№ 23 июнь 1999

еженедельное
ПРИЛОЖЕНИЕ К ГАЗЕТЕ
«ПЕРВОЕ СЕНТЯБРЯ»

ИНФОРМАТИК

Программирование на языке Visual Basic 5

А.И. Маргулев

Выпуск 1

Жаркое 99 лето

Содержание

ЯЗЫК БЕЗ ЛИШНИХ СЛОВ

*Кто это говорил, что деревья — это всего-навсего недорезанные бревна?**

Visual Basic 5 — это дружественный язык, позволяющий писать приложения "под Windows" (95(8) или NT) с интерфейсом профессионального уровня и без углубления в системные и инструментальные тонкости, как это требуется при использовании языка C++. Именно с появлением этой версии языка Visual Basic скептицизм большинства профессиональных программистов свелся, по существу, к "хорошей мине": существенно более высокая скорость создания приложений на Visual Basic 5 заставляет их все чаще обращаться именно к нему (прежде всего в задачах, связанных с базами данных).

Особый интерес Visual Basic 5 представляет для тех, кто делает в программировании первые шаги, самостоятельно или в курсе обучения, позволяя уже с самого начала почувствовать вкус к созданию приложений с "профессиональным" интерфейсом, будь то калькулятор или редактор, броузер графических или звуковых файлов, мультимедийная игра или обучающая программа и т.д. Освоившись на уровне простейших приложений, начинающий программист может приступить к экспериментам по их переносу в документы ActiveX, обеспечивая возможность их инсталляции и запуска из броузера Internet. Потом ему захочется создавать собственные элементы управления и размещать их на своей web-странице...

Но не только эти особенности Visual Basic 5 делают его привлекательным для изучения в школах, колледжах и вузах.

Visual Basic 5 позволяет свести воедино "старый", математико-алгоритмический ("программирование — вторая грамотность"), и "новый", информационно-технологический, подходы к изучению информатики, которые до этого момента сосуществовали в едином курсе практически независимо друг от друга. Отказ от математико-алгоритмического подхода привел бы к сокращению интеллектуально-логического аспекта обучения, в то время как отказ от изучения современных информационных технологий затруднил бы формирование основ общей информационной культуры. Visual Basic 5, во-первых, сам является таким же проводником объектно-ориентированной технологии Microsoft и идеологии

* Все эпиграфы в книге являются цитатами из произведений Венедикта Ерофеева.

совместно используемых ресурсов, как и MS Office 97 или Internet Explorer (вплоть до использования их компонентов), а с другой стороны — предоставляет структурированный (а также простой и удобный) язык записи и отладки алгоритмов, используемых как при создании приложений, так и для программирования в тех же офисных продуктах. То есть Visual Basic 5 дает учащемуся возможность взглянуть на информационные технологии "изнутри", в более интеллектуальном и творческом ключе.

Именно с этой целью в проектах настоящего пособия рассматриваются такие вопросы, как COM ("компонентно-объектная модель"), OLE-автоматизация, средства повторного использования кода, технология "клиент — сервер".

Структура книги

В первой главе излагаются принципы объектного и событийно-управляемого программирования в среде Visual Basic 5, описываются элементы сервиса, предлагаемого этой средой.

Во второй главе на примере создания простого приложения рассматривается практическая реализация этих принципов.

В третьей главе на примере создания обучающей программы рассмотрена работа средствами Visual Basic 5 со списками и базами данных, включая запросы на SQL.

Четвертая глава посвящена концепции COM, теории и практике повторного использования кода и OLE-автоматизации. Навыки обращения с объектами отрабатываются на примере создаваемого объекта "Стек". Здесь же создаются практически полезный пользовательский элемент управления "Часы-Будильник" и использующее его приложение.

В пятой затронуты вопросы программирования для Internet.

Автор благодарит ведущего методиста Московского института повышения квалификации работников образования Владимира Владимировича Морозова за проявленную им постоянную готовность поделиться опытом.

Глава 1

ОБЪЕКТНЫЙ ПОДХОД

Бонапарт рекомендовал как можно чаще оперировать понятиями, ничего не выражающими и все объясняющими, например, "судьба".

Базовые элементы интерфейса Windows 95

В начале главы напомним некоторые базовые сведения, относящиеся к работе пользователя с любым приложением в операционной системе Windows 95(8). Иллюстраций приводить не будем, так как уже после двух десятков сеансов работы в этой графической системе можно без труда представлять, о каких именно объектах идет речь.

Интерфейс — это набор средств взаимодействия программы с пользователем. Windows устанавливает *единообразный* графический интерфейс, в котором пользователь оперирует при помощи манипулятора "мышь". Основные элементы интерфейса — *меню* и *окна* (приложений, документов и диалоговые). Предполагается, что пользователь имеет начальные навыки по работе с мышью: умеет указывать указателем мыши объекты экранного изображения и производить последующие операции *"щелчок"*, или *"клик"* (однократное нажатие и отпускание клавиши), *"двойной клик"* (два щелчка с минимальным интервалом времени между ними), а также *"дрэг энд дроп"*, или *перетаскивание* ("хватание" объекта нажатием левой клавиши, перемещение схваченного объекта без отпускания клавиши и ее отпускание при достижении требуемого места).

Меню — это список объектов (как правило, тематически сгруппированных), из которых необходимо сделать выбор (помещая на объект указатель мыши и произведя однократный клик). Выбор пункта меню — это выполнение определенной команды программы. Все меню обладают общими свойствами:

- могут иметь несколько уровней — когда при выборе указателем мыши пункта меню (имеющего после названия указатель "➢") открывается подменю этого пункта (меню следующего уровня, дочернее меню);
- могут иметь пункты, недоступные для выбора в данный момент (в данный момент выглядят "блеклыми");

- допускают возможность одновременного выбора нескольких объектов;
- могут иметь пункты (их названия заканчиваются многоточием — "..."), при выборе которых открываются *диалоговые окна* (см. ниже).

Окно приложения (программы) — это элемент интерфейса, в котором выполняется любая запущенная на выполнение в операционной системе Windows прикладная программа (приложение). *Открыть* или *закрыть* окно приложения — то же, что и запустить программу на выполнение или завершить ее. Кнопки управления окном в его правом верхнем углу обеспечивают закрытие окна, а также режимы *"полное окно"*, *"восстановить"* (к некоему *"исходному"* размеру, который можно менять) и *"свернуть"* (кнопку со значком данного приложения на панели задач, рядом с кнопкой **Пуск** окна Windows 95). В незакрытом окне приложения (хотя бы и свернутом) всегда выполняется программа. Переход от одного выполняющегося приложения к другому (хотя бы и свернутому) осуществляется циклически по комбинации клавиш Alt Tab; при этом свернутое окно восстанавливается. Заметим, что указанные кнопки управления окном дублируют соответствующие команды *системного меню* окна, открываемого щелчком мыши по *иконке* (синоним *пиктограммы*) окна, расположенной в его верхнем *левом* углу. Когда выполняется одновременно несколько приложений, только у одного из них окно (или пиктограмма!) *активно*, то есть воспринимает команды; оно располагается поверх остальных, и его заголовок выделен цветом.

"Хватая" боковую или нижнюю рамку окна указателем мыши (он принимает при этом вид двунаправленной стрелки), мы можем менять его размер по горизонтали или вертикали (или оба размера одновременно, "ухватив" за правый нижний угол). Окно приложения всегда содержит зону *заголовка* (содержащую название приложения) и горизонтальное меню, являющееся *главным меню* приложения. "Схватив" окно за заголовок, мы можем передвигать его по экрану. При выборе почти каждого пункта главного меню появляется *ниспадающее,* или *падающее, меню первого уровня.*

Окно документа — это элемент интерфейса, содержащий объект обработки приложения (данные). В полноэкранном варианте является частью окна соответствующего приложения. Всегда содержит зону заголовка (содержащую имя документа), часто — *полосы прокрутки* (появляющиеся, когда документ не помещается полностью в окне) и *линейки.* Открытое окно документа может находиться в активном либо пассивном

состоянии. Имеет те же кнопки управления, что и окно приложения, причем при свертывании сохраняет минимизированную зону заголовка и кнопки управления. Если окно находится в пассивном состоянии (зона заголовка не выделена цветом), то, щелкнув по любой его части мышью, можно перевести его в активное состояние. Тот же переход обычно позволяет сделать команда, представляющая имя соответствующего окна из пункта главного меню **Окно** (в приложениях с *многодокументным интерфейсом*). В приложениях других типов (с однодокументным интерфейсом) нужное из открытых окон документов может быть активизировано в результате поиска циклическим переключением окон комбинацией клавиш Alt Tab.

Диалоговое окно (или просто ***диалог***) выдается приложением и служит для ввода или установки параметров, представляющих данные для управления приложением. Оно содержит зону заголовка и различные *элементы управления*, поддерживающие диалог с пользователем. К ним в основном относятся:

- *командные кнопки* (надпись на которых поясняет их функции); например, кнопки с надписями **OK** позволяют закрыть диалог с сохранением настроек;
- *переключатели,* или "*флажки*" (*Check Box*), представляющие собой квадратные окошечки, которые независимо друг от друга могут находиться в двух состояниях: пустом (режим, к которому относится флажок, выключен) и с "птичкой" (режим включен); переход из одного состояния в другое осуществляется щелчком мыши в окошечке;
- *радиокнопки,* или *поля выбора* (*Options*), объединенные в группу альтернативных режимов; обозначаются кружочками, из которых один отмечен центральной точкой (выбранный режим), а остальные пусты; выбор нужного режима осуществляется щелчком мыши в соответствующем кружочке;
- *списки* (*List*), содержащие перечни каких-либо объектов и размещаемые в подокнах (имеющих полосу вертикальной прокрутки списка в окне, если оно не умещает весь список); для выбора объекта (пункта перечня) используется однократный щелчок мыши; иногда предусмотрен и двукратный щелчок ("двойной клик") по объекту для вызова нового диалогового окна, в котором свойства данного объекта могут быть *отредактированы* (изменены).

Частным случаем списка является *раскрывающийся список*, представляющий собой заголовочное окошко, содержащее выбранный в данный момент объект списка, и расположенную справа от окошка кнопку; первый щелчок по кнопке раскрывает список, повторный — закрывает; после выбора объекта щелчком мыши список закрывается и выбранный объект остается в его заголовке; другим частным случаем списка является нераскрывающийся список значений с элементом управления "спин" (*Spin*), или "вороток", состоящим из пары кнопок с указателями направления прокрутки "вверх" и "вниз"; щелкая по кнопкам мышью, мы меняем текущие значения в окошке с заданными шагами;

Еще один вариант списка — *иерархический,* или *дерево* (*Tree*), примерами являются списки с элементами файловой системы в Windows 95;

- *поля ввода* (*Text Box*), предназначенные для ввода символьных строк (чисел, текста); для начала редактирования указываем мышью позицию ввода и щелчком левой клавиши устанавливаем в нее курсор.

  С полем ввода может быть сопряжен список (какого-либо из приведенных вариантов), служащий для выбора требуемых значений (*комбинированное поле* ввода).

Диалоговое окно может быть разбито на тематические подокна, снабженные *вкладками* **(Tab)**; такие подокна располагаются каскадом (одно под другим), а их вкладки — мозаикой (рядом друг с другом), причем наверху находится подокно с выбранной мышью вкладкой. Такие трехмерные диалоговые окна напоминают картотеку.

Существуют и другие элементы управления, причем их количество не определено заранее. Подобные элементы управления мы и сами будем создавать средствами Visual Basic 5.

Элементы управления как объекты программ

Три составляющие Windows-приложения

Мы рассмотрели основные элементы интерфейса программ, работающих под управлением Windows, — "приложений Windows" (будем называть их просто "приложениями"). Их совокупность образует *визуальную составляющую* любого такого приложения. Система програм-

мирования VB5, как и другие Visual-системы, позволяет формировать эту визуальную составляющую без написания кода, а просто средствами элементарного графического редактирования (*компоновки*). Не случайно такое программирование называют *визуальным*.

Но сформированная визуальная составляющая должна еще определенным образом реагировать на те или иные *события*, происходящие вовне. К таковым можно отнести сигнал от мыши или клавиатуры, выполнение определенной команды центральным процессором, получение сообщения от другого приложения или другого окна данного приложения и т.д. Поэтому наряду с визуальной составляющей приложения имеется еще и другая его составляющая — *системная*.

Дело в том, что элементы интерфейса — это просто видимые части тех "устройств", из которых, как из "кирпичиков", слагается любое приложение. Эти устройства принято называть *объектами*.

Для изготовления таких объектов-"кирпичиков" в интегрированной среде VB5 используются "формочки" (шаблоны), каждая из которых определяет у изготавливаемых ею объектов "форму" (набор функциональных характеристик). В терминологии объектного программирования эти "формочки"-шаблоны называются *классами*. Забегая вперед, отметим, что если считать объекты новой (наряду со строковой, числовой, логической и т.д.) категорией данных, то класс, по существу, задает своим именем *тип* такого данного. Используя имя класса, можно в дальнейшем объявлять переменную, "значениями" которой будут объекты данного класса, — так же, как имена целого типа **Integer** или строкового типа **String** используют для объявления обычных переменных, представляющих значения соответствующих категорий. (Разница лишь в том, что объектная переменная представляет не сам объект, а его адрес в оперативной памяти.)

Объект, таким образом, суть *экземпляр объекта заданного класса*. Но это слишком длинно. Сразу нужно привыкнуть к тому, что в одном контексте под объектом подразумевают именно *экземпляр*, а в другом — соответствующий *класс*. Держите в уме это обстоятельство!

Приведем пример. Типичными объектами в VB5 являются элементы управления — такие, например, о которых мы упоминали, описывая интерфейс Windows 9x: поля ввода, списки, командные кнопки и т.д. В интегрированной среде VB5 эти объекты представлены своими *классами*, на которые ссылаются отображения этих элементов в специальном окне — *палитре инструментов*. В процессе визуального проектирования приложения мы будем в этой палитре выбирать мышью отображение того или иного элемента (выбирая тем самым ссылаемый этим отображением *класс* объекта) и размещать этот элемент в форме приложения уже как *экземпляр* объекта. Таких экземпляров мы можем разместить сколько угодно; все они будут различаться своими именами.

В процессе функционирования приложения составляющие его объекты могут обмениваться *сообщениями*. Необходимость такого обмена возникает всякий раз, когда происходит то или иное значимое для объектов приложения *событие*. Эти события возникают как вне приложения, так и в каком-либо его объекте. В последнем случае уместнее говорить не о возникновении события, а о его *возбуждении* объектом. Сообщение о произошедшем событии приходит в тот или иной объект от операционной системы. Именно *системная составляющая* приложения и определяет свойства каждого произошедшего события, формируя далее сообщение тому объекту, в котором предусмотрена реакция на это событие. Так, например, при нажатии на кнопку мыши именно системной составляющей приложения предстоит определить тот объект, которым данное нажатие должно управлять, и сформировать для него соответствующее сообщение. И то и другое будет зависеть скорее всего от взаимоположения указателя мыши и визуальных составляющих объектов (*активного* приложения) и их частей. Таким образом, произойдет, скажем, полное раскрытие окна приложения. Или установка флажка в диалоге. Или выбор элемента списка. Или его прокрутка. Или что-нибудь еще, с чем хорошо знакомы все работающие в операционной системе Windows 9x.

Системная составляющая приложения скрыта от глаз прикладного программиста. Ее существование проявляется для него только в том, что активное приложение, чем бы оно ни занималось, находится в любой момент времени еще и в состоянии ожидания какого-либо внешнего события. Для знакомых с программированием это выглядит так, как если бы приложение представляло собой обычную программу "под DOS", имеющую охватывающий цикл, в котором отслеживаются и обрабатываются внешние воздействия, такие, например, как нажатия клавиш пользователем. Подобный цикл (обработки сообщений) существует и в самом деле, но не в той части приложения, которую создает пользователь, а в системной составляющей. И обрабатывает он не непосредственно сами события, а сообщения, сформированные на основе событий операционной системой и помещенные ею в очередь приложения либо в единую системную очередь.

Подведем некоторые итоги.

С помощью объектов — элементов управления программист средствами визуальной компоновки создает визуальную составляющую приложения, и потому программирование на VB5 является *визуальным*. Кроме того, с помощью системной составляющей приложение через объекты управляется пользователем или другими приложениями посредством

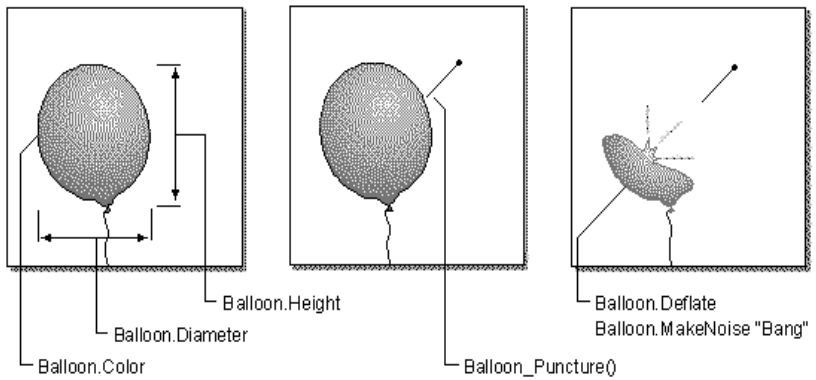
возбуждения и обработки событий, и потому программирование на VB5 является программированием *с управлением по событиям*. А это означает, что почти вся работа прикладного программиста на VB5 состоит просто в написании подпрограмм — обработчиков тех или иных событий. *Обработчики событий* являются, таким образом, третьей составляющей создаваемого на VB5 Windows-приложения. Их в основном мы и будем учиться писать.

Объект и его "паспорт"

Мы уже сказали, что любое написанное на VB5 приложение — это просто набор устройств — объектов (*Objects*). Так же как, скажем, компьютер — набор съемных модулей: плат, приводов и т.д. Для каждого такого объекта, как и для модулей компьютера, можно составить описание-паспорт, в котором описать, во-первых, основные свойства объекта (*Properties*), во-вторых — события (*Events*), на которые объект может реагировать, в-третьих — способы реагирования (*Perform Methods*), представляющие собой отдельные действия, которые объект способен производить в качестве реакций на события (*Respond Events*). Сразу отметим, что традиционно способы реагирования, являющиеся внутренними процедурами VB5, принято называть *методами*.

Рассмотрим все три указанные понятия на примере воздушного шарика из справочной подсистемы VB5 "Books Online" (если она была установлена при инсталляции системы VB5).

Открыв данную подсистему из пункта **Help** главного меню запущенной системы VB5 либо из подменю группы "Microsoft Visual Basic 5.0" (**Пуск ➢ Программы ➢ Microsoft Visual Basic 5.0 ➢ Books Online**), мы получаем: слева — окно оглавления, справа — окно просмотра содержимого раздела. Откроем в оглавлении книгу "Forms, Controls and Menus" ("Формы, элементы управления и меню") и в ней — раздел "Understanding Properties, Methods and Events" ("Понятие о свойствах, методах и событиях"). В разделе в качестве объекта рассматривается детский воздушный шарик **(Balloon)**, наполненный гелием. Среди его свойств указываются высота **(Height)**, диаметр поперечника **(Diameter)**, цвет **(Color)**, "надутость" **(Inflated)**, возраст **(Age)**. Событиями, вызывающими реакцию данного объекта, являются его протыкание **(Puncture)** и отпускание **(Release)**. В качестве способов реагирования (методов) определяются характерный хлопок **(MakeNoise "Bang")**, выброс газа **(Deflate)** и подъем **(Rise)**:



Описанные характеристики объекта позволяют программировать решение связанных с ним возможных задач. Вот как, например, будет выглядеть фрагмент установки **(Set)** указанных свойств объекта (кроме возраста):

```
Balloon.Color = Red
Balloon.Diameter = 10
Balloon.Inflated = True
```

Как мы видим, обращение к свойствам осуществляется по составным именам, в которых имени свойства предшествует имя объекта, отделяемое *навигационной точкой* (такая конструкция подобна обращению к элементу структуры в языке Си, к элементам записей в языке Pascal). А если бы наш объект **Balloon** входил в состав другого объекта — *формы* проекта, например, с именем **Form1**:

> **Форма проекта служит заготовкой для окна проектируемого приложения; форма настраивается при проектировании интерфейса вашего приложения**

— то из другой формы (**Form2,** например) обращение, допустим, к свойству **Color** нашего объекта производилось бы по составному имени **Form1.Balloon.Color**.

В приведенных предложениях (*Statements*) гипотетической программы на Visual Basic справа от знака равенства указаны определенные значения из числа допустимых для соответствующих свойств. Значения имеют такие характеристики, как *тип* и *диапазон*. Значения **Red** и **10** относятся, по-видимому, к целому типу; разница между ними та, что первое представлено именованной, а второе — обычной константами. Значение **True** ("истина") является *логическим*; для свойства **Inflated** оно означа-

ет, что шарик действительно надут; противоположное значение **False** ("ложь") означало бы, что шарик сдут.

Вызов методов нашего объекта может быть таким:

```
Balloon.Inflate
Balloon.Deflate
Balloon.Rise 5
```

Синтаксис вызова подобен обращению к свойству; разница лишь в том, что именем метода является глагол.

А вот как может выглядеть подпрограмма-обработчик, вызываемая при возникновении определенного события — протыкания шарика, например:

```
Sub Balloon_Puncture()
   Balloon.Deflate
   Balloon.MakeNoise "Bang"
   Balloon.Inflated = False
   Balloon.Diameter = 1
End Sub
```

В первом предложении подпрограммы вызывается метод **Deflate**, во втором — метод **MakeNoise** с аргументом **"Bang"**, в третьем предложении свойство **Inflated** устанавливается в значение **False**, в четвертом — свойство **Diameter** устанавливается в **1**.

Имя подпрограммы-обработчика **Balloon_Puncture** является комбинацией соединенных через знак подчеркивания имен объекта **(Balloon)** и события **(Puncture)**. Можно представить, что это экзотическое событие возникает в динамической компьютерной игре при соприкосновении экранного контура шарика с каким-либо "острым" контуром (т.е. с его сходящимся к точке элементом). Если же вместо этого для нашего шарика была бы предусмотрена возможность "хлопать" его кликом мыши, то имя приведенной подпрограммы должно было быть **Balloon_Click**. Если бы были предусмотрены оба эти события, то должны были бы иметься две подпрограммы с разными именами и одинаковыми телами.

Ну а можно ли подпрограмму-обработчик вызывать из программного кода как обычную подпрограмму? Безусловно, причем синтаксис такого вызова может быть полностью идентичен синтаксису вызова метода (т.е. даже без ключевого слова **Call** и без круглых скобок). Действительно, и обработчик, и метод представляют собой некоторые процедуры; разница лишь в том, что код метода мы не можем ни прочесть, ни поменять.

В приведенном примере мы устанавливали свойства объекта в программном коде, т.е. в фазе выполнения программы (*runtime mode*). Большинство свойств можно также устанавливать и в фазе разработки (*design*

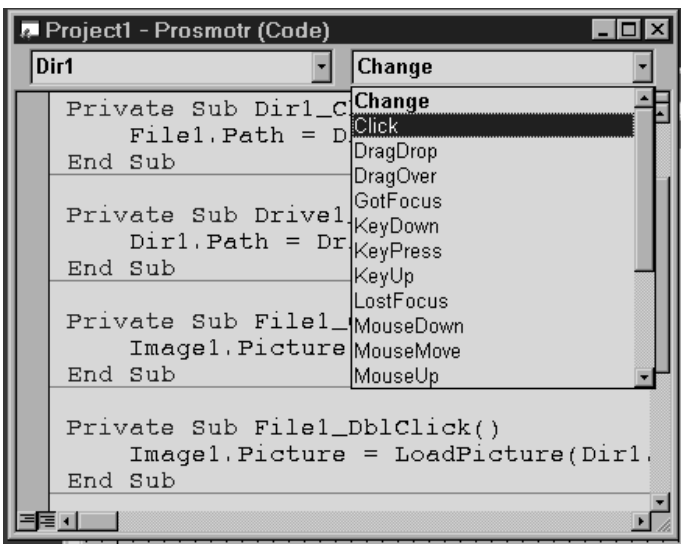
mode) попутно с компоновкой объектов в экранной форме. Некоторые из свойств, однако, изменять в режиме выполнения нельзя (а можно только читать); другие, наоборот, доступны для установки только в режиме выполнения.

Итак, паспортные данные объекта — свойства, события и методы — участвуют своими именами в создаваемом программистом коде. Неприятная неожиданность, если учесть, что эти имена сплошь и рядом включают в себя каждое по нескольку слов! Поэтому спешим успокоить: никакой необходимости набирать эти имена или держать в голове нет. В окне редактора программного кода под именем окна располагаются два раскрывающихся списка: левый содержит имена всех объектов проекта **(Project)**:

> **Проект — это набор файлов, с которыми мы работаем в процессе создания приложения. Кроме того, под проектом понимают само приложение в процессе его создания**

— а правый — имена всех событий, определенных для выбранного в левом списке объекта.

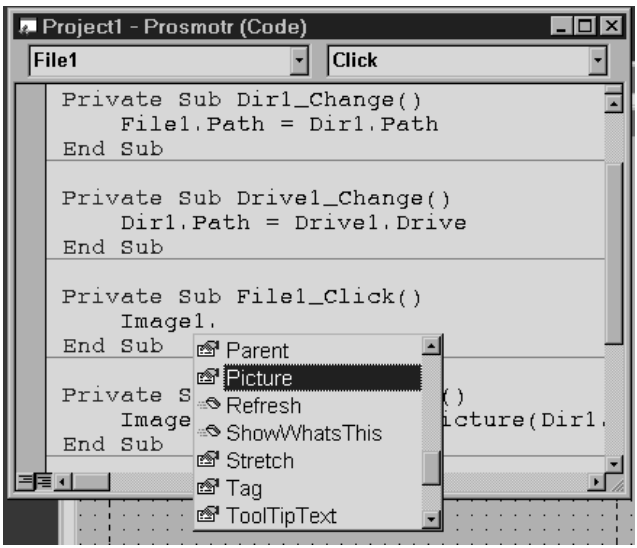
В следующем примере для объекта с именем **Dir1**, являющегося списком директорий (папок), в правом раскрывающемся списке событий выбрано событие **Change**, наступающее при смене выделенной директории списка:



Заметим, что всю информацию и о данном объекте, и о его событиях, а также о свойствах и методах мы могли предварительно узнать, выделив данный объект на форме приложения и войдя в режим помощи нажатием клавиши F1.

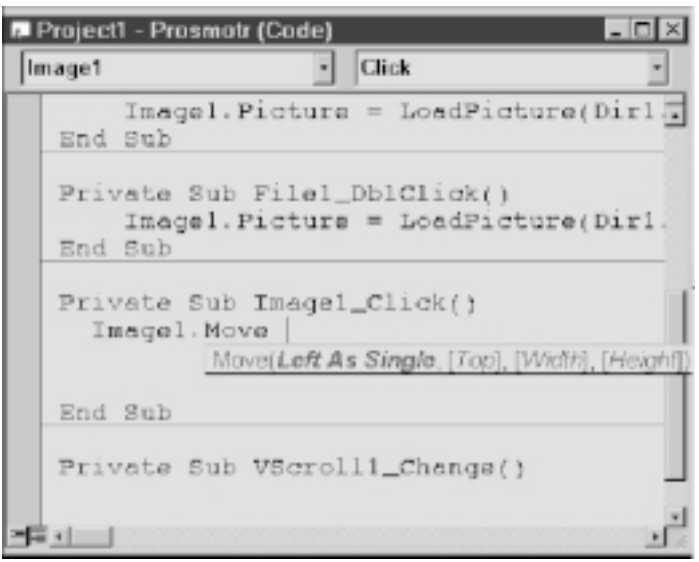
Выбрав из левого списка имя объекта, а из правого — имя события, мы получаем немедленно в поле окна заготовку соответствующей подпрограммы-обработчика, содержащую заголовок подпрограммы с правильно составленным ее именем.

Пусть теперь при наборе, например, текста подпрограммы **File1_Click** мы хотим обратиться к свойству или методу некоторого определенного в данном проекте объекта **Image1** (по его имени можно догадаться, что он представляет картинку в проектируемом приложении). Набрав **"Image1."**, мы немедленно получаем окно списка, в котором перечислены все имена свойств (значок) и методов ( ):



Этот сервис называется **Auto List Member** (автоматический список участников). Двойной клик на выбранном имени (или нажатие на клавишу Tab после выделения) приводит к копированию его вслед за точкой в набираемое составное имя. Разумеется, можно осуществить и обычный набор нужного имени, игнорируя указанный список.

Auto List Member — это лишь один из примеров дружественного сервиса, облегчающего жизнь программиста в системе VB5. Существует еще ряд подобных услуг системы с приставкой **Auto**: **Complete Word** (автодописывание служебного слова, когда набрано достаточное для его идентификации количество начинающих его символов); **Data Tips** (отображение значения переменной во всплывающем окошечке при помещении над ней указателя мыши); **List Constant** (открываемый список констант для свойств объекта). И, наконец, **Auto Quick Info**, представляющий собой вывод синтаксического шаблона обращения к методу, имя которого набрано (в ответ на пробел после этого имени), либо к встроенной функции VB5 (в ответ на открытие круглой скобки для записи аргументов вызова). Вот пример такой синтаксической подсказки для метода **Move**:



Кстати, о подсказке. По клавише F1 мы можем не только входить в систематический каталог справочной системы, не только получать информацию о выделенном на форме объекте, но и контекстную подсказку по выделенному курсором в программном коде встроенному программному объекту (имени стандартной функции VB5, методу, свойству и т.д.). Чем чаще мы будем обращаться к таким подсказкам, тем лучше изучим VB5 и... английский язык.

Глава 2

ОСНОВЫ ПРОЕКТИРОВАНИЯ

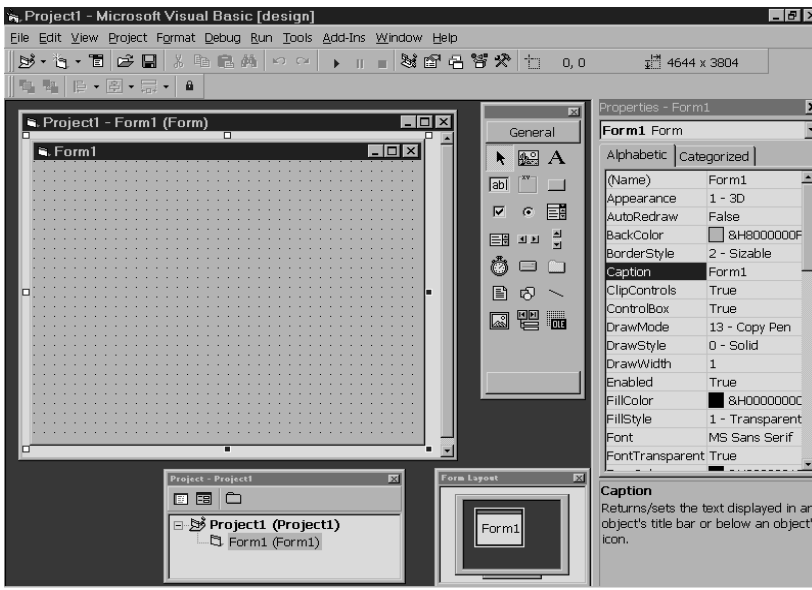
Следует вполне полагаться на судьбу и твердо веровать, что самое скверное еще впереди.

Интегрированная среда разработки

Запустив Visual Basic 5.0, мы получим окно **New Project**:

Перед нами три вкладки — **New**, **Existing** и **Recent**. Верхняя вкладка **New** содержит пиктограммы различных типов проектов и сервисных модулей. Например, **Vb Application Wizard** ("Мастер приложений") позволяет автоматизировать создание заготовки нового приложения. Вкладки **Existing** и **Recent** служат для открытия расположенных на накопителях *существующих* или только нескольких из *последних* модифицированных проектов соответственно. Возможности вкладок **Existing** и **Recent**

сохраняются и после выбора пиктограммы **Standart EXE** на вкладке **New** (а именно этот вариант проекта ("стандартный") мы и будем использовать при создании наших приложений). Чтобы переходить сразу в режим создания стандартного проекта по умолчанию, после выбора пиктограммы **Standart EXE** и перед нажатием на кнопку **Открыть** (иногда ее имя **Open**) нужно установить флажок **Don' t show this dialog in the future** (Не показывать этот диалог в дальнейшем). При этом, если в дальнейшем нам все же понадобится работать с окном **New Project**, то мы включим радиокнопку **Prompt** (Запрос), путь к которой из главного меню: **Tools ➢ Options ➢ Environment ➢ Prompt**. Ну а сейчас после открытия стандартного проекта мы увидим его главное окно следующего вида:



Впрочем, расположение элементов этого окна (будем называть его *главным*) будет у вас наверняка несколько другим, и в качестве первого упражнения попытайтесь привести его к данному виду.

Рассмотрим три верхние строки главного окна.

Первая из них — *строка заголовка*. Мы уже знаем, что в ее левом конце располагается значок системного меню, позволяющего изменять размер окна и его местоположение, а также производить над ним те же действия, что и при помощи трех известных нам кнопок изменения статуса окна (свертывания, максимизации/восстановления, закрытия) в правом конце строки. В центральной части строки располагается имя окна. Оно состоит из имени проекта (**Project1**), после которого через тире указана программная среда **(Microsoft Visual Basic)**. Далее выражением [**design**] указан текущий режим приложения — проектирование. В режиме выполнения проекта текст в квадратных скобках заменяется на [**run**]. Забегая вперед, отметим, что при максимизации активного окна документа с формой (заготовкой окна будущего приложения) область заголовка этого окна документа совмещается с областью заголовка главного окна и имя окна документа (в квадратных скобках) приписывается справа к имени главного окна через тире.

Под строкой заголовка располагается *строка главного меню*. При максимизации активного окна документа с какой-либо формой в правый конец этой же строки переходят три кнопки изменения состояния указанного окна документа.

Наконец, под строкой главного меню располагается *стандартная панель инструментов*. В ее левой части имеется двойной вертикальный рельеф, схватив за который мы можем "вырвать" эту панель из ее стандартного местоположения и разместить в любом другом месте экрана. Отсоединенная, эта панель получает собственную строку заголовка с именем **Standard** и кнопкой закрытия. Двойным кликом по этой строке можно вернуть панель на ее обычное место.

Стандартная панель инструментов содержит пять групп пиктограмм команд, дублирующих наиболее "ходовые" команды, доступные через те или иные пункты главного меню. В правой же части этой панели располагается группа из двух координатных пар: *положения* (относительно левой и верхней внутренних границ главного окна в режиме выполнения) и *размеров* текущего (выделенного) объекта. (Эта группа присутствует только при *максимизированном* статусе главного окна.) Когда проект только что открылся, таким объектом является расположенная в окне документа *экранная форма* будущего приложения с именем **Form1**.

Координатные пары (у нас они **0**, **0** и **4644**, **3804**) отображают значения следующих свойств выделенного объекта **(Form1)**: **Left** (Левый край), **Top** (Верхний край), **Width** (Ширина), **Height** (Высота) соответственно, выраженные в *твипсах* (*twips*) — условных единицах, равных 1/1440 дюйма на принтерной распечатке. Все эти параметры устанавливаются

при проектировании в процессе компоновки формы будущего приложения и могут программно изменяться в процессе его работы. Положение на экране окна проектируемого приложения во время его выполнения устанавливается в специальном окне **Form Layout** (Внешний вид формы), о котором пойдет речь ниже.

Перейдем к рассмотрению *инструментальных окон*, которые расположены на рабочей поверхности главного окна. Эти окна предоставляют нам те или иные инструментальные средства, используемые при проектировании приложения.

Важнейшие инструментальные окна

Перечислим их.

- Окно, на котором располагается множество пиктограмм различных элементов управления (*рис.* 2.1), называется **Toolbox** (Палитра инструментов). Выбрав щелчком мыши по соответствующей пиктограмме нужный элемент управления, мы устанавливаем его на форме проектируемого приложения. Установка заключается в том, что мы помещаем указатель мыши в предполагаемой точке расположения на форме левого верхнего угла элемента управления и, зафиксировав левую кнопку мыши в нажатом положении, тянем образующийся прямоугольный контур до предполагаемой точки расположения правого нижнего угла элемента, где и отпускаем. Другой способ состоит в выполнении двойного клика по пиктограмме элемента в палитре инструментов, после чего соответствующий элемент появляется в произвольном месте на активной форме, имея некоторые "стандартные" размеры. После этого его положение на форме и размеры регулируются при помощи мыши обычным образом.

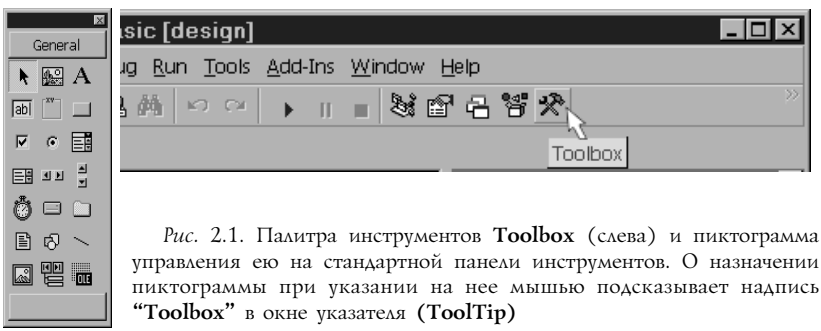
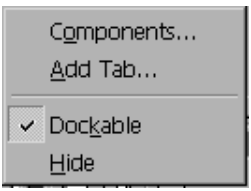


Рис. 2.1. Палитра инструментов **Toolbox** (слева) и пиктограмма управления ею на стандартной панели инструментов. О назначении пиктограммы при указании на нее мышью подсказывает надпись **"Toolbox"** в окне указателя **(ToolTip)**

Сейчас самое время заметить, что в интегрированной среде VB5 активно используются *контекстные меню*, вызываемые кликом по объекту *правой* клавишей мыши (либо комбинацией Shift F10). Сделав такой клик по палитре инструментов, мы получаем следующее меню:



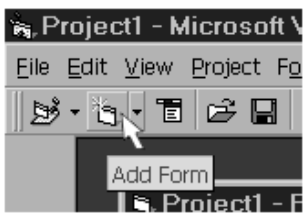
Горизонтальной чертой меню разделено на две группы. Верхняя имеет две команды. **Components** (Компоненты) отвечает за добавление в палитру инструментов новых элементов управления; она является дубликатом одноименной команды из пункта **Project** (Проект) главного меню и может выполняться по комбинации *клавиш быстрого вызова* (*Shortcuts*) Ctrl T. По этой команде открывается окно компонентов, с которым нам еще предстоит работать при проектировании учебных приложений. **Add Tab** (Добавить вкладку) используется при нехватке места в палитре (и нежелании увеличивать ее размер) и заключается в создании именованных вкладок, на которых можно размещать дополнительные компоненты; первичная вкладка имеет, как мы видим, имя **General** (Главная). Нижняя группа предоставляет возможность убирать палитру инструментов с экрана, щелкая по состоянию **Hide** (Скрытая), причем того же достигают, щелкая по кнопке закрытия инструментального окна. Когда палитра на экране, галочкой отмечено ее состояние **Dockable** (Выставленная). Пункты **Hide** и **Dockable** имеются в контекстных меню и для ряда других типов окон.

- Окно документа, в котором располагается экранная форма проектируемого приложения, является *окном формы*. Оно имеет строку заголовка, подобную по структуре строке заголовка главного окна. Имя окна состоит из имени проекта (у нас — **Project1**), к которому через тире приписано имя размещенной в нем формы (у нас — **Form1**); за этим именем в круглых скобках указан класс формы **(Form)**.

Каждой экранной форме соответствует входящий в проект одноименный файл с расширением FRM. По умолчанию при открытии нового проекта открывается одно окно экранной формы **Form1**; при надобности можно открывать дополнительные окна форм, имена которых по умолчанию **Form2**, **Form3** и т.д. Все эти окна форм будут равноправны и независимы друг от друга. Интерфейс создаваемого приложения с такого типа независимыми окнами называется *Single Document Interface* — SDI ("однодокументный интерфейс"). Бывают приложения и с другим типом интерфейса — *Multiple Document Interface* — MDI ("многодокументный интерфейс"), в которых существует единственное главное ("родительское") окно, содержащее дочерние окна документов (как, например, в текстовом процессоре Microsoft Word или в самой интегрированной среде VB5).

Созданием приложений с интерфейсом типа MDI мы заниматься не будем; научившись создавать однодокументные приложения, мы, воспользовавшись услугами мастера приложений (VB Application Wizard), запускаемого из окна **New Project** (см. начало главы), без труда получим необходимую заготовку.

А пока откроем еще одно окно формы в открытом нами SDI-проекте. Это можно сделать несколькими способами; мы воспользуемся самым очевидным — с помощью известной нам стандартной панели инструментов, как показано на *рис.* 2.2. Выполнив **Открыть** в появившемся диалоге с выделенной по умолчанию пиктограммой **Form**, мы получаем в главном окне проекта новое окно формы **Form2**:



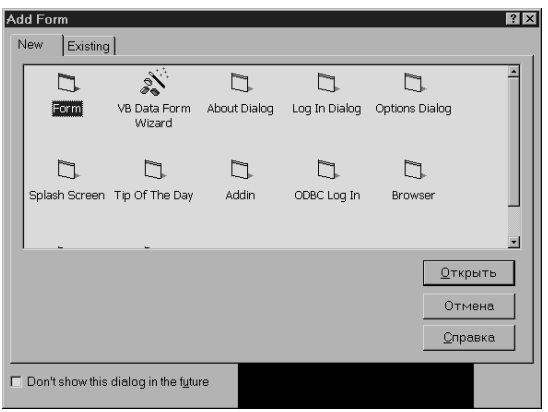
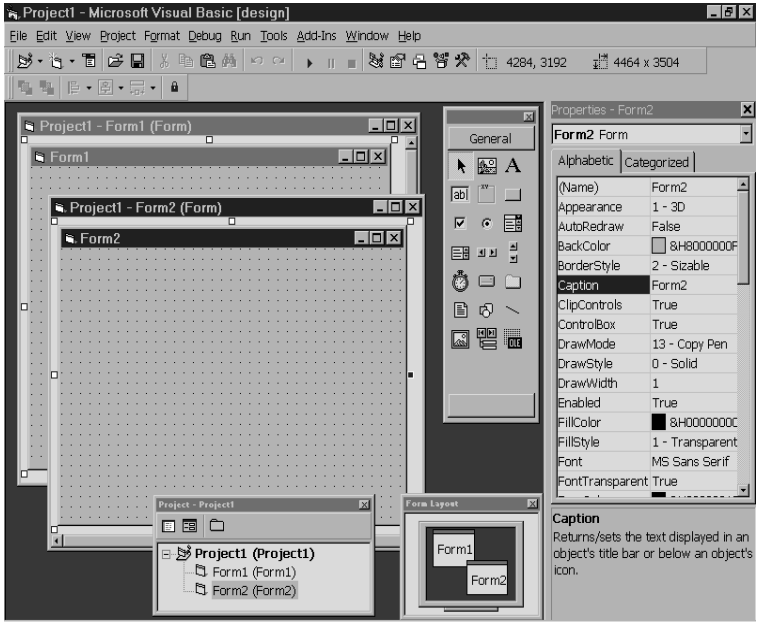


Рис. 2.2. Выбор пиктограммы **Add Form** (Добавить форму) на стандартной панели инструментов. Справа — диалог, открываемый при выполнении команды **Add Form**



Заметим, что вновь открытое окно формы является активным, а старое потеряло активность ("выцвело"). Обратим также внимание, что изменения произошли и в трех еще не рассмотренных нами подокнах проекта.

Окно экранной формы всегда имеет "оборотную сторону", представляющую собой *окно кода* (точнее — окно *редактора* кода). Дело в том, что в файле формы всегда присутствует и относящийся к обработчикам связанных с нею событий код. К нему мы и можем перейти, вызывая окно кода двойным кликом по форме либо через значок **View Code** окна **Проводника проекта**.

- В окне **Проводника проекта (Project Explorer)**, имеющего имя **Project**, к которому через дефис прибавляется имя проекта (см. *рис.* 2.3), перечисляются все файлы, формирующие наш проект. Собственно файл проекта, содержащий информацию о всех включенных в проект файлах, имеет расширение VBP. Файлы экранных форм имеют расширение FRM, модулей (отдельно хранимых текстов подпрограмм, не отвечающих за визуальную составляющую приложения) — BAS. В нашем примере окно проводника проекта отображает файл проекта **Project1.vbp** и два файла форм —

Form1.frm и **Form2.frm**. В проект для каждой формы может входить по одному файлу двоичных данных с расширением FRX, хранящему свойства, представляющие картинки **(Picture)** и значки **(Icon)**. Кроме того, проект может содержать файлы модулей классов (CLS), управляющих элементов стандарта ActiveX (OCX) и один файл ресурсов (RES).

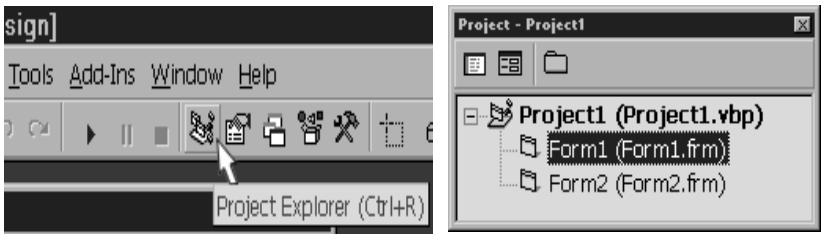


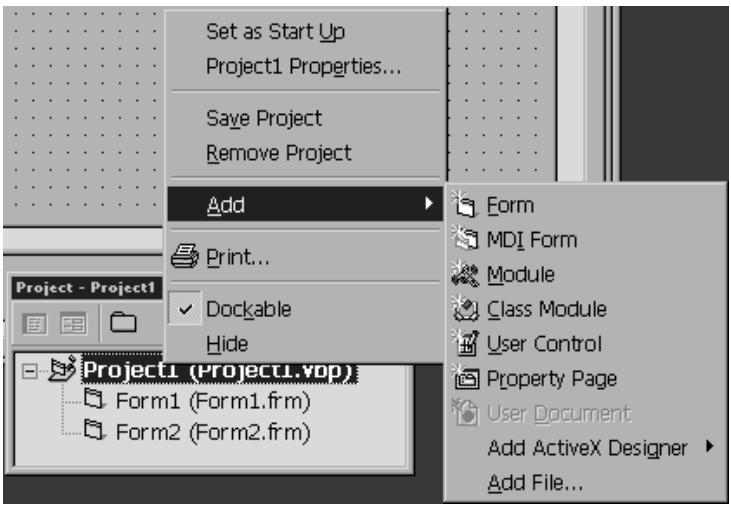
Рис. 2.3. Окно **Проводника проекта (Project Explorer)** (справа) и пиктограмма его вызова на стандартной панели инструментов

Рассмотрим вторую сверху строку в изучаемом окне. В ней находятся 3 пиктограммы; подводя к ним указатель мыши, в окне указателя мы можем прочесть их названия. Левая пиктограмма называется **View Code** (Просмотр кода), средняя — **View Object** (Просмотр объекта), правая — **Toggle Folders** (Переключение папок). Первые две пиктограммы обеспечивают проектировщику приложения возможность перехода от его визуальной составляющей к программной (т.е. к кодам обработчиков событий). Третья пиктограмма переключает режим отображения структуры файлов проекта в окне: в форме обычного списка либо в виде дерева.

Окно проводника проекта предназначено для быстрого доступа к той или иной экранной форме. Так, посредством двойного клика по имени содержащего экранную форму файла мы активизируем ее и отображаем на экране поверх остальных. Щелкнув по пиктограмме просмотра кода **(View Code)**, мы входим в окно текстового редактора кода VB5, обеспечивающее доступ к обработчикам событий для активной формы и содержащихся в ней управляющих элементов.

Контекстное меню, вызываемое по щелчку правой клавишей внутри окна проводника проекта, обеспечивает быстрый доступ ко многим командам, дублирующим команды пунктов **File** и

Project главного меню. Состав этого меню зависит от того, какой тип файла выделен в настоящий момент в списке окна, — файл проекта или файл формы. Более полным является меню при выделенном файле проекта (для полноты картины показано также подчиненное меню команды **Add**):



Пункт **Set as Start Up** (Установить как стартовый) имеет смысл в случае одновременной работы с несколькими проектами. Второй пункт (**Project1 Properties...**) вызывает диалоговое окно свойств проекта, дублируя соответствующую команду из пункта **Project** главного меню. Другие команды из пункта **Project** дублирует и команда **Add** с набором вариантов из подчиненного меню. Команда **Print** (Печать) позволяет распечатывать как код файла экранной формы, так и вид самой формы. Команды **Print, Remove Project** (Удалить проект) и **Save Project** (Сохранить проект) дублируют команды пункта **File** главного меню.

- Следующим важнейшим для работы окном является окно свойств **(Properties Window)**. Оно предназначено для установки свойств объектов на стадии проектирования. Для его выставления можно использовать пиктограмму стандартной панели инструментов, соответствующую команду из пункта **View** главного меню либо клавишу F4 (*рис.* 2.4).

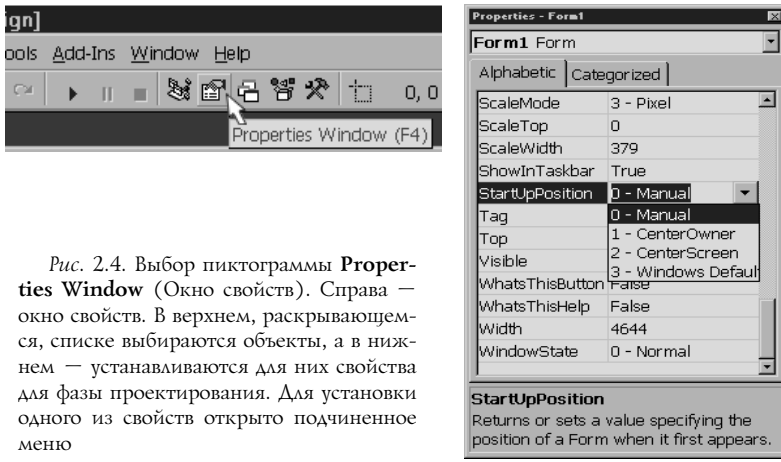


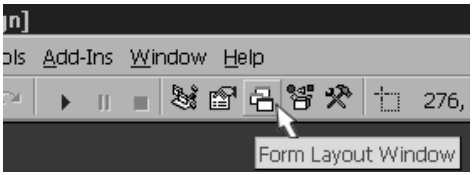
Рис. 2.4. Выбор пиктограммы **Properties Window** (Окно свойств). Справа — окно свойств. В верхнем, раскрывающемся, списке выбираются объекты, а в нижнем — устанавливаются для них свойства для фазы проектирования. Для установки одного из свойств открыто подчиненное меню

Список свойств разделен на две колонки. В левой находятся имена свойств указанного в верхнем окошке объекта, а в правой — значения, установленные для этих свойств по умолчанию либо программистом. Как видно из рисунка, для некоторых свойств предусмотрена возможность выбора из списка, раскрывающегося после клика по значку ⮟, появляющемуся в поле значения после выделения имени свойства.

Список свойств присутствует на двух вкладках — **Alphabetic** (Алфавитный) и **Categorized** (По категориям), обозначающих принцип упорядочения списка свойств: по алфавиту либо по категориям. Переключаться на вкладку **Categorized** может быть удобно тогда, когда имя свойства забыто, но известны его функциональные признаки.

Отметим, наконец, что под списком свойств в окне **Properties** выводится краткое описание текущего выделенного свойства.

- Последним из инструментальных окон является окно **Form Layout** (Внешний вид формы), служащее для задания положения экранной формы на экране при его загрузке **(Load)** в фазе выполнения проекта (*рис.* 2.5). Размеры формы устанавливаются в окне формы перетаскиванием размерных маркеров (темных квадратиков), расположенных на правой и нижней границах формы. А вот задавать положение формы с установленными размерами на экране во время выполнения можно, вызывая окно **Form Layout** и перетаскивая форму мышью внутри стилизованного изображения экрана



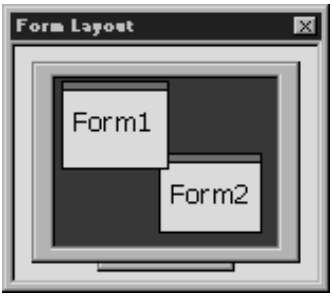


Рис. 2.5. Выбор пиктограммы **Form Layout Window** (Окно внешнего вида). Справа — размещение на экране с помощью окна двух форм

монитора. Для точной установки формы в центре экрана можно вызвать контекстное меню окна и в подчиненном меню пункта **Startup Position** (Стартовая позиция) выбрать значение **Center Screen** (Центр экрана). Альтернативными значениями являются: **Manual** (Задавать вручную), **Center Owner** (По центру области приложения) и **Windows Default** (По умолчанию — форма в верхнем левом углу экрана).

С основными инструментальными окнами мы ознакомились. Займемся теперь проектированием нашего первого приложения.

Проект "Калькулятор"

Калькулятор относится к числу наиболее простых приложений: всем известны и сценарий работы с ним, и реализующие его работу алгоритмы. Есть у этого приложения как у учебного и недостаток: оно уже многократно реализовано, в том числе и как стандартное приложение MS Windows. Как мы увидим, недостаток этот мнимый: ничто не мешает нам постепенно делать наш калькулятор более удобным и многофункциональным, изучая при этом используемые средства VB5.

Размещение элементов

Главный, как нам кажется, недостаток стандартного калькулятора — это единое окно для ввода одного или двух операндов и для вывода результата операции. Стоит чему-то отвлечь наше внимание — и мы забываем, на каком шаге вычисление сложного выражения прервалось. Поэтому начертим на бумаге схему нашего будущего калькулятора, в котором окна ввода операндов и чтения результата разнесены.

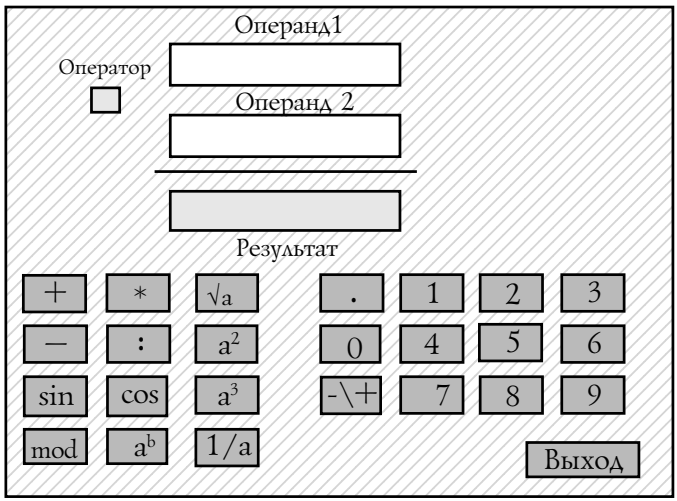


Рис. 2.6. Начальная схема калькулятора

На *рис.* 2.6 приведен пример начального варианта такой схемы. На ней имеются два поля с именами **Операнд 1** и **Операнд 2**, предназначенные для ввода операндов; имеются поля отображения выполняемой операции и результата (**Оператор** и **Результат** соответственно); имеются восемь командных кнопок со стандартным для калькуляторов назначением, а также кнопка **Выход** для окончания работы.

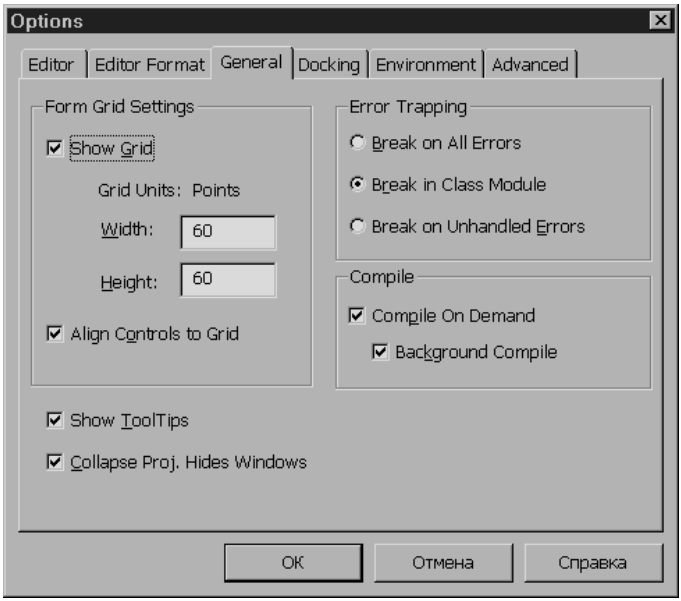
Сценарий работы пользователя с калькулятором таков: в полях ввода вводится необходимое для операции число операндов, после чего кликом по командной кнопке выполняется требуемая команда, отображаемая в поле оператора; в поле результата читается результат. Если результат должен далее участвовать как операнд в следующей операции, то кликом мыши в поле результата мы копируем его в поле первого операнда.

Перейдем непосредственно к проектированию интерфейса программы.

Подготовка среды проектирования

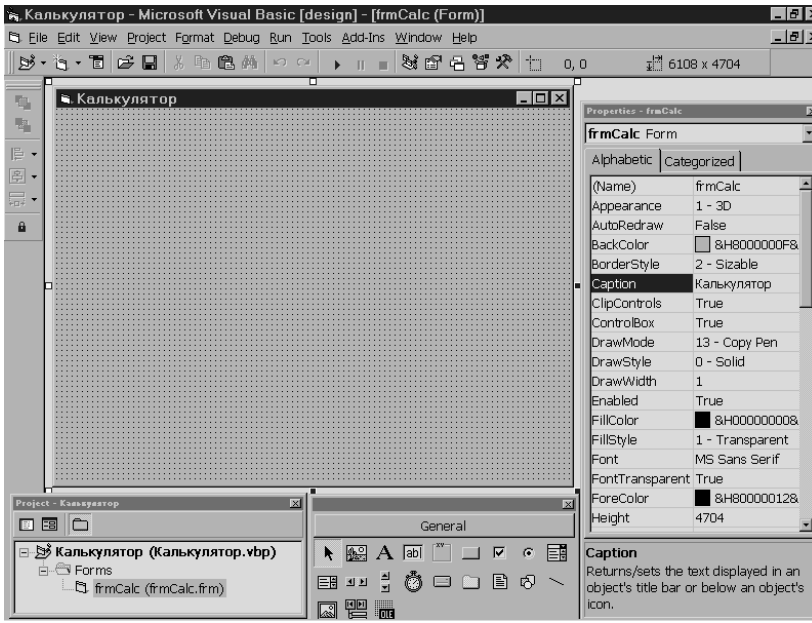
Открыв новый проект, установим все нужные нам инструментальные окна (без окна **Form Layout**) так, чтобы форму можно было растянуть, не "налезая" на них, как можно больше, сохраняя приблизительно пропорции нашей схемы. Получается форма размером примерно 106×77 мм.

Обратим внимание на точечную сетку, которой покрыта форма. Ее назначение — помогать проектировщику выравнивать элементы управления и их части по горизонтали и вертикали. Настройка сетки производится на вкладке **General** (Общее), путь на которую — **Tools ➢ Options ➢ General**:



На этой вкладке в области **Form Grid Settings** (Параметры сетки формы) мы установим вдвое более мелкий по сравнению со стандартным 120×120 шаг сетки **(Grid Units)**; включенный переключатель **Align Controls to Grid** означает, что границы элементов управления будут "притягиваться" к параллельным рядам точек сетки, оказавшись вблизи них.

Изменим имя нашего проекта с **"Project1"** на **"Калькулятор"**. Для этого щелкнем по файлу **Project1.vbp** в окне проекта и введем для его единственного свойства **(Name)** в окне свойств новое имя. Щелкнем по форме (или по файлу **Form1.frm** в окне проекта) и изменим имя формы (свойство **Name**) на **frmCalc**, а надпись (свойство **Caption**), которая указывается в заголовке окна формы, — на **"Калькулятор"**. Получим следующий исходный вид экрана перед началом проектирования:



Размещение элементов управления

❐ Разместим (двойным щелчком и последующим перемещением) поля ввода операндов (элемент панели инструментов **TextBox**: ab). Сначала установим поле ввода первого операнда, отрегулируем его размеры. Элемент получает по умолчанию имя **Text1**. Чтобы не регулировать размеры равного ему поля ввода второго операнда, создадим последний посредством копирования первого в буфер обмена **(Clipboard)**, что дает возможность вставки из него любого числа данных копий.

Войдя на выделенном поле в контекстное меню (щелчок правой клавишей), выберем команду **Copy**. Затем, войдя повторно в контекстное меню, выберем **Paste** (Вставить) (вместо контекстного меню можно пользоваться привычными комбинациями Ctrl C и Ctrl V соответственно). Появляется диалог с запросом: **“You already have a control “Text1”. Do you want to create a control array?”** (“У вас уже есть элемент “Text1”. Вы

хотите создать массив элементов?") — и вариантами ответов: **"Да"** и **"Нет"**. Выбрав **"Нет"**, мы получаем копию элемента **Text1** с именем **Text2** (убедитесь в этом с помощью окна свойств) и той же надписью (свойство **Caption**) внутри поля.

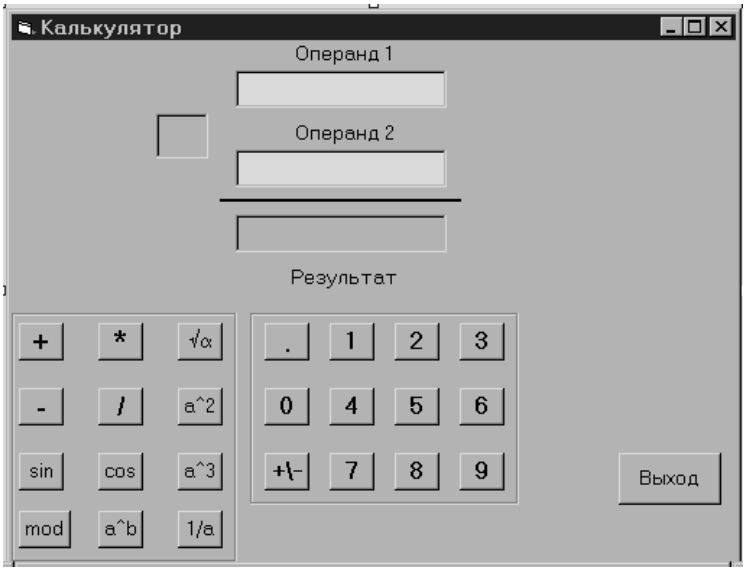
- Создание программы — это итеративный процесс, когда на некотором этапе приходится возвращаться назад и корректировать уже спроектированное. Почему, например, мы выбрали вариант копирования, а не создания массива элементов (т.е. элементов, к которым обращаются по единому имени, но со своим индексом, записываемым после имени в круглых скобках)? Нам показалось, что раз элементов будет всего два, никаких преимуществ в представлении их элементами массива нет. В дальнейшем, однако, выяснилось, что это не так. Оказалось, что, вводя переменную **i**, принимающую два значения: **0** — если последним активизировалось ("получало фокус") поле ввода для первого операнда и **1** — если для второго, — мы можем написать *единые* обработчики для каждого события, действительного для обоих полей. С учетом этого удалим скопированный элемент (выделяя и нажимая Del) и повторим описанный процесс вплоть до запроса на создание массива, где выберем уже ответом **"Да"**. После этого на форме, точно так же, как и в предыдущем варианте, появится второе поле ввода с надписью **Text1**, но, заглянув в список элементов окна свойств, мы обнаружим, что вместо элементов **Text1** и **Text2** у нас появились элементы **Text1(0)** и **Text1(1)**. По соглашению об именах объектов (введенному фирмой Microsoft) изменим, пользуясь свойством **Name** в окне свойств, общее имя **Text1** этих элементов на **txtOp**; надписи (значения свойства **Caption**) **"Text1"** удалим.

- Разместим блок из 12 командных кнопок (элемент панели инструментов **CommandButton**:), из которых 10 будут отвечать за ввод в активное поле ввода цифр (от 0 до 9), 11-я — за ввод точки, отделяющей дробную часть от целой (если в системе для этой цели используется запятая, то нужно либо поменять данную установку — на вкладке **Числа** раздела **Язык и стандарты** Панели управления Windows 95, — либо самим использовать запятую), и 12-я — за изменение знака введенного числа. Первые 11 из указанных кнопок создадим в форме массива **cmdNum** (индекс меняется

от 0 до 10), 12-ю — отдельным элементом с именем **cmdOpposit**. В качестве надписей для первых 11 кнопок будут введены соответствующие им вводимые символы, для кнопки изменения знака — надпись **"–\+"**.

- Разместим посредством копирования 12 командных кнопок для выполнения операций как бинарных (для двух операндов) — аддитивных (**"+"**, **"–"**), мультипликативных (**"*"**, **"/"**), возведения в степень (a^b), вычисления остатка от деления **(mod)**, так и операций вычисления функций от активного (получавшего фокус последним) операнда (функции **sin, cos,** $\sqrt{a}$ **,** a^2**,** a^3**,** $1/a$). Для каждой из кнопок в окне свойств изменим имена объектов (на **cmdPlus, cmdMinus, cmdMult, cmdDevide, cmdPower, cmdMod, cmdSin, cmdCos, cmdRoot, cmdSquare, cmdCobe, cmdReverse** соответственно) и установим для них соответствующие математическим обозначениям их операций надписи (вместо надписей вида a^b мы вынуждены писать $a\hat{}b$).
- Каждую из описанных групп командных кнопок возьмем в рамку (элемент панели инструментов **Frame**:).
- Установим 6 ярлыков (элемент панели инструментов **Label**: **A**). Ярлыки используются как для создания постоянных надписей на форме, так и для вывода данных в фазе работы программы. Так, два из наших ярлыков будут служить для вывода результата **(lblRes)** и знака выполненной бинарной операции **(lblOper)**. Их надписи устанавливают в фазе проекта пустыми. Кроме того, для большей выразительности мы сделаем наши ярлыки "утопленными", установив свойство **BorderStyle** (Стиль границы) в **1 – Fixed Single** (Фиксированная одиночная). Три оставшихся ярлыка создадут постоянные надписи "Операнд 1" (**lblOp1**), "Операнд 2" (**lblOp2**) и "Результат" **(lblResult)** над соответствующими полями ввода и под ярлыком **lblRes**. Изменим свойство **Alignment** (Выравнивание) для надписей у этих ярлыков на **2 – Center** (Центральное). Последний ярлык, не видимый в фазе проекта (т.е. с пустой надписью и без "утопления"), установим над группой операционных кнопок. Назначение этого ярлыка **lblMes** — выводить предупреждающие сообщения при попытках некорректных вычислений (деления на 0, например).
- Для стилизации под вычисление "столбиком" отделим операнды от результата горизонтальной линией (элемент панели инструментов **Line**:).

Начальный вариант проекта **"Калькулятор"** готов. Для более удобного (чем щелчок указателем мыши по маленькой кнопке закрытия окна) выхода из программы создадим еще командную кнопку **cmdExit** с надписью **"Выход"**. Уберем ненужную теперь сетку, выключив опцию **Show Grid** на рассмотренной выше вкладке. Спроектированная форма нашего приложения имеет следующий вид:

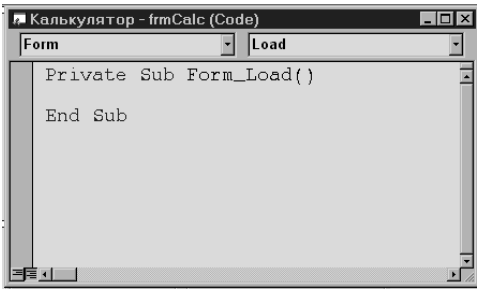


В дальнейшем мы можем обнаружить, что во время работы с калькулятором щелканье мышью по кнопкам с надписями типа *a^b* приводит к тому, что эти надписи "рассыпаются" на несколько строк. Чтобы избежать этого, нужно слегка увеличить ширину этих кнопок, вводя новые значения в поле их свойства **Width** (Ширина).

Программирование набора операндов

Теперь мы познакомимся с окном программного кода и сервисов встроенного редактора, представленного этим окном.

Как мы помним, вызвать это окно можно двумя способами: щелкнув по пиктограмме просмотра кода в окне проводника проекта либо двойным щелчком по какому-либо элементу управления, например, по самой форме. После этого мы окажемся в окне примерно такого вида:



Устройство этого окна мы уже разбирали в предыдущей главе. Так, под строкой заголовка мы видим два раскрывающихся списка: в левом — элементы управления (они представлены своими "экземплярными" именами), а также форма, которая представлена своим типовым именем **Form**), в правом — события, определенные для выбранного элемента или формы. Под списками расположено окно редактирования кода, причем курсор располагается внутри кода того обработчика, которым обрабатывается выбранное в правом списке событие.

В левом списке первым пунктом всегда является **General**, обозначающий *секцию общих объявлений (General Declaration section)* нашего программного модуля (представляющего в данном случае файл с текстом программного модуля экранной формы). В этой секции объявляются переменные уровня модуля **(Private)** либо общедоступные **(Public)**.

Термин "переменная" нами еще не использовался, и его нужно определить. Ведь до сих пор мы говорили только об объектах — элементах управления, имеющих свойства и методы. В подглаве "Объект и его "паспорт" мы видели, как свойствам объекта можно присваивать числовые, логические или текстовые значения с помощью констант — самоопределенных данных, представляющих в программе именно то значение, которое выражено их записью. А вот переменные — это такие элементы программы, которые записываются с помощью имен-идентификаторов, а представлять в программе могут разные значения, которые тем или иным способом им *присваиваются*. В какой-то степени свойства — это частный случай переменных, существующих только вместе с объектом и имеющих заданные создателями этих объектов имена. Но так же как в VB5 есть необходимость в процедурах, не являющихся обработчиками, так в нем есть нужда и в именованных величинах, не связанных с какими-либо объектами. Их и принято называть "переменными". (Сказанное не относится, разумеется, к уже упоминавшимся объектным переменным, хранящим адреса объектов.)

Переменная — это представленный именем-идентификатором элемент программы, который служит для обращения к данному определенного *типа*, хранимому в ячейке оперативной памяти, и обеспечивает связь с одной и той же ячейкой в течение определенной фазы времени работы программы (имеет определенное "*время жизни*").

Идентификатор, представляющий имя переменной, строится по определенным правилам (пока будем использовать в качестве таких идентификаторов просто строчные буквы английского алфавита). И, кроме того, условимся, что перед использованием переменной ее необходимо явно объявить, т.е. указать, что переменная, имеющая такое-то имя, имеет такой-то тип. В этом же объявлении можно также указать, какое время жизни имеет данная переменная. Подобное объявление переменной осуществляется с помощью *инструкции* (говорят еще — *оператора*) **Dim**. В простейшем случае оно имеет вид:

Dim Имя_переменной As Имя_типа

Здесь на месте элементов, названных по-русски, в "настоящем" объявлении стоят конкретные имена переменных и требуемые имена типов, например:

```
Dim i As Integer 'объявляется переменная i целого типа
Dim s As String 'объявляется переменная s типа строки символов
```

(апостроф используется для указания того, что после него и до конца данной текстовой строки идет *комментарий* (вообще говоря, набор *любых* символов), обычно представляющий замечания программиста по написанному им фрагменту программы). В одной инструкции **Dim** можно объявить несколько переменных, указывая тип для каждой из них.

В отличие от имен переменных, идентификаторы которых составляет сам программист, имена типов (**Integer**, **String** и т.д.), равно как и имена инструкций (**Dim**, например), являются установленными раз и навсегда, *зарезервированными*. Понятно, что не стоит их использовать в качестве идентификаторов переменных (а идентифицировать объекты, кстати, можно!).

Точно так же, как наряду с одним экземпляром объекта мы можем создавать их массив, обращаясь далее к нужному экземпляру с помощью индекса, мы, кроме одиночной переменной, можем объявлять массив переменных с заданным именем:

```
Dim a(500) As Integer
```

— обращаясь далее в программе к конкретному элементу массива ("индексированной переменной") с помощью точно такого же индекса:

```
a(i) = a(i)+1 'Увеличили i-й элемент массива на 1
```

Индекс — это просто номер элемента в массиве; в качестве этого номера можно использовать целую константу, целую переменную, а также арифметическое выражение более общего вида, то есть в котором используются константы, переменные, обращения к функциям (разновидностям процедур), знаки арифметических действий:

```
j = a(80*(m-1)+n) 'В скобках после a - индексное выражение
```

При обращении к индексированной переменной стоящее на месте ее индекса арифметическое выражение вычисляется и приводится, если оно не целое, к целому числу путем округления дробной части; полученное значение является номером элемента в массиве, к которому осуществляется обращение.

Что же необходимо поместить нам в секцию **General**?

Как уже было намечено при размещении на форме полей ввода операндов нашего калькулятора, представляющих элементы массива из двух элементов (индекс первого — 0, второго — 1), мы должны иметь переменную, которая все время работы нашей программы должна сохранять индекс того элемента — поля ввода, которое последним получало фокус. Кроме того, эта переменная должна быть "видна" в любом обработчике. Оба эти условия (время жизни и область видимости) обеспечиваются объявлением данной переменной не внутри какой-либо процедуры, а в секции общих объявлений в "самом верху" нашего программного модуля:

```
Dim i As Integer
```

Перейдем теперь непосредственно к программированию набора операндов.

Собственно, поля ввода именно и служат для того, чтобы этот ввод вообще не нужно было программировать. Так, если мы установим курсор в поле ввода (возьмем этот объект тем самым "в фокус") и начнем набор символов на клавиатуре, то набираемая строка на каждом шаге набора будет значением свойства **Text** (Текст) поля ввода и в качестве этого значения будет в поле ввода отображаться. Но все это будет возможно при вводе данных *с клавиатуры*. Мы же хотим создать калькулятор, в котором все управление, а значит, и набор операндов, осуществляется *мышью* — как в стандартном калькуляторе, входящем в состав Windows. Для этого-то мы и вводили кнопки с цифрами и десятичной точкой. Значит, нам нужно самим запрограммировать обработчик "нажатий" на эти кнопки мышью так, чтобы при каждом

нажатии изображенный на кнопке символ "подцеплялся" к строке, являющейся значением свойства **Text**.

Вернемся через кнопку **View Object** (Просмотр объекта) окна проекта к форме и сделаем двойной щелчок по любой из перечисленных кнопок. Перед нами возникнет следующая заготовка обработчика:

```
Private Sub cmdNum_Click(Index As Integer)
End Sub
```

— представляющая собой "процедурные скобки": строку заголовка процедуры и строку ее завершения. Тело процедуры пока пусто; эту пустоту мы и должны заполнить нужным нам кодом.

Подцепление символа производится с помощью одной из двух операций — "+" или "&" (перед значком "&" — "амперсандом" — нужно обязательно самим ставить пробел!). Таким образом, наш обработчик может быть таким:

```
Private Sub cmdNum_Click(Index As Integer)
 If Index < 10 Then
  txtOp(i).Text = txtOp(i).Text & Index 'Приписываем цифру,
 Else
  txtOp(i).Text = txtOp(i).Text & "." 'или точку
 End If
End Sub
```

Несмотря на свою простоту, текст приведенного обработчика нуждается в ряде замечаний.

- Переменная **Index** по своему смыслу является *параметром*, то есть такой переменной, которая получает значение в момент вызова процедуры. В данном случае вызов происходит в момент возникновения события **Click** при щелчке мышью по кнопке, являющейся элементом массива объектов — командных кнопок **cmdNum**. Эти элементы идентифицируются значениями индексов от 0 (кнопка с цифрой "0") до 10 (кнопка с точкой); индексы цифровых кнопок равны изображенным на них цифрам. Именно значения этих индексов и передаются параметру **Index** данного обработчика, единого для всех кнопок — элементов массива, указывая таким образом обработчику, к какому именно элементу относится произошедшее событие **Click**, а следовательно, какую именно цифру надо присоединять к значению свойства **Text**.

  Переменная **Index** объявлена внутри процедуры и потому является *локальной*; ее область видимости — только текст данного обработчика. Время ее жизни — от момента вызова обработчика до окон-

чания его работы (*автоматическая* переменная). Любая локальная переменная является автоматической, если она не объявлена с ключевым словом **Static** либо если с этим словом не объявлена процедура, содержащая объявление этой переменной. В обоих этих случаях локальная переменная будет *статической*, т.е. сохраняющей свои значения между вызовами процедуры.

Объявление *параметра*, как мы видим, производится не так, как мы рассказали выше, а в круглых скобках после имени обработчика. Тем самым одновременно указывается и то, что эта переменная является параметром, и ее тип **(Integer)**.

- В данном обработчике использована управляющая конструкция, называемая *ветвление*. Ведь нам надо присоединять к тексту в поле ввода либо символьное представление величины **Index**, если **Index** < **10**, либо точку (символ "**.**"). Мы так и записываем:

```
Если Index < 10 То
   txtOp(i).Text = txtOp(i).Text & Index
Иначе
   txtOp(i).Text = txtOp(i).Text & "."
```

На языке Visual Basic это предложение записывают тем же самым образом, но ключевые слова **Если**, **То**, **Иначе** пишутся по-английски: **If**, **Then**, **Else**. Кроме того, мы должны как-то указать окончание второй ветви (после **Иначе**). В естественном языке для этого мы используем точку. В тексте программы на VB используют словосочетание **Конец Если (End If)**.

- Ключевое слово **Private** (Закрытая) означает, что данную процедуру можно вызвать по имени только из содержащего ее программного модуля (файла). Без этого слова, а также с ключевым словом **Public** (Общедоступная) процедура доступна и из других модулей проекта. Оба эти слова можно использовать и в объявлениях переменных вместо слова **Dim**; по умолчанию закрытой является локальная переменная, а общедоступной — объявленная на уровне модуля (секция **General**).

- Обратите внимание: "умный" компилятор понимает, что мы хотим к символьной величине **Text** присоединить именно символ, являющийся десятичным изображением числа **Index**. То есть происходит автоматическое преобразование величины целого типа к величине символьного. Если бы этого не происходило, то подцеплялся бы не символ "5" (имеющий код по кодовой таблице 53), а псевдосимвол (имеющий по кодовой таблице код 5).

- Мы неспроста решили отделять дробную часть вводимого числа точкой, а не запятой, что в принципе допускается настройкой Windows. Если, однако, у вас настройкой предусмотрена именно запятая, то поменять надо именно настройку (во избежание очень неприятных неожиданностей при работе с некоторыми функциями, например, **Val**)! Для этого из окна **Панели управления** надо войти в окно **Язык и стандарты** и на вкладке **Числа** поменять разделитель целой и дробной частей с запятой на точку. Проконтролируйте это заранее!

Завершение проекта

Действия в полях ввода **txtOp(i)**, где **i** принимает значения 0 или 1, мы запрограммировали. Осталось, по существу, запрограммировать вычисление результатов операций при нажатии на каждую из операционных кнопок (событие **Click**) и вывод результатов либо в активное поле ввода (если операция имеет один операнд — обратная величина, противоположная величина, квадрат, куб операнда и квадратный корень из него, другие функции одной переменной от операнда), либо в ярлыке результата **lblRes** для *бинарных* операций (сложение, вычитание, умножение, деление, возведение в степень, остаток от деления). Учтем, что для бинарных операций их знак должен отображаться ярлыком **lblOper**. Наконец, предусмотрим такую ситуацию: нас что-то отвлекло во время многоэтапных вычислений, а когда мы вернулись к ним, то не помним, является ли число в ярлыке результата действительно результатом отображаемой ярлыком **lblOper** операции над теми операндами, которые находятся в данный момент в полях ввода, или какой-то из этих операндов мы успели изменить для последующих вычислений. Поэтому запрограммируем изменение фона (свойство **BackColor**) у ярлыков **lblOper** и **lblRes** при выполнении бинарной операции и восстановление у них фона при изменении содержимого любого поля ввода (событие **Change**). Таким образом, действителен будет только тот результат, который расположен на измененном фоне ярлыка.

Программный код для обработчиков событий следующий:

```
Dim i As Integer ' - Секция общих объявлений (деклараций)

Private Sub cmdCos_Click()                  'Косинус
 txtOp(i).Text = Cos(Val(txtOp(i).Text))
End Sub
Private Sub cmdCube_Click()                 'Возведение в куб
 txtOp(i).Text = Val(txtOp(i).Text) ^ 3
End Sub
```

```
Private Sub cmdDevide_Click()                'Деление
 If Val(txtOp(1).Text) <> 0 Then
   lblRes.Caption = Val(txtOp(0).Text) / Val(txtOp(1).Text)
 Else
   lblMes.Caption = "Делить на 0 не умею!!!"
 End If
 lblRes.BackColor = &H80000009
 lblOper.Caption = "/"
 lblOper.BackColor = &H80000009
End Sub
Private Sub cmdMinus_Click()                  'Вычитание
 lblRes.Caption = Val(txtOp(0).Text) - Val(txtOp(1).Text)
 lblRes.BackColor = &H80000009
 lblOper.Caption = "-"
 lblOper.BackColor = &H80000009
End Sub
Private Sub cmdMod_Click()                   'Остаток от деления
 lblRes.Caption = Val(txtOp(0).Text) Mod Val(txtOp(1).Text)
 lblRes.BackColor = &H80000009
 lblOper.Caption = "mod"
 lblOper.BackColor = &H80000009
End Sub
Private Sub cmdMult_Click()                   'Умножение
 lblRes.Caption = Val(txtOp(0).Text) * Val(txtOp(1).Text)
 lblRes.BackColor = &H80000009
 lblOper.Caption = "*"
 lblOper.BackColor = &H80000009
End Sub
Private Sub cmdOpposit_Click()               'Противоположное
 txtOp(i).Text = -Val(txtOp(i).Text)
End Sub
Private Sub cmdPlus_Click()                   'Сложение
 lblRes.Caption = Val(txtOp(0).Text) + Val(txtOp(1).Text)
 lblRes.BackColor = &H80000009
 lblOper.Caption = "+"
 lblOper.BackColor = &H80000009
End Sub
Private Sub cmdPower_Click()             'Возведение в степень
 If Val(txtOp(0).Text) = 0 And Val(txtOp(1).Text) <= 0 Then
  lblMes.Caption = "Думай, что делаешь!"
 Else
  lblRes.Caption = Val(txtOp(0).Text) ^ Val(txtOp(1).Text)
  lblRes.BackColor = &H80000009
  lblOper.Caption = "^"
  lblOper.BackColor = &H80000009
 End If
End Sub
Private Sub cmdReverse_Click()                'Обратное
 If Val(txtOp(i).Text) <> 0 Then
  txtOp(i).Text = 1 / Val(txtOp(i).Text)
 Else
  lblMes.Caption = "Делить на 0 не умею!!!"
 End If
End Sub
```

```
Private Sub cmdRoot_Click()                 'Квадратный корень
 If Val(txtOp(i).Text) < 0 Then
    lblMes.Caption = "Корень из отрицательного - это круто!"
 Else
  txtOp(i).Text = Sqr(Val(txtOp(i).Text))
 End If
End Sub
Private Sub cmdSin_Click()                          'Синус
 txtOp(i).Text = Sin(Val(txtOp(i).Text))
End Sub
Private Sub cmdSquare_Click()           'Возведение в квадрат
 txtOp(i).Text = Val(txtOp(i).Text) ^ 2
End Sub
```

В последнем фрагменте мы оформляем присвоение начального значения глобальной переменной **i**, хранящей индекс поля ввода, бывшего в фокусе последним. Начальные присвоения делаются обычно при загрузке формы (событие **Load**):

```
Private Sub Form_Load()             'Загрузка формы
 i = 0 'Присвоение начального значения переменной i
End Sub
```

В процессе же работы калькулятора значения этой переменной будут присваиваться при каждом получении фокуса тем или иным полем ввода (а возможные предупреждающие сообщения от предыдущей операции гасятся):

```
Private Sub txtOp_GotFocus(Index As Integer) 'Получение
 lblMes.Caption = ""                          'фокуса полем ввода
i = Index 'Запоминаем в i индекс поля ввода, получавшего фокус последним
End Sub
```

При любых изменениях в полях ввода восстанавливается первоначальный цвет фона у ярлыков операции и результата:

```
Private Sub txtOp_Change(Index As Integer) 'Изменение поля
  lblOper.BackColor = &H8000000A            'ввода
  lblRes.BackColor = &H8000000A
End Sub
```

Для быстрой очистки полей ввода мы используем двойной клик:

```
Private Sub txtOp_DblClick(Index As Integer) 'Очистка поля
 txtOp(i).Text = ""                           'ввода
End Sub
```

И, наконец, запрограммируем кнопку выхода (хотя всегда есть возможность выйти путем закрытия окна программы стандартным средством управления окном):

```
Private Sub cmdExit_Click()
 End
End Sub
```

Первая версия калькулятора создана! В ней всего 60 строк кода, введенного нами с клавиатуры.

Запустим нашу задачу в среде VB5. Для этого выполним команду **Start** из пункта **Run** главного меню, либо нажмем на F5, либо щелкнем по пиктограмме **Start** стандартной панели инструментов:

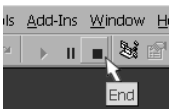
Если для текущего варианта кода мы делаем это в первый раз, то системе VB5 потребуется определенное время, чтобы скомпилировать наш текст (перевести его в машинные коды) и осуществить компоновку исполнимого модуля программы (подключить необходимые библиотеки с используемыми вами и системой функциями). При последующих запусках в одном сеансе работы с VB5 программа будет запускаться мгновенно (если код ее не менялся).

Итак, если вдруг вы ввели код программы *без ошибок*, то после запуска программы на экране поверх других окон появляется окно нашего калькулятора. Протестируем его — и убедимся, что он вполне работоспособен. Вы спросите: о каких ошибках, собственно, идет речь? Ведь при вводе кода интеллектуальный редактор VB5 анализировал вводимый текст и немедленно выдавал предупреждения о сделанных ошибках... Но дело в том, что ошибки программиста бывают разными. Редактор же позволяет отслеживать только синтаксические.

Процесс выявления и устранения ошибок в программе называется отладкой. Англоязычный эквивалент — "*debug*" ("избавление от жучков"). Полезно помнить клавиши быстрого вызова, позволяющие выполнять те или иные отладочные действия: устанавливать точки останова в той или иной строке, использовать окна наблюдений и т.д. Мы не будем рассматривать эту сторону создания приложения, оправдывая себя лишь тем, что необходимость в знании средств отладки появляется при программировании существенно более серьезных задач, а значит, этот пробел можно будет восполнить на последующих этапах (само)образования.

Ну а сейчас научимся прерывать выполнение запущенной программы. Конечно, если она работает нормально, мы и выйти из нее можем регламентированным образом: стандартным средством закрытия окна программы либо специально запрограммированной кнопкой. А если программа прервалась по какой-либо ошибке фазы выполнения (*run-time error*)?

Тогда мы должны закончить фазу выполнения с помощью щелчка по пиктограмме **End** стандартной панели:



Если же в процессе отладки мы не хотим выходить из фазы выполнения, а лишь приостановить выполнение, мы должны щелкнуть по пиктограмме **Break**:

Обе пиктограммы дублируют соответствующие команды пункта **Run** главного меню.

А теперь работайте сами!

Наш замечательный (тем, что он *наш*) калькулятор имеет ряд недочетов. Перечислим их.

- Представим, что мы набрали "длинный" операнд и ошиблись в нескольких цифрах в середине числа. Мы можем выделить их мышью и либо удалить, нажав клавишу Del (или BackSpace), либо при надобности замены сразу набирать с клавиатуры новые значения. Но вот сделать эти действия мышью, без участия клавиатуры, мы не сможем. Более того, скоро выясняется, что даже простую вставку символов внутри набранного операнда мы можем осуществлять только с клавиатуры: "набор" с помощью мыши и командных кнопок добавляет символы только в конец операнда независимо от текущего положения курсора в поле ввода! Ничего другого использованная нами операция конкатенации и не могла нам обеспечить. Что же делать?

  Самая очевидная идея — отслеживать положение курсора внутри поля ввода — неожиданно завела в тупик: таких средств в Visual Basic попросту не оказалось (а зря!). Выход состоит в использовании инструкции **SendKeys Строка**, которая имитирует передачу активному окну одного или нескольких нажатий клавиш с символами, представленными в параметре **Строка**, как если бы они были действительно нажаты на клавиатуре.

Каждое такое "нажатие" мы сопроводим удалением выделенного фрагмента операнда, если таковой существует. Такой фрагмент доступен для чтения и изменения в свойстве **SelText** поля ввода. Вот как будет выглядеть обработчик **cmdNum_Click** после модификации (замененный вариант оставлен как комментарий):

```
Private Sub cmdNum_Click(Index As Integer)
 txtOp(i).SelText = ""  'Удаляем выделенное (если оно есть) перед
                        'его заменой
 txtOp(i).SetFocus
 If Index < 10 Then
   ' txtOp(i).Text = txtOp(i).Text & Index
   SendKeys Index 'Приписываем цифру
 Else
   ' txtOp(i).Text = txtOp(i).Text & "."
   SendKeys "." 'или точку
 End If
End Sub
```

- Еще одно средство, которого мы пока не имеем, — это имитация действий клавиш BackSpace и Del. Поэтому введем эти клавиши (с именами **cmdBksp** и **cmdDel**) и используем ту же инструкцию **SendKeys**, в которой специальным образом, с помощью встроенных констант (их имена приведены в справке по данной инструкции), можно задавать имитацию нажатий несимвольных клавиш и даже их комбинаций:

```
Private Sub cmdBksp_Click()
 txtOp(i).SetFocus
 SendKeys "{BKSP}"
End Sub

Private Sub cmdDel_Click()
 txtOp(i).SetFocus
 SendKeys "{DEL}"
End Sub
```

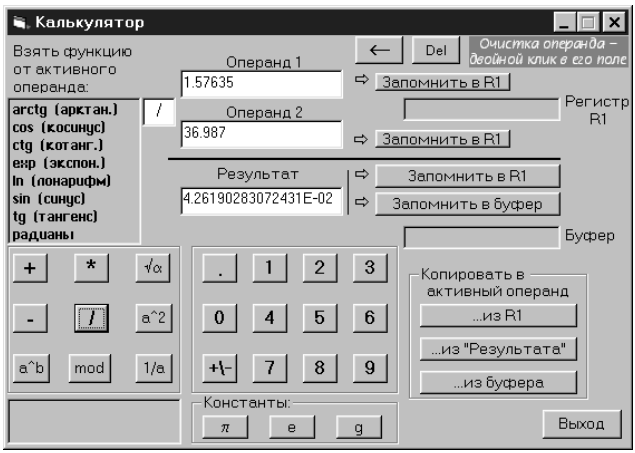
- Последнее, для чего мы еще вынуждены использовать клавиатуру, — это запоминание результата в буфере обмена **(Clipboard)** и копирование из него значения в активный (находящийся в фокусе) операнд. Оказывается, у нас всегда имеется возможность обратиться с этими целями к объекту **Clipboard**, метод которого **SetText** запоминает в нем текстовую строку (в том числе значение свойства какого-либо другого объекта):

```
Private Sub cmdToClip_Click()
 Clipboard.SetText lblRes.Caption 'Запоминаем результат
                                  'в буфере
End Sub
```

— а метод **GetText** копирует в строку из буфера обмена в строковую переменную (в том числе свойство):

```
Private Sub cmdFromClip_Click()
 txtOp(i).Text = Clipboard.GetText 'Копируем из буфера
                                   'обмена
End Sub
```

Ну а теперь в качестве упражнения предлагаем дополнить нашу программу кодом, обеспечивающим работу вот такой "продвинутой" модели нашего калькулятора:



Программирование работы с каждым из дополнительно размещенных элементов управления уже не должно вызывать затруднений, не считая, быть может, работы со списком, содержащим имена функций. Что ж, запрограммируйте все остальное, а к программированию работы со списком вернитесь после изучения следующей главы, в начале которой эта тема и будет разобрана (а заодно и инструкция **Select Case** *множественного выбора*). Заметим, что элемент "список" дает нам возможность без сложностей наращивать и изменять в нашем калькуляторе набор доступных функций.

И последнее. Действия пользователя, даже неверные, не должны приводить к аварийному завершению работы приложения. То есть, если мы набрали не число, программа должна выявить и обработать эту ситуацию ранее системного обработчика. Универсальным средством является использование инструкции **On Error GoTo**. Но можно еще до возникновения ошибки проверять, является ли вводимое значение числом, с помощью функции **IsNumeric** (Выражение).

Глава 3

РАБОТА С БАЗАМИ ДАННЫХ

И если уж гнаться, то не меньше как за двумя зайцами.

Проект "Словарной обучалки"

В данной главе мы будем создавать обучающую программу для пополнения словарного запаса по иностранному, в данном случае английскому, языку. Программа будет иметь *две* версии: первая проще и компактнее, зато вторая более профессиональная, так как работает с "настоящей" базой данных.

Сценарий работы будущей программы предполагается следующий.

В окне "обучалки" имеется список английских слов, расположенных по алфавиту. Программа последовательно и случайным образом выдает в специальном поле русские слова-переводы соответствующих английских слов из списка и одновременно выводит в другом поле изображения обозначаемых этими словами объектов. Обучаемый должен выбрать из списка соответствующее английское слово. Если он делает это правильно, то выбранное слово "произносится" компьютером, если же неверно, то выдается звуковой сигнал и поперек картинки надпись: "Неверно!" После этого выдается следующее русское слово и картинка и т.д.

Помимо данного "пассивного" режима, в "обучалке" имеется и режим тестирования на оценку, который включается специальной кнопкой. В этом режиме обучаемый выполняет то же задание, но параллельно с отсчетом времени (60 секунд). По истечении времени выдается результат — количество данных ответов, из них число верных и оценка.

Основная проблема, которая встает перед нами, — это организация работы с *разнородными* данными. Действительно, с одной стороны, это массивы *текстовых* данных — наименований предметов, с другой — это графические файлы с *изображениями* этих предметов и звуковые файлы с *произнесениями* названий предметов. При этом надо учесть, что ни названия предметов, ни их количество, ни имена файлов в коде программы ни в каком случае присутствовать не должны; все они могут задаваться в виде свойств только на стадии проектирования. Ну а в идеале сама работающая программа должна предусматривать настройку на те или иные данные. Обычно все подобные проблемы организации работы с данными решаются при помощи специальных *систем управления базой данных* (СУБД).

Немножко о базах данных

СУБД работают с данными, которые организованы в *базу данных* (*Database*).

База данных (БД) — это файл специального формата, содержащий структурированные данные.

Структура БД — это совокупность *таблиц*, каждая из которых описывает, как правило, одну *сущность*. Каждая строка в таблице называется *записью (Record)* (записью о чем? — об *экземпляре* какой-либо сущности).

Сущность — это явление или объект, имеющий свойства, а также признаки, отличающие один экземпляр сущности от другого.

Каждый столбец представляет *поле (Field)*, обозначающее определенный элемент (свойство, характеристику, атрибут сущности) в записи таблицы. Некоторые поля служат для хранения *ключей данных (Data keys)*, которые уникальным образом идентифицируют запись и обеспечивают связи между таблицами. Что это за связи и для чего они нужны? И для чего бывает нужно в одной базе данных иметь несколько таблиц, когда, например, в базе данных интегрированного пакета WORKS, изучение которого широко распространено в школьном курсе, данные расположены только в одной?

Недостатки однотабличного представления становятся видны на тех базах данных, где мы вынуждены заведомо дублировать в записях одну и ту же информацию. В школьном курсе до таких примеров обычно не доходят руки. Скажем, в базе данных "Отдел кадров" единственная таблица, хранящая в виде записей личные данные сотрудников, никакого такого заведомого дублирования не предусматривает. Но представьте себе, что вам нужно вести сквозной учет всех товаров, хранящихся на нескольких складах единой организации. Опять-таки никакого заведомого дублирования не возникло бы, если бы каждый товар хранился только на одном из имеющихся складов; в реальности, разумеется, один и тот же товар может храниться на разных складах. Рассмотрим, например, следующую сводную таблицу (очень условную, разумеется), отражающую текущее состояние "товара на складах" (так мы назовем эту сущность) организации:

N п/п	Наименование товара	Единица измерения	Количество единиц	Учетная цена ед. (руб.)	Номер склада
1	Дискеты 3.5''	Коробка	5	400	3
2	Дискеты 3.5''	Коробка	28	400	5
3	Дискеты 5.25''	Коробка	1	200	1
4	Дискеты 5.25''	Коробка	2	200	2
5	Дискеты 5.25''	Коробка	1	200	3

Видно, что в пяти записях таблицы указаны товары только двух видов; несколько записей, используемых для каждого вида, обусловлены различием места хранения (номера склада). Более половины объема таблицы составляет, таким образом, избыточная, дублированная информация.

На данном примере видно, что база данных на основе единственной сущности (и, соответственно, одной таблицы) может быть крайне неэкономной. Поэтому в "настоящих" базах данных используется *реляционная* (от слова *relate* — относиться) модель хранения, при которой информация разбивается на сущности — таблицы, в которых имеются ключевые используемые для межтабличных связей поля.

Обратимся к приведенному примеру базы данных, отражающей сущность "товар на складах". Как было известно заранее, один и тот же поступивший товар (но пришедший, скажем, в разное время), имея определенные свойства — наименование, учетную цену, количество и т.д., может храниться на разных складах. Поэтому атрибут "номер склада" не является присущим сущности "товар". Естественно разделить сущность "товар на складах" на две: "товар" и "товар на складе". Так как один товар может храниться на разных складах, то между этими двумя сущностями имеет место тип связи ("отношения") "*один ко многим*", или, по-другому, "родитель — потомок". При этом в родительской таблице "товар" каждый товар будет идентифицироваться уникальным значением, а содержащее его поле называться *первичным ключом*. Соответствующее поле в дочерней таблице "товар на складе" может содержать по нескольку одинаковых значений первичного ключа и называется *внешним ключом*. Именно такого типа связи наиболее часты в реальных базах данных.

Нам осталось выяснить, по какому атрибуту (ключевому полю) будет осуществляться связь между этими сущностями. Поле наименования

товара использовать нельзя: могут быть товары с одним наименованием, но разной учетной ценой. Кажется естественным использовать порядковый номер записи о товаре в таблице **"Товар"** (таблица, как мы уже говорили, является формой отображения соответствующей сущности). Порядковый номер записи — это номер очередности ввода записи в таблицу (поэтому его еще называют физическим номером). Для того чтобы в любой момент иметь доступ к этому номеру, используют введение отдельного поля *счетчика*, значение которого для каждой новой записи автоматически увеличивается на 1 (операция увеличения на единицу называется "инкрементом"). Забегая вперед, скажем, что достигается это путем установки соответствующего значения свойства **Attributes** (Атрибуты) объекта **Field** с помощью глобальной константы **dbAutoIncrField**. Недостатком такого ключа будет только отсутствие у него смыслового значения. В реальности же для идентификации экземпляра сущности стараются использовать какие-либо атрибутивные коды:

N п/п	Наименование товара	Единица измерения	Учетная цена ед. (руб.)	Идентификационный номер товара
1	Дискеты 3.5''	Коробка	400	102
2	Дискеты 5.25''	Коробка	200	103

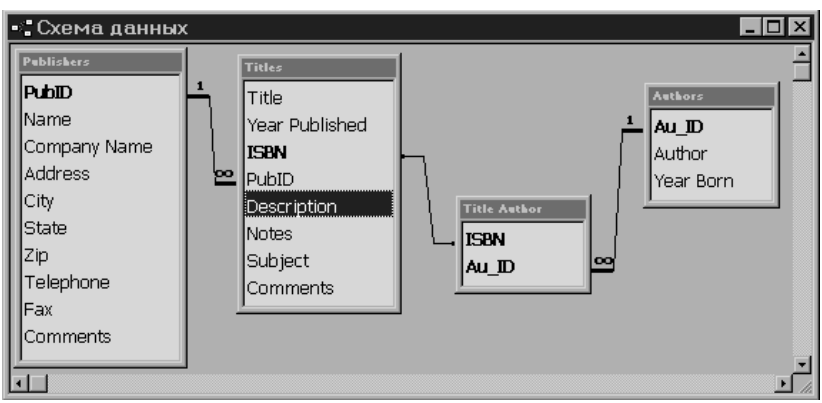
После этого таблица **"Товар на складе"** примет вид:

N п/п	Идентификационный номер товара	Количество	Номер склада
1	103	1	1
2	103	2	2
3	102	5	3
4	103	1	3
5	102	28	5

Отметим, что под номером склада мы также понимаем в общем случае идентификационный номер склада, вводимый в отдельной таблице **"Склады"**. В данном случае номер склада скорее всего является значением поля — счетчика таблицы **"Склады"**.

Проделанная нами процедура реорганизации базы данных, направленная на устранение дублирования, называется *нормализацией*. Много ли мы от нее выгадали? Даже в нашем упрощенном примере выигрыш по объему таблицы составляет более 50%. В реальных задачах информация по товарам включает в себя гораздо большее количество полей-атрибутов: источник партии товара, приходная цена, цена продажи, покупатель партии, дата поступления и дата продажи, дата оформления и номер сопроводительного документа и т.п. При хранении в единой таблице масштаб паразитного расходования объема памяти возрос бы многократно. Кроме того, усложнилась бы организация работы с такой базой данных. Используя же реляционный подход и вводя необходимые связанные по ключевым полям с соответствующими идентификационными номерами таблицы-сущности, такие, как **"Товар отпущенный"**, **"Организация"**, и другие, мы находим возможность реализации гибкой и эффективной системы товарного учета.

В пакете VB5 (папка VB) имеется модельная реляционная база данных **Biblio.mdb**, созданная в формате СУБД Access. Эта база данных обеспечивает работу с данными о книгах, их авторах и издательствах. Файл базы данных хранит несколько таблиц, связанных по ключевым полям. Система Access предоставляет нам возможность просмотра схемы связей этих таблиц. Вот как выглядит окно с этой схемой (имена ключевых полей с идентификационными номерами выделены жирным):



Из схемы видно, что две связи (по полям **PubID** — идентификационный номер публикации и **AuID** — идентификационный номер автора) имеют тип "один ко многим" (когда идентифицирующие значения указанного поля в одной таблице уникальны **(1)**, а в другой нет **(∞)**). Имеется и одна связь (по полю **ISBN** — международному стандартному номеру книги) типа "один к одному" (когда в обоих таблицах идентификационные значения указанного поля уникальны и взаимно однозначны). Эти два типа связи являются преобладающими в правильно организованной базе данных (отношение "многие ко многим" используется только во вспомогательных целях). Подобную схематизацию нужно проводить при проектировании любой базы данных реляционной модели.

Вернемся к нашей задаче.

База данных, которая вполне удовлетворила бы ее условиям, двумерная, поскольку отображает только одну сущность — **"Предмет"** — совокупностью его атрибутов: наименованиями на английском и русском языках, изображением и произношением (на английском). Единая таблица такой базы данных имела бы следующее строение (фрагмент из трех записей):

EnglWord	RusWord	Image	Wave
DOG	Собака	<ссылка на файл>	<ссылка на файл>
DOLPHIN	Дельфин	<ссылка на файл>	<ссылка на файл>
DONKEY	Осел	<ссылка на файл>	<ссылка на файл>

Записи должны отображаться в отсортированном по первому полю виде. Предусмотренное сценарием указание очередного предмета, отображаемого картинкой и наименованием на русском языке, производится путем генерации случайного целого числа, равномерно распределенного по всему диапазону значений поля счетчика (с коррекцией для избежания повторов). На форме располагается окно со списком английских слов-переводов, представляющее собой копию поля **"Слово на английском"** нашей базы данных (будем в дальнейшем использовать для краткости аббревиатуру БД), из которого обучаемый производит выбор соответствующего. При совпадении слова, выбранного обучаемым, со словом, стоящим в поле-оригинале выбранной записи БД, вызывается звуковой объект в данной записи; если слова не совпадают, регистрируется ошибка, после чего весь процесс повторяется.

Осталось дело за малым: научиться создавать БД и работать с ней.

Сразу скажем, что в языке VB5 есть все объекты, необходимые для работы с базой данных, например, в формате Access. Они образуют целое семейство DAO *(Data Access Objects)* — "объектов доступа к данным". Например, в этом семействе есть объект **Database** (База данных); через него можно работать с объектом **Recordset** (Набор записей), через него — с коллекцией объектов **Fields** (Поля). Но мы работу с этими объектами пока отложим: наша двумерная база данных настолько проста, что будет полезно смоделировать ее с помощью неких подручных средств, в качестве которых мы сейчас рассмотрим стандартные встроенные элементы управления, представляющие *списки*: **ListBox** и **ComboBox**.

Элементы управления ListBox и ComboBox

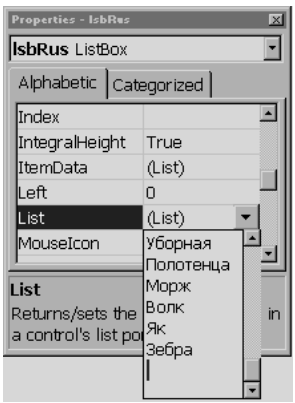
О простом и комбинированном списках мы упоминали как о стандартных элементах управления в среде Windows 95 еще в первой главе. Они призваны обеспечивать пользователю возможность выбора элемента-строки из их вертикально оформленного перечня. Комбинированный список наряду с этой возможностью обеспечивает также возможность задания выбираемого строкового элемента путем набора в поле ввода (кроме варианта, когда его свойство **Style** имеет значение 2).

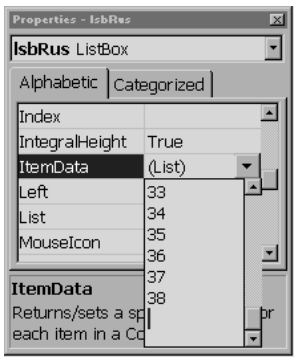
Для организации работы со списком (элемент панели инструментов **ListBox**: ) или с комбинированным списком (элемент панели инструментов **ComboBox**:) нужный элемент устанавливается на форме обычным образом. В момент создания элемента управления сам массив отображаемых им строковых элементов пуст. Этот строковый массив представляется свойством **List**; первый строковый элемент (первый пункт перечня) имеет индекс 0, последний — **ListCount** — 1, где **ListCount** — свойство (доступное только в фазе выполнения), возвращающее (но не устанавливающее!) количество строковых элементов в массиве **List** (пунктов в перечне). Строковые элементы массива мы можем вводить в фазе проекта построчно, открывая массив **List** прямо в окне свойств, как это будет показано ниже.

Очень важно, что при задании (в фазе проекта) значений массива **List** в элементах **ListBox** и **ComboBox** в них синхронно создается еще один массив — числовой (элементов типа **Long**), представляемый свойством **ItemData**. По умолчанию этот массив заполняется нулевыми значениями, однако, так же как и строковые элементы массива **List**, числовые элементы массива **ItemData** могут быть введены в фазе проектирования в окне свойств (в пределах количества элементов массива **List**!). Для чего же их используют?

Как было сказано, массив **ItemData** создается только синхронно с созданием (заполнением строками) массива **List** (в фазе выполнения эта

синхронность сохраняется). Это значит, что между элементами обоих массивов существует взаимно-однозначное соответствие. Приложив столбцы этих массивов друг к другу соответственными элементами, мы получим... таблицу, в одном столбце которой — значения некоторого атрибута (наименования определяемого предмета в данном случае), а в другом — идентификационные номера этих предметов, что же еще! В качестве таких номеров введем просто порядковые номера строк, введенных в массив **List** (иначе — номера пунктов перечня, отображаемого элементом **ListBox** по мере ввода этих строк). И строки, и номера вводятся в фазе проекта построчно с помощью открывающегося комбинированного списка:





Теперь становится ясно, что нашу однотабличную БД можно разбить на 4 таблицы (по числу полей) и связать эти таблицы, размещаемые в элементах-списках, с помощью идентификационного номера каждого определяемого предмета (отношение между таблицами — "один к одному"). При этом список с английскими наименованиями (единственный, который будет определен видимым в фазе выполнения) будет постоянно отсортирован по алфавиту (свойство **Sorted** установлено в **True**). Так как мы не можем хранить в элементе-списке OLE-объекты, то будем хранить соответствующие имена файлов с картинками и звукозаписью. Все эти файлы будем хранить в отдельной папке с предопределенным именем, например, _PIC-WAV. Такая организация потребует от пользователя приложения, перед началом работы с ним, с помощью специального броузера (от слова **Browse** — пролистывать; по-русски кнопку вызова броузера называют обычно **Обзор**) установить эту директорию как рабочую (обычно для броузера предусматривают собственное диалоговое окно, но в нашем случае мы пренебрежем этим обычаем).

В броузерах также используются стандартные элементы-списки, но специализированные: для работы со списками логических дисков (элемент панели инструментов **DriveListBox**: ), с иерархическим списком каталогов текущего диска (элемент панели инструментов **DirListBox**: ) и списком файлов рабочего каталога (текущего каталога на текущем диске) (элемент панели инструментов **FileListBox**: ). Но для функций броузера эти элементы проще программировать (стандартным образом), чем рассказывать об их свойствах. Поэтому перейдем непосредственно к проектированию.

Выбор и размещение элементов управления

Если организация хранения данных и последующей их выборки вроде бы прояснилась, то не совсем ясным остался вопрос о программном отображении графических объектов и проигрывании звуковых.

Для отображения графических объектов используются два стандартных элемента панели инструментов — **Image**:  и **PictureBox**:  .

Элемент **Image** значительно беднее по своим возможностям, но и ресурсов потребляет существенно меньше. Подробно останавливаться на особенностях свойств каждого из этих элементов нам нет смысла. Укажем только существенные для нынешней задачи моменты.

Оба элемента имеют свойство **Picture**, которому в качестве значения устанавливается полное (с именем диска и путем) имя графического файла. При проектировании эта установка производится через вызов диалога **Load Picture** (Загрузка картинки) путем нажатия на кнопку с многоточием в строке свойства **Picture** окна **Properties**: 

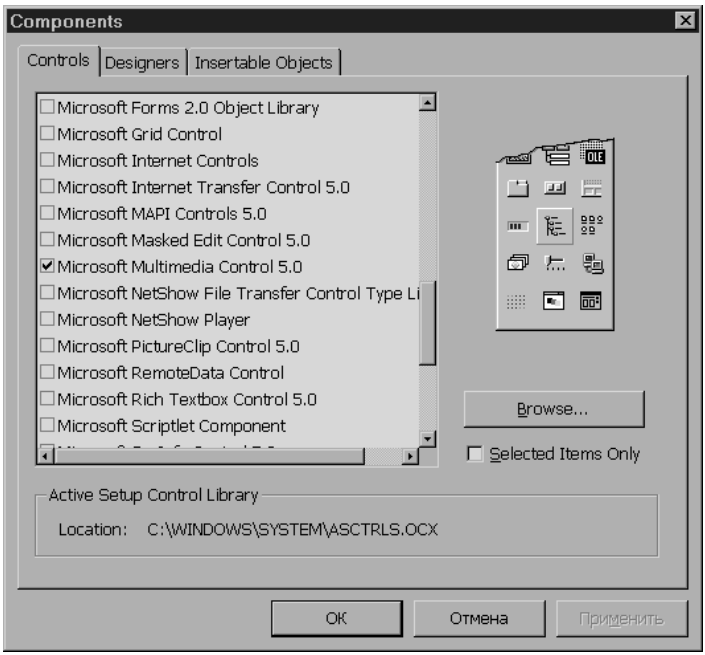
. В фазе выполнения установка файла производится с помощью функции **LoadPicture()**.

По-разному эти элементы работают в случаях, когда картинка по размеру больше рабочей области элемента. Элемент **Image** изменяет свои размеры под размер картинки, если свойство **Stretch** (Растягивание) установлено в **False** (по умолчанию); в противном случае, при установке его в **True**, картинка сжимается (или растягивается) под размер элемента. Элемент **Picture** обрезает не влезающую часть картинки, но может и подстроиться под ее размер при установке свойства **AutoSize** (Авторазмер) в **True** (по умолчанию **False**).

Помимо данных стандартных элементов панели **ToolBox**, нам потребуется — для проигрывания звукозаписи — и один нестандартный,

созданный по технологии ActiveX (компонент ActiveX). Это элемент управления мультимедиа **MMControl**. Компоненты ActiveX перечислены в диалоге **Components** (Компоненты), который вызывается либо, как уже упоминалось, через одноименный пункт контекстного меню, вызванного в панели инструментов, либо командой Ctrl T, дублирующей команду **Components** из пункта **Project** главного меню.

Вызовем диалог **Components** и во вкладке **Controls** установим флажок напротив названия компонента **Microsoft Multumedia Control 5.0**, после чего щелкнем по кнопкам **Применить** и **OK**:



После этого на панели инструментов появится пиктограмма элемента **MMControl**: . Наконец, остался не рассмотренным еще один очень важный стандартный элемент — таймер (**Timer**:). Этот невидимый в фазе выполнения элемент предназначен для одной-единственной цели — генерировать событие **Timer** через промежуток времени, заданный в миллисекундах значением свойства **Interval** (Интервал). Ну

а в обработчике этого события мы задаем с помощью программного кода те действия, которые нам необходимо выполнять в момент возникновения события. Так как тип значения **Interval** целый, то максимальный интервал, который можно задать для таймера, — чуть более минуты. Для задания больших интервалов между периодическими выполнениями требуемого программного кода мы можем ввести переменную-счетчик (статическую), накапливающую число обращений к обработчику, и при каждом обращении проверять на равенство нулю (в инструкции ветвления **If**) остаток от деления значения счетчика на целое число, задающее коэффициент удлинения интервала срабатывания таймера по сравнению со значением свойства **Interval**. Требуемый программный код должен содержаться в той ветви инструкции **If**, которая выполняется при истинности условия.

Наиболее часто таймер используется для целей анимации. Но он может использоваться и для задания времени работы каких-либо процедур, и для временных задержек каких-либо действий программы. Для этих целей у таймера имеется свойство, аналогичное пусковому устройству секундомера: таймер ведет отсчет времени только тогда, когда его свойство **Enabled** (Возможный) установлено в **True** (по умолчанию), а значение **Interval** не равно **0**. Таким образом, программно изменяя какое-либо из этих свойств, мы имеем возможность запускать таймер (например, устанавливая **Enabled=True**) и снова его останавливать через заданный для него интервал (помещая в обработчик установку **Enabled=False**).

Займемся размещением на форме ТЕСТ нашего приложения элементов управления, как это показано на *рис. 3.1*. Наиболее важным из них дадим "правильные" (по рекомендуемому фирмой Microsoft соглашению) имена, другим для лучшего различения "правильных" имен оставим "естественные".

Охарактеризуем устанавливаемые элементы.

- Элемент мультимедиа **MMControl1** не должен быть виден в фазе выполнения (**Visible=False**). Его установки по умолчанию рассчитаны на работу с файлами звукозаписи с расширением WAV (**DeviceType="WaveAudio"**). Так как элемент **MMControl** не встроенный, а создан по объектной технологии, являясь, таким образом, "пользовательским" элементом (**UserControl**), то команды управления им задаются (в фазе выполнения) как значения его свойства **Command**. Например, инструкция **MMControl1.Command = "Open"** открывает доступ к мультимедийному устройству, опреде-

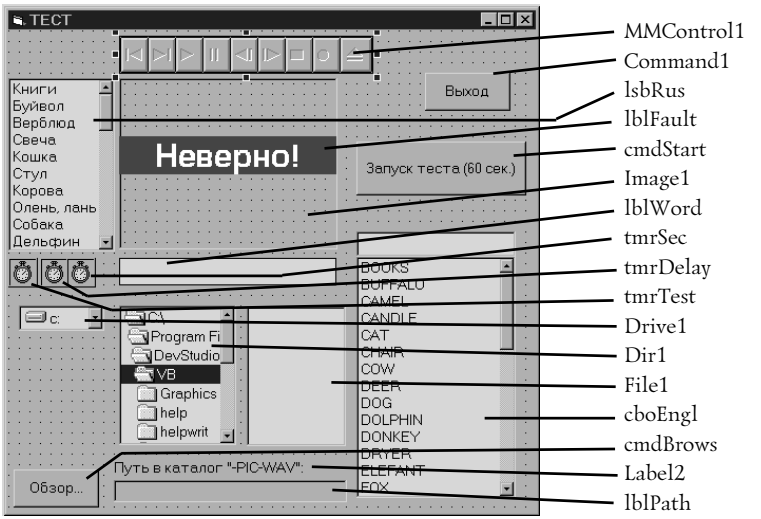


Рис. 3.1. Форма проекта с размещенными элементами управления

ленному свойством **DeviceType** ("устройство" — это совокупность аппаратного и программного обеспечения, например, звуковая карта и те или иные драйверы). Помимо "**WaveAudio**", имеется еще 11 типов устройств, с которыми работает данный элемент, — сканер, проигрыватель компакт-дисков, видеодисков, MIDI-файлов, цифрового и обычного видео и т.д.

- Кнопка (элемент **Command**) с именем **Command1** предназначена для выхода из приложения (**Caption = "Выход"**).
- Список (элемент **ListBox**) с именем **lsbRus** предназначен для ввода (в фазе проектирования) и хранения русских слов (свойство **List**) с их идентификационными номерами (свойство **ItemData**). В фазе выполнения предполагается невидимым.
- Ярлык (элемент **Label**) с именем **lblFault** предназначен для вывода надписи (**Caption = "Неверно!"**) при неверном выборе английского слова-перевода. Размер шрифта (свойство **Font**) установлен 18 пунктов, начертание жирное; изменены цвет шрифта **(ForeColor = &H8000000E&)** и цвет фона **(BackColor= &H80000017&)**.
- Кнопка с именем **cmdStart** служит для запуска минутного теста **(Caption="Запуск теста 60 с")**. После запуска теста в значении свойства **Caption** отображается число секунд, прошедших от начала тестирования.

- Графическое окно (элемент **Image**) с именем **Image1** предназначено для вывода сопутствующей русскому слову картинки.
- Ярлык с именем **lblWord** предназначен для отображения выбранного из списка **lsbRus** случайным образом русского слова. Размер шрифта установлен 12 пунктов.
- Таймер с именем **tmrSec** предназначен для отсчета секунд, отображаемых в надписи (свойство **Caption**) на кнопке **cmdStart** после запуска теста.
- Таймер с именем **tmrDelay** генерирует задержку времени между указанием английского слова-перевода и выводом наименования и изображения следующего случайно выбранного предмета. Использование этого таймера не является обязательным.
- Таймер с именем **tmrTest** задает время тестирования, по истечении которого обработчик события **Timer** подытоживает и выводит результаты тестирования.
- Списки с именами **Drive1, Dir1** и **File1** используются в "броузере файлов" приложения для установки рабочей директории с графическими и звуковыми файлами, соответствующими наименованиям предметов. В элементе — списке файлов **File1** предусмотрена фильтрация для отображения имен только графических файлов формата **JPEG: Pattern="*.JPG"** (так как именно в этом формате мы будем хранить наши графические файлы).
- Комбинированный список с именем **cboEngl** предназначен для ввода (в фазе проекта) и работы с английскими словами-переводами (свойство **List**) и их идентификационными номерами (свойство **ItemData**) — теми же, что и у соответствующих русских слов в элементе **cboRus**. Этот список должен быть видим в фазе выполнения и отсортирован в лексикографическом ("по алфавиту") порядке (**Sorted=True**). Почему мы используем не обычный (**ListBox**), а комбинированный список? В будущем, особенно в том случае, если количество слов будет достаточно велико, предполагается развить наше приложение так, что искомое слово-перевод можно будет вводить в поле ввода этого списка с клавиатуры, причем список будет программно "подтягиваться" к правильно набираемому слову. Приложение сможет служить, таким образом, и клавиатурным тренажером.

- Кнопка с именем **cmdBrows** служит для активизации работы с броузером и окончания работы с ним. При активизации работы первоначальная надпись на кнопке **(Caption="Обзор...")** сменяется надписью "**OK**", а затем, по окончании работы, восстанавливается.
- Ярлык с именем **Label2** содержит надпись, подсказывающую пользователю броузера, какой именно путь (**Caption = "Путь в каталог _PIC-WAV"**) должен быть выведен в надписи нижележащего ярлыка с именем **lblPath**.

Наверное, у вас уже возник вопрос: почему мы не разместили, как планировали, списки с именами графических и звуковых файлов (и соответствующими идентификационными номерами)? Да просто нашелся способ еще более упростить наш проект. Чтобы не заниматься вводом в списки имен файлов, просто заранее дадим графическим и звуковым файлам имена, представляющие (без учета расширения) соответствующие идентификационные номера (точнее — их символьные десятичные записи) из списков русских и английских наименований соответствующих предметов. Тогда по идентификационному номеру выбранного слова мы сможем легко строить составные имена соответствующих графического и звукового файлов (они будут различаться расширениями).

Строковое представление номера (да и вообще числа) дает функция **Str(Номер)**. В случае положительного числа, однако, эта функция в начале строки помещает пробел. Чтобы он не мешал, мы делаем "вырезку" остатка данной строки начиная с ее второго символа. Такую вырезку возвращает функция **Mid(2, Строка)** (когда нужна вырезка не до конца строки, а определенной длины, эту длину задают третьим параметром данной функции, в нашем случае опущенным). Таким образом, имя файла возвращает композиция этих функций **Mid(2, Str(Номер))**.

МИНИСТЕРСТВО ОБЩЕГО И ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ РФ
ФОНД НОВЫХ ТЕХНОЛОГИЙ В ОБРАЗОВАНИИ "БАЙТИК"
ДЕПАРТАМЕНТ ПО ОБРАЗОВАНИЮ МОСКОВСКОЙ ОБЛАСТИ
ЦЕНТР НОВЫХ ПЕДАГОГИЧЕСКИХ ТЕХНОЛОГИЙ
ИНСТИТУТ ЮНЕСКО ПО ИНФОРМАЦИОННЫМ ТЕХНОЛОГИЯМ В ОБРАЗОВАНИИ
COMPUTER USING EDUCATORS INC, USA

X МЕЖДУНАРОДНАЯ КОНФЕРЕНЦИЯ
"ПРИМЕНЕНИЕ НОВЫХ ТЕХНОЛОГИЙ В ОБРАЗОВАНИИ"

Троицк, Московская область
30 июня — 3 июля 1999 года

Приглашаем вас принять участие в юбилейной X международной конференции "Использование новых технологий в образовании" в рамках the 10th International Technology Institute, которая пройдет в период с 30 июня по 3 июля 1999 года на базе фонда новых технологий в образовании "Байтик". В конференции примут участие американские представители организации Computer Using Educators (CUE) и специалисты других стран.

Официальные языки — русский и английский. Конференция посвящена вопросам практического использования новых технологий в образовании и будет проводиться по следующим направлениям:

1. Новые технологии для детей дошкольного и младшего школьного возраста.
2. Преподавание школьных дисциплин: информатика, естественные предметы, гуманитарные предметы, экономика и иностранные языки.
3. Новые технологии в учебном процессе, дистанционное обучение и Интернет.
4. Информационные технологии в профессиональной подготовке.
5. Учебники и учебные пособия.
6. Олимпиады и конкурсы по информатике.
7. Методики контроля знаний обучаемых.
8. Подготовка специалистов в области информационных технологий.
9. Компьютер для администрации в образовательном учреждении.

В программу конференции будут включены:

• "Круглые столы" для обсуждения проблемных вопросов компьютерного образования.

• Выставка-ярмарка программных и технических средств, учебников, а также методических разработок российских и зарубежных фирм.

Полную информацию о программе конференции, порядке оформления материалов, оплате оргвзноса вы можете получить в оргкомитете.

Адрес:
142092, Московская обл., г. Троицк, Сиреневый б-р, д. 11.
Фонд "Байтик"
Тел./факс (095)334-03-67

E-mail:
cue@bytic.troitsk.ru
cue@relay.bytic.troitsk.ru

Web:
http://www.bytic.troitsk.ru/russian/cue_reg99.html
http://stuff.bytic.troitsk.ru/cue/cue_reg99.html

Выставка “Информационные технологии в образовании” в ВВЦ

С 28 мая по 2 июня в ВВЦ прошла выставка “Информационные технологии в образовании”, в которой приняло участие “Первое сентября”, представленное приложениями “Информатика”, “Управление школой” и сервером **www.1september.ru**. Выставка оставила крайне противоречивое впечатление. С одной стороны, организаторы создали для работы участников неплохие условия, например, обеспечили подключение к Интернету, благодаря чему мы получили возможность продемонстрировать наш сервер в полном объеме (на выставке “Школа-99”, о которой мы писали в № 12/99, такой возможности не было), с другой — выставка, по-видимому, побила все рекорды непосещаемости. В причинах этого мы разбираться не станем — думается, организаторам они ясны и без наших советов.

Отдельно надо сказать о названии выставки. Все привыкли к тому, что конференция-выставка с таким названием проходит осенью в московском лицее № 1511 при МИФИ (напомним, что очередная пройдет с 9 по 12 ноября). Данная выставка не имела к осенней никакого отношения, что для некоторых посетителей было не вполне очевидно. Будем надеяться, что чувство неудовлетворенности, которое осталось у них от посещения этой выставки, не помешает им посетить осеннюю, на которой “Информатика” также будет представлена. Будем рады видеть вас на нашем стенде, приходите!

Самое популярное профессиональное издание
для учителей информатики — газета

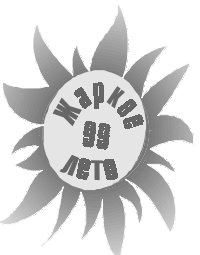
ИНФОРМАТИК

пятый год издания . 4 номера в месяц . 48 номеров в год . индекс подписки 32291

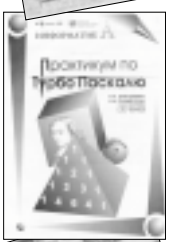
Дорогие читатели!

“Информатика” выходит с января 1995 года. Цель и назначение газеты — быть надежной методической опорой любому учителю информатики.

Преподаватели с многолетним стажем и начинающие, обладатели современного компьютерного класса и те, кто учит детей, довольствуясь самым скромным оборудованием, те, кто ведет профильные курсы, и те, кто работает по минимальному учебному плану в самой обычной школе, находили и обязательно найдут на наших страницах материал, для них предназначенный. В течение прошедших четырех лет сформировались основные направления и рубрики. На страницах газеты — “Задачи”, “Экзамены”, “Олимпиады”, “Языки программирования”, “Новые информационные технологии”, “Как это делаю я”, “Учебники” (новые учебники!), “Документы” (официальные документы, их квалифицированное толкование, ответы на вопросы), “Материалы к уроку”, “Круглый стол”. Мы с трудом умещаемся в заданный газетный объем и стараемся, чтобы каждая публикация была существенной помощью учителю при подготовке к уроку.



Серия из 12 спецвыпусков “Жаркое лето 99-го” — бесценный подарок учителю к новому учебному году.



Москва, 121165, ул. Киевская, 24. Тел. 249-48-96. E-mail: inf@1september.ru http://www.1september.ru

Гл. редактор
С.Л.Островский

Зам. гл. редактора
Е.Б.Докшицкая

Дизайн и компьютерная верстка:
Н.И.Пронская

Корректоры:
Е.Л. Володина,
С.М.Подберезина

Регистрационный номер 012868

Отпечатано в типографии ОАО ПО "Пресса-1", 125865, ГСП, Москва, ул. Правды, 24
Тираж 7000 экз.
Заказ №

ИНДЕКС ПОДПИСКИ
для индивидуальных подписчиков — 32291
для организаций и предприятий — 32591
комплекта приложений — 32744

121165, Москва, Киевская, 24
Тел. 249 4896
Отдел рекламы
Тел. 249 9870

Тел. (095)249 3138, факс (095)249 3184

internet:inf@1september.ru
WWW:http://www.1september.ru

05.06