О. В. Герман
Ю. О. Герман
Программирование
на
JAVA
и
C#
для
студента
bhv
+ CD

Олег Герман
Юлия Герман

Программирование на JAVA и C# для студента

Санкт-Петербург
«БХВ-Петербург»
2005

УДК 681.3.06
ББК 32.973.26-018.1
Г38

Герман О. В., Герман Ю. О.

Г38 Программирование на Java и C# для студента. — СПб.: БХВ-Петербург, 2005. — 512 с.: ил.

ISBN 5-94157-710-9

Рассмотрены основные вопросы программирования на языках JAVA и C#, включая их сравнительное описание как двух важнейших и весьма сходных прикладных платформ для создания современных сетевых приложений. Книга содержит теоретическую часть, объясняющую основные моменты программирования, и практическую, включающую задания, контрольные вопросы и много законченных примеров с подробными объяснениями и комментариями, которые позволяют эффективно перейти к самостоятельному написанию программ на языках JAVA и C#. На компакт-диске размещены листинги примеров, рассмотренных в книге.

Для студентов, преподавателей и программистов

УДК 681.3.06
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Дарья Масленникова*
Компьютерная верстка	*Татьяны Олоновой*
Корректор	*Наталия Першакова*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 28.09.05.
Формат $60\times90^{1}/_{16}$. Печать офсетная. Усл. печ. л. 32.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-710-9

© Герман О. В., Герман Ю. О., 2005
© Оформление, издательство "БХВ-Петербург", 2005

Оглавление

Введение 9

Сравнительный анализ Java и C#: общность платформ и отличительные особенности 11
Зачем нужно знать и Java и C#? 15

ЧАСТЬ I. JAVA 17

Глава 1. Основы программирования на языке Java 19

Программирование "без классов" 23
Классы 40
Объявление переменных и методов, использование их в программе 52
Создание визуального интерфейса 63
Программирование обработки событий от элементов, мыши и клавиатуры 67
Программирование ввода-вывода с использованием файлов 78
Работа со строками 89
Использование массивов 91
Апплеты 93
Обработка исключительных ситуаций 96
Работа с графикой 98

Глава 2. Практические занятия по Java 109

Основы HTML 109
  Цель занятия 109
  Краткие теоретические сведения 109

Задание......119
Контрольные вопросы......119
Использование скриптов JavaScript в документах HTML......120
Цель занятия......120
Краткие теоретические сведения......120
Задание......132
Контрольные вопросы......133
Введение в Java......133
Цель занятия......133
Краткие теоретические сведения......134
Задание......144
Контрольные вопросы......145
Реализация взаимодействия между апплетами......145
Цель занятия......145
Краткие теоретические сведения......146
Задание......155
Контрольные вопросы......155
Внутренняя база данных апплета......156
Цель занятия......156
Краткие теоретические сведения......156
Задание......163
Контрольные вопросы......164
Работа с формами и меню......164
Цель занятия......164
Краткие теоретические сведения......164
Задание......174
Контрольные вопросы......174
Java и базы данных......175
Цель занятия......175
Краткие теоретические сведения......175
Задание......183
Контрольные вопросы......184
Основы XML. Преобразование XML-HTML.
Использование JavaScript......184
Цель занятия......184
Краткие теоретические сведения......184
Задание......192
Контрольные вопросы......192
Взаимодействие XML-Java-JavaScript......193
Цель занятия......193
Краткие теоретические сведения......193

Задание..203
Контрольные вопросы 203
Чтение XML-файла с использованием файлового диалога 203
Цель занятия 203
Краткие теоретические сведения..203
Задание..212
Контрольные вопросы 213
Потоки в Java...213
Цель занятия 213
Краткие теоретические сведения..213
Задание..221
Контрольные вопросы 222
Создание приложений "клиент-сервер" 222
Цель занятия 222
Краткие теоретические сведения..223
Задание..235
Дополнительные сведения...236
Контрольные вопросы 246
Доступ к серверной базе данных из клиента..........................247
Цель занятия 247
Краткие теоретические сведения..247
Задание..275
Контрольные вопросы 276
Использование Java Beans в других средах 277
Цель занятия 277
Краткие теоретические сведения..277
Задание..287
Контрольные вопросы 288
Изучение механизма сериализации 288
Цель занятия 288
Краткие теоретические сведения..288
Задание..300
Контрольные вопросы 300
Создание сервлетов..300
Цель занятия 300
Краткие теоретические сведения..301
Задание..307
Контрольные вопросы 308

Создание почтовой службы в стандартном Java 308
Цель занятия 308
Краткие теоретические сведения 309
Задание 312
Контрольные вопросы 312
Создание JSP-страниц 313
Цель занятия 313
Краткие теоретические сведения 313
Задание 316
Контрольные вопросы 316
Создание простого браузера 317
Цель занятия 317
Краткие теоретические сведения 317
Задание 323
Контрольные вопросы 323
Сводка основных использованных команд Java 324

ЧАСТЬ II. C# 331

Глава 3. Основы программирования на языке C# 333

Введение в язык C# 333
Платформа C# для Java-программистов 342
Программирование "без классов" 345
Использование классов 350
Использование подпрограмм 362
Объявление массивов 367
Работа с файлами 371
Сериализация объектов 376
Создание приложений на основе формы 379
Работа со строками 386
Создание сборок 389
Создание Web-приложений 392
Реализация API-вызовов 398

Глава 4. Практические занятия по C# 403

Файловый ввод-вывод в C# 403
Цель занятия 403
Краткие теоретические сведения 403

Задание....................417
Контрольные вопросы....................419
Работа с базами данных в C#....................419
Цель занятия....................419
Краткие теоретические сведения....................419
Задание....................426
Контрольные вопросы....................426
Простейшее рисование в C#....................426
Цель занятия....................426
Краткие теоретические сведения....................427
Задание....................433
Контрольные вопросы....................433
Изучение механизма потоков для смены графических изображений....................433
Цель занятия....................433
Краткие теоретические сведения....................433
Задание....................443
Контрольные вопросы....................443
Создание собственных компонентов....................444
Цель занятия....................444
Краткие теоретические сведения....................444
Задание....................452
Контрольные вопросы....................453
Клиент-серверное взаимодействие на основе протоколов TCP и HTTP....................453
Цель занятия....................453
Краткие теоретические сведения....................453
Задание....................461
Контрольные вопросы....................461
Работа с классом таймера....................461
Цель занятия....................461
Краткие теоретические сведения....................462
Задание....................468
Контрольные вопросы....................469
Обработка польской записи....................469
Цель занятия....................469
Краткие теоретические сведения....................469
Задание....................480
Контрольные вопросы....................481

Работа с коллекциями 481
Цель занятия 481
Краткие теоретические сведения..481
Задание..491
Контрольные вопросы 491
Сводка основных использованных команд C# 491

Приложение. Описание компакт-диска 499

Содержимое компакт-диска..499
Инструкции по работе с листингами программ на Java и C# 499
Запуск классов Java..500
Запуск апплетов Java..500
Запуск приложений C# 502
Установка Tomcat и Java 503

Список литературы 505

Предметный указатель 507

Введение

Практическое пособие посвящено программированию на языках Java и C#. Выбор именно этих языков продиктован весьма сходными концептуальными посылками, а освоение любого из них делает изучение второго языка достаточно тривиальным. Попутно (но вполне достаточно для самостоятельной работы с ними) рассматриваются языки JavaScript, HTML, JSP, XML, которые очень часто применяются совместно с Java и C# (через Web-приложения) и пользуются высоким спросом на рынке труда программистов. Владение этими языками, вне всяких сомнений, значительно увеличивает шансы будущих программистов-профессионалов на получение неплохой работы.

Необходимость написания этого пособия продиктована тем, что обычно в ходе семестра студенты не успевают изучить огромные справочники по языкам Java и C#. Изучение Java или C# начинается, как правило, после знакомства с Delphi или Visual Basic на младшем курсе. Поэтому концепции программирования этих новых языков должны быть изложены достаточно элементарно и сжато.

Начинающие изучать серьезно тот или иной язык нуждаются в поэтапном вхождении в проблемную область. Вовсе нет необходимости "обрушивать" на неподготовленный ум лавину информации. Представляется, что пользователю не хватает именно практических пособий по языкам программирования, поэтому авторы надеются, что предлагаемый практикум поможет в какой-то мере восполнить этот пробел. Для чего применяется язык Java (C#), что в нем особенного, какова его общая концепция, как строить программы в этом языке и как решать типичные задачи — все эти вопросы отражены в настоящей книге.

Скажем, обычный пользователь не станет заострять внимание на типах данных (он возьмет на вооружение только три-четыре) и не будет истощать свою память различными перегруженными методами языка. Вообще, из объектно-ориентированного программирования только наследование должно пониматься по-настоящему глубоко. И вряд ли студентом будут активно использоваться инкапсуляция и полиморфизм, ибо он не является профессиональным программистом. Инкапсуляция связана с ограничением доступа к членам классов (что, вообще говоря, никак не влияет на работоспособность создаваемых программ), а полиморфизм — с использованием одноименных методов, но с различными типами аргументов и, вообще говоря, различным их числом. Однако всегда можно назвать методы разными именами, так что такое ухищрение, как полиморфизм, — лишь более удобный способ записи функциональности программы.

Данный практикум прошел апробацию в учебном процессе на кафедре информационных технологий автоматизированных систем Белорусского государственного университета информатики и радиоэлектроники (Минск) и предназначен студентам, самостоятельно изучающим языки Java и C#, и преподавателям, проводящим занятия по данным языкам и интернет-технологиям. Он охватывает двухсеместровый курс по современным технологиям программирования на Java и односеместровый — по программированию на C#. С нашей точки зрения, необходимы третий и, возможно, четвертый семестры для углубленного изучения тем, не вошедших в пособие (прежде всего, это программирования в J2EE, CORBA, .NET, изучение OLE-автоматизации). Практикум предназначен для выполнения двухчасовых практических (лабораторных) работ и снабжен необходимым теоретическим введением и подробным разбором рассматриваемых примеров. Каждому практическому занятию предпослано теоретическое введение по теме занятия, а всем занятиям вместе — общее теоретическое введение по соответствующему языку. Именно эти обстоятельства и делают настоящее пособие полезным для лиц, приступающих к овладению такими в высшей степени замечательными и широко востребованными программными продуктами, как Java и C#. Из сказанного следует, что пособие ориентировано на лиц, владеющих основами программирования на каком-нибудь языке программирования. Каких-либо более серьезных ограничений нет.

Остановимся более подробно на структуре предлагаемого пособия. Оно состоит из двух частей. Первая часть посвящена языку Java и включает общетеоретическое введение в этот язык (кстати говоря, полезное и при изучении C#), а также двухсеместровый цикл практических занятий, посвященных различным прикладным задачам. Перечень этих задач охватывает основные разделы полного курса программирования на стандартном Java. В каждом практическом занятии формулируется цель, рассматривается теоретическая сторона вопроса и приводится программная реализация с необходимыми пояснениями. Практическое усвоение темы достигается тогда, когда читатель хорошо разберется в программе и самостоятельно выполнит предлагаемое задание.

Вторая часть пособия целиком посвящена языку C#, содержит общетеоретическое введение и односеместровый цикл практических занятий. Структура практических занятий аналогична представленной в первой части. Темы практических занятий на C# пересекаются с темами по Java. Пособие построено так, что дает возможность параллельно рассматривать оба языка ввиду их несомненной концептуальной общности. Оно полезно для тех, кто не знает ни один из рассматриваемых языков, либо знает один из них и хотел бы научиться программировать на другом. При этом, несомненно, владение обоими языками открывает перед вами более широкие перспективы.

Сравнительный анализ Java и C#: общность платформ и отличительные особенности

Язык Java создан фирмой Sun для Интернета в середине 90-х годов прошлого столетия. Можно было бы сказать, что это еще один вариант C++, только без указателей, без необходимости сборки мусора (очистки памяти от неиспользуемых объектов), наличия лучшей системы защиты и пр., если бы не одно фундаментальное обстоятельство: Java позволял создавать интерактивные приложения (апплеты) непосредственно в клиентских сайтах. Массовость аудитории под общим названием "интернет-

клиенты" обеспечила мощный импульс для распространения Java. Выяснилось, что Java отлично продуман: громоздкие и тяжеловесные конструкции C++ в нем обрели компактный и изящный вид. Помимо этого, язык Java давал возможность писать распределенные приложения для локальных сетей и баз данных. Наконец, специальные приложения — сервлеты, написанные на Java, позволяли также строить серверные приложения, работающие на удаленных компьютерах и обслуживающие запросы от клиентов. Фирма Microsoft, поняв, что теряет крупный сегмент рынка, с определенным запозданием отреагировала на этот вызов созданием Visual J++. Однако существовало мнение, что Visual J++ — просто переопределенная версия Java. Развернулась конкурентная борьба. Дошло до того, что фирма Microsoft вообще отказалась поддерживать виртуальную машину Java (JVM) в браузере Internet Explorer. Sun возбудила судебное дело и выиграла его.

В процессе развития своих программных концепций Sun создала несколько новых важных технологий: RMI, CORBA, JSP, JavaBeans и др. Венцом этого развития стало появление технологии J2EE (Java 2 Enterprise Edition), поддерживающей работу с удаленными объектами и позволяющей создавать серверы приложений. Чтобы не отстать и продолжить борьбу, фирма Microsoft создала нечто большее, чем новую Java-подобную версию языка, а именно — платформу .NET, которая содержит виртуальную машину, библиотеки, механизмы, поддерживающие работу с удаленными объектами на основе технологии COM+, средства разработки ASP-страниц и пр. Не будет преувеличением сказать, что одним из ключевых компонентов .NET является язык C#. Хотя, кроме C#, в рамках этой платформы имеются еще и другие языки — C++.NET, VB.NET, ASP.NET, Visual J#.NET, программы на которых компилируются в предназначенный для исполнения виртуальной машиной CLI (Common Language Infrastructure) промежуточный код на языке MSIL (Microsoft Intermediate Language).

C#, без сомнения, замечательный язык, удачно сочетающий в себе элементы и идеи Java (в наибольшей степени), Delphi (в плане реализации среды разработки, обработки событий и интерфейса с элементами), Visual Basic (в плане использования

свойств `Put` и `Get`) и С++ (как общего родительского языка). Мотивация создания языка С# осталась прежней — борьба за крупнейший сегмент рынка и первенство в области новейших технологий. Ситуация, в которую эта борьба так или иначе завела, может быть резюмирована следующим образом: Java выступил как базовая концепция для С#. Java и С# теперь стоят в целом на различных (но "идеологически" общих) платформах. Коротко охарактеризуем эти платформы.

Прежде всего, начнем с универсальности Java — поддерживается возможность исполнения Java-приложений под управлением различных операционных систем: Windows, UNIX, Solaris и др. Это достигается за счет того, что исходные тексты на языке Java можно скомпилировать в промежуточный байт-код, который затем выполняется виртуальной машиной Java (JVM). Следовательно, Java-программы выполняются везде, где установлена JVM. Аналогичную роль играет виртуальная машина среды .NET — CLI. Однако у CLI есть и преимущества — во-первых, функции одних языков среды .NET можно использовать в других языках, и, во-вторых, во всех языках .NET объекты реализуются одинаково (например, класс, созданный в С++.NET, можно использовать как родительский в VB.NET, и наоборот).

Второй важной особенностью сравниваемых платформ является работа с удаленными объектами. *Удаленный объект* — это экземпляр некоторого класса, содержащий методы и свойства, которые образуют интерфейс. Ясно, что удаленные объекты создаются на других сетевых компьютерах и становятся доступными через сеть. Фирма Sun разработала пакет классов и интерфейсов, образующих множество объектов EJB (Enterprise Java Beans), которые размещены в EJB-контейнерах и доступны через сеть. Объекты EJB имеют определенную функциональную ориентацию:

- для работы с базами данных;
- для обмена сообщениями;
- для выполнения сложных вычислений.

Технология EJB позволяет реализовать так называемые серверы приложений. Аналогом этой технологии в .NET является COM+. До появления .NET фирма Microsoft использовала технологии

COM/DCOM (Component Object Model/Distributed Component Object Model), общий смысл которых можно передать следующим образом. Создается класс COM-компонента, свойства и методы которого хранятся в двоичном файле, так что для запуска методов порождаемых из него COM-объектов не имеет значения, из какой программной среды они вызываются. Доступ к этим методам выполняют специальные сервисные программы, образующие интерфейс с COM-компонентами и использующие информацию, хранящуюся в реестре Windows. Клиентская сторона выполняет запрос на создание и получение (через сеть) COM-объекта. Создается объект некоторого класса COM с определенной функциональностью (методами и свойствами) После создания объект становится доступным на другом сетевом компьютере, связанном с тем, где зарегистрирован исходный класс. Недостаток COM/DCOM состоит в привязке к реестру Windows. В этом случае ни о какой платформенной независимости не может идти речи. В .NET такой привязки к реестру нет. Используется аналог службы имен Java. Таким образом, технология COM+ доведена до уровня платформенной независимости, изначально характерной для Java.

Важной особенностью Java является возможность создания Web-серверных приложений на базе технологии страниц JSP. Задача Web-серверного приложения состоит в получении информации от интернет-клиента, ее обработке и возврате результатов на сторону клиента. Web-серверное приложение на языке Java запускается на выполнение программой Web-сервера (в качестве которого, например, можно использовать Tomcat). Аналогом этой технологии в C# является создание в среде разработки приложений .NET ASP-страниц, написанных на C#. Следует отметить, что разработка ASP-страниц в среде .NET автоматизирована в значительно большей степени, чем в Java.

Программы в Java и C# состоят из классов (class) — основной структурной единицы приложения. И в Java, и в C# разрешается наследование только от одного класса; вместе с тем, можно подключать к классу более одного интерфейса. C# позволяет программисту не заботиться об удалении объектов (сборке мусора) — это делается, как и в Java, автоматически. Аналогичным образом в Java и C# реализован механизм области видимости имен. В C# этот механизм назван пространством имен, в Java —

использованием пакетов (packages). И в Java, и в C# не используются указатели, а рабочим языком в обоих случаях является C, так что некоторые фрагменты кода на Java и C# могут быть неотличимы. В деталях некоторые отличия все же есть. Например, в Java главная точка входа в приложение есть `public static void main(String [] args)`. Этот синтаксис является неизменным. В C# метод `Main()` может быть переопределен. Например, таким образом:

```
public static void Main()
```

В C# имеется оператор `goto` (перехода на метку), в языке Java — нет. В C# можно определять структуры и перечисления; в Java эта возможность исключена. Имеются различия в реализации оператора `switch`. Оба языка поддерживают многопоточность. Эти общие черты и отличия вполне конкретизируются в материалах практических занятий.

Специфическими компонентами .NET являются службы каталогов и службы Windows, о которых можно прочитать в специальной литературе [3, 9].

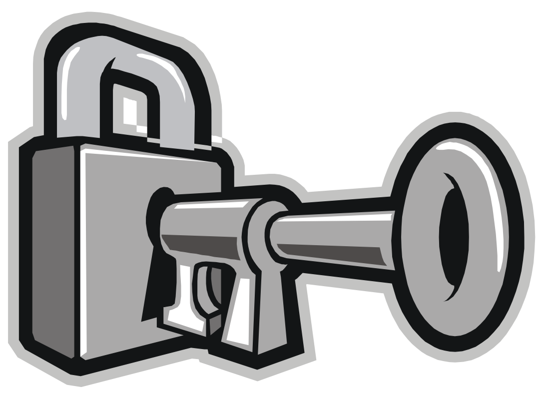
Зачем нужно знать и Java и C#?

Ответ частично сформулирован выше. Уместно систематизировать рассуждения следующим образом.

- Java и C# ориентированы на общий стремительно растущий сегмент рынка. Они призваны решать одни и те же задачи, используя, в принципе, одни и те же концептуальные установки.
- Позиции Sun и Microsoft на момент написания этой книги достаточно сильны. В настоящее время спрос на специалистов по Java и по C# сравнительно высок. Знание обоих языков создает более сильную позицию в перспективе.
- Изучение одного языка при знании другого, как пишет Гиббонз [3], — дело нескольких часов. Мы, разумеется, полагаем, что времени требуется побольше, но с мыслью Гиббонза согласны, поскольку:
    - концептуальные основы сравниваемых языков в целом одинаковы;

- коды приложений базируются на синтаксисе языка C и имеют много общего;
- современное программирование требует скорее понимания, чем запоминания.

- В случае, если какой-то из сравниваемых языков "сдаст позиции", у вас сохраняется задел в области другого. Это создает большую уверенность в своих возможностях и силах. Имейте в виду, что соперничество этих фирм, вероятно, приведет к созданию новых технологий, и какая из них окажется победителем, сейчас сказать трудно.
- Сравнительное изучение позволяет рассмотреть возможности языков более критично.

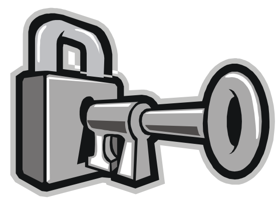


ЧАСТЬ I

JAVA

ГЛАВА 1. Основы программирования на языке Java

ГЛАВА 2. Практические занятия по Java



Глава 1

Основы программирования на языке Java

Язык Java — это объектно-ориентированный язык, прототипом которого можно считать C++. Java в значительной степени упростил написание программ и позволил выполнять их на разных платформах (под управлением различных операционных систем). Простота этого языка, его хорошая защищенность и возможности, предоставляемые для программирования в Интернете, сделали его весьма популярным.

Java является языком сетевого программирования, в первую очередь ориентированным на среду Интернет. Концепция программирования в Интернете базируется на рассмотрении двух сторон: стороны *клиента* и стороны *сервера*. К серверу обращаются многие клиенты. Обычно обращение происходит за информацией из базы данных либо для выполнения некоторой полезной программы. Клиентом является подключенный к Интернету конечный пользователь. Такой пользователь, работая на своем индивидуальном компьютере, должен знать интернет-адрес сервера и имя той программы, с которой он хочет связаться. Для доступа к серверу клиент должен создать свой сайт на языке HTML или построить *апплет* на языке Java и ввести в него функциональную часть, выполняющую обращение к серверу. Посредником между клиентской и серверной сторонами является специальная программа — *браузер*. Содержимое сайта браузер передает на сервер. На сервере выполняется программа *Web-сервер*, которая взаимодействует с

браузером через определенный порт и активизирует обработчик скрипта или сервлет (если приложение сервера, обрабатывающее сайт, написано на языке Java). Обработчик скрипта выполняет конкретную прикладную задачу, используя принятые данные сайта. Например, обработчик скрипта может обратиться к базе данных, передав в свою очередь запрос серверу баз данных, размещенному, как правило, на той же машине, что и Web-сервер. Результаты своей работы (как правило, так же в форме сайта) обработчик скрипта или сервлет возвращает клиенту.

Внешне после запуска сайт может иметь, например, такой вид (рис. 1.1).

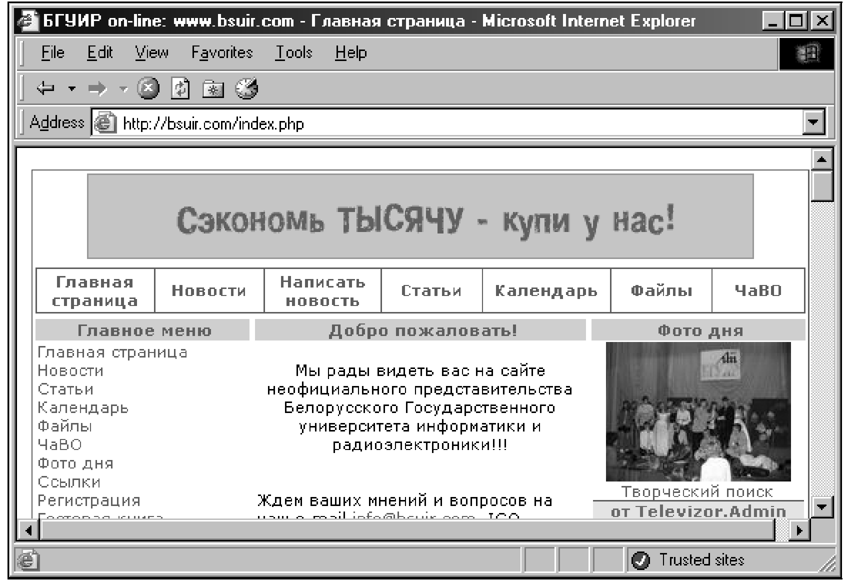


Рис. 1.1. Неофициальный сайт Белорусского государственного университета информатики и радиоэлектроники (Минск)

Поскольку язык HTML используется для программирования сайтов очень широко, ему посвящено наше первое практическое занятие. На сайте можно разместить много самой разнообразной информации. Эту информацию можно передать серверному приложению средствами специального посредника (браузера) —

например, Internet Explorer. Браузер связывается с Web-сервером по каналу связи через определенный порт (как правило, с номером 8080) и передает ему информацию в определенном формате (протоколе). Обычно таким протоколом является HTTP (Hypertext Transfer Protocol, протокол передачи гипертекстовых файлов). В сайты, написанные на HTML, можно вставлять *апплеты (applets)* — динамические окна, запрограммированные средствами языка Java. Сервер, приняв документ от браузера, передает управление серверному приложению — обработчику скрипта или сервлету, который может получить значения элементов формы — текстовых полей, выделенных элементов списков, переключателей. Типичная задача серверного приложения — зайти в базу данных и найти информацию, затребованную клиентом. Эта задача возникает не только в Интернете, но и при работе в локальных сетях, например, в условиях фирмы, учреждения или производства. Таким образом, создание распределенных приложений на Java является одним из основных назначений данного языка.

Перед тем как перейти к описаниям практических занятий, мы предпошлем им общие теоретические сведения по языку Java.

Основными *предварительными* моментами в освоении этого языка являются следующие:

- использование классов и классовых переменных как основных структурных единиц программ;
- объявление переменных и методов, их использование в программе;
- создание визуального интерфейса (форм, кнопок, списков, меню и пр.);
- программирование обработки событий от элементов, мыши и клавиатуры;
- программирование ввода/вывода через файлы и т. д.

Последующее изложение языка включает более сложные вопросы, например, использование потоков, клиент-серверных технологий, работу с XML (eXtended Markup Language, расширенный язык разметки) и пр. Вам потребуется установить у себя Java SDK 1.*x* (например, $x = 4$). Эту систему можно бесплатно скачать через

Интернет с сервера фирмы Sun, расположенного по адресу **http://Java.sun.com/j2se/1.4/download-windows.html**, либо последнюю версию J2SE SDK (Java 2 Standard Edition Software Development Kit), которую можно скачать с сайта фирмы Sun по адресу **http://Java.sun.com/j2se/**. Основными программами, которые задействует этот практикум, являются java.exe и javac.exe. Последняя из этих программ осуществляет создание классов путем компиляции исходных Java-файлов. Программа java.exe выполняет созданные классы. Java-классы выполняются на различных платформах: Windows, UNIX, Linux, Solaris и др. при условии, что на компьютере установлена вирутальная машина Java (JVM), а java.exe является ее ядром. Отметим, что в большинстве современных браузеров виртуальная машина Java установлена по умолчанию.

Каталог, в котором размещена система Java SDK, может иметь, например, такой вид (рис. 1.2).

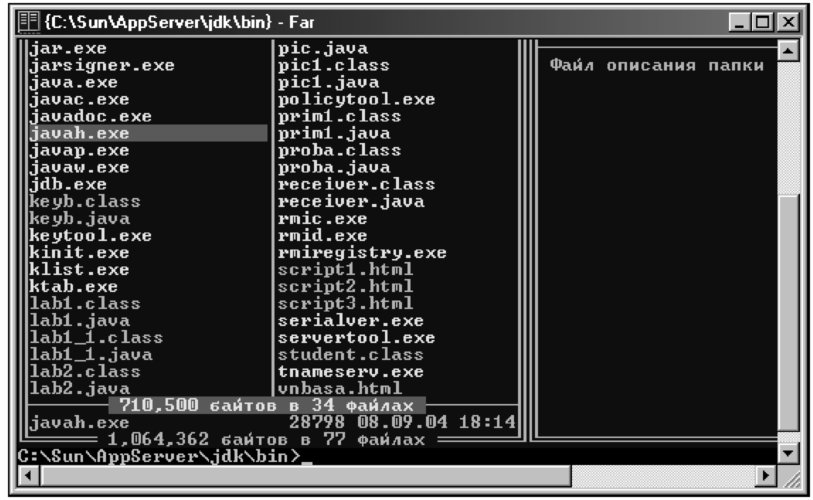


Рис. 1.2. Каталог с размещенной в нем системой Java SDK

В этом практикуме нам понадобятся следующие программы SDK:

- javac.exe — компилятор исходных Java-файлов;
- java.exe — программа выполнения классов;

- jar.exe — программа создания архивов;
- packager.exe — программа создания компонентов Java BEANS.

При первоначальной работе с Java полезны такие утилиты, как javadoc.exe — используется для документирования программ, и appletviewer.exe — используется для запуска апплетов.

Программирование "без классов"

Программировать без классов уже практически нельзя — это не модно, да и становится крайне тяжело, когда речь идет о сколько-нибудь приемлемом интерфейсе. Имейте в виду, что Java-приложение состоит из классов, где класс — это как функция в языке С или процедура в Delphi. Поэтому заголовок этого раздела не следует понимать буквально, а словосочетание "без классов" взято в кавычки. Разумеется, можно использовать консольные приложения, т. е. приложения без привычных нам окон. Но вот уже для работы с файлами или вывода на консоль вам понадобятся классы. В этом разделе мы намереваемся по возможности использовать в программе простые переменные. Например, мы хотели бы использовать операции:

```
x = 2;
y[0] = x - 1;
```

с переменными `x` и `y` и т. д., и т. п.

Итак, создадим простейшее Java-приложение, в котором запрашивается имя пользователя и выводится приветствие. Увы, для первого знакомства это приложение (листинг 1.1) не очень простое!

Листинг 1.1. Приложение, выполняющее приветствие

```
import java.awt.*;
import java.io.*;
public class proba
{
 static int b;
```

```
 static String sname;
 static char[] charray;
 public static void main(String args[])
 {
  charray = new char[20];
  int i=0;
  // Вывод вопроса о вашем имени:
  System.out.println("What is Your name?");
  try
   {
    while((b=System.in.read()) !=13)
    {
     charray[i++]=(char) b;}
    }
  catch(IOException e)
  {
   System.out.println("The error "+e);
  }
  sname=new String(charray);
  //Вывод строки приветствия:
  System.out.println("Hello, fellow-programmer, "+sname);
 }
}
```

Разберем приведенный в листинге код более подробно. Сначала идет подключение библиотек (библиотечных классов):

```
import java.awt.*;
import java.io.*;
```

В языке C вы подключаете их с помощью инструкции `#include`. В Delphi вы используете команду `use`. Так что здесь нет ничего необычного. Дальше идет объявление класса (этого мы не избежали!):

```
public class proba
{  ...  ...}
```

Наш класс называется `proba`. Файл, где его следует сохранить, набрав текст программы в каком-нибудь редакторе (например, можно использовать Блокнот или встроенный редактор FAR manager, Norton Commander), также должен называться proba.java. Все же это еще не классы в традиционном смысле. Объявляем используемые *переменные*:

```
static int b;
static String sname;
static char[] charray;
```

Во всех языках программирования используются переменные разных типов. У нас это `int`, `String` и `char` (целое число, строка и символ соответственно). Слово `static` нам пришлось использовать, так как мы работаем с классами нетрадиционно, т. е. не создаем объекты этих классов. Итак, запомним, что если мы объявляем переменные класса и не создаем объектов класса, то переменные должны дополняться словом `static`. Наконец, добрались до единственного метода — `main()`:

```
public static void main(String args[])
{
charray = new char[20];
int i=0;
System.out.println("What is Your name?");
try{
  while((b=System.in.read()) !=13)
{charray[i++]=(char) b;  }
 }
catch(IOException e)
{System.out.println("The error"+e);}
sname=new String(charray);
System.out.println("Hello, fellow-programmer, "+sname);
}
```

Метод `main()` всегда объявляется так, как записано (запоминаем). Аргументы этого метода — параметры командной строки,

использованной для запуска приложения. Пока что нам это не потребуется. Затем снова следует два объявления переменных:

```
// Объявляется массив символов, состоящий из 20 элементов:
charray = new char[20];
int i=0;
```

Где слово `static`? Запомним, что внутри методов это слово использовать нельзя. *Переменные класса* должны объявляться вне методов, но внутри объявлений классов. Внутри методов объявляются локальные переменные методов.

Далее идет команда:

```
System.out.println("What is Your name?");
```

Эта команда, между прочим, использует имя класса `System`. Запомним эту команду для вывода строк на консоль. Результат ее действия мы вскоре увидим. А вот далее в цикле выполняется чтение символов до тех пор, пока не будет прочитан символ `<Enter>` (его код 13). Отсюда начинается то программирование "без классов", о котором мы и намеревались поговорить. Пока не будем "разжевывать" эту программу, но приведем на рис. 1.3 результат ее выполнения.

```
cmd

C:\Sun\AppServer\jdk\bin>java proba
What is Your name?
oleg
Hello, fellow-programmer, oleg

C:\Sun\AppServer\jdk\bin>
```

Рис. 1.3. Результат выполнения программы, приведенной в листинге 1.1

Для того чтобы скомпилировать программу, используем командную строку MS-DOS:

```
javac proba.java
```

Программа javac — это и есть компилятор Java-программ. Чтобы ввести эту команду, например, из программы Norton Commander, наберите в командной строке `>cmd`.

Если ошибок нет, система создаст файл proba.class и разместит его в том же каталоге, где находится файл proba.java. Если файл proba.java лежит в каталоге, отличном от того, в котором размещен файл javac.exe, то следует указать путь к файлу proba.java, например:

```
javac c:\temp\proba.java
```

(указан путь `c:\temp\proba.java`).

Запускаем класс на выполнение командой

```
java proba
```

(Найдите этот файл на рис. 1.3.)

Заметим, что если java.exe находится в каталоге, отличном от того, в котором расположен созданный нами класс `proba.class`, то следует предварительно выполнить команду:

```
set classpath=%classpath%;c:\temp\proba
```

Эту команду следует ввести из командной строки DOS; такую возможность предоставляет также, например, FAR manager или Norton Commander.

Теперь поговорим о программировании. Предполагается, что у вас уже есть кое-какой опыт, но, тем не менее, двигаемся все же с "нуля".

Традиционных команд сравнительно мало:

- присваивание (=);
- проверка условия (`if`);
- циклы (`for` и `while`).

Существует еще команда `goto`, но в данном пособии ее практически не используем (как утверждают специалисты, про нее лучше забыть сразу, потому как в Java это ключевое слово зарезервировано, но самой команды нет).

Традиционное программирование предполагает правильную запись команд для получения задуманного результата. Программы, написанные людьми, не имеющими опыта (даже если это доктора физико-математических наук), зачастую вызывают недоумение.

При написании кода вы работаете с именами переменных. Эти имена должны что-то выражать. Например, пусть требуется ввести какое-нибудь слово и подсчитать его длину (число символов). Обозначим это слово переменной `slovo`, объявим ее и присвоим ей значение пустой строки. *Присваивание* реализуется оператором = следующим образом:

```
static String slovo="";
```

Теперь значение переменной `slovo` равно пустой строке "". Если вам не нравится пустая строка, то присвоим переменной `slovo` иное значение. Например:

```
slovo="Ур-ра".
```

Длина слова возвращается в результате выполнения команды `slovo.length()`.

Поскольку метод `System.in.read()` читает байты (единицы информации, соответствующие символам строки, набираемым на клавиатуре), то объявляем массив байтов и создаем его в памяти:

```
static char[] charray = new char[20];
static int i=0;
```

Массив — это множество переменных; имена переменных совпадают с именем массива, но дополняются номерами (индексами). Например, `charray[0]` — первая переменная массива `charray`; `charray[1]` — вторая переменная массива `charray` и т. д. Создание массива выполняет команда `new`, которая использует базовый тип (в данном случае `char`) и размерность массива (указывается в квадратных скобках).

Организуем цикл для чтения:

```
try{
  while(( int b=System.in.read()) !=13)
{charray[i++]=(char) b;} //тело цикла
}
```

Цикл `while` работает так. В скобках записывается условие, необходимое для его выполнения:

```
((int b=System.in.read()) !=13)
```

А именно, указывается, что `b = System.in.read()`, а само проверяемое условие таково: `b != 13` (`b` не равно 13). Если нажимаемая клавиша имеет код, отличный от 13, то символ попадает в массив `charray` в ячейку `i` и немедленно `i` увеличивается на 1 (это выполняется командой `i++`). Нам надо внимательно следить, чтобы число вводимых символов не превысило 20, так как мы объявили массив именно этого размера. Запомним, что любой цикл имеет тело. *Тело цикла* ограничивается фигурными скобками `{…}`, следующими сразу за условием цикла. То, что в них записывается, выполняется многократно, пока верно условие цикла. Выйти из цикла можно с помощью команды `break`. Итак, внутри нашего цикла считывается очередной символ с клавиатуры по команде:

```
int b=System.in.read().
```

Но значение считанного символа записывается в целую переменную `b`. Конечно, вы знаете, что символы имеют свои числовые коды, представляемые байтами. Например, код пробела — 32. Однако при записи в массив `charray` числовой код преобразуется в настоящий символ, который данный код определяет. То, что выполняет команда

```
charray[i++]=(char) b;
```

означает преобразование числа `b` в соответствующий символ `(char)b` (`b` и `(char)b`, по сути, не одно и то же). Это и есть *преобразование типов*. В нашей программе мы еще раз используем преобразование типов при выводе строки на консоль:

```
// строка формируется из символов — представителей другого
// типа:
slovo=new String(charray);
System.out.println("The word  "+slovo+ " was read with length "+slovo.trim().length());
//выводим строку на консоль
```

Команда `slovo.trim().length()` содержит две последовательно исполняемые операции: `trim()` — удаляет пробелы с начала и конца строки, и `length()` — возвращает длину строки.

Теперь приведем итоговую программу (листинг 1.2), снабженную комментариями. *Комментарии* указываются после двух разделительных наклонных черт ("слэшей").

Листинг 1.2. Приложение, выполняющее подсчет длины строки

```
import java.awt.*; // Подключаем библиотеки классов
import java.io.*;

public class proba  // Объявляем единственный класс
{
 static int b; // Объявляем типы переменных класса
 static String slovo="";
 // Объявляем массив символов и создаем пустой массив:
 static char[] charray=new char[20];
 // Определяем единственный метод приложения:
 public static void main(String args[])
 {
  int i=0;  // Внутреннее объявление переменной i в методе
            // main()
  System.out.println("Input a word ->"); // Вывод строки
                                         // с подсказкой
  try
   {
    // Конструкция try  catch используется для защиты,
    // о чем будет рассказано позже
    while((b=System.in.read()) !=13) // Организуем цикл,
    // try пока не рассматриваем — это к циклу не имеет
    // отношения
    {
     charray[i++]=(char) b;  } // Тело цикла
    }
  catch(IOException e)
   {
    System.out.println("The error"+e);
   }
  // Преобразуем массив charray символов в строку slovo:
  slovo=new String(charray);
```

```
  System.out.println("The word  "+slovo+ " was read with
length "+
   slovo.trim().length()); // Вывод ответа
 }
}
```

Заинтересованный читатель спросит о назначении команд `try catch`. Забегая вперед, отметим, что они используются для "перехвата" и обработки возможных ошибок, например, при вводе данных. Этот вопрос будет обсуждаться в отдельном разделе.

Теперь изменим несколько эту программу, чтобы в модифицированном виде она была бы применима для чтения двух чисел и отыскания их наибольшего общего делителя.

Чтение целого числа — это уже некоторая задача. Сначала покажем, как прочесть всего одно число (листинг 1.3).

Листинг 1.3. Приложение, выполняющее ввод числа с клавиатуры

```
import java.awt.*;
import java.io.*;
public class proba
{
 static int b;
 static String slovo="";
 static char[] charray;
 public static void main(String args[])
 {
  charray = new char[20]; // В массив попадают коды символов
                          // '0'..'9'
  int i=0;
  System.out.println("Input a number ->"); // Приглашение
                                           // ввода числа
  try
   {
```

```
    // Чтение символов числа выполняется в цикле:
    while((b=System.in.read()) !=13) {charray[i++]=(char) b;
}
   }
  catch(IOException e)
   {
    System.out.println("The error"+e);
   }
  slovo=new String(charray); // В строке slovo — прочитанное
                             // число
  System.out.println("The word  "+slovo+ " was read with
  length"+slovo.trim().length());
  double j=0; // Число j используем для преобразования в него
              //  слова slovo
  // Тип double — это тип вещественного числа с фиксированной
  // точкой удвоенной точности
  int k=0;
  int l=slovo.trim().length();
  for(i=0;i<=(l-1); i++) // В этом цикле конструкция
                         // slovo.charAt(i) дает i-ый
                         // символ слова slovo
  {
   switch(slovo.charAt(i)) // Формируем цифру k в зависимости
                           // от i-го символа слова slovo
    {
     case '0': k=0; break; // если символ '0', то полагаем
                           // k=0;
     case '1': k=1; break; // если символ '1', то полагаем
                           // k=1;
     case '2': k=2; break; // если символ '2', то полагаем
                           // k=2;
     case '3': k=3; break; // если символ '3', то полагаем
                           // k=3;
     case '4': k=4; break; // если символ '4', то полагаем
                           // k=4;
```

```
      case '5': k=5; break; // если символ '5', то полагаем
                            // k=5;
      case '6': k=6; break; // если символ '6', то полагаем
                            // k=6;
      case '7': k=7; break; // если символ '7', то полагаем
                            // k=7;
      case '8': k=8; break; // если символ '8', то полагаем
                            // k=8;
      case '9': k=9; break; // если символ '9', то полагаем
                            // k=9;

       default : k=0;break;
     }
    // Напомним, что 123=1×10^2 + 2×10^1 +3×10^0
    j=j+k*Math.pow(10,l-i-1);
    // Поэтому преобразуем цифры  k из слова в частичные суммы
    // k×10^i
    // Функция Math.pow(x,y) вычисляет в Java число x^y
   }
  System.out.println("The word  "+slovo+ " was read with value
"+ j);
  // Результирующее число j выводим на консоль

 }
}
```

В этой программе мы использовали цикл `for` и команду `switch`.

Цикл `for` применяется для повторения одного и того же блока операций (тела) фиксированное число раз:

```
   for(i=0;i<=(l-1); i++)
{ тело цикла}
```

Переменная цикла `i` определяет номер итерации. Начальное значение `i` задает оператор `i=0`.

Условие `i<=(l - 1)` проверяется перед каждым очередным прохождением цикла. Если условие истинно, то итерация цикла выполняется, если нет, то происходит выход из цикла.

Оператор `i++` увеличивает номер переменной цикла на 1 после завершения очередной итерации. Наш цикл имеет такой вид:

```
for(i=0;i<=(l-1); i++)
// В этом цикле конструкция slovo.charAt(i) дает i-й символ
// слова
   {
      switch(slovo.charAt(i))
   {
     case '0': k=0; break; // если символ '0', то полагаем
                            // k=0;
     case '1': k=1; break; // если символ '1', то полагаем
                           // k=1;
     case '2': k=2; break; // если символ '2', то полагаем
                           // k=2;
     case '3': k=3; break; // если символ '3', то полагаем
                           // k=3;
     case '4': k=4; break; // если символ '4', то полагаем
                           // k=4;
     case '5': k=5; break; // если символ '5', то полагаем
                           // k=5;
     case '6': k=6; break; // если символ '6', то полагаем
                           // k=6;
     case '7': k=7; break; // если символ '7', то полагаем
                           // k=7;
     case '8': k=8; break; // если символ '8', то полагаем
                           // k=8;
     case '9': k=9; break; // если символ '9', то полагаем
                           // k=9;

      default : k=0;break;
    }
```

Тело цикла начинается с оператора `switch(slovo.charAt(i))`. Этот оператор применяется для проверки значения `i`-го разряда слова `slovo`. Таким разрядом должна быть по нашему замыслу десятичная цифра 0..9. Какая именно цифра — устанавливает оператор `case`. Например, если цифра равна `'5'` (в символьном представлении), то срабатывает оператор:

```
case '5' : k=5; break;
```

Следует обратить внимание на то, что `break` используется в операторе `case` для выхода из тела `switch() {}`, а не из цикла `for`, в котором использована команда `switch`. Таким образом, команда `switch` позволяет присвоить переменной `k` значение (в числовом формате) `i`-го разряда слова `slovo`, содержащего цифру в символьном формате (запомните, что `'5'` и `5` — это не одно и то же!).

Теперь нетрудно понять логику нашего цикла `for`: на каждой итерации `i` читается очередной `i`-ый разряд слова `slovo`, выполняется его преобразование в числовое представление и к `j` добавляется величина `k*Math.pow(10,l-i-1)`:

```
j=j+k*Math.pow(10,l-i-1);
```

в соответствии с представлением десятичного числа, например: `123=1*102 + 2*101 +3*100`. Команда `Math.pow(10,l-i-1)` вычисляет 10^{l-i-1}.

Наконец, оператор `if` проверяет условие таким схематическим образом:

```
if (условие)
   { блок команд 1}
else
   {блок команд 2}
```

Проверяется условие, указываемое в скобках после слова `if`. Если оно истинно, то выполняется блок команд 1, иначе выполняется блок команд 2. С помощью оператора `if` можно проверять сложные логические условия. Например, следующий оператор:

```
if( x>0) && (y>0)
  { A}
else
  { B}
```

проверяет два условия: `x>0` и `y>0`. Связка `&&` соответствует логической операции И. Если оба условия истинны, то выполняется блок A, иначе (одно или оба условия ложны) — блок B.

Следующий оператор:

```
if( x>0) || (y>0)
```

```
  { A}
else
  { B}
```

выполняет проверку на истинность хотя бы одного условия: `x>0` или `y>0`. Связка `||` соответствует логической операции ИЛИ. Сложное условие, которое связывает с помощью связки ИЛИ несколько простых условий, истинно тогда и только тогда, когда истинно хотя бы одно простое условие.

В заключение этого раздела мы приводим расширенный текст программы (листинг 1.4), в результате выполнения которой считываются два целых положительных числа и возвращается их наибольший общий делитель. Пусть эти числа — A и B. Алгоритм такой:

```
Считать A;
Преобразовать A в число AA;
Считать B;
Преобразовать B в число BB;
while (AA !=BB)
 {
  if (AA>BB)
   AA=AA-BB;
  else
   if BB>AA
    BB=BB-AA;
 }
Вывод на консоль AA;
```

Поскольку основные моменты были рассмотрены выше, то приводим код программы без комментариев.

Листинг 1.4. Отыскание наибольшего общего делителя двух чисел

```
import java.awt.*;
import java.io.*;
```

```
public class proba
{
 static int b;
 static String slovo="";
 static char[] charray;
 static char[] charray1;
 public static void main(String args[])
{
charray = new char[20];
int i=0;
System.out.println("Input a first number ->");
try{
  while((b=System.in.read()) !=13)
{charray[i++]=(char) b;  }
 }
catch(IOException e)
{System.out.println("The error"+e);}
slovo=new String(charray);
double A=0;
double B=0;
int k=0;
int l=slovo.trim().length();
   for(i=0;i<=(l-1); i++)
   {
      switch(slovo.charAt(i))
   {
     case '0': k=0;break;
     case '1': k=1;break;
     case '2': k=2; break;
     case '3': k=3;break;
     case '4': k=4; break;
     case '5': k=5; break;
     case '6': k=6;break;
```

```
     case '7': k=7; break;
     case '8': k=8;break;
     case '9': k=9; break;

      default : k=0;break;
    }
    A=A+k*Math.pow(10,l-i-1);}

charray1 = new char[20];
i=0;
System.out.println("Input a second number");
try{
  while((b=System.in.read()) !=13)
{charray1[i++]=(char) b;  }
 }
catch(IOException e)
{System.out.println("The error"+e);}
String slovo1=new String(charray1);
  k=0;
 l=slovo1.trim().length();
  slovo1=slovo1.trim();
   for(i=0;i<=(l-1); i++)
   {
      switch(slovo1.charAt(i))
     {
      case '0': k=0; break;
      case '1': k=1; break;
      case '2': k=2; break;
      case '3': k=3; break;
      case '4': k=4; break;
      case '5': k=5; break;
      case '6': k=6; break;
      case '7': k=7; break;
```

```
      case '8': k=8; break;
      case '9': k=9; break;
      default : k=0;break;
    }
  B=B+k*Math.pow(10,l-i-1);}
  System.out.println("first is:"+ A);
  System.out.println("second is:"+ B);
   while (A!=B)
    {
   if (A>B)
      A=A-B;
   else
    if (B>A)
      B=B-A;
  else break;
}
    System.out.println("Common devisor is: "+ A);//Вывод
   // наибольшего общего делителя чисел A, B
}
}
```

Мы рассмотрели побайтовый ввод символов с клавиатуры. Однако имеется и более простая возможность. Для ее реализации следует воспользоваться стандартным классом BufferedReader (как видите, классы навязчиво предлагают себя в практике программирования). Следующий пример (листинг 1.5) осуществляет ввод строки, считываемой с клавиатуры с помощью класса BufferedReader.

Листинг 1.5. Использование класса BufferedReader

```
import java.awt.*;
import java.io.*;
public class keyb
```

```
{
 static String slovo="";
 public static void main(String args[])
{
// Создаем переменную keyb1 для чтения строки с клавиатуры;
BufferedReader keyb1 = new BufferedReader(new
InputStreamReader(System.in));
// Класс BufferedReader предоставляет возможность читать
// строки с помощью метода readLine();
System.out.println("What is Your name ?");// Просим ввести имя
try{
slovo=keyb1.readLine(); } // Читаем строку slovo с клавиатуры;
catch (Exception e)
 {}
//Вывод приветствия на консоль
System.out.println("Hellow, fellow-programmer  "+slovo); try{
System.in.read();
  }
catch(Exception e)
  {}
}
}
```

Итак, программирование без классов обладает сравнительно небольшими возможностями и предполагает следующее: умение ввести исходные данные, обработать их нужным образом и вывести результат. Но даже ввод чисел заставляет нас прибегать к особым приемам, так как при нажатии на клавишу читается байт кода этой клавиши. Все считываемые байты можно помещать в массив, как мы и делали, после чего из массива строить либо числа, либо строки, либо переменные иных типов.

Классы

Классы являются основой объектно-ориентированного программирования. Физически классу соответствует некоторая сущность.

Например, можно рассматривать класс автомобилей, класс водителей, класс дорожных знаков, класс бездомных животных и т. п. Для того чтобы что-то можно было делать с классом, его следует описать. Схематически это можно представить таким образом:

```
class автомобиль
 {
   описание свойств класса автомобиль
   описание методов класса автомобиль
 }
```

Свойствами класса `автомобиль` могут быть стоимость, марка, тип, срок эксплуатации. Каждое свойство для различных автомобилей различное, но оно принадлежит некоторому диапазону значений. Диапазон значений называется в программировании *типом*. С учетом сказанного можно несколько детализировать описание класса `автомобиль`.

```
class автомобиль
 {Число стоимость;
  Строка марка;
  Строка тип;
  Число срок;
  описание методов класса автомобиль
 }
```

Допустим, в классе `автомобиль` определен метод `show()`, который выдает на дисплей информацию о стоимости и марке автомобиля:

```
show()
 {
  выдать на дисплей <марка>;
  выдать на дисплей <стоимость>;
}
```

Теперь *определение класса* получает следующий вид:

```
class автомобиль
```

```
{
 Число стоимость;
 Строка марка;
 Строка тип;
 Число срок;
 show()
 {
   выдать на дисплей <марка>;
   выдать на дисплей <стоимость>;
 }
}
```

Разумно также определить метод для установки начальных значений всех свойств. Такой метод по терминологии, принятой в объектном программировании, называется *конструктором*. Название конструктора совпадает с названием класса. Его можно было бы определить так:

```
автомобиль (s,m,t,srok)
{
 стоимость = s;
 марка = m;
 тип = t;
 срок = srok;
}
```

Объединяя все сказанное о классе `автомобиль`, получим следующее его описание:

```
class автомобиль
{
 Число стоимость;
 Строка марка;
 Строка тип;
 Число срок;
 show()
```

```
 {
  выдать на дисплей <марка>;
  выдать на дисплей <стоимость>;
 }
 автомобиль (s,m,t,srok)
 {
  стоимость = s;
  марка = m;
  тип = t;
  срок = srok;
 }
}
```

Теперь вы могли бы поинтересоваться, что же делать с этим описанием? Ответ таков: на основании этого описания можно построить один, два или сотню объектов, которые являются индивидуальными *экземплярами класса* `автомобиль`. С каждым из созданных экземпляров класса (автомобилем) можно работать индивидуально, т. е. устанавливать значения свойств, просматривать их на дисплее. Создание нового автомобиля реализуется, например, таким образом:

```
автомобиль myauto = new автомобиль(3000,
"opel","дизель",10);
```

Можно создать еще один автомобиль таким же образом:

```
автомобиль myauto2 = new автомобиль(3500,
"audi","дизель",7);
```

Переменные `myauto` и `myauto2` называются *объектными переменными*. Каждая из этих переменных получила значения свойств при выполнении конструктора. Можно воспользоваться методом `show()`, чтобы показать, каковы эти значения — например, выполнить команду:

```
myauto2.show();
```

Эта команда должна вывести на экран дисплея информацию об индивидуальном автомобиле `myauto2`.

На основании класса `автомобиль` можно построить другой класс, например, `автомобиль_на_продажу`. Мы хотели бы, чтобы

в этом классе присутствовало свойство `продажная_цена` и свойство `покупатель`, а также новый метод — `продан()`, который возвращает значение "продан", если свойство `покупатель` имеет не пустое значение.

Новый класс наследует *свойства* и *методы класса* `автомобиль` и добавляет новые свойства и методы. Это делается следующим образом:

```
class автомобиль_на_продажу extends автомобиль
{
 Число продажная_цена;
 Строка    покупатель;
 продан ()
 {
  если покупатель==""
  return "продан";
  иначе
   return "не продан";
 }
}
```

Для данного класса используется *конструктор класса* `автомобиль`. Но этот конструктор не определяет значения свойств `покупатель` и `продажная_цена`. Поэтому, вообще говоря, нам следует позаботиться о написании нового конструктора, например:

```
автомобиль_на_продажу (s,m,t,srok,pc,pok)
{
 автомобиль(s,m,t,srok); // вызываем родительский конструктор
 продажная_цена=pc;
 покупатель=pok;
}
```

Обратим внимание на то, что в новом конструкторе нам пришлось вызвать конструктор класса `автомобиль`. В Java *дочерний*

класс при использовании собственного конструктора должен вызывать конструктор *родительского класса* с помощью команды `super(список аргументов)`. Таково техническое решение, предложенное разработчиками языка. Хорошо оно, или плохо — нам выбирать не приходится.

Заметим, что хотя свойства `стоимость`, `марка`, `тип`, `срок` не определены в классе `автомобиль_на_продажу`, в конструкторе мы явно ссылаемся на эти свойства. Таким образом, мы рассмотрели одно из важнейших свойств класса — наследование. Также классы обладают еще такими свойствами, как полиморфизм и инкапсуляция, о чем более подробные сведения можно почерпнуть из [6].

Следует ли уничтожать создаваемые объекты? Вообще говоря — да, однако Java снимает эту обязанность с программиста. Java "понимает", когда созданный объект больше не требуется, и удаляет его из памяти. Это одно из больших достоинств Java.

Теперь мы перейдем к рассмотрению синтаксиса классов в языке Java.

Начнем с примера приложения (листинг 1.6), демонстрирующего использование классов.

Листинг 1.6. Демонстрация использования классов

```
import java.awt.*;
import java.awt.event.*;
class Book
{
 String title;
 int  price;
 Book(String s, int i)
 {
  this.title=s;
  this.price=i;
 }
}
```

```
public class lab1 extends Frame implements ActionListener
{
 Book book;
 Button b1=new Button("Exit");
 Button b2=new Button("Create Object");
 Button b3=new Button("Show Object");
 public lab1()
 {
  setLayout(null);
  add(b1);
  add(b2);
  add(b3);
  b1.addActionListener(this);
  b2.addActionListener(this);
  b3.addActionListener(this);
  b1.setBounds(20,20,100,20);
  b2.setBounds(20,50,100,20);
  b3.setBounds(20,80,100,20);
  setBackground(new Color(120,200,120));
 }
 public void actionPerformed (ActionEvent e)
 {
  if (e.getSource()==b1)
   System.exit(0);
  else
   if (e.getSource()==b2)
    {
     if (book == null)
      book=new Book("Pinoccio",200);
     else
      System.out.println("The object already exists");
    }
```

```
   else
    if (e.getSource()==b3)
     {
      if(book!=null)
      {
       Graphics g= getGraphics();
       g.drawString(book.title,20,200);
      }
     }
 }
public static void main(String [] args)
 {
  lab1 f= new lab1();
  f.resize(500,500);
  f.setVisible(true);
 }
}
```

В этом примере первым объявлен класс `Book`:

```
class Book
{
 String title;
 int price;
 Book(String s, int i)
 {
  this.title=s;
  this.price=i;
 }
}
```

В данном классе объявлены два члена (переменные) класса — `title` и `price`. Переменная `title` имеет тип `String` (строковый), переменная `price` имеет тип `int` (простой целочисленный тип). Таким образом, создавая объект данного класса, мы

рассчитываем, что он будет иметь эти два свойства: `title` и `price`. Вот то место в программе, где мы создаем объект класса `Book`:

```
{
 if (book == null)
  book=new Book("Pinoccio",200);
```

Переменная `book` — это не строка и не число. Она называется *объектной переменной*. Коротко — *объектом* (или *экземпляром класса*). Объект является чем-то реальным (соответствует чему-то реальному). В самом деле, конкретная книга есть некий реальный объект, который имеет название и цену. Можно указывать и другие свойства объекта-книги, но в нашем примере это не требуется.

В классе `Book` объявлен *конструктор* — метод, который используется при создании объекта. Оператор:

```
book=new Book("Pinoccio",200);
```

как раз и обращается к конструктору класса `Book`. Следует помнить, что конструктор имеет то же имя, что и сам класс. Для чего нужен конструктор? Для того, главным образом, чтобы инициализировать члены класса.

Инициализировать члены класса — значит, присвоить им какие-то начальные значения. Такими значениями в нашем примере являются название книги "Pinoccio" и цена книги 200. Само присваивание этих значений реализуется в теле конструктора следующим образом

```
Book(String s, int i)
{
 this.title=s;
 this.price=i;
}
```

Идентификатор `this` вообще говоря, избыточен. Он указывает на тот объект, который инициализируется в конструкторе. Не забываем, что конструктор может использоваться для создания

многих объектов. Поскольку вызов конструкторов — частое явление, то запомним, как это схематически реализуется:

```
имя_объектной_переменной=new
имя_класса_объектной_переменной(параметры);
```

Итак, описание класса `Book` завершено. Вторым классом является класс `lab1`. Этот класс объявлен как `public`. Только один класс в приложении Java может содержать метод `main()` и должен быть объявлен как `public`. В этом классе помещается *точка входа* во все приложение — метод с именем `main()`. Файл, в котором необходимо сохранить данное приложение, должен называться lab1.java — именно так, как называется класс `public`. В классе `lab1` объявляются объектная переменная `book` ранее описанного класса `Book` и программные кнопки (объекты класса `Button`):

```
Book book;
Button b1=new Button("Exit");
Button b2=new Button("Create Object");
Button b3=new Button("Show Object");
```

При этом кнопки не только объявляются, но и создаются в окне приложения (теперь мы будем всегда рассматривать конструкцию с ключевым словом `new` как обращение к конструктору).

В классе `lab1` также имеется конструктор:

```
public lab1()
{
 setLayout(null);
 add(b1);
 add(b2);
 add(b3);
 b1.addActionListener(this);
 b2.addActionListener(this);
 b3.addActionListener(this);
 b1.setBounds(20,20,100,20);
 b2.setBounds(20,50,100,20);
 b3.setBounds(20,80,100,20);
 setBackground(new Color(120,200,120));
}
```

Этот конструктор добавляет кнопки на форму:

```
add(b1);
add(b2);
add(b3);
```

устанавливает их размеры и места расположения:

```
b1.setBounds(20,20,100,20);
b2.setBounds(20,50,100,20);
b3.setBounds(20,80,100,20);
```

включает для них *прослушиватель событий*:

```
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
```

Благодаря прослушивателю событий можно для программных кнопок написать код обработки щелчка мышью.

В классе `lab1` объявлены три метода:

- конструктор `lab1()`;
- обработчик событий `public void actionPerformed (ActionEvent e)`;
- главный метод класса, с которого начинается выполнение всей программы: `main()`.

Созданный файл (например, в программе Блокнот) следует сохранить как lab1.java.

Компиляция этого файла реализуется из командной строки MS-DOS:

```
javac lab1.java
```

Чтобы ввести эту команду, например, из программы NC (Norton Commander), наберите в командной строке `>cmd`.

Программа `javac` — это и есть компилятор Java-программ. Если ошибок нет, то будут построены классы `lab1.class` и `Book.class`. Все эти файлы будут размещены в одном каталоге.

Теперь выполним приложение, введя в командной строке:

```
java lab1
```

Увидим такую форму (рис. 1.4).

Рис. 1.4. Визуальный интерфейс программы, код которой представлен в листинге 1.6

На форме показан результат нажатия кнопок **Create Object** и **Show Object**.

Спросим, почему, собственно, появилось окно? Рассмотрим снова объявление класса `lab1`:

```
public class lab1 extends Frame implements ActionListener
```

Слова `extends Frame` в этом объявлении означают, что класс `lab1` наследует открытые свойства и методы класса `Frame`. Класс `Frame` есть родитель класса `lab1`. В Java в качестве родителя может указываться только один класс или ни одного. Класс `Frame` как раз и представляет собой окно (форму). Так что ясно, что и `lab1` суть окно. Класс `Frame` стандартный. Это значит, что в дистрибутиве `Java` поставляется пакет, содержащий описание данного класса. Такие пакеты подключаются к программе в самом ее начале командой `import`. Оконные классы подключаются через пакеты `java.awt.*`. Выполним программу, предварительно убрав строку:

```
import java.awt.*;
```

Система выдаст 23 сообщения об ошибке (у вас, возможно, их будет и не 23). Знание стандартных классов очень важно, так

как они содержат огромное число полезных методов и свойств, без которых пришлось бы писать очень длинные программы. Мы еще вернемся к использованию классов чуть позже в этой главе. Общие принципы использования классов можно найти в [6].

Объявление переменных и методов, использование их в программе

После того как ситуация с классами прояснилась, обратимся к переменным и методам класса. Переменные используются для хранения значений, а методы, в общем случае, — для выполнения преобразований, вычислений, поиска, сохранения, а также других целей. Начнем рассмотрение с метода `main()`, с которого стартует приложение:

```
public static void main(String [] args)
{
 lab1 f= new lab1();
 f.resize(500,500);
 f.setVisible(true);
}
```

В метод `main()` всегда передаются аргументы командной строки — массив `args`. Далее следующая строка уже может быть нами объяснена:

```
lab1 f= new lab1();
```

Здесь создается объект с именем `f` — экземпляр класса `lab1`. Естественно, что обращение

```
new lab1()
```

используется для вызова конструктора. Конструктор данного класса как раз и создает форму, которую мы видим на рис. 1.4.

Члены класса должны быть объявлены и созданы. Объявления — это строки наподобие следующих:

```
Book book;
String title;
int price;
```

Можно одновременно объявлять и создавать члены класса:

```
Button b1=new Button("Exit");
Button b2=new Button("Create Object");
Button b3=new Button("Show Object");
```

При этом следует иметь в виду, что с помощью конструктора создаются только объектные переменные, а простые переменные наподобие `int` не имеют конструкторов вовсе. Однако в Java есть тип `Integer`, который позволяет создавать объектные целочисленные переменные. Отличие его от `int` заключается в том, что `Integer` предоставляет целое множество методов для работы с целыми числами, а `int` таких методов не предоставляет. Класс `String` — это класс, позволяющий создавать объектные строковые переменные. Для последующего запомним, что в Java типы, название которых начинается с большой буквы, являются *объектными*; типы, название которых начинается с малой буквы, являются *простыми*.

Приведем названия ряда важнейших типов языка Java:

`Boolean`	простой логический тип
`char`	простой символьный тип
`byte`	байт
`short`	короткое целое
`int`	32-разрядное целое
`long`	64-битовое целое число со знаком
`float`	вещественное число с плавающей точкой
`double`	вещественное число с плавающей точкой удвоенной точности
`ActionEvent`	класс события и действия
`Applet`	класс апплета — окна, отображаемого браузером
`Boolean`	составной логический тип
`Button`	кнопка
`Canvas`	графический класс для рисования
`Character`	составной символьный тип
`CheckBox`	флажок

`Choice`	выпадающий список
`Color`	цвет
`Connection`	соединение (сетевое)
`Container`	контейнер
`Date`	дата + время
`DataInputStream`	высокоуровневый класс для чтения данных
`DataOutputStream`	высокоуровневый класс для вывода данных
`Dialog`	диалоговое окно
`Dimension`	размеры экрана
`Event`	класс события (например, нажатия кнопки мыши)
`File`	класс для доступа к файлам
`FileDialog`	окно диалога при выборе файла
`FileInputStream`	низкоуровневый класс для чтения файла
`FileOutputStream`	низкоуровневый класс для записи в файл
`Font`	шрифт
`Frame`	окно приложения (форма)
`Graphics`	графический класс
`GridLayout`	класс для размещения элементов в окне в матрице
`HashTable`	ассоциативная таблица базы данных
`HttpRequest`	класс для чтения данных HTML-документа
`HttpResponse`	класс для вывода документа клиенту
`Image`	рисунок
`ImageObserver`	класс для просмотра и загрузки рисунков
`InetAddress`	адрес компьютера в Интернете
`Integer`	составной класс целых чисел
`KeyEvent`	класс событий от клавиатуры

`Label`	ярлык
`List`	список
`Menu`	элементы линейного меню
`MenuBar`	панель меню
`MenuItem`	элементы выпадающего меню
`MouseEvent`	класс событий от мыши
`Object`	класс объектов
`Panel`	панель
`Point`	точка в окне
`Rectangle`	прямоугольная область в окне
`ServerSocket`	порт на стороне сервера
`Socket`	порт
`Statement`	инструкция SQL
`String`	строковый класс
`StringBuffer`	буфер символов
`StringTokenizer`	класс для выделения лексем
`TextField`	текстовое поле
`TextArea`	текстовое окно
`Thread`	поток
`Vector`	вектор
`URL`	(universal resource locator) адрес ресурса в Интернете

Переменные составного (объектного) типа обладают методами своего класса. Так, переменную `title` класса `String` можно вывести большими жирными красными буквами следующим образом (рис. 1.5).

```
{
 if(book!=null)
 {
  Graphics g= getGraphics();
  Font fnt=new Font("Courier",Font.BOLD,24);
```

```
  g.setFont(fnt);
  setForeground(new Color(250,0,0));
  g.drawString(book.title.toUpperCase(),20,200);
 }
}
```

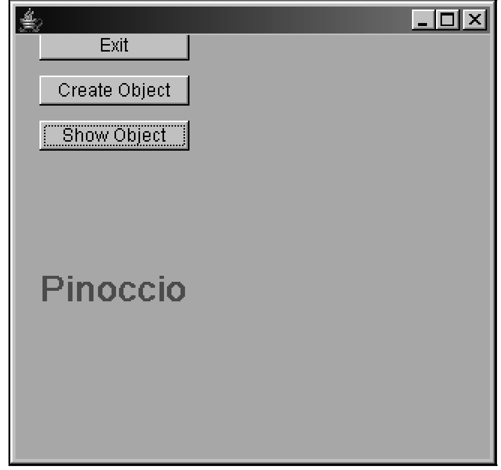


Рис. 1.5. Использование методов классов `Graphics` и `Font`

Вывод строки реализует оператор

```
g.drawString(book.title.toUpperCase(),20,200);
```

Здесь `g` — графическая объектная переменная класса `Graphics`. В классе `Graphics` есть метод для вывода строки — `drawString()`. Поэтому конструкция `g.drawString()` выполняет вывод строки, помещенной в круглые скобки. Сама строка, выводимая этим оператором, такова:

```
book.title.toUpperCase().
```

Здесь `book` — имя объектной переменной, созданной из класса `Book`. У этой переменной есть свойство `title` (член класса). Но `title` относится к классу `String`, а у класса `String` есть метод `toUpperCase()`, переводящий буквы строки к заглавному виду. Следовательно, вся конструкция `book.title.toUpperCase()` дает переменную `title` объекта `book` класса `Book` в форме заглавной

строки. Прежде чем выводить строку, создаем новый шрифт (`font`) командой

```
Font fnt=new Font("Courier",Font.BOLD,24);
```

а командой

```
g.setFont(fnt);
```

устанавливаем этот шрифт на форме. Конструктор класса `Font` использует название шрифта, тип и размер соответственно в первом, втором и третьем параметре.

Теперь о красном цвете. Мы его получаем оператором

```
setForeground(new Color(250,0,0));
```

либо оператором

```
g.setColor(new Color(250,0,0));
```

(Заметим, что в классе `Graphics` нет метода `setForeground()`, который определен в классе `Frame`.)

Конструктор класса `Color` использует комбинацию трех цветов: красного, зеленого и голубого для получения нужного цвета. Значение каждого из цветов передается целым числом от 0 до 255 (0 соответствует отсутствию цвета, 255 — максимальному уровню насыщения). Оператор для установки цвета шрифта — это `setForeground()`. Перед ним не записано имя объектной переменной! Поэтому остается предположить (и это на самом деле так), что данный оператор относится либо к объекту `this`, либо к какому-то классу. Чтобы проверить, какая гипотеза верна, запишем так:

```
{
 if(book!=null)
 {
  Graphics g= getGraphics();
  this.setForeground(new Color(250,0,0));
  g.drawString(book.title.toUpperCase(),20,200);
 }
}
```

и выполним программу. Никаких ошибок компилятор не выдаст. Следовательно, опущено слово `this`, которое фактически указывает на тот объект, который выполняет данный оператор.

Этим объектом является объект `f` класса `lab1`, созданный в методе `main()`:

```
public static void main(String [] args)
{
 lab1 f= new lab1();
 f.resize(500,500);
 f.setVisible(true);
}
```

ибо метод `actionPerformed`, в котором выполняется оператор

```
this.setForeground(new Color(250,0,0));
```

как раз и определяется в классе `lab1`.

Однако следует иметь в виду, что если название метода записано без ссылки на объектную переменную, то данный метод может принадлежать классу, а не объекту. Такие методы называются статическими (определяются словом `static`). Например, видим такой метод в нашей программе — `main()`. Изменим этот метод:

```
public static void main(String [] args)
{
 lab1 f= new lab1();
 setForeground(new Color(250,0,0));
 f.resize(500,500);
 f.setVisible(true);
}
```

Компилятор отреагирует на это сообщением об ошибке (рис. 1.6).

```
C:\Sun\AppServer\jdk\bin>javac lab1.java
lab1.java:176: non-static method setForeground(java.awt.Color) cannot be
 static context
     setForeground( new Color(0,0,250));
     ^
Note: lab1.java uses or overrides a deprecated API.
Note: Recompile with -deprecation for details.
1 error

C:\Sun\AppServer\jdk\bin>_
1Левая  2Правая 3Смотр. 4Редак. 5Печать 6Связь  7Искать 8Истор  9Видео
```

Рис. 1.6. Неверное использование нестатического метода

Сообщение будет содержать слова о том, что нестатический метод `setForeground()` адресуется из статического контекста. Иначе говоря, все тело статического метода `main` рассматривается как *статический контекст*. Теперь запомним, что в статическом контексте ссылка `this` бессмысленна. А коль так, то методы, используемые в статическом контексте без ссылки на объектную переменную, должны быть объявлены как статические (`static`). Для исправления ошибки достаточно написать так:

```
public static void main(String [] args)
{
 lab1 f= new lab1();
 f.setForeground(new Color(250,0,0));
 f.resize(500,500);
 f.setVisible(true);
}
```

Теперь рассмотрим *модификаторы доступа* к переменным, методам и классам: `public`, `protected`, `private`. Модификатор `public` указывает на общедоступность метода, переменной или класса. Модификатор `protected` разрешает доступ к переменной, методу или классу только из родительского или дочернего класса. Наконец, модификатор `private` ограничивает доступ к переменной, методу только классом, где она объявлена. Мы знаем теперь, что непосредственно обратиться к переменной или методу класса можно, если эта переменная или метод объявлены как статические. Доступ к таким переменным и методам регулируется через модификаторы `public`, `protected`, `private`. Кстати, отсутствие модификатора доступа указывает, что класс доступен в пределах активного каталога, из которого запущено приложение. Однако если создаем объект некоторого класса `ClassA` и этот класс доступен из того места `ClassB`, где мы создаем объект, то получаем доступ ко всем переменным, методам класса `ClassA`.

В заключение этого подраздела рассмотрим, как передаются аргументы в процедуры. У нас есть обращение к конструктору:

```
if (e.getSource()==b2)
```

```
 {
  if (book == null)
   book=new Book("Pinoccio",200);
```

Именно в строке `new Book("Pinoccio",200);` конструктору передаются два аргумента: строка "Pinoccio" и число 200. Аргументы должны разделяться запятыми. Аргументы передаются либо непосредственно (как в нашем случае), либо передаются переменные, которым предварительно присвоены нужные значения, т. е.:

```
if (e.getSource()==b2)
 {
  if (book == null)
   {
    String s="Pinoccio";
    int z= 200;
    book=new Book(s,z);
   }
```

В объявлении конструктора следует указать типы аргументов:

```
Book(String s, int i)
{
 this.title=s;
 this.price=i;
}
```

Переменная `s` объявлена как `String`, переменная `i` объявлена как `int`.

Метод может возвращать или не возвращать значения. Если метод не возвращает никакие значения, то в его объявлении следует указать ключевое слово `void`, например:

```
public static void main().
```

Но методы могут возвращать значения, тогда нужно указать в объявлении метода тип возвращаемого им значения. Поясним этот момент примером. Добавим в класс `lab1` новый метод, который возвращает значение свойства `title` объекта `book`.

```
public String showTitle(Book z)
{
 return z.title;
}
```

Этот метод поместим в тело класса `lab1`, как показано далее:

```
public class lab1 extends Frame implements ActionListener
{
 Book book;
 Button b1=new Button("Exit");
 Button b2=new Button("Create Object");
 Button b3=new Button("Show Object");
 public lab1()
 {
  setLayout(null);
  add(b1);
  add(b2);
  add(b3);
  b1.addActionListener(this);
  b2.addActionListener(this);
  b3.addActionListener(this);
  b1.setBounds(20,20,100,20);
  b2.setBounds(20,50,100,20);
  b3.setBounds(20,80,100,20);
  setBackground(new Color(120,200,120));
 }
 // Здесь вставлено описание и объявление нового метода
 // showTitle()
 public String showTitle(Book z)
 {
  return z.title;
 }
 public void actionPerformed (ActionEvent e)
```

```
 {
  if (e.getSource()==b1)
   System.exit(0);
  else
   if (e.getSource()==b2)
    {
     if (book == null)
      book=new Book("Pinoccio",200);
     else
      System.out.println("The object already exists");
    }
   else
    if (e.getSource()==b3)
     {
      if(book!=null)
      {
       Graphics g= getGraphics();
       g.drawString(this.showTitle(book),20,200);
       // здесь мы обратились к новому методу
      }
     }
 }
 public static void main(String [] args)
 {
  lab1 f= new lab1();
  f.resize(500,500);
  f.setVisible(true);
 }
}
```

Методом `showTitle()` мы воспользовались в обработчике события, связанного с нажатием кнопки в окне приложения. Данный метод возвращает строковое значение, поэтому в объявлении этого метода указано ключевое слово `String`. Особенностью передачи в метод в качестве аргумента объекта некоторого класса является то, что фактически передается ссылка на объект. Поэтому любые

изменения членов объекта в методе проводятся на исходном объекте. Поясним сказанное следующим фрагментом кода:

```
import java.awt.*;
import java.io.*;
public class peredaczaobj
{
 int b;
 public void method(peredaczaobj x)
 {
  x.b=5; // Этот оператор изменяет значение b переданного
         // объекта
 }
 public static void main(String args[])
 {
  peredaczaobj ob=new peredaczaobj();
  ob.b=1; // Начальное значение члена класса
  System.out.println("Before method:"+ob.b);
  ob.method(ob);
  System.out.println("After method:"+ob.b);
 }
}
```

При запуске этого класса на консоль будет выведено:

```
Before method:1
After method:5.
```

Создание визуального интерфейса

Под *визуальным интерфейсом* понимается окно с размещенными на нем элементами — кнопками, текстовыми полями, меню, списками и пр. В предыдущем примере мы размещали кнопки. Это делалось в конструкторе так:

```
public lab1()
{
 setLayout(null);
```

```
 add(b1);
 add(b2);
 add(b3);
 b1.addActionListener(this);
 b2.addActionListener(this);
 b3.addActionListener(this);
 b1.setBounds(20,20,100,20);
 b2.setBounds(20,50,100,20);
 b3.setBounds(20,80,100,20);
 setBackground(new Color(120,200,120));
}
```

Кнопки добавляются так:

```
add(b1);
add(b2);
add(b3);
```

Позиции размещения кнопок и их размеры задаются так:

```
b1.setBounds(20,20,100,20);
b2.setBounds(20,50,100,20);
b3.setBounds(20,80,100,20);
```

Метод `setBounds(x,y,w,h)` использует в качестве аргументов `x`, `y` — координаты верхнего левого угла элемента на форме (апплете); `w` — ширина элемента в пикселах, `h` — его длина. В рассматриваемой программе мы сами размещали элементы в окне. Java имеет ***компоновщики*** для автоматического размещения элементов. Компоновщик включается в программу оператором `setLayout`. В нашем случае использование автоматического компоновщика блокируется указанием параметра `null` в операторе `setLayout`. В качестве альтернативного примера приведем использование компоновщика `BorderLayout`. Этот компоновщик размещает всего пять элементов: в центре (`BorderLayout.CENTER`), справа (`BorderLayout.EAST`), слева (`BorderLayout.WEST`), сверху (`BorderLayout.NORTH`) и внизу (`BorderLayout.SOUTH`).

Вот как это следует сделать (приводим фрагмент класса `lab1` с конструктором):

```
public class lab1 extends Frame implements ActionListener
{
 Book book;
 Button b1=new Button("Exit");
 Button b2=new Button("Create Object");
 Button b3=new Button("Show Object");
 Label lb=new Label("INTRODUCTION in JAVA"); // добавлено
 TextField tf = new TextField(); // добавлено

public lab1()
{
 setLayout(new BorderLayout()); // изменено
 add(tf,BorderLayout.SOUTH); // добавлено
 add(b1,BorderLayout.WEST); // изменено
 add(b2,BorderLayout.CENTER); // изменено
 add(b3,BorderLayout.EAST); // изменено
 add(lb,BorderLayout.NORTH); // добавлено
 b1.addActionListener(this);
 b2.addActionListener(this);
 b3.addActionListener(this);
 b1.setBounds(20,20,100,20); // бесполезен
 b2.setBounds(20,50,100,20); // бесполезен
 b3.setBounds(20,80,100,20); // бесполезен
 setBackground(new Color(120,200,120));
}
```

В окно добавлены еще ярлык (`Label`) и текстовое поле (`TextField`). Результат изменения конструктора такой (рис. 1.7).

Теперь вывод строки "Pinoccio" осуществляется не непосредственно в окно, а в текстовое поле. С этой целью следует несколько изменить *обработчик события* от программной кнопки (`Button`):

```
public void actionPerformed (ActionEvent e)
```

```
{
 if (e.getSource()==b1)
  System.exit(0);
 else
  if (e.getSource()==b2)
   {
    if (book == null)
     book=new Book("Pinoccio",200);
    else
     System.out.println("The object already exists");
   }
  else
   if (e.getSource()==b3)
    {
     if(book!=null)
      {
       Graphics g= getGraphics();    // бесполезна
       // this.setForeground(new Color(250,0,0)); // удалено
       // g.drawString(book.title.toUpperCase(),20,200);
                                                   // удалено
       tf.setText(book.title.toUpperCase());      // добавлено
      }
     }
   }
```

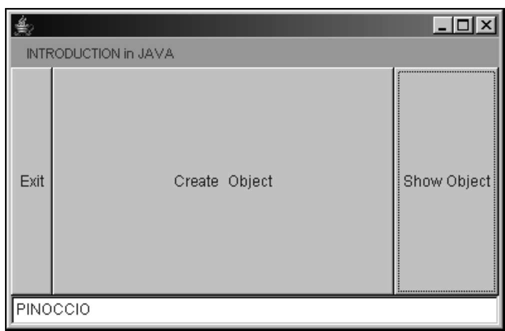


Рис. 1.7. Использование компоновщика `BorderLayout`

Переменная `tf` представляет объектную переменную типа текстового поля. Добавление в текстовое поле текста реализуется методом `setText()` класса `TextField`.

Итак, следует понимать, что создание *визуального интерфейса* выполняется на основе окна. В `Java` окнами являются объекты типа `Frame`, `Panel`, `Dialog` и `Applet`. Апплеты мы будем рассматривать позже. Это окна, которые отображает и обрабатывает программа Internet Explorer. Для добавления элементов в окно используется метод `add()` класса `Frame (Applet)`. Для добавления на `Panel` используется класс `Container`. Элементы должны быть размещены в окне, что выполняется пользователем или системой. Для использования автоматического укладчика его требуется создать. Мы делали это командой:

```
setLayout(new BorderLayout());
```

Имеются другие типы укладчиков, отличные от `BorderLayout`.

Программирование обработки событий от элементов, мыши и клавиатуры

Для того чтобы перехватывать события от мыши, нужно подключить *прослушиватель событий*. Для различных элементов используются различные прослушиватели событий. Для программных кнопок, текстовых полей и меню прослушиватели событий строятся на основе интерфейса `ActionListener`. Для выпадающих списков и флажков используется интерфейс `ItemListener`, для клавиатуры — `KeyListener`, для мыши — `MouseListener` и т. д. Подключение интерфейсов реализуется сразу же в объявлении класса. Так, в объявлении класса `lab1` подключаем интерфейс `ActionListener`:

```
public class lab1 extends Frame implements ActionListener
```

Мы добавляем прослушиватель с помощью слов `implements ActionListener`.

`ActionListener` — это *абстрактный класс* (интерфейс — это вид абстрактного класса, в котором методы только объявлены, но не определены. Абстрактный класс вообще — это такой класс, в

котором есть хотя бы один объявленный, но не реализованный метод). В `ActionListener` такой метод, подлежащий переопределению, — `actionPerformed()`. Его реализацию мы выполнили следующим образом:

```
public void actionPerformed (ActionEvent e)
{
 if (e.getSource()==b1)
  System.exit(0);
 else
  if (e.getSource()==b2)
  {
   if (book == null)
    book=new Book("Pinoccio",200);
   else
    System.out.println("The object already exists");
  }
  else
   if (e.getSource()==b3)
   {
    if(book!=null)
    {
     Graphics g= getGraphics();
     g.drawString(book.title,20,200);
    }
  }
}
```

Следует понимать, что объявления абстрактных классов вполне конкретные, так что попытка пользователя сделать свое объявление строго пресекается. Например, не пройдет объявление без `public`:

```
void actionPerformed (ActionEvent e)
{
 if (e.getSource()==b1)
```

```
  System.exit(0);
}
```

В метод `actionPerformed()` передается в качестве аргумента событие `e`, принадлежащее классу `ActionEvent`. Данный класс содержит метод `getSource()`, который дает возможность узнать, от какого источника пришло данное событие. Обратите внимание на то, как применяется метод `getSource()` в программе. Это его типичное использование. Запомним, что метод `actionPerformed()` используется для прослушивания событий от программных кнопок, меню, текстового поля (при нажатии клавиши <Enter>) и списка. Часто в программе возникает необходимость реагировать на события, связанные с нажатием клавиш на клавиатуре. Для прослушивания *событий от клавиатуры* следует подключить интерфейс `KeyListener`. В данном интерфейсе объявлены три метода: `keyPressed()`, `keyReleased()` и `keyTyped()`. Все эти три метода пользователь обязан переопределить, причем переопределение может быть пустым:

```
public void keyTyped(KeyEvent event)
{
}
```

Метод `keyPressed()` активизируется при нажатии на клавишу. Метод `keyReleased()` активизируется при отпускании клавиши. Метод `keyTyped()` активизируется также при нажатии на клавишу, но не позволяет получить значения клавиш-модификаторов, если таковые нажимались одновременно с клавишами. (К клавишам-модификаторам относятся клавиши <Ctrl>, <Shift>, <Alt> и др.). Для получения кода нажатой клавиши следует использовать метод `getKeyCode()` класса `KeyEvent`; для получения самой нажатой клавиши — метод `getKeyChar()`. Воспользуемся этими методами для выхода из программы. В разбираемом нами примере выход был реализован по нажатию на кнопку **Exit** (программное имя `b1`):

```
public void actionPerformed (ActionEvent e)
{
 if (e.getSource()==b1)
```

```
  System.exit(0);
 else
 ...
```

Завершение программы реализует оператор:

```
System.exit(0); // 0 указывается в качестве кода возврата
```

Теперь изменим программу следующим образом. Во-первых, в объявление класса `lab1` включим интерфейс `KeyListener`:

```
public class lab1 extends Frame implements
ActionListener,KeyListener
{
 Book book;
 Button b1=new Button("Exit");
 Button b2=new Button("Create Object");
 Button b3=new Button("Show Object");
 Label lb=new Label("INTRODUCTION in JAVA");
 TextField tf = new TextField();
 public lab1()
 {
   setLayout(null);
   add(tf);
   add(b1);
   add(b2);
   add(b3);
   add(lb);
   addKeyListener(this);
   b1.addActionListener(this);
   b2.addActionListener(this);
   b3.addActionListener(this);
   lb.setBounds(20,20,200,20);
   tf.setBounds(20,160,200,20);
   b1.setBounds(20,60,100,20);
   b2.setBounds(20,90,100,20);
```

```
   b3.setBounds(20,120,100,20);
   setBackground(new Color(120,200,120));
  }
 public String showTitle(Book z)
 {
  return z.title;
 }
 public void keyTyped(KeyEvent ke)
 {}

 public void keyReleased(KeyEvent ke)
 {}

public void keyPressed(KeyEvent ke)
 {
  if(ke.getKeyChar()=='a') // выход из приложения по вводу
                           // символа 'a'
   System.exit(0);
 }

 public void actionPerformed (ActionEvent e)
 {
  if (e.getSource()==b1)
   System.exit(0);
...
```

В конструкторе класса `lab1` мы добавили оператор подключения *прослушивателя событий от клавиатуры* для объекта этого класса:

```
addKeyListener(this);
```

Затем объявили в теле класса `lab1` методы для обработки событий от клавиатуры. Но чтобы ввод с клавиатуры символа `'a'` завершил программу, нужно, чтобы окно приложения имело фокус ввода, а для этого используется оператор `requestFocus()`. Мы вставили его в метод `main()`:

```
public static void main(String [] args)
```

```
{
 lab1 f= new lab1();
 f.setForeground( new Color(0,0,250));
 f.resize(500,500);
 f.setVisible(true);
 f.requestFocus();
}
```

Так что имеем в виду, что после развертывания окна приложения оно имеет фокус ввода и среагирует на нажатие клавиши <a>. Если начать щелкать кнопками, то окно приложения потеряет фокус ввода и реагировать на нажатие клавиши <a> перестанет.

Рассмотрим более общее приложение (листинг 1.7), содержащее флажки, выпадающий список, кнопки, ярлык (`Label`) и текстовое поле (`TextField`) на предмет использования прослушивателей событий.

Флажок — это элемент (маленькое квадратное окошко), в который при щелчке на нем мышью помещается символ галочки или крестика, свидетельствующий о том, что данный флажок установлен. Флажки можно объединять в группы, так что установка любого флажка из группы приводит к сбросу всех других флажков этой группы.

Выпадающий список — это список, который можно развернуть (открыть его элементы) и произвести выбор щелчком мыши любого из них.

Ярлык — это текстовое поле, в котором разрешается только отображать текст, но его нельзя редактировать.

Текстовое поле — это поле, допускающее редактирование текста.

В рассматриваемом приложении важно обратить внимание на использование прослушивателей событий.

Листинг 1.7. Пример использования прослушивателей событий

```
import java.awt.*;
import java.awt.event.*;
//Подключение прослушивателей:
public class Compn extends Frame implements ActionListener,
```

```
                    ItemListener,MouseListener
{
 Button exit=new Button("Exit");
 CheckboxGroup gr= new CheckboxGroup(); // Объявление группы
                                         // флажков
 Checkbox op1= new Checkbox("One"); // Объявление первого
                                     // флажка
 Checkbox op2= new Checkbox("Two"); // Объявление второго
                                     // флажка
 Choice ch= new Choice(); // Объявление выпадающего списка
 Label lb= new Label("YourChoice"); // Объявление ярлыка
 //Объявление текстового поля:
TextField tf = new TextField("To edit ....");
 Compn() // Конструктор
 {
  setLayout(null); // Размещаем элементы вручную
  add(exit); // Добавляем на форму кнопку
  add(ch); // Добавляем выпадающий список
  add(lb); // Добавляем ярлык
  add(tf); // Добавляем текстовое поле
  ch.add("Good"); // Добавляем элемент в выпадающий список
  ch.add("Very Good"); // Добавляем элемент в выпадающий
                         // список
  ch.add("Excellence");
  op1.setCheckboxGroup(gr); // Присваиваем флажку группу
  op2.setCheckboxGroup(gr); // Присваиваем второму флажку
                              // ту же группу
  add(op1); // Добавляем первый флажок
  add(op2); // Добавляем второй флажок
  op1.setBounds(10,80,100,20); // Задаем размеры и координаты
                                 // флажка
  op2.setBounds(120,80,100,20);
  ch.setBounds(10,110,60,70); // Задаем размеры и координаты
                                // списка
```

```
lb.setBounds(130,110,100,20); // Задаем размеры и координаты
                              // ярлыка
lb.setBackground(Color.white); // Устанавливаем белый фон
                               // ярлыка
 // Задаем размеры и координаты текстового поля:
 tf.setBounds(130,140,100,20);
 // Разрешаем редактирование текста в текстовом поле:
 tf.setEditable(true);
 tf.setBackground(Color.yellow); // Задаем фон текстового
                                 // поля
 // Добавляем прослушиватель выпадающего списка:
 ch.addItemListener(this);
 op1.addItemListener(this); // Добавляем прослушиватель
                            // флажка
 op2.addItemListener(this);
 // Добавляем прослушиватель текстового поля,
 // который реагирует на нажатие клавиши <ENTER>:
 tf.addActionListener(this);
 // Добавляем прослушиватель событий от ярлыка,
 // который реагирует на щелчок мышью:
 lb.addMouseListener(this);
 exit.addActionListener(this); // Добавляем прослушиватель
                               // для кнопки
 // Устанавливаем координаты и размеры кнопки:
 exit.setBounds(10,20,100,20);
}
// Переопределяем методы интерфейса MouseListener
public void mouseEntered(MouseEvent ev)
{
}

public void mouseExited(MouseEvent ev)
{
}
```

```
public void mousePressed(MouseEvent ev)
 {
 }

 public void mouseReleased(MouseEvent ev)
 {
 }

 public void mouseClicked(MouseEvent ev) // При щелчке мышью
                                          // на ярлыке
 // его содержимое изменяется и устанавливается из текстового
 // поля tf
 {
  if(ev.getSource()==lb)
   lb.setText(tf.getText());
 }

 public void itemStateChanged(ItemEvent ie) // Реакция на
                                             // выбор флажка
 {
  if(ie.getSource()==op1)
   {
    setBackground(Color.red);
   } // При выборе первого флажка изменяем цвет формы
     // на красный
  else
   if(ie.getSource()==op2)
    {setBackground(Color.green);
    } // При выборе второго флажка устанавливаем зеленый цвет
      // формы
   else
    if(ie.getSource()==ch) // При выборе элемента выпадающего
                           // списка
    // устанавливаем содержимое ярлыка равным выбранному
    // значению
```

```
    lb.setText(ch.getSelectedItem());
 }
 //Обработка событий от кнопки и текстового поля
 public void actionPerformed(ActionEvent e)
 {
  if(e.getSource()==exit)
   System.exit(0); //Выход из приложения
  else
   if(e.getSource()==tf)
    tf.setText(""); // Сброс содержимого текстового поля
                    // при нажатии <ENTER>
 }

 public static void main(String[] args)
 {
  Compn cm=new Compn();
  cm.resize(300,300);
  cm.setBackground(new Color(200,0,180));
  cm.setVisible(true);
 }
}
```

Подробные комментарии в коде программы разъясняют многие вопросы. Отметим, что содержимое ярлыка и текстового поля получаем командой `getText()`, а установку нового содержимого ярлыка и текстового поля осуществляем командой `setText()`. Значение выбранного элемента выпадающего списка получаем в строке:

```
ch.getSelectedItem());
```

Номер выделенного элемента списка возвращает метод `getSelectedIndex()`.

Обратим внимание также на то, что текстовое поле прослушивается с помощью интерфейса `ActionListener`, который распознает событие, связанное с нажатием клавиши <Enter>. Наконец, запомним следующее. Каждый ***прослушиватель событий***

требует определить все свои нереализованные методы. Прослушиватель `MouseListener` содержит пять таких методов:

- `public void mouseEntered(MouseEvent ev)`
- `public void mouseExited(MouseEvent ev)`
- `public void mousePressed(MouseEvent ev)`
- `public void mouseReleased(MouseEvent ev)`
- `public void mouseClicked(MouseEvent ev)`

При реализации нужно прописывать все методы интерфейса. Более удобная возможность предоставляется абстрактным классом `MouseAdapter`. Поясним использование этого класса следующим примером, снабженным всеми необходимыми комментариями.

```
import java.awt.*;
import java.awt.event.*;
class Adapter extends MouseAdapter
{
 // Использует класс MouseAdapter
 mousead x;
 Adapter(mousead md)
 // В конструктор передаем ссылку на форму mousead
 {
   x=md;
 }

 public void mousePressed(MouseEvent me)
 //Описываем только метод mousePressed()
 {
  x.setBackground(new Color((int)(
250*Math.random()),100,100));
  //при щелчке мышью на форме последняя изменяет фоновый цвет;
  //x - объект класса mousead
 }
}
public class mousead extends Frame
```

```
{
 mousead() // Конструктор основного класса
  {
   Adapter ad=new Adapter(this); // Подключение адаптера в
                                 // основной класс
   this.addMouseListener(ad); // Подключение прослушивателя
                              // от мыши
  }

 public static void main(String[] args) // Главный метод
приложения
  {
   mousead nn=new mousead();
   nn.resize(100,200);
   nn.show();
  }
}
```

Программирование ввода-вывода с использованием файлов

Рассмотрим еще один аспект работы в Java — *чтение из файла* и *запись в файл.* В совокупности с рассмотренными выше вопросами — это одна из важнейших сторон программирования в Java.

Ввод-вывод играет большую роль в системах обработки информации. Часто просто требуется ввести какую-нибудь строку или число.

Например, мы бы хотели запросить название книги `title` из окна диалога. Сделаем это. Нам потребуется создать новое окно для ввода информации, которое будет содержать текстовое поле и кнопку для закрытия окна. Данное окно следует объявить как отдельный класс на основе `Frame`:

```
class Dlgwin extends Frame implements ActionListener
{
 Book bb;
```

```
String param;
Button bexit=new Button("Exit From Dialog");
TextField tfi=new TextField();

public Dlgwin(Book z)
{
 bb=z;
 param=z.title;
 setLayout(null);
 add(bexit);
 add(tfi);
 bexit.addActionListener(this);
 tfi.setBounds(20,160,200,20);
 tfi.setText(param);
 bexit.setBounds(20,60,100,20);
 setBackground(new Color(200,200,100));
}

public void actionPerformed (ActionEvent e)
{
 if (e.getSource()==bexit)
  {
   bb.title= tfi.getText();
   this.setVisible(false);
  }
}
}
```

Наш новый класс содержит переменную типа `Book`. В нем многое очень похоже на класс `lab1`. В конструктор класса `Dlgwin` передаем объект `book` в виде объектной переменной `z`.

Оператор `bb=z` позволяет присвоить объектной переменной `bb` ссылку на объект `z`, поскольку в момент выполнения присваивания переменная `bb` не инициализирована. Мы используем переменную `bb` в обработчике события от мыши чуть ниже.

Остальные действия конструктора — добавление на форму кнопки и текстового поля и подключение к ним прослушивателя. *Обработчик события* от программной кнопки заносит в поле `title` объекта `bb` значение из текстового поля `tfi`. Команда

```
this.setVisible(false);
```

скрывает окно диалога. Итак, после вызова окна диалога пользователю нужно набрать в текстовом поле название книги (по умолчанию там отобразится "Pinoccio") и нажать кнопку **Exit from dialog**.

Теперь посмотрим, как вызвать диалог; создание соответствующей диалоговой переменной связываем с обработчиком события от кнопки **Create Object** класса `lab1`:

```
public void actionPerformed (ActionEvent e)
{
 if (e.getSource()==b1)
  System.exit(0);
 else
  if (e.getSource()==b2)
   {
    if (book == null)
     {
      book=new Book("Pinoccio",200); // создаем переменную
                                     // типа lab1
      dlg=new Dlgwin(book); // создаем переменную класса
                            // Dlgwin
      dlg.setSize(300,300);
      dlg.setVisible(true);    // отображаем переменную Dlgwin
     }
    else
     System.out.println("The object already exists");
...
```

Реализация этой части кода приведет к тому, что после запуска приложения и нажатия на кнопку **Create Object** откроется новое окно диалога (рис. 1.8).

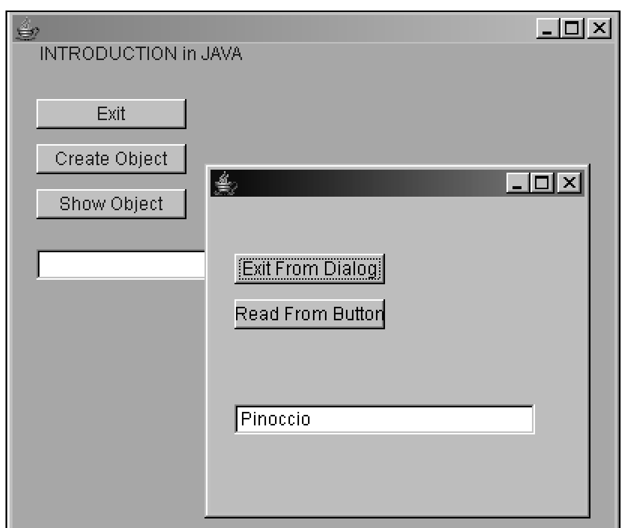


Рис. 1.8. Новое окно диалога

Нажатие на кнопку **Exit From Dialog** закроет *диалоговое окно*, но при этом в поле `title` объекта `book` будет записано содержимое текстового поля диалогового окна.

Весьма простой вывод текстовой информации реализует оператор:

```
JOptionPane.showMessageDialog(null,book.title);
```

Второй аргумент этого оператора определяет выводимую строку. Чтобы этот оператор сработал, нужно импортировать пакет директивой:

```
import javax.swing.*;
```

Теперь рассмотрим работу с файлами. Допустим, мы хотим, чтобы вводимое в окне диалога значение поля `tfi` также сохранялось и в файле. Для этого размещаем в диалоговом окне новую кнопку для чтения ранее сохраненной строки в файле. Эта строка читается прямо в текстовое поле `tfi`. Сохранение строки в файле выполняется при нажатии на кнопку **Exit** в диалоговом окне. Все это реализуется в новом классе `Dlgwin` (листинг 1.8).

Листинг 1.8. Определение диалогового класса `Dlgwin`

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;

class Book
{
 String title;
 int price;

 Book(String s, int i) // Конструктор класса Book
 {
  this.title=s;
  this.price=i;
 }
}

class Dlgwin extends Frame implements ActionListener
                       // Диалоговое окно
{
 Book bb;
 String param;
 // Кнопка для выхода из диалога:
 Button bexit=new Button("Exit From Dialog");
 // Кнопка для чтения из файла:
 Button bread=new Button("ReadFromButton");
 // Текстовое поле для отображения названия книги:
 TextField tfi=new TextField();

 // Конструктор диалогового окна; ему передается объект класса
 // Book.
 public Dlgwin(Book z)
 {
```

```
 bb=z; // bb получает ссылку на объект Book, переданный
       // конструктору
 param=z.title;
 setLayout(null);
 add(bexit);
 add(bread);
 add(tfi);
 bexit.addActionListener(this); // Добавление прослушивателя
 bread.addActionListener(this);
 tfi.setBounds(20,160,200,20); // Ручная компоновка элементов
                               // в окне
 tfi.setText(param);
 bexit.setBounds(20,60,100,20);
 bread.setBounds(20,90,100,20);
 setBackground(new Color(200,200,100));
}

public void actionPerformed (ActionEvent e)
{
 if (e.getSource()==bexit)
  {
   // При выходе из диалога инициализируем свойство title:
   bb.title= tfi.getText();
   try
    {
     FileOutputStream os=new FileOutputStream("d1.dat");
     DataOutputStream dos=new DataOutputStream(os);
     // Производим запись в файл d1.dat названия книги:
     dos.writeUTF(bb.title);
     dos.close();
    }
   catch(IOException err)
    {
     JOptionPane.showMessageDialog(null,"Error"+err);
    }
```

```
    this.setVisible(false);
    JOptionPane.showMessageDialog(null,bb.title);
   }
  else
   if (e.getSource()==bread)
    {
     try
      {
       // Чтение из файла названия книги:
       FileInputStream is=new FileInputStream("d1.dat");
       DataInputStream dis=new DataInputStream(is);
       // Отображение названия в текстовом поле:
       tfi.setText(dis.readUTF());
       dis.close();
      }
     catch(IOException err)
      {
       JOptionPane.showMessageDialog(null,"Error"+err);
      }
    }
 }
}

// Основной класс:
public class lab1 extends Frame implements
ActionListener,KeyListener    {
 Dlgwin dlg; // Объявление окна диалога
 Book book;
 Button b1=new Button("Exit");
 Button b2=new Button("Create Object");
 Button b3=new Button("Show Object");
 Label lb=new Label("INTRODUCTION in JAVA");
 TextField tf = new TextField();
```

```
public lab1()
{
 setLayout(null);
 add(tf);
 add(b1);
 add(b2);
 add(b3);
 add(lb);
 addKeyListener(this);
 b1.addActionListener(this);
 b2.addActionListener(this);
 b3.addActionListener(this);
 lb.setBounds(20,20,200,20);
 tf.setBounds(20,160,200,20);
 b1.setBounds(20,60,100,20);
 b2.setBounds(20,90,100,20);
 b3.setBounds(20,120,100,20);
 setBackground(new Color(120,200,120));
}

public String showTitle(Book z) // Показывает название книги
{
 return z.title;
}

public void keyTyped(KeyEvent ke)
{
}

public void keyReleased(KeyEvent ke)
{
}

public void keyPressed(KeyEvent ke)
```

```
{
 if(ke.getKeyChar()=='a')
  System.exit(0);
}

public void actionPerformed (ActionEvent e)
{
 if (e.getSource()==b1)
  System.exit(0);
 else
  if (e.getSource()==b2)
   {
    if (book == null) // Создание объекта-книги
     {
      book=new Book("Pinoccio",200);
      // Открытие диалогового окна с передачей ссылки
      // на Book:
      dlg=new Dlgwin(book);
      dlg.setSize(300,300);
      dlg.setVisible(true);
     }
    else
     System.out.println("The object already exists");
   }
  else
   if (e.getSource()==b3)
    {
     if(book!=null)
      {
       Font fnt=new Font("Courier",Font.BOLD,24);
       Graphics g= getGraphics();
       this.setForeground(new Color(250,0,0));
       g.setFont(fnt);
       g.drawString(book.title.toUpperCase(),20,300);
```

```
        tf.setText(this.showTitle(book).toUpperCase());
       }
     }
 }

 public static void main(String [] args)
 {
  lab1 f= new lab1();
  f.setForeground( new Color(0,0,250));
  f.resize(500,500);
  f.setVisible(true);
  f.requestFocus();
 }
}
```

Теперь подробно рассмотрим, как здесь выполняется запись в файл и чтение из файла. Запись в файл реализует следующий фрагмент кода:

```
public void actionPerformed (ActionEvent e)
{
 if (e.getSource()==bexit)
  {
   bb.title= tfi.getText();
   try
   {
    FileOutputStream os=new FileOutputStream("d1.dat");
    DataOutputStream dos=new DataOutputStream(os);
    dos.writeUTF(bb.title); // Записали строку в файл d1.dat
    dos.close();
   }
   catch(IOException err)
    {
     JOptionPane.showMessageDialog(null,"Error"+err);
    }
```

```
    this.setVisible(false);
    JOptionPane.showMessageDialog(null,bb.title);
   }
```

Сначала создаем *файловую переменную* `os`:

```
FileOutputStream os=new FileOutputStream("d1.dat");
```

Эта переменная принадлежит к классу `FileOutputStream`. Данный класс содержит низкоуровневые методы для записи, выполняющие пересылку отдельных байтов. Поэтому используется более высокоуровневый вывод на основе класса `DataOutputStream`, который играет роль "обертывающего" класса по отношению к `FileOutputStream`. Видим, что переменная `dos` создается с помощью конструктора, который получает в качестве входного аргумента переменную `os` класса `FileOutputStream`. Запись в файл реализует оператор:

```
dos.writeUTF(bb.title);
```

Этот оператор выполняет запись строковых величин в формате Unicode. Содержимое файла d1.dat после записи выглядит следующим образом:

```
Buratino
```

Фрагмент записи в файл помещен внутрь конструкции:

```
try
 {
. . . .
 }
catch(IOException err)
 {
  ...
 }
```

Такая конструкция называется охраной. *Охрана* нужна для того, чтобы в случае возникновения ошибки передать управление на часть кода, записанную внутри `catch(IOException err) {...}`.

Несмотря на то, что в этом случае программа все равно завершается, обработчик ошибки позволяет принять меры для корректного завершения программы. Использованная конструкция `catch` отлавливает ошибки типа `IOException` (ошибки ввода-вывода). Это значит, что любая ***ошибка ввода-вывода***, вызванная внутри блока `try`, будет передана на обработку в блок `catch`. Использование охраны имеет целью повысить надежность программ на языке `Java`. Впрочем, охрана применяется и в других современных языках.

Наконец, ***чтение из файла*** реализует следующий фрагмент кода:

```
if (e.getSource()==bread)
{
 try
 {
  FileInputStream is=new FileInputStream("d1.dat");
  DataInputStream dis=new DataInputStream(is);
  tfi.setText(dis.readUTF());// прочитали строку из файла
  dis.close(); //закрыли файл
 }
 catch(IOException err)
 {
  JOptionPane.showMessageDialog(null,"Error"+err);
 }
}
```

Без труда можно проследить аналогию между этим фрагментом и фрагментом для записи, только сейчас мы имеем дело с базовыми классами `FileInputStream` и `DataInputStream`.

Работа со строками

Работа со строками составляет важный аспект любого языка программирования. В языке `Java` для создания строки следует использовать оператор:

```
String s="New string";
```

Можно также создать строку из массива символов, например:

```
char charray []={'a','b','b','a'};
String s=new String(chrarray);
```

Определение длины строки выполняется так:

```
int i=s.length();
```

Соединение двух строк `s1`, `s2` производится таким образом:

```
String s= s1+s2;
```

Проверка на равенство двух строк выполняется так:

```
if(s1.equals(s2))
 {…}
else
 {…}
```

Проверка на равенство без учета регистра выполняется таким образом:

```
if(s1.equalsIgnoreCase(s2))
 { …}
else
 {… }
```

Преобразование числа в строку осуществляем таким путем:

```
int i=10;
String s="" + i;
```

Получение конкретного символа строки по его индексу реализуется следующим образом:

```
int i=2;
// Получаем значение символа, стоящего в третьей позиции:
char c = s.charAt(2);
```

Выделение подстроки реализуется, например, таким образом:

```
String s1="Hello, World";
String s1=s2.substring(1,3); // s1 получает значение
// подстроки s2, содержащей символы со второго по четвертый —
// нумерация символов начинается с нуля, поэтому номер 1
// соответствует второму символу; 3 соответствует четвертому
// символу и т. д.
```

*Замена одного символ*а строки на другой выполняется, например, согласно следующему оператору:

```
s = s.replace('a', 'o'); // заменяет в строке s повсюду
// символ 'a' на 'o'.
```

Можно определить, с какого места одна строка входит во вторую:

```
int i =s.indexOf("he");
```

Если подстрока `"he"` не содержится в строке `s`, то возвращается значение −1.

Далее можно проверить, начинается ли строка с заданной подстроки. Это делается так:

```
if (s.startsWith("O-o-h"))
 {...}
```

Аналогично проверяем, заканчивается ли строка заданной подстрокой:

```
if (s.endsWith("O-o-h"))
 {...}
```

Наконец, важный аспект при работе со строками — преобразование строки в другой тип.

Например, "10" — преобразуем в 10, "2.4" — в 2.4. Пусть, например, `s="10"`. Тогда требуемые преобразования выполняют следующие операторы:

```
Integer i = Integer.valueOf(s);
Float f=Float.valueOf(s);
int k= i.intValue();
float ff= f.floatValue();
```

Эти действия можно объединить:

```
int k=Integer.valueOf(s).intValue();
float ff=Float.valueOf(s).floatValue();
```

Использование массивов

Объявление массивов выполняется следующим стандартным способом:

```
Тип переменная [] = new Тип [размер]
```

Например:

```
int z[] =new int[100]; // Объявляем и создаем
        // неинициализированный массив из 100 целых чисел
```

Другой способ создания — явное определение массива перечислением его элементов:

```
int z[]={1,2,3};
```

Создается массив из трех целых чисел: 1, 2, 3.

Запомним, что нумерация элементов массивов начинается с нуля. Количество элементов массива можно получить так:

```
int le=z.length;
```

Несколько более специфично выполняется *создание массива объектов*, например:

```
Book [] barray= new Book[3]; // создаем массив объектов класса
                             // Book
// Создаем первый элемент:
barray[0]= new Book("Герой нашего времени","Лермонтов");
// Создаем второй элемент:
barray[1]= new Book("Аэлита","А.Толстой");
// Создаем третий элемент:
barray[2]= new Book("12 стульев","И.Ильф, Е.Петров");
```

Рассмотрим пример передачи массива в качестве параметра метода. В следующем примере передается целочисленный массив и возвращается число его элементов.

```
import java.awt.*;
public class prim1
{
 static int count(int [] xar)
 //этот метод возвращает размер массива xar
 {
  return xar.length;
 }

 public static void main(String args[])
 {
```

```
  int [] intarray = {1,2,3,4,5}; // явное перечисление
                                 // элементов массива
  int i=0;
  // Ниже помещено обращение к методу count с передачей ему
  // (ссылки) на массив intarray; метод count должен быть
  // static, поскольку не создан объект класса prim1, который
  // этим методом обладает
  System.out.println("The array size is:"+count(intarray));
  try
  {
   System.in.read();
  }
  catch(Exception e){}
 }
}
```

Апплеты

Апплеты — это приложения, работающие в Интернете. Апплет открывается браузером и должен быть описан в HTML-файле. Собственно, апплет есть то же, что и форма, только окно апплета открывается браузером (например, Internet Explorer или другим браузером, поддерживающим Java-апплеты). Апплет имеет определенную структуру, о которой мы поговорим несколько позже. Приведем пример простейшего апплета, выводящего строку приветствия и текущее время (рис. 1.9).

Для построения этого апплета нужно создать два файла: Java-файл и HTML-файл. Последний должен иметь следующий вид:

```
<html>
<APPLET code="firstapplet.class" width=300 height=400"
</Applet>
</html>
```

Язык HTML мы рассматриваем в цикле практических занятий. Пока же отметим, что апплет вставляется с помощью тегов:

```
<APPLET code="firstapplet.class" width=300 height=400"
</Applet>
```

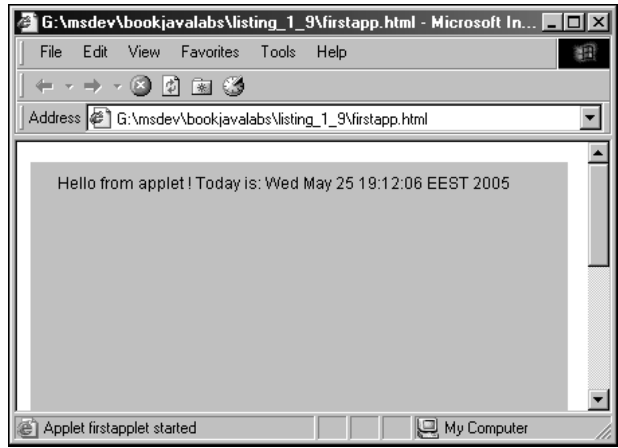


Рис. 1.9. Окно простейшего апплета

Параметр `code` определяет имя созданного файла класса; `width` — задает ширину, а `height` — высоту апплета. Исходный файл firstapplet.java имеет такой вид (листинг 1.9).

Листинг 1.9. Приложение на основе апплета

```
import java.applet.Applet;
import java.awt.*;
import java.util.*;

public class firstapplet extends Applet
{
 private String str1;
 private Date date;

 public void init()
 {
  str1="Hello from applet !"; // строка приветствия
  date=new Date(); // получаем дату и время
  str1=str1+" Today is: "+date;
 }
```

```
 public void paint(Graphics g)
 {
  g.drawString(str1,20,20); // вывод приветствия
                            // и даты-времени
 }
}
```

Название файла совпадает с именем главного класса. При построении апплетов необходимо обязательно подключать пакет:

```
import java.applet.Applet;
```

Строка класса апплета обязательно определяет наследование от класса `Applet`:

```
public class firstapplet extends Applet
```

Апплет в общем реализует следующие методы: `start()`, `init()`, `run()`, `paint()`, `destroy()` и `stop()`, правда, не все они обязательны для реализации. Метод `init()` использован для инициализации (роль, аналогичная конструктору). Его код очень прост:

```
public void init()
{
 str1="Hello from applet !";
 date=new Date();
 str1=str1+" Today is: "+date;
}
```

Отметим, что оператор

```
date= new Date();
```

получает системную дату.

Метод `paint()` используется для перерисовки окна апплета. Перерисовку можно вызвать перетаскиванием окна апплета или с помощью команды `repaint()`. При первом открытии окна апплета метод `paint()` вызывается так же. В апплете осуществляется вывод строки так же, как и на форме.

Метод `start()` вызывается всякий раз (многократно) при возврате к данной странице, а метод `stop()` — при выходе из страницы.

Метод `destroy()` вызывается один раз при уничтожении окна апплета из памяти. Таким образом, назначение методов апплета вполне конкретно. Заметим, что в апплетах нельзя, в общем случае, выполнять работу с файлами и базами данных. Таковы ограничения безопасности, устанавливаемые приложением браузера Internet Explorer (другие браузеры, впрочем, могут давать послабления). Отмеченное обстоятельство следует постоянно иметь в виду.

Для запуска апплета нужно открыть HTML-файл (двойным щелчком мышью на его имени). Предварительно компилируется Java-файл выполнением команды:

```
javac firstapplet.java
```

Запомним, что апплет запускается иначе, чем приложение Java, а именно: апплет открывается как обычный HTML-файл двойным щелчком мыши на имени файла HTML, содержащего теги `<Applet></Applet>`.

Обработка исключительных ситуаций

В языке Java предусмотрена обработка ошибочных ситуаций, связанных, как правило, с выполнением таких системных действий, как ввод-вывод через файлы, запуск приложений, отличных от Java, чтение портов и пр. И даже мы встречались с такими ситуациями уже не раз. Рассмотрим следующий пример из того, что приводилось ранее (см. листинг 1.7).

```
if (e.getSource()==bread)
 {
  try
   {
    FileInputStream is=new FileInputStream("d1.dat");
    DataInputStream dis=new DataInputStream(is);
    tfi.setText(dis.readUTF()); // прочитали строку из файла
    dis.close();
   }
  catch(IOException err)
```

```
    { JOptionPane.showMessageDialog(null,"Error"+err);  }
  }
}
```

В блоке `try` осуществляется создание потоковой переменной по имени `is` класса `FileInputStream`, на основе которой далее создается *объектная переменная* `dis`, позволяющая читать информацию из файла по команде `dis.readUTF()`.

Операция чтения связана с выполнением системных действий, которые могут привести к *ошибке ввода-вывода*. Поэтому блок `catch` выполняет перехват именно этого типа ошибок:

```
catch(IOException err)
 { JOptionPane.showMessageDialog(null,"Error "+err);  }
```

Класс ошибок ввода-вывода называется `IOException`. Если при чтении/записи из файла возникнет ошибка класса `IOException`, то она будет перехвачена данным блоком `catch`, что приведет к выполнению единственного оператора внутри этого блока:

```
JOptionPane.showMessageDialog(null,"Error"+err);
```

Данный оператор просто выведет сообщение об ошибке в диалоговом окне. Наиболее общим классом ошибок (исключений) является класс `Exception`. Этот класс позволяет перехватывать вообще все исключения, так что предыдущий пример можно было переписать таким образом:

```
if (e.getSource()==bread)
 {
  try
  {
   FileInputStream is=new FileInputStream("d1.dat");
   DataInputStream dis=new DataInputStream(is);
   tfi.setText(dis.readUTF()); // Прочитали строку из файла
   dis.close();
  }
  catch(Exception err) // Здесь изменен класс исключений
   { JOptionPane.showMessageDialog(null,"Error"+err);  }
 }
```

По мере рассмотрения практических занятий мы будем вводить новые классы исключений.

Работа с графикой

Простейшее рисование на формах и апплетах выполняют методы класса `Graphics`. В апплете имеется стандартный метод `paint()(Graphics g)`, который использует переменную графического контекста `g` для рисования простейших фигур и вывода текста. Например, вывод текстовой строки выполняется таким образом:

```
g.drawString(str,x,y);
```

Здесь `str` — строковая переменная, а `x`, `y` — координаты первого символа строки. Достаточно часто, однако, возникает необходимость выполнить вывод строки или нарисовать фигуру не в методе `paint()`, а в любом другом методе. В этом случае нужно создать *объектную графическую переменную* и затем использовать ее методы рисования. Для создания объектной графической переменной `z` используем команду следующего вида:

```
Graphics z = getGraphics();
```

Приведем пример, в котором на форме выполняется вывод строки при нажатии на программную кнопку (листинг 1.10).

Листинг 1.10. Графический вывод вне метода `paint()`

```
import java.awt.*;
import java.awt.event.*;

public class painting extends Frame implements ActionListener
{
 private String str1;
 Button print=new Button("Print"); // Кнопка для вывода строки
 Button clear=new Button("Clear"); // Кнопка для очистки
                                   // области экрана
 Button exit=new Button("Exit"); // Кнопка для выхода из
                                 // приложения
 painting()  // Конструктор класса painting
```

```
{
 setLayout(null); // Размещение элементов выполняется вручную
 str1="Hello from Frame !";
 add(print); // Добавление кнопки
 add(clear);
 add(exit);
 print.addActionListener(this);  // Добавление прослушивателя
 clear.addActionListener(this);
 exit.addActionListener(this);
 print.setBounds(10,20,100,20);
 clear.setBounds(10,50,100,20);
 exit.setBounds(10,80,100,20);
}

public void actionPerformed(ActionEvent e) // Обработчик
                                            // событий
                                            // от кнопок
{
 if(e.getSource()==exit)
  System.exit(0); //Завершение приложения
 else
  if (e.getSource()==print)
   {
    Graphics z=getGraphics(); // Создаем объектную
                              // графическую переменную z
    z.drawString(str1,10,120); // Вывод строки str1
   }
  else
   if(e.getSource()==clear)
    {
     Graphics z=getGraphics();
     z.clearRect(8,110,110,35); // Очистка и заливка цветом
                                // фона прямоугольной области
    }
}
```

```
 public static void main(String[] args)
 {
  painting pt= new painting(); // Создаем форму
  pt.resize(400,400); // Задаем размеры формы
  pt.setBackground(new Color(120,100,180)); // Устанавливаем
                                             // цвет формы
  pt.show();
 }
}
```

В этом примере используем графическую объектную переменную для вывода строки, *очистки прямоугольной области экрана* и заливки ее цветом фона:

```
if(e.getSource()==clear)
{
 Graphics z=getGraphics();
 z.clearRect(8,110,110,35); // Очистка и заливка цветом фона
                            // прямоугольной области
}
```

В цикле практических занятий будем рассматривать различные графические методы класса `Graphics`. Заметим, что класс `Graphics` не позволяет "удерживать" изображение (например, рисунок). Полезен и другой класс — `Canvas`. Объект этого класса можно перемещать по форме (апплету) или панели (`Panel`). Можно управлять его размерами, создавать на его основе дочерние классы и пр. Предыдущее приложение незначительно изменено (листинг 1.11) таким образом, что по нажатию на кнопку **Print** строка выводилась в случайной позиции экрана.

Листинг 1.11. Графический вывод на основе класса `Canvas`

```
import java.awt.*;
import java.awt.event.*;
public class painting extends Frame implements ActionListener
```

```
{
 private String str1;
 // Построена классовая переменная на базе Canvas:
 private Canvas cs=new Canvas();
 Button print=new Button("Print");
 Button clear=new Button("Clear");
 Button exit=new  Button("Exit");

painting()
 {
  setLayout(null);
  str1="Hello from Frame !";
  add(print);
  add(clear);
  add(exit);
  add(cs);
  print.addActionListener(this);
  clear.addActionListener(this);
  exit.addActionListener(this);
  print.setBounds(10,20,100,20);
  clear.setBounds(10,50,100,20);
  exit.setBounds(10,80,100,20);
  cs.setBounds(10,110,180,150);
  // Объект Canvas можно размещать в произвольном месте формы
  cs.setBackground(Color.yellow); // Объект Canvas можно
                                  // раскрашивать
 }

 public void actionPerformed(ActionEvent e)
 {
  if(e.getSource()==exit)
   System.exit(0);
  else
   if (e.getSource()==print)
    {
```

```
     int x=(int) Math.round(( Math.random()*20+10));
     //x,y — случайные координаты для вывода объекта Canvas
     int y=(int) Math.round((Math.random()*100+110));
     cs.move(x,y); // Перемещаем объект Canvas в новую позицию
     // Выводим строку в объекте Canvas:
     cs.getGraphics().drawString(str1,10,10);
    }
   else
    if(e.getSource()==clear)
     {
      cs.getGraphics().clearRect(0,0,170,135); // Очищаем
                                               // область
                                             // объекта Canvas
     }
 }

 public static void main(String[] args)
 {
   painting pt= new painting();
   pt.resize(400,400);
   pt.setBackground(new Color(120,100,180));
   pt.show();
 }
}
```

Рассмотрим теперь, как получить картинку. Держателем картинки является класс `Image`. Поэтому первоначально следует сделать объявление наподобие следующего:

```
Image img;
```

Затем в строковой переменной следует указать адрес картинки:

```
String s="c:/work/1.gif";
```

Java не всегда отображает файлы с расширением .bmp (используем при работе с графикой файлы .gif и .jpg). После этого нужно инициализировать переменную `img`:

```
img=getToolkit().getImage(s);
```

И, наконец, рисуем картинку:

```
Graphics g= getGraphics();
g.drawImage(img,20,150,this);
```

Заметим, что *рисование картинки* следует предварить операцией ее подготовки:

```
prepareImage(img,this);
```

поскольку в противном случае картинка может не отобразиться.

Следующее небольшое приложение (листинг 1.12) объединяет все сказанное ранее.

Листинг 1.12. Рисование картинки на форме

```
import java.awt.*;
import java.awt.event.*;

class Pic extends Frame implements ActionListener
// Приложение создается на основе Frame

{
 private Image img; // Объявляем картинку
 Button exit=new Button("Exit");
 Button pict=new Button("Picture"); // По нажатию на эту
                                    // кнопку картинка
                                    // отображается на форме

 Pic()
 {
  setLayout(null);
  add(exit);
  add(pict);
  exit.addActionListener(this);
  pict.addActionListener(this);
  exit.setBounds(10,20,100,20);
  pict.setBounds(10,50,100,20);
  String s="C:/work/1.gif"; // Формируем строку с адресом
                            // картинки
```

```
  img= getToolkit().getImage(s); // Создаем переменную класса
                                 // Image для хранения
                                 // картинки
  prepareImage(img,this); // Переменная img заранее
                          // инициализируется, но еще не
                          // отображается
 }

 public void actionPerformed(ActionEvent e)
 {
  if(e.getSource()==exit)
   System.exit(0); // выход из приложения
  else
   if(e.getSource()==pict)
    {
     Graphics g =getGraphics(); // Создаем переменную
                                // графического контекста
                                // для рисования картинки
     g.drawImage(img,10,80,this); // Рисуем картинку
     g.drawLine(10,200,200,200); // Рисуем линию
     g.setColor(Color.blue); // Изменяем цвет пера
     g.drawLine(10,210,200,210); // Рисуем линию голубым
                                 // цветом
    }
 }

 public static void main(String[] args)
 {
  Pic fr=new Pic(); // Создаем объект класса приложения
  fr.resize(400,400); // Устанавливаем размеры формы
                      // в пикселах
  fr.setVisible(true); // делаем форму приложения видимой
 }
}
```

В приведенном листинге следует обратить внимание на оператор:

```
img= getToolkit().getImage(s);
```

который присваивает переменной `img` битовый образ, хранимый в файле с адресом `s`.

Отметим, что в апплете для этих целей используется команда

```
img=getImage(URL u, "name.gif");
```

Здесь первый операнд задает URL-адрес картинки в сети Интернет, второй — имя картинки.

Результат работы приложения приведен на рис. 1.10. В прилагаемом к книге CD нужно предварительно подготовить файл с картинкой с именем 1.gif и разместить его в каталоге c:\work.

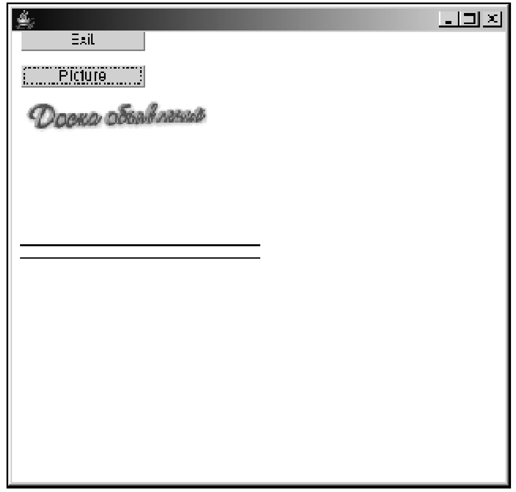


Рис. 1.10. Рисование картинки и линий

Более продвинутые возможности для рисования предоставляет пакет Graphics2D. Например, средствами этого пакета можно выполнять *градиентную заливку* фигур, управлять шириной пера, поворачивать фигуры и пр. В качестве иллюстрации приведем приложение, демонстрирующее некоторые возможности пакета Graphics2D (листинг 1.13).

Листинг 1.13. Рисование с градиентной заливкой на базе пакета Graphics2D

```
import java.awt.*;
import java.awt.geom.*;

public class Pic1 extends Frame
{
 private Ellipse2D.Double circle= new
 Ellipse2D.Double(20,30,100,100); // создаем переменную circle
                                  // класса Ellipse2D.Double

 Pic1()
 {
  resize(300,300);
  setBackground(Color.yellow);
 }
 public void paint(Graphics g)
 {
  Graphics2D g2=(Graphics2D) g; // Преобразуем к классу
                                // Graphics2D переменную
                                // Graphics
  //Устанавливаем режим градиентной заливки
  g2.setPaint(new
GradientPaint(0,0,Color.red,110,110,Color.green,true));
  g2.fill(circle); // Заливаем окружность
  g2.setPaint(Color.white); // Устанавливаем цвет пера
                            // в значение  Color.white
  g2.drawString("JAVA+",30,60); // Выводим строку
 }

 public static void main(String [] args)
 {
  Pic1 fr=new Pic1();
  fr.resize(400,400);
```

```
  fr.show();
 }
}
```

Результат работы программы из листинга 1.13 приведен на рис. 1.11.

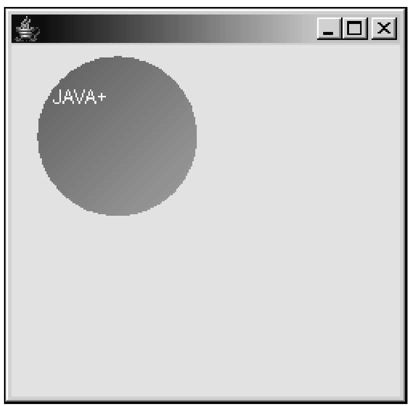


Рис. 1.11. Градиентная заливка

Подключаем пакет Graphics2D командой:

```
import java.awt.geom.*;
```

Объявление переменной типа эллипса ясно из листинга 1.13. Важно обратить внимание на способ получения графического контекста `Graphics2D`, выполненный на основе преобразования типов:

```
Graphics2D g2=(Graphics2D) g;
```

Рисование окружности (эллипса) с заливкой реализует команда:

```
g2.fill(circle);
```

При этом заранее устанавливается режим градиентной заливки:

```
g2.setPaint(new GradientPaint
(0,0,Color.red,110,110,Color.green,true));
```

В этой команде указаны два цвета — Color.red, Color.green и координаты области, в которой выполняется заливка (в той части

этой области, где содержится замкнутая часть рисуемой фигуры). Красный цвет начинается с точки с координатами (0,0), зеленый — с точки с координатами (110,110).

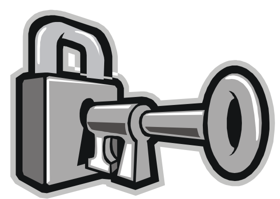
В этом же листинге осуществлен вывод строки с предварительной установкой цвета пера:

```
g2.setPaint(Color.white); // Устанавливаем цвет пера
                          // в значение Color.white
g2.drawString("JAVA+",30,60); // Выводим строку
```

Более интересные подробности, касающиеся Graphics2D, можно найти в [13].

Наше предварительное введение в язык Java завершено. Основные концепции этого языка, необходимые для успешного и быстрого старта, изложены.

Приступаем к практическим занятиям, в которых будут вводиться новые понятия и конструкции, а также будут использоваться уже объясненные в этой главе.



Глава 2

Практические занятия по Java

Основы HTML

Цель занятия

Целью настоящего занятия является изучение принципов создания HTML-документов. Рассматривается краткое введение в язык HTML, который в этом практическом пособии неоднократно используется при создании клиентских приложений во взаимодействии с Java, JavaScript и XML. Более подробные сведения заинтересованный читатель может найти в [2, 13].

Краткие теоретические сведения

Структура HTML-документа определяется упорядоченным набором *тегов* следующего вида:

```
<HTML>
<HEAD> текст
</HEAD>
<BODY> управляющие_теги и текст
</BODY>
</HTML>
```

Программа-клиент браузер при просмотре файлов с этими тегами выполняет отображение документа в окне. Каждый такой файл имеет расширение html или htm, а набрать его можно в любом текстовом редакторе. Теги играют роль команд и заставляют браузер выполнить предписываемые ими действия. *Область действия тега* определяется тем местом, где он указан, и тем местом, где он закрыт (записан в угловых скобках с предшествующей косой чертой). В общем случае теги имеют параметры. Рассмотрим некоторые из них.

```
<BODY alink=color background=url bgcolor=color
scroll=yes|no bgproperties=fixed> ...</BODY>
```

Параметр `alink` задает цвет гиперссылок (по умолчанию голубой). Параметр `background` задает адрес рисунка, на фоне которого отображается документ. Параметр `bgcolor` задает цвет фона документа. Параметр `scroll` задает возможность прокрутки документа (`yes`). Параметр `bgproperties=fixed` устанавливает невозможность прокрутки фонового рисунка документа при прокрутке содержимого документа.

В качестве возможных для использования цветов можно указывать: `aliceblue` — синий; `aqua` — голубой; `azure` — азурный; `gold` — золотой; `gray` — серый; `green` — зеленый; `crimson` — кремовый; `brown` — коричневый; `darkmagenta` — темно-малиновый; `ivory` — цвет слоновой кости; `orange` — оранжевый; `lime` — цвет извести; `purple` — розовый; `sienna` — фиолетовый; `khaki` — зеленый-защитный и др.

Для форматирования текста в документе следует использовать следующие теги:

- от `<H1>` до `<H6>` — изображает текст от наименьшего до максимального размера. Этот тег использует параметр `align = center| left|right`, который устанавливает смещение текста в окне (по центру, влево, вправо). Например:

  ```
  <H1 align=center> Никогда не забывай то, что учишь</H1>
  ```
  ;
- `<P>` — задает начало и конец параграфа;
- `<CENTER>` — парный тег, размещает текст и рисунки по центру;
- `<BR>` — перевод каретки на следующую строку. Тег этот непарный;

- <HR> — горизонтальная линия. Используется для подчеркивания. Длину линии в пикселах можно задать с помощью параметра `width`= число, цвет линии — параметром `color` = цвет; толщину линии — параметром `size` = число.

Создайте следующий документ:

```
<HTML>
<BODY bgcolor=floralwhite>
<H1> Simple document <HR align=left size=4 color=red
width=200 >
</HR></H1>
</Body>
</HTML>
```

Наберите текст в Блокноте (NotePad). Сохраните его в файле (укажите тип файла: **all** (все), задайте имя и расширение, например: lab00.html) и запустите файл на выполнение из программы FAR (или из Проводника или браузера).

Можно также использовать теги:

- <B> — задает жирный вариант линий;
- <I> — задает курсивный вариант текста;
- <U> — задает подчеркивание текста;
- <Strike> — устанавливает перечеркивание текста;
- <CITE> — используется для выделения цитат;
- <PRE> — используется для учета пробелов, например: `<PRE> Here to write </PRE>`
- <FONT color=цвет face=шрифт size=n> — задает цвет, название и размер шрифта. Пример:

  ```
  <Font color=magenta face="Arial" size=24> OK </Font>
  ```

  Проверьте работу этого тега на несколько усложненном примере:

```
<HTML>
<BODY bgcolor=floralwhite>
<H1>
<Font color=magenta face="Arial" size=24> OK </Font>
Simple document <HR align=left size=4 color=red width=280>
</HR></H1>
```

```
</Body>
</HTML>
```

- <!-- помещаем комментарий --> — приведенные теги используются для вставки комментария в тело HTML-документа.

Для размещения в документе рисунка наберите:

```
<IMG src=url alt=text border=n height=n1 width=n2 >
```

Здесь параметры имеют следующий смысл:

- src — адрес файла с изображением или видеоклипом;
- alt — текст, выводимый на месте изображения, если оно не загружено браузером;
- border — толщина рамки;
- height и width — соответственно, высота и ширина изображения.

Пример:

```
<IMG src="globe.jpeg" border=2 alt="Not available"
width=200 height=200 align=right>
```

Рассмотрим, как в документе размещаются гиперссылки. *Гиперссылка* — это строка, при щелчке на которой мышью выполняется переход к другому документу, связанному с этой гиперссылкой. Задание гиперссылки реализуется так:

```
<A href=url target=window_name| _blank| _parent|_self>
текст_гиперссылки | рисунок | элемент_управления </A>
```

Пример:

```
<A href="w.html"> press this hyperlink </A>
```

В данном примере в документе будет выведена строка "press this hyperlink", выделенная синим или другим цветом (вспомните параметр alink тега <BODY>). Щелчок кнопкой мыши в области гиперссылки приведет к загрузке нового документа из файла w.html. Новый документ загружается в отдельном окне. Но можно указать новую дислокацию документа, если использовать параметр target. Значения этого параметра таковы:

- window_name — имя окна или фрейма;
- _blank — вывод в новое окно;
- _self — вывод в текущее окно или фрейм;
- _parent — загрузка в родительский фрейм для данного окна.

Пример использования в качестве *гиперссылки* рисунка:

```
<A href="w.html" target=_self> <IMG src="www.gif"></A>
```

Для произвольного размещения элемента в документе следует использовать параметр STYLE, который имеет следующие подпараметры:

- Position — задает абсолютное или относительное положение элемента;
- z-index — задает, на каком уровне (сверху или ниже других) располагается элемент;
- top, left, width, height — верхняя координата, левая координата, ширина и высота.

Пример:

```
<HTML>
<BODY bgcolor=blue>
<DIV STYLE="position:absolute; top:100; left:50;
width:200;height:200">
    <FONT color=red size=18>
       YESSS!!!
     </FONT>
   </DIV>
</BODY>
</HTML>
```

Тег <DIV> используется для группировки нескольких элементов в указанной области. В примере выше тег <DIV> охватывает область (100, 50, 200, 200). В этой области выводится текст "YESSS!!!" В документе можно определить несколько тегов <DIV>, что позволит наложить одну группу элементов на другую. Пример накладки текстов:

```
<HTML>
 <BODY bgcolor=blue>
<DIV STYLE="position:absolute; top:100; left:50;
width:200;height:200;z-index:0">
    <FONT color=red size=18>
       YESSS!!!
     </FONT>
```

```
   </DIV>
<DIV STYLE="position:absolute; top:90; left:50;
width:200;height:200; z-index:1">
    <FONT color=yellow size=18>
       YESSS!!!
     </FONT>
   </DIV>
</BODY>
</HTML>
```

Откройте браузером данный HTML-файл или щелкните мышью дважды на его имени в окне Norton Commander (можно использовать также, например, FAR или диспетчер Windows).

Для создания списка следует использовать теги `<OL> <LI>...</LI></OL>`

Пример:

```
<OL>
<LI type=i> One </LI>
<LI type=i> Two </LI>
</OL>
```

В качестве маркера списка можно использовать графическое изображение. Для этого нужно для тега `<LI>` определить свойство `list-style-image` атрибута `style`.

Пример:

```
<LI style="list-style-image:url(pic1.gif) ">пример</LI>
```

Рассмотрим теперь пример создания таблицы:

```
<TABLE border=1 bordercolor=green>
<CAPTION valign=center> table of names </Caption>
<tr>
    <td> name</td> <td> johns </td>
</tr>
<tr>
<td> name</td> <td> silvestr</td></tr>
</TABLE>
```

Теги `<table></table>` определяют начало и конец таблицы. Тег `<Caption>` задает заголовок таблицы. Тег `<td>` задает содержимое

ячейки, а тег `<tr>` задает начало строки таблицы. Теперь приведем HTML-документ следующего вида, который аккумулирует все ранее сказанное, а результат его выполнения (отображение в браузере) представлен на рис. 2.1.

```
<HTML>
<BODY bgcolor=floralwhite>
<H1>
<Font color=magenta face="Arial" size=24> OK </Font>
 Simple document <HR align=left size=4 color=red width=280>
 </HR></H1>
<OL>
 <LI style="list-style-
image:url(d:/msdev/labsjava/route.gif)"> one </LI>
 <LI type=i> two </LI>
</OL>
<TABLE border=1 bordercolor=green>
<CAPTION valign=center> table of names </Caption>
<tr>
    <td> name</td> <td> johns </td>
</tr>
<tr>
<td> name</td> <td> silvestr</td></tr>
</table>
</Body>
</HTML>
```

Обратите внимание на то, что с пунктом списка `one` связан рисунок. При работе с таблицами важно уметь объединять ячейки. Для объединения смежных ячеек строки следует использовать атрибут `rowspan`, а для объединения смежных ячеек столбца — атрибут `colspan`. Приведем пример:

```
<Table border bordercolor=blue>
<tr>
<td colspan=2 align=center><B>HELLO</B></td>
</tr>
<tr>
<td> YOU </td>
```

```
<td> &YOU</td>
</tr>
</Table>
```

Рис. 2.1. Пример результирующего документа

В данном примере строка `<td colspan=2 align=center> <B>HELLO</B></td>` объединяет две ячейки (два столбца). Поэтому сообщение "`HELLO`" размещается по центру (атрибут `align=center`) этой объединенной области. Использование таблиц важно для структурирования и правильного размещения элементов документа. В ячейках таблиц можно размещать кнопки, картинки, текстовые поля и другие элементы интерфейса.

Фреймы и формы

Фреймы — это окна, в которых размещаются отдельные документы. В отличие от обычных окон, фреймы нельзя перетаскивать. Простой пример, иллюстрирующий создание двух фреймов, такой:

```
<HTML>
<FRAMESET cols="40%,60%">
```

```
<Frame frameborder=yes src="pr1.html">
<Frame frameborder=yes src="pr2.html">
</FRAME>
</HTML>
```

В этом примере создаются два фрейма: первый загружает документ pr1.html, второй — pr2.html. Первый фрейм занимает 40% общего числа столбцов экрана. Второй — 60%. Фреймы должны определяться внутри тегов `<FRAMESET>...</FRAMESET>`.

Горизонтальное разбиение на фреймы дает такой код:

```
<HTML>
<FRAMESET rows="40%,60%">
<Frame frameborder=yes src="pr1.html">
<Frame frameborder=yes src="pr2.html">
</FRAME>
</HTML>
```

Результат выполнения такого HTML-файла можно увидеть на рис. 2.2.

Рис. 2.2. Горизонтальные фреймы

Формы используются для размещения на сайте элементов *визуального интерфейса*: кнопок, флажков, текстовых полей, списков и др. Разместим на форме кнопку следующим образом:

```
<Form>
<Input type="button" name="b1" value="PRESS"></INPUT>
</Form>
```

Отображение кнопки продемонстрировано на рис. 2.2.

Для отображения текстового поля следует набрать строку:

```
<input type="text"  name="c1" cols=20 value="something">
```

Размер текстового поля определяется параметром `cols`. Рассмотрим следующий пример:

```
<html>
<Body>
<Form method="post" Action="c:\Program
Files\ActivePerl\Perl\bin\first.cgi">
<input type="text" value="dderr">
<input type="Submit" value="SEND">
</form>
</body>
<html>
```

Обратим внимание на следующие два момента. Первое: в теге `<Form>` заданы два важных параметра: `method` и `Action`. Параметр `Action` указывает сетевой адрес программы обработки для сайта, находящейся на стороне сервера (отметим, что и клиент, и сервер могут находиться на одном и том же компьютере). Параметр `method` указывает, каким образом будут передаваться параметры обработчику сайта. Эта программа-обработчик будет запущена при нажатии на кнопку `Submit`. Второе: кнопка `Submit` — это специальная кнопка, назначением которой как раз и является вызов обработчика сайта. Программы обработки сайтов реализуются на серверах. В качестве языков программирования они, в принципе, могут использовать любые языки. В настоящее время широко применяется язык PERL. Можно писать обработчики сайтов на Java и C#. Обработчики, написанные на Java, называются *сервлетами*. Для запуска сервлета нужно установить на компьютере подходящую программу сервера, например,

TOMCAT. Эта программа свободно распространяется в Интернете.

Нажатие кнопки `Submit` вызывает сервлет. Запрос на запуск сервлета формирует браузер. Запрос содержит адрес сервлета и строку параметров в виде: `name = value .... name = value`. В методе `GET` строка параметров передается вместе с адресом сервлета. В методе `POST` она передается отдельным файлом. Java-сервлеты извлекают параметры одинаковым образом как для `POST`-варианта, так и для `GET`-варианта.

На этом завершаем краткое введение в разработку HTML-файлов.

Задание

Создать собственный сайт, используя язык HTML. Примерный список тем для сайтов следующий.

- Расписание занятий.
- Товары.
- Киноафиша.
- Футбольные команды.
- Автомобили.
- Компьютеры.

Представить сайт в форме `FRAMESET`. На сайте должны быть картинки, таблицы, кнопки, списки и другие элементы интерфейса. Обеспечить смену документов во фреймах с помощью гиперссылок.

Контрольные вопросы

1. Что такое сайт?
2. Какова структура документа HTML?
3. Как разместить на сайте гиперссылку, рисунок, таблицу?
4. Для чего нужен тег `<PRE>`?
5. Что выполняет программа обработки скрипта?

6. Как в документе указывается адрес программы обработки скрипта?
7. Для чего используется сервер TOMCAT?
8. Как разбить сайт на фреймы?
9. Как объединить ячейки таблицы?
10. Какие элементы можно размещать на форме? Является ли таблица элементом формы?

Использование скриптов JavaScript в документах HTML

Цель занятия

Целью занятия является изучение принципов создания программ (скриптов) JavaScript в документах HTML. Приводится синтаксис программ на языке JavaScript, показываются способы вызова функций *скриптов* на примере небольших клиентских программ. Более подробную информацию можно получить в [4, 13].

Краткие теоретические сведения

В документы HTML можно вставлять программы (скрипты), написанные на языках JavaScript и VBScript. В этом практическом пособии мы будем использовать программы на языке JavaScript. Для вставки скрипта в тело HTML-документа понадобится придерживаться следующего синтаксиса:

```
<html>
<script>
<!--
  function hello()
    { alert("Hello from script");}
-->
</script>
<body bgcolor=#AABBCC>
<form name="frm1">
```

```
<input type="Button" value="Press to activate hello"
onClick="hello()">
</form>
</body>
</html>
```

Тело скрипта представлено одной-единственной функцией `hello()`. Данная функция просто выводит сообщение "Hello from script". Вызов функции, как правило, привязывается к событию `onClick` (щелчок мышью на кнопке или другом элементе). Этот вызов помещен в теге следующим образом:

```
<input type="Button" value="Press to activate hello"
onClick="hello()">
```

Имеются и другие *события*:

- `onLoad` — загрузка документа;
- `onFocus` — получение элементом фокуса (при щелчке мышью);
- `onBlur` — потеря элементом фокуса;
- `onSelect` — выделение текста в поле ввода;
- `onOver` — перемещение курсора мыши над элементом;
- `onChange` — выбор нового элемента выпадающего списка;
- `onExit` — выход курсора за границы элемента и др.

Синтаксис JavaScript аналогичен синтаксису языка C. *Скрипт* ограничивается тегами `<script> … </script>`. Тело скрипта записывается как комментарий — начало комментария открывает тег `<!--` , завершение комментария — тег `-->` . Если теги комментария убрать, то скрипт будет выполняться немедленно при загрузке документа. Скрипт, помещенный в комментарий, выполняется по событию. В нашем примере таким событием является `onClick` (щелчок мышью на кнопке). Получение значений элементов формы выполняется по ссылкам вида:

```
document.имя_формы.имя_элемента.value
```

Наоборот, присваивание значения элементу выполняется по следующему шаблону:

```
document.имя_формы.имя_элемента.value=значение
```

В теле скрипта записывают, вообще говоря, несколько функций, каждая из которых может вызвать любую другую. Функциям, как правило, передают аргументы. В JavaScript нет необходимости указывать типы переменных. Они определяются из контекста. *Переменные*, объявленные до объявления функций, являются *глобальными*. Такие переменные попадают в область видимости каждой функции. Переменные, объявленные внутри какой-либо функции, являются *локальными*. Значения таких переменных доступны только в пределах той функции, в которой данные переменные объявлены. Рассмотрим пример, в котором на форме будут располагаться два списка типа `Select`. В одном из них перечисляются цены товаров, во втором — их количества. Пользователь осуществляет выбор цены товара щелчком мыши, выбранная цена умножается на соответствующее количество, а результат помещается в текстовое поле. Вот пример этого скрипта (листинг 2.1).

Листинг 2.1. Вычисления общей стоимости товара. Скрипт в теле HTML-документа

```
<HTML>
     <body bgcolor=#aabbcc> <!--задание цвета фона
     шестнадцатеричным числом-->
   <script>
<!--
   function calc(x,y)  // функция calc() находит произведение
                       // своих аргументов
    {
document.forma1.tf.value=""+x*y;// результат помещается
                                // в текстовом поле tf
     }
  -->
    </script>
    <form name="forma1">
      <select name=s1
onChange="calc(document.forma1.s1.options

[document.forma1.s1.selectedIndex].value,document.forma1.s2.op
tions
```

```
[document.forma1.s1.selectedIndex].value)">
// вызов функции calc() выполняется при выборе элемента
// выпадающего списка— событие onChange
      <option name=op1 value=1000> 1000</option>
      <option name=op2 value=2000> 2000</option>
      <option name=op3 value=3000> 3000</option>
    </select>
     <select name=s2>
      <option name=op12 value=10> 10</option>
      <option name=op22 value=20> 20</option>
      <option name=op32 value=30> 30</option>
     </select>
  <input type="TextField" name=tf value="">
  </form>
  </body>
</html>
```

В этом примере мы описываем выпадающий список таким образом:

```
      <select  name=s1
onChange="calc(document.forma1.s1.options
[document.forma1.s1.selectedIndex].value,document.forma1.s2.op
tions
[document.forma1.s1.selectedIndex].value)">
// вызов функции calc  выполняется при выборе элемента
// выпадающего списка — событие onChange
      <option name=op1 value=1000> 1000</option>
      <option name=op2 value=2000> 2000</option>
      <option name=op3 value=3000> 3000</option>
    </select>
```

Опции выпадающего списка описываются индивидуально: указывается имя опции (`name=`), значение опции (`value=`) и значение, отображаемое в выпадающем списке для этой опции (оно, вообще говоря, не обязано совпадать с `value`).

Записана реакция на выбор нового элемента выпадающего списка:

```
onChange="calc(document.forma1.s1.options
```

```
[document.forma1.s1.selectedIndex].value,
document.forma1.s2.options
[document.forma1.s1.selectedIndex].value)">
```

Видим, что для получения значения опции (параметра `value`) следует указать ссылку вида:

```
document.forma1.s1.options[document.
forma1.s1.selectedIndex].value
```

Здесь в квадратных скобках указывается индекс выделенного элемента списка — адресация выполняется через ключевое слово `selectedIndex`. За словом `document` следует разделяемая знаком точки последовательность имен элементов — сначала *формы* (`forma1`), затем имени списка (`s1`), затем ссылки на опцию списка посредством указания не имени, а `options[индекс]` — такой вариант также допустим.

Еще одним элементом формы в этом скрипте является текстовое поле:

```
<input type="TextField" name=tf  value="">
```

Значение этого текстового поля задано в виде пустой строки.

Приведем пример еще одного скрипта (листинг 2.2), который демонстрирует игру "крестики-нолики" (размер доски 3×3). В этой игре нам нужно помнить очередность ходов игроков. При нажатии на кнопку на ней отображается "крестик" или "нолик" в зависимости от этой очередности. Соответствующую переменную `xod` мы объявляем глобальной. Наш вариант скрипта во многом несовершенен, поскольку допускает писать "крестики" или "нолики" на уже использованных клетках таблицы. Результат выполнения скрипта помещен на рис. 2.3.

Листинг 2.2. Скрипт для игры в "крестики-нолики", размещенный в теле HTML-документа

```
      <HTML>
     <body bgcolor=#aabbcc>
       <script>
<!--
  var xod=1;
```

```
function clear1()
        {
              j=0;
           while(j<=9){
     if(document.forma1.elements[j].name==10)
        {
                  j++;
                  continue;}
     document.forma1.elements[j].value="   ";
                 j++;
                   }
                 xod=1;
           return;
                }
                function show(x)
              {
                 j=0;
                while(j<10)
                 {
                   if (document.forma1.elements[j].name==x)
                    {
                      if(xod==1)
                        {document.forma1.elements[j].value="
                        x ";
                         xod=2;
                         break;
                        }
                      else
                      {document.forma1.elements[j].value="
                      0 ";
                       xod=1;
                       break;
                      }
                   }
                   j++;
```

```
                }
              }
            -->
            </script>
    <form name="forma1">
    <table border=2 bgcolor=#aaccff>
    <tr>
<td><input type="Button" name=1 value="   "
onClick="show('1')"></td>
<td><input type="Button" name=2 value="   "
onClick="show('2')"></td>
<td><input type="Button" name=3 value="   "
onClick="show('3')"></td>
     </tr>
     <tr>
<td><input type="Button" name=4  value="   "
onClick="show('4')"></td>
<td><input type="Button" name=5  value="   "
onClick="show('5')"></td>
<td><input type="Button" name=6  value="   "
onClick="show('6')"></td>
     </tr>
     <tr>
<td><input type="Button" name=7 value="   "
onClick="show('7')"></td>
<td><input type="Button" name=8 value="   "
onClick="show('8')"></td>
<td><input type="Button" name=9 value="   "
onClick="show('9')"></td>
     </tr>
    </table>
<br><br>
<Input type="Button" name=10 value="Clear" onClick="clear1()">

 </form>
 </body>
</html>
```

Рис. 2.3. Игра в "крестики-нолики"

Игровая доска реализуется как таблица HTML:

```
     <table border=2 bgcolor=#aaccff>
       <tr>
<td><input type="Button" name=1 value="   "
onClick="show('1')"></td>
<td><input type="Button" name=2 value="   "
onClick="show('2')"></td>
<td><input type="Button" name=3 value="   "
onClick="show('3')"></td>
       </tr>
       <tr>
<td><input type="Button" name=4  value="   "
onClick="show('4')"></td>
<td><input type="Button" name=5  value="   "
onClick="show('5')"></td>
<td><input type="Button" name=6  value="   "
onClick="show('6')"></td>
```

```
        </tr>
        <tr>
<td><input type="Button" name=7 value="   "
onClick="show('7')"></td>
<td><input type="Button" name=8 value="   "
onClick="show('8')"></td>
<td><input type="Button" name=9 value="   "
onClick="show('9')"></td>
        </tr>
       </table>
```

В каждой ячейке таблицы размещается кнопка:

```
<td><input type="Button" name=1 value="  " onClick=
"show('1')"></td>
```

Для каждой кнопки предусмотрен вызов функции `show()`, в которую передается имя кнопки:

```
onClick="show('7')"
```

Теперь рассмотрим функцию `show()`:

```
function show(x)
{
        j=0;
 while(j<10)
 {
  if (document.forma1.elements[j].name==x)
   {
     if(xod==1)
    {document.forma1.elements[j].value=" x ";
     xod=2;
     break;
    }
     else
   {document.forma1.elements[j].value=" 0 ";
     xod=1;
     break;
   }
  }
```

```
    j++;
  }
```

В цикле `while` выполняем просмотр всех *элементов формы* (элементы имеют индексы от 0 до 9). Поскольку `x` — номер кнопки, то сравниваем его со значением параметра `name`, которому присвоено значение номера кнопки:

```
if (document.forma1.elements[j].name==x)
```

При совпадении проверяем, какой символ, "крестик" или "нолик", нарисовать на элементе (в зависимости от переменной `xod`):

```
if(xod==1)
 {
  document.forma1.elements[j].value=" x ";
  xod=2;
  break;
 }
else
 {
  document.forma1.elements[j].value=" 0 ";
  xod=1;
  break;
 }
```

Команда `break` реализует выход из цикла.

Функция `clear()` осуществляет стирание надписей на кнопках, сделанных в процессе игры.

Подробное изучение языка JavaScript не входит в наши задачи. Заинтересованный читатель может адресоваться к списку литературы для ознакомления, помещенному в конце пособия. В этой работе рассмотрим, как создать анимацию. Для этого нам потребуется запускать программы по *таймеру*. Это делается так:

```
timerID= setTimeout("functionname()",1000);
```

Эта команда запускает функцию `functionname()` через интервал в 1000 мс (т. е. 1 с). Следовательно, если рекурсивно вызывать этот оператор, данная функция будет активизироваться с периодом в 1 с.

Кроме этого, нам потребуется сбрасывать таймер, поскольку если он не сброшен, то использование рекурсивных вызовов таймера приведет к переполнению памяти. Делается это таким образом:

```
clearTimer(timerID);
```

Здесь `timerID` — идентификатор ранее созданного таймера.

Теперь можно привести полный код сайта (листинг 2.3).

Листинг 2.3. Скрипт для динамического отображения времени

```
<html>
<form name="for1">
 <Input type="TextField" name="txt1" size=60 value="00:00"/>
<SCRIPT>
<!--
  var timerID=null;
  var timerRunning=false;
  function stopclock()
    {
     if(timerRunning)
      { clearTimeout(timerID);}
        timerRunning=false;
        return ;
    }
   function startclock()
      {
    stopclock();
    showtime();
    return ; //оператор выхода из функции
      }
    function showtime()
   {
     var now= new Date();
```

```
      for1.txt1.value= now.toLocaleString();
      timerID= setTimeout("startclock()",1000);
      timerRunning=true;
   }
 -->
  </Script>
  </Form>
<Body bgcolor=#aabbff onLoad="startclock()">
</body>
</html>
```

Данный *скрипт* просто отображает текущее время в текстовом поле. Это делается по команде:

```
for1.txt1.value= now.toLocaleString();
```

Ссылка на текстовое поле осуществляется в следующей последовательности: имя формы — имя текстового поля — `value`. Объектная переменная `now` держит текущее время. Эта переменная получает значение в строке:

```
var now= new Date();
```

Запуск скрипта происходит по событию `onLoad` (загрузки окна документа). Вызывается функция `startclock()`. Проследите сами, как развиваются действия дальше.

Для прорисовки сменяющих друг друга картинок следует подготовить две картинки, которые будут последовательно накладываться одна на другую. Для размещения картинки в сайте нужно использовать контейнер вида:

```
<img src="address" width=400 height=200 visibility=
"hidden" name="img1">
```

Для смены картинки следует выполнить команду:

```
document.images[0].src="newaddress";
```

Вам нужно продумать, как реализовать эту команду в скрипте.

Важной возможностью языка JavaScript является работа с документом cookie. Под cookie следует понимать вспомогательную и учетную информацию (например, адрес сайта, время сохранения сайта, число его посещений и др.), передаваемую браузером от

клиента к серверу и в обратном направлении. Информция в `cookie` записывается в виде `parameter=value;parameter=value`. Это позволяет легко получить нужное значение. Строку cookie можно получить в программе с помощью оператора:

```
s= document.cookie;
```

Значение же cookie можно сформировать, например, операторами:

```
document.cookie="name=Petrov; ";
document.cookie="age=45; ";
```

Для того чтобы распаковать значение параметра, нужно использовать операторы `indexOf` и `substring`. Оператор `indexOf` возвращает номер позиции в строке, с которого данная подстрока впервые входит в строку; оператор `substring` формирует из строки подстроку, расположенную в исходной строке, начиная с указанного места и включая указанное количество символов. Так, с помощью команд

```
s=document.cookie;
i=s.indexOf("name");
```

получаем в переменной `i` номер позиции в `s`, с которой начинается слово `name`. Далее используем команды:

```
s1=s.substring(i+4, s.length);
i=s1.indexOf("=");
s2=s1.substring(i+1, s1.length);
i=s2.indexOf(";");
s3=s2.substring(1,i-1);
```

Этот последний фрагмент позволяет получить в `s3` строковое значение "Petrov".

Задание

Подготовьте две или более сменяющие друг друга картинки и, используя материал, изложенный выше, продемонстрируйте эффект анимации сайта. Сделайте так, чтобы адреса картинок находились в cookie и чтобы для работы было необходимо предварительно эти адреса оттуда извлечь. Содержимое cookie сформируйте по событию `onLoad` загрузки документа, т. е. в HTML-документе укажите:

```
<body bgcolor=#AABBCC onLoad="formcookie()">
```

```
...
```

где `formcookie()` — функция для формирования cookie.

Контрольные вопросы

1. Для чего предназначены скрипты?
2. Как скрипт вставляется в тело HTML-документа и как он выполняется?
3. Как определяются функции в JavaScript?
4. Как осуществляется возврат значений из функций?
5. Приведите пример вызова функции по событию выбора элемента списка.
6. Укажите способ объявления таймера.
7. Почему таймер следует уничтожать при использовании рекурсивных вызовов?
8. Укажите, каким образом получить значение элемента списка, текстового поля?
9. Как использовать cookie?
10. Как получить значения параметров cookie?

Введение в Java

Цель занятия

Ознакомление с основами программирования в среде Java. Предполагается изучение краткого теоретического введения и разработка программы рисования графика наподобие изображенного на рис. 2.4. Работа связана с более подробным изучением апплетов, их методов и способов вывода информации. Рекомендуемая литература [1, 2, 11, 15, 17].

Следует обратить внимание на приводимые в этом разделе сведения о командах для создания рисунков, что используется в последующем.

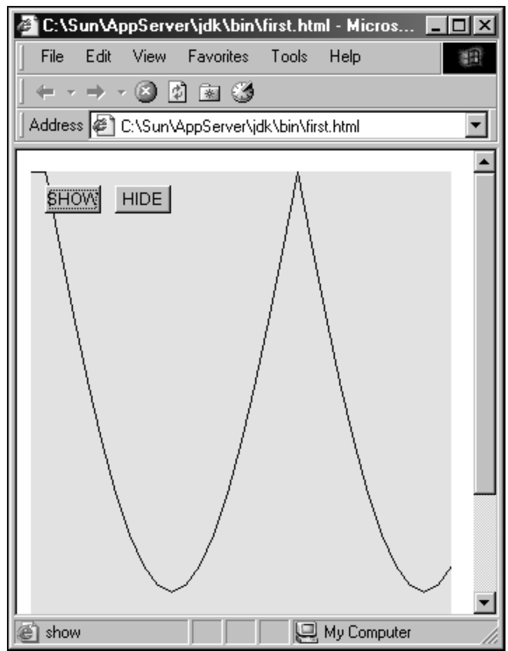


Рис. 2.4. Апплет для рисования графика

Краткие теоретические сведения

Основным достоинством Java является возможность программировать в сетевой среде, например, в Интернете. Java позволяет создавать приложения на основе технологии "клиент-сервер". Приложение-сервер размещается на мощном сетевом компьютере и обрабатывает запросы клиентов, выполняющихся на локальных ЭВМ.

Приложения для Интернета называются *апплетами*, и один из них мы создадим на этом занятии. Приложения Java, не предназначенные для работы в Интернете, являются приложениями, основанными на фреймах (или формах, если они используют базовый класс `Frame` или `JFrame`.

Апплет запускается на выполнение браузером Internet Explorer, когда браузер читает созданный нами HTML-файл. Апплет состоит из секций: `start()`, `init()`, `stop()`, `destroy()`, `paint()`.

Не все секции следует указывать в апплете в общем случае. Секция `init()` выполняется только один раз при первом запуске апплета. Секция `start()` выполняется каждый раз при рестарте апплета из состояния ожидания. Секция `stop()` запускается при выходе из апплета, но при его сохранении. Секция `destroy()` запускается при окончательном закрытии окна апплета. Ясно, что ввиду разного временного порядка вызова этих секций их программирование существенно разнится по целям. Секция `paint()` служит для перерисовки изображения в окне апплета. Именно эта секция потребуется для рисования графика.

Программа на языке Java состоит из классов. Главный класс апплета должен быть объявлен, например, так:

```
public class lab1 extends Applet
```

Здесь `lab1` — имя класса. Ключевое слово `public` является модификатором доступа. Главный класс апплета должен иметь тип `public`, т. е. быть общедоступным. Другим возможным вариантом является отсутствие модификатора доступа, тогда класс доступен только из того пакета, в котором он описан. Java для каждого public-класса после компиляции создает файл с расширением class, причем имя этого файла совпадает с именем исходного Java-файла. Имя исходного Java-файла должно совпадать с именем класса, объявленного как `public`. В нашем примере следует сохранить текст программы для апплета в файле labl.java. Таким образом, логично создавать в программе только один публичный класс.

Программа, как правило, использует стандартные функции. Стандартные функции применяются для ввода-вывода, работы с файлами, событиями, сетевыми адресами и пр. Для подключения стандартных функций в программу нужно предварительно импортировать их с помощью инструкций следующего типа:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
```

Пакет java.applet.* содержит классы для поддержания работы с апплетами. Пакет java.awt.* содержит классы для обработки системных событий и пр.

Приведем пример апплета, результатом выполнения которого является приветствие, выведенное в окне браузера (листинг 2.4).

Листинг 2.4. Апплет, выдающий приветствие

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class lab1 extends Applet
{
 public void paint(Graphics g)
 {
  g.drawString("HELLO",10,10);
 }
}
```

Откомпилируйте этот апплет, задав в командной строке операционной системы (DOS): javac lab1.java. Единственная эффективная команда `g.drawString("HELLO",10,10)` приведенной программы служит для вывода строки (первый аргумент) в окно апплета с указанной позиции (`x`-координата, `y`-координтата). Переменная `g` используется для хранения графического контекста (тип `Graphics`). Запомните эту команду.

При работе с пакетом JSDK 1.* вам потребуется какой-нибудь редактор для DOS (например, Norton Commander или FAR). Если ошибок синтаксиса нет, то будет создан файл labl.class. Для запуска апплета на выполнение вам потребуется создать HTML-файл (файл с расширением html). Вот его возможный вид:

```
<html>
<applet code="lab1.class"
width=400
height=400>
</applet>
</html>
```

Параметр `code` задает главный класс апплета. Сохраните этот файл под любым именем, но с расширением html. Для запуска апплета щелкните дважды по имени этого файла.

Команды языка Java имеют синтаксис C. Указатели не используются. Определение класса помещается в фигурные скобки. Теперь приведем полный текст программы для рисования графика (листинг 2.5) и дадим к нему пояснения.

Листинг 2.5. Приложение на основе апплета, выполняющее рисование графика

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class lab1 extends Applet implements ActionListener
{
 Button btnshow=new Button("SHOW");
 Button btnhide=new Button("HIDE");
 String msg="";

 public void paint(Graphics g)
 {
  int x,y,xold,yold,step=10;
  if (msg.equals("show"))
   {
    xold=0;
    yold=(int)Math.abs(Math.sin(Math.PI*xold/180)*
                      (getSize().height-  100));
    while (xold<399)
    {
     x=xold+step;
     y=(int)Math.abs( Math.sin(Math.PI*xold/180)*
                    (getSize().height-100));
     g.drawLine(xold,yold,x,y);
     xold=x;
     yold=y;
    }
   }
```

```
 }
 public void graph(String msg1)
 {
  Graphics g=getGraphics();
  if(msg1.equals("show"))
   repaint();
  else
   if (msg1.equals("hide"))
    {
     g.setColor(getBackground());
     g.fillRect(0,0,getSize().width,getSize().height);
    }
  showStatus(msg1);
 }
public void init()
 {
  setLayout(null);
  add(btnshow);
  add(btnhide);
  btnhide.setBounds(60,10,40,20);
  btnshow.setBounds(10,10,40,20);
  btnshow.addActionListener(this);
  btnhide.addActionListener(this);
  setBackground(Color.yellow);
 }

 public void actionPerformed(ActionEvent a_e)
 {
  if (a_e.getSource()==btnshow)
   {
    msg="show";
    graph(msg);
   }
  else
```

```
   if (a_e.getSource()==btnhide)
    {
     msg="hide";
     graph(msg);
    }
 }
}
```

Разберем более подробно код этой программы.

Определение *главного класса апплета* таково:

```
public class lab1 extends Applet implements ActionListener
```

Слова `implements ActionListener` используются для подключения в класс апплета интерфейса `ActionListener`, который используется для написания реакции на нажатие кнопок и других элементов визуального интерфейса. В этом интерфейсе имеется метод:

```
public void actionPerformed(ActionEvent a_e),
```

который и определяет реакцию на нажатие кнопок.

Все используемые в программе переменные, кнопки и пр. должны быть объявлены. Как правило, объявления помещаются перед описанием методов. В нашем примере эти объявления таковы:

```
Button btnshow=new Button("SHOW");
Button btnhide=new Button("HIDE");
String msg="";
```

Здесь объявлены две кнопки: `btnshow`, `btnhide` (тип `Button`) и строка `msg` (тип `String`). Java различает заглавные и строчные буквы, поэтому `Button` и `button` суть разные объекты. Заметим, что в первых двух объявлениях программные кнопки прямо создаются *конструкторами* класса `Button`: `new Button("SHOW")`. Аргументом команды является заголовок кнопки. Но можно было разделить объявление и создание кнопки:

```
Button btnshow;
btnshow= new Button("SHOW");
```

В логическом порядке далее следует рассмотреть метод `init()` *апплета*, где объявленные и созданные кнопки добавляются в окно апплета.

```
public void init()
{
 setLayout(null);
 add(btnshow);
 add(btnhide);
 btnhide.setBounds(60,10,40,20);
 btnshow.setBounds(10,10,40,20);
 btnshow.addActionListener(this);
 btnhide.addActionListener(this);
 setBackground(Color.yellow);
}
```

Добавление кнопок выполняется командами `add(btnshow)`, `add(btnhide)`; установка размеров и координат кнопок — командами `btnhide.setBounds(60,10,40,20)`, `btnshow.setBounds(10,10,40,20)`; установка прослушивателя событий от кнопок — командами `btnshow.addActionListener(this)`, `btnhide.addActionListener(this)`.

Команда `setBackground(Color.yellow)` устанавливает цвет фона окна. Цвет — это объект класса `Color`. Поэтому в команду установки цвета передан аргумент `Color.yellow`. Команда `setLayout(null)` указывает, что для размещения кнопок в окне апплета не надо применять стандартный класс для размещения — на это указывает параметр `null`. В любом случае параметр `null` определяет, что объект или переменная не имеет значения.

Программа реакции на события от кнопок такова:

```
public void actionPerformed(ActionEvent a_e)
{
 if (a_e.getSource()==btnshow)
  {
   msg="show";
   graph(msg);
```

```
  }
 else
  if (a_e.getSource()==btnhide)
   {
    msg="hide";
    graph(msg);
   }
}
```

Единственный аргумент этой функции `a_e` имеет тип `ActionEvent`. Данный тип предоставляет метод `getSource()`, который возвращает источник события. Нетрудно сообразить, как этот метод используется и для чего. Видим, что в зависимости от того, какая кнопка нажата, переменная `msg` получает значение `"show"` (показать график) или `"hide"` (скрыть график). Переменная `msg` передается в качестве аргумента в метод `graph()`. Этот метод является единственным нестандартным методом в классе. Он анализирует, какое сообщение (`msg1`) пришло, и производит адекватную реакцию. Если пришло сообщение `"hide"`, то уничтожение графика выполняется следующим образом:

```
else
 if (msg1.equals("hide"))
  {
   g.setColor(getBackground());
   g.fillRect(0,0,getSize().width,getSize().height);
  }
```

Команда `g.fillRect(0,0,getSize().width,getSize().height)` заливает прямоугольник с координатами `(0, 0, getSize().width, getSize().height)` цветом фона, предварительно установленным командой `g.setColor(getBackground())`. Напомним, что `g` — это переменная графического контекста, а способ ее получения таков:

```
Graphics g=getGraphics();
```

Метод `getGraphics()` принадлежит классу апплета. Этому же классу принадлежат методы `getSize().width`, `getSize().height`, которые получают размеры окна апплета.

Как видим, метод `graph()` вызывает перерисовку окна апплета посредством обращения к `repaint()`, который уже непосредственно вызывает метод `paint()`. В методе `paint()` рисование линии графика выполняется в цикле `while`:

```
{
 xold=0;
 yold=(int)Math.abs(
 Math.sin(Math.PI*xold/180)*(getSize().height-100));
 while (xold<399)
 {
  x=xold+step;
  y=(int)Math.abs(
  Math.sin(Math.PI*xold/180)*(getSize().height-100));
  g.drawLine(xold,yold,x,y);
  xold=x;
  yold=y;
 }
}
```

Класс `Math` содержит набор методов и констант для выполнения алгебраических операций и преобразований. Пример обращения к этому классу дает команда:

```
Math.sin(Math.PI*xold/180)
```

которая вычисляет синус угла (в радианах) `PI*xold / 180`; а сам угол `xold` задается в градусах. Обратим внимание также на *приведение типа* в выражении `y=(int)Math.abs(...)` — правая часть приводится к типу `int` (целый).

Итак, программа рисования графиков разобрана.

В заключение теоретической части приведем сведения по некоторым важным графическим методам класса `Graphics`:

- `public void drawRect(int x,int y, int width, int height)` — рисует прямоугольник, левый верхний угол которого имеет координаты `(x, y)`, ш ирина — `width`, высота — `height`;
- `public void drawLine(int x1,int y1, int x2, int y2)` — рисует линию, связывающую две точки с заданными координатами;

- `public void setColor(Color a)` — устанавливает цвет рисуемых графических объектов. Цвет апплета можно установить методом `setBackground(Color c)`, который принадлежит классу апплета;
- `public void drawOval(int x,int y, int width, int height)` — рисует овал, вписанный в прямоугольник с заданными координатами;
- `public void fillOval(int x,int y, int width, int height)` — заполняет прямоугольник текущим цветом, установленным командой `setColor`;
- `public void drawArc(int x,int y, int width, int height,int startangle,int endangle)` — рисует дугу (часть окружности), вписанную в прямоугольник заданного размера, заключенную между двумя лучами с углами `int startangle`, `int endangle`.

Приведем пример использования:

```
import java.applet.*;
import java.awt.*;
public class t1 extends Applet
{
 public void paint(Graphics g)
  {
   for(int x=10; x<200; x+=20)
   {
    g.drawArc(20,30,20+x,30+x,0,270);
   }
  }
}
```

Рисование многоугольника произвольной формы обеспечивает метод:

`public void drawPolygon(int[] xpoints, int []ypoints, int number).`

`number` задает число вершин, координаты вершин берутся из массивов `xpoints` (`x`-координаты) и `ypoints`(`y`-координаты).

Заполнение многоугольника цветом выполняет метод:

```
public void drawPolygon(int[] xpoints, int [] ypoints, int number)
```

с тем же набором аргументов. Пример объявления и использования:

```
int x[]={200,230,400,430};
int y[]={100,20,40,60};
int num=4;
Graphics g=getGraphics();
g.drawPolygon(x,y,num);
```

Для получения рисунка его нужно предварительно объявить:

```
public Image img;
```

Далее нужно загрузить рисунок в память:

```
img=getImage(URL u, "name.gif");
```

Переменная `u` типа `URL` должна указывать на *сетевой адрес* рисунка. Если рисунок находится там же, где и апплет, то следует использовать команду:

```
img=getImage(getDocumentBase(),"name.gif");
```

Теперь остается отобразить рисунок:

```
public boolean drawImage(Image img, int x, int y, ImageObserver ob)
```

В качестве последнего параметра команды `drawImage()` следует указать `this`.

Задание

В настоящей работе требуется написать программу для рисования важнейших зависимостей теории массового обслуживания, которые указываются в вариантах заданий. Требуется также предусмотреть возможность задания масштаба графика.

1. Нарисовать график не сплошной непрерывной линией, а маленькими кружками.
2. Нарисовать график $f(t)= \lambda \times e^{-\lambda t}$, задающий плотность пуассоновского закона распределения вероятностей, где λ представляет интенсивность потока событий; `t` — время.

3. Нарисовать график $P(t,k)=e^{-\lambda*t}\times(\lambda t)\times k\ /(k!)$ значения вероятности наступления k событий за время t при интенсивности потока событий в единицу времени λ.
4. Нарисовать график для формулы вероятности состояния простоя одноканальной системы массового обслуживания с числом мест ожидания m:

   $P0=(1+\rho+\rho^2+...+\rho^{m+1})^{-1}$, где $\rho = \lambda/\mu$ и μ определяет интенсивность обслуживания заявок.
5. Нарисовать график $e^t = 1+t/1! + t^2/2! + ..$ для различных степеней приближения.

   Предусмотреть возможность масштабирования графика в отношении 1:1, 1:2, 1:10.

Контрольные вопросы

1. Как создать апплет?
2. Каковы стандартные методы, используемые апплетом в цикле жизни? Для чего они служат?
3. Как осуществляется прорисовка апплета?
4. Каким способом можно вывести строку в апплете?
5. Объяснить, как можно отобразить в апплете рисунок?
6. Объяснить программу для рисования графика.
7. Каким образом установить цвет и толщину линии на графике?
8. Что выполняет команда `setBounds()`?
9. Что выполняет команда `setLayout()`?

Реализация взаимодействия между апплетами

Цель занятия

Целью занятия является изучение механизма передачи информации из одного апплета в другой, а также способа вызова методов одного апплета из другого и механизм передачи параметров. Дополнительно можно рекомендовать [2, 17].

Краткие теоретические сведения

Взаимодействие апплетов предполагает передачу информации из одного апплета в другой. Далее приведен текст главного апплета, который обращается к вспомогательному (второму) апплету (листинг 2.6).

Листинг 2.6. Текст главного апплета

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class lab2 extends Applet implements ActionListener
{
 private String msg="Red";
 CheckboxGroup cbg=new CheckboxGroup();
 Checkbox cbRed= new Checkbox("Red",cbg,true);
 Checkbox cbBlue= new Checkbox("Blue",cbg,false);
 Checkbox cbYellow= new Checkbox("Yellow",cbg,false);
 Button btn=new Button("Send Color");
 public void init()
 {
  setLayout(null);
  add(cbRed);
  add(cbBlue);
  add(cbYellow);
  add(btn);
  btn.setBounds(10,10,80,18);
  cbRed.setBounds(10,30,50,18);
  cbBlue.setBounds(10,50,50,18);
  cbYellow.setBounds(10,70,50,18);
  btn.addActionListener(this);
 }
 public void actionPerformed(ActionEvent e)
 {
  Applet receiver=null;
```

```
  String receiverName="ReceiverApplet";
  receiver=getAppletContext().getApplet(receiverName);
  if(receiver !=null)
   {
    if (!(receiver instanceof Receiver))
     {
      showStatus("Not an applet we have been looking for");
     }
    else
     {
      showStatus("OK");
      ((Receiver) receiver).processRequest(returnValue());
     }
   }
  else
  showStatus("Can't find the specified applet");
 }

 public String returnValue()
 {
  if (cbRed.getState())
   return "Red";
  else
   if (cbBlue.getState())
    return "Blue";
   else return "Yellow";
 }
}
```

Соответствующий этой программе HTML-файл имеет следующий вид:

```
<html>
<body>
<APPLET   code="lab2.class"
          width=300
          height=100>
```

```
</APPLET>
<APPLET    code="Receiver.class"
           width=300
           height=100>
<PARAM name="name" value="ReceiverApplet">
</APPLET>
</Body>
</Html>
```

Видим, что в этом HTML-файле определены два апплета, соответствующие двум различным `public`-классам, которые запускаются браузером одновременно. Пока обратим внимание лишь на то, что во втором апплете задается тег: `<PARAM name="name" value="ReceiverApplet">`. Запомните, в Java имеется специальная команда, которая возвращает в качестве результата апплет — т. е. целый класс со своими методами и переменными. Это команда `getApplet()`. В качестве аргумента данная команда использует значение `value` из тега `PARAM`. Теперь найдем то место в программе, где данная команда применена. Вот оно:

```
Applet receiver=null;
String receiverName="ReceiverApplet";
receiver=getAppletContext().getApplet(receiverName);
if(receiver !=null)
 {
  if (!(receiver instanceof Receiver))
   {
    showStatus("Not an applet we have been looking for");
   }
  else
   {
    showStatus("OK");
    ...
```

Рассмотрим этот фрагмент подробно. Во-первых, он содержится в методе реализации события выбора одного из флажков ведущего апплета, после того как оба апплета будут отображены браузером (рис. 2.5).

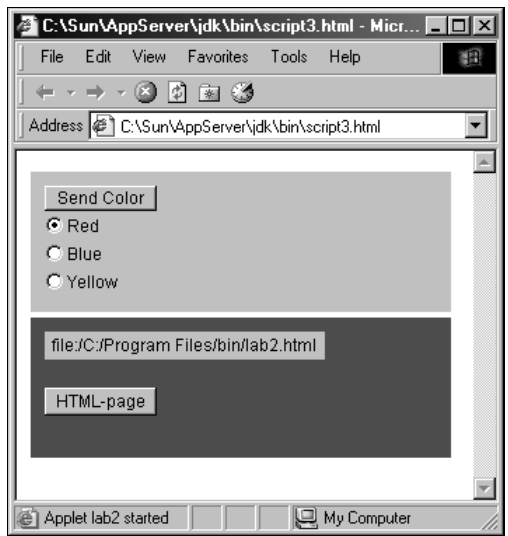


Рис. 2.5. Два взаимодействующих апплета

При выборе соответствующего флажка и нажатии кнопки **SEND** второй апплет окрашивается в тот цвет, который соответствует радиокнопке.

В строке `Applet receiver=null` объявляется `receiver` как переменная типа `Applet`, которой присваивается изначально пустое значение `null`.

Следующие две строки:

```
String receiverName="ReceiverApplet";
receiver=getAppletContext().getApplet(receiverName);
```

объявляют строковую переменную `receiverName` и возвращают в переменную `receiver` указатель на апплет, содержащий в параметре `value` значение переменной `receiverName`, т. е. "`ReceiverApplet`". Запомним эту команду:

```
receiver=getAppletContext().getApplet(receiverName).
```

Далее в программе выполняется проверка того, что получен действительно нужный апплет:

```
{
 if (!(receiver instanceof Receiver))
```

```
  {
   showStatus("Not an applet we have been looking for");
  }
 else
 {
  showStatus("OK");
```

Обратим внимание на следующее. `Receiver` — это экземпляр (объект) класса `Receiver`, определенного в файле Receiver.java. Команда:

```
if (!(receiver instanceof Receiver))
```

как раз и проверяет, что `receiver` является экземпляром класса `Receiver`. Результаты проверки просто отображаются в виде информационной строки в поле статуса апплета командой `showStatus()`. Если результат проверки положителен, то вслед за командой

```
showStatus("OK");
```

выполняется "главная команда":

```
((Receiver) receiver).processRequest(returnValue());
```

Эта команда вызывает метод `processRequest()` из другого апплета и передает ему единственный параметр — результат выполнения метода `returnValue()`. Вызов варианта `receiver.processRequest(returnValue())` принципиально не пройдет, так как Java будет считать в этом случае `receiver` классом или экземпляром класса, объявленного в данном файле или в одном из импортируемых файлов, но никак не в другом апплете. Так что следует запомнить и этот вариант вызова. Метод `returnValue()` не требует каких-либо дополнительных разъяснений. Рассмотрим теперь файл Receiver.java (листинг 2.7).

Листинг 2.7. Класс `Receiver`

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
public class Receiver extends Applet
```

```
{
 private String receivedMsg="Red";
 private Label label=new Label("zdem",Label.CENTER);
 public Button btnShow= new Button("HTML-page");
 URL myurl=null;
 String s;

 public void init()
 {
  setLayout(null);
  add(label);
  add(btnShow);
  label.setBounds(10,10,200,20);
  btnShow.setBounds(10,50,80,20);
  showStatus("init is working");
  repaint();
  try
  {
   s="file:/C:/Program Files/bin/lab2.html";
   myurl=new URL(s);
   label.setText(""+ myurl);
  }
  catch(MalformedURLException e)
   {
    showStatus("error of url"+e.getMessage());
    label.setText("Badforurl");
   }
 }
 public boolean action(Event evt,Object arg)
 {
  if (evt.target instanceof Button)
   {
    label.setText("changing url");
```

```
    getAppletContext().showDocument(myurl,"_blank");
    return(true);
   }
  return false;
 }
 public void paint(Graphics g)
 {
  g.drawString("Hello",20,50);
  if (receivedMsg.equals("Red"))
   {
    g.setColor(Color.red);
   }
  else
   if (receivedMsg.equals("Blue"))
    {
     g.setColor(Color.blue);
    }
   else
    g.setColor(Color.yellow);
  g.fillRect(0,0,getSize().width,getSize().height);
 }

 public void processRequest(String senderName)
 {
  label.setText(senderName);
  receivedMsg=senderName;
  repaint();}
 }
```

Начнем с главного метода:

```
public void processRequest(String senderName)
{
 label.setText(senderName);
```

```
 receivedMsg=senderName;
 repaint();}
}
```

Данный метод отображает в текстовом поле `label` принятый текст из главного апплета, присваивает его переменной `receivedMsg` и перерисовывает апплет с помощью команды `repaint()`, вызывающей метод `paint()` апплета. Рассмотрим метод `paint()`:

```
public void paint(Graphics g)
{
 g.drawString("Hello",20,50);
 if (receivedMsg.equals("Red"))
  {
   g.setColor(Color.red);
  }
 else
  if (receivedMsg.equals("Blue"))
   {
    g.setColor(Color.blue);
   }
  else
   g.setColor(Color.yellow);
  g.fillRect(0,0,getSize().width,getSize().height);
}
```

Изменение цвета апплета выполняет команда `g.setColor()`, которая устанавливает цвет для заливки, а саму заливку выполняет команда `g.fillRect(0,0,getSize().width,getSize().height)`, знакомая нам по первому практическому занятию. В заключение рассмотрим следующий метод:

```
public boolean action(Event evt, Object arg)
{
 if (evt.target instanceof Button)
  {
```

```
    label.setText("changing url");
    getAppletContext().showDocument(myurl,"_blank");
    return(true);
   }
 return false;
}
```

Данный метод применяется для описания реакции на какое-либо событие от визуального компонента (например, кнопки). Также он использовался в начальных версиях Java и сохранен для совместимости, так как взамен используется метод `actionPerformed()`. Подробно останавливаться на методе `action()` мы здесь не будем. Отметим, что параметр `evt` класса `Event` имеет ряд свойств, причем свойство `target` определяет объект, инициировавший *событие*. Другими важными свойствами параметра `evt` являются:

- `key` — код клавиши для события от клавиатуры;
- `when` — время происхождения события;
- `x,y` — координаты курсора мыши при возникновении события;
- `id` — идентификатор типа события (целое число).

Второй аргумент метода `action` позволяет конкретизировать объект, выдавший событие. Это делается, например, выполнением команды `if (arg.equals("SEND")) {...}`. Метод `equals()` используется для проверки названия кнопки. Запомните, что в конце метода `action()` следует указать команду `return false`, если предполагается, что окончательная обработка событий не завершена в методе `action()`, или `return true`, если никакая дальнейшая обработка событий не предполагается. Иначе говоря, метод `action()` может обрабатывать только часть событий, остальные должны переадресовываться операционной системе.

Теперь остается последняя нерассмотренная команда обработчика `action()`:

```
getAppletContext().showDocument(myurl,"_blank");
```

Эта команда отображает в окне документ с *сетевым адресом* `myurl` (обратите внимание, как он формируется в программе).

Второй параметр `blank` указывает, что новый документ должен открываться в новом окне. Другие варианты:

- `parent` — документ открывается в родительском окне;
- `top` — документ открывается в верхней части окна браузера.

Итак, мы полностью рассмотрели вызов апплета из другого апплета с помощью команды `showDocument()`.

Задание

1. Передать в другой апплет объект типа `Image`.
2. Передать в другой апплет и отобразить время часы:минуты:секунды.

   Для получения времени подключите пакет `java.util/*;` и создайте объект типа `Date`:

   ```
   Date now= new Date();
   ```

   Для получения часов использовать `now.getHours()`; минут — `now.getMinutes()`; секунд — `now.getSeconds()`.
3. Передать в другой апплет сетевой URL третьего апплета и показать его с помощью `showDocument()`.
4. Передать в другой апплет код клавиши (стрелка вверх, стрелка вниз, стрелка вправо, стрелка влево) и, в соответствии с переданным кодом клавиши, нарисовать линию методом `drawLine()`.
5. Управлять размерами отображаемой окружности во втором апплете из первого апплета с помощью клавиш-стрелок.

Контрольные вопросы

1. Как записывается апплет в HTML-файле?
2. Как выполнить апплет?
3. Как получить параметр из HTML-файла?
4. Укажите назначение команды `getApplet()`.
5. Объясните способ вызова метода другого апплета в данном. Найдите соответствующее место в программе.
6. Какие параметры использует команда `showDocument()`?

7. Объясните смысл оператора `instanceof`.
8. Сравните методы `action()` и `actionPerformed()`.

Внутренняя база данных апплета

Цель занятия

Создать интерфейс для работы с базой данных апплета. Рассмотреть использование класса `Hashtable` и его методов `put` и `get`, создание записей с различными типами полей, а также научиться сохранять внутреннюю базу данных в файле на диске. Дополнительные сведения можно найти в [2, 17].

Краткие теоретические сведения

Создадим *апплет*, который позволит пользователю работать с внутренней базой данных. Работа с базой данных предполагает построение интерфейса для добавления, удаления и поиска записей. Для реализации *базы данных* апплета используем класс `Hashtable`. Этот класс позволяет реализовать ассоциативное обращение к данным (для доступа к записям базы данных используются не номера записей, а содержимое полей базы данных). Таблицы Hashtable позволяют хранить данные различных типов. Доступ к данным выполняется безотносительно к их типу, поэтому нам потребуется выполнить приведение типов. То, что мы намереваемся сделать, показано на рис. 2.6 в виде окна апплета.

В апплете расположены кнопки **Clear**, **Add**, **Delete**, **Search**. Кнопка **Clear** используется для очистки полей текущей записи. Кнопка **Add** используется для добавления записи в базу. Кнопка **Delete** используется для удаления записи из базы. Для хранения таблицы базы данных используется класс `Hashtable` с переменной `bd` этого класса. Класс `Hashtable` является стандартным и импортируется с помощью инструкции:

```
import java.util.*;
```

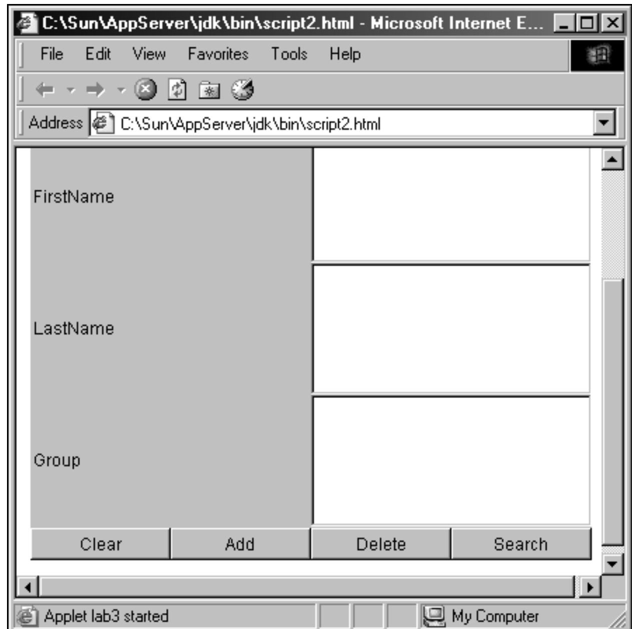


Рис. 2.6. Таблица базы данных апплета

У этого класса имеются удобные методы для добавления, удаления и поиска записей. Познакомимся с ними. Итак, `bd` — это объект класса `Hashtable`. Тогда добавление записи в `bd` реализует команда `bd.put(id,st)`. Здесь `id` — значение ключа добавляемой записи. Сама запись есть `st`. Заметим, что запись `st` состоит из полей. В примере это: имя, фамилия и группа (по-английски: `FirstName`, `SecondName`, `Group`). Чтобы сформировать такую запись, объявляем класс:

```
class Student
{
 protected String id;
 protected String fname;
 protected String lname;
 protected String group;
 Student(String id, String fname, String lname, String group)
```

```
 {
  this.id=id;
  this.fname=fname;
  this.lname=lname;
  this.group=group;
 }
} /*end of class*/
```

Класс `Student` содержит строковые переменные:

```
protected String id;
protected String fname;
protected String lname;
protected String group,
```

а также единственный метод — конструктор этого класса. Вспомните, что имя конструктора должно совпадать с именем класса, если конструктор есть. Таким образом, добавляемая в базу запись — это экземпляр (а точнее, объект) класса `Student`. Вот фрагмент кода, в котором осуществляется добавление записи при нажатии на кнопку **Add**:

```
String id=idfld.getText();
if(id!=null)
 {
  Student st=new Student(id,fnamefld.getText(),
                             lnamefld.getText(),
                            groupfld.getText());
  bd.put(id,st);
 }
```

Команда добавления записи — это `bd.put(id,st)`. Любопытно заметить, что объекту `bd` не важно, какой именной объект добавляется командой `put()`. Поэтому `Hashnable` может хранить вперемешку все — записи, числа, строки, даты, картинки. По-английски слово "hash" означает "мешанина". Фрагмент кода для удаления записи таков:

```
if (source==deletebtn)
```

```
{
  String id=idfld.getText();
  if(bd.remove(id)!=null)
   {
    clearFields();
    showStatus("Deleted !!!");
   }
```

Метод `clearFields()` очищает текстовые поля после удаления записи стандартным методом `bd.remove(id)` класса `Hashtable`, который, при успешном выполнении, возвращает значение логической истины (`true`).

Приведем теперь полный текст программы (листинг 2.8). Заметим лишь, что в методе `init()` выполняется стандартный набор действий по добавлению и размещению на панели кнопок, текстовых полей, что нам уже знакомо по предыдущим занятиям.

Просмотрите текст апплета и найдите в нем уже описанные фрагменты. Постарайтесь разобраться в том, какие действия выполняются в каждом из блоков программы.

Листинг 2.8. Текст приложения для работы с внутренней базой данных апплета

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.applet.*;

class Student
{
 protected String id;
 protected String fname;
 protected String lname;
 protected String group;

 Student(String id, String fname, String lname,
          String group)
```

```
 {
  this.id=id;
  this.fname=fname;
  this.lname=lname;
  this.group=group;
 }
} /*end of class*/

public class lab3 extends Applet implements ActionListener
{
 Panel dat;
 Panel but;
 Hashtable bd;

 TextField idfld;
 TextField fnamefld;
 TextField lnamefld;
 TextField groupfld;

 Button clearbtn;
 Button addbtn;
 Button deletebtn;
 Button searchbtn;

 public void init()
 {
  bd=new Hashtable();
  setLayout(new BorderLayout());
  dat=new Panel();
  dat.setLayout(new GridLayout(4,2));
  dat.add(new Label("ID"));
  dat.add(idfld=new TextField());
  dat.add(new Label("FirstName"));
  dat.add(fnamefld=new TextField());
```

```
 dat.add(new Label("LastName"));
 dat.add(lnamefld=new TextField());
 dat.add(new Label("Group"));
 dat.add(groupfld=new TextField());
 but=new Panel();
 but.setLayout(new GridLayout(1,4));
 but.add(clearbtn=new Button("Clear"));
 but.add(addbtn=new Button("Add"));
 but.add(deletebtn=new Button("Delete"));
 but.add(searchbtn=new Button("Search"));
 add("Center",dat);
 add("South",but);
 clearbtn.addActionListener(this);
 addbtn.addActionListener(this);
 deletebtn.addActionListener(this);
 searchbtn.addActionListener(this);
}
public void clearFields()
{
 idfld.setText("");
 fnamefld.setText("");
 lnamefld.setText("");
 groupfld.setText("");
}
public void actionPerformed(ActionEvent ae)
{
 Object source=ae.getSource();
 if (source==clearbtn)
   {
    clearFields();
   }
 else if(source==addbtn)
   {
    String id=idfld.getText();
```

```
    if(id!=null)
     {
      Student st=new Student(id,fnamefld.getText(),
                                lnamefld.getText(),
                               groupfld.getText());
      bd.put(id,st);
     }
    else
     {
      showStatus("Need an ID");
     }
   }
  else
   if (source==deletebtn)
    {
     String id=idfld.getText();
     if(bd.remove(id)!=null)
      {
       clearFields();
       showStatus("Deleted !!!");
      }
    }
   else
    if(source==searchbtn)
     {
      String id=idfld.getText();
      Object ob=bd.get(id);
      if (ob!=null)
       {
        fnamefld.setText(((Student)ob).fname);
        lnamefld.setText(((Student)ob).lname);
        groupfld.setText(((Student)ob).group);
        idfld.setText(id);
       }
```

```
      else
        {
         showStatus("ID not found");
        }
      }
  }
}
```

Для выполнения этого апплета создадим HTML-файл следующего вида:

```
<html>
<body>
<APPLET   code="lab3.class"
          width=300
          height=300>
</APPLET>
</body>
</html>
```

Параметр `code` в этом документе определяет имя класса, в который скомпилирован исходный файл с апплетом, приведенный в листинге 2.8.

Задание

Разберитесь с программой. Вам надлежит написать программу по образу и подобию представленной, используя следующие варианты.

1. База данных должна содержать картинки (адреса картинок, которые отображаются при выборе записи из базы данных). Обращаем внимание на то, что картинку нельзя вывести с помощью метода `drawImage()` на базе переменной графического контекста `Graphics` на панели. Метод `drawImage()` рисует картинку на апплете или на форме. Поэтому вам следует разместить панели `but` и `dat` на апплете так, чтобы освободить место для изображения рисунка.
2. База данных должна содержать небольшие тексты.
3. База данных должна содержать сетевые `URL`, а также должна быть реализована возможность их просмотра.

Контрольные вопросы

1. Каким образом осуществляются чтение и запись в таблицы Hashtable?
2. В чем особенности ассоциативных баз данных?
3. Приведите пример объявления записей, содержащих поля разных типов.
4. Продумайте, как сохранить содержимое Hashtable в файле на диске.
5. Продумайте, как прочитать записи, сохраненные в предыдущем пункте, обратно в Hashtable.
6. Продемонстрируйте, как визуализировать таблицу с помощью компоновщика `GridLayout`.

Работа с формами и меню

Цель занятия

Изучить основные принципы работы с формами, научиться использовать горизонтальные и всплывающие меню. Рассмотреть способы обработки событий от элементов меню. Научиться создавать и вызывать вспомогательные формы. Следует вспомнить материал *главы 1*, посвященный обработке событий от мыши и клавиатуры, а также разработке визуального интерфейса *(см. главы 1, 2)*. Дополнительные сведения следует искать в [2, 11, 13, 15, 17].

Краткие теоретические сведения

Рассмотрим сначала, как создавать горизонтальное *меню*. Панель меню объявляется в приложении с помощью класса `MenuBar`. Элементы горизонтального меню объявляются как объекты класса `Menu` и добавляются на панель `MenuBar`. Пункты меню являются членами класса `MenuItem` и добавляются в соответствующие элементы меню `Menu`. Обработка событий, связанных с выбором пунктов меню, выполняется на основе

прослушивателя `ActionListener` так же, как и от кнопок. Для этих же целей можно задействовать метод `action()`.

Прежде всего, создадим меню в конструкторе класса так, как показано далее:

```
import java.awt.*;
import java.util.*;
import java.lang.*;

public class Forma extends Frame
{
 MenuBar mb = new MenuBar();
 Menu m1=new Menu("Operations");
 Menu m2=new Menu("TheEnd");
 Forma()
 {
  m1.add(new MenuItem("Show"));
  m2.add(new MenuItem("Quit"));
  mb.add(m1);
  mb.add(m2);
  setMenuBar(mb);
  setBackground(new Color(200,20,100));
 }
```

Как видно, меню определяется через компоненты `MenuBar`, `Menu`, `MenuItem`. Последние генерируют событие при их выборе и допускают обработку. *Обработчик событий* от пунктов меню с именами **Quit** и **Show**:

```
public boolean action (Event evt, Object ob)
{
 String lbl=(String) ob;
 if (lbl.equals("Quit"))
  {
   System.exit(0);
   return true;
  }
```

```
 else
  {
   if (lbl.equals("Show"))
    {
     Framemenu fr= new Framemenu();
     fr.resize(300,300);
     fr.show();
     return true;
    }
  }
 return false;
}
```

Метод `public boolean action(Event evt, Object ob)` является универсальным (позволяет обрабатывать события и от программных кнопок, и от других компонентов). Его следует запомнить. Аргументами этого метода являются имя события и объект, сгенерировавший событие. В тексте метода проверяется, от какого объекта возникло событие.

Следующие строки кода показывают, как конструируются (создаются) объекты в Java:

```
Framemenu fr= new Framemenu();
fr.resize(300,300);
fr.show();
```

`Framemenu` — это имя класса, который имеет такое описание:

```
class Framemenu extends Frame
{
 Button ex=new Button("Return");
 Framemenu(){
 add(ex);
 setLayout(null);
 ex.setBounds(20,40,100,20);
 setBackground(new Color(100,60,120));
}
```

В классе `Framemenu` определены два метода.

```
public void paint(Graphics g)
{
 g.drawString("Hello, Dears!",120,100);
}
public boolean action (Event evt, Object ob)
{
 String lb=(String) ob;
 if (lb.equals("Return"))
  dispose();
 return true;
}
```

Метод `paint()` используется для вывода текстовой строки, метод `action()` — обработчик событий; в данном случае при нажатии на кнопку **Return** вызывается метод `dispose()`, который удаляет вспомогательную форму.

Таким образом, на главной форме создано *меню*, которое позволяет запускать вспомогательную форму. Полный текст приложения (листинг 2.9) приведен далее.

Листинг 2.9. Текст приложения для работы с горизонтальным меню

```
import java.awt.*;
import java.util.*;
import java.lang.*;

public class Forma extends Frame
{
 MenuBar mb = new MenuBar();
 Menu m1=new Menu("Operations");
 Menu m2=new Menu("TheEnd");
```

```
Forma()
 {
  // Пункт меню для демонстрации вспомогательной формы:
  m1.add(new MenuItem("Show"));
  // Пункт меню для завершения приложения:
  m2.add(new MenuItem("Quit"));    mb.add(m1);
  mb.add(m2);
  setMenuBar(mb);
  setBackground(new Color(200,20,100));
 }

 public boolean action (Event evt, Object ob)
 {
  String lbl=(String) ob;
  if (lbl.equals("Quit"))
   {
    System.exit(0);
    return true;
   }
  else
   {
    if (lbl.equals("Show"))
     {
      Framemenu fr= new Framemenu(); // Создаем
                                     // вспомогательную форму
      fr.resize(300,300);
      fr.show(); // Делаем вспомогательную форму видимой
      return true;
     }
   }
  return false;
 }
```

```
 public static void main(String args[])
 {
  Forma f=new Forma();
  f.resize(400,400);
  f.show();
 }
}
class Framemenu extends Frame  // Класс вспомогательной формы
{
 Button ex=new Button("Return");
 Framemenu()
 {
  add(ex);
  setLayout(null);
  ex.setBounds(20,40,100,20);
  setBackground(new Color(100,60,120));
 }
 public void paint(Graphics g)
 {
  g.drawString("Hello, Dears!",120,100);
 }
 public boolean action (Event evt, Object ob)
 {
  String lb=(String) ob;
  if (lb.equals("Return"))
   dispose();
  return true;
 }
}
```

Теперь обратимся к всплывающему *меню*, которое относится к классу `PopupMenu` и использует пункты меню `MenuItems`. Для работы с всплывающим меню следует подключить два интерфейса:

ActionListener и MouseListener. Интерфейс MouseListener используется для его отображения, которое выполняется на основе метода pressMouseEvent() следующим образом:

```
public void pressMouseEvent(MouseEvent ev)
{
 if(ev.isPopupTrigger())
  menu.show(ev.getComponent(), ev.getX(), ev.getY());
 super.processMouseEvent(ev);
}
```

В этом фрагменте menu — программное имя всплывающего меню (переменной класса PopupMenu). Метод getComponent() класса MouseEvent возвращает объект, создавший событие ev; метод getX() возвращает координату по оси *X* курсора мыши в момент генерации события, а метод getY() — координату по оси *Y*. Последняя команда приведенного фрагмента выполняет вызов метода processMouseEvent() суперкласса.

Обработка событий, связанных с выбором пунктов всплывающего меню, реализуется традиционным образом через интерфейс ActionListener. Всплывающее меню добавлено на форму в тексте конструктора.

```
Forma()
{
 setLayout(null);
 add(exit); // Добавлена кнопка для выхода из приложения
 addMouseListener(this); // Добавлен прослушиватель
                         // MouseListener для формы
 add(menu); // Добавлено всплывающее меню
 menu.add(it1); // Присоединение пунктов it1 и it2
 menu.add(it2);
 it1.addActionListener(this); // Добавление прослушивателя
                              // для пунктов всплывающего меню
 it2.addActionListener(this);
 enableEvents(AWT.MOUSE_EVENT_MASK); // Активизация событий от
                                     // мыши
```

```
 exit.addActionListener(this); // Добавление прослушивателя
                               // для кнопки
 exit.setBounds(10,20,100,20); // Установка размеров и
                               // координат кнопки
}
```

Приведем код программы, в которой показан пример создания всплывающего меню (листинг 2.10).

Листинг 2.10. Текст приложения для работы со всплывающим меню

```
import java.awt.*;
import java.util.*;
import java.lang.*;
import java.awt.event.*;
public class Forma extends Frame implements ActionListener,
MouseListener
{
 Button exit=new Button("Exit");
 PopupMenu menu= new PopupMenu("pop"); // Создается
                                       // всплывающее меню
 MenuItem it1= new MenuItem("Show"); // Используется для
                                     // открытия
                                     // вспомогательной формы
 MenuItem it2= new MenuItem("Quit"); // Используется для
                                     // выхода из приложения
 Forma()  //Конструктор
 {
  setLayout(null);
  add(exit);
  addMouseListener(this); // Добавление прослушивателя для
                          // формы
  add(menu);
  menu.add(it1); // Добавление пунктов во всплывающее меню
  menu.add(it2); // Добавление пунктов во всплывающее меню
```

```
 it1.addActionListener(this); // Добавление прослушивателя
                              // для пунктов всплывающего
                              // меню
 it2.addActionListener(this);
 enableEvents(AWTEvent.MOUSE_EVENT_MASK); // Активизация
                                          // событий от мыши
 exit.addActionListener(this);
 exit.setBounds(10,20,100,20);
}
public void mouseEntered(MouseEvent ev) // Приведенные методы
                                        // интерфейса
                                        // MouseListener
                                        // не реализованы
{}
public void mouseExited(MouseEvent ev)
{}
public void mousePressed(MouseEvent ev)
{}
public void mouseReleased(MouseEvent ev) // Всплывающее меню
                                         // нужно создавать
                                         // в этом методе
{
 if(ev.isPopupTrigger())
  menu.show(ev.getComponent(), ev.getX(), ev.getY());
                                    // Отобразить
                                    // всплывающее меню
}
public void mouseClicked(MouseEvent ev)
{
}
public void actionPerformed(ActionEvent e)
{
 if(e.getSource()==exit)
  {
   System.exit(0);
```

```
   }
  else
   {
    if (e.getSource()==it1) // Выбран пункт it1 всплывающего
                            // меню
     {
      setBackground(Color.yellow);
      Framemenu fr= new Framemenu(); // Создаем
                                     // вспомогательную форму
      fr.resize(300,300);
      fr.show(); // Делаем вспомогательную форму видимой
     }
   }
 }

 public static void main(String args[])
 {
  Forma f=new Forma();
  f.resize(400,400);
  f.setBackground(new Color(100,60,120));
  f.show();
 }
}
class Framemenu extends Frame  // Класс вспомогательной формы
{
 Button ex=new Button("Return");
 Framemenu()
 {
  add(ex);
  setLayout(null);
  ex.setBounds(20,40,100,20);
 }
```

```
 public void paint(Graphics g)
 {
  g.drawString("Hello, Dears!",120,100);
 }
 public boolean action (Event evt, Object ob)
 {
  String lb=(String) ob;
  if (lb.equals("Return"))
   dispose(); // Закрываем и удаляем вспомогательную форму
  return true;
 }
}
```

Обратим внимание на следующее:

- всплывающее меню относится к классу `PopupMenu`
- его пункты обрабатываются с помощью метода `actionPerformed()`
- само меню нужно показать командой:

  `menu.show(ev.getComponent(), ev.getX(), ev.getY())`
- эту команду следует поместить в обработчик `mouseReleased()`.

Задание

Объединить данную работу с работой практического занятия *"Введение в Java"*, в которой строился график функции, так, чтобы теперь график строился во вспомогательном окне.

Контрольные вопросы

1. Как описывать горизонтальное меню в приложении?
2. Как описывается всплывающее меню в приложении?
3. Укажите, какие интерфейсы следует подключить для всплывающего меню, и для чего они нужны?
4. Каким образом активизировать события от мыши?

5. Каким методом отображается всплывающее меню?
6. Как подключить прослушиватель событий от мыши к форме?
7. Объясните назначение слова `this` в следующем коде:

```
public void mouseClicked(MouseEvent ev)
{
 if(ev.getSource()==this)
  this.setrBackground(Color.yellow);
}
```

8. Объясните и продемонстрируйте создание вспомогательных форм.
9. Объясните, какой смысл заключается в использовании ключевого слова `this`?

Java и базы данных

Цель занятия

Целью настоящего занятия является изучение механизма взаимодействия программы на языке Java и базы данных. Надлежит усвоить то, каким образом устанавливается соединение с базой данных с помощью подсистемы *Администратор BDE*, как подключается интерфейс JDBC-ODBC, как исполняются SQL-запросы. Сведения по языку SQL достаточно подробно изложены в [8, 10, 11]. Дополнительные сведения о работе с базой данных можно почерпнуть из [2, 10].

Краткие теоретические сведения

Работу с *базами данных* выполняет подсистема JDBC (Java DataBase Connectivity). Имеется два основных варианта работы с базами данных. Первый вариант реализуется через взаимодействие с системой ODBC (Open DataBase Connectivity), ставшей в некотором смысле стандартом и реализованной в Windows. Система ODBC реализует множество различных драйверов для работы с широким диапазоном баз данных. Для работы с ODBC Java включает своего рода интерфейс JDBC-ODBC. Именно с этим вариантом работы мы познакомимся в этом практическом занятии. Второй вариант работы с базами данных состоит в подключении к ним "напря-

мую". Этот вариант работы мы не рассматриваем. Он реализован на основе ANSI SQLJ (American National Standards Institute SQL for Java — SQL для Java Американского национального института стандартов), а описание можно найти, например, в [10].

Для работы с базой данных в Java необходимо выполнить следующие действия. Подключить системный драйвер для работы с конкретной базой. Для этого выберите иконку **My Computer** (Мой компьютер) на рабочем столе, далее — пункт **Control Panel** (Панель управления). Затем выберите пункт **Administrative Tools** (Средства администрирования) и **Data Sources ODBC** (Источники данных ODBC). Теперь выберите вкладку **System DSN** (Системный DSN; DSN — Data Source Name), если создаете источник данных для многих пользователей и регистрируете его в реестре, либо **User DSN** (Пользовательский DSN), если создаете источник данных, доступный только на вашем компьютере. Появится окно администратора баз данных следующего вида (рис. 2.7).

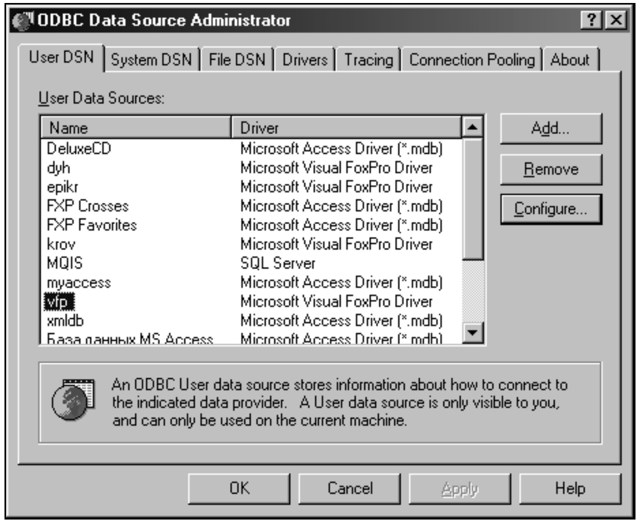


Рис. 2.7. Окно администратора баз данных

В этом окне уже добавлены системные ODBC-драйверы для базы данных FoxPro и Access. Поскольку предполагается, что изначально этих установок нет, то выберите кнопку **Add** (Добавить).

Затем двойным щелчком мыши выберите нужный драйвер из списка драйверов. Получите следующее окно (рис. 2.8).

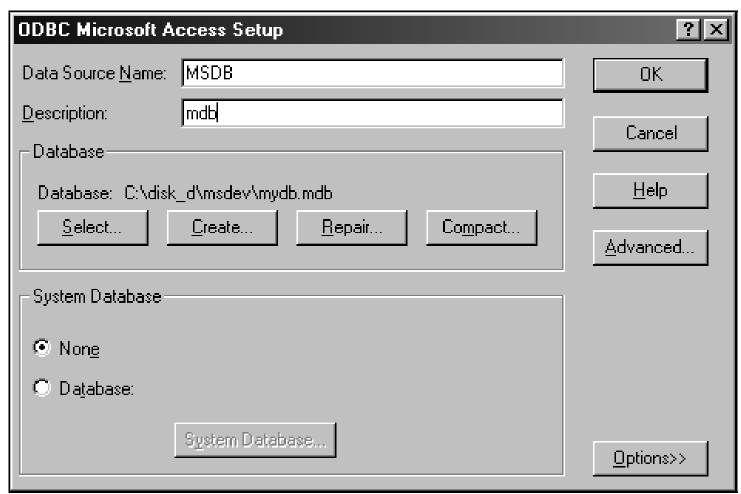


Рис. 2.8. Создание источника данных

В поле **Data Source Name** (Имя источника) введите, например, `MSDB`; в поле **Description** (Описание) — `mdb` для Access (для FoxPro введите, соответственно, `VFP` или `DBF` — *см. Замечание далее*). Если собираетесь работать с конкретной базой данных, то она должна уже быть создана, и в этом случае нужно будет ввести название пути к ней. Закройте окна для установки драйверов. Данный пункт завершен.

Перед началом работы с базой данных она должна быть закрыта, поэтому вам потребуется установить соединение с ней. Для этого в Java используется команда:

```
Connection db=DriverManager.getConnection(url);
```

Здесь `url` задает ссылку на драйвер базы данных. Переменная `url` может быть объявлена так:

`String url="jdbc:odbc:vfp";` — для FOXPRO

или

`String url="jdbc:odbc:msdb";` — для ACCESS.

Замечание

Вы вовсе не обязаны использовать имена `vfp` и `msdb` для именования своих источников данных. Вопрос выбора имен — сугубо индивидуальное дело. В своем приложении вы можете назвать их иначе.

Следует помнить, что база данных — это не отдельная таблица, а хранилище таблиц. Поэтому если вы хотите получить работоспособную программу, позаботьтесь о предварительном создании пустой базы, в которую вы сможете успешно добавить свою новую таблицу, и не одну.

Для работы с базой данных потребуется вспомнить SQL для того, чтобы писать запросы, в том числе и запрос на создание таблицы. Это делается так:

```
Statement sq=db.createStatement(); // формируем пустую команду
                                   // SQL
String  sq_str="SELECT * FROM myt"; // Определяем строку
                                    // запроса на выборку
ResultSet rs= sq.executeQuery(sq_str); // Выполняем команду
                                       // SQL
```

И это почти все. В приведенном примере сформирована строка запроса:

```
String  sq_str="SELECT * FROM myt";
```

и выдана команда на его выполнение:

```
ResultSet rs=sq.executeQuery(sq_str);
```

Ясно, что переменная `rs` будет содержать все найденные записи. Для доступа к этим записям можно организовать следующий цикл:

```
while(rs.next())
{
    String s=rs.getString("Name");
    System.out.println("my-"+s);}
    System.out.println("OK");
}
```

Команда `rs.next()` выдает очередную запись и возвращает `true` или `false` в зависимости от того, достигнут конец набора или нет. Команда `rs.getString("Name")` возвращает содержимое поля `Name`, которое должно иметь тип `char` (`varchar`). Для получения значений полей другого типа следует использовать следующие команды:

- `rs.getBoolean(имя_столбца);` `//` — для поля таблицы типа `boolean`
- `rs.getInt(имя_столбца);` `//` — для поля таблицы типа `int`
- `rs.getDate(имя_столбца);` `//` — для поля таблицы типа `Date`
- `rs.getFloat(имя_столбца);` `//` — для поля таблицы типа `Numeric`

Далее (листинг 2.11) приведен полный текст программы, из которого все сказанное выше следует как само собой разумеющееся (хотя некоторые аспекты все еще нуждаются в комментариях).

Листинг 2.11. Текст приложения для работы с базой данных

```
import java.awt.*;
import java.net.*;
import java.sql.*;
public class lab6
{
 public static void main(String args[])
 {
   String url="jdbc:odbc:vfp"; // Подключение источника данных
                               // vfp, подготовленного с помощью
                               // администратора ODBC
   try
    {
     Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    } //нуждается в комментарии
   catch(Exception e)
    {
     System.out.println("Classdefnotfound"+e);
```

```
    }
   try
    {
     // Объявление соединения с источником данных:
     Connection db=DriverManager.getConnection(url);
     Statement sq=db.createStatement();// Создаем пустую
                                       // команду SQL
     // Строка запроса на выборку данных:
     String  sq_str="SELECT * FROM myt";
     ResultSet rs= sq.executeQuery(sq_str); // Переменная rs
                                            // получает набор
                                           // выбранных записей
     while(rs.next())
     { //Выполняем просмотр записей набора:
       String s=rs.getString("Name"); // Получаем содержимое
                                      // поля Name текущей
                                      // записи
       System.out.println("my-"+s); // Выводим содержимое поля
                                    // Name на консоль
     }
     System.out.println("OK");
     // db.close(); // Эта строка не является необходимой
     // Но все же рекомендуется закрывать базу после
     // завершения работы
    }
   catch(Exception er)
    {
     System.out.println("Error has arised here:"+er);}
    }
 }
```

Дадим дополнительные комментарии к программе. Строка:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

должна указываться обязательно; она подключает интерфейс для перехода Java-ODBC, без которого Java не сможет связаться с установленным вами ODBC-драйвером. Таким образом, основная

часть работы связывается с формированием *SQL-запросов*. Запрос имеет следующее схематическое представление:

```
<Команда> <Таблица> <Данные>
```

Основными командами являются следующие:

- создание таблицы (`CREATE table`);
- удаление таблицы (`DROP table`);
- вставка записи в таблицу (`INSERT into`);
- обновление таблицы путем изменения данных (`UPDATE`);
- выборка записей по условию (`SELECT`).

Условия задаются с помощью ключевого слова `where`, после которого записывается простое или сложное логическое условие, например:

`where age>20` (выбор записей, где поле `age` больше 20), или

`where (age>20) and (age<40)` (выбор записей, где поле `age` больше 20 и меньше 40), или

`where not(name="Ivanov")` (выбор записей, где поле `name` не содержит слово `"Ivanov"`) и т. д.

Пример создания таблицы:

```
CREATE table MYTABLE (Name char(50), Age int(3), Location
varchar(20))
```

В этом примере создается таблица по имени `MYTABLE`, состоящая из полей `Name`, `Age` и `Location`. Для каждого поля при создании указывается тип. Для поля `Name` указан строковый тип фиксированной длины `(char(50))`; для поля `Age` — целочисленный тип с тремя значащими цифрами; для поля `Location` — строковый тип переменной длины (первоначальная длина составляет 20 символов). Наряду с указанными выделим другие типы данных SQL:

- `Date` (дата);
- `Numeric(m,n)` (вещественный тип с фиксированной точкой; `m` определяет общее число разрядов, а `n` — число разрядов после запятой);
- `Float` (вещественный с плавающей точкой);

- `DateTime` (дата и время);
- `Blob` (большие двоичные наборы данных).

Пример удаления таблицы:

```
DROP table MYTABLE
```

Вставка записи в таблицу:

```
INSERT INTO MYTABLE VALUES("Jan",140, "Paradise")
```

Данные изменяются в таблице запросом:

```
UPDATE MYTABLE SET LOCATION="Paradise" WHERE AGE>110
```

Запись удаляется из таблицы запросом:

```
DELETE FROM MYTABLE WHERE AGE>110
```

Осуществим выборку записей, у которых значение поля `LOCATION` принадлежит множеству `{"Минск", "Куйбышев", "Гродно"}`:

```
SELECT * FROM MYTABLE WHERE
LOCATION IN("Минск","Куйбышев","Гродно")
```

Для формирования упорядоченного результирующего набора используется ключевое слово `ORDER BY`, за которым указываем имя столбца, по которому выполняется упорядочение записей:

```
SELECT * FROM MYTABLE WHERE
LOCATION IN("Минск","Куйбышев","Гродно") ORDER BY Location
```

Подробности о языке SQL можно найти в [8].

Интерес представляет чтение полей, в которых хранятся картинки. Такие поля описываются с типом `Blob`. Например, при создании таблицы средствами языка SQL, одно из полей которой должно содержать рисунок, следует ввести команду следующего типа:

```
CREATE table MYTABLE (Name char(50), Age int(3), Photo
blob(64k))
```

Для выборки объектов `Blob` следует использовать команду `getBlob()`. Приведем короткий фрагмент, который прояснит суть вопроса:

```
Label lb=new Label();
  ...
```

```
rs= s.executeQuery("Select * from MYTABLE"");
if (rs.next())
  {
   Blob bl =rs.getBlob(3);
   // 3 — номер поля в заголовке таблицы с рисунком
   byte [ ] imgbytes = blob.getBytes(1,60*60);
   // картинка имеет размер 60*60 пикселов
   lb.setIcon(new ImageIcon(imgbytes));
   ...
  }
```

Таким образом, предварительно созданную базу данных с картинками можно просто просматривать, как показано выше.

В приведенном приложении (листинг 2.9) использовалась команда `executeQuery()`, которая позволяла построить набор записей по запросу `SELECT`. Для выполнения других запросов SQL (например, `CREATE` или `UPDATE`) следует использовать команду `execute()`. Следующий фрагмент кода поясняет ситуацию.

```
Statement sq=db.createStatement(); // Создаем пустую команду
                                   // SQL
String sq_str="Create table myt(Name char(20), Age int(3),
Location char(30))
sq.execute(sq_str); // Выполнение запроса на создание таблицы
```

Разумеется, никакого результирующего набора при выполнении таких запросов не создается.

Задание

Прочитайте теоретическую часть и убедитесь, что все понятно. Напишите программу, базируясь на представленном в теоретической части аналоге для работы с базой данных. Ваша программа должна выполнять следующие операции:

- создавать таблицу и удалять ее;
- просматривать таблицу;
- выполнять поиск по вводимому значению для поля;
- добавлять записи.

Контрольные вопросы

1. Что такое база данных и источник данных ODBC?
2. Как создать источник данных ODBC?
3. Каким образом Java получает доступ к источнику данных ODBC?
4. Назовите известные вам команды SQL.
5. Покажите, как осуществляется выборка записей (добавление записей).
6. Объясните, как получить доступ к полям просматриваемой записи.
7. Укажите, как записывать и считывать из базы данных рисунки.
8. Назовите известные вам типы данных SQL.
9. Является ли вариант связи ODBC-JDBC единственным механизмом работы с базами данных в Java?

Основы XML. Преобразование XML-HTML. Использование JavaScript

Цель занятия

Знакомство с технологией XML. Изучение возможности представления документов XML в HTML. Использование скриптов JavaScript для навигации по таблице XML и организации поиска данных по условию. Рекомендуемая литература [4, 13, 16].

Краткие теоретические сведения

Технология XML (eXtensible Markup Language) была создана в конце 90-х годов прошлого столетия. Основные достоинства текста XML:

- имеет структуру базы данных, доступен ЭВМ и человеку;
- удобно обрабатывается средствами современных языков программирования;
- легко переводится в HTML.

Рассмотрим следующий пример текстовой базы данных, написанной на XML:

```
<?xml version="1.0"?>
<root>
     <book>
       <title> Three men in the boat</title>
        <author> Jerom-K-Jerom</author>
        <price> 12000</price>
     </book>
     <book>
       <title> Notre Domme de Paris</title>
        <author> V.Hugo</author>
        <price> 15000 </price>
     </book>
     <book>
       <title> A War and Peace</title>
        <author> L.Tolstoy</author>
        <price> 16500</price>
     </book>
     <book>
       <title> Angelika — the misstress of ghosts</title>
        <author> A and S. Gallen</author>
        <price> 9000</price>
     </book>
</root>
```

Это пример правильно составленного документа XML, элементами которого являются теги `<root>`, `<notes>`, `<book>`, `<title>`, `<author>`, `<price>`.

Элементы в тексте расположены по типу дерева с головным элементом `<root>`. Каждый элемент имеет сопряженный с ним закрывающий элемент. Область действия каждого элемента ограничена открывающим и закрывающим элементами. Не допускается пересечения области действия элементов, т. е. области либо вложены одна в другую, либо вовсе не пересекаются. Элемент `<root>`, область действия которого содержит области действия всех других

элементов, называется корневым. XML-документ можно рассматривать как текстовую базу данных. Значением элемента является информация, помещенная между тегами, определяющими данный элемент. Так, значением первого элемента `<title>` является строка `Three men in the boat.`

Наберите этот текст в любом редакторе и сохраните его как простой текстовый файл с расширением xml — например, дайте этому файлу имя textbd.xml. Можно просмотреть этот файл браузером Internet Explorer так же, как вы просматривали HTML-файлы. В случае ошибки интерпретатор XML выдаст подробную информацию о дислокации и сути ошибки.

Теперь покажем, как перевести этот вывод в табличную форму HTML, что выполняется средствами HTML. Создадим следующий файл HTML (листинг 2.12).

Листинг 2.12. HTML-документ для отображения таблицы XML

```
<HTML>
  <BODY bgcolor=#AA00EE>
     <h1> Our first lesson in xml-technology</h1>
     <XML src="textbd.xml" ID="myxml"></XML>
     <Table id="tb" border="2" width="80%" datasrc="#myxml">
    <Thead style="background-color:aqua">
  <TH> The Book Title</TH>
  <TH> The author </TH>
   <TH> The price </TH>
     </Thead>
  <TR>
    <TD><SPAN DATAFLD="title"/></TD>
    <TD><SPAN DATAFLD="author"/></TD>
    <TD><SPAN DATAFLD="price"/></TD>
  </TR>
</TABLE>
</BODY>
</HTML>
```

Сохраним этот HTML-файл под именем textbd.html. Теперь откроем его браузером. Результат будет таким (рис. 2.9).

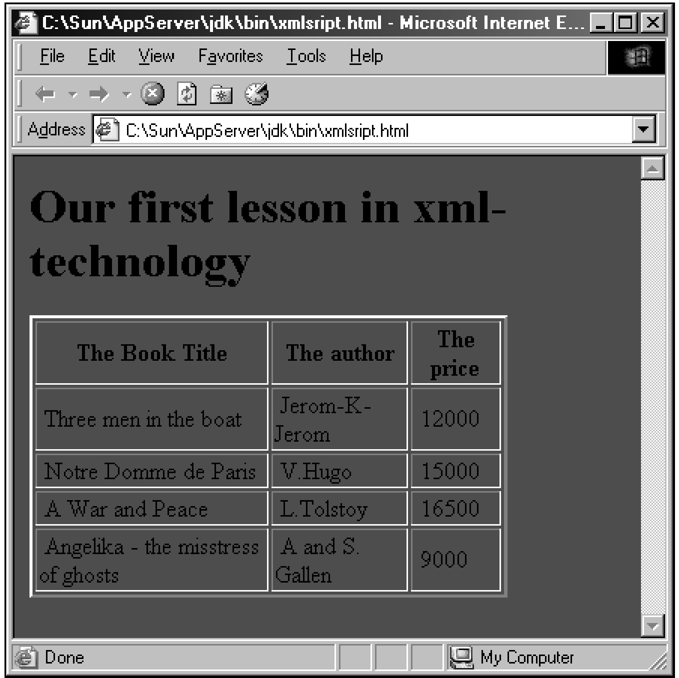


Рис. 2.9. Отображение документа XML в документе HTML

Для подключения созданного ранее XML-файла и связывания его с таблицей используются теги:

```
<XML src="textbd.xml" ID="myxml"></XML>
<Table id="tb" border="2" width="80%" datasrc="#myxml">
```

Для отображения данных в таблице используются теги для ячеек в следующем виде:

```
<TD><SPAN DATAFLD="title"/></TD>
<TD><SPAN DATAFLD="author"/></TD>
<TD><SPAN DATAFLD="price"/></TD>
```

Тег `<SPAN>` используется в качестве контейнера. Параметр `DATAFLD` содержит значение отображаемого элемента XML.

При работе с базами данных одним из основных вопросов является поиск требуемой информации. В этой работе осуществим такой поиск с помощью средств JavaScript. Поскольку база может быть достаточно большой, то вывод ее целиком в таблице HTML-документа очень неэффективен. Поэтому будем отображать не всю таблицу, а, скажем, только две записи. Кроме того, поставим кнопки, чтобы листать базу <вверх-вниз>. Для этого изменим наш HTML-документ следующим образом (листинг 2.13).

Листинг 2.13. Модифицированный HTML-документ для отображения таблицы XML

```
<html>
  <body bgcolor=#AA00EE>
     <h1> Our first lesson in xml-technology</h1>
     <XML src="textbd.xml" ID="myxml"></XML>
     <Table id="tb" border="2" width="80%" datasrc="#myxml"
     datapagesize="2">
    <Thead style="background-color: acqua">
  <TH> The Book Title</TH>
  <TH> The author </TH>
   <TH> The price </TH>
     </Thead>
  <TR>
    <TD><SPAN DATAFLD="title"/></TD>
    <TD><SPAN DATAFLD="author"/></TD>
    <TD><SPAN DATAFLD="price"/></TD>
   </TR>
   </TABLE>
   <Button onClick="tb.nextPage()">&gt</Button> <!--Команда
   отображает следующую страницу-->
   <Button onClick="tb.previousPage()">&lt</Button><!--Команда
   отображает предыдущую страницу загруженного в память XML-
   документа-->
   <br/>
   <br/>
   </BODY>
    </HTML>
```

Терм `>` используется для прорисовки стрелки вправо, терм `<` — стрелки влево. При этом указываем, что нужно отображать только две записи в таблице:

```
<Table id="tb" border="2" width="80%" datasrc="#myxml"
datapagesize="2">
```

Теперь создадим для нашего сайта функциональное наполнение. Его смысл будет заключаться в том, что будем вводить название книги целиком или какие-то его фрагменты, и по нажатию кнопки система должна выдавать другие реквизиты книги: автора и цену либо сообщать, что книга не найдена. Теперь понадобится привлечь JavaScript. Собственно потребуется всего несколько команд.

- `getElementByTagName("title").item(i).text;`

  Эта команда возвращает значение элемента `<title>` из XML-файла, который является i-м по порядку перечисления этих элементов `<title>`.

- `getElementsByTagName("title").length;`

  Эта команда возвращает общее число элементов `<title>` из XML-документа.

- `String.indexOf(string1);`

  Эта команда возвращает позицию, с которой строка `string1` входит в строку `String` либо `-1`, если вхождений нет.

Теперь приведем расширенный HTML-код для этой задачи (листинг 2.14).

Листинг 2.14. Расширенный HTML-документ для отображения таблицы XML

```
<html>
<head>
<script>
<!--
function showelement()
{
 // Подключение XML-документа:
 var odoc=new ActiveXObject("Microsoft.XMLDOM");
```

```
odoc.async=false; // Приостановка программы,
                  // пока загрузка не завершится
odoc.load("textbd.xml"); // Загрузка XML-документа в память
var string1=document.myform.mytext.value;
z=odoc.getElementsByTagName("title").length;// Получение
                                            // длины элемента
                                            // с тегом
                                            // <title>
if(z>0)
 {
  for(i=0;i<z;i++) // Цикл для проверки на совпадение
  // указанного названия и названий, содержащихся
  // в теге <title> считанного XML-документа
  {
   var s=new
   String(odoc.getElementsByTagName("title").item(i).text);
   if(s.indexOf(string1) == -1)
    {
     continue;
    }
   else // Совпадение строк установлено.
        // Отображаем содержимое тега <author>
    {
var s2=new
String(odoc.getElementsByTagName("author").item(i).text);
     document.myform.mytext.value="Author:"+s2;
     break;
    }
  }
 }
else
 {
  alert ("No entry found for"+string1);
 }
}
-->
```

```
</script>
</head>
<body bgcolor=#AABBEE>
<h1> Our first lesson in xml-technology</h1>
<XML src="textbd.xml" ID="myxml"></XML>
<Table id="tb" border="2" width="80%" datasrc="#myxml"
datapagesize="2">
<Thead style="background-color:aqua">
<TH> The Book Title</TH>
<TH> The author </TH>
<TH> The price </TH>
</Thead>
<TR>
<TD><SPAN DATAFLD="title"/></TD>
<TD><SPAN DATAFLD="author"/></TD>
<TD><SPAN DATAFLD="price"/></TD>
</TR>
</TABLE>
<br/>
<br/>
<Button onClick="tb.nextPage()">&gt</Button>
<Button onClick="tb.previousPage()">&lt</Button>
<br/>
<br/>
<form name="myform">
  <br>
    <input type="Text" name="mytext"   value=""/>
   <br>
<center>
 <Input type="Button" value="Find" onClick="showelement()">
  <br>
  <Input type="Reset" value="Clear"/>
</form>
</BODY>
</HTML>
```

Команды:

```
var odoc=new ActiveXObject("Microsoft.XMLDOM");
odoc.async=false;
odoc.load("textbd.xml");
```

используются для создания объекта `odoc`, содержащего набор записей XML. Имя объекта `odoc` выбрано произвольно. Функция скрипта реализована между тегами `<script>`, `</script>`. Для активизации функции нужно в текстовом поле:

```
<input type="Text" name="mytext"   value=""/>
```

набрать полное или сокращенное название книги и нажать кнопку **Find**. Для этого предварительно следует очистить содержимое текстового поля кнопкой **Clear**.

Задание

1. Создать свою собственную XML-базу, например: футбольные команды, отдел кадров, косметика и пр.
2. Разработать HTML-сайт для отображения данных.
3. Реализовать поиск информации по нескольким условиям (например, год выпуска (создания, рождения) = 2000 и цена <10 000).

В качестве результата выполнения задания продемонстрировать действующие сайт и XML-базу.

Контрольные вопросы

1. Что представляют собой документы XML?
2. Объясните синтаксис документов XML.
3. Как отобразить документ XML в таблице HTML?
4. Как JavaScript позволяет выполнять поиск информации в XML-документе?
5. Что выполняет команда

   `var odoc=new ActiveXObject("Microsoft.XMLDOM");` ?
6. Что делает команда

   `odoc.getElementsByTagName("title").item(i).text;`?

Взаимодействие XML-Java-JavaScript

Цель занятия

Познакомиться с механизмом совместного использования XML, Java, JavaScript. Изучить технику вызова методов Java из скриптов JavaScript и передачу им параметров. Передать данные из XML-таблицы в апплет и отобразить их графически. Дополнительную информацию можно найти в [13].

Краткие теоретические сведения

На предыдущем занятии были рассмотрены вопросы, связанные с работой с документами XML средствами JavaScript и HTML. Теперь мы включим еще и механизмы Java, что позволит существенно расширить наши пользовательские возможности. Поставим такую задачу. Пусть в XML-базе данных хранятся сведения по котировке валют:

```
<?xml version="1.0"?>
<root>
<item>
<name> usa dol.</name>
<value>25</value>
</item>
<item>
<name> austr dol.</name>
<value>130</value>
</item>
<item>
<name> canadian dol.</name>
<value>240</value>
</item>
<item>
<name> south africa dol.</name>
<value>80</value>
```

```
</item>
<item>
<name> new zeland dol.</name>
<value>300</value>
</item>
<item>
<name> Filippines dol.</name>
<value>100</value>
</item>
</root>
```

Это правильно составленный документ XML, элементами которого являются `<root>`, `<item>`, `<name>`, `<value>`.

Наберите этот текст в любом редакторе и сохраните его как простой текстовый файл с расширением .xml, например, с именем kotirovka.xml. Теперь просмотрите этот файл браузером Internet Explorer.

Создайте следующий html-файл (листинг 2.15).

Листинг 2.15. Документ kotirovka.html

```
<html>
<head>
<H1> Example of interaction XML-JAVASCRIPT-JAVA</h1>
<script>
<!--
  function graphic()
 {
  //first -load xml base
      var odoc=new ActiveXObject("Microsoft.XMLDOM");
         odoc.async=false;
         odoc.load("kotirovka.xml");

        z=odoc.getElementsByTagName("name").length;
        var valutaval =new String(""); // строка значений
                                        // курсов
        var valutaname=new String(""); // строка имен валют
```

```
//loading the arrays with values from database
   for(var i=0;i<z;i++)
   {

valutaval+=odoc.getElementsByTagName("value").item(i).text;
                             // добавляем очередной курс
    valutaval=valutaval+"?"; // добавляем разделительный
                             // символ

valutaname+=odoc.getElementsByTagName("name").item(i).text;
    // добавляем имя валюты
    valutaname=valutaname+"?"; // добавляем разделительный
                               // символ
   }
    document.aplet.graph("SHOW",valutaval,valutaname);
    // вызов метода graph() java-апплета
    // передаем ему параметры: строку "SHOW", строку valutaval
    // и строку valutename
}
-->
</script>
</head>
<body bgcolor=#aabbcc>
<Br/>
<applet code="lab1.class" width=500 height=300 name="aplet">
// апплет, который вызываем из JavaScript
</applet>
<br/>
<Button onClick="graphic()">Launching Java</Button>
</body>
</html>
```

Сохраним этот HTML-файл под именем kotirovkajava.html. В этом файле реализованы три момента.

Во-первых, чтение XML-базы в объектную переменную `odoc` в реализации функции JavaScript с именем `graphic()`:

```
<script>
<!--
  function graphic()
 {
  //first -load xml base
      var odoc=new ActiveXObject("Microsoft.XMLDOM");
         odoc.async=false;
         odoc.load("kotirovka.xml");
....   ....   ....
```

Последняя команда загружает собственно интересующий нас документ.

Во-вторых, далее в цикле формируются две длинные строки `valutaval` и `valutaname`:

```
for(var i=0;i<z;i++)
   {

valutaval+=odoc.getElementsByTagName("value").item(i).text;
    valutaval=valutaval+"?";

valutaname+=odoc.getElementsByTagName("name").item(i).text;
    valutaname=valutaname+"?";
   }
```

Строка `valutaname` содержит названия валют, разделенных знаком `"?"`. Строка `valutaval` содержит котировки валют (целые числа), разделенные знаком `"?"`.

Теперь, в-третьих, когда эти строки подготовлены, наступает решающий момент — вызов функции Java из тела скрипта JavaScript. Вот этот вызов:

```
document.aplet.graph("SHOW",valutaval,valutaname);
```

Здесь вызывается метод по имени `graph()`. Указывается имя апплета — `aplet`, записанное в параметре `name` тега `<applet>` документа HTML. Методу `graph()` передаются три параметра, два из которых — длинные строки, сформированные выше. Этот вызов передается HTML-объекту `document.aplet`. Данный объект объявлен в HTML-файле таким образом:

```
<applet code="lab1.class" width=500 height=300
name="aplet">
```

```
</applet>
```

Теперь уже должно быть ясно, как вызывается Java-апплет: код апплета содержится в классе `lab1.class`.

Мы пока не привели код Java, но результат его выполнения представлен на рис. 2.10.

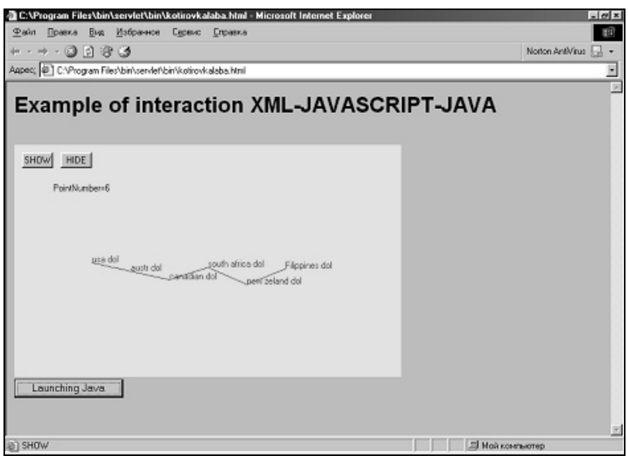


Рис. 2.10. Представление котировок валют

Заметим, что вполне понятно естественное желание передать в метод Java не длинные строки, а массивы, однако (увы!), это не делается просто.

Теперь приводим текст Java-апплета (листинг 2.16), который содержит почти в точности Java-программу из листинга 2.5, рисующую график. Но там рисовали синусоиду, а здесь — просто набор ломаных линий.

Листинг 2.16. Приложение на основе апплета для рисования графика котировок валют

```
import java.applet.*;
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class lab1 extends Applet implements ActionListener
```

```
{
 Button btnshow=new Button("SHOW"); // Кнопка для демонстрации
                                       // графика курса валют
 Button btnhide=new Button("HIDE"); // Кнопка для очистки
 String msg="";
 // В этот массив будут помещены курсы валют:
 public int [] bar={50,40,20,80,100,10,40,20,50,30};
 // В этот массив будут помещены названия валют:
 public String[]
 names={"One","Two","Three","Four","Five","Six","Seven",
                        "Eight","Nine","Ten"};
 public int pointnumber=10; // Число точек массива
 public void paint(Graphics g)
 {
  int x,y,xold,yold,step=50;
  g.setColor(Color.blue);
  g.drawString("PointNumber="+pointnumber,50,60);
  g.setColor(Color.red);
  if (msg.equals("show")) // Отображение графика курсов при
                            // msg="show"
   {
    xold=100;
    yold=bar[0]+150;
    g.drawString(names[0],xold,yold); // Вывод названия первой
                                        // валюты
    int i=1;
    while (i<pointnumber)
    {
     x=xold+step;
     y=bar[i]+150;
     g.drawLine(xold,yold,x,y); // Рисование ломаных линий,
                                  // связывающих курсы валют
     g.drawString(names[i],x,y); // Вывод названий других
                                   // валют
     xold=x;
```

```
    yold=y;
    i++;
   }
 }
}

public void graph(String msg1,String msgvalues, String
msgnames)
// Этот метод вызываем из JavaScript; он должен быть объявлен
// как public
{
 if(msg1.equals("SHOW"))
  {
   int j=0;
   while((msgnames.length()>0)&&(msgvalues.length()>0))
   {
    int k1=msgnames.indexOf("?");
    int k2=msgvalues.indexOf("?");
    String s1=msgnames.substring(0,k1-1);
    String s2=msgvalues.substring(0,k2-1);

    bar[j]=Integer.valueOf(s2).intValue();
    names[j]=s1;
    j++;
    pointnumber=j;
    if (msgnames.length()>(k1+1))
     {
      msgnames=msgnames.substring(k1+1);
     }
    else
     {
      break;
     }

    if (msgvalues.length()>(k2+1))
```

```
      {
       msgvalues=msgvalues.substring(k2+1);
      }
     else
      {
       break;
      }
    }
   }
  Graphics g=getGraphics();
  if(msg1.equals("SHOW"))
   {
    msg="show";
    repaint();
   }
  else
   if(msg1.equals("show"))
    repaint();
   else
    if (msg1.equals("hide"))
     {
      g.setColor(getBackground());
      g.fillRect(0,0,getSize().width,getSize().height);
     }
  showStatus(msg1);
 }
 public void init()
 {
  setLayout(null);
  add(btnshow);
  add(btnhide);
  btnhide.setBounds(60,10,40,20);
  btnshow.setBounds(10,10,40,20);
  btnshow.addActionListener(this);
```

```
  btnhide.addActionListener(this);
  setBackground(Color.yellow);
 }

 public void actionPerformed(ActionEvent a_e)
 {
  if (a_e.getSource()==btnshow)
   {
    msg="show";
    graph(msg,msg,msg);
   }
   else
    if (a_e.getSource()==btnhide)
     {
      msg="hide";
      graph(msg,msg,msg);
     }
 }
}
```

Находим здесь метод `show()`. Он должен быть объявлен как `public`. Мы видим, что его смысл — распаковать длинные строки и сформировать из них *массивы*: один для названий валют, второй — для их котировок. В качестве разделителей между словами в длинных строках использовался символ `"?"`. Построенные массивы прорисовывает метод `paint()`.

Следует обратить внимание на то, как объявлены и используются массивы.

```
public int [] bar={50,40,20,80,100,10,40,20,50,30};
public String[] names=
 {"One","Two","Three","Four","Five","Six","Seven","Eight",
 "Nine","Ten"};
```

В обоих приведенных случаях с левой стороны стоит объявление массива, а с правой после знака `"="` — выражение для инициализации перечислением элементов массива в фигурных скобках.

Ссылка на элемент массива в тексте программы выполняется так:

```
bar[j]=Integer.valueOf(s2).intValue();
names[j]=s1;
```

Видим по объявлению, что массив `bar` есть набор целых чисел. Команда:

```
bar[j]=Integer.valueOf(s2).intValue();
```

использует метод `valueOf()` класса `Integer`. Здесь `s2` — строка, содержащая целое число. Приведенная выше команда преобразует значение `s2` в целое число. Поскольку преобразование строки в число — довольно часто возникающая задача, то следует запомнить эту команду, основанную на использовании класса `Integer`. В этой работе также используем команды сравнения строк, выделения подстроки, получения позиции вхождения подстроки в строку:

```
if(msg1.equals("SHOW"))
 {
  int j=0;
  while((msgnames.length()>0)&&(msgvalues.length()>0))
  {
   int k1=msgnames.indexOf("?");
   int k2=msgvalues.indexOf("?");
...
```

Сравнение строк выполняет оператор: `msg1.equals("SHOW")`. Этот оператор возвращает значение `true`, если строка `msg1` совпадает со строкой "`SHOW`", и `false` — в противном случае. Оператор `int k1=msgnames.indexOf("?")` присваивает переменной `k1` значение первой позиции (по порядку с начала строки), в которой встретился символ `"?"`. Оба рассмотренных оператора являются методами класса `String`, поэтому, если s — объектная переменная класса `String`, то можно выполнять присваивания типа `boolean z = s.equals("Интересно")` или `int i = s.indexOf("тересно")`.

Таким образом, поставленная цель достигнута. Теперь следует выполнить индивидуальное задание, помещенное далее.

Задание

1. Создать свою собственную XML-базу. Например: футбольные команды, отдел кадров, косметика и пр.
2. Разработать HTML-сайт для отображения данных.
3. С помощью Java осуществить вывод данных из XML-базы в виде простой таблицы.

Контрольные вопросы

1. Как вызвать метод Java-апплета из скрипта JavaScript?
2. Как передать массив параметров в метод Java-апплета.?
3. С каким модификатором доступа должен быть объявлен метод Java, вызываемый из апплета?
4. Можно ли передать в качестве параметра из JavaScript в Java целое число? переменную типа даты? (Проверьте.)
5. Объясните способ распаковки строки-параметра, переданной в метод `show()`.

Чтение XML-файла с использованием файлового диалога

Цель занятия

В этом занятии надлежит познакомиться с работой компонента файлового диалога для выбора XML-базы и загрузки ее в текстовую область для выполнения последующего поиска. Будет использована база данных, содержащая сведения о курсах валют. Поиск котировки валюты осуществляется по ее названию. Диалоговый интерфейс предполагает использование компонента `JOptionPane` пакета `Swing`. Более подробно о компонентах пакета `Swing` и их методах можно прочитать в [11].

Краткие теоретические сведения

Класс файлового диалога называется `FileDialog`. Этот класс позволяет открыть окно для выбора имени файла, содержимое которого требуется прочитать или записать. Данный класс со-

держит методы `setFile()` для установки отображаемых файлов по их расширению, `getFile()` для получения имени выбранного файла и др. При запуске *файлового диалога* по команде `show()` откроется окно диалога следующего вида (рис. 2.11).

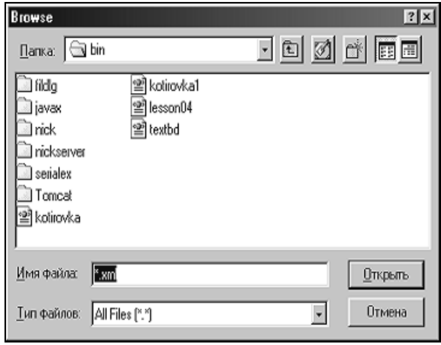


Рис. 2.11. Окно файлового диалога

В этом окне следует просто выбрать требуемый XML-файл, содержимое которого откроется в текстовой области, как показано на рис. 2.12.

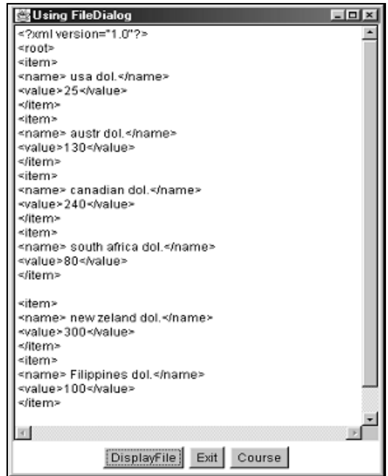


Рис. 2.12. Открытие содержимого XML-файла в текстовой области

По щелчку мыши на кнопке **Course** появится приглашение ввести название валюты (рис. 2.13).

Рис. 2.13. Приглашение для ввода названия валюты

Для отображения окна на рис. 2.13 используется команда:

```
String s=JOptionPane.showInputDialog(null, "Input currency
title").toUpperCase();
```

Результат работы этого диалога заключается в формировании ответа в строке `s`, приведенной к верхнему регистру. На основании введенного названия валюты осуществляется поиск соответствующей записи в XML-файле. Теперь по нажатию **OK** появится и соответствующая котировка.

Для объявления переменной типа файлового диалога используем команду `private FileDialog loader`.

Переменная `loader` создается конструктором так:

```
loader=new FileDialog(this,"Browse",FileDialog.LOAD);
loader.setFile("*.xml"); // Устанавливаем тип отображаемых
                         // файлов
```

Открытие диалогового файла выполняет команда `loader.show()`. Выбранный файл должен быть прочитан. Для этой цели вводим объявления:

```
File file=new File(filenamestr);
FileInputStream in=new FileInputStream(file);
DataInputStream dis= new DataInputStream(in);
```

Чтение файла и отображение его содержимого в текстовой области выполняется методом `display()` таким образом:

```
fileArea.setText("");
String line=dis.readLine(); // Читаем строку из файла
while (line !=null)
{
 //Добавляем прочитанную строку в текстовую область:
 fileArea.append(line+"\n");
 // Проверяем, содержит ли строка тег <name>:
 if (line.indexOf("name") >=0)
 {
  if (line.indexOf("name") >=0)
   {
    // Если содержит, то переводим символы строки в верхний
    // регистр:
    String line1=line.toUpperCase();
    if (line1.indexOf(s)>=0) // Проверяем, совпало ли имя
                             // валюты в теге <name>
     { priz=1;} //Если совпало, то priz=1
   }
  line=dis.readLine(); // Считываем строку с тегом <value>
  if((priz==1)&&(line.indexOf("value")>=0))
   {
    priz=0;
    JOptionPane.showMessageDialog(null,line);
   } // Выводим котировку валюты
 }
}
```

Объединим (листинг 2.17) эти сведения в единое приложение, добавив необходимые кнопки, назначение которых пояснено в комментариях.

Листинг 2.17. Работа с диалоговым файлом

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;
import javax.swing.event.*;
public class FileDialogS extends Frame // Базовый класс
                                       // приложения
implements ActionListener
{
 public static void main(String[] args)
 {
  new  FileDialogS();
 }

 public Button loadButton; // Кнопка для открытия файлового
                           // диалога
 public Button exitButton; // Кнопка для выхода из приложения
 public TextArea fileArea; // Текстовая область
                           // для вывода содержимого файла
 public Button tfPoisk; // Кнопка для ввода названия валюты и
                        // инициализации поиска котировки

 private FileDialog loader; // Этот компонент используется для
                            // выбора файла из диалогового
                            // окна
 public String filenamestr;  // Здесь помещается имя
                             // выбранного файла

 public FileDialogS()
```

```
{
 super("Using FileDialog"); // Вызов конструктора
                            // родительского класса
 loadButton=new Button("DisplayFile");
 loadButton.addActionListener(this);
 tfPoisk=new Button("Course"); // Подключение прослушивателя
 exitButton=new Button("Exit");
 exitButton.addActionListener(this);
 tfPoisk.addActionListener(this);
 Panel buttonPanel=new Panel(); // Создание панели
                                // для размещения кнопок
 // Добавление кнопки на панель
 buttonPanel.add(loadButton,BorderLayout.SOUTH);
 // Используется компоновщик BorderLayout, располагающий
 // компоненты
 // по сторонам света
 buttonPanel.add(exitButton,BorderLayout.NORTH);
 buttonPanel.add(tfPoisk,BorderLayout.CENTER);
 // Сама панель добавляется на форму:
 add(buttonPanel,BorderLayout.SOUTH);
 fileArea=new TextArea();
 // Текстовая область добавляется на форму:
 add(fileArea,BorderLayout.CENTER);
 // Создание переменной файлового диалога:
 loader=new FileDialog(this,"Browse",FileDialog.LOAD);
 // Установка расширений просматриваемых файлов:
 loader.setFile("*.xml");
 setSize(350,500); // Задаем размеры формы
 setVisible(true); // Делаем форму видимой
}
// Обработка событий от кнопок:
public void actionPerformed(ActionEvent event)
{
 if (event.getSource()==loadButton)
```

```
   {
    loader.show(); // Отображает диалоговое окно для выбора
                   // имени файла
    filenamestr=loader.getFile(); // Получаем имя файла по
                                  // getFile()
    displayFile(filenamestr); // Отображает содержимое
                              // выбранного файла
   }
  else
   if(event.getSource()==exitButton)
    {
     System.exit(0); //Завершение приложения
    }
   else
    if(event.getSource()==tfPoisk) // Поиск котировки валюты
     {
      String s=JOptionPane.showInputDialog(null,
         "Input currency title").toUpperCase();
      // команда JOptionPane.showInputDialog() отображает окно
      // для ввода названия валюты
      try
      {
       // Выполняем поиск котировки указанной валюты
       // в файле filenamestr
       int priz=0;
       File file=new File(filenamestr);
       FileInputStream in=new FileInputStream(file);
       DataInputStream dis= new DataInputStream(in);
       // Создаем потоковую переменную dis
       // для чтения XML-файла filenamestr
       fileArea.setText("");
       String line=dis.readLine(); // Читаем строку из файла
       while (line !=null)
       {
        fileArea.append(line+"\n");
```

```
       if (line.indexOf("name") >=0) // Проверяем наличие
                                     // тега <name>
        {
         String line1=line.toUpperCase();
         if (line1.indexOf(s)>=0) // Проверяем, совпало ли
                                  // имя валюты в теге <name>
          {priz=1;}
        }
       line=dis.readLine(); // Если да, то считываем строку
                            // с тегом <value>
       if((priz==1)&&(line.indexOf("value")>=0))
        {
         priz=0;
         JOptionPane.showMessageDialog(null,line);
        } // Выводим котировку
      }
     }
     catch(IOException oe)
     {
      fileArea.setText("Cannot read file "+filenamestr);}
     }
}
// Этот метод выводит содержимое файла в текстовую область:
public void displayFile(String filename)
{
 try
 {
  File file=new File(filename);
  FileInputStream in=new FileInputStream(file);
  int fileLength=(int) file.length();
  byte[] fileContents =new byte[fileLength];
  in.read(fileContents); // Читаем содержимое файла
                         // в массив байтов fileContents
  // Используем низкоуровневый файловый ввод:
  String fileContentsString=new String(fileContents);
```

```
   // Преобразуем массив байтов в строку
    fileArea.setText(fileContentsString); // Выводим строку
                                          // в текстовую
                                          // область
   }
   catch (IOException ioe)
   {
    fileArea.setText("Error:"+ioe);
   }
  }
}
```

Здесь компонент *файлового диалога* объявлен как `loader`:

```
loader=new FileDialog(this,"Browse",FileDialog.LOAD);
loader.setFile("*.xml");
```

Его запуск реализован в обработчике события, связанного с щелчком мыши на кнопке:

```
if (event.getSource()==loadButton)
   {
    loader.show();
    filenamestr=loader.getFile();
    displayFile(filenamestr);
   }
```

Поиск котировки введенной валюты выполнен так:

```
else
 if(event.getSource()==tfPoisk)
   {
    // Открывается диалоговое окно для ввода котировки
    // (рис. 2.13):
    String s=JOptionPane.showInputDialog(null,"Input currency
                                         title").toUpperCase();
    try
     {
      int priz=0;
```

```
File file=new File(filenamestr); // Объявление потоковых
                                  // переменных
FileInputStream in=new FileInputStream(file);
DataInputStream dis= new DataInputStream(in);
fileArea.setText("");
String line=dis.readLine(); // Чтение строки из файла
while (line !=null)
{
 fileArea.append(line+"\n"); // Добавление строки
                             // в текстовую область
 // К строке добавляется символ конца строки - \n
 if (line.indexOf("name") >=0) // Проверка на наличие
                               // в строке тега <name>
  {
   String line1=line.toUpperCase();
   if (line1.indexOf(s)>=0) // Проверка на совпадения
                            // названия введенной валюты
    {priz=1;}
  }
 line=dis.readLine();
 if((priz==1)&&(line.indexOf("value")>=0))
  {
   priz=0;
   JOptionPane.showMessageDialog(null,line);
   } // Вывод котировки в диалоговом окне
}
```

Видим, что процедура поиска котировки снова считывает файл и определяет позицию строки с введенным названием валюты.

Задание

1. Добавьте в программу возможность добавления новой валюты в XML-базу.
2. Добавьте в программу возможность удаления валюты из XML-базы.

Контрольные вопросы

1. Какой класс реализует окно файлового диалога? Назовите методы этого класса.
2. Назовите команду для ввода строки в диалоговом окне.
3. Какой командой выполняется проверка того, что данная строка содержится в другой строке?
4. Объясните работу метода `displayFile()` из листинга 2.15.
5. Объясните реализованный в листинге 2.15 метод поиска котировки валюты с известным названием.

Потоки в Java

Цель занятия

Познакомиться с классом потоков `Thread` и возможностями его использования для реализации динамических функций приложения (апплета). Изучить использование главного метода потока `run()`, а также применение интерфейса `Runnable`. Подробное описание потоков изложено в [15].

Краткие теоретические сведения

Поток представляет собой процесс, выполняющийся параллельно с основной программой. Поток нужно объявить, создать и запустить. Описание класса потока можно проиллюстрировать следующим примером. Предположим, у нас имеется клиент банка, который периодически производит вклад денег на свой счет или их изъятие со счета. Тогда параллельно основной программе следует создать поток, моделирующий поведение клиента. Далее приведен пример такого потока:

```
class mythread extends Thread
{
 int sum,i;

 public void run()
```

```
 {
  while(true)
   {
    if (Math.random()>0.4)
     {i=1;} else {i=-1;}
    try
     {sleep(100);}
    catch(Exception err)
     {System.out.println(""+err);}
    Account.putmoney((int) (i*Math.random()*100));
   }
  }
}
```

Класс потока должен расширять базовый класс `Thread`. В примере класс потока назван `mythread`. Запомните, что у потока главным методом является `run()`. Как только этот метод выполнится, поток будет уничтожен. Поэтому при программировании потока следует записать свой собственный код для `run()`. В приведенном примере метод `run()` запрограммирован так, что он выполняется в бесконечном цикле `while(true) {...}`. На каждой очередной итерации цикла поток вызывает метод из другого класса: `Account.putmoney()`, который и моделирует занесение денег или их изъятие со счета в сумме, определяемой как `i*Math.random()*100)`. Заметим, что если `i==1`, то деньги добавляются к счету; если `i==-1`, то деньги изымаются со счета. Наконец, отмечаем, что команда `sleep(100)` приостанавливает работу потока на 100 миллисекунд (1 милисекунда = 0.001 с). Поэтому очередная банковская операция клиента происходит с задержкой на это время. Некоторые команды могут повлечь за собой возникновение системных ошибок. Java требует такие команды *охранять*. Охрана состоит в применении конструкции `try-catch`. То, что записывается в фигурных скобках после `try`, — есть охраняемая последовательность команд. То, что записывается в фигурных скобках после `catch`, — есть последовательность действий, выполняемая в случае возникновения системной ошибки. В нашем примере это выдача сообщения об ошибке на системную консоль:

`System.out.println(""+err);`.

Теперь обратимся к классу `Account`. Он определен следующим образом.

```
class Account
{
 static int balance=10000;
 static public void putmoney(int amount)
 {
  balance=balance+amount;
 }
}
```

Этот класс имеет следующее назначение: в нем находится статическая переменная `balance`, содержащая текущий счет, а также метод для моделирования операций со счетом:

```
static public void putmoney(int amount)
 {balance=balance+amount;}
```

Суть этого метода проста: он получает на входе сумму `amount` и добавляет ее к балансу `balance`. Итак, запомним, что метод `Account.putmoney()` вызывается из потока, имитирующего клиента, а *не реализуется* в самом потоке.

Остается отметить назначение самого апплета. На рис. 2.14 представлено его окно после запуска.

На апплете размещена кнопка, при нажатии которой отображается текущее состояние счета клиента.

Событие от программной кнопки запрограммировано так:

```
public void actionPerformed(ActionEvent a_e)
{
 summa=Account.balance;
 repaint();
}
```

Переменной `summa` присваивается текущее значение переменной `balance` класса `Account`. Одновременно обратите внимание на то, как выполняется обращение к этой переменной. Завершающая команда `repaint()` обязательна для перерисовки окна апплета. Команда `repaint()` запускает метод `paint()`.

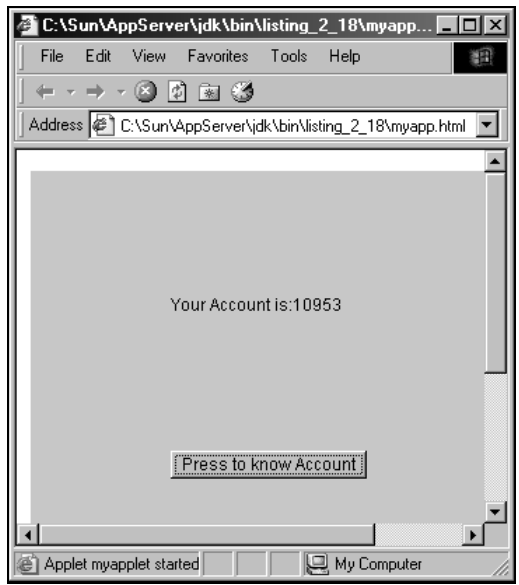


Рис. 2.14. Окно апплета с данными банковского счета

Метод `paint()` выполняет отображение суммы в окне апплета:

```
public void paint(Graphics g)
{
 setBackground(Color.pink);
 g.setColor(Color.white);
 g.drawString("Your Account is:"+summa,100,100);}
}
```

Обратите внимание на то, что аргумент `"Your Account is:"+summa` есть строка, хотя он получается путем присоединения к строке целого числа, представленного переменной `summa`.

Наконец, приведем текст инициализации окна апплета:

```
public class myapplet extends Applet implements ActionListener
{
 mythread myth=new mythread();
 int summa=0;
 Button btn=new Button("Press to know Account");
```

```
 public void init()
 {
  setLayout(null);
  add(btn);
  btn.setBounds(100,200,120,20);
  btn.addActionListener(this);
  myth.start();
 }
...
```

Здесь следует обратить внимание на следующее. Объявление потока производится так:

```
mythread myth=new mythread();
```

Здесь `mythread` — это имя класса потока (определено ранее). Переменная `myth` — это имя потока. Запуск потока на выполнение параллельно с основной программой реализуется в строке:

```
myth.start();
```

Теперь объединим все рассмотренные секции в единое приложение (листинг 2.18).

Листинг 2.18. Работа с потоком

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class myapplet extends Applet implements ActionListener
{
 mythread myth=new mythread();
 int summa=0;
 Button btn=new Button("Press to know Account");

 public void init()
 {
  setLayout(null);
  add(btn);
  btn.setBounds(100,200,120,20);
  btn.addActionListener(this);
```

```
  myth.start();
 }

 public void actionPerformed(ActionEvent a_e)
 {
  summa=Account.balance;
  repaint();
 }
 public void paint(Graphics g)
 {
  setBackground(Color.pink);
  g.setColor(Color.white);
  g.drawString("Your Account is:"+summa,100,100);
 }
}

class Account
{
 static int balance=10000;
 static public void putmoney(int amount)
 {
  balance=balance+amount;
 }
}
class mythread extends Thread
{
 int sum,i;
 public void run()
 {
  while(true)
   {
    if (Math.random()>0.4)
     { i=1; }
    else {i=-1; }
    try
     {sleep(100);}
    catch(Exception err)
```

```
    {System.out.println(""+err);}
   Account.putmoney((int) (i*Math.random()*100));
  }
 }
}
```

Пусть требуется перерисовывать окно апплета по инициативе не кнопки, а потока. Для этого нужно динамически вызывать метод `repaint()` апплета. Попытки написать вызов типа `myapplet.repaint()` либо `myapplet.paint()` приведут нас к неразрешимой проблеме, связанной с тем, что метод `repaint()` объявлен как `static`. Напомним, что если метод объявлен как `static`, то и все его переменные должны быть `static`. Проблема состоит в том, что в вызове `myapplet.repaint()` `myapplet` — есть имя класса, а не объекта. А ссылки на класс всегда предполагают тип `static`.

Имеется иная возможность — породить апплет-поток, т. е. апплет, совмещающий в себе поток с его методами и главным методом `run()`. Рассмотрите тот же пример, но без кнопки (листинг 2.19).

Листинг 2.19. Запуск потока без кнопки

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class myapplet2 extends Applet implements Runnable
{
 Thread myth;
 int i,summa=0;

 public void init()
 {
  myth=new Thread(this);
  myth.start();
 }
 public void paint(Graphics g)
 {
  setBackground(Color.pink);
  g.setColor(Color.white);
```

```
  g.drawString("Your Account is:"+summa,100,100);
 }
 public void run()
 {
  while(true)
  {
   if (Math.random()>0.4)
    {i=1;}
   else {i=-1;}
   try
    {myth.sleep(1000);}
   catch(Exception err)
    {System.out.println(""+err);}
   summa=summa+((int) (i*Math.random()*100));
   repaint();
  }
 }
}
```

Поток `myth`, созданный в апплете, имеет доступ к методу `paint()` апплета и осуществляет его вызов в теле метода `run()`:

```
public void run()
{
 {
  while(true)
  {
   if (Math.random()>0.4)
    {i=1;}
   else
    {i=-1;}
   try
    {myth.sleep(1000);}
   catch(Exception err)
    {System.out.println(""+err);}
   summa=summa+((int) (i*Math.random()*100));
```

```
    repaint();
   }
  }
}
```

Такой способ реализации потоков в апплете позволяет динамически перерисовывать апплет в соответствии с логикой потока. Это дает возможность получать эффекты типа бегущей строки, мерцаний, смены рисунков и пр.

Задание

По образу и подобию представленной программы (листинги 2.18, 2.19) напишите свою собственную согласно варианту. Вам потребуются дополнительные указания к выполнению этих заданий, помещенные ниже. Варианты заданий:

1. Поток инициирует появление в окне апплета в случайные моменты времени цветных кружков в различных местах экрана.
2. Поток инициирует запись в окне апплета большими буквами текущей системной даты.
3. Поток инициирует отображение бегущей строки.
4. Напишите программу игры двух человек в спички. На экране отображается текущее число спичек в куче, например, 12. Игроки ходят по очереди. Когда игрок ходит, он нажимает кнопку (для каждого игрока — своя кнопка). Программа в процедуре `actionPerformed()` распознает, кто ходит, для определения победителя. Перед ходом игрок вводит в текстовом поле типа `TextField` число забираемых из кучи спичек: 1, 2 или 3. Проигрывает тот, перед ходом которого в куче остается 4 или менее спичек, но не менее одной.

Указания

Чтобы отобразить текст в окне апплета большими буквами следует использовать объект класса `Font`. Пример:

```
public void paint(Graphics g)
{
 String text="HY-HY";
```

```
  Font ff=new Font("Arial",Font.PLAIN,20);
  g.setFont(ff);
  g.setColor(new Color(200,100,30));
  g.drawString(text,20,20);
}
```

Для программирования бегущей строки и появления шаров используйте второй вариант программы с интерфейсом `Runnable`. Задание 4-го варианта выполняется по примеру первой программы.

Чтение текстовой строки из текстового поля выполняется командой `txt.getText()`, где `txt` — имя поля. Если `z` — переменная типа `int`, то команда `z=Integer.valueOf(txt.getText()).intValue()` присваивает переменной `z` значение из текстового поля.

Контрольные вопросы

1. Объясните концепцию потоков в Java. Каково их назначение и возможные способы применения?
2. Как запустить поток в апплете? Объясните назначение интерфейса `Runnable`.
3. Объясните работу программы листинга 2.19.
4. Объясните реализованный в листинге 2.17 метод поиска котировки валюты с известным названием.
5. Самостоятельно выясните, как задавать и определять имя потока, а также получать данные потока?

Создание приложений "клиент-сервер"

Цель занятия

Научиться создавать приложения на основе технологии "клиент-сервер". Изучить способ описания потока клиента и потока сервера. Освоить технику передачи данных от сервера клиенту. За подробной информацией следует обращаться в [15].

Краткие теоретические сведения

В приложениях "клиент-сервер" имеется два параллельно выполняющихся *потока*: *поток-сервер* и *поток-клиент*. В общем случае эти потоки выполняются на разных машинах. Поток-сервер обслуживает клиентов в централизованном режиме. Это значит, что клиенты обращаются к нему за общими данными из централизованной базы данных. Сервер анализирует поступившие запросы и выполняет поиск в базе, затем возвращает результат клиенту. Клиент также может передавать данные серверу, обеспечивая двунаправленную передачу информации.

Для установления связи между клиентом и сервером в программе как клиента, так и сервера нужно создать сокетное (socket — розетка (гнездо)) соединение.

В клиенте это делается так:

```
class clientThread extends Thread
{
 DataInputStream dis=null;
 Socket s=null;
 public clientThread()
 {
  try{
   s=new Socket("127.0.0.1",2525);
   ...
```

Необходимо объявить переменную типа `Socket` и затем создать соответствующий объект командой:

```
s=new Socket("127.0.0.1",2525);
```

Параметрами конструктора являются *сетевой адрес* "127.0.0.1" и номер порта 2525, задаваемый произвольно.

Замечание

Приведенный здесь сетевой адрес определяет собственный компьютер, т. е. и клиент и сервер функционируют на одном и том же компьютере. Этот адрес полезно запомнить, поскольку для отладки сетевых программ в этом случае нет необходимости располагать настоящим сервером.

> Номер порта, устанавливаемый в команде, не должен совпадать с некоторыми стандартными портами, например, с номером 80. Можно узнать сетевое имя и сетевой адрес компьютера Это выполняется таким образом: **My Computer** (Мой компьютер) | <правая кнопка мыши> | **Properties** (Свойства) | **Network Identification** (Сетевая идентификация), а затем по имени компьютера определить его сетевой номер, введя в командной строке меню **Start** (Пуск) | **Run** (Выполнить инструкцию):
>
> `ping сетевое_имя_компьютера`

На сервере создается поток, который также объявляет сокетное соединение:

```
ServerSocket server;
String amountstring;
static int amount=200;
public void run()
{
 try
  {
   server= new ServerSocket(2525);
  }
 catch(Exception e)
  {
    System.out.println("ERRSOCK+"+e);
  }
```

Небольшое отличие от *клиента*: используется класс и конструктор `ServerSocket`.

После создания сокетного соединения нужно дождаться фактического соединения с сервером. Сервер все время активен и прослушивает *сокет*. Вот как это делается в программе:

```
while(true)
 {
  Socket s=null;
  try
   { s=server.accept(); }
  catch(Exception e)
```

```
   {System.out.println("ACCEPTER"+e);}
  try
   { (что-то делается с реальным сокетом s)}
```

Реальное подключение клиента к серверу реализует команда `s=server.accept()`, которая устанавливает "физический" контакт при наличии сообщения от клиента. Заметим, что приложение сервера "замирает" на команде `s=server.accept()`, ожидая подключения клиента.

Как только такое соединение установлено, нужно установить канал ввода-вывода между сервером и клиентом. В работе *сервер* посылает клиенту данные о счете, моделируемые как случайные числа. Значит, сервер в системе ввода-вывода должен записать данные в сокет, а клиент — их прочитать. Вот как это делается на стороне сервера:

```
class Account extends Thread
{
 ServerSocket server;
 String amountstring;
 static int amount=200;

 public void run()
 {
  try
   {
    server= new ServerSocket(2525); // Номер сокета 2525
                                    // выбран произвольно
   }
  catch(Exception e)
   { System.out.println("ERRSOCK+"+e); }
  while(true)
  {
   Socket s=null;
   try
    {
     s=server.accept(); // Ожидание соединения с клиентом
    }
```

```
   catch(Exception e)
    {System.out.println("ACCEPTER"+e);}
   try
    {
     PrintStream ps=new PrintStream(s.getOutputStream());
     // Класс PrintStream предназначен для текстового вывода
      int amountcur=((int)(Math.random()*1000));
      // Определяется
     // случайная величина текущего вклада с учетом знака;
     // отрицательный вклад – это снятие части денег со счета
     if (Math.random()>0.5)
      amount-=amountcur; // Сумма на счете изменяется случайно
     else
      amount+=amountcur;
     Integer x=new Integer(amount);
     amountstring=x.toString(); // Преобразование целого числа
                                // в строку
     ps.println("Account:"+amountstring); // Здесь выполняется
                                          // передача строки клиенту
     ps.flush();
     s.close(); // Сокетное соединение закрывается
    }
   catch(Exception e)
    {
     System.out.println("PSERROR"+e);
    }
 }
```

Поток вывода для сервера создается командой:

```
PrintStream ps=new PrintStream(s.getOutputStream());
```

Объект `ps` будет держать все методы вывода, включая главный:

```
ps.println("Account:"+amountstring);
```

На счет добавляется случайная величина:

```
amount=((int)(Math.random()*1000));
```

Класс `Math` содержит метод `random()` для генерации случайного числа от 0 до 1. Мы видим, что значение `Math.random()*1000` не является в общем случае целым числом, поэтому выполняем явное преобразование типа, записав перед `Math.random()*1000` ключевое слово (`int`). Добавление или уменьшение суммы в переменной `amount` выполняется путем внесения или снятия случайной величины со счета на основании проверки:

```
int amountcur=((int)(Math.random()*1000));
// Определяется случайная величина текущего вклада с учетом
// знака; отрицательный вклад – это снятие части денег
// со счета
if (Math.random()>0.5)
 amount-=amountcur; // Сумма на счете изменяется случайно
else
 amount+=amountcur;
```

Обратим внимание на то, что в конце цикла (заметим, бесконечного) реальное соединение сервера и клиента разрывается:

```
ps.flush(); // Эта команда выполняет выталкивание
            // содержимого буфера вывода
s.close();  // Сокетное соединение закрывается
```

Теперь о реализации сервера. Он написан не как апплет, а как приложение Java с главным методом:

```
public static void main(String args[])
{
 serv f=new serv();
 f.resize(400,400);
 f.show();
 new Account().start();
}
```

Отличие приложений Java от апплетов заключается в том, что браузеры *не работают* с приложениями. Значит, приложение должно как-то иначе использоваться в Интернете. Приложение-сервер может, вообще говоря, использоваться в любой локальной сети. Для Интернета установлены жесткие ограничения на

апплеты: невозможность записи информации в файлы сервера. Попытка создать сокетное соединение через апплет будет пресечена. Для написания серверов в Java на основе апплетов применяется другой подход: использование *сервлетов*. Их изучение является самостоятельной задачей. Итак, запомним: приложения Java работают как автономные программы, не доступные браузерам. Следовательно, весь сетевой интерфейс должен быть разработан пользователем.

Базовым классом нашего приложения является следующий:

```
public class serv extends Frame
{
 public boolean handleEvent(Event evt) // Используется
                                       // обработчик событий
                                       // ранних версий Java
 {
  if (evt.id==Event.WINDOW_DESTROY) // Попытка закрыть окно
                                    // приложения
   {System.exit(0);}
  return super.handleEvent(evt);
 }
 public boolean mouseDown(Event evt,int x,int y)
 // Обработчик события от мыши
 {
  new clientThread().start(); // Запуск клиента
  return(true);
 }

 public static void main(String args[])
 {
  serv f=new serv();
  f.resize(400,400);
  f.show();
  new Account().start();
 }
}
```

Разберемся с этим классом. Главный метод `main()` стартует сразу при запуске приложения. Он создает окно приложения:

```
serv f=new serv();
f.resize(400,400);
f.show();
```

и запускает *поток*-сервера:

```
new Account().start();
```

Переменная `f` имеет тип `serv`, который базируется на оконном типе `Frame`. Наш базовый класс реализует следующие методы. Первый — обработка события закрытия окна:

```
public boolean handleEvent(Event evt)
{
 if (evt.id==Event.WINDOW_DESTROY)
  {System.exit(0);}
 return super.handleEvent(evt);
}
```

Этот метод используется для анализа и локальной обработки событий, объявляемых как `Event evt`. Тип события проверяется командой

```
if (evt.id==Event.WINDOW_DESTROY)
```

т. е. анализируется событие закрытия окна сервера. Заметим, что сервер следует закрыть явно в программе, иначе он останется активным. Команда закрытия сервера:

```
System.exit(0);
```

Второй метод рассматриваемого класса:

```
public boolean mouseDown(Event evt,int x,int y)
{
 new clientThread().start();
 return(true);
}
```

реагирует на щелчок мыши в окне сервера. В предыдущих работах мы использовали для перехвата событий от мыши методы интерфейсов `ActionListener`, `MouseListener`. В этой работе

показан другой вариант, характерный для первых версий Java. По щелчку мыши в окне сервера запускается клиент:

```
new clientThread().start();
```

Клиент моделируется своим собственным потоком. И сервер, и клиент в данной программе существуют как отдельные потоки, функционирующие в одном общем приложении. Поток клиента определен следующим образом:

```
class clientThread extends Thread
{
 DataInputStream dis=null;
 Socket s=null;
 public clientThread()
 {
  try
   {
    s=new Socket("127.0.0.1",2525);
    dis= new DataInputStream(s.getInputStream());
   }
  catch(Exception e)
  {  System.out.println("ERROR:"+e); }
 }
 public void run()
 {
  while (true)
  {
   try
    {sleep(100); }
   catch(Exception er)
    {System.out.println("BDD"+er); }
   try
    {
     String msg=dis.readLine();
     if(msg==null)
      break;
```

```
     System.out.println(msg);
    }
   catch(Exception e)
    {System.out.println("ERRORR+"+e); }
  }
 }
}
```

Инициализация клиента выглядит таким образом:

```
DataInputStream dis=null;
Socket s=null;
public clientThread()
{
 try
  {
   s=new Socket("127.0.0.1",2525);
   dis= new DataInputStream(s.getInputStream());
  }
```

Клиент пытается получить *сокет* `Socket("127.0.0.1",2525)`. Это именно тот же сокет, куда пишет данные сервер. Заметим, что клиент должен указать *сетевой адрес* компьютера, где расположен сервер (этот адрес называется *IP-адресом* и представляет собой последовательность номеров, разделенных точкой, или сетевое имя; в любом случае адрес указывается как строковый аргумент в кавычках). Клиенту нужна объектная переменная `dis` для чтения данных из сокета. Клиент читает данные из сокета в своем методе `run()`:

```
public void run()
{
 while (true)
 {
  try
   {sleep(100); // Клиент выполняет попытку чтения из сокета
                // каждые 100 миллисекунд
   }
```

```
  catch(Exception er)
   {System.out.println("BDD"+er); }
  try
   {
    String msg=dis.readLine(); // Здесь клиент пытается
                               // прочитать строку из сокета
    if(msg==null)
     break;
    System.out.println(msg); // Прочитанная строка выводится
                             // на консоль
   }
```

Теперь можно собрать класс клиента и класс сервера в одно приложение (листинг 2.20).

Листинг 2.20. Клиент и сервер в одном приложении

```
import java.io.*;
import java.net.*;
import java.awt.*;
import java.lang.*;
class clientThread extends Thread
{ //Потоковый класс клиента
 DataInputStream dis=null; // Клиент читает данные из сервера
                           // на базе DataInputStream
 Socket s=null;
 public clientThread()
 {
  try
   {
    s=new Socket("127.0.0.1",2525); // Создаем сокет для
                                    // клиента
    dis= new DataInputStream(s.getInputStream());
    //dis используется для чтения из сокета клиента
   }
  catch(Exception e)
   {  System.out.println("ERROR:"+e);}
```

```
 }
public void run()
 {
  while (true)
  {
   try
    {
     sleep(100); // Клиент подключается к сокету каждые
                 // 100 миллисекунд
    }
   catch(Exception er)
    {System.out.println("BDD"+er); }
   try
    {
     String msg=dis.readLine(); // Чтение сообщения от сервера
     if(msg==null)
      break;
     System.out.println(msg); // Вывод прочитанного сообщения
    }
   catch(Exception e)
    {System.out.println("ERRORR+"+e); }
  }
 }
}
class Account extends Thread
{ // Потоковый класс сервера
 ServerSocket server;
 String amountstring;
 static int amount=200;
 public void run()
 {
  try
   {
    server= new ServerSocket(2525); // Сокет сервера
   }
```

```
  catch(Exception e)
   { System.out.println("ERRSOCK+"+e); }
  while(true)
  {
   Socket s=null;
   try
    {
     s=server.accept(); // Прослушивает подключение к сокету
                        // клиента
    }
   catch(Exception e)
    {System.out.println("ACCEPTER"+e);}
   try
    {
     PrintStream ps=new PrintStream(s.getOutputStream());
     int amountcur=((int)(Math.random()*1000));
     // Определяется случайная величина текущего вклада
     // с учетом знака; отрицательный вклад — это снятие
     // части денег со счета
     if (Math.random()>0.5)
      amount-=amountcur; // Сумма на счете изменяется случайно
     else
      amount+=amountcur;
     Integer x=new Integer(amount);
     amountstring=x.toString();
     ps.println("Account:"+amountstring); // Выводит в сокет
                                          // сообщение для клиента
     ps.flush();
     s.close();
    }
   catch(Exception e)
    {System.out.println("PSERROR"+e); }
  }
 }
}
```

```
public class serv extends Frame
{ // Форма сервера
 public boolean handleEvent(Event evt)
 {
  if (evt.id==Event.WINDOW_DESTROY)
   {System.exit(0);}
  return super.handleEvent(evt);
 }
 public boolean mouseDown(Event evt,int x,int y)
 {
  new clientThread().start(); // Поток клиента порождается
                              // щелчком мыши в окне сервера
  return(true);
 }
 public static void main(String args[])
 {
  serv f=new serv();
  f.resize(400,400);
  f.show();
  new Account().start();
 }
}
```

Замечание

Для запуска потоков сервера и клиента после развертывания приложения следует выполнить щелчок мышью на окне приложения.

Задание

Внимательно ознакомьтесь с работой. Выполните программу, рассмотренную в качестве примера. Создайте свой вариант программы, используя приведенные ниже варианты заданий.

1. Написать приложение для двух клиентов: клиента-мужа и клиента-жены, которые работают с одним счетом. Каждому клиенту соответствует свой поток.

2. Создать приложение "клиент-сервер" с некоторым визуальным интерфейсом, который выполняет запуск клиента по нажатию кнопки и отражает состояние счета в текстовом поле *(см. приведенные далее дополнительные сведения о создании визуального интерфейса для приложений).*
3. Написать приложение "клиент-сервер" как раздельные программы.

Далее помещены дополнительные сведения к настоящему практическому занятию, расширяющие функциональные возможности клиента и сервера. Второе приложение (листинги 2.22 и 2.23) демонстрирует раздельную реализацию клиента и сервера.

Дополнительные сведения

Приведенный далее вариант программы (листинг 2.21) запускает *поток* клиента по нажатию кнопки (это есть единственное отличие от приведенного ранее приложения "клиент-сервер"). Кнопки и другие визуальные элементы добавляются в конструкторе основного класса приложения, который в общем случае имеет в качестве родительского класса `Frame`. Вот пример того, как это делается:

```
public class servcopy extends Frame
{
 Button b=new Button("ActivateClient");
 public servcopy(String title)
  {
   super(title);
   setLayout(null);
   add(b);
   b.setBounds(150,50,100,20);
  }
```

Для прослушивания кнопки в классе `servcopy` помещен стандартный метод:

```
public boolean action(Event evt, Object arg)
 { if (evt.target==b)
     new clientThread().start();
    return(true);
}
```

Впрочем, можно было бы организовать прослушивание кнопки и уже известным нам интерфейсом `ActionListener`. Оператор `if (evt.target==b)` просто выясняет, от какого элемента пришло событие (в нашем примере `b` — это имя кнопки).

Листинг 2.21. Запуск клиента по нажатию кнопки

```
import java.io.*;
import java.net.*;
import java.awt.*;
import java.lang.*;
// Клиент не изменился
class clientThread extends Thread
{
 DataInputStream dis=null;
 Socket s=null;
 public clientThread()
 {
  try
   {
    s=new Socket("127.0.0.1",2525);
    dis= new DataInputStream(s.getInputStream());
   }
  catch(Exception e)
   {  System.out.println("ERROR:"+e);}
 }

 public void run()
 {
  while (true)
  {
   try
    { sleep(100); }
   catch(Exception er)
    {System.out.println("BDD"+er); }
```

```
   try
    {
     String msg=dis.readLine();
     if(msg==null)
      break;
     System.out.println(msg);
    }
   catch(Exception e)
    {System.out.println("ERRORR+"+e); }
  }
 }
}
class Account extends Thread
{
 ServerSocket server;
 String amountstring;
 static  int amount=200;
 public void run()
 {
  try
   { server= new ServerSocket(2525); }
  catch(Exception e)
   { System.out.println("ERRSOCK+"+e); }
  while(true)
  {
   Socket s=null;
   try
    { s=server.accept(); }
   catch(Exception e)
    {System.out.println("ACCEPTER"+e);}
   try
    {
     PrintStream ps=new PrintStream(s.getOutputStream());
```

```
     int amountcur=((int)(Math.random()*1000));
     // Определяется случайная величина текущего вклада
     // с учетом знака; отрицательный вклад — это снятие части
     // денег со счета
     if (Math.random()>0.5)
      amount-=amountcur; // Сумма на счете изменяется случайно
     else
      amount+=amountcur;
     Integer x=new Integer(amount);
     amountstring=x.toString();
     ps.println("Account:"+amountstring);
     ps.flush();
     s.close();
    }
   catch(Exception e)
    {System.out.println("PSERROR"+e); }
  }
 }
}
// Основной класс претерпел изменения в связи с добавлением
// кнопки
public class servcopy extends Frame
{
 Button b=new Button("ActivateClient"); // Добавлена кнопка
                                        // для запуска клиента
 public servcopy(String title)
 {
  super(title);
  setLayout(null);
  add(b);
  b.setBounds(150,50,100,20);
 }
```

```
public boolean handleEvent(Event evt)
 {
  if (evt.id==Event.WINDOW_DESTROY)
   {System.exit(0);}
  return super.handleEvent(evt);
 }

 public boolean action(Event evt, Object arg)
 {
  if (evt.target==b)
   new clientThread().start(); // Клиент запускается по
                               // нажатию кнопки
   return(true);
 }

 public static void main(String args[])
 {
  servcopy f=new servcopy("MyFrame");
  f.resize(400,400);
  f.show();
  new Account().start();
 }
}
```

Второй вариант реализации приложений "клиент-сервер" предусматривает их автономность: клиент реализуется в одном приложении, сервер — в другом.

Приведем пример приложения "клиент-сервер" с раздельными программами для сервера и клиента. Обе программы реализованы как потоки. Поток сервера передает клиенту строку. Поток клиента считывает строку из сокета и отображает ее в текстовом поле. Обратим внимание на то, что сокетное соединение на стороне сервера поддерживается в бесконечном цикле. В потоке клиента сокетное соединение устанавливается всякий раз, как только получено очередное сообщение.

Приложение сервера имеет следующий вид (листинг 2.22).

Листинг 2.22. Отдельное приложение сервера

```
import java.io.*;
import java.net.*;
import java.awt.*;
import java.util.*;
class serverThread extends Thread
{ // Поток сервера
 static ServerSocket server;
 static int n=0;
 serverThread()
 {
  try
   { server=new ServerSocket(2525);}
  catch(IOException e)
   { System.out.println("ERROR:"+e); }
 }

 public void  run() // Главный метод потока сервера
 {
  while(true)
  {
   n=n+1; // n – номер сообщения
   lab4Server.label.setText(""+n); // Номер сообщения заносим
                                   // в текстовое поле
   Socket s=null;
   Try
    {
     s=server.accept(); // Соединяемся с клиентом
     if (s!=null)
```

```
      {
       lab4Server.label.setText("ACCEPTED SOCKET "+n);
       // Подтверждаем соединение
      }
    }
   catch(IOException e)
    { System.out.println("ERROR:"+e); }
   try
    {
     PrintStream ps=new PrintStream(s.getOutputStream());
     int x=(int) (Math.random()*1000);
     lab4Server.label.setText("sended"+ n);
     ps.println("Hello, from server::"+n);// Эту строку
                                          // отсылаем клиенту
     ps.flush();
     s.close(); // Разрываем соединение
                // сообщение на стороне клиента
    }
   catch(IOException e)
    { System.out.println("ERROR:"+e); }
  }
 }
}
public class lab4Server extends Frame // Объявляем главное
                                      // окно сервера
{
 static  Label label=new Label("INFO");
 Button b=new Button("EXIT"); // Кнопка для завершения
                              // приложения сервера
 static int n=0;
 lab4Server()
 { // Конструктор приложения сервера
  add(label);
```

```
  add(b);
  b.setBounds(10,100,100,20);
  label.setBounds(10,60,350,20);
  serverThread sf= new serverThread(); // Объявляем поток
                                       // сервера
  sf.start();// Поток сервера запущен
 }
public boolean action(Event evt,Object ob)
 {
  if (evt.target instanceof Button)
   { System.exit(0);
     return false; }
  return true;
 }
 public static void main(String args[])
 {
  lab4Server f= new lab4Server();
  f.setLayout(null);
  f.resize(500,400);
  f.show();
 }
}
```

На этом приложение сервера завершено. Оно строится отдельно от приложения клиента и компилируется также независимо. Заметим, что программа сервера должна запускаться первой. Приведенная программа сервера мало чем отличается от ранее рассмотренных. Вам предлагается внимательно разобраться в каждом выполняемом действии.

Приложение клиента представлено в листинге 2.23.

Листинг 2.23. Отдельное приложение клиента

```
import java.awt.*;
import java.net.*;
import java.applet.*;
```

```
import java.io.*;

class ClientThread extends Thread // Поток клиента мало
// изменился; в основной класс клиента добавлен метод sh() для
// показа принятой от сервера строки
{
 // Потоковая переменная для чтения строки из сокета:
 DataInputStream dis=null;
 Socket s=null;
 String s1="HELLO FROM CLIENT";
 static int z=0;
 public void run() // Главный метод потока клиента
 {
  while(z<=40)
  {
   z+=1;
   try
    {
     s=new Socket("127.0.0.1",2525); // Клиент подключается к
                                      // серверу
     dis=new DataInputStream(s.getInputStream());
    }
   catch (Exception e)
    {
     s1="Error in SocInet ::"+e;
     lab4Client.sh(s1);
    }  // Обратите внимание на вызов статического метода sh()
       // класса lab4Client
   try
    {
     sleep(1000);
    }
```

```
   catch(InterruptedException e1)
    {
     s1="sleepERROR"+e1;
     lab4Client.sh(s1);
    }
   try
    {
     s1=dis.readLine(); // Клиент читает строку из сокета
     if (s1==null)
      {
       s1="No information transmitted ";
       lab4Client.sh(s1); // Вывод принятой строки в текстовое
                          // поле
       continue;
      }
     lab4Client.sh(s1);
    }
   catch (IOException e)
    {
     s1="ERROR IN IO PROC."+e;
     lab4Client.sh(s1);
    }
  }
 }
}
// Основной класс приложения клиента
public class lab4Client extends Frame
{
 static  Label label=new Label("For Server Info");
 // Переменная label используется в статическом методе sh(),
 // поэтому объявляется как static
 lab4Client() // Конструктор клиента
 {
  super("MYCLIENT");
  add(label);
```

```
  setLayout(null);
  label.setBounds(10,100,250,20);
  ClientThread  myclient = new ClientThread(); // Объявление
                                             // потока клиента
  myclient.start(); // Запуск потока клиента
 }
 public static void main(String args[])
 {
  lab4Client f=new lab4Client();
  f.resize(400,400);
  f.show();
 }
 public static void sh(String msg) // метод sh() объявлен как
                                   // static
 {
  label.setText(msg);
 }
} //end of lab4Client definition
```

Скомпилируйте эти программы по отдельности и запустите их в таком порядке: сначала приложение сервера, затем — приложение клиента.

Контрольные вопросы

1. Для чего используется класс `Socket`?
2. С помощью какой команды сервер соединяется с клиентом?
3. С помощью какого класса осуществляется вывод информации в сокет?
4. Объясните программу листинга 2.21.
5. Объясните программу листинга 2.22.
6. Объясните программу листинга 2.23.
7. Как выполняется разрыв соединения между клиентом и сервером?
8. Из каких соображений выбирается номер порта?

Доступ к серверной базе данных из клиента

Цель занятия

Изучение механизма доступа к *базе данных*, расположенной на сервере, из клиента. Реализация передачи и распаковки SQL-запросов. Получение результата выполнения SQL-запроса и передача результата на сторону клиента. Организация работы в сети. По данной проблеме можно также рекомендовать [1, 10, 11, 17].

Краткие теоретические сведения

Настоящее практическое занятие объединяет знания, полученные на практических занятиях, посвященных работе с базами данных, потоками и технологии "клиент-сервер". Хорошее усвоение материала этих занятий позволит вам без особого труда разобраться с материалом этого занятия.

Рассмотрим механизмы доступа к расположенной на сервере базе данных из клиента. Технология "клиент-сервер" применяется для централизованного доступа из удаленных ЭВМ к общей базе данных, расположенной на сервере. В языке Java механизм клиент-серверного взаимодействия основан на использовании потоков и сокетных соединений. Для клиента надлежит создать два потока: один поток для работы на ввод, второй — для работы на вывод. Аналогичная ситуация с сервером. В качестве языка запросов используем язык SQL.

Клиент должен обращаться в базу с сформированным SQL-запросом. Сервер должен выполнять запрос и возвращать ответ. Окно программы сервера предствлено на рис. 2.15.

Запуск сервера на выполнение реализует кнопка **Start Server**. По этой кнопке запускаются два потока. Один поток работает на ввод данных от клиента, второй — на вывод. Выход из сервера реализуется по кнопке **Exit**.

Рис. 2.15. Окно приложения сервера

Текст потока, работающего на вывод, помещен далее (листинг 2.24). Этот поток выдает на сторону клиента результат выборки из базы данных по принятому SQL-запросу. Факт формирования результата выборки отслеживается с помощью переменной `act`. Строка с результатом выборки (`ser2`) формируется в другом потоке сервера, работающем на ввод. Объявление сокета и подключение к нему должны быть ясны из листинга 2*.24 (см. также главу 2).*

Листинг 2.24. Поток сервера server2, работающий на вывод

```
class server2 extends Thread
{
 ServerSocket serverr;
 Socket outserver;
 static String ser2="accepted from 2526"; // Порт 2526
                                          // назначен здесь
                                          // для вывода из сервера
 PrintStream  serverprint; // Объявляем потоковый класс для
                           // вывода
 int nu=1;
```

```
public void run()
{
 yield(); // Эта команда позволяет пропустить вперед
          // выполнение более приоритетных потоков
 try
  {
   serverr=new ServerSocket(2526); // Назначаем сокет для
                                   // вывода
  }
 catch(Exception err)
  {ServerBD.lb.setText("B2ErrorSocket:"+err);}
 while(ServerBD.act==false)
 // статическая переменная act определяет (act=true) наличие
 // сформированного результата выборки по команде SQL
 {
  try
   {
    yield();
    ServerBD.lb.setText("Is waiting:"+nu);
    sleep(1000);
    nu+=1;}
    catch(Exception err)
     {ServerBD.lb.setText("H2ErrorSleep:"+err);}
   }
  ServerBD.lb.setText("Sending answer to client");
  yield();
  while(true)
  {
   try
   {
    // С этого места реализуется вывод информации из сервера
    // через сокет
    outserver=serverr.accept(); // Ожидание подключения
                                // клиента
    yield();
```

```
    if (outserver !=null)
     {
      serverprint=new
PrintStream(outserver.getOutputStream());
      serverprint.println(ser2); // Выводится на сторону
                                 // клиента строка
      ServerBD.act=false; // Вывод завершен. Сервер ожидает
      // новый сформированный результат для вывода
     }
    }
   catch(Exception err)
    {
     ServerBD.lb.setText("A2ConnectError:"+err);
    }
   yield();
  }
 }
}
```

Из сервера передается обратно клиенту строка с именем `ser2`, формируемая в первом потоке. Поток сервера, принимающий SQL-запрос от клиента, выполняется в классе `public class ServerBD extends Frame implements Runnable`. Рассмотрим поток сервера, работающий на ввод, а начнем с его основного метода `run()`.

```
public void run()
{
 lb.setText("Server is launched");
 try
  {
   server=new ServerSocket(2525); // Создается новое гнездо
                                  // для прослушивания
  }
 catch(Exception err)
  { lb.setText("BErrorSocket:"+err);}
```

```
try
 {
  db=DriverManager.getConnection(bdurl); // Получаем
  // соединение с базой данных, определенной через
  // переменную String bdurl="jdbc:odbc:vfp"
 }
catch(Exception err1)
 { lb.setText("CC_DBconnectErr:"+err1);}
try
 { sqlst=db.createStatement(); }// Создаем команду SQL
catch(Exception err2)
 {lb.setText("FError:"+err2);}
while(true)
{
 try
  {
   inserver=server.accept(); // Подключаемся к клиенту
   if (inserver !=null)
    {
     // Создаем потоковую переменную для ввода из сокета:
     serverinput=new
     DataInputStream(inserver.getInputStream());
     lb.setText(serverinput.readLine());
     // Принятый SQL-запрос отображаем в текстовом поле
     // с именем lb
    }
  }
 catch(Exception err)
  { lb.setText("A Connection Error:"+err); continue;}
 tar.setText(lb.getText());
 String str_sql=lb.getText(); // В переменной str_sql
                              // помещаем текст SQL-запроса
 int priz=4; // Переменная priz определяет
            // распознанный тип SQL-запроса
```

```
  String
sword=((str_sql.trim()).substring(0,6)).toUpperCase();
  // Выделяем подстроку строки SQL запроса длиной 7 символов,
  // имея в виду, что SQL-команда одна из: SELECT, CREATE,
  // UPDATE, INSERT (6 символов + 1 символ пробела)
  if (swork.equals("SELECT"))
   { priz=1;}
  else // Устанавливаем переменную priz в зависимости
       // от принятой SQL-команды
   if (swork.equals("CREATE"))
       {priz=2;}
      else priz=3;
  try
   {
    // Переменная priz определяет, какая SQL-команда получена
    // (см. по тексту)
    switch(priz)
    {
     case 1:
     // Если команда SELECT, то она выполняется так:
     rs= sqlst.executeQuery(str_sql);
     lb.setText("1:SELECT");
     // processing the results of selection
     //обработка результатов выборки:
     tar.setText("");
     int colsnumber=rs.getMetaData().getColumnCount();
     / Определяем число выбранных столбцов в результирующем
     // наборе;
     String ss=""; // Переменная ss служит для формирования
                   // записей для отправки клиенту
     while(rs.next()) // в rs — очередная запись из базы
                      // данных
```

```
{
 tar.append("\n");
 for (i=1 ;i<=colsnumber;i++)
 {
  // Определяем тип столбца из rs:
  int coltype=rs.getMetaData().getColumnType(i);
  if (coltype==1) // Тип 1 соответствует строковому
   {
    ss=ss+" "+rs.getString(i); // Читаем содержимое i-го
                               // столбца строкового типа
   }
  else
   if (coltype==2) // Тип 2 соответствует целому числу
    { ss=ss+" "+rs.getInt(i); }
 }
 tar.append(ss); // Считанные значения добавляем в
                 // текстовую область с именем tar
 server2.ser2=ss; // Этим оператором формируется строка
 // ser2, передаваемая обратно клиенту из потока сервера,
 // работающего на вывод
 ss="";
}
rs.close();
lb.setText(server2.ser2);
act=true;
Thread.yield();
break;
case 2:
sqlst.execute(str_sql); // Команда execute() выполняется
// при создании таблицы по SQL-запросу CREATE TABLE
lb.setText("2:CREATE");
break;
case 3:
sqlst.executeUpdate(str_sql); // Команда на обновление
                              // базы данных
```

```
     lb.setText("3:INSERT/UPDATE");
     break;
     default:
     tar.setText("NONO");
     break;
    }
   }
  catch(Exception err1)
   {
    lb.setText("CH_DBconnectErr:"+err1);
    tar.setText(str_sql);
   }
 }
}
```

Далее (листинг 2.25) приводится полный текст приложения сервера, снабженный необходимыми комментариями.

Листинг 2.25. Полный текст приложения сервера

```
import java.awt.*;
import java.io.*;
import java.sql.*;
import java.net.*;
//Серверная часть, работающая на вывод, была разобрана ранее.
class server2 extends Thread
{
 ServerSocket serverr;
 Socket outserver; // Сокет для вывода результата обработки
                   // запроса
 static String ser2="accepted from 2526";
 PrintStream  serverprint; // Потоковая переменная для вывода
                           // в сокет
 int nu=1;
 public void run()  // Главный метод потока
```

```
{
 yield();
 try
  {
   serverr=new ServerSocket(2526); // Для вывода используем
                                   // порт с номером 2526
  }
 catch(Exception err)
  { ServerBD.lb.setText("B2ErrorSocket:"+err);}
 while(ServerBD.act==false) // "холостой ход", пока
                            // act==false
 {
  try
   {
    yield();
    ServerBD.lb.setText("Is waiting:"+nu);
    sleep(1000);
    nu+=1;
   }
  catch(Exception err)
   {ServerBD.lb.setText("H2ErrorSleep:"+err);}
 }
 ServerBD.lb.setText("Sending answer to client");
 yield();
 while(true)
 {
  try
   {
    outserver=serverr.accept(); // Ждем подключения клиента
                                // для отсылки ему строки
    yield();
    if (outserver !=null)
     { serverprint=new
     PrintStream(outserver.getOutputStream());
```

```
      serverprint.println(ser2); // Отсылка строки ser2
                                 // клиенту
      ServerBD.act=false;
     }
    }
   catch(Exception err)
    {ServerBD.lb.setText("A2ConnectError:"+err);}
   yield();
  }
 }
}
// Класс сервера, содержащий поток, работающий на ввод
public class  ServerBD extends Frame implements Runnable
{
 int i;
 static boolean act=false; // Переменная act=true, если
                           // сформирован результат выборки
 Thread serverthread; // Объявление потока сервера,
                      // работающего на ввод
 ServerSocket server;
 String bdurl="jdbc:odbc:vfp"; // Строка для установления
                               // соединения
                               // с базой данных
 Socket inserver; // Сокет для ввода SQL-запроса от клиента
 DataInputStream serverinput; // Объявление потоковой
                              // переменной для чтения данных
 Button btn_exit=new Button("Exit Server"); // Кнопка для
                                            // завершения
                                       // приложения сервера
 Button btn_launch=new Button("StartServer"); // Кнопка для
                                              // запуска
                                          // потоков сервера
 static Label  lb=new Label("Server is Sleeping");
```

```
static TextArea tar=new TextArea(); // Текстовая область
// для размещения результатов выборки
Connection db; // Переменная соединения
Statement sqlst; // Переменная для SQL-запроса
ResultSet rs; // Результирующий набор команды Select

ServerBD() // Конструктор, формирующий окно приложения
           // сервера
{
 add(btn_exit);
 add(lb);
 add(btn_launch);
 add(tar);
 setLayout(null);
 setBackground(new Color(50,20,150));
 lb.setForeground(Color.yellow);
 btn_exit.setBounds(20,20,100,20);
 btn_launch.setBounds(130,20,100,20);
 lb.setBounds(20,60,450,20);
 tar.setBounds(20,80,450,200);
 try
  {
   // Подключение интерфейса JDBC-ODBC:
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  }
 catch(Exception err)
  {lb.setText("ClassForNameErr"+err);}
}
public boolean action(Event evt, Object arg)
// Обработка событий от программных кнопок
{
 if (evt.target==btn_exit)
  { System.exit(0); return true; }
 else
```

```
   if (evt.target==btn_launch) // Событие от кнопки для
                               // запуска потоков
    {
     try
      {
       serverthread= new Thread(this); // Создаем поток
                                       // сервера для ввода
       serverthread.start(); // Запускаем поток сервера для
                             // ввода
       server2 serv2 =new server2(); // Создаем поток сервера
                                     // для вывода
       serv2.start(); // Запускаем поток сервера для вывода
       lb.setText("Server Thread is Activated");
      }
     catch(Exception err)
      {lb.setText("ERROR in Thread:"+err);}
     return true;
    }
   else
    { return false; }
 }
 public void run() // Главный метод потока сервера для ввода
 {
  lb.setText("Server is launched");
  try
   {
    server=new ServerSocket(2525); // Создаем сокет для ввода
                                   // SQL-запроса на базе
                                   // порта 2525
   }
  catch(Exception err)
   {lb.setText("BErrorSocket:"+err);}
  try
```

```
 {
  db=DriverManager.getConnection(bdurl); // Попытка
  // установить соединение с базой данных для выполнения
  // принятого SQL-запроса
 }
catch(Exception err1)
 { lb.setText("CC_DBconnectErr:"+err1);}
try
 {
  sqlst=db.createStatement();//Создаем команду SQL
 }
catch(Exception err2)
 { lb.setText("FError:"+err2); }
while(true)
{
 try
  {
   inserver=server.accept(); // Ожидаем передачи от клиента
                             // SQL-запроса
   if (inserver !=null)
    {
     serverinput=new
     DataInputStream(inserver.getInputStream());
     lb.setText(serverinput.readLine()); // Отображаем
                                         // принятую строку
                                         // запроса
                                     // в текстовом поле lb
    }
  }
 catch(Exception err)
  {lb.setText("A ConnectError:"+err); continue;}
 tar.setText(lb.getText());
 String str_sql=lb.getText(); // В переменную str_sql
                              // помещаем текст SQL-запроса
```

```
int priz=4;
String
swork=((str_sql.trim()).substring(0,6)).toUpperCase();
// Выделяем подстроку swork, содержащую код SQL-команды,
// из строки запроса.
if (swork.equals("SELECT"))
 { // Проверяем код команды и, соответственно,
   // формируем переменную priz
   priz=1;
 }
else
 if (swork.equals("CREATE")) {priz=2;}
 else priz=3;
try
 {
  switch(priz)
  {
   case 1:
   rs= sqlst.executeQuery(str_sql); // Формируем
                                    // результирующий
                                    // набор для команды
SELECT
   lb.setText("1:SELECT");
   // Обработка результирующего набора
   tar.setText("");
   int colsnumber=rs.getMetaData().getColumnCount();
   // В colsnumber заносим число столбцов набора
    String ss=""; // В строку ss заносим содержимое полей
                  // текущей записи набора rs
   while(rs.next()) // Обработка записей набора rs
   {
    tar.append("\n"); // Записи отделяются символом
                      // конца строки //"\n"
    for (i=1 ;i<=colsnumber;i++)
     {
```

```
    int coltype=rs.getMetaData().getColumnType(i);
    // В coltype — тип столбца
    // Тип столбца таблицы определяет способ чтения
    // его содержимого
    if (coltype==1) // Столбец имеет строковый тип
     {
      ss=ss+" "+rs.getString(i); // В ss записываем
                                 // строковое
                                 // значение из столбца
     }
    else
     if (coltype==2) // Столбец имеет целочисленный тип
      {
       ss=ss+" "+rs.getInt(i); // В ss записываем целое
                               // значение из столбца
      }
    }
   tar.append(ss);
   server2.ser2=ss; // Переменная ss с занесенным в нее
                    // набором rs передается обратно
                    // на сторону клиента
   ss="";
 }
 rs.close();
 lb.setText(server2.ser2);
 act=true;
 Thread.yield();
 break;
 case 2: // Команда SQL есть CREATE TABLE
 sqlst.execute(str_sql);//Выполнение команды CREATE
 lb.setText("2:CREATE");
 break;
```

```
      case 3: // Остальные команды SQL проходят в этом
              // варианте оператора case
      sqlst.executeUpdate(str_sql); // Выполнение команд,
                                    // отличных от
                                    // SELECT, CREATE
      lb.setText("3:INSERT/UPDATE");
      break;
      default:
      tar.setText("No valid SQL code");
      break;
     }
    }
   catch(Exception err1)
    {
     lb.setText("CH_DBconnectErr:"+err1);
     tar.setText(str_sql);
    }
  }
 }

 public static void main(String args[])
 {
  ServerBD sbd=new ServerBD();
  sbd.resize(500,360);
  sbd.show();
 }
}
```

Рассмотрение серверной части закончено. В приложении к этому занятию мы рассмотрим кратко SQL-запросы. Переходим к стороне клиента, окно которого прдставлно на рис. 2.16.

Кнопки в окне клиента позволяют вставить в окно редактирования каркас SQL-запроса и затем сформировать его целиком. Результат запроса инициируется кнопкой `EXECUTE QUERY`. Пример представления результата дает следующее окно (рис. 2.17).

Рис. 2.16. Окно клиента

Рис. 2.17. Пример результата SQL-запроса

Аналогично серверу, клиентская программа реализует два потока. Первый поток служит для передачи SQL-запроса в сервер. Второй поток принимает результат работы сервера. Поток клиента для передачи запроса серверу имеет такой вид:

```
// Поток клиента на вывод — передача серверу SQL-запроса
public void run() // Главный метод потока
{
 try
  {
   clienttransmit=new Socket("127.0.0.1",2525); // Эта команда
   // и обеспечивает сокетное соединение с сервером
   // Получаем выходной поток для клиента:
   clientprint=new
   PrintStream(clienttransmit.getOutputStream());
   clientprint.println(str_query); // Передаем SQL-запрос
                                   // серверу
```

```
    lbl.setText("Query is sent To Server");
    clientprint.flush();
    clienttransmit.close(); // Закрываем сокетное соединение
                            // со стороны клиента
    client2= new Client2BD();
    client2.start(); // Запускаем поток клиента на ввод
                     // результата от сервера
   }
  catch(Exception err)
  {
   lbl.setText("The Error in connection:"+err);
  }
}
```

Теперь приведем полный текст программы клиента с комментариями (листинг 2.26).

Листинг 2.26. Полный текст приложения клиента

```
import java.awt.*;
import java.lang.*;
import java.net.*;
import java.io.*;
class DialogBD extends Frame  // Здесь создается диалоговое
// окно клиента для подтверждения выбранного действия
// на стороне клиента
{
 Button btn_ok=new Button("Click if YES");
 Button btn_no=new Button("Click if NO");
 Label  lb=new Label("CONFIRM THE SOLUTION");
 Event dialogevent;
 ClientBD dialogBD; // Объявляется класс диалогового окна,
                    // на котором будет выведен запрос для
                    // подтверждения действия
```

```
DialogBD(Event evt, ClientBD cldbcopy)
{
 super("DESICIONDIALOG");
 dialogevent=evt;
 dialogBD=cldbcopy;
 Font fnt=new Font("Courier",Font.BOLD,18);
 lb.setFont(fnt);
 lb.setForeground(new Color(100,20,250));
 add(btn_ok);
 add(btn_no);
 add(lb);
 setLayout(null);
 btn_ok.setBounds(20,60,100,20);
 btn_no.setBounds(120,60,100,20);
 lb.setBounds(10,15,250,40);
}

public void spaint()
{
 setBackground(new Color(77,150,120));
}
public boolean action(Event evt, Object arg)
{
 if (evt.target==btn_ok)
  {
   ClientBD.Solution=1; // Переменная Solution=1, если выйти
                        // из приложения клиента
   dispose(); // Команда закрытия окна диалога
   dialogBD.procaction(dialogevent);
   return true;
  }
 else
  if (evt.target==btn_no)
```

```
    {
     ClientBD.Solution=0;
     dispose(); // Команда закрытия окна диалога
     dialogBD.procaction(dialogevent);
     return true;
    }
   else return false;
 }
}

class Client2BD extends Thread
{
 Socket clientreceiver;
 // Поток клиента для ввода данных — результата SQL-запроса к
 // серверу
 DataInputStream clientinput=null;
 public void run()
 {
  try
   {
    clientreceiver=new Socket("127.0.0.1",2526); // Создаем
                                                 // сокет
                                                 // для ввода
    // Потоковая переменная для ввода:
    clientinput=new
    DataInputStream(clientreceiver.getInputStream());
    String ssr=clientinput.readLine(); // Чтение результата,
                          // возвращенного сервером, из сокета
    ClientBD.lbl.setText("Answer from Server");
    ClientBD.ta.setText(ssr); // Вывод ответа сервера
                              // в текстовое окно
   }
  catch(Exception err)
  { ClientBD.lbl.setText("The Error in connection:"+err); }
 }
}
```

```
// Основное окно клиента
public class ClientBD extends Frame implements Runnable
{
 Socket clienttransmit; // Сокет для вывода
 PrintStream clientprint=null; // Потоковый класс для вывода
 String str_query=""; // Строка запроса
 String sss;
 static int Solution=2;
 ClientBD cldbcopy;
 Client2BD client2;
 // Все эти элементы нетрудно увидеть в окне клиента
 Button btn_exec=new Button("EXECUTE QUERY"); // Основная
 // кнопка для посылки запроса на сервер
 Button btn_create=new Button("CREATE TABLE"); // Формирует
                                               // каркас
                                               // команды
CREATE
 Button btn_delete=new Button("DELETE"); // Формирует каркас
                                         // команды DELETE
 Button btn_write=new Button("WRITE RECORD");
 Button btn_update=new Button("UPDATE"); // Формирует каркас
                                         // команды UPDATE
 Button btn_find=new Button("SELECT"); // Формирует каркас
                                       // команды SELECT
 Button btn_clear=new Button("CLEAR"); // Очищает текстовую
                                       // область
 Button btn_exit=new Button("EXIT"); // Запуск диалога
                                     // на выход из приложения
                                     // клиента
 static TextArea ta= new TextArea();
 Thread clientthread; // Поток клиента на ввод
 static Label lbl=new Label("");

 public ClientBD() // Конструктор клиента инициализирует
                   // интерфейсную часть
```

```
{
 super("JAVACLIENTINTERFACE");
 cldbcopy=this;
 setLayout(null);
 add(btn_exec);
 add(btn_create);
 add(btn_delete);
 add(btn_write);
 add(btn_update);
 add(btn_find);
 add(btn_clear);
 add(btn_exit);
 add(lbl);
 add(ta);
 btn_exec.setBounds(0,20,600,20);
 btn_create.setBounds(0,40,100,20);
 btn_delete.setBounds(100,40,100,20);
 btn_write.setBounds(200,40,100,20);
 btn_update.setBounds(300,40,100,20);
 btn_find.setBounds(400,40,100,20);
 btn_clear.setBounds(500,40,100,20);
 btn_exit.setBounds(0,60,100,20);
 lbl.setBounds(100,60,520,20);
 ta.setBounds(0,80,600,320);
 ta.setEditable(true);
}
// Программирование реакции на нажатие кнопок
public boolean action(Event evt,Object arg)
{
 if (evt.target instanceof Button) // Независимо от того,
 // какая кнопка нажимается, запускается окно диалога –
 // объект класса DialogBD
  {
   DialogBD dlg=new DialogBD(evt, cldbcopy);
```

```
    // Конструктору DialogBD передается объект-событиеevt и
    // объект-копия
    // класса ClientBD
    dlg.resize(300,100); // Задание параметров диалогового
                         // окна
    dlg.spaint(); // Установка фонового цвета диалогового окна
    dlg.move(200,200); // Изменение позиции на экране
                       // диалогового окна
    dlg.show(); // Отображение диалогового окна
    return true;
   }
  return false;
 }
 // После закрытия окна диалога, если нет завершения клиента,
 // запускается метод procaction() для продолжения обработки
 // события evt в соответствии с принятым решением Solution
 public boolean procaction(Event evt)
 {
  // Solution==1 означает, что действие в диалоге подтверждено
  if((evt.target == btn_exit)&&(Solution==1))
   {
    System.exit(0); // Завершение клиента
    return true;
   }
  else
   if ((evt.target==btn_exec)&&(Solution==1))
    {
     str_query=ta.getText();
     if (clientthread ==null)
      {
       // Создание потока клиента на вывод
       clientthread= new Thread(this);
       clientthread.start(); // Запускается поток клиента на
                       // вывод текста SQL-запроса
                       // на сервер
```

```
        lbl.setText("QUERY is ACCEPTED");
       }
      else
       {
        lbl.setText("Cannot run At the moment");
        // Предупреждающее сообщение; Требуется повторить
        clientthread=null;
       }
      Solution=2;
      return true;
     }
    else
     if((evt.target==btn_create)&&(Solution==1)) // Выводится
                      // шаблон (каркас) команды CREATE TABLE
      {
       String str_t1=ta.getText();
       int len_str=str_t1.length();
       ta.replaceText("",0,len_str);
       String str_create="CREATE TABLE ?TABLENAME(?PAR1
       char(50), "+
       "?PAR2 char(50))\n"+"//insert real values instead of
       ??";
       // Строка str_create содержит каркас команды CREATE
       // TABLE
       ta.appendText(str_create);
       Solution=2;
       return true;
      }
     else
      if ((evt.target== btn_delete)&&(Solution==1))
       {
        String str_t1=ta.getText();
        int len_str=str_t1.length();
        ta.replaceText("",0,len_str);
```

```
       String str_delete="DELETE WHERE ?CONDITION?n"+
       "// insert a CONDITION instead of ??"; // Шаблон
        // команды DELETE
       ta.appendText(str_delete);
       Solution=2;
       return true;
      }
     else
      if ((evt.target==btn_write)&&(Solution==1))
       {
        String str_t1=ta.getText();
        int len_str=str_t1.length();
        ta.replaceText("",0,len_str); // Выводится шаблон
                                      // команды INSERT
        String str_insert="INSERT INTO ?TABLE
        VALUES(?PAR1,?PAR2,...)\n"+
         "// insert real values instead of ??";
        ta.appendText(str_insert);
        Solution=2;
        return true;
       }
      else
       if ((evt.target==btn_update)&&(Solution==1))
        {
         String str_t1=ta.getText();
         int len_str=str_t1.length();
         ta.replaceText("",0,len_str); // Выводится шаблон
                                       // команды UPDATE
         String str_update="UPDATE ?TABLE SET ?PAR=?VAL WHERE
         ?CONDIT\n"+
         "//insert NAME  and VALUE of PAR and set a CONDITION
         ??";
         ta.appendText(str_update);
         Solution=2;
         return true;
        }
```

```
      else
       if ((evt.target==btn_find)&&(Solution==1))
        {
         String str_t1=ta.getText();
         int len_str=str_t1.length();
         ta.replaceText("",0,len_str); // Выводится шаблон
                                       // команды SELECT
         String str_select="SELECT * FROM ?TABLE WHERE
         ?CONDITION))\n"+
         "// set Table Name and CONDITION marked with ??";
         ta.appendText(str_select);
         Solution=2;
         return true;
        }
       else
        if ((evt.target==btn_clear)&&(Solution==1))
         {
          String str_t1=ta.getText();
          int len_str=str_t1.length();
          ta.replaceText("",0,len_str);
          Solution=2;
          return true;
         }
        else
         {
          Solution=2;
          return false;
         }
}
public void run()
{
 try
  {
   // Создается сокетное соединение на вывод SQL-запроса на
   // сервер
```

```
    clienttransmit=new Socket("127.0.0.1",2525);
    clientprint=new
    PrintStream(clienttransmit.getOutputStream());
    clientprint.println(str_query); // Вывод запроса
  // реализуется оберточным классом PrintStream,
  // осуществляющим вывод строк
    lbl.setText("Query is sent To Server");
    clientprint.flush();
    clienttransmit.close();
    client2= new Client2BD();
    client2.start(); // Запускается поток клиента
                     // на ввод ответа от сервера
   }
  catch(Exception err)
  { lbl.setText("The Error in connection:"+err); }
 }
 public static void main(String args[])
 {
  ClientBD cldb=new ClientBD();
  cldb.resize(600,400);
  cldb.show();
 }
}
```

Отдельный класс используется для создания диалогового окна, в котором выдается запрос на подтверждение действия:

```
class DialogBD extends Frame
{
 Button btn_ok=new Button("Click if YES"); // Кнопка для
                              // подтверждения действия
 Button btn_no=new Button("Click if NO"); // Кнопка для
                                    // отмены действия
 Label  lb=new Label("CONFIRM THE SOLUTION");
 Event dialogevent; // Событие dialogevent держит копию
                    // события evt основного окна клиента
```

```
ClientBD dialogBD;
DialogBD(Event evt, ClientBD cldbcopy)
{
 super("DESICIONDIALOG");
 dialogevent=evt; // dialogevent и evt — одно и то же событие
 dialogBD=cldbcopy; // Переменная dialogBD относится к классу
                    // ClientBD
 Font fnt=new Font("Courier",Font.BOLD,18);
 lb.setFont(fnt);
 lb.setForeground(new Color(100,20,250));
 add(btn_ok);
 add(btn_no);
 add(lb);
 setLayout(null);
 btn_ok.setBounds(20,60,100,20);
 btn_no.setBounds(120,60,100,20);
 lb.setBounds(10,15,250,40);
}
public void spaint()
{
 setBackground(new Color(77,150,120));
}
public boolean action(Event evt, Object arg)
{
 if (evt.target==btn_ok)
  {
   ClientBD.Solution=1; // Статическая переменная Solution
   // определяет выбранное действие в диалоге и
   // обрабатывается в основном классе клиента в методе
   // procaction()
   dispose();
   dialogBD.procaction(dialogevent); // Метод procaction()
   // обрабатывает событие после закрытия диалога
```

```
    return true;
   }
  else
   if (evt.target==btn_no)
    {
     ClientBD.Solution=0;
     dispose();
     dialogBD.procaction(dialogevent);
     return true;
    }
   else
    return false;
 }
}
```

Задание

1. Создайте свою базу данных, например в Access.
2. Создайте соединение для этой базы данных, как это мы делали на практическом занятии, посвященном работе с базами данных.
3. Измените с учетом вашей базы данных и вашего соединения текст сервера и клиента.
4. Скомпилируйте клиента и сервер и запустите сначала сервер, затем клиента.
5. Выполните следующие команды SQL:
   - `CREATE TABLE stud(name char(30), group int)`

   Эта команда служит для создания таблицы с именем `stud` с полями `name` и `group`. Каждое поле имеет собственный тип данных. Поле `name` объявлено как строковое с типом `char`. В скобках указывается максимальная длина записи. Поле `group` объявлено как целочисленное с типом `int`.

   - `INSERT INTO stud values("Petrov",12);`

   Добавляет в таблицу новую запись `<"Petrov",12>`.

- `SELECT * FROM stud WHERE group=12`

Данная команда выбирает все записи из таблицы `stud`, где `group=12`. Символ `*` указывает, что подлежат выборке значения всех полей таблицы. Если следует выбрать только имя, то запрос следует переписать так:

`SELECT name FROM stud WHERE group=12.`

- `UPDATE stud set name="Petrikov" WHERE name="Petrov"`

Данная команда везде, где `name="Petrov"`, запишет `name="Petrikov"`.

- `DELETE FROM STUD WHERE name="Petrov"`

Удаляет из таблицы все записи, где `name="Petrov"`.

Контрольные вопросы

1. Какие потоки должны обеспечивать функциональность клиента и сервера? Найдите их описание в листингах этого занятия.
2. Как в приведенных программах обеспечивается синхронизация потоков, работающих на ввод и вывод?
3. Возвращает ли сервер клиенту в качестве ответа одну строку или много строк?
4. Объясните смысл диалогового окна `DialogBD`. Как выполняется передача аргументов в конструктор класса диалогового окна?
5. Используют ли потоки сервера и клиента общие порты (сокеты)?
6. Объясните способ формирования строки ответа на стороне сервера.
7. Подумайте, как реализовать доступ к базе данных сервера по паролю.
8. Может ли сервер обслуживать более одного клиента, как это реализовать?
9. Должен ли сервер быть все время активен?

10. Объясните работу потока сервера, работающего на ввод по листингу 2.25.
11. Объясните работу потока сервера, работающего на вывод по листингу 2.25.
12. Объясните работу потока клиента, работающего на ввод по листингу 2.26.
13. Объясните работу потока клиента, работающего на вывод по листингу 2.26.

Использование Java Beans в других средах

Цель занятия

Целью настоящего занятия является демонстрация возможностей применения технологии Java Beans. Предполагается решить следующие задачи: изучить способы создания компонентов *ActiveX* из бинов (*bean* — боб), а также рассмотреть их включение в приложения, созданные на других платформах. Требуется построить бин на базе собственного класса пользователя. В качестве дополнительных источников информации отметим [2, 5, 13].

Краткие теоретические сведения

Прежде всего следует научиться создавать компоненты ActiveX из *бинов*. Под ActiveX понимается объект, созданный в одной среде программирования и используемый в других средах. В Java имеется утилита `sun.beans.ole.Packager`, которая выполняет требуемую задачу. Следовательно, чтобы успешно выполнить работу, нужно иметь уже созданные бины. В настоящей работе воспользуемся архивом бинов, поставляемым вместе с Java.

Имейте в виду, что бины предназначены для вставки объектов Java в чужеродные приложения, например, приложения Visual

Basic или HTML. В этой работе мы как раз и внедрим бины в HTML-документы.

Итак, по порядку.

Прежде всего, нужно из имеющегося бина создать объект ActiveX. Запустите из командной строки следующее приложение:

```
>java -cp rt.jar; jaws.jar sun.beans.ole.Packager
```

Предварительно убедитесь, что файлы rt.jar, jaws.jar находятся в текущем каталоге (откуда запускается программа java.exe). В противном случае либо скопируйте их в текущий каталог, либо пропишите полный путь к этим файлам. (Вообще говоря, можно не указывать rt.jar — бин создается и без него.)

В результате должно быть запущено приложение с таким окном, как показано на рис. 2.18.

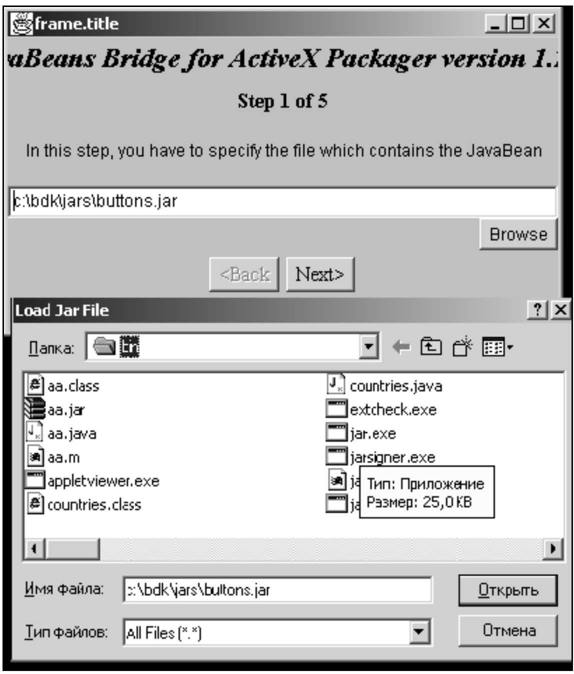


Рис. 2.18. Окно приложения sun.beans.ole.Packager

Нажмите кнопку **Browse** и выберите нужный бин. Эти бины следует искать в подкаталоге lib/dt. Выбрав этот подкаталог, нажмите кнопку **Next**. Затем в списке бинов выберите `JtextField` (например) и снова нажмите **Next**. На приглашение зарегистрировать компонент дайте подтверждение и укажите, куда поместить регистрационный файл. Регистрационный файл имеет расширение reg. Этот текстовый файл можно просмотреть. Вот каково его содержимое (листинг 2.27).

Листинг 2.27. Содержимое файла регистрации бина

```
REGEDIT4
[HKEY_CLASSES_ROOT\JTextField.Bean]
@= "JTextField Bean Control"
[HKEY_CLASSES_ROOT\JTextField.Bean\CLSID]
@= "{F84A5040-0873-11D7-8D09-444553540001}"
[HKEY_CLASSES_ROOT\JTextField.Bean\CurVer]
@= "1"
[HKEY_CLASSES_ROOT\JTextField.Bean.1]
@= "JTextField Bean Control"
[HKEY_CLASSES_ROOT\JTextField.Bean.1\Insertable]
[HKEY_CLASSES_ROOT\JTextField.Bean.1\CLSID]
@= "{F84A5040-0873-11D7-8D09-444553540001}"
[HKEY_CLASSES_ROOT\CLSID\{F84A5040-0873-11D7-8D09-
444553540001}]
@= "JTextField Bean Control"
[HKEY_CLASSES_ROOT\CLSID\{F84A5040-0873-11D7-8D09-
444553540001}\InprocServer32]
@= "C:\\Program Files\\JavaSoft\\JRE\\1.2\\bin\\beans.ocx"
"ThreadingModel" = "Apartment"
[HKEY_CLASSES_ROOT\CLSID\{F84A5040-0873-11D7-8D09-
444553540001}\ToolboxBitmap32]
@= "C:\\Program Files\\JavaSoft\\JRE\\1.2\\bin\\beans.ocx,0"
[HKEY_CLASSES_ROOT\CLSID\{F84A5040-0873-11D7-8D09-
444553540001}\TypeLib]
@= "{F84A5041-0873-11D7-8D09-444553540001}"
```

```
[HKEY_CLASSES_ROOT\CLSID\{F84A5040-0873-11D7-8D09-
444553540001}\ProgID]
@= "JTextField.Bean.1"
[HKEY_CLASSES_ROOT\CLSID\{F84A5040-0873-11D7-8D09-
444553540001}\VersionIndependentProgID]
@= "JTextField.Bean"
[HKEY_CLASSES_ROOT\CLSID\{F84A5040-0873-11D7-8D09-
444553540001}\JarFileName]
@= "C:\\Program Files\\bin\\servlet\\lib\\dt.jar"
[HKEY_CLASSES_ROOT\CLSID\{F84A5040-0873-11D7-8D09-
444553540001}\JavaClass]
@= "javax.swing.JTextField"
[HKEY_CLASSES_ROOT\CLSID\{F84A5040-0873-11D7-8D09-
444553540001}\InterfaceClass]
@= "sun/beans/ole/OleBeanInterface"
[HKEY_CLASSES_ROOT\CLSID\{F84A5040-0873-11D7-8D09-
444553540001}\Control]
[HKEY_CLASSES_ROOT\CLSID\{F84A5040-0873-11D7-8D09-
444553540001}\Programmable]
[HKEY_CLASSES_ROOT\CLSID\{F84A5040-0873-11D7-8D09-
444553540001}\Insertable]
[HKEY_CLASSES_ROOT\CLSID\{F84A5040-0873-11D7-8D09-
444553540001}\MiscStatus]
@= "0"
[HKEY_CLASSES_ROOT\CLSID\{F84A5040-0873-11D7-8D09-
444553540001}\MiscStatus\1]
@= "18833"
[HKEY_CLASSES_ROOT\CLSID\{F84A5040-0873-11D7-8D09-
444553540001}\DefaultIcon]
@= "C:\\Program Files\\bin\\servlet\\bin\\awt.ico"
[HKEY_CLASSES_ROOT\CLSID\{F84A5040-0873-11D7-8D09-
444553540001}\Version]
@= "1.0"
[HKEY_CLASSES_ROOT\CLSID\{F84A5040-0873-11D7-8D09-
444553540001}\DataFormats\GetSet\0]
@= "2,1,16,1"
[HKEY_CLASSES_ROOT\CLSID\{F84A5040-0873-11D7-8D09-
444553540001}\DataFormats\GetSet\1]
```

```
@= "3,1,32,1"
[HKEY_CLASSES_ROOT\CLSID\{F84A5040-0873-11D7-8D09-
444553540001}\DataFormats\GetSet\2]
@= "14,1,64,1"
[HKEY_CLASSES_ROOT\CLSID\{F84A5040-0873-11D7-8D09-
444553540001}\DataFormats\GetSet\3]
@= "1,1,1,1"
[HKEY_CLASSES_ROOT\CLSID\{F84A5040-0873-11D7-8D09-
444553540001}\verb\0]
@= "&Edit,0,2"
[HKEY_CLASSES_ROOT\CLSID\{F84A5040-0873-11D7-8D09-
444553540001}\verb\-1]
@= "Show,0,0"
[HKEY_CLASSES_ROOT\CLSID\{F84A5040-0873-11D7-8D09-
444553540001}\verb\-2]
@= "Open,0,0"
[HKEY_CLASSES_ROOT\CLSID\{F84A5040-0873-11D7-8D09-
444553540001}\verb\-3]
@= "Hide,0,1"
[HKEY_CLASSES_ROOT\TypeLib\{F84A5041-0873-11D7-8D09-
444553540001}]
@= "JTextField Bean Control Type Library"
[HKEY_CLASSES_ROOT\TypeLib\{F84A5041-0873-11D7-8D09-
444553540001}\1.0]
@= "JTextField Bean Control "
[HKEY_CLASSES_ROOT\TypeLib\{F84A5041-0873-11D7-8D09-
444553540001}\1.0\0\win32]
@= "C:\\Program Files\\bin\\servlet\\bin\\JTextField.tlb"
[HKEY_CLASSES_ROOT\TypeLib\{F84A5041-0873-11D7-8D09-
444553540001}\1.0\FLAGS]
@= "2"
[HKEY_CLASSES_ROOT\TypeLib\{F84A5041-0873-11D7-8D09-
444553540001}\1.0\HELPDIR]
@= "C:\\Program Files\\bin\\servlet\\bin"
```

Несмотря на обилие регистрационной информации, вам потребуется не так много. Прежде всего, это регистрационный номер,

задаваемый параметром `CLSID`, под которым компонент зарегистрирован в Windows:

```
[HKEY_CLASSES_ROOT\JTextField.Bean\CLSID]
@= "{F84A5040-0873-11D7-8D09-444553540001}" // Это требуемый
                                            // номер
```

Теперь создадим HTML-файл, например такой (листинг 2.28).

Листинг 2.28. HTML-документ с внедренными в него бинами

```
<HTML>
<H1> CALLING BEANS</H1>
<body bgcolor=coral>
<form name=fora>
<br>
<Input type="Button" value="Show BEAN" name=b1 onClick="f1()">
<br>
<OBJECT ID="TXT" CLASSID="CLSID:F84A5040-0873-11D7-8D09-
444553540001"
 WIDTH=150 HEIGHT=20>
</OBJECT>
<br>
<br>
 <OBJECT ID="Area" CLASSID="CLSID:B4D08C60-0893-11D7-8D09-
444553540001"
  WIDTH=250 HEIGHT=100>
</OBJECT>
<br>
 <OBJECT ID="Prg" CLASSID="CLSID:C894CDE0-089A-11D7-8D09-
444553540001"
  WIDTH=250 HEIGHT=10>
</OBJECT>
<SCRIPT language="JavaScript">
<!--
  var x=0;
```

```
 function f1()
  {
d= new Array("\none","\ntwo","\nthree");
fora.TXT.setText("FROM BEAN");
fora.Area.append(d[2]);
fora.Prg.setValue(x);
x+=10;
 if(x==50){x=0;}
}
-->
</SCRIPT>
</form>
</body>
</HTML>
```

Обратите внимание на то, как в сайт вставляется объект ActiveX, созданный нами:

```
<OBJECT ID="TXT" CLASSID="CLSID:F84A5040-0873-11D7-8D09-
444553540001"
  WIDTH=150 HEIGHT=20>
</OBJECT>
```

Теперь должно быть ясно, где используется регистрационный номер компонента. Помимо текстового поля мы добавили компонент `progressbar` и текстовую область, также построенные из бинов.

На рис. 2.19 представлен результат открытия документа из листинга 2.28.

Лишь кнопка **Show BEAN** не является бином. Слово "`three`" выведено в текстовой области `JtextArea`. Видим также объект прогресс-бар (`progressbar`), заполняемый с каждым нажатием кнопки. Но для того, чтобы заставить элементы работать, нам потребовалось использовать язык JavaScript. В этой работе, как и ранее, обходимся лишь минимумом этого языка. Достаточно напомнить, что JavaScript использует синтаксис языка C, а тело JavaScript собрано из функций.

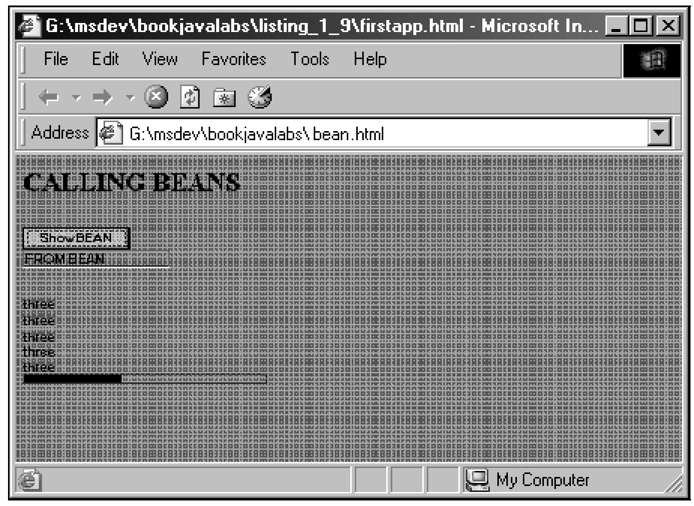


Рис. 2.19. Окно документа с бинами

```
<SCRIPT language="JavaScript">
<!--
 var x=0;
function f1()
  {
d= new Array("\none","\ntwo","\nthree");
fora.TXT.setText("FROM BEAN"); // Эта команда выводит текст
    //в текстовое поле, используя стандартный метод компонента
JTextField - setText()
fora.Area.append(d[2]); // Добавление элемента массива d
                        // в текстовую область
fora.Prg.setValue(x);
x+=10;
 if(x==50){x=0;}
}
-->
</SCRIPT>
```

Вызов функции привязан к кнопке **Show BEAN**.

В этом занятии потребуется использовать таймер в среде JavaScript. Для этой цели используйте команду `setTimeout("f1()",1000)`.

Во-первых вспомните, что JavaScript чувствителен к написанию заглавных и строчных букв. Что касается команды `setTimeout()`, то первый ее аргумент — это имя функции, которая запускается через число миллисекунд, указываемое вторым аргументом. При этом запуск выполняется однократно, а не каждые 1000 мс. Следовательно, внутри функции `f1()` нужно сбросить таймер, а затем вызвать снова саму себя. Это делается так:

```
<HTML>
<H1> CALLING BEANS</H1>
<body bgcolor=coral>
<form name=fora>
<br>
<Input type="Button" value="Show BEAN" name=b1 onClick="f1()">
<br>
<OBJECT ID="TXT" CLASSID="CLSID:F84A5040-0873-11D7-8D09-
444553540001"
  WIDTH=350 HEIGHT=20>
</OBJECT>
<br>
<br>
 <OBJECT ID="Area" CLASSID="CLSID:B4D08C60-0893-11D7-8D09-
444553540001"
  WIDTH=250 HEIGHT=100>
</OBJECT>
<br>
 <OBJECT ID="Prg" CLASSID="CLSID:C894CDE0-089A-11D7-8D09-
444553540001"
  WIDTH=250 HEIGHT=10>
</OBJECT>
<SCRIPT language="JavaScript">
<!--
  var x=0;
```

```
  var id;
  function f1()
  {id=setTimeout("f2()",1000);}
 function f2()
  {
clearTimeout(id);
d=new Date();
scs=d.toLocaleString();
fora.TXT.setText(scs);
fora.Prg.setValue(x);
x+=10;
 if(x==50){x=0;}
   f1();
}
-->
</SCRIPT>
</form>
</body>
```

Откройте этот документ и посмотрите его в действии.

Более интересно создавать бины на основе собственных классов. Такие бины должны быть упакованы в jar-архивы. Посмотрим, как это сделать. В качестве базового класса для бина используем следующий класс.

```
import java.awt.*;
import java.io.Serializable; // Этот пакет обязателен
import java.util.*;
public class MyBean implements Serializable // Интерфейс
                                            // Serializable
                                            // обязателен
{
 public Date dt; // Переменные и методы сериализуемого класса
                 // обязаны быть public
 public String getData()
```

```
 {
  dt=new Date();
  return dt.toLocaleString(); // Метод getData() возвращает
                              // строку с датой-временем
 }
```

Итак, наш бин содержит один-единственный метод `getData()`, возвращающий в строковом формате дату и время. Запомним, что класс бина должен реализовывать интерфейс `Serializable` и содержать только публичные (`public`) методы. Скомпилируем этот файл и, если компиляция пройдет успешно, получим файл MyBean.class.

Теперь следует создать так называемый *манифест-файл*, например, с именем manifest.tmp:

```
Manifest-Version: 1.0
Name: MyBean.class
Java-Bean: True
```

Запускаем Java-архиватор:

```
jar cfvm MyBean.jar manifest.tmp MyBean.class
```

Аргументами здесь являются: имя создаваемого jar-файла, имя манифест-файла и имя класса бина.

Теперь следует преобразовать jar-файл в объект ActiveX. Для этого следует сделать так, как мы описали в начале этого занятия, т. е. использовать утилиту:

```
>java –cp rt.jar; jaws.jar sun.beans.ole.Packager
```

А затем указать созданный jar-файл MyBean.jar при выборе файла в процедуре `sun.beans.ole.Packager`.

Задание

Используя таймер, напишите программу, в которой скорость смены картинок в HTML-документе управляется ползунком. Чтобы разместить картинку в документе, используйте тег:

```
<img name=i1 src="c:\...\pic.bmp" width=40 height=40>
```

Картинки следует размещать в контейнерах (тег `<div>`), и делать их поочередно невидимыми с частотой, управляемой ползунком. Вспомните работу по созданию HTML-сайта.

Контрольные вопросы

1. Что такое `bean`?
2. Что такое объект ActiveX?
3. Как создать объект ActiveX на основе библиотеки бинов?
4. Как создать свой собственный бин на базе какого-нибудь класса?
5. Как внедрить объект ActiveX в HTML-документ?
6. Как использовать методы объекта ActiveX в скрипте JavaScript?
7. Как `запустить` таймер в скрипте JavaScript?
8. Для чего следует уничтожать объект-таймер при рекурсивном обращении к таймеру?

Изучение механизма сериализации

Цель занятия

Целью практического занятия является изучение работы механизма сериализации объектов в языке Java. Необходимо научиться сохранять на форме простые объекты (не контейнеры), а также массивы однотипных объектов. Кроме этого, требуется восстанавливать записанные в файле на диске объекты и их функциональность. Рекомендуемая литература [15, 17].

Краткие теоретические сведения

Сериализация — это запись и чтение экземпляров классов в поток ввода-вывода. Мы знаем, что можно сохранять и читать из файлов простые типы данных: числа, строки, символы, логические переменные. Современные объектно-ориентированные

языки позволяют сохранять и читать объекты. Далее будем под сериализацией понимать сохранение объектов, а под *десериализацией* — их чтение из файлов.

Следующий фрагмент программы демонстрирует выполнение сериализации:

```
if (evt.target==b_serialize)
 {
  try
   {
    fos= new FileOutputStream("tmpserial");
    oos=new ObjectOutputStream(fos);
    oos.writeObject(bexit); // Запись в файл кнопки
    oos.writeObject(b_serialize);
    oos.writeObject(b_deserialize);
    oos.writeObject(tf);// Запись в файл текстовго поля
    fos.close();
    return true;
   }
  catch(Exception err)
   {System.out.println("Error:"+err);}
```

Как видим, для сохранения объектов в файле используется метод `writeObject()`. Этот метод принадлежит классу `ObjectOutputStream`. Процедура сериализации привязана к кнопке. В программной реализации создается главное окно с тремя кнопками и текстовым полем, которые подлежат сериализации. Запомним, что сериализовать можно только простые объекты, не содержащие других объектов! Контейнеры, включая формы и панели, сериализовать нельзя. Переменные `fos`, `oos` объявлены как потоковые переменные в главном классе. Эти объявления таковы:

```
public class lab14 extends Frame
{
 ObjectOutputStream oos;
 FileOutputStream fos;
 ObjectInputStream ois;
 FileInputStream fis;
```

```
 static lab14  form;
 Button bexit=new Button("Exit");
 Button b_serialize=new Button("Serial"); // Кнопка для
сериализации
 Button b_deserialize=new Button("Deserialize"); // Кнопка для
                                                // десериализации
  TextField tf=new TextField("100");
  Lab14(String s)
  { //конструктор
   super(s);
   setLayout(null);
   add(bexit);
   add(b_serialize);
   add(b_deserialize);
   add(tf);
   setBackground(new Color(100,20,150));
   bexit.setBounds(10,20,60,20);
   b_serialize.setBounds(10,40,100,20);
   b_deserialize.setBounds(10,60,100,20);
   tf.setBounds(10,100,200,20);
  }
  public boolean action(Event evt, Object ob)
  {
   if (evt.target==bexit)
    {
     System.exit(0);
     return true;
    }
   else
    if (evt.target==b_deserialize)
...
```

Исходная форма отображается на экране так, как показано на рис. 2.20.

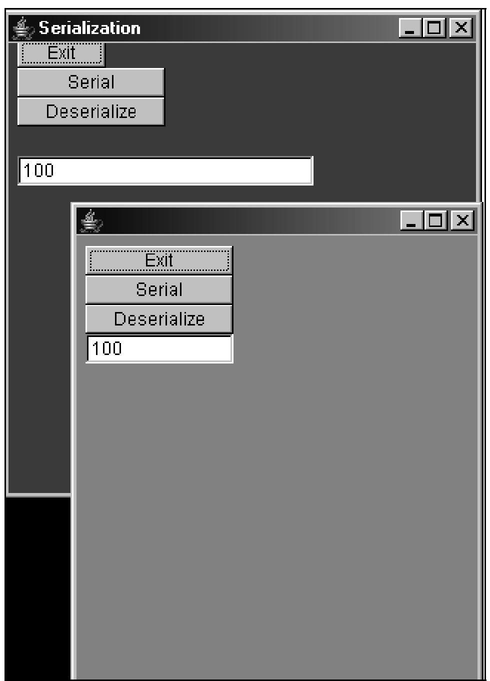


Рис. 2.20. Окно исходной и десериализованной формы

Основное окно на рис. 2.20 показано слева. Справа размещается восстановленное (десериализованное) окно. Кнопка сериализации запрограммирована следующим образом:

```
if (evt.target==b_serialize)
 {
  try
   {
    fos= new FileOutputStream("tmpserial"); // Создаем файл
                                            // для хранения
                                            // сериализованных
объектов
    oos=new ObjectOutputStream(fos);
    oos.writeObject(bexit); //Сохраняем кнопку bexit
    oos.writeObject(b_serialize); // Сохраняем кнопку
                                  // b_serialize
```

```
    oos.writeObject(b_deserialize); // Сохраняем кнопку
                                    // b_deserialize
    oos.writeObject(tf);//Сохраняем текстовое поле
    fos.close();
    return true;
   }
  catch(Exception err)
   {System.out.println("Error:"+err);}
 }
}
```

По нажатию кнопки десериализации создается новое окно-форма и в нем отображаются считываемые (десериализуемые) объекты. Чтение объектов из файла реализует команда `readObject()`.

При чтении необходимо выполнять *приведение типов*. Запомните, что после восстановления из файла десериализованный объект не содержит методов, которыми он обладал до сохранения. Таким образом, все методы десериализованных объектов должны быть прописаны в классе-окне, где они восстанавливаются. В примере таким классом является `dlg`. Обратим внимание на то, что после создания десериализованной формы с кнопками и текстовым полем основной формы может быть проведена сериализация этой второй формы и получена третья десериализованная форма и т. д. Однако вся информация об объектах записывается в один и тот же файл. Заметим также, что описание класса `dlg` практически копирует описание основного класса `lab14`. Приводим описание класса `dlg`:

```
class dlg extends Frame
{
 FileInputStream fis;
 FileOutputStream fos;
 ObjectInputStream ois;
 ObjectOutputStream oos;
 Button bexitD;
 Button b_serializeD;
 Button b_deserializeD;
```

```
 TextField tfD;
 dlg()
 {
  setBackground(new Color(180,100,150));
  setLayout(null);
  resize(400,500);
  move(50,100);
 }

 public boolean action(Event evt, Object ob)
 {
  if (evt.target==bexitD)
   {
    this.dispose();
    return true;
   }
  else
   if (evt.target==b_deserializeD)
      {
```

и т. д.

Теперь приведем полный текст приложения (листинг 2.29).

Листинг 2.29. Приложение для сериализации/десериализации объектов

```
import java.awt.*;
import java.util.*;
import java.io.*;
class dlg extends Frame
{ // Класс на основе формы для десериализации
 FileInputStream fis; // Объявление файловой переменной для
                      // чтения
 FileOutputStream fos; // Объявление файловой переменной для
                       // записи
```

```
ObjectInputStream ois; // ois используется для чтения
                       // объектов
ObjectOutputStream oos; // oos используется для записи
объектов
Button bexitD; // Кнопка для выхода
Button b_serializeD; // Кнопка для сериадизации
Button b_deserializeD; // Кнопка для десериализации
TextField tfD; // Текстовое поле
dlg() // Конструктор десериализуемой формы
{
 setBackground(new Color(180,100,150));
 setLayout(null);
 resize(400,500);
 move(50,100);
}
public boolean action(Event evt, Object ob)
{
 if (evt.target==bexitD)
  {
   this.dispose();
   return true;
  }
 else
  if (evt.target==b_deserializeD)
   {
    File  fl=new File("tmpserial"); // Выполняется
                                    // десериализация
                                    // из файла tmpserial
    if (fl.exists()) // Проверяем, существует ли этот файл
     {
      tfD.setText("FileExists");
      dlg dd= new dlg(); // Создаем новую форму
      try
       {
        fis= new FileInputStream("tmpserial");
```

```
        ois=new ObjectInputStream(fis);
        dd.bexitD= (Button) ois.readObject(); // Читаем из
                                         // файла кнопку
                                         // и выполняем
                                         // приведение типа
        dd.bexitD.setBounds(10,30,100,20); // Устанавливаем
                                           // ее размеры
                                           // и координаты
        dd.add(dd.bexitD); // Добавляем кнопку на форму
        dd.b_serializeD=(Button) ois.readObject(); // Те же
                                     // действия выполняем
                                     // со второй кнопкой
        dd.b_serializeD.setBounds(10,50,100,20);
        dd.add(dd.b_serializeD);
        dd.b_deserializeD=(Button) ois.readObject();
        dd.b_deserializeD.setBounds(10,70,100,20);
        dd.add(dd.b_deserializeD);
        dd.tfD=(TextField) ois.readObject(); // Читаем из
        // файла текстовое поле и выполняем приведение типа
        dd.tfD.setBounds(10,90,100,20);
        dd.add(dd.tfD);
        dd.show(); // Отображаем десериализованную форму
        fis.close();
        return true;
       }
      catch(Exception err)
       {System.out.println("Error:"+err);}
     }
    else
     {tfD.setText("FileNotExists");}

    return true;
   }
  else
   if (evt.target==b_serializeD) // Кнопка для сериализации
```

```
    {
     try
      {
       fos= new FileOutputStream("tmpserial"); // Создаем
                                               // файл для
                                               // сериализации
       oos=new ObjectOutputStream(fos); // Создаем потоковую
                                        // переменную
                                        // для записи в файл
                                        // объектов
       oos.writeObject(bexitD); // Пишем в файл кнопки и
                                // текстовое поле
       oos.writeObject(b_serializeD);
       oos.writeObject(b_deserializeD);
       oos.writeObject(tfD);
       fos.close();
       return true;
      }
    catch(Exception err)
      {System.out.println("Error:"+err);}
   }

  return false;
 }
}
public class lab14 extends Frame
{ // Форма основного класса
 ObjectOutputStream oos; // Объектная переменная для записи
                         // объектов в файл
 FileOutputStream fos; // Переменная для чтения объектов из
                       // файла
 ObjectInputStream ois;
 FileInputStream fis;
 static lab14 form;
 Button bexit=new Button("Exit"); // Кнопка для выхода
```

```
Button b_serialize=new Button("Serial"); // Кнопка для
                                         // сериализации
Button b_deserialize=new Button("Deserialize"); // Кнопка для
                                                // десериализации
TextField tf=new TextField("100");
lab14(String s)
{ //Конструктор основной формы
 super(s);
 setLayout(null);
 add(bexit);
 add(b_serialize);
 add(b_deserialize);
 add(tf);
 setBackground(new Color(100,20,150));
 bexit.setBounds(10,20,60,20);
 b_serialize.setBounds(10,40,100,20);
 b_deserialize.setBounds(10,60,100,20);
 tf.setBounds(10,100,200,20);
}

public boolean action(Event evt, Object ob) // Обработчик
                                            // событий
{
 if (evt.target==bexit) // Выход из приложения
  {
   System.exit(0);
   return true;
  }
 else
  if (evt.target==b_deserialize) // Запуск десериализации
   {
    File  fl=new File("tmpserial"); // Проверяем, есть ли
                                    // файл с записанными
                                    // объектами
```

```
if (fl.exists())
 {
  tf.setText("FileExists");
  dlg d= new dlg(); // Создаем новую форму на базе класса
                  // dlg
  try
   {
    fis= new FileInputStream("tmpserial"); // Создание
    // потоковой переменной для низкоуровневого ввода
    ois=new ObjectInputStream(fis); // Создание потоковой
    // переменной для высокоуровневого ввода
    d.bexitD= (Button) ois.readObject(); // Читаем на
    // новую форму объекты. При чтении выполняем
    // приведение типа
    d.bexitD.setBounds(10,30,100,20);
    d.add(d.bexitD);
    d.b_serializeD=(Button) ois.readObject();
    d.b_serializeD.setBounds(10,50,100,20);
    d.add(d.b_serializeD);
    d.b_deserializeD=(Button) ois.readObject();
    d.b_deserializeD.setBounds(10,70,100,20);
    d.add(d.b_deserializeD);
    d.tfD=(TextField) ois.readObject();
    d.tfD.setBounds(10,90,100,20);
    d.add(d.tfD);
    d.show(); // Отображаем новую форму
    fis.close(); // Закрываем поток для ввода
    return true;
   }
  catch(Exception err)
   {System.out.println("Error:"+err);}
 }
else
 {tf.setText("FileNotExists");}
```

```
    return true;
    }
   else
    if (evt.target==b_serialize) // Активизировано событие
сериализации
     {
      try
       {
        fos= new FileOutputStream("tmpserial");
        oos=new ObjectOutputStream(fos); // Запись в файл
                                         // кнопок
                                         // и текстового поля
        oos.writeObject(bexit);
        oos.writeObject(b_serialize);
        oos.writeObject(b_deserialize);
        oos.writeObject(tf);
        fos.close();
        return true;
       }
      catch(Exception err)
       {System.out.println("Error:"+err);}
     }
   return false;
  }

  public static void main(String[] args)
  {
   form = new lab14("Serialization");
   form.resize(400,400);
   form.move(200,100);
   form.show();
  }
}
```

Задание

Разберитесь с теоретическим материалом. Основная задача, которую вы должны реализовать? — это сериализовать невизуальный экземпляр класса. Вы должны удостовериться, что все классовые переменные сохранятся в том виде, в каком они находились в момент сериализации. Конкретное содержание вашего невизуального класса может быть произвольным — пусть, например, это будет класс **банковский счет**, содержащий дату последней чековой операции и состояние счета, а также шифр владельца.

Научитесь сериализовать массив однотипных объектов. Особенно обратите внимание на то, что создание массива объектов выполняется в два этапа. На первом этапе создается массив в целом обычным образом, а на втором каждый объект массива создается отдельной командой.

Контрольные вопросы

1. Что такое сериализация и десериализация объектов?
2. Можно ли сериализовать объекты-контейнеры?
3. Сохраняются ли при сериализации методы объектов?
4. Какой класс обеспечивает сериализацию?
5. Поясните листинг 2.29.
6. Можно ли сериализовать массив объектов?
7. Укажите, когда следует использовать сериализацию?

Создание сервлетов

Цель занятия

Научиться писать *сервлеты* и тестировать их выполнение на сервере. Рассмотреть вопросы, связанные с инсталляцией Web-сервера TOMCAT. Научиться правильно размещать классы сервлетов и адресовать сервлеты из документов HTML. В качестве дополнительной литературы можно использовать [10, 14].

Краткие теоретические сведения

Сервлет — это Java-программа, работающая на стороне сервера. Для запуска сервлета следует установить на стороне сервера программу Web-сервера. В качестве Web-сервера используем Tomcat 3.0 (можно работать также с версиями 3.1, 4.0, 5.0). Вам следует скачать этот бесплатный Web-сервер, используя Internet-адрес **http://www.jakarta.apache.org/tomcat/**. На этом же сайте вы можете прочитать инструкции по установке Tomcat. При запуске приложения Web-сервер должен быть активным (запущенным из командной строки MS-DOS). При инсталляции Tomcat 3.0 следует обратить внимание на следующее.

- В конфигурационном файле server.xml замените значение порта 8080 на 80 в строке

  ```
  <ContextManager port=8080 hostName=" "  inet=" ">
  ```

- Запишите в файле startup.bat значение переменной `JAVA_HOME` в первой строке этого файла так:

  ```
  set JAVA_HOME=путь к каталогу с файлом java.exe
  ```

- Аналогичным образом установите там же значение переменной `CATALINA_HOME`, присвоив ей значение пути, где расположен Tomcat:

  ```
  set CATALINA_HOME=путь к каталогу Tomcat:
  ```

- При работе с Windows 98 установите в сеансе MS-DOS в окне свойств размер памяти Initial Environment с Avto на 2816. При работе в более старших версиях Windows этого делать не надо.

Сервлеты пишут с помощью пакета javax. Этот пакет следует скопировать из дистрибутива Tomcat в корневой каталог Java.

Сервлет обрабатывает строки запросов от браузеров типа Internet Explorer. Браузер передает на сервер запрос, содержащий данные сайта (HTML-документ). Далее считаем, что Web-сервер и клиент расположены на одном и том же компьютере. HTML-документ пишется на языке HTML, в нашей работе он будет таким:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<H1> Получение данных от формы</H1>
</HEAD>
```

```
<BODY BGCOLOR=#34AAFF>
<FORM
ACTION="http://127.0.0.1:80//examples/servlet/MyServlet">
<INPUT type="TEXT " name="tf" value="Input Something
Pleasant"></Input>
<Br>
<INPUT type="SUBMIT" value="SEND FORM TO SERVLET"></INPUT>
</Form>
</BODY>
</HTML>
```

Этот документ открывается браузером в следующем виде (рис. 2.21).

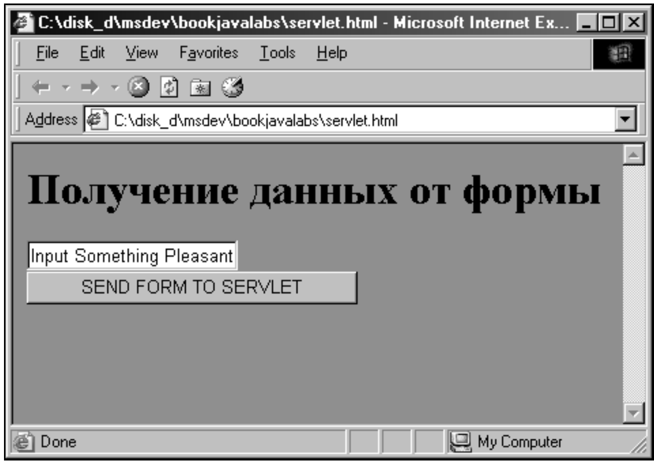


Рис. 2.21. Окно HTML-документа для отсылки Web-серверу

Видим, что на форме расположены кнопка и текстовое поле. Этого вполне достаточно для разъяснения сути. Отметим, что кнопка имеет тип `SUBMIT`. Кнопка этого типа используется для соединения с сервлетом.

В HTML-документе необходимо в теге `<ACTION>` указать адрес программы сервлета, которому будет передан сам документ при нажатии на кнопку типа `SUBMIT`. Соотвествующая строка в тексте документа такова:

```
<FORM ACTION="http://127.0.0.1:80//examples/servlet/
MyServlet">
```

Из этой строки следует, что имя класса сервлета есть `MyServlet.class`. Часть адреса `http://127.0.0.1:80` задает тип протокола, сетевой адрес компьютера и номер порта (в нашем случае — 80). Для создания класса сервлета следует подготовить исходный Java-файл сервлета. Этот файл нужно скомпилировать обычным образом и получить класс сервлета. Файл с классом сервлета следует поместить, как это принято в Tomcat, в каталог ...//TOMCAT/webpages/WEB-INF/classes. Однако все же следует самостоятельно проверить, откуда Web-сервер выполняет запуск сервлетов. Для этого активизируйте окно программы Internet Explorer и введите следующий адрес: **http://localhost**. До выполнения этих действий Tomcat уже должен быть запущен. После нажатия кнопки <Enter> откроется домашняя страница Tomcat (рис. 2.22). Выберите на ней ссылку **Servlet Examples** и откройте текст HTML-документа из меню браузера (**View Source**). Найдите в тексте документа ссылку на сервлет.

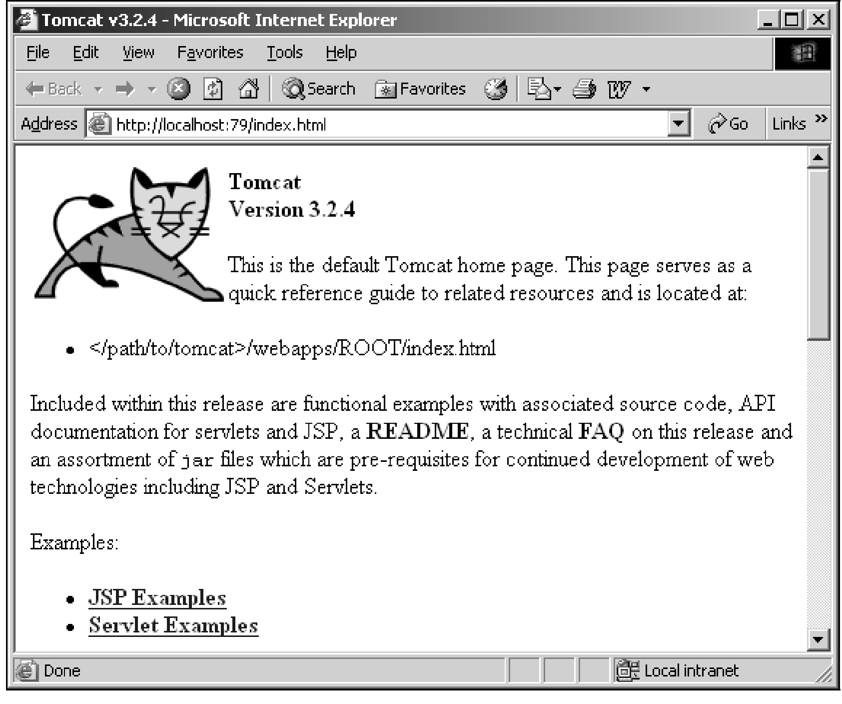


Рис. 2.22. Окно Web-сервера Tomcat

Напишем следующий исходный код сервлета (листинг 2.30).

Листинг 2.30. Текст сервлета

```
import java io.*;
import javax servlet.*;
import javax servlet.http.*;
public class MyServlet extends HttpServlet
{
 public void doGet (HttpServletRequest request,
                   HttpServletResponse response)
 throws ServletException, IOException
 {
  response.setContentType("text/html");
  printWriter out= response.getWriter();
  String title="Получение данных от формы";
  String DocType= "<!DOCTYPE HTML PUBLIC \"-//W3//DTD HTML
  4.0"+
                          "Transitional//EN\">\n";
  out.println ("<HTML>\n"+
                  "<HEAD><H1> Получение данных от
                  формы</H1></HEAD>\n"+
                 "<BODY BGCOLOR=34AAFF><BR><BR>");
  String s=request.getParameter("tf");
  out.println("TEXT FROM TEXT-FIELD is"+s);
  out.println("</BODY></HTML>");
 }
}
```

Как видим, код сервлета получился небольшим. Дадим следующие пояснения. Сервлет не только должен прочитать переданные ему браузером данные документа, но и вернуть результаты своей работы обратно на сторону клиента и, как правило, в форме HTML-документа. Строка:

```
<!DOCTYPE  HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
```

используется для указания браузеру того, что требования к правильности написания тегов не являются жесткими (параметр — `Transitional`).

Выводу тела документа из сервлета должен предшествовать вывод типа документа:

```
response.setContentType("text/html");
```

Это обстоятельство следует иметь в виду всегда, поскольку браузер по-разному интерпретирует различные виды документов.

Значения переменных формы получаем с помощью команды наподобие следующей:

```
String s=request.getParameter("tf");
```

Здесь переменная класса `HttpServletRequest request` позволяет прочитать все данные формы, переданные браузером, используя команду `getParameter()`. Операндом команды является имя текстового поля, объявленное в HTML-документе:

```
<INPUT type="TEXT" name="tf" value="Input Something Pleasant"></Input>
```

Таким образом, можно получить значения всех визуальных элементов формы. Другая объектная переменная класса `HttpServletResponse response` позволяет вернуть информацию обратно клиенту. С помощью этой переменной можно вернуть клиенту обычный HTML-документ. В приведенном ранее примере видим строки:

```
printWriter out= response.getWriter();
...
out.println ("<HTML>\n"+
                "<HEAD><H1> Получение данных от
                формы</H1></HEAD>\n"+
               "<BODY BGCOLOR=34AAFF><BR><BR>");
```

где как раз и используется данная объектная переменная: на ее основе создается потоковая переменная для вывода `out`, которая затем выполняет прямой вывод в формате HTML. Например, строки:

```
out.println ("<HTML>\n"+
                "<HEAD><H1> Получение данных от
                формы</H1></HEAD>\n"+
               "<BODY BGCOLOR=34AAFF><BR><BR>");
```

```
String s=request.getParameter("tf");
out.println("TEXT FROM TEXT-FIELD is"+s);
out.println("</BODY></HTML>");
```

как раз и обеспечивают вывод окна документа, показанного на рис. 2.21.

Для активизации сервлета нажмите кнопку `Submit` (в документе на ней имеется надпись "`SEND FORM TO SERVLET`"). Кнопка `Submit` выполняет попытку соединения с URL-адресом, прописанным в параметре `ACTION` тега `FORM`:

```
ACTION="http://127.0.0.1:80//examples/servlet/MyServlet">
```

Этот адрес должен определять место нахождения сервлета `MyServlet.Class`. Это место регламентируется документацией по Tomcat, так что определите его исходя из примеров, установленных в директории EXAMPLES TOMCAT. Обратим внимание на то, что в тексте сервлета используется метод обработки формы `doGet()`. Это наиболее быстрый способ, ориентированный на передачу серверу небольших сайтов. Альтернативным способом является метод `doPost()`, который здесь не рассматривается.

Наконец, остается привести базовую структуру сервлета:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletClass extends HttpServlet
{
 public void doGet(HttpServletRequesr request,
                   HttpServletResponse response)
 throws ServletException, IOException
 {
  // Переменная request используется для чтения полей формы
  // Переменная response используется для вывода ответа
  // в формате HTML
  PrintWriter out = response.getWriter();
  // Используем out.println для вывода документа на стороне
  // клиента
 }
}
```

Итак, резюмируем наши знания по этому практическому занятию.

- Перед выполнением сервлета необходимо запустить Web-сервер Tomcat. Запуск и настройка сервера Tomcat являются важной составной частью настоящей работы. При работе с Tomcat 3.0 в файле server.xml измените строку:

  ```
  <ContextManager port="8080" HostName=" " inet=" ">
  ```

  на

  ```
  <ContextManager port="80" HostName=" " inet=" ">
  ```

  тем самым задав новое значение порта для связи с браузером.

- Далее в файлах startup.bat и startserver.bat задайте значение переменной `JAVA_HOME`, равной пути, где расположен дистрибутив JAVA SDK. Поскольку файлы startup.bat и startserver.bat являются командными файлами MS-DOS, то следует поместить команду

  ```
  set JAVA_HOME=c:\Program Files\JAVA\...
  ```

  указав путь к каталогу с программой `java.exe` в самом начале этих файлов.

  Аналогичным образом следует установить параметр `CATALINA_HOME` равным значению пути к каталогу Tomcat.

- Для запуска сервера Tomcat запустите из среды программы Norton Commander или FAR (или из командной строки MS-DOS) командный файл startup.bat.

- Теперь скомпилируйте сервлет и устраните ошибки. Пусть файл MyServlet.class успешно создан. Чтобы этот файл был доступен из вашего сайта, его нужно поместить в нужный каталог. По умолчанию таковым является

  ```
  c:\TOMCAT\webpages\WEB-INF\classes.
  ```

  Однако проще выяснить, откуда запускаются сервлеты, используемые в Tomcat в качестве примеров, как было описано ранее.

Задание

Разберитесь с теоретическим материалом. Напишите собственный сервлет, который получает данные от формы, а именно: групповой идентификатор пользователя и пароль, и сравнивает его с известным паролем, читая последний из

файла. При совпадении выдать приветствие, иначе — сообщение о нарушении прав доступа. Для ввода пароля используйте элемент HTML `<INPUT type="PASSWORD" name="MYPSWD">`.

Контрольные вопросы

1. Что такое сервлет?
2. Для чего нужен Web-сервер?
3. Как указывается адрес сервлета в документе HTML?
4. Для чего нужна кнопка SUBMIT?
5. Как открыть окно Web-сервера Tomcat?
6. Где взять пакет javax для работы с сервлетами?
7. Чем отличаются методы `doGet()` и `doPost()` друг от друга?
8. Как получить значение параметра формы в сервлете?
9. Как сервлет возвращает результат своей работы на сторону клиента?
10. Поясните код программы из листинга 2.30.
11. Как установить и запустить Tomcat?
12. Из какого каталога Tomcat запускает классы сервлетов? Как практически это узнать?
13. Объясните общее взаимодействие клиентской и серверной сторон.

Создание почтовой службы в стандартном Java

Цель занятия

Ознакомиться с построением простого почтового клиента для сети Internet. Научиться формировать почтовое сообщение и получать ответные сообщения от почтового сервера. Дополнительные сведения можно найти в [15].

Краткие теоретические сведения

Рассмотрим сравнительно небольшой вопрос реализации связи с почтовым сервером. Нам надлежит создать программу почтового клиента для отсылки письма почтовому серверу. Последний доставит письмо по указанному в нем адресу.

Почтовый клиент использует протокол SMTP (Mail Transporting Protocol), имеющий следующий набор команд:

- HELO localhost/domain.name — устанавливает соединение с сервером;
- MAIL FROM **user@email.com** — определяет адрес отправителя письма;
- RCPT TO **user@mail.com** — определяет адрес получателя письма;
- DATA — определяет начало данных письма:

  ======= текст письма =====
- QUIT — разрыв соединения.

Таким образом, наш почтовый клиент должен вставить указанные выше команды в письмо и направить его через почтовый сокет серверу.

Следующая программа (листинг 2.31) реализует указанную цель и составляет предмет исследования в этом занятии. Приводим ее текст с комментариями по ходу программы.

Листинг 2.31. Программа почтового клиента

```
import java.io.*;
import java.net.*;

public class mailclient
{
 static PrintStream ps=null; // Статическая объектная
                             // переменная для
                             // класса вывода в сокет
 static DataInputStream dis=null; // Статическая объектная
                                  // переменная
                                  // для класса ввода
```

```
public static void send (String str)
throws IOException
{ // Программа для отправки письма
 ps.println(str); // Текст  письма – в str
 ps.flush(); // Этот метод всегда используется
            // для очистки буфера вывода
 System.out.println("Сообщение отослано");
}
public static void receive() // Получение ответа SMTP-сервера
throws IOException
{
 String readstr=dis.readLine();
 // Вывод ответа сервера SMTP:
 System.out.println("SMTP-response: "+readstr);
}

public static void main(String [] args)
{
 String hello="HELO "; // Заголовок письма
 String mailfrom="MAIL FROM:vovochka@koshkindom.com";
 // Напишите, откуда идет письмо
 String rcptto="RCPT TO: naderevniu@dedushke.com";
 // Напишите, куда идет письмо
 String subject= "SUBJECT: JAVA is COOL ! "; // Тема письма
 String data="This is to my uncle"; // Текст письма
 String body="Congratulations with first SMTP-channel
 to YOU.";
 // Финальный аккорд – точка - завершает послание

 Socket smtp=null; // Объявление почтового сокета
 try
```

```
 {
  // 25 — это стандартный номер порта SMTP
  smtp= new Socket("smtp.any.com",25);
  OutputStream os=new smtp.getOutputStream();
  ps= new PrintStream(os);
  InputStream is= smtp.getInputStream();
  dis= new DataInputStream(is);
 }
catch( IOException e)
 {}

try
 {
  String loc=InetAddress.getlocalHost().getHostName();
  // Получаем Интернет-адрес хоста, на котором
  // располагается реальный почтовый сервер

  send(hello+loc); // Посылка первой команды
  receive(); // Получаем ответ почтового сервера
  send(mailfrom); // Посылка второй команды
  receive(); // Получаем ответ почтового сервера
  send(rcptto); // Посылка третьей команды
  receive(); // Получаем ответ почтового сервера
  send(data); // Посылаем данные
  receive(); // Получаем ответ почтового сервера
  send(subject); // Посылка темы
  receive(); // Получаем ответ почтового сервера
  send(body); // Посылка тела сообщения
  receive(); // Получаем ответ почтового сервера
  smtp.close();
 }
catch(IOException  e)
```

```
    {}
  System.out.println("Mail sent");
 }
}
```

Вы должны понимать, что в письме указывается адрес отправителя и адрес получателя. Это обычные e-mail-адреса. Вместе с этими адресами следует определить адрес почтового сервера. Этот адрес следует занести в переменную `loc`.

Для запуска данного приложения следует заменить адреса получателя и отправителя и адрес домена на реальные адреса. Нам нужно получить адрес домена, определяющего дислокацию почтового сервера. Например, воспользуйтесь адресом **www.mail.ru**, где имеется почтовый сервер, либо **yahoo.com**.

Замечание

Работу следует выполнять при активном соединении с сетью Internet. Вам необходимо направить письмо самому себе при наличии у вас, разумеется, почтового ящика. Желаем успеха!

Задание

Разберитесь с теоретическим материалом. Вспомните, как использовались сокетные соединения. Воспользуйтесь сервисом, предоставляемым почтовым сервером **Mail.ru**. Подготовьте и отправьте себе или другу электронное письмо.

Контрольные вопросы

1. Какой порт использует сервер SMTP?
2. Какие команды обрабатывает почтовый сервер при работе с клиентом?
3. Объясните текст листинга 2.31.
4. Следует ли подключиться к Internet при работе с почтовым сервером?
5. Возвращает ли сервер ответные сообщения?
6. Какой класс используется для записи содержимого письма в сокет сервера?

Создание JSP-страниц

Цель занятия

Ознакомиться с технологией использования JSP-страниц. Научиться использовать методы Java внутри страниц, а также подключать требуемые пакеты. Научиться формировать HTML-документ для отправки на сторону клиента. Дополнительные сведения по данной теме можно найти в [2, 13].

Краткие теоретические сведения

JSP-страницы (Java Server Pages) — это в некотором смысле упрощенные варианты сервлетов. JSP-страницы представляют собой HTML-документы (сайты), содержащие специальные теги, связанные с языком Java. Например, такими тегами могут быть фрагменты программ на языке Java (они называются *скриплетами*). JSP-страницы подобно сервлетам размещаются на сервере и получают запросы от браузера — Internet Explorer. На клиентскую сторону обратно возвращается заготовленный в JSP-странице HTML-документ.

Далее помещаем пример простейшей JSP-страницы (листинг 2.32).

Листинг 2.32. Пример простой JSP-страницы

```
<%@ page import="java.util.*; java.awt.*;"%>

<HTML>

<HEAD><Title> Простейшая страница JSP</Title>

<META HTTP-equiv=Content-Type content="text/html;
charset=windows-1251">

</Head>

<BODY>

HELLO FROM JSP

<BR>
```

```
<P>
<%! String getDate() {
        Date sdf= new Date();
       return sdf.toLocaleString();}%>
TODAY IS : <%=getDate()%>
<BR>
ALL RIGHTS RESERVED
</BODY>
</HTML>
```

Все новые теги, отличные от HTML, взяты в `<% ... %>`. Это и есть *JSP-теги*. Они различаются по своему обозначению и назначению, как показано далее:

- `<% --` — комментарий;
- `<%!` — объявление переменных и методов;
- `<%=` — получение и вставка значения;
- `<%@ page import = ...` — подключение внешнего файла;
- `<%@ page contentType= ...` — определение типа документа, передаваемого скрипту браузером.

Строка

```
<%@ page content="text/HTML; charset=windows-1251"%>
```

объявляет браузеру о типе возвращаемого документа (`text/HTML`) и используемом шрифте.

Строка

```
<%@ page import="java.util.*; java.awt.*; "%>
```

указывает подключаемые пакеты для скриплетов, помещенных в тело JSP-страницы.

Строки

```
<%! String getDate() {
        Date sdf= new Date();
       return sdf.toLocaleString();}%>
```

содержат объявление и реализацию метода `getDate()` с указанием типа значения, возвращаемого методом.

Наконец, строка

```
TODAY IS : <%=getDate()%>
```

обеспечивает вставку значения, возвращаемого методом `getDate()` в тело HTML-документа.

Поскольку JSP-страница все равно преобразуется в сервлет, то ей доступны некоторые объекты сервлетов, а именно:

- `out` — для вывода текста в HTML-документ;
- `request` — позволяет читать значения параметров документа (объект типа `HttpRequest`);
- `response` — позволяет писать параметры HTML-документа (`HttpResponse`).

Пример чтения данных из формы представлен в лиситнге 2.33.

Листинг 2.33. Пример чтения данных из формы

```
<%@ page import="java.util.* "%>
<HTML><BODY> Чтение параметров
<br><P>
<response.setContentType("text/html");
<% Enumeration fields=request.getParameterNames();
      while(fields.hasMoreElements())
     {String nm=(String) fields.nextElement();
       String v1=request.getParameterValues(nm)[0];
       out.println("Параметр: "+nm+" значение="+v1);}
%>
</BODY>
</HTML>
```

В этом примере используется класс `Enumeration` для получения списка имен параметров HTML-страницы. Под параметром HTML-страницы понимается имя элемента формы, например: текстового поля или выпадающего списка. Список имен параметров получаем командой:

```
Enumeration fields=request.getParameterNames();
```

Затем в цикле для каждого очередного элемента `nm` списка имен параметров `fields`, получаемого по команде:

```
String nm=(String) fields.nextElement();
```

получаем значение параметра по команде

```
String v1=request.getParameterValues(nm)[0];
```

Имея в виду, что параметр может иметь более одного значения, берем первое, для чего используем индекс 0 в массиве возвращаемых значений команды `getParameterValues(nm)[0]`.

Технология использования JSP-страниц

Прежде всего наберите текст страницы и сохраните его в файле с расширением jsp. Затем нужно поместить этот файл в то место каталога Tomcat, где расположены HTML-страницы. Страницы компилировать не надо. Ошибки возвращает сам браузер. Однако для получения подробных сведений об ошибках нужно в меню браузера выбрать пункт **Tools | Internet Options | Advanced** и сбросить флажок **Show Friendly HTTP-error messages**.

Для запуска JSP-страницы необходимо построить сайт и адресовать его к JSP-странице так же как и в случае сервлета.

Задание

Создать JSP-страницу, осуществляющую возврат содержимого текстового файла обратно в сайт, если введен правильный пароль. Содержимое текстового файла должно состоять из одной-двух строк, отображаемых в поле **TextArea** на сайте.

Контрольные вопросы

1. В чем специфика JSP-страниц в сравнении с сервлетами?
2. Какие теги используют JSP-страницы?
3. Как подключаются пакеты Java в JSP-страницы?
4. Как описываются методы в JSP-страницах?
5. Объясните технологию использования JSP-страниц.

Создание простого браузера

Цель занятия

Освоить возможности реализации средствами языка Java простейшего браузера. Изучить возможности и функции браузера. Познакомиться с системой навигации по сайтам. Дополнительные источники информации [13].

Краткие теоретические сведения

Для создания браузера как объекта необходимо к базовому классу подключить интерфейс `HyperlinkListener`. Браузер будет открывать указываемый ему сайт на панели `JEditorPane`, используя параметр-объект типа URL (Universal Resource Locator — универсальный адрес ресурсов):

```
URL url=new URL(String initialUrl);
JEditorPane htmlPane= new JEditorPane(url);
htmlPane.setPage(url);
```

Для навигации по гиперссылкам необходимо подключить прослушиватель:

```
htmlPane.addHyperlinkListener(this);
```

Данный прослушиватель реагирует на событие, связанное со сменой документов.

Мы будем выполнять навигацию так: набираем в текстовом поле новый URL, щелкаем по текстовому полю мышью и обрабатываем событие от текстового поля так, как показано далее:

```
Public void actionPerformed(Actionevent ae)
{
 String url;
 if(ae.getSource()== urlField)
  {
   url= urlField.getText();
   JOptionPane. showMessageDialog(null, "url="+url);
   try
    {
```

```
     htmlPane.setPage(new URL(url));
     JOptionPane.showMessageDialog(null,
     "ActivatedFrom>"+url);
    }
   catch (IOException io)
    {}
 else
...
```

В этом примере использован объект `htmlPane` класса `JEditorPane`, который открывает HTML-документ с помощью команды `setPage(URL url)`. Класс `JEditorPane` описан в пакете javax.swing. Открытие в окне документа и есть реализация функций простейшего браузера. Класс `JEditorPane` позволяет отображать только текстовую информацию HTML-документа, но не предоставляет взможности отображать, например, рисунки. В приложении мы снабжаем объект `JEditorPane` полосой вертикальной прокрутки для просмотра больших сайтов. Это делается так:

```
JScrollPane sp= new JScrollPane(htmlPane); // полоса
                         // прокрутки для больших сайтов
getContentPane().add(sp, BorderLayout.CENTER);
```

После набора в текстовом поле нового URL и нажатия клавиши <Enter> будет открыт новый документ. Заметим, что адрес документа следует записывать с указанием слова `file` в самом начале:

```
file:/c:/Sun/1.html
```

Нажатие клавиши в активном текстовом поле обеспечивает срабатываение метода интерфейса `ActionListener`:

```
// Обработчик событий от кнопки и текстового поля:
public void actionPerformed(ActionEvent ae)
{
 String url;
 if(ae.getSource()==urlField) // Событие возникло при нажатии
                              // клавиши для активного
                               // текстового поля
                               // с адресом URL
```

```
  {
   url=urlField.getText();
   JOptionPane.showMessageDialog(null,"url="+url);
   try
    {
     // Смена сайта по этой команде:
     htmlPane.setPage(new URL(urlField.getText()));
    }
   catch (IOException ioe)
    {}
  }
 else
  if (ae.getSource()==b1)
   System.exit(0);
}
```

Приведем также текст обработчика события `HyperlinkEvent`:

```
public void hyperlinkUpdate(HyperlinkEvent hle) // Это
// обработчик событий от смены сайта. Просто выводим
// сообщение:
{
 if(hle.getSource()== HyperlinkEvent.EventType.ACTIVATED)
  {
   JOptionPane.showMessageDialog(null,"NewURLLoaded");
  }
}
```

Теперь приведем полный текст программы, в котором вам уже будет нетрудно разобраться (листинг 2.34).

Листинг 2.34. Простой браузер

```
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
```

```
import java.awt.event.*;
import java.net.*;
import java.io.*;

public class Browser extends JFrame
implements HyperlinkListener, ActionListener // Подключение
                                             // интерфейсов
{
 public static void main(String[]args)
 {
  Browser br= new Browser("file:/c:/Sun/1.html");
                              // Указываем, какой сайт открыть
  br.setSize(600,800);
  br.setBackground(new Color(100,200,200));
  br.setVisible(true);
 }
 private JButton b1; // Кнопка для завершения приложения
 private JTextField urlField; // Текстовое поле для ввода url
 private JEditorPane htmlPane; // Панель для отображения сайта
 private String initialUrl;
 public Browser(String initurl)
 {
  this.initialUrl= initurl;
  JPanel topPanel = new JPanel(); // Панель для размещения
                                  // кнопки и текстового поля
  urlField=new JTextField(40);
  urlField.setText(this.initialUrl); // Адрес сайта
                                     // в текстовое поле
  urlField.addActionListener(this); // Нажатие кнопки вызывает
                                    // смену сайта
  topPanel.add(urlField);
  b1= new JButton("EXIT");
  b1.addActionListener(this);
```

```
  topPanel.add(b1);
  Container knt=getContentPane();
  knt.add(topPanel,BorderLayout.NORTH);
  try
   {
    JOptionPane.showMessageDialog(null,"Try to create URL");
    URL url = new URL(this.initialUrl);

JOptionPane.showMessageDialog(null,""+url.toExternalForm());
    // Команда toExternalForm() преобразует url в строку
    htmlPane= new JEditorPane(url); // htmlPane отображает
                                    // сайт
    htmlPane.setEditable(false); // Содержимое сайта нельзя
                                 // редактировать
    htmlPane.addHyperlinkListener(this); // Этот
                                         // прослушиватель
                                         // реагирует на смену
                                         // сайтов
    htmlPane.setBackground(new Color(200,200,200));
    htmlPane.setPage(url); // Отображаем сайт этой командой
    JScrollPane sp= new JScrollPane(htmlPane); // Полоса
                                               // прокрутки
                                               // для больших
                                               // сайтов
    getContentPane().add(sp, BorderLayout.CENTER);
   }
  catch(IOException ioe)
  {}
 }

 public void actionPerformed(ActionEvent ae) // Обработчик
                  // событий от  кнопки и текстового поля
```

```
 {
  String url;
  if(ae.getSource()==urlField) // Событие возникло при нажатии
  // клавиши для активного текстового поля с адресом URL
   {
    url=urlField.getText();
    JOptionPane.showMessageDialog(null,"url="+url);
    try
     {
      htmlPane.setPage(new URL(urlField.getText())); // Смена
                                                      // сайта
                                             // по этой команде
     }
    catch (IOException ioe)
     {}
   }
  else
   if (ae.getSource()==b1)
    System.exit(0);
 }
 public void hyperlinkUpdate(HyperlinkEvent hle) // Это
 // обработчик событий от смены сайта. Просто выводим
 // сообщение.
 {
  if(hle.getSource()== HyperlinkEvent.EventType.ACTIVATED)
   {
    JOptionPane.showMessageDialog(null,"NewURLLoaded");
   }
 }
}
```

Окно созданного браузера и результат отображения документа приведены на рис. 2.23.

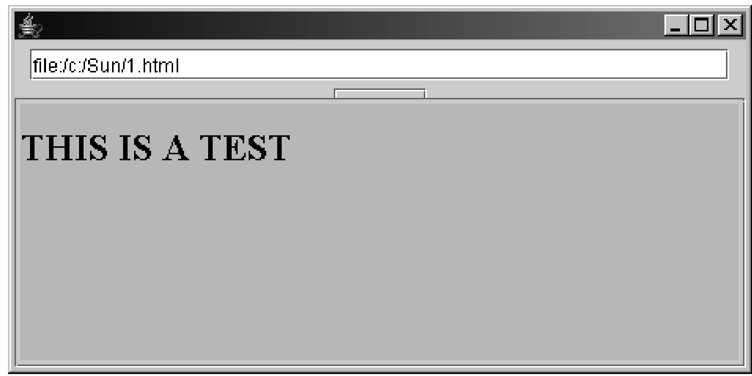


Рис. 2.23. Окно простейшего браузера

Резюмируя материал занятия, обратим внимание на два обстоятельства.

- При указании пути к сайту нужно помещать префикс `file:/`, например:

  `file:/c:/Sun/1.html`

- Созданный нами браузер отображает текстовую информацию и не отображает рисунки.

Задание

Изменить программу так, чтобы в новом ее варианте стало две панели для открытия сайтов (на каждой панели можно было бы открыть свой сайт).

Контрольные вопросы

1. Какова задача браузера?
2. Какой класс позволяет создавать простой браузер?
3. Каковы ограничения, накладываемые на простой браузер?
4. Как записывается URL?
5. Что такое URL?
6. С помощью какого метода выполняется отображение сайта?

7. Как подключить полосу прокрутки к панели?
8. Для обработки каких событий предназначен интерфейс `HyperlinkListener`?

Сводка основных использованных команд Java

`actionPerformed(ActionEvent e)` — метод интерфейса `ActionListener` для обработки событий от кнопок, пунктов меню и текстовых полей (при нажатии клавиши <Enter>.

`add(b1)` — добавляет на форму элемент с именем `b1`; например, кнопку.

`add(b1, BorderLayout.NORTH)` — размещает элемент с именем `b1` на верхней части формы с помощью диспетчера компоновки `BorderLayout`.

`btn.addActionListener(this)` — добавление прослушивателя событий к кнопке.

`addMouseListener(this)` — добавление прослушивателя событий от мыши для формы.

`amountstring=x.toString()` — создает строку `amountstring` из объектной целочисленной переменной `x` типа `Integer` (*см.* `Integer x=`).

`bar.length()` — возвращает длину массива с именем `bar` (*см.* `int bar[]=`).

`b1.addActionListener(this)` — добавляется прослушиватель событий для элемента с именем `b1`.

`b1.addKeyListener(this)` — добавляется прослушиватель событий от клавиатуры для элемента `b1` (при условии, что он имеет фокус).

`b1.dispose()` — удаляет элемент с именем `b1` из памяти.

`b1.hide()` — прячет элемент с именем `b1`.

`b1.move(x,y)` — перемещает элемент с именем `b1` в позицию с координатами `(x,y)`.

`b1.show()` — делает элемент с именем `b1` видимым.

`b1.setBounds(20,20,200,300)` — размеры элемента `b1` таковы: ширина — 200, высота — 300, левый верхний угол по `x` — 20, левый верхний угол по `y` — 20.

`b1.request()` — элемент с именем `b1` получает фокус (становится активным).

`b1.resize(400,300)` — изменяет размеры элемента (формы) с именем `b1`.

`b1.setFont(fnt)` — добавляет на элемент `b1` (например, кнопку или ярлык) новый шрифт.

`b1.setVisible(true)` — делает элемент `b1` видимым. При `setVisible(false)` — невидимым.

`Book book = new Book("Pinoccio",200)` — создает объект `book` класса `Book`, вызывая его конструктор и передавая ему два параметра: строку `"Pinoccio"` и число 200.

`ClassForName("sun.jdbc.odbc.JdbcOdbcDriver")` — подключает драйвер JDBC-ODBC.

`Connection dbc=DriverManager.getConnection(url)` — получает соединение с указанной базой. Переменная `url` должна определять ранее созданное соединение, например, `String url ="jdbc:odbc:vfp"` — здесь указывается ранее созданное `odbc`-соединение с именем `vfp`.

`Date d=new Date()` — создание объектной переменной типа даты.

`d.toLocaleString()` — перевод даты в строковое представление.

`dis.readUTF()` — читает строку в формате Unicode из потока `DataInputStream` с именем `dis`.

`dosob = (Button) ois.readObject()` — чтение из файла ранее сериализованного в нем объекта и приведение его к типу `Button`.

`dos.writeUTF("Text")` — пишет строку `"Text"` в формате Unicode в поток `DataOutputStream` с именем `dos`.

`dos.writeObject(sf)` — сериализация (запись в файл) объекта с именем `sf`.

`dos.Close()` — закрывает поток.

`FileDialog loader=new FileDialog(this, "browse", FileDialog.LOAD)` — создает переменную типа файлового диалога для чтения файла.

`Font fnt= new Font("Times new Roman", Font.BOLD, 16)` — создает переменную типа `Font` с указанным названием шрифта, жирностью и высотой символов.

`getAppletContext().showDocument(URL url, "_blank")` — открывает HTML-документ с заданным сетевым адресом URL вне какого-либо окна.

`evt.getSource()` — получение ссылки на объект, инициировавший событие. Применяют, например, в методе `actionPerformed()`.

`getSize().width()` — возвращает ширину формы (апплета).

`getSize().height()` — возвращает высоту формы (апплета).

`g.drawArc(x,y,x1,y1,a,b)` — рисуется дуга окружности в прямоугольной области с координатами левого верхнего угла `x,y`, шириной — `x1`, высотой — `y1`. Начальный угол дуги — `a`, конечный угол дуги — `b`. Углы измеряются против часовой стрелки.

`g.drawString("Hello",20,200)` — выводит строку на форму с позиции (20,200); `g` — переменная класса `Graphics`.

`g.fillRect(0,0,20,100)` — рисуется закрашенный прямоугольник с указанными координатами левого верхнего угла (0,0), шириной — 20 и высотой — 100.

`g.setColor(getBackground())` — переменная графического контекста `g` получает значение цвета фона, совпадающего с цветом фона апплета (формы).

`Graphics g=getGraphics()` — получает переменную `g` графического контекста.

`if (cb.getState()) {…}` — получает значение поля `CheckBox` (`true` — выделено, `false` — нет).

`if (str.equals("Hello") {…}` — проверяет, совпадает ли строка `str` с `"Hello"`.

`Image myimg=getImage(URL url, "name.gif")` — переменная `myimg` получает значение битового образа (рисунка) с именем `name.gif` по сетевому адресу `url`. Значение `url` можно получить командой `getDocumentBase()`, которая возвращает URL HTML-документа, содержащего объявление апплета: `img=getImage(getDocumentBase(),"name.gif")`.

`init()` — метод инициализации апплета.

`Integer X=new Integer(amount)` — создает объектную целочисленную переменную `X` на основании простой целочисленной переменной `amount`.

`int bar[]={50,40,60}` — создает целочисленный массив с именем `bar` из трех чисел: 50, 40, 60.

`int l=bar.length()` — возвращает число элементов массива `bar`.

`int i=Integer.valueOf(str).intValue()` — получает целое число типа `int` из строки, например, "15".

`JOptionPane.showMessageDialog(null,"Hello");` — выводит строку в диалоговом окне.

`ke.getKeyChar()` — получает символ, соответствующий нажатой клавише.

`loader.setFile("*.dat")` — устанавливает тип отображаемых файлов для переменной `loader` типа `FileDialog` (*см.* `FileDialog`).

`loader.show()` — показывает на экране окно для выбора имени файла (*см.* `FileDialog`).

`Math.random()` — возвращает случайное число от 0 до 1.

`Math.sin((a/180)*3.14)` — вычисляет синус угла `a`, заданного в градусах.

`MenuBar mb= new MenuBar()` — создает пустую панель меню.

`Menu m1= new Menu("opera")` — создает горизонтальный пункт меню.

`m1.add(new MenuItem("Show"))` — создает в горизонтальном пункте меню `m1` вертикальный элемент с названием `Show`.

`paint(Graphics g)` — метод для рисования на апплете и форме. Графические методы для рисования предоставляет класс `Graphics`.

`prg.setValue(pos)` — установка нового положения ползунка: объектная переменная `prg` типа `JSlider`.

`repaint()` — перерисовывает форму (апплет).

`request.getParameter("tf")` — получение значения элемента формы HTML с именем `tf`.

`ResultSet rs= sq.executeQuery(sq_str)` — формируется множество записей по SQL-запросу `sq_str`. Переменная `sq` относится к типу `Statement` *(см. далее)*.

`return z` — возвращает переменную `z` из метода.

`rs.getString("Student")` — получаем значение текстового поля с именем `Student` из текущей выбранной записи из таблицы базы данных (*см.* `ResultSet`).

`rs.getInt("Group")` — получаем значение целочисленного поля с именем `Group` из текущей выбранной записи из таблицы базы данных (*см.* `ResultSet`).

`ServerSocket srv= new ServerSocket(2525)` — получение порта с номером 2525 на сервере.

`setBackground(new Color(120,200,255))` — задается цвет фона формы или апплета: 120 — красный, 200 — зеленый, 255 — голубой.

`setForeground(new Color(120,200,255))` — задает цвет текста, выводимого на форме (апплете/канве).

`setLayout(new BorderLayout())` — подключает диспетчер компоновки `BorderLayout()`.

`setLayout(new GridLayout(4,2))` — подключает диспетчер компоновки типа `GridLayout` (матрицы) с 4 строками и 2 столбцами (всего на $8 = 4 \times 2$ элементов).

`setLayout(null)` — диспетчер компоновки не используется. Пользователь сам добавляет и размещает элементы на форме.

`setMenuBar(mb)` — панель меню с именем `mb` выводится на форме.

`showStatus("OK")` — выводит строку "OK" в строку состояния апплета.

`sleep(100)` — ожидание текущим потоком 100 миллисекунд.

`Socket s = new Socket("127.0.0.1", 2525)` — получение сокета (порта) с номером 2525.

`s=srv.accept()` — сервер ожидает подключения клиента; `s` — переменная типа `Socket`.

`Statement sq=dbc.createStatement()` — создается пустая SQL-команда. `dbc` — переменная типа SQL-соединения (*см.* `Connection`).

`sq.executeQuery(strsql)` — выполняет SQL-запрос типа `SELECT` и формирует результирующий набор записей.

`sq.execute(strsql)` — выполняет SQL-запрос типа `CREATE` и `DELETE`. Результат не возвращает.

`sq.executeUpdate(strsql)` — выполняет SQL-запрос типа `INSERT` и `UPDATE`. Результат не возвращает.

`str1.indexOf("?")` — возвращает индекс (номер позиции) символа `"?"` в строке `str1`.

`str1.toUpperCase()` — строка пишется заглавными буквами

`str1.toLowerCase()` — строка пишется строчными буквами.

`String s= str1.substring(0,10)` — возвращает подстроку строки `str1` с нулевого по 10-й символ.

`String sq_str ="SELECT * FROM myt"` — пример строки запроса на языке SQL.

`System.exit(0)` — завершает приложение на основе формы.

`System.Threading.TimerCallback tc=new System.Threading.TimerCallback (showstr) t= new System.Threading.Timer(tc,null,100,350)` — создание и использование таймера в программе с последующим его запуском через 100 мс после создания и затем каждые 350 мс. Процедура, запускаемая таймером, называется `showstr`.

`System.out.println("HELLO")` — выводит на консоль строку `"HELLO"`.

`System.in.read()` — читает код (один байт) нажатой клавиши.

`textarea.append("Hello")` — добавляет в текстовую область с именем `textarea` строку `"Hello"`.

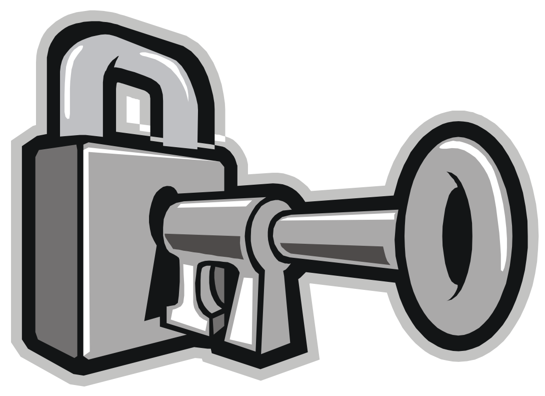
`tf.getText()` — получает строку, записанную в текстовом поле с именем `tf`.

`tf.setText("HELLO")` — вывод строки `"HELLO"` в элемент с именем `tf`. Это может быть текстовое поле или ярлык (`Label`, `JLabel`).

`thread.start()` — запуск потока с именем `thread`.

`yield()` — процесс приостанавливается и пропускает другие процессы.

`while (rs.next()) {…}` — читает очередную запись в цикле `while` из набора записей, прочитанного по команде `ResultSet rs= sq.executeQuery(sq_str)`.

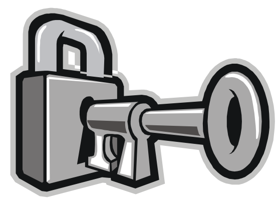


ЧАСТЬ II

C#

ГЛАВА 3. Основы программирования на языке C#

ГЛАВА 4. Практические занятия по C#



Глава 3

Основы программирования на языке C#

Введение в язык C#

Язык C# — это объектно-ориентированный язык, напоминающий C++ и Java. По аналогии с Java, C# не поддерживает указателей, использует встроенный механизм сборки мусора (garbage collection), что позволяет программисту не задействовать деструкторы и освобождает от необходимости отслеживать процесс выделения памяти. В языке нет глобальных переменных, множественного наследования и ряда других конструкций.

C# реализован на платформе .NET, ядро которой работает на уровне общего языка, известного под названием Common Language Subset (CLS, также называется Common Language Specification), и обеспечивает связь между всеми языками и библиотекой классов. Это означает, что C# имеет доступ ко многим средствам, которые можно реализовать на Visual Basic .NET, Visual C++ .NET и Visual J++ .NET).

Сходство концептуальных основ Java и C# очевидно. Однако C#, несомненно, не просто переопределенный вариант Visual J++. Многие конструкции Java вполне узнаются в C#. Рассмотрим каркас приложения C#:

```
using System;
namespace MyFirstApp
```

```
{
 class Class1
 {
  static void Main(string[] args)
  {
   // код приложения
  }
 }
}
```

Инструкция `using` аналогична инструкции `import` в Java. Она предназначена для подключения *библиотеки классов*. Объявление класса такое же, как в Java. Главная точка входа в приложение, — как и ранее, `static void Main()`. Объявление `namespace` определяет так называемое *пространство имен* — аналог объявления `Package`. Как и Java, C# не использует указателей. В последующем подобные аналогии будут усматриваться и в других аспектах. И вместе с тем C# является представителем новой платформы .NET. По аналогии с Java, платформа .NET содержит *виртуальную машину* для выполнения NET-приложений. Исходные тексты C# транслируются в промежуточный язык MSIL (Microsoft Intermediate Language), который обрабатывает виртуальная машина платформы .NET. Все это в рамках собственной реализации есть и в Java. Вместе с виртуальной машиной платформа .NET содержит библиотеки классов, определяющие свои пространства имен. Например, ими являются [3]:

- WinForms — соответствует AWT (Java);
- ASP.NET — аналог JSP;
- ADO.NET — аналог JDBC;
- компоненты .NET — аналог Java Beans и ActiveX;
- COM+ — аналог EJB (Enterprise Java Beans);
- службы Windows — нет аналога в Java.

Виртуальная машина .NET позволяет работать с языками Visual C# .NET, Visual C++ .NET, Visual Basic .NET, JScript .NET, Visual J++ .NET. Все эти языки используют одни и те же библиотеки и одинаково реализуют объекты. Например, класс, созданный в Visual Basic, можно объявить и использовать в C#.

Единицей приложения является *проект*. Проекты могут собираться в сборки.

Для работы с C# вам необходимо приобрести и инсталлировать систему Visual Studio.NET 2003. В отличие от J2SE SDK, она не поставляется бесплатно. Окно инсталлированной системы представлено на рис. 3.1.

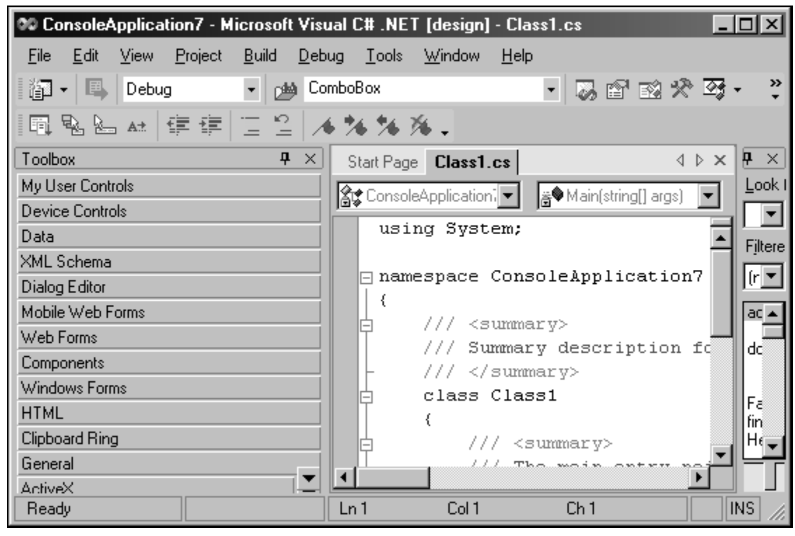


Рис. 3.1. Окно создания проектов C#

Для *создания проекта* выбираем **File | New | New Project**.

В окне из предложенных вариантов приложений нужно выбрать тип будущего проекта:

- **Windows Application** — оконное приложение Windows;
- **Console Application** — консольное приложение (без окна);
- **Control Library** — создание новых элементов;
- **WebForm Application** — создание динамических Web-страниц и др.

Начнем изучение с *консольных приложений*. Выберите тип проекта **Console Application**, затем введите его имя (в поле **Name**) и место хранения (в поле **Location**), и на экране дисплея вы увидите показанное на рис. 3.1 окно.

Заметим, что файлы проектов в C# имеют расширение csproj, а файлы классов — cs.

Простейшее приложение содержит единственный класс и единственный метод `Main()`. Изменим содержимое метода `Main()` следующим образом:

```
static void Main(string[] args)
{
 System.Console.WriteLine("Your name, please=>");
 string s=System.Console.ReadLine();
 System.Console.WriteLine("Hello, "+s);
 System.Console.ReadLine();
}
```

Выполним эту программу. Выберем пункт меню **Debug**, затем подпункт **Start**.

Команда `System.Console.WriteLine("Your name, please=>")` выводит на консоль строку приглашения. `Console` — это имя класса, содержащего методы для работы с консолью. Одним из таких методов является вывод строки — `WriteLine()`. Программирование на языке C# осуществляется на основе классов. В приложении может быть несколько классов, но только в одном классе определяется функция `Main()` — главная точка входа в приложение. Именно эта функция получает управление первой при запуске приложения. В C#, как и в Java, имеет значение регистр букв. В начале программы указываются подключаемые пакеты (с помощью директивы `using`). Каждый пакет содержит описание стандартных классов. Ключевое слово `namespace` используется для указания *пространства имен*, в пределах которого действительны имена переменных и функций, используемых программистом. Определения функций и классов заключаются в фигурные скобки — `{}`.

C# использует синтаксис языка C без указателей. Простыми типами данных в C# являются следующие:

- `short` — короткое целое от −32 768 до +32 768;
- `ushort` — беззнаковое целое от 0 до 65 535;
- `int` — то же, что и `short`;
- `uint` — то же, что и `ushort`;

- `sbyte` — целое от −127 до +127;
- `byte` — целое от 0 до 255;
- `char` — символьный тип (например, `'a'`);
- `string` — строковый тип (например, `"abc"`);
- `long` — длинное целое;
- `ulong` — беззнаковое длинное целое;
- `float` — вещественное число;
- `double` — вещественное число удвоенной точности от $+/-5.0\times10^{-324}$ до $+/-3.4\times10^{308}$.

Допускается частичное преобразование типов, например:

```
short x;
int y=2;
x= (short) y;
```

Для написания кодов используются операторы присваивания (=), проверки условия (`if () {…} else {…}`), цикла (`for`, `foreach`, `while/do while`), множественного выбора (`switch case`), переходов (`break`, `continue`, `goto`).

Оператор присваивания: `string s=Console.ReadLine().` Здесь строковая переменная `s` получает значение, вводимое с клавиатуры.

Продемонстрируем работу *оператора множественного выбора* на следующем фрагменте кода, предполагая наличие команды `using System` в начале программы.

```
string s=Console.ReadLine(); // Имя System можно опустить,
// т.к. соответствующий пакет подключен командой using System
switch (s)
{
 case "apple":
  Console.WriteLine("Fruit");
 break;
 case  "cherry":
  Console.WriteLine("Fruit");
 break;
```

```
 case "stone" :
  Console.WriteLine("Not a fruit");
 break;
 default :
  Console.WriteLine("Think about it Yourself");
 break;
}
```

В этом фрагменте сначала выполняется чтение строки `s` с клавиатуры. Оператор `switch` осуществляет разбор вариантов. Каждый проверяемый вариант начинается словом `case`, за которым следует значение переменной, соответствующее этому варианту. Каждый вариант, вообще говоря, следует завершать командой `break` для выхода за пределы конструкции `switch`. Если оператор `break` не указан, то выполняется переход для проверки следующего варианта `case` и т. д. Вариант `default` соответствует значениям проверяемой переменной или выражения, не указанным в вариантах `case`. Реализуем тот же фрагмент с помощью оператора `if`:

```
string s=Console.ReadLine();
if (s=="apple")
 Console.WriteLine("Fruit");
else
if (s== "cherry")
 Console.WriteLine("Fruit");
else
 if (s== "stone" )
  Console.WriteLine("Not a fruit");
 else
  Console.WriteLine("Think about it Yourself");
}
```

Оператор `if` сначала проверяет условие, указанное в круглых скобках, например, `if(s== "stone") {...}`.

Если условие истинно, выполняется блок команд, записанных в фигурных скобках. Если в блоке только одна команда, то фигурные скобки указывать не обязательно. В случае ложности

условия осуществляется выполнение блока команд, записанных непосредственно за ключевым словом `else`.

Обратимся к циклам. Приведем фрагмент кода, который осуществляет ввод строк до тех пор, пока не будет введена строка `"exit"`.

```
string s;
do
{
 string s=Console.ReadLine();
 if (s=="apple")
  Console.WriteLine("Fruit");
 else
  if (s== "cherry")
   Console.WriteLine("Fruit");
  else
   if (s== "stone" )
    Console.WriteLine("Not a fruit");
   else
    Console.WriteLine("Think about it Yourself");
}
while (!(s=="exit");
```

Выполняется *тело цикла* `do` до тех пор, пока истинно выражение, записанное в условии `while`. Символ `"!"` соответствует логическому оператору отрицания. Два знака равенства используются для проверки на совпадение. Другая форма цикла `while` имеет следующий вид:

```
while (условие)
{
 тело цикла
}
```

В этом варианте цикл выполняется до тех пор, пока истинно условие (цикл может не выполниться ни разу, если при входе в него условие цикла ложно).

Цикл `for` позволяет использовать переменную цикла и ограничивать число итераций фиксированным значением, например:

```
for (int i=1; Math.sin(i)<0.5; i++)
{
 Console.WriteLine(""+Math.sin(i));
}
```

Приведенный здесь цикл выполняется, начиная со значения `i=1`, до тех пор, пока `sin(i)<0.5`. После завершения очередной итерации цикла и вывода значения синуса `i` на консоль переменная цикла `i` увеличивается на 1 с помощью оператора `i++`. В записи условий цикла `for` указывают: начальное значение переменной цикла (`int i=1`), условие выполнения очередной итерации и оператор, выполняющийся по завершении очередной итерации (`i++`). Все варианты записи циклов взаимозаменяемы. Для преждевременного ***выхода из цикла*** используется команда `break`. Для преждевременного перехода к очередной итерации используют команду `continue`.

Приведем пример:

```
for (;;) // Бесконечный цикл без каких-либо условий
{
 string s=Console.ReadLine(); // Читаем строку с клавиатуры
 if (s=="apple") // Выход из цикла по вводу "apple"
  break;
 else
  continue; // Продолжение итераций
}
```

Цикл `foreach` будет рассмотрен далее. Команду `goto` мы по возможности не используем, солидаризуясь с разработчиками Java. Сравнивая коды, написанные на C# и Java, видим, что C# "более либерален" в отношении использования исключительных ситуаций и охраны (конструкции `try ... catch`). Читателю снова рекомендуем прочитать *разд. "Обработка исключительных ситуаций" главы 1.* Здесь ограничимся следующим типичным примером. Запишем код, в котором выполним деление на 0, но при этом перехватим исключительную ситуацию так, как показано в этом примере:

```
try
 {
  int z=0;
  float d= 15/z;
 }
catch (System.Exception e)
 {
  Console.WriteLine("Error in Your program");
}
```

Как и следовало ожидать, получим сообщение: `Error in Your program`. При возникновении ошибки внутри блока `try` проверяется, какой тип ошибок перехватывает блок `catch`. В нашем случае блок `catch` перехватывает ошибки типа `System.Exception`. Этот тип ошибок является наиболее общим. Если тип ошибок, указанный в блоке `catch`, не соответствует типу возникшей ошибки, программа завершается аварийно. Для одного блока `try` можно указать несколько различных блоков `catch`, перехватывающих разные типы ошибок. После обработки блока `catch` программа продолжает выполняться с оператора, непосредственно следующего за этим блоком `catch`. В блоке `catch` указывается объектная переменная `e` класса `System.Exception`. Эта переменная имеет несколько полезных свойств, которые доступны в приложении:

- `Message` — содержит текст ошибки;
- `TargetSite` — содержит имя метода, вызвавшего ошибку;
- `Source` — содержит имя приложения, вызвавшего ошибку,

и др.

Наконец, можно искусственно создать ошибочную ситуацию, например, таким образом:

```
try
 {
  ArithmeticException e1=new ArithmeticException(); // Создаем
                                                    // исключение
  throw e1; // Вызываем исключение искусственно
 }
catch(System.Exception e) // Перехватываем исключение
```

```
{
 Console.WriteLine(e.Message);
 Console.ReadLine();
}
```

Исключение вызывается командой `throw`.

Более подробное знакомство с вопросом об исключительных ситуациях и их обработке можно получить из [9].

Платформа C# для Java-программистов

Полезной в этом смысле является книга [3]. Читателю, владеющему основами Java, изложенными в первой части книги, в действительности овладение C# должно быть интересно. Новым является наличие визуальной среды проектирования приложений. Здесь, несомненно, чувствуется влияние Delphi и Visual Basic. Программисту удобно пользоваться окном элементов управления, "находящимся под рукой" (рис. 3.2).

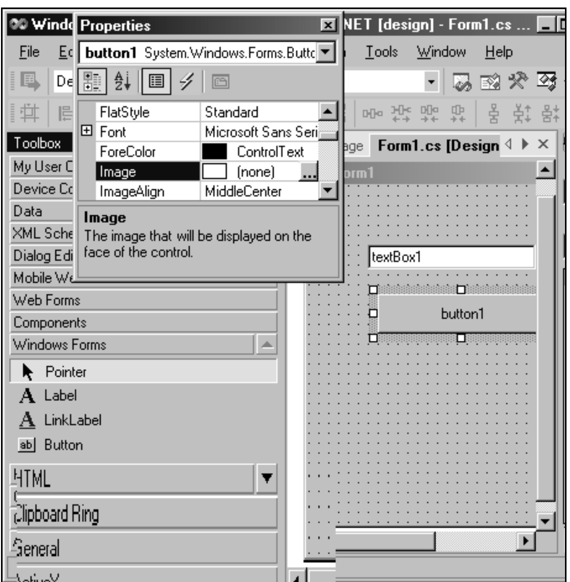


Рис. 3.2. Окно элементов управления

В окне элементов управления вы увидите закладки:

- **Data**
- **XML Scheme**
- **WebForms**
- **Components**
- **Windows Forms**
- **HTML**

и др.

Чтобы на форме поместить кнопку, текстовое поле или какой-нибудь другой элемент, выберите закладку **Windows Forms**, затем щелчком левой кнопки мыши на нужном элементе выделите его и прорисуйте мышью контуры элемента на форме, не отпуская при этом левую кнопку мыши. В Java создание и размещение элементов требовалось выполнять программно. *Свойства элементов* (окно **Properties** на рис. 3.2) доступны через пункт **Properties** контекстного меню, которое можно вызвать щелчком правой кнопки мыши на элементе. В окне свойств можно добраться и до обработчика событий, связанных с элементом (рис. 3.3), раскрыв окно событий щелчком мыши на пиктограмме с изображенным на ней значком молнии в окне **Properties**.

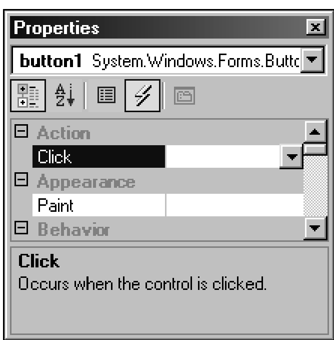


Рис. 3.3. Окно событий элемента

В материале, посвященном C#, мы обратимся к более подробному рассмотрению программирования приложений на основе форм. После установки Visual Studio .NET доступна система

документации по .NET, которую можно активизировать следующим образом: **Пуск | Программы | Visual Studio .NET | Visual Studio .NET Documentation**. *Справочная система* содержит сведения по языкам, входящим в .NET, и включает в себя описания классов и примеры программ. При работе с C# иногда будет требоваться использование командной строки MS-DOS — например, при создании новых компонентов и dll-функций. Эта строка (command prompt) доступна через **Пуск | Программы | Visual Studio .NET | Visual Studio .NET Tools | Command Prompt**. На этот момент следует обратить внимание.

Для запуска приложений следует выбрать пункт главного меню **Debug**, а затем подпункт **Start** (или **Start without debugging**). Если в приложении имеются ошибки, то они выводятся в окне, расположенном в нижней части экрана (рис. 3.4).

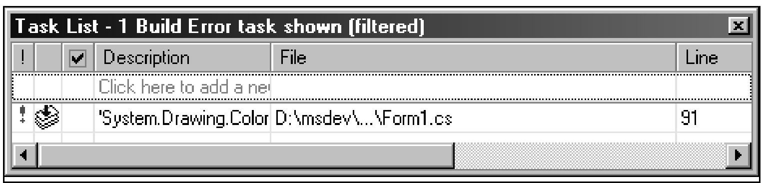


Рис. 3.4. Окно для вывода сообщений об ошибках

Если щелкнуть дважды мышью на строке сообщения об ошибке, то система отобразит то место в программе, где данная ошибка имела место.

Большим достоинством системы создания приложений .NET является встроенная система отладки. Щелкните мышью слева в строке, где вы хотите поместить точку прерывания. Эта строка будет выделена большим красным кружком (рис. 3.5).

После запуска приложения, когда программа дойдет до помеченного места, она приостановится и в окне отладки можно будет увидеть все значения переменных, полученные к этому моменту. Пошаговое выполнение программы реализуется с помощью клавиши <F5>. Если требуется продолжить автоматическое выполнение программы, следует выполнить команду меню **Debug | Continue**.

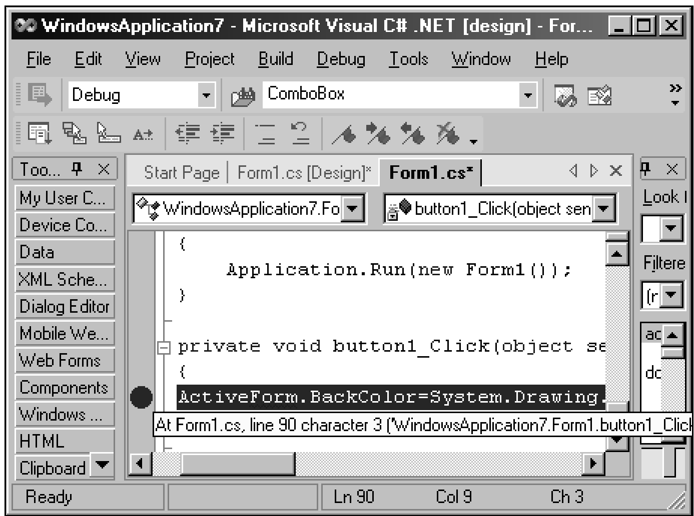


Рис. 3.5. Указание места прерывания для отладки

При открытии ранее сохраненного проекта вы можете не увидеть на экране его формы и текста приложения. В этом случае выполните команду меню **View | Solution Explorer**, а затем в окне структуры проекта щелкните дважды мышью на имени формы.

Итак, после программирования на Java следует познакомиться со средой создания проектов .NET и визуальным интерфейсом этой среды. Очень неплохо, если у вас есть опыт программирования на Delphi или Visual Basic. Однако изучение среды не настолько серьезное препятствие, чтобы остановиться. Скорее всего, постепенно накапливая опыт, вы просто перестанете замечать эти начальные трудности.

Программирование "без классов"

Программирование "без классов" — это программирование без использования объектов, порождаемых из классов. Как правило, это программирование консольных приложений. Фактически мы достаточно подробно останавливались на нем в одноименном разделе, посвященном языку Java; *см. разд. "Программирование без классов" и "Классы" главы 1.* Рассмотрим снова

приложение для чтения двух целых чисел и нахождения их общего делителя. Это приложение представлено в листинге 3.1.

Листинг 3.1. Консольное приложение для нахождения наибольшего общего делителя

```
using System;
namespace ConsoleApplication5
{
 /// <summary>
 /// Summary description for Class1
 /// </summary>
 class Class1
 {
  static int A;
  static int B;
  static string s;
  /// <summary>
  /// The main entry point for the application.
  /// </summary>
  [STAThread]
  static void Main(string[] args)
  {
   //
   // TODO: Add code to start application here
   //
   Console.WriteLine("Vvedi czislo A");
   s=Console.ReadLine();
   A=Convert.ToInt32(s);
   Console.WriteLine("Vvedi czislo B");
   s=Console.ReadLine();
   B=Convert.ToInt32(s);
   while (A!=B)
   {
    if (A>B)
```

```
     A=A-B;
    else
     if (B>A)
      B=B-A;
     else
      break;
   }
   Console.WriteLine("Common Divisor="+A);
   Console.ReadLine();
  }
 }
}
```

Как видим, чтение строки с клавиатуры выполняется командой `Console.ReadLine()`. Полученную строку далее преобразуем в целое число командой `Convert.IntTo32()`. Цикл `while(A !=B) {}` выполняется до тех пор, пока переменная `A` не равна переменной `B`. Теперь заметим, что язык C#, как и Java, использует тот же синтаксис для цикла `while`, так что остается лишь указать, что без классов (условно, конечно, так как `Console` и `Convert` — это классы) программируют лишь отчаянные консерваторы (а мы не из их числа).

В C#, в отличие от Java, можно использовать структуры. *Структура* — это тот же класс, однако структуры не используют механизм наследования. Следующий код (листинг 3.2) демонстрирует использование структуры по имени `Point`. В этой структуре объявлены два поля `x` и `y`. Определен конструктор и метод сложения координат двух точек.

Листинг 3.2. Консольное приложение, использующее структуру `Point`

```
using System;
namespace StructureApp
{
 /// <summary>
 /// Summary description for Class1.
```

```
/// </summary>
/// <summary>
/// Example of declaring and using a struct
/// </summary>
public class StructExample
{
 struct Point  // Объявляется структура Point
 {
  public int x; // Объявляются поля x и y
  public int y;
  public Point(int x, int y) // Объявляется конструктор
  {
   this.x = x;
   this.y = y;
  }
  public Point Add(Point pt) // Сложение двух точек
                             // покоординатно
  {
   Point newPt;
   newPt.x = x + pt.x;
   newPt.y = y + pt.y;
   return newPt;
  }
 }
 static void Main(string[] args)
 {
  Point pt1 = new Point(0, 1); // Создаем две точки pt1,pt2
  Point pt2 = new Point(2, 3);
  Point pt3;
  pt3 = pt1.Add(pt2); // Третья точка получается как сумма
                      // первых
  Console.WriteLine("The new point coordinates are "+pt3.x+":
  "+ pt3.y);
  Console.ReadLine();
```

```
  }
 }
}
```

Этот же пример можно вполне реализовать на основе классов:

```
using System;
namespace StructureApp
{
 /// <summary>
 /// Summary description for Class1.
 /// </summary>
 /// <summary>
 /// Example of declaring and using a struct
 /// </summary>
 public class ClassExample
  {
   class Point  // Объявляется класс Point внутри класса
                // ClassExample
   {
    public int x; // Объявляются поля x и y
    public int y;
    public Point(int x, int y) // Объявляется конструктор
     this.x = x;
     this.y = y;
    }
   public Point Add(Point pt) // Сложение двух точек
                              // покоординатно
   {
    Point newPt=new Point(0,0);
    newPt.x = x + pt.x;
    newPt.y = y + pt.y;
    return newPt;
   }
```

```
  }
  static void Main(string[] args)
  {
   Point pt1 = new Point(0, 1); // Создаем две точки pt1,pt2
   Point pt2 = new Point(2, 3);
   Point pt3;
   pt3 = pt1.Add(pt2); // Третья точка получается как сумма
                       // первых двух
   Console.WriteLine("The new point coordinates are "+pt3.x+":
   "+ pt3.y);
   Console.ReadLine();
  }
 }
}
```

Отличий практически не видим. Однако имеем в виду, что классы в сравнении со структурами являются более мощными и эффективными средствами.

Использование классов

Прежде всего, прочитайте *разд. "Классы"* теоретического введения в Java до того места, с которого начинается описание, специфическое для Java.

Как правило, классы объявляются независимо друг от друга, что позволяет использовать механизм наследования. Приведем следующий пример использования двух классов в приложении (листинг 3.3).

Листинг 3.3. Консольное приложение, работающее с двумя классами

```
using System;
namespace Test
{
 class Worker
```

```
  {
   public int age=0;
   public string name;
  }
 class Test
 {
  static void Main()
  {
   Worker w1= new Worker();
   w1.age=32;
   w1.name="Petrov";
   Console.WriteLine(w1.name+":"+w1.age);
   Console.ReadLine();
  }
 }
}
```

Модификатор доступа `public` означает доступность переменных и методов данного класса из других классов. Имеются также другие модификаторы:

- `protected` — переменные и методы, объявленные с этим модификатором, доступны в пределах данного класса и из всех дочерних классов;
- `private` — переменные и методы с этим модификатором доступны в пределах только данного класса;
- `internal` — переменные и методы данного класса доступны в пределах единицы компиляции (приложения);
- `Published` — позволяет объявлять поле в окне свойств (**Properties**) объекта.

В C# добавлена возможность включать в классы свойства, доступные посредством методов `get` и `put`. Достаточно часто имеет место необходимость получать доступ к закрытым (`private`) членам класса из других классов. В C# такая возможность как

раз и реализована посредством открытых свойств и методов `get` и `put`.

В языке C# можно использовать свойства, которые позволяют получать значения полей при помощи метода `get` и записывать значения полей при помощи метода `put`. Рассмотрим следующее *консольное приложение* (листинг 3.4).

Листинг 3.4. Доступ к закрытым членам через свойства `get/put`

```
using System;
namespace properties
{
 /// <summary>
 /// Summary description for Class1.
 /// </summary>
 public  class Car:MoreCar // Наследуем свойства и поля класса
                          // MoreCar
 {
  /// <summary>
  /// The main entry point for the application.
  /// </summary>
  [STAThread]
  static void Main(string[] args)
  {
   Car onecar= new Car(); // Создаем объект с помощью
                          // конструктора
   onecar.Marka="Ford"; // Заносим значение в открытое поле
   // Marka, которое также устанавливается в закрытом
   // (private) поле marka
   onecar.price=120; // Этот оператор вызовет ошибку,
   // поскольку совершается попытка установить значение поля
   // типа private в дочерних классах, что недопустимо
   Console.WriteLine(onecar.Marka+" "+onecar.price);
```

```
   Console.ReadLine();
   //
   // TODO: Add code to start application here
   //
  }
 }
}
public class MoreCar
// Определяется класс с защищенными полями marka и price
{
 private string marka;
 public string Marka
 {
  get
  {
   return marka;
  }  // Возвращает значение приватного поля marka
  set
  {
   marka=value;
  } // Устанавливает значение приватного поля marka
    // через переменную value, значение которой определяется
    // системой
 }
 private int price;
}
```

Напомним, что поля и методы с модификатором доступа `private` не доступны уже в дочерних классах. Однако хорошим тоном программирования является не использование модификатора доступа `public`, а именно доступ к полям `private` посредством методов `get`, `put` для открытых свойств. Для исправления

ошибки наше приложение следует переписать как показано в листинге 3.5.

Листинг 3.5. Устранение ошибки доступа к закрытому члену класса

```
using System;
namespace properties
{
 /// <summary>
 /// Summary description for Class1.
 /// </summary>
 public class Car:MoreCar // Наследуем свойства и поля класса
                          // MoreCar
 {
  /// <summary>
  /// The main entry point for the application.
  /// </summary>
  [STAThread]
  static void Main(string[] args)
  {
   Car onecar= new Car(); // Создаем объект с помощью
                          // конструктора
   onecar.Marka="Ford"; // Заносим значение в открытое
   // свойство Marka, которое также устанавливается в закрытом
   // (private) поле marka
   onecar.Price=120; // Этот оператор не вызовет ошибки
   Console.WriteLine(onecar.Marka+" "+onecar.Price);
   Console.ReadLine();
   //
   // TODO: Add code to start application here
   //
  }
 }
}
public class MoreCar
```

```
{
 private string marka;
 public string Marka
 {
  get
   {
    return marka;
   }
  set
  {
   marka=value;
  }
 }

 private int price;
 public int Price // Использование открытого свойства Price
 {
 get
  { return price;}
 set
  {price=value;}
 }
}
```

Открытое свойство, например, `Marka`, объявлено вместе с методами `get` и `put` так:

```
public string Marka
 {
 get
  { return marka;}
 set
  {marka=value;}
}
```

Метод `get` возвращает значение закрытого поля `marka` командой `return marka`; метод `set` устанавливает значение закрытого поля

`marka` оператором `marka=value`. Здесь `value` — стандартная системная переменная для установки значения свойства

Для классов используется механизм наследования. Указать в программе факт наследования можно с помощью конструкции `Class1: Class2` (класс `Class1` наследует класс `Class2`). Однако родителем данного класса может быть *только один* класс (отличие от C++). Дочерние классы наследуют свойства и методы родителей. Приведем следующий пример (листинг 3.6).

Листинг 3.6. Пример приложения с несколькими классами

```
using System;
 namespace Test
 {
  class Worker
  {
   public int age=0;
   public void setAge(int age)
   {
    if ((age>0) && (age<100))
     this.age=age;
    else
     this.age=0;
   }

   public int getAge()
   {
    return this.age;
   }
  }

  class Boss:Worker
  {
   public int numofworkers;
   public new void setAge(int age)
```

```
   {
    if((age>0)&&(age<45)) // Логическая проверка двух условий
     this.age=age;
    else
     this.age=0;
   }
  }

  class Test
  {
   static void Main()
   {
    Worker w1= new Worker();
    Boss b1=new Boss();
    w1.setAge(50);
    b1.setAge(50);
    Console.WriteLine("Рабочий: "+w1.age);
    Console.WriteLine("Босс:  "+b1.age);
    Console.ReadLine();
   }
  }
 }
```

В данном приложении используются три класса. Класс `Boss` наследует класс `Worker`. Это делается так:

```
class Boss:Worker
{}
```

Из этого факта следует то, что в класс `Boss` попадают члены-переменные класса `Worker`:

```
public int age=0;
```

а также его методы:

```
public void setAge(int age)
public int getAge()
```

Однако в самом классе `Boss` метод `setAge()` переопределен так:

```
public new void setAge(int age)
```

```
{
 if((age>0)&&(age<45))
  this.age=age;
 else
  this.age=0;
}
```

Метод `getAge()` остался без изменений. Теперь уже в методе `Main()` класса `Test` строки

```
w1.setAge(50);
b1.setAge(50);
```

будут обращаться к двум различным методам.

Рассмотрим особый вид классов — интерфейсы. *Интерфейс* — это класс, в котором объявлены методы и свойства с методами `get` и `put`, однако реализации методов в интерфейсах отсутствуют. Класс, который наследует интерфейс, должен реализовать и методы, объявленные в интерфейсе. От интерфейса нельзя строить объекты. Интерфейс является разновидностью абстрактного класса. *Абстрактный класс* содержит хотя бы одно объявление метода, для которого не приведена реализация. Такие методы называются *виртуальными*. От абстрактного класса нельзя строить объекты. Перейдем к рассмотрению примеров интерфейсов (листинг 3.7).

Листинг 3.7. Пример приложения с использованием интерфейсов

```
using System;
namespace ConsoleApplicationInter
{
 interface IParentInterfaceOne // Первый интерфейс
 {
  void ParentInterfaceOneMethod(); // Объявление метода без
                                   // реализации
 }

 interface IMyInterfaceTwo : IParentInterfaceOne // Второй
                                  // интерфейс наследует первый
```

```
 {
  void MethodToImplementTwo();
 }
 class InterfaceImplementerEx : IMyInterfaceTwo
 // Класс наследует интерфейс
 {
  static void Main()
  {
   InterfaceImplementerEx iImp = new InterfaceImplementerEx();
   Console.WriteLine("REALLY");
   Console.ReadLine();
   iImp.MethodToImplementTwo();
   iImp.ParentInterfaceOneMethod();
  }

  public void MethodToImplementTwo() // Реализация метода
                                     // интерфейса
  {
   Console.WriteLine("MethodToImplementTwo() called.");
   string s=Console.ReadLine();
   Console.WriteLine(s);
  }
  public void ParentInterfaceOneMethod() // Реализация метода
                                         // интерфейса
  {
   Console.WriteLine("ParentInterfaceOneMethod() called.");
   string s=Console.ReadLine();
   Console.WriteLine(s);
  }
 }
}
```

В этом примере объявлены два интерфейса: `interface IparentInterfaceOne` и `interface IMyInterfaceTwo: IparentInterfaceOne`. Второй из этих интерфейсов наследует первый (используется двоеточие). Это значит, что объявления методов

первого интерфейса переносятся во второй интерфейс в силу *наследования*. В данном приложении объявлен класс `InterfaceImplementerEx`, который наследует второй интерфейс `ImyInterfaceTwo`. В классе `InterfaceImplementerEx` как раз и необходимо реализовать методы, объявленные или унаследованные интерфейсом `ImyInterfaceTwo`. Мы видим реализации обоих методов:

```
public void MethodToImplementTwo()
{
 Console.WriteLine("MethodToImplementTwo() called.");
 string s=Console.ReadLine();
 Console.WriteLine(s);
}
public void ParentInterfaceOneMethod()
{
 Console.WriteLine("ParentInterfaceOneMethod() called.");
 string s=Console.ReadLine();
 Console.WriteLine(s);
}
```

Итак, если класс наследует интерфейс, то в нем должна быть представлена реализация методов интерфейса. Рассмотрим второй пример (листинг 3.8).

Листинг 3.8. Пример с неопределенным методом `get`

```
using System;
namespace ConsoleApplicationInter
{
 interface FirstInterface // Объявлен первый интерфейс
 {
  string Stroka // Объявлено открытое свойство с методом get
  {
   get;
  }
  void FirstMethod(); //Объявлен метод
 }
```

```
interface SecondInterface : FirstInterface // Объявлен второй
                                            // интерфейс
// Он наследует метод FirstMethod() первого интерфейса
{
 void SecondMethod();
}

class InterfaceBased : SecondInterface
// Наследование интерфейса с двумя методами и открытым
// свойством Stroka
{
 private string str="REALLY";
 static void Main()
 {
  InterfaceBased iImp = new InterfaceBased();
  Console.WriteLine(iImp.Stroka); // Обращение к закрытому
                                  // полю
  // через метод get
  Console.ReadLine();
  iImp.FirstMethod();
  iImp.SecondMethod();
 }

 public string Stroka // Реализация метода get
 {
  get
   {
    return str;
   }
 }

 public void FirstMethod() // Реализация метода FirstMethod()
 {
  Console.WriteLine("First Method() is called.");
  Console.ReadLine();
```

```
  }

  public void SecondMethod() // Реализация метода SecondMethod
  {
   Console.WriteLine("SecondMethodMethod() called.");
   Console.ReadLine();
  }
 }
}
```

Реализация первого метода FirstMethod() такова:

```
public void FirstMethod()
{
 Console.WriteLine("First Method() is called.");
 Console.ReadLine();
}
```

При запуске он объявит себя строкой:

```
First Method() is called
```

При запуске второго метода будет выведена строка:

```
SecondMethodMethod() called
```

Интерфейсы мы часто использовали в Java для подключения обработчиков событий. Их более общее назначение состоит в том, чтобы объявлять некоторую функционально законченную часть проекта, содержащую набор методов для реализации в конкретных условиях.

Использование подпрограмм

Примером использования подпрограмм может служить следующий:

```
using System;
class Ex
{
 static void Main()
```

```
 {
  Console.WriteLine("Вызываем метод Jump …");
  Jump();
 }

 static void Jump()
 {
  Console.WriteLine("Привет из Jump …");
  Console.ReadLine();
 }
}
```

Когда перед именем метода (функции) не указывается имя объекта, то предполагается, что используется метод класса, где данный метод объявлен. Такой метод (функция) называется *статическим*. При записи такого метода нужно указывать ключевое слово `static`. Правила здесь такие же, как и в Java. Ключевое слово `static` указывает на принадлежность поля или метода классу, а не объекту. Если требуется выйти из метода заблаговременно, то следует использовать оператор возврата `return`.

В метод в общем случае можно передавать параметры или получать параметры. Следующий пример демонстрирует, как это делается:

```
using System;
class Color
{
 public Color()
 {
  this.red=255;
  this.green=0;
  this.blue=255;
 }

 protected int red;
 protected int green;
 protected int blue;
```

```
  public void GetColors(out int red, out int green, out int
  blue)
 {
  red=this.red;
  green=this.green;
  blue=this.blue;
 }

 static void Main()
  {
   Color c= new Color();
   int red;
   int green;
   int blue;
   c.GetColors(out red, out green, out blue);
   Console.WriteLine("red color is "+red);
   Console.ReadLine();
  }
 }
}
```

В этом примере объявлен единственный класс `Color`. В классе `Color` объявлены: конструктор `public Color()` (конструктор можно не задавать), переменные:

```
protected int red;
protected int green;
protected int blue;
```

а также два метода `Main()` и `GetColors()`. По своей структуре программа имеет прямую аналогию с программой Java. *Конструктор* имеет то же имя, что и класс, в котором он определен. Задача конструктора — инициализировать переменные класса:

```
{
 this.red=255;
 this.green=0;
```

```
 this.blue=255;
}
```

В методе `Main()` выполняем создание объекта класса `Color`:

```
Color c= new Color();
```

Затем вызываем метод `GetColors()`:

```
c.GetColors(out red, out green, out blue);
```

При вызове метода `GetColors()` в скобках перечисляем аргументы и указываем для них *модификатор* `out` (или `in`). Модификатор `out` соответствует выходному параметру, т. е. такому, который получает значение в методе; модификатор `in` соответствует входному параметру (такой параметр имеет значение и не изменяет его при выходе). Можно указывать тип составной `in out` (такой параметр передается в метод и в этом методе его значение изменяется). Остается лишь обратить внимание на то, как объявляется метод `GetColors()` в классе:

```
public void GetColors(out int red, out int green, out int
blue)
```

Здесь указывается список параметров метода, их типы и модификаторы. Вместо модификатора `out` можно использовать модификатор `ref`, но при условии, что передаваемый с данным модификатором параметр получил значение (инициализирован).

В C# реализован механизм ссылок на методы, использующий так называемые *делегаты*. Делегат можно объявить, например, следующим образом:

```
public delegate double DelegateFunction (double x, double y);
```

Теперь имя метода `DelegateFunction()` можно связать с любым методом, который возвращает значение типа `double` и использует два аргумента также типа `double`. Иллюстрацию использования делегата можно дать на следующем примере (листинг 3.9).

Листинг 3.9. Использование делегатов

```
using System;
public delegate double DelegateFunction (double x, double y);
public class Class1
```

```
{
 public static double Distance(double x, double y)
 {
  return Math.Pow((Math.Pow(x,2)+Math.Pow(y,2)),0.5);
 }
 public static double Sum(double x, double y)
 {
  return x+y;
 }
}
class Example
{
 public static void Main()
 {
  double a=10;
  double b=20;
  DelegateFunction df= new DelegateFunction(Class1.Distance);
  Console.WriteLine("FirstDelegate returned "+ df(a,b));
  df= new DelegateFunction(Class1.Sum);
  Console.WriteLine("SecondDelegate returned "+ df(a,b));
  Console.ReadLine();
  }
 }
}
```

В данном примере объявлен делегат:

```
public delegate double DelegateFunction (double x, double
y);
```

В классе `Class1` также представлены реализации методов `Distance()` и `Sum()`. Видим, что эти методы возвращают значение типа `double` и используют два аргумента типа `double`. В таком случае говорят, что оба метода имеют одинаковую *сигнатуру*, которая совпадает с сигнатурой объявленного делегата. Этого обстоятельст-

ва достаточно для того, чтобы связать делегат сначала с одним методом, затем — со вторым. Первое связывание выполняется так:

```
DelegateFunction df= new DelegateFunction
(Class1.Distance);
```

Здесь в конструктор класса делегата передается имя первого метода. Далее реализуется вывод на консоль значения метода `Distance()`, хотя обращение выполняется к делегату:

```
Console.WriteLine("FirstDelegate returned "+ df(a,b));
```

По аналогии выполняется связывание делегата со вторым методом и вывод значения, получаемого методом `Sum()`:

```
df= new DelegateFunction(Class1.Sum);
Console.WriteLine("SecondDelegate returned "+ df(a,b));
```

Заметим, что методы, связываемые с делегатом, объявлены как статические, поэтому и в конструкторе класса делегата указывается имя класса, а затем через точку — имя метода.

Объявление массивов

Примеры объявлений массивов таковы:

```
int [] k;  // Объявлен массив k, но не создан
k=new int[10]; // Создан массив из 10 целых чисел, но не
               // заполнен
k[0]=1; // Первый элемент массива получил значение 0.
        // Нумерация с 0
int [] z={5,6,7}; // Массив z объявлен и инициализирован
                  // тремя значениями
int [] m =new int[5]; // Массив m из 5 элементов объявлен
                      // и создан
int [,] w = new int[4,4]; // Двумерный массив w размером
                          // 4*4 объявлен и создан
```

Рассмотрим небольшой пример работы с массивом. Пусть требуется инвертировать одномерный *массив*, в котором непосредственно заданы значения элементов. Приведем программу, которая это делает (листинг 3.10).

Листинг 3.10. Инвертирование массива

```
using System;
namespace ArrayEx
{
 /// Summary description for Class1.
 /// </summary>
 class Class1
 {
  /// <summary>
  /// The main entry point for the application.
  /// </summary>
  [STAThread]
  static void Main(string[] args)
  {
   int [] ar1={1,2,3,4,5,6,7,8,9}; // Объявляем массив,
                                   // подлежащий инверсии
   int []  ar2=      new int[9]; // В этом массиве записываем
                                 // результат
   String s="";
   for (int i=ar1.Length;i>0;i--)
   {
    ar2[9-i]=ar1[i-1]; // Последний элемент массива ar1
                       // становится первым в ar2
    s+=" "+ar2[9-i]; // Строка для вывода результата
                     // последовательно
                     // заполняется элементами массива ar2
   }
   Console.WriteLine(s);
   Console.ReadLine();
   //
   // TODO: Add code to start application here
  }
 }
}
```

При работе с массивами часто возникает необходимость получать в цикле доступ к элементам массива. Для этой цели используют цикл типа `foreach`. Такого цикла в Java нет. Следующий пример (листинг 3.11) осуществляет простой вывод на консоль строк, являющихся элементами строкового массива.

Листинг 3.11. Вывод элементов строкового массива

```
using System;
namespace foreachex
{
 /// <summary>
 /// Summary description for Class1.
 /// </summary>
 class Class1
 {
  static String [] strar={"one","two","three"}; // Объявляем и
                                        // создаем массив строк
  /// <summary>
  /// The main entry point for the application.
  /// </summary>
  [STAThread]
  static void Main(string[] args)
  {
   foreach( String s in strar) // Организуем перебор элементов
                               // массива с помощью цикла
                               // foreach
   {
    Console.WriteLine("The string is: "+s);
    // На каждой итерации цикла получаем доступ
    // к очередной строке массива
   }
  Console.ReadLine();
  //
```

```
  // TODO: Add code to start application here
  //
  }
 }
}
```

Обратим внимание на то, что массивы передаются по ссылке. Рассмотрим пример:

```
int [] k= new int[2];
k[0]=1;
k[1]=2;
int []kk=k;
kk[0]=10;

Console.WriteLine("The kk[0]="+kk[0]+" k[0]="+ k[0]);
Console.ReadLine();
```

В этом примере `kk` и `k` суть один и тот же массив, поскольку оператор присваивания `int []kk=k` присвоит переменной `kk` адрес переменной `k`, следовательно, будут выведены одни и те же значения. По ссылке также передаются объектные переменные (экземпляры классов).

Это значит, что если `ob1` и `ob2` представляют две объектные переменные, то присваивание `ob1=ob2` присваивает объектной переменной `ob1` адрес объектной переменной `ob2`.

Рассмотрим теперь создание массива объектов. Как и в языке Java, процесс создания отдельного экземпляра выполняется в два этапа: сначала создаем массив с помощью конструктора, а затем создаем экземпляр класса, например:

```
Point [] masp= new Point[2];
masp[0]=new Point(3,14);
masp[1]=new Point(5,6);
```

Здесь создан массив структур `Point` *(см. разд. "Программирование «без классов»")* и выполнена инициализация его элементов.

Работа с файлами

Рассмотрим, как организовать работу с файлами, на примерах. В первом из них обратимся к низкоуровневой работе с файлами (листинг 3.12).

Листинг 3.12. Работа с файлами

```
using System;
using System.IO;
using System.Text;
class Test
{
 public static void Main()
 {
  string path=@"c:\my.txt";
  if (File.Exists(path))
   File.Delete(path);
  FileStream fs= File.Create(path,1024);
  Byte[] info =new UTF8Encoding(true).GetBytes("Hello");
  fs.Write(info,0,info.Length);
  fs.Close();
  StreamReader sr= File.OpenText(path);
  string s="";
  while((s=sr.ReadLine()) != null)
   Console.WriteLine(s);
  Console.ReadLine();
 }
}
```

В этом примере сначала проверяем, существует файл или нет:

```
string path=@"c:\my.txt"; // Задает путь к файлу;
// @ используется, чтобы подавить действие спецсимволов,
// начинающихся с косой черты \
if  (File.Exists(path)) // Проверка существования файла с
                         // с путем path
 File.Delete(path); // Если файл существует, то его удаляем
```

Затем создается файловая переменная для низкоуровневой записи в файл:

```
FileStream fs= File.Create(path,1024);
```

Класс `FileStream` является низкоуровневым классом для работы с байтами. Параметр 1024 задает размер создаваемого файла в байтах. Далее формируем массив байтов и записываем его в файл:

```
Byte[] info =new UTF8Encoding(true).GetBytes("Hello");
fs.Write(info,0,info.Length); // Производим запись
                              // массива байтов info
```

Система кодировки символов UTF8Encoding является альтернативной для ASCII. Она использует для представления каждого символа 16 бит (ASCII — соответственно 8 бит), т. е. количество представляемых символов в UTF8Encoding намного превосходит количество символов в ASCII.

Далее выполняем чтение на консоль сохраненных данных. Для этого создаем файловую переменную для ввода:

```
StreamReader sr= File.OpenText(path);
string s="";
```

Класс `StreamReader` является более высокоуровневым в сравнении с `FileStream`. Чтение и печать на консоль строк выполняются в цикле:

```
while((s=sr.ReadLine()) != null)
 Console.WriteLine(s);
```

Оператор `s=sr.ReadLine()` читает строку символов, записанную ранее как массив символов. Конец файла соответствует чтению неинициализированной строки `null`.

В следующем примере рассмотрим, как можно выбрать имя файла из окна файлового диалога (листинг 3.13).

Листинг 3.13. Работа с окном файлового диалога

```
using System;
using System.Window.Forms;
```

```
class My1
{
 public static void Main()
 {
  OpenFileDialog dlgop= new OpenFileDialog();
  if (dlgop.ShowDialog()==DialogResult.OK)
   Console.WriteLine(dlgop.FileName);
 }
```

Объявление окна файлового диалога реализует команда `OpenFileDialog dlgop= new OpenFileDialog()`. Если выбран файл (нажата кнопка **OK**), то срабатывает оператор `if (dlgop.ShowDialog()==DialogResult.OK)`, после чего имя выбранного файла выводится на консоль:

```
Console.WriteLine(dlgop.FileName)
```

Еще один пример (листинг 3.14), в котором мы теперь рассмотрим работу с низкоуровневым классом `FileStream`. В этом примере вводится имя файла из консоли. Затем выполняется чтение информации из этого файла и запись считанных данных в файл с тем же именем, но с расширением bat.

Листинг 3.14. Класс `FileStream`

```
using System;
using System.Windows.Forms;
using System.IO;
class Example
{
 public static void Main()
 {
  String FileName;
  Console.WriteLine("Input File Name->");
  FileName=Console.ReadLine();
  // Открываем файл для чтения:
  FileStream inf = File.OpenRead(FileName);
  // Открываем другой файл для записи:
  FileStream outf=File.OpenWrite(FileName+".bat");
```

```
  int b; // Переменная, куда записывается считываемый байт.
         // Байт всегда неотрицателен
  while ((b=inf.ReadByte()) >-1)
  // Читаем байт; если -1, то конец файла
   outf.WriteByte((byte)b); // Записываем байт b в другой файл
  outf.Flush(); // Очищаем выходной поток
  outf.Close(); // Закрываем выходной поток
  inf.Close(); // Закрываем входной поток
 }
}
```

В листинге 3.15 приведен пример работы с классами `StreamReader` и `StreamWriter`.

Листинг 3.15. Классы `StreamReader` и `StreamWriter`

```
using System;
using System.IO;
class Ex
{
 public static void Main()
 {
  FileStream ous=File.Create("c:\\my.txt"); // Создаем файл;
  // Удваиваем спецсимвол \ (обратный слэш) в строке
  //  "c:\\my.txt"; использование конструкции @"c:\my.txt" не
  // требует удваивать спецсимвол.
  StreamWriter sw= new StreamWriter(ous); // Создаем выходную
                                  // потоковую переменную sw
  sw.WriteLine("Gratuliren Sie"); // Пишем строку в файл
  sw.Flush(); // Выталкиваем байты из выходного потока в файл
  sw.Close();
  StreamReader sr= new StreamReader("c:\\my.txt"); // Создаем
  // входную потоковую переменную sr
  String sline;
```

```
  sline= sr.ReadLine(); // Читаем строку из файла
  Console.WriteLine(sline); // Выводим прочитанную строку на
                            // консоль
 }
}
```

В листинге 3.16 приведен пример, в котором рассмотрены запись и чтение из файлов переменных других типов. Запись и чтение переменных других типов, отличных от `byte` и `string`, осуществляются с помощью классов `BinaryWriter` и `BinaryReader`.

Листинг 3.16. Запись и чтение переменных различных типов

```
using System;
using System.IO;
class Ex
{
 public static void Main()
 {
  FileStream ous= File.Create(@"c:\my.dat");
  BinaryWriter bw= new BinaryWriter(ous); // Создаем
  // переменную bw на базе низкоуровневого класса FileStream
  bw.Write((int) 32); // Запись целого числа 32
  bw.Write((decimal) 4.67); // Запись вещественного числа 4.67
  string s="Es ist gut";
  bw.Write(s); // Запись строки
  bw.Flush();
  bw.Close();
  // Запись двух чисел и строки завершена; приступаем к чтению
  FileStream ins= File.OpenRead(@"c:\my.dat");
  BinaryReader br = new BinaryReader(ins); // Создаем
  // переменную для ввода br на базе низкоуровневого класс
  // FileStream
  int i= br.ReadInt32(); // Читаем первое целое число
  decimal d = br.ReadDecimal(); // Читаем второе вещественное
                                // число
```

```
  string s1=br.ReadString(); // Читаем строку
  // Теперь все выводим на консоль:
  Console.WriteLine(""+i); // Присоединяем к пустой строке
                           // число, получаем строку
  Console.WriteLine(""+d);
  Console.WriteLine(""+s1);
  Console.ReadLine();
  br.Close();
 }
}
```

Сериализация объектов

Сериализация позволяет сохранить на диске объекты, а затем обратно прочитать их в память. Сериализация выполняется по тем же правилам, что и в Java. Даже если вы не читали первую часть этой книги, рекомендуем просмотреть *главу 2*. Мы ограничимся лишь небольшим примером с комментариями (листинг 3.17).

Листинг 3.17. Сериализация объектов

```
using System;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
[Serializable] // Этот спецификатор необходимо помещать при
               // сериализации
class Customer
{
 private int CustomerNumber; // Члены класса Customer -
                             // CustomerNumber, CustomerName
 private string CustomerName;

 public void WriteCustomer()
```

```
 {
  Console.WriteLine("Number is: "+this.CustomerNumber);
  Console.WriteLine("Name is: "+ this.CustomerName);
 }

 public Customer (
 // определение конструктора класса Customer;
 // ему передаются два параметра
                int NewNumber,
                string NewName)
 {
  this.CustomerNumber=NewNumber; // Инициализация переменных
                                 // класса
  this.CustomerName=NewName;
 }
}
class Example
{
 public static void Main()
 {
  // Создаем объект класса Customer:
  Customer my=new Customer(1, "vovan");
  my.WriteCustomer(); // Запускаем метод WriteCustomer()
                      // объекта
  FileStream fs= new FileStream("c.dat",
  FileMode.Create);
  // Еще один способ создать файл через конструктор
  // FileStream с указанием режима FileMode.Create
  BinaryFormatter bf= new BinaryFormatter();
  // Класс BinaryFormatter используется для сериализации
  // объектов
  bf.Serialize(fs,my); // Вот это и есть сериализация объекта
                       // my
```

```
    fs.Flush(); // Очищаем выходной поток
    fs.Close(); // Закрываем выходной поток

// Теперь прочитаем объект из файла:
    FileStream rs = new FileStream("c.dat",FileMode.Open);
    Customer newone=(Customer) bf.Deserialize(rs); // Создаем
    // новый объект и читаем объект из файла перед оператором
    // bf.Deserialize(rs) указываем в скобках тип
    // считываемого объекта для приведения типов;
    // такой порядок следует запомнить на будущее
    newone.WriteCustomer();//выполняем вывод для нового объекта
    System.Console.ReadLine();
  }
}
```

В этом примере сериализуем объект класса `Customer` следующим образом:

```
BinaryFormatter bf= new BinaryFormatter();
// Класс BinaryFormatter используется для сериализации
объектов
bf.Serialize(fs,my); // Вот это и есть сериализация объекта my
                     // класса Customer
```

Чтение сериализованного объекта выполняют следующие строки кода:

```
// Теперь прочитаем объект из файла
FileStream rs = new FileStream("c.dat",FileMode.Open);
Customer newone=(Customer) bf.Deserialise(rs); // Создаем
// новый объект и читаем объект из файла
```

Замечание

Сериализуются только простые объекты (не контейнеры), которые не содержат других объектов. Методы при сериализации не сохраняются.

Создание приложений на основе формы

Формы — это окна приложений. C# предлагает достаточно мощный и простой в освоении инструментарий для создания приложений на основе форм. Можно сказать, что эта технология во многом общая с Delphi или Visual Basic. Разработчику нужно разместить на окне визуальные компоненты (кнопки, списки, меню, текстовые поля, рисунки и пр.) и запрограммировать события, связанные с этими компонентами. При этом каждый компонент имеет достаточно большое число событий, доступ к которым следует реализовать через окно свойств (**Properties**) компонента. Пользователь также может связать с событиями компонента собственные методы (что будет показано далее). Наконец, пользователь может создавать новые визуальные и невизуальные компоненты и работать с ними так же естественно, как со стандартными компонентами C#.

Для создания приложений на основе формы выполним команду меню **File | New | Project | Windows Application**.

На экране монитора появится пустая форма, на которой разместим компоненты `Label` и TrackBar (рис. 3.6).

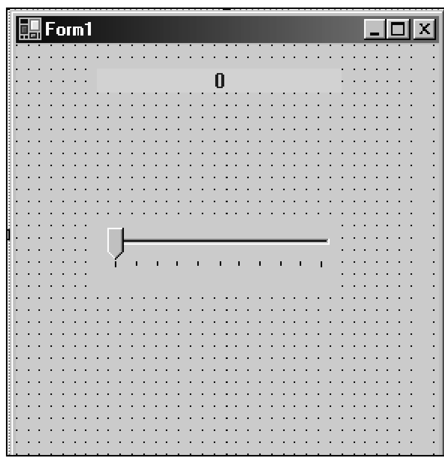


Рис. 3.6. Простое приложение на основе формы

Для этого откройте палитру элементов **ToolBox**, а если ее нет, то используйте пункт основного меню **View**. Для того чтобы придать элементам требуемый вид, нажмите на элемент правой кнопкой мыши и выберите пункт **Properties**. Откроется окно свойств данного элемента. Например, свойство `Value` компонента `TrackBar` определяет позицию ползунка. Свойство `Text` компонента `Label` определяет его текст. Свойство `BackColor` компонентов `Label` и `Form` определяет цвет этих компонентов. Свойство `TextAlign` компонента `Label` определяет расположение текста и т. д. Программная установка этих свойств реализуется, как показано далее на примере для установки цвета фона:

```
this.label1.BackColor=System.Drawing.Color.FromArgb
(((System.Byte)(255)), ((System.Byte)(224)),
(System.Byte)(100));
```

Здесь записана команда для установки цвета фона компонента с именем `label1` (класса `Label`), размещенного на форме (предполагается, что ссылка `this` указывает на форму).

Пусть предназначение нашей программы заключается в том, чтобы фиксировать в ярлыке `Label` положение ползунка. Для этого дважды щелкнем по компоненту `TrackBar` и введем следующий код с позиции, где устанавливается курсор.

```
static void Main()
{
 Application.Run(new Form1());
}


private void trackBar1_Scroll(object sender,
System.EventArgs e)
{
 label1.Text=""+trackBar1.Value; // Добавлено нами
}
```

Запустим программу на выполнение командой меню **Debug | Start**. Изменяя позицию ползунка, убедимся, что его координаты отображаются в поле `Label`.

Итак, визуальная среда проектирования приложений в .NET имеет много общего с Delphi и Visual Basic: с компонентами

связан набор свойств и событий. События обрабатываются стандартными методами. Задача пользователя — создать свой код обработки события. Познакомимся более подробно с работой списков, расширив наше приложение. Расположим на форме список `ComboBox` и кнопку `Button`. По нажатию на кнопку значение ползунка будет записываться в список. Запрограммируем кнопку таким образом:

```
private void button1_Click(object sender, System.EventArgs e)
{
 string s=""+trackBar1.Value;
 int i=comboBox1.FindString(s);
 if (i<0)
  comboBox1.Items.Add(s);
 else
  MessageBox.Show("Элемент уже есть");
}
```

Добавление нового элемента в список реализуется командой:

```
comboBox1.Items.Add(s);
```

Команда `int i=comboBox1.FindString(s)` присваивает переменной `i` значение номера строки, где в списке находится строка `s`. Если строки `s` в списке нет, то функция `FindString(s)` возвращает значение −1.

Теперь пусть по щелчку мыши на элементе списка устанавливается новая координата ползунка. Это можно реализовать так:

```
private void comboBox1_SelectedIndexChanged(object sender,
System.EventArgs e)
{
 string s=comboBox1.Items[comboBox1.SelectedIndex].ToString();

 trackBar1.Value=Convert.ToInt32(s);
 label1.Text=s;
}
```

Чтобы записать этот метод, просто щелкнем мышью на элементе `comboBox1`. Ссылка `comboBox1.SelectedIndex` определяет номер

выделенного элемента списка. Значение `comboBox1.Items[comboBox1.SelectedIndex].ToString()` определяет сам выделенный элемент. Метод `Convert.ToInt32(s)` преобразует строку в целое число.

Теперь сделаем так, чтобы по двойному щелчку мыши на форме список очищался. Прежде всего, откройте список свойств формы. Найдите значок молнии и откройте его. Найдите слово `DoubleClick` и щелкните по нему (рис. 3.7).

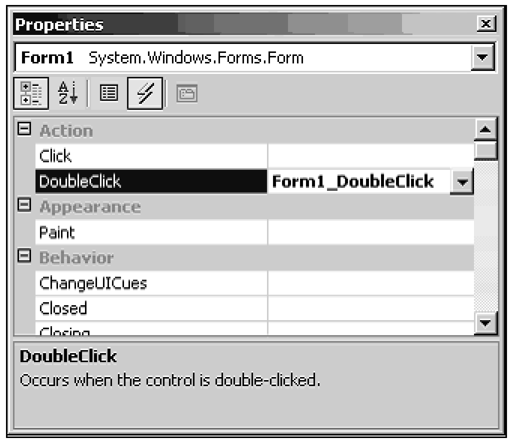


Рис. 3.7. Программирование событий `DoubleClick`

Запишите следующий код:

```
private void Form1_DoubleClick(object sender,
System.EventArgs e)
{
 comboBox1.Items.Clear();
}
```

Наконец, научимся удалять элемент из списка путем щелчка на нем правой кнопкой мыши. Для этого выделим выпадающий список, откроем его окно свойств, щелкнем на значке молнии и найдем свойство `MouseDown`. Щелкнем по нему и в окне редактора кода напишем следующие строки:

```
private void comboBox1_MouseDown(object sender,
System.Windows.Forms.MouseEventArgs e)
```

```
{
 if (e.Button==MouseButtons.Right)
  comboBox1.Items.RemoveAt(comboBox1.SelectedIndex);
}
```

Обратим внимание на то, что сначала необходимо выбрать элемент списка щелчком левой кнопки мыши, а затем щелкнуть правой кнопкой мыши на значении элемента в текстовом поле списка. Команда

```
comboBox1.Items.RemoveAt(comboBox1.SelectedIndex)
```

удаляет выделенный элемент списка.

Команда

```
comboBox1.Items.Insert(string s,int i);
```

вставляет строку `s` в позицию `i`.

Для того чтобы сделать список невидимым, используйте `comboBox1.Hide()`, а для того чтобы сделать его обратно видимым — `comboBox1.Show()`. Чтобы вовсе удалить список, используйте `comboBox1.Dispose()`.

Следующие команды выделяют элемент списка, совпадающий со строкой `textBox2.Text`.

```
int index = comboBox1.FindString(textBox2.Text);
comboBox1.SelectedIndex = index;
```

Имеется возможность связать с событием свой собственный обработчик. Тем не менее список аргументов этого обработчика должен соответствовать стандартному. Так, изменим обработчик события `comboBox1_MouseDown` на свой собственный. Для этого в модуль инициализации формы нужно поместить строку:

```
this.comboBox1.MouseDown+=new System.
EventHandler(this.myownmethod);
```

Тем самым мы указываем, что событие `OnMouseDown` закрепляется за обработчиком по имени `myownmethod`. Реализуем этот метод так:

```
private void myownmethod(object sender,
System.Windows.Forms.MouseEventArgs e)
{
 if (e.Button==MouseButtons.Right)
```

```
  comboBox1.Items.RemoveAt(comboBox1.SelectedIndex);
}
```

Вставим его следующим образом. Выберем компонент Form1 (щелчком мыши на форме) и откроем метод InitializeComponent() (рис. 3.8).

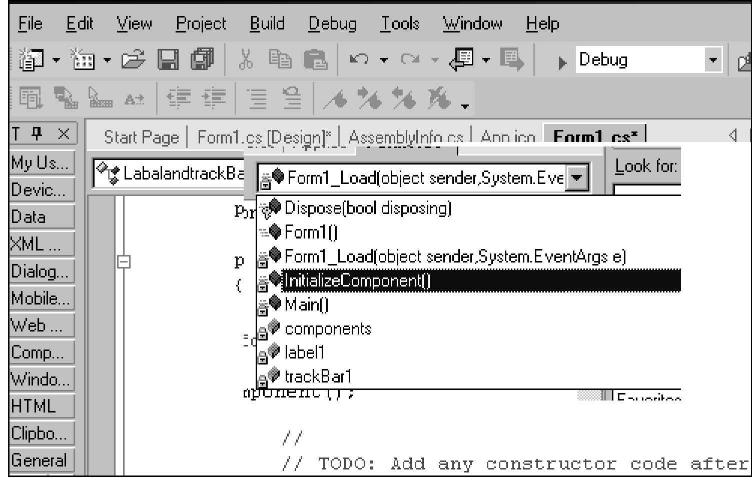


Рис. 3.8. Окно метода инициализации формы

Заменим строку

```
this.comboBox1.MouseDown+=new
System.Windows.Forms.MouseEventHandler
(this.comboBox1_MouseDown);
```

строкой

```
this.comboBox1.MouseDown+=new
System.Windows.Forms.MouseEventHandler(this.myownmethod);
```

Эта новая строка указывает на то, что обработчиком события, связанного с нажатием кнопки мыши на списке comboBox1, является метод myownmethod.

Теперь изменим содержимое класса Form1 следующим образом:

```
static void Main()
```

```
{
 Application.Run(new Form1());
}
private void trackBar1_Scroll(object sender,
System.EventArgs e)
{
 label1.Text=""+trackBar1.Value;
}
private void button1_Click(object sender, System.EventArgs e)
{
 string s=""+trackBar1.Value;
 int i=comboBox1.FindString(s);
 if (i<0)
  comboBox1.Items.Add(s);
 else
 MessageBox.Show("Элемент уже есть");
}
private void comboBox1_SelectedIndexChanged(object sender,
System.EventArgs e)
{
 string s=comboBox1.Items[comboBox1.SelectedIndex].ToString();
 trackBar1.Value=Convert.ToInt32(s);
 label1.Text=s.Insert(0,"hi");
}
private void Form1_DoubleClick(object sender,
System.EventArgs e)
{
 comboBox1.Items.Clear();
}
// Следующий метод теперь бесполезен
private void comboBox1_MouseDown(object sender,

System.Windows.Forms.MouseEventArgs e)
```

```
{
 if (e.Button==MouseButtons.Right)
  comboBox1.Items.RemoveAt(comboBox1.SelectedIndex);
}
// Добавлено нами
private void myownmethod(object sender,
                        System.Windows.Forms.MouseEventArgs e)
{
 if (e.Button==MouseButtons.Right)
  comboBox1.Items.RemoveAt(comboBox1.SelectedIndex);
}
```

Итак, в этом разделе вами было получено представление о работе с формой и элементами управления. Следует обратить внимание на использование свойств элементов в программе и программирование событий от элементов.

Работа со строками

Работа со строками составляет важную часть процессов программирования приложений. Операции над строками включают: создание строки, копирование строки, вставку строки, извлечение подстроки, извлечение символа и др. Базовыми классами для строк являются `String` и `string`. Первый из них содержит полный арсенал средств для работы со строками, второй является простым классом, но также предоставляет ряд методов для оперирования строками. Количество символов в строке определяется с помощью свойства `Length`, например:

```
int i=mystr.Length;
```

Команда `int pos= mystr.IndexOf("страна")` определяет позицию первого вхождения подстроки `"страна"` в строке `mystr`.

Команда `int pos= mystr.IndexOf("страна",5)` отыскивает первое вхождение подстроки `"страна"`, начиная с пятого символа строки `mystr`, а команда `int pos= mystr.IndexOf("страна",5, 20)` ограничивает поиск между 5-м и 20-м символами.

Команда `string s = mystr.Substring(ind)` позволяет получить подстроку строки `mystr`, начиная с символа `ind`, и до конца. Команда `string s= mystr.Substring(i1,i2)` позволяет выделить подстроку с позиции `i1` и содержащую число символов `i2`.

Соединение строк осуществляется следующими способами:

```
string s= String.Concat(s1,s2);
```

Здесь строка `s` получает значение конкатенации (соединения) строк `s1` и `s2`.

```
string s = s1+s2; // Соединение выполняет перегруженный
                  // оператор +.
```

Оператор + позволяет добавить к строке не-строку:

```
string s= ""+DateTime.Now; // Добавление к пустой строке
// нами широко использовалось ранее
```

Сравнение строк выполняется так:

```
int i= String.Compare(s1,s2,true);
```

Результат этой команды определяется следующим образом:

- `i=1`, если строка `s1>s2` (">" понимается в лексикографическом смысле);
- `i=0`, если строка `s1=s2` ("=" понимается как совпадение);
- `i=-1`, если строка `s1<s2` ("<" понимается в лексикографическом смысле).

Можно сравнивать строки и так:

```
if(s1==s2) {}
```

или

```
if (s1.Equals(s2)) {…}.
```

Вставку подстроки в строку выполняет команда:

```
string s = mystr.Insert(index, "что вставляем");
```

Эта команда вставляет подстроку "`что вставляем`" в строку `mystr`.

Удаление части строки реализует оператор:

```
string s= mystr.Remove(ind1, ind2);
```

Удаляются символы, ограниченные позициями `ind1` и `ind2`.

Замена символов реализуется так:

```
string s = mystr.Replace('a','b'); // Буква a заменяется на
                                   // букву b
```

Замена одной подстроки на другую выполняется так:

```
string s= mystr.Replace("hello","good bye");
```

Перевод строки в верхний и нижний регистры выполняются, соответственно, так:

```
s=s.ToUpper();
s=s.ToLower();
```

Проверка начала строки с указанной подстроки реализуется таким образом:

```
if (mystr.StartsWith("Hello")) {…}
```

Аналогично, проверка завершения строки указанной подстрокой реализуется таким образом:

```
if (mystr.EndsWith("Hello")) {…}
```

Оператор `Split` позволяет сформировать массив строк из одной большой строки, используя символ расщепления строки, как показано далее:

```
String s="Hello, You, there ";
string [] mystrarray=s.Split(",");
foreach (string st in mystrarray)
{
 Console.WriteLine(st);
}
```

Здесь разделителем является символ запятой. Будет сформирован строковый массив из трех строк: `"Hello"`, `"You"`, `"there"`.

Интересным классом для работы со строками является класс `StringBuilder`. Этот класс работает со строками как с последовательностями символов. Он позволяет добавлять, удалять, вставлять и замещать символы. Например:

```
String s= "Hello";
StringBuilder sb= new StringBuilder(s);// Создает переменную
                                       // класса StringBuilder
```

```
sb.Append(", You — programmer of my heart ! "); // Добавляет
                                                // к ней новую
                                                // строку
sb.Insert(4, "Hello again"); // Вставляет фразу "Hello again"
                             // после 4—го символа
sb.Remove(0,4); // Удаляет первое слово Hello
s=sb.ToString(); // Выполняет присваивание строке s
```

Создание сборок

До сих пор мы выполняли наши программы, запуская их через выполнение команды меню **Debug | Start**. Программа выполнялась непосредственно в среде C#. Но может возникнуть необходимость создать самостоятельное приложение, выполнение которого никак не должно привязываться к среде программирования C#. Такие самостоятельные приложения называются *сборками*. Слово "сборка" означает, что приложение собрано из отдельных составляющих частей. Именно в сборку могут входить откомпилированные классы (каждый в своем файле), из которых один класс обязательно должен содержать метод `Main()`, dll-функции, ресурсы (прежде всего, иконки и битмап-образы), а также манифест сборки. Мы не будем использовать манифесты. Отметим, что однофайловую сборку можно создать, используя команду **Build** основного меню среды разработки, выбрав **Build** имя_приложения. Будет создан exe-файл. Наша цель — показать, как создать многофайловую сборку. Пусть текст первого файла будет следующим (листинг 3.18).

Листинг 3.18. Первый файл сборки

```
using System;
namespace StringConversion
{
 class My
 {
  string privatestring;
  public string instring;
```

```
   {
    get
    {
     return privatestring;
    }
    set
    {
     privatestring=value;
    }
   }

  public void upper(out string upperstr)
   {
    upperstr=privatestring.ToUpper();
   }
  }
}
```

Этот файл следует создать не как проект, а как обычный текстовый файл. Обратим внимание на конструкцию `get-set`. Она применяется для объявления публикуемых свойств класса. Это перенято из Visual Basic. Публикуемое свойство получает значение методом `set`, а прочитать значение этого свойства можно методом `get`. Фрагмент объявления публикуемого свойства таков:

```
public string instring
 {
  get
  {
   return privatestring;
   }
  set
  {
   privatestring=value;
   }
 }
```

Чтобы этот файл с именем, скажем, ex1.cs использовать в качестве файла сборки, нужно запустить из командной строки MS-DOS компилятор C#:

```
csc /t: module ex1.cs
```

Для этого выберите **Пуск | Программы | Visual Studio .NET 2003 | Tools | Command Prompt**. Если в программе ошибок нет, то будет создан файл с именем ex1.netmodule.

Теперь наберите текст второго главного файла сборки с именем ex2.cs (листинг 3.19).

Листинг 3.19. Второй файл сборки

```
using System;
using StringConversion;
class MyMain
{
 public static void Main()
 {
  string localstring;
  My ms = new My();
  ms.instring ="hello from …";
  ms.upper(out localstring);
  Console.WriteLine(localstring);
 }
}
```

Видим, что этот класс использует класс `My`, определенный в первом файле (листинг 3.18). Для того чтобы это стало технически возможным, подключаем пространство имен первого файла посредством `using StringConversion`.

Теперь нужно скомпилировать и этот файл, но он — главный, поэтому строка компиляции изменится так:

```
csc /addmodule : ex1.netmodule /t: module ex2.cs
```

Теперь нужно собрать все в одно приложение:

```
al ex1.netmodule
```

```
ex2.netmodule /main: ex2 ex2.Main
/out: ex0.exe /t:exe
```

Часть

```
al  ex1.netmodule
ex2.netmodule
```

в общем-то понятна; `al` — это сокращение от `assemble` (от англ. — `"собирать")`. В этой части указываются имена объединяемых модулей.

Опция `/main: ex2 ex2.Main` указывает, что имя главного файла `ex2`, а имя главной функции — `Main`.

Опция `/out: ex0.exe` указывает, что имя создаваемого файла `ex0.exe`. Опция `/t:exe` определяет исполняемый тип файла.

Итак, мы "оторвали" приложение от среды C#. Кроме того, сборки позволяют писать программу отдельными модулями, т. е. разбить ее на функционально законченные части и отдать разным людям на исполнение.

Создание Web-приложений

Как нам известно, при программировании в Web строят отдельно клиентские и серверные приложения. Задача серверного приложения — обработать данные сайта, переданные браузером. C# позволяет создавать серверные приложения. Полезным при чтении этого раздела будет знакомство с материалом практического занятия *главы 2 "Использование JSP-страниц"*. Общий характер процесса разработки Web-приложения такой же, как и для обычного приложения Windows. Для создания Web-приложений необходимо установить и запустить Web-сервер IIS (Internet Information Server), см. далее. Действия по созданию Web-приложений сводятся к следующим.

1. Создать новый *проект* на базе шаблона ASP.NET Web Application и присвоить ему имя, например, MyWebApplication1. При вводе имени приложения указать также в поле **Location** адрес: **http://localhost/MyWebApplication1**. Создаваемые файлы будут помещены в папку MyWebApplication1 каталога wwwroot. По умолчанию IIS использует каталог c:\InetPub\wwwroot, однако этот каталог можно изменить, используя окно установки свойств IIS.

2. Расположить на фор `ней` ме визуальные компоненты, например, `Label` (либо `TextBox`) и `Button`. Запрограммировать кнопку хотя бы так:

   ```
   { label1.Text = "Hello From Server! ";}
   ```

3. Чтобы проверить работу формы, необходимо просто вызвать ее на выполнение командой **Debug | Start** либо использовать комбинацию клавиш <Ctrl>+<F5>. В результате компиляции проекта система создаст необходимые файлы Web-приложения в указанном каталоге MyWebAppication1.

Надо полагать, что наш простейший пример не вызовет ошибок. Система построит для созданного Web-приложения два файла: WebForm1.aspx, содержащий код HTML и ASP.NET, и файл WebForm1.aspx.cs, содержащий код C#. Файл с кодом HTML используется для возврата клиенту формы документа с результатом действия нашей кнопки. Файл с кодом C# выполняет программу для кнопки.

Содержимое файла WebForm1.aspx будет иметь следующий вид (листинг 3.20).

Листинг 3.20. Файл WebForm1.aspx

```
<%@ Page language="c#" Codebehind="WebFormsApp1.aspx.cs"
AutoEventWireUp="false"
Inherits="MyWebApplication.WebForm1" %>
<!DOCTYPE HTML PUBLIC "~//W3C//DTD HTML 4.0 Transitional//EN"
>
<HTML>
<HEAD>
 <title> WebForm1</title>
 <meta content="Microsoft Visual Studio 7.0"
name="Generation">
 <meta content="C#" name="CODE_LANGUAGE">
 <meta content="JavaScript" name="vs_defaultClientScript">
 <meta content=http://schemas.microsoft.com/intellisense/ie5
 name="vs_targetSchema">
</HEAD>
```

```
<body MS_POSITIONING="GridLayout">
 <form id="Form1" method="post" runat="server">
  <asp:TextBox id="TextBox1" style="Z-INDEX: 101; LEFT: 24px;
  POSITION: absolute; TOP: 20px" runat="server"
  Width="411px" Height="187px"
  TextMode="MultiLine"> </asp:TextBox>
  <asp:Button id="Button1" style="Z-INDEX: 102; LEFT: 40px;
  POSITION: absolute; TOP: 312px" runat="server" Width="104px"
     Height="27px"
  Text=" PRESS" > </asp:Button>
 </form>
</body>
</HTML>
```

Приведенный документ — это ASP.NET-страница (Active Server Pages — активные серверные страницы). (Вспомните уже рассмотренные нами JSP-страницы.) ASP-страница содержит вперемежку теги HTML и ASP-теги. ASP-тег

```
<%@ Page language="c#" Codebehind="WebForm1.aspx.cs"
AutoEventWireUp="false"
Inherits="MyWebApplication.WebForm1" %>
```

указывает, что рабочим языком страницы является C#, а имя программного модуля, поддерживающего Web-форму, — WebForm1.0aspx.cs. Указывается, что классом формы является `Inherits="MyWebApplication.WebForm1"`.

Наконец, сообщается через `AutoEventWireUp="false"`, что процессор ASP.NET не будет автоматически вызывать методы обработки событий инициализации и загрузки ASP-страницы. Чтение остальных тегов не вызывает затруднений.

Теперь, чтобы обратиться из клиентской стороны (сайта) к данному серверному приложению, нужно в окне браузера ввести адрес:

```
http://localhost/MyWebApplication1
```

Если обращение к ASP-странице должно производиться с другого компьютера, то вместо `localhost` следует указать сетевой адрес компьютера, содержащего ASP-страницу.

Нужно понимать, что для активизации созданных нами ASP-страниц на стороне сервера должно быть активным приложение Web-сервера. Если для Java использовали в качестве Web-сервера Tomcat, то для C# используем в качестве Web-сервера IIS. Этот Web-сервер входит в состав Windows и его следует инсталлировать и запустить.

Для инсталляции IIS выберите **Пуск | Настройка | Панель управления | Установка/удаление программ | Установка/удаление компонентов Windows**. Затем отметьте окошко IIS и нажмите кнопку **Next**. Система пригласит вас вставить компакт-диск с версией Windows, что вам и надлежит сделать.

После инсталляции IIS этот сервер нужно запустить. Для этого снова выберите **Пуск | Настройка | Панель управления | Сервисы**. Найдите в списке сервисов Internet Information Services и активизируйте контекстное меню щелчком правой кнопки мыши на имени IIS. Выберите пункт **Start**. Попытайтесь связаться с главной страницей IIS, открыв окно Internet Explorer и указав в качестве URL в поле адреса **http://www.localhost**. Если IIS успешно инсталлирован и запущен, то откроется домашняя страница этого Web-сервера. Имейте в виду, что Web-сервер IIS требует размещать создаваемые ASP-страницы в каталоге C:\Inetpub\wwwroot. Запомните это. Следовательно, создаваемые визуальной средой C# страницы ASP следует размещать в этом каталоге.

Другая возможность создания Web-приложений состоит в использовании шаблонов Empty Web Project. Эта возможность является аналогом сервлетов языка Java. Создадим подобное простейшее приложение.

1. Выберите шаблон **Empty Web Project** и присвойте ему имя `webHandle`. Включите в этот проект ссылку на библиотеку `System.Web`, для чего следует щелкнуть правой кнопкой мыши в строке **References**, выбрать в контекстном меню пункт **Add Reference**, а затем выбрать из выпадающего списка опцию **`System.Web.dll`**, щелкнув по ней дважды.
2. Откройте контекстное меню проекта и выберите **Add | Add New Item | Text File**. Присвойте новому файлу имя `webHandlerC.ashx`. Введите в него следующий код:

```
<%@ WebHandler Language="c#"
Codebehind="webHandlerC.ashx.cs"
class ="webHandle.webHandlerC" %>
```

3. Теперь нужно добавить в проект файл на языке C#, именованный как `webHandlerC`. Для этого щелкните правой кнопкой мыши на проекте и выберите **Add | Add New Item | Code File**. Присвойте файлу имя `webHandlerC.ashx.cs`. Введите следующий код:

```
using System.Web;
namespace webHandle
{
public class webHandlerC : IHttpHandler
{
public void ProcessRequest (HttpContext context)
{
HttpRequest request=context.Request;
HttpResponse resp=context.Response;
resp.ContentType="text/plain";
resp.Write(" Hello from C#-script");
resp.End;
}

public bool IsReusable
{
get
{return true;}
}
}
}
```

4. Откройте панель свойств `webHandlerC.ashx` и установите свойство **Build Action** в значение **Content**. После этого выберите команду **Set as Start Page** в контекстном меню `webHandlerC.ashx`.
5. Добавьте в проект конфигурационный файл:

   Add | Add New Item | Web Configuration File

6. Запустите проект на выполнение: <Ctrl>+<F5>.

Построенный нами проект есть аналог сервлета языка Java. Обращение к созданному Web-приложению осуществляется через URL: **http://localhost/webHandle**.

В приведенном примере ключевую роль играют переменные `resp` и `request`.

Переменная `request` позволяет получить данные из формы клиентского сайта. Наиболее просто это сделать, используя по аналогии приведенные далее команды:

```
int i = int.Parse(context.Request["tfage"]); // Получаем
// значение элемента формы HTML-документа с именем tfage и
// преобразуем его к целому числу
s= context.Request["Uname"];
```

Здесь в квадратных скобках после `Request` указаны имена текстовых полей, размещенных на форме клиента и переданных браузером нашему Web-приложению. Команда `int.Parse` преобразует строку в целое число.

Переменная `resp` позволяет вернуть на сторону клиента HTML-документ. Пояснение дает следующий пример.

```
int i1, i2; //индексы для массивов
NameValueCollection params=Request.QueryString;
// Получаем коллекцию имен и значений параметров
// В массив arr заносим имена параметров
String[] arr = params.AllKeys;
for (i1 = 0; i1 < arr.Length; i1++)
{
 // Выводим на сторону клиента имя параметра в цикле
 Response.Write("Key: " + Server.HtmlEncode(arr[i1]) +
"<br>");
 // Получаем значения параметра в массиве arr2
 String[] arr2 = params.GetValues(arr[i1]);
 for (i2 = 0; i2 < arr2.Length; i2++)
 {
```

```
  // Выводим значения параметра на сторону клиента
  Response.Write("Value " + i2 + ": " +
Server.HtmlEncode(arr2[i2])
                  + "<br>");
 }
}
```

Эффективная обработка клиентских форм средствами языка ASP выходит за пределы нашего рассмотрения.

Реализация API-вызовов

API-вызовы — это вызовы функций операционной системы или dll-функций. Для использования API-вызовов нужно их объявить как внешние методы с помощью директивы компилятора `DllImport`. Проблема использования API-вызовов упирается в необходимость согласования типов аргументов этих вызовов с типами C#. Приведем пример использования API-вызовов (листинг 3.21).

Листинг 3.21. Пример использования API-вызовов

```
using System;
using System.Text;
using System.Runtime.InteropServices;
namespace ConsoleApplication7API
{
 /// <summary>
 /// Summary description for Class1.
 /// </summary>
 class Class1
 {
  [DllImport("kernel32.dll")]
  static extern bool GetComputerName(StringBuilder name, ref
  ulong size);
```

```
  /// <summary>
  /// The main entry point for the application.
  /// </summary>
  [STAThread]

  static void Main(string[] args)
  {
   //
   // TODO: Add code to start application here
   //
   ulong size=256;
   StringBuilder name= new StringBuilder((int) size);
   bool success=GetComputerName(name, ref size);
   System.Console.WriteLine(name.ToString());
   System.Console.ReadLine();
  }
 }
}
```

Для использования API-вызова мы подключаем пакет:

```
using System.Runtime.InteropServices;
```

Далее включаем директиву компилятора:

```
[DllImport("kernel32.dll")]
```

Эта директива указывает, где находится интересующая нас API-функция.

Затем объявляем саму API-функцию:

```
static extern bool GetComputerName
(StringBuilder name,ref ulong size);
```

Эта функция возвращает имя компьютера, созданное при инсталляции операционной системы.

Вызов самой функции реализуется так:

```
bool success=GetComputerName(name, ref size);
```

Более сложно реализовать передачу структур в качестве параметров API-функций. Структуру необходимо предварительно объявить. Для этого используется директива компилятора `[StructureLayout]`. При объявлении структуры можно указывать явно смещения полей относительно начала структуры либо неявно. Поскольку второй способ не требует знания размеров занимаемой памяти соответствующими типами данных, нам он представляется более удобным. В связи со сказанным рассмотрим иллюстрацию применения структур (листинг 3.22).

Листинг 3.22. Передача структуры в API-вызове

```
using System;
using System.Runtime.InteropServices;
namespace ConsoleApplication6
{
 /// <summary>
 /// Summary description for Class1.
 /// </summary>
 [StructLayout(LayoutKind.Sequential)]
 struct SYSTEMTIME // Объявляется структура
 {
  public uint wYear;
  public uint wMonth;
  public uint wDayOfWeek;
  public uint wDay;
  public uint wHour;
  public uint wMinute;
  public uint wSecond;
  public uint wMilisecond;
 }

 class Class1
```

```
 {
  /// <summary>
  /// The main entry point for the application.
  /// </summary>
  [DllImport("kernel32.dll")]
  static extern  void GetSystemTime( out SYSTEMTIME st);
  [STAThread]

  static void Main(string[] args)
  {
   SYSTEMTIME st=new SYSTEMTIME();
   GetSystemTime( out st);
   Console.WriteLine("Сейчас="+ st.wHour+":"+ st.wMinute);
   Console.ReadLine();
  }
 }
}
```

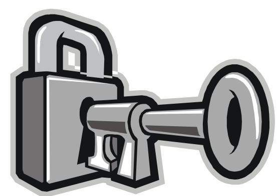
В данном примере объявляется *структура*:

```
[StructLayout(LayoutKind.Sequential)]
struct SYSTEMTIME
{
 public uint wYear;
 public uint wMonth;
 public uint wDayOfWeek;
 public uint wDay;
 public uint wHour;
 public uint wMinute;
 public uint wSecond;
public uint wMilisecond;
}
```

Инструкция `[StructLayout(LayoutKind.Sequential)]` объявляет, что поля структуры имеют размеры, определяемые используемыми типами. Далее следует объявление метода API:

```
[DllImport("kernel32.dll")]
static extern  void GetSystemTime( out SYSTEMTIME st);
```

Обратим внимание на то, что поиск метода `GetSystemTime()` (получения системного времени) выполняется в библиотеке kernel32.dll. Опция `out` указывает на то, что метод `GetSystemTime()` возвращает заполненные поля структуры.



Глава 4

Практические занятия по C#

Файловый ввод-вывод в C#

Цель занятия

Изучить простые способы работы с файлами в C#.

Краткие теоретические сведения.

В данной работе будут изучены не только механизмы файлового ввода-вывода, но и другие важные программные принципы C#. Дополнительно рекомендуем [9].

Прежде всего, запустите Visual Studio .Net и выберите опцию **New Project**, а затем **Windows Application**. На экране появится форма, на которую вам следует перетащить элементы управления: `ComboBox`, `PictureBox` и `Button`, причем через свойство **Items/Collection** окна свойств элемента `Combobox1` занесите начальные значения в `ComboBox` (рис. 4.1).

Собственно, это и есть форма проекта, с которой будем иметь дело в работе. Смысл работы заключается в том, что у нас имеется (будет вами создан) некоторый файл, содержащий информацию о расположении рисунков. Например, его содержимое: Мотоцикл: mot.bmp, Автомобиль: avto.bmp, Трактор: lorry.bmp.

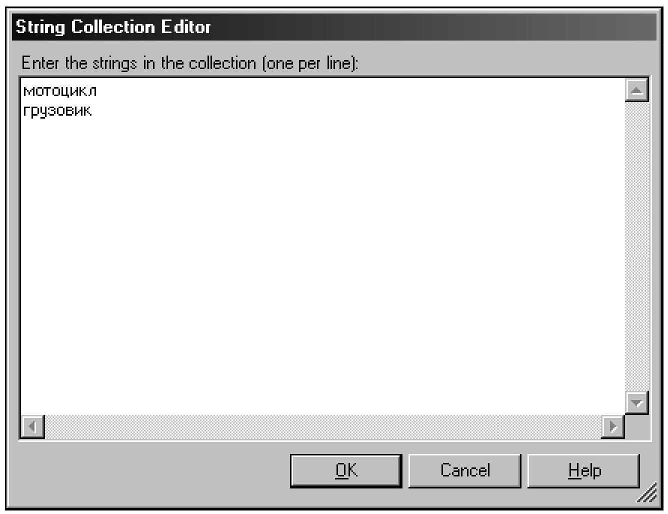


Рис. 4.1. Добавление элементов в выпадающий список

Пользователь будет выбирать из выпадающего списка (`ComboBox`) название элемента — например, **мотоцикл**. Затем, нажимая кнопку **Show Picture**, он будет получать изображение мотоцикла (рис. 4.2).

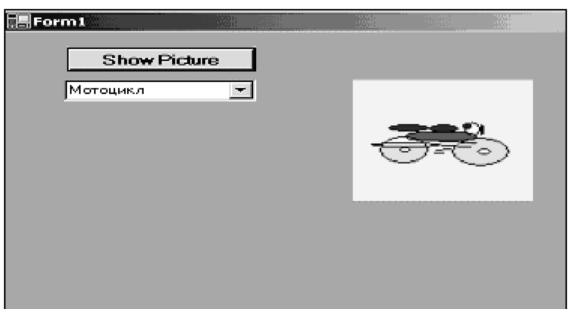


Рис. 4.2. Окно приложения для отображения рисунков

Для решения этой задачи нужно научиться работать с `ComboBox` и `PictureBox`, выполнять работу с файлами и уметь обрабатывать строки. Для первого занятия по C# этого вполне достаточно. Сначала о работе с элементами. Выбор элементов

осуществляется из окна **ToolBox**, расположенного слева на экране (а если его нет, то включите его, воспользовавшись командой меню **View**). Выберите вкладку **Window Forms** окна **ToolBox**. Элементы добавляются так же, как и в среде программирования Delphi. Взгляните на конструкцию:

```
if (comboBox1.SelectedIndex<0)
 {
  MessageBox.Show("No items selected");
 }
```

Это чистая конструкция языка C#. *Терм* `comboBox1.SelectedIndex` получает значение индекса выделенного элемента списка `ComboBox`. Отметим, что терм `comboBox1.SelectedItem` имеет значение выделенного элемента списка `ComboBox`. В этой работе нам пока достаточно и этого. Чтобы добавить элемент в *список*, используйте команду `comboBox1.Items.Add("NewElement")`, а чтобы удалить элемент из списка — команду `comboBox1.Items.Remove("NewElement")`.

Работу с файлами лучше пояснить прямо из текста программы:

```
StreamReader sr= File.OpenText(test.path);
string s="";
s=sr.ReadLine();
if (s!=null)
 {}
```

Объявляется *потоковая переменная* `sr` типа `StreamReader`. Этой переменной присваивается значение файловой переменной `File.OpenText(имя_файла)`. Переменная `sr` определяет текстовый файл для ввода, хотя, заметим, что представление информации в этом файле не соответствует ASCII (будет использоваться кодировка UTF8Encoding). *Чтение строки* из файла осуществляет команда `s=sr.ReadLine()`. Если прочитан хотя бы один байт, то это устанавливает проверка `if (s!=null) {}`. Таким образом, если в нашем файле одна-единственная строка, а пока это так и есть, то переменная `s` имеет именно следующий вид:

```
s
="Мотоцикл:mot.bmp,Автомобиль:avto.bmp,Трактор:lorry.bmp"
```

Теперь основополагающий принцип. Программирование в C# выполняется как и в Java на основе классов. В каком-то одном из этих классов должен быть реализован метод `Main()`. В программе создаются объекты классов — как обычно, с помощью конструкторов. Временно оторвемся от нашей программы и создадим что-то очень простое: на форме разместим кнопку, по щелчку мыши на которой будем выдавать сообщение. Но в обработчике события от кнопки создадим объект, а для вывода сообщения воспользуемся методом этого объекта. В листинге 4.1 приведен программный код приложения, которое выполняет эти действия.

Листинг 4.1. Каркас простого приложения с кнопкой

```
using System;
class Ex
{ // Объявление собственного класса с методом ShowStr()
 public void ShowStr()
 {
  MessageBox.Show("Ну, что, получили?");
  return ;
 }
}
public class Form1 : System.Windows.Forms.Form
{
.... .... ... ...  ... ...
 static void Main()
 {
  Application.Run(new Form1());
 }

 private void button1_Click(object sender, System.EventArgs e)
 {
  Ex myObj= new Ex();
  myObj.ShowStr();
 }
}
```

Многоточие заменяет все производимые системой подстановки, в которых нам пока нет никакой нужды. Главное, что мы объявили свой собственный класс:

```
class Ex
{
 public void ShowStr()
 {
  MessageBox.Show("Ну, что, получили?");
  return ;
 }
}
```

И вот теперь его используем общепринятым образом:

```
Ex myObj= new Ex();
myObj.ShowStr();
```

Но все это известно из других языков. Теперь возвращаемся снова к нашей программе и приводим ее полный текст (листинг 4.2).

Листинг 4.2. Полный текст приложения для чтения из файла и рисования картинки

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.IO;
using System.Text;
namespace ComboandFiles
{
 /// <summary>
 /// Summary description for Form1.
 /// </summary>
 public class Form1 : System.Windows.Forms.Form
```

```
{
 private System.Windows.Forms.ComboBox comboBox1;
 private System.Windows.Forms.PictureBox pictureBox1;
 private System.Windows.Forms.Button button1;
 /// <summary>
 /// Required designer variable.
 /// </summary>
 private System.ComponentModel.Container components = null;
 private Bitmap MyImage;
 public Form1()
 {
  //
  // Required for Windows Form Designer support
  //
  InitializeComponent();
  //
  // TODO: Add any constructor code after InitializeComponent
  // call
  //
 }
 /// <summary>
 /// Clean up any resources being used.
 /// </summary>

 protected override void Dispose( bool disposing )
 {
  if( disposing )
   {
    if (components != null)
     {
      components.Dispose();
     }
   }
  base.Dispose( disposing );
 }
```

```
class Test
{
 public string path;
 public Test()
 {
  path=@"d:/msdev/test.txt";
 }

 public string getinfo()
 {
  if (!File.Exists(path))
   {return ("File does not exists");}
  else {return ("OK");}
 }
}
#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>

private void InitializeComponent()
{
 this.comboBox1 = new System.Windows.Forms.ComboBox();
 this.pictureBox1 = new System.Windows.Forms.PictureBox();
 this.button1 = new System.Windows.Forms.Button();
 this.SuspendLayout();
 //
 // comboBox1
 //
 this.comboBox1.Items.AddRange(new object[] {"Мотоцикл",
                              "Автомобиль", "Трактор"});
 this.comboBox1.Location = new System.Drawing.Point(40, 48);
 this.comboBox1.Name = "comboBox1";
```

```
this.comboBox1.Size = new System.Drawing.Size(128, 21);
this.comboBox1.TabIndex = 0;
this.comboBox1.Text = "comboBox1";
//
// pictureBox1
//
this.pictureBox1.BackColor =
System.Drawing.SystemColors.Control;
this.pictureBox1.Location = new System.Drawing.Point(232,
48);
this.pictureBox1.Name = "pictureBox1";
this.pictureBox1.Size = new System.Drawing.Size(208, 232);
this.pictureBox1.TabIndex = 1;
this.pictureBox1.TabStop = false;
//
// button1
//
this.button1.BackColor=System.Drawing.Color.FromArgb(
((System.Byte)(255)), ((System.Byte)(192)),
((System.Byte)(192)));
this.button1.Font = new System.Drawing.Font("Microsoft Sans
Serif",
9.75F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((System.Byte)(204)));
this.button1.Location = new System.Drawing.Point(42, 16);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(126, 24);
this.button1.TabIndex = 2;
this.button1.Text = "Show Picture ";
this.button1.Click += new
System.EventHandler(this.button1_Click);
//
// Form1
//
this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
this.BackColor = System.Drawing.Color.RosyBrown;
```

```
 this.ClientSize = new System.Drawing.Size(456, 397);
 this.Controls.Add(this.button1);
 this.Controls.Add(this.pictureBox1);
 this.Controls.Add(this.comboBox1);
 this.Name = "Form1";
 this.Text = "Form1";
 this.ResumeLayout(false);
}
#endregion
/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]

static void Main()
{
 Application.Run(new Form1());
}
private void button1_Click(object sender,
System.EventArgs e)
{
 if (comboBox1.SelectedIndex<0)
  {MessageBox.Show("No items selected");}
 else
  {
   Test test=new Test();
   string stest=test.getinfo();
   if (stest=="OK")
    {
     if (MyImage!=null)
      { MyImage.Dispose();}
     StreamReader sr= File.OpenText(test.path);
     string s="";
     s=sr.ReadLine();
     if (s!=null)
```

```
      {
       string s1=  comboBox1.SelectedItem.ToString();
       int k= s.IndexOf(s1,0,s.Length);
       if (k<0)
        {
         MessageBox.Show("ComBoItem not Found");
         goto labelfin;
        }
       k=s.IndexOf(':',k,s.Length-k);
       if (k<0)
        {
         MessageBox.Show(": not found after specified combo
         Item");
         goto labelfin;
        }
       s=s.Substring(k+1,s.Length-k-1);
       k=s.IndexOf(',',0,s.Length);
       if (k>=0)
        {s=s.Substring(0,k);}
       s=@"d:\msdev\"+s;
       pictureBox1.SizeMode=
       PictureBoxSizeMode.StretchImage;
       MyImage= new Bitmap(s);
       pictureBox1.ClientSize=new Size(120,120);
       pictureBox1.Image=(Image) MyImage;
       labelfin: ;
      }
    }
   else
    {
     MessageBox.Show(stest);
     FileStream fs=File.Create(test.path,1024);
      {
      Byte[]info=new UTF8Encoding(true).GetBytes("Мотоцикл:
      mot.bmp,
```

```
        Автомобиль: avto.bmp, Трактор: lorry.bmp");
        fs.Write(info,0,info.Length);
        MessageBox.Show("File Created.Repeat then");
        }
      }
    }
  }
 }
}
```

Займемся теперь разъяснением этой программы с учетом того, что очень важные вещи мы уже объяснили. Большая часть этого кода была автоматически добавлена самой системой. Фрагмент:

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.IO;
using System.Text;
```

используется для подключения *библиотек классов*, содержащих описания системных функций. Например, функции для работы с файлами содержатся в классе `System.IO`. Строка `namespace ComboandFiles` используется для объявления пространства имен, что является новшеством, впервые введенным в C++. *Пространство имен* содержит определения имен, используемых в некотором контексте. Например, в программе может встретиться имя функции `ReadLine()`, которое конфликтует с именем стандартной функции. В этом случае для системы важно, какое пространство имен используется в качестве контекста. Далее идет длинное описание класса `public class Form1: System.Windows.Forms.Form`. Это описание ничем не занимательно, так как система создает его сама. Но мы поместили внутрь данного описания следующее определение собственного встроенного класса:

```
class Test
{
```

```
 public string path;
 public Test()
 {
  path=@"d:/msdev/test.txt";
 }

 public string getinfo()
 {
  if (!File.Exists(path))
   {return ("File does not exists");}
  else {return ("OK");}
 }
}
```

Это описание включает объявление переменной `path`, конструктора `Test()`, который данную переменную инициализирует, и метода `getinfo()`, проверяющего, имеется ли данный файл в указанном каталоге. Кстати, запомните эту простую команду: `File.Exists(path)`. Остается привести код обработчика события от нажатия кнопки и пояснить его, что сделано с помощью помещенных рядом с командами комментариев.

```
private void button1_Click(object sender, System.EventArgs e)
{
 if (comboBox1.SelectedIndex<0) // Проверяет, выделен ли
 // элемент в комбосписке
  {MessageBox.Show("No items selected");} // Сообщение, если
                                          // не выделен
 else
  {
   Test test=new Test(); // Иначе создает объект класса Test
   string stest=test.getinfo(); // Метод getinfo()класса Test
   // Сообщает о наличии файла данных
   if (stest=="OK")
    {
     if (MyImage!=null) // Проверяет наличие рисунка
     // и удаляет текущий рисунок
```

```
 { MyImage.Dispose();}
StreamReader sr= File.OpenText(test.path); // Создает
// файловую переменную для ввода строки из файла
// с адресами рисунков
string s="";
s=sr.ReadLine(); // Читает строку из файла, имя которого
// помещено в переменной test.path
if (s!=null) // Строка не пуста?
 {
  string s1=comboBox1.SelectedItem.ToString(); //
Преобразует
  // в подстроку выделенный элемент списка
  int k= s.IndexOf(s1,0,s.Length); // Определяет
  // местонахождение данной подстроки в строке из файла
  if (k<0)
   {
    MessageBox.Show("ComBoItem not Found");
    goto labelfin; // Редкое использование оператора goto
   }
  k=s.IndexOf(':',k,s.Length-k); // Определяет позицию
  // ":" вслед за именем подстроки
  if (k<0)
   {
    MessageBox.Show(": not found after specified combo
    Item");
    goto labelfin;
   }
  s=s.Substring(k+1,s.Length-k-1); // Определяет адрес
  // файла с рисунком
  k=s.IndexOf(',',0,s.Length);
  if (k>=0)
   {s=s.Substring(0,k);}
```

```
        s=@"d:\msdev\"+s; // Добавляет к имени файла
        // полный путь; в вашей работе этот оператор следует
        // изменить
        pictureBox1.SizeMode= PictureBoxSizeMode.StretchImage;
        MyImage= new Bitmap(s); // Создает битовый образ
                                // картинки из файла
        pictureBox1.ClientSize=new Size(120,120); // Задает
                                                  // размеры
                                                  // картинки
        pictureBox1.Image=(Image) MyImage; // Преобразует
                                           // BitMap в Image
        labelfin: ;
       }
     }
    else // Файл с адресами картинок не существует, и он
        // создается
     {
       MessageBox.Show(stest);
       FileStream fs=File.Create(test.path,1024);
       { Byte [] info=new
UTF8Encoding(true).GetBytes("Мотоцикл: mot.bmp,
         Автомобиль: avto.bmp, Трактор: lorry.bmp");
        fs.Write(info,0,info.Length); // Занесение информации
                                      // в файл из буфера
        MessageBox.Show("File Created.Repeat then");
       }
     }
    }
   }
  }
}
```

Для отображения рисунка на форме необходимо разместить компонент класса `PictureBox`. У этого компонента имеется свойство `Image`, которое в качестве значения получает массив

пикселов (или иначе — битмап-образ). В программе это реализовано следующим образом:

```
MyImage= new Bitmap(s); // Создает битовый образ картинки из
                        // файла
pictureBox1.ClientSize=new Size(120,120); // Задает размеры
                                          // картинки
pictureBox1.Image=(Image) MyImage;
```

Переменная `MyImage` объявлена в самом начале класса `Form1` следующим образом:

```
private Bitmap MyImage;
```

В этом приложении неоднократно использовалось диалоговое окно для вывода текстового сообщения `MessageBox.Show`.

Следует также обратить внимание на получение массива байтов, кодирующих отдельные символы строки:

```
Byte [] info=new UTF8Encoding(true).GetBytes("Мотоцикл:
mot.bmp, Автомобиль: avto.bmp, Трактор: lorry.bmp");
```

Задание

Итак, поставленная в работе цель достигнута. Вам надлежит несколько усложнить задачу, а именно: смоделировать простой SQL-запрос к базе данных в текстовом файле. Следует реализовать задачу поиска автомобиля, например, не старше 1990 года выпуска и стоимостью не выше $1000. Для этого вам нужно включить в текстовый файл поля :

```
avto: ... , price: ..., year: ... , photo : ... , ... ...
...
avto: ... , price: ..., year: ... , photo : ...
```

Далее потребуется несколько дополнительных возможностей. Во-первых, показываем, как преобразовать строку в целое число на примере:

```
string s= textBox1.Text;
int i=Convert.ToInt32(s,10);
```

Во-вторых, рассмотрим, как осуществлять запись и чтение в текстовый файл не одной-единственной строки, а нескольких строк.

В C# для этих целей используют классы `StreamWriter` и `StreamReader`. Покажем это на примере:

```
using System;
using System.IO;
class Ex2
{
 public static void Main()
 {
  FileStream ous=File.Create(@"c:\data.txt");
  StreamWriter sw= new StreamWriter(ous);
  sw.WriteLine("one");
  sw.WriteLine("two");
  sw.Flush();
  sw.Close();
  StreamReader sr= File.OpenText(@"c:\data.txt");
  string s1;
  while((s1=sr.ReadLine())!=null)
  {
    Console.WriteLine(s1);
  }
  Console.ReadLine();
sr.Close();
 }
}
```

Здесь создаем объект `StreamWriter` для записи в файл и `StreamReader` — для чтения. Предварительно порождаем файловую переменную `FileStream ous`. Запись в файл реализуем командой `WriteLine()`, чтение — `ReadLine()`. Признаком достижения конца файла является условие `sr.ReadLine()==null`. Теперь вы должны самостоятельно построить простейшее приложение для поиска автомобиля по цене и году.

Контрольные вопросы

1. Как разместить визуальный компонент на форме и задать его свойства?
2. Как запрограммировать обработчик события для компонента? Как создать свой обработчик?
3. Объясните то место в программе, где выполняется создание и отображение рисунка.
4. Объясните то место в программе, где выполняется чтение строки из файла.
5. Как прочитать из файла несколько строк?
6. Объясните назначение класса `Test`.

Работа с базами данных в C#

Цель занятия

Познакомиться с принципами работы с базами данных в C#. Научиться читать, записывать и выбирать информацию из таблиц баз данных, используя SQL-запросы. Освоить работу с классами ADO.NET [8, 9].

Краткие теоретические сведения

В настоящем практическом занятии рассматривается работа с базами данных на основе классов ADO.NET. Имеется три способа использования этих классов через соединения:

- `SqlConnection`;
- `OleDbConnection`;
- `OdbcConnection`.

Соединение `SqlConnection` предназначено для работы с сервером баз даных MS SQL-server 2000. В этой работе мы его не рассматриваем. Соединение `OleDbConnection` используется для связи "напрямую" с локальной базой данных (поддерживаются только базы данных Microsoft). Соединение `OdbcConnection`

использует механизм драйверов ODBC для связи с широким диапазоном баз данных.

Начнем с `OleDbConnection`. Приводим сразу работоспособный пример и затем комментируем его (листинг 4.3).

Листинг 4.3. Работа с базой данных на основе `OleDbConnection`

```
using System;
using System.Data;
using System.Data.OleDb;
namespace lessonbasaconsole
{
 /// <summary>
 /// Summary description for Class1.
 /// </summary>
 class Class1
 {
  /// <summary>
  /// The main entry point for the application.
  /// </summary>
  [STAThread]
  static void Main(string[] args)
  {
   //
   // TODO: Add code to start application here
   //
   string connectingString
   =@"provider=Microsoft.Jet.OLEDB.4.0;
   data source= c:\disk_d\msdev\mydb.mdb";
   OleDbConnection myConn = new OleDbConnection
   (connectingString);
   //Сформировать строку запросов
   string selectString ="Select name,group from stud";
   OleDbCommand myCommand= myConn.CreateCommand();
   myCommand.CommandText= selectString;
```

```
    OleDbDataAdapter oda= new OleDbDataAdapter();
    oda.SelectCommand=myCommand;
    DataSet myDataset= new DataSet();
    myConn.Open();
    string dataTableName="studA";
    oda.Fill(myDataset,dataTableName);
    DataTable myDataTable= myDataset.Tables[dataTableName];
    foreach (DataRow dr in myDataTable.Rows)
    {
     Console.WriteLine("Name="+dr["name"]);
     Console.WriteLine("Group="+dr["group"]);
    }
    string s= Console.ReadLine();
    myConn.Close();
   }
  }
}
```

Сразу обратим внимание на подключаемые библиотеки классов. А теперь рассмотрим фрагмент, реализующий соединение с базой. Сначала строим строку соединения с локальной базой:

```
string connectingString=@"provider=Microsoft.Jet.OLEDB.4.
0;
data source= c:\disk_d\msdev\mydb.mdb";
```

Указанная здесь база данных `mydb.mdb` построена в Access. Она содержит таблицу `stud`, которая, в свою очередь, содержит всего два поля: `name` и `group`.

Следующая строка

```
OleDbConnection myConn = new OleDbConnec-
tion(connectingString);
```

создает новое соединение с этой базой.

Затем с помощью трех строк:

```
string selectString ="Select name,group from stud";
OleDbCommand myCommand= myConn.CreateCommand();
```

```
myCommand.CommandText= selectString;
```

строим SQL-команду для выполнения выборки из таблицы `stud`.

Затем нужно подключить адаптер для связи с базой:

```
OleDbDataAdapter oda= new OleDbDataAdapter();
oda.SelectCommand=myCommand;
```

Созданную ранее SQL-команду присваиваем полю `SelectCommand` объекта-адаптера.

Далее создаем объект типа `DataSet`, который хранит выбираемые наборы данных из таблиц и хранит, в том числе, сами таблицы. После этого открываем соединение:

```
DataSet myDataset= new DataSet();
myConn.Open();
```

Теперь с помощью команд:

```
string dataTableName="studA";
oda.Fill(myDataset,dataTableName);
```

как раз и осуществляем выборку данных из базы и помещение их в условную таблицу `studA` (заметим, что имя реальной таблицы — `stud`).

Итак, `OleAdapter` располагает командой для связи с базой, и с помощью этой команды соединен с объектом-соединением (`connection`). Таким образом, при открытии соединения данные могут быть считаны адаптером в память. Адаптер выбирает данные и помещает их в таблицу `studA` командой `oda.Fill(myDataset,dataTableName)`.

Теперь остается только прочитать данные из таблицы `studA` и отобразить их на консоли:

```
DataTable myDataTable= myDataset.Tables[dataTableName];
foreach (DataRow dr in myDataTable.Rows)
{
 Console.WriteLine("Name="+dr["name"]);
 Console.WriteLine("Group="+dr["group"]);
}
```

Результат работы представлен на рис. 4.3.

Описанный здесь порядок действий при работе с базой данных является общим и для двух других типов соединений.

Рис. 4.3. Вывод на консоль записей таблицы `stud`

Приводим пример той же программы для работы с ODBC (листинг 4.4). Заметим, что предварительно нужно создать ODBC-соединение с помощью утилиты ODBC Administrator, как это было сделано в Java *(см. разд. "Java и базы данных" главы 2).*

Листинг 4.4. Пример работы с базой данных на основе интерфейса ODBC

```
using System;
using System.Data;
using System.Data.Odbc;
namespace odbcexample
{
 /// <summary>
 /// Summary description for Class1.
 /// </summary>
 class Class1
 {
  /// <summary>
  /// The main entry point for the application.
  /// </summary>
  [STAThread]
  static void Main(string[] args)
  {
   //
   // TODO: Add code to start application here
   //
   string connectingString ="DSN=myaccess";
```

```
    OdbcConnection myConn = new OdbcConnection
    (connectingString);
    string mySelectQuery = "SELECT name,group FROM stud";
    OdbcCommand myCommand = new OdbcCommand(mySelectQuery);
    myCommand.Connection= new OdbcConnection(connectingString);
    OdbcDataAdapter oda= new OdbcDataAdapter();
    oda.SelectCommand=myCommand;
    //Сформировать строку запроса
    DataSet myDataset= new DataSet();
    myConn.Open();
    string dataTableName="studA";
    oda.Fill(myDataset,dataTableName);
    DataTable myDataTable= myDataset.Tables[dataTableName];
    foreach (DataRow dr in myDataTable.Rows)
    {
     Console.WriteLine("Name="+dr["name"]);
     Console.WriteLine("Group="+dr["group"]);
    }

    string s= Console.ReadLine();
    myConn.Close();
   }
  }
}
```

Обратим внимание на строки:

```
OdbcCommand myCommand = new OdbcCommand(mySelectQuery);
myCommand.Connection= new OdbcConnection(connectingString);
```

которыми отличается этот способ от предыдущего, а именно: нужно определять свойство `Connection` у объекта `OdbcCommand`. В остальном сходство обоих листингов вполне усматривается.

Запросы, отличные от `SELECT`, реализуются, как показано далее на примере команды `INSERT` для вставки записи (листинг 4.5).

Листинг 4.5. Реализация запроса `INSERT`

```
using System;
using System.Data;
using System.Data.Odbc;
namespace odbcexample
{
 /// <summary>
 /// Summary description for Class1.
 /// </summary>
 class Class1
 {
  /// <summary>
  /// The main entry point for the application.
  /// </summary>
  [STAThread]
  static void Main(string[] args)
  {
   //
   // TODO: Add code to start application here
   //
   string myConnectionString="DSN=myaccess";
   string myExecuteQuery="insert into stud
   values('han',2,null)";
   OdbcConnection myConnection = new
   OdbcConnection(myConnectionString);
   OdbcCommand myCommand = new OdbcCommand(myExecuteQuery,
   myConnection);
   myConnection.Open();
   myCommand.ExecuteNonQuery();
   myConnection.Close();
   string s=Console.ReadLine();
  }
```

```
  }
}
```

Видим, что достаточно воспользоваться в этом случае только объектами `OdbcConnection` и `OdbcCommand`. Сам запрос выполняется на открытом соединении командой `myCommand.ExecuteNonQuery()`. Этот запрос, разумеется, не создает результирующего набора записей. Проверить результат его выполнения можно непосредственно, открыв таблицу `stud` в Access.

Задание

Создать консольное приложение, которое позволяло бы не только выбирать записи из базы данных, но и добавлять и удалять их. Для этих целей следует использовать язык запросов SQL.

Контрольные вопросы

1. Какие классы реализуют работу с базами данных в C#?
2. Какая функция класса адаптера?
3. В каком объекте записывается SQL-запрос?
4. Что такое соединение, как его создать?
5. Где сохраняется результирующий набор по запросу `SELECT`?
6. Чем отличается запрос на выборку записей из таблицы от запроса на вставку записей или их обновление?

Простейшее рисование в C#

Цель занятия

В настоящей работе необходимо познакомиться с рисованием пером в C#. Надлежит построить график синусоиды или другой математической функции. Дополнительные сведения можно найти в [7].

Краткие теоретические сведения

Для рисования линий и фигур нужно получить сначала графический контекст. Затем выдать команды рисования. Следующий фрагмент поясняет сказанное:

```
Graphics g= e.Graphics; // получаем графический контекст
Pen p = new Pen(Color.Bisque,5); // создаем перо для рисования
                                 // линий
g.DrawRectangle(p,10,10,200,200); // рисуем прямоугольник
```

Графическому контексту передается обработчик события `OnPaint()`. Поэтому можно "привязаться" именно к этому обработчику. Смысл наших действий таков: поставить кнопку и закрепить за ней обработчик события `OnClick()`. В этом пункте обработчику следует просто вызвать перерисовку формы, где и получить фигуры. Это делается так:

```
private void button1_Click(object sender, System.EventArgs e)
{
 show=1;
 this.Invalidate();
}
```

Мы намеренно используем переменную `show` для того, чтобы отменить рисование, когда `show==0`. Метод `Invalidate()` как раз и активизирует событие `OnPaint()`. Это обстоятельство полезно запомнить.

Собственные обработчики событий закрепляются за событиями в конструкторе:

```
this.button2.Click += new
System.EventHandler(this.button2_Click);
```

Эта команда закрепляет программу `button2_Click` за событием нажатия кнопки.

```
this.Paint+=new
System.Windows.Forms.PaintEventHandler(this.showgraph);
```

Приведенная команда аналогичным образом закрепляет программу в качестве обработчика события `OnPaint()`.

Результатом работы описываемого приложения является следующая форма после нажатия на кнопку **Painting** (рис. 4.4).

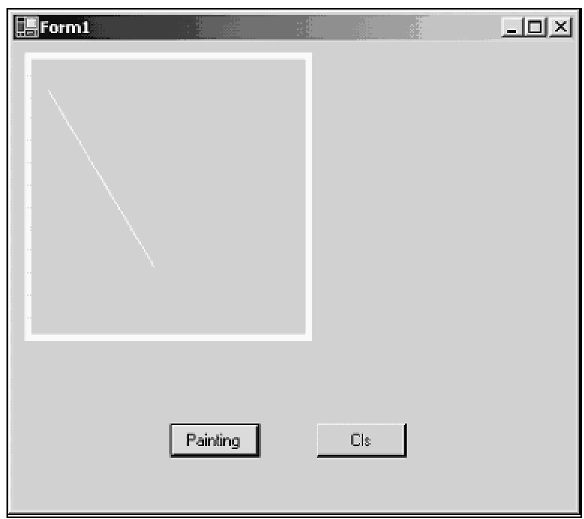


Рис. 4.4. Простейшее рисование на форме

Полный текст приложения приведен в листинге 4.6.

Листинг 4.6. Приложение для рисования на форме

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
namespace painting
{
 /// <summary>
 /// Summary description for Form1.
 /// </summary>
 public class Form1 : System.Windows.Forms.Form
 {
  private System.Windows.Forms.Button button1;
```

```
/// <summary>
/// Required designer variable.
/// </summary>
private System.ComponentModel.Container components = null;
private System.Windows.Forms.Button button2;
private short show = 0; // Значение show = 0 запрещает
                        // рисование

public Form1()
{
 //
 // Required for Windows Form Designer support
 //
 InitializeComponent();
 //
 // TODO: Add any constructor code after InitializeComponent
 // call
 //
}
/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
 if( disposing )
  {
   if (components != null)
    {
     components.Dispose();
    }
  }
 base.Dispose( disposing );
}
#region Windows Form Designer generated code
/// <summary>
```

```
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>

private void InitializeComponent() // Стандартная программа
// инициализации формы позволяет вставить свои собственные
// команды
{
 this.button1 = new System.Windows.Forms.Button();
 this.button2 = new System.Windows.Forms.Button();
 this.SuspendLayout();
 //
 // button1
 //
 this.button1.BackColor = System.Drawing.Color.FromArgb((
 (System.Byte)(255)), ((System.Byte)(192)),
 ((System.Byte)(192)));
 this.button1.Location = new System.Drawing.Point(112, 272);
 this.button1.Name = "button1";
 this.button1.Size = new System.Drawing.Size(64, 24);
 this.button1.TabIndex = 0;
 this.button1.Text = "Painting";
 this.button1.Click += new
 System.EventHandler(this.button1_Click);
 //
 // button2
 //
 this.button2.Location = new System.Drawing.Point(216, 272);
 this.button2.Name = "button2";
 this.button2.Size = new System.Drawing.Size(64, 24);
 this.button2.TabIndex = 1;
 this.button2.Text = "Cls";
 this.button2.Click += new
 System.EventHandler(this.button2_Click);
```

```
 //
 // Form1
 //
 this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
 this.BackColor =
 System.Drawing.Color.FromArgb(((System.Byte)(192)),
 ((System.Byte)(192)), ((System.Byte)(255)));
 this.ClientSize = new System.Drawing.Size(400, 333);
 this.Controls.Add(this.button2);
 this.Controls.Add(this.button1);
 this.Name = "Form1";
 this.Text = "Form1";
 this.Paint += new
 System.Windows.Forms.PaintEventHandler(this.showgraph);
 // Закрепление события OnPaint() за обработчиком
 // с названием showgraph
 this.ResumeLayout(false);
}
#endregion
/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static void Main()
{
 Application.Run(new Form1());
}
private void showgraph(object sender,
                      System.Windows.Forms.PaintEventArgs e)
{
 if (show==1)
  { // Рисование, когда show=1
   Graphics g= e.Graphics; // Формируем графический контекст
```

```
      Pen p = new Pen(Color.Bisque,5); // Создаем перо для
                                        // рисования
      g.DrawRectangle(p,10,10,200,200); // Рисуем прямоугольник
      Pen p2= new Pen(Color.Beige,1);
      g.DrawLine(p2,25,35,100,160); // Вторым пером рисуем
                                    // линию
      show=0;
     }
   }
   private void button1_Click(object sender,
   System.EventArgs e)
   {
    show=1;
    this.Invalidate(); // Активизируем событие OnPaint()
   }
   private void button2_Click(object sender,
   System.EventArgs e)
   {
    show=0;
    this.Invalidate();
   }
  }
}
```

Другой способ использования графического контекста состоит в его прямом создании следующим образом:

```
Graphics g = control.CreateGraphics();
```

Здесь в качестве `control` может выступать форма.

Итак, резюмируем сведения по простейшему рисованию.

Для рисования следует использовать *графический контекст*; графический контекст передается в метод, закрепленный за событием `OnPaint()`, но можно создать графический контекст, как было показано ранее.

Класс `Graphics` содержит методы для рисования прямоугольников и линий (и др.):

```
g.DrawRectangle(p,10,10,200,200); //рисуем прямоугольник
```

```
Pen p2= new Pen(Color.Beige,1);
g.DrawLine(p2,25,35,100,160);//вторым пером рисуем линию
show=0;
```

Для активизации события `OnPaint()` следует использовать команду `Invalidate()`. Эта команда заодно очищает форму от нарисованных фигур.

Задание

1. Напишите программу для рисования синусоиды.
2. Напишите программу для рисования гистограммы.
3. Напишите программу для рисования многоугольника.

Контрольные вопросы

1. Что такое графический контекст?
2. Как создать графический контекст?
3. Как закрепить за событием `OnPaint()` свой собственный обработчик, какие аргументы ему следует передать?
4. Как установить ширину и цвет линии фигуры?
5. Как нарисовать овал и прямоугольник?

Изучение механизма потоков для смены графических изображений

Цель занятия

Изучить возможности механизма потоков на примере смены графических изображений. *Дополнительно см. разд. "Потоки в Java" главы 2.*

Краткие теоретические сведения

Механизм потоков позволяет параллельно с основным потоком управления в программе выполнять один или несколько других потоков управления. Этот механизм нам уже знаком по курсу Java. Нам надлежит научиться порождать поток и выполнять задержку его выполнения на некоторое фиксированное время.

Каждый раз, когда эта задержка иссякает и поток активизируется, происходит смена картинки. В результате выполняется анимация изображения.

Нам потребуется также овладеть основами вывода графики в C#. С этого, пожалуй, проще всего и начать. Следующее приложение создается на основе формы Windows Application и текст его мы помещаем полностью (листинг 4.7).

Листинг 4.7. Еще одно приложение для рисования на форме

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
namespace lesson3graphics
{
 /// <summary>
 /// Summary description for Form1.
 /// </summary>

 public class My : System.Windows.Forms.Form
 {
  /// <summary>
  /// Required designer variable.
  /// </summary>
  private System.ComponentModel.Container components = null;

  public My()
  {
   //
   // Required for Windows Form Designer support
   //
   InitializeComponent();
   //
```

```
   // TODO: Add any constructor code after InitializeComponent
   // call
   //
  }
  /// <summary>
  /// Clean up any resources being used.
  /// </summary>
  protected override void Dispose( bool disposing )
  {
   if( disposing )
    {
     if (components != null)
      {
       components.Dispose();
      }
    }
   base.Dispose( disposing );
  }

  #region Windows Form Designer generated code
  /// <summary>
  /// Required method for Designer support - do not modify
  /// the contents of this method with the code editor.
  /// </summary>
  private void InitializeComponent()
  {
   this.components = new System.ComponentModel.Container();
   // this.Size = new System.Drawing.Size(300,300);
   // this.Text = "Form1";
   this.BackColor=System.Drawing.Color.Bisque;
   this.ClientSize=new System.Drawing.Size(400,400);
   this.Name="Lesson-Graphics";
   this.Text="LESSONC#";
   this.Paint+=new
   System.Windows.Forms.PaintEventHandler(this.MyPaint);
```

```
  }
  #endregion
  /// <summary>
  /// The main entry point for the application.
  /// </summary>
  [STAThread]
  static void Main()
  {
   Application.Run(new My());
  }
  private void MyPaint(object sender,
  System.Windows.Forms.PaintEventArgs
                      e)
  {
   Graphics g=e.Graphics;
   Pen p= new Pen(Color.Black,10);
   g.DrawLine(p,25,25,300,300);
   p.Color=Color.Red;
   g.DrawEllipse(p,50,50,100,100);
  }
 }
}
```

Результат работы этого приложения показан на рис. 4.5.

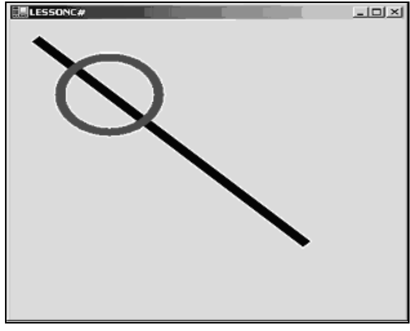


Рис. 4.5. Еще один пример рисования на форме

Программа рисует овал (эллипс) и прямую линию. Этой простейшей графики нам будет достаточно. Прежде всего, мы заполнили следующий текст инициализации:

```
private void InitializeComponent()
{
 this.components = new System.ComponentModel.Container();
 //this.Size = new System.Drawing.Size(300,300);
 //this.Text = "Form1";
 this.BackColor=System.Drawing.Color.Bisque;
 this.ClientSize=new System.Drawing.Size(400,400);
 this.Name="Lesson-Graphics";
 this.Text="LESSONC#";
 this.Paint+=new
System.Windows.Forms.PaintEventHandler(this.MyPaint);
}
#endregion
```

В тексте инициализации управляем цветом формы:

```
this.BackColor=System.Drawing.Color.Bisque;
```

размером формы:

```
this.ClientSize=new System.Drawing.Size(400,400);
```

и указываем обработчик события `OnPaint()`. Вы, вероятно, догадались, что это выполняет строка

```
this.Paint+=new
System.Windows.Forms.PaintEventHandler(this.MyPaint);
```

Запомните синтаксис этой конструкции, так как и для других событий ваши собственные обработчики указываются так же. Процедура-обработчик — `MyPaint`. Само имя `MyPaint` здесь можно изменить на любое другое. Текст этой процедуры помещен в самом приложении — найдите его. Обратите внимание на наличие аргументов, передаваемых в процедуру, — они стандартные, но нам потребуется объект-событие `System.Windows.Forms.PaintEventArgs e`. Этот объект предоставляет компонент `Graphics`, который собственно и нужен для рисования. Мы видим, как он используется:

```
Graphics g=e.Graphics;
Pen p= new Pen(Color.Black,10);
```

```
g.DrawLine(p,25,25,300,300);
p.Color=Color.Red;
g.DrawEllipse(p,50,50,100,100);
```

У компонента `Graphics` очень много других методов и свойств, но нам пока достаточно и этих. Рисование выполняется пером, как и в других языках. В заключение два новых важных замечания.

Замечание

Событие `OnPaint()` всегда происходит при смене размеров или местоположения окна приложения. Но его можно инициировать искусственно командой `Invalidate()`. Этой командой мы воспользуемся чуть позже.

Замечание

В этой работе мы займемся мультипликацией. Нам понадобится смена рисунков. Отображение рисунка можно реализовать так:

```
Bitmap b= new Bitmap("Face.bmp");
g.DrawImage(b,10,10,200,200);
// Здесь g — это объект Graphics. DrawImage рисует
// картинку в указанном прямоугольнике
```

На этом с графикой пока все. Приступаем к потокам. Опять же приведем нечто простое — консольное приложение (листинг 4.8), в котором создается и используется поток.

Листинг 4.8. Приложение с использованием потока

```
using System;
using System.Threading;
class Example
{
 public static void CountUp()
 {
  for(int i=0;i<100; i++)
```

```
    {
     Console.WriteLine("Counter now is->"+ i.ToString());
     Thread.Sleep(100);
    }
  }
 public static void Main()
  {
   Thread t2= new Thread(new ThreadStart(CountUp));
   t2.Start();
   t2.Join();
  }
}
```

Здесь представлено консольное приложение с единственным классом `Example`. Для создания приложения с потоками нужно выполнить подключение пакета

```
using System.Threading;
```

В классе `Example` определены два метода: `CountUp()`, `Main()`. Поток создается в методе `Main()`:

```
Thread t2= new Thread(new ThreadStart(CountUp));
```

В конструкторе потока указывается, какой метод поток начнет выполнять по команде `Start`. В нашем случае — это `CountUp()`. Заметим, что в методе `CountUp()` нет аргументов, однако в общем случае метод может содержать аргументы, в силу чего их нужно было бы указать в вызове (отмечено многоточием):

```
new ThreadStart(CountUp(...))
```

Поток запускается так: `t2.Start()`. Команда `t2.Join()` используется с тем, чтобы основная программа была приостановлена до завершения потока `t2`.

В методе `CountUp()` используется команда приостановки выполнения на заданное число миллисекунд (1 мс = 1/1000 с): `Thread.Sleep(100)`.

Созданный поток выполняет счет с задержкой и выводом на экран. И этого опять же достаточно. Теперь мы приведем полный текст приложения с формой, которое выводит динамически кружки синего цвета в разных местах экрана с

задержкой. Рисование кружков связывается с событием `OnPaint()`, которое генерирует поток с помощью команды `Invalidate()`. Вот полный текст работоспособного приложения (листинг 4.9).

Листинг 4.9. Приложение с потоком для анимации изображения

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Threading;
namespace graphandflows
{
 /// <summary>
 /// Summary description for Form1.
 /// </summary>
 public class Fora : System.Windows.Forms.Form
 {
  /// <summary>
  /// Required designer variable.
  /// </summary>
  private System.ComponentModel.Container components = null;

  public Fora()
  {
   //
   // Required for Windows Form Designer support
   //
   InitializeComponent();
   //
   // TODO: Add any constructor code after InitializeComponent
   // call
```

```
 //
}
/// <summary>
/// Clean up any resources being used.
/// </summary>

protected override void Dispose( bool disposing )
{
 if( disposing )
  {
   if (components != null)
    {
     components.Dispose();
    }
  }
 base.Dispose( disposing );
}
#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
 this.components = new System.ComponentModel.Container();
 this.Size = new System.Drawing.Size(400,400);
 this.Text = "FORA";
 this.BackColor=System.Drawing.Color.Bisque;
 this.Paint+=new
 System.Windows.Forms.PaintEventHandler(this.painting);
}
#endregion
public void painting(object sender,
                    System.Windows.Forms.PaintEventArgs e)
```

```
  {
   int i=0;
   Graphics g=e.Graphics;
   Pen p =new Pen(Color.Blue,5);
   Random r=new Random();
   i=r.Next(100);
   g.DrawEllipse(p,10+r.Next(300),10+r.Next(200),i,i);
  }

  public void cycl()
  {
   int i=0;
   while(i<50)
   {
    this.Invalidate();
    Thread.Sleep(400);
    i++;
    continue;
   }
  }

  /// <summary>
  /// The main entry point for the application.
  /// </summary>
  [STAThread]
  static void Main()
  {
   Fora f=new Fora();
   Thread t2= new Thread(new ThreadStart(f.cycl));
   t2.Start();
   Application.Run(f);
  }
 }
}
```

В этом приложении поток реализует метод `cycl()`. В отличие от Java для потока нет предопределенного метода `run()`. Перерисовка формы происходит каждые 400 мс:

```
this.Invalidate();
Thread.Sleep(400);
i++;
continue;
```

Нет необходимости объяснять этот текст более подробно, поскольку он очень прост, но следует обратить внимание на метод `Main()` и способ генерации потока в нем, а именно: поток запускается до запуска основной программы в строке `Application.Run(f)`.

Кроме того, процедура запуска потока указана как `f.cycl`, а не просто `cycl()`. Здесь действуют те же правила именования, что и в Java. Если вы пишете просто `cycl()`, то предполагается обращение к методу класса, а не методу порожденного объекта этого класса. Но тогда процедура `cycl()` должна быть объявлена в классе как `static`. У нас этого нет. Способ запуска потока в форме `f.cycl` вынуждает систему выполнения C# запустить именно метод `cycl()` объекта `f`, порожденного выше как `Fora f=new Fora()`.

Задание

Используя знания, сообщенные вам в этом занятии, создайте приложение, демонстрирующее анимацию простых графических файлов, созданных в PaintBrush.

Контрольные вопросы

1. Как создать поток?
2. Как создать рисунок на форме?
3. Как выполнить приостанов потока на требуемое время?
4. В чем отличие потоков C# от потоков в Java?
5. Использует ли поток статические методы класса?

Создание собственных компонентов

Цель занятия

Целью настоящего практического занятия является изучение способов создания собственных компонентов в среде C#. Аналогом этого занятия является создание компонентов Java Beans. В качестве дополнительной литературы можно рекомендовать [3].

Краткие теоретические сведения

В C# можно создавать следующие компоненты: COM *(см. Введение к этой книге)*, компоненты на основе визуальных классов и пользовательские компоненты. *Компоненты COM* могут использоваться в других средах программирования, например, в JavaScript. В этой работе мы их не рассматриваем. Компоненты на основе визуальных классов позволяют вам придать некоторую новую функциональность стандартным визуальным компонентам, например, на основе класса файлового диалога дополнительно предусмотреть выдачу информации о свойствах файла. Пользовательские компоненты позволяют вам создавать любые функциональности для своего класса. Наконец, в C# можно внедрить чужой элемент ActiveX, например, календарь.

Итак, начнем с компонентов, которые строятся на основе визуальных классов. Рассмотрим создание компонента, представляющего собой текстовое поле и полосу вертикальной прокрутки. При перемещении ползунка в текстовом поле будет отображаться его положение.

Для создания такого компонента следует выполнить следующие действия.

1. Создать новый проект на базе шаблона **Windows Control Library** (**File | New Project | Windows Control Library**).
2. Присвоить имя создаваемому компоненту, например, `MYSCROLL`.
3. Разместить на форме текстовое поле (задайте его свойство `ReadOnly = true`) и полосу вертикальной прокрутки из панели элементов **Window Forms**.
4. Создать следующий код (листинг 4.10).

Листинг 4.10. Код компонента

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Windows.Forms;
namespace WindowsControlLibrary1
{
 /// <summary>
 /// Summary description for UserControl1.
 /// </summary>
 public class MYSCROLL : System.Windows.Forms.UserControl
 {
  private int m_min=int.MinValue;
  private int m_max=int.MaxValue;
  private int m_current=0;
  private System.Windows.Forms.TextBox textBox1;
  private System.Windows.Forms.VScrollBar vScrollBar1;
  /// <summary>
  /// Required designer variable.
  /// </summary>
  private System.ComponentModel.Container components = null;
  public int Min
  {
   get
    {return m_min;}
   set
    { m_min=value;}
  }

  public int Max
  {
   get
```

```
      {return m_max;}
     set
      { m_max=value;}
   }

    public int Current
    {
    get
    {return m_current;}
    set
    {
     if((value>m_max)||(value<m_min))
      {
       throw new ArgumentOutOfRangeException("Current");
      }
     m_current=value;
     textBox1.Text=m_current.ToString();
    }
   }
  }
 }
```

В классе MYSCROLL мы определили три свойства: Min, Max и Current, задающие максимальную позицию, минимальную позицию и текущую позицию ползунка. Эти свойства являются публикуемыми (published, отображаемыми в окне свойств). Это значит, что они будут доступны в окне свойств объекта класса MYSCROLL, размещенного на форме обычным образом. Обратите внимание, как для каждого из этих свойств задаются методы get и set, используемые для получения (get) и установки (set) значения данного свойства у объекта.

Теперь надо запрограммировать реакцию на перемещение ползунка. Дважды щелкните мышью на ползунке на форме и вставьте следующий код:

```
private void vScrollBar1_Scroll(object sender,
```

```
    System.Windows.Forms.ScrollEventArgs e)
    {
     if (e.Type==ScrollEventType.SmallIncrement)
     {
      try
       { Current-=1; }
      catch
       {}
     }
    else
     if (e.Type==ScrollEventType.SmallDecrement)
      {
       Current+=1;
      }
   }
```

Событие `SmallIncrement` происходит при смещении ползунка вниз, событие `SmallDecrement` — при смещении ползунка вверх. Эти события анализируются в методе `vScrollBar1_Scroll()`, одним из аргументов которого является переменная `ScrollEventArgs e`, которая дает доступ к типу события из оператора `if`:

```
if (e.Type==ScrollEventType.SmallDecrement)
```

Созданный файл следует сохранить, например, под именем example.cs.

5. Скомпилируйте данный файл (вспомните, как выполняется компиляция из командной строки). Однако теперь мы должны построить dll, а это делается так:

```
csc /out: newelement.dll /t:library example.cs
```

Здесь предполагается, что исходный файл назван example.cs. Обратим внимание, что у него нет метода `Main()`. В результате будет построен выходной библиотечный dll-файл с именем newelement.dll.

6. Закройте проект.

Теперь выясним, как разместить созданный компонент на форме. Для этого создайте проект на основе шаблона **Windows Application**. Создайте далее в окне элементов новую закладку, для чего нажмите правую кнопку мыши на этом окне и выберите

пункт **Add Tab**. Введите имя закладки: `MYCOMPONENTS`. Выделите эту закладку щелчком мыши и нажмите снова правую кнопку мыши. Появится окно такого вида (рис. 4.6).

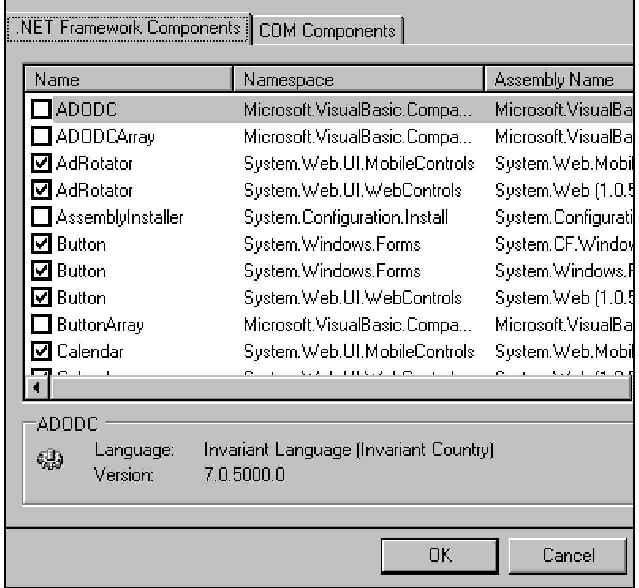


Рис. 4.6. Окно для установки компонентов

Выберите закладку **.NET Framework Components**. Затем кнопку **Browse**. Найдите, где был сохранен ваш dll-файл. Он сохраняется в том месте, которое вы указали при открытии пустого проекта в каталоге ... папка_с_проектами/debug/bin/... . Когда вы отыщете ваш dll-файл, нажмите **OK**. Ваш класс появится в списке, приведенном на рисунке выше. Выделите имя этого класса и нажмите **OK**. Теперь класс будет размещен в окне элементов. С ним можно работать как с обычной кнопкой. То, что вы создали, иллюстрируется ниже (рис. 4.7).

Теперь рассмотрим, как создавать собственные невизуальные компоненты. Технология в целом почти такая же.

Итак, продолжаем с компонентов, которые строятся на основе собственных пользовательских классов. Рассмотрим создание

компонента, представляющего собой закрашенный эллипс и имеющего редактируемое свойство `Message`.

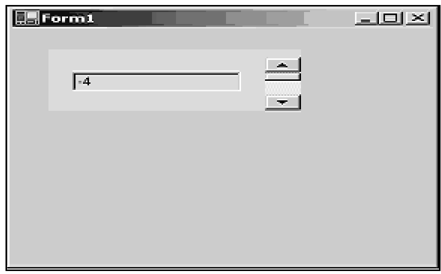


Рис. 4.7. Созданный компонент пользователя

Для создания такого компонента следует выполнить следующие действия.

1. Создайте новый проект на базе шаблона **Windows Control Library** (**File | New Project | Windows Control Library**).
2. Присвойте имя создаваемому компоненту: например, `MYCOMP`.
3. В окне **Solution Explorer** щелкните правой кнопкой мыши на узле `UserControl1` и удалите его из проекта.
4. Щелкните правой кнопкой мыши в окне **Solution Explorer** на названии библиотеки и выберите команду **Add | Add Inherited Control**. Выберите шаблон **Custom Control** и присвойте файлу имя, например, `WindowsControlLibrary2.cs`.
5. Создайте, например, следующий код (листинг 4.11).

Листинг 4.11. Код компонента на основе собственного класса

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Windows.Forms;
namespace WindowsControlLibrary2
```

```
{
 /// <summary>
 /// Summary description for CustomCTRL1.
 /// </summary>
 public class CustomCTRL1 : System.Windows.Forms.Control
 {
  private string m_mes;
  public string Message
  {
   get
   {return m_mes;}
   set
   {m_mes=value;}
  }

  /// <summary>
  /// Required designer variable.
  /// </summary>
  private System.ComponentModel.Container components = null;
  public CustomCTRL1()
  {
   // This call is required by the Windows.Forms Form
   // Designer.
   InitializeComponent();
   ResizeRedraw =true;
   m_mes="INITIALLY";
   // TODO: Add any initialization after the InitComponent
   // call
  }
  /// <summary>
  /// Clean up any resources being used.
  /// </summary>
  protected override void Dispose( bool disposing )
  {
   if( disposing )
```

```
    {
     if( components != null )
      components.Dispose();
    }
   base.Dispose( disposing );
  }
  #region Component Designer generated code
  /// <summary>
  /// Required method for Designer support - do not modify
  /// the contents of this method with the code editor.
  /// </summary>
  private void InitializeComponent()
  {
   components = new System.ComponentModel.Container();
  }
  #endregion
  protected override void OnPaint(PaintEventArgs pe)
  {
   // TODO: Add custom paint code here
   // Calling the base class OnPaint
   SolidBrush b= new SolidBrush (Color.SteelBlue);
   pe.Graphics.FillEllipse(b,0,0,50,50);
   base.OnPaint(pe);
  }
 }
}
```

Прорисовку эллипса мы поместили в обработчик события `OnPaint()`:

```
protected override void OnPaint(PaintEventArgs pe)
{
 // TODO: Add custom paint code here
 // Calling the base class OnPaint
 SolidBrush b= new SolidBrush (Color.SteelBlue);
 //Создает твердую кисть для заполнения цветом
```

```
 pe.Graphics.FillEllipse(b,0,0,50,50);
 // Рисует закрашенный эллипс;
 base.OnPaint(pe);
}
```

Объявление редактируемого свойства представлено следующим блоком:

```
private string m_mes;
public string Message
{
 get
 {return m_mes;}
 set
 {m_mes=value;}
}
```

Теперь достаточно просто скомпилировать проект. При этом создается dll-файл, который вам нужно установить на палитру элементов так же, как и ранее созданный компонент на основе визуальных классов.

После этого можно работать с данным элементом, например, как с кнопкой.

Задание

Создайте собственный невизуальный элемент, который имеет два метода: один используется для получения головной части строки, второй — для получения хвостовой части. Например, первый метод объявлен как:

```
String Head(String str, String mask)
```

Этот метод получает исходную строку, например: `"Hello, friends"` и строку-разделитель `","`. Данный метод отыскивает в строке `"Hello,friends"` первое вхождение строки-разделителя и возвращает часть исходной строки, стоящую до разделителя. Если же разделитель не встретится, то возвращается вся исходная строка.

Метод, возвращающий хвостовую часть, отличается от описанного только тем, что возвращает часть строки, стоящую после

разделителя. Для выполнения задания вам потребуются методы для работы со строками:

- `public int indexOf(String s)` — возвращает номер позиции, где встречается первое вхождение строки `s`, например:

```
String s1="Hello, Friends";
int i =s1.indexOf(",");
```

- `public String Substring(int i1,int2)` — выделяет из строки подстроку в диапазоне от `i1` до `i1+i2`.

Контрольные вопросы

1. Как создать компонент на основе визуального класса?
2. Как создать компонент на основе собственного невизуального класса?
3. Как установить для компонентов публикуемые свойства в окне **Properties**?
4. Где располагаются созданные dll-файлы и как их разместить в приложении?

Клиент-серверное взаимодействие на основе протоколов TCP и HTTP

Цель занятия

Познакомиться с возможностями взаимодействия приложений на C# через локальную сеть на основе протоколов TCP и HTTP. Изучить способы реализации клиента и сервера. Сравнить способы реализации клиент-серверного взаимодействия в Java и C#. Дополнительно см. [3, 10].

Краткие теоретические сведения

В этом занятии надлежит написать программу сервера и программу клиента и запустить их как раздельные приложения. Программа-клиент вызовет метод программы-сервера и получит

результат работы последней. Как и ранее, можно для запуска приложения клиента и приложения сервера использовать один и тот же компьютер, задав сетевой адрес 127.0.0.1. Можно запустить эти приложения на разных компьютерах. В этом случае следует использовать сетевое имя компьютера сервера для указания его в клиентском приложении. Это имя можно получить через **Мой компьютер | Свойства | Сетевая идентификация**.

Сначала приведем и подробно разберем текст программы-сервера (листинг 4.12). Это приложение реализует взаимодействие на основе протокола TCP.

Листинг 4.12. Приложение сервера

```
using System;
using System.IO;
using Sysem.Net.Sockets;
namespace Server
{
 public class ServerExample
 {
  private void Listen() // Это единственный метод сервера,
                        // который мы используем
   {
    IPAddress ipAddress =
    Dns.Resolve("localhost").AddressList[0];
    TcpListener tcpl = new TcpListener(50001); // Создаем
                                              // прослушиватель
                                              // порта 50001
    tcpl.Start(); // Запускаем прослушиватель,
                  // ожидание подключение клиента
    Socket newSocket = tcpl.AcceptSocket(); // Прослушиватель
    // ожидает подключения клиента
    if (newSocket.Connected) // Если соединение установлено, то
    // пересылаем данные клиенту
```

```
    {
     NetworkStream ns= new NetworkStream(newSocket);
     // Пересылка данных
     byte [] buf={(byte) ' S', (byte) 'O', (byte) ' ', (byte)
     'M', (byte)
     'U', (byte) 'S', (byte) ' ', (byte) 'T', (byte) ' ',
     (byte) 'B',
     (byte) 'E', (byte) ' ', (byte) 'I', (byte) ' ', (byte)
     'S', (byte)
     'A', (byte) 'Y',};
     ns.Write(buf,0,3); // Отправляем данные клиенту
     // Очистка
     ns.Flush();
     ns.Close();
     break;
    }
  }
  static void Main()
  {
   //
   // TODO: Add code to start application here
   ServerExample se=new ServerExample();
   se.Listen();
  }
 }
}
```

Итак, чтобы сервер возвратил строку `"SO MUST BE I SAY"`, необходимо подключение к нему клиента через порт 50001. Номер порта можно использовать и другой. Мы видим, что механизм сервера весьма прост и во многом напоминает тот, который использует классический Java. Прослушивание сокета реализуется командой

```
Socket newSocket = tcpl.AcceptSocket();
```

Эта команда "зависает" до тех пор, пока клиент не соединится с сервером. Если соединение успешно, то срабатывает оператор `if` и осуществляется вывод строки на сторону клинета:

```
if (newSocket.Connected) // Если соединение установлено,
                         // то пересылаем  данные клиенту
```

```
 {
  NetworkStream ns= new NetworkStream(newSocket);
  //пересылка данных
  byte [] buf={(byte) ' S', (byte) 'O', (byte) ' ',
  (byte) 'M',
  (byte) 'U', (byte) 'S', (byte) ' ', (byte) 'T', (byte)
  ' ', (byte)
  'B', (byte) 'E', (byte) ' ', (byte) 'I', (byte) ' ',
  (byte) 'S',
  (byte) 'A', (byte) 'Y',};
  ns.Write(buf,0,3);  //отправляем данные клиенту
```

Обратим внимание на то, что клиенту передается массив байтов. Байты получаются из объектов типа `char` путем явного преобразования типа.

Нужно теперь скомпилировать текст сервера таким образом: **Пуск | Программы | Visual Studio Net 2003 | Command Prompt** и ввести команду компиляции:

```
csc server.cs
```

Теперь нужно создать новый проект — приложение клиента. Клиент должен знать сетевой адрес сервера и номер порта. Вот его текст (листинг 4.13).

Листинг 4.13. Приложение клиента

```
using System;
using System.IO;
using Sysem.Net.Sockets;
class clientexample
{
 public static void Main()
 {
  TcpClient newSocket=
  new TcpClient("localhost",50001); // Создаем объект типа
                                    // TcpClient
                                    // для связи с сервером
```

```
  NetworkStream ns= newSocket.GetStream(); // ns — потоковая
   // переменная на базе сокета
  // Чтение массива байтов от сервера из входного потока:
  byte [] buf= new byte [20];
  ns.Read(buf,0,10);
  char [] buf2= new char[20]; // Теперь выполним
                              // преобразование байтов
                              // в тип char:
  for(int i=0; i<20;i++)
  {
   buf2[i]= (char) buf[i];
  }
  Console.WriteLine(buf2); // Выводим строку от сервера
                           // на консоль клиента
  ns.Close();
  newSocket.Close();
 }
}
```

Скомпилируйте приложение клиента:

```
csc clientexample.cs
```

Вот собственно и все.

В результате компиляции приложений клиента и сервера строятся файлы с расширением exe.

Сначала из командной строки запустите программу-сервер (рис. 4.8), затем отдельно запустите клиента (рис. 4.9).

```
The FAR manager, version 1.70 beta 5 (build
Copyright (C) 1996-2000 Eugene Roshal, Copy
Пожалуйста, зарегистрируйте Вашу копию
D:\msdev\csharpprojects\Server>server.exe
Press To Stop Server
_
```

Рис. 4.8. Запуск приложения сервера

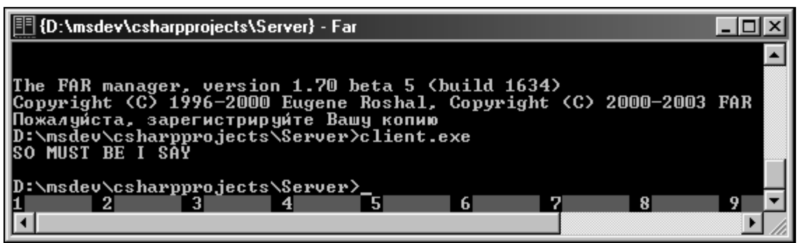


Рис. 4.9. Запуск приложения клиента

Убедитесь в правильности работы программы.

Имеется возможность связи клиента и сервера через протокол HTTP (этот протокол использует Internet Explorer). Рассмотрим, как осуществить такую связь. Отметим, что теперь нет необходимости указывать номер порта. Сервер реализуется как удаленный объект, так что нам потребуется посредник между этим объектом и клиентом — так называемый *интерфейс*. В интерфейсе нужно объявить методы сервера, которые мы хотим сделать доступными из клиента. Вот пример очень простого интерфейса:

```
using System;
public interface ISimpleObject
{
 String ToUpper(String inString);
}
```

Мы видим, что единственным методом сервера является метод `ToUpper`, который получает в качестве входного аргумента строку `inString`. Реализацию данного метода следует перенести в сервер. Вот, собственно, и само приложение сервера (листинг 4.14).

Листинг 4.14. Приложение сервера на основе протокола HTTP

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
namespace Server
```

```
{
 public class SimpleObject: MarshalByRefObject, ISimpleObject
 {
  public String ToUpper(String inString)
  {
   return(inString.ToUpper());
  }

 class Example8_8
 {
  static void Main(string[] args)
   {
    //
    // TODO: Add code to start application here
    HttpChannel channel = new HttpChannel(54321);
    ChannelServices.RegisterChannel(channel);
    RemotingConfiguration.RegisterWellKnownServiceType(
    typeof(SimpleObject),
    "SOEndPoint", WellKnownObjectMode.Singleton);
    Console.WriteLine("Press To Stop Server");
    Console.ReadLine();
   }
 }
}
```

Этот сервер не совсем обычен. Обратим внимание на то, что в объявление класса сервера подключены уже созданный нами интерфейс и еще один интерфейс: `MarshalByRefObject`. Дальше следует реализация метода `ToUpper()`. Метод `Main()` сервера занимается созданием соединения с клиентом:

```
HttpChannel channel = new HttpChannel(54321); // Создаем канал
// связи снова через порт!
ChannelServices.RegisterChannel(channel); // Регистрируем
// канал в операционной системе
```

```
RemotingConfiguration.RegisterWellKnownServiceType(
 typeof(SimpleObject),
 "SOEndPoint", WellKnownObjectMode.Singleton);
//определяем тип соединения "Simple Object End Point"
// и режим
// его работы Singleton (обслуживание одного клиента
// одним удаленным объектом)
```

Наконец, рассмотрим сторону клиента (листинг 4.15).

Листинг 4.15. Приложение клиента на основе протокола HTTP

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
class client
{
 public static void Main()
 {
  HttpChannel h1=new HttpChannel(0);
  ChannelServices.RegisterChannel(h1);
  Object remoteOb=
  RemotingServices.Connect(typeof(ISimpleObject),
   "http://localhost:54321/SOEndPoint");
   ISimpleObject so=remoteOb as ISimpleObject;
  Console.WriteLine(so.ToUpper("so must be I say"));
 }
}
```

Клиент должен получить доступ к удаленному объекту сервера. Это делается с помощью команды:

```
RemotingServices.Connect(typeof(ISimpleObject),
 "http://localhost:54321/SOEndPoint");
```

Здесь использовано *сетевое имя* `localhost` и через двоеточие — номер порта. Сетевое имя уникально идентифицирует компьютер в сети. Команда:

```
ISimpleObject so=remoteOb as ISimpleObject;
```

создает у клиента экземпляр объекта сервера. Отметим при этом, что используется имя интерфейса, а имя класса сервера вовсе не задействуется. Ну и наконец, обращение к методу выполняется так:

```
Console.WriteLine(so.ToUpper("so must be I say"));
```

Не правда ли, весьма оригинальное взаимодействие двух распределенных приложений!

Задание

Напишите свои собственные приложения клиента и сервера, которые обмениваются:

- текущим временем;
- содержимым файла, имя которого передает серверу клиент.

Контрольные вопросы

1. Объяснить программы из листингов 4.12 и 4.13.
2. Как передать информацию от клиента серверу?
3. Как вы понимаете работу удаленного объекта на сервере?
4. В чем особенности клиент-серверного приложения на основе протокола HTTP?

Работа с классом таймера

Цель занятия

Научиться использовать *класс таймера* для обеспечения динамики в программе. Надлежит выяснить, как объявлять таймер, а также запускать ту или иную процедуру по событию от таймера. В качестве приложения с таймером в этом занятии реализована программа "бегущая строка". Более подробную информацию можно получить в [9].

Краткие теоретические сведения

Таймер — это механизм, который позволяет через указанное время многократно активизировать некоторую процедуру. Работу с таймером мы проводили и в Java, и в JavaScript. Рассмотрим, как реализовать *бегущую строку* на форме. Для этого разместим на форме две кнопки, одна из которых запускает бегущую строку (**StartTimer** (рис. 4.10)).

Рис. 4.10. Форма для демонстрации бегущей строки

Для работы с таймером необходимо подключить следующие библиотеки:

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Threading;
```

Последняя из этого списка как раз и содержит класс `Timer`. Объявим переменную `t` класса `Timer`, а также саму бегущую строку и ее координаты в классе так:

```
public string str;
public int xcoord;
```

```
public int ycoord;
private System.Threading.Timer t;
```

Объявление переменной класса таймера должно полностью определять путь к этому классу. Например, не пройдет такое указание:

```
private Timer t;
```

Теперь запрограммируем кнопку для запуска таймера так:

```
private void button1_Click(object sender, System.EventArgs e)
{
 System.Threading.TimerCallback tc=new
 System.Threading.TimerCallback(showstr);
 t= new System.Threading.Timer(tc,null,100,350);
}
```

Здесь используется так называемый класс `delegate` (делегат) `TimerCallback`, которому следует передать в качестве аргумента название метода, запускаемого по таймеру, что, собственно, и объявляет инструкция:

```
t= new System.Threading.Timer(tc,null,100,350);
```

Инструкция указывает, что таймер следует запустить через 100 мс первый раз, а затем запускать каждые 350 мс. Метод `showstr()` выводит строку на экран и объявляется таким образом:

```
public void showstr(Object state)
{
 Graphics g = this.CreateGraphics();
 this.Refresh();
 Font drawFont = new Font("Times New Roman", 16);
 SolidBrush drawBrush = new SolidBrush(Color.Red);
 g.DrawString(str,drawFont,drawBrush,xcoord,ycoord);
 xcoord+=5; // Наращивание координаты по оси X для вывода
 // строки при очередном запуски этого метода
}
```

Вывод строки производит команда:

```
g.DrawString(str,drawFont,drawBrush,xcoord,ycoord);
```

Ее аргументами являются выводимая строка `str`, шрифт — `drawFont`, кисть (цвет) — `drawBrush` и координаты. Переменные `drawFont` и `drawBrush` созданы с помощью своих конструкторов так:

```
Font drawFont = new Font("Times New Roman", 16);
SolidBrush drawBrush = new SolidBrush(Color.Red);
```

После отображения строки увеличиваем координату по оси *X* для последующего вывода строки при очередном срабатывании таймера. Отметим также, что прежде чем рисовать строку, нужно очистить форму по команде:

```
this.Refresh();
```

Отображение текста, как и рисование, выполняется на базе методов класса `Graphics`. Поэтому мы создали переменную графического контекста таким образом:

```
Graphics g = this.CreateGraphics();
```

Наконец, объявление метода `showstr()` требует обязательно указывать в качестве аргумента переменную типа `Object` (таково свойство делегатов).

Кнопка **ClearForm** запрограммирована таким образом:

```
private void button2_Click(object sender, System.EventArgs e)
{
 this.Refresh();
 t.Dispose(); //удаление таймера
 t=null;
 xcoord=20;
}
```

Во-первых, она очищает экран, а во-вторых, производит уничтожение таймера: `t.Dispose()` и его деинициализацию: `t=null`.

Суммируя все сказанное, получаем следующую итоговую программу (листинг 4.16).

Листинг 4.16. Приложение типа "бегущая строка"

```
using System;
using System.Drawing;
```

```
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Threading;

namespace Timerapp
{
 /// <summary>
 /// Summary description for Form1.
 /// </summary>
 public class Form1 : System.Windows.Forms.Form
 {
  public string str;
  public int xcoord;
  public int ycoord;
  private System.Threading.Timer t;
  private System.Windows.Forms.Button button1;
  private System.Windows.Forms.Button button2;
  /// <summary>
  /// Required designer variable.
  /// </summary>
  private System.ComponentModel.Container components = null;

  public Form1()
  {
   //
   // Required for Windows Form Designer support
   //
   InitializeComponent();
   str="Hello";
   xcoord=20;
   ycoord=100;
   //
```

```
 // TODO: Add any constructor code after InitializeComponent
 // call
 //
}

/// <summary>
/// Clean up any resources being used.
/// </summary>
public void showstr(Object state)
{
 Graphics g = this.CreateGraphics();
 this.Refresh();
 Font drawFont = new Font("Times New Roman", 16);
 SolidBrush drawBrush = new SolidBrush(Color.Red);
 g.DrawString(str,drawFont,drawBrush,xcoord,ycoord);
 xcoord+=5;
}

protected override void Dispose( bool disposing )
{
 if( disposing )
  {
   if (components != null)
    {
     components.Dispose();
    }
  }
 base.Dispose( disposing );
}

#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
```

```
private void InitializeComponent()
{
 this.button1 = new System.Windows.Forms.Button();
 this.button2 = new System.Windows.Forms.Button();
 this.SuspendLayout();
 //
 // button1
 this.button1.Location = new System.Drawing.Point(16, 232);
 this.button1.Name = "button1";
 this.button1.Size = new System.Drawing.Size(72, 32);
 this.button1.TabIndex = 0;
 this.button1.Text = "StartTimer";
 this.button1.Click += new
 System.EventHandler(this.button1_Click);
 //
 // button2
 //
 this.button2.Location = new System.Drawing.Point(120, 232);
 this.button2.Name = "button2";
 this.button2.Size = new System.Drawing.Size(88, 32);
 this.button2.TabIndex = 1;
 this.button2.Text = "ClearForm";
 this.button2.Click += new
 System.EventHandler(this.button2_Click);
 //
 // Form1
 //
 this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
 this.ClientSize = new System.Drawing.Size(292, 273);
 this.Controls.Add(this.button2);
 this.Controls.Add(this.button1);
 this.Name = "Form1";
 this.Text = "Form1";
 this.ResumeLayout(false);
}
```

```
  #endregion

  /// <summary>
  /// The main entry point for the application.
  /// </summary>
  [STAThread]
  static void Main()
  {
   Application.Run(new Form1());
  }

  private void button1_Click(object sender,
  System.EventArgs e)
  {
   System.Threading.TimerCallback tc=new
   System.Threading.TimerCallback(showstr);
   t= new System.Threading.Timer(tc,null,100,350);
  }

  private void button2_Click(object sender,
  System.EventArgs e)
  {
   this.Refresh();
   t.Dispose();
   t=null;
   xcoord=20; // Вывод строки стартует с этой координаты
  }
 }
}
```

Задание

Построить:

- две бегущие строки;
- бегущую картинку;
- анимированную картинку.

Контрольные вопросы

1. Для чего нужен таймер и что он позволяет сделать?
2. Как объявить переменную таймера, можно ли объявить несколько таймеров в одном приложении?
3. Как работает таймер?
4. Объясните программу листинга 4.16.
5. Вспомните, как использовался таймер в приложениях Java и скриптах JavaScript?

Обработка польской записи

Цель занятия

Закрепить на практике знания по работе со строками. Изучить возможности представления командных строк в польской записи. Освоить принципы работы со стеком. Дополнительные сведения можно найти в [6, 12].

Краткие теоретические сведения

Польская запись — это запись арифметических выражений, в которой знак операции записывается после аргументов, например:

```
a b + // Это, конечно, a+b
a c + b * // Это (a+c)*b
```

Достоинство польской записи в том, что она не использует скобки. Наша задача будет состоять в том, чтобы программно вычислить значение простого арифметического выражения, представленного в польской записи, которое использует только целые числа и знаки операций +, − и * (умножить). Следовательно, мы должны научиться вычислять, к примеру, выражения типа такого: 9 45 7 + *.

Несколько упростим себе задачу, используя в качестве разделителей между *термами* выражения не пробел, а запятую. Таким образом, предыдущая запись превратится в такую: 9,45,7,+,*. Эта запись соответствует обычному представлению вида: 9 * (45 + 7) (ответ: 468).

На форме разместим одно текстовое поле для ввода польской записи, другое текстовое поле — для вывода ответа и кнопку для выполнения расчета. Именно на кнопку и "ложится" вся логика нашего приложения. Наличие разделителя "," позволяет использовать очень "сильную" команду `Split` , которая формирует массив лексем (читай — термов), из которых состоит строка. Наша экранная форма должна иметь следующий вид (рис. 4.11).

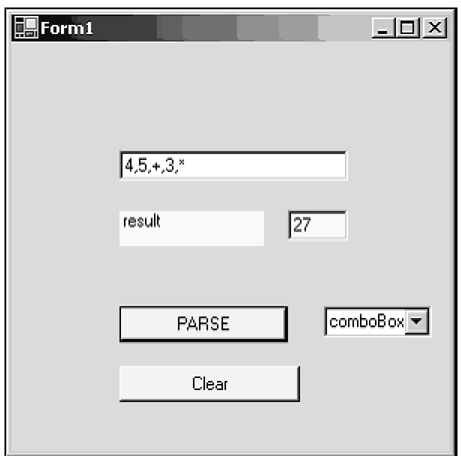


Рис. 4.11. Окно приложения для работы с польской записью

Кнопка **PARSE** осуществляет вычисление польской записи. Кнопка **Clear** очищает текстовые поля и список. В списке хранятся все разобранные лексемы входного выражения. *Лексемы* — это "строительные" элементы выражения: переменные и знаки операций в нашем случае.

Разбор польской записи выполняется с помощью *стека* — особого вида памяти, для которой возможны следующие операции:

- записать в стек;
- прочитать (вытолкнуть) из стека;
- очистить стек.

В стеке доступен только верхний элемент (иначе — верхушка стека). При записи очередного элемента остальные проталкиваются

вглубь стека на одну ячейку, а новый записывается в верхушку. При чтении (выталкивании), напротив, каждый элемент поднимается на одну ячейку вверх.

Алгоритм разбора выполняется таким образом:

(i) Прочитать очередную лексему из входной строки.

(ii) Если эта лексема не является операцией, то записать ее в стек, иначе — переходим к (iii).

(iii) Вытолкнуть из стека столько элементов, сколько должно участвовать в данной операции (у нас выталкивается два элемента). Выполнить над ними операцию, а результат загрузить обратно в стек.

(iv) Перейти к шагу (i), если в строке есть еще лексемы, иначе — конец.

Сначала приведем всю программу (листинг 4.17), реализующую этот алгоритм, затем разберем ее подробно.

Листинг 4.17. Приложение для обработки польской записи

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace polish
{
 /// <summary>
 /// Summary description for Form1.
 /// </summary>
 public class Form1 : System.Windows.Forms.Form
 {
  public string initstr;
  public string[] masstr;
  private System.Windows.Forms.TextBox textBox1;
```

```
private System.Windows.Forms.Button button1;
private System.Windows.Forms.TextBox textBox2;
private System.Windows.Forms.Label label1;
private System.Windows.Forms.ComboBox comboBox1;
private System.Windows.Forms.Button button2;
/// <summary>
/// Required designer variable.
/// </summary>
private System.ComponentModel.Container components = null;

public Form1()
{
 // Required for Windows Form Designer support
 //
 InitializeComponent();
 //
 // TODO: Add any constructor code after InitializeComponent
 // call
 //
}

/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
 if( disposing )
  {
   if (components != null)
    {
     components.Dispose();
    }
  }
 base.Dispose( disposing );
}
```

```
#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent() // Инициализация формы —
// это совокупность действий, выполненных системой, поэтому
// здесь на ней не следует останавливаться
{
 this.textBox1 = new System.Windows.Forms.TextBox();
 this.button1 = new System.Windows.Forms.Button();
 this.textBox2 = new System.Windows.Forms.TextBox();
 this.label1 = new System.Windows.Forms.Label();
 this.comboBox1 = new System.Windows.Forms.ComboBox();
 this.button2 = new System.Windows.Forms.Button();
 this.SuspendLayout();
 //
 // textBox1
 //
 this.textBox1.Location = new System.Drawing.Point(72, 72);
 this.textBox1.Name = "textBox1";
 this.textBox1.Size = new System.Drawing.Size(152, 20);
 this.textBox1.TabIndex = 0;
 this.textBox1.Text = "";
 //
 // button1
 //
 this.button1.BackColor =
 System.Drawing.Color.FromArgb(((System.Byte)(255)),
 ((System.Byte)(255)), ((System.Byte)(128)));
 this.button1.Location = new System.Drawing.Point(72, 176);
 this.button1.Name = "button1";
 this.button1.Size = new System.Drawing.Size(112, 24);
 this.button1.TabIndex = 1;
```

```
this.button1.Text = "PARSE";
this.button1.Click += new
System.EventHandler(this.button1_Click);
//
// textBox2
//
this.textBox2.BackColor =
System.Drawing.Color.FromArgb(((System.Byte)(255)),
((System.Byte)(255)), ((System.Byte)(192)));
this.textBox2.Location = new System.Drawing.Point(184,
112);
this.textBox2.Name = "textBox2";
this.textBox2.Size = new System.Drawing.Size(40, 20);
this.textBox2.TabIndex = 2;
this.textBox2.Text = "";
//
// label1
//
this.label1.BackColor =
System.Drawing.Color.FromArgb(((System.Byte)(255)),
((System.Byte)(255)), ((System.Byte)(192)));
this.label1.Location = new System.Drawing.Point(72, 112);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(96, 24);
this.label1.TabIndex = 3;
this.label1.Text = "result";
//
// comboBox1
//
this.comboBox1.Location = new System.Drawing.Point(208,
176);
this.comboBox1.Name = "comboBox1";
this.comboBox1.Size = new System.Drawing.Size(72, 21);
this.comboBox1.TabIndex = 4;
this.comboBox1.Text = "comboBox1";
```

```
    //
    // button2
    //
    this.button2.BackColor =
    System.Drawing.Color.FromArgb(((System.Byte)(255)),
    ((System.Byte)(255)), ((System.Byte)(128)));
    this.button2.Location = new System.Drawing.Point(72, 216);
    this.button2.Name = "button2";
    this.button2.Size = new System.Drawing.Size(120, 24);
    this.button2.TabIndex = 5;
    this.button2.Text = "Clear";
    this.button2.Click += new
    System.EventHandler(this.button2_Click);
    //
    // Form1
    //
    this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
    this.BackColor =
    System.Drawing.Color.FromArgb(((System.Byte)(255)),
    ((System.Byte)(192)), ((System.Byte)(192)));
    this.ClientSize = new System.Drawing.Size(292, 273);
    this.Controls.Add(this.button2);
    this.Controls.Add(this.comboBox1);
    this.Controls.Add(this.label1);
    this.Controls.Add(this.textBox2);
    this.Controls.Add(this.button1);
    this.Controls.Add(this.textBox1);
    this.Name = "Form1";
    this.Text = "Form1";
    this.ResumeLayout(false);
  }
  #endregion

  /// <summary>
  /// The main entry point for the application.
```

```
/// </summary>
[STAThread]
static void Main()
{
 Application.Run(new Form1());
}

private void button1_Click(object sender,
System.EventArgs e)
{
 initstr=textBox1.Text; // Это — разбираемое выражение
 masstr=initstr.Split(','); // masstr — массив, в который
                            // записываются лексемы
 for(int j=0;j<masstr.Length;j++)
  { comboBox1.Items.Add(masstr[j]); } // В комбо-список
                                      // заносим
                                      // все распознанные
                                      // лексемы
 string[]stack=new string[masstr.Length];
 decimal res=0; // В переменной res получаем окончательный
                // результат
 for( int i=0;i<masstr.Length;i++) // В этом цикле
 // выполняется разбор и обработка польской записи,
 // лексемы которой сохранены в массиве строк masstr
 {
  if ((masstr[i]=="+")|(masstr[i]=="-")| (masstr[i]=="*")|
  (masstr[i]=="/"))
   {
    string s1=stack[1];
    string s0=stack[0];
    for (int iw=2;iw<stack.Length;iw++)
    { stack[iw-1]=stack[iw]; }
    decimal i1=Convert.ToDecimal(s1);
    decimal i2=Convert.ToDecimal(s0);
    switch (masstr[i])
```

```
      {
       case "+":
         res=i1+i2;
         stack[0]=""+res;
         break;
       case "-":
         res=i1-i2;
         stack[0]=""+res;
         break;
       case "*":
         res=i1*i2;
         stack[0]=""+res;
         break;
       case "/":
         res=i1/i2;
         stack[0]=""+res;
         break;
      }
     }
    else
     {
      if(stack.Length==0)
       {stack[0]=masstr[i];}
      else
       {
        for (int j=0;j<stack.Length-1;j++)
         stack[j+1]=stack[j];
        stack[0]=masstr[i];
       }
     }
  }
  textBox2.Text=""+res;
 }
```

```
  private void button2_Click(object sender,
  System.EventArgs e)
  {
   initstr="";
   textBox1.Text="";
   textBox2.Text="";
   comboBox1.Items.Clear();
  }
 }
}
```

Самая простая — это кнопка **Clear**:

```
private void button2_Click(object sender, System.EventArgs e)
{
 initstr="";
 textBox1.Text="";
 textBox2.Text="";
 comboBox1.Items.Clear();
}
```

Пояснять ее, вероятно, не следует. Обратимся к кнопке **PARSE**. Сначала мы формируем массив лексем:

```
initstr=textBox1.Text;  // Это — разбираемое выражение
masstr=initstr.Split(',');
```

Отметим, что в качестве разделителя используется запятая.

Разбор лексем выполняется так. Если встречается число, то оно грузится в стек — массив `stack`. Если встречается знак операции, то выполняются следующие команды:

```
if ((masstr[i]=="+")|(masstr[i]=="-")| (masstr[i]=="*") |
(masstr[i]=="/")) // "|" — Знак "ИЛИ"; выполняется проверка
                  // того, что лексема суть операция
 { // Встретился знак операции
  string s1=stack[1];
```

```
  string s0=stack[0]; // Выгружаем из стека верхние два
                      // операнда
  for (int iw=2;iw<stack.Length;iw++) // Передвигаем остальные
  // операнды в стеке наверх на одну ячейку
  {
   stack[iw-1]=stack[iw];
  }
  decimal i1=Convert.ToDecimal(s1); // Преобразуем прочитанные
  // два операнда из стека в вещественные числа — тип decimal
  decimal i2=Convert.ToDecimal(s0);
  switch (masstr[i])
  { // Анализируем знак операции
    case "+":
      res=i1+i2; // Если +, то складываем
      stack[0]=""+res;
      break;
    case "-":
      res=i1-i2; // Если -, то вычитаем
      stack[0]=""+res;
      break;
    case "*":
     res=i1*i2; // Умножение для знака *
     stack[0]=""+res;
     break;
    case "/":
      res=i1/i2;  // Деление для знака /
      stack[0]=""+res;
      break;
  }
}
```

Если лексема не оператор, то ее загружаем в стек:

```
...
else
```

```
 {
  if(stack.Length==0)
   {stack[0]=masstr[i];}
  else
   {
    for (int j=0;j<stack.Length-1;j++)
     stack[j+1]=stack[j];
    stack[0]=masstr[i];
   }
 }
}
```

Если стек не пуст, то загрузку выполняем так:

```
for (int j=0;j<stack.Length-1;j++) stack[j+1]=stack[j];
// Элементы проталкиваем на следующую ячейку стека
stack[0]=masstr[i]; // Считанная лексема грузится в верхушку
                    // стека
```

Итак, основная процедура обработки польского выражения нами последовательно разобрана.

Задание

Выполните алгоритм разбора польской записи сначала вручную и убедитесь в том, что вы его хорошо понимаете. Затем разберитесь в программе.

Требуется расширить функциональные возможности программы следующим образом:

- разделителем лексем должен быть символ пробела;
- программа должна обнаруживать неправильные синтаксические конструкции типа 2 + 3 *;
- числа должны быть вещественными с фиксированной точкой: например: 20.6;
- предусмотреть возможность работы с отрицательными числами;
- предусмотреть возможность использования операторов присваивания. В этом случае вам понадобится хранить значения

переменных, например, используя для этих целей еще один массив с двумя столбцами: в первом столбце записываем имена переменных, во втором — их значения.

Контрольные вопросы

1. Какое главное достоинство польской записи?
2. Как обрабатываются выражения, представленные в польской записи?
3. Какие операции можно выполнять на стеке?
4. Объясните программу листинга 4.17.
5. Подумайте над тем, как получить польскую запись из обычного выражения со скобками?

Работа с коллекциями

Цель занятия

Освоить работу с коллекциями на примере класса `ArrayList`. Изучить методы этого класса. Создать и разобраться в демонстрационном приложении. Дополнительно см. [9, 17].

Краткие теоретические сведения

Класс `ArrayList` позволяет хранить объекты разных типов. Мы создадим на его основе внутреннюю базу данных. Эта база данных будет содержать сведения о студентах: имя и группу. Простейший интерфейс с базой данных будет реализован на основе формы (рис. 4.12).

Имеется всего две кнопки: одна (**Add**) для добавления записи в базу, другая (**Find**) — для поиска номера группы по имени. Записи будем создавать на основе класса `Student`, объявленного следующим образом:

```
class Student
{
 public string name;
 public int group;
}
```

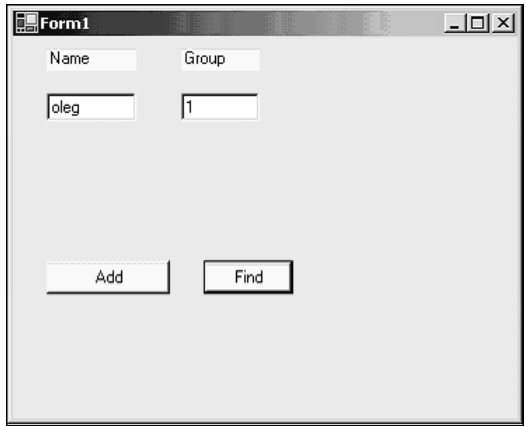


Рис. 4.12. Окно приложения для работы с классом `ArrayList`

Этот класс содержит два поля: `name` (имя) и `group` (номер группы). Обратим внимание на то, что класс `ArrayList` позволяет хранить объекты разных классов. Это его большое достоинство. Дальше приводим реакцию на нажатие кнопки **Add**:

```
private void button1_Click(object sender, System.EventArgs e)
{
 Student      stud=new Student(); // Создаем объект
 stud.name=textBox1.Text;
 stud.group=Convert.ToInt32(textBox2.Text);
 myal.Add(stud); // Добавляем объект в коллекцию
 MessageBox.Show("Ready");
}
```

Видим, что сначала создается объект типа `Student`. Затем его поля `name` и `group` получают значения, а затем объект `stud` добавляется в коллекцию `myal`, созданную на основе класса `ArrayList` при начальной загрузке формы:

```
private void Form1_Load(object sender, System.EventArgs e)
{
 myal=new ArrayList();
}
```

Наконец, по нажатию кнопки **Find** реализуется следующий код:

```
private void button2_Click(object sender, System.EventArgs e)
{
 int priz=0;
 IEnumerator inum=myal.GetEnumerator();
 while(inum.MoveNext())
 {
  Student studcur=(Student) inum.Current;
  String s1=studcur.name.Trim();
  String s2=textBox1.Text.Trim();
  if(String.Compare(s1,s2)==0)
   {
    textBox2.Text=""+studcur.group;
    priz=1;
    break;
   }
  continue;
 }
 if(priz==0)
  MessageBox.Show("Not Found");
}
```

Этот код требует дополнительных разъяснений. Прежде всего, заметим, что поиск в *коллекции объектов* реализуется путем последовательного перехода к очередному объекту:

```
while(inum.MoveNext()) { . . .  }
```

Для этого нам необходимо создать переменную класса `IEnumerator`:

```
IEnumerator inum=myal.GetEnumerator();
```

Свойство `Current` переменной `inum` дает очередной объект типа `Student`. Затем мы просто проверяем на совпадение имени в текстовом поле `textBox1` и значения поля `name` текущей записи:

```
String s1=studcur.name.Trim();
String s2=textBox1.Text.Trim();
if(String.Compare(s1,s2)==0)
```

Класс `ArrayList` позволяет выполнить вставку объекта `ob` в указанную позицию `i` командой:

```
myal.Insert(int i, Object ob);
```

Очистка списка выполняется командой

```
myal.Clear();
```

Удаление объекта на позиции `i` из списка реализует команда

```
myal.RemoveAt(int i);
```

Позицию объекта `Ob` в списке определяет команда:

```
int i = myal.IndexOf(Ob);
```

Если объекта в списке нет, то возвращается значение −1. В этих примерах `myal` является объектом типа `ArrayList`.

Теперь приводим полный текст программы (листинг 4.18) рассмотренного приложения.

Листинг 4.18. Работа с `ArrayList`

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace WindowsApplication5
{
 /// <summary>
 /// Summary description for Form1.
 /// </summary>

 public class Form1 : System.Windows.Forms.Form
 {
  class Student // Объявление класса Student, объекты которого
                // попадают в ArrayList
  {
```

```
   public string name; // Поля класса Student
   public int group;
  }

  private ArrayList myal;
  // Объявления элементов формы, сделанные системой
  private System.Windows.Forms.Button button1;
  private System.Windows.Forms.TextBox textBox1;
  private System.Windows.Forms.TextBox textBox2;
  private System.Windows.Forms.Label label1;
  private System.Windows.Forms.Label label2;
  private System.Windows.Forms.Button button2;
  /// <summary>
  /// Required designer variable.
  /// </summary>
  private System.ComponentModel.Container components = null;

  public Form1() // Конструктор формы
  {
   //
   // Required for Windows Form Designer support
   //
   InitializeComponent();
   //
   // TODO: Add any constructor code after InitializeComponent
   // call
   //
  }
  /// <summary>
  /// Clean up any resources being used.
  /// </summary>
  protected override void Dispose( bool disposing )
  {
   if( disposing )
```

```
    {
     if (components != null)
      { components.Dispose(); }
    }
   base.Dispose( disposing );
  }

  #region Windows Form Designer generated code
  /// <summary>
  /// Required method for Designer support - do not modify
  /// the contents of this method with the code editor.
  /// </summary>
  private void InitializeComponent() // Код инициализации
                                     // формы, построенный
                                     // системой
  {
   this.button1 = new System.Windows.Forms.Button();
   this.textBox1 = new System.Windows.Forms.TextBox();
   this.textBox2 = new System.Windows.Forms.TextBox();
   this.label1 = new System.Windows.Forms.Label();
   this.label2 = new System.Windows.Forms.Label();
   this.button2 = new System.Windows.Forms.Button();
   this.SuspendLayout();
   //
   // button1
   //
   this.button1.BackColor =
   System.Drawing.Color.FromArgb(((System.Byte)(255)),
   ((System.Byte)(224)), ((System.Byte)(192)));
   this.button1.Location = new System.Drawing.Point(24, 160);
   this.button1.Name = "button1";
   this.button1.Size = new System.Drawing.Size(88, 24);
   this.button1.TabIndex = 0;
   this.button1.Text = "Add";
```

```
this.button1.Click += new
System.EventHandler(this.button1_Click);
//
// textBox1
//
this.textBox1.Location = new System.Drawing.Point(24, 40);
this.textBox1.Name = "textBox1";
this.textBox1.Size = new System.Drawing.Size(64, 20);
this.textBox1.TabIndex = 1;
this.textBox1.Text = "";
//
// textBox2
//
this.textBox2.Location = new System.Drawing.Point(120, 40);
this.textBox2.Name = "textBox2";
this.textBox2.Size = new System.Drawing.Size(56, 20);
this.textBox2.TabIndex = 2;
this.textBox2.Text = "";
//
// label1
//
this.label1.BackColor =
System.Drawing.Color.FromArgb(((System.Byte)(255)),
((System.Byte)(224)), ((System.Byte)(192)));
this.label1.Location = new System.Drawing.Point(24, 8);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(64, 16);
this.label1.TabIndex = 3;
this.label1.Text = "Name";
//
// label2
//
this.label2.BackColor =
System.Drawing.Color.FromArgb(((System.Byte)(255)),
```

```
((System.Byte)(224)), ((System.Byte)(192)));
this.label2.Location = new System.Drawing.Point(120, 8);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(56, 16);
this.label2.TabIndex = 4;
this.label2.Text = "Group";
//
// button2
//
this.button2.BackColor =
System.Drawing.Color.FromArgb(((System.Byte)(255)),
((System.Byte)(224)), ((System.Byte)(192)));
this.button2.Location = new System.Drawing.Point(136, 160);
this.button2.Name = "button2";
this.button2.Size = new System.Drawing.Size(64, 24);
this.button2.TabIndex = 5;
this.button2.Text = "Find";
this.button2.Click += new
System.EventHandler(this.button2_Click);
//
// Form1
//
this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
this.BackColor =
System.Drawing.Color.FromArgb(((System.Byte)(192)),
((System.Byte)(255)), ((System.Byte)(192)));
this.ClientSize = new System.Drawing.Size(360, 273);
this.Controls.Add(this.button2);
this.Controls.Add(this.label2);
 this.Controls.Add(this.label1);
this.Controls.Add(this.textBox2);
this.Controls.Add(this.textBox1);
this.Controls.Add(this.button1);
this.Name = "Form1";
this.Text = "Form1";
```

```
 this.Load += new System.EventHandler(this.Form1_Load);
 this.ResumeLayout(false);
}
#endregion

/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static void Main()
{
 Application.Run(new Form1());
}

private void Form1_Load(object sender, System.EventArgs e)
{
 myal=new ArrayList(); // Создаем объект типа ArrayList
 // при загрузке формы
}

private void button1_Click(object sender,
System.EventArgs e)
// Добавление объекта в список ArrayList
{
 Student stud=new Student(); // Создаем объект stud
 stud.name=textBox1.Text; // Записываем значения полей name,
                          // group
 stud.group=Convert.ToInt32(textBox2.Text);
 myal.Add(stud);
 MessageBox.Show("Ready");
}

private void button2_Click(object sender,
System.EventArgs e)
// Поиск объекта по имени name
```

```
  {
   int priz=0; // Переменная priz равна 1, если найден нужный
               // объект в списке
   IEnumerator inum=myal.GetEnumerator(); // inum выполняет
                                          // переход к новому
                                          // объекту
   while(inum.MoveNext()) // Организуем цикл просмотра
                          // объектов списка
    {
     Student studcur=(Student) inum.Current; // studcur —
                                             // текущий объект
                                             // в списке
     String s1=studcur.name.Trim(); // Проверяем, совпадает ли
     // его поле name с искомым, заданным в текстовом поле
     // textBox1
     String s2=textBox1.Text.Trim();
     if(String.Compare(s1,s2)==0) // Оператор
     // String.Compare(s1,s2) возвращает 0, если строки
     // совпадают
      {
       textBox2.Text=""+studcur.group; // Отображаем номер
                                       // группы
       priz=1; // Объект найден - выходим из цикла по break
       break;
      }
     continue;
    }
  if(priz==0)
   MessageBox.Show("Not Found"); // Выводим сообщение, если
                                 // запись не найдена
  }
 }
}
```

Задание

В предлагаемом задании следует реализовать следующие возможности:

- добавить в программу возможность удалять элемент из списка;
- добавить в программу возможность поиска по номеру группы;
- добавить в программу возможность работы с базой только при предъявлении пароля. Пароль вводить в поле `TextBox`, предварительно задав его свойство `PasswordChar="*"`;
- при добавлении объекта в базу проверять наличие такого объекта и выдавать сообщение;
- предусмотреть возможность обновления записей в базе;
- реализовать возможность сериализации списка в файле и чтения его из файла.

Контрольные вопросы

1. Что такое коллекция?
2. Как добавлять, удалять и вставлять элементы в коллекцию?
3. Как выполнять поиск в коллекции?
4. Объясните программу листинга 4.18.

Сводка основных использованных команд C#

`Application.Run(new Form1())` — запуск класса `Form1` на выполнение (управление передается методу `Main()`).

`ArrayList arl= new ArrayList()` — создание пустой коллекции объектов.

`arl.Count` — значение числа объектов в списке.

`arl.Add(Object ob)` — добавление элемента в список.

`arl.Insert(int i, Object ob)` — вставка элемента в список в позицию `i`.

arl.RemoveAt(int i) — удаление объекта из списка, занимающего позицию i.

BinaryReader bir = new BinaryReader(ins) — открывает двоичный файл для чтения, где ins создается так: FileStream ins=File.OpenRead("c:\\my.dat").

BinaryWriter biw=new BinaryWriter(ous) — открывает двоичный файл для записи. Переменная ous — типа FileStream (*см.* FileStream).

biw.Write((decimal) 4.67) — запись в двоичный файл вещественного числа.

biw.Write((string) "Hello") — запись в двоичный файл строки.

Boss b1= new Boss() — создает объект класса Boss

Byte [] info = new UTF8Encoding(true).GetBytes("Hello") — заполняет байтовый массив info байтами, соответствующими слову "Hello".

comboBox1.dispose() — удаляет список.

comboBox1.Items.Add(s) — добавляет строку s в список comboBox1.

comboBox.Items.AddRange(new object[]{"a","b","c"}) — добавляет в список набор объектов, которыми являются строки "a","b","c".

comboBox1.Items.Clear() — очищает список.

comboBox1.Items.Insert("Hello",i) — вставляет строку "Hello" на позицию элемента с номером i.

comboBox1.Items.RemoveAt(comboBox1.SelectedIndex) — удаляет выделенный элемент из списка.

comboBox1.SelectedIndex=index — выделяет элемент списка с номером index.

controlx.Location=new System.Drawing.Point(184, 112) — установка позиции элемента controlx на форме.

controlx.Size = new System.Drawing.Size(64, 16) — установка размеров элемента с именем controlx на форме.

Console.WriteLine(str1) — выводит строку str1 на консоль.

`decimal i1=Convert.ToDecimal(s1)` — преобразование строки в вещественное число.

`int i= Convert.ToInt32(s1)` — преобразование строки в целое число.

`decimal d= bir.ReadDecimal()` — чтение вещественного числа из двоичного файла (*см.* `BinaryReader`).

`delegate double DelegateFunc( double x, string s)` — объявление функции-делегата с аргументами `x` и `s`.

`FileStream fs= File.Create(path,1024)` — создает потоковую переменную для низкоуровневой работы с файлом на основе байтов.

`fs.Write(info,0,info.Length)` — записывает в файл `fs` массив байтов `info`, начиная с нулевого байта и заканчивая последним байтом массива.

`File.Create(path,size)` — создает файл с именем `path` и размером `size`.

`File.Delete(path)` — уничтожает файл `path`.

`FileStream ous = File.Create("c:\\my.dat")` — создание файла.

`FileStream fout= File.OpenWrite(path)` — открывает файл для вывода.

`fout.Flush()` — сбрасывает содержимое буфера вывода в файл.

`fout.Close()` — закрывает файл.

`fout.WriteByte(byte(b))` — выводит байт в файл.

`get {return prop;}` — возвращает значение открытого свойства.

`goto labela` — переход на метку `labela`.

`IEnumerator inum= myal.GetEnumerator()` — создается объект `IEnumerator` для доступа к содержимому списка `ArrayList`.

```
while(inum.MoveNext())   // в цикле осуществляется
                        // последовательный доступ
                        // к объектам из списка
```

```
        {
          Student st=(Student) inum.Current; // получение
                                             // очередного объекта
       ...
         }
```

`if (File.Exists(path))` — проверяет существование файла `path`.

`Image my= new Bitmap(s)` — создает рисунок в памяти; `s` — адрес рисунка на диске.

`int index = arl.IndexOf(Object ob)` — возвращает позицию объекта `ob` в коллекции `ArrayList`. В качестве объекта может выступать, например, строка.

`int i= Convert.ToInt32(str1)` — преобразует строку `str1` в целое число `i`.

`int i=comboBox1.FindString(s)` — определяет номер строки списка, совпадающей со строкой `s`.

`int [] k= new int[10]` — создает массив для хранения 10 целых чисел.

`int [] k={5,6,7}` — явное создание массива из трех чисел.

`MessageBox.Show("Hello")` — вывод строки в диалоговом окне.

`OpenFileDialog dg= new OpenFileDialog()` — создание переменной `dg` типа файлового диалога. Проверка выбора имени файла выполняется так:

```
if (dlg.ShowDialog()==DialogResult.OK) {...}
```

`pictureBox1.Image=(Image) new Bitmap(@"c:\msdev\1.bmp")` — отображает в компоненте `pictureBox` картинку из файла `c:\msdev\1.bmp`

`ReadByte(b)` — чтение из файла байта `b` на основе метода низкоуровневого класса `FileStream`.

`set { marka=value;}` — установка значения открытого свойства через закрытый член класса `marka`.

`str1.IndexOf("страна")` — номер позиции в строке `str1`, с которой начинается подстрока `"страна"`.

`str1.Length` — длина строки `str1`.

`StreamReader sr= new StreamReader("c:\\my.txt")` — открытие потока для высокоуровневого чтения.

`str1=str1.ToLower()` — переводит строку в нижний регистр.

`str1=str1.ToUpper()` — переводит строку в верхний регистр.

`str1=str1.Trim()` — удаляет из строки начальные и хвостовые пробелы.

`String s=bir.ReadString()` — чтение строки из двоичного файла `BinaryReader bir`.

`String s= sr.ReadLine()` — высокоуровневое чтение.

`String s=str1.Remove(i1,i2)` — удаляет из строки `str1` подстроку, начиная с символа `i1` и включая последующие `i2` символов.

`String s=str1.Replace("abc","cba")` — получает из строки `str1` подстроку `s` заменой слова `"abc"` на слово `"cba"`.

`String str1= @"c:\work\my.dat"` — присваивает значение строке: `@` отменяет необходимость дублировать символ наклонной черты `\`.

`String [] strarray= s.Split(",")` — формирует массив из слов, входящих в `s` и разделенных символом запятой.

`String s= str1.Substring(i1,i2)` — строка `s` получает значение подстроки `str1` между символами с номерами `i1` и `i2`.

`String s= comboBox1.Items[comboBox1.SelectedIndex].ToString()` — чтение выделенного элемента списка.

`String s= String.Concat(s1,s2)` — строка `s` получает значение соединения строк `s1` и `s2`.

`String s= Console.ReadLine()` — выполняет чтение строки с консоли.

```
// Пример работы с классом StringBuilder
String sr= "Hello";
StringBuilder srb= new StringBuilder(sr); // создает
// переменную класса StringBuilder
```

```
srb.Append(", You — programmer of my heart ! ");
                              //добавляет к ней новую строку
srb.Insert(4, "Hello again"); // вставляет слова Hello
                                 // again после 4-го символа
srb.Remove(0,4); // удаляет первое слово "Hello"
sr=srb.ToString(); // выполняет присваивание строке sr
```

`System.Threading.TimerCallback tc=new System.Threading.TimerCallback(showstr)` — создается объект для запуска по таймеру метода `showstr`.

`System.Threading.Timer t=new System.Threading.Timer(tc, null,100,350)` — создается и запускается таймер спустя 100 мс, по которому каждые 350 мс запускается метод `showstr()`.

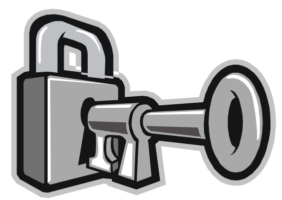
`textBox1.Text` — содержимое текстового поля с именем `textBox1`.

`textBox1.PasswordChar` — символ для замены вводимого символа при наборе пароля.

`this.BackColor=System.Drawing.Color.FromArgb( ((System.Byte)(255)),((System.Byte)(224)),(System.Byte) (100))` — установка цвета формы (предполагается, что `this` ссылается на форму).

`this.Controls.Add(this.label1)` — добавление на форму элемента с именем `label1` (здесь предполагается, что `this` является ссылкой на форму).

`WriteByte(b)` — запись в файл байта `b` на основе метода низкоуровневого класса `FileStream`.



Приложение

Описание компакт-диска

Содержимое компакт-диска

Папки, размещенные в каталоге Listings и имена которых начинаются со слова listing, содержат листинги программ на языках Java и C#.

В каталоге Java содержится дистрибутив Java SDK 1.4. Для инсталляции Java необходимо разархивировать с помощью утилиты WinRar содержимое этого каталога и выполнить программу setup.exe.

В каталоге Tomcat содержится Tomcat 3.2, не требующий инсталляции.

Инструкции по работе с листингами программ на Java и C#

Листинги содержат отлаженные приложения и исходные тексты программ. Листинги пронумерованы так же, как и в тексте книги. Например, listing_1_1 соответствует листингу 1.1 книги.

Не все листинги из книги приведены на CD. Например, задание по созданию бобов Java Beans не представлено на CD, поскольку

каждый раз при создании боба (читай: объекта ActiveX) система регистрирует его в системном реестре, т. е. делает системно-зависимым.

Имеется несколько дополнительных листингов, снабженных литерой в конце названия листинга.

Запуск классов Java

Запуск приложений на основе форм выполняется из среды командного редактора (например, Norton Commander или FAR manager) так:

```
java listing_1_1/proba
```

Папка listing_1_1 содержит файлы proba.java и proba.class.

В исходном тексте файла proba.java указывается самая первая строка:

```
package listing_1_1
```

В ней сообщается, что класс proba.class находится в папке listing_1_1.

Таким образом следует запускать все приложения на основе форм.

Запуск апплетов Java

Приложения на основе апплетов запускаются путем двойного щелчка на имени файла с расширением html. Например, из папки Listing_1_9 следует запустить апплет через файл firstapp.html .В этом HTML-файле задается тег `<applet>` и содержится указание имени класса Java-апплета в параметре `code`:

```
<html>
<applet code="firstapplet.class"
width=400
height=400>
</applet>
</html>
```

Для выполнения программ на языке Java следует создать текстовый файл в каком-нибудь редакторе, например, в стандартной программе Блокнот, а затем сохранить созданный файл, указав при сохранении тип **Все файлы** и введя в качестве имени файла название главного класса приложение в файле с расширением java, например: proba.java.

В качестве примера рассмотрим файл proba.java из папки Листинг_1_1:

```
package listing_1_1;
import java.awt.*;
import java.io.*;

public class proba
{
 static int b;
 static String sname;
 static char[] charray;

 public static void main(String args[])
 {
  charray = new char[20];
  int i=0;
  System.out.println("What is Your name?"); // Вывод
                                   // вопроса о Вашем имени

  try
   {
    while((b=System.in.read()) !=13)
    { charray[i++]=(char) b; }
   }
  catch(IOException e)
   {System.out.println("The error"+e);}

  sname=new String(charray);
```

```
    // Вывод строки приветствия:
    System.out.println("Hello, fellow-programmer, "+sname);
  }
}
```

Здесь главный класс объявлен как `public class proba`. Под главным классом мы понимаем здесь и далее класс, содержащий метод с именем `main()`. Поэтому создаваемый файл должен иметь имя proba.java.

Обращаем ваше внимание на то обстоятельство, что при использовании в качестве первой строки

```
package listing_1_1
```

мы тем самым указываем, что исходный текст Java-файла размещается в каталоге с именем listing_1_1. Этот каталог должен, в свою очередь, быть подкаталогом каталога с именем bin, содержащего программу java.exe.

Следовательно, строка для запуска компилятора Java будет иметь такой вид:

```
javac listing_1_1/proba.java
```

А строка для запуска класса на выполнение будет иметь такой вид:

```
java listing_1_1/proba
```

В этом втором запуске имя класса указывается без расширения.

В некоторых листингах, предложенных на данном CD, не предусмотрен выход из работающего приложения. В таком случае вам нужно использовать комбинацию клавиш <CTRL>+ +<ALT>+<DEL> для запуска диспетчера задач и снятия программы вручную.

Запуск приложений C#

Приложения C# имеют расширение exe, так что запуск их выполняется традиционным образом, а именно: щелчком мыши на имени исполняемого exe-файла. Для загрузки исходного файла в окно проекта следует запустить Visual Studio .NET, а в ней выбрать пакет C#.NET. В окне разработки проектов выбрать пункт

Open, а затем с помощью кнопки **Browse** найти нужный проект. Проекты имеют расширение csproj.

Установка Tomcat и Java

Для установки Java с CD вам следует разархивировать содержимое папки Java и выполнить программу setup.exe.

Для установки Tomcat setup.exe выполнять не надо. Достаточно разархивировать папку Tomcat, которую вы найдете на данном CD, и настроить файлы start.bat и config.xml так, как описано в книге.

Список литературы

1. Будилов В. Основы программирования для Интернета. — СПб.: Питер, 2003. — 716 с.
2. Гарнаев А., Гарнаев С. Web-программирование на Java и JavaScript. — СПб.: БХВ-Петербург, 2002. — 1022 с.
3. Гиббонз П. Платформа .NET для Java-программистов. — СПб.: Питер, 2003. — 326 с.
4. Дарнелл Рик. JavaScript. Справочник. — СПб.: Питер, 2000. — 192 с.
5. Дунаев С. Технологии Интернет-программирования. — СПб.: Питер, 2001. — 474 с.
6. Ирэ П. Объектно-ориентированное программирование с использованием С++. — Киев: ДиаСофт, 1995. — 478 с.
7. Лабор В. В. Си Шарп. Создание приложений для Windows. — Минск: Харвест, 2003. — 382 с.
8. Пирогов В. MS SQL Server 2000. Управление и программирование. — СПб.: Питер, 2005. —582 с.
9. Прайс Дж., Гандэрлой М. Visual C#.NET. Полное руководство. — Киев: Век, 2004. — 958 с.
10. Смирнов Н. Java 2 Enterprise. Основы практической разработки распределенных корпоративных приложений: — М.: Кудиц-Образ, 2002. — 236 с.
11. Хабибуллин И. Создание распределенных приложений на Java 2. — СПб.: БХВ-Петербург, 2002. — 692 с.

12. Холзнер С. Visual C++6. Учебный курс. — СПб.: Питер, 1999. — 568 с.
13. Холл М., Браун Л. Программирование для Web. — М.: Вильямс, 2002. — 1264 с.
14. Холл М. Сервлеты и Java Server Pages. — СПб.: Питер, 2001. — 494 с.
15. Чен М., Грифис С., Изи Э. Программирование на Java. Наиболее полное руководство. — Минск: Попурри, 1997. — 1016 с.
16. Эдди С. XML-справочник. — СПб.: Питер, 2000. — 474 с.
17. Эккель Б. Философия Java. — СПб.: Питер, 2001. — 876 с.

Предметный указатель

A

ActiveX 277
API-вызов 398
ASP-страница 394, 395

C

char 25

D

dll-файл 452

H

HTML 93, 186
HTML-документ 109, 115, 120, 397
HTTP 21

I

int 25
Internet Explorer 21, 194
Internet Information Server 395
IP-адрес 231

J

Java 134, 154, 176, 180, 193
JavaScript 195
JDBC 175
JSP-страница 313, 316
JSP-тег 314

O

ODBC 175

S

SQL-запрос 181, 247, 250
String 25

W

Web-сервер 19, 300

X

XML 21, 184, 185

А

Администратор BDE 175
Апплет 19, 93, 95, 134, 140, 145, 148, 153, 155, 156, 197, 227
Аргументы командной строки 52

Б

База данных 156, 163, 175, 178, 247, 481
Бегущая строка 462
Библиотека классов 334, 413
Бин 277
Блок:
- catch 97
- try 89

Браузер 19

В

Визуальный интерфейс 63, 67, 118
Виртуальная машина 334
Вставка подстроки 387
Вывод текстовой строки 98
Выделение подстроки 90
Выход из цикла 340

Г

Гиперссылка 112, 113
Градиентная заливка 105
Графический контекст 432

Д

Делегат 365
Десериализация 289
Диалоговое окно 81
Длина слова 28
Документ cookie 131

З

Замена символа 91, 388
Запись в файл 78, 87
Запуск апплета 96

И

Идентификатор this 48
Инициализация членов класса 48
Интернет 308
Интерфейс 358, 458
- ItemListener 67
- Runnable 213
- Serializable 287
- ActionListener 67

К

Класс 40
- ADO.NET 419
- ArrayList 481, 484
- BinaryWriter 375
- BufferedReader 39
- Canvas 100
- Enumeration 315
- Exception 97
- FileInputStream 97
- FileOutputStream 88
- FileStream 372
- Font 57
- Frame 51
- Graphics 56, 98, 100, 464
- Hashtable 157
- HttpServletRequest 305

Image 102
Integer 202
IOException 97
JEditorPane 318
KeyEvent 69
Math 142, 227
MouseAdapter 77
String 53, 56, 202
StringBuilder 388
System 26
Thread 213, 214
абстрактный 67, 358
апплета 139
дочерний 45
конструктор 44
объект (экземпляр)
класса 48
родительский 45
свойства 44
таймера 461
Клиент 19, 224, 230, 247
Ключевое слово void 60
Коллекция объектов 483
Комментарий 29
Компоненты COM 444
Компоновщик 64
Конструктор 42, 48, 52, 139, 158, 236, 364, 367
класса 44, 48

Л

Лексема 470

М

Манифест-файл 287
Массив 26, 28, 201, 367
Меню 164, 167, 169
Метод:
GET 119
keyPressed() 69
keyReleased() 69
keyTyped() 69
main() 25, 52
paint() 98, 167
POST 119
виртуальный 358
класса 44
статический 363
Механизм потоков 433
Многофайловая сборка 389
Модификатор 365
доступа 59

Н

Наследование свойств 356

О

Область действия тега 110
Обработка событий 154, 170
Обработчик событий 65, 80, 165, 343, 424
Объект:
удаленный 13
Объектная переменная 97
Объявление массива 91
Оператор:
&& 35
|| 36
if 35
Split 388
множественного
выбора 337
Определение:
длины строки 90
класса 41
Опции выпадающего
списка 123

Охрана 88
Очистка прямоугольной области экрана 100
Ошибка ввода-вывода 89, 97

П

Пакет Swing 203
Переменная 25, 112
    класса 26
    объектная 43, 48
    объектная графическая 98
    потоковая 405
    простая 53
    цикла 33
Платформа .NET 333
Польская запись 469
Преобразование:
    типов 29
    числа в строку 90
Приведение типов 142, 292
Приложение консольное 335
Присваивание 28
Проверка на равенство двух строк 90
Программа для рисования графика 137
Программирование в Web 392
Проект 335, 392
Прослушиватель:
    ActionListener 165
    событий 50, 67, 76
    событий от клавиатуры 71
Простейшее Java-приложение 23
Пространство имен 334, 336, 413
Протокол:
    HTTP 458
    SMTP 309
    TCP 454

Р

Размер массива 92
Рисование:
    картинки 103
    окружности 107

С

Сборка 389
Свойства элементов 343
Секция:
    init() 135
    paint() 135
    start() 135
Сервер 19, 225, 240, 247
Сервлет 118, 228, 300, 304
Сериализация 288, 289, 376
Сетевое имя 461
Сетевой адрес 144, 154, 223, 231
Сигнатура 366
Скриплет 313
Скрипт 120, 121, 131
Событие 121, 154
События от клавиатуры 69
Соединение строк 90, 387
Создание:
    массива объектов 92
    проекта 335
Сокет 224, 231, 248
Список 405
Справочная система 344
Сравнение строк 202, 387
Статический контекст 59
Стек 470, 478
Структура 347, 401

Т

Таблица HTML 127
Таймер 129, 462
Тег 109
    <Form>, 118
    , 114
    <table>, 114
    <td>, 114
    <tr>, 115
    DIV 113
Тело цикла 29, 339
Терм 405, 469
Тип 41
    объектный 53
    простой 53
    события 229
Типы:
    данных SQL 181
    языка Java 53
Точка входа 49

У

Утилита ODBC Administrator 423

Ф

Файловая переменная 88
Файловый диалог 204, 211
Форма 118, 124, 379
Фрейм 116

Ц

Цикл:
    for 33
    while 28, 129, 142
    foreach 369

Ч

Чтение:
    из файла 78, 87, 89
    строки 405
    файла 206
    чисел 31

Э

Экземпляр класса 43
Элементы 129