КИПР

ТУСУР

Кафедра конструирования
и производства радиоаппаратуры

Основы создания графического программного интерфейса в среде Lazarus

Томск, 2017

Кобрин Юрий Павлович

Основы создания графического программного интерфейса в среде Lazarus. Методические указания к лабораторной работе и по организации самостоятельной работы по дисциплинам «Информатика» и «Информационные технологии» для студентов очного и заочного обучения специальностей 211000.62 (бакалавриат) и 162107.65 «Информатика и информационные технологии» (специалитет). - Томск: Томский государственный университет систем управления и радиоэлектроники (ТУСУР), кафедра КИПР, 2017. – 29 с.

Lazarus — открытая бесплатная кроссплатформенная среда визуальной разработки программного обеспечения для компилятора *Free Pascal*, максимально приближённая к *Delphi*. Методические указания посвящены освоению основ объектно-ориентированного программирования (ООП) в интегрированной среде *Lazarus*, а также с особенностями технологии визуального программирования программ с графическим интерфейсом. Для создания графического интерфейса приложения в Lazarus используют готовые компоненты, значки которых находятся на панели компонентов. После того как пользователь помещает компонент на форме, программный код для него генерируется автоматически. Вручную остаётся запрограммировать только те действия, которые будет выполнять это приложение.

Методические указания предназначены для помощи в подготовке бакалавров и магистрантов в Информатике, выполнения курсовых и дипломных проектов.

Оглавление

1 Цели и задачи работы

1) Приобретение начальных навыков работы в интегрированной среде объектно-ориентированного программирования (ООП) *Lazarus*.

2) Освоение работы с компонентами ***TForm*** (Форма), ***TLabel*** (Надпись) и ***TButton*** (Командная кнопка).

В ходе выполнения работы следует научиться:

- работать в интегрированной среде Lazarus;
- усвоить правила использования основных типов объектов в *Lazarus*;
- освоить приёмы создания простейших графических интерфейсов в среде Lazarus.

2 Порядок выполнения работы

Перед выполнением этой работы следует:

1) Ознакомиться с теоретической частью (раздел «Основы объектно-ориентированного программирования (ООП)»). В качестве дополнительной литературы использовать [1,2,3,4,5,6,7,8].

2) Изучить правила работы с интегрированной средой программирования Lazarus), а также ознакомиться со структурой программ в системе Lazarus.

3) Войти в свой личный каталог, загрузить и настроить систему программирования ***Lazarus***.

Создать **Шаблон для собственных разрабатываемых приложений** (раздел **«**приложения.

4) Создание шаблона для разрабатываемых программ**»)** и сохранить его в личном каталоге на сервере (имя каталога - фамилия студента).

5) Добиться, чтобы при компиляции **Приложения-шаблона** не было сообщений об ошибках.

6) Ответить на контрольные вопросы.

7) Оформить отчёт и защитить его у преподавателя.

3 Отчётность

Отчёт должен быть выполнен в соответствии с [9] и состоять из следующих разделов:

1) Цель работы.
2) Откомпилированный текст разработанного приложения (в электронном виде).
3) Ответы на контрольные вопросы.
4) Выводы.

Для получения зачёта студент должен:

- уметь отвечать на контрольные вопросы;
- показать, что разработанное приложение работает правильно;
- уметь пояснить назначение элементов разработанного приложения;
- продемонстрировать навыки работы в интегрированной среде ***Lazarus***.

4 Контрольные вопросы

1) Дайте определение объектно-ориентированному программированию. Какими достоинствами и недостатками обладает данная технология?

2) Что собой представляют классы, объекты, интерфейсы?

3) В чем заключается сущность принципов инкапсуляции, полиморфизма, наследования?

4) Какие принципы положены в основу технологии ООП?

5) Что собой представляет визуальное проектирование интерфейса?

6) Перечислите основные элементы пользовательского интерфейса Lazarus.

7) Какие функции выполняет палитра компонентов в Lazarus?

8) Для чего используется Инспектор объектов?

9) Разъясните назначение основных файлов, входящих в проект Lazarus.

10) Из каких компонентов состоит IDE Lazarus?

11) В чем отличие невизуальных от визуальных компонентов?

12) С помощью какого свойства меняется заголовок у компонента?

13) Проекты сохраняются в одном файле или нет?

14) Какие команды текстового редактора Вы знаете?

15) Что такое блок текста программы и как его выделить? Какие операции с блоками вы знаете?

16) Как осуществить контекстный поиск и замену?

17) Что представляет собой строка комментариев?

18) Объясните назначение интерфейсной секции (*interface*), секции реализации (*implementation*) и секции инициализации модуля *unit*.

19) Верно ли, что в модуле *unit*:

a) количество подпрограмм в интерфейсной секции должно совпадать с количеством подпрограмм в секции реализации;

b) количество подпрограмм в интерфейсной секции может быть меньше количества подпрограмм в секции реализации;

c) количество подпрограмм в интерфейсной секции может быть больше количества подпрограмм в секции реализации.

20) Покажите на примерах, как осуществляется:

a) запуск и выход из интегрированной среды;

b) загрузка и сохранение файлов проекта;

c) компиляция и запуск программы.

5 Основы объектно-ориентированного программирования (ООП)

5.1 Основные понятия ООП

Объектно-ориентированное программирование (ООП) представляет собой технологию разработки программ с использованием объектов. Основная идея ООП заключается в объединении данных, с которыми работает программа и подпрограмм[1], которые эти данные обрабатывают, в единое целое – *объект*. **Объект** — это любая часть окружающей действительности (предмет, процесс, явление), воспринимаемая человеком как единое целое.

В языках программирования понятие объекта реализовано как упрощённое, идеализированное описание реальной сущности предметной области. Это совокупность **свойств** (структур данных, характерных для данного объекта), **методов** их обработки (подпрограмм изменения их свойств) и **событий**, на которые данный объект может реагировать и, которые приводят, как правило, к изменению свойств объекта. Для каждого объекта существует определённый набор событий. Например, командная кнопка (компонент *TButton*) поддерживает события *Click* (щелчок), *MouseDown* (нажата кнопка мыши), *MouseUp* (отпущена кнопка мыши).

---

[1] **Подпрограмма -** это отдельная поименованная функционально независимая часть компьютерной программы, содержащая описание определённого набора действий. Подпрограмма может быть многократно вызвана из разных частей программы с разными входными данными, избавляя от необходимости многократно повторять в тексте программы аналогичные фрагменты. В языках программирования используется два типа подпрограмм: *процедуры* и *функции*.

При решении задачи из какой-либо предметной области, программист выделяет отдельные объекты, исходя из особенностей задачи. Этот процесс называется **объектной декомпозицией**. Объекты состоят из **данных**, описывающих свойства этих объектов и методов (подпрограмм, обрабатывающих эти данные).

Например, при разработке базы данных студентов университета, можно выделить объект с **именем** «Студент». Данными (**свойствами**) для этого объекта могут выступать: фамилия и имя студента, адрес, курс, группа, его оценки и т.п. При этом можно определить некоторые подпрограммы (**методы**), которые выполняют определённые действия над объектами, устанавливающие, как объект взаимодействует с окружающим миром. Например, процедуру вычисления средней оценки за семестр (эту процедуру можно в дальнейшем использовать для установления размера стипендии), процедуру перевода с курса на курс, процедуру отчисления из университета и т.д.

Для работы с объектами в *Lazarus* введено понятие *класса*. **Класс** (англ. *class*) - это абстрактное понятие, сравнимое с понятием категория в его обычном смысле. Это структура, включающая в себя описание данных, процедуры и функции, которые могут быть выполнены над объектом.

С другой стороны, **Класс** - это шаблон, на основе которого может быть создан конкретный программный объект (англ. *object*), он описывает свойства и методы (англ. *methods*), определяющие поведение объектов этого класса. Класс образуют объекты с одинаковыми свойствами, то есть с одинаковыми наборами переменных состояния и методов. Каждый конкретный объект, имеющий структуру этого класса, называется **экземпляром класса**. В программировании отношения объекта и класса можно сравнить с описанием переменной, где сама переменная (объект) является экземпляром какого-либо типа данных (класса).

Все данные объекта хранятся в его полях. **Поля** с данными - это обычные переменные. Доступ к полям осуществляется по их имени. Обычно тип данных каждого поля задаётся в описании класса, членом которого является поле. Допускаются любые разрешённые в языке типы переменных, в том числе поле может иметь тип класса. С помощью полей описываются состояние объекта, а с помощью методов – поведение объекта.

Для доступа к полям данных в классе лучше использовать **свойства**. **Свойство** - это специфический способ для организации связи с данными класса. Свойства отображают состояние объекта (например, определяют его внешний вид) и описываются с помощью переменных (полей) классов (например, размер, шрифт, цвет и т.д.). Для работы со свойствами предусматривают два специальных метода - для записи и чтения значения поля. Преимущество свойства в том, что методы чтения и записи полностью скрыты. Это позволяет вносить изменения в код класса не изменяя использующий его внешний код.

Поведенческую сторону объекта определяют совокупность **событий**. В качестве обработчика события должна быть предусмотрена процедура, которая выполняет те или иные действия в ответ на наступление события. Другими словами, с помощью такой процедуры (обработчика события) реализуется реакция объекта на событие, например на щелчок мыши. При разработке программы следует определить необходимые свойства объектов в его приложении и написать обработчики тех событий, на которые должен реагировать тот или иной объект приложения.

Связь между основными понятиями изображена на Рис. 5.1.

Инспектор объектов позволяет определить обработчики событий, на которые должна реагировать форма или её компоненты.

Под **объектом** понимается конкретный экземпляр (сущность) класса. В программе *объект описывается как переменная типа класс.* Переменная типа класс содержит не сами данные, а ссылку на них, т.е. является указателем, а сам объект размещается в динамической памяти. Перед использованием объекта необходимо выделить для него память путём создания нового экземпляра класса или присвоением существующего экземпляра переменной типа класс. После завершения работы с этим экземпляром класса (объектом) память необходимо освободить.

В ООП существуют три основных понятия: *инкапсуляция, наследование и полиморфизм.*

Объединение данных и свойственных им методов их обработки (подпрограмм) в одно целое – объект, называется **инкапсуляцией** и является одним из важнейших принципов ООП. Внутреннее устройство объекта может быть достаточно сложным, но оно скрыто от чужих глаз. Для связи с внешним окружением используются лишь небольшие объёмы данных, причём количество и тип этих данных строго контролируется, что существенно повышает надёжность программы. **Принцип сокрытия информации** – один из важнейших принципов ООП. Скрывая внутреннее устройство класса, мы абстрагируемся от ненужных деталей. Кроме того, это позволяет защитить члены класса от несанкционированного доступа извне.

Наследование - предусматривает создание новых классов на базе существующих и позволяет классу потомку иметь (наследовать) все свойства класса - родителя. Если имеется класс, который Вас в принципе почти удовлетворяет, но чего-то в нём не хватает, то заново создавать новый класс не нужно. Достаточно создать класс на основе *существующего* и просто добавить в него недостающие члены (поля, методы и свойства). Это и есть наследование. Новый класс называется **наследником** (**потомком**, **дочерним классом)**. А старый класс называется **родительским классом** (**предком, базовым классом**).

Иерархия классов – набор классов, связанных отношением наследования.

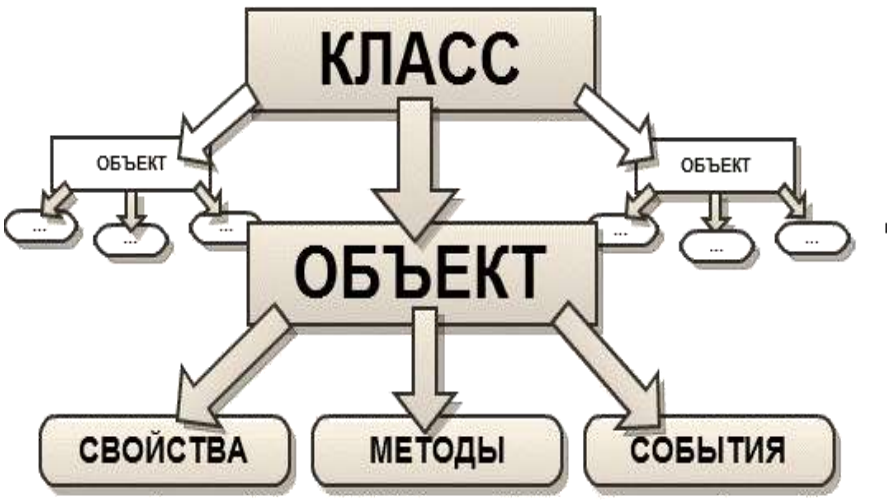


Рис. 5.1 - Связь между основными понятиями ООП

Под **полиморфизмом** понимается возможность одних и тех же функций или процедур по-разному обрабатывать данные, принадлежащие разным объектам. **Полиморфизм** означает, что рождённые объекты обладают информацией о том, какие методы они должны использовать в зависимости от того, в каком месте цепочки они находятся. Например, пусть у вас имеется объект «Геометрическая фигура» и пусть у неё имеется функция вычисления площади. Если необходимо вычислить площадь прямоугольника, то функция будет вычислять её по одному алгоритму, а если это треугольник, то её площадь функция будет вычислять по совсем другому алгоритму.

Прямое обращение к полям не рекомендуется, так как может стать источником ошибок. **Сообщение** (англ. *message*) - это практически тоже самое, что и вызов функций в обычном программировании. В ООП обычно употребляется выражение «послать сообщение» какому-либо объекту. Мы не можем напрямую изменить состояние объекта и должны как бы послать сообщение объекту, что мы хотим так и так изменить его состояние. Объект сам меняет своё состояние, а мы только его просим об этом посылая сообщения.

В терминах библиотеки *Lazarus Component Library* (*LCL*) элемент графического интерфейса программы называют **компонентом**. Каждый компонент реализуется как объект соответствующего класса, который наследуется от некоторого базового класса. Компоненты бывают *видимыми* и *невидимыми*.

Объявление класса в общем случае производится следующим образом:

Type

<Имя класса>[2] ***= class***(<имя класса родителя>)

private

<Поля и методы, доступные только в пределах модуля>

<методы и переменные класса>

protected

<поля и методы, доступные только в классах-потомках>

public

<поля и методы, доступные из других модулей>

published

<поля и методы, видимые в инспекторе объектов>

end;

Имя класса родителя — имя класса, наследником которого является данный класс.

Поля, свойства и методы секции *public* не имеют ограничений на видимость. Они доступны из других функций и методов объектов как в данном модуле, так и во всех прочих, ссылающихся на него.

Поля, свойства и методы секции *private* доступны только в методах класса и в функциях, содержащихся в том же модуле, что и описываемый класс. Это позволяет полностью

---

[2] Имена классов принято начинать с буквы T (от слова *Type*), хотя это и не обязательно. В качестве имени класса можно использовать любой допустимый в *Lazarus* идентификатор

скрыть детали внутренней реализации класса. Вызов приватных методов осуществляется из публичных.

Публикуемый (*published*) — это раздел, содержащий свойства, которые пользователь может устанавливать на этапе проектирования и они доступны в любом модуле.

Защищённый (*protected*) — это раздел, содержащий поля и методы, которые доступны внутри класса, а также любым его классам-потомкам, в том числе и в других модулях.

При программировании с использованием классов программист должен решить, какие члены и методы должны быть объявлены публичными, а какие приватными.

5.2 Структура программ в системе Lazarus

Программа на *Lazarus* состоит из совокупности модулей.

Проект - это набор связанных файлов различного типа, который разрабатывает программист в Lazarus. В результате компиляции получается **исполняемая программа-приложение**, с которой может работать пользователь без помощи Lazarus.

Модуль - это автономно компилируемая программная единица исходного кода, включающая в себя различные компоненты раздела описаний (типы, константы, переменные, процедуры и функции), выполненная в виде файла с расширением **.pas*.

Может использоваться другими программами или модулями путём вызова его подпрограмм по именам. Автономно использоваться не может.

Текст модуля на языке программирования называется исходным модулем. После компиляции создаётся так называемый *объектный модуль*. **Объектный модуль** (англ. *object file* - объектный файл) - файл с промежуточным представлением отдельного модуля программы, полученный в результате обработки исходного кода компилятором. Это программа на машинном языке с неразрешёнными внешними ссылками. На этапе компоновки (сборки) с помощью программы **Редактор связей** (англ. *link editor, linker*) необходимые модули включаются в программу (компонуются) для обеспечения правильных вызовов функций и процедур. Например, главная программа вызывает некую процедуру из модуля A, в модуле B происходит вызов функции из модуля C и т.д. В итоге подобного процесса, называемого *разрешением связей,* собирается исполняемая программа.

Текст модуля мы можем видеть и редактировать в **Редакторе кода**. При создании окна создаётся два файла: **модуль** - файл ****.pas*** с исходным кодом, и файл ****.lfm***, в котором содержатся настройки используемых на форме компонентов.

Структура программы в среде *Lazarus* зависит от её типа. Тип выбирается в меню **Файл** [10] с помощью которого выполняют всевозможные работы с файлами проектов *Lazarus*. Команды для создания новых файлов:

1) ***Создать модуль*** (англ. *New Unit* - Создать новый модуль *Unit*, исходный код на *FreePascal*);

2) ***Создать форму*** (англ. *New Form* - Создать новую форму, то есть два файла: саму форму и соответствующий ей файл с *FreePascal*-кодом);

3) ***Создать...*** Открывает всплывающее окно (Рис. 5.2) в котором приведены различные типы проектов, которые можно создать.

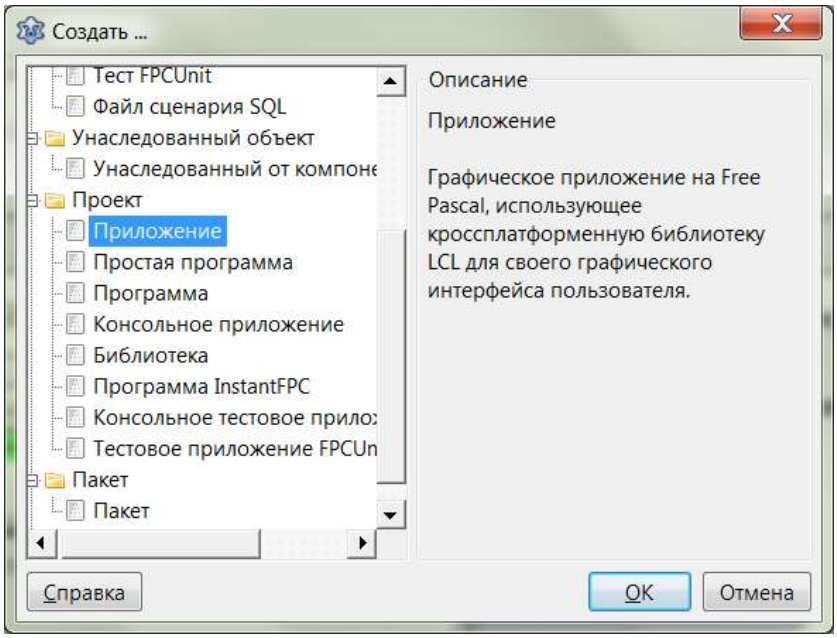


Рис. 5.2 - Всплывающее окно **Создать** в меню **Файлы**

Чаще всего применяемые типы проектов:

– ***Custom Program*** - пользовательская неграфическая программа на языке *FreePascal*. Это **консольное приложение** (*Console application*), т.е. программа, имитирующая работу в операционной системе *MS-DOS* (или в окне *DOS*), для которой устройством ввода является клавиатура, а устройством вывода — монитор, работающий в режиме отображения *символьной информации* (буквы, цифры и специальные знаки). Допустимо использование функций *Windows*.

– ***Application*** **-** графическое оконное приложение *Windows* на языке *FreePascal*.

Чтобы файлы разных проектов не смешивались, рекомендуется *для каждого проекта* создавать *свою папку*. Количество и типы файлов проекта зависят от вида программы и используемых режимов работы при конструировании программы.

Расширения и назначение важнейших типов файлов проекта Lazarus следующие:

- Исходный код проекта *Lazarus* (****.lpr***) (англ. ***l****azarus* ***p****roject,* ***lpr***) на языке *FreePascal.* Создаётся *Lazarus* автоматически.
- Информация о конфигурации проекта *Lazarus* ***(*.lpi***) (англ. ***l****azarus* ***p****roject* ***i****nformation,* ***lpi***). Создаётся *Lazarus* автоматически.
- Программный модуль *Lazarus* **(**.pas;*.pp*)** - исходный текст модуля на языке *FreePascal*.
- Описание **Форм** *Lazarus* ***(*.lfm;*.dfm***) (англ. *Lazarus form* ***p****roject,* ***lfm***) - файлы с данными о позициях, размерах и т.п., размещённых в форме компонентов и т.п. Создаётся *Lazarus* автоматически.
- Компилированный (****.compiled***) - содержит сведения о конфигурации проекта, нужные для формирования объектного файла. Создаётся *Lazarus* автоматически.

- Ассемблерный (***.ppu***), создаётся *Lazarus* автоматически при компиляции.
- Объектный код проекта (***.o***). (англ. ***o****bject*, ***o***), необходимый для сборки проекта. Создаётся *Lazarus* автоматически.
- Ресурсный файл (***.lrs***) -информация о курсорах, иконках и др. (англ. ***l****azarus* ***re****source,* ***lrs***)..
- Исполняемый файл программы (***.exe***) (англ. ***exe****cution* - выполнение, ***exe***). Создаётся *Lazarus* автоматически после успешной компиляции.
- Файл с «иконкой» проекта (***.ico**) – изображение в виде лапы гепарда, появляющееся в верхнем левом углу окна программы.

Автоматически создаются и резервные копии файлов, помещаемые во вложенную папку проекта *backup*.

При создании консольного приложения *Lazarus* автоматически создаёт шаблон программы. Код программы нужно заносить между блочными скобками *begin .. end*.

Любой модуль состоит из нескольких частей:

unit имя_модуля; //**Заголовок модуля** - за зарезервированным словом *unit* следует имя модуля и точка с запятой

interface /**Раздел описаний -** открывается служебным словом *interface* и описывают программные элементы - типы, классы, процедуры и функции:

- ***uses*** список_модулей;
- ***type*** список типов;
- ***const*** список_констант;
- ***var*** список_переменных;
- ***procedure*** имя_процедуры;
- ***function*** имя_функции;

implementation //**Раздел реализаций -** содержит *программный код*, реализующий механизм работы описанных программных элементов (тексты процедур обработки событий, процедуры и функции, созданные программистом)

end. //**Конец модуля.**

Модуль начинается со служебного слова $unit$ (блок, модуль), за которым следует имя модуля. В случае если данный модуль использует другие модули, после слова *interface* необходимо поместить служебное слово *uses* (использовать) и список используемых в программе модулей.

Интерфейсный раздел модуля начинается со служебного слова *interface*[3]. В этом разделе можно определять константы, типы данных, переменные, процедуры и функции, которые доступны для всех программ и модулей, использующих данный модуль. Глобальные переменные, помещённые в интерфейсной секции, могут быть использованы в основной программе.

Раздел реализации модуля начинается служебным словом *implementation* (реализация, ввод в эксплуатацию, внедрение). Этот раздел содержит программный код, реализующий механизм работы описанных программных элементов (тексты процедур обработки событий, процедуры и функции, созданные программистом). Описанные в секции интерфейса константы, типы данных, переменные, процедуры и функции являются *видимыми* в секции реализации. Все процедуры и функции, которые описаны в интерфейсной секции, описываются ещё раз в секции реализации, причём их заголовок должен быть точно таким же, как тот, который указан в секции интерфейса. В секции реализации могут находиться и свои описания, невидимые для программ и модулей, использующих данный модуль.

Всегда в проекте есть **главный модуль, которому** обычно дают имя ***Main*** (англ. *main* - главный). Для обращения к другим модулям следует придумать уникальные имена (свойство *Name* компонента TForm). Имена модулям разумно давать исходя из их назначения, например, *Optimization, Editor* и т.п.

С целью уменьшения размера основной программы готовые подпрограммы целесообразно оформлять в виде модуля. Со временем у Вас появится множество подобных «своих собственных» модулей-библиотек с подпрограммами.

Имя библиотечного модуля должно совпадать с именем файла, под которым хранится текст модуля на диске.

Исходный текст библиотечного модуля имеет расширение $*.pas$

$(<$ имя модуля $>.pas)$.

Можно посмотреть, какие файлы создал *Lazarus*, если выполнить последовательность команд **Главное меню** «**Сервис** ⇒ **Параметры**», и в ветке «**Окружение**» перейти в раздел «**Файловые фильтры**».

---

[3] **Интерфейс** (англ. *interface* — средство осуществления взаимного воздействия, взаимосвязи) — совокупность возможностей, способов и методов одновременного действия двух имеющих общее разграничение информационных систем, устройств или программ.

5.3 Проектирования приложение с помощью Формы

С помощью формы Вы проектируете вид *окна Вашего приложения*, которое увидит пользователь во время выполнения этого приложения. При создании нового проекта появляется пустая форма (Рис. 5.3) - это то, что Вы увидите в начале проектирования интерфейса будущего приложения. Форме соответствует класс, производный от базового класса ***TForm***.

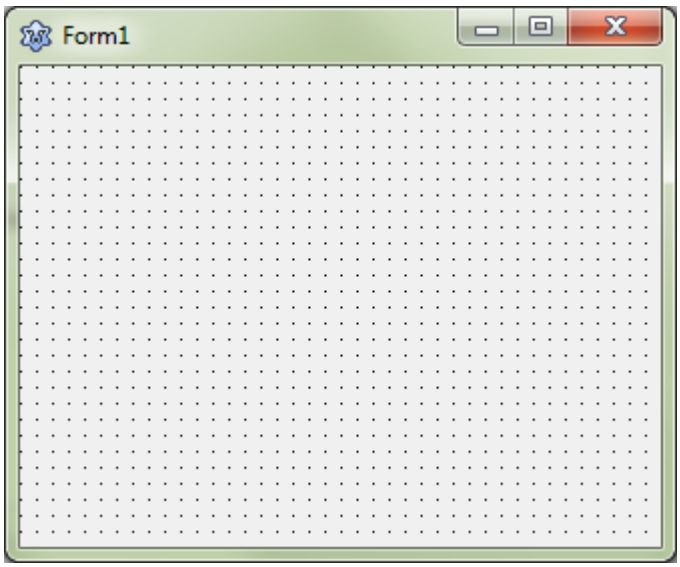


Рис. 5.3 - Окно формы

Для удобства, рабочая область окна формы заполнена точками координатной сетки, видимыми только на этапе проектирования приложения. Вначале окно с формой содержит только стандартные элементы - строку заголовка и кнопки развёртывания, свёртывания и закрытия. Форму следует дополнить разнообразными интерфейсными *компонентами*[4] из *панели компонентов*[5] lazarus, создавая тем самым внешний вид своей программы. Окно инспектора объектов располагается слева от окна редактирования.

Для запуска каких-то действий или команд в форму добавляют элементы управления: кнопки, меню, строки ввода, полосы прокрутки и т.п.

Щелчок по кнопке мышью по элементу управления, нажатие клавиши клавиатуры и т.п. вызывает некоторое событие, инициирующее выполнение каких-либо действий в ответ на наступление события. Реакция объекта на событие, реализуется **обработчиком события** - процедурой, которая выполняет действия, соответствующие этому событию.

Информация об выделенном объекте приложения содержится в **Окне инспектора объектов,** имеющем три вкладки: *Свойства, События, Избранное*. Эти вкладки используются для редактирования свойств объекта и описания событий, на которые будет реагировать данный объект.

Таким образом, проектирование приложения в Lazarus сводится к тому, чтобы разместить в форме необходимые объекты управления, задать их свойства и написать обработчики тех событий, на которые должен реагировать этот объект приложения.

---

[4] *Компонент* - это некоторый функциональный элемент интерфейса, размещаемый на форме (окна, кнопки, переключатели, поля ввода и т.п.), обладающий определёнными свойствами.

[5] Элементы графического интерфейса, а также множество других в среде Lazarus размещены в специальной библиотеке LCL (*Lazarus Component Library*), предоставляющей целую палитру визуальных компонентов.

6 Создание шаблона для разрабатываемых программ

6.1 Назначение шаблона разрабатываемых программ

Шаблон проектирования (англ. *Design pattern*) в разработке программного обеспечения — повторимая конструкция, позволяющая снизить сложность разработки за счёт использования готовых абстракций для решения некоторых повторяющихся задач. С подобными элементарными шаблонами (заготовками программ) сталкиваются в своей повседневной деятельности практически все разработчики программного обеспечения. Хороший правильный шаблон проектирования позволяет, отыскав удачное решение, пользоваться им снова и снова, снижая количество ошибок и время разработки программ. Разумно постоянно модернизировать, улучшать шаблон, используя приобретаемый опыт.

Начиная разработку нового программного проекта, следует загрузить шаблон и сохранить его под новым именем. Модифицировать всегда легче и быстрее, чем создавать всё заново!

6.2 Запустить интегрированную среду Lazarus

Откройте *Lazarus*, если он у вас закрыт, или закройте старый проект и начните новый. Для запуска интегрированной среды (IDE) *Lazarus* сделайте двойной щелчок ЛК мышки на значке  . На экране появятся несколько окон *Windows* (Рис. 6.1).

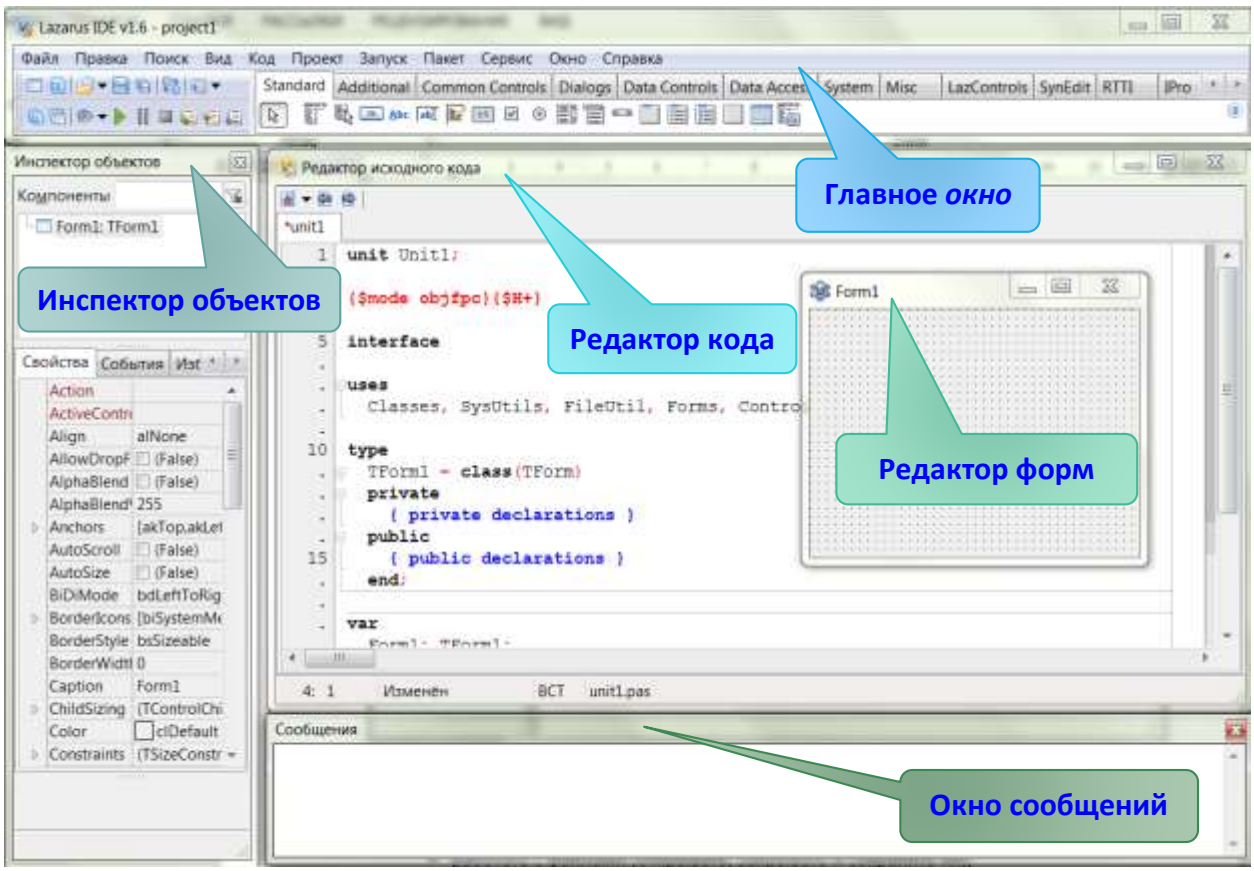


Рис. 6.1 - Окна интегрированной среды разработки программ ***Lazarus***

Главное окно *Lazarus* управляет проектом создаваемой программы. Оно размещено вверху и содержит:

1) **Главное меню,** позволяющее активизировать **выпадающие меню** с перечнем сгруппированных по их предназначению команд (работа с файлами, отладка, компиляция, справка, запуск различных вспомогательных утилит и т.п.).

2) **Панель инструментов с** наиболее часто используемыми командами **Главного меню** (команды сохранения и открытия, запуска, останова, трассировки и т.п.).

3) **Палитру (панель) компонентов –** содержащую множество вкладок, позволяющих выбрать стандартные компоненты, используемые при конструировании формы – будущего окна приложения.

Инспектор Объектов (*Object Inspector*), расположенный слева, содержит:

1) В верхней части - **Дерево объектов**, в котором в иерархической виде располагаются все объекты, используемые в текущей Форме.
2) В нижней части **Инспектора Объектов** - **вкладки** *Свойства, События, Избранное, Ограничения*, в которых настраивают различные параметры текущего компонента.

В окне **Редактора Форм** (окно будущего приложения с именем ***Form1*** по умолчанию) осуществляется редактирование формы - положения и размеров компонентов, размещённых на этой форме.

Занимающий большую часть экрана **Редактор исходного кода** *Lazarus* (*Lazarus Source Editor*) содержит исходный код программы, который мы должны создать.

В **Окно сообщений** (Messages) выводятся различные сообщения: о найденных ошибках, о завершении компиляции, о наличии объявленных, но неиспользуемых переменных и т.п.

6.3 Создать новый проект

Новый проект создаётся с помощью последовательности команд **Проект ⇒ Создать проект**… В появившемся диалоговом окне (Рис. 6.2) следует выбрать опцию **Приложение** и нажать кнопку **ОК**.

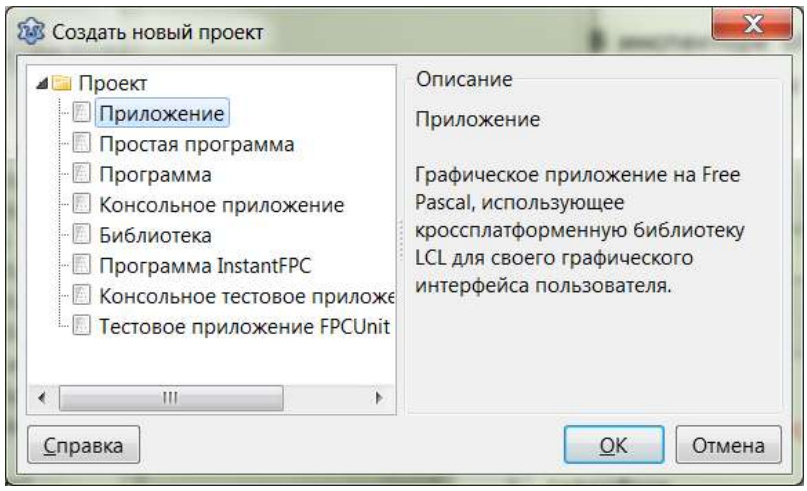


Рис. 6.2 - Диалоговое окно **Создать новый проект**

6.4 Сохранить созданный проект

Сохранить созданный проект можно с помощью последовательности команд **Проект ⇒ Сохранить проект как…** В открывшемся окне **Сохранить проект (**Рис. 6.3 **необходимо создать новую папку с именем *Шаблон Lazarus* для шаблона файлов** Ваших **проектов (проект будет содержать** несколько **файлов), открыть её, набрать в строке Имя файла, например, *Template*[6] и щёлкнуть по кнопке Сохранить.** В результате мы сохраним файл ***Template***, содержащий сведения о проекте. Заметим, что в Lazarus (как и в Pascal) строчные и заглавные буквы не различаются. Тем не менее для удобочитаемости кода лучше приучиться использовать заглавные буквы, чтобы выделять названия.

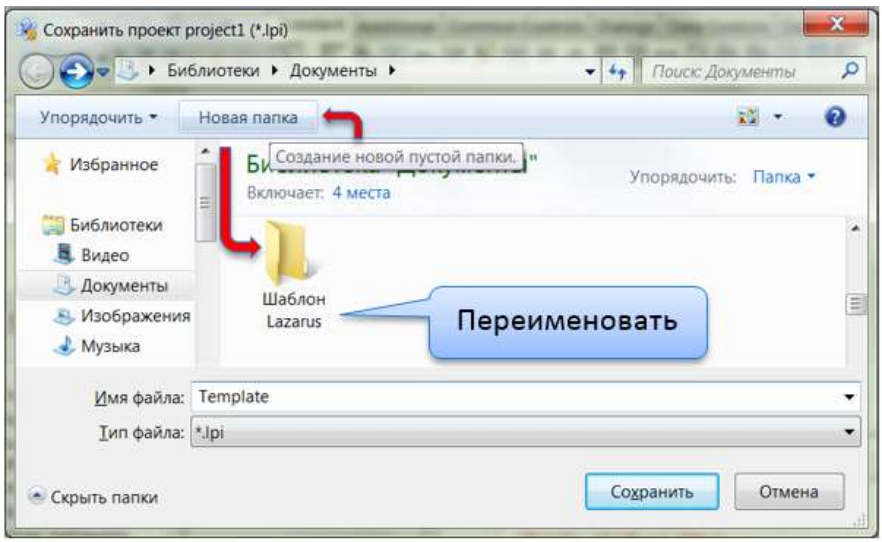


Рис. 6.3 - Диалоговое окно **Сохранить проект**

Автоматически откроется диалоговое окно **Сохранить** для сохранения программного кода проекта, которое по умолчанию имеет заголовок ***Unit1***. Дадим ему имя, например, ***Main*** или ***unitivanov*** (фамилию, конечно, свою!) (файл ***unitivanov.pas***), в котором также необходимо щёлкнуть по кнопке **Сохранить**.

Кроме этих двух файлов в папке проекта создаётся автоматически ещё несколько файлов, в том числе – ***unitivanov.lfm***, который представляет собой файл с полными данными о проектировщике формы – о позициях, размерах и т.п. размещённых в форме компонентов. В папке проекта теперь должны содержаться следующие файлы (Рис. 6.4):

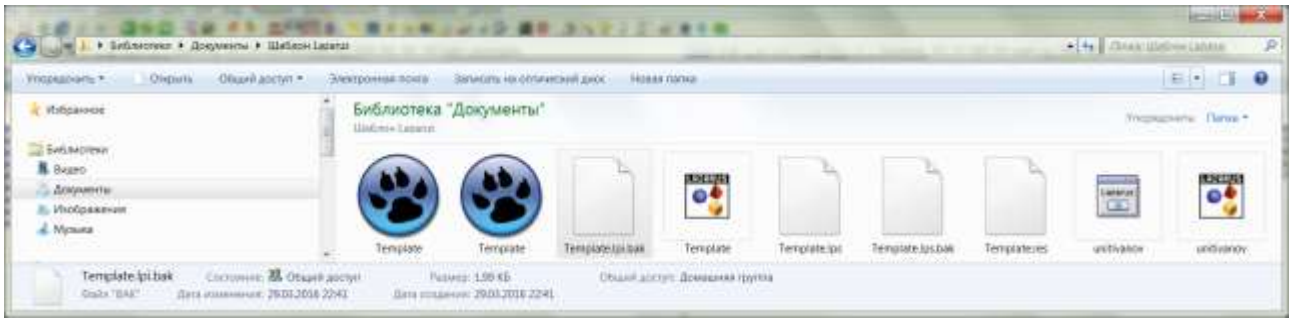


Рис. 6.4 - Содержание папки **Шаблон Lazarus** после сохранения проекта

---

[6] ***Template*** (*англ.*) - шаблон, лекало

Примечание. Откуда появляются файлы с расширением **.bak*?

- Это страховочные копии (англ. ***bacup***) предыдущих редакций исходных файлов до момента их последнего сохранения.
- Если нужно вернуться к предыдущей редакции файла (например, если сохранённая версия содержит ошибки и требуется вернуть то, что было), то переименуйте страховочную копию - вместо *bak* запишите расширение исходного файла.

6.5 Создание Формы

Каждая программа *Lazarus* с графическим интерфейсом содержит как минимум одно окно, внутри которого отображаются остальные элементы интерфейса. Окно является объектом класса TForm (**Форма**) и обладает всеми свойствами стандартных графических окон. Оно как правило имеет заголовок. Свойства **Формы** определяют вид окна Вашей программы. Его можно свернуть, можно развернуть во весь экран, можно менять размеры. Окно можно перемещать в любое место экрана. Таблица 6.1 представляет назначение основных свойств компонента TForm**,** Таблица 6.2 **–** *его* методов, а Таблица 6.3 - обрабатываемых событий.

Таблица 6.1 – Некоторые часто используемые свойства компонента класса ***TForm*** (Форма)

Свойство	Тип	Описание
Name	Строка	**Имя формы.** В программе могут быть несколько форм. Имя формы в программе используется для управления соответствующей формой и доступа к компонентам формы. Имя не должно содержать недопустимые символы, пробелы и т.д.
Caption	Строка	**Текст заголовка окна.** Обычно здесь выводят название программы и имя документа, связанного с этой программой или краткое содержание программы, например «Расчёт катушки индуктивности».
Top	Целое число	Расстояние от верхней границы формы до верхней границы экрана.
Left	Целое число	Расстояние от левой границы формы до левой границы экрана.
Width, ***Height***	Целое число	Ширина, высота формы. Размеры задаются в пикселях.
Icon		**Значок в заголовке диалогового окна**, обозначающий кнопку вывода системного меню.
Color		**Цвет фона.**
Font	Объект *TFont*	**Шрифт элементов интерфейса.** Шрифт, используемый по «умолчанию» для компонентов, находящимися на поверхности формы.
Canvas		Поверхность, на которую можно вывести графику.
Position		**Положение окна при запуске:** *poDesigned* – положение окна и его размеры остаются такими же, что и при проектировании; *poDefault* – положение окна и его размеры определяется автоматически операционной системой;

Свойство	Тип	Описание
		poDefaultPosOnly – положение окна определяется автоматически операционной системой, а размеры соответствуют установкам при проектировании; *poDefaultSizeOnly* – размеры окна определяется автоматически операционной системой, а положение соответствуют установкам при проектировании; *poScreenCenter* или *poDesktopCenter* – окно выводится в центре экрана, размер определяется при проектировании; *poMainFormCenter* – форма отображается в центре главной формы, размер определяется при проектировании, если имеется только одна главная форма, то этот параметр соответствует *poScreenCenter*; *poOwnerFormCenter* – форма отображается в центре той формы, которая является владельцем данной формы.
Align		**Управление положением формы на экране:** *alNone* – положение формы и его размеры не меняются; *alBottom* – форма располагается внизу экрана; *alLeft* – форма располагается в левой части экрана; *alRight* – форма располагается в правой части экрана; *alTop* – форма располагается вверху экрана; *alClient* – форма занимает весь экран.

Таблица 6.2 – Назначение некоторых часто используемых методов компонента класса ***TForm***

Метод	Аргументы	Возвращаемое значение	Описание
Show	Нет	Нет	Показывает окно на экране
ShowModal	Нет	Целое число	Показывает окно как модальное
Close	Нет	Нет	Закрывает окно

Примечание. Форма может быть **модальной** и **немодальной**.

Модальная форма это такая форма, которая не позволяет открывать другие формы, пока она сама не будет закрыта.

- К модальным формам обычно относятся диалоговые окна. Чтобы отобразить форму в модальном режиме необходимо вызвать метод ***ShowModal***.
- По умолчанию главная форма приложения открывается в *немодальном режиме*.

Таблица 6.3 – Назначение некоторых обрабатываемых событий компонента класса ***TForm***

Событие	Описание
OnResize	Происходит при изменении размеров окна
OnShow	Происходит при появлении окна на экране
OnHide	Происходит при исчезновении окна
OnActive	Происходит при активации окна
OnDeactive	Происходит при деактивации окна

6.6 Уточнение заголовка формы

В инспекторе объектов (компонент ***Form1: TForm1***) исправить значение параметра ***Caption*** – *заменить, например, текст* ***Form1*** *на* ***Шаблон Иванов И.И. 235-1****, после чего нажать Enter* (Рис. 6.5).

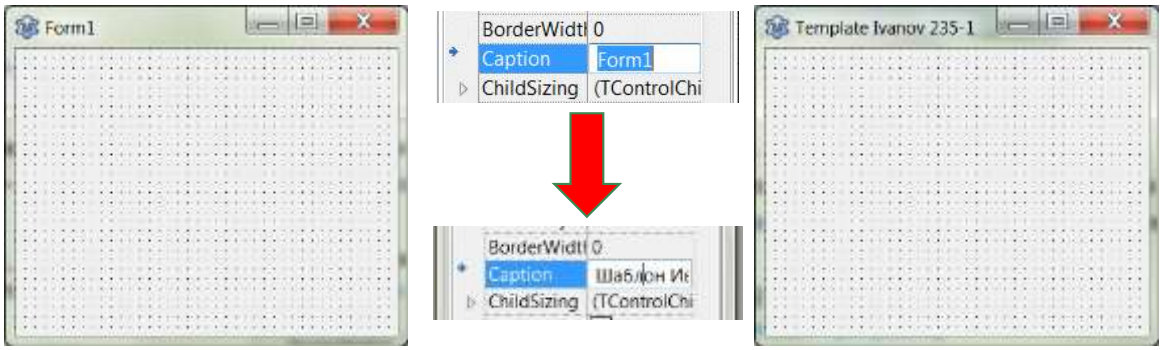


Рис. 6.5 - Уточнение заголовка формы

Примечание. Изменение размеров **Формы** выполняется стандартными действиями с помощью мышки, как это принято в *Windows*:

- Чтобы изменить размеры окна формы, щёлкните ЛК на его любой угол, верхнюю, нижнюю, левую или правую границу. Если указатель превратится в соответствующую двунаправленную стрелку, показывающую, что окно формы готово к изменению размеров, то перетащите указатель в нужном направлении.
- Нельзя изменить размер окна, если оно свёрнуто или развёрнуто во весь экран.

6.7 Формирование информационных строк

Для вывода на Форму текста, который пользователь не может изменить во время выполнения программы применяют компонент ***TLabel*** (Надпись, Метка) (Рис. 6.6).

Рис. 6.6 - Положение компонента ***TLabel*** во вкладке «Standard» палитры и его вид

Таблица 6.4 знакомит с часто используемыми параметрами компонента ***TLabel.***

Таблица 6.4 – Часто используемые параметры компонента ***Label*** (Надпись)

Свойство	Описание
Name	**Имя компонента.** Используется в программе для доступа к этому компоненту и его свойствам.
Caption	**Отображаемый текст** в поле надписи.
Left	Расстояние от левой границы поля вывода до левой границы формы.

Свойство	Описание
Top	Расстояние от верхней границы поля вывода до верхней границы формы.
Width, Height	**Ширина и высота поля вывода** (в пикселях).
AutoSize	Признак того, что размер поля определяется его содержимым. *True* – вертикальный и горизонтальный размер надписи зависит от длины текста, *False* – размеры компонента устанавливаются вручную, выравнивание текста производится свойством *Alignment*.
WordWrap	Признак того, что слова, которые не помещаются в текущей строке, автоматически переносятся на следующую строку (значение свойства *AutoSize* должно быть *False*). Если *WordWrap* дадим значение *True*, то производится перенос текста по словам, т.е. текст надписи отображается в несколько строк, иначе текст выводится в одну строку.
Alignment	**Способ выравнивания текста** внутри поля: *taLeftJustify* - выравнивание по левому краю; *taCenter* – выравнивание по центру; *taRightJustify* – Выравнивание по правому краю
Font	**Параметры шрифта, используемые для отображения текста**: *Font.Name* – вид шрифта; *Font.Size* – размер шрифта; *Font.Color* – цвет шрифта.
ParentFont	Признак наследования компонентом характеристик шрифта формы, на которой находиться компонент. Если значение свойства равно *True*, то текст выводиться шрифтом, установленным для формы.
Color	**Цвет фона** области вывода текста.
Transparent	Управляет отображением фона области вывода текста. Значение *True* делает область вывода текста прозрачной, (область не закрашивается цветом, заданным свойством *Color*).
Visible	Позволяет скрыть текст (*False*) или сделать его видимым (*True*) во время выполнения приложения.

Примечание. Разместить нужный компонент на **Форме** можно следующими способами:

- Дважды щёлкнуть левой кнопкой мыши (ЛК) на значке компонента, расположенного на палитре компонентов. При этом компонент попадёт в левый верхний угол **Формы**, а не в то место, куда Вы хотите. Но его затем можно перетащить в желаемое место с помощью мышки (как это принято в *Windows*).
- Щёлкнуть на значке компонента (при этом он выделяется) и щёлкнуть в любое желаемое место на **Форме**. Таким образом, компонент можно поместить, куда было задумано.

Мы будем использовать компоненты ***TLabel*** для формирования информационных строк с назначением программы, фамилией её автора, номером варианта и условиями индивидуального задания (Рис. 6.7).

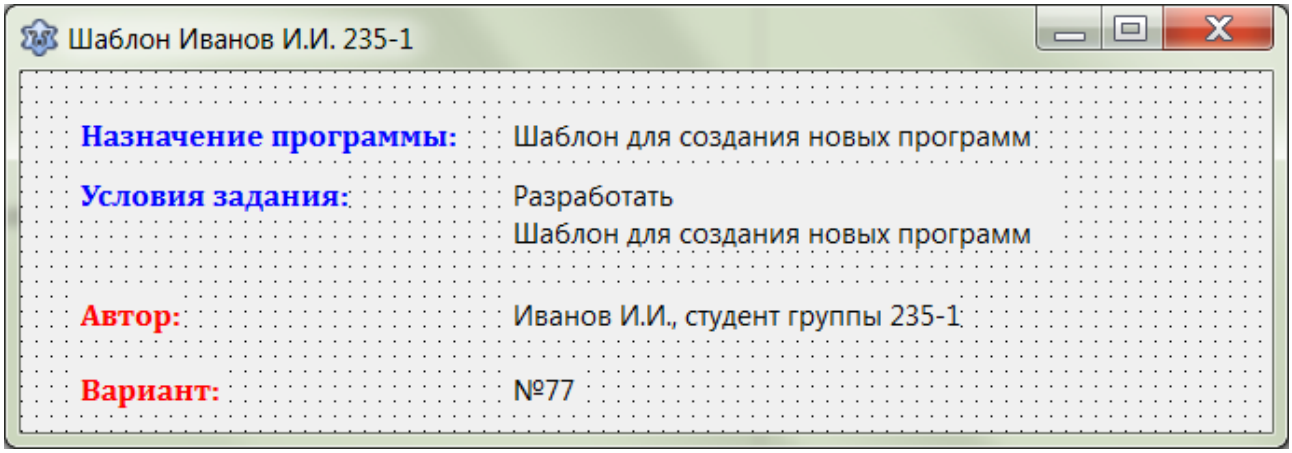


Рис. 6.7 - Создание информационых строк в Форме

Так как в создаваемой Форме использованы тексты с разным оформлением (цвет текста, начертание), то потребовалось добавить 8 компонентов ***TLabel***. Обратите внимание – как изменяется дерево компонентов в **Инспекторе объектов,** а также программный код программы в окне **Редактора исходного кода** при добавлении компонентов ***TLabel***. С помощью мышки размещаем компоненты ***TLabel*** в нужных местах **Формы.** Используйте вспомогательные линии, появляющиеся при перемещении этих компонентов для выравнивания надписей по вертикали и горизонтали.

Для каждого компонента вводим требуемый текст в качестве параметра свойства ***Caption***. Нажимаем кнопку  у свойства ***Font*** и в появившемся диалоговом окне уточняем параметры шрифта отображаемого текста (Рис. 6.8).

Чтобы длинный текст в условий задания мог отображаться в нескольких строках, установим значение свойства ***AutoSize*** в ***False***, а свойству ***WordWrap*** присвоим значение ***True.*** Нажмём кнопку  свойства ***Caption***, и в открывшемся окне Диалог ввода строк (**Ошибка! Источник ссылки не найден.**) отредактируем многострочный текст.

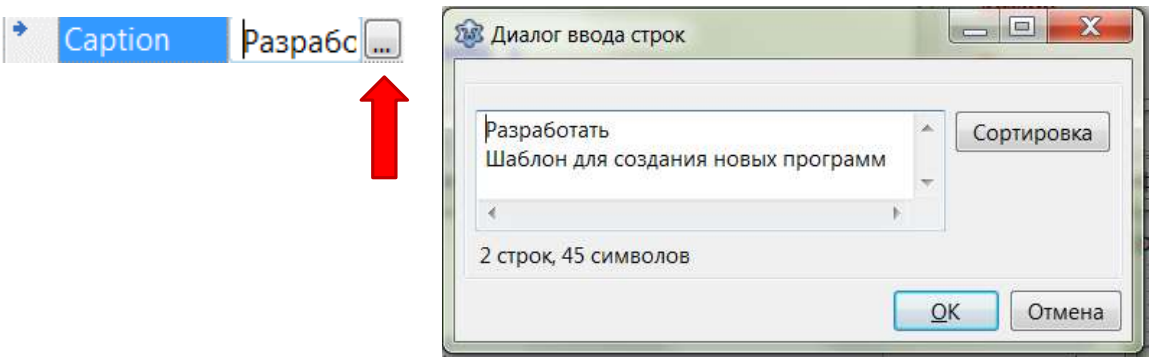


Рис. 6.8 - Редактирование многострочной надписи

Уточняем окончательно размеры **Формы** и надписей

6.8 Создание командной кнопки

Компонент ***TButton*** (Кнопка) – командная кнопка, с помощью которой пользователь может вызывать выполнение какого-либо действия (Рис. 6.9).

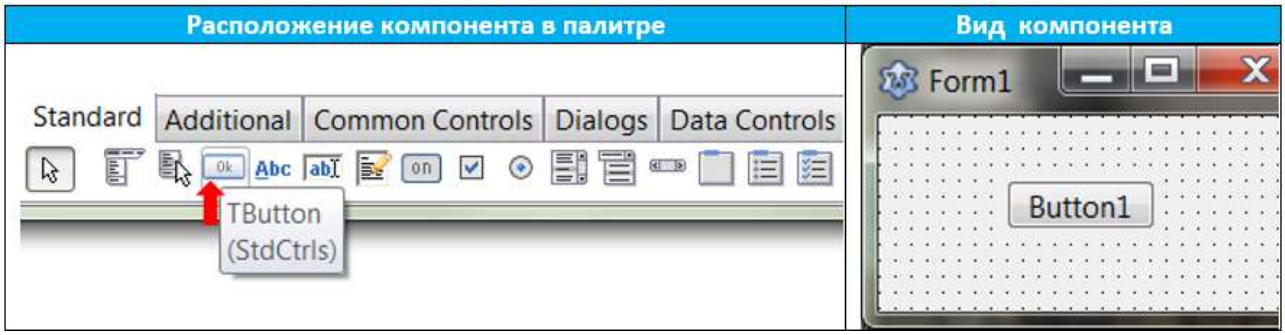


Рис. 6.9 - Положение компонента ***TButton*** (Кнопка) во вкладке «Standard» палитры и его вид

Таблица 6.5 представляет назначение основных свойств компонента ***TButton***а

Таблица 6.5 - Основные свойства компонента ***TButton*** (Кнопка)

Свойство	Описание
Name	**Имя компонента.** Используется в программе для доступа к компоненту и его свойствам.
Caption	**Текст на кнопке.**
Left	Расстояние от левой границы кнопки до левой границы формы.
Top	Расстояние от верхней границы кнопки до верхней границы формы.
idth, Height	**Ширина, высота кнопки.**
Enabled	**Признак доступности кнопки.** $True$ -кнопка доступна, $False$ – кнопка недоступна (надпись на кнопке имеет бледный вид, нажатие на кнопку не приводит ни к каким действиям, даже если имеется обработчик события $OnClick$).
Visible	**Позволяет скрыть текст.** $False$ – текст видим, $True$ – текст невидим.
Hint	**Контекстная подсказка** – текст, который появляется рядом с указателем мыши при наведении указателя (для того чтобы текст появился, надо чтобы значение свойства $ShowHint$ было $True$).
ShowHint	Разрешает ($True$) или запрещает ($False$) отображение подсказки при наведении указателя на кнопку.
Default	Если установлено значение $True$, то нажатие клавиши $Enter$ будет эквивалентно щелчку по кнопке мышью.
Cancel	Если установлено значение $True$, то нажатие клавиши Esc будет эквивалентно щелчку по кнопке мышью.

Таблица 6.6 представляет обрабатываемые события этого компонента.

Таблица 6.6 - События компонента ***TButton***

Событие	Описание
OnClick	Щелчок на кнопке
OnFocus	Получение фокуса[7]

Щелчок по кнопке мышью вызывает событие ***OnClick*** в обработчике которого программист инициирует выполнение каких-либо действий, команд и процедур.

Вставьте в **Форму** ещё один компонент ***TLabel. В окне Инспектора объектов появится новый объект Label9:TLabel. В Инспекторе объектов уберите в свойстве Caption значение Label9*** (создаём пустую надпись). Используя диалоговое окно свойства ***Font*** для компонента **Label9** подберите параметры выводимого текста - шрифт, начертание, размер, цвет (Рис. 6.10).

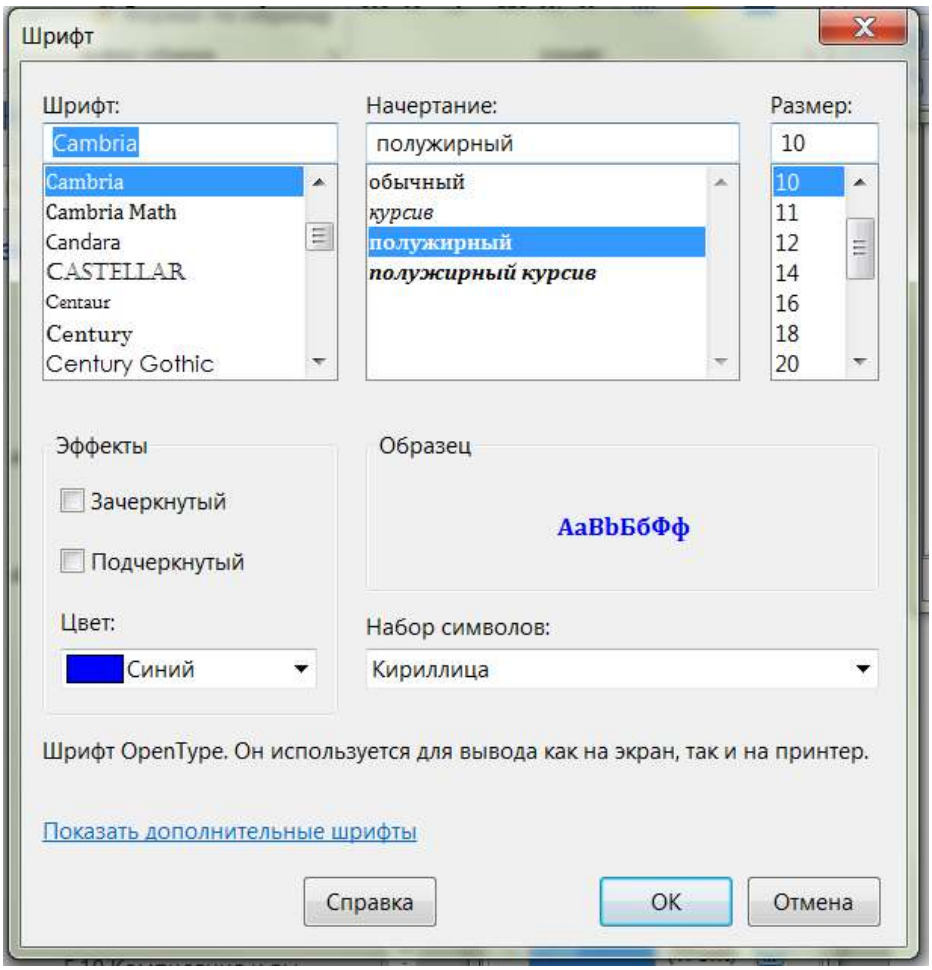


Рис. 6.10 - Диалоговое окно Шрифт. Установка параметров выводимого текста

Разместите в **Форме** кнопку ($TButton$). Определите значение свойства Caption (название кнопки) **Важное сообщение**. Высоту, ширину и положение кнопки на форме отрегулируйте с помощью мышки.

---

[7] Обычно фокусировка на элементе автоматически происходит при нажатии на элементе мышкой, но также можно перейти на нужный элемент клавиатурой – через клавишу ***Tab***, нажатие пальцем на планшете и так далее.

Напишите программный код для процедуры *обработчика события* **Щелчок на кнопке**. Пока мы этого не сделаем, кнопка работать не будет - при нажатии на кнопку ничего происходить не будет.

Для этого выполните двойной щелчок по кнопке. Откроется **Редактор исходного кода**, в котором после кода, созданного автоматически, добавится новая подпрограмма-процедура (Рис. 6.11) - $TForm1.Button1Click$ – обработчик события **Щелчок на кнопке** (англ. *Button* – кнопка, *Click* – щелчок). Пока это лишь пустая заготовка процедура обработчика события, не содержащая ни одного оператора. Конечно, при нажатии кнопки она ничего не будет делать.

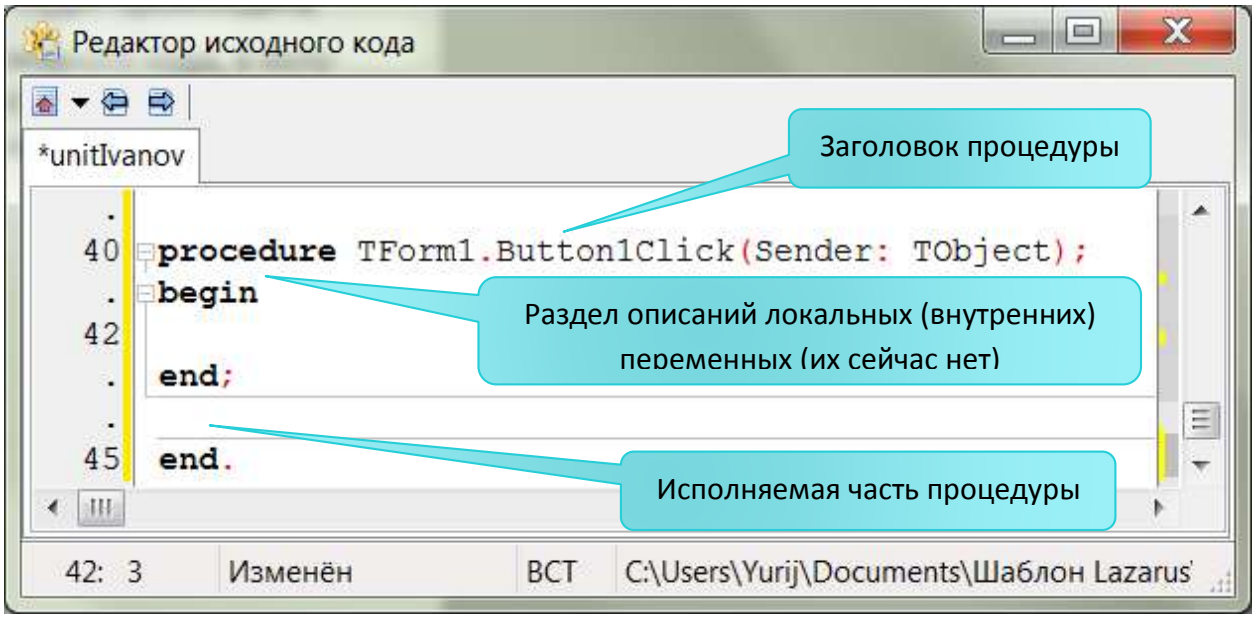


Рис. 6.11 – Пустой шаблон процедуры обработчика события **Щелчок на кнопке**

Структура подпрограммы-процедуры содержит **Заголовок, Раздел описаний и Исполняемую часть:**

```
Procedure <имя процедуры>(список формальных параметров);
  //Раздел локальных описаний процедуры (действуют только внутри процедуры):
  // констант, переменных, типов, меток, процедур, функций
begin
  //Операторы исполняемой части процедуры>
  ...
end;
```

Формальные параметры в заголовке процедур записываются в виде:

var имя параметра: имя типа

и отделяются друг от друга точкой с запятой. Если параметр не возвращает результат расчёта, то ключевое слово ***var*** может отсутствовать. Если параметры однотипны, то их имена можно перечислять через запятую, указывая общее для них имя типа. При описании параметров можно использовать только стандартные (встроенные) имена типов, либо имена пользовательских типов, определённые с помощью оператора ***type***.

Список формальных параметров может отсутствовать.

Процедура вызывается по имени:

< имя процедуры > (список фактических параметров);

Список фактических параметров - это их перечисление через запятую. Значение каждого фактического параметра при вызове процедуры как бы подставляется вместо формального параметра, стоящего на том же месте в заголовке.

После передачи *входных параметров*, управление временно передаётся процедуре - выполняются операторы исполняемой части. Возврат значений *выходных параметров* (только для тех, которые помечены ***var***) в вызывающую программную единицу происходит непосредственно после работы исполняемой части процедуры. После завершения работы процедуры управление возвращается в вызывающую программную единицу и её работа продолжается.

Каждый формальный параметр указывается вместе со своим типом. Соответствующий ему фактический параметр указывается без типа. Между формальными и фактическими параметрами должно быть соответствие по количеству параметров, по их типу и порядку следования.

Чтобы процедура выполнила необходимые действия, напишите соответствующий код между операторными скобками *begin* и *end*. В нашем случае это оператор присваивания, который в случае щелчка ЛК мышью по кнопке **Важное сообщение** изменяет свойство $\mathbf{Caption}$ (текст надписи) объекта *Label9* на новое значение «Я, студент Иванов И.И. – великий программист!» вместо отсутствующего текста.

Вводя код, обратите внимание на подсказку, появляющуюся после ввода точки (нужно немного подождать), следующей за label9 (Рис. 6.12). Подсказка представляет собой всплывающее меню, в котором перечислены допустимые свойства и методы компонента label9.

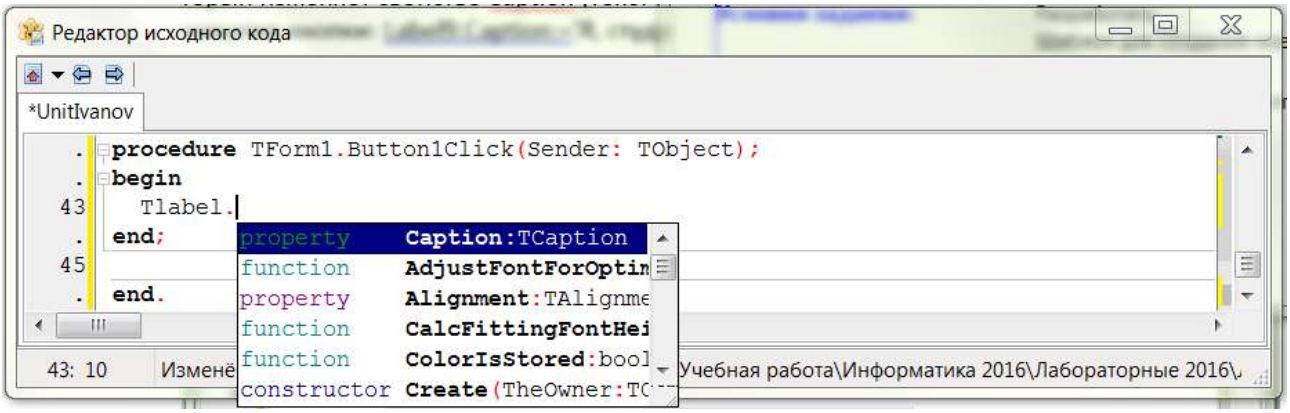


Рис. 6.12 - Всплывающее меню, в котором перечислены допустимые свойства и методы компонента $label1$.

С помощью мыши Вы при необходимости можете выбрать из списка нужное свойство (англ. ***property***) - в нашем случае ***property Caption,*** или метод (подпрограммы типов *procedure, function, constructor*). Вы также можете начать вводить имя нужного свойства или метода, при этом *Lazarus* автоматически прокручивает список и находит имена, первые

буквы которых совпадают с вводимыми буквами. Это поможет Вам, если Вы забыли точное имя свойства или метода. Если теперь нажать <*Пробел*> или <*Enter*>, то *Lazarus* вместо Вас автоматически завершит ввод имени.

Окончательный вид процедуры обработчика события кнопки «Важное сообщение» приведён на Рис. 6.13.

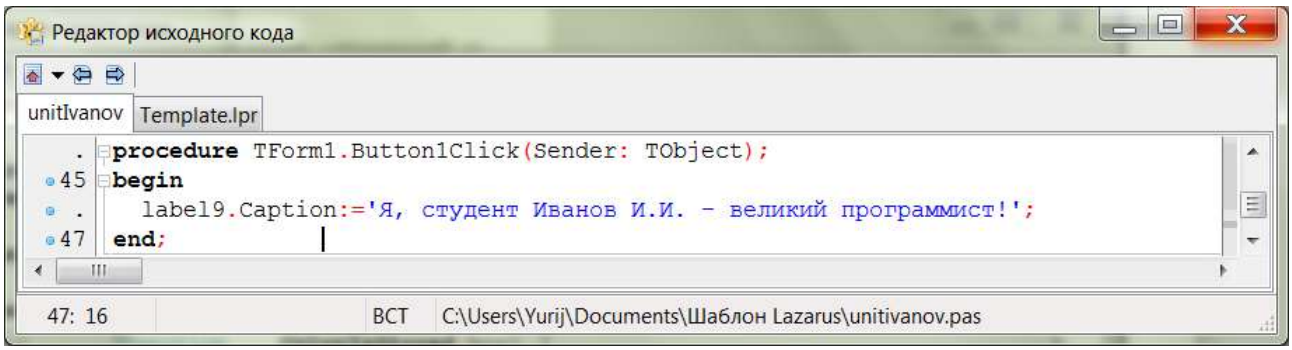


Рис. 6.13 – Окончательный вид процедуры обработчика события кнопки «Важное сообщение»

6.9 Создание кнопки «Завершить работу»

Добавьте из палитры на **Форму** ещё одну кнопку ***Button2: TButton***. Установите значение свойства ***Caption*** для этой кнопки – **Завершить работу**.

Напишите программный код для процедуры *обработчика события* **Щелчок на кнопке Завершить работу**. Откройте **Редактор исходного кода** с помощью двойного щелчка по кнопке **Завершить работу** на **Форме**, в котором добавится заготовка новой процедуры *TForm1.Button2Click* для нашей **Формы** - обработчика события **Щелчок на кнопке *Button2***.

Чтобы процедура выполнила необходимые действия по закрытию окна, напишите между операторными скобками *begin* и *end* оператор вызова метода закрытия окна ***Close*** (см. Таблица 6.2). На Рис. 6.14 представлен вид процедуры обработчика события кнопки «Завершить работу»

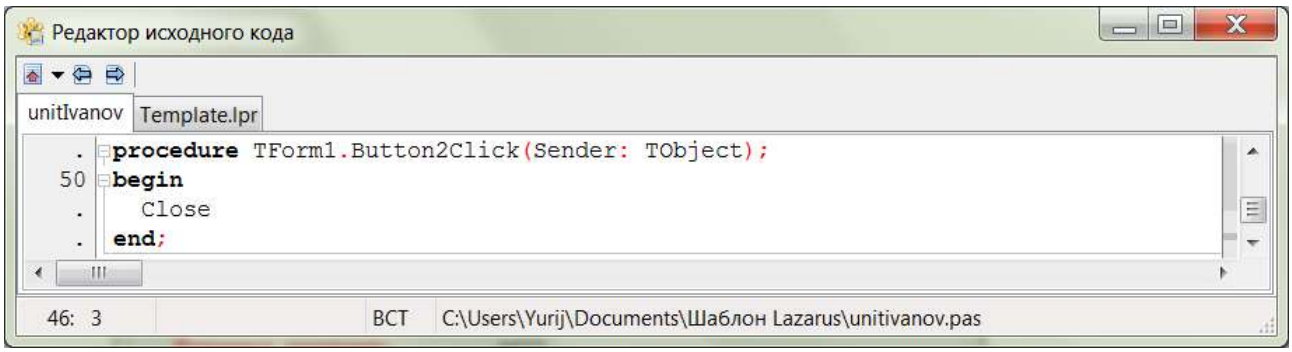
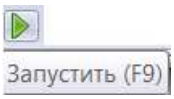


Рис. 6.14 - Вид процедуры обработчика события кнопки «Завершить работу»

6.10 Компиляция и выполнение программы

Выполнить программу можно одним из способов:

1) щёлкнуть по кнопке 

на панели инструментов;
2) выбрать команду **Запуск (Run) ⇒ Запустить (Run)** в главном меню;
3) нажать клавишу <***F9***>.

Происходит процесс компиляции, в результате которого в папке проекта создаётся *exe*-файл. В **Окне сообщений** выводится протокол сборки проекта (Рис. 6.15):

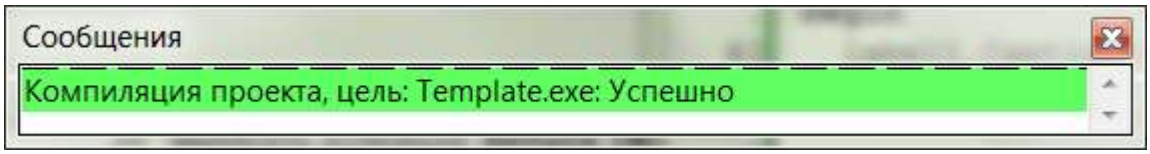


Рис. 6.15 - Сообщение об успешной компиляции

В случае, если были допущены ошибки, соответствующее сообщение об этом появится в окне сообщений.

Если компиляция прошла успешно, то на экране появится **Окно** с кнопками, однако пока что без надписи. Если теперь щёлкнуть ЛК по кнопке **Важное сообщение**, то в окне появится надпись «Я, студент Иванов И.И. – великий программист!» (Рис. 6.16).

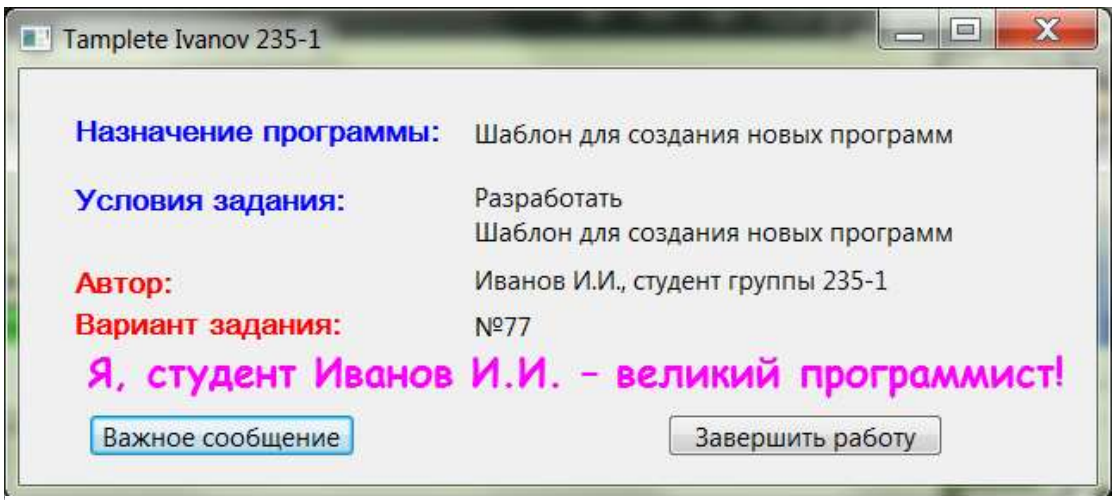


Рис. 6.16 - Результаты работы Вашей программы

Если щёлкнуть ЛК мыши по кнопке **Завершить работу**, то работа созданного Вами приложения завершится.

Таким образом, вы создали приложение, реагирующее на действия пользователя. Скомпилированная программа сохраниться в папке проекта в виде файла с расширением $.exe$. Он может быть выполнен на компьютере без среды разработки Lazarus.

6.11 Сохранение файлов проекта

Для этого выполните команду ***Проект ⇒ Сохранить*** или ***Файл ⇒ Сохранить***

7 Список литературы

1. Алексеев Е.Р., Чеснокова О.В., Кучер Т.В. Free Pascal и Lazarus: Учебник по программированию / Е.Р. Алексеев, О.В. Чеснокова, Т.В. Кучер. - М.: Издательский дом ДМК-пресс, 2010. - 440 с.
2. Алексеев Е.Р., Чеснокова О.В., Кучер Т.В.. Самоучитель по программированию на Free Pascal и Lazarus.. - Донецк: ДонНТУ, Технопарк ДонНТУ УНИТЕХ, 2011. - 503 с.
3. Кетков Ю.Л. Свободное программное обеспечение. FREE PASCAL для студентов и школьников / Ю.Л. Кетков, А.Ю. Кетков. — СПб.: БХВ-Петербург, 2011. — 384 с.
4. Мансуров К.Т. Основы программирования в среде Lazarus. - М.: Нобель пресс, 2013. – 772 с.
5. Фаронов В.В. Turbo Pascal. Наиболее полное руководство (в подлиннике). — СПб.: БХВ-Петербург, 2004. — 1056 с.
6. Фленов М.Е. Библия Delphi. — 2-е изд., перераб. и доп. - СПб.: БХВ-Петербург, 2008. - 800 с.
7. Lazarus Tutorial/ru [Электронный ресурс] // База знаний о Free Pascal, Lazarus и родственных проектах: [сайт]. URL: http://wiki.freepascal.org/Lazarus_Tutorial/ru
8. Программирование на Lazarus [Электронный ресурс] // «ИНТУИТ» Национальный открытый университет: [сайт]. URL: http://www.intuit.ru/studies/courses/13745/1221/lecture/23276?page=1
9. ОС ТУСУР 01-2013 (СТО 02069326.1.01-2013). Работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления. - Томск: ТУСУР, 2013. – 57 с.
10. Кобрин Ю.П. Приложение к лабораторной работе "Работа в интегрированной среде Lazarus (free Pascal)". Томск: ТУСУР, 2016. - 21 с.