Министерство образования Республики Беларусь
УО «Полесский государственный университет»

В.Н. ШТЕПА,
Л.Н. БАЗАКА,
Д.А. ДЕРКАЧ,
А.А. ДМИТРАНИЦА

ВИЗУАЛЬНЫЕ СРЕДСТВА РАЗРАБОТКИ ПРОГРАММНЫХ ПРИЛОЖЕНИЙ

Методические рекомендации
по выполнению лабораторных работ для студентов
по специальности 1-40 05 01
«Информационные системы и технологии»
(по направлениям)

Пинск
ПолесГУ
2016

УДК 004.4(072)
ББК 32.973.26я73
В41

Р е ц е н з е н т ы:
кандидат технических наук, доцент
Ю.М. Вишняков;
кандидат экономических наук, доцент
И.А. Янковский

У т в е р ж д е н о
научно-методическим советом ПолесГУ

Штепа, В.Н.

В41 Визуальные средства разработки программных приложений : методические рекомендации по выполнению лабораторных работ / В.Н. Штепа [и др.]. – Пинск : ПолесГУ, 2016. – 72 с.

ISBN 978-985-516-447-1

Предназначено для студентов по специальности 1-40 05 01 «Информационные системы и технологии» (по направлениям).

УДК 004.4(072)
ББК 32.973.26я73

ISBN 978-985-516-447-1

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Среда разработки Visual Studio .Net – интегрированная среда разработки программного обеспечения, продукт компании Microsoft [1–3].

Обычно среда разработки включает [9]:

– текстовый редактор;
– компилятор и / или интерпретатор;
– средства автоматизации сборки;
– отладчик.

Visual Studio включает поддержку языков C++, C# и Visual Basic .NET, а также языка F# [4, 10], отсутствующего в предыдущих версиях. Framework.NET (произносится Dot Net) – программная платформа, выпущенная компанией Microsoft в 2002 г.

Основой платформы является исполняющая среда:

– CLR способна выполнять как обычные программы, так и серверные веб-приложения;
– .NET Framework поддерживает создание программ, написанных на различных языках программирования.

В Framework .Net можно выделить два основных компонента:

– статический – FCL (Framework Class Library – библиотека классов) [11];
– динамический – CLR (Common Language Runtime – общеязыковая исполнительная среда).

Entity Framework представляет специальную объектно-ориентированную технологию на базе фреймворка .NET для работы с данными. Если традиционные средства ADO.NET позволяют создавать подключения, команды и прочие объекты для взаимодействия с базами данных, то Entity Framework представляет собой более высокий уровень абстракции [5], который позволяет абстрагироваться от самой базы данных и ра-

ботать с данными независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает Entity Framework, мы уже работает с объектами.

Первая версия Entity Framework – 1.0 вышла еще в 2008 г. и представляла очень ограниченную функциональность, базовую поддержку ORM (object-relational mapping – отображения данных на реальные объекты) и один-единственный подход к взаимодействию с базой данных (БД) – Database First [10]. С выходом версии 4.0 в 2010 г. многое изменилось – с этого времени Entity Framework стал рекомендуемой технологией для доступа к данным, а в сам Framework были введены новые возможности взаимодействия с БД – подходы Model First и Code First. Дополнительные улучшения функционала последовали с выходом версии 5.0 в 2012 г. В 2013 г. был выпущен Entity Framework 6.0 [5, 7], обладающий возможностью асинхронного доступа к данным.

Центральной концепцией Entity Framework является понятие сущности или entity. Сущность представляет набор данных, ассоциированных с определенным объектом [4]. Поэтому данная технология предполагает работу не с таблицами, а с объектами и их наборами.

1. ОСНОВЫ СОЗДАНИЯ ПРОГРАММНЫХ ПРИЛОЖЕНИЙ В WINDOWS FORMS

Windows Forms – интерфейс программирования приложений (API), отвечающий за графический интерфейс пользователя и являющийся частью Microsoft .NET Framework.

Данный интерфейс упрощает доступ к элементам интерфейса Microsoft Windows за счет создания обёртки для существующего Win32 API в управляемом коде. Причём управляемый код – классы, реализующие API для Windows Forms, не зависят от языка разработки. Таким образом, программист одинаково может использовать Windows Forms как при написании программного обеспечения (ПО) на C#, C++, так и на VB.Net, J# и др. языках.

С одной стороны, Windows Forms рассматривается как замена более старой и сложной библиотеки MFC, изначально написанной на языке C++. С другой стороны, WF не предлагает парадигму, сравнимую с MVC. Для исправления этой ситуации и реализации данной функциональности в WF существуют сторонние библиотеки.

Однако следует отметить, что самостоятельно C++ с Windows Forms взаимодействовать не может, поэтому на самом деле при работе с библиотеками .NET Framework используется промежуточный язык программирования C++/CLI.

1.1 Создание нового проекта

Чтобы создать проект в Windows Forms, нужно запустить Microsoft Visual Studio C++, нажать “Создать проект”, выбрать (слева) “CLR”, далее выберите “Приложение Windows Forms” и назовите свой проект, например, “first_lesson”.

В последних версиях MS Visual Studio больше нет возможности создать приложение Windows Forms автоматически. Однако соответствующая функциональность сохранена. Для этого необходимо создать пустой проект CLR и добавить в него форму "вручную".

Вот весь процесс, расписанный по шагам:

Меню Создать проект, Visual C++, CLR, Пустой проект CLR. После создания проекта нажать на вкладку Проект.

Добавить новый элемент, UI, форма Windows Form.

Добавить следующий код в файл MyForm.cpp:

```
#include "MyForm.h" //Здесь пишем имя h-файла вашей формы!
using namespace Example1; //Здесь пишем имя вашего проекта!
[STAThreadAttribute]
int main(array<System::String ^> ^args) {
Application::EnableVisualStyles();
Application::SetCompatibleTextRenderingDefault(false);
Application::Run(gcnew MyForm()); //Также пишем имя своей формы, если оно не MyForm
return 0;
}
```

Далее нажимаем вкладку Проект, Свойства имя_текущего_проекта.

Выбираем Свойства конфигурации, Компоновщик, Система, справа в поле Подсистема вставляем Windows (строку / SUBSYSTEM:WINDOWS).

Затем Свойства конфигурации, Компоновщик, Дополнительно, справа поле Точка входа, вставляем строку main.

Также в MyForm.cpp можно добавить следующий код:

```
#include "MyForm.h"
using namespace System;
using namespace System::Windows::Forms;
[STAThread]
void main(array<String^>^ arg) {
Application::EnableVisualStyles();
Application::SetCompatibleTextRenderingDefault(false);
Project1::MyForm form;//Вместо Project1 – имя вашего проекта;
Application::Run(%form);
}
```

Далее будут рассматриваться отдельные приемы работы с основными элементами управления Windows Forms, часто используемые в реализации проектов.

Более подробная информация о классах System.Windows. Forms и System.Windows.Forms.Control приведена на сайте msdn.microsoft.com.

1.2 Элементы button, textBox и label

Здесь продемонстрировано создание первой простой, но полноценной программы. Суть ее в том, что вы пишете в текстовом поле какое-то предложение, нажимаете на копку, и на форме появляется надпись – то же самое предложение.

Создайте новый проект с единственной пустой формой.

Итак, познакомимся сначала с интерфейсом компилятора: справа от вас находится панель всевозможных инструментов (элементов), а слева, собственно, сама форма, на которую вы и будете размещать выбранные вами элементы (**Рис. 1.1**).

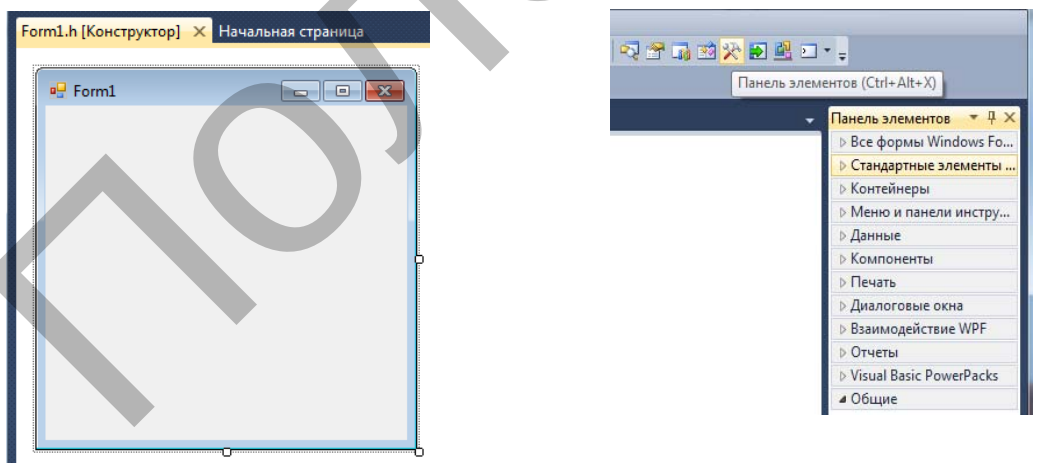

Рис. 1.1 – Интерфейс компилятора

Нажмём на “Стандартные элементы” и выберем из них "button" "textbox" и "label", перетаскивая их поочерёдно на форму (**Рис. 1.2**).

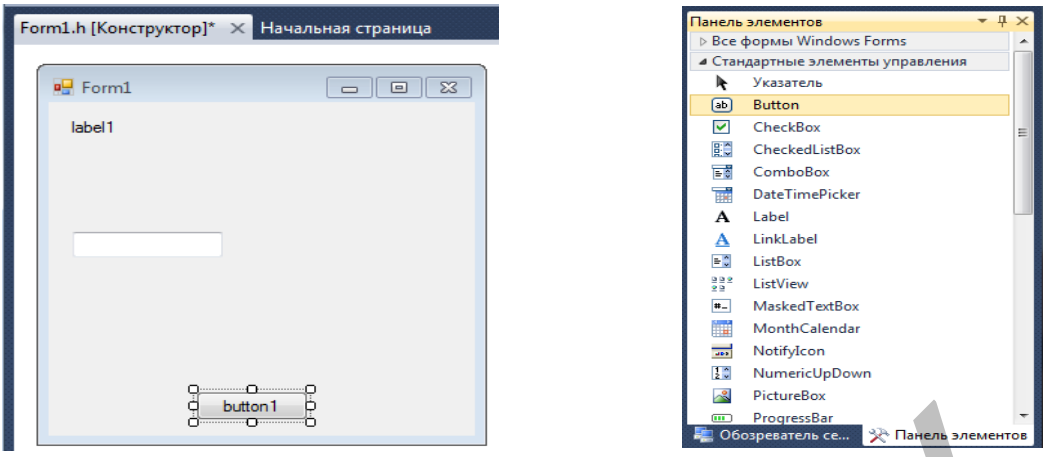

Рис. 1.2 – Стандартные элементы на форме

Нажав, например, на кнопку "button1", справа вы увидите её свойства, их там очень много.

Для начала выберите свойство "Text" и напишите там вместо "button1" "Копировать" (**Рис. 1.3**).

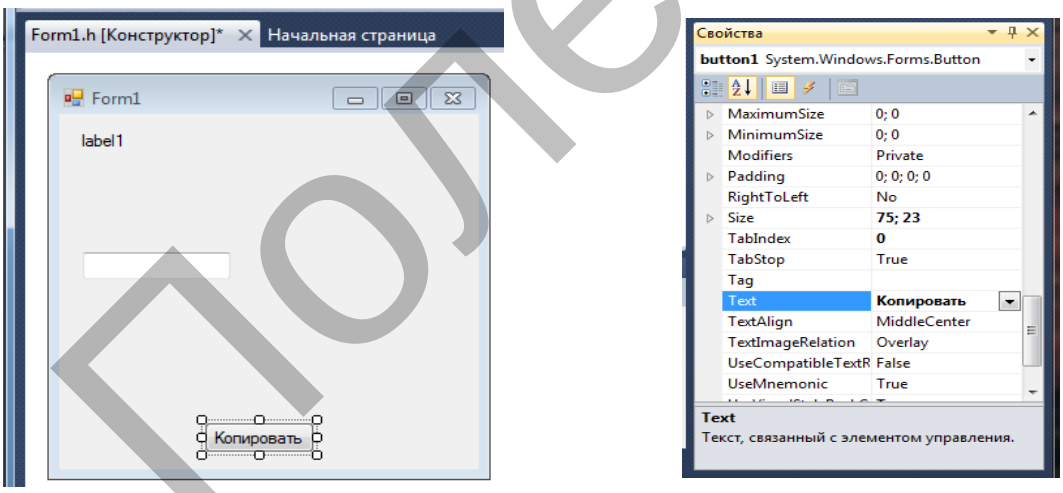

Рис. 1.3 – Описание свойств кнопки

Также вы, безусловно, знаете, что размер формы и элементов можно изменять.

Измените расположение и размеры элементов управления формы для более компактного размещения (**Рис. 1.4**).

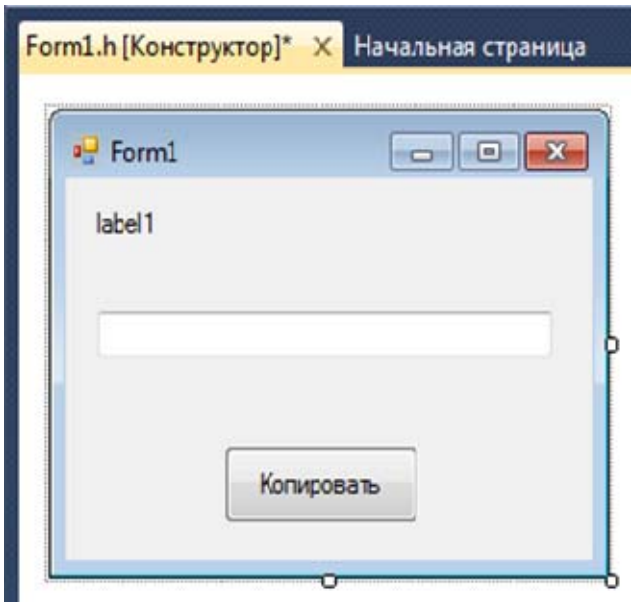

Рис. 1.4 – Изменение размера формы

Далее щёлкните два раза по форме – перед вами раскроется новое окно с программным кодом, в котором мы и будем, собственно, писать код. В данном случае откроется событие Form_Load, где вам нужно написать label1 –> Text = ""; Эта строка говорит о том, что, когда произойдёт запуск (загрузка) программы, текст "label1" изменится на "" (**Рис. 1.5**).

```
            this->PerformLayout();

        }
#pragma endregion
    private: System::Void Form1_Load(System::Object^  sender, System::EventArgs^  e) {
                 label1->Text = "";
             }
```

Рис. 1.5 – Событие Form_Load

После этого нажмите наверху "Form1.h[Конструктор]". Откроется Событие "button1_Click".

Напишите в нём label1 –> Text = textBox1 –> Text**;** это будет говорить о том, что, когда при загрузке формы вы нажмёте («clicknete») на кнопку, произойдёт описанное действие: текст, написанный в "textBox1", скопируется в "label1".

Далее приведён код на C++:

```
    #pragma endregion
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
    {
```

```
   label1 -> Text = "";
        }
```

private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)

```
{
   label1 -> Text = textBox1 -> Text;
}
```

1.3 Элемент MessageBox

Здесь продемонстрированно создание простой программы, в которой при нажатии на кнопку будет «выскакивать» небольшое окошко, сообщающее нам о том, какой текст был записан в текстовом поле. Для этого понадобятся всего лишь два элемента – "button" и "text-Box". Ну а теперь создайте проект, назовите его – "second_lesson" и перенесите на форму "button" и "textBox", а также два "label". Далее кликните два раза сначала по форме, а затем по кнопке (**Рис. 1.6**).

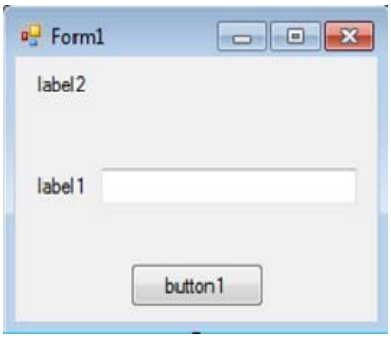

Рис. 1.6 – Полученная форма

Далее напишите ниже находящийся код. Вы увидите, как к тексту "MessageBox" прибавляется текст, написанный вами в "textBox1", далее мы даём название «выскочившему окошку» – так же как и названию вашей формы.

```
        }
#pragma endregion
    private: System::Void Form1_Load(System::Object^  sender, System::EventArgs^  e) {
             }
    private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {
             }
    };
}
```

Рис. 1.7 – Описание события в коде

Ниже приведен код:

```
#pragma endregion
private: System::Void Form1_Load(System::Object^ sender,
System::EventArgs^ e)
{
this -> Text = "Форма приветствия";
label1 -> Text = "Name:";
label2 -> Text = "Напишите ваше имя.";
button1 -> Text = "Ввод";
}
private: System::Void button1_Click (System::Object^ sender,
System::EventArgs^ e)
{
MessageBox::Show("Здравствуй " + textBox1 -> Text + "!", "При-
ветствие");
}
```

1.4 Событие MouseHover

Сейчас вы познакомитесь с таким весьма нужным «событием», как "MouseHover". Идея вполне простая: если вы наводите курсором на один из элементов, на котором описано данное событие, то элемент как-либо изменяется – как напишете, так и изменится. В данном примере при наведении курсора мыши на надпись "label" изменяется текст и цвет текста, а также «выскакивает» MessageBox – с каким-нибудь текстом. Создайте проект в Windows Forms, назовите его “third_lesson”, после чего перенесите на форму элемент "label". Теперь нажмите на "label" – справа вы должны увидеть список его свойств; помимо этого, наверху будет значок, похожий на жёлтую молнию, – нажимайте на него и выбирайте событие "MouseHover", два раза кликнув на него (**Рис. 1.8**).

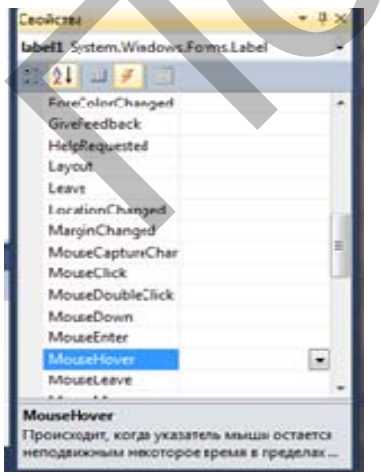

Рис. 1.8 – Событие MouseHover

Далее вам надо будет прописать для текста "label" – text align center, чтобы надпись была в середине относительно самой себя.

Теперь перейдём к коду:

```
#pragma endregion
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
    {
  Form1::Text = "Hover";
  label1 -> TextAlign = ContentAlignment::MiddleCenter;
  label1 -> Text = "Не трогай.";
}

private: System::Void label1_MouseHover(System::Object^ sender, System::EventArgs^ e) {
  label1 -> TextAlign = ContentAlignment::MiddleCenter;
  label1 -> Text = "ERROR!!!";
  label1 -> ForeColor = Color::Red;
  MessageBox::Show("Написано же\nНЕтрогать!", "Fatal ERROR!",
  MessageBoxButtons::OK, MessageBoxIcon::Error);
}
```

1.5 Подсказка ToolTip

Здесь мы рассмотрим очень маленький по своей сути, но в то же время очень актуальный элемент – подсказку "TollTip".

Для примера возьмём программу из предшествующего урока. Нам останется дописать буквально пару строк. Задача следующая – при наведении курсора мыши на текстовое поле будет появляться маленькое текстовое поле, сообщающее о том, что здесь нужно ввести имя. Перенесите на форму из панели инструментов два "label", один "button" и подсказку "ToolTip". Подсказка должна быть привязана к элементу управления, а текст подсказки не должен быть пустым (если вам не нужна пустая подсказка).

Вот код реализации:

```
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{
   this -> Text = "Форма приветствия";
   label1 -> Text = "Name:";
   label2 -> Text = "Напишите ваше имя.";
   button1 -> Text = "Ввод";
   //------------ реализация ToolTip
   toolTip1 -> SetToolTip(textBox1, "Введите\n ваше имя");
   toolTip1 -> IsBalloon = true;
}

private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
MessageBox::Show("Здравствуй" + textBox1 -> Text + "!",
"Приветствие");
}
```

1.6 Изменение шрифта текста и цвета формы и элементов

Вполне может оказаться, что ваша программа требует более оригинального оформления, чем стандартное – задать цвет кнопки, задать фон формы, загрузив изображение.

Рис. 1.9 – Необходимое изображение

Для наглядного примера создадим проект, в котором будем записывать на фоне изображения текст, задав изображения для заднего фона ("BackGroundImage") формы и изменив цвет кнопки. Для этого нам понадобятся "textBox", "label", кнопка "button" и изображение, показанное на **Рис. 1.9**.

Создав новый проект в "Windows Forms", нажмите на форму, слева вы увидите её свойства – нам нужно "BackGroundImage" (**Рис. 1.10**). После этого нажмите на кнопку "…" и выберите "Локальный ресурс", после чего нажмите на кнопку "Импорт". Откроется проводник – вам нужно открыть в нем сохранённое изображение, пример которого был показан выше (можете сохранить его в папку прямо со страницы). Далее нажимаете на элемент "button", выбираете в его свойствах "BackColor" и ставите нужный цвет (**Рис. 1.11**).

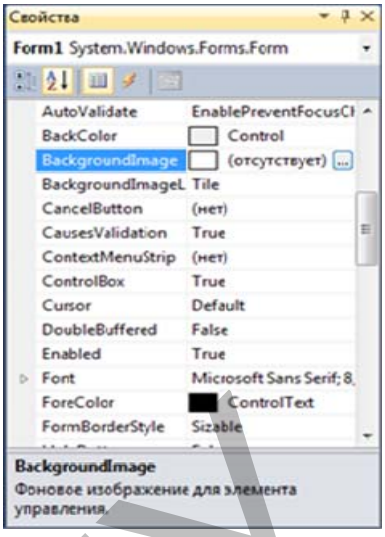

Рис. 1.10 – Свойство "BackGroundImage"

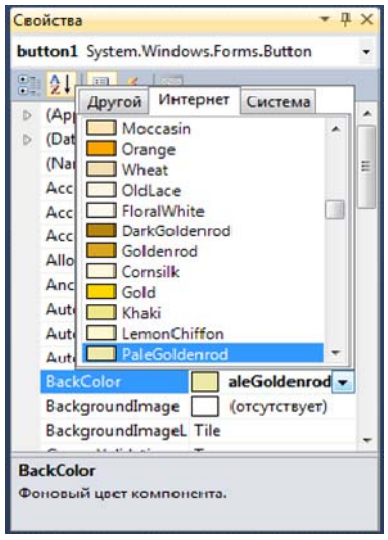

Рис. 1.11 – Свойство "BackColor"

Теперь нужно изменить шрифт элемента "label". Для этого нажмите на него и выберите свойство "Font", нажав на кнопку "…".

Далее выберите нужный размер и стиль шрифта (**Рис. 1.12**).

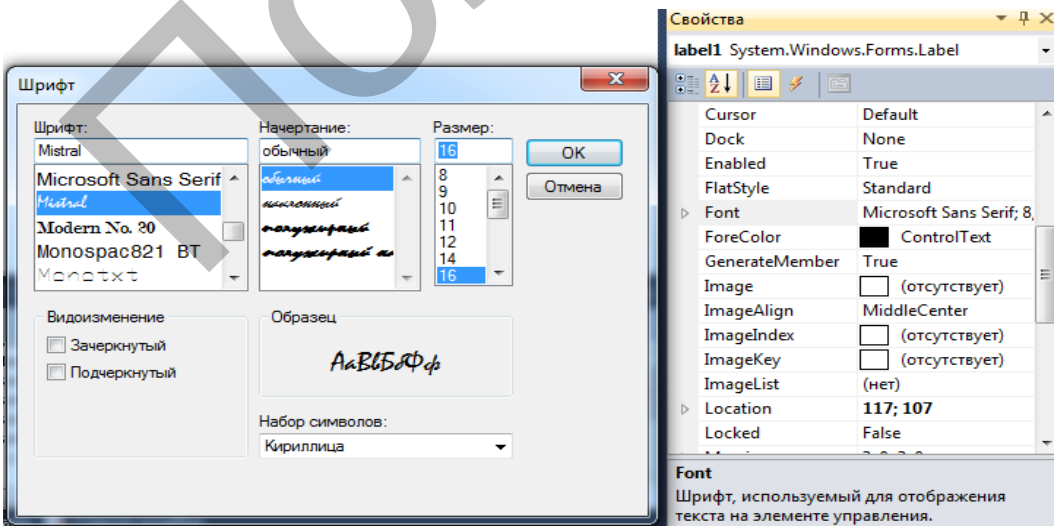

Рис. 1.12 – Свойство "Font"

Вот что должно получиться (**Рис. 1.13**).

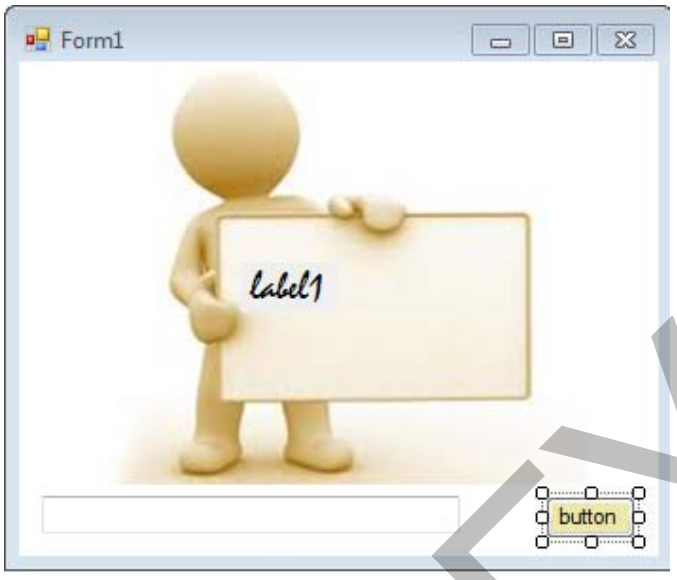

Рис. 1.13 – Свойство

Теперь перейдём к коду – он очень прост:

```
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
        {
this -> Text = "Доска объявлений";
label1 -> Text = "";
}

private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
        {
label1 -> Text = textBox1 -> Text;
// Если стиль шрифта стал опять обычный, то остановите программу и
// ещё раз измените шрифт элемента "label"
}

};
}
```

1.7 Загрузка изображения в PictureBox при помощи ComboBox

Суть программы следующая – есть четыре картинки (**Рис. 1.14**):

Рис. 1.14 – Необходимые изображения

Они лежат на диске или на флеш-носителе. На форме есть "comboBox", в котором находится список из четырёх слов.

При выборе одного из слов в списке должна появляться картинка, а в "label" её название – для того чтобы занести в "comboBox", нужно найти в панели свойств свойство "Items" и написать через "enter" слова (**Рис. 1.15**).

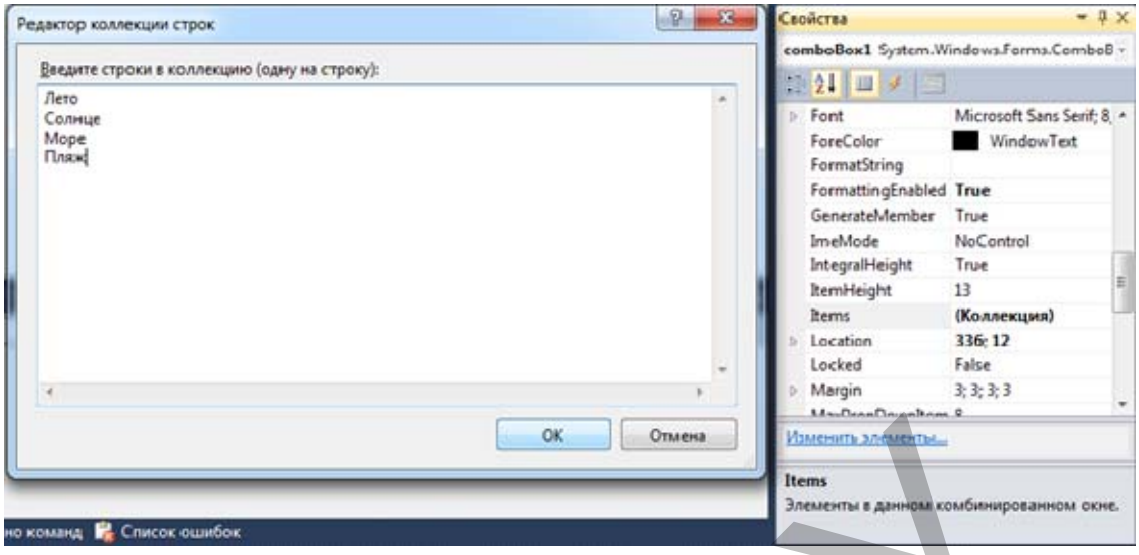

Рис. 1.15 – Свойство "Items"

Помимо "comboBox", перенесите на форму элементы – "lable" и "PictureBox".

Стиль текста "label1" вы можете выбрать сами (**Рис. 1.16**).

Рис. 1.16 – Заготовка программы

Перейдём к коду:

```
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{
   this -> Text = "Фотогалерея";
   label1 -> Text = "";
   comboBox1 -> Text = "Список";
```

```
}
private: System::Void comboBox1_SelectedIndexChanged (System::Object^ sender, System::EventArgs^ e)
{
  switch (comboBox1 -> SelectedIndex)
  {
    case 0: pictureBox1 -> Image = Image::FromFile("d:\\7.0.png");
    label1 -> Text = "Лето"; break;
    case 1: pictureBox1 -> Image = Image::FromFile("d:\\7.1.png");
    label1 -> Text = "Солнце"; break;
    case 2: pictureBox1 -> Image = Image::FromFile("d:\\7.2.png");
    label1 -> Text = "Море"; break;
    case 3: pictureBox1 -> Image = Image::FromFile("d:\\7.3.png");
    label1 -> Text = "Пляж"; break;
  }
}
```

1.8 Задание по первому разделу

Список заданий для самостоятельного выполнения (согласно заданию преподавателя):

1. Калькулятор энергопотребления. Приложение должно осуществлять расчет нагрузки на электросеть при подключении пользовательского набора электроприборов и отображать предупреждение в случае превышения допустимой безопасной нагрузки на проводку. Минимальная реализация:

– форма для составления пользовательского набора подключенных электроприборов;

– форма для добавления электроприборов и их характеристик в базу данных;

– база данных (в любой реализации) для хранения информации об электроприборах;

– форма настроек параметров сети.

2. План производства. Приложению должно осуществлять расчет оптимального плана производства продукции на предприятии реального сектора экономики в плановом пе-

риоде – на основании данных о ценах на продукцию, производственных издержках, производственных мощностях предприятия и / или любых других показателях, использование которых будет уместным. Кроме того, приложение должно осуществлять запись, хранение и предоставление информации о проведенных расчетах в базе данных (в любой реализации) с указанием даты. Минимальная реализация:

– форма ввода данных и предоставления результата;

– форма предоставления информации о расчетах;

– база данных (в любой реализации) для хранения информации о проведенных расчетах;

– форма настроек параметров (если необходимо).

3. Фотогалерея. Приложение должно предоставлять пользователю инструменты просмотра, оценки и работы с графическими изображениями.

Кроме того, приложение должно вести учет и предоставлять пользователю информацию обо всех операциях с изображениями (просмотр, редактирование, экспорт в др. приложения). Минимальная реализация:

– форма просмотра изображения (основной режим);

– форма просмотра изображений (слайд-шоу);

– форма просмотра информации о проведенных операциях;

– функция экспорта изображения в специализированные редакторы;

– база данных (в любой реализации) для хранения информации об оценках графических изображений;

– форма настроек параметров приложения.

4. Физический калькулятор. Приложение должно предоставлять пользователю набор инструментов для расчета физических величин. Выбор рассчитываемых величин (не менее 5), а также используемых уравнений (не менее 10) остается за студентом. Каждое уравнение должно быть представлено в отдельной форме (расчет различных физических величин с использованием одного уравнения должен быть реализован в одной форме).

Данные о проведенных расчетах, а также используемые табличные значения физических величин должны храниться в базе данных (в любой реализации). Минимальная реализация:

– формы для расчета физических величин с предоставлением информации об используемом уравнении, а также краткой справки обо всех фигурирующих в уравнении величинах – как выбираемых пользователем, так и табличных;

– форма для просмотра информации о проведенных операциях;

– форма для просмотра значений хранимых в базе данных табличных величин;

– база данных (в любой реализации) для хранения информации о проведенных расчетах и табличных величинах.

5. Система поддержки принятия решений (СППР). Приложение должно предоставлять инструменты для осуществления выбора (выбор сферы применения остается за студентом; варианты сфер применения: бюро знакомств, подбор персонала, выбор поставщика) на основании оценки не менее 8 параметров. Минимальная реализация:

– форма отчета, предоставляющая пользователю результат работы приложения;

– форма настроек приоритета и значимости параметров при оценке;

– форма инструментов управления базой кандидатов;

– база данных (в любой реализации), предоставляющая информацию о кандидатах.

6. Аудиопроигрыватель. Приложение должно предоставлять пользователю набор инструментов для воспроизведения и оценки аудиофайлов. Минимальная реализация:

– форма воспроизведения выбранного файла;

– форма плейлиста с функциями добавления / удаления композиций;

– база данных (в любой реализации), хранящая данные о пользовательских оценках файлов;

– форма настроек приложения.

7. Ежедневник. Приложение должно предоставлять пользователю набор инструментов для хранения информации и напоминания о важных событиях и мероприятиях. Минимальная реализация:

– форма, предоставляющая информацию о ближайших событиях;

– форма, предоставляющая набор инструментов для управления записями;

– база данных (в любой реализации) для хранения пользовательских записей;

– форма настроек параметров приложения.

8. Текстовый редактор. Приложение должно предоставлять пользователю набор инструментов для работы с текстовыми файлами. Минимальная реализация:

– форма редактирования текста;

– чтение и запись текстовой информации;

– форма, ориентированная только на чтение пользователем текстовой информации;

– форма настроек интерфейса;

– форма, предоставляющая информацию о выполненных приложением операциях с файлами;

– база данных (в любой реализации), предназначенная для хранения истории работы приложения.

9. Электронный словарь. Приложение должно предоставлять сведения о предметах и терминах определенной области (право выбора области оставляется за студентом). Каждая статья, если возможно, должна быть снабжена, по крайней мере, одной иллюстрацией. Минимальная реализация:

– форма навигации;

– форма вывода выбранной статьи;

– реализация поиска по последовательности символов;

– база данных (в любой реализации), содержащая предоставляемую приложением информацию;

– форма настроек интерфейса.

10. Программа-переводчик. Приложение должно предоставлять пользователю набор инструментов для перевода текстовой информации с одного языка на другой. Минимальная реализация:

– форма перевода слов;

– реализация прямого и обратного перевода с русского языка на английский;

– форма редактирования словарей;

– база данных (в любой реализации) для хранения словарей;

– форма настроек интерфейса.

11. Свой вариант. За студентом оставляется право на реализацию любого иного приложения на C ++ / CLI, если оно:

– не совпадает по тематике с одним из приведенных выше вариантов;

– его реализация включает в себя, по крайней мере:

- основные элементы управления Windows Forms;
- взаимодействие нескольких форм;
- взаимодействие с базой данных (в любой реализации);
- форму настроек пользовательского интерфейса;

– приложение может быть использовано в каких бы то ни было практических целях.

Примечание: приведенная в заданиях функциональность приложений является ориентировочной. Любые оригинальные идеи и решения приветствуются.

2. ВЗАИМОДЕЙСТВИЕ ФОРМ В ПРОЕКТАХ VISUAL STUDIO

Довольно часто окно приложения должно как взаимодействовать с другими окнами (формами), так и получать данные из окон стандартных системных диалогов.

2.1 Пример конструирования и программного вызова формы

```
Form ^ form2 = gcnew Form(); //gcnew и ^ в Managed C++ (C++ CLI) – аналоги new и *
Button^ button2 = gcnew Button();
button2 -> Text = L"OK";
button2 -> Location = Point(10,10);
form2 -> Text = L"Моё окно";
form2 -> HelpButton = true;
form2 -> FormBorderStyle = System::Windows::Forms::FormBorderStyle::FixedDialog;
//Одна из типовых проблем – указание пространства имен
//Если начинается с :: – то глобальный идентификатор
form2 -> StartPosition = FormStartPosition::CenterScreen;
form2 -> Controls -> Add( button2 );
form2 -> ShowDialog();
```

Для добавления обработчика нажатия программно сгенерированной кнопки button2 достаточно перед последней строкой кода написать:

```
button2 -> Click + = gcnew System::EventHandler (this, & Form1::button2_Click);
```

– до того, как будет вызван метод form2 -> ShowDialog() или form2 -> Show();

При этом код обработчика размещён в текущем модуле Form1.h:

```
private: System::Void button2_Click(System::Object^sender, System::EventArgs^e) {
MessageBox::Show("Here");
}
```

2.2 Вызов формы из формы

В меню выберем Проект – Добавить новый элемент – Форма – имя Form2

Добавим оператор:

```
#include "Form2.h"
```

перед первым namespace в Form1.h

Включим указатель на экземпляр класса в секцию public класса Form1:

```
Form2 ^ F2;
```

Добавим код там, где нужно создать и вызвать вторую форму:

```
F2 = gcnew Form2();
F2 -> Show();
```

Для программного удаления второй формы подойдёт код

```
delete F2;
```

Следует учесть, что указатель хранит адрес только одной формы – той, что создана последней. Если мы последовательно создали таким кодом несколько форм, удалена будет только последняя из них. Как вариант попробуйте массив форм.

Опишем нужные данные в классе формы Form1 (здесь имя и namespace проекта Tabulator, если нужно, замените своим):

```
static const int MAX_FORMS = 100; //Максимальное коли-
чество форм
int FormCount; //Счётчик форм
array <Tabulator::Form2 ^> ^F2; //Указатель на массив форм
```

Потом инициализируем данные по событию Load главной формы:

```
FormCount = 0;
F2 = gcnew array<Tabulator::Form2 ^>(MAX_FORMS);
```

Затем реализуем код для создания очередной формы:

```
if (FormCount<MAX_FORMS) {
  F2[FormCount++] = gcnew Form2();
  F2[FormCount-1] -> Show();
}
else MessageBox::Show("Слишком много форм!");
```

и её удаления:

```
if (FormCount) { delete F2[FormCount-1]; FormCount--; }
```

Если мы хотим создавать дочерние формы не отдельно, а внутри родительской формы, то в свойствах Form1 нужно указать, что она "предок" (установить свойство IsMdiParent = true), а перед показом дочерней формы оператором F2[FormCount-1] -> Show() пометить её как потомка Form1:

```
F2[FormCount-1] -> MdiParent = this;
```

2.3 Наладка взаимодействия родительской и дочерней форм

Так как нам из первой формы нужно иметь доступ ко второй, а из второй к первой, то будет возникать проблема перекрестных ссылок (когда Form1.h ссылается на Form2.h, который, в свою очередь, вновь ссылается на Form1.h).

Нам едва ли обойтись без привлечения файлов .cpp.

Распишем процесс по шагам.

1) Имеются 2 формы – Form1 и Form2, на Form1 располагаются Button (button1, будет открывать вторую форму) и Label (label1, здесь будем менять текст). На Form2 – button1, по нажатии которой будет происходить смена текста в label1.

2) Так как нам из первой формы нужно иметь доступ ко второй, а из второй к первой, то будет возникать указанная выше проблема перекрестных ссылок. Чтобы этого избежать, код первой формы (Form1), который будет иметь доступ ко второй форме (Form2), мы вынесем из .h-файла в .cpp файл. Таким образом, нужно создать файл Form1.cpp (Проект – Добавить новый элемент – Файл C++ – Form1).

3) Объявить открытый метод Set в Form1.h для того, чтобы можно было изменить текст label1 (код можно написать в конце файла, после #pragma endregion):

```
public: void Set(String^ text) {
label1 -> Text = text;
}
```

4) В файле Form2.h подключаем Form1.h (в начале):

```
#include "Form1.h"
```

и создаем конструктор, который будет принимать и сохранять ссылку на первую форму для дальнейшего использования:

```
Form2(Form1^ parent){
InitializeComponent();
parentForm = parent;
}
//ниже сразу можно прописать ссылку: private: Form1^ parentForm;
```

5) По клику кнопки в Form2 будем вызывать метод Set родительской формы:

```
private: System::Void button1_Click(System::Object^sender, System::EventArgs^e) {
  parentForm -> Set("hello from form2");
  parentForm -> Show();
  this -> Hide();
}
```

6) Осталось в первой форме сделать открытие второй формы. Для этого из Form1.h обработчик нажатия кнопки переносим в Form1.cpp, а в .h-файле оставляем только его объявление.

Код в файле Form1.cpp:

```
#include "StdAfx.h"
#include "Form1.h"
#include "Form2.h"
namespace ChildToParent {
 System::Void  Form1::button1_Click(System::Object^sender,
 System::EventArgs^e) {
  Form2^ f2 = gcnew Form2(this);
  f2 -> Show();
  this -> Hide();
 }
}
```

В Form1.h вставляем только строку:

```
private: System::Void button1_Click(System::Object^ sender,
System::EventArgs^ e);
```

2.4 Загрузка файла в текстовое поле, использование стандартных диалогов

На форму добавлено:

– многострочное поле textBox1 (MultiLine = true);

– кнопки для открытия (button1) и закрытия (button2) файлов;

– диалоги openFileDialog1 и saveFileDialog1.

код:

```
private: System::Void button1_Click(System::Object^sender, System::EventArgs^e) {
openFileDialog1 -> ShowDialog();
 if (openFileDialog1 -> FileName = = nullptr) return;
```

```
        try { // Создание экземпляра StreamReader для чтения
        из файла
          System::Text::Encoding^ Coding = System::Text:: Encod-
          ing::Get Encoding(1251);
          auto Reader = gcnew IO::StreamReader(openFileDialog1 ->
          File Name,Coding);
          textBox1 -> Text = Reader -> ReadToEnd();
          Reader -> Close();
          textBox1 -> Modified = false;
        }
        catch (IO::FileNotFoundException^ e) {
          MessageBox::Show(e -> Message + "\nНет такого файла",
          "Ошибка",
            MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
        }
        catch (Exception^ e) { // Отчет о других ошибках
          MessageBox::Show(e -> Message, "Ошибка",MessageBox
          Buttons::OK,MessageBoxIcon::Exclamation);
        }
      }

void Write (void) {
  try { // Создание экземпляра StreamWriter для записи в файл:
    System::Text::Encoding^ Coding = System::Text::Encoding::Get
    Encoding("utf-8");
            // заказ кодовой страницы UTF-8
      auto Writer = gcnew IO::StreamWriter(saveFileDialog1 -> File
      Name, false, Coding);
      Writer -> Write(textBox1 -> Text);Writer -> Close();
      //или IO::File::WriteAllText(saveFileDialog1 -> FileName, text
      Box1 -> Text); вместо этих строк
      textBox1 -> Modified = false;
    }
    catch (Exception^ e) {
      MessageBox::Show(e -> Message, "Ошибка", MessageBox-
      Buttons::OK,MessageBoxIcon::
      Exclamation);
```

```
  }
 }

 private: System::Void button2_Click(System::Object^sender,
 System::EventArgs^e) {
  saveFileDialog1 -> FileName = openFileDialog1 -> FileName;
  if (saveFileDialog1 -> ShowDialog() = = Windows::Forms::
  DialogResult::OK) {
   Write();
  }
 }

 private: System::Void textBox1_KeyUp(System::Object
 ^sender,
  System::Windows::Forms::KeyEventArgs^e) {
  if (Char::IsControl(e -> KeyValue) = = false) textBox1 ->
  Modified = true;
 }
 private: System::Void Form1_FormClosing(System::Object
 ^sender,
  System::Windows::Forms::FormClosingEventArgs^e) {
  if (textBox1 -> Modified = = false) return;
  auto MBox = MessageBox::Show("Текст был изменен.
  \nСохранить изменения?","Простой редактор", Message-
  Box Buttons::YesNoCancel,MessageBoxIcon::Exclamation);
  if (MBox = = Windows::Forms::DialogResult::No) return;
  if (MBox = = Windows::Forms::DialogResult::Cancel) e ->
  Cancel = true;
  if (MBox = = Windows::Forms::DialogResult::Yes) {
   if (saveFileDialog1 -> ShowDialog() = = Windows::Forms::
   DialogResult::OK) {Write(); return; }
   else e -> Cancel = true; // Передумал выходить
  } // DialogResult::Yes
 }
```

Задание: см. «Список заданий для самостоятельного выполнения».

3. РАССМОТРЕНИЕ РЯДА ЭЛЕМЕНТОВ ВИЗУАЛИЗАЦИИ VISUAL STUDIO

3.1 Элемент MenuStrip и свойство Anchor

Здесь будет рассмотрен довольно важный и распространённый элемент "MenuStrip", который вы можете встретить: в текстовом редакторе, выбирая "Файл –> Сохранить как", в графическом редакторе "Файл –> Вставить" и во многих других программах. "MenuStrip" – это, по сути, выпадающее меню, в котором есть определённые пункты.

Также будет рассмотрено свойство "Anchor" – это свойство, при котором определяется, к каким сторонам формы будет привязан элемент: если к правой, то при увеличении или уменьшении размера формы правая сторона "textBox" будет также увеличиваться или уменьшаться; если поставлена кнопка "button", то она станет перемещаться за стороной формы, к которой привязана.

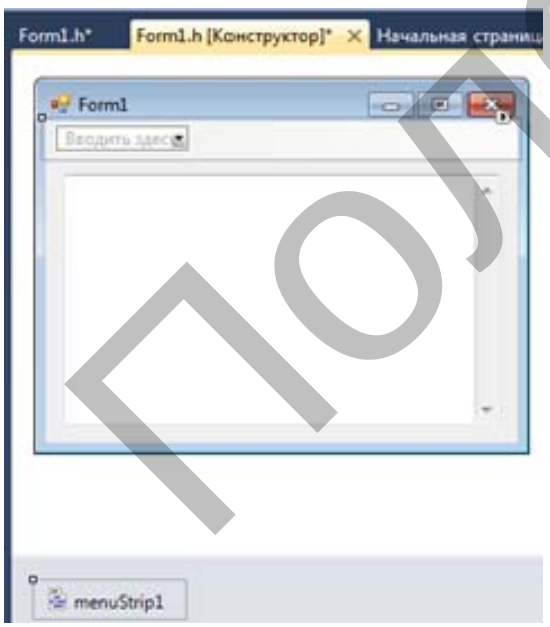

Рис. 3.1 – Изображение элемента "MenuStrip" и элемент "textBox" на форме

Перенесите на форму элемент "Menu-Strip" и элемент "text Box". У элемента "textBox" включите "Multiline" и свойство "Scroll-Bars" –> "Vertical (**Рис. 3.1**). Далее нажмите на текстовое поле "Вводить здесь" и напишите "Файл" (**Рис. 3.2**). После этого в нижнем, новом появившемся текстовом поле, напишите "Открыть", после чего появится ещё одно, в котором напишите "Сохранить как" (**Рис. 3.3**).

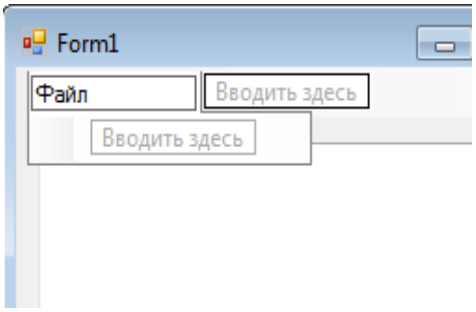

Рис. 3.2 – Ввод начений в элемент "MenuStrip"

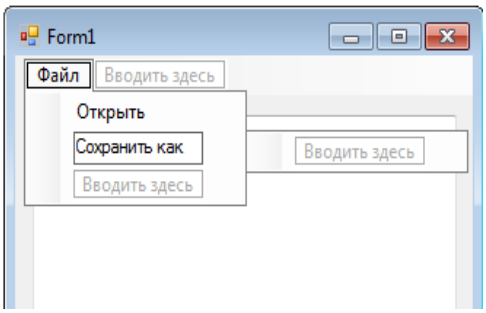

Рис. 3.3 – Ввод значений в элемент "MenuStrip"

Теперь займёмся свойством "Achor" – для этого нажмите на "textBox". После чего на панели свойств, расположенной слева, выберите свойство "Achor" (**Рис. 3.4**).

Как в можете видеть – "textBox" привязан к левой и верхней части формы.

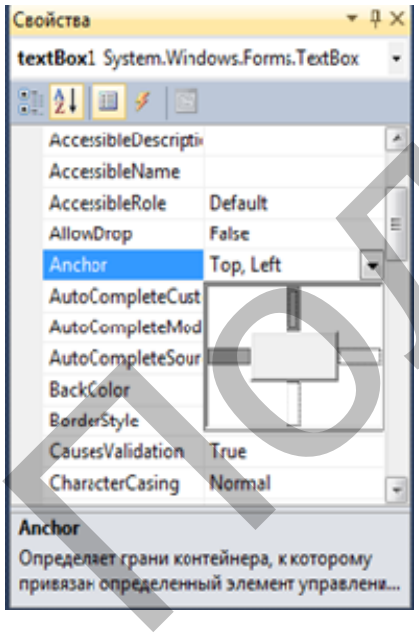

Рис. 3.4 – Использование свойства "Achor"

Теперь выделите также правую и нижнюю часть, после чего размер элемента "textBox" будет изменяться пропорционально размеру формы (**Рис. 3.5**).

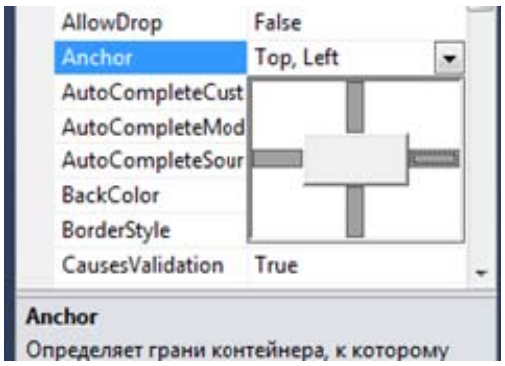

Рис. 3.5 – Использование свойства "Achor"

Запустите программу и проверьте изменённые свойства (**Рис. 3.6, 3.7**).

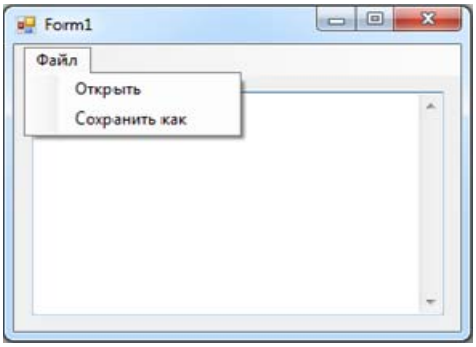

Рис. 3.6 – Проверка свойств элемента "MenuStrip"

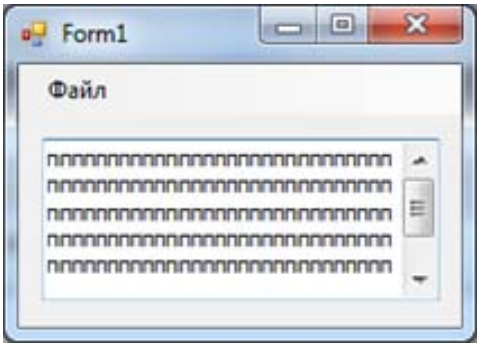

Рис. 3.7 – Проверка свойства "Achor"

3.2 Открытие и запись текстового файла

В данной программе мы будем открывать и редактировать какой-либо уже созданный текстовый файл или же сами писать текст и сохранять его, как новый текстовый файл, в нужную вам папку. Ещё один явный признак текстового редактора: вы что-то написали и нажимаете крестик, чтобы выйти, при этом программа спрашивает: «Сохранить изменения?». Для этого понадобятся следующие элементы: "MenuStrip", "textBox", "openFile-dialog", "saveFiledialog". Перетащите все эти элементы на форму, назовите заголовок "Menu-Strip" "Файл", создайте в нём три пункта: "Открыть", "Сохранить как", "Выход", привяжите "textBox" ко всем сторонам формы, если не знаете, включите "Multeline" и "Scroll-Bars –> Vertical" (**Рис. 3.8**). В коде программы будут созданы **"MyReader"** и **"MyWriter"**, с помощью которых программа будет читать и записывать текст в файл. Помимо этого, в коде выбирается кодировка, благодаря которой программа будет понимать русский текст.

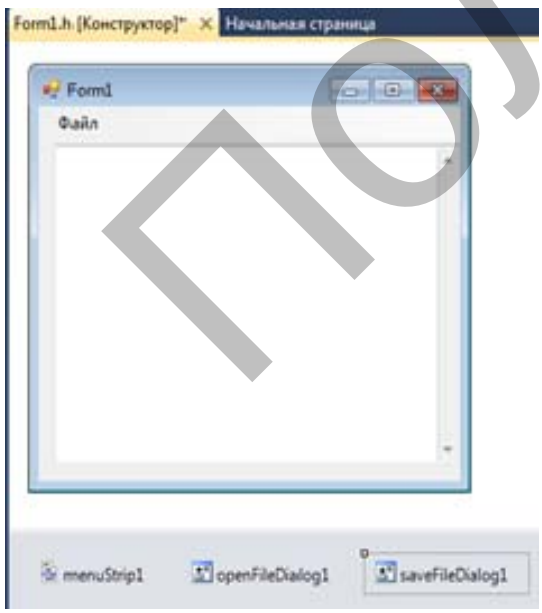

Рис. 3.8 – Форма текствого редактора

У формы нужно вызвать событие "FormClosing".

Код программы:

```
#pragma endregion
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
    {
  this -> Text = "Текстовый редактор";
  openFileDialog1 -> FileName = "D:\\ВУЗ\\Text2.txt";
  openFileDialog1 -> Filter = "Текстовые файлы (*.txt)|*.txt|All files (*.*)|*.*";
  saveFileDialog1 -> Filter = "Текстовые файлы (*.txt)|*.txt|All files (*.*)|*.*";
}

private: System::Void открытьToolStripMenuItem_Click(System::Object^ sender, System::EventArgs^ e)
    {
  openFileDialog1 -> ShowDialog();
  if (openFileDialog1 -> FileName = = nullptr) return;
  try
  {
    auto MyReader = gcnew IO::StreamReader(openFileDialog1 -> FileName, System::Text::Encoding::GetEncoding(1251));
    textBox1 -> Text = MyReader -> ReadToEnd();
    MyReader -> Close();
  }
  catch (IO::FileNotFoundException^ Ситуация)
  {
    MessageBox::Show(Ситуация -> Message+"\nФайл не найден", "Ошибка" MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
  }
    catch (Exception^ Ситуация)
  {
```

```
    MessageBox::Show(Ситуация -> Message, "Ошибка", Mes-
    sageBoxButtons::OK, MessageBoxIcon::Exclamation);
  }
}
private: System::Void сохранитьКакToolStripMenuItem_Click
(System::Object^ sender, System::EventArgs^ e)
    {
  saveFileDialog1 -> FileName = openFileDialog1 -> FileName;
  if (saveFileDialog1 -> ShowDialog() = = Windows::Forms::Dia-
  logResult::OK) Save();
}
void Save()
{
  try
  {
// Создание экземпляра StreamWriter для записи в файл:
    auto MyWriter = gcnew IO::StreamWriter(saveFileDialog1 ->
    FileName, false, System::Text::Encoding::GetEncoding(1251));
    MyWriter -> Write(textBox1 -> Text);
    MyWriter -> Close(); textBox1 -> Modified = false;
}
catch (Exception^ Ситуация)
{
  MessageBox::Show(Ситуация -> Message, "Ошибка", Mes-
  sageBoxButtons::OK, MessageBoxIcon::Exclamation);
  }
}

private: System::Void выходToolStripMenuItem_Click(System::Ob-
ject^ sender, System::EventArgs^ e)
      {
  this -> Close();
}
private: System::Void Form1_FormClosing(System::Object^ send-
er, System::Windows::Forms::FormClosingEventArgs^ e)
      {
  if (textBox1 -> Modified = = false) return;
```

```
    auto MeBox = MessageBox::Show("Текст был изменён. \n Со-
    хранить изменения?", "Простой редактор", MessageBoxBut-
    tons::YesNoCancel, MessageBoxIcon::Exclamation);
    if (MeBox = = Windows::Forms::DialogResult::No) return;\
    if (MeBox = = Windows::Forms::DialogResult::Cancel) e -> Can-
    cel = true;
    if (MeBox = = Windows::Forms::DialogResult::Yes)
    {
      if (saveFileDialog1 -> ShowDialog() == Windows::Forms:: Di-
      alogResult::OK)
      {
        Save(); return;
      }
      else e -> Cancel = true;
    }
}
```

3.3 Создание простейшего Веб-браузера

Сейчас вы увидите, как создать свой, пусть и не с огромным количеством функций, но всё же полностью рабочий web-браузер. В его функции будут входить: переход на сайт по указанному адресу в текстовом поле, переход на страницу назад или вперёд, переход на домашнюю страницу, выбор одного из сайтов, занесённых в «избранное» (в список элемента "ComboBox"), и кнопка печати отображаемой страницы.

Некоторые пользователи думают, что для того, чтобы загрузить интернет-страничку, нужно будет писать огромный код. Делать этого совершенно не придётся.

В "Windows Forms" есть уже готовый элемент – "WebBrowser", который нужно просто перенести на форму и нужным образом связать с другими элементами. Для этого понадобятся: 5"button", 1""textBox", 1"WebBrow-ser" и 1"ComboBox". Перенесите все эти элементы на форму, как показано ниже; если элемент "WebBrowser" растянулся на всю форму – кликните на указатель в WebBrowser «Задачи»: "Открепить в родительском контейнере" (**Рис. 3.9**).

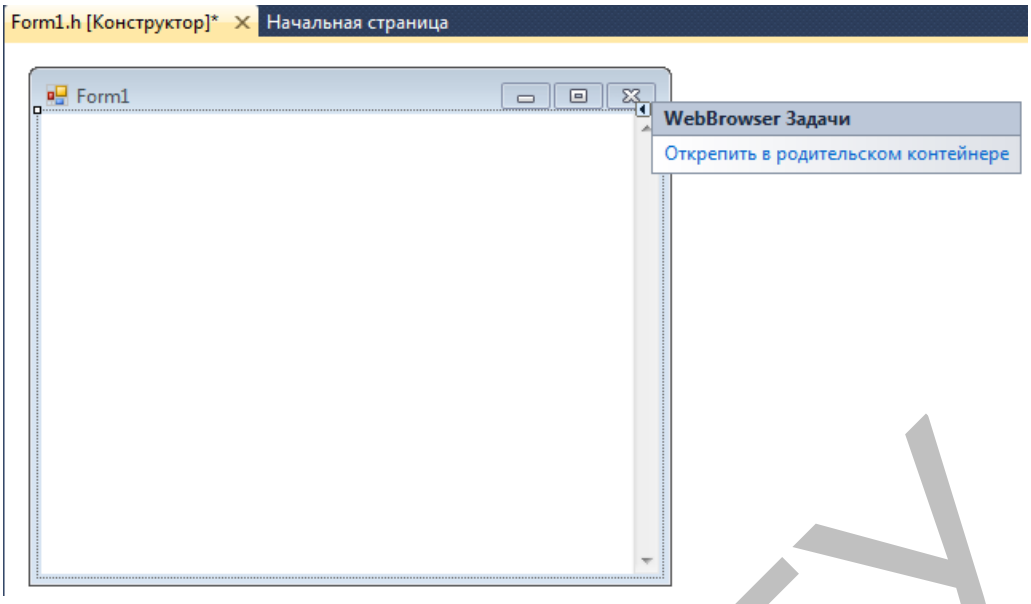

Рис. 3.9 – Открытие родительского контейнера

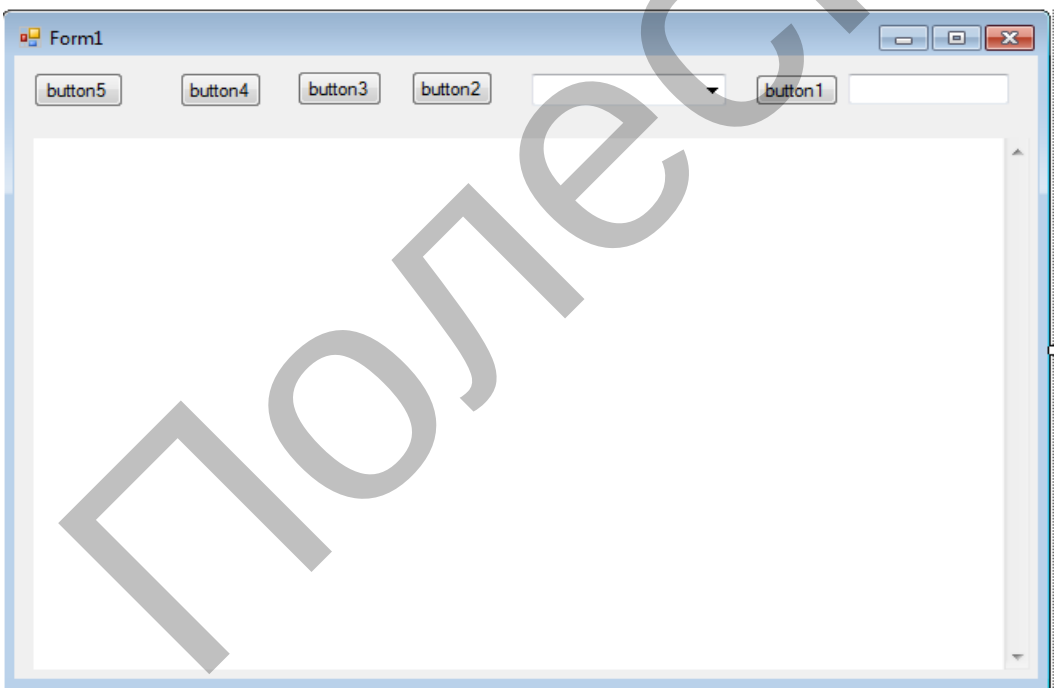

Рис. 3.10 – Готовая форма "WebBrowser"

У элемента "textBox" в свойстве "Anchor" к уже выделенным "Top, Left" добавьте "Rich", выделив его.

Теперь в свойстве "Text" задайте всем кнопкам "button" соответствующие названия:

Button1 – "Поиск";
Button2 – " –> ";
Button3 – "<–";
Button4 – "Home";
Button5 – "Печать";

У элемента "comboBox" в свойстве "Text" напишите: "Избранное", далее в свойстве "Items" напишите следующий список:

www.vk.com
olocoder.ru
youtube.com
google.com
yandex.ru
www.mail.ru

Вид формы программы после изменений представлен на **Рис. 3.11.**

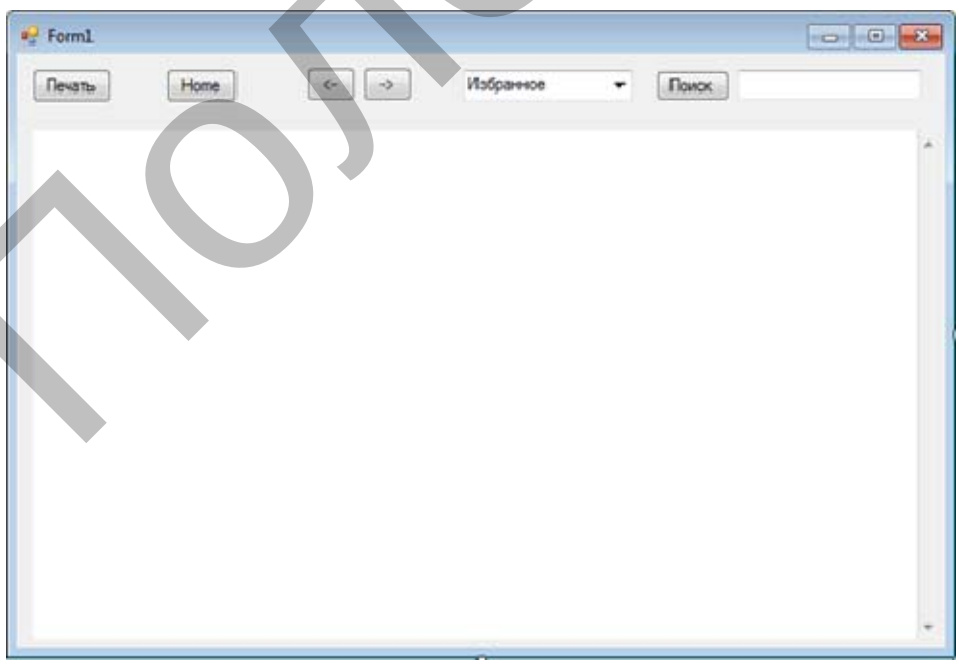

Рис. 3.11 – Форма "WebBrowser" после изменений

Теперь перейдём к коду программы:

```
#pragma endregion
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e) {
this -> Text = "My BrowWwser";
}

private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {
webBrowser1 -> Navigate(textBox1 -> Text);
}

private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e) {
webBrowser1 -> GoForward();
}

private: System::Void button3_Click(System::Object^ sender, System::EventArgs^ e) {
webBrowser1 -> GoBack();
}

private: System::Void button4_Click(System::Object^ sender, System::EventArgs^ e) {
webBrowser1 -> Navigate("olocoder.ru");
}

private: System::Void button5_Click(System::Object^ sender, System::EventArgs^ e) {
this -> webBrowser1 -> Print();
}

private: System::Void comboBox1_SelectedIndexChanged (System::Object^ sender, System::EventArgs^ e) {
switch (comboBox1 -> SelectedIndex)
{
```

```
case 0: webBrowser1 –> Navigate("www.vk.com"); break;
case 1: webBrowser1 –> Navigate("olocoder.ru"); break;
case 2: webBrowser1 –> Navigate("www.youtube.com"); break;
case 3: webBrowser1 –> Navigate("www.google.com"); break;
case 4: webBrowser1 –> Navigate("www.yandex.ru"); break;
case 5: webBrowser1 –> Navigate("www.mail.ru"); break;
}
}

};
}
```

3.4 Создание базы данных элемент DataGrindView

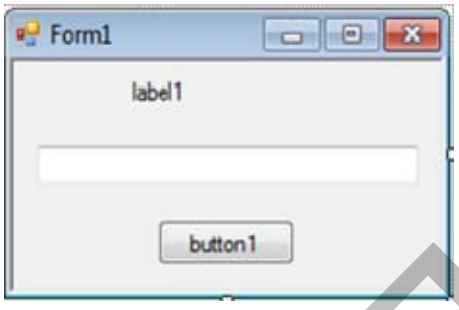

Рис. 3.12 – Вид форм программы "Form1"

Суть данной программы следующая: на "Form1" находится тестовое поле, куда нужно ввести пароль, и кнопка, после нажатия которой программа проверяет введённый пароль. Если пароль – «admin», то появится вторая форма, где будет находиться таблица и активная кнопка "Сохранить". Пользователь вводит данные в таблицу, активирует кнопку, после чего в папке проекта создаётся документ baza.xml, где будут храниться данные, которые можно дополнять или удалять, сохраняя изменения. Если пользователь вводит пароль «reader», то появляется вторая форма – "form2", с той же таблицей и данными, но кнопка «Сохранить» неактивна – пользователь может только просматривать данные, но не изменять. Если же пароль неверный, то появляется окошко "Incorrect password" (**Рис. 3.13**).

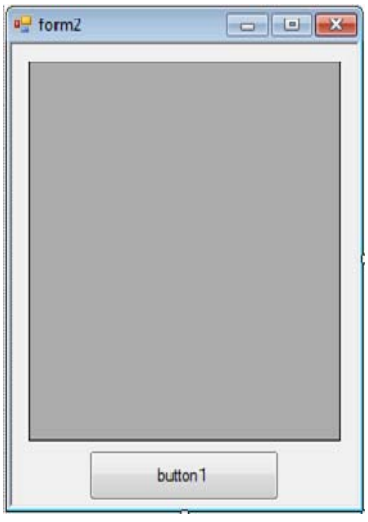

Рис. 3.13 – Вид форм программы "Form2"

Создайте проект в приложении "Windows Forms": на "Form1" перенесите – "label", "textBox" и "button".

Добавьте новую форму "form2" и перенесите на неё "DataGrindView" и "button".

Когда первая форма становится невидима, появляется вторая, но, закрыв вторую форму, первая не закроется.

Поэтому, если вы проверяете программу в компиляторе, а не запускаете отдельно, то просто «перезайдите» в компилятор, тогда проект будет закрываться.

Если вы запускаете программу отдельно из папки "Debug" (.exe), тогда всё будет в порядке.

Код программы:

--- Form1:

```
#pragma once
#include "form2.h"

namespace BD_Number {

using namespace System;

#pragma endregion
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e) {
this -> Text = "Вход в БД";
label1 -> Text = "Введите пароль";
button1 -> Text = "Ввод";
textBox1 -> PasswordChar = '*';
textBox1 -> TextAlign = HorizontalAlignment::Center;
}

private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {
form2^ gform2 = gcnew form2;
gform2 -> Show();
if (textBox1 -> Text = = "reader")
{
gform2 -> Visible = true;
```

```
this -> Visible = false;
gform2 -> button1 -> Enabled = false;
}
if (textBox1 -> Text = = "admin")
{
gform2 -> Visible = true
this -> Visible = false;
gform2 -> button1 -> Enabled = true;
}
if (textBox1 -> Text ! = "reader" && textBox1 -> Text ! = "admin")
{
gform2 -> Visible = false;
MessageBox::Show("Incorrect password!");
}
}
};
}
--- Form2

private: System::Windows::Forms::DataGridView^dataGridView1;
public: System::Windows::Forms::Button^ button1;
protected:
private:
/// <summary>
/// Требуется переменная конструктора.
/// </summary>
.
.
.
#pragma endregion

DataTable ^ Таблица;
DataSet ^ ВводДанных;

private: System::Void form2_Load(System::Object^ sender, System::EventArgs^ e) {
this -> Text = "Телефонная книга";
button1 -> Text = "Сохранить";
```

```
this -> Visible = false;
Таблица = gcnew DataTable();
ВводДанных = gcnew DataSet();
if (IO::File::Exists("baza.xml") = = false)
{
dataGridView1 -> DataSource = Таблица;
Таблица -> Columns -> Add("Имена");
Таблица -> Columns -> Add("Номера Телефонов");
ВводДанных -> Tables -> Add(Таблица);
}
else
{
ВводДанных -> ReadXml("baza.xml");
String ^ СтрокаXML = ВводДанных -> GetXml();
dataGridView1 -> DataMember = ("Название таблицы");
dataGridView1 -> DataSource = ВводДанных;
}
}
```

private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {

```
Таблица -> TableName = "Название таблицы";
ВводДанных -> WriteXml("baza.xml");
}

};
}
```

Задание: см. «Список заданий для самостоятельного выполнения».

4. СОЗДАНИЕ ПРОГРАММНЫХ ПРИЛОЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ ORM ENTITY FRAMEWORK

С английского ORM – это objectrelation mapping, или, по-русски, объектно-реляционное отражение. Зачастую данные в базе данных хранятся не совсем в том виде или структуре, с которой нам (разработчикам) хотелось бы работать, а так, как от нас требуют модели нормализации реляционных БД, либо данные в БД хранятся в структуре таблиц, повышающей производительность.

Давайте вспомним, как мы работали с данными с использованием технологии ADO.NET. Мы создавали подключение к БД (SqlConnection), выбирали параметры (если нужна была полная манипуляция) и создавали объекты SqlCommand. Когда запрос содержал параметры (SqlParameter), то создавали и их – объекты наборов данных (DataSet, DataTable), создавали объекты SqlDataAdapter; открывали и закрывали «ручками» подключения к БД, заполняли из адаптера наборы данных – и только лишь потом могли работать с данными.

Модель данных – это некое представление той структуры и иерархии конечных объектов, которое нам предоставляет технология доступа к данным. Программируя на объектно-ориентированных языках программирования высокого уровня, таких как Java и C#, даталогическая модель в редких случаях является полезной, так как не описывает объект как автономную, полную и законченную объект-сущность, композицию (речь идет не об объектах представления View Model архитектурного паттерна MVVM). Нам не нужны объекты таблицы БД, а нужны полноценные объекты предметной области, концепции – как раз таки набор данных объектов, их связи и зависимости представляют собой концептуальную модель.

EF предоставляет нам такую возможность – манипуляцию концептуальной моделью. Помимо богатого, насыщенного и гибкого функционала EF, к его плюсам можно также добавить и отличную интеграцию с родной нам IDE Visual Studio (далее VS). Туда входит: генерирование БД из классов (сущностей) (подход Code First), генерирование классов (сущно-

стей) из БД (подход Code First from Database), генерирование концептуальной диаграммы и модели (подход Database First), проектирование модели в конструкторе VS и последующая генерация БД, миграция БД прямо из VS.

На сегодняшний день, думаю, неспроста EF – официальная и первичная технология Microsoft для доступа к данным.

Кроме EF существуют и другие ORM для .NET Framework (далее .NET), такие как NHibernate (перекочевала из Java как Hiber-nate) и LINQ to SQL (даталогическая модель).

4.1 Основы работы ORM Entity Framework

Перед тем как работать с EF, давайте разберемся, как его устанавливать.

EF – это набор dll-библиотек, которые проще всего установить, используя менеджер пакетов NuGet (далее NuGet). Но можно и просто выкачать библиотеки и подключить их «ручками» к проекту. Для того, что бы установить EF в решение через NuGet, необходимо кликнуть правой клавишей мыши по проекту и далее –> Manage NuGet Packages –> Tab "Browse" –> Search "Entity Framework" –> Click Install (**Рис. 4.1**).

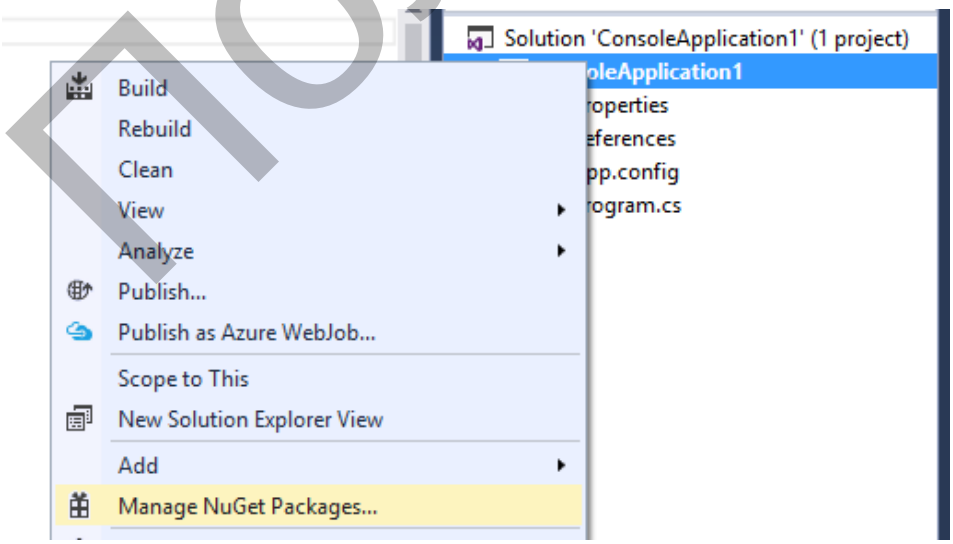

Рис. 4.1 – Вход в NuGet

После успешной установки EF проект вы должны увидеть в окне Output сообщение "Finished", в проекте появятся ссылки на сборки EntityFramework, EntityFramework.SqlServer (**Рис. 4.2**).

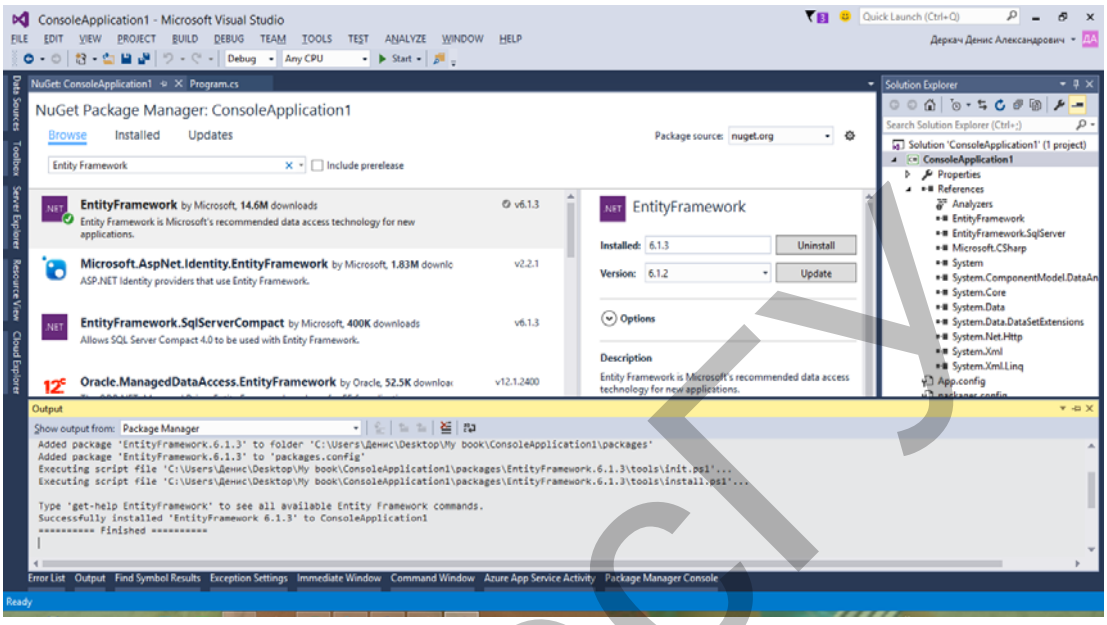

Рис. 4.2 – Результат установки EF

Если устанавливать «ручками», то необходимо проделать следующие шаги: кликнуть в проекте правой кнопкой мыши по References –> Add reference –> Browse –> Browse –> выбрать dll-библиотеки EntityFramework, EntityFramework.SqlServer из файловой системы (**Рис. 4.3**).

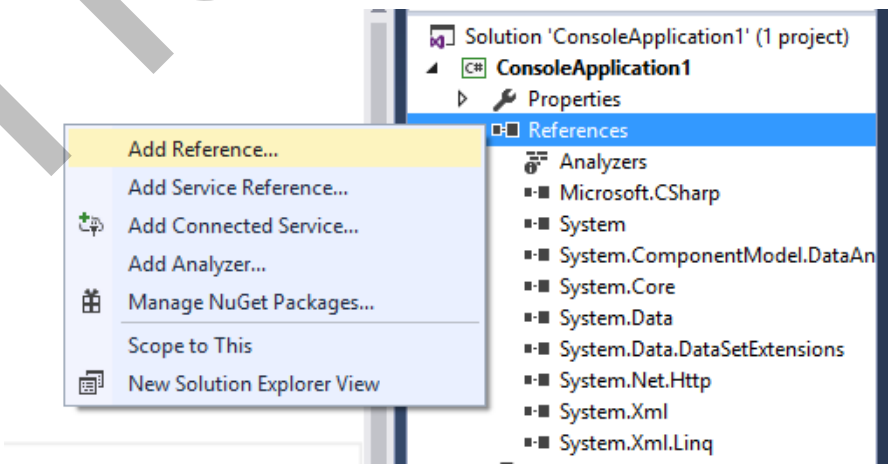

Рис. 4.3 – Переход к подключению библиотек к проекту

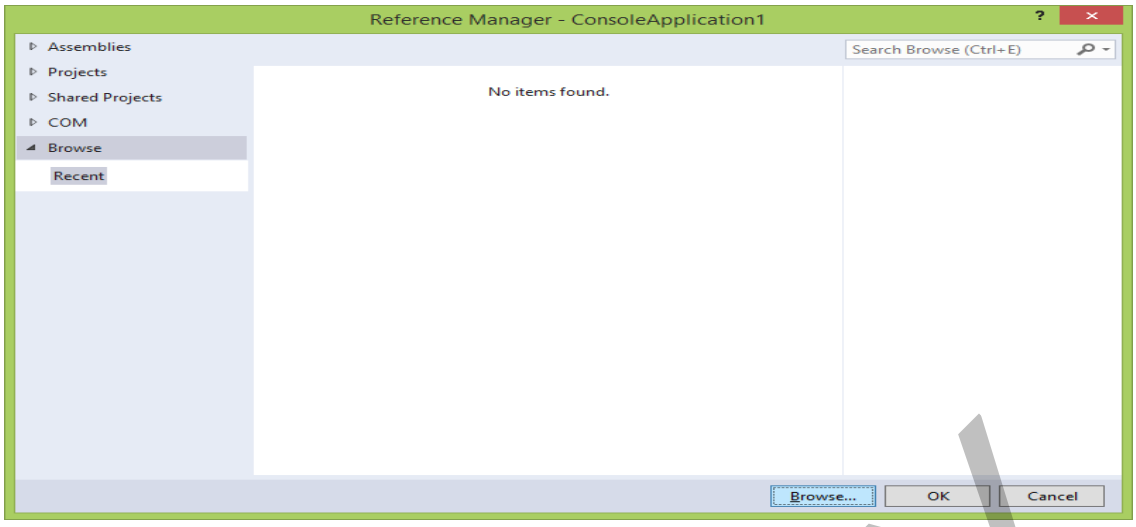

Рис. 4.4 – Добавление библиотек EF в ссылки проекта

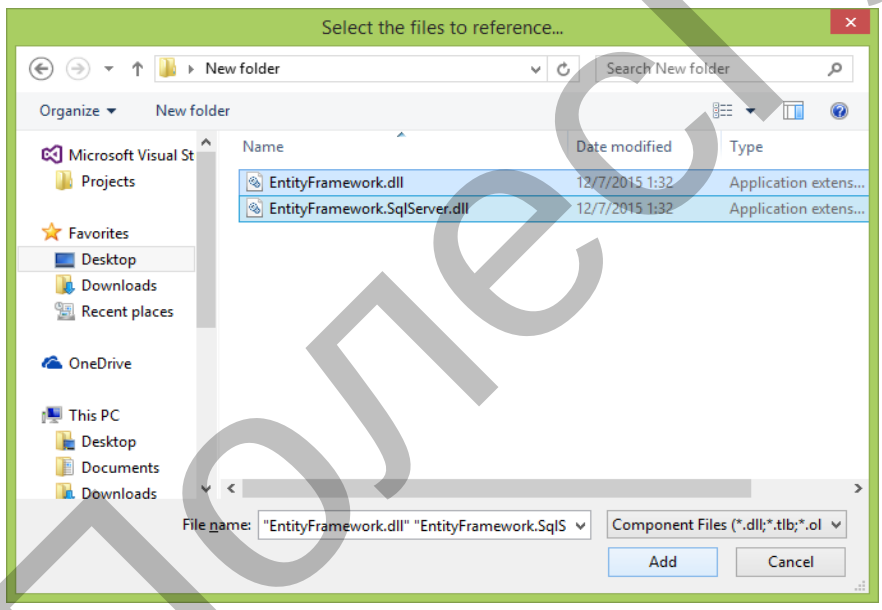

Рис. 4.5 – Выбор библиотек EF из файловой системы

EF 6 входит в состав .NET 4.5 и поставляется дефолтно (по умолчанию) с VS2013 и выше, но если хотите использовать самую «свежую» версию, то стоит воспользоваться NuGet и иметь при себе подключение к сети Интернет. EF дефолтно подключается к проектам типа ASP.NET и используется как первичный и интегрируемый поставщик данных к моделям представления Razor. В проектах этого типа имеет смысл только обновить EF до более новой версии через NuGet.

4.2 Структура и компоненты EF

К основным компонентам EF можно отнести контекст данных (Context), его сущности (Entities) и FluentAPI (интерфейс управления сущностями).

Контекст данных представляет собой некую «коробочку», в которую складываются сущности, при этом эта «коробочка» умеет работать с хранилищем, запросами, определять, изменился ли объект сущности, и др. функции.

Сущность EF – это описание объекта; класс, который взаимодействует с контекстом данных. Правильные сущности должны полностью описывать объект. Сущности EF могут отражать конечный объект как из одной, так и нескольких таблиц, декорироваться специальными управляющими атрибутами, управлять связями и ограничениями, наследоваться TPH, TPT, TPC.

FluentAPI – интерфейс контекста, выполняющий широкий спектр операций – как сущностей, так и контекста (направлению и инициализации связей, установки целостных ограничений, первичных и внешних ключей и др.).

4.3 Создание контекста данных

Давайте создадим новое решение с именем "EfStruct" типа "Console Application" и подключим к нему EF-библиотеки.

В корне проекта создадим класс с именем "AppContext" и подключим в файле класса сборку EntityFramework.dll, импортируя пространство имен конструкцией using System.Data.Entity. Также унаследуем базовый класс контекста EF следующим образом "AppContext:DbContext". Должен получиться класс (**Рис. 4.6**). Сразу возникает вопрос, зачем наследовать класс DbContext?

```
using System.Data.Entity;

namespace EfStruct
{
    class AppContext:DbContext
    {
    }
}
```

Рис. 4.6 – Класс контекста AppContext

DbContext – базовый класс EF, содержащий в себе реализацию логики для работы с БД, ее инициализации, манипуляцией сущностями, инициализацией БД и др. Наследуя класс DbContext, мы тем самым говорим, что хотим использовать EF как поставщик данных. Одна из перегрузок конструктора базового класса DbContext должна принимать один входящий параметр типа «string». Этот параметр может быть как имя строки подключения в файле ".config" (о нем поговорим позже) или же готовая строка подключения, причем, если мы передаем название строки подключения из файла ".config", то формат входящей строки должен иметь следующий вид "name = имя_строки_ подключения". Рефакторив наш класс AppContext для строки подключения с именем "ConnectionString" и входящим параметром самой строки подключения, должны получиться классы (**Рис. 4.7**, **4.8**).

```
using System.Data.Entity;

namespace EfStruct
{
    class AppContext:DbContext
    {
        public AppContext():base("name=ConnectionString")
        {
        }
    }
}
```

Рис. 4.8 – Передача имени подключения в базовый конструктор

```
using System.Data.Entity;

namespace EfStruct
{
    class AppContext:DbContext
    {
        public AppContext():base("Data Source=derk_pc;Initial Catalog=HealthDB;Integrated Security=True")
        {
        }
    }
}
```

Рис. 4.9 – Передача строки подключения в базовый конструктор

Возникает вопрос: зачем, в таком случае, передавать строку подключения к БД в конструктор, если можно просто передать название строки подключения, описанной в файле .config?

Зачастую сервера БД имеют непостоянный адрес или с течением времени разворачиваются на другом сервере, тогда приложению необходимо динамически формировать строку соединения (например, получать надстройки сервера БД из пользовательского файла настроек при запуске приложения). Это позволяет не пересобирать приложения при изменении адреса БД, что добавляет приложению гибкости.

4.4 Создание сущностей EF(Entity)

Сущность – это класс, который описывает концептуальный объект, а с ним и таблицу (иногда не одну!). Меньше слов – больше дела! Давайте усовершенствуем наш проект и добавим в него один класс "Faculty" и добавим в него два открытых свойства – public int Id, public string Title. (**Рис. 4.10**).

```
class Faculty
{
    public int Id { get; set; }
    public string Title { get; set; }
}
```

Рис. 4.10 – Класс-сущность "Faculty"

А теперь в наш класс добавим класс контекста "AppContext" virtual-свойство типа обобщенного набора DbSet<Faculty> с именем "Faculties". Класс контекста должен принять следующий вид (**Рис. 4.11**):

```
class AppContext:DbContext
{
    public AppContext():base("name=ConnectionString")
    {
    }

    public virtual DbSet<Faculty> Faculties { get; set; }
}
```

Рис. 4.11 – Контекст "AppContext"
с набором сущностей типа "Faculty"

4.5 Первый взгляд на .config

Файлы .config – конфигурационные файлы проекта, имена этих файлов зависят от типа разрабатываемого проекта. Для приложений типа ASP.NET и WCF файл .config будет иметь имя Web.config, а для остальных App.config.

В .config-файлах содержится служебная информация проекта, такая как версия .net framework, строки подключения, конфигурации конечных точек сервиса и др. Там же и хранятся строки подключения для EF (**Рис. 4.12**).

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <connectionStrings>
    <add name="ConnectionString"
         connectionString="Data Source=.;Initial Catalog=DbEF;Integrated Security=True"
         providerName="System.Data.SqlClient"/>
  </connectionStrings>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237468 -->
    <section name="entityFramework"
             type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0, Culture
             requirePermission="false" />
  </configSections>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.2" />
  </startup>
  <entityFramework>
    <defaultConnectionFactory type="System.Data.Entity.Infrastructure.LocalDbConnectionFactory, EntityFramework">
      <parameters>
        <parameter value="mssqllocaldb" />
      </parameters>
    </defaultConnectionFactory>
    <providers>
      <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramewo
    </providers>
  </entityFramework>
</configuration>
```

Рис. 4.12 – Файл App.config

Файлы .config описываются языком XML (eXtensible Mark-up Language – расширяемый язык разметки), и имеют иерархическую древовидную структуру, каждая ветка которого описывает объект и называется узлом.

Первый узел документа описывает версию XML документа и кодировку (**Рис. 4.13**).

```
<?xml version="1.0" encoding="utf-8"?>
```

Рис. 4.13 – Узел описания XML-документа App.config

Сейчас нас интересует лишь узел "connectionStrings", его подузлы и параметры.

```
<connectionStrings>
  <add name="ConnectionString"
       connectionString="Data Source=.;Initial Catalog=DbEF;Integrated Security=True"
       providerName="System.Data.SqlClient"/>
</connectionStrings>
```

Рис. 4.14 – Узел "connectionStrings"

Узел "connectionStrings" содержит набор подузлов "add", описывающих сроки подключения к БД. Подузлов "add" может быть несколько, так как приложение может работать не с одним подключением и не одно БД. Узел "add" описывает строку подключения, а именно он содержит в себе ряд параметров, таких как "name", "connectionString", "providerName".

Параметр "name" содержит в себе название строки подключения, которое передается в конструктор DbContext.

Параметр "connectionString" содержит саму строку подключения к БД, описываемую по согласованному стандарту описания строки подключения Microsoft.

Параметр "providerName" содержит в себе значения драйвер-провайдера СУБД, на которой развернута наша БД.

На сегодняшний момент EF поддерживает такие СУБД как MSSQL, MySQL, PostgreSQL.

О работе с некоторыми из них поговорим чуть позже.

4.6 Первый старт EF(CodeFirst from Database)

В этом примере будет использоваться подход "CodeFirst from Database".

Прежде чем мы попробуем подход "CodeFirst from Database", мы с вами создадим БД "University" со следующими таблицами, структура которых довольно проста (**Рис. 4.15**).

Все столбцы, имеющие тип данных int и помеченные как первичный ключ, также помечены свойством "Identity Specification".

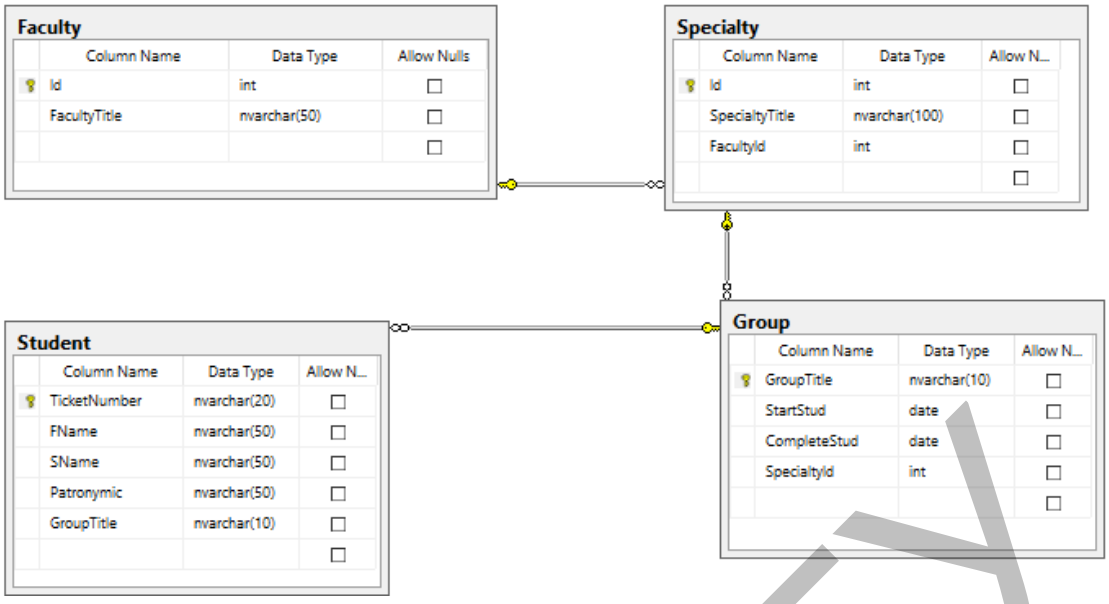

Рис. 4.15 – Схема таблиц и связей БД «University»

Теперь создайте решение типа "Console Application" с названием "FirstEfApp" и подключите к нему библиотеки EF.

На проекте кликаем правой кнопкой мыши Add –> New Item (**Рис. 4.16**).

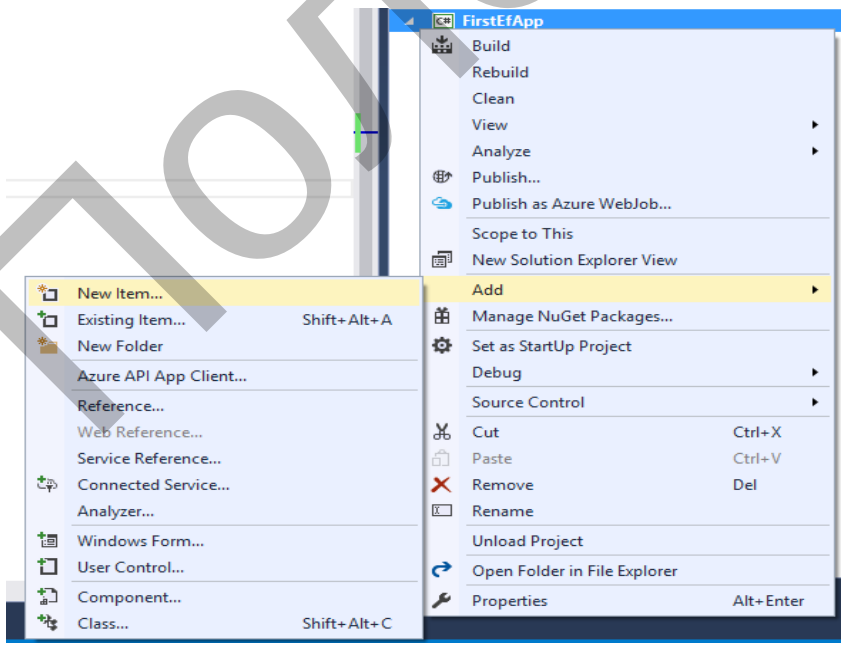

Рис. 4.16 – Добавление нового элемента к проекту

Затем слева в списке типов элементов выберем "Data", а в списке элементов "ADO.NET Entity Data Model" и в поле снизу называем элемент "DbModel" (DbModel – будущее имя класса-контекста). Кликаем на кнопочку "Add" (**Рис. 4.17**).

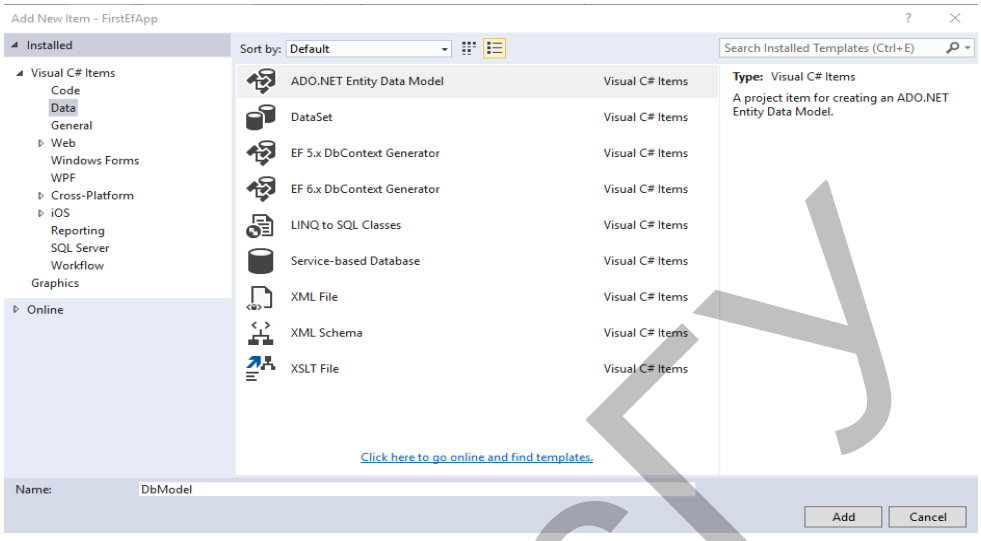

Рис. 4.17 – Добавление элемента "ADO.NET Entity Data Model"

После чего перейдем к шагу выбора подхода. Из существующих четырех выбираем нужный нам подход "Code First from database" и кликаем кнопку "Next >", после чего перейдем к шагу настройки подключения к БД (**Рис. 4.18**).

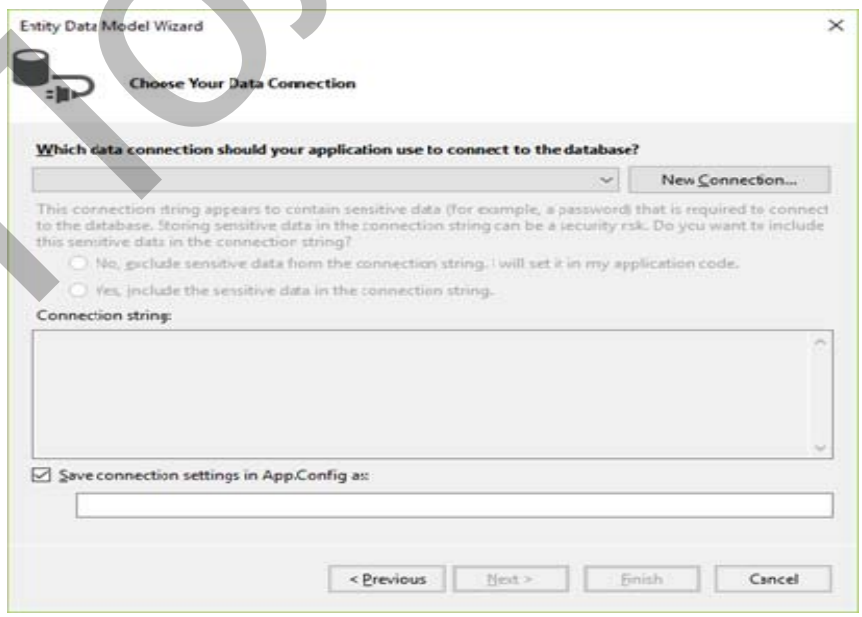

Рис. 4.18 – Настройки подключения к БД

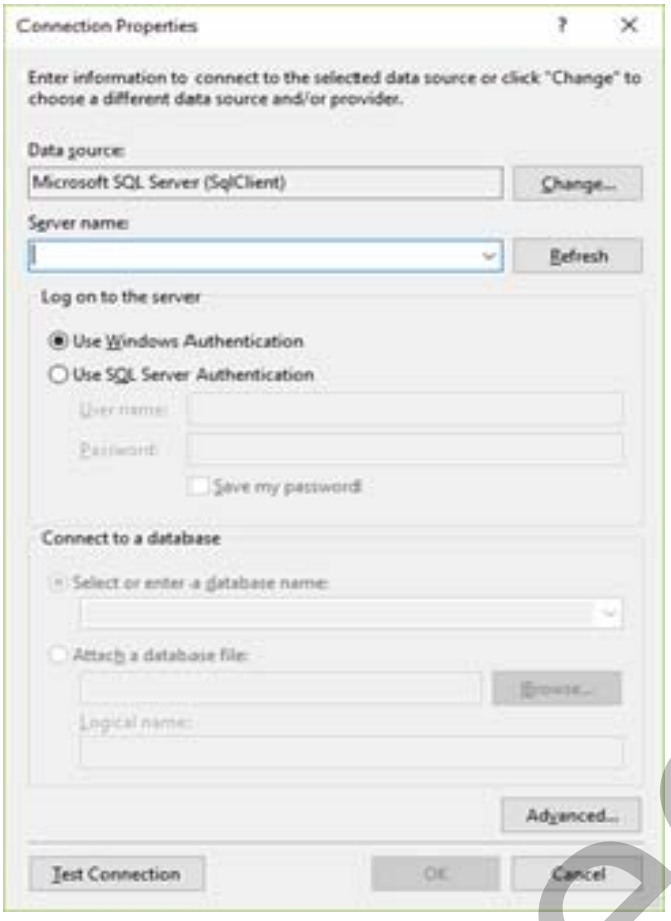

Рис. 4.19 – Диалог создания подключения

Так как мы еще не создавали подключения к БД, то кликаем на кнопочку "New Connection…" – откроется диалог создания подключения (**Рис. 4.19**). В выпадающем редактируемом списке "Server name" вводим название СЕРВЕРА БД.

ВНИМАНИЕ!

Если экземпляр сервера не установлен как экземпляр по умолчанию (имя MSSQLSERVER), тогда название сервера будет выглядеть следующим образом:

ИМЯ_МАШИНЫ\ИМЯ_ЭКЗЕМПЛЯРА_СЕРВЕРА_БД (например, MYCOMP\SQLEXPRESS).

Название сервера нечувствительно к регистру. Если экземпляр сервера установлен по умолчанию, тогда достаточно вписать имя машины, просто поставить точку или написать localhost (например, localhost, MYCOMP). Используем просто точку (.).

Следующий шаг – это аутентификация. Тут предлагается выбрать один из пунктов: использовать Windows-аутентификацию или же использовать SQL Server-аутентификацию.

Вы можете использовать и SQL Server-аутентификацию, если у вас выделен пользователь для БД, но в примере будет использоваться Windows-аутентификация. В зависимости от того, какой тип аутентификации мы выберем, будет варьироваться строка подключения.

Следующий шаг – это выбор БД. VS нам предлагает два варианта: прикрепить файл БД (LocalDB) "Attach a database file" или же использовать БД из имеющихся подключенных к серверу БД. Будем использовать нашу БД "University" из сервера БД "Select or enter a database name". Подключаемые БД LocalDB, их плюсы и минусы рассмотрим при использовании EF в web-приложениях ASP.NET (**Рис. 4.20**).

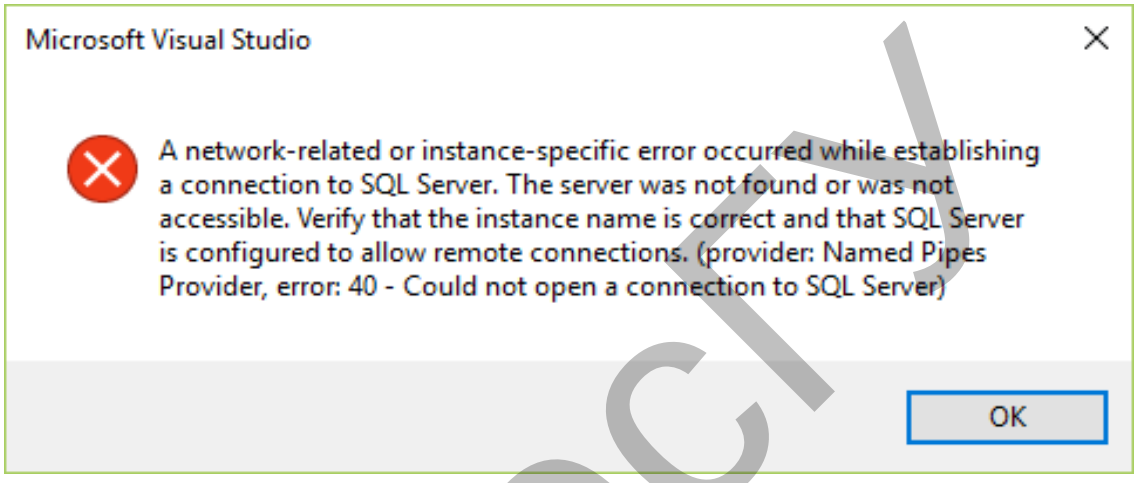

Рис. 4.20 – Ошибка из-за неправильного имени сервера БД (не удалось открыть соединение с SQL сервером)

Введя правильное имя сервера, у нас активируется выпадающий список для выбора БД "Select or enter a database name". Из него выберем нужную нам БД "University".

Если же список с выбором БД загружается долго (таймаут по умолчанию – 30 сек.), и через таймаут вы получаете ошибку, тогда вы неверно указали имя сервера БД либо не прошли аутентификацию (зависит от содержимого полученной ошибки) (**Рис. 4.21**).

После того как все поля заполнены верно, мы можем протестировать соединение, нажав соответствующую кнопку "Test connection". В случае успеха мы получим сообщение об успешном тестировании соединения (**Рис. 4.22**).

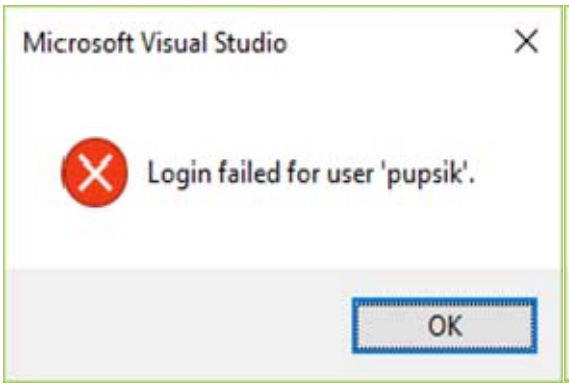

Рис. 4.21 – Ошибка из-за неправильного имени пользователя или пароля при SQL Server-аутентификации

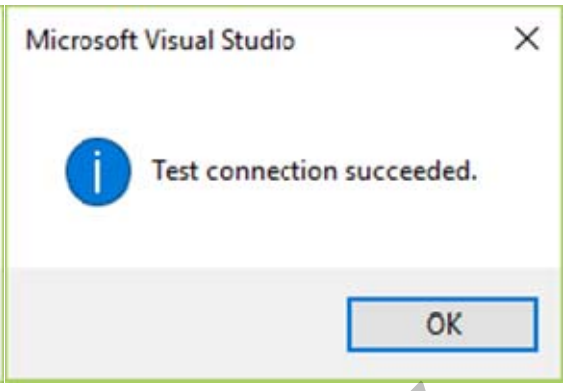

Рис. 4.22 – Сообщение об успешном тестировании соединения

В случае неудачи соответственно появляется сообщение об ошибке (**Рис. 4.23**).

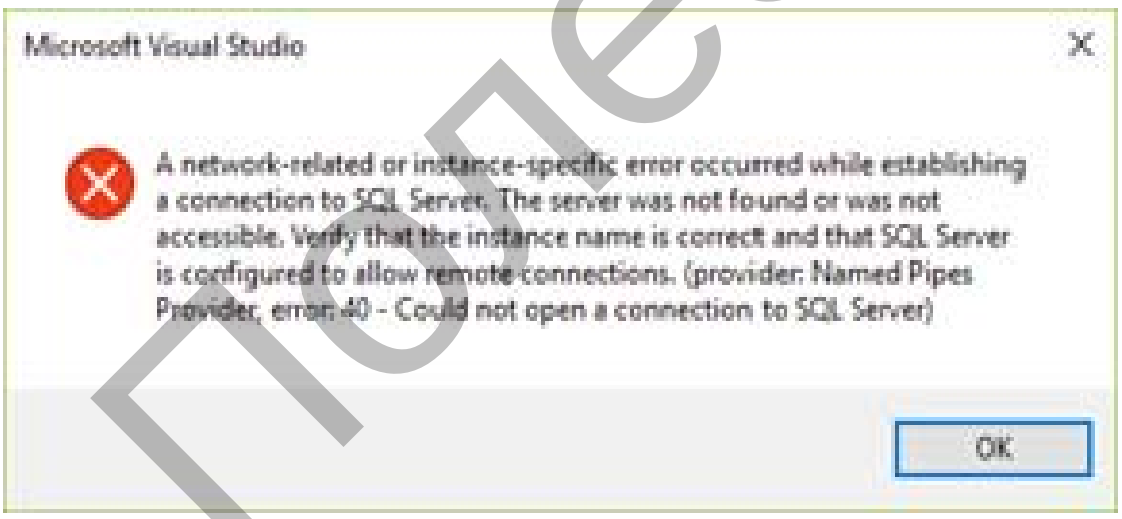

Рис. 4.23 – Сообщение о неудачном тестировании соединения (не удалось открыть соединение с SQL сервером)

Если тест прошел «на ура», тогда мы должны получить приблизительно следующий заполненный диалог. В диалоге кликаем "OK" (**Рис. 4.24**).

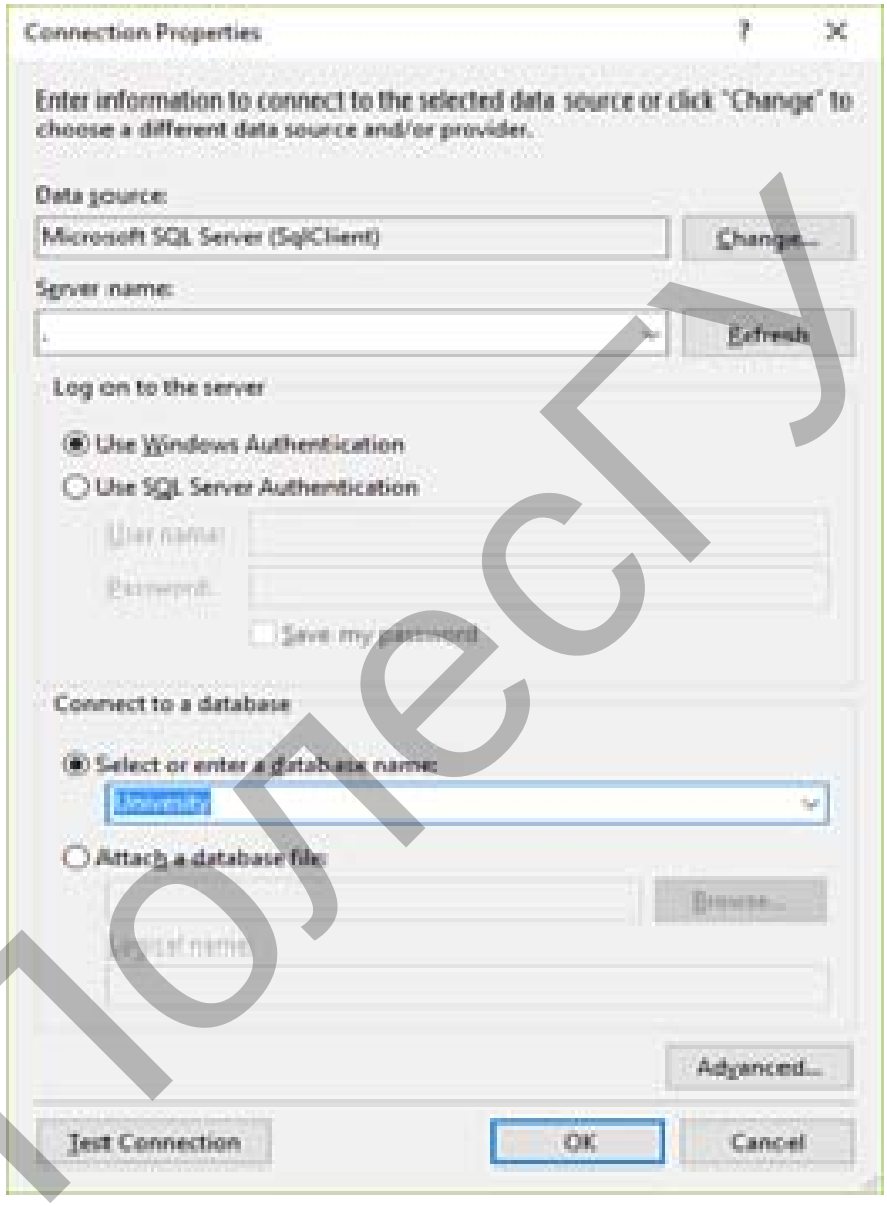

Рис. 4.24 – Заполненный диалог создания соединения

5. ПРИКЛАДНАЯ РАБОТА В ORM ENTITY FRAMEWORK

После того как диалог закроется, в выпадающем списке выбора подключения будет выбрано только что созданное подключение. Оставим его выбранным.

В прокручиваемом readonly-поле "Connection string" отобразится строка подключения. Чтобы сохранить нашу строку подключения в файле App.config, установим флажок "Save connection strings in App.Config as" в состояние «Включено».

Также здесь имеется поле с именем строки подключения. Давайте переименуем имя подключения как "DefaultConnection". Это параметр "name" узла "add" в файле .config. Это же имя и передается в конструктор базового класса DbContext контекста данных согласно формата "name = имя_строки_подключения". Далее кликаем "Next >".

Следующий шаг – выбор таблиц для генерирования сущностей и контекста. Развернём список таблиц и отметим все таблицы. Если в БД вы создавали диаграмму таблиц, то таблицу "sysdiagrams" отмечать не нужно!

Далее отметим флажок "Генерировать имена объектов" во множественном числе "Pluralize or singularize generated object names". Кликаем кнопочку "Finish". В окне вывода "Output" мы должны получить сообщение об успешном генерировании модели.

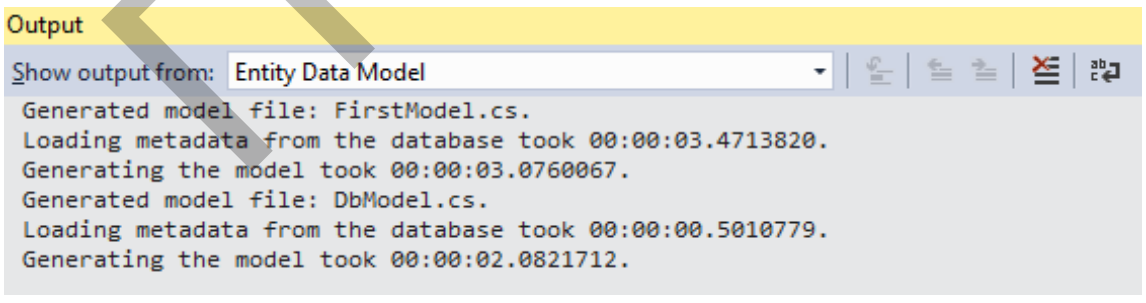

Рис. 5.1 – Сообщение-вывод об успешном генерировании модели

Можно заметить, что в корне проекта у нас появилось 5 новых классов: "DbModel" (контекст), "Group" (сущность), "Faculty" (сущность), "Specialty" (сущность), "Student" (сущность).

Сейчас давайте вернемся в наш класс "Program" и в методе Main напишем код (**Рис. 5.2**):

```
13          Console.WriteLine("Привет, EF!");
14          Console.WriteLine("Друзья, EF валит!");
15
16          //оборачиваем объект-контекста в контсрукцию using для явного уничтожения объекта и вызова дестуктора
17          //что повлечет закрытие соединения с БД
18          using (DbModel db = new DbModel())
19          {
20              //из набора DbSet<Faculty> Fcaculties объекта-контекста получаем кол-во факультетов
21              int count_fac = db.Faculties.Count();
22              Console.WriteLine("Было факультетов: {0}", count_fac);
23
24              //создаем обект-сущность
25              Faculty new_fac = new Faculty() { FacultyTitle = "Факультет банковского дела" };
26              //добавляем его в набор контекста
27              db.Faculties.Add(new_fac);
28              //сохраняем изменения в контексте и БД
29              db.SaveChanges();
30              Console.WriteLine("Ничёсе! Я создал факультет!");
31
32              //опять из набора DbSet<Faculty> Fcaculties объекта-контекста получаем кол-во факультетов
33              count_fac = db.Faculties.Count();
34              Console.WriteLine("Стало факультетов: {0}\n", count_fac);
35
36              //получаем коллекцию факультеов и проходим по всем факультетам
37              foreach (Faculty fac in db.Faculties.ToList())
38              {
39                  Console.WriteLine("Всем привет! Я великий '{0}'!", fac.FacultyTitle);
40              }
41          }
42          Console.ReadKey();
```

Рис. 5.2 – Добавление и вывод первого факультета. Метод Main

Отправим наше приложения на выполнение(Ctrl+F5) и, при точном соблюдении всех шагов, указанных выше, мы получим вывод в консольном приложении (**Рис. 5.3**).

```
Привет, EF!
Друзья, EF валит!
Было факультетов: 0
Ничёсе! Я создал факультет!
Стало факультетов: 1

Всем привет! Я великий 'Факультет банковского дела'!
```

Рис. 5.3 – Результат выполнения приложения «FirstEfApp»

А теперь посмотрим в нашу таблицу "" и увидим, что таблица не пуста, а в ней уже есть одна запись – "Факультет банковского дела" (**Рис. 5.4**).

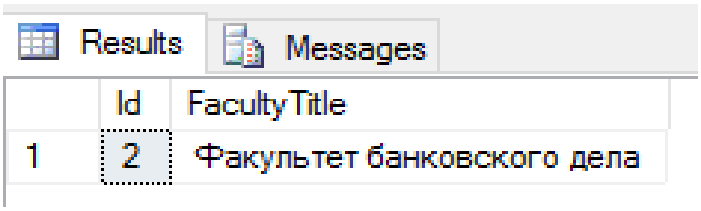

Рис. 5.4 – Записи в таблице "Faculty"

Разберем, зачем оборачивали объект контекста (DbModel) конструкцией using (18-я строка). В этой строке создали объект контекста.

Дело в том, что контекст EF берет на себя такую обязанность, как работа с соединениями БД. Так вот, когда объект уничтожается, все соединения контекста с БД закрываются в деструкторе. Может показаться, что когда объект контекста создается, то открывается соединение, но на самом деле НЕТ! Соединение с БД открывается контекстом лишь тогда, когда объект контекста производит свой первый запрос к БД.

Когда оборачивали объект контекста конструкцией using, мы явно определили рамки жизни объекта контекста, т.е. это означает, что когда выполнение алгоритма выйдет за пределы конструкции using, то сборщику мусора будет послана команда об уничтожении объекта контекста, и при уничтожении сработает деструктор, в котором явно закрываются все соединения с БД. Это нужно для того, что бы выиграть в производительности, не оставить случайно открытыми соединения к БД.

Конечно, использование конструкции using вовсе не обязательно, но считается правилом хорошего тона!

В 21 строке получили количество записей в БД, используя свойство-набор DbSet<Faculty> Faculties контекста.

В 25 строке просто создали новый объект типа "Faculty" и установили его свойство "FacultyTitle", равное "Факультет банковского дела".

В 27 строке добавили только что созданный объект типа "Faculty" в набор контекста DbSet<Faculty> Faculties.

В 29 строке сохранили изменения контекста в БД вызовом метода контекста SaveChanges();

В 33 строке вновь получили количество записей в БД, используя свойство-набор DbSet<Faculty> Faculties контекста.

А теперь давайте посмотрим, что понадобилось, для того, чтобы получить CRUD (create-read-update-delete) к БД, используя EF и используя ADO.NET.

```
DbModel db = new DbModel();

// create
Faculty new_fac = new Faculty() { FacultyTitle = "Экономический факультет" };
db.Faculties.Add(new_fac);
db.SaveChanges();

// read
List<Faculty> facs = db.Faculties.ToList();

// update
Faculty u_fac = db.Faculties.First();
u_fac.FacultyTitle += " NEW";
db.Entry(u_fac).State = System.Data.Entity.EntityState.Modified;
db.SaveChanges();

// delete
Faculty d_fac = db.Faculties.First();
db.Faculties.Remove(d_fac);
db.SaveChanges();
```

Рис. 5.5 – CRUD с EF

```
string connectionString = "Data Source=.;Initial Catalog=Univesity;Integrated Security=True";
string selectQuery = "SELECT * FROM Faculty";
string insertQuery = "INSERT INTO Faculty (FacultyTitle) VALUES(@iFacTitle)";
string deleteQuery = "DELETE FROM Faculty WHERE Id=@dFacId";
string updateQuery = "UPDATE Faculty SET(FacultyTitle=@uFacTitle) WHERE Id=@uFacId";

SqlCommand insertCommand = new SqlCommand(insertQuery);
```

```
SqlParameter iTitlePar = new SqlParameter("@iFacTitle", SqlDbType.NVarChar, 50, "FacultyTitle");
insertCommand.Parameters.Add(iTitlePar);

SqlCommand deleteCommand = new SqlCommand(deleteQuery);
SqlParameter dIdPar = new SqlParameter("@dFacId", SqlDbType.Int, 32, "Id");
deleteCommand.Parameters.Add(dIdPar);

SqlCommand updateCommand = new SqlCommand(updateQuery);
SqlParameter uTitlePar = new SqlParameter("@uFacTitle", SqlDbType.NVarChar, 50, "FacultyTitle");
SqlParameter uIdPar = new SqlParameter("@uFacId", SqlDbType.Int, 32, "Id");
updateCommand.Parameters.Add(uTitlePar);
updateCommand.Parameters.Add(uIdPar);

DataTable dTable = new DataTable();

SqlDataAdapter adapter = new SqlDataAdapter(selectQuery, connectionString);

// read
adapter.Fill(dTable);

// create
DataRow new_dRow = dTable.NewRow();
new_dRow[1] = "Биотехнологический факультет";
dTable.Rows.Add(new_dRow);
adapter.Update(dTable);

// update
dTable.Rows[0][2] = "ФОЗОЖ";
adapter.Update(dTable);

// delete
dTable.Rows.RemoveAt(2);
adapter.Update(dTable);
```

Рис. 5.6 – CRUD с ADO.NET

Из вышеприведенных примеров очевидно, что EF выигрывает у ADO.NET как по объему написанного кода, так и по принципу управления объектами.

Если учесть еще тот момент, что для реализаций операций CRUD в ADP.NET требуется написание запросов для всех таблиц, то в EF-модель охватывает все таблицы сразу.

EF реализует несколько подходов, а именно 4:

– CodeFirts;
– DatabaseFirst;
– CodeFirst from Database;
– ModelFirst.

CodeFirst – подход, при котором разработчик проектирует и разрабатывает классы сущностей, а также контекст данных, при первом использовании которого генерируются БД и соответствующие таблицы, согласно правилам и аннотациям EF (стратегия по умолчанию).

DatabaseFirst – подход, при котором уже имеется готовая БД, и разработчику с использованием инструментария EF остается лишь сгенерировать классы сущностей, схему модели и контекст, исходя из структуры БД.

CodeFirst from Database – тот же подход, что и DatabaseFirst, за исключением того, что схема модели не создается.

ModelFirst – подход, при котором имеется лишь модель данных, согласно которой генерируются классы сущностей и контекст (при необходимости и БД).

6. ЗАДАНИЯ ПО САМОСТОЯТЕЛЬНОМУ ОСВОЕНИЮ РАБОТЫ В ORM ENTITY FRAMEWORK

1. Вопрос: Инсталляция / деинсталляция EF

1.1. Установить EF, используя NuGet. Деинсталлировать, используя NuGet.

1.2. Установить EF, используя библиотеки EntityFramework.dll и EntityFramework.SqlServer.dll из файловой системы. Деинсталлировать сборки EF из проекта.

1.3. Установить EF, используя NuGet. Деинсталлировать сборки EF из проекта, не используя NuGet.

2. Вопрос: Создание подключений (в таблицах БД использовать "Identity Specification"). При необходимости (в зависимости от типа аутентификации) добавить пользователя для аутентификации к БД. Полученные подключения не сохранять!!!

2.1. Создать БД "ФАМИЛИЯ_ТРАНСЛИТОМ_НА_АНГЛ_ВАРИАНТ_ЦИФРА" с таблицей "People" (Id<int>, FirstName<nvarchar(50)>, SecondName<nvarchar(50)>). Создать подключение к БД, используя Windows аутентификацию; используя средства VS, протестировать его. Соединение должно пройти проверку с использованием "Test connection"!!!

2.2. Создать БД "ФАМИЛИЯ_ТРАНСЛИТОМ_НА_АНГЛ_ВАРИАНТ_ЦИФРА" с таблицей "Cars" (Id<int>, Number< nvarchar(10)>, Color<nvarchar(10)>, Mark<nvarchar(20)>). Создать подключение к БД, используя SQL Server-аутентификацию; используя средства VS, протестировать его. Соединение должно пройти проверку с использованием "Test connection"!!!

2.3. Создать БД "ФАМИЛИЯ_ТРАНСЛИТОМ_НА_АНГЛ_ВАРИАНТ_ЦИФРА" с таблицей "Developers" (Id<int>, Company<nvarchar(50)>, DevelopPlatphorm<nvarchar(15)>). Создать подключение к БД, используя Windows-аутентификацию; используя средства VS, протестировать его. Соединение должно пройти проверку с использованием кнопочки "Test connection"!!!

3. Строка подключения

3.1. В блокноте Windows написать строку подключения к БД "ФАМИЛИЯ_ТРАНСЛИТОМ_НА_АНГЛ_ВАРИАНТ_ЦИФРА", использующую Windows-аутентификацию, где ИМЯ_МАШИНЫ – это имя вашего компьютера, а ИМЯ_ЭКЗЕМПЛЯРА_СЕРВЕРА_БД – "sqlexpress". В блокноте Windows написать строку подключения к БД "ФАМИЛИЯ_ТРАНСЛИТОМ_НА_АНГЛ_ВАРИАНТ_ЦИФРА", SQL Server-аутентификацию, где ИМЯ_МАШИНЫ – это IP-адрес 192.168.1.1, а ИМЯ_ЭКЗЕМПЛЯРА_СЕРВЕРА_БД установлено по умолчанию.

3.2. В блокноте Windows написать строку подключения к БД "ФАМИЛИЯ_ТРАНСЛИТОМ_НА_АНГЛ_ВАРИАНТ_ЦИФРА", использующую Windows-аутентификацию, где ИМЯ_МАШИНЫ – это имя вашего компьютера, а ИМЯ_ЭКЗЕМПЛЯРА_СЕРВЕРА_БД установлено по умолчанию.

4. Вопрос: Файлы .config. При необходимости (в зависимости от типа аутентификации) добавить пользователя для аутентификации к БД

4.1. Создать проект типа "Console Application1", отредактировать файл App.config, добавив в него необходимые узлы строки подключения EF к БД "ФАМИЛИЯ_ТРАНСЛИТОМ_НА_АНГЛ_ВАРИАНТ_ЦИФРА". Строка подключения должна быть работоспособной, так как, исходя из нее, в последующих заданиях устанавливается соединение с БД. Имя строки подключения должно соответствовать имени БД.

4.2. Создать проект типа "Console Application2", отредактировать файл App.config, добавив в него необходимые узлы строки подключения EF к БД "ФАМИЛИЯ_ТРАНСЛИТОМ_НА_АНГЛ_ВАРИАНТ_ЦИФРА". Строка подключения должна быть работоспособной, так как в последующих заданиях, исходя из нее, устанавливается соединение с БД. Имя строки подключения должно соответствовать имени БД.

4.3. Создать проект типа "Console Application2", отредактировать файл App.config, добавив в него необходимые узлы строки подключения EF к БД "ФАМИЛИЯ_ТРАНСЛИТОМ_НА_АНГЛ_ВАРИАНТ_ЦИФРА". Строка подключения долж-

на быть работоспособной, так как в последующих заданиях устанавливается, исходя из нее, соединение с БД. Имя строки подключения должно соответствовать имени БД.

5. Вопрос: CodeFirst from database (по выполнению задания удалить сгенерированные классы)

5.1. Создать контекст и сущности БД "ФАМИЛИЯ_ТРАНСЛИТОМ_НА_АНГЛ_ВАРИАНТ_ЦИФРА", используя подход CodeFirst from database. Добавить одну запись в таблицу "People", вывести на экран количество записей, удалить запись из таблицы. Реализация добавления, получения количества записей и удаления записи должна быть реализована в методе Main класса Program.

5.2. Создать контекст и сущности БД "ФАМИЛИЯ_ТРАНСЛИТОМ_НА_АНГЛ_ВАРИАНТ_ЦИФРА", используя подход CodeFirst from database. Добавить одну запись в таблицу "Cars", вывести на экран количество записей, удалить запись из таблицы. Реализация добавления, получения количества записей и удаления записи должна быть реализована в методе Main класса Program.

5.3. Создать контекст и сущности БД "ФАМИЛИЯ_ТРАНСЛИТОМ_НА_АНГЛ_ВАРИАНТ_ЦИФРА", используя подход CodeFirst from database. Добавить одну запись в таблицу "Developers", вывести на экран количество записей, удалить запись из таблицы. Реализация добавления, получения количества записей и удаления записи должна быть реализована в методе Main класса Program.

6. Вопрос: Создание контекста с использованием строки подключения из App.config

6.1. Разработать контекст без наборов с использованием строки подключения из App.config и конструктора базового класса. Имя класса контекста должно соответствовать имени строки подключения в App.config. Создать объект контекста в методе Main класса Program. Приложения должно выполняться без исключений (Exception), иначе строка подключения в App.config записана неверно (**зад. 4** провалено)!

6.2. Разработать контекст без наборов с использованием строки подключения из App.config и конструктора базового класса. Имя класса контекста должно соответствовать имени строки подключения в App.config. Создать объект контекста в методе Main класса Program. Приложения должно выполняться без исключений (Exception), иначе строка подключения в App.config записана неверно (**зад. 4** провалено)!

6.3. Разработать контекст без наборов с использованием строки подключения из App.config и конструктора базового класса. Имя класса контекста должно соответствовать имени строки подключения в App.config. Создать объект контекста в методе Main класса Program. Приложения должно выполняться без исключений (Exception), иначе строка подключения в App.config записана неверно (**зад. 4** провалено)!

7. Вопрос: Создание сущности

7.1. Создать сущность "Person" согласно соответствующей таблице "People" в БД "ФАМИЛИЯ_ТРАНСЛИТОМ_НА_АНГЛ_ВАРИАНТ_ЦИФРА".

7.2. Создать сущность "Car" согласно соответствующей таблице "Cars" в БД "ФАМИЛИЯ_ТРАНСЛИТОМ_НА_АНГЛ_ВАРИАНТ_ЦИФРА".

7.3. Создать сущность "Developer" согласно соответствующей таблице "Developers" в БД "ФАМИЛИЯ_ТРАНСЛИТОМ_НА_АНГЛ_ВАРИАНТ_ЦИФРА".

8. Вопрос: Создание свойств-наборов в контексте

8.1. Добавить в контекст (из **зад. 6**) набор сущностей (из **зад. 7**). В свойствах-наборах должны присутствовать все дополнительные атрибуты (задание одно на три варианта, результат у каждого варианта разный).

9. Вопрос: Использование контекста, сущностей, конструкций вызывающих сборщик мусора, динамических строк подключения

9.1. Реализовать интерфейс выбора строки подключения: из App.config или динамическая строка. При выборе пункта динамической строки подключения запросить ввод с клавиа-

туры имени сервера БД, имя экземпляра БД, имя БД, тип аутентификации, имя пользователя и пароль (при SQL Server-аутентификации) и использовать динамическую строку в контексте (при выборе этого пункта). Реализовать добавление записи в БД, вывод количества записей, удаление добавленной записи. При выполнении задания использовать конструкции, гарантирующие закрытие соединений с БД. В приложении не должно быть не отлавливаемых исключений! Реализация логики должна быть расположена в методе Main класса Program. Тестирование приложения должно ОБЯЗАТЕЛЬНО включать тестирование с использованием динамической строки подключения. (Задание одно на три варианта, результат у каждого варианта разный).

10. Вопрос: CRUD ADO.NET, CRUD EF

10.1. Используя полученный контекст (из **зад. 8**), при реализации задания который придется подправить, и сущности (из **зад. 7**), реализовать операции CRUD на EF, а также разработать операции CRUD с использованием ADO.NET. Создать текстовый документ в проекте и описать преимущества EF над ADO.NET в вашем проекте. При выполнении задания БЫЛО БЫ РАЗУМНО использовать дополнительные классы, отделяющие реализацию ADO.NET от EF.

При выполнении задания использовать конструкции, гарантирующие закрытие соединений с БД. В приложении не должно быть не отлавливаемых исключений! Реализация должна содержать самодокументирующиеся XML-комментарии к классам и методам, а также разумное количество простых комментариев к логике.

Названия переменных, классов и методов символами из набора кириллицы СТРОГО НАКАЗЫВАТСЯ! (задание одно на три варианта, результат у каждого варианта разный).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Хортон, А. Visual C++ 2008 : базовый курс / А. Хортон. – М. : Издательский дом «Вильямс», 2009. – 284 с.
2. Мешков, А. Visual C++ и MFC : в 3 т. / А. Мешков, Ю. Тихомиров. – СПб. : БХВ-Петербург, 2006. – 198 с.
3. Рихтер, Дж. Программирование на языке Visual C++. Windows via C/C++ / Дж. Рихтер, К. Назар. – СПб. : Питер, 2009. – 249 с.
4. Тихомиров, Ю. Самоучитель MFC / Ю. Тихомиров. – СПб. : БХВ-Петербург, 2002. – 256 с.
5. Круглински, Д. Программирование на Microsoft Visual C++ 6.0. Для профессионалов / Д. Круглински [и др.]. – СПб. : Питер, М. : Русская Редакция, 2001. – 145 с.
6. Чернышов, Ю.Н. Информационные технологии в экономике : учебное пособие / Ю.Н. Чернышов. – 2-е изд., испр. и доп. – М. : Горячая линия-телеком, 2008. – 240 с.
7. Смоленцев, Н.К. Matlab: программирование на Visual C#, Borland JBuilder,VBA : учебный курс / Н.К. Смоленцев. – СПб. ; М. : Питер, 2009. – 464 с.
8. Разработка приложений в среде Visual Basic / Министерство образования Республики Беларусь, Белорусский национальный технический университет ; сост.: И.Е. Ругалёва, Н.В. Дашкевич. – Минск : БНТУ, 2012. – 113 с.
9. Коренская, И.Н. Основы алгоритмизации и программирования в среде Visual C++ / И.Н. Коренская [и др.]. – Министерство образования Республики Беларусь, УО «Белорусский государственный университет информатики и радиоэлектроники», Институт повышения квалификации БГУИР, Институт информационных технологий. – Минск : БГУИР, 2011.– 57 с.
10. Павловская, Т.А. C/C++. Программирование на языке высокого уровня : учебник / Т.А. Павловская. – Издательская программа «300 лучших учебников для высшей школы». – СПб. : ПИТЕР, 2007. – 461 с.
11. Бобровский, С.И. Технологии C#Builder. Разработка приложений для бизнеса : учебный курс / С.И. Бобровский. – СПб. : Питер, 2007. – 672 с.

Учебное издание

Штепа Владимир Николаевич
Базака Людмила Николаевна
Деркач Денис Александрович
Дмитраница Александр Александрович

Визуальные средства разработки программных приложений

Методические рекомендации

Ответственный за выпуск *П.Б. Пигаль*

Редактор *Т.И. Сакович*

Подписано в печать 09.03.2016 г. Формат 60×84/16.
Бумага офсетная. Гарнитура «Таймс». Ризография.
Усл. печ. л. 4,19. Уч.-изд. л. 1,79.
Тираж 60 экз. Заказ № 86.

Отпечатано в редакционно-издательском отделе
Полесского государственного университета
225710, г. Пинск, ул. Днепровской флотилии, 23.