

Глава 1

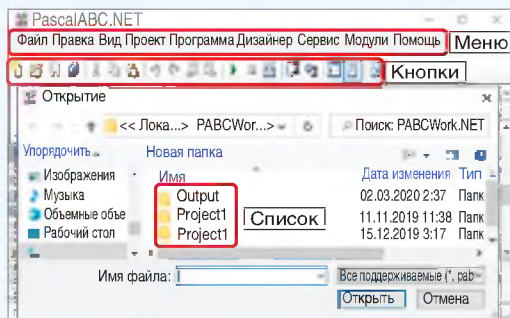
ВВЕДЕНИЕ В ОБЪЕКТНО-СОБЫТИЙНОЕ ПРОГРАММИРОВАНИЕ

§ 1. Объектно-событийная модель работы программы



Основателем RAD считается сотрудник IBM, британский консультант по информационным технологиям Джеймс Мартин (1933—2013), который в начале 1990-х гг. сформулировал основные принципы RAD, основываясь на идеях Барри Бойма и Скотта Шульца.

Пример 1.1. После загрузки какого-либо редактора пользователь может открыть файл для редактирования. При этом он выбирает меню **Файл**, находит в списке команду **Открыть**, выбирает нужный файл, нажимает кнопку **Открыть**. Как мы видим, чтобы открыть файл, пользователь взаимодействует с такими элементами управления, как меню, список, кнопка.



1.1. Элементы управления в приложениях с графическим интерфейсом

Современные программы, с которыми сегодня работают пользователи компьютера, отличаются от тех, которые вы создавали раньше. Основное отличие — взаимодействие пользователя с программой.

Программы, которые вы создавали в 7—10-м классах, взаимодействовали с пользователем посредством текстового интерфейса (часто его называют интерфейсом командной строки). После запуска программы вы вводили данные, программа выполнялась, и вы видели результат. И ввод, и вывод данных осуществлялся в алфавитно-цифровой форме.

Операционные системы с графическим оконным интерфейсом (например, Windows) предполагают общение пользователя с программой посредством элементов управления. К элементам управления относят: кнопки, разнообразные меню, текстовые сообщения, списки и др. При работе программы пользователь выбирает какой-либо элемент управления и совершает с ним определенное действие (пример 1.1). Если такое действие для выбранного элемента было определено, то программа его выполняет, иначе выдает сообщение об ошибке.

Многие системы программирования позволяют создавать программы с оконным интерфейсом. Такие программы называют **оконными приложениями** (Windows Form Application).

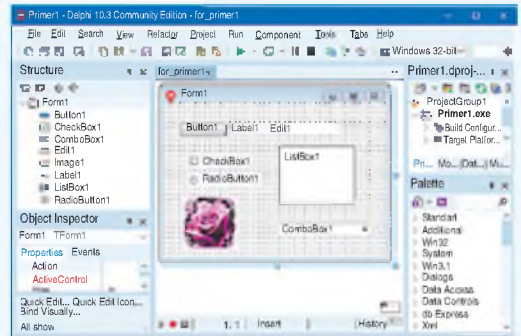
Проектирование интерфейса окна программы можно выполнять с использованием RAD-технологии (**Rapid Application Development — быстрая разработка приложений**). Технология RAD характерна для многих систем программирования. Быстрая разработка стала возможной за счет того, что элементы управления были визуализированы и собраны в специальные библиотеки — VCL (Visual Component Library — визуальная библиотека компонентов).

Различные элементы управления можно перетаскивать с палитры компонентов на форму с помощью мыши. Процесс создания интерфейса будущей программы представляется аналогом работы с неким конструктором. Программирование в RAD-средах является визуальным, поскольку код по созданию объекта не пишется, а генерируется средой. Задача программиста — написание кода по управлению готовыми компонентами.

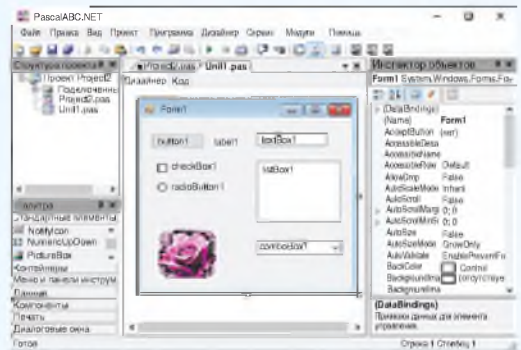
Визуальное программирование поддерживается в PascalABC и Delphi (код пишется на языке Pascal), VisualBasic, C# и др. (пример 1.2). Для обучения учащихся младших классов используется визуальное программирование в среде Скретч (Scratch).

Пример 1.2. Среда программирования, в которых реализована поддержка парадигмы визуального программирования.

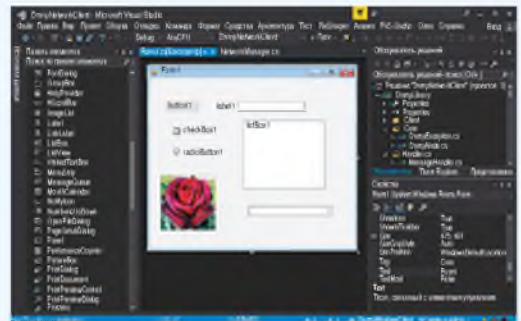
Delphi:



PascalABC:



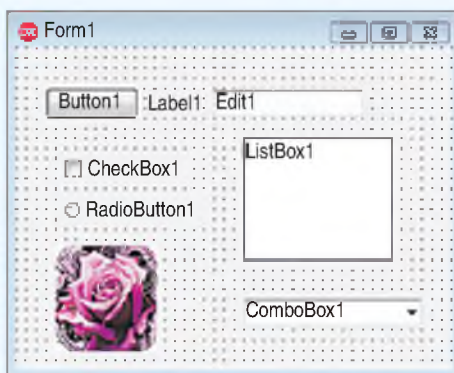
Visual Studio для языка C#:



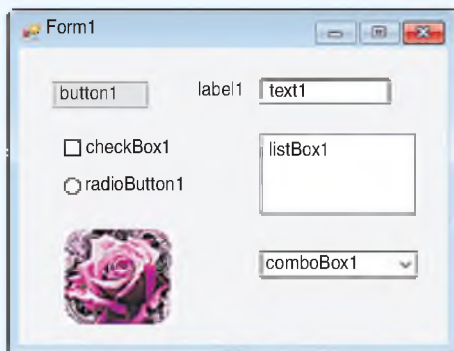
Пример 1.3. Основные элементы интерфейса:

Элемент управления	Имя
Кнопка	Button
Надпись	Label
Поле для ввода текста	TextBox (Edit)
Флажок	CheckBox
Радиокнопка	RadioButton
Список	ListBox
Выпадающий список	ComboBox
Рисунок	PictureBox (Image)

Элементы управления на форме в среде программирования Delphi:



Элементы управления на форме в среде программирования PascalABC:



Многие элементы управления в разных средах имеют одинаковые или синонимичные имена (пример 1.3).

Создаются оконные приложения как проект и состоят из нескольких файлов. Внешний вид окна будущего приложения строится на форме. Для формы сохраняются два файла — один содержит описание внешнего вида формы, другой — описание действий при выборе пользователем того или иного элемента управления. Главный файл проекта содержит описание его структуры, а также команды по созданию формы и запуску приложения.

Все элементы, размещенные на форме, и сама форма образуют систему взаимодействующих объектов. Способ их взаимодействия основан на объектно-ориентированном программировании.

Объектно-ориентированное программирование (ООП) — технология создания программ, основанная на использовании системы объектов. Каждый объект обладает набором **свойств**, которые описывают его состояние, и **методов**, характеризующих его поведение.

Объект — совокупность данных и методов работы с ними.

Организация данных внутри объекта скрыта от пользователя. Данные и способы их чтения и записи являются свойствами объекта, их можно изменять. Методы — процедуры и функции для обработки данных.

1.2. События

Организация взаимодействия между программой и пользователем управляется **событиями**: пользователь может нажать на клавишу мыши или клавиатуры, ввести текст и др.

Метод программирования, основанный на управлении событиями, называют **событийно-ориентированным программированием**.

Каждое событие связано с каким-либо объектом, которому передается управление в тот момент времени, когда происходит событие. Среди основных событий можно выделить три категории: события мыши, события клавиатуры и системные события (примеры 1.4—1.6).

Процедура (или функция), инициируемая событием, называется **обработчиком события**.

Запущенный на выполнение проект находится в ждущем режиме, реагируя на события, учтенные при его создании, вызываемые действиями пользователя или возникающими в самой программе.

Объектно-событийная модель программы предполагает следующее:

- создание объектов с присущими им свойствами и методами;
- описание событий, при которых объект может выполнять алгоритм обработки данных.

Пример 1.4. События мыши возникают в том случае, если пользователь производит какие-либо действия с мышью:

Click	Нажатие левой кнопки мыши
DblClick	Двойной щелчок левой кнопкой мыши
MouseDown	Нажатие на любую кнопку мыши. Параметры обработчика события позволяют определить, какая из кнопок была нажата и в какой точке
MouseUp	Освобождение кнопки мыши, которая была нажата
MouseMove	Перемещение указателя мыши

Пример 1.5. События клавиатуры происходят при нажатии клавиш на клавиатуре:

KeyPress	Нажатие клавиши с текстовым символом
KeyDown	Нажатие любой клавиши
KeyUp	Освобождение клавиши

Пример 1.6. Системные события управляются функциями операционной системы:

Paint	Возникает, когда элемент необходимо перерисовать
Resize	Происходит, когда размеры элемента изменяются
Enter	Возникает, когда элемент управления становится активным
Leave	Происходит, когда элемент управления перестает быть активным



1. Какие программы называют оконными приложениями?
2. Что понимают под событийным программированием?
3. Какие типы событий вы можете назвать?



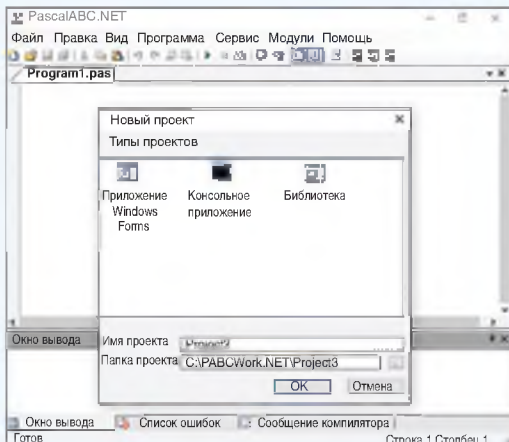
§ 2. Визуальная среда разработки программ

Работа по созданию оконных приложений рассматривается в среде программирования PascalABC.Net.

Пример 2.1. Файлы проекта:

Имя	Тип	Размер
Project1.pabcproj	PascalABC.NET Pr...	2 КБ
Project1.pas	Файл "PAS"	1 КБ
Unit1.fmabc	Файл "FMABC"	15 КБ
Unit1.Form1.inc	Free Pascal includ...	1 КБ
Unit1.pas	Файл "PAS"	1 КБ

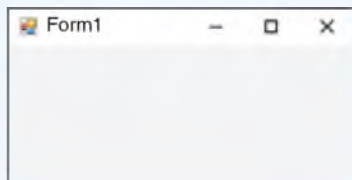
Пример 2.2. Создание проекта в PascalABC.Net:



Пример 2.3. Создание и выполнение проекта:

1. Создать папку с именем Primer1.
2. Создать проект.
3. Запустить проект на выполнение.

Вид окна приложения:



Для сохранения изменений используют команду Сохранить все (кнопка).

2.1. Структура проекта

При создании оконного приложения работают с проектом, состоящим из нескольких файлов. В разных средах программирования проект может состоять из различного количества файлов. Обязательными файлами являются следующие:

- файл формы (1), содержащий описание внешнего вида окна приложения;
- файл программного модуля (2), содержащий описание функций-обработчиков для объектов на форме;
- файл проекта (3), позволяющий связать структурные элементы проекта между собой.

(Рассмотрите пример 2.1.)

Файлы одного проекта обычно хранятся внутри отдельной папки. При компиляции приложения создается файл с расширением `.exe` и именем, совпадающим с именем проекта. Этот файл запускает работающее приложение без загрузки среды программирования. (Как скомпилировать приложение, чтобы файл с расширением `.exe` не удалялся после закрытия окна, см. в *Приложении*, с. 102).

Для создания проекта в среде PascalABC.Net нужно выполнить команды **Файл • Новый проект • Приложение Windows Form** (пример 2.2).

При создании проекта файлы сохраняются автоматически (пример 2.3).

2.2. Интерфейс среды программирования

Полное окно среды программирования PascalABC.Net при создании приложений Windows Form можно посмотреть в *Приложении* (с. 102).

Рассмотрим основные элементы.

Основное меню и панель быстрого доступа (пример 2.4) содержат команды для управления проектом: сохранение, загрузка, выполнение и др.

Форма (пример 2.5) служит для визуального отображения окна приложения. Во время проектирования приложения на форме отображается точечная сетка, позволяющая выравнивать помещаемые на форму компоненты.

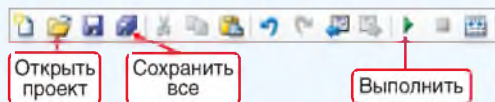
Инспектор объектов (пример 2.6) отображает свойства (или события) выбранного объекта.

В левом столбце вкладки **Свойства** перечислены все свойства объекта, которыми пользователь может управлять при проектировании приложения. В правом столбце указаны значения свойств, которые могут выбираться из списка или вводиться с клавиатуры.

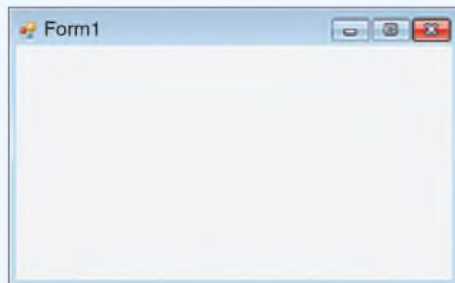
Вкладка **События** содержит список событий для объекта. Для каждого события может быть определен свой обработчик. Если обработчик для события определен, напротив события будет прописано имя процедуры (функции) обработчика.

В нижней части инспектора объектов размещено описание выбранного свойства или обработчика событий.

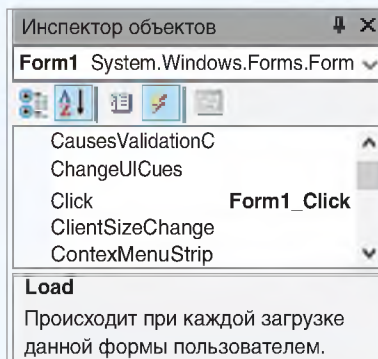
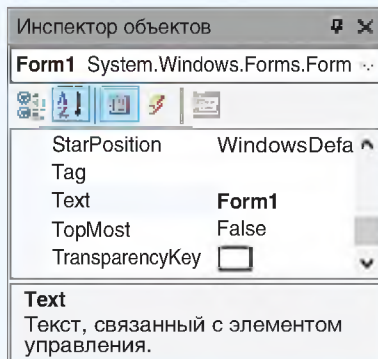
Пример 2.4. Меню и панель быстрого доступа:



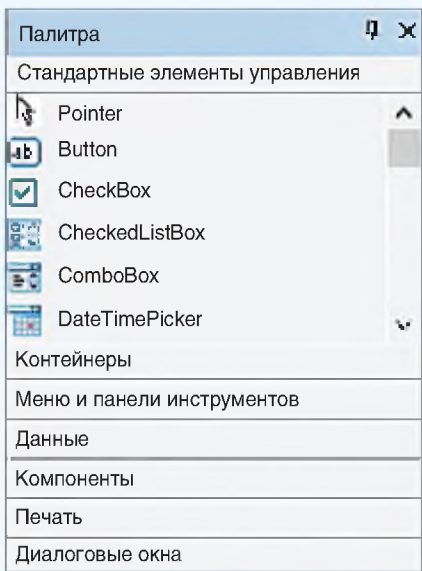
Пример 2.5. Форма:



Пример 2.6. Инспектор объектов. Отображаются свойства формы:



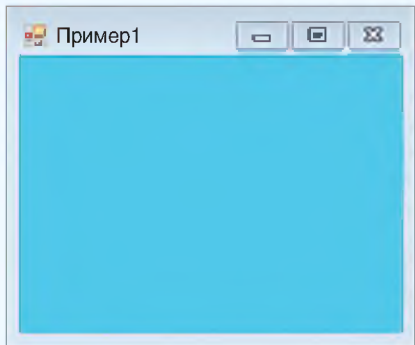
Пример 2.7. Палитра компонентов:



Пример 2.8. Изменение свойств формы в инспекторе объектов:

Свойство	Значение
Text	Пример 1
BackColor	clAqua (выбрать из списка на вкладке Интернет)
Size	250; 250
Location	200; 200

После изменения значений свойств в инспекторе объектов изменится внешний вид формы:



Палитра компонентов (пример 2.7) содержит список визуальных компонентов, объединенных в группы. Раскрытие группы происходит по щелчку с названием группы.

2.3. Работа с формой

Форма является объектом и служит для визуального отображения окна приложения. Как любой объект, форма обладает свойствами (пример 2.8).

Свойство	Назначение
Text	Заголовок формы отображается в строке заголовка окна при запуске приложения. По умолчанию — Form1
BackColor	Цвет формы. Может быть выбран один из стандартных (перечислены в списке) или задан вручную тремя числами, соответствующими RGB
Size	Высота и ширина формы. Можно указать два числа через «;» или развернуть свойство, нажав значок  , и получить возможность ввода значений Width и Height
Location	Горизонтальная и вертикальная координаты положения верхнего левого угла окна формы на экране. Можно указать два числа через «;» или развернуть свойство, нажав значок  , и получить возможность ввода значений X и Y
(Name)	Имя (внутреннее) формы. Используется в программном коде для обращения к объекту. Является идентификатором

Для создания обработчика событий формы нужно в инспекторе объектов перейти на вкладку **События** (⚡), выбрать событие. Процедура генерируется автоматически при двойном клике мышью в пустой строке напротив выбранного события. После этого среда переключается на страницу, на которой пишется код (пример 2.9).

Имя процедуры-обработчика состоит из названия компонента, над которым происходит событие, и названия события (`Form1_Click`).

Для каждого объекта определен обработчик по умолчанию, который создается при двойном клике по объекту. Для формы таким обработчиком будет `Form1_Load` — событие, которое происходит при загрузке формы.

Для переключения между окном программного кода и конструктором дизайна формы можно использовать вкладки **Дизайнер** и **Код** в верхней части окна приложения: Дизайнер Код.

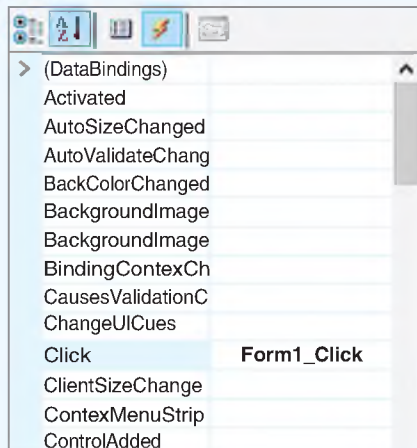
При создании процедур-обработчиков свойства объектов можно изменять программно. Для этого нужно обратиться к свойству по его имени и присвоить новое значение. Например, для изменения цвета формы нужно записать следующую команду:

```
BackColor := Color.Red;
```

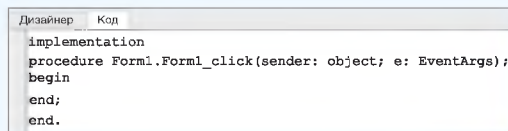
Система Pascal позволяет упростить ввод сложных имен в код программы. После того как вы наберете часть сложного имени, на экране появится список со всеми свойствами и методами, которые относятся к этому объекту (пример 2.10).

Пример 2.9. Создание обработчика события Click (клик левой клавишей мыши) для формы.

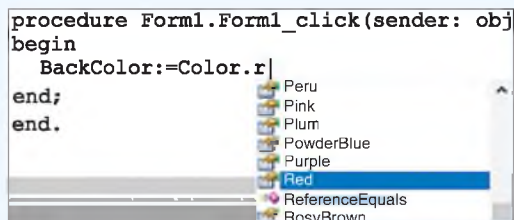
1. Выбор события в инспекторе объектов:



2. Окно программного кода со вставленным обработчиком:



Пример 2.10. Подсказка системы при вводе свойств объекта:



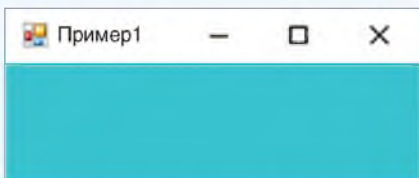
Если вы введете первые буквы названия свойства (метода), то курсор переместится в списке к тем свойствам и методам, названия которых начинаются на эти буквы. После этого нужное свойство можно вставить в программу щелчком мыши или нажатием клавиши Enter. Если список не появился, его можно вызвать комбинацией клавиш Ctrl + пробел.

Пример 2.11. Код процедуры-обработчика:

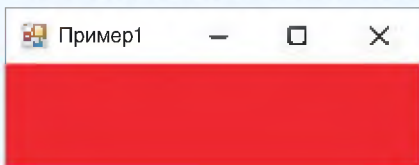
```
procedure Form1.Form1_Click
(sender: Object; e: EventArgs);
begin
  BackColor := Color.Red;
end;
```

Чтобы убедиться в правильности работы программы, нужно запустить проект и проверить, что при выполнении щелчка мышью по форме ее цвет меняется на красный.

Форма после запуска проекта:



Форма после щелчка мыши по ней:



Пример 2.11. Создать обработчик события для щелчка левой клавишей мыши по форме, в результате которого цвет формы должен поменяться на красный (продолжить работу с примером 2.8).

Этапы выполнения задания

1. Перейти на вкладку **Events** в окне инспектора объектов.
2. Выполнить двойной щелчок в поле напротив события **OnClick**.
3. В окне редактора кода в процедуре

```
Form1.Form1_Click(sender: Object;
e: EventArgs);
вписать команду
BackColor := Color.Red;
```

Все изменения свойств формы, которые производили в примере 2.8, можно описать программно. Для этого создается обработчик события `Form1_Load`.

4. Сохранить изменения в проекте.

- ?**
1. Какие элементы среды PascalABC отображаются на экране после создания проекта?
 2. Какие файлы входят в состав приложения, создаваемого в PascalABC?
 3. Для чего предназначена форма?
 4. Для чего используют инспектор объектов?
 5. Какие свойства форм вы можете назвать?
 6. Как создать обработчик события?



Упражнения

1. Внесите изменения в проект из примера 2.11 так, чтобы цвет формы менялся случайно. Изменять цвет можно с помощью функции `FromArgb`. У этой функции четыре параметра: прозрачность (альфа-канал, интенсивность красного цвета, интенсивность зеленого цвета, интенсивность синего цвета). Генерация случайных чисел происходит следующим образом. Сначала создается переменная, являющаяся объектом класса `Random` (команда `var rnd: Random := new Random();`). Каждое новое случай-

ное число можно получить, обращаясь к методу `next(x)`, где `x` задает полуинтервал $[0, x)$. Команда смены цвета будет выглядеть следующим образом:

```
BackColor :• Color.FromArgb(255, rnd.next(256), rnd.next(256), rnd.next(256));
```

2 Создайте проект, в котором при двойном клике мыши по форме ее размеры будут увеличиваться на 5.

1. Создайте и сохраните в новой папке проект.
2. Измените свойство `Text` формы на **Упражнение 2**.
3. Создайте обработчик события мыши `DbClick`.
4. Для изменения ширины и высоты формы можно воспользоваться командами:

```
Width :• Width + 5;
Height :• Height + 5;
```

5. Сохраните изменения в проекте.
6. Запустите проект и проверьте его работу.

3 Создайте проект, в котором цвет формы будет меняться при наведении на нее мыши, например с желтого на зеленый.

1. Измените свойство `Text` у формы на **Упражнение 3**.
2. Установите желтый цвет формы.
3. Создайте обработчики для двух событий мыши: `MouseEnter` и `MouseLeave`.
4. В коде события `MouseEnter` установите зеленый (`Green`) цвет формы, а коде события `MouseLeave` — желтый (`Yellow`).
5. Сохраните изменения в проекте.
6. Запустите проект и проверьте его работу.

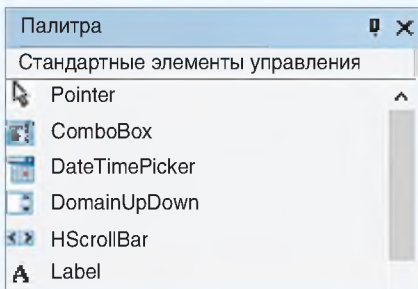
§ 3. Проектирование интерфейса оконного приложения с использованием элементов управления

3.1. Основные элементы управления

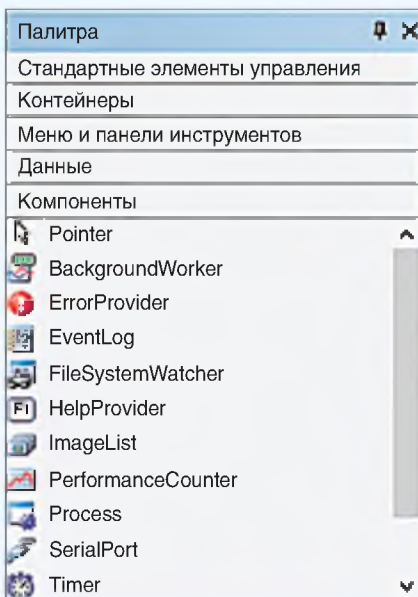
Элементами управления называются объекты, которые используются для отображения данных или организации взаимодействия между пользователем и приложением с помощью мыши или клавиатуры. Они собраны в специальные библиотеки компонентов, которые ОС использует для обеспечения единообразного интерфейса

Для элементов управления используется и другое название — виджеты. Слово употребляется примерно с 1920-х гг. в американском английском для обозначения простой, но необходимой вещи, маленького изделия. Одним из вариантов происхождения этого слова считается словослияние «window gadget» (букв. «оконное приспособление»), также произошедшее в начале XX в.

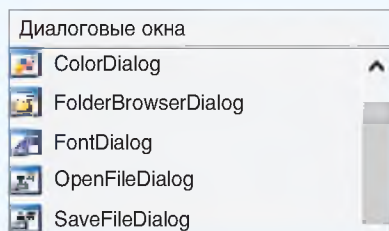
Пример 3.1. Палитра Стандартные элементы управления:



Пример 3.2. Палитра Компоненты:



Пример 3.3. Палитра Диалоговые окна:



прикладных программ. Наиболее распространенными элементами управления являются: кнопки, редактируемые поля, списки выбора, флажки, переключатели и т. д.

Компоненты библиотеки размещаются на различных страницах палитры компонентов. Каждая страница имеет свое название. На странице **Стандартные элементы управления** (пример 3.1) размещены наиболее употребляемые компоненты:

- Кнопка (Button)
- Надпись (Label)
- Поле для ввода текста (TextBox)
- Флажок (CheckBox)
- Радиокнопка (RadioButton)
- Список (ListBox)
- Выпадающий список (ComboBox)
- Рисунок (PictureBox)

Внешний вид этих компонентов на форме был представлен в примере 1.4.

Другие страницы палитры компонентов показаны в примерах 3.2 и 3.3.


Одним из наиболее используемых компонентов палитры **Компоненты** является компонент Таймер (Timer).

Палитра **Меню и панели инструментов** содержит компоненты, необходимые для создания главного меню программы или контекстных меню для различных объектов, помещенных на форму.

Палитры **Печать** и **Диалоговые окна** содержат компоненты, обеспечивающие стандартные диалоги операционной системы: открытие и сохранение файла, выбор цвета, установки параметров шрифта, настройки принтера и управление печатью.


Палитра **Данные** содержит компоненты для работы с таблицами баз данных.

3.2. Элемент управления кнопка (Button)

Компонент **кнопка** относится к элементам управления. На панели компонентов **Стандартные элементы управления** кнопка изображена в виде  Button, имя объекта — button. Кнопка, помещенная на форму, получает имя buttonN, где N — номер 1, 2, 3... (пример 3.4). При необходимости кнопку можно переместить в любое место формы. Ключевые точки позволят установить нужный размер кнопки.

Некоторые свойства кнопки перечислены в таблице (пример 3.5).

Как видно из таблицы, многие свойства кнопки совпадают по именам и назначениям со свойствами формы, поэтому в дальнейшем для компонентов будут указываться только те свойства, которые отличны от уже описанных для других компонентов.

Основным событием кнопки является Click. Для создания обработчика события Click для кнопки можно поступить так же, как и при создании аналогичного обработчика для формы: выбрать событие на вкладке **События** () и выполнить двойной щелчок в поле напротив события Click. Можно просто выполнить двойной щелчок по кнопке. (Для формы основным событием является событие Load, поэтому при двойном щелчке по форме создается обработчик события Load.)

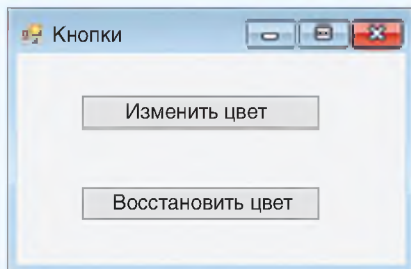
Пример 3.4. Компонент *кнопка* на форме:



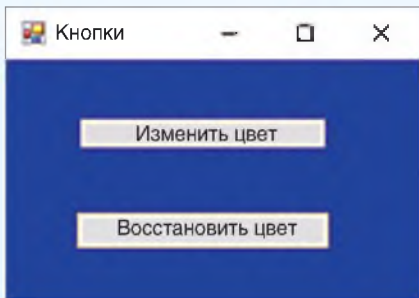
Пример 3.5. Свойства кнопки:

Некоторые свойства кнопки Button	
Свойство	Назначение
Text	Заголовок (внешнее имя) кнопки, текст, который отображается на кнопке. По умолчанию — button1
Font	Свойства шрифта для подписи заголовка. Свойство Font является сложным, о чем свидетельствует значок  , при нажатии на который раскрываются все свойства шрифта
Height	Высота кнопки
Weight	Ширина кнопки
Left	Горизонтальная координата положения верхнего левого угла кнопки на форме
Top	Вертикальная координата положения верхнего левого угла кнопки на форме
(Name)	Имя (внутреннее) кнопки. Используется в программном коде для обращения к объекту. Является идентификатором
Enabled	Значение True этого свойства обеспечивает доступность кнопки для мыши или клавиатуры
Visible	Значение True этого свойства обеспечивает видимость кнопки во время выполнения приложения


Пример 3.6. Внешний вид формы в режиме конструктора дизайна:



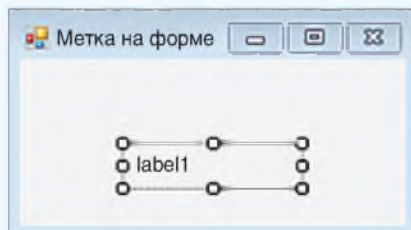
Внешний вид формы при выполнении:



Свойство кнопки `BackColor` позволяет менять ее цвет. Для обращения к этому свойству в программе используется запись: `button1.BackColor`.

Свойство кнопки `BackgroundImage` позволяет вставить на кнопку изображение, хранящееся в графическом файле. Для вставки можно использовать кнопку . Далее выбрать файл с рисунком.

Пример 3.7. Компонент *метка* на форме:



Пример 3.6. Создать проект, разместив на форме две кнопки. При нажатии на одну из них цвет формы должен измениться на синий, а при нажатии на вторую — должен восстановиться исходный цвет.

Этапы выполнения задания


1. Создать на форме две кнопки.
2. Изменить свойство `Text` у кнопки `button1` на **Изменить цвет**.
3. Изменить свойство `Text` у кнопки `button2` на **Восстановить цвет**.
4. Создать обработчик события `Click` для кнопки `button1` и изменить цвет формы. Команда

```
BackColor :• Color.Blue;
```

5. Создать обработчик события `Click` для кнопки `button2` и изменить цвет формы на первоначальный (название цвета формы указано в поле `Color` инспектора объектов). Команда `BackColor :• SystemColors.Control`;

6. Сохранить изменения в проекте. Название цвета `SystemColors.Control` задает не какой-то определенный цвет. Это цвет элемента управления, заданный цветовой схемой `Windows`. Поэтому он не обязательно будет серым.

3.3. Элемент управления *метка* (Label)

Компонент *метка* предназначен для отображения текста на форме. На панели компонентов **Стандартные элементы управления** метка изображена в виде , имя объекта — `label`. Кнопка, помещенная на форму, получает имя `labelN`, где `N` — номер 1, 2, 3... (пример 3.7).

Некоторые свойства метки, отличные от свойств кнопки, перечислены в таблице (пример 3.8). Основным событием для метки является Click.

Пример 3.9. Создать проект, в котором описана возможность выполнять следующие действия: после запуска программы в окне с именем «Работаем с кнопкой и меткой» при щелчке мыши по кнопке «Приветствие» появляется сообщение «Здравствуй, мир!».


Этапы выполнения задания

1. Изменить свойство Text формы на «Работаем с кнопкой и меткой».

2. Добавить на форму кнопку button1.

3. Изменить свойство Text кнопки на «Приветствие».

4. Добавить на форму метку label1.

5. Изменить свойства шрифта для компонента label1. Нажать кнопку 

в поле Font (цвет шрифта — синий, размер — 20, стиль — жирный курсив).


6. Очистить поле Text у метки.

7. Установить значение true у свойства метки Autosize.

8. В обработчик события Click для кнопки button1 вписать команду

```
label1.Text := 'Здравствуй, мир!';
```

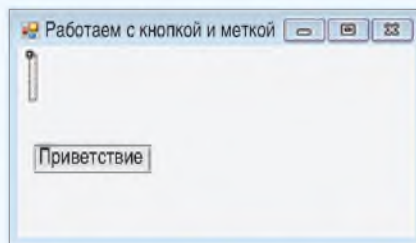
3.4. Элемент управления текстовое поле (Edit)

Текстовое поле — компонент, который предназначен для ввода и вывода текстовой информации. На панели компонентов **Стандартные элементы управления** текстовое поле изображено в виде , имя объекта — `TextBox`.

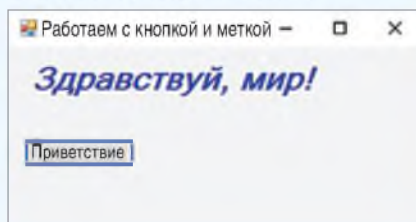
Пример 3.8. Свойства метки:

Свойство	Назначение
Text	Отображает введенный текст на форме
BackColor	Устанавливает цвет фона метки, который по умолчанию совпадает с цветом формы. Фон метки можно сделать прозрачным, задав свойству BackColor значение <code>Color.Transparent</code>
AutoSize	Значение true этого свойства приводит к автоматическому изменению размеров метки в соответствии с длиной текста
TextAlign	Выравнивание текста относительно границ метки: <div data-bbox="921 727 1122 873" data-label="Image"> </div>

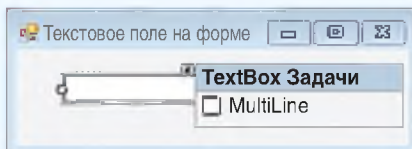
Пример 3.9. Форма на этапе конструирования:



Работающее приложение:



Пример 3.10. Компонент *текстовое поле* на форме:



Пример 3.11. Свойства текстового поля:

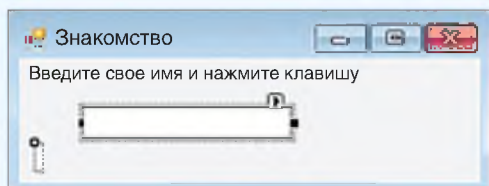
Свойство	Назначение
BorderStyle	Определяет границу вокруг текстового поля: None (нет границы), FixedSingl или Fixed3d (есть граница)
MaxLength	Ограничивает количество символов, которые можно ввести в TextBox
ReadOnly	Значение true запрещает редактирование текста, отображаемого в TextBox
Text	Содержит вводимый или выводимый текст

Текстовое поле часто называют текстовым редактором, поскольку оно снабжено такими функциями, как:

- копирование выделенного текста в буфер обмена (Ctrl+C);
- вырезание выделенного текста в буфер обмена (Ctrl+X);
- вставка текста из буфера обмена в позицию курсора (Ctrl+V);
- отмена последней команды редактирования (Ctrl+Z).

Свойство Multiline определяет, каким будет редактор — однострочным или многострочным.

Пример 3.12. Форма на этапе конструирования:



Текстовое поле, помещенное на форму, получает имя TextBoxN, где N — номер 1, 2, 3... (пример 3.10).

В отличие от ранее рассмотренных компонентов, свойство Text у текстового поля по умолчанию пусто (у других — совпадает с именем компонента). Некоторые свойства компонента TextBox приведены в таблице (пример 3.11).

Значение свойства Text компонента *текстовое поле* может изменяться программно или при вводе с клавиатуры. Основным событием для TextBox является TextChanged, которое происходит при изменении компонента. Наиболее часто программируют событие KeyPress, которое позволяет определить, какая клавиша была нажата.

Пример 3.12. Создать проект, в котором пользователя попросят ввести его имя, а затем, после нажатия клавиши Enter, будет выдано сообщение «Имя, приятно с Вами познакомиться!»

Этапы выполнения задания

1. Изменить свойство Text у формы на «Знакомство».
2. Разместить на форме две метки и текстовое поле.
3. Изменить свойство Text у label1 на «Введите свое имя и нажмите клавишу Enter».
4. Очистить поле свойства Text у Label2.
5. Написать обработчик события KeyPress для компонента Edit1, который будет проверять нажатие клавиши ввода (код клавиши Enter — 13). Если клавиша нажата, то поменять свойство Text у label2:

```

if e.KeyChar • #13 then
  label2.Text := TextBox1.Text +
    ', приятно с Вами познакомиться!';

```

Текстовое поле `TextBox` используется также для ввода и вывода чисел. При этом необходимо использовать функции для преобразования строк в числа и чисел в строки. Эти функции приведены в таблице:

Название функции	Действие
Ввод с помощью Edit	
<code>StrToInt</code>	Преобразование строки в целое число
<code>StrToFloat</code>	Преобразование строки в значение с плавающей запятой
Вывод с помощью Edit	
<code>IntToStr</code>	Преобразование целого числа в строку
<code>FloatToStr</code>	Преобразование вещественного числа в строку

В русскоязычной версии Windows в качестве разделителя целой и дробной части числа по умолчанию используется запятая. Если при вводе чисел в текстовые поля использовать точку, то будет возникать ошибка преобразования типов.

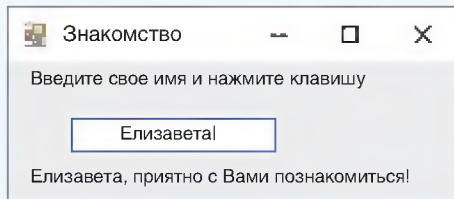
Пример 3.13. Создать проект, в котором пользователь сможет ввести число, получить его значение в квадрате и квадратный корень из этого числа.

Этапы выполнения задания

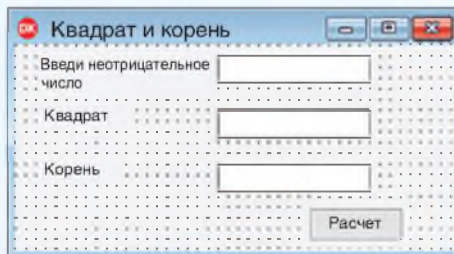
1. Изменить свойство `Text` у формы на «Квадрат и корень».

2. Разместить на форме три метки, три текстовых поля и кнопку.

Пример 3.12. Продолжение.
Работающее приложение:



Пример 3.13. Форма на этапе конструирования:



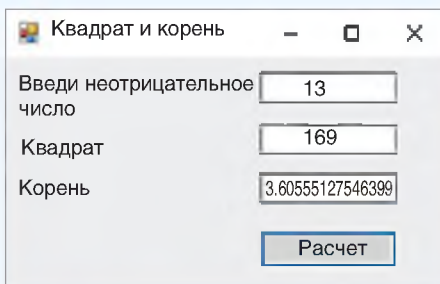
Обработчик события `OnClick` для `Button1`:

```

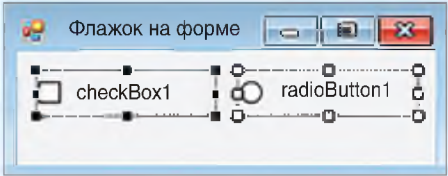
procedure Form1.button1_Click
(sender: Object; e: EventArgs);
var a, b: integer;
    c: real;
begin
  a := StrToInt(TextBox1.Text);
  b := a * a;
  c := sqrt(a);
  TextBox2.Text := IntToStr(b);
  TextBox3.Text := FloatToStr(c);
end;

```

Работающее приложение:



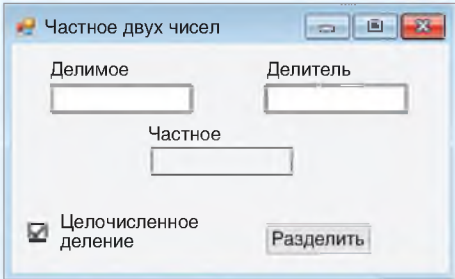
Пример 3.14. Компоненты флажок и радиокнопка на форме:



Пример 3.15. Свойства компонента флажок:


Свойство	Назначение
Checked	Значение true у этого свойства показывает, что установлена галочка —  , пустое окно индикатора —  соответствует значению false
Text	Надпись возле компонента checkBox
RightToLeft	Определяет, с какой стороны кнопки размещается надпись: Yes (надпись слева), No (надпись справа)
CheckState	Определяет состояние компонента: Unchecked — не выделен, Checked — выделен, Indeterminate ( — промежуточное состояние. Первые два состояния соответствуют свойству Checked


Пример 3.16. Форма на этапе конструирования:



- 3. Изменить свойство Text у Label1 на «Введите неотрицательное число».
- 4. Изменить свойство Text у Label2 на «Квадрат».
- 5. Изменить свойство Text у Label3 на «Корень».
- 6. Изменить свойство Text у Button1 на «Расчет».
- 7. Написать обработчик Click для кнопки.

3.5*. Элементы управления флажок (CheckBox) и переключатель (RadioButton)

Флажок используется в приложениях для включения или выключения каких-либо опций. На панели компонентов **Стандартные элементы управления** флажок изображен в виде  CheckBox, имя объекта — CheckBox. Флажок, помещенный на форму, получает имя checkBoxN, где N — номер 1, 2, 3... (пример 3.14). Некоторые свойства компонента checkBox приведены в таблице (пример 3.15).

Аналогичным образом используется компонент **Переключатель (радиокнопка)**. На панели компонентов Standard радиокнопка изображена в виде  RadioButton, имя объекта — RadioButton. Переключатель, помещенный на форму, получает имя radioButtonN, где N — номер 1, 2, 3... (пример 3.14).

Обычно радиокнопки образуют группы взаимосвязанных индикаторов, позволяющих выбрать только одну из нескольких взаимоисключающих альтернатив. При размещении на форме нескольких переключате-

лей включенным должен быть только один из них (контейнер GroupBox).

Пример 3.16. Создать проект для вычисления частного от деления одного целого числа на другое. Числа задаются в текстовых полях. Результат вычисляется при нажатии на кнопку «Частное» и помещается в третье текстовое поле. Результат зависит от состояния флажка.

Этапы выполнения задания

1. Поместить на форму текстовые поля (3), надписи (3), флажок и кнопку.

2. Для компонента textBox3 установить значение true для свойства ReadOnly.

3. Изменить свойство Text у компонентов label («Делимое», «Делитель», «Частное»).

4. Изменить свойство Text компонента button1 на «Разделить».

5. Изменить свойство Text компонента checkBox1 на «Целочисленное деление».

6. Написать обработчик события Click для компонента button1.

6.1. Проверить, что поля компонентов textBox1 и textBox2 не пусты. Иначе вывести сообщение «Одно из полей не заполнено».

6.2. Проверить состояние переключателя checkBox. Если он включен, то выполнить целочисленное деление, иначе обычное деление.

6.3. Вывести результат.

7. Выполнить программу для различных значений. Проверить работу приложения, когда одно из полей textBox1 или textBox2 (или оба поля) пусты.

Обработчик события OnClick для Button1:

```
procedure Form1.button1_Click
(sender: Object; e: EventArgs);
var a, b, c : integer;
    d : real;
begin
    if (TextBox1.Text <> '') and
       (TextBox2.Text <> '') then
    begin
        a := StrToInt(TextBox1.Text);
        b := StrToInt(TextBox2.Text);
        if CheckBox1.Checked then
        begin
            c := a div b;
            TextBox3.Text := IntToStr(c);
        end
        else
        begin
            d := a / b;
            TextBox3.Text := FloatToStr(d);
        end;
    end
    else
        MessageBox.Show('Одно из
полей не заполнено');
end;
```

Работающее приложение:



1. Какие компоненты относят к элементам управления?
2. Как поместить компонент на форму?
3. Какие свойства компонента `button` вы можете назвать?
4. Какое событие является основным для компонента `button`?
5. Для чего предназначен компонент `label`?
6. В каких случаях используется компонент `TextBox`?
7. Для чего предназначены компоненты `checkBox` и `radioButton`?



Упражнения

1 Откройте проект из примера 3.9 и дополните его кнопкой «Очистить»¹. Кнопка «Очистить» должна очищать текст метки (Свойству `Caption` присвоить значение пустой строки: `""`). Сделайте случайным выбор цвета и размера шрифта у метки.

2 Откройте проект из примера 3.12 и добавьте на форму три метки и две кнопки.

1. Измените свойства компонентов в соответствии с указаниями в таблице.

Компонент	Свойство	Значение свойства
button1	Text	Да
button1	Visible	False
button2	Text	Нет
button2	Visible	False
label3	Text	Вы хотите работать в ИТ?
label3	Visible	False
label4	Text	Замечательно! Успехов в изучении информатики! Она Вам понадобится!
label4	Visible	False
label5	Text	Другие профессии тоже требуют знания информатики
label5	Visible	False

2. Добавьте в обработчик события `KeyPress` команду, которая делает надпись `Label3` и кнопки видимыми.

¹ Перед изменением скопируйте проект в новую папку

```

procedure Form1.textBox1_KeyPress(sender: Object;
e: KeyPressEventArgs);
begin
    if e.KeyChar • #13 then
        begin
            label2.Text := TextBox1.Text + ', приятно с Вами познакомиться!';
            Label3.Visible := True;
            Button1.Visible := True;
            Button2.Visible := True;
        end;
    end;
end;

```

3. Напишите обработчики Click для кнопок Button1 и Button2. Сделайте видимыми соответствующие надписи.

Форма на этапе проектирования	Работающее приложение после запуска
<p style="text-align: center;">Работающее приложение до ответа на вопрос</p>	<p style="text-align: center;">Работающее приложение после ответа на вопрос</p>

4*. Добавьте в приложение еще один вопрос. Форму ответа выберите самостоятельно.

3 Создайте проект **Калькулятор**. Разместите на форме три поля `TextBox` и три надписи: «Первое число», «Второе число», «Результат». Добавьте кнопки для вычисления суммы, разности, произведения и частного. Запретите редактирование в поле с ответом. *Добавьте проверку деления на ноль.

Форма на этапе проектирования	Работающее приложение

4 Создайте проект, в котором вычисляется доход по вкладу. Программа должна обеспечивать расчет денежных сумм для простых или капитализированных вкладов. Если вклад простой, то процентная ставка начисляется от исходной суммы, и каждый месяц она одинаковая, а если капитализированный, то процентная ставка начисляется каждый месяц от суммы вклада в предыдущем месяце.

Форма на этапе проектирования	Работающее приложение


Проверьте, заполнены ли поля с исходными данными. Если нет, то выведите соответствующее сообщение.

5 Реализовать «убегающую кнопку», т. е. при наведении указателя мыши на кнопку она должна случайным образом поменять место.


6 Добавить в упражнение 5 кнопку «Домой», которая должна передвинуть «убегающую» в верхний левый угол формы.

§ 4. Элементы управления для работы с графикой

4.1. Элемент управления для вставки рисунка (PictureBox)

При создании приложений нередко возникает необходимость украсить их графическим изображением. В этом случае можно воспользоваться компонентом **изображение**. На панели компонентов **Стандартные элементы управления** компонент изображение представлен в виде  PictureBox, имя объекта PictureBox. Компонент PictureBox, помещенный на форму, получает имя PictureBox N, где N — номер 1, 2, 3... (пример 4.1).

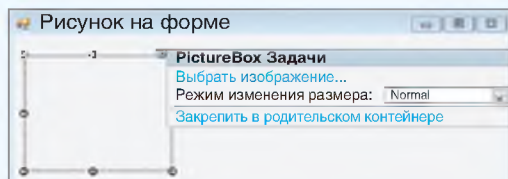
Компонент представляет собой контейнер, в который помещается изображение. Некоторые свойства компонента PictureBox приведены в таблице (пример 4.2).

Используя свойство Image, можно выбрать и загрузить изображение на этапе проектирования приложения. Изображение может быть выбрано в контейнере при нажатии на кнопку  в правом верхнем углу компонента. В этом случае рисунок сохраняется в файле формы и для работы приложения отдельного файла с рисунком не требуется.

Компонент поддерживает вставку рисунков в форматах JPEG, PNG, BMP, GIF. Если требуется обработка изображения (любые изменения рисунка), то рисунок должен быть сохранен в формате BMP. Для рисунков формата PNG или GIF с прозрачным фоном при загрузке сохраняется прозрачность.

Рисунок можно загрузить как фон формы. Для этого предназначено свойство формы BackgroundImage.

Пример 4.1. Компонент *изображение* на форме:



Пример 4.2. Некоторые свойства компонента *изображение*:

Свойство	Назначение
Image	Используется для отображения изображений
SizeMode	Определяет способ отображения рисунка: <i>Normal</i> — левый верхний угол рисунка совмещен с левым верхним углом контейнера; <i>StretchImage</i> — рисунок вписывается в контейнер; <i>AutoSize</i> — размер подгоняется под размер рисунка; <i>CenterImage</i> — рисунок будет отцентрирован относительно компонента; <i>Zoom</i> — при изменении размеров контейнера будут сохраняться пропорции рисунка
Margin	Определяет промежутки между полями изображения и полями другого компонента (значения All, Left, Top, Right, Bottom)
BackColor	Цвет фона изображения. Может быть прозрачным

Свойство Image компонента PictureBox обладает методом Save, который используется для сохранения изображения. Метод Load компонента PictureBox может быть использован для загрузки изображения при открытии приложения. В этом случае файл с рисунком должен находиться в папке проекта (или нужно прописать полный путь к файлу).

Пример 4.3. Форма на этапе конструирования:



Обработчик события Click для Button1:

```
procedure Form1.button1_Click
(sender: Object; e: EventArgs);
begin
    PictureBox2.Top := 220;
    var rnd: Random := new Random();
    PictureBox2.Left := rnd.
    Next(300);
    PictureBox2.Visible := True;
end;
```

Работающее приложение

До нажатия на кнопку:



После нажатия на кнопку:



Поскольку горизонтальное положение медведя задается случайным образом, то при каждом нажатии на кнопку медведь будет прорисован в новом месте.

Пример 4.3. Создать проект, разместить в нем фоновое изображение на форме. При нажатии на кнопку поверх фонового изображения должно появиться другое изображение.

Этапы выполнения задания

1. Установить размеры формы Height • 450, Width • 670.

2. Загрузить фоновое изображение для формы. Задать для свойства формы BackgroundImageLayout значение Stretch.

3. Поместить на форму компонент *изображение* и кнопку.

4. Для компонента PictureBox установить значение для свойства Visible • False (изображение невидимо при запуске приложения). Размеры Height • 120, Width • 200. Свойство SizeMode • StretchImage.

5. Загрузить изображение в компонент PictureBox. Изображение может быть формата PNG, или GIF с прозрачным фоном, или формата BMP с фоном однородного цвета. Свойство BackColor • Transparent.

6. Написать обработчик события OnClick для компонента Button1.

Если при запуске приложения изображение мерцает, то устранить мерцание можно с помощью включения двойной буферизации:

```
DoubleBuffered := true;
```

Эта команда должна быть прописана в обработчике события Load для формы.

4.2. Построение графиков функций

Пространство имен System.Drawing обеспечивает доступ к функциональным возможностям графического интерфейса Windows. Класс Graphics

предоставляет методы для рисования графических примитивов.

Основное событие для построения изображений — Paint.

Если возникает необходимость рисовать по точкам, то для этого используется класс `Bitmap` (точечное изображение). Каждая точка имеет координаты X и Y . Система координат такая же, как и для графического окна `PascalABC.Net` — точка с координатами $(0, 0)$ расположена в верхнем левом углу, ось OY направлена вниз. Каждая точка имеет координаты X и Y . Координаты измеряются в пикселях. Важнейшее свойство пикселя — его цвет. Для задания цвета в `PascalABC.Net` можно воспользоваться несколькими способами (пример 4.4). Точка изображается с помощью команды `SetPixel(x1, y1, Color);`

Класс `Graphics` содержит большое количество свойств и методов, позволяющих строить изображения. Многие из методов `Graphics` совпадают с процедурами, которые использовались в библиотеке `GraphABC` среды программирования `PascalABC.Net`. Описание этих методов приведено в приложении.

Пример 4.5. Создать проект и построить график функции $y = x \sin x$ на промежутке, заданном пользователем.

Этапы выполнения задания

1. Поместить на форму компоненты: `PictureBox`, два компонента `Label`, два компонента `TextBox` и компонент `Button`.

2. Изменить свойства `Text` у компонентов `Label1`, `Label2` на x_0 и x_1 соответственно.

Пример 4.4. Способы задания цвета в `PascalABC.Net`:

1. Задание цвета с помощью констант. Константы хранят имя цвета (например, `Color.SkyBlue` — небесно-синий, `Color.Red` — красный). Возможные значения появляются в выпадающем списке после ввода в программу ключевого слова `Color`.

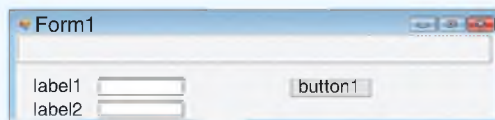


2. Для задания цвета можно воспользоваться функцией `Color.FromArgb(A, R, G, B)`, параметры которой задают прозрачность (альфа-канал) и интенсивность красного, синего и зеленого цветов соответственно. Значения параметров могут изменяться от 0 до 255.

3. Задание цвета с помощью шестнадцатеричных чисел. Пары цифр шестнадцатеричного числа задают прозрачность и интенсивность красного, зеленого и синего цветов соответственно. Полученное шестнадцатеричное число записывается как параметр функции `FromArgb`. Например, `Color.FromArgb($FF0000FF)` — синий цвет. Значения для других цветов: `$FFFFFF0000` — красный, `$FF00FF00` — зеленый, `$FF000000` — черный, `$FFFFFFFF` — белый.

Шестнадцатеричные числа можно перевести в десятичную систему счисления и пользоваться этими значениями. Например, `Color.FromArgb(4278190335)` — синий цвет.

Пример 4.5. Форма на этапе конструирования:



Пример 4.5. Продолжение.

Обработчик события Click для компонента Button1:

```
procedure Form1.button1_Click
(sender: Object; e: EventArgs);
var x, y, h, k, x0, xn: real;
    x1, y1, n, c_x, c_y: integer;
    gr: Graphics;
    bm: Bitmap;
    p_c: Pen;
begin
    //подготовка графической области
    //для рисования по пикселям
    bm := new Bitmap (PictureBox1.Width,
        PictureBox1.Height);
    pictureBox1.Image := (Image) (bm);
    gr := Graphics.FromImage
        (pictureBox1.Image);
    //закраска графической области белым
    //цветом
    gr.Clear(Color.White);
    // количество точек
    n := 10000;
    // концы промежутка
    x0 := StrToFloat(TextBox1.Text);
    xn := StrToFloat(TextBox2.Text);
    // центр области построения
    c_x := PictureBox1.Width div 2;
    c_y := PictureBox1.Height div 2;
    // масштабный коэффициент
    k := PictureBox1.Width / (xn - x0);
    // шаг
    h := (xn - x0) / n;
    x := x0;
    //оси
    p_c := new Pen(Color.Black, 1);
    gr.DrawLine(p_c, 0, c_y, 2*c_x, c_y);
    gr.DrawLine(p_c, c_x, 0, c_x, 2*c_y);
    for var i := 1 to n do
    begin
        y := x * sin(x);
        x1 := trunc(x * k) + c_x;
        y1 := trunc(-y * k) + c_y;
        //проверка попадания точки
        //в графическую область
        if (y1 >= 0) and (y1 < 2*c_y) then
            bm.SetPixel(x1, y1, Color.Blue);
        x := x + h;
    end;
end;
```

3. Изменить свойства Text у компонентов Edit1 и Edit2 на -20 и 20 соответственно.

4. Изменить свойства Text у компонента Button1 на «Построить график».

5. Написать обработчик события Click для компонента Button1 и построить в нем график функции по точкам.

5.1. Нарисовать оси координат в виде двух перпендикулярных линий, пересекающихся в центре компонента PictureBox.

5.2. Чтобы получить видимость сплошной линии, количество точек, которые образуют график функции, должно быть не менее 10 000 ($n \cdot 10\,000$).

5.3. Шаг изменения значения x определяется как $h = \frac{x_n - x_0}{n}$.

5.4. При построении нужно учитывать масштаб: ширина компонента PictureBox должна соответствовать длине заданного промежутка. Тогда масштабный коэффициент можно рассчитать по формуле

$$h = \frac{\text{PictureBox1.Width}}{x_n - x_0}.$$

5.5. Поскольку расположение осей координат на экране не совпадает с расположением осей, принятым в математике, то нужно преобразовать координаты: точке (0; 0) должна соответствовать точка в центре компонента PictureBox. Для этого полученное значение x нужно увеличить на величину $c_x \cdot \text{PictureBox1.Width} \div 2$, а значение y на $c_y \cdot \text{PictureBox1.Height} \div 2$. Так как ось Y направлена вниз, а не вверх, то у значения Y нужно еще поме-

нять знак на противоположный. На канве будет закрашиваться точка с координатами $(x_{\text{екр}} \cdot x \cdot k + c_x, y_{\text{екр}} \cdot -y \cdot k + c_y)$.

5.6. Необходимо учитывать, что при вычислении значения x и y будут вещественными, а значения графических координат могут быть только целыми. Поэтому перед прорисовкой точки нужно преобразовать вещественные числа в целые с помощью функции *trunc*.

5.7. Некоторые точки графика при построении могут оказаться за пределами графической области, поэтому необходима проверка значения $y1$: значение должно быть неотрицательным и меньше высоты графической области.

4.3. Построение диаграмм

Основные принципы построения гистограмм и круговых диаграмм разбирались в 10-м классе (примеры 4.6 и 6.8). Используя аналогичные методы Graphics, можно построить диаграммы в оконных приложениях, созданных в PascalABC.Net.

Пример 4.6. Создать проект и построить гистограмму по данным массива из n элементов ($n \cdot 10$). Описать массив с константными данными или добавить данные в массив случайным образом.

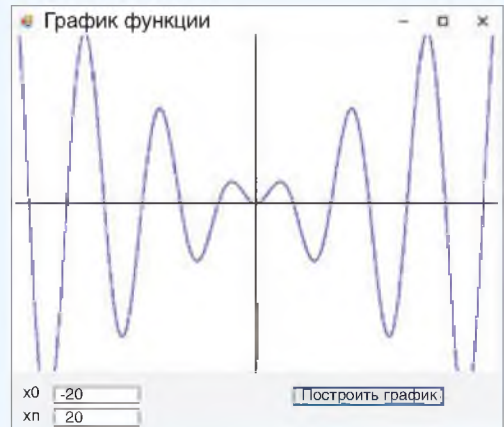
Этапы выполнения задания

1. Поместить на форму компоненты: PictureBox и два компонента Button.

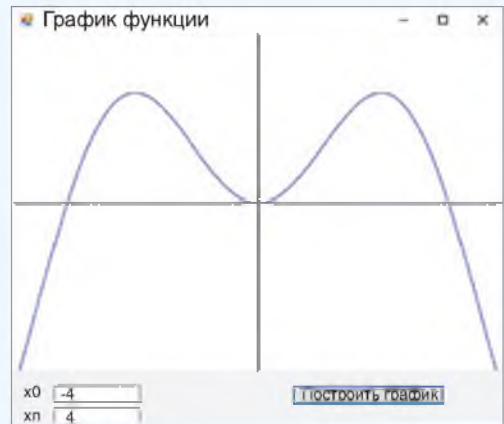
2. Изменить свойства Text у компонента Button1 на «Диаграмма с константными данными».

Пример 4.5. Продолжение.

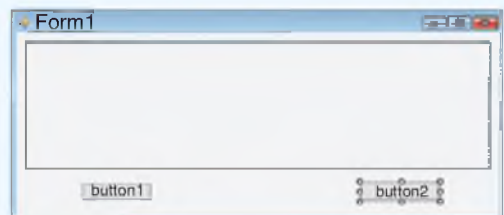
Работающее приложение:



Изменение начальных значений концов промежутка:



Пример 4.6. Форма на этапе конструирования:

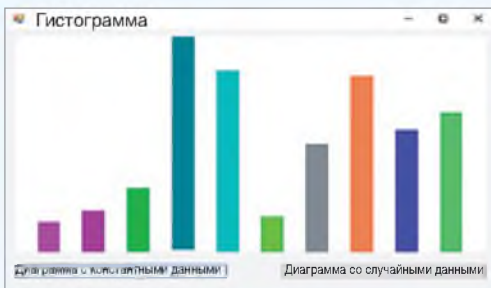


Пример 4.6. Продолжение.

Обработчик события Click для компонента Button1:

```
procedure Form1.button1_Click
(sender: Object; e: EventArgs);
const a: array[1..10] of integer =
(10, 14, 22, 75, 63, 12, 37, 61, 42, 48);
n = 10;
var max, x, y1, y2, h, i, cr, cg,
cb: integer;
m: real;
gr: Graphics;
rnd: Random;
sb: SolidBrush;
begin
max := a[1];
for i := 2 to n do
if a[i] > max then
max := a[i];
h := trunc(PictureBox1.Width/(2*n+1));
m := PictureBox1.Height / max;
x := h;
//подготовка графической области
//для рисования примитивов
gr := PictureBox1.CreateGraphics;
gr.Clear(Color.White);
rnd := new Random();
for i := 1 to n do
begin
cr := rnd.next(256);
cg := rnd.next(256);
cb := rnd.next(256);
sb := new SolidBrush(Color.FromArgb
(cr, cg, cb));
y1 := PictureBox1.Height-1;
y2 := y1-trunc(a[i] * m)-1;
gr.FillRectangle(sb, x, y2, h, y1);
x := x + 2 * h;
end;
end;
```

Работающее приложение:



3. Изменить свойства Text у компонента Button2 на «Диаграмма со случайными данными».

4. Написать обработчик события Click для компонента Button1, в котором диаграмма строится с помощью прямоугольников.

4.1. Найти максимальный элемент в массиве *max*.

4.2. Рассчитать масштабный коэффициент:

$$m = \frac{PictureBox1.Height}{max}.$$

4.3. В цикле строить *n* прямоугольников одинаковой ширины. Ширина прямоугольника

$$h = \frac{PictureBox1.Width}{2n + 1}.$$

5. Обработчик для компонента Button2 будет отличаться от обработчика для компонента Button1 только способом получения элементов массива.

5.1. Массив должен быть описан в разделе var: *a: array[1..10] of integer*;


5.2. Элементы массива со значениями от 20 до 100 можно получать следующим образом:

```
rnd := new Random();
for i := 1 to n do
a[i] := rnd.next(80) + 20;
```

4.4. Анимация

Эффект анимации достигается за счет того, что перед взглядом пользователя происходит быстрая смена изображений. Каждый из кадров анимации остается на экране очень небольшой промежуток времени.

Для замера интервалов времени можно использовать компонент *таймер*.

Он расположен на панели **Компоненты** и представлен в виде  Timer, имя объекта Timer. Компонент Timer помещается в отдельную область ниже формы и получает имя TimerN, где N — номер 1, 2, 3... (пример 4.7).

Некоторые свойства компонента Timer приведены в таблице:

Свойство	Назначение
Enabled	Значение true обозначает, что таймер запущен
Interval	Время в миллисекундах, через которое происходит срабатывание таймера и вызов обработчика Tick

Компонент имеет единственный обработчик — Tick, в котором описываются действия, происходящие по истечении интервала срабатывания таймера.

Пример 4.8. Создать проект, в котором самолет будет пролетать над городом.

Этапы выполнения задания

1. Поместить на форму компоненты PictureBox и Button, добавить компонент Timer.

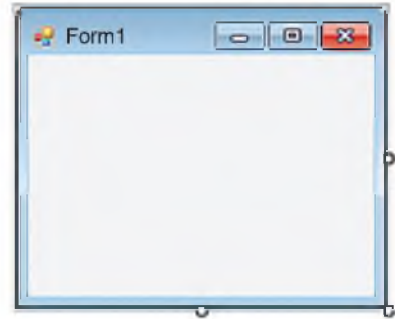
2. Загрузить изображение города в компонент как фон формы (свойство формы BackgroundImage).

3. Установить прозрачный цвет фона для компонента PictureBox1 (значение Transparent у свойства BackColor). Установить режим изменения размера — AutoSize (свойство SizeMode).

4. Написать обработчик события Load для формы и описать начальное

Пример 4.7. Компонент таймер:

Дизайнер Код



Компонент Timer не виден при работе приложения, поэтому он размещается в области, специально предназначенной для невидимых компонентов.

Пример 4.8. Форма на этапе конструирования:



Обработчик события Load для формы:

```
procedure Form1.Form1_Load (sender:
Object; e: EventArgs);
begin
    PictureBox1.Load ('plane.png');
    x := -PictureBox1.Width;
    y := 20;
    PictureBox1.Left := x;
    PictureBox1.Top := y;
end;
```

Пример 4.8. Продолжение.

Обработчик события Click для компонента Button1:

```
procedure Form1.button1_Click
(sender: Object; e: EventArgs);
begin
  Timer1.Enabled := True;
end;
```

Обработчик события Tick для компонента Timer1:

```
procedure Form1.timer1_Tick
(sender: Object; e: EventArgs);
begin
  x := x + 1;
  PictureBox1.Left := x;
  if PictureBox1.Left > Width +
    + PictureBox1.Width then
    x := -PictureBox1.Width;
end;
```

Работающее приложение:



Если при запуске анимации возникает мерцание, то включить двойную буферизацию.

положение самолета, указав координаты верхнего левого угла PictureBox1 за пределами формы. Загрузить в PictureBox1 изображение из файла с рисунком самолета.

5. Изменить свойства Text у компонента Button1 на «Полетели!».

6. Установить значение False у свойства таймера Enabled в инспекторе объектов.

7. Установить в инспекторе объектов время срабатывания таймера, равным 10.

8. Написать обработчик события Click для компонента Button1, запустить таймер.

9. В инспекторе объектов установить прозрачность для компонента PictureBox1.

10. Написать обработчик события Tick и менять в нем значение свойства Left у компонента PictureBox1. Если самолет вылетел за границу, то вернуть его в начальное положение.

Двойная буферизация позволяет сделать анимацию более плавной, поскольку все операции рисования сначала выполняются в памяти, а лишь затем на экране компьютера. После завершения всех операций рисования содержимое буфера копируется из памяти непосредственно на связанную с ним область экрана.

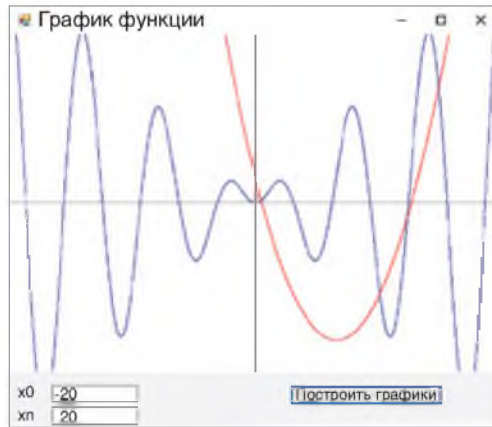


1. Какой компонент используется для размещения изображений?
2. Какое свойство позволяет загрузить готовое изображение в компонент PictureBox?
3. С помощью какого свойства можно установить прозрачный фон для изображения?
4. Какой класс представляет методы для рисования графических примитивов?
5. Какой метод канвы позволяет закрасить пиксель?
6. Какой компонент используется для отсчета времени?



Упражнения

- 1 Добавьте в проект из примера 4.4. еще одно животное (например, белку). Для размещения белки нужно добавить еще одну кнопку. Местоположение определить случайным образом в верхушке какого-либо дерева.
- 2 Добавьте в проект 4.5 перечисленные возможности.
 1. Оси координат со стрелками и подписями.
 2. Единичный отрезок на оси X.
 3. График функции $y = 0,3x^2 - 4x + 2$ в той же системе координат красным цветом.



Постройте в одной системе координат графики следующих функций. Каждую кривую рисовать своим цветом. Для ввода значений параметров a , b и c добавьте на форму компоненты Text:

1. $y = ax^2$, $y = b \cos x^2$;
2. $y = ax^2 + bx + c$, $y = \frac{1}{x^2 + 3}$;
3. $y = ax^3$, $y = \frac{x}{(x+3)^2}$, $y = bx \sin x$.

- 3 Измените проект из примера 4.6 так, чтобы строилась линейчатая диаграмма (столбики расположены горизонтально).
- 4* Измените проект из примера 4.6 так, чтобы значения в массив можно было вводить. Для этого числа необходимо записывать в компонент TextBox, считывать строку из чисел и пробелов, выделять цифры и преобразовывать их в числовые значения.
- 5 Добавьте в проект из примера 4.8 кнопку «Стоп», при нажатии на которую самолет остановится. *После остановки самолета должен появиться парашют и опуститься вниз.
- 6* Создайте анимацию движения Луны вокруг Земли. Для расчета координат верхнего левого угла PictureBox1, содержащего Луну, можно воспользоваться параметрическим уравнением окружности: $x = R \sin(t)$, $y = R \cos(t)$, где R — радиус, t — параметр, изменяющий свое значение от 0 до 2π .

§ 5. Создание приложений

Пример 5.1. Рекомендации по созданию оконных приложений.

1. В приложении рекомендуется разместить главное меню и инструментальную панель быстрых кнопок, дублирующих основные разделы меню.

2. Желательно, чтобы объекты приложения обладали контекстными меню, появляющимися при нажатии правой клавишей мыши на объекте.

3. Для объектов рекомендуется прописать подсказки, всплывающие при наведении указателя мыши на объект.

4. Рекомендуется реализовать строку состояния, используемую для выдачи различной информации.

5. При нажатии клавиши F1 должен загружаться файл справки.

6. В программе желательно реализовать возможность настройки и сохранения настроек, чтобы при следующем сеансе работы их не пришлось устанавливать заново.

7. Если результат работы приложения зависит от каких-либо параметров, обязательно укажите значения по умолчанию. Они позволят ускорить взаимодействие пользователя с программой, а также являются примером того, в каком формате данные следует вводить.

Мощным воздействием на психику человека является цвет, поэтому с ним нужно обращаться очень осторожно. Нужно стремиться использовать ограниченный набор цветов и уделять внимание их правильному сочетанию. Восприятие цвета у человека очень индивидуально, поэтому не стоит навязывать всем свое видение цвета. Желательно, чтобы основной цвет формы был нейтральным (например, у большинства приложений Microsoft это светло-серый цвет).

5.1. Разработка оконных приложений

Создание любого оконного приложения осуществляется, как правило, в три этапа:

1. Создание интерфейса приложения, т. е. средств взаимодействия пользователя с программой.

2. Разработка сценария работы будущего приложения. На этом этапе определяют, какая информация будет выводиться на экран, какие события будут происходить при использовании различных компонентов, как приложение должно завершить работу, какие результаты и в каком виде сохранить и т. д.

3. Разработка алгоритма решения поставленной задачи.

Большинство приложений в операционной системе Windows выглядят и ведут себя сходным образом. Компания Microsoft предложила рекомендации для разработки программного обеспечения¹, направленные на то, чтобы пользователь не тратил время на освоение нюансов пользовательского интерфейса новой программы, а сразу начал продуктивно ее использовать. Эти рекомендации основаны на психофизиологических особенностях человека и существенно облегчат жизнь будущим пользователям вашей программы.

Приведем некоторые рекомендации по разработке графического интерфейса оконных приложений (пример 5.1).

¹ <https://docs.microsoft.com/ru-ru/windows/uwp/design/basics/design-and-ui-intro>

5.2. Стандартные диалоги

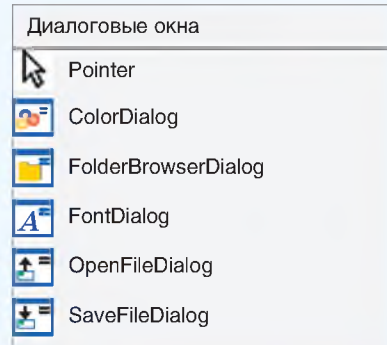
Практически любое приложение Windows использует стандартные диалоги, встроенные в операционную систему, для открытия и сохранения файлов, выбора атрибутов шрифта или установки цвета, поиска текста, печати. В библиотеку VCL включены компоненты, реализующие соответствующие окна Windows. Они размещены на панели **Диалоговые окна** (пример 5.2). В примере 5.3 приведен перечень компонентов для реализации стандартных диалогов.

Объекты на странице **Диалоговые окна** невидимы во время выполнения, поэтому они размещаются в специальной области под формой (пример 5.4). Внешний вид окна диалога зависит от версии Windows.

Вызов и обработка диалогов происходит программно. Для всех диалогов определен метод `ShowDialog()` (пример 5.5). С помощью этого метода происходит открытие окна соответствующего диалога. В свойствах компонента-диалога запоминается выбор пользователя, который затем можно обработать.

Диалоги для открытия и сохранения файлов используются в различных приложениях. Основное свойство компонентов `OpenDialog` и `SaveDialog`, в котором возвращается в виде строки имя файла, — это свойство **FileName**. Если задать данное свойство на этапе конструирования в окне инспектора объектов, то при открытии диалога

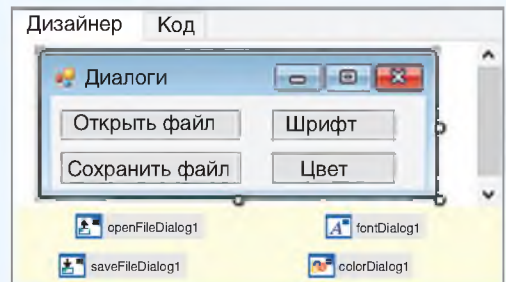
Пример 5.2. Панель Диалоговые окна.



Пример 5.3. Список некоторых стандартных диалогов.

Компонент	Назначение
OpenFileDialog	Создание окна диалога «Открыть файл»
SaveFileDialog	Создание окна диалога «Сохранить файл»
FontDialog	Создание окна диалога «Шрифт» — выбор атрибутов шрифта
ColorDialog	Создание окна диалога «Цвет» — выбор цвета

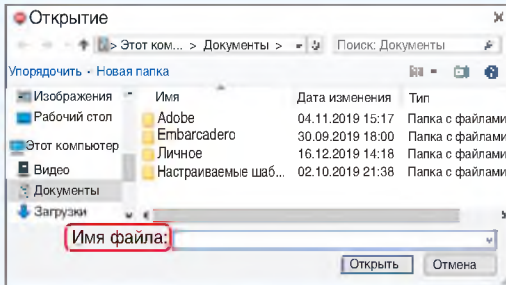
Пример 5.4. Диалоговые компоненты и кнопки для их вызова на форме:



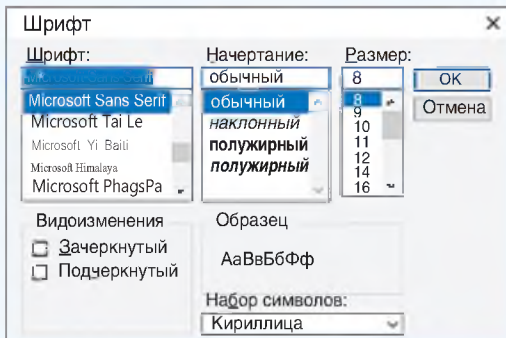
Пример 5.5. Стандартное обращение к диалогу:

```
<имя диалога>. ShowDialog();
```

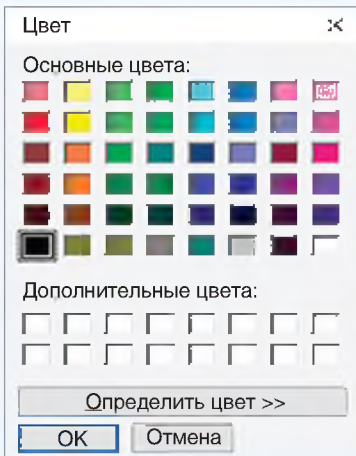
Пример 5.6. Стандартный диалог для открытия файла:



Пример 5.7. Стандартный диалог для настроек шрифта:



Пример 5.8. Стандартный диалог для выбора цвета:



оно будет появляться в строке **Имя файла** (пример 5.6).



Для вызова стандартного окна установки атрибутов шрифта можно использовать компонент **FontDialog** (пример 5.7). В окне **Шрифт** пользователь может выбрать имя шрифта, его стиль, размер. Основное свойство компонента — **Font**.

Для вызова стандартного окна установки цвета используется компонент **ColorDialog** (пример 5.8). В нем можно выбрать цвет из базовой палитры. Основное свойство компонента **ColorDialog** — **Color**. Это свойство соответствует тому цвету, который пользователь выбрал в диалоге.

5.3. Создание меню

Практически любое приложение должно иметь меню, которое дает удобный доступ к функциям программы. Существует несколько типов меню:

- **главное меню** с выпадающими списками разделов;
- **каскадные меню**, в которых разделу первичного меню ставится в соответствие список подразделов;
- **контекстные меню**, появляющиеся при нажатии правой клавишей мыши на объекте.

В **PascalABC.Net** меню создаются компонентами  **MenuStrip** (главное меню) и  **ContextMenuStrip** (контекстное меню), расположенными на панели **Меню** и панели **инструментов**. Во время выполнения программы сами компоненты не видны, поэтому раз-

мещаются в специальной области под формой (пример 5.9). На этапе выполнения программы главное меню будет помещено на свое стандартное место — наверху формы, контекстное меню появится только после нажатия правой кнопки мыши по тому компоненту, к которому оно относится.

Для добавления новых пунктов меню нужно кликнуть левой клавишей мыши в верхней части формы (там, где обычно располагается меню). Затем заполнить ячейки, соответствующие пунктам меню (пример 5.10).

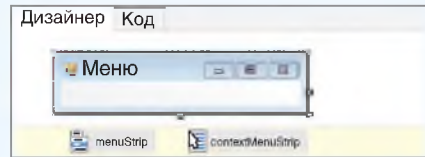
Каждый пункт меню является отдельным объектом. Список всех компонентов, относящихся к меню, можно увидеть в выпадающем списке в инспекторе объектов. Названия пунктов меню прописываются в свойстве Text в окне инспектора объектов (пример 5.11).

Для каждого пункта меню основным событием является событие Click.

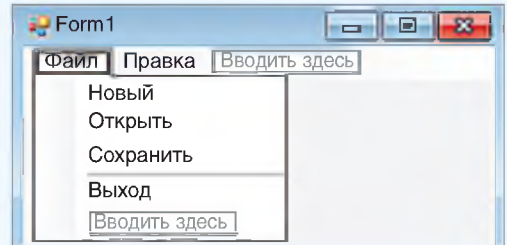
Создание контекстных меню аналогично созданию главного меню. Сначала нужно выбрать компонент на нижней панели, а затем заполнить ячейки. Для того чтобы при щелчке правой кнопкой мыши на некотором компоненте появлялось контекстное меню, нужно написать имя контекстного меню в свойстве ContextMenuStrip для выбранного компонента (пример 5.12).

Написание обработчиков для меню и диалогов будет рассмотрено в следующих пунктах.

Пример 5.9. Меню:

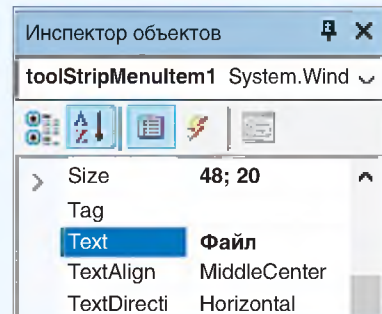


Пример 5.10. Редактирование меню:

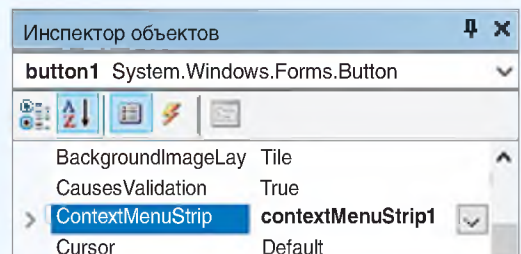


Если в качестве значения свойства Caption ввести «—», то вместо пункта меню появится разделитель.

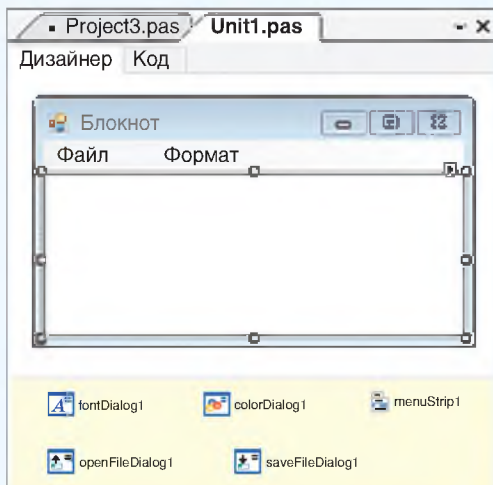
Пример 5.11. Название пункта меню в инспекторе объектов:



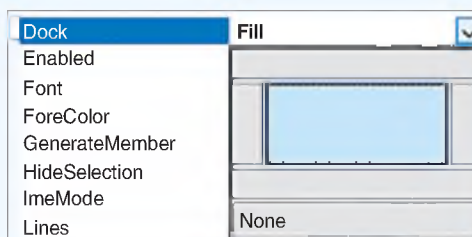
Пример 5.12. Контекстное меню для компонента Button1:



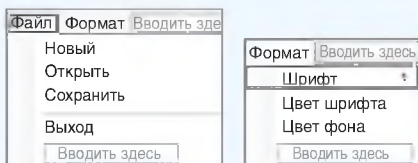
Пример 5.13. Форма на этапе конструирования:



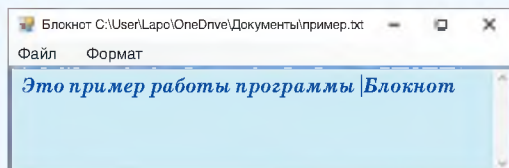
Пример 5.14. Настройка клиентской области:



Пример 5.15. Структура меню:



Пример 5.16. Работающее приложение:



5.4. Создание приложения «Блокнот»

Программа Блокнот должна давать возможность открыть и сохранить текстовый файл, выбрать цвет текста и цвет фона.

Разместить на форме (пример 5.13) следующие компоненты:

- рабочая область для текста — `TextBox1`;
- диалоги работы с файлами — `OpenFileDialog1`, `SaveFileDialog1`;
- диалоги для настройки внешнего вида приложения — `FontDialog1`, `ColorDialog1`;
- главное меню — `MenuStrip1`.

Компонент `TextBox1` предназначен для набора и редактирования текста. Текст может набираться в несколько строк, поэтому нужно установить значение `true` для свойства `MiltiLine`. Для того чтобы компонент занимал всю клиентскую часть формы, необходимо установить у свойства `Dock` значение `Fill` (пример 5.14). Установить значение `Vertical` для свойства `ScrollBars` (вертикальная полоса прокрутки).

Структура меню представлена в примере 5.15. Для написания обработчиков пунктов меню нужно в инспекторе объектов выбрать соответствующий пункт меню, перейти на вкладку `Events` и выбрать событие `Click`. Поскольку событие `Click` является событием по умолчанию, то двойной клик по пункту в редакторе меню создаст процедуру-обработчик.

Окно работающего приложения показано в примере 5.16.

Обработчики событий для каждого из пунктов меню представлены в примере 5.17.

Для сохранения и загрузки файлов опишем глобальную переменную `F_N`:

```
var F_N: String;
```

Обработчик пункта меню **Новый** (`StripMenuItem4`) очищает строки компонента `TextBox1` от введенного ранее текста.

Обработчики пунктов меню **Открыть** (`StripMenuItem5`) и **Сохранить** (`StripMenuItem6`) работают с файлом. Имя файла добавляется к заголовку окна.

Обработчик пункта меню **Выход** (`StripMenuItem8`) закрывает главную форму проекта.

Обработчик пункта меню **Шрифт** (`StripMenuItem9`) приписывает шрифту, связанному с компонентом `TextBox1`, свойства, выбранные пользователем.

Обработчики пунктов меню **Цвет текста** (`StripMenuItem10`) и **Цвет фона** (`StripMenuItem11`) устанавливают для `TextBox1` цвета текста и фона, выбранные пользователем.

Для компонента `TextBox` определены следующие действия: Копировать (`Ctrl + C`), Вырезать (`Ctrl + X`), Вставить (`Ctrl + V`), Отменить (`Ctrl + Z`).

5.5. Создание приложения «Графический редактор»

Программа «Графический редактор» должна давать возможность открыть и сохранить файл, выбрать цвет линии и цвет фона, установить толщину линии. Рисование производится выбранным цветом линии при нажатой левой клавише мыши. Клик правой клавишей мыши внутри замкнутой области используется для заливки ограниченной области выбранным цветом фона.

Пример 5.17. Обработчики событий для пунктов меню:

```
var s:string;

procedure
Form1.toolStripMenuItem4_Click
(sender: Object; e: EventArgs);
begin
    //Файл—Новый
    TextBox1.Clear;
end;

procedure
Form1.toolStripMenuItem5_Click
(sender: Object; e: EventArgs);
begin
    //Файл—Открыть
    openFileDialog1.ShowDialog();
    s:= openFileDialog1.FileName;
    Text := 'Блокнот ' + s;
    TextBox1.Lines := ReadAllLines(s);
end;

procedure
Form1.toolStripMenuItem6_Click
(sender: Object; e: EventArgs);
begin
    //Файл—Сохранить
    saveFileDialog1.ShowDialog();
    F_N:= saveFileDialog1.FileName;
    WriteAllLines(F_N,
        TextBox1.Lines);
    Text := 'Блокнот ' + F_N;
end;

procedure
Form1.toolStripMenuItem8_Click
(sender: Object; e: EventArgs);
begin
    //Файл—Выход
    close;
end;
```

Пример 5.17. Продолжение.

```

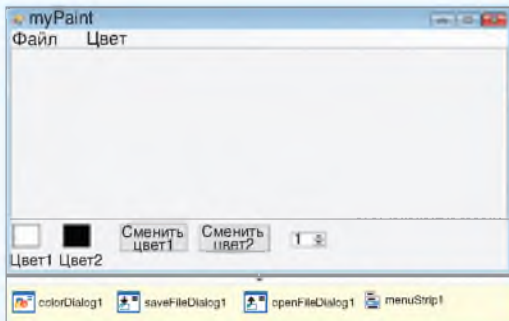
procedure Form1.
  toolStripMenuItem9_Click
  (sender: Object; e: EventArgs);
begin
  //Формат—Шрифт
  fontDialog1.ShowDialog();
  TextBox1.Font := fontDialog1.
  Font;
end;

procedure Form1.
  toolStripMenuItem10_
  Click(sender: Object; e:
  EventArgs);
begin
  //Формат—Цвет текста
  colorDialog1.ShowDialog();
  TextBox1.ForeColor :=
    colorDialog1.Color;
end;

procedure Form1.
  toolStripMenuItem11_
  Click(sender: Object; e:
  EventArgs);
begin
  //Формат—Цвет фона
  colorDialog1.ShowDialog();
  TextBox1.BackColor :=
    colorDialog1.Color;
end;

```

Пример 5.18. Форма на этапе конструирования:



Сначала спроектируем форму, разместив на ней следующие компоненты (пример 5.18):

- область для рисования — PictureBox;
- компоненты, отображающие выбранный цвет для рисования и цвет фона — Panel1, Panel2;
- кнопки для смены цвета;
- компонент выбора цвета — ColorDialog1;
- компонент для выбора толщины линии — numericUpDown1 (панель компонентов Стандартные элементы управления);
- главное меню — menuStrip1 и компоненты для работы с файлами — OpenFileDialog1, SaveFileDialog1.

На этапе конструирования установить значение свойства BackColor у компонентов Panel1 и Panel2 — Black и White соответственно.

У свойств Value и Minimum для компонента numericUpDown1 установить значение 1.

Структура меню показана в примере 5.19. Создание рисунка и загрузка файла в приложение приведены в примере 5.20.

В обработчике события Load для формы прописаны первоначальные установки для создания графического объекта, заданы параметры кисти и карандаша.

В обработчике события MouseDown для компонента PaintBox1 задаем переменной m_d значение true — кнопка нажата. Здесь же запоминаем координаты точки, поскольку от этой точки начнем строить линию. В обработчи-

ке `MouseUp` — значение переменной `m_d • false` — кнопка не нажата.

Для отслеживания траектории движения мыши по компоненту `PaintBox1` создаем обработчик события `MouseMove`. Если кнопка нажата, то можем строить линию. Параметры `e.x`, `e.y` возвращают координаты точки, в которой произошло нажатие кнопки.

Для перемещения мыши нужно использовать метод `DrawLine (x1, y1, e.x, e.y)` — рисование линии, соединяющей две точки. После прорисовки обновляем координаты.

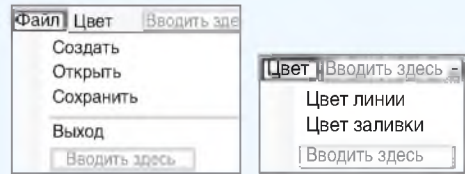
Толщина линии определяется значением свойства `Value` для компонента `numericUpDown1`. Обработчик события — `ValueChanged`.

Обработчики событий для компонентов `OpenFileDialog1`, `SaveFileDialog1` вызываются из соответствующих пунктов меню и аналогичны обработчикам, описанным для программы Блокнот. Для сохранения и загрузки файлов нужно описать глобальную строковую переменную `FileName`. Приложение может сохранять и загружать файлы формата BMP.

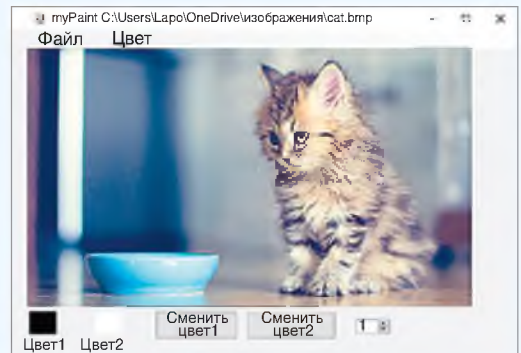
С помощью компонента `ColorDialog1` можно выбрать цвет линии или заливки. Пункты меню **Цвет** позволяют выбрать цвет линии или заливки соответственно.

В примере 5.21 приведено описание глобальных переменных, которые используются для создания приложения «Графический редактор». Обработчики всех описанных событий приведены в приложении.

Пример 5.19. Структура меню:



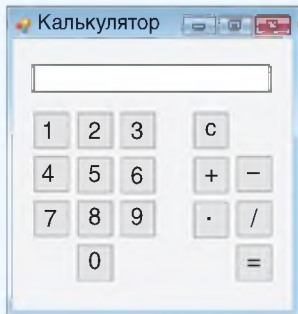
Пример 5.20. Работающее приложение:



Пример 5.21. Описание глобальных переменных.

```
var gr: Graphics;
    bm: Bitmap;
    p_c: Pen;
    s_b: SolidBrush;
    c_f, c_b: Color;
    w: decimal;
    x1, y1, x2, y2: integer;
    m_d: boolean;
    F_N: string;
```

Пример 5.22. Форма на этапе конструирования:



Пример 5.23. Обработчики событий:

```
//описание глобальных переменных
var n1, n2: integer;
    znak: char;

procedure Form1.button1_Click
(sender: Object; e: EventArgs);
begin
    //приписывание цифры к числу
    TextBox1.Text := TextBox1.Text + '1';
end;

procedure Form1.button9_Click
(sender: Object; e: EventArgs);
begin
    //приписывание цифры к числу
    TextBox1.Text := TextBox1.Text + '9';
end;
```

Для остальных цифровых кнопок нужно изменить только '1' на соответствующую цифру.

```
procedure Form1.button12_Click
(sender: Object; e: EventArgs);
begin
    n1 := StrToInt(TextBox1.Text);
    //запоминание знака операции
    znak := '+';
    TextBox1.Clear;
end;
```

Для остальных кнопок со знаками арифметических действий нужно изменить только '+' на соответствующий знак.

5.6. Создание приложения «Калькулятор»

Создание калькулятора начнем с конструирования формы. На ней нужно разместить: поле `TextBox` для ввода/вывода чисел, 10 кнопок с цифрами, 4 кнопки с арифметическими действиями, кнопку «•» и кнопку «C» — очистить (пример 5.22).

При нажатии на кнопку с цифрой программа должна дописать эту цифру к числу в поле `TextBox`. При нажатии на кнопку с арифметическим действием нужно запомнить число, которое в данный момент находится в поле `TextBox`, и очистить поле для ввода второго числа. Числа будем хранить в двух переменных `n1`, `n2` типа `integer`. Знак операции будем хранить в переменной `znak` типа `char`. Переменные описываются как глобальные. При нажатии на кнопку «•» выполняется арифметическое действие и выводится результат.

Кнопки могут содержать рисунок на поверхности (например, изображения с цифрами). Свойство для размещения рисунка — `BackgroundImage`.

Установить значение `FixedSingle` для свойства `FormBorderStyle` формы. В этом случае граница формы не позволит менять ее размеры.

Коды процедур-обработчиков приведены в примере 5.23.

Для каждой кнопки на форме нужно создать обработчик события `Click`.

Обработчики событий для всех цифровых кнопок будут идентичны.

Обработчики для кнопок арифметических действий будут отличаться

только значением запоминаемой операции.

Основные вычисления происходят в обработчике кнопки «•». Преобразуем в число `n2` значение поля `Edit` и выполняем арифметическую операцию в зависимости от значения переменной `znak`. После этого обнуляем переменные.

В обработчике кнопки «С» (от англ. `clear` — очистить) происходит обнуление переменных и очистка поля `Edit`.

Созданный калькулятор имеет большое количество ограничений в своей работе, поскольку рассчитан на вычисления только с натуральными числами.

Пример 5.23. Продолжение.

```
procedure Form1.button16_Click
(sender: Object; e: EventArgs);
begin
  n2 := StrToInt (TextBox1.Text);
  case znak of
    '+': TextBox1.Text:= IntToStr (n1+n2);
    '-': TextBox1.Text:= IntToStr (n1-n2);
    '*': TextBox1.Text:= IntToStr (n1*n2);
    '/': TextBox1.Text:= IntToStr (n1 div n2);
  end;
  n1 := 0; n2 := 0;
  znak := ' ';
end;

procedure Form1.button15_Click
(sender: Object; e: EventArgs);
begin
  TextBox1.Clear;
  n1 := 0; n2 := 0;
  znak := ' ';
end;
```



Упражнения

- 1 Дополните проект Блокнот следующими возможностями:
 1. Пункт меню Сохранить файл заменить двумя: Сохранить и Сохранить как...
 2. Добавить Пункт меню Переносить по словам.
 3. Добавьте возможность управления полосами прокрутки: установить горизонтальную, вертикальную, обе.
 - 4*. Добавьте следующие диалоги: Параметры страницы, Печать, Поиск, Замена и соответствующие пункты в меню.
 5. Добавьте контекстное меню для управления компонентом Мемо. Команды контекстного меню, дублирующие команды основного меню, должны вызывать те же обработчики.
 6. Предложите свои возможности.
- 2 Для проекта Графический редактор добавьте следующие возможности:
 1. Рисовать отрезки, овалы и прямоугольники.
 2. Предложите свои возможности.
- 3 Для проекта Калькулятор добавьте следующие возможности:
 1. Возможность работы с вещественными числами.
 2. Возможность вычислять значения функций: извлечение квадратного корня, тригонометрические функции (для градусов и радиан) и др.
 - 3*. Возможность перевода чисел в другие системы счисления.