**ЛАБОРАТОРНАЯ РАБОТА № 1.**

Тема. **Основы объектно-ориентированного программирования. Система визуального программирования Borland Delphi. Создание интерфейса программы. Настройка свойств формы (окна).**

Основные вопросы:

* основные понятия объектно–ориентированного программирования – ООП;
* структура проекта в Delphi;
* создание формы оконного приложения, задание свойств формы с помощью Инспектора объектов и программным путѐм с использованием процедуры OnActivate;
* применение основных свойств формы для практического использования в разработке интерфейса программы.

**Теоретическая часть**.

**Основные понятия объектно-ориентированного программирования. Классы, объекты, методы, свойства.**

Структурное программирование – написание программы с использованием нескольких основных структурных единиц: линейная структура, ветвление, циклы. Для укрупнения структурных единиц программы используются подпрограммы (в Pascal процедуры и функции). Написание и отладка программы происходит сверху вниз.

К середине 90-х годов сформировался новый подход в программировании – объектно-ориентированный. Программа рассматривается не только как последовательность выполняемых команд, но и как набор объектов, у каждого из которых имеются свои заранее определенные свойства. С каждым объектом программы предусмотрены заранее определенные действия – методы.

Объектно-ориентированное программирование (ООП) — это методика разработки программ, в основе которой лежит понятие объект. Объект — это некоторая структура, соответствующая объекту реального мира, его поведению. Задача, решаемая с использованием методики ООП, описывается в терминах объектов и операций над ними, а программа при таком подходе представляет собой набор объектов и связей между ними.

Преимущество объектного подхода в программировании: логическая простота построения программы. Ко всему прочему, программист практически не ограничен в своих действиях. Он может разрабатывать объекты любого назначения и с любыми свойствами.

Основные понятия ООП:

* Класс – это сложная структура, включающая, помимо описания данных, описание процедур и функций, которые могут быть выполнены над представителем класса – объектом. В Pascal классу соответствует тип (type) данных, создаваемых программистом, однако в описании типа не предусмотрено описание действий над объектами данного типа.
* Объект – переменная данного класса.
* Методы класса (процедуры и функции, объявление которых включено в описание класса) выполняют действия над объектами класса. Для того чтобы метод был выполнен, необходимо указать имя объекта и имя метода, отделив одно имя от другого точкой.
* Свойства объекта – одна из характеристик объекта. Каждое свойство объекта имеет своѐ значение. Любое свойство объекта – характеристика объекта, задаваемое в классе объектов. Сам класс содержит описание возможных значений каждого свойства. Само значение свойства задается в программе.

Пример 1. Схема класса «БИБЛИОТЕКА»

*Объекты класса* – множество всех книг библиотеки.

*Методы класса* – всевозможные предусмотренные атрибуты книг:

* автор;
* наименование;
* издательство;
* год издания;
* количество страниц;
* жанр;
* и т.д.

*Свойства объектов данного* *класса* - всевозможные предусмотренные действия с книгами:

* прием книги в библиотеку;
* выдача книги читателю;
* прием книги от читателя;
* списание книги.
* и т.д.

Объектно-ориентированный язык программирования характеризуется тремя основными свойствами: инкапсуляция; наследование; полиморфизм.

*Инкапсуляция.* Комбинирование записей с процедурами и функциями, манипулирующими полями этих записей, формирует новый тип данных – объект. Кроме того, инкапсуляция это скрытие внутренней структуры объекта за интерфейсом. То есть извне из всего объекта виден один интерфейс. Пользователю не обязательно знать, как устроена машина, чтобы на ней ездить.

Класс может быть использован любым другим классом через интерфейс и никакой другой класс (если это особо не оговорено модификатором доступа «публичный») не может получить доступ к внутренней структуре объекта.

Этим термином обозначают скрытие внутреннего устройства объекта. Объект рассматривается как – черный ящик. Известны свойства объекта, то есть доступные извне переменные. Известны методы объекта, то есть доступные извне функции. Но как устроены эти функции, по каким алгоритмам они работают, программисту не сообщается.

Программист, пользователь объекта, не знает и о том, есть ли у объекта дополнительные внутренние вспомогательные функции и переменные и как они связаны с доступными свойствами и методами.

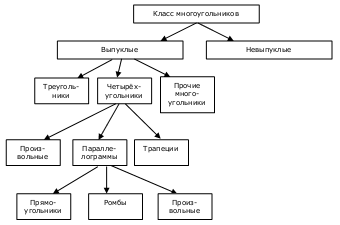
*Наследование*. При создании нового класса объектов программист задает тип элементов этого класса (свойства) и функции (методы), выполняемые над объектами этого класса. Однако объекты данного класса могут быть использованы в виде элементов другого класса, который является в иерархии классов подклассом (дочерним). Ранее определенный класс является родительским для дочернего.

Все свойства родительского класса, его методы автоматически наследуются всеми объектами дочернего класса. Наследование – уточнение структуры родительского класса. Наследующий класс может дополнять родительский методами и полями, сохраняя те, которые унаследовал от него.

*Полиморфизм*. При разработке подпрограмм иногда возникает необходимость создания функций, выполняющих однотипные действия с различными типами данных, например, нахождение модуля числа. Для разных типов числовых данных (целых и вещественных) в языках программирования, не являющихся объектно-ориентированными, необходимо каждой из таких функций задать разные имена. В объектно-ориентированных языках программирования можно разным функциям задавать одинаковые имена, но при условии, что каждая из таких функций будет иметь параметры разного типа или отличаться числом параметров. В этом и состоит суть полиморфизма.

При вызове таких функций из основной программы (модуля) будет вызвана соответствующая функция в зависимости от числа параметров или от их типа. Полиморфизм или перегрузка – по сути, объединение различных функций (методов) с разными входными параметрами под одним именем. Особенно актуально если действия функций аналогичны, и мы хотим, чтобы однородные действия были видны под одним именем. Хороший пример – оператор «+» понимаемый как сложение для различных типов объектов оператор записывается одинаково.

*Пример многоуровневой классификации.*



Для Родительского класса и любого дочернего подкласса можно выделить характерные свойства, присущие объектам любого подкласса: число вершин (значением данного параметра будет количество вершин для объектов каждого подкласса), число сторон, площадь фигуры. Однако, для каждого подкласса можно выделить и характерные только для объектов данного подкласса свойства и методы.

Например, свойством объектов любого подкласса будет площадь фигуры, но методы еѐ вычисления будут разными. Действительно, формулы вычисления площади для объектов класса *Треугольники* и *Трапеции* разные.

**Введение. Основные компоненты интерфейса Borland Delphi.**

Среда программирования Borland Delphi является объектно-ориентированной средой. В состав языка Delphi (а, по сути, этим языком является Object Pascal) входят десятки классов с множеством методов и свойств. Все эти классы созданы программистами Borland. Для решения большей части задач по созданию программ этих классов вполне достаточно. По этой причине Delphi является самой эффективной средой программирования с точки зрения скорости создания программы. Однако с точки зрения скорости выполнения программы, созданной на Delphi, это далеко не так. Удобство работы программиста происходит за счет скорости выполнения программы, созданной программистом.

Основным элементом интерфейса является палитра компонентов (рис 1), представленная в виде древовидной структуры. Каждая ветвь содержит различные элементы программы. Элементы (объекты) программы могут быть визуальными (видимыми при прогоне программы) и невизуальным (невидимыми при прогоне программы).

*Дерево объектов* (*структура*, рис. 1) содержит ярлыки всех компонентов программы, как визуальных, так и невизуальных. В дереве объектов можно выделять объекты и управлять ими (копировать, перемещать, удалять). В дереве объектов они расположены в виде иерархической структуры, т.е. по подчиненности.

*Инспектор объектов* (рис 2) является одним из основных элементов настройки элементов программы. Он содержит вкладки со свойствами (*Properties*) и методами (*Events*) текущего объекта. Данная панель является динамической, т.е. еѐ состав зависит от выделенного объекта. Некоторые свойства объектов являются составными. Слева от имени такого свойства расположен символ «+».

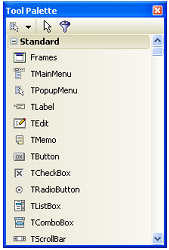
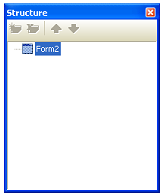
 

Рис.1. Палитра компонентов Delphi и дерево объектов проекта

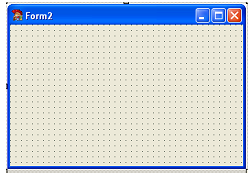
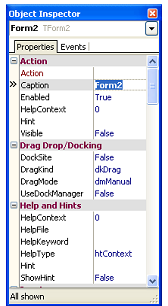


Рис. 2. Инспектор объектов проекта и окно проекта (формы)

Окно с программным кодом (рис.3) данной формы содержит процедуры всех объектов (как визуальных, так и невизуальных), относящихся к данной форме. Код пустой формы создается автоматически и называется модулем.

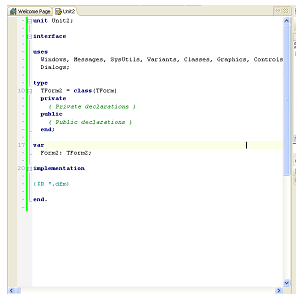


Рис. 3. Окно с программным кодом

Любой модуль состоит из стандартных секций:

unit Unit1;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs;

type

TForm1 = class(TForm)

private

{ Private declarations }

public

{ Public declarations }

end;

var

Form1: TForm1;

implementation

{$R \*.dfm}

end.

Любой модуль начинается с заголовка

unit <имя модуля>;

Секция интерфейсных объявлений содержит описания программных элементов. Начинается секция служебным словом interface.

Программные элементы – типы, классы, процедуры и функции. Секция реализации начинается служебным словом implementation и содержит описания действий процедур и функций.

Терминатором модуля является служебное слово end.

Для наглядности текста программы желательно при написании текста процедур и функций пояснять их с помощью комментариев. Он может быть однострочным – // текст комментария – или многострочным – {текст комментария}.

**Сохранение проекта**

Проект – множество файлов, связанных одной программой. Желательно для каждого проекта выделять отдельную папку. Основной файл проекта имеет расширение DPR. В окне кода программы он, как правило, не отображается. Является текстовым файлом и имеет типовой вид:

program Project1;

uses

Forms,

Unit1 in 'Unit1.pas' {Form1};

{$R \*.res}

begin

Application.Initialize;

Application.CreateForm(TForm1, Form1);

Application.Run;

end.

Файлы модулей имеют тип PAS. После компиляции для каждого модуля создается файл с расширением DFM. Файлы важны для проекта и удаление одного из них приведет к невозможности повторной компиляции проекта. Файл DFM является текстовым и имеет вид:

object Form1: TForm1

Left = 192

Top = 165

Width = 696

Height = 480

Caption = 'Form1'

Color = clBtnFace

Font.Charset = DEFAULT\_CHARSET

Font.Color = clWindowText

Font.Height = -11

Font.Name = 'MS Sans Serif'

Font.Style = []

OldCreateOrder = False

PixelsPerInch = 96

TextHeight = 13

end

При каждой повторной компиляции проекта создается заново файл ресурсов с расширением RES. В нѐм хранится дополнительная служебная информация о проекте – значок программы, номер версии программы и т.д.

Файлы настройки – файлы с настройкой компилятора (CFG), проекта (DOF), среды Delphi (DSK). Редактировать их нежелательно. Исполнимый файл проекта (EXE) создается после компиляции проекта.

**Форма и её основные свойства**

Форма – это окно программы. В неѐ можно добавлять визуальные компоненты. Для настройки свойств окна используются свойства формы. Это можно делать на стадии конструирования интерфейса в Инспекторе объектов во вкладке Свойства (Properties) или программно в окне с кодом проекта.

**Основные свойства формы.**

*Caption* – заголовок окна.

BorderStyle – определяет стиль рамки (границы) окна и его стиль. Фактически данное свойство определяет тип окна – обычное программное или диалоговое.

Форма может быть контейнером для размещения на ней дочерних компонентов. Однако и для самой формы предусмотрено множество стандартных событий, наступление которых может привести к изменению свойств самой формы или еѐ дочерних элементов. Событие – это процедура, код которой выполняется при наступлении указанного события. Для начала использования какого–либо события формы или дальнейшего редактирования программного кода для этого события можно в Инспекторе объектов во вкладке

События дважды щѐлкнуть в строке нужного события во втором столбце. При первом использовании этого приѐма для любого события создаѐтся процедура, программный код которой разработчик заполняет самостоятельно.

При каждом повторном использовании этого приѐма, т.е. при двойном щелчке на имени процедуры в строке события, открывается программный код для просмотра и редактирования.

**Практическая часть**

**Ход работы.**

1. Создать новый проект. Новый проект создается автоматически при запуске Delphi или через меню

***Файл → Новый → Приложение (File → New → VCL Form Application – Delphi for Win 32).***

1. В *Инспекторе объектов* во вкладке Свойства (Properties) задать следующие свойства формы (окна):

* *Caption* – Лабораторная работа 1 – заголовок окна;
* *Height* – 600 – высота окна в пикселях. Данное свойство может быть изменено при конструировании;
* формы графически;
* Width – 450 – ширина окна в пикселях;
* Color – clMoneyGreen – цвет активного окна.

1. Запустить программу на исполнение: меню *Запуск → Запуск* (*Run → Run*) или клавиша *F9*. В окне программы попробовать изменить его размер с помощью кнопок управления окном и графическим способом. Завершить работу программы.

1. В Инспекторе объектов последовательно задать в свойствах формы следующее из свойств – BorderStyle – вид оборки окна. Данное свойство позволяет указать стиль окна – обычное программное или диалоговое.

Значение по умолчанию

– bsSizeable – обычное окно. Возможные другие значения:

* *bsDialog* – диалоговое. Окно не масштабируется, из кнопок управления окном – только кнопка *Закрыть*;
* *bsNone* – окно не имеет заголовка, не масштабируется, для его закрытия можно использовать только контекстное меню в Панели задач на ярлыке этого приложения;
* *bsSingle* – рамка толщиной в 1 пиксел, такое окно не может изменять свои размеры графически, но с помощью кнопок управления оно может быть масштабировано;
* *bsSizeToolWin* – диалоговое окно с уменьшенным заголовком, окно может быть масштабировано;
* *bsToolWindow* – подобно bsSingle, но с уменьшенным по высоте заголовком.

Запустить программу на выполнение при каждом изменении данного параметра и попробовать графически и с помощью кнопок управления окном изменить масштаб окна.

1. Для данной формы создать процедуру, которая выполняется при активизации формы. Для этого в *Инспекторе объектов* раскрыть вкладку *События* (*Events*) и для события *OnActivate* дважды щелкнуть мышью в пустое поле справа. Автоматически в списке событий будет создана процедура *FormActivate* и откроется окно программного кода для данного модуля. Созданная процедура будет иметь вид:

**procedure TForm1.FormActivate(Sender: TObject);**

**begin**

**end;**

Весь необходимый программный код этой процедуры должен быть записан между служебными словами begin … end;

1. Теперь можно программным путем задать некоторые свойства формы. Для ввода команд используйте визуальные подсказки, выдаваемые редактором кода Delphi.

Настройте свойство для вывода заголовка окна при запуске программы – свойство *Caption*. Для этого в теле процедуры ввести следующую команду с параметрами:

**Form1.Caption := 'Лабораторная работа № 1';**

В данном случае *Form1* – имя объекта, *Caption* – свойство объекта. Символ точки является разделителем между именем объекта и его свойством. Обычно после ввода точки редактор Delphi выводит подсказку по всем доступным свойствам (*Properties*), процедурам (*Procedure*) и функциям (*Function*), которые можно выбрать из списка после ввода нескольких первых символов имени свойства, процедуры и функции. Кроме того, в подсказке всегда отображается для свойств и функций тип этого объекта. Например, для свойств *Width* и *Height* выводится подсказка *Integer*. Это означает, что значением данного свойства может быть только число целого типа.

Задайте дополнительно к свойству заголовка в созданную процедуру и введите команды для настройки свойств:

* ширины окна – 700 пикселей;
* высоты окна – 500 пикселей;
* цвет окна – clCream.

Проверьте работоспособность проекта, запустив программу на выполнение.

1. К установленным ранее свойствам формы необходимо добавить дополнительно свойство:

* *всплывающая подсказка* (свойство *Hint*) при наведении на него указателя мыши должна появиться соответствующая текстовая строка. Например:

**Form1.Hint := 'Эта подсказка по форме!';**

Проверить, что, несмотря на установку данного свойства при наведении указателя мыши на окно программы, подсказка всплывать не будет. Для того чтобы «разрешить» подсказке появляться, необходимо задать свойство *ShowHint*:

**Form1.ShowHint := True;**

Данное свойство является логическим и может принимать значение либо *True*, либо *False* (по умолчанию).

1. Для оформления текста программы используйте комментарии – пояснительный текст, который может быть однострочным или многострочным.

Однострочный комментарий начинается символом двойного слеша **//** далее текст комментария. Свойство. Многострочным комментарием может быть любой текст в несколько строк, заключѐнный в символы фигурных скобок **{}**.

Комментарии позволяют пояснять фрагменты программы или на время выключать из программного кода часть команд.

Еще одним вариантом многострочного комментария может быть любой произвольный текст, заключенный в парные символы **(\* \*)**.

1. Сохранить данный проект необходимо в отдельной папке на диске. Для этого необходимо вызвать через меню ***File → Save Project As…*** и сохранить файлы в заранее подготовленной папке, назовем ее, к примеру «Delphi files».

При сохранении файла–модуля (файл с типом *pas*) задайте ему имя, например Module1. При сохранении файла–проекта (файл с типом *dpr*) задайте ему – Lab1.