Министерство науки и высшего образования
Российской Федерации
Братский педагогический колледж
федерального государственного бюджетного
образовательного учреждения высшего образования
«Братский государственный университет»

МДК 02.02. ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

методические рекомендации
по выполнению лабораторных занятий

для студентов 4 курса
очной формы обучения
специальности
09.02.07 Информационные системы и программирование

Автор: Л.Д. Разумова

Братск, 2021

Инструментальные средства разработки программного обеспечения. Методические рекомендации по выполнению практических занятий. /Сост. Л.Д. Разумова.- Братск, 2021.- 83 с.

Методические рекомендации предназначены для студентов четвертого курса специальности 09.02.07 Информационные системы и программирование при изучении междисциплинарного курса «Инструментальные средства разработки программного обеспечения».

Печатается по решению научно-методического совета
Братского педагогического колледжа ФГБОУ ВО «БрГУ»
665709, г. Братск, ул. Макаренко, 40

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Лабораторные занятия являются одним из основных видов учебных занятий, направленных на экспериментальное подтверждение теоретических положений и формирование учебных и профессиональных практических умений. Они составляют важную часть теоретической и профессиональной практической подготовки.

Лабораторные занятия направлены на экспериментальное подтверждение теоретических положений и формирование учебных и профессионально-значимых умений обучающихся.

Выполнение лабораторных занятий проводится с целью:

- формирования практических умений в соответствии с требованиями к уровню подготовки студентов, установленными рабочей программой дисциплины по конкретным разделам (темам);
- обобщение, систематизацию, углубление, закрепление полученных теоретических знаний;
- совершенствование умений применять полученные знания на практике, реализацию единства интеллектуальной и практической деятельности;
- развитие интеллектуальных умений у будущих специалистов: аналитических, проектировочных, конструктивных и др.;
- выработку при решении поставленных задач таких профессионально значимых качеств, как самостоятельность, ответственность, точность, творческая инициатива.

В части освоения основного вида профессиональной деятельности (ВПД): «Участие в интеграции программных модулей» и соответствующих профессиональных компетенций (ПК) обучающийся должен:

3.1. Анализировать проектную и техническую документацию на уровне взаимодействия компонент программного обеспечения.

3.2. Выполнять интеграцию модулей в программную систему.

3.3. Выполнять отладку программного продукта с использованием специализированных программных средств.

3.4. Осуществлять разработку тестовых наборов и тестовых сценариев.

3.5. Производить инспектирование компонент программного продукта на предмет соответствия стандартам кодирования.

3.6. Разрабатывать технологическую документацию.

С целью овладения указанным видом профессиональной деятельности и соответствующими профессиональными компетенциями обучающийся в ходе освоения профессионального модуля должен:

иметь практический опыт:

– участия в выработке требований к программному обеспечению;

– участия в проектировании программного обеспечения с использованием специализированных программных пакетов;

уметь:

– владеть основными методологиями процессов разработки программного обеспечения;

– использовать методы для получения кода с заданной функциональностью и степенью качества;

знать:

– модели процесса разработки программного обеспечения;

– основные принципы процесса разработки программного обеспечения;

– основные подходы к интегрированию программных модулей;

– основные методы и средства эффективной разработки;

– основы верификации и аттестации программного обеспечения;

– концепции и реализации программных процессов;

– принципы построения, структуры и приемы работы с инструментальными средствами, поддерживающими создание программного обеспечения;

– методы организации работы в коллективах разработчиков программного обеспечения;

– основные положения метрологии программных продуктов, принципы построения, проектирования и использования средств для измерений характеристик и параметров программ, программных систем и комплексов;

– стандарты качества программного обеспечения;

– методы и средства разработки программной документации.

Общие указания по выполнению работ на лабораторных занятиях

О проведении лабораторного занятия студентам сообщается заблаговременно: когда предстоит лабораторное занятие, какие вопросы нужно повторить, чтобы ее выполнить. Просматриваются задания, оговаривается объем и время выполнения. Критерии оценки сообщаются перед выполнением каждого лабораторного занятия.

Студенты получают распечатанный или электронный вариант задания с описанием этапов выполнения работы.

При выполнении лабораторного занятия студент придерживается следующего алгоритма:

1. Записать в тетради дату, тему и цель занятия.
2. Ознакомиться с правилами и условиями выполнения задания.
3. Повторить теоретические задания, необходимые для рациональной работы и других практических действий.
4. Выполнить работу по предложенному алгоритму действий.
5. Обобщить результаты работы, сформулировать выводы по работе.
6. Записать в тетрадь ответы на контрольные вопросы.

Задания студентами сдаются в электронном виде. Все выполненные работы нумеруются, в соответствии с номером практического задания, и сохраняются в отдельной папке.

ЛАБОРАТОРНАЯ РАБОТА № 1
ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ VISUAL STUDIO. ПЛАТФОРМА .NET
ТЕХНОЛОГИЯ РАЗРАБОТКИ WINDOWS ПРИЛОЖЕНИЙ. ПРИЛОЖЕНИЯ WINDOWS FORMS.

Цель: освоить инструмент «элементов управления» с целью создания различных приложений.

Задание на лабораторную работу

1. Создать учебные примеры (программы1-9) и разобрать принцип их работы.
2. Отчет представить в электронном виде: листинг кода с Вашими комментариями, скриншоты результатов работы программы. В кодах должны быть подобранные именно Вами файлы формата *. jpg , *.bmp, *.xml и в заголовках форм- номер программы, Ваша фамилия, номер группы.
3. При защите лабораторной работы демонстрируете преподавателю понимание работы кода.
4. Проекты сохраняете на своем флэш-накопителе для работы в следующих лабораторных работах.

Как создается и выполняется Windows-проект

По умолчанию он содержит *класс* Form1 - наследника *класса* Form. Этот *класс* содержит точку входа в проект - процедуру Main, вызывающую *статический метод* Run *класса* Application, который создает *объект класса* Form1 и открывает *форму* - видимый образ объекта - для интерактивной работы пользователя. Открываемая *форма* содержит *пользовательский интерфейс* - окошки, *кнопки*, списки, другие *элементы управления*, *меню*. Все эти *элементы* способны реагировать на события, возникающие при выполнении пользователем каких-либо действий - нажатии *кнопок*, ввода текста, выбора *пунктов меню*.

Приложения Windows Forms

Компания Майкрософт предоставила в составе библиотеки классов *.NET* Framework огромное количество "элементов управления", которые можно помещать на формы. Освоив этот инструмент, вы сможете быстро создавать эффектные приложения.

Если вы захотите писать программы, похожие на привычные приложения *Windows*, то наверняка воспользуетесь классами из пространства имен System.Windows.Forms. Они позволяют задействовать кнопки, списки, текстовые поля, *меню*, окна сообщений и множество других "элементов управления". *Элементы управления* — это то, что вы помещаете в форму. Они нужны для вывода информации, например, текстовой (элемент управления Label) или графической (элемент управления PictureBox), либо для выполнения определенных действий, например, выбора значения или перехода к другой форме после нажатия кнопки. Все *элементы управления* помещаются на форму.

Понятие "форма", принятое в программировании, родственно понятию "форма анкеты" или "форма документа" из обычной жизни. На форме можно в определенном порядке расположить различные *элементы (текст*, картинки, поля для заполнения и т. д.). Когда нам дают готовую форму документа и просят ее *заполнить*, мы обычно *читаем* содержащуюся в ней типовую информацию, а затем *вписываем недостающие данные в* определенные строки.

Каждый вид *элементов управления* описывается собственным *классом*. Библиотека FCL содержит большое число *классов*, задающих различные *элементы управления*. Одним из типов проектов, доступных на C#, является проект, создающий *элемент управления*, так что ничто не мешает создавать собственные *элементы управления* и размещать их на *формах* наряду со встроенными элементами. Многие фирмы специализируются на создании *элементов управления* - это один из видов повторно используемых компонентов.

В каких отношениях находятся *класс* Form, *класс* Control, *классы элементов управления*? На рис. 1 показана иерархия отношений, связывающих эти *классы*.

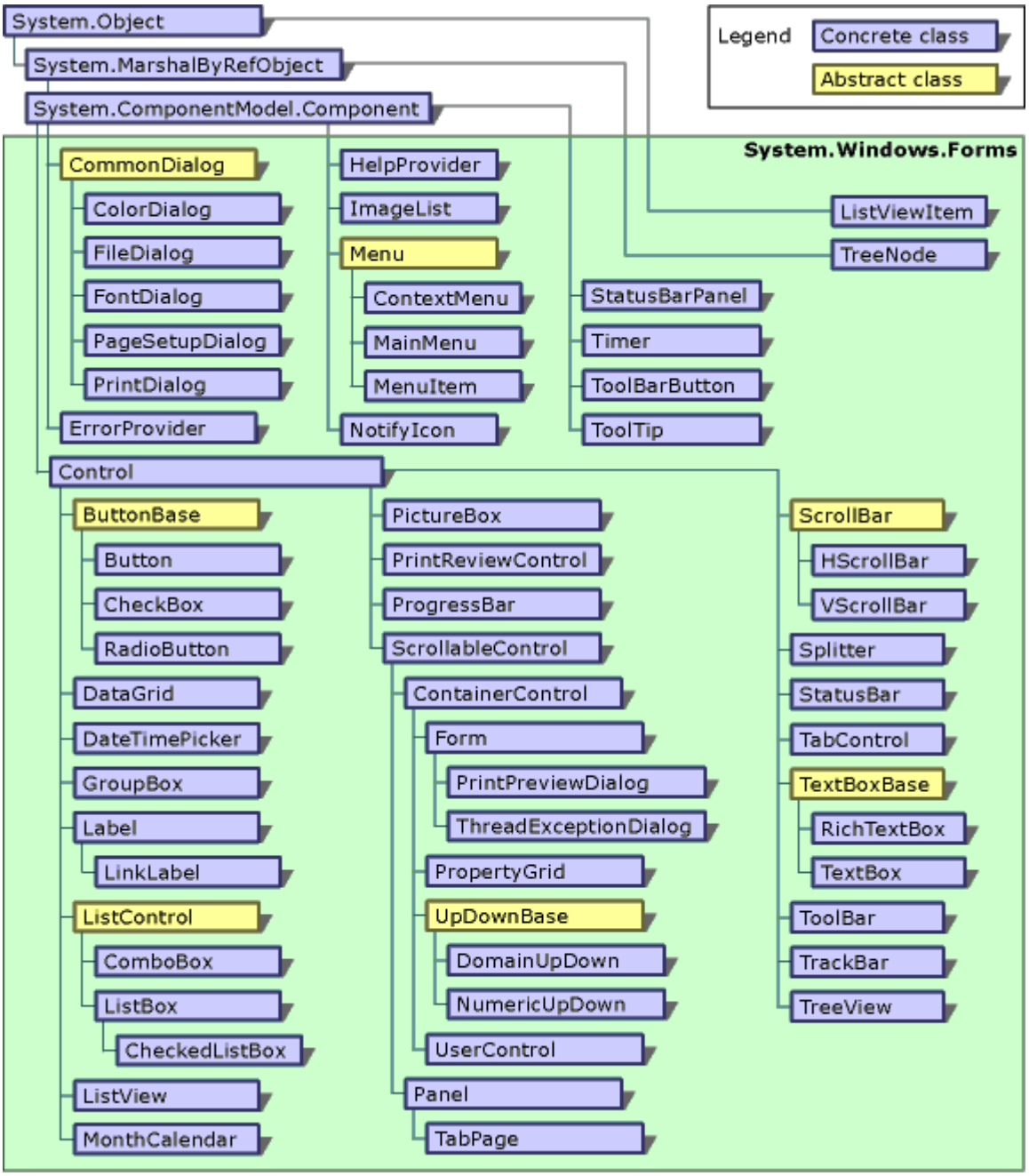


Рис. 1. Иерархия классов элементов управления

Естественно, все эти *классы* являются потомками прародителя - *класса* Object. Заметьте, *класс* Control в иерархии *классов* занимает довольно высокое положение, хотя и у него есть два важных родительских *класса* - *класс* Component, определяющий возможность *элементам управления* быть компонентами, и *класс* MarshalByRefObject, задающий возможность передачи *элементов управления* по сети. *Класс* Control задает важные свойства, методы и события, наследуемые всеми его потомками. Все *классы элементов управления* являются наследниками *класса* Control. Чаще всего, это прямые наследники, но иногда они имеют и непосредственного родителя, которым может быть абстрактный *класс* - это верно для *кнопок*, списков, текстовых *элементов управления*. Может показаться удивительным, но *класс* Form является одним из потомков *класса* Control,

так что *форма* - это *элемент управления* со специальными свойствами.Будучинаследником *классов* ScrollableControl и ContainerControl, *форма* допуска ет прокрутку и размещение *элементов управления*.

Некоторые полезные классы из пространства имен System.Windows.Forms

Вот некоторые элементы управления, которые можно размещать на формах:

- Label (Надпись).
- Button (Кнопка).
- ListBox (Список).
- CheckBox (Флажок).
- RadioButton (Переключатель).
- MessageBox (Окно сообщений).
- Menu (Меню).
- TabControl (Управление вкладками).
- Toolbar (Панель инструментов).
- *TreeView* (Дерево).
- *DataGrid* (Сетка данных).
- PictureBox (Изображение).
- RichTextBox (Текстовое поле с поддержкой формата *RTF*).

Работа с примерами программ Windows Forms в Visual C#

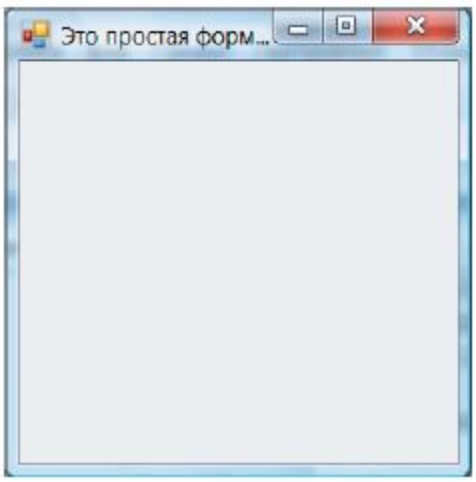
Возможно, вы предпочтете не использовать уже заготовленные примеры проектов, а *разрабатывать их "с нуля"*. В таком случае нужно учесть, что для каждого проекта C# сразу же создает два файла (с именами **Form1.cs** и **Program.cs**) и наполняет их исходным кодом на языке C#, то есть вы изначально получаете простейшую, но полноценную программу. Предлагаемый нами способ работы с уже полученным проектом состоит в выполнении следующих действий:

- Удалите файл Form1.cs.
- Замените код в файле Program.cs на код примера, с которым вы работаете.

Пример программы № 1

Рассмотрим пример простейшего приложения Windows Forms. Оно всего лишь создает новую форму и выводит определенный текст в заголовок окна формы. В тексте выводимом в заголовке должна быть указана номер программы, Ваша фамилия, номер группы.

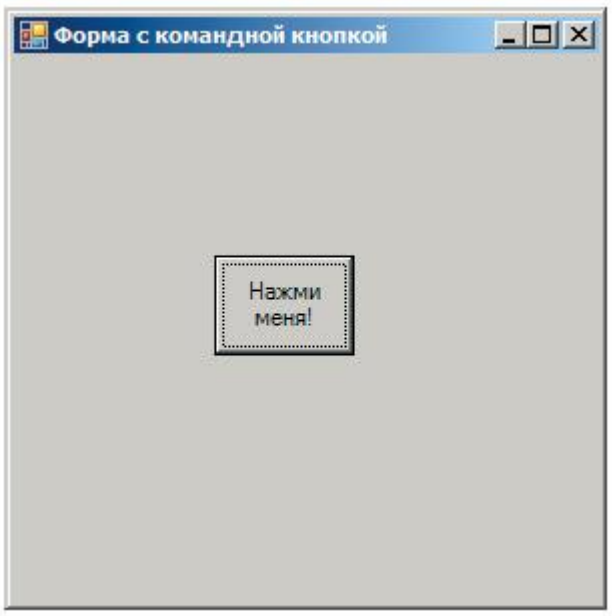
```
using System.Windows.Forms;
 class SimpleWindowsForm : Form
 {
  // Метод-конструктор нашего класса
  public SimpleWindowsForm()
  {
   // Указываем заголовок окна
   this.Text = "Это простая форма с заголовком";
  }
  static void Main()
  {
   // Создаем новый экземпляр класса
   //и запускаем его на выполнение
   // В результате на экране дисплея откроется форма
   Application.Run(new SimpleWindowsForm());
  }
 }
```



Пример программы № 2

Следующий пример — размещаем на форме кнопку.

```
using System.Windows.Forms;
 class SimpleWindowsFormWithButton : Form
 {
  Button button1;

  // Метод-конструктор нашего класса
  public SimpleWindowsFormWithButton()
  {
   // Указываем заголовок окна
   this.Text = "Форма с командной кнопкой";

   // Добавляем кнопку в коллекцию элементов управления формы
   // Хотя на кнопке написано: "Нажми меня!",
   // пока при нажатии ничего не происходит!
   button1 = new Button();
   button1.Text = "Нажми меня!";
   button1.Top = 100;
   button1.Left = 100;
   button1.Height = 50;
   button1.Width = 70;
   this.Controls.Add(button1);
  }

  static void Main()
  {
   // Создаем и запускаем форму
   Application.Run(new SimpleWindowsFormWithButton());
  }
 }
```

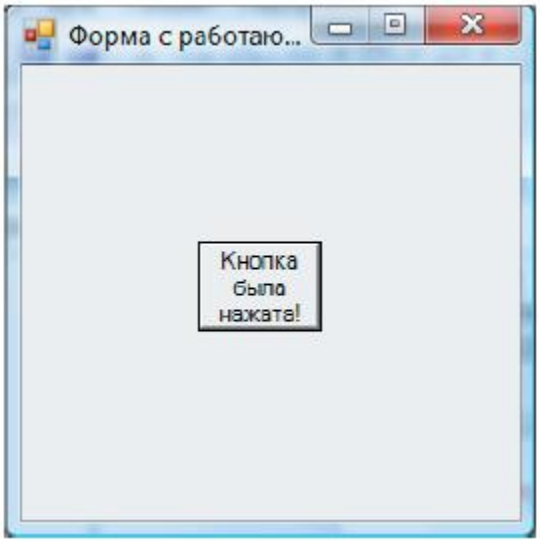


Пример программы № 3

Кнопку на форму мы поместили, но при нажатии на нее ничего не происходит. Нам нужно описать метод, который будет выполнять какое-либо действие при нажатии на кнопку. Пусть при этом текст в заголовке окна будет меняться. Поскольку такой метод отслеживает наступление некоторого события (в нашем случае – нажатие на кнопку) и затем каким-то образом обрабатывает его, он, напомним, называется "обработчик события". Кроме того, надо привязать обработчик события к соответствующему событию, то есть к нажатию на кнопку.

```
using System;
using System.Windows.Forms;
using System.Drawing;

class FormWithWorkingButton : Form
{
 Button mrButton;
 // Метод-конструктор нашего класса
 public FormWithWorkingButton()
 {
  // Указываем заголовок окна
  this.Text = "Форма с работающей кнопкой!";
  // Добавляем кнопку и привязываем ее к обработчику события
  mrButton = new Button();
  mrButton.Text = "Нажми меня";
  mrButton.Top = 100;
  mrButton.Left = 100;
  mrButton.Height = 50;
  mrButton.Width = 70;
  mrButton.Click += new System.EventHandler(mrButton_Click);
  this.Controls.Add(mrButton);
 }

 static void Main()
 {
```

```
  // Создаем и запускаем форму
  Application.Run(new FormWithWorkingButton());
 }

 // Обработчик события, срабатывающий при нажатии кнопки
 void mrButton_Click(object sender, EventArgs e)
 {
  // Изменяем текст
  mrButton.Text = "Кнопка была нажата!";
 }
}
```



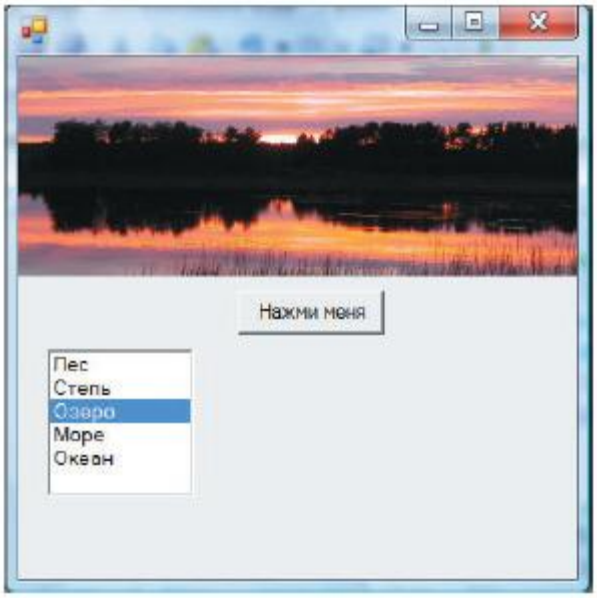
Пример программы № 4

Наша программа умеет выполнять основные действия. Теперь добавим на форму несколько новых элементов управления, аккуратно разместим их и немного поработаем с ними. Возьмем элементы управления 4-х типов: Button, ListBox, MessageBox и PictureBox.

Обратите внимание: кроме System.Windows.Forms в этом примере упоминается пространство имен System.Drawing. Дело в том, что мы используем элемент управления PictureBox, а для работы с изображениями требуются классы Drawing.

```
using System.Windows.Forms;
using System.Drawing;

class MyForm : Form
{
 // Объявим элемент ListBox как поле класса:
 // нам придется обращаться к нему из разных методов
 ListBox listBox1;

 // Метод-конструктор нашего класса
 public MyForm()
 {
  //Размеры формы
  this.Size = new Size(400, 400);
  // Создадим элемент PictureBox, поместим в него изображение,
  // добавим его на форму
  PictureBox pictureBox1 = new PictureBox();
```

```
  pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
  Bitmap image1 = new Bitmap ("..//..//images//Zakat.jpg");
  pictureBox1.ClientSize = new Size(this.Width, 150);
  pictureBox1.Image = (Image)image1;
  this.Controls.Add(pictureBox1);
  // Создаем объект Button, определяем некоторые из его свойств
  Button button1 = new System.Windows.Forms.Button();
  button1.Location = new Point(150, 160);
  button1.Size = new Size(100, 30);
  button1.Text = "Нажми меня";
  button1.Click += new System.EventHandler(button1_Click);
  this.Controls.Add(button1);

  // Создаем ListBox, определяем свойства и добавляем на форму
  listBox1 = new System.Windows.Forms.ListBox();
  listBox1.Location = new System.Drawing.Point(20, 200);
  listBox1.Size = new Size(100, 100);
  listBox1.Items.Add("Лес");
  listBox1.Items.Add("Степь ");
  listBox1.Items.Add("Озеро");
  listBox1.Items.Add("Море");
  listBox1.Items.Add("Океан");
  listBox1.SelectedIndex = 2;
  this.Controls.Add(listBox1);
 }

 // Обработчик события, срабатывающий при нажатии кнопки
 void button1_Click(object sender, System.EventArgs e)
 {
  // Выводим сообщение с указанием выбранного в списке пункта
  MessageBox.Show(this, "Вы выбрали " + listBox1.SelectedItem,
  "Уведомление", MessageBoxButtons.OK);
 }

 static void Main()
 {
  // Создаем и запускаем форму
  Application.Run(new MyForm());
 }

 private void InitializeComponent()
 {
  this.SuspendLayout();
  //
  // MyForm
  //
  this.BackColor = System.Drawing.SystemColors.Control;
  this.ClientSize = new System.Drawing.Size(292, 273);
  this.Name = "MyForm";
  this.ResumeLayout(false);
 }
}
```



Пример программы № 5

Программа вам пригодится, когда нужно будет вспомнить, для чего нужен тот или иной элемент управления.

Программа весьма интересна и стоит того, чтобы вы тщательно с ней поработали. После ее запуска открывается форма, в которой меню построено при помощи *элемента управления TreeView*. Пункты меню представлены в виде листьев на ветке дерева. Выбор того или иного листочка приводит к тому, что обработчик события заселяет форму новыми элементами управления. Поскольку эти элементы задаются определенным пунктом меню, их тип зависит от того, какой именно пункт выбран. С добавляемыми элементами тоже можно работать.

Благодаря этому проекту появляется возможность познакомиться с самыми разными элементами управления, а главное — понять, как они создаются и как их использовать в собственных проектах.

Напоминаем: для того чтобы использовать элементы управления PictureBox и DataGridView, требуются пространства имен System.Drawing, System.Data и System.Xml.

```
using System;
using System.Windows.Forms;
using System.Drawing;
using System.Data;
using System.Xml;

class FormWithManyControls : Form
{
 TreeView treeView1;
 Panel panel1;
 CheckBox checkBox1, checkBox2;
 RadioButton radioButton1, radioButton2;
 ListBox listBox1;

 // Метод-конструктор нашего класса
 public FormWithManyControls()
```

```
{
  // Указываем размеры и заголовок окна

  this.Text = "Форма, включающая различные элементы управления!";
  this.Height = 800; this.Width = 900;
  // Добавляем элемент TreeView в качестве своеобразного меню

  treeView1 = new TreeView();
  treeView1.BackColor = Color.BurlyWood;
  treeView1.Dock = DockStyle.Left;
  treeView1.AfterSelect +=
  new System.Windows.Forms.TreeViewEventHandler(treeView1_AfterSelect);

  TreeNode tn = new TreeNode("Элементы");
  tn.Expand();

  tn.Nodes.Add(new TreeNode("[Очистить]"));
  tn.Nodes.Add(new TreeNode("Label"));
  tn.Nodes.Add(new TreeNode("Button"));
  tn.Nodes.Add(new TreeNode("CheckBox"));
  tn.Nodes.Add(new TreeNode("RadioButton"));
  tn.Nodes.Add(new TreeNode("ListBox"));
  tn.Nodes.Add(new TreeNode("TextBox"));
  tn.Nodes.Add(new TreeNode("TabControl"));
  tn.Nodes.Add(new TreeNode("DataGridView"));
  tn.Nodes.Add(new TreeNode("MainMenu"));
  tn.Nodes.Add(new TreeNode("ToolBar"));
  tn.Nodes.Add(new TreeNode("PictureBox"));
  tn.Nodes.Add(new TreeNode("RichTextBox"));

  treeView1.Nodes.Add(tn);

  this.Controls.Add(treeView1);

  // Добавляем панель для размещения остальных элементов управления

  panel1 = new Panel();
  panel1.Dock = DockStyle.Right;
  panel1.BorderStyle = BorderStyle.Fixed3D;
  panel1.Width = this.Width - treeView1.Width;

  this.Controls.Add(panel1);
}

// Обработчик событий, срабатывающий при выборе одного из узлов дерева
// TreeView
private void treeView1_AfterSelect
(object sender, System.Windows.Forms.TreeViewEventArgs e)
{
  // Выполнение соответствующего действия при выборе любого из узлов

  if (e.Node.Text == "[Очистить]")
```

```
{
  // Удаляем с панели все элементы управления
  panel1.Controls.Clear();
}
else if (e.Node.Text == "Button")
{
  // Добавляем на панель кнопку

  Button button1 = new Button();
  button1.Text = "Нажми меня!";
  button1.Location = new Point(300, 20);
  button1.Width = 120;
  button1.Height = 40;
  button1.Click += new EventHandler(button1_Click);

  panel1.Controls.Add(button1);
}
else if (e.Node.Text == "Label")
{
  // Добавляем на панель метку

  Label label1 = new Label();
  label1.Text =
  "Это надпись. Используется для вывода текста на экран!";

  label1.Location = new Point(180, 70);
  label1.Width = 400;
  label1.Click += new EventHandler(label1_Click);
  panel1.Controls.Add(label1);
}
else if (e.Node.Text == "CheckBox")
{
  // Добавляем на панель несколько флажков

  checkBox1 = new CheckBox();
  checkBox1.Text = "Я способный!";
  checkBox1.Location = new Point(20, 40);
  checkBox1.Width = 150;
  checkBox1.CheckedChanged +=
  new EventHandler(CheckBox_CheckedChanged);
  panel1.Controls.Add(checkBox1);

  checkBox2 = new CheckBox();
  checkBox2.Text = "Я скромный!";
  checkBox2.Location = new Point(20, 80);
  checkBox2.Width = 150;
  checkBox2.CheckedChanged +=
  new EventHandler(CheckBox_CheckedChanged);
  panel1.Controls.Add(checkBox2);
}
else if (e.Node.Text == "RadioButton")
{
```

```
  // Добавляем на панель несколько переключателей

  radioButton1 = new RadioButton();
  radioButton1.Text = "Я добрый!";
  radioButton1.Location = new Point(20, 120);
  radioButton1.Width = 150;
  radioButton1.Height = 30;
  //radioButton1.Size = new Size(20, 100);

  radioButton1.CheckedChanged +=
  new EventHandler(RadioButton_CheckedChanged);

  panel1.Controls.Add(radioButton1);

  radioButton2 = new RadioButton();
  radioButton2.Text = "Я трудолюбивый!";
  radioButton2.Location = new Point(20, 160);
  radioButton2.Width = 150;
  radioButton2.Height = 30;
  //radioButton2.Size = new Size(20, 100);

  radioButton2.CheckedChanged +=
  new EventHandler(RadioButton_CheckedChanged);

  panel1.Controls.Add(radioButton2);
 }
 else if (e.Node.Text == "ListBox")
 {
  // Добавляем на панель список

  listBox1 = new ListBox();
  listBox1.Items.Add("Зеленый");
  listBox1.Items.Add("Желтый");
  listBox1.Items.Add("Голубой");
  listBox1.Items.Add("Серый");

  listBox1.Location = new Point(20, 250);
  listBox1.Width = 100; listBox1.Height = 100;
  listBox1.SelectedIndexChanged +=
  new EventHandler(listBox1_SelectedIndexChanged);

  panel1.Controls.Add(listBox1);
 }
 else if (e.Node.Text == "TextBox")
 {
  // Добавляем на панель текстовое поле

  TextBox textBox1 = new TextBox();
  textBox1.Multiline = true;
  textBox1.Text = "Это текстовое окно. Сюда можно вводить текст!" +
  "\r\n" + " Сотрите этот текст и введите свой!";
  textBox1.Location = new Point(180, 100);
```

```
  textBox1.Width = 400; textBox1.Height = 40;

  panel1.Controls.Add(textBox1);
 }
 else if (e.Node.Text == "DataGridView")
 {
  // Добавляем на панель таблицу, заполненную данными из файла xml

  DataSet dataSet1 = new DataSet("Пример DataSet");
  dataSet1.ReadXml("..//..//images//marks.xml");

  DataGridView dataGridView1 = new DataGridView();
  dataGridView1.Width = 250;
  dataGridView1.Height = 150;
  dataGridView1.Location = new Point(20, 500);
  dataGridView1.AutoGenerateColumns = true;
  dataGridView1.DataSource = dataSet1;
  dataGridView1.DataMember = "предметы";
  //dataGridView1.DataMember = "оценка";

  //dataGridView1.ColumnCount = 2;
  panel1.Controls.Add(dataGridView1);
 }
 else if (e.Node.Text == "TabControl")
 {
  // Добавляем на панель элемент управления вкладками
  // и наполняем каждую вкладку содержимым

  TabControl tabControl1 = new TabControl();
  tabControl1.Location = new Point(190, 150);
  tabControl1.Size = new Size(300, 300);

  TabPage tabPage1 = new TabPage("Вадик");
  PictureBox pictureBox1 = new PictureBox();
  pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
  pictureBox1.Image = new Bitmap("..//..//images//Vadik.jpg");

  pictureBox1.Size = new Size(300, 200);
  tabPage1.Controls.Add(pictureBox1);
  Label labelV = new Label();
  labelV.Top = 200;
  labelV.Size = new Size(300, 50);
  labelV.Text = "Это Вадик! Он любит купаться и работать на компьютере!";
  tabPage1.Controls.Add(labelV);
  tabControl1.TabPages.Add(tabPage1);

  TabPage tabPage2 = new TabPage("Его компьютер");
  PictureBox pictureBox2 = new PictureBox();
  pictureBox2.SizeMode = PictureBoxSizeMode.StretchImage;
  pictureBox2.Image = new Bitmap("..//..//images//comp.jpg");
  pictureBox2.Size = new Size(300, 200);
  tabPage2.Controls.Add(pictureBox2);
```

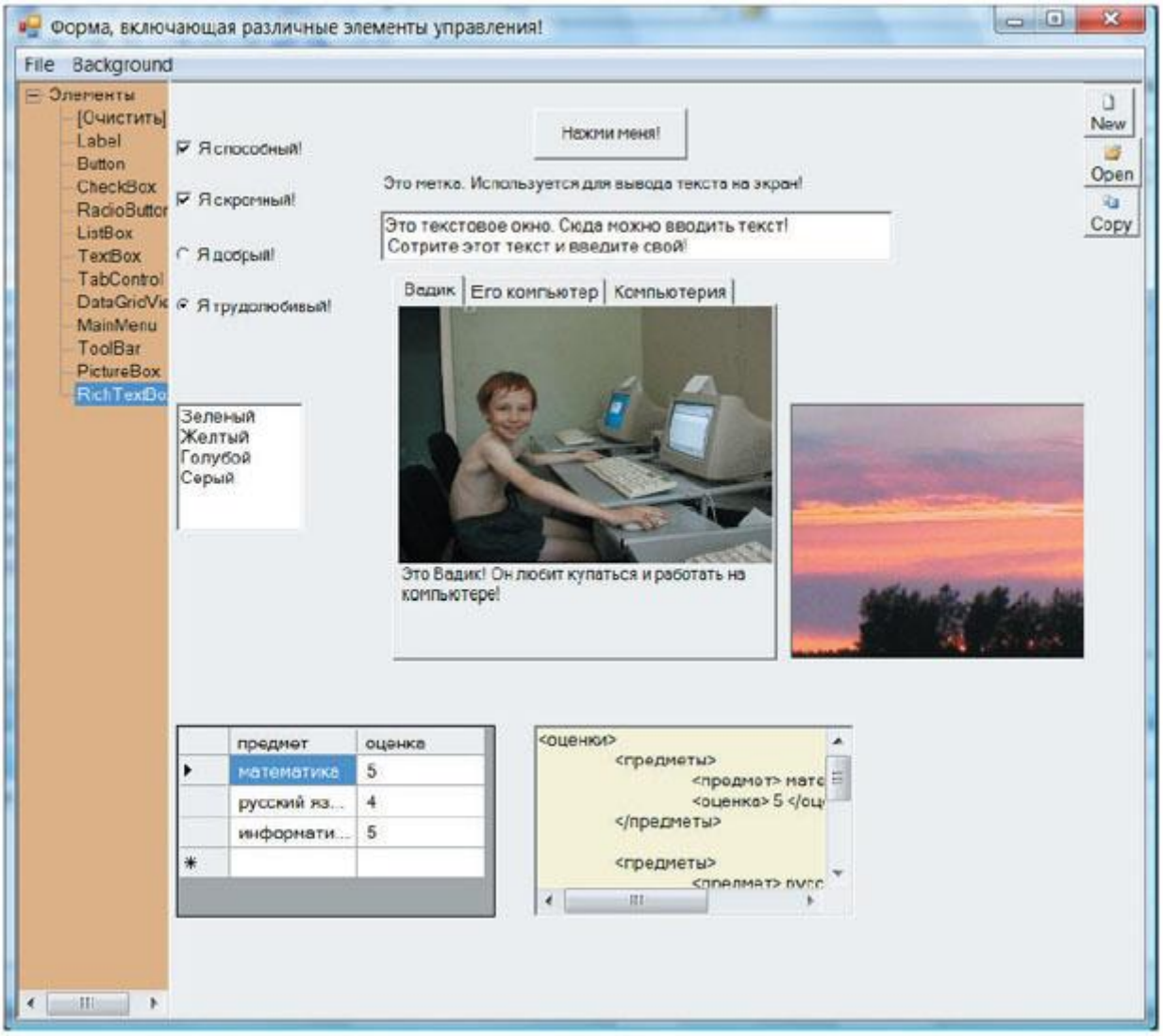
```
  Label labelC = new Label();
  labelC.Top = 200;
  labelC.Size = new Size(300, 50);
  labelC.Text = "Это компьютер Вадика! Пока Вадик купается, " +
  "он разрешает работать на компьютере ящерице!";
  tabPage2.Controls.Add(labelC);
  tabControl1.TabPages.Add(tabPage2);

  TabPage tabPage3 = new TabPage("Компьютерия");
  PictureBox pictureBox3 = new PictureBox();
  pictureBox3.SizeMode = PictureBoxSizeMode.StretchImage;
  pictureBox3.Image = new Bitmap("..//..//images//terra.jpg");
  pictureBox3.Size = new Size(300, 200);
  tabPage3.Controls.Add(pictureBox3);
  Label labelT = new Label();
  labelT.Top = 200;
  labelT.Size = new Size(300, 50);
  labelT.Text = "Это страна Компьютерия!" +
  " Она расположена на берегу реки Тверца в Тверской области!";
  tabPage3.Controls.Add(labelT);
  tabControl1.TabPages.Add(tabPage3);

  panel1.Controls.Add(tabControl1);
 }
 else if (e.Node.Text == "PictureBox")
 {
  // Добавляем на панель изображение

  PictureBox pictureBox1 = new PictureBox();
  pictureBox1.Image = new Bitmap("..//..//images//Zakat.jpg");
  pictureBox1.BorderStyle = BorderStyle.Fixed3D;
  pictureBox1.Location = new Point(500, 250);
  pictureBox1.Size = new Size(250, 200);

  panel1.Controls.Add(pictureBox1);
 }
 else if (e.Node.Text == "RichTextBox")
 {
  // Добавляем поле для ввода текста с форматированием
  // Загружаем в него данные из файла XML

  RichTextBox richTextBox1 = new RichTextBox();
  richTextBox1.LoadFile("..//..//images//marks.xml",
  RichTextBoxStreamType.UnicodePlainText);
  richTextBox1.WordWrap = false;
  richTextBox1.BorderStyle = BorderStyle.Fixed3D;
  richTextBox1.BackColor = Color.Beige;
  richTextBox1.Size = new Size(250, 150);
  richTextBox1.Location = new Point(300, 500);
  // panel1.Height - richTextBox1.Height - 5);

  panel1.Controls.Add(richTextBox1);
```

```
}
else if (e.Node.Text == "MainMenu")
{
  // Добавляем классическое "меню" (появляется в верхней части окна)
  MainMenu mainMenu1 = new MainMenu();

  MenuItem menuItem1 = new MenuItem("File");
  menuItem1.MenuItems.Add("Exit",
  new EventHandler(mainMenu1_Exit_Select));
  mainMenu1.MenuItems.Add(menuItem1);

  MenuItem menuItem2 = new MenuItem("Background");
  menuItem2.MenuItems.Add("Choose",
  new EventHandler(mainMenu1_ColorOwn_Select));
  menuItem2.MenuItems.Add("White",
  new EventHandler(mainMenu1_ColorWhite_Select));
  mainMenu1.MenuItems.Add(menuItem2);

  this.Menu = mainMenu1;

  MessageBox.Show("Главное меню добавлено в окно " +
  "Испытайте его после нажатия OK.");
}
else if (e.Node.Text == "ToolBar")
{
  // Добавляем на панель элемент "панель управления" с кнопками
  // быстрого вызова
  ToolBar toolBar1 = new ToolBar();
  toolBar1.Size = new Size(100, 100);
  toolBar1.Dock = DockStyle.Right;
  ImageList imageList1 = new ImageList();
  imageList1.Images.Add(new Bitmap("../../images/new.gif"));
  imageList1.Images.Add(new Bitmap("../../images/open.gif"));
  imageList1.Images.Add(new Bitmap("../../images/copy.gif"));
  toolBar1.ImageList = imageList1;

  ToolBarButton toolBarbutton1 = new ToolBarButton("New");
  toolBarbutton1.ImageIndex = 0;
  toolBar1.Buttons.Add(toolBarbutton1);

  ToolBarButton toolBarbutton2 = new ToolBarButton("Open");
  toolBarbutton2.ImageIndex = 1;
  toolBar1.Buttons.Add(toolBarbutton2);

  ToolBarButton toolBarButton3 = new ToolBarButton("Copy");
  toolBarButton3.ImageIndex = 2;
  toolBar1.Buttons.Add(toolBarButton3);
  toolBar1.ButtonClick +=
  new ToolBarButtonClickEventHandler(toolBar1_Click);

  panel1.Controls.Add(toolBar1);
}
```

```
}
/* Обработчики событий для добавленных выше элементов управления */

// Обработчик события, срабатывающий при щелчке мышью на метке
void label1_Click(object sender, System.EventArgs e)
{
  MessageBox.Show
  ("Да, у меток тоже есть событие Click. Но для них включение событий -
  редкость.");
}

// Обработчик события, срабатывающий при нажатии кнопки
void button1_Click(object sender, System.EventArgs e)
{
MessageBox.Show("Пора, наконец-то вы нажали меня!");
}
// Обработчик события, срабатывающий при установке или снятии флажка
void CheckBox_CheckedChanged(object sender, System.EventArgs e)
{
  if (checkBox1.Checked && checkBox2.Checked)
  {
    MessageBox.Show("У Вас все получится!");
  }
  else if (checkBox1.Checked)
  {
    MessageBox.Show("Не здорово быть умным и не скромным!");
  }
  else if (checkBox2.Checked)
  {
    MessageBox.Show("Скромность украшает. Хорошо бы еще быть умным!");
  }
  else
  {
    MessageBox.Show("Ни скромности, ни таланта?");
  }
}

// Обработчик события, срабатывающий при нажатии переключателя
void RadioButton_CheckedChanged(object sender, System.EventArgs e)
{
  if (radioButton1.Checked)
  {
    MessageBox.Show("Доброту все любят!");
  }
  else if (radioButton2.Checked)
  {
    MessageBox.Show("Это замечательно!");
  }
}

// Обработчик события, срабатывающий при выборе одного из пунктов списка
void listBox1_SelectedIndexChanged(object sender, System.EventArgs e)
```

```
{
  switch (listBox1.SelectedItem.ToString())
  {
    case ("Зеленый"): treeView1.BackColor = Color.Green; break;
    case ("Желтый"): treeView1.BackColor = Color.Yellow; break;
    case ("Голубой"): treeView1.BackColor = Color.Blue; break;
    case ("Серый"): treeView1.BackColor = Color.Gray; break;
  }
}

// Обработчик события, срабатывающий при выборе в меню пункта "White"
void mainMenu1_ColorWhite_Select(object sender, System.EventArgs e)
{
  treeView1.BackColor = Color.White;
}

// Обработчик события, срабатывающий при выборе в меню цвета
void mainMenu1_ColorOwn_Select(object sender, System.EventArgs e)
{
  ColorDialog colorDialog1 = new ColorDialog();
  colorDialog1.Color = treeView1.BackColor;
  colorDialog1.ShowDialog();
  treeView1.BackColor = colorDialog1.Color;
}

// Обработчик события, срабатывающий при выборе в меню пункта "exit"
void mainMenu1_Exit_Select(object sender, System.EventArgs e)
{
  if (
  MessageBox.Show("Вы уверены, что хотите закончить работу?",
  "Exit confirmation", MessageBoxButtons.YesNo)
  == DialogResult.Yes
  )
  {
    this.Dispose();
  }
}

// Обработчик события, срабатывающий при нажатии кнопки на панели инструментов
void toolBar1_Click
(object sender, System.Windows.Forms.ToolBarButtonClickEventArgs e)
{
  if (e.Button.Text == "Open")
  {
    MessageBox.Show("Здесь мог бы открываться файл!");
  }
  else if (e.Button.Text == "New")
  {
    MessageBox.Show("Здесь мог бы создаваться файл!");
  }
  else if (e.Button.Text == "Copy")
  {
```

```
    MessageBox.Show("Здесь мог бы копироваться файл!");
   }
  }
  static void Main()
  {
  // // Создаем и запускаем форму
  Application.Run(new FormWithManyControls());
  }
}
```



Рисование

Классы, объединенные в *пространство* имен Drawing, позволяют работать с различными изображениями. Существует два основных типа компьютерных изображений.

Растровые представляют собой набор точек. Примером могут служить фотографии и значки.

Векторная графика — это изображения, составленные из геометрических фигур: линий, окружностей, прямоугольников и т. д. Например, план дома удобно представлять в виде векторного изображения.

Для начала продемонстрируем работу с *растровой графикой*. На компьютере часто приходится выполнять обработку изображений, например, фотографий. Для этого в библиотеке классов *.NET* Framework имеется немало полезных средств.

Пример программы № 6

Для создания программы, отображающей на форме рисунок, хранящийся в файле, нам понадобится специальный элемент управления. Для этой цели прекрасно подходит PictureBox.

```
using System;
using System.Windows.Forms;
using System.Drawing;

class PictureDisplayer : Form
{
  Bitmap image1;
  PictureBox pictureBox1;

  // Метод-конструктор нашего класса
  public PictureDisplayer()
  {
    // Указываем размеры и заголовок окна

    this.Text = "Искусство аборигенов";
    this.Size = new Size(302, 240);

    // Подготавливаем поле для размещения изображения

    pictureBox1 = new PictureBox();
    pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
    pictureBox1.BorderStyle = BorderStyle.Fixed3D;
    pictureBox1.ClientSize = new Size(300, 196);

    // Добавляем изображение в элемент PictureBox

    image1 = new Bitmap(@"../../images/Iskusstvo.jpg");
    pictureBox1.Image = (Image)image1;

    // Добавляем PictureBox (с изображением) на форму

    this.Controls.Add(pictureBox1);
  }
  static void Main()
  {
    // Создаем и запускаем форму
    Application.Run(new PictureDisplayer());
  }
}
```

Пример программы № 7

Следующая *программа* загружает фотографию с диска и после нажатия кнопки "*flip*" (Перевернуть) позволяет получить ее зеркальное отражение, расположенное по горизонтали:

```
using System;
using System.Windows.Forms;
using System.Drawing;

class PictureFlipper : Form
{
  Button button1;
  Bitmap image1;
  PictureBox pictureBox1;

  // Метод-конструктор нашего класса
  public PictureFlipper()
  {
    // Указываем размеры и заголовок окна

    this.Text = "Поворот рисунка";
    this.Size = new Size(302, 240);

    // Добавляем на форму кнопку

    button1 = new Button();
    button1.Text = "Поворот рисунка";
    button1.Location = new Point(100, 150);
    button1.Size = new Size(70, 40);
    button1.Click += new System.EventHandler(button1_Click);
    this.Controls.Add(button1);

    // Добавляем элемент PictureBox на форму

    pictureBox1 = new PictureBox();
    pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
    pictureBox1.BorderStyle = BorderStyle.Fixed3D;
    pictureBox1.ClientSize = new Size(300, 196);

    // Добавляем изображение в элемент PictureBox
```

```
    image1 = new Bitmap(@"../../images/Giraf.jpg");
    pictureBox1.Image = (Image)image1;

    // Добавляем на форму элемент PictureBox

    this.Controls.Add(pictureBox1);
  }
  static void Main()
  {
    // Создаем и запускаем форму
    Application.Run(new PictureFlipper());
  }

  // Обработчик события, срабатывающий при нажатии кнопки
  void button1_Click(object sender, EventArgs e)
  {
    // Отображает изоражение относительно оси X (горизонтально)
    image1.RotateFlip(RotateFlipType.RotateNoneFlipX);

    // Повторно вставляем изображение в элемент PictureBox
    pictureBox1.Image = (Image)image1;

    // Обновляем заголовок окна
    this.Text = "Рисунок после поворота!";
  }
}
```

Теперь перейдем к примерам работы с *векторной графикой* — изображениям, составленным из геометрических фигур.

Во всех примерах будут создаваться кнопка и обработчик событий, отвечающий за то, чтобы работа с графикой начиналась только после нажатия кнопки.

Необходимо усвоить несколько важных принципов. Они вполне логичны, но все-таки следует уяснить их, чтобы избежать возможных затруднений.

1. В обычном мире, прежде чем нарисовать линию, окружность, прямоугольник или иную фигуру, необходимо выбрать карандаш нужного цвета с грифелем определенной толщины. Для отрисовки на компьютере простейших фигур надо сначала

создать **объект** *Pen* (**Перо**). Например, с помощью данного фрагмента кода создается объект *Pen*, который рисует зеленую линию толщиной 3 пикселя:

```
Pen myGreenPen = new Pen(Color.Green, 3);
```

2. Для рисования **фигур с заливкой** потребуется нечто вроде кисти с красками. Предварительно следует создать объект Brush (**Кисть**), а затем выбрать цвет заливки и один из многочисленных типов кисти. В следующем фрагменте кода создается объект SolidBrush (**Сплошная кисть**) голубого цвета:

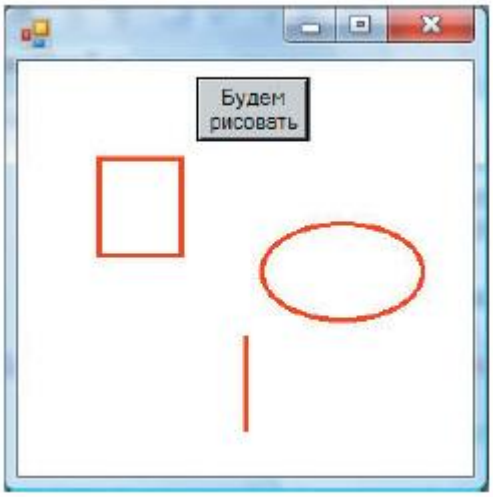
```
SolidBrush myBlueBrush = new SolidBrush(Color.Blue);
```

Пример программы № 8

В этой программе в методе, названном DrawSomeShapes, рисуется линия, *прямоугольник* и *эллипс*.

```
using System;
using System.Windows.Forms;
using System.Drawing;

class SimpleShapeMaker : Form
{
  // Метод-конструктор нашего класса
  public SimpleShapeMaker()
  {
    // Меняем цвет фона формы на белый

    this.BackColor = Color.White;

    // Добавляем на форму кнопку и привязываем ее к обработчику событий

    Button button1 = new Button();
    button1.Text = "Будем рисовать!";
    button1.Location = new Point(110, 10);
    button1.Size = new Size(70, 40);
    button1.BackColor = Color.LightGray;
    button1.Click += new System.EventHandler(button1_Click);
    this.Controls.Add(button1);
  }

  // Обработчик события, срабатывающий при нажатии кнопки
  void button1_Click(object o, System.EventArgs e)
  {
    // Вызов метода
    DrawSomeShapes();
  }

  // Метод для отрисовки на поверхности формы нескольких фигур
  void DrawSomeShapes()
  {
    // Подготовка области рисования на форме
    Graphics g = this.CreateGraphics();

    // Подготавливаем перо, рисующее красную линию толщиной 3 пикселя
    Pen redPen = new Pen(Color.Red, 3);
```

```
    // С помощью пера рисуем прямую линию, прямоугольник и эллипс
    g.DrawLine(redPen, 140, 170, 140, 230);
    g.DrawRectangle(redPen, 50, 60, 50, 60);
    g.DrawEllipse(redPen, 150, 100, 100, 60);

    // Очистка
    g.Dispose();
  }
  static void Main()
  {
    // Создаем и запускаем форму
    Application.Run(new SimpleShapeMaker());
  }
}
```



Пример программы № 9

Теперь попробуем поиграть с мышкой – работать с графикой удобнее при помощи мыши, а не клавиатуры. Мы будем обрабатывать как растровые, так и *векторные* изображения, используя некоторые события мыши.

Постараемся освоить некоторые новые действия, а именно — действия с точечными рисунками. Некоторые принципы работы приведенного ниже кода.

- Компьютерные программы формируют изображение на экране монитора, управляя цветом и яркостью маленьких точек, которые называются *пикселями*.
- Цвет пикселя определяется тремя цветовыми компонентами: красной (red), зеленой (green) и синей (blue) – в языках программирования часто используется сокращение RGB. Цвет и яркость пикселя можно изменять, регулируя интенсивность компонентов RGB, в пределах от 0 до 255 единиц. Например:
  - если red=255, green=0, blue=0 — цвет пикселя будет ярко-красным;
  - если red=255, green=255, blue=0 — цвет пикселя желтый.
- Компьютер может отслеживать положение курсора мыши, определяемое координатами X и Y (горизонтальная и вертикальная координаты). Так, верхний левый угол экрана имеет координаты X=0 и Y=0.

```
using System;
```

```
using System.Windows.Forms;
using System.Drawing;

class FunWithTheMouse : Form
{
  // Объявляем объекты, доступные для разных методов

  PictureBox pictureBox1;
  Label label1;
  Point spotClicked;

  // Метод-конструктор нашего класса
  public FunWithTheMouse()
  {
    // Задаем размеры окна

    this.Size = new Size(640, 480);

    // Загружаем рисунок в элемент PictureBox и вставляем в форму

    pictureBox1 = new PictureBox();
    pictureBox1.Image = (Image)new Bitmap(@"../../images/Dog.bmp");
    pictureBox1.SizeMode = PictureBoxSizeMode.Normal;
    pictureBox1.Dock = DockStyle.Fill;
    this.Controls.Add(pictureBox1);

    // Добавляем метку с инструкциями в нижнюю часть экрана

    label1 = new Label();
    label1.BackColor = Color.Wheat;
    label1.Dock = DockStyle.Bottom;
    label1.Text =
    "При нажатой левой кнопке мыши можно рисовать прямоугольники. " +
    "Нажатая правая кнопка изменяет яркость прямоугольника " +
    "Нажав SHIFT и перемещая мышь, рисуем желтые кружки.";
    label1.TextAlign = ContentAlignment.MiddleCenter;
    this.Controls.Add(label1);

    // Привязываем PictureBox к обработчикам событий мыши

    this.pictureBox1.MouseDown += new MouseEventHandler(MouseButtonIsDown);
    this.pictureBox1.MouseUp += new MouseEventHandler(MouseButtonIsUp);
    this.pictureBox1.MouseMove += new MouseEventHandler(TheMouseMoved);
  }
  // Обработчик событий, срабатывающий при ПЕРЕМЕЩЕНИИ мыши
  public void TheMouseMoved(object sender, MouseEventArgs e)
  {
    // Если на клавиатуре нажата клавиша SHIFT
    if ((Control.ModifierKeys & Keys.Shift) == Keys.Shift)
  {
    // Подготовка области рисования на изображении
    System.Drawing.Graphics g =
```

```
    this.pictureBox1.CreateGraphics();

    // Используем желтое перо
    System.Drawing.Pen yellowPen = new
    System.Drawing.Pen(Color.Yellow, 3);

    // Рисуем окружность (эллипс, вписанный в квадрат)
    // Верхний левый угол квадрата имеет координаты X и Y
    // текущего положения мыши.
    g.DrawEllipse(yellowPen, e.X, e.Y, 40, 40);

    // Очистка
    g.Dispose();
  }
}

// Обработчик событий, срабатывающий при НАЖАТИИ кнопки мыши
public void MouseButtonIsDown(object sender, MouseEventArgs e)
{
  // Запоминаем точку, в которой произошло нажатие кнопки мыши.
  // Когда кнопка будет отпущена, нам понадобятся ее координаты

  spotClicked.X = e.X;// горизонтальная координата
  spotClicked.Y = e.Y;// вертикальная координата
}

// Обработчик событий, срабатывающий при ОТЖАТИИ кнопки мыши
public void MouseButtonIsUp(object sender, MouseEventArgs e)
{
  /* Пользователь отпустил кнопку мыши! */

  // Создаем прямоугольник (пока он еще не виден), ограничивающий
  // область изображения, с которой пользователь будет работать

  Rectangle r = new Rectangle();

  // Левый верхний угол прямоугольника соответствует точке,
  // в которой была нажата кнопка мыши.
  // Мы сохранили ее координаты.

  r.X = spotClicked.X;
  r.Y = spotClicked.Y;

  // Ширина и высота прямоугольника вычисляется
  // путем вычитания координат мыши в точке нажатия
  // из текущих координат (в точке отжатия кнопки).

  r.Width = e.X – spotClicked.X;
  r.Height = e.Y – spotClicked.Y;

  if (e.Button == MouseButtons.Left)
  {
```

```
    /* Если была нажата и отпущена левая кнопка мыши,
    рисуем видимый контур прямоугольника */

    // Подготовка области рисования на изображении
    Graphics g = this.pictureBox1.CreateGraphics();

    // Рисуем красный контур прямоугольника

    Pen redPen = new Pen(Color.Red, 2);
    g.DrawRectangle(redPen, r);
   }
   else
   {
    // Если была нажата другая кнопка, вызываем
    // метод, подсвечивающий область изображения

    ChangeLightness(r);
   }
}

// Метод, увеличивающий яркость выбранного участка изображения
// путем увеличения яркости каждого пикселя этого участка
public void ChangeLightness(Rectangle rect)
{
  int newRed, newGreen, newBlue;
  Color pixel;

  // Копируем изображение, загруженное в PictureBox

  System.Drawing.Bitmap picture = new
  Bitmap(this.pictureBox1.Image);

  // Операция увеличения яркости может занять много времени,
  // пользователя предупреждают, если выбран большой участок.

  if ((rect.Width > 150) || (rect.Height > 150))
  {

    DialogResult result = MessageBox.Show(
    "Выделенная область велика! " +
    "Изменение яркости может требовать значительного времени!",
    "Warning", MessageBoxButtons.OKCancel);

    // При нажатии кнопки Cancel (Отмена) выходим из метода
    // и возвращаемся к месту его вызова

    if (result == DialogResult.Cancel) return;
  }
  /* Перебираем последовательно все пиксели данного участка
  и удваиваем значение яркости компонент RGB пикселей */

  // Перебор по горизонтали слева направо...
```

```
 for (int x = rect.X; x < rect.X + rect.Width; x++)
 {
  // и по вертикали сверху вниз...

  for (int y = rect.Y; y < (rect.Y + rect.Height); y++)
  {
   // Считываем текущий пиксель

   pixel = picture.GetPixel(x, y);

   // Увеличиваем яркость цветовых компонент пикселя

   newRed = (int)Math.Round(pixel.R * 2.0, 0);
   if (newRed > 255) newRed = 255;
   newGreen = (int)Math.Round(pixel.G * 2.0, 0);
   if (newGreen > 255) newGreen = 255;
   newBlue = (int)Math.Round(pixel.B * 2.0, 0);
   if (newBlue > 255) newBlue = 255;

   // Присваиваем пикселю новые цветовые значения

   picture.SetPixel
   (x, y, Color.FromArgb(
   (byte)newRed, (byte)newGreen, (byte)newBlue));
  }
 }
 // Помещаем измененную копию изображения в PictureBox,
 // чтобы изменения отобразились на экране

 this.pictureBox1.Image = picture;
 }
 static void Main()
 {
  // Создаем экземпляр класса формы
  Application.Run(new FunWithTheMouse());
 }
}
```

ЛАБОРАТОРНАЯ РАБОТА № 2
ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ VISUAL STUDIO. ПЛАТФОРМА .NET
ТЕХНОЛОГИЯ РАЗРАБОТКИ WINDOWS ПРИЛОЖЕНИЙ. ПРИЛОЖЕНИЯ WINDOWS FORMS.
ОСНОВНАЯ МЕТОДИКА СЧИТЫВАНИЯ ФАЙЛА XML В НАБОР ДАННЫХ И СОЗДАНИЯ СХЕМЫ НА ОСНОВЕ СОДЕРЖИМОГО ФАЙЛА XML

Цель: освоить инструмент «элементов управления» с целью создания различных приложений.

Задание на лабораторную работу

1. Создать учебные примеры (примеры №1-№5) и разобрать принцип их работы.
2. Отчет представить в электронном виде: листинг кода с Вашими комментариями, скриншоты результатов работы программы. Заголовок формы –Ваша фамилия, группа.
3. При защите лабораторной работы демонстрируете преподавателю понимание работы кода.
4. Проекты сохраняете на своем флэш-накопителе для работы в следующих лабораторных работах.
5. Вернуться к лабораторной работе № 1 и в примере № 5 отработать с полученным в этой лабораторной работе файлом authors.xml . (Создание набора данных, чтение XML-файла и отображение его в элементе управления DataGridView. Добавление кода для отображения XML-схемы, основанной на XML-файле, в элементе управления TextBox или Rich Textbox)

ADO.NET предоставляет простые методы обработки XML-данных. В этом пошаговом руководстве описывается создание приложения Windows, загружающего XML-данные в набор данных. Набор данных будет затем показан в DataGridView. В конце примера в текстовом поле отображается схема XML на основе содержимого файла XML.

Пример включает в себя пять основных этапов:

1. Создание нового проекта.
2. Создание файла XML, считываемого в набор данных.
3. Создание пользовательского интерфейса
4. Создание набора данных, чтение XML-файла и отображение его в элементе управления DataGridView.
5. Добавление кода для отображения XML-схемы, основанной на XML-файле, в элементе управления TextBox.

Примечание

Отображаемые диалоговые окна и команды меню могут отличаться от описанных в справке в зависимости от текущих настроек или выпуска. Чтобы изменить параметры, в меню **Сервис** выберите команду **Импорт и экспорт параметров**. Дополнительные сведения см. в разделе Customizing Development Settings in Visual Studio.

Создание нового проекта

На данном этапе создается проект Visual C#, в рамках которого будет реализован этот пример.

Чтобы создать новый проект Windows

1. В меню **Файл** создайте новый проект.
2. Назовите проект ReadingXML.
3. Выберите **Приложение Windows** и нажмите кнопку **OK**.

Проект **ReadingXML** создан и добавлен в "Обозреватель решений".

Пример № 1

Создание файла XML, считываемого в набор данных.

Поскольку в пошаговом руководстве основной упор делается на считывание данных XML в набор данных, предоставляется содержимое некоторого файла XML.

Создание файла XML, считываемого в набор данных.

1. В меню **Проект** выберите **Добавить новый элемент**. (или в обозревателе решений на ReadingXML правой кнопкой мышки – добавить-создать элемент- **XML-файл)**
2. Выберите пункт **XML-файл**, назовите файл authors.xml и нажмите кнопку **Добавить**.

Файл XML загрузится в конструктор в режиме редактирования.

3. Вставьте следующий код в редактор после объявления XML.

XML

```
<Authors_Table>
  <authors>
    <au_id>172-32-1176</au_id>
    <au_lname>White</au_lname>
    <au_fname>Johnson</au_fname>
    <phone>408 496-7223</phone>
    <address>10932 Bigge Rd.</address>
    <city>Menlo Park</city>
    <state>CA</state>
    <zip>94025</zip>
    <contract>true</contract>
  </authors>
  <authors>
    <au_id>213-46-8915</au_id>
    <au_lname>Green</au_lname>
    <au_fname>Margie</au_fname>
    <phone>415 986-7020</phone>
    <address>309 63rd St. #411</address>
    <city>Oakland</city>
    <state>CA</state>
    <zip>94618</zip>
    <contract>true</contract>
  </authors>
  <authors>
    <au_id>238-95-7766</au_id>
    <au_lname>Carson</au_lname>
    <au_fname>Cheryl</au_fname>
    <phone>415 548-7723</phone>
    <address>589 Darwin Ln.</address>
    <city>Berkeley</city>
    <state>CA</state>
    <zip>94705</zip>
    <contract>true</contract>
  </authors>
  <authors>
    <au_id>267-41-2394</au_id>
    <au_lname>Hunter</au_lname>
```

```
    <au_fname>Anne</au_fname>
    <phone>408 286-2428</phone>
    <address>22 Cleveland Av. #14</address>
    <city>San Jose</city>
    <state>CA</state>
    <zip>95128</zip>
    <contract>true</contract>
  </authors>
  <authors>
    <au_id>274-80-9391</au_id>
    <au_lname>Straight</au_lname>
    <au_fname>Dean</au_fname>
    <phone>415 834-2919</phone>
    <address>5420 College Av.</address>
    <city>Oakland</city>
    <state>CA</state>
    <zip>94609</zip>
    <contract>true</contract>
  </authors>
</Authors_Table>
```

4. В меню **Файл** выберите **Сохранить authors.xml**.

Пример № 2

Создание пользовательского интерфейса

Ниже перечислены элементы, образующие пользовательский интерфейс создаваемого приложения:

- Элемент управления DataGridView, отображающий содержимое файла XML в виде данных.
- Элемент управления TextBox, отображающий схему XML файла XML.
- Два элемента управления Button.
  - Первая кнопка инициирует считывание файла XML в набор данных и отображение его в элементе управления DataGridView.
  - Нажатием второй кнопки извлекается схема из набора данных, и через StringWriter она отображается в элементе управления TextBox.

Для добавления элементов управления в форму:

1. Откройте Form1 в Конструкторе.
2. Из **Панели элементов** перетащите на форму следующие элементы управления:
   - Один элемент управления DataGridView
   - Один элемент управления TextBox
   - Два элемента управления Button
3. Задайте следующие свойства:

Элемент управления	Свойство.	Параметр
TextBox1	**Multiline**	true
	ScrollBars	**Вертикальный**
Button1	**Имя (name)**	ReadXmlButton
	Текст (text)	Прочитать XML
Button2	**Имя (name)**	ShowSchemaButton
	Текст (text)	Показать схему

Пример № 3

Создание набора данных, получающего данные XML

В следующей процедуре будет создан новый набор данных с именем authors. За дополнительными сведениями о наборах данных обратитесь к разделу Работа с наборами данных в Visual Studio.

Для создания нового набора данных, который будет получать XML-данные:

1. Выбрав исходный файл для **Form1** в **обозревателе решений**, нажмите кнопку **Открыть в конструкторе** в панели инструментов **обозревателя решений**.
2. Перетащите **Набор данных (DataSet)** из Вкладка "Данные", панель элементов на **Form1**.
3. Выберите **Нетипизированный набор данных** на **Добавление набора данных** диалоговое окно " и нажмите кнопку**ОК**.

Dataset1 добавляется в область компонентов.

4. В окне **Свойства** задайте свойствам **Имя** и DataSetName значение AuthorsDataSet.

Пример № 4

Создание обработчика событий, считывающего XML в набор данных

Кнопка **Прочитать XML** инициирует считывание файла XML в набор данных и задает свойства элемента управления DataGridView, определяющие его привязку к набору данных.

Добавление кода в обработчик событий ReadXmlButton_Click

1. В **обозревателе решений** выберите **Form1** и нажмите кнопку **Открыть в конструкторе** в панели инструментов **обозревателя решений**.
2. Дважды щелкните кнопку **Прочитать XML**.

Редактор кода откроется на обработчике событий ReadXmlButton_Click.

3. Введите следующий код в обработчик событий ReadXmlButton_Click:

C#

```
private void ReadXmlButton_Click(object sender, EventArgs e)
{
   string filePath = "Complete path where you saved the XML file";

   AuthorsDataSet.ReadXml(filePath);

   dataGridView1.DataSource = AuthorsDataSet;
   dataGridView1.DataMember = "authors";
}
```

4. В коде обработчика событий ReadXMLButton_Click измените параметр filepath = на правильный путь.

Пример № 5

Создание обработчика событий, отображающего схему XML в текстовом поле

Кнопка **Показать схему** создает объект StringWriter, который заполняется схемой и отображается в TextBox.

Добавление кода в обработчик событий ShowSchemaButton_Click.

1. В **обозревателе решений** выберите **Form1** и нажмите кнопку **Открыть в конструкторе**.
2. Дважды щелкните кнопку **Показать схему**.

Редактор кода откроется на обработчике событий ShowSchemaButton_Click.

3. Введите следующий код в обработчик событий ShowSchemaButton_Click:

C#

```
private void ShowSchemaButton_Click(object sender, EventArgs e)
{
   System.IO.StringWriter swXML = new System.IO.StringWriter();
   AuthorsDataSet.WriteXmlSchema(swXML);
   textBox1.Text = swXML.ToString();
}
```

Проверка

Теперь можно проверить форму, чтобы убедиться, что она работает так, как ожидалось. Чтобы проверить форму, выполните следующие действия:

1. Нажмите клавишу F5 для запуска приложения.
2. Нажмите кнопку **Прочитать XML**.

Сетка данных DataGridView заполнится содержимым файла XML.

3. Нажмите кнопку **Показать схему**.

В текстовом поле отобразится схема XML для файла XML.

ЛАБОРАТОРНАЯ РАБОТА № 3
ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ VISUAL STUDIO. ЖИЗНЕННЫЙ ЦИКЛ РАЗРАБОТКИ ПРИЛОЖЕНИЯ В VISUAL STUDIO. МОДЕЛИРОВАНИЕ СТРУКТУРЫ ПРОГРАММЫ НА ЯЗЫКЕ UML.

Цель: освоение инструментария моделирования, проектирования, реализации, рефакторинга.

Задание на лабораторную работу:

1. Создать диаграмму классов согласно пункта 3 « **Пример выполнения лабораторной работы» (Создание диаграммы классов для сценария "Добавить новый заказ" прецедента "Работа с заказом").** Для каждого класса необходимо задать атрибуты и операции. Каждый класс должен быть подробно задокументирован - необходимо задать текстовое описание самого класса, описания его атрибутов и операций.
2. Создать пакеты для группировки классов.
3. Сгруппировать классы в пакеты.
4. Проекты сохраняете на своем флэш-накопителе для работы в следующих лабораторных работах.

Содержание отчета

1. Созданные диаграммы классов в виде скринштов.

Для создания *UML*-моделей используется специальный вид проекта **Modeling Project**. (Проекты моделирования). В среде Visual Studio выберем в главном *меню* **File / NewProject и вид проекта Modeling Project (Проекты моделирования)**

Создание новой диаграммы классов в MS Visual Studio

1. В меню Архитектура выберите пункт Создать схему.
2. В диалоговом окне Добавление новой схемы выберите требуемый тип схемы моделирования.

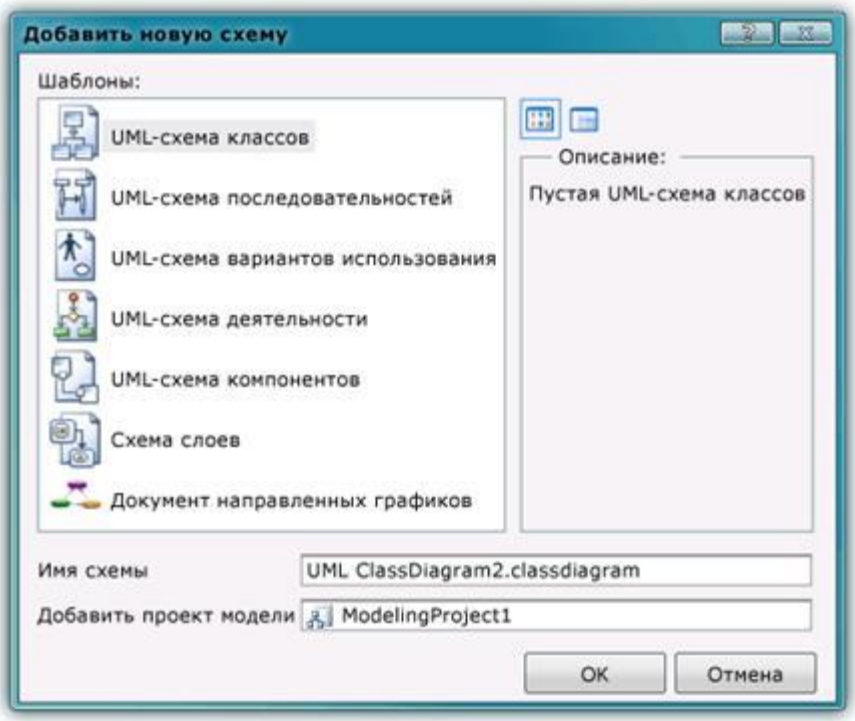


Рис. 2.3 Выбор UML-схемы в Visual Studio

3. Введите имя новой схемы.

4. В окне Добавить в проект моделирования выполните следующее.

5. Выберите проект моделирования, который уже существует в решении, и нажмите кнопку ОК.

6. Выберите Создать новый проект моделирования и нажмите кнопку ОК.

7. В диалоговом окне Создание нового проекта моделирования введите имя и расположение нового проекта, затем нажмите кнопку Создать.

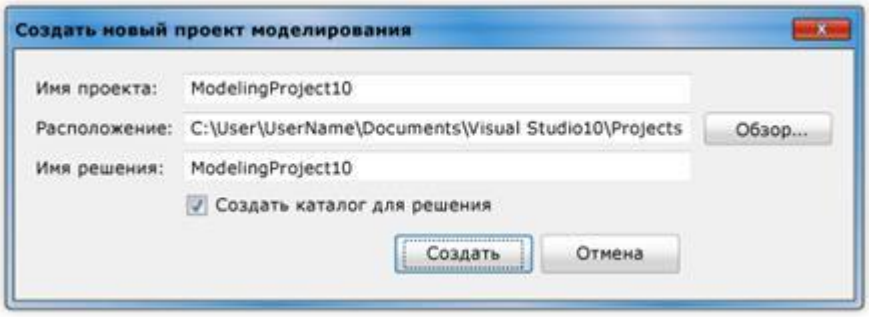


Рис. 2.4 Создание проекта моделирования в Visual Studio

Создание нового класса

Для создания нового класса нужно щелкнуть по кнопке *Класс* на *Панели элементов* и затем по свободному месту окна диаграммы (рис. 2.5). После создания класса нужно определить его свойства. Для этого нужно вызвать для него контекстное меню и выбрать пункт *Свойства,* после чего откроется меню класса, содержащее ряд вкладок (рис. 2.6).

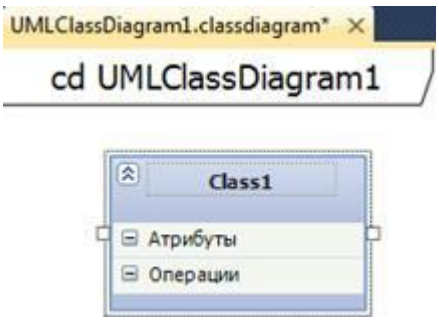


Рис. 2.5 Созданный класс Class1

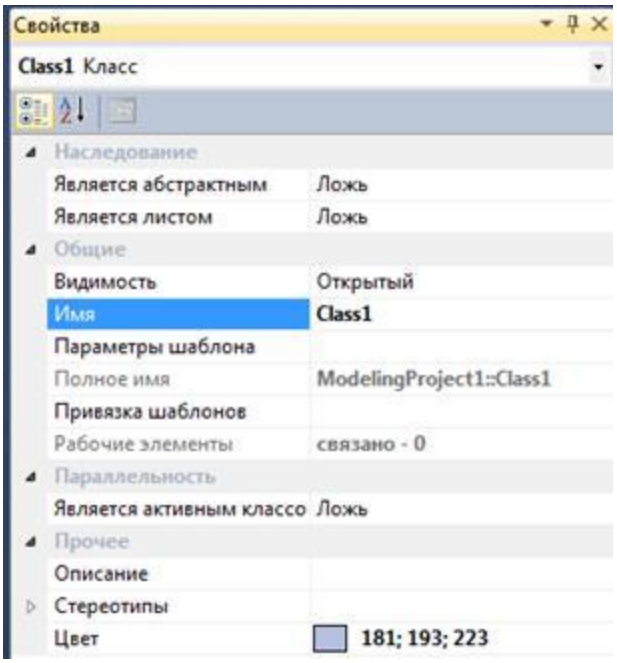


Рис. 2.6 Свойства класса

вызвать контекстное меню для строки *Атрибуты* и выбрать пункт *Добавить,* либо вызвать контекстное меню класса, выбрать пункт *Добавить* (рис. 2.7) и выбрать пункт *Атрибут.*

Добавление новых операций к классу

Для добавления новой операции к классу нужно вызвать контекстное меню для строки *Операции* и выбрать пункт *Добавить,* либо вызвать контекстное меню класса, выбрать пункт *Добавить* (рис. 2.7) и выбрать пункт *Операции.*

Добавление параметров к операции класса

Для добавления параметров к операции класса необходимо открыть свойства операции, выбрать вкладку *Параметры*, в ней по кнопке с тремя точками, после чего откроется окно настройки параметров (рис. 2.8).

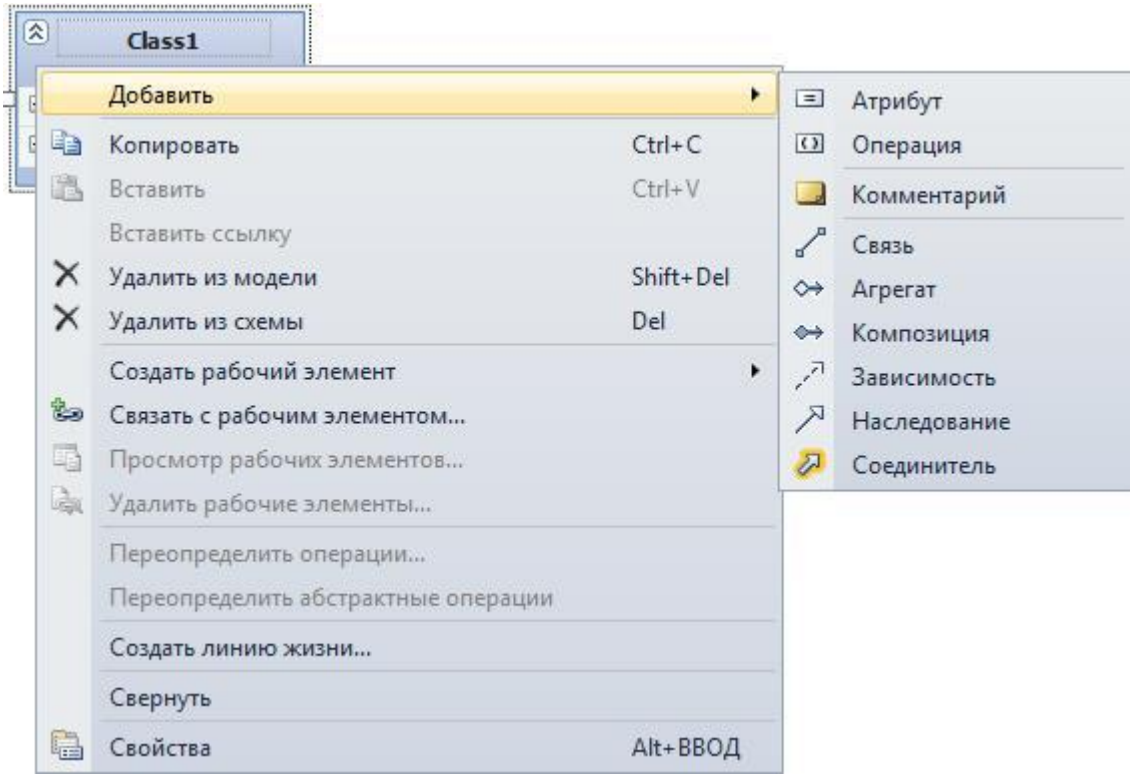


Рис. 2.7 Контекстное меню класса.

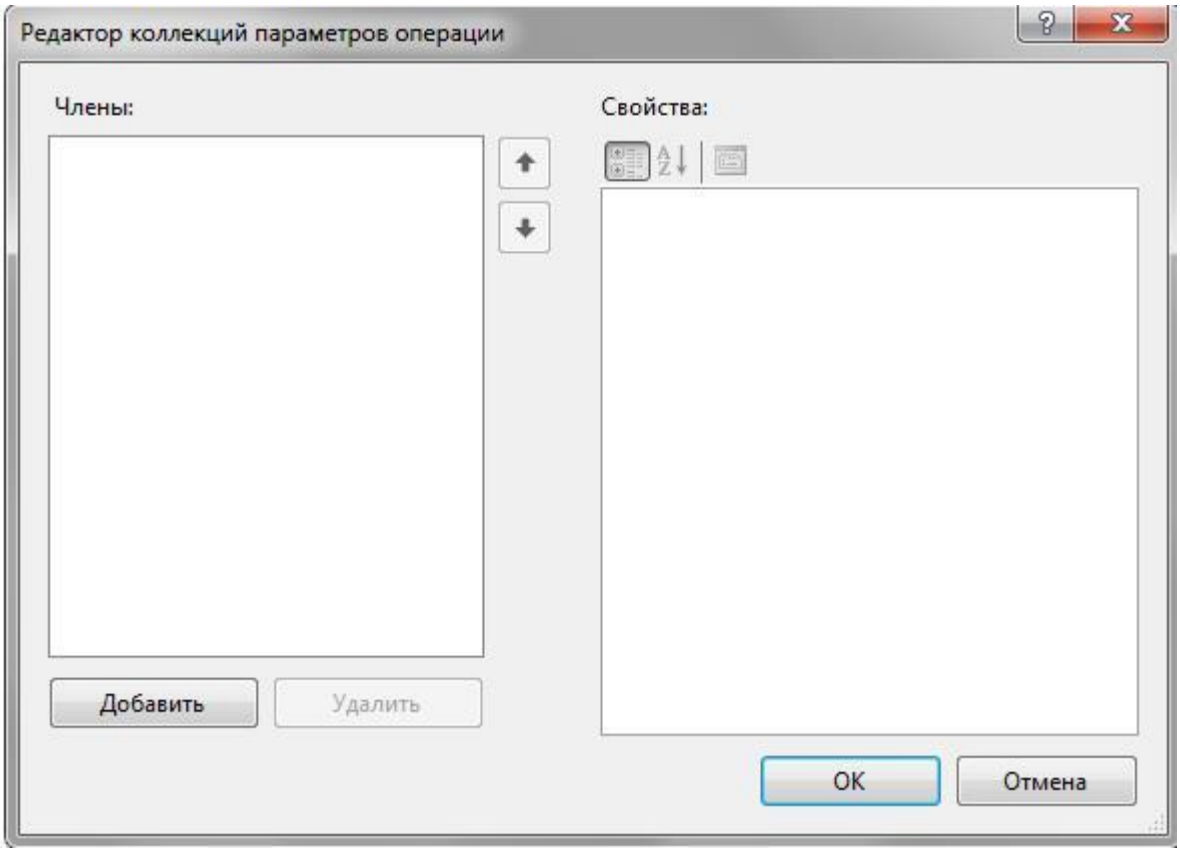


Рис. 2.8 Окно добавления параметров

Создание отношений между классами.

Общее замечание. Для любого типа отношений задание его свойств осуществляется одинаковым способом - вызвать для нее контекстное меню и выбрать пункт *Свойства ...* .

Отношение зависимости.

Является наиболее общей формой отношения в языке UML. Все другие типы отношений можно считать частным случаем данного отношения. Отношение зависимости показывает, что изменение одного класса влечет изменение другого класса. Чаще всего применяется, когда один класс использует другой в качестве аргумента. Изображается пунктирной линией со стрелкой, направленной от зависимого класса к независимому.

Для создания отношения зависимости следует выбрать кнопку *Зависимость* на панели,затем щелкнуть мышкой по зависимому классу и не отпуская кнопки мыши перетащить стрелку на независимый класс.

Отношение ассоциации, агрегации и композиции.

Поскольку отношения ассоциации, агрегации и композиции отличаются очень незначительно, то рассмотрим работу с ними в общем.

Отношение ассоциации показывает, что один класс каким-то образом связан с другим классом (аналог связи в диаграмме «Сущность-Связь»). Изображается сплошной линией, соединяющей классы.

Отношение агрегации - частный случай ассоциации. Представляет собой отношение типа «целое/часть». Изображается в виде простой ассоциации с незакрашенным ромбом со стороны «целого». Агрегация не является наследованием, поскольку все классы-«части» в агрегации являются вполне самостоятельными, со своими атрибутами и операциями, отличающимися от атрибутов и операций класса-«целое».

Отношение композиции - частный случай агрегации. Служит для выделения специальной формы отношения «целое-часть», при которой составляющие части в некотором смысле находятся внутри целого. Специфика взаимосвязи между ними заключается в том, что части не могут выступать в отрыве от целого, т.е. с уничтожением целого уничтожаются и все его составные части. Изображается в виде сплошной линии с закрашенным ромбом возле класса «целое».

Для создания одного из приведенных выше отношений следует выбрать соответствующую кнопку на панели,затем щелкнуть мышкой по одному классу и не отпуская кнопки мыши перетащить стрелку на другой класс.

Отношение наследования

Это отношение между двумя элементами модели, при котором один элемент (клиент) реализует поведение, заданное другим (поставщиком). Изображается в виде пунктирной линии с большой незакрашенной стрелкой, указывающей на поставщика. Чаще всего наследование используют для определения отношений между интерфейсом и классом или компонентом, который предоставляет объявленные в интерфейсе операции или услуги.

Для создания отношения наследования следует выбрать кнопку *Наследование* на панели, затем щелкнуть мышкой по объекту-клиенту и не отпуская кнопки мыши перетащить стрелку на объект-поставщик.

3. Пример выполнения лабораторной работы

3.1. Создание диаграммы классов для сценария "Добавить новый заказ" прецедента "Работа с заказом"

Диаграммы классов будем рассматривать с концептуальной точки зрения. Для упрощения задачи и чтобы не загромождать диаграммы несущественными деталями методы setX, getX для каждого атрибута X классов задавать не будем.

Создадим в Логическом представлении браузера новую диаграмму классов и назовем ее "*Add New Order*". В поле документации запишем для нее следующий текст: "*Диаграмма классов для сценария "Добавить новый заказ" прецедента "Работа с заказом"*".

Заполнение диаграммы начнем с определения классов-сущностей. Рассматриваемый сценарий состоит из:

· самого *заказа*;

· *клиента*, который делает заказ;

· *комплектующих изделий*, которые входят в заказ.

Создадим классы-сущности *Order (Заказ)*, *Client (Клиент)* и *ComponentPart (Комплектующее изделие)*. Поскольку в один заказ может входить много разных комплектующих изделий, и одно комплектующее изделие может входить во много заказов, то введем еще один класс-сущность *OrderItem (Состав заказа)*. Опишем каждый класс.

Класс *Client:*

Параметр	Значение
Комментарий	Класс, представляющий собой клиента фирмы
Атрибуты	name : String - наименование клиента address : String - адрес клиента phone : String - телефон клиента Все атрибуты имеют модификатор доступа - private
Операции	AddClient() - добавление нового клиента RemoveClient() - удаление существующего клиента GetInfo() - получить информацию о клиенте Все операции имеют модификатор доступа - public

Класс Order:

Параметр	Значение
Комментарий	Класс, представляющий собой заказ, который делает клиент
Атрибуты	orderNumber : Integer - номер заказа orderDate : Date - дата оформления заказа orderComplete : Date - дата выполнения заказа Все атрибуты имеют модификатор доступа - private
Операции	Create() - создание нового заказа SetInfo() - занести информацию о заказе GetInfo() - получить информацию о заказе Все операции имеют модификатор доступа - public

Класс OrderItem:

Параметр	Значение
Комментарий	Класс, представляющий собой пункт заказа, который делает клиент
Атрибуты	itemNumber : Integer - номер пункта заказа quantity : Integer - количество комплектующих изделий price : Double - цена за единицу Все атрибуты имеют модификатор доступа - private
Операции	Create() - создание новой строки заказа SetInfo() - занести информацию о строке заказа GetInfo() - получить информацию о строке заказа Все операции имеют модификатор доступа - public

Класс ComponentPart:

Параметр	Значение
Комментарий	Класс, представляющий собой комплектующие изделия
Атрибуты	name : String - наименование manufacturer : String - производитель price : Double - цена за единицу description - описание Все атрибуты имеют модификатор доступа - private
Операции	AddComponent() - добавление нового комплектующего изделия RemoveComponent() - удаление комплектующего изделия GetInfo() - получить информацию о комплектующем изделии Все операции имеют модификатор доступа - public

Результат создания классов-сущностей показан на рис. 2.9:

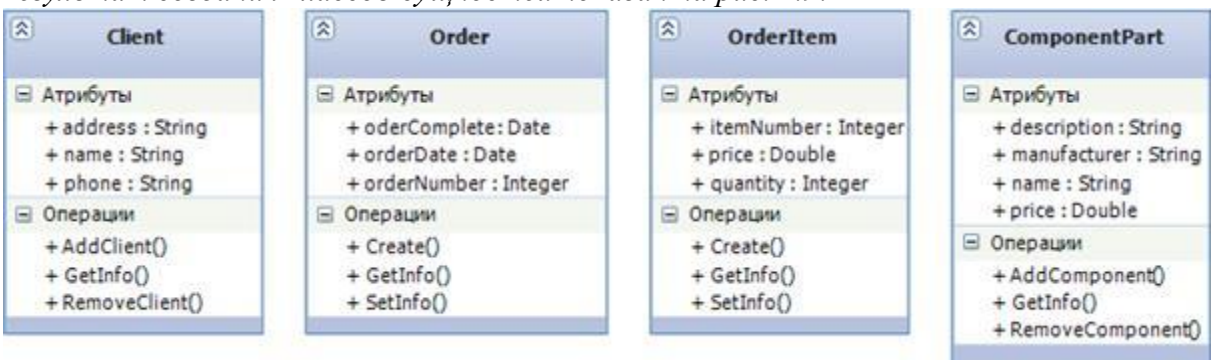


Рис. 2.9 *Созданные классы-сущности*

Добавим отношения между классами (рис. 2.10):

· класс *Client* и *Order* - отношение ассоциации, поскольку данные два класса просто связаны друг с другом и никакие другие типы связей здесь применить нельзя. Один клиент может сделать несколько заказов, каждый заказ поступает только от одного клиента, поэтому кратность связи со стороны класса *Client* - 1, со стороны *Order* - 1..n;

· класс *Order* и *OrderItem* - отношение композиции, поскольку строка заказа является частью заказа, и без него существовать не может. В один заказ может входить несколько строк заказа, строка заказа относится только к одному заказу, поэтому кратность связи со стороны *Order* - 1, со стороны *OrderItem* - 1..n;

· класс *OrderItem* и *ComponentPart* - отношение агрегации, поскольку комплектующие изделия являются частями строки заказа, но и те, и другие, являются самостоятельными классами. Одно комплектующее изделие может входить во много строк заказа, в одну строку заказа входит только одно комплектующее изделие, поэтому кратность связи со стороны *ComponentPart* - 1, со стороны *OrderItem* - 1..n.

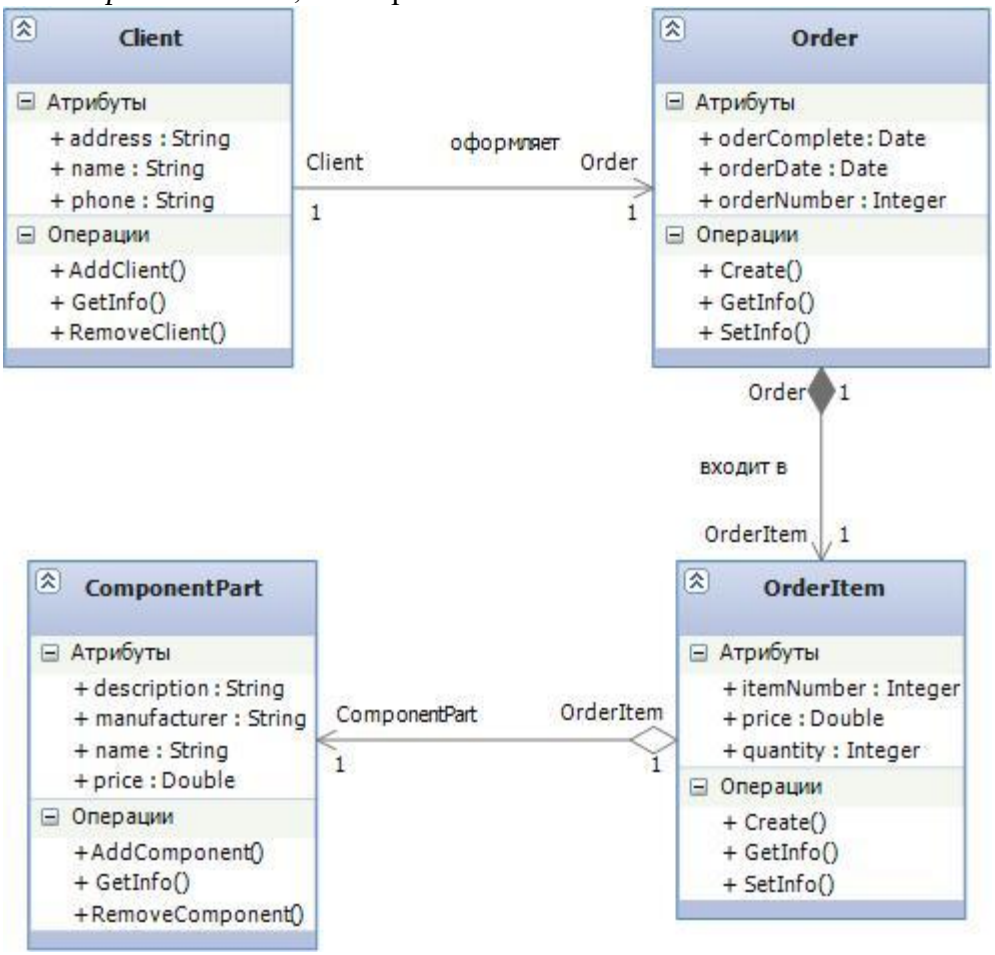


Рис. 2.10 *Классы-сущности и отношения между ними*

Создание пакетов

Пакеты предназначены для группировки элементов в группы по определенным критериям. В простейшем случае классы можно группировать по их стереотипам. Создадим пакет *Entities*(классы-сущности):

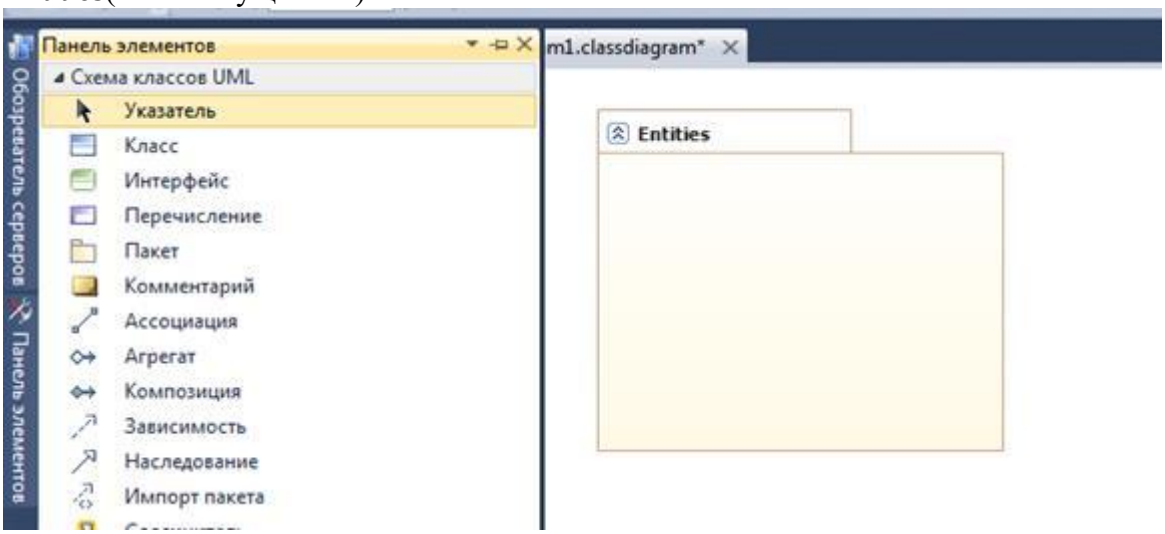


Рис. 2.11 Пакет *Entities*

Группировка классов в пакеты

Группировка классов в пакеты осуществляется путем перетаскивания в Логическом представлении браузера соответствующего класса в соответствующий пакет:

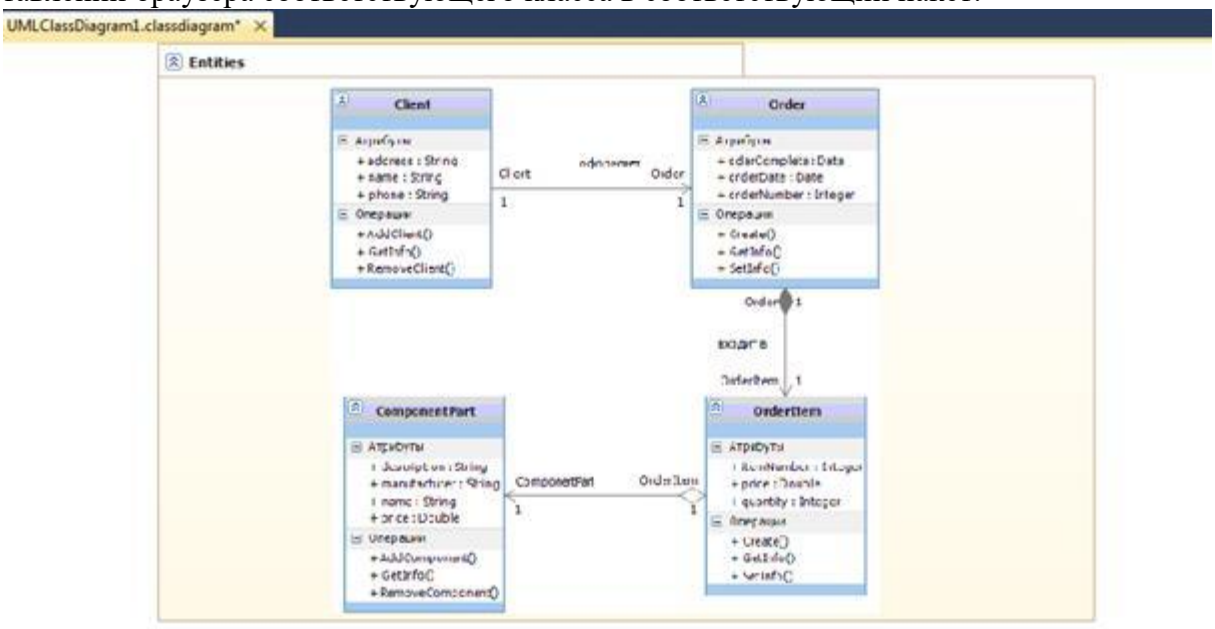


Рис. 2.13 *Классы и пакеты для сценария "Добавление нового заказа"*

ЛАБОРАТОРНАЯ РАБОТА № 4
ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ VISUAL STUDIO. ЖИЗНЕННЫЙ ЦИКЛ РАЗРАБОТКИ ПРИЛОЖЕНИЯ В VISUAL STUDIO. МОДЕЛИРОВАНИЕ СТРУКТУРЫ ПРОГРАММЫ НА ЯЗЫКЕ UML.

Цель: освоение инструментария моделирования, проектирования, реализации, рефакторинга.

Задание на лабораторную работу:

1. Изучить лекционный материал, раздел 1 (теоретическая часть) данной лабораторной работы.
2. Выполнить **ЗАДАНИЕ НА СОЗДАНИЕ ДИАГРАММЫ КЛАССОВ и ПРОВЕДЕНИЕ РЕФАКТОРИНГА (РАЗДЕЛ 2.)**
3. Проекты сохраняете на своем флэш-накопителе для работы в следующих лабораторных работах.

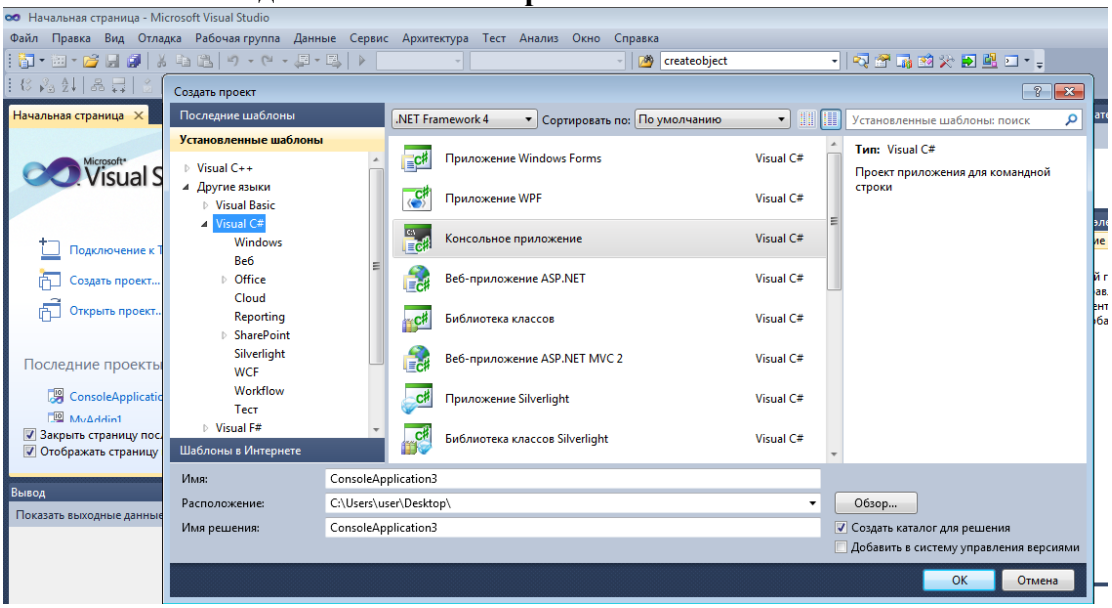
Содержание отчета

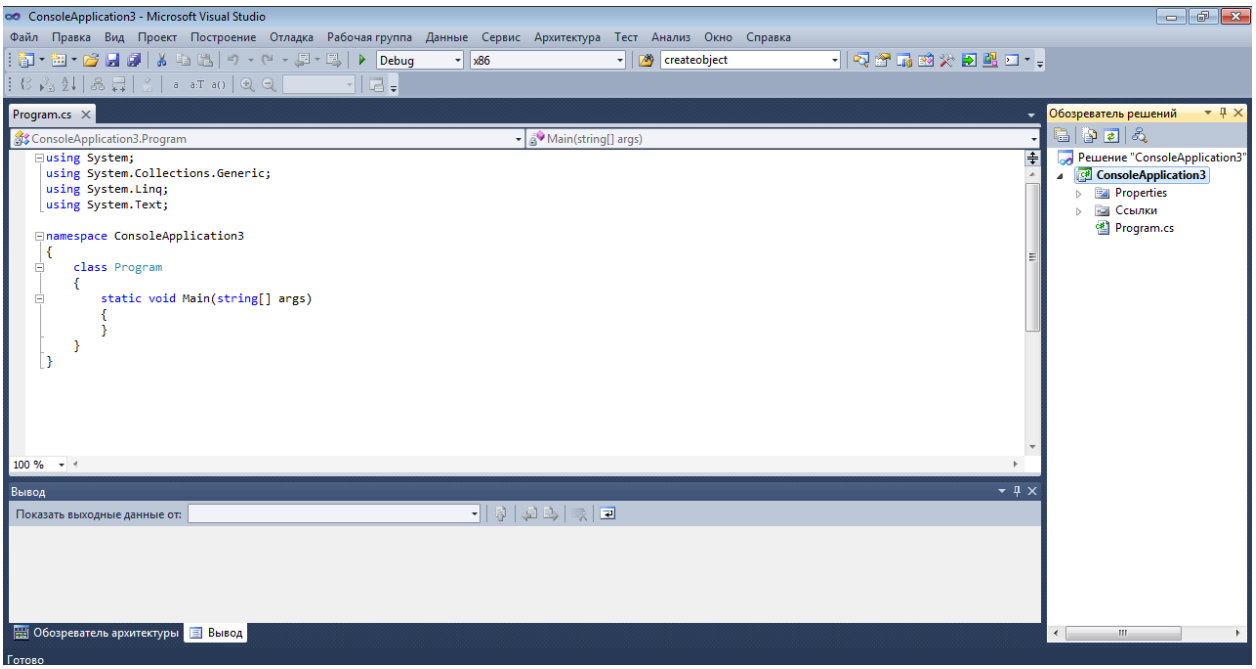
1. Созданные диаграммы классов в виде скринштов.
2. Листинги полученных кодов на языке C# с комментариями.

Раздел 1. Создание новой диаграммы классов в MS Visual Studio и рефакторинг кода.

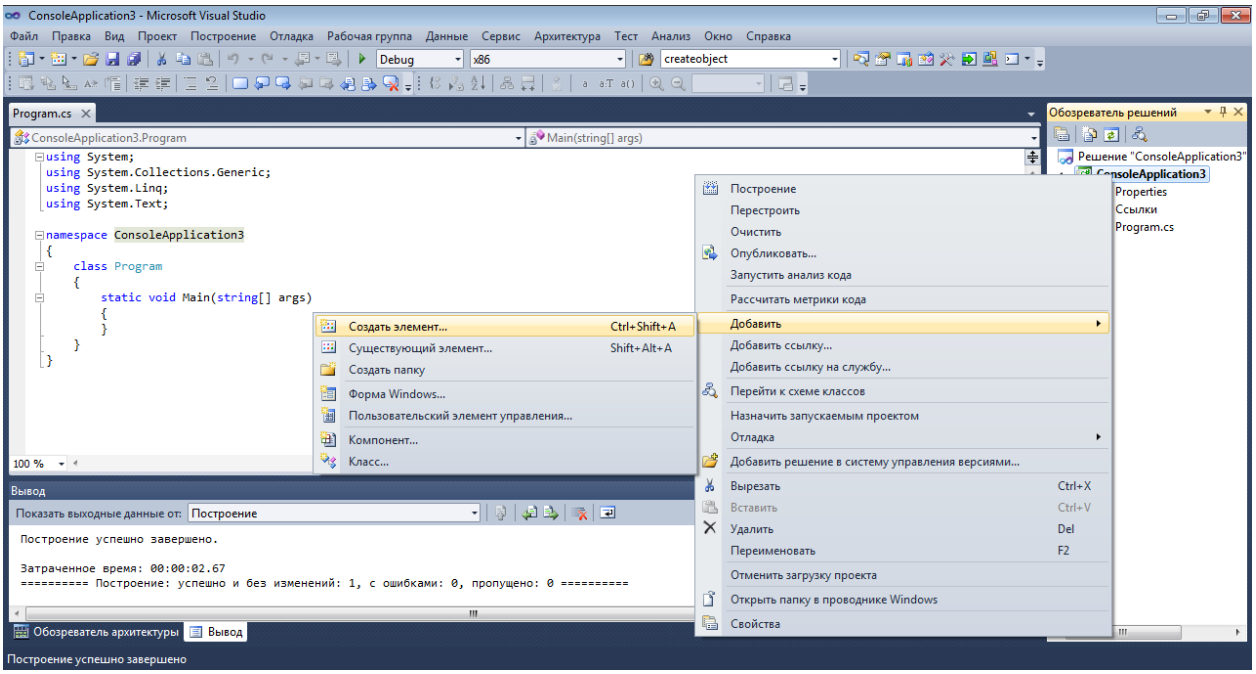
(теоретическая часть)

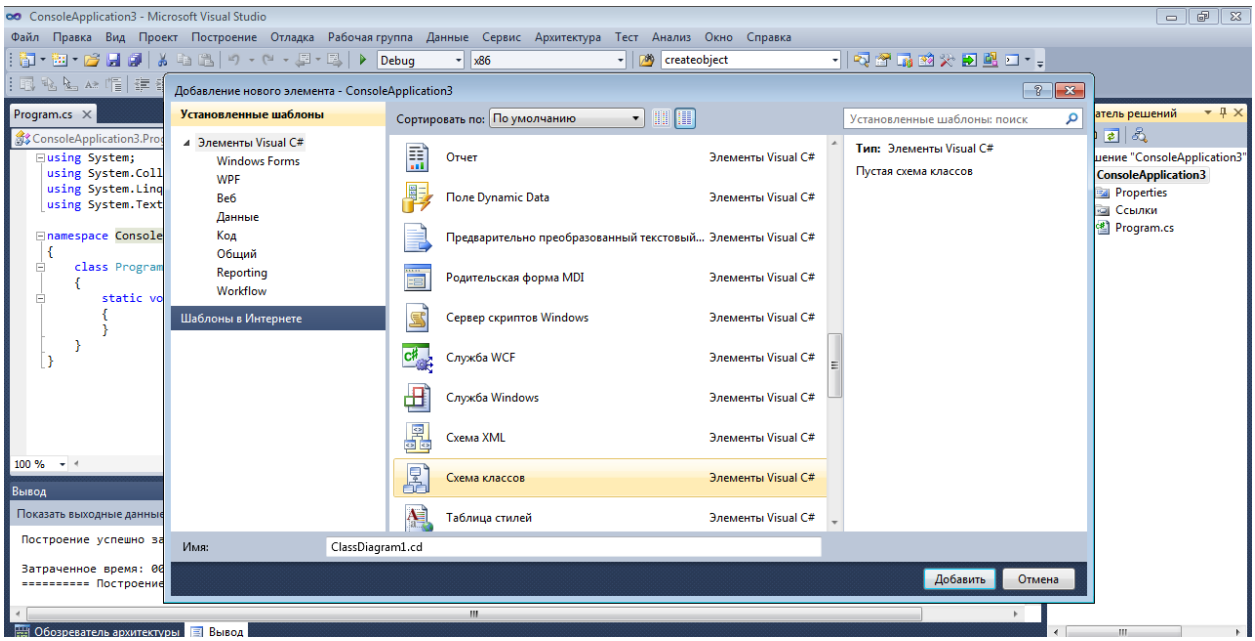
1. Создать консольное приложение

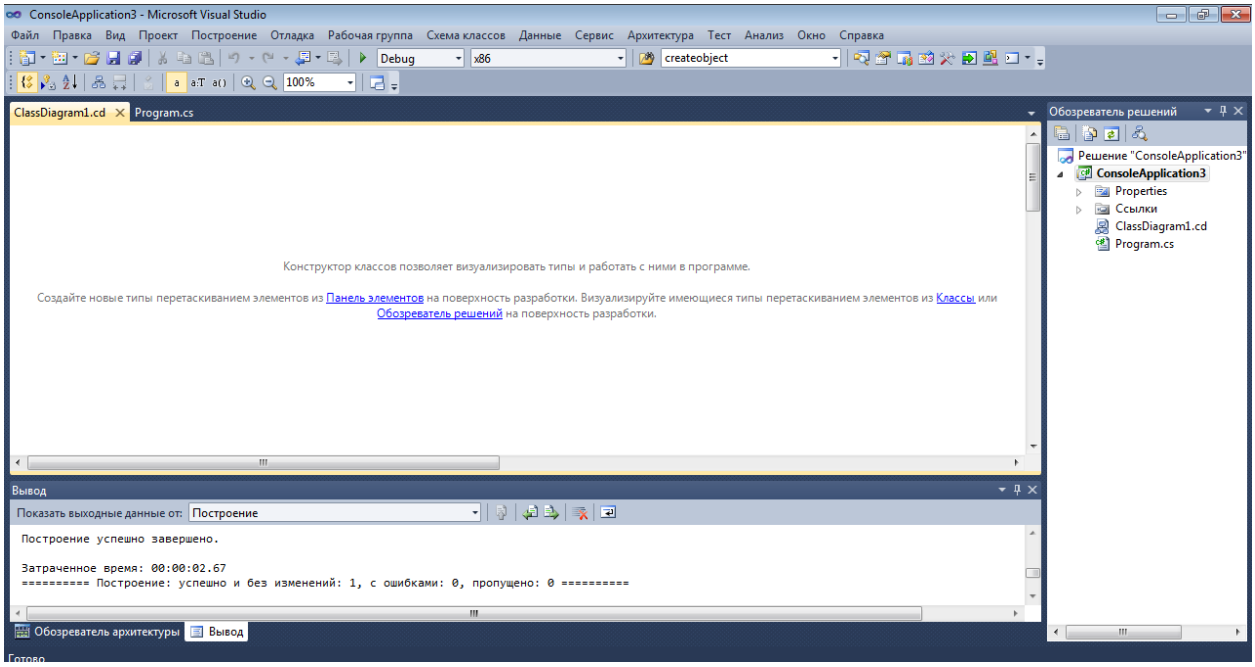




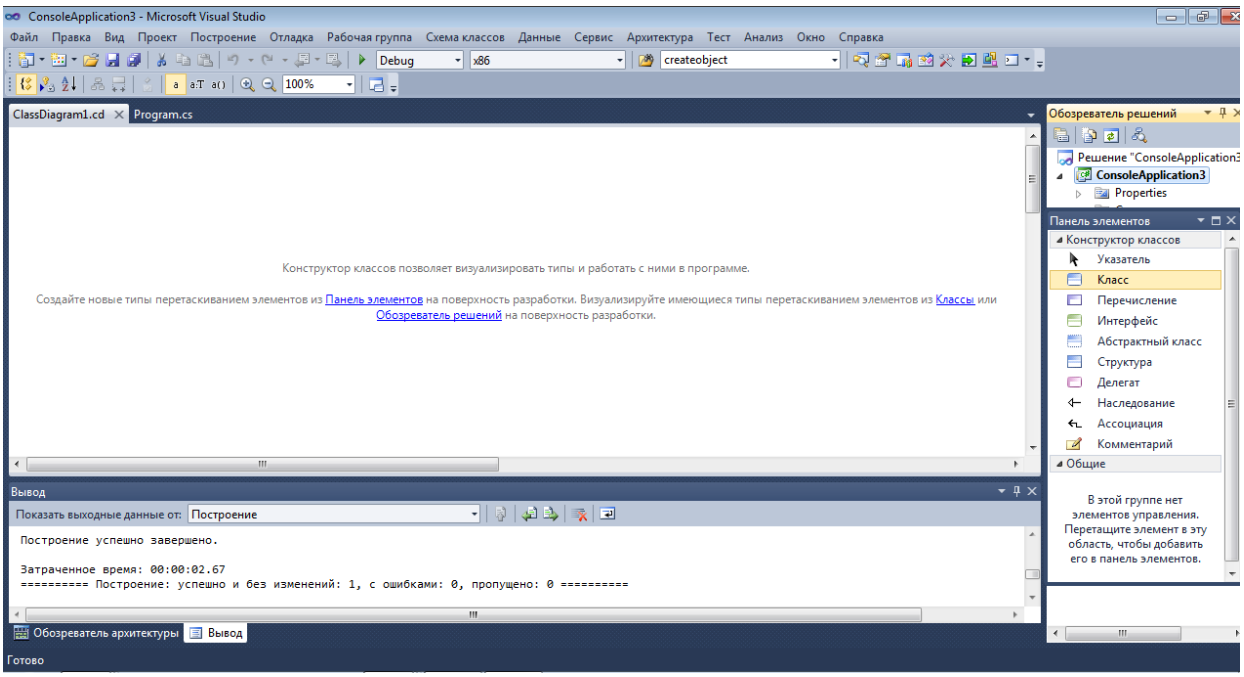
2. Добавить в проект «диаграмму классов».



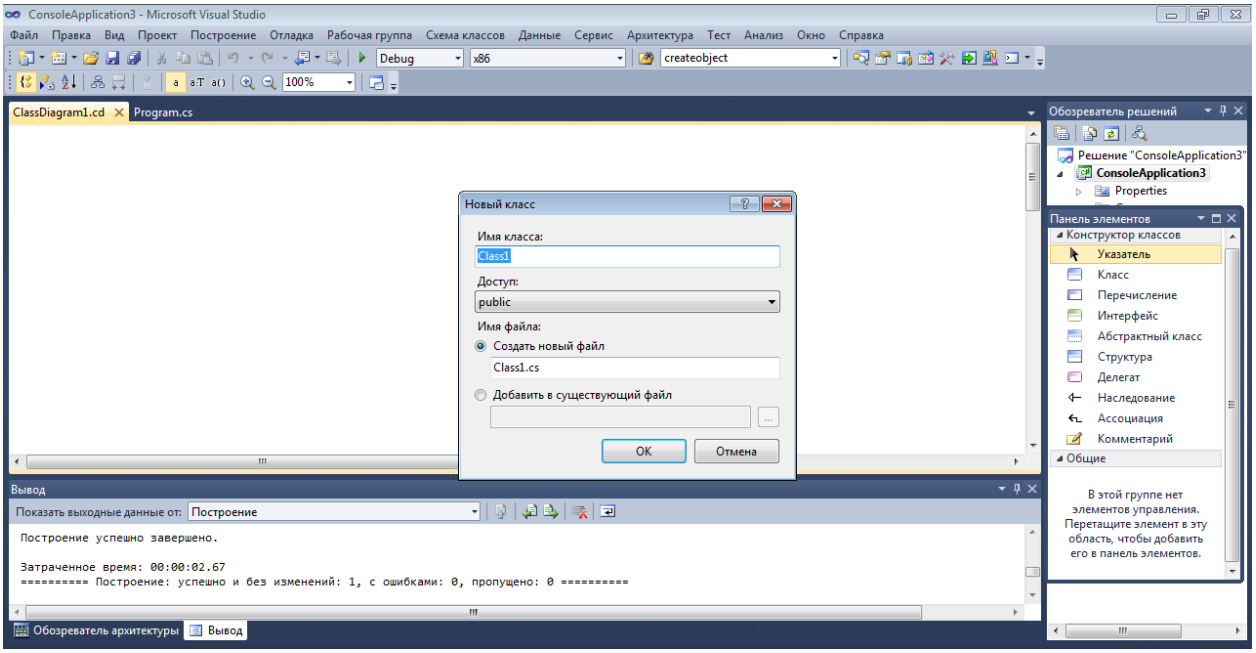




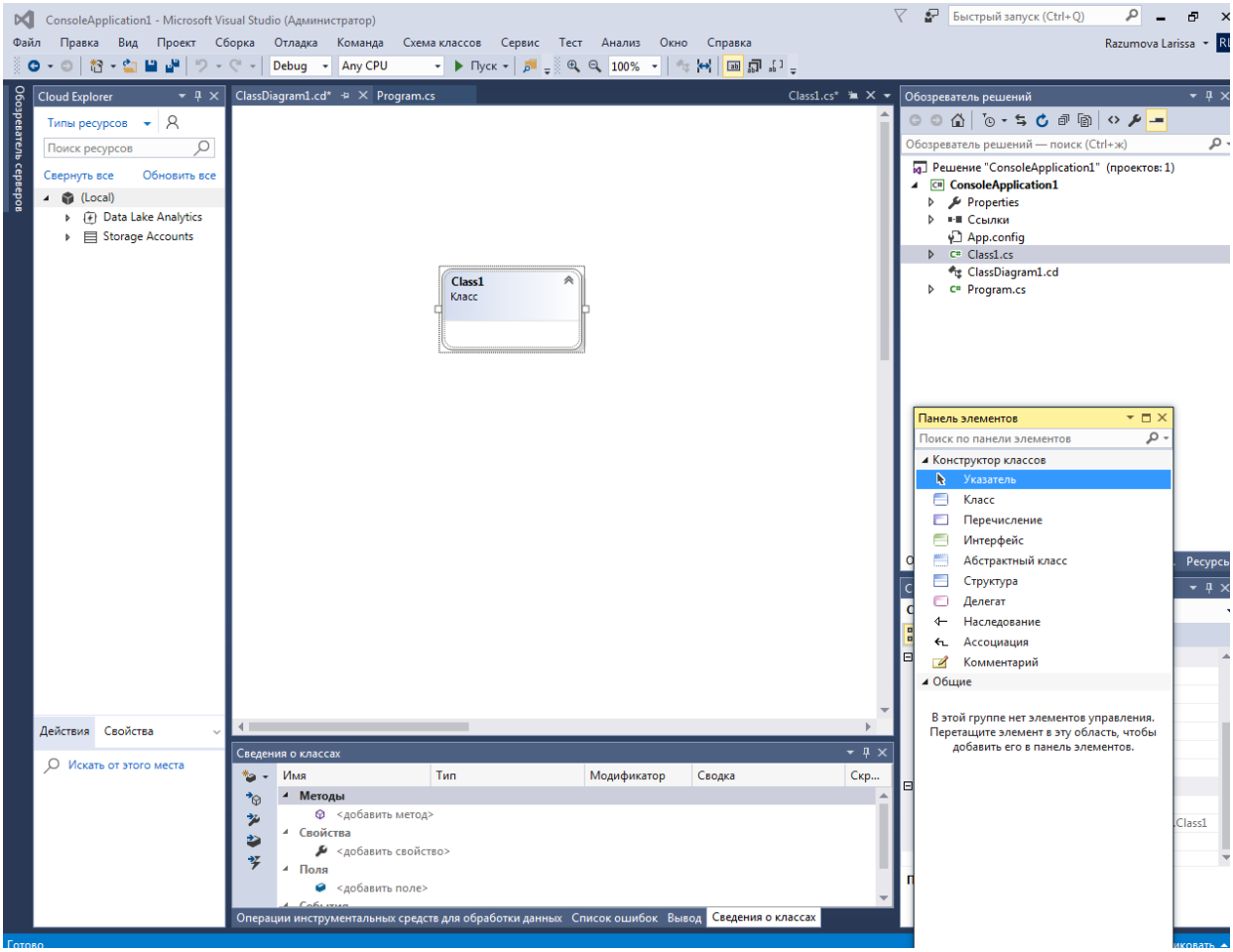
3. Построение диаграммы выполняется с помощью инструментов «Панель элементов»



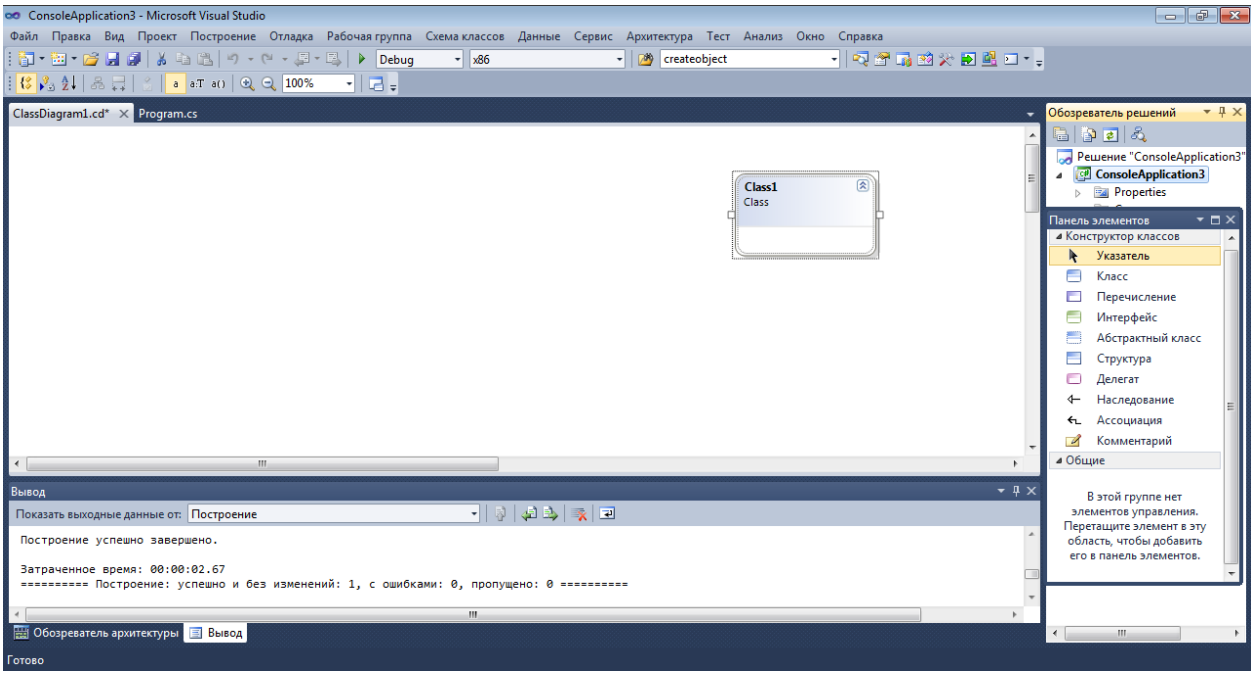
4.Для создания класса перетащите инструмент «Класс» на поле диаграммы классов. Выводится диалоговое окно для указания имени класса, области видимости и названия файла для создаваемого класса.



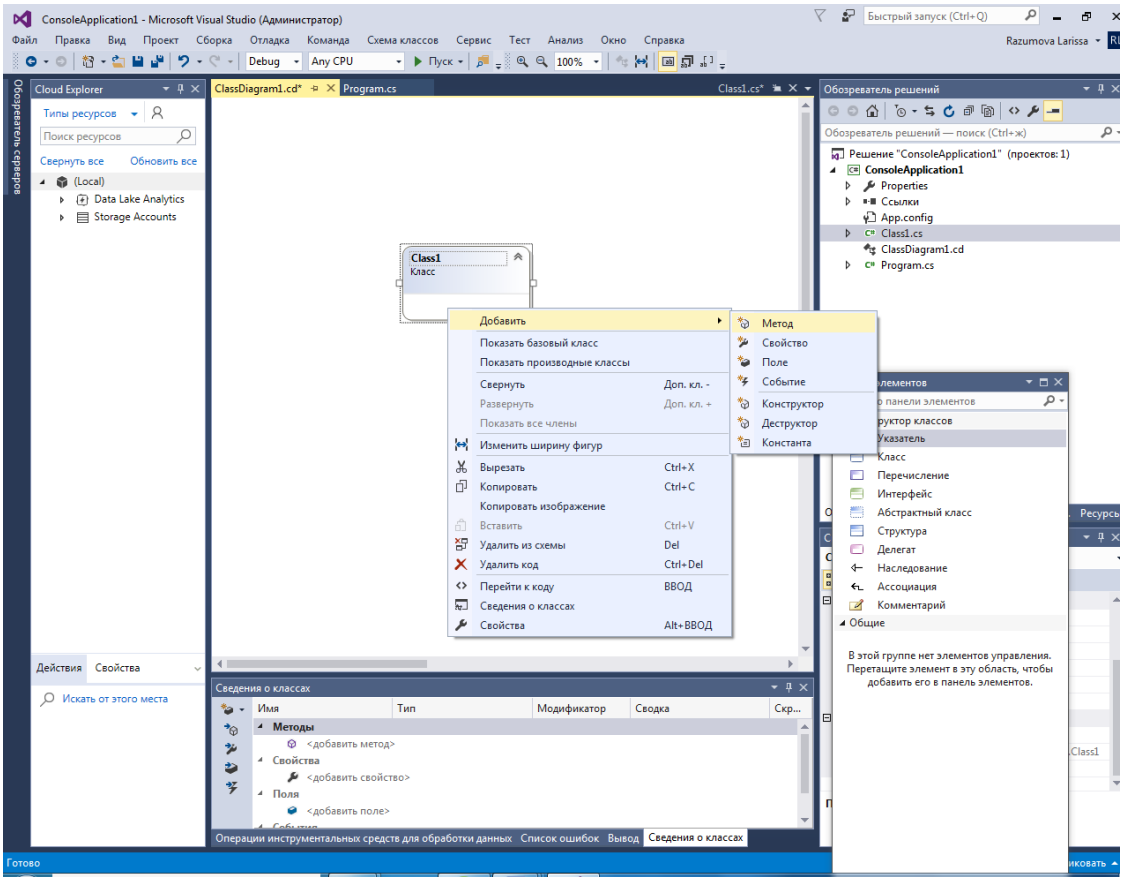
Созданный класс на диаграмме классов отображается в виде прямоугольника, при этом параметры класса указываются в окне «Class Details» (Сведения о классах).

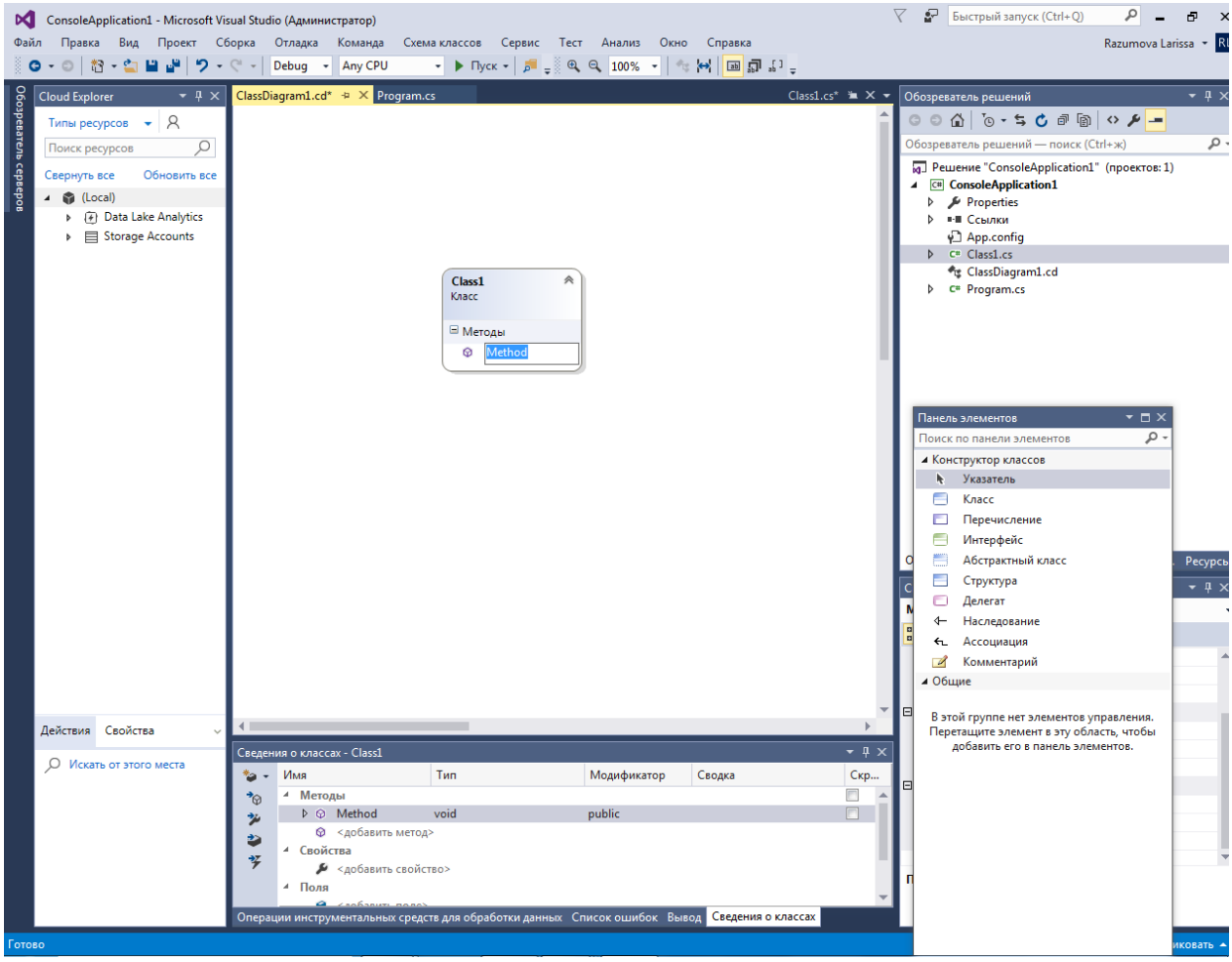


Или

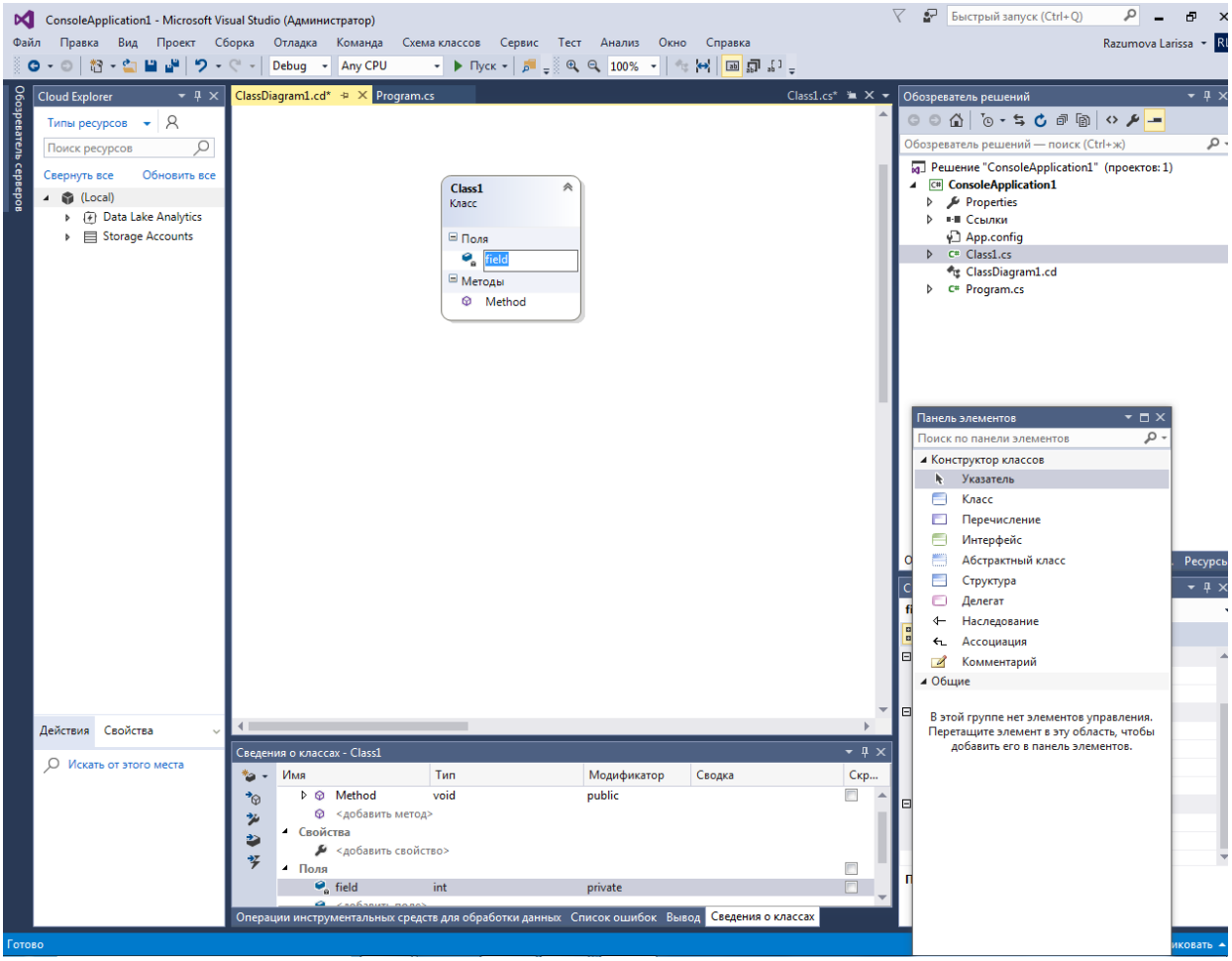


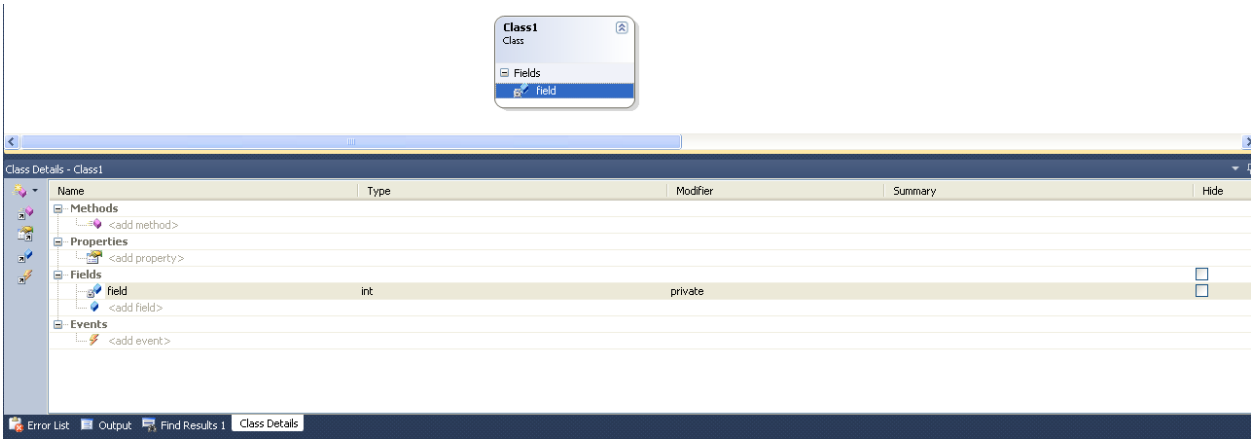
Для добавления полей и методов (в том числе конструктора и деструктора) в класс используется или окно «Class Details» (Сведения о классах) или контекстное меню, после нажатия правой кнопки мыши на прямоугольнике, обозначающего созданный класс.


ConsoleApplication1 - Microsoft Visual Studio (Администратор)
Файл Правка Вид Проект Сборка Отладка Команда Схема классов Сервис Тест Анализ Окно Справка
Debug
Any CPU
Пуск
Cloud Explorer
Типы ресурсов
Поиск ресурсов
Свернуть все
Обновить все
(Local)
Data Lake Analytics
Storage Accounts
ClassDiagram1.cd*
Program.cs
Class1.cs*
Class1
Класс
Добавить
Показать базовый класс
Показать производные классы
Свернуть
Развернуть
Показать все члены
Изменить ширину фигур
Вырезать
Копировать
Копировать изображение
Вставить
Удалить из схемы
Удалить код
Перейти к коду
Сведения о классах
Свойства
Метод
Свойство
Поле
Событие
Конструктор
Деструктор
Константа
Обозреватель решений
Решение "ConsoleApplication1" (проектов: 1)
ConsoleApplication1
Properties
Ссылки
App.config
Class1.cs
ClassDiagram1.cd
Program.cs
Указатель
Класс
Перечисление
Интерфейс
Абстрактный класс
Структура
Делегат
Наследование
Ассоциация
Комментарий
Общие
В этой группе нет элементов управления. Перетащите элемент в эту область, чтобы добавить его в панель элементов.
Действия
Свойства
Искать от этого места
Сведения о классах
Имя
Тип
Модификатор
Сводка
Методы
<добавить метод>
Свойства
<добавить свойство>
Поля
<добавить поле>
Операции инструментальных средств для обработки данных
Список ошибок
Вывод
Сведения о классах
Готово



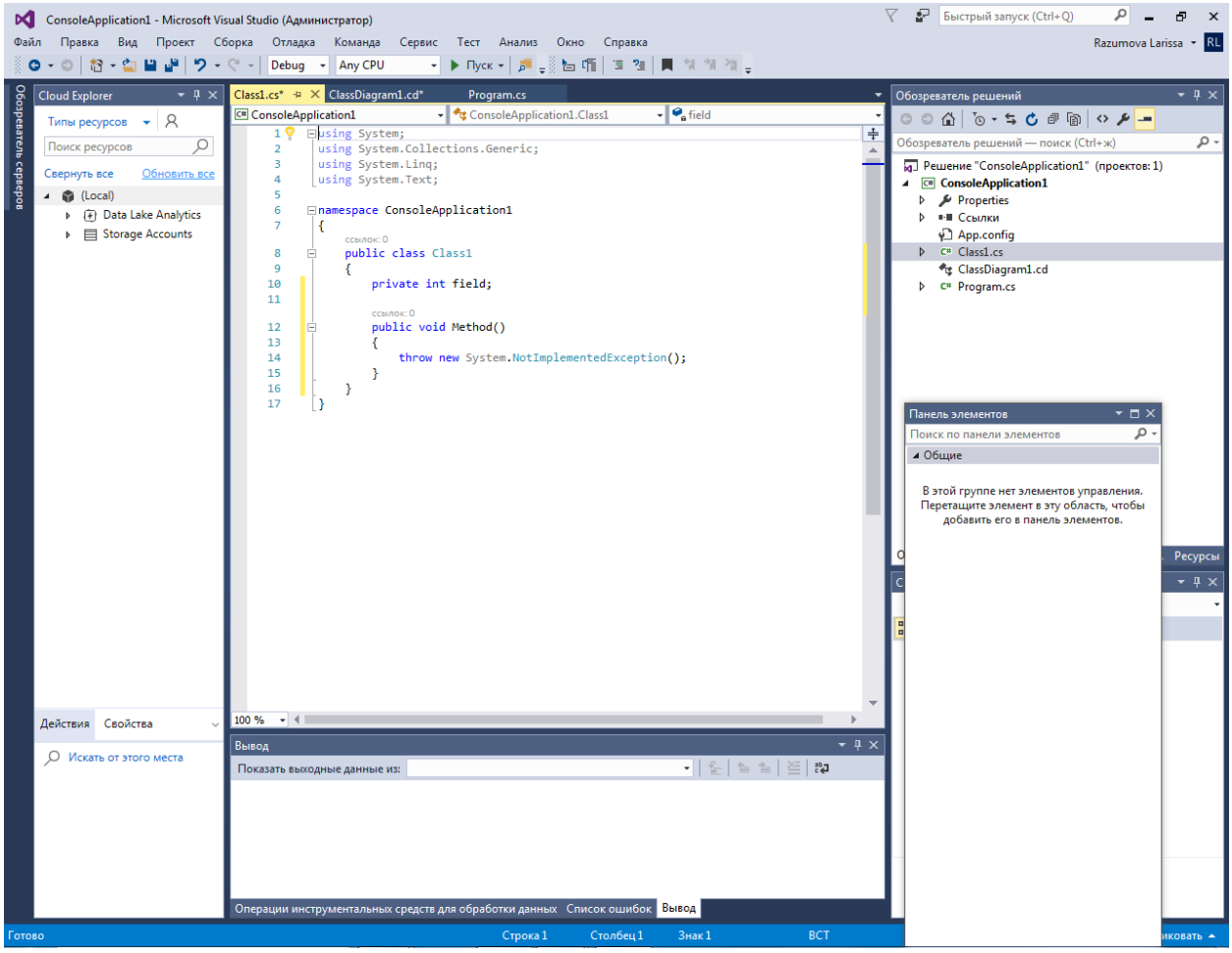
При добавлении метода указывается тип возвращаемого значения и набор передаваемых параметров.





При добавлении поля указывается имя поля, область видимости и тип.

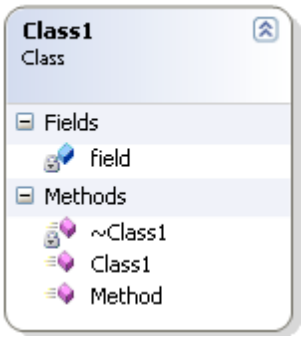
Добавленные в класс поля и методы прописываются в виде кода также и в файле, связанном с данным классом.


ConsoleApplication1 - Microsoft Visual Studio (Администратор)
Файл Правка Вид Проект Сборка Отладка Команда Сервис Тест Анализ Окно Справка
Debug
Any CPU
Пуск
Cloud Explorer
Class1.cs*
ClassDiagram1.cd*
Program.cs
ConsoleApplication1
ConsoleApplication1.Class1
field
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
public class Class1
{
private int field;
public void Method()
{
throw new System.NotImplementedException();
}
}
}
Обозреватель решений
Решение "ConsoleApplication1" (проектов: 1)
ConsoleApplication1
Properties
Ссылки
App.config
Class1.cs
ClassDiagram1.cd
Program.cs
Панель элементов
Общие
В этой группе нет элементов управления. Перетащите элемент в эту область, чтобы добавить его в панель элементов.
Вывод
Показать выходные данные из:
Операции инструментальных средств для обработки данных
Список ошибок
Вывод
Готово
Строка 1
Столбец 1
Знак 1
ВСТ

Class1.cs* ClassDiagram1.cd* Program.cs

ConsoleApplication4.Class1

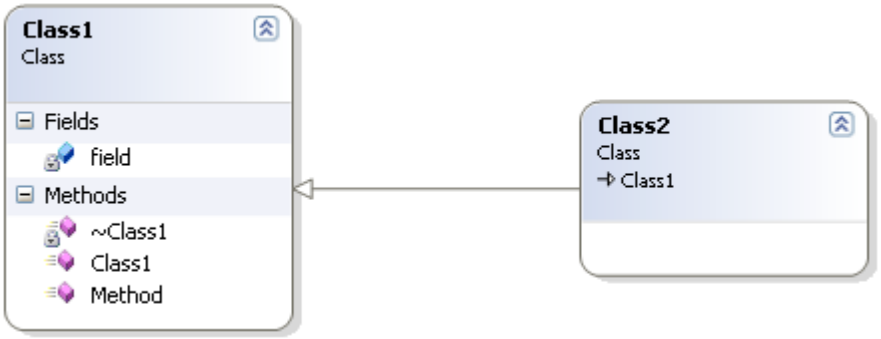
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication4
{
    public class Class1
    {
        private float field;

        public Class1()
        {
            throw new System.NotImplementedException();
        }

        ~Class1()
        {
            throw new System.NotImplementedException();
        }

        public int Method(int p1)
        {
            throw new System.NotImplementedException();
        }
    }
}
```



5. Взаимосвязь между классами

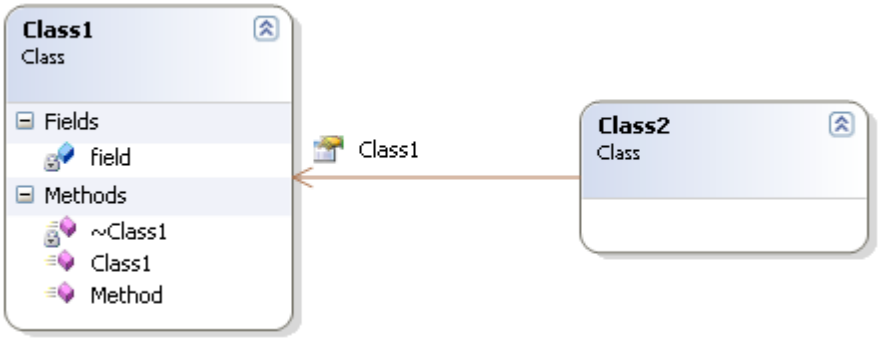
Поддерживается два вида взаимосвязей между классами Inheritance (наследование) и Association (ассоциация).

Связь Inheritance (наследование)



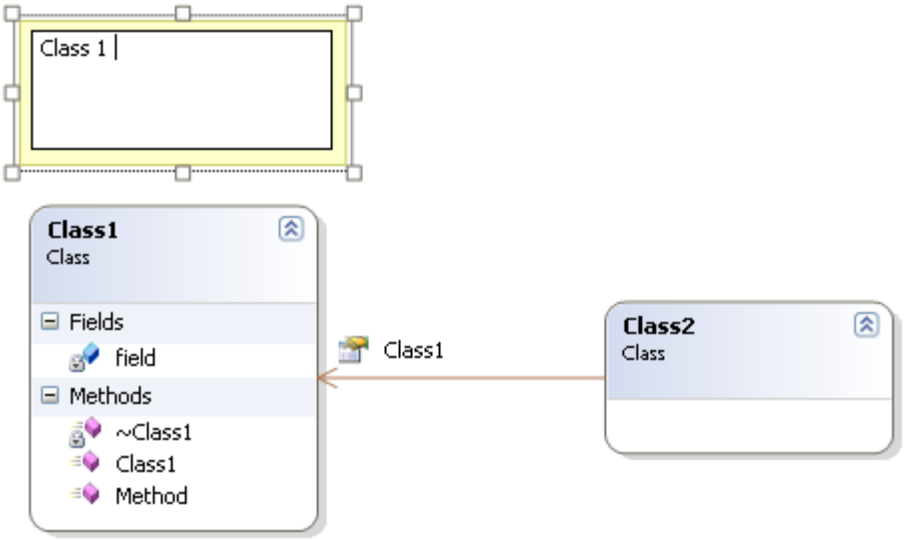

```
ConsoleApplication4.Class2
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication4
{
    public class Class2 : Class1
    {
    }
}
```

Связь Association (ассоциация)



```
namespace ConsoleApplication4
{
    public class Class2
    {
        public Class1 Class1
        {
            get
            {
                throw new System.NotImplementedException();
            }
            set
            {
            }
        }
    }
}
```

Создание примечания (комментария)



6.Рефакторинг кода

Особое *место* при разработке и модификации кода занимает **рефакторинг** - систематическая модификация и *улучшение* существующего кода, без коренного изменения его семантики, с помощью автоматических преобразований, осуществляемых средой.

Типичный пример - **изменение имени метода**. Задача состоит в том, чтобы изменить имя метода и в его определении, и во всех его использованиях. В общем случае, если проект достаточно велик, вручную выполнять решение подобной задачи весьма неудобно. Почти наверняка при этом какие-либо использования метода будут забыты, и к ним придется возвращаться уже после получения ошибок при компиляции (сборке) проекта.

Рассмотрим рефакторинг в среде на примере простого консольного приложения. (рис. 6.1.)

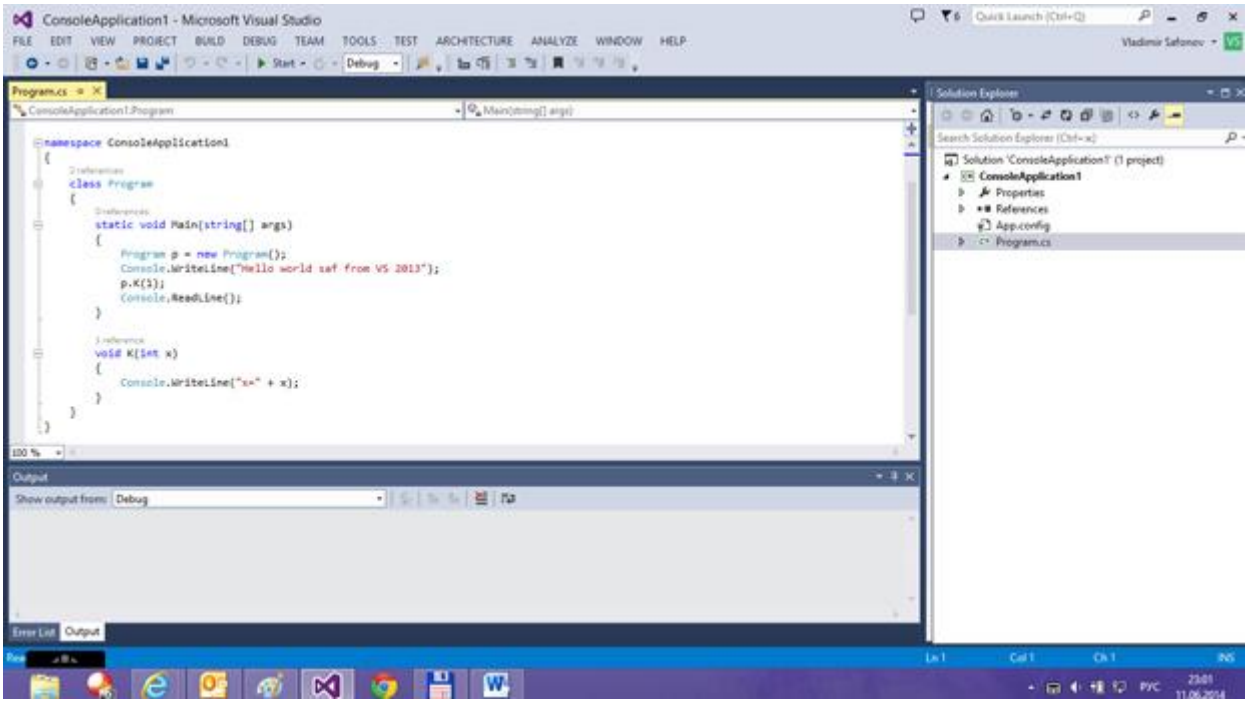


Рис.6.1.Консольное приложение

В программе определен *класс* Program, *статический метод* Main и *метод экземпляра* K с аргументом X.

Пусть необходимо изменить имя метода K на M (естественно, не только в определении, но и во всех его использованиях).

Текущую позицию в редакторе текста устанавливаем на первую строчку определения метода K (заголовок) и выбираем в контекстном *меню пункт* **Refactor** (рефакторинг)(рис.6.2)

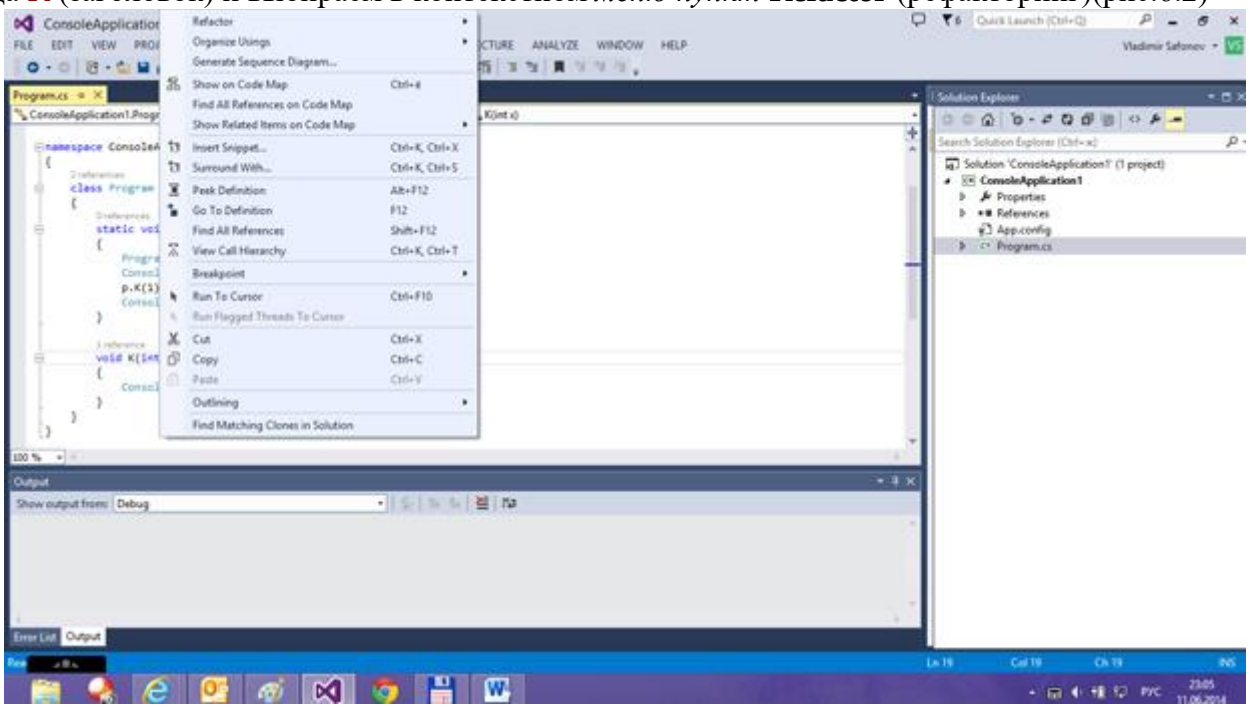


Рис. 6.2.. Выбор действия по рефакторингу для метода K

Далее выделяем имя метода - K и затем выбираем в контекстном *меню пункт* **Refactor / Rename** (если мы забыли выделить имя метода, среда подскажет нам, что это необходимо сделать) - см. рис.6.3.

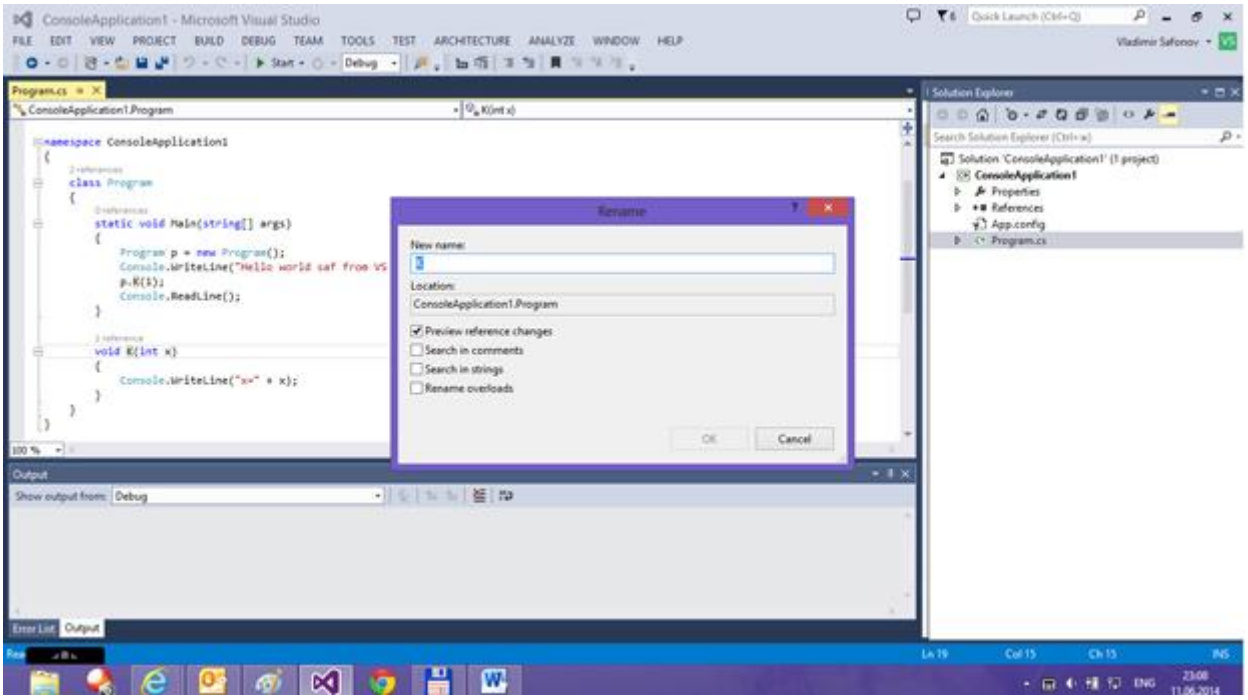


Рис. 6.3.. Выбор действия замены имени метода K

В окне рис.6.3.задаем новое имя метода - M и выбираем действие Preview Changes (оно выбрано *по* умолчанию) - см. рис.6.4.

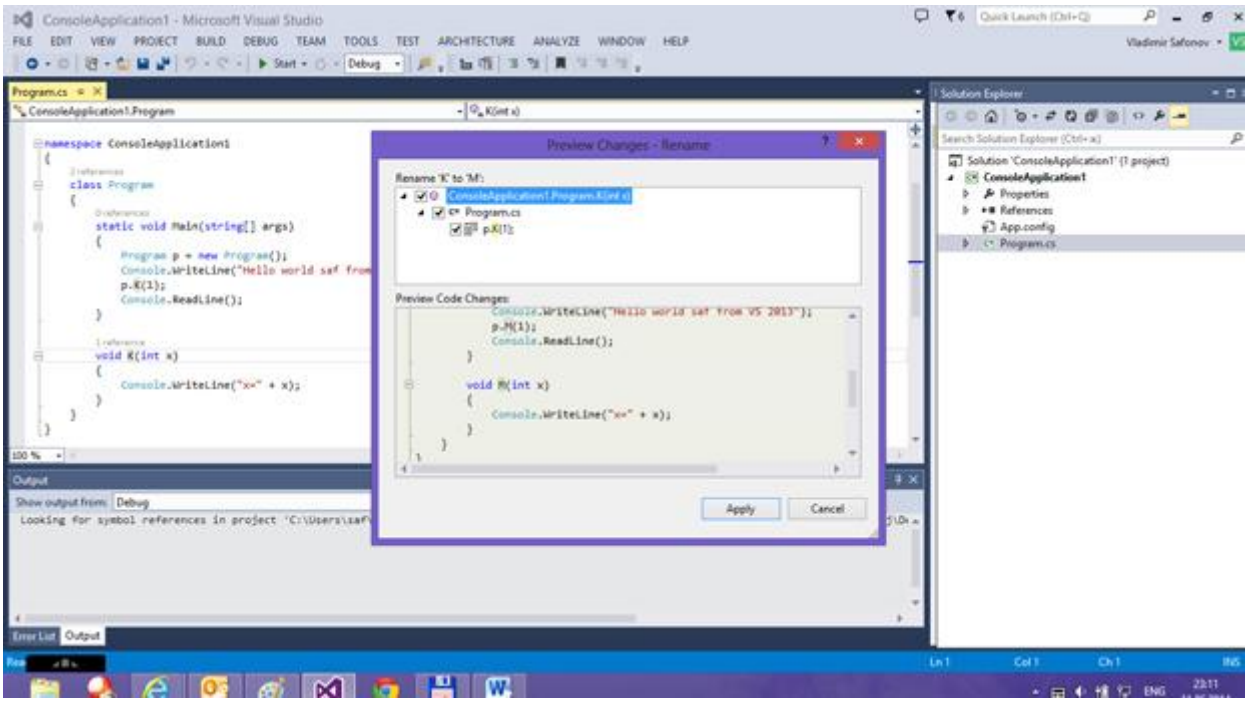


Рис. 6.4. Предварительный просмотр планируемых изменений в коде

Поскольку все предлагаемые изменения нас устраивают, нажимаем **Apply**.

На рис.6.5. показан результат рефакторинга.

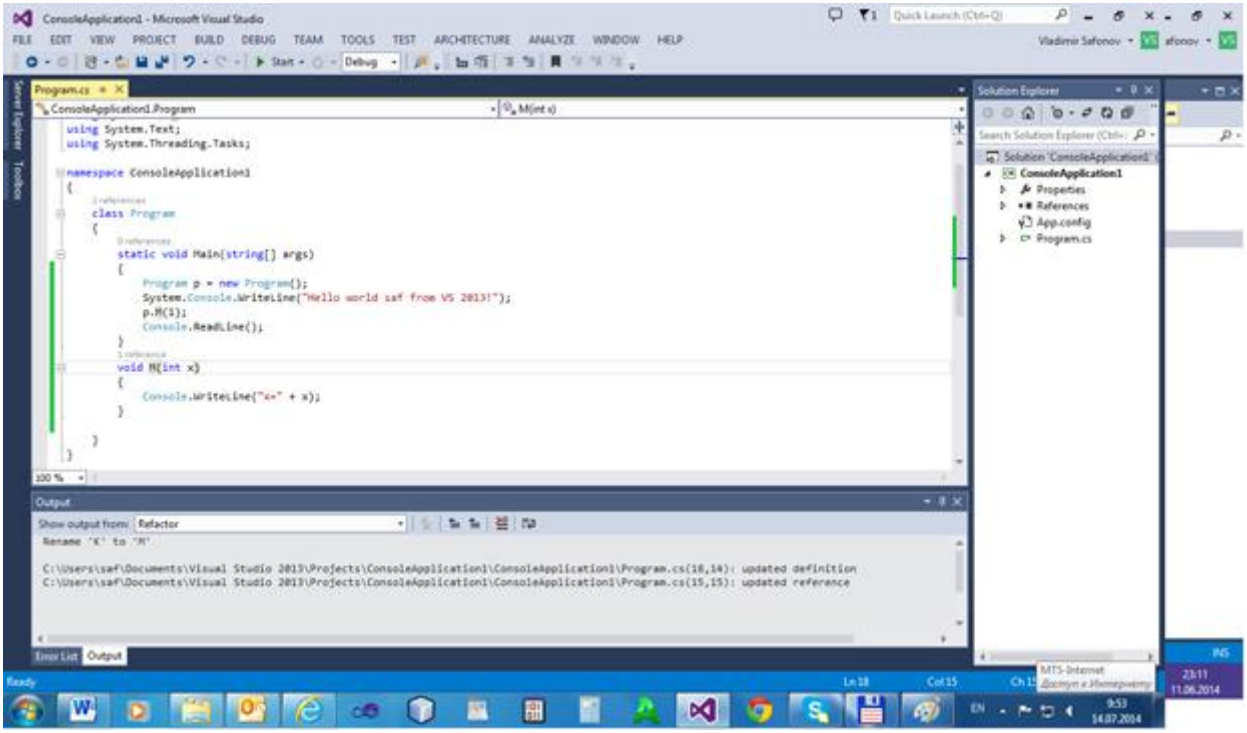


Рис.6.5. Результат рефакторинга

Чтобы убедиться в правильности выполненных средой преобразований, нужно сделать сборку проекта. В окне **Refactor** (рефакторинг)(внизу) дана *информация* о том, какие преобразования произведены.

Среда Visual Studio предоставляет следующие виды рефакторинга:

- Rename - замена имени сущности;
- Extract Method - извлечь метод: преобразовать выделенный фрагмент кода в метод с заданным именем;
- Encapsulate Field - инкапсулировать поле, сделав его приватным, но добавить public-свойство для доступа к нему;
- Extract Interface - извлечь интерфейс: выделить текст класса и автоматически сформировать для него соответствующий интерфейс (если это возможно);
- Remove parameters - удалить часть параметров метода;
- Reorder parameters - изменить порядок параметров метода.

РАЗДЕЛ 2. ЗАДАНИЕ НА СОЗДАНИЕ ДИАГРАММЫ КЛАССОВ и ПРОВЕДЕНИЕ РЕФАКТОРИНГА
(практическая часть)

а. Создание диаграммы классов для сценария "Добавить новый заказ" прецедента "Работа с заказом"

Заполнение диаграммы начнем с определения классов-сущностей. Рассматриваемый сценарий состоит из:

· самого *заказа*;

· *клиента*, который делает заказ;

· *комплектующих изделий*, которые входят в заказ.

Создадим классы-сущности *Order (Заказ)*, *Client (Клиент)* и *ComponentPart (Комплектующее изделие)*. Поскольку в один заказ может входить много разных комплектующих изделий, и одно комплектующее изделие может входить во много заказов, то введем еще один класс-сущность *OrderItem (Состав заказа)*. Опишем каждый класс.

Класс *Client:*

Параметр	Значение
Комментарий	Класс, представляющий собой клиента фирмы
поля	name : String - наименование клиента address : String - адрес клиента phone : String - телефон клиента Все атрибуты имеют модификатор доступа - private
методы	AddClient() - добавление нового клиента RemoveClient() - удаление существующего клиента GetInfo() - получить информацию о клиенте Все операции имеют модификатор доступа - public

Класс Order:

Параметр	Значение
Комментарий	Класс, представляющий собой заказ, который делает клиент
поля	orderNumber : Integer - номер заказа orderDate : Date - дата оформления заказа orderComplete : Date - дата выполнения заказа Все атрибуты имеют модификатор доступа - private
методы	Create() - создание нового заказа SetInfo() - занести информацию о заказе GetInfo() - получить информацию о заказе Все операции имеют модификатор доступа - public

Класс OrderItem:

Параметр	Значение
Комментарий	Класс, представляющий собой пункт заказа, который делает клиент
поля	itemNumber : Integer - номер пункта заказа quantity : Integer - количество комплектующих изделий price : Double - цена за единицу Все атрибуты имеют модификатор доступа - private
методы	Create() - создание новой строки заказа SetInfo() - занести информацию о строке заказа GetInfo() - получить информацию о строке заказа Все операции имеют модификатор доступа - public

Класс ComponentPart:

Параметр	Значение
Комментарий	Класс, представляющий собой комплектующие изделия
поля	name : String - наименование manufacturer : String - производитель price : Double - цена за единицу description - описание Все атрибуты имеют

	модификатор доступа - private
методы	AddComponent() - добавление нового комплектующего изделия RemoveComponent() - удаление комплектующего изделия GetInfo() - получить информацию о комплектующем изделии Все операции имеют модификатор доступа - public

Добавитьотношения между классами :

· класс *Client* и *Order* - отношение ассоциации, поскольку данные два класса просто связаны друг с другом и никакие другие типы связей здесь применить нельзя. Один клиент может сделать несколько заказов, каждый заказ поступает только от одного клиента, поэтому кратность связи со стороны класса *Client* - 1, со стороны *Order* - 1..n;

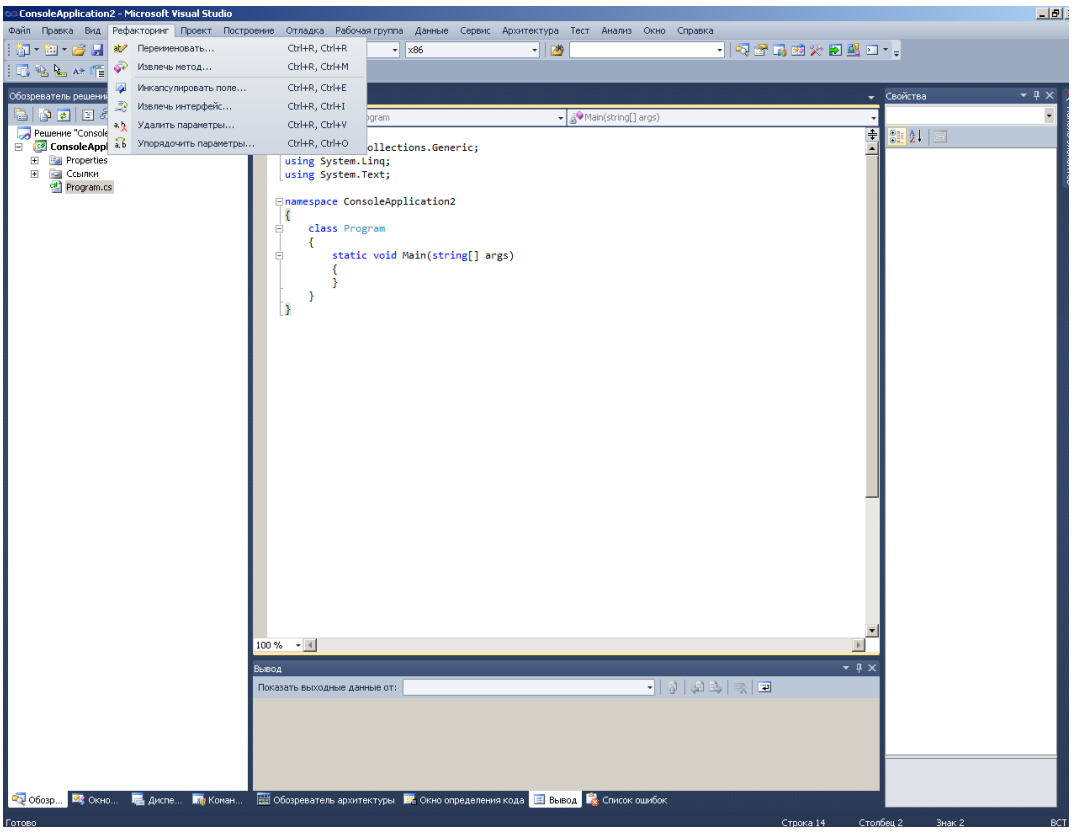
· класс *Order* и *OrderItem* - отношение ассоциации, поскольку строка заказа является частью заказа, и без него существовать не может. В один заказ может входить несколько строк заказа, строка заказа относится только к одному заказу, поэтому кратность связи со стороны *Order* - 1, со стороны *OrderItem* - 1..n;

· класс *OrderItem* и *ComponentPart* - отношение ассоциации, поскольку комплектующие изделия являются частями строки заказа, но и те, и другие, являются самостоятельными классами. Одно комплектующее изделие может входить во много строк заказа, в одну строку заказа входит только одно комплектующее изделие, поэтому кратность связи со стороны *ComponentPart* - 1, со стороны *OrderItem* - 1..n.

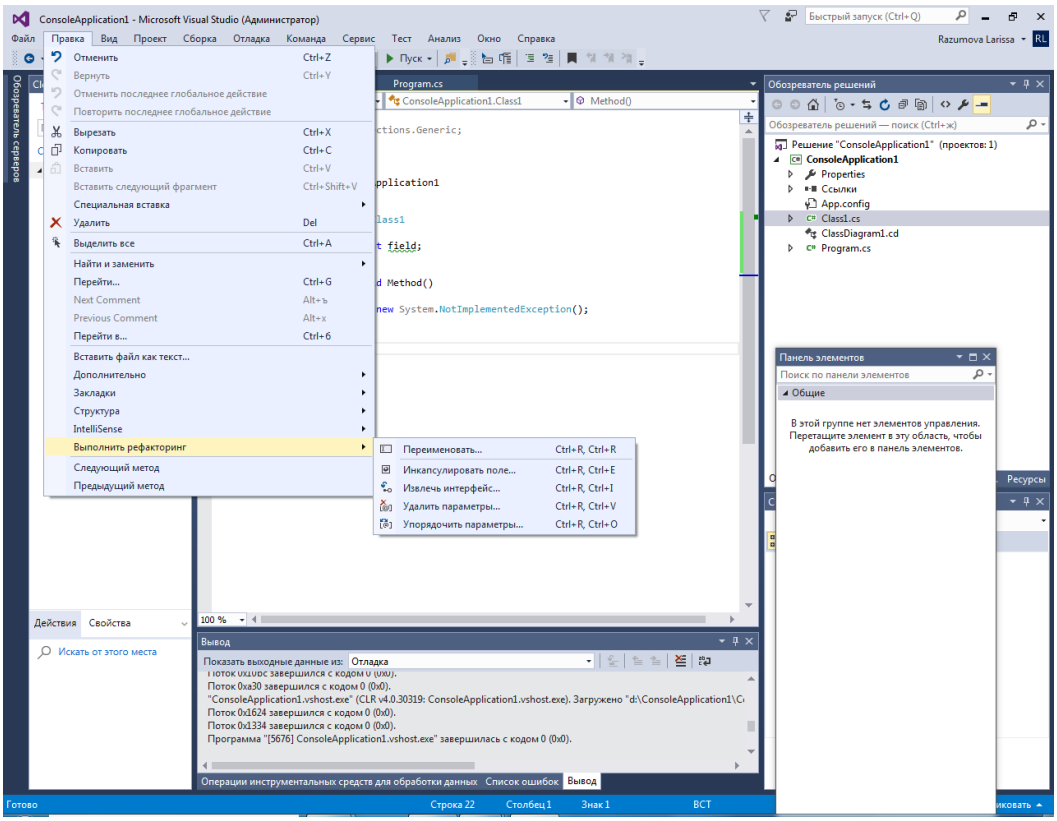
2.1.Рефакторинг

Провести все возможные на полученном коде виды рефакторинга. Листинги в отчете с комментариями **о виде рефакторинга.**

Запуск рефактиронга


ConsoleApplication2 - Microsoft Visual Studio
Файл Правка Вид Рефакторинг Проект Построение Отладка Рабочая группа Данные Сервис Архитектура Тест Анализ Окно Справка
Переименовать... Ctrl+R, Ctrl+R
Извлечь метод... Ctrl+R, Ctrl+M
Инкапсулировать поле... Ctrl+R, Ctrl+E
Извлечь интерфейс... Ctrl+R, Ctrl+I
Удалить параметры... Ctrl+R, Ctrl+V
Упорядочить параметры... Ctrl+R, Ctrl+O
Обозреватель решений
Свойства
Properties
Ссылки
Program.cs
using System.Linq;
using System.Text;
namespace ConsoleApplication2
{
class Program
{
static void Main(string[] args)
{
}
}
}
Вывод
Показать выходные данные от:
Обозреватель архитектуры
Окно определения кода
Вывод
Список ошибок
Готово
Строка 14
Столбец 2
Знак 2

ИЛИ


ConsoleApplication1 - Microsoft Visual Studio (Администратор)
Файл Правка Вид Проект Сборка Отладка Команда Сервис Тест Анализ Окно Справка
Отменить Ctrl+Z
Вернуть Ctrl+Y
Вырезать Ctrl+X
Копировать Ctrl+C
Вставить Ctrl+V
Удалить Del
Выделить все Ctrl+A
Найти и заменить
Перейти... Ctrl+G
Выполнить рефакторинг
Переименовать... Ctrl+R, Ctrl+R
Инкапсулировать поле... Ctrl+R, Ctrl+E
Извлечь интерфейс... Ctrl+R, Ctrl+I
Удалить параметры... Ctrl+R, Ctrl+V
Упорядочить параметры... Ctrl+R, Ctrl+O
Обозреватель решений
Решение "ConsoleApplication1" (проектов: 1)
ConsoleApplication1
Properties
Ссылки
App.config
Class1.cs
ClassDiagram1.cd
Program.cs
Панель элементов
В этой группе нет элементов управления. Перетащите элемент в эту область, чтобы добавить его в панель элементов.
Вывод
Операции инструментальных средств для обработки данных
Список ошибок
Готово
Строка 22
Столбец 1
Знак 1
ВСТ

ЛАБОРАТОРНАЯ РАБОТА № 5
ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ VISUAL STUDIO. ПЛАТФОРМА ASP .NET
ПАТТЕРН MVC. ИСПОЛЬЗОВАНИЕ ШАБЛОНОВ ПРЕДСТАВЛЕНИЙ MVC С ФОРМИРОВАНИЕМ ШАБЛОНОВ ДАННЫХ

Цель: освоить паттерн MVC с целью создания веб- приложений.

Задание на лабораторную работу.

1. Создать 2 задания из учебного примера проекта MVC и разобрать принцип работы.
2. Отчет представить в электронном виде: листинги кода, скриншоты результатов работы программы.
3. При защите лабораторной работы демонстрируете преподавателю понимание работы кода.
4. Проект сохраняете на своем флэш-накопителе для работы в следующих лабораторных работах.

ПРИМЕР «СОЗДАНИЕ НОВОГО ПРОЕКТА MVC»

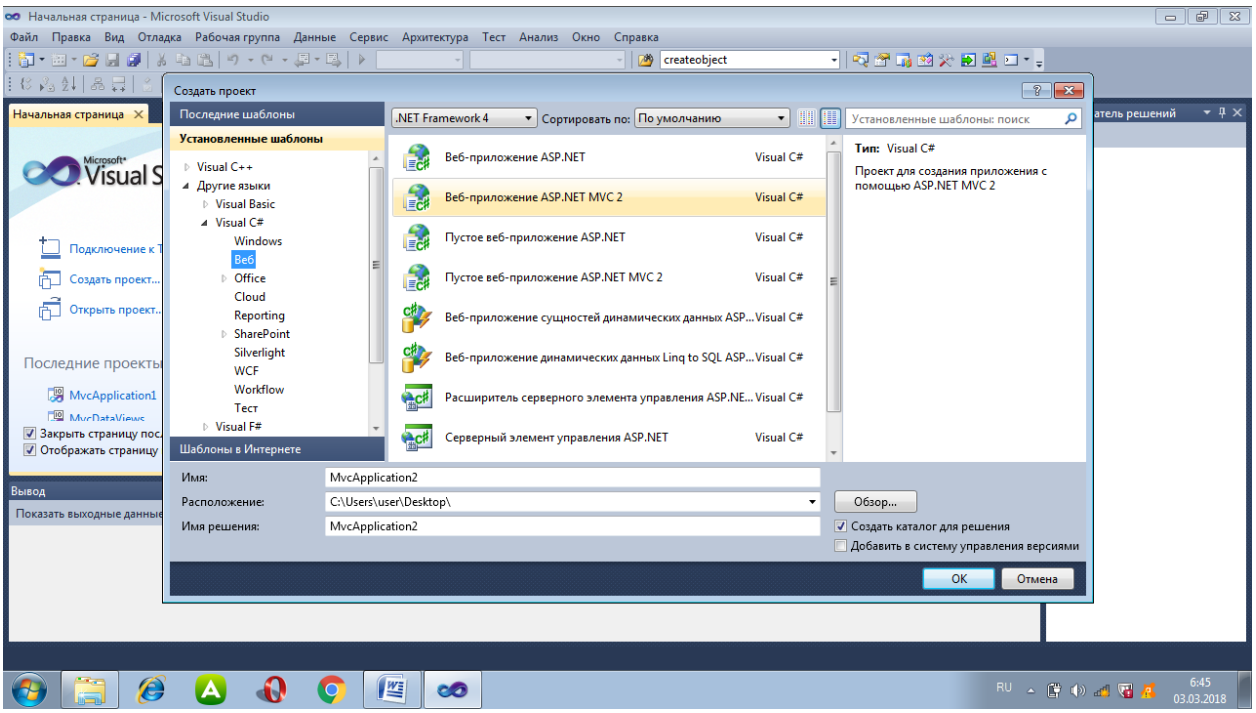
Задание № 1

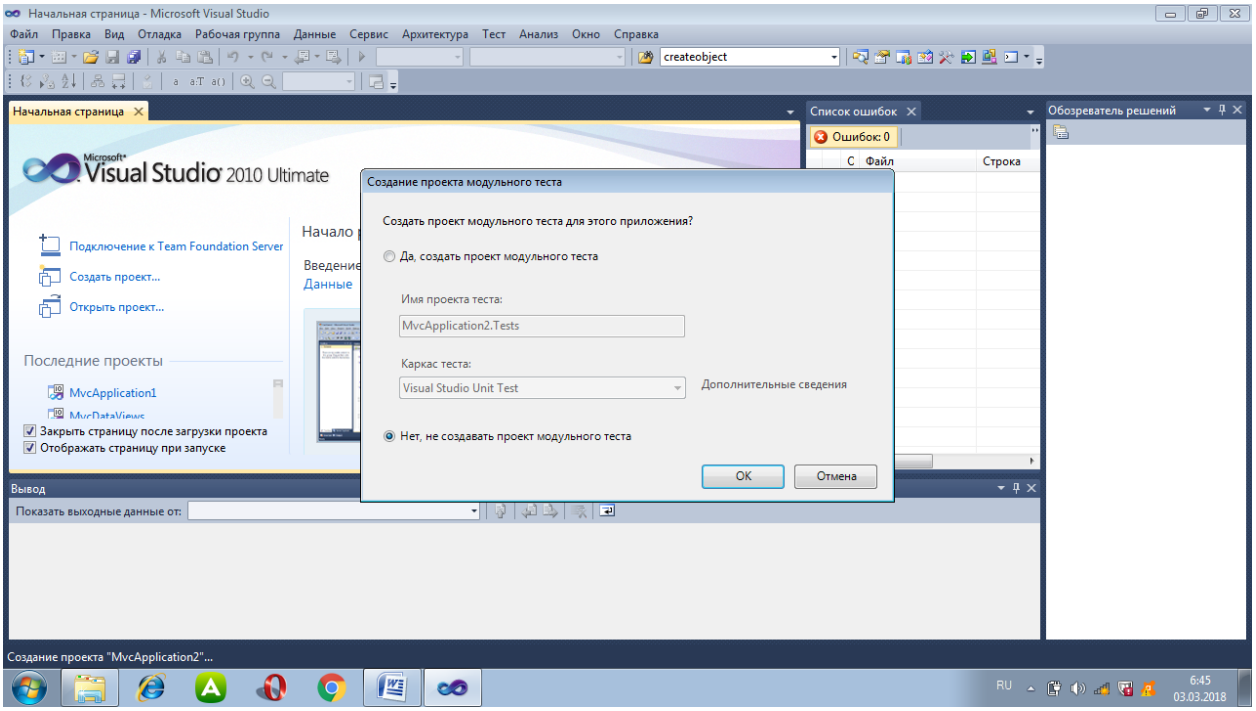
1.Создание шаблона нового проекта

Создание нового проекта MVC

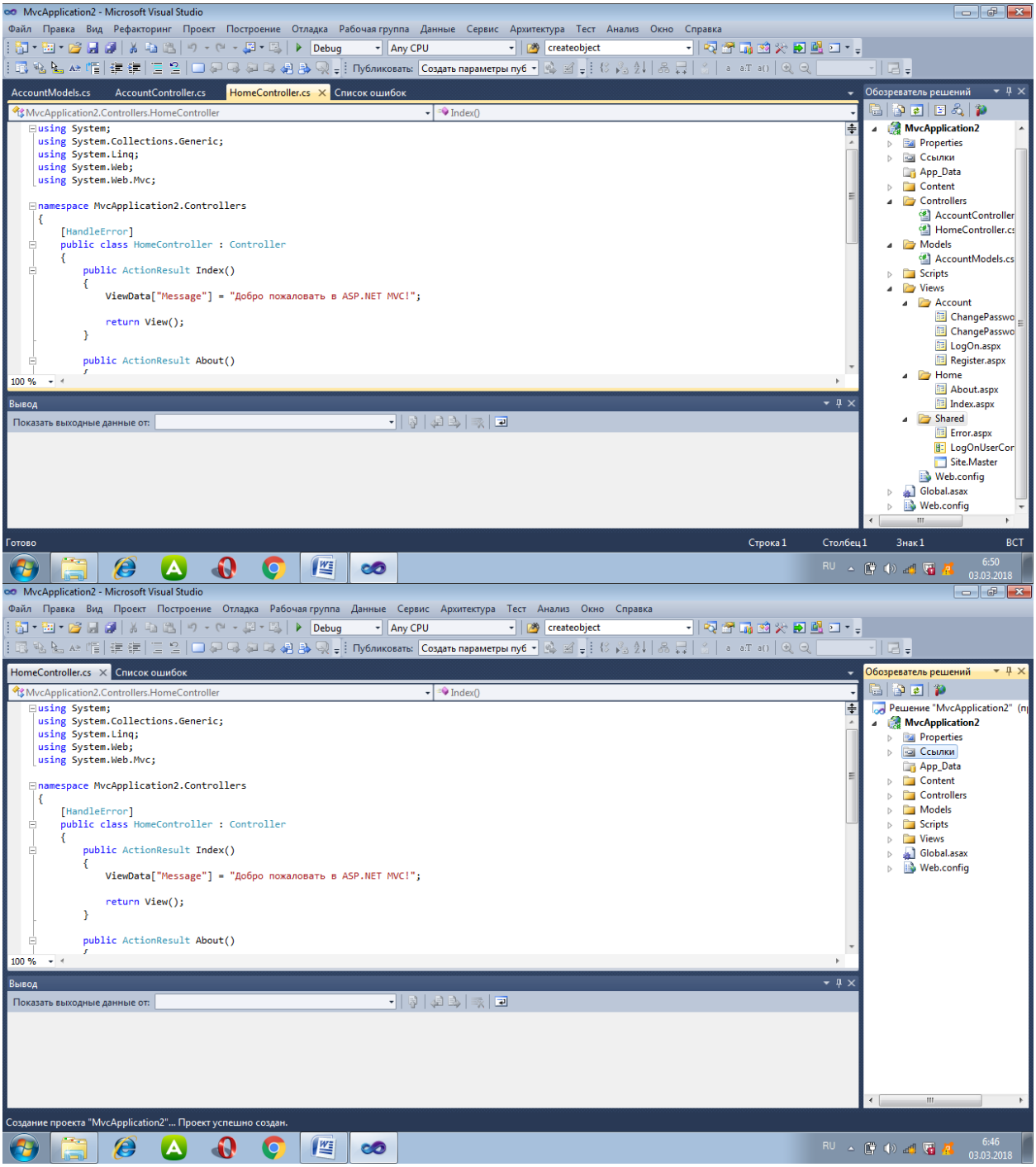
1. В меню **Файл** выберите команду **Создать проект**.
2. В диалоговом окне **Новый проект** перейдите к разделу **Типы проектов**, разверните узел **l C#**, а затем щелкните элемент **Веб**.
3. В группе **Установленные шаблоны Visual Studio** выберите пункт **Веб-приложение ASP.NET MVC 2**.
4. В поле **Имя** введите
5. В поле **Расположение** введите имя папки проекта.
6. Установите флажок **Создать каталог для решения**.
7. Нажмите кнопку **ОК**.
8. В диалоговом окне **Создание проекта тестирования** выберите элемент **Нет, не создавать проект модульного теста**.
9. Нажмите кнопку **ОК**.

Будет создан новый проект MVC-приложения.

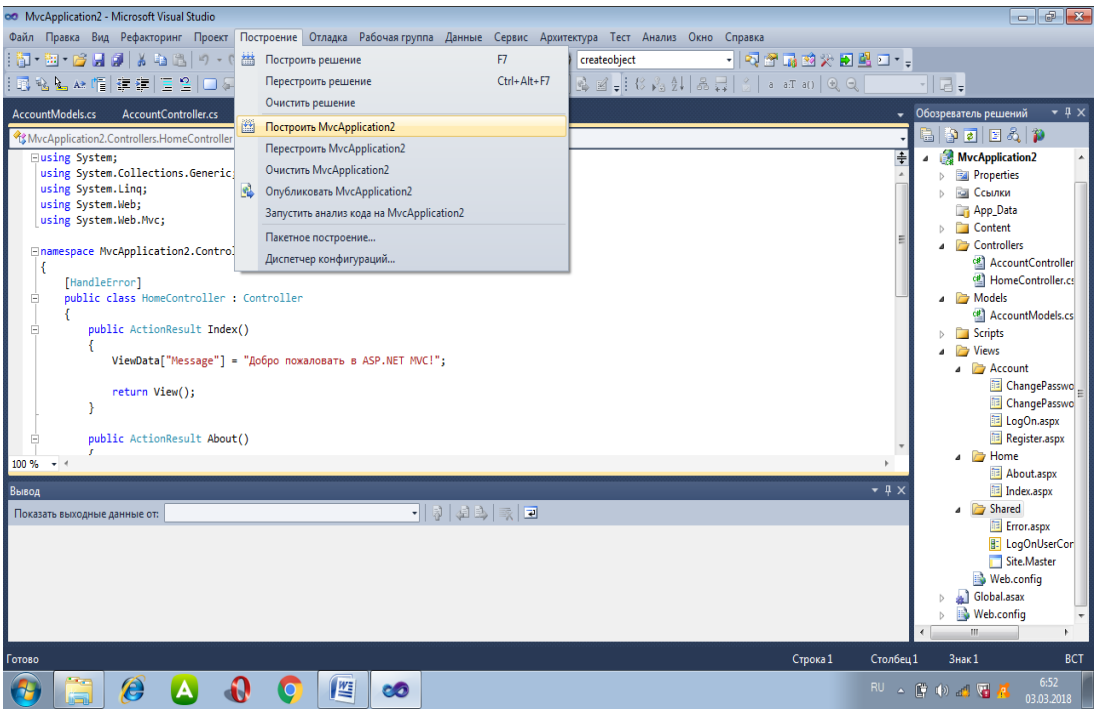




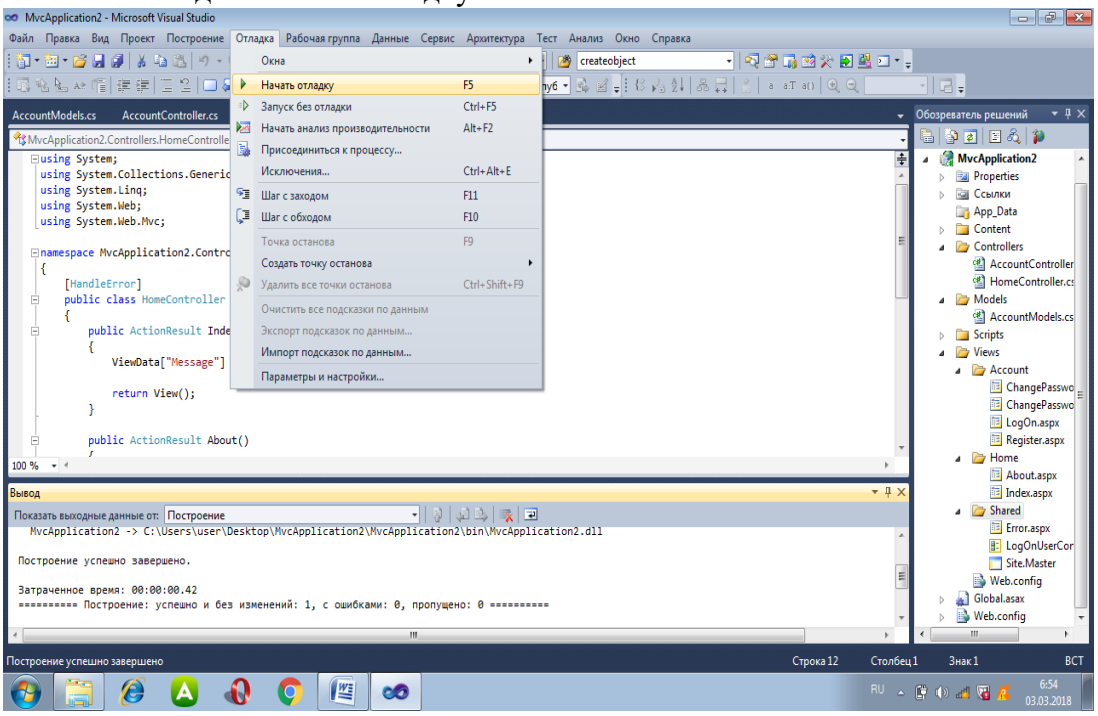
Изучите файлы полученного проекта через «обозреватель» решений. Папки Controllers, Models, Vievs.



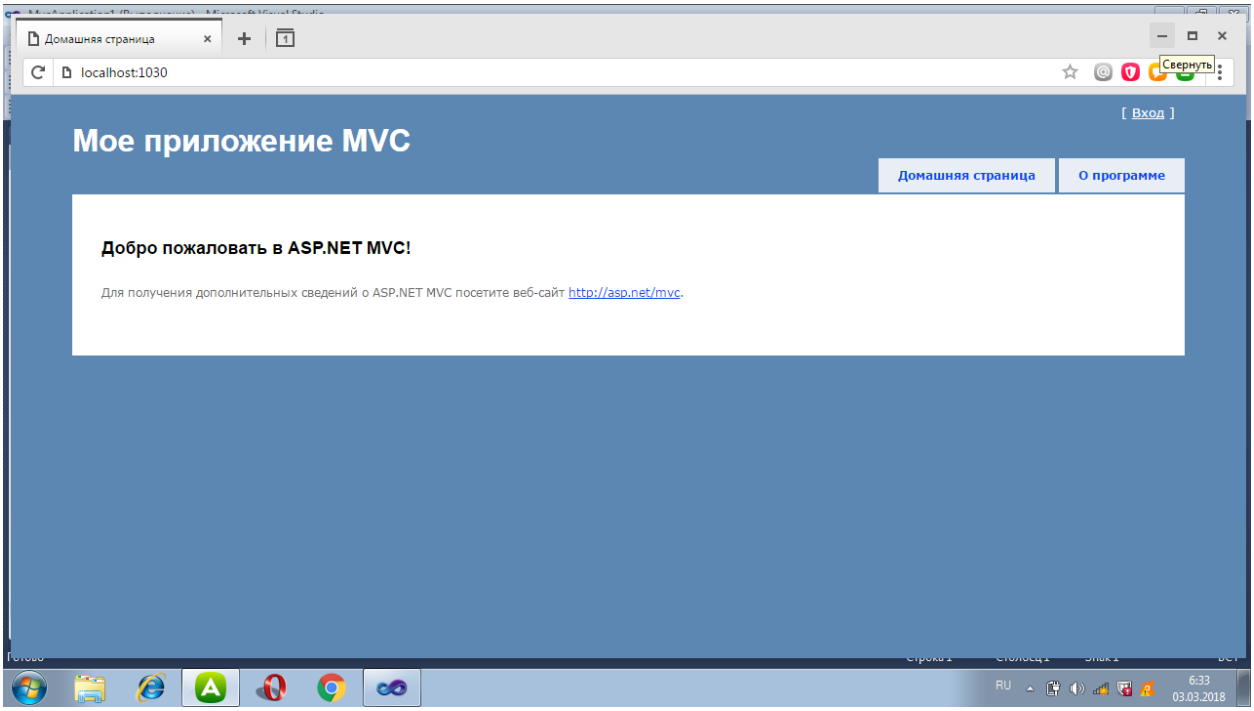
В меню «Построение» построить решение.

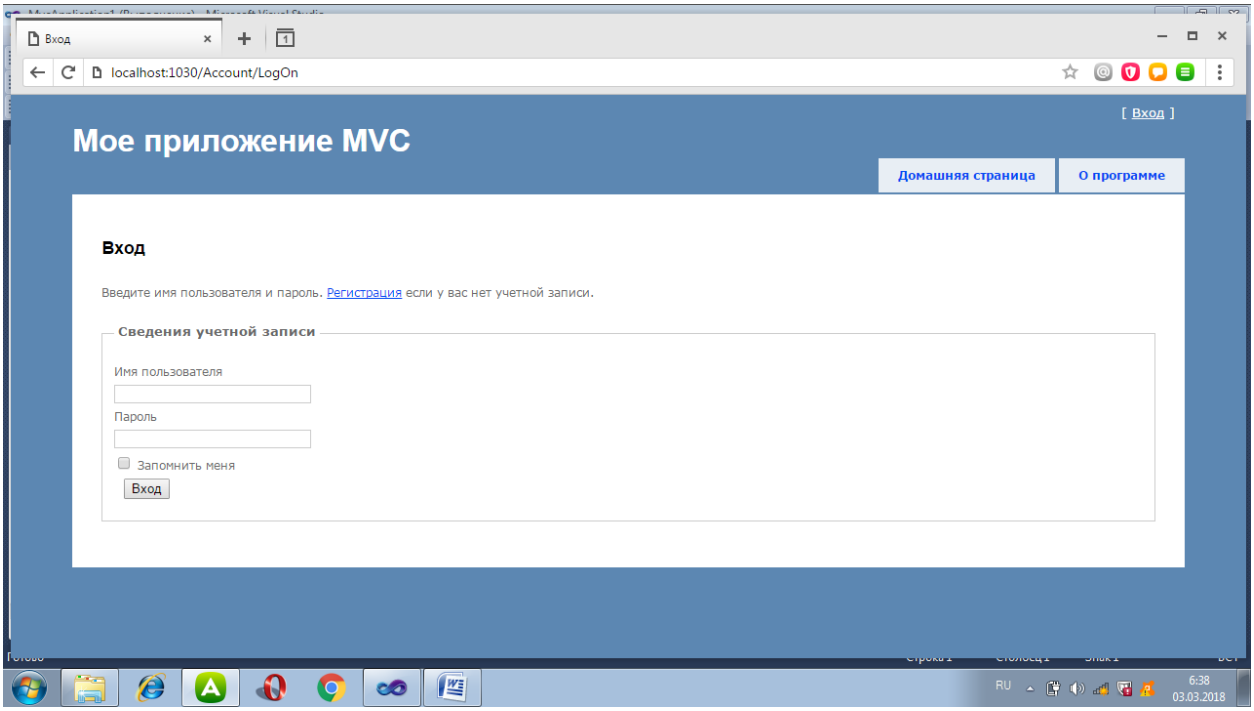


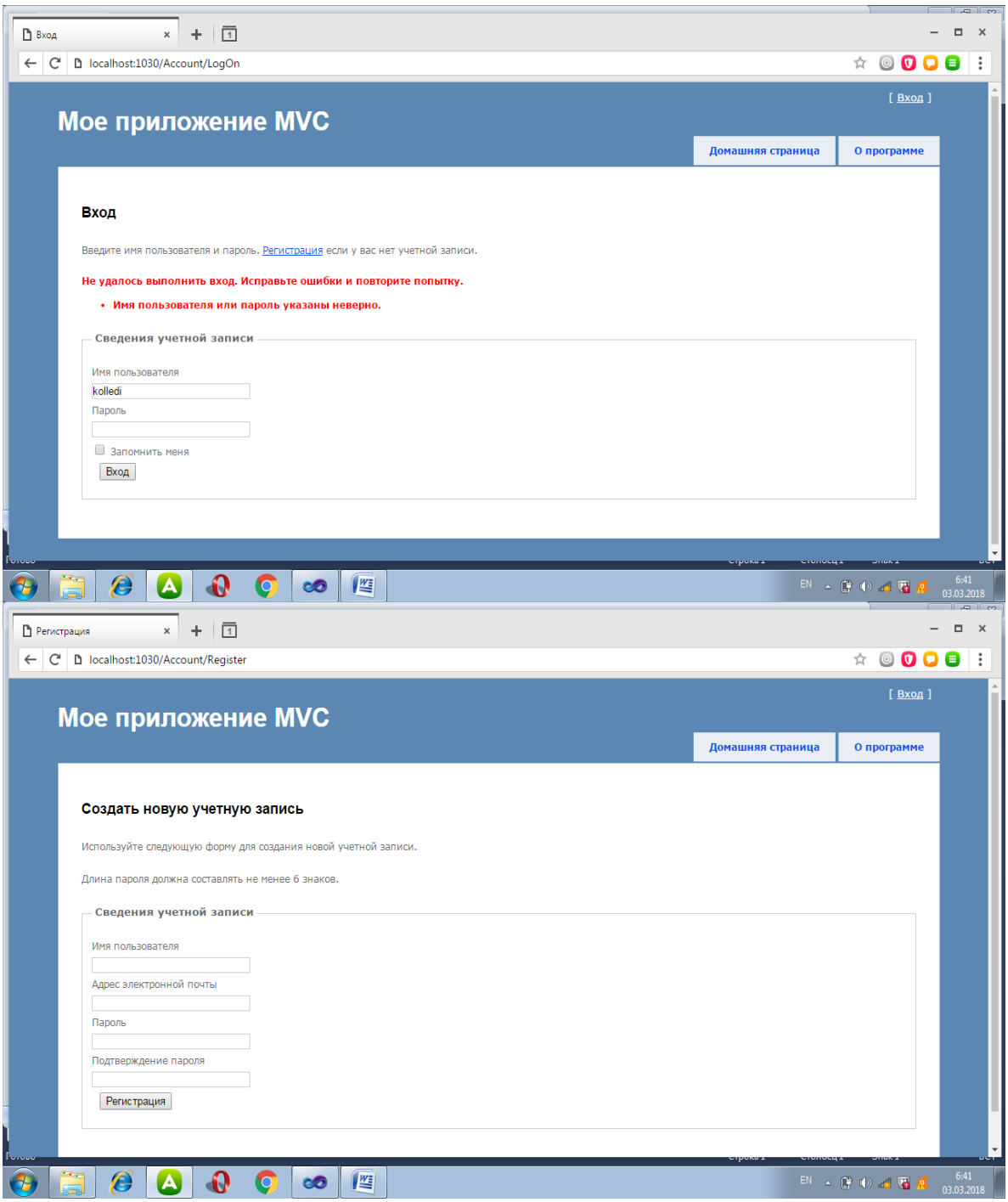
В меню «Отладка» начать отладку



Получить результаты работы представленные ниже.


Домашняя страница
localhost:1030
Свернуть
[Вход]
Мое приложение MVC
Домашняя страница
О программе
Добро пожаловать в ASP.NET MVC!
Для получения дополнительных сведений о ASP.NET MVC посетите веб-сайт http://asp.net/mvc.
RU
6:33
03.03.2018


Вход
localhost:1030/Account/LogOn
[Вход]
Мое приложение MVC
Домашняя страница
О программе
Вход
Введите имя пользователя и пароль. Регистрация если у вас нет учетной записи.
Сведения учетной записи
Имя пользователя
Пароль
Запомнить меня
Вход
RU
6:38
03.03.2018



Изучить код в файлах проекта, изменить текст «Мое приложение MVC» - « Фамилия , номер группы» и в отчете для каждого полученного скриншота результата написать имя файла, в котором находится код и «нарезки» кода, подтверждающие данное действие .

Задание № 2

При создании MVC-приложений ASP.NET в Visual Studio можно создавать контроллеры и представления, поддерживающие формирование шаблонов данных. В этом примере рассказывается, как создать простое MVC-приложение, использующее шаблоны данных для контроллеров и представлений, поддерживаемых Visual Studio для модели MVC.

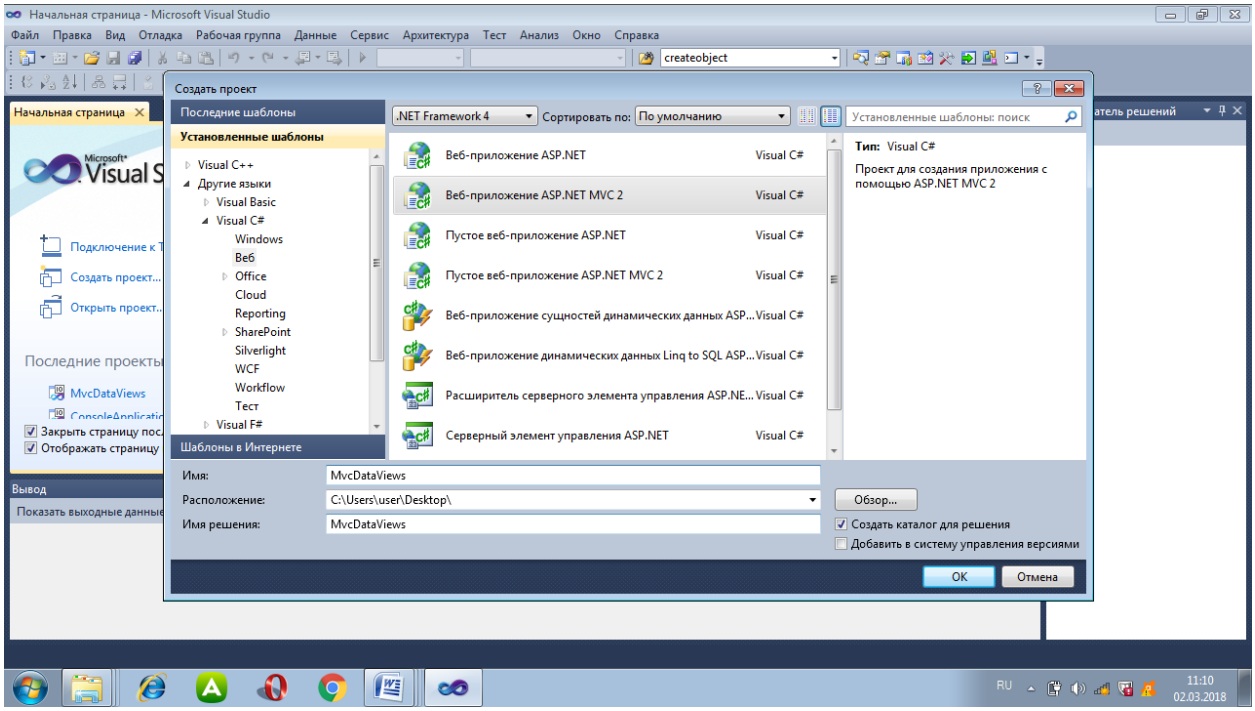
В этом примере будет добавлен контроллер, уже содержащий методы действий для отображения, изменения и обновления данных модели. Использование встроенной в MVC функции формирования шаблонов данных позволит создать представления, основанные на определенной таким образом модели.

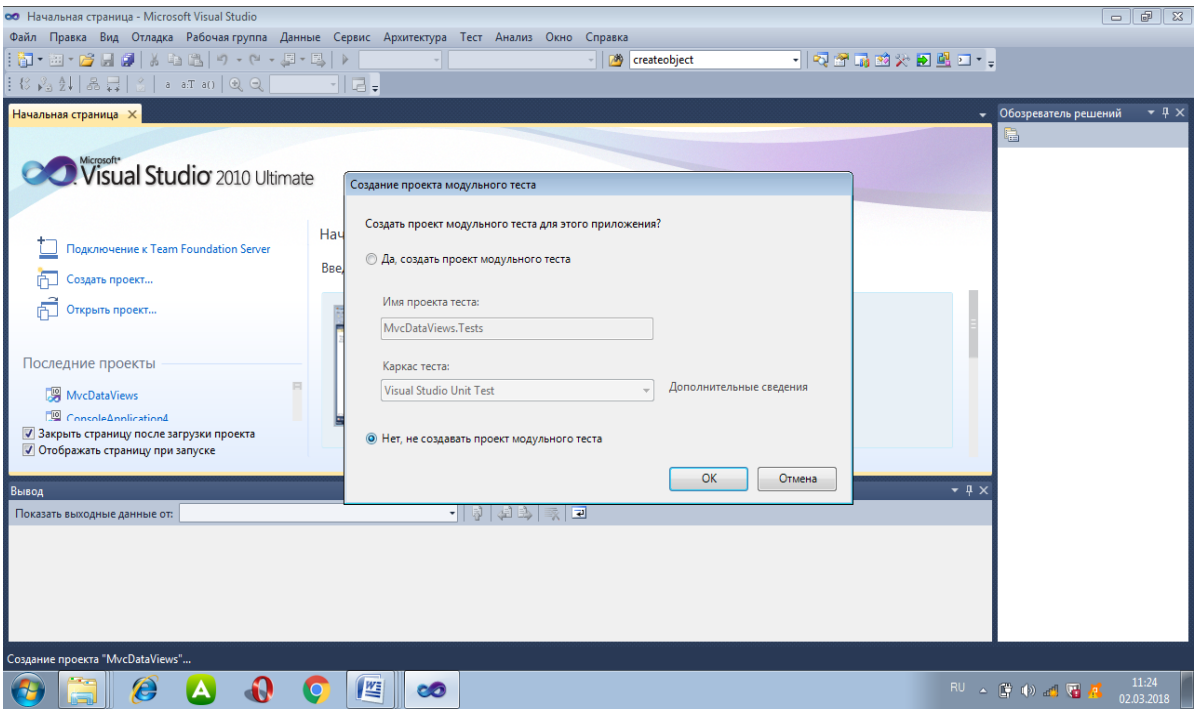
Создание нового проекта MVC

1. В меню **Файл** выберите команду **Создать проект**.
2. В диалоговом окне **Новый проект** перейдите к разделу **Типы проектов**, разверните узел **C#**, а затем щелкните элемент **Веб**.
3. В группе **Установленные шаблоны Visual Studio** выберите пункт **Веб-приложение ASP.NET MVC 2**.
4. В поле **Имя** введите ***MvcDataViews.***
5. В поле **Расположение** введите имя папки проекта.
6. Установите флажок **Создать каталог для решения**.
7. Нажмите кнопку **ОК**.
8. В диалоговом окне **Создание проекта тестирования** выберите элемент **Нет, не создавать проект модульного теста**.

9. Нажмите кнопку **ОК**.

Будет создан новый проект MVC-приложения.





Создание класса модели

В этом примере используется простая модель данных, значения в которой будут добавляться, изменяться и отображаться с помощью шаблонов представлений. Будет создан класс, представляющий данные о человеке. В этом примере у данного класса будут иметься свойства для идентификатора, имени и возраста.

Создание класса модели

1. Откройте **обозреватель проектов**, щелкните папку **Models** правой кнопкой мыши и последовательно выберите пункты **Добавить** и **Класс**.
2. Измените имя класса на Person.cs.
3. Добавьте следующий код для класса Person:

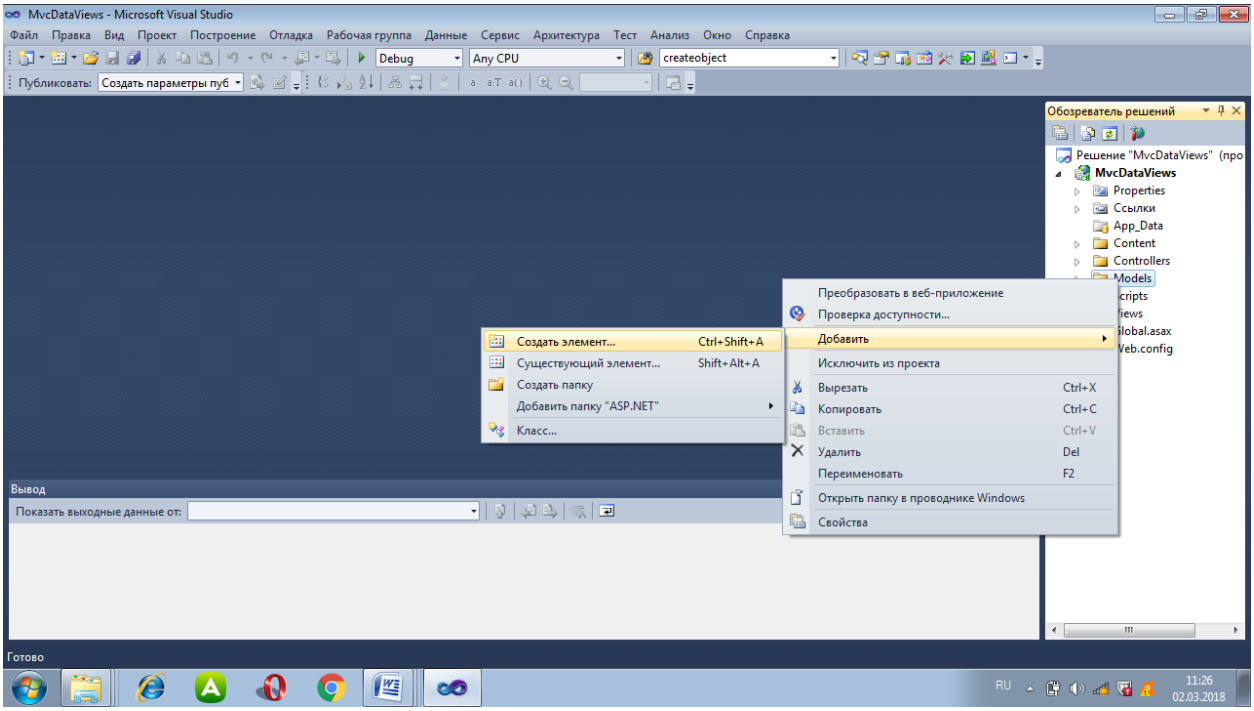
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;

namespace MvcDataViews.Models
{
   public class Person
   {
      [Required(ErrorMessage = "The ID is required.")]
      public int Id { get; set; }

      [Required(ErrorMessage = "The name is required.")]
      public string Name { get; set; }

      [Range(1, 200, ErrorMessage = "A number between 1 and 200.")]
      public int Age { get; set; }

      [RegularExpression(@"((\(\d{3}\) ?)|(\d{3}-))?\d{3}-\d{4}",
```

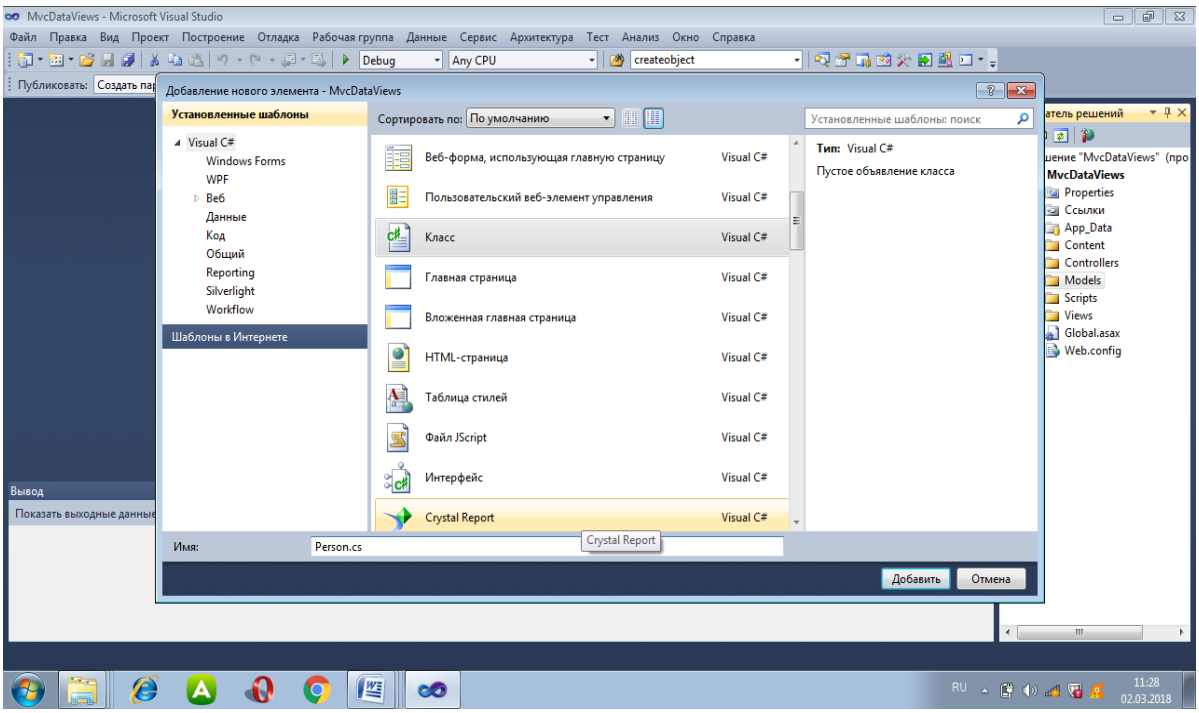
```
        ErrorMessage = "Invalid phone number.")]
    public string Phone { get; set; }

    [RegularExpression(@"^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$",
        ErrorMessage = "Invalid email address.")]
    public string Email { get; set; }
  }
}
```

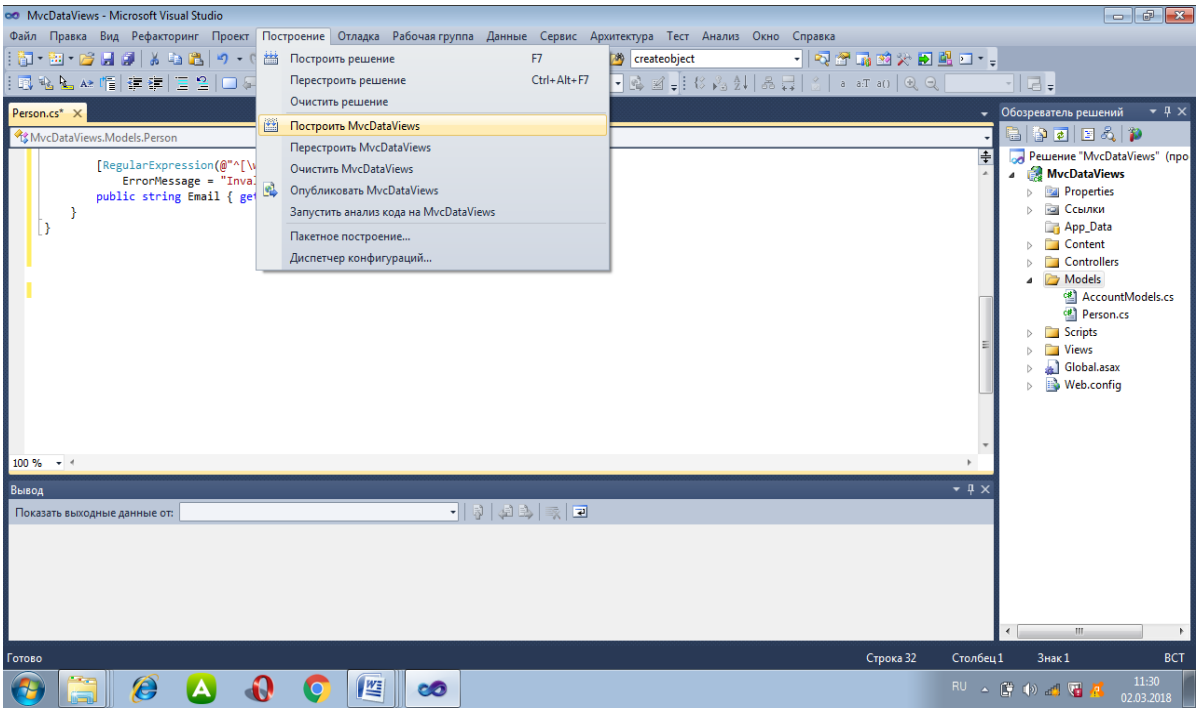
4. В меню **Построение** выберите команду **Построить MvcDataViews**, чтобы построить проект и создать экземпляр объекта Person.

Примечание
Этот этап необходим, поскольку Visual Studio использует экземпляр модели при создании кода контролера и разметки представления по шаблонам MVC.

5. Сохраните и закройте файл.







Добавление контроллера

Далее будет создан контроллер, содержащий заглушки методов действий, которые будут отображать представления для создания, изменения и показа списка объектов Person.

Добавление контроллера-обработчика данных

1. В **обозревателе решений** щелкните правой кнопкой мыши папку **Контроллеры**, а затем последовательно выберите пункты **Добавить** и **Контроллер**.
2. Назовите контроллер PersonController.
3. Установите флажок **Добавить методы действий для сценариев создания, обновления и получения сведений**.
4. Нажмите кнопку **Добавить**.

Новый контроллер будет добавлен в приложение. Контроллер содержит следующие методы действий: Index, Details,Create (для HTTP GET), Create (для HTTP POST), Delete (для HTTP GET), Delete (для HTTP POST), Edit (для HTTP GET) и Edit(для HTTP POST).

5. Добавьте следующий код в начале класса PersonController:

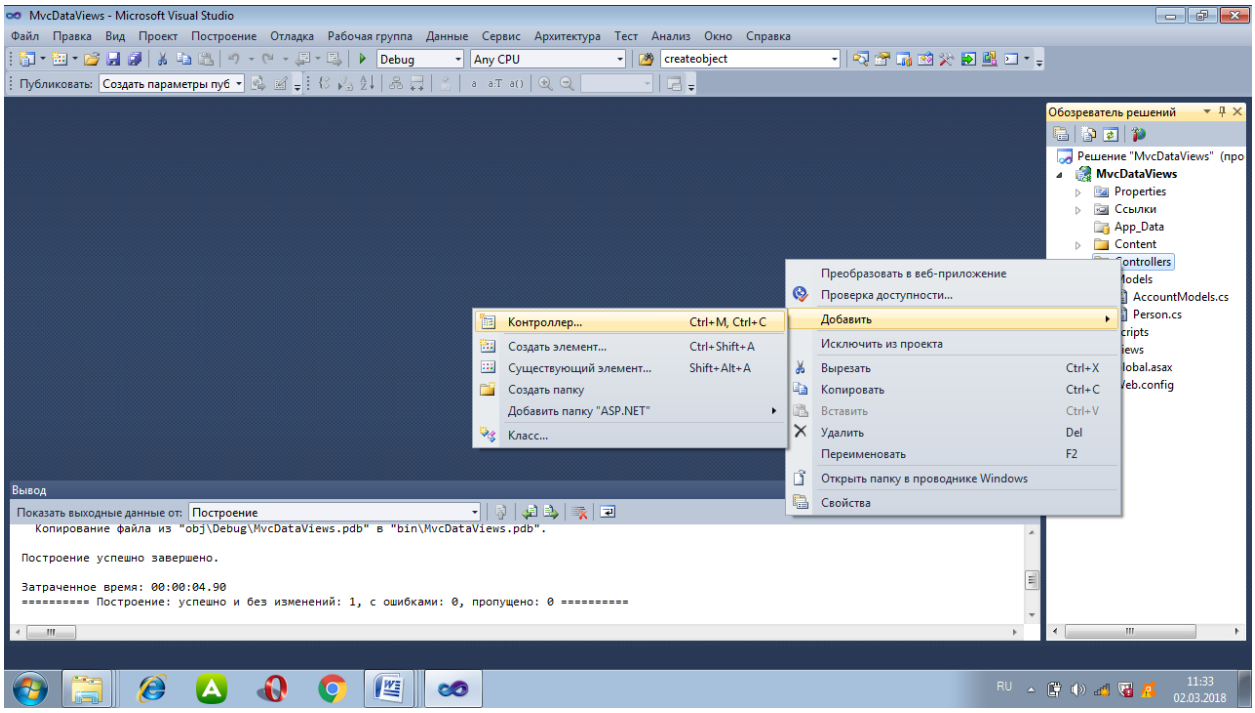
```
static List<Person> people = new List<Person>();
```

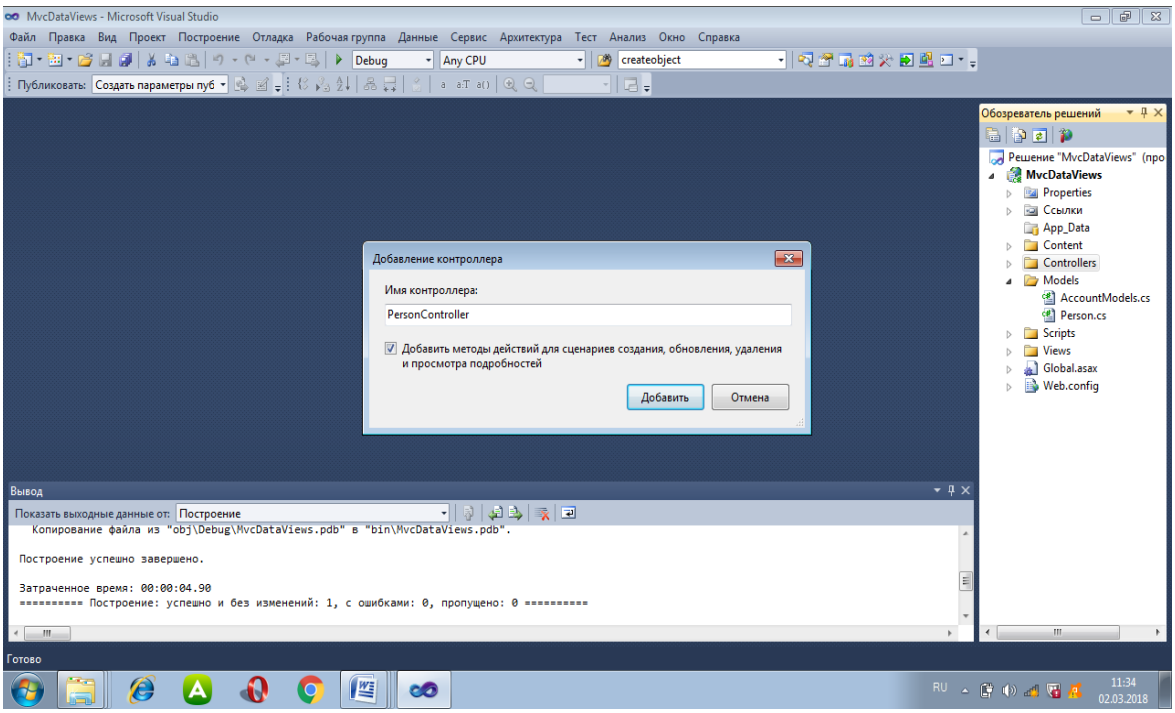
и в using

```
using MvcDataViews.Models;
```

Этот код создает список объектов Person.

6. Сохраните файл.





Добавление представления списка

Теперь можно добавить представления данных. Первое из них — это представление списка, отображаемое методом действия Index.Представление будет отображать созданные объекты Person в сетке. В каждой строке также будут ссылки для отображения детального представления или представления изменения сведений о человеке.

Добавление представления списка

1. Откройте в редакторе класс PersonController и найдите метод действия Index.
2. Щелкните метод действия Index правой кнопкой мыши и выберите пункт **Добавление представления**.
3. В поле **Имя представления** введите Index.
4. Установите флажок **Создать представление со строгой типизацией**.
5. Выберите MvcDataViews.Models.Person в списке **Класс представления данных**.
6. Выберите элемент **List** в списке **Содержимое представления**.

Примечание
Оставьте флажок **Выбрать главную страницу** в установленном положении.

7. Нажмите кнопку **Добавить**.

В новой папке **Person** будет создано представление Index. Представление Index будет содержать шаблон для отображения списка данных.

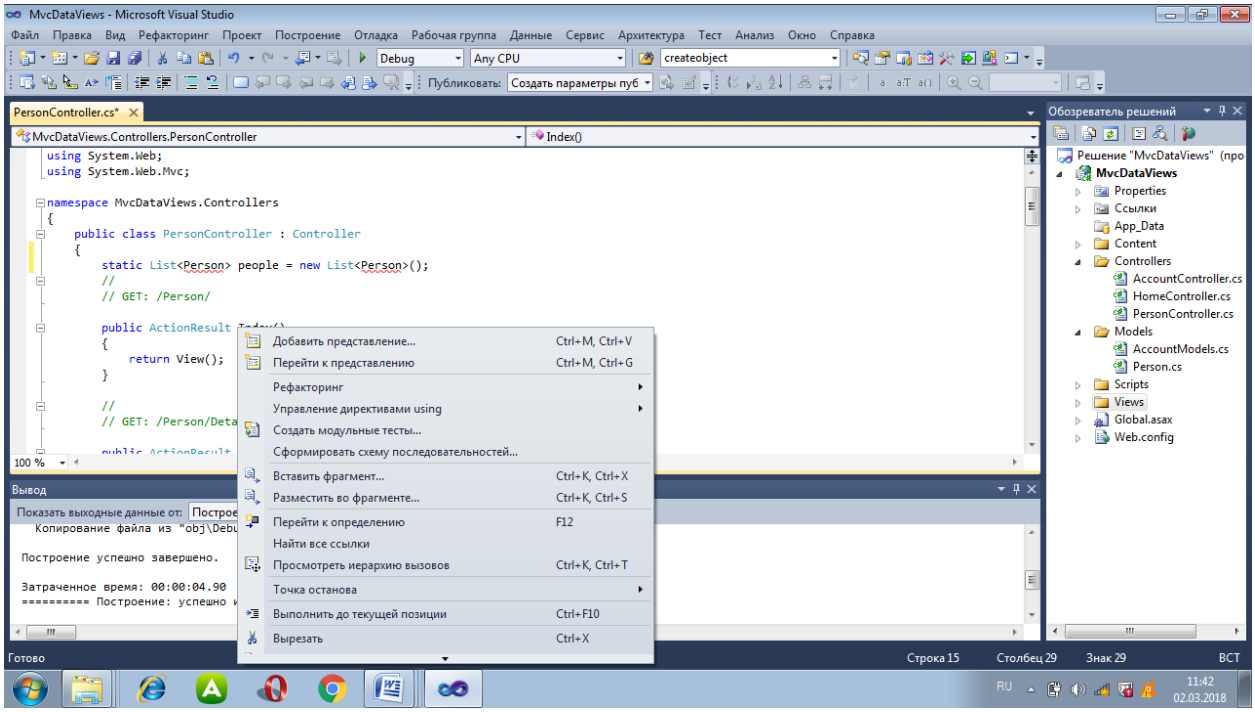
8. В представлении Index найдите элементы управления Html.ActionLink и измените их так, как показано в следующем примере:

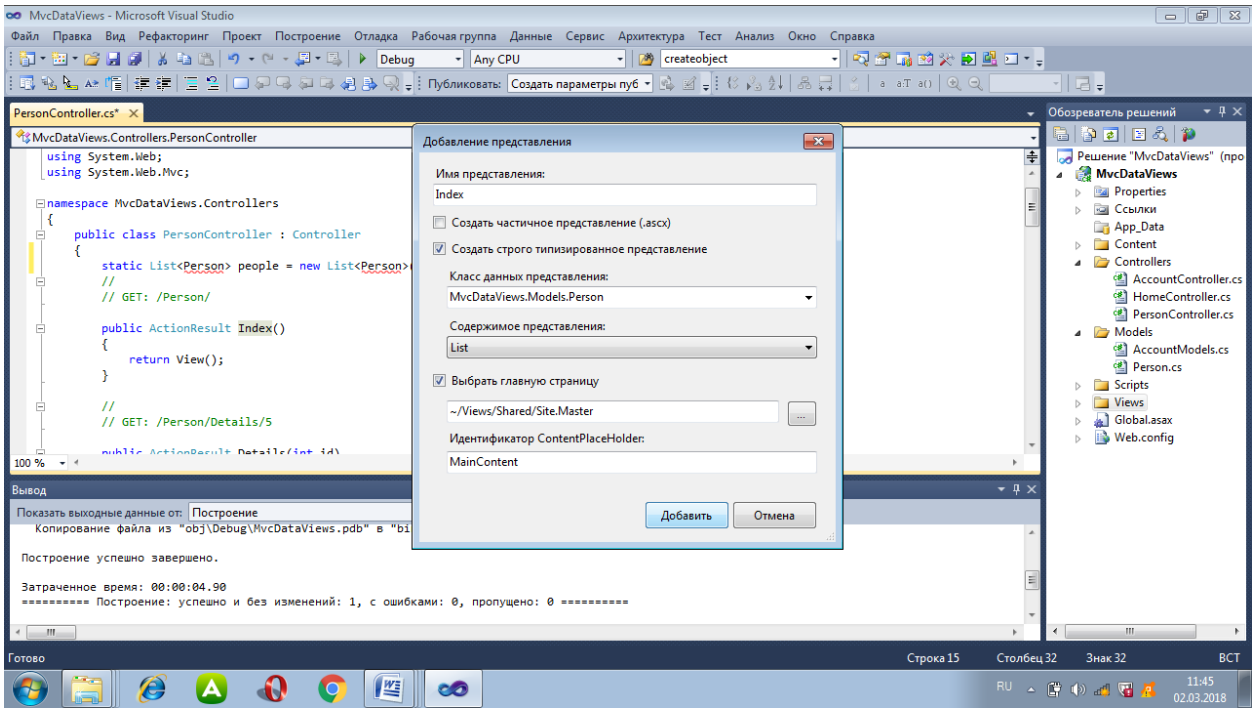
```
<%= Html.ActionLink("Edit", "Edit", new { id=item.Id }) %> |
<%= Html.ActionLink("Details", "Details", item )%> |
<%= Html.ActionLink("Delete", "Delete", new { id=item.Id })%>
```

9. В классе PersonController замените метод действия Index следующим кодом:

```
public ActionResult Index()
{
   return View(people);
}
```

10. Сохраните файлы.





Добавление представления создания

Далее добавим представление для создания объектов Person. При создании объекта Person определяется имя, возраст и идентификатор человека. Класс PersonController содержит два метода действия Create. Первый метод действия Create получает запрос HTTP GET и отображает представление создания. Другой метод действия Create получает запрос HTTP POST от представления создания, проверяет допустимость данных, добавляет их в список и выполняет перенаправление к методу действияIndex.

Добавление представления создания

1. Откройте в редакторе класс PersonController и найдите метод действия Create, обрабатывающий запрос HTTP GET.
2. Щелкните метод действия Create правой кнопкой мыши и выберите пункт **Добавление представления**.
3. В поле **Имя представления** введите Create.
4. Установите флажок **Создать представление со строгой типизацией**.
5. Выберите MvcDataViews.Models.Person в списке **Класс представления данных**.
6. Выберите элемент **Create** в списке **Содержимое представления**.

Примечание
Оставьте флажок **Выбрать главную страницу** в установленном положении.

7. Нажмите кнопку **Добавить**.

В проект будет добавлено представление Create.

8. В классе PersonController замените метод действия Create, обрабатывающий запрос HTTP POST, на следующий код:

```
[HttpPost]
public ActionResult Create(Person p)
{
   if (!ModelState.IsValid)
   {
      return View("Create", p);
   }

   people.Add(p);

   return RedirectToAction("Index");
}
```

9. Сохраните файлы.

Добавление представления сведений

Представление сведений будет отображать значения для отдельного объекта Person. Представление также будет содержать ссылки на представление изменения и на представление списка.

Добавление представления сведений

1. Откройте в редакторе класс PersonController и найдите метод действия Details.

2. Щелкните метод действия Details правой кнопкой мыши и выберите пункт **Добавление представления**.
3. В поле **Имя представления** введите Details.
4. Установите флажок **Создать представление со строгой типизацией**.
5. Выберите MvcDataViews.Models.Person в списке **Класс представления данных**.
6. Выберите элемент **Details** в списке **Содержимое представления**.

Примечание
Оставьте флажок **Выбрать главную страницу** в установленном положении.

7. Нажмите кнопку **Добавить**.

В проект будет добавлено представление Details.

8. В представлении Details найдите элемент управления Html.ActionLink, ссылающийся на представление Edit, и измените его так, как показано в следующем примере:

```
<%= Html.ActionLink("Edit", "Edit", new { id=Model.Id }) %> |
<%= Html.ActionLink("Back to List", "Index") %>
```

9. В классе PersonController замените метод действия Details следующим кодом:

```
public ActionResult Details(Person p)
{
   return View(p);
}
```

10. Сохраните файлы.

Добавление представления удаления

Представление удаления позволяет пользователю удалять объект Person из списка. Пользователь может либо удалить выбранный объект Person, либо вернуться к представлению списка. Класс PersonController содержит два метода действия Delete. Первый метод действия Delete получает запрос HTTP GET и отображает представление удаления. Другой метод действия Delete получает запрос HTTP POST от представления удаления, удаляет выбранный объект и выполняет перенаправление к методу действия Index.

Добавление представления сведений

1. Откройте в редакторе класс PersonController и найдите метод действия Delete .
2. Щелкните метод действия Delete правой кнопкой мыши и выберите пункт **Добавление представления**.
3. В поле **Имя представления** введите Delete.
4. Установите флажок **Создать представление со строгой типизацией**.
5. Выберите пункт MvcDataViews.Models.Person в списке **Класс представления данных**.
6. Выберите элемент Delete в списке **Содержимое представления**.

Примечание

Оставьте флажок **Выбрать главную страницу** в установленном положении.

7. Нажмите кнопку **Добавить**.

В проект будет добавлено представление с именем Delete.

8. В классе PersonController замените метод действия Delete, обрабатывающий запрос HTTP GET, следующим кодом:

```
public ActionResult Delete(int id)
{
   Person p = new Person();
   foreach (Person pn in people)
   {
      if (pn.Id == id)
      {
         p.Name = pn.Name;
         p.Age = pn.Age;
         p.Id = pn.Id;
         p.Phone = pn.Phone;
         p.Email = pn.Email;
      }
   }

   return View(p);
}
```

9. *Теперь нужно заменить метод действия Delete, обрабатывающий запрос HTTP POST,* ***кодом, который Вы должны разработать самостоятельно.***

Но пока ничего не заменяем, работаем дальше по созданию проекта.

А когда Вы проект будете тестировать Обратите внимание как работает Delete, и тогда вернитесь к этому пункту и попробуйте провести изменения.

10. Сохраните файлы.

Добавление представления изменения

Представление изменения позволяет менять значения объекта Person. Класс PersonController содержит два метода действия Edit. Первый метод действия Edit получает запрос HTTP GET и отображает представление изменения. Другой метод действия Editполучает запрос HTTP POST от представления изменения, проверяет допустимость данных, изменяет их в нужном объекте Person и выполняет перенаправление к методу действия Index.

Добавление представления изменения

1. Откройте в редакторе класс PersonController и найдите метод действия Edit, обрабатывающий запрос HTTP GET.

2. Замените метод действия Edit следующим кодом:

```
public ActionResult Edit(int id)
{
   Person p = new Person();
   foreach (Person pn in people)
```

```
        {
            if (pn.Id == id)
            {
                p.Name = pn.Name;
                p.Age = pn.Age;
                p.Id = pn.Id;
                p.Phone = pn.Phone;
                p.Email = pn.Email;
            }
        }

        return View(p);
    }
```

3. Щелкните метод действия Edit правой кнопкой мыши и выберите пункт **Добавление представления**.
4. В поле **Имя представления** введите Edit.
5. Установите флажок **Создать представление со строгой типизацией**.
6. Выберите MvcDataViews.Models.Person в списке **Класс представления данных**.
7. Выберите элемент **Edit** в списке **Содержимое представления**.
8. Оставьте флажок **Выбрать главную страницу** в установленном положении.
9. Нажмите кнопку **Добавить**.

В проект будет добавлено представление Edit.

10. В классе PersonController замените метод действия Edit, обрабатывающий запрос HTTP POST, на следующий код:

```
    [HttpPost]
    public ActionResult Edit(Person p)
    {
        if (!ModelState.IsValid)
        {
            return View("Edit", p);
        }

        foreach (Person pn in people)
        {
            if (pn.Id == p.Id)
            {
                pn.Name = p.Name;
                pn.Age = p.Age;
                pn.Id = p.Id;
                pn.Phone = p.Phone;
                pn.Email = p.Email;
            }
        }

        return RedirectToAction("Index");
    }
```

11. Сохраните и закройте файлы.

Ссылка с начальной страницы

В завершение добавим на начальной странице ссылку на список людей.

Добавление ссылки на список людей

1. В папке **Home** откройте представление Index.
2. Добавьте следующую разметку в строке после тега </h2>:

```
<p>
  <%=Html.ActionLink("Open Person List", "Index", "Person") %>
</p>
```

6. Сохраните и закройте файл.

Тестирование приложения

Теперь можно скомпилировать и запустить приложение.

Проверка работы приложения

1. Нажмите сочетание клавиш CTRL+F5 для запуска приложения.
2. На начальной странице щелкните ссылку **Open Person List** (Открыть список людей).
3. Щелкните ссылку **Create New** (Создать).
4. Введите значения в полях **Name** (Имя), **Age** (Возраст), **Phone** (Телефон), **Email** (Электронная почта) и **Id** (Идентификатор), после чего нажмите кнопку **Create** (Создать).

   Отобразится представление Index, в котором будет присутствовать добавленный человек.
5. Повторите предыдущий шаг несколько раз, чтобы добавить еще несколько записей.
6. В представлении Index щелкните одну из ссылок **Details** (Сведения).

   Отображается представление Details.
7. Щелкните ссылку **Back to List** (Обратно к списку).
8. Щелкните одну из ссылок **Edit** (Изменить).

   Отображается представление Edit.
9. Измените имя или возраст человека и выберите команду **Update** (Обновить).

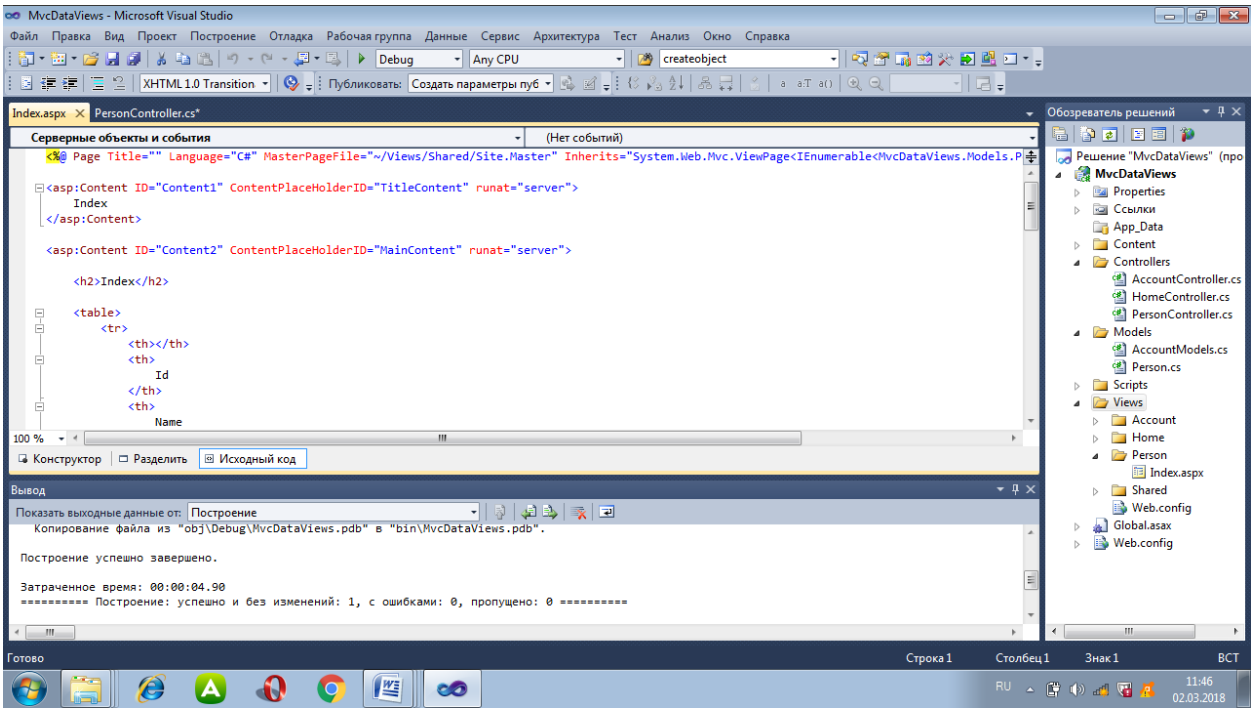
   Снова отобразится представление Index, в котором будут показаны обновленные поля данных.
10. Щелкните одну из ссылок **Delete** (Удалить).

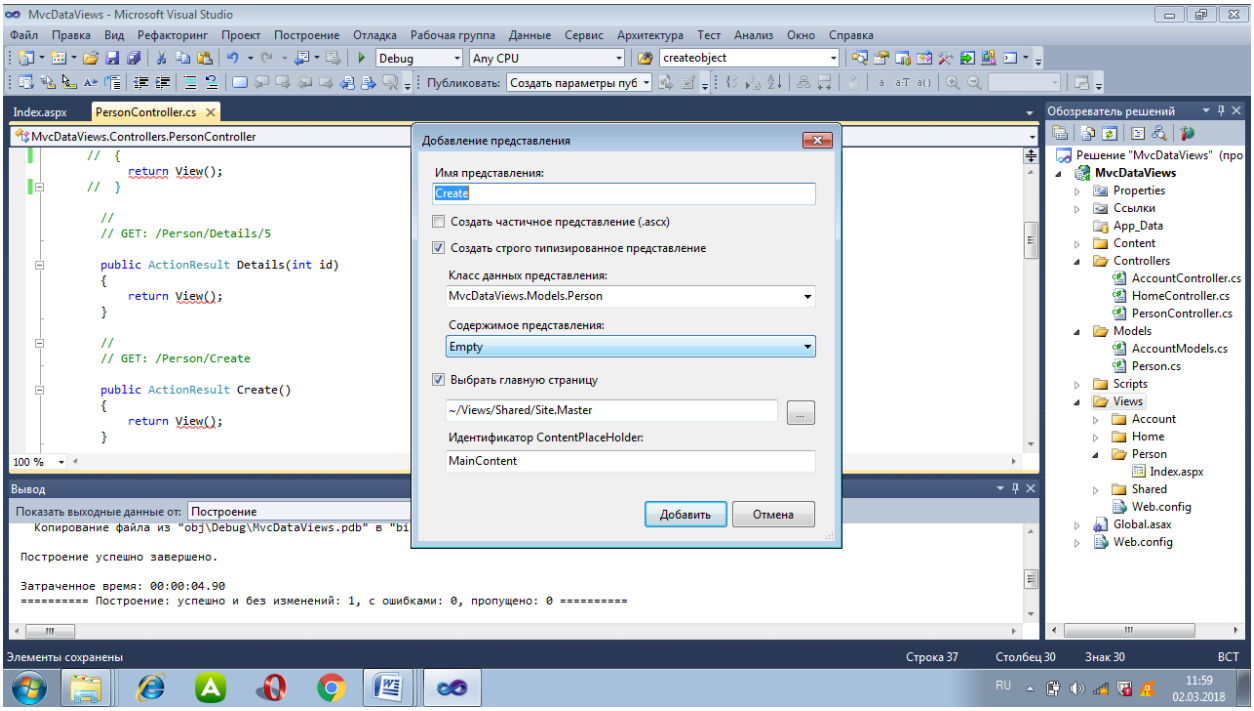
    Отображается представление Delete.
11. Нажмите кнопку **Удалить**.

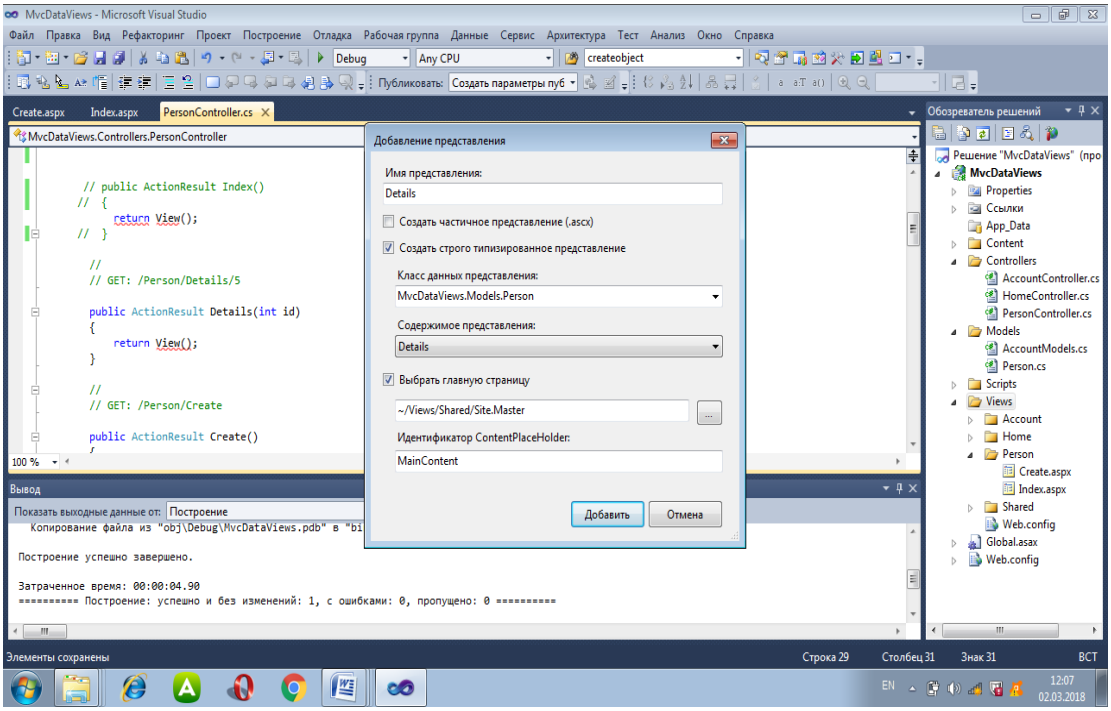
    Представление Index отображается повторно с удаленной выбранной записью.

В отчете представить скриншоты результатов работы приложения и листинги кодов.

Обратите внимание, что нужно будет вернуться к пункту « Добавление представления удаления»






MvcDataViews - Microsoft Visual Studio
Файл Правка Вид Рефакторинг Проект Построение Отладка Рабочая группа Данные Сервис Архитектура Тест Анализ Окно Справка
Debug
Any CPU
createobject
Create.aspx
Index.aspx
PersonController.cs
MvcDataViews.Controllers.PersonController
// public ActionResult Index()
// {
return View();
// }
//
// GET: /Person/Details/5
public ActionResult Details(int id)
{
return View();
}
//
// GET: /Person/Create
public ActionResult Create()
Добавление представления
Имя представления:
Details
Создать частичное представление (.ascx)
Создать строго типизированное представление
Класс данных представления:
MvcDataViews.Models.Person
Содержимое представления:
Details
Выбрать главную страницу
~/Views/Shared/Site.Master
Идентификатор ContentPlaceHolder:
MainContent
Добавить
Отмена
Вывод
Показать выходные данные от: Построение
Копирование файла из "obj\Debug\MvcDataViews.pdb" в "bi
Построение успешно завершено.
Затраченное время: 00:00:04.90
========== Построение: успешно и без изменений: 1, с ошибками: 0, пропущено: 0 ==========
Обозреватель решений
Решение "MvcDataViews" (про
MvcDataViews
Properties
Ссылки
App_Data
Content
Controllers
AccountController.cs
HomeController.cs
PersonController.cs
Models
AccountModels.cs
Person.cs
Scripts
Views
Account
Home
Person
Create.aspx
Index.aspx
Shared
Web.config
Global.asax
Web.config
Элементы сохранены
Строка 29
Столбец 31
Знак 31
ВСТ
12:07
02.03.2018

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Основные источники:

1. Введение в разработку приложений для ОС Android / Ю.В. Березовская, О.А. Юфрякова, В.Г. Вологдина и др. - 2-е изд., испр. - Москва: Национальный Открытый Университет «ИНТУИТ», 2016. - 434 с.: ил. - Библиогр. в кн.; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=428937.
2. Зубкова Т.М. Технология разработки программного обеспечения: учебное пособие /Т.М. Зубкова; Министерство образования и науки Российской Федерации, Федеральное государственное бюджетное образовательное учреждение высшего образования «Оренбургский государственный университет», Кафедра программного обеспечения вычислительной техники и автоматизированных систем. - Оренбург: ОГУ, 2017. - 469 с.: ил. - Библиогр.: с. 454-459. - ISBN 978-5-7410-1785-2; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=485553.

Дополнительные источники:

1. Антонов В.Ф. Методы и средства проектирования информационных систем: учебное пособие / В.Ф. Антонов, А.А. Москвитин; Министерство образования и науки Российской Федерации, Федеральное государственное автономное образовательное учреждение высшего профессионального образования «Северо-Кавказский федеральный университет». - Ставрополь: СКФУ, 2016. - 342 с.: ил. - Библиогр. в кн.; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=458663.
2. Васильев А.Н. Самоучитель Java с примерами и с программами. - СПб.: Наука и техника, 2016. -368с.
3. Введение в программные системы и их разработку / С.В. Назаров, С.Н. Белоусова, И.А. Бессонова и др. - 2-е изд., испр. - Москва: Национальный Открытый Университет «ИНТУИТ», 2016. - 650 с. : схем., табл., ил. - Библиогр. в кн.; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=429819.
4. Введение в разработку приложений для ОС Android / Ю.В. Березовская, О.А. Юфрякова, В.Г. Вологдина и др. - 2-е изд., испр. - Москва: Национальный Открытый Университет «ИНТУИТ», 2016. - 434 с.: ил. - Библиогр. в кн.; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=428937.
5. Долженко А.И. Технологии командной разработки программного обеспечения информационных систем / А.И. Долженко. - 2-е изд., исправ. - Москва: Национальный Открытый Университет «ИНТУИТ», 2016. - 301 с.: схем., ил. - Библиогр. в кн.; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=428801.
6. Мельников В.П., Схиртладзе А.Г. Методы и средства хранения и защиты компьютерной информации. – Старый Оскол: ТНТ, 2016. -400 с.
7. Митина О.А. Методы и средства проектирования информационных систем и технологий: курс лекций / О.А. Митина; Министерство транспорта Российской Федерации. - Москва: Альтаир: МГАВТ, 2016. - 76 с.: ил. - Библиогр. в кн.; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=482395.
8. Нужнов Е.В. Мультимедиа технологии: учебное пособие / Е.В. Нужнов; Министерство образования и науки РФ, Южный федеральный университет. - 2-е, перераб. и дополн. - Таганрог: Издательство Южного федерального университета, 2016. - Ч. 2. Виртуальная реальность, создание мультимедиа продуктов, применение мультимедиа технологий в профессиональной деятельности. - 180 с.: ил. - Библиогр. в кн. - ISBN 978-5-9275-2171-5; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=493255.
9. Сафонов В.О. Развитие платформы облачных вычислений Microsoft Windows Azure / В.О. Сафонов. - 2-е изд., испр. - Москва: Национальный Открытый Университет

«ИНТУИТ», 2016. - 393 с.: ил.; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=428823.
10. Черников Б.В. Оценка качества программного обеспечения: Практикум. – М.: ФОРУМ: ИНФРА-М, 2017. - 400 с.
11. Шклярова Е.И. Управление качеством, стандартизация и сертификация: курс лекций / Е.И. Шклярова; Министерство транспорта Российской Федерации. - Москва: Альтаир: МГАВТ, 2016. - 103 с.: ил. - Библиогр. в кн.; [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=482413.

Интернет-ресурсы:

1. Компьютерные книги. Режим доступа: [http://computers.plib.ru/programming/Books.VBasic6/index.html 09.04.2021];
2. Технология программирования. Электронное пособие по дисциплине "Технология Программирования". Чернев Дмитрий Алексеевич. Режим доступа: [http://www.tehprog.ru 09.04.2021];
3. Межрегиональных испытательный центр Сертификация программного обеспечения СИ, АСУ, АСУТП, БД на требования ГОСТ. Режим доступа: [http://www.testrussia.ru/ 09.04.2021].
4. Все для программиста! Режим доступа: [http://www.codenet.ru/ 09.04.2021];
5. On-line библиотека свободно доступных материалов по информационным технологиям. Режим доступа: [http://digitland.ru 09.04.2021].

1.