ЧПОУ ВОЛОГОДСКИЙ КООПЕРАТИВНЫЙ КОЛЛЕДЖ

ЧПОУ Вологодский кооперативный колледж

А.А. Дроздова

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

Основы объектно-ориентированного программирования

Методические рекомендации к практическим занятиям для студентов очной формы обучения по специальности

09.02.04 Информационные системы (по отраслям)

Вологда–2018

Составитель: Дроздова Анна Александровна, преподаватель высшей квалификационной категории ЧПОУ Вологодский кооперативный колледж.

Дроздова А.А. Основы алгоритмизации и программирования. Основы объектно-ориентированного программирования: методические рекомендации к практическим занятиям для студентов очной формы обучения по специальности 09.02.04 Информационные системы (по отраслям) /сост. А.А. Дроздова. – Вологда: ЧПОУВКК, 2018. – 82с.

В методических рекомендациях представлены одиннадцать практических занятий по разделу Основы объектно-ориентированного программирования, которые содержат: цель занятия, требования к результатам освоения дисциплины, список формируемых общих и профессиональных компетенций, оборудование, технические и программные средства, практические задания и методические указания, контрольные задания и задания для внеаудиторной самостоятельной работы, критерии оценивания.

Методические рекомендации составлены с целью оказания помощи студентам специальности 09.02.04 Информационные системы (по отраслям) в организации самостоятельной работы при подготовке к практическим занятиям по учебной дисциплине Основы алгоритмизации и программирования, а также могут быть использованы преподавателями при проведении учебных занятий.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Практические работы занимают центральное место в изучении учебной дисциплины Основы алгоритмизации и программирования специальности 09.02.04 Информационные системы (по отраслям). Их целью является организация и управление самостоятельной работой студента в процессе практических занятий, а также:

- формирование практических умений по разработке Windows-приложений, мобильных приложений и навыков работы в среде программирования;
- обобщение, углубление, систематизация и закрепление полученных теоретических знаний;
- формирование умений применять полученные знания на практике;
- развитие таких профессионально значимых качеств личности, как наблюдательность, умение сравнивать, обобщать, самостоятельность, ответственность, творческая инициатива.

В результате выполнения практических работ студент должен:

знать:

- общие принципы построения алгоритмов, основные алгоритмические конструкции;
- понятие системы программирования;
- основные элементы процедурного языка программирования, структуру программы, операторы и операции, управляющие структуры, структуры данных, файлы, кассы памяти;
- подпрограммы, составление библиотек подпрограмм;
- объектно-ориентированную модель программирования, понятие классов и объектов, их свойств и методов.

уметь:

- использовать языки программирования, строить логически правильные и эффективные программы.

Предлагаемые во второй части методических рекомендаций задания охватывают основные возможности объектно-ориентированной среды программирования по созданию приложений. Приобретенные в процессе выполнения практических работ умения служат основой для последующего быстрого освоения других языков и сред программирования.

Во второй части методических указаний представлены девятнадцать практических работ. Каждая практическая работа рассчитана на выполнение в течение двух академических часов. В процессе выполнения некоторых практических работ предлагается использовать вспомогательные файлы.

Практические работы следует выполнять последовательно, не пропуская предыдущие. Каждая последующая работа раскрывает дополнительные возможности изучаемой среды программирования, основываясь на знаниях, полученных при изучении материала предыдущих практических работ.

Методические указания предназначены для использования на аудиторных занятиях преподавателями и студентами средних профессиональных учебных заведений, обучающихся на 2 курсе специальности 09.02.04 Информационные системы (по отраслям) при изучении учебной дисциплины Основы алгоритмизации и программирования.

ПРАКТИЧЕСКАЯ РАБОТА №1
ИЗУЧЕНИЕ ИНТЕГРИРОВАННОЙ СРЕДЫ РАЗРАБОТЧИКА. СОЗДАНИЕ ПРОСТОГО ПРОЕКТА

Цель работы: сформировать умения по использованию и настройке интегрированной среды разработчика **VisualStudio** для создания**Windows-**приложений, сформировать умения по созданию простейших приложений.

Формируемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.

В результате освоения практических заданий обучающийся должен

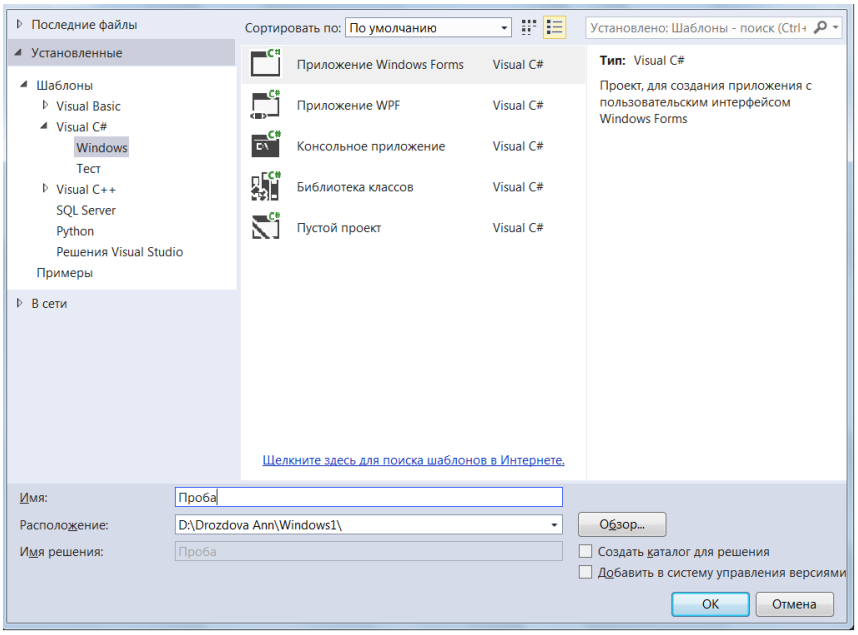
- уметь: использовать языки программирования, строить логически правильные и эффективные программы
- знать: объектно-ориентированную модель программирования, понятие системы программирования, основные алгоритмические конструкции.

Материально-техническое обеспечение занятия: персональный компьютер, среда программирования **VisualStudio**.

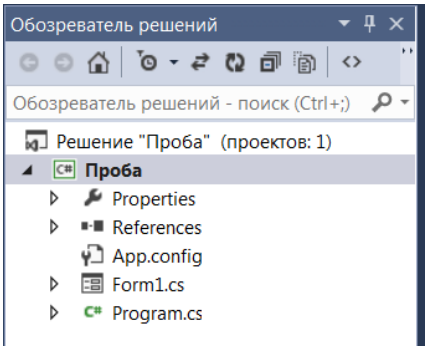
Практические задания и методические указания

Задание 1. Изучение интегрированной среды разработчика

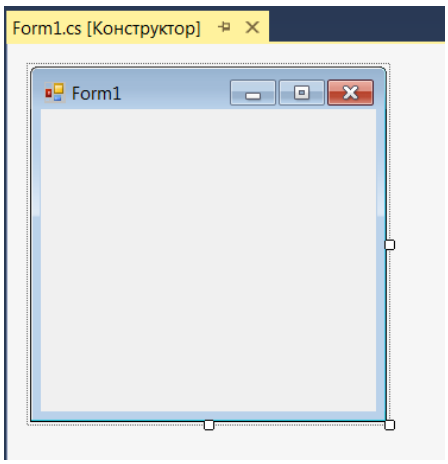
1. Запустите среду программирования **VisualStudio**. На начальной странице выберите пункт **Создать проект**. В разделе шаблонов выберите **Приложение Windows Forms**. Имя проекта и приложения – **Проба**. Папка для размещения проекта **Windows1**.



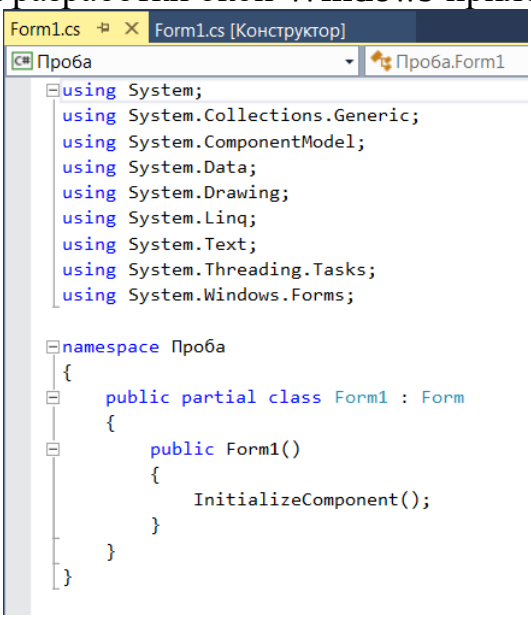
2. В результате будет создана заготовка нового проекта. В окне **Обозревательрешений** виден состав нашего проекта. Он не пуст – в папках уже имеются компоненты. Именно те, которые нужны для организации **Windows**-приложения. Каждое решение в **C#** может содержать несколько проектов. В нашем случае – это один проект под названием **Проба**. В составе проекта есть два программных файла – **Program.cs** (такой файл вы уже видели в консольном приложении) и **Form1.cs**.



3. На экране слева вы видите отображение содержимого файла **Form1.cs**. Обратите внимание на закладку, соответствующую этому отображению. Закладка помечена словом **[Конструктор]**. Это не случайно. Файл отображается визуально. Показано, какие объекты в нем используются. Правда, пока в нём нет никаких объектов, не считая самого окна, но само оно тоже объект.



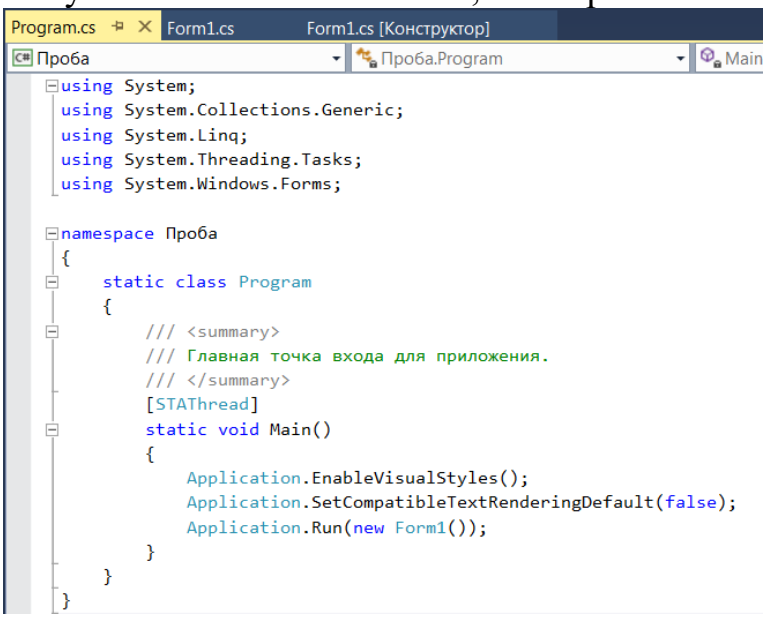
4. Установите мышку на окно и нажмите правую кнопку. Отобразится контекстное меню, связанное с окном. Выберите режим **Перейти к коду**. Появится новая закладка с именем **Form1**, но без пометки **Конструктор**. Отображено содержимое файла **Form1.cs**, но уже не в виде объектов, а в виде программного кода на языке **C#**. Операторов, подключающих к программе системные пространства имён, здесь значительно больше, чем в консольном приложении. Есть уже известное нам пространство имен **System**. Другие строки подключают подпространства имен пространства **System**. Есть среди них, в частности, и пространство **System.Windows.Forms**, которое содержит всё необходимое для разработки окон **Windows**-приложения.



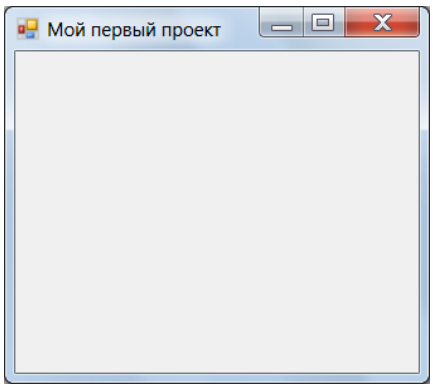
Имеется пространство имен нашей программы – **Проба**. Внутри него имеется описание класса с именем **Form1**. Класс – это структурная единица программы, ее отдельный модуль, где описываются какие-либо алгоритмы. Описания алгоритмов оформлены в виде функций (методов). В составе класса **Form1** уже имеется метод **Form1**(). Это особый метод: его называют конструктором объектов класса.

Конструктор объектов есть в любом классе. В данном случае – это конструктор окна **Form1**. Кроме конструктора в составе класса будут и другие методы, которые напишет программист. Все эти методы будут иметь отношение именно к окну **Form1**.

5. Рассмотрите модуль **Program.cs**. Выберите его мышкой в окне **Обозреватель решения**. Структура этого модуля нами уже изучена. В этом модуле есть класс – **Program**. Он содержит метод **Main()**. Вы помните, что это главный метод программы. Однако, в отличие от консольного приложения здесь этот метод не пуст – в нем уже есть три оператора. Если последовательно навести указатель мышки на каждый оператор, то можно получить сведения о назначении каждого оператора. В целом смысл операторов таков. Два первых оператора разрешают использовать визуальные стили в процессе выполнения программы и задают некоторые стандартные режимы управления программой. Третий оператор обеспечивает запуск процесса, который связан с окном **Form1**. Для этого используется метод **Run** из класса **Application**. В круглых скобках указано имя того объекта, с которым связан процесс.



6. С помощью закладки **Form1.cs [Конструктор]** переключитесь в окно визуального отображения формы **Form1**. Установите курсор мышки на окне, вызовите контекстное меню и выберите **Свойства**. Отобразится дополнительное окно свойств формы. Свойств много, они различны, знакомиться с ними нужно постепенно, по мере надобности. Для начала отметим свойство **Name**, которое определяет идентификатор окна. Фактически это имя переменной: **Form1**. Именно это имя будет использоваться в программном коде при различных обращениях к объекту окно **Form1**.
7. Свойство **Text** содержит заголовок окна – сейчас заголовок совпадает с именем переменной. Это свойство можно изменить, что приведёт к изменению названия окна. Измените значение данного свойства на **Мой первый проект**. При этом на форме изменится заголовок окна.
8. Откомпилируйте приложение, выбрав из главного меню команду **Сборка – Собрать решение**.
9. Для запуска приложения выберите из главного меню команду **Отладка – Начать отладку (F5)**. Приложение запустится в отладочном режиме и на экране появится разработанное окно.

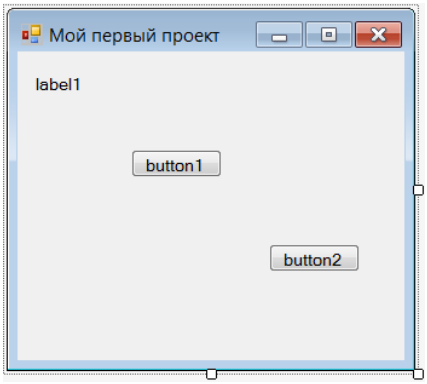


10. Для закрытия окна щелкните мышью на кнопке закрытия.

Задание 2. Создание простого проекта

Разработать программу, которая при нажатии на кнопку **«Hello»** выводит сообщение **«Hello, мир Windows!»**, при нажатии на кнопку **«Выход»** программа завершит работу.

1. Разместите на форме компоненты в соответствии с рисунком. Для этого откройте вкладку **Панель элементов** и разверните список **Стандартные элементы управления**. Затем выберите необходимый элемент управления и щелкните по окну формы. Установите элементы управления в требуемое место на форме с помощью мыши.



2. Выделите кнопку **button2**, перейдите в окно **Свойства**, найдите свойство **Текст** и измените его на **Выход**.
3. Сохраните проект, нажав на соответствующую кнопку на панели инструментов.
4. Для связывания функций кнопки с функцией закрытия окна необходимо создать обработчик события на нажатие кнопки. Для этого сделайте двойной щелчок на кнопке. В результате в коде приложения сформируется шаблон функции обработчика события **Click** для кнопки. В полученный шаблон добавьте функцию закрытия окна.

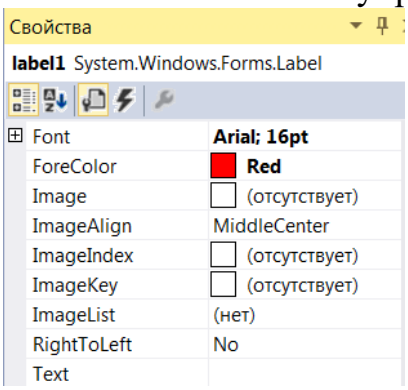
```
private void button2_Click_1(object sender, EventArgs e)
{
    Application.Exit();
}
```

Класс**Application** можно рассматривать как «класс низшего уровня», позволяющий нам управлять поведением приложения **Windows Forms**. Кроме того, этот класс определяет набор событий уровня всего приложения, например закрытие приложения или простой центрального процессора.

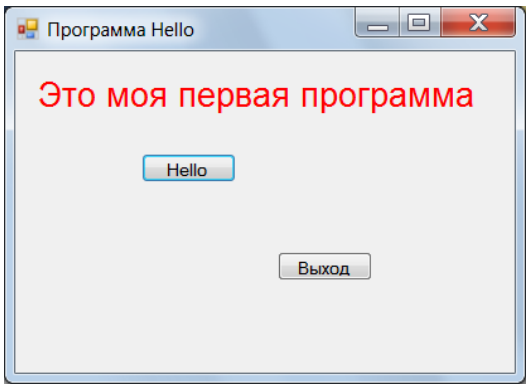
5. Сохраните проект. Запустите проект. Нажмите на кнопку **Выход** и вы вернетесь из режима просмотра программы в режим разработки проекта.
6. Выделите кнопку **button1**, перейдите в окно **Свойства**, найдите свойство **Текст** и измените его на **Hello**.
7. Для связывания функций кнопки с функцией вывода сообщения в соответствующую надпись необходимо создать обработчик события на нажатие кнопки. Для этого сделайте двойной щелчок на кнопке. В результате в коде приложения сформируется шаблон функции обработчика события **Click** для кнопки. В полученный шаблон добавьте необходимые операторы:

```
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = "Hello, мир Windows!";
}
```

8. Сохраните проект еще раз, запустите и протестируйте его, т.е. запустите проект и нажмите кнопку **Hello** – должна появиться ранее написанная вами фраза, нажмите кнопку **Выход**, и программа закроется.
9. Измените содержание выводимой на экран реплики на «**Это моя первая программа**». Для этого дважды щелкните по кнопке **Hello** и измените соответствующий оператор.
10. Задайте шрифт и размер выводимой реплики следующим образом: размер – **16**, цвет – **красный**, гарнитура шрифта - **Arial**. Для этого измените свойство **Font** элемента управления **label1** на панели **Свойства**.
11. Очистите содержимое свойства **Text** элемента управления **label1**.



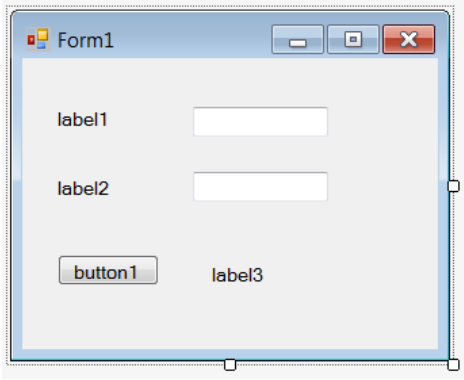
12. Запустите свой проект не в среде **VisualStudio**, а в среде **Windows**. Для этого сверните или закройте **VisualStudio**. Перейдите в папку вашего проекта **Windows1** и найдите в ней папку **Проба – bin – Debug**. Найдите в папке **Debug** файл **Проба.exe**. Щелкните мышкой по этому файлу – файл, как и положено файлу с расширением **.exe**, запустится. Вы увидите, что ваш проект работает в **Windows** без запуска среды **VisualStudio**.
13. Озаглавьте окно проекта «**Программа Hello**». Для этого выделите форму **Form1**, найдите свойство **Text**на панели Свойства и замените надпись на **Программа Hello**.
14. Сохраните изменения и запустите проект на исполнение.



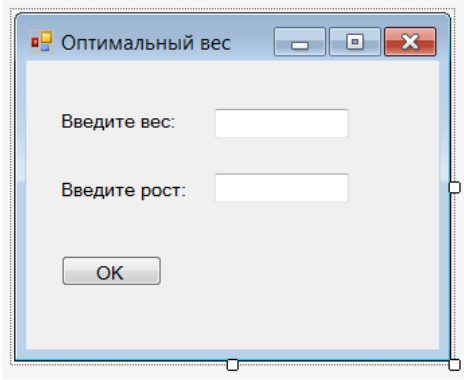
Задание 3. Создание простого проекта

Разработать программу с помощью, которой пользователь, введя свой рост и фактический вес, мог бы определить худой он или полный, и на сколько ему нужно поправиться или похудеть. Для разработки программы воспользуйтесь тем, что оптимальный вес человека определяется так: рост человека минус 100. Если фактический вес человека меньше оптимального, то человек худой, и наоборот, если больше, то нужно похудеть.

1. Создайте в папке своей группы новую папку и назовите её **Weight**.
2. Создайте новое **Приложение Windows Forms**. Имя проекта и приложения – **Weight**. Папка для размещения проекта **Windows1**.
3. Разместите на форме компоненты в соответствии с рисунком. В **TextBox1** будет вводиться вес в кг, а в **TextBox 2** – рост в см.



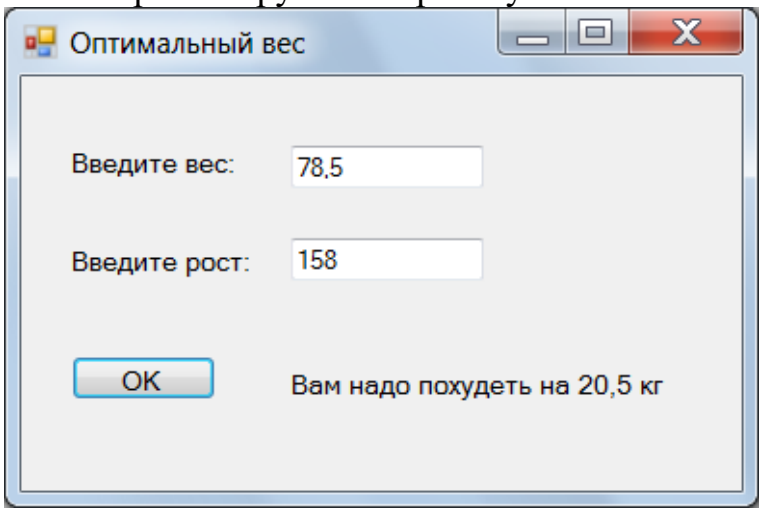
4. Задайте для элементов управления значение свойства **Text** в соответствии с рисунком. Для **label3** значение свойства **Text** оставьте пустым, для элементов управления **TextBox1** и **TextBox2** значение свойства **Text** также оставьте пустым.



5. Выделите кнопку **button1,** дважды щелкните по ней мышью. Оказавшись в коде программы, а точнее, в заготовке процедуры кнопки **button1** заполните её следующим кодом:

```
private void button1_Click(object sender, EventArgs e)
{
    int faktW = Convert.ToInt32(textBox1.Text);
    int Rost = Convert.ToInt32(textBox2.Text);
    int optW = Rost - 100;
    int Delta = Math.Abs(faktW - optW);
    if (optW == faktW)
    {
        label3.Text = "Ваш вес оптимален";
    }
    else
    {
        if (optW > faktW)
        {
            label3.Text = "Вам надо поправится на " + Delta + " кг";
        }
        else
        {
            label3.Text = "Вам надо похудеть на " + Delta + " кг";
        }
    }
}
```

   Вес и рост заданы в объектах **textBox1** и **textBox2** через свойство **Text**, а это строковое значение – тип **string**. Следовательно, эти значения надо преобразовать в целое число. Встроенный класс **Convert** содержит нужные методы преобразования, в частности метод **ToInt32**.
6. Сохраните проект и протестируйте работу приложения.
7. Усовершенствуйте программу так, чтобы можно было бы вводить десятичные величины. Для этого задайте тип переменных не **Int**, а **Double**.
8. Метод **ToDouble**встроенного класса **Convert** преобразует помещенную в скобки переменную типа **string** в переменную вещественного типа. Замените в тексте программного кода **ToInt32** на **ToDouble**.
9. Озаглавьте окно проекта **Оптимальный Вес**.
10. Сохраните проект и протестируйте его работу.



Задания для внеаудиторной самостоятельной работы

- Составьте опорный конспект, содержащий основные элементы рабочей среды **VisualStudio** и их описание.
- Заполните таблицу основных методов **VisualStudio**.

Метод	Назначение
ToDouble()	
ToInt32()	
Abs()	
Exit()	

Критерии оценивания на практическом занятии:

Балл	Уровень освоения компетенций	Критерии оценивания уровня освоения компетенций
5	Максимальный	Практическая работа выполнена в полном соответствии с требованиями, студент представил все задания практической работы без погрешностей и замечаний, на все вопросы при защите практической работы дал правильные ответы.
4	Высокий	Практическая работа выполнена в полном соответствии с требованиями, студент представил все задания практической работы с небольшими погрешностями в выполнении на персональном компьютере, на защите практической работы затруднялся при ответах на некоторые вопросы, нуждался в уточняющих вопросах и подсказках со стороны преподавателя.
3	Средний	Практическая работа выполнена в соответствии с требованиями, студент представил все задания практической работы с существенными погрешностями в выполнении на персональном компьютере, неспособен правильно интерпретировать полученные результаты, на защите затруднялся и/или не ответил на большинство вопросов, нуждался в уточняющих вопросах и подсказках со стороны преподавателя.
2	Ниже среднего	Студент несамостоятельно выполнил практическую работу, неспособен пояснить выполнение ни одного задания, не ответил ни на один контрольный вопрос на защите.

ПРАКТИЧЕСКАЯ РАБОТА №2
СОЗДАНИЕ ПРОЕКТА С ИСПОЛЬЗОВАНИЕМ КНОПОЧНЫХ КОМПОНЕНТОВ

Цель работы: сформировать навыки разработки приложений с использованием кнопочных компонентов в среде программирования **VisualStudio**, изучить особенности их использования.

Формируемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.

В результате освоения практических заданий обучающийся должен

- уметь: использовать языки программирования, строить логически правильные и эффективные программы
- знать: объектно-ориентированную модель программирования, основные алгоритмические конструкции.

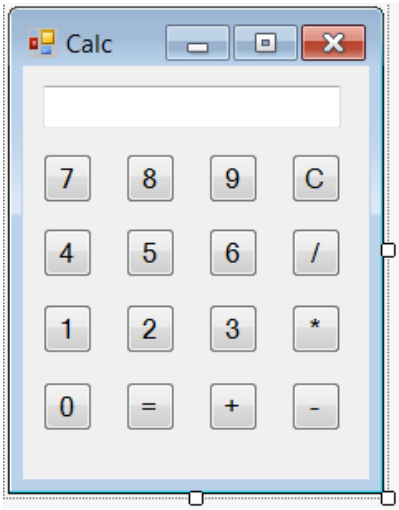
Материально-техническое обеспечение занятия: персональный компьютер, среда программирования **VisualStudio**.

Практические задания и методические указания

Задание 1.

Используя кнопочные компоненты **button,** разработать программу – калькулятор, выполняющий простейшие действия.

1. Запустите среду программирования **VisualStudio**. Создайте новое **Приложение Windows Forms**. Имя проекта и приложения – **Калькулятор**. Папка для размещения проекта **Калькулятор**.
2. Задайте для формы следующие свойства: **Text** – Calc, **Font - Size** – 10.
3. Разместите на форме одно поле ввода, дайте ему имя **Disp** и очистите свойство **Text**.
4. Добавьте на форму 12 кнопок **button** для цифр, арифметических действий, знака «равно» и операции «сброс». Установите для всех кнопок размеры 30 на 30 пикселей и разместите их так, как на рисунке.



5. Дайте кнопкам-действиям имена **btnPlus**, **btnMinus**, **btnMul**, **btnDiv** (сложение, вычитание, умножение и деление), а кнопке «равно» — имя **btnCalc**.
6. Для действий и кнопки **C** установите жирный шрифт (свойство **Font – Bold**), а для кнопки **C** дополнительно — красный цвет шрифта.
7. Кнопка **C** должна просто стирать содержимое поля ввода **Disp**. То есть, нужно вызвать метод **Disp.Clear()**. Добавьте обработчик события **OnClick** для кнопки **C**.
8. Понятно, что когда пользователь щелкнул по кнопке-цифре, нужно добавить эту цифру в конец текста поля ввода. Добавьте обработчик события **OnClick** для кнопки **0**. Добавьте следующий код:

```
{
    Disp.Text += 0;
}
```

9. Добавьте аналогичный код для всех кнопок-цифр.
10. Сохраните программу и протестируйте работу кнопок.
11. При работе программы вы увидели, что с клавиатуры можно ввести буквы, которые нам совсем не нужны. Когда пользователь нажмет клавишу в поле ввода, возникает событие **OnKeyPress**, которое можно перехватить, установив соответствующий обработчик. Создайте обработчик события **OnKeyPress** для поля ввода **Disp**. Для этого перейдите на вкладку **Form1 [Конструктор]**, выделите поле вводы **Disp**, на панели **Свойства** перейдите на вкладку **События**, щелкнув по значку  . Найдите событие **KeyPress** и дважды щелкните мышью в поле справа от него, будет сгенерирована заготовка метода. В тело обработчика события добавьте следующий код:

```
private void Disp_KeyPress(object sender, KeyPressEventArgs e)
{
    if (!char.IsDigit(e.KeyChar))
    {
        e.Handled = true;
    }
}
```

Для проверки ввода используется метод **char.IsDigit**, который возвращает **true**, если введенный символ является десятичной цифрой и **false**, если нет.

Свойство **Handled** используется для определения того, было ли событие обработано. Установив значение **Handled** в **true**, событие ввода не будет передано операционной системе для обработки по умолчанию.

Запустите программу и попробуйте вводит буквы.

12. Далее необходимо организовать вычисления. Нам нужны две переменных для хранения чисел и одна символьная переменная, в которую будем записывать тип операции. Поскольку при расчетах могут получиться числа с дробной частью, для хранения чисел будем использовать вещественные переменные. В простейшем случае для хранения операции можно использовать переменную типа **Char** (один символ), но мы объявим ее как символьную строку, так как при доработке программы могут понадобиться и многосимвольные названия операций. Объявите в начале программы две вещественных переменные **x1** и **x2** типа **Double** и одну символьную строку **oper**.

```
public partial class Form1 : Form
{
    double x1;
    double x2;
    string oper;
```

13. Когда мы нажимаем на одну из кнопок-операций, нужно запомнить введенное число в переменной **x1** и тип операции в переменной **oper**. Тип операции (надпись на кнопке) легко узнать, обратившись к свойству **Text**. Выделите кнопку **+** и создайте для нее обработчик события **OnClick**:

```
private void btnPlus_Click(object sender, EventArgs e)
{
    x1 = Convert.ToDouble(Disp.Text);
    oper = btnPlus.Text;
}
```

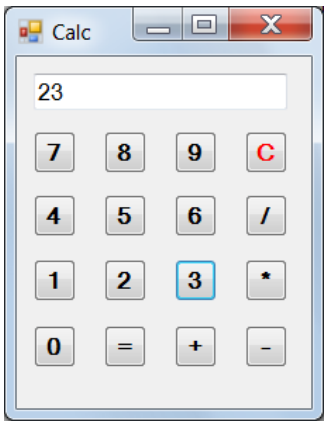
14. Создайте аналогичный обработчик для всех остальных кнопок-операций.
15. При нажатии на кнопку = нужно прочитать из поля ввода второе число и выполнить операцию. При этом первое число и тип операции уже должны находиться в переменных **x1** и **oper**. Поскольку в результате деления может получиться число с дробной частью, переменная для хранения результата (назовем ее **res**) тоже должна быть вещественной. Объявите в начале программы данную переменную.
16. Введите обработчик события **OnClick** для кнопки =:

```
private void btnCalc_Click(object sender, EventArgs e)
{
    double x2 = Convert.ToDouble(Disp.Text);
    if (oper=="+")
    {
        res=x1+x2;
    }
    if (oper == "-")
    {
        res = x1 - x2;
    }
    if (oper == "*")
    {
        res = x1 * x2;
    }
    if (oper == "/")
    {
        res = x1 / x2;
    }
    Disp.Text = Convert.ToString(res);
}
```

17. Запустите программу и проверьте ее работу. Учтите, что для ввода второго числа нужно сначала очистить экран кнопкой «**C**» (позже мы исправим это неудобство).
18. Конечно, очень неудобно, что перед вводом второго числа нужно очищать поле ввода, нажимая на кнопку **C**. Хотелось бы делать это автоматически. Запустите стандартную программу **Калькулятор** и посмотрите, в какой момент стирается первое число. Наверное, вы увидели, что число из поля ввода автоматически стирается, когда после нажатия на кнопки-действия или кнопку = пользователь набирает новое число. Мы введем логическую переменную **newNumber**, которой будем присваивать значение **True** в том случае, если нужно начинать вводить новое число. Объявите логическую переменную **newNumber**.
19. В конце обработчиков события **OnClick** для кнопок-действий и кнопки равно добавьте строку: **newNumber = true;**
20. Очистку экрана будем делать перед вводом нового числа. Дополните обработчики кнопок-цифр. В начало обработчика события **OnClick** для кнопок-цифр добавьте код:

```
if (newNumber)
{
    Disp.Clear();
    newNumber = false;
}
```

21. Запустите программу и проверьте ее работу. После этого закройте проект.



Контрольные задания

1. Составьте программу, которая переводит суммы из рублей в доллары.
2. Составьте программу, которая переводит суммы из рублей в доллары.

Задания для внеаудиторной самостоятельной работы

Составьте опорный конспект по теме. Запишите в тетрадь основные методы, которые используются для обработки событий проекта.

Критерии оценивания на практическом занятии

Балл	Уровень освоения компетенций	Критерии оценивания уровня освоения компетенций
5	Максимальный	Практическая работа выполнена в полном соответствии с требованиями, студент представил все задания практической работы и контрольные задания без погрешностей и замечаний, на все вопросы при защите практической работы дал правильные ответы.
4	Высокий	Практическая работа выполнена в полном соответствии с требованиями, студент представил все задания практической работы и контрольные задания с небольшими погрешностями в выполнении на персональном компьютере, на защите практической работы затруднялся при ответах на некоторые вопросы, нуждался в уточняющих вопросах и подсказках со стороны преподавателя.
3	Средний	Практическая работа выполнена в соответствии с требованиями, студент представил все задания практической работы и контрольные задания с существенными погрешностями в выполнении на персональном компьютере, неспособен правильно интерпретировать полученные результаты, на защите затруднялся и/или не ответил на большинство вопросов, нуждался в уточняющих вопросах и подсказках со стороны преподавателя.
2	Ниже среднего	Студент несамостоятельно выполнил практическую работу, неспособен пояснить выполнение ни одного задания, не ответил ни на один контрольный вопрос на защите.

ПРАКТИЧЕСКАЯ РАБОТА №3
СОЗДАНИЕ ПРОЕКТА С ИСПОЛЬЗОВАНИЕМ КОМПОНЕНТОВ ДЛЯ РАБОТЫ С ТЕКСТОМ

Цель работы: сформировать умения по использованию компонентов для работы с текстом в среде программирования **VisualStudio**, сформировать умения по созданию приложений с компонентами для работы с текстом в среде программирования **VisualStudio**.

Формируемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.

В результате освоения практических заданий обучающийся должен

- уметь: использовать языки программирования, строить логически правильные и эффективные программы
- знать: объектно-ориентированную модель программирования, основные алгоритмические конструкции.

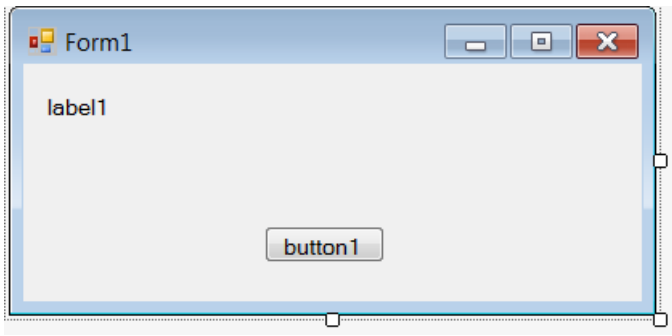
Материально-техническое обеспечение занятия: персональный компьютер, среда программирования **VisualStudio**.

Практические задания и методические указания

Задание 1.

Разработать программу, которая при нажатии на кнопку **«Output»** выводит сообщение **«Моя первая программа на языке C#»**, а затем при повторном нажатии на эту же кнопку сообщение исчезает. При повторном выводе цвет надписи должен быть красным.

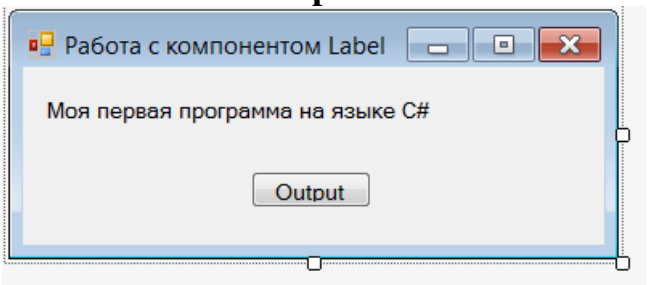
1. Запустите среду программирования **VisualStudio**. Создайте новое **Приложение WindowsForms**. Имя проекта и приложения **Label**. Папка для размещения проекта **Текст**.
2. Разместите на форме компонент **label** и кнопку **button** вкладки панели элементов **Стандартные элементы управления**.



Компоненты ввода — вывода данных можно условно разделить на несколько различных блоков: компоненты вывода текстовой информации на экран; однострочные поля ввода текстовой и числовой информации; многострочные поля ввода.

Для вывода определенной информации на экран, кроме уже ранее используемого компонента **label**, есть и другие компоненты. Текст, который будет отображен, можно задавать как на этапе разработки формы, так и в процессе выполнения программы, присвоив значение свойству **Text**.

3. Задайте для формы заголовок **«Работа с компонентом Label»**.
4. Выделите надпись **label1**, найдите на панели **Свойства** свойство **Text** и вставьте новое название надписи **Моя первая программа на языке C#.**
5. Выделите кнопку **button1**, найдите на панели **Свойства** свойство **Text** и вставьте новое название кнопки **Output**.

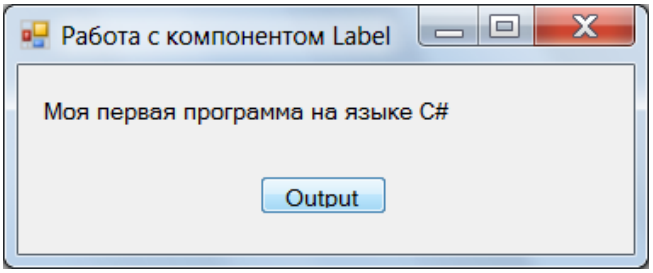


6. Перейдите на панели **Свойства** на страницу **События**, найдите событие **Click** и справа от него дважды щелкните мышкой. Оказавшись в коде программы, но теперь в процедуре кнопки **Button1**, напишите следующий программный код:

```
private void button1_Click(object sender, EventArgs e)
{
    label1.Visible = !label1.Visible;
    if (label1.Visible)
    {
        label1.ForeColor = System.Drawing.Color.Red;
    }
}
```

В этой программе при каждом очередном нажатии происходит изменение свойства **Visible**, вследствие чего надпись то появляется, то исчезает с экрана, а также происходит изменение свойства **ForeColor**.

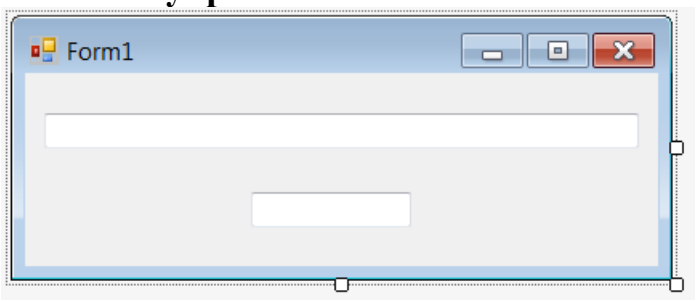
7. Сохраните изменения и запустите проект. Протестируйте его работу.



Задание 2.

Разработать программу, которая при вводе текста в первый компонент **textBox1**, во втором компоненте **textBox 2** отображает реальную длину вводимой строки. Кроме этого, при выходе из компонента **textBox 1** его содержимое копируется в буфер обмена и удаляется, а при возвращении в программу появляется снова.

1. Создайте новое **Приложение WindowsForms**. Имя проекта и приложения **textBox1**. Папка для размещения проекта **Текст**.
2. Разместите на форме два компонент **textBox** вкладки панели элементов **Стандартные элементы управления**.



3. Задайте для формы заголовок **«Работа с компонентом textBox»**.
4. Выделите текстовое поле **textBox1**, найдите на панели **Свойства** свойство **Text** и оставьте его пустым. Аналогичные действия выполните со вторым текстовым полем.
5. Выделите компонент **textBox1**, на панели **Свойства** перейдите на вкладку **События** и найдите событие **TextChanged**и справа от него дважды щелкните мышкой. Оказавшись в коде программы, но теперь в процедуре текстового поля **textBox1**, напишите следующий программный код:

```
private void textBox1_TextChanged(object sender, EventArgs e)
{
    textBox2.Text = Convert.ToString(textBox1.Text.Length);
}
```

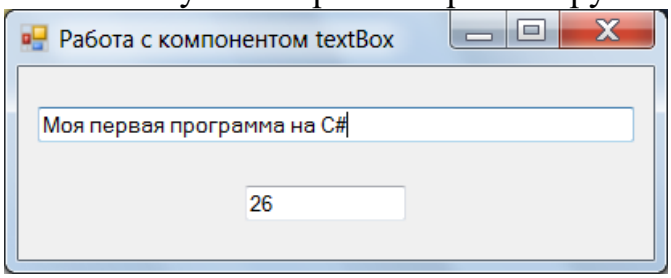
6. Выделите компонент **textBox1**, на панели **Свойства** перейдите на вкладку **События** и найдите событие **Enter,** справа от него дважды щелкните мышкой. Оказавшись в коде программы, но теперь в процедуре текстового поля **textBox1**, напишите следующий программный код:

```
private void textBox1_Enter(object sender, EventArgs e)
{
    textBox1.Paste();
}
```

7. Выделите компонент **Edit1**, на панели **Свойства** перейдите на вкладку **События** и найдите событие **Leave**и справа от него дважды щелкните мышкой. Оказавшись в коде программы, но теперь в процедуре текстового поля **Edit1**, напишите следующий программный код:

```
private void textBox1_Leave(object sender, EventArgs e)
{
    textBox1.SelectAll();
    textBox1.Copy();
    textBox1.Clear();
}
```

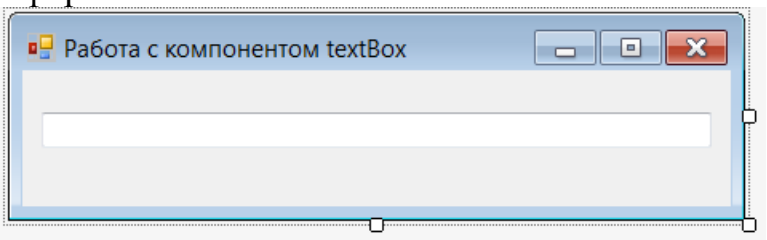
8. Сохраните изменения и запустите проект. Протестируйте его работу.



Задание 3.

Разработать программу, которая запрещает ввод в компонент **textBox1** подряд двух одинаковых символов.

1. Создайте новое **Приложение WindowsForms**. Имя проекта и приложения **textBox2**. Папка для размещения проекта **Текст**.
2. Разместите на форме компонент **textBox1**.



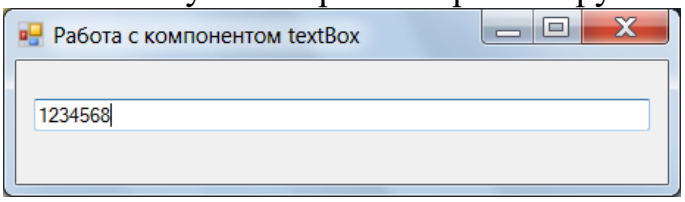
3. Перейдите в код программы. Объявите глобальную переменную **ch** типа **char**, в которой будет храниться последний нажатый символ.

```
protected char ch;
private void textBox1_KeyPress(object sender, KeyPressEve
```

4. Задайте для формы заголовок **«Работа с компонентом textBox»**.
5. Выделите текстовое поле **textBox1**, найдите на панели **Свойства** свойство **Text** и оставьте его пустым.
6. Создайте процедуру обработки события **KeyPress** текстового поля **textBox1**, параметр **Key** данной процедуры содержит символ нажатой клавиши. Если вновь введенный символ совпадает с только что нажатым символом, то он игнорируется. В противном случае, новый символ запоминается в переменной **ch**.

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
   if (ch==e.KeyChar)
   {
       e.KeyChar = Convert.ToChar(0);
   }
   else
   {
       ch = e.KeyChar;
   }
}
```

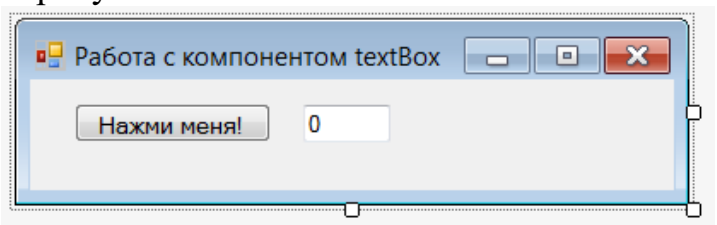
7. Сохраните изменения и запустите проект. Протестируйте его работу.



Задание 4.

Разработать программу, которая считает количество нажатий на кнопку и выдает это значение в компоненте **textBox**.

1. Создайте новое **Приложение WindowsForms**. Имя проекта и приложения **textBox3**. Папка для размещения проекта **Текст**.
2. Разместите на форме компонент **textBox** и кнопку **button**.
3. Используя панель **Свойства,** задайте значения свойств компонентов формы в соответствии с рисунком.



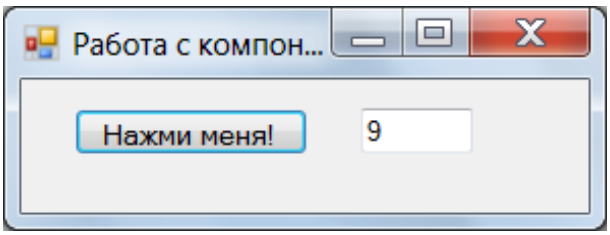
4. Если в целочисленной переменной **i** будем считать количество нажатий, то процедура обработки события **Click** кнопки может быть записана в виде:

```
private void button1_Click(object sender, EventArgs e)
{
    i = i + 1;
    textBox1.Text = Convert.ToString(i);
}
```

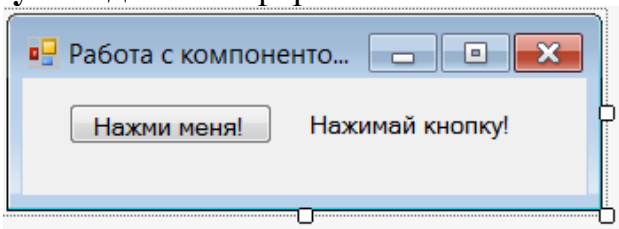
5. Однако остается вопрос, где описывать данную переменную **i**. Если сделать это внутри данной процедуры, то также необходимо осуществлять обнуление переменной, а это приведет к получению одного и того результата, равного единице. Следовательно, переменная **i** должна быть глобальной переменной. Перейдите в код программы опишите глобальную переменную **i** типа **int**, в которой будет храниться последний нажатый символ.

```
protected int i;
private void button1_Click(object sender, EventArgs e)
```

8. Сохраните изменения и запустите проект. Протестируйте его работу.



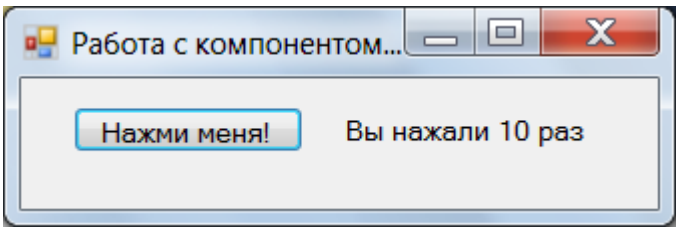
9. Данную программу можно легко модифицировать так, чтобы после определенного количества нажатий появлялось некоторое сообщение или кнопка блокировалась, или приложение автоматически закрывалось. Результат можно отображать не только посредством компонента **textBox**, но и через не редактируемый текст, т. е. компонент **label**, что в данном случае является более естественным.
   Свойству **Visible** компонента **label** присваиваем **False**, т. е. при открытии формы надпись отражаться не будет. Затем, как и ранее, при нажатии на кнопку переменная **i** увеличивается на **1**. Когда значение переменной **i** будет равно **10, 20, 30** или **40** компонент **label** становится видимым, а свойству **Text** надписи присваиваем значение «**Вы нажали i раз**». При следующем нажатии надпись становится невидима. Когда **i** станет равной **50**, кнопку необходимо сделать неактивной, для чего необходимо изменить значение свойства **Enabled** с **True** — включено на **False** — выключено.
10. Разместите на форме компонент **label**. Задайте свойство **Text** равным «**Нажимай кнопку!**». Удалите с формы компонент **textBox1**.

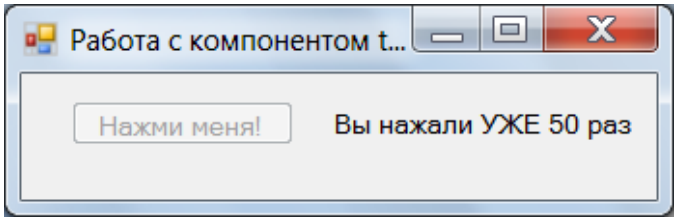


11. Перейдите в окно редактирования кода и внесите изменения в код процедуры обработки события **Click** кнопки.

```
private void button1_Click(object sender, EventArgs e)
{
    label1.Visible = false;
    i = i + 1;
    if ((i==10)||(i==20)||(i==30)||(i==40))
    {
        label1.Visible = true;
        label1.Text = "Вы нажали " + Convert.ToString(i) + " раз";
    }
    if (i==50)
    {
        label1.Visible = true;
        label1.Text = "Вы нажали УЖЕ " + Convert.ToString(i) + " раз";
        button1.Enabled = false;
    }
}
```

12. Сохраните проект и протестируйте работу приложения.

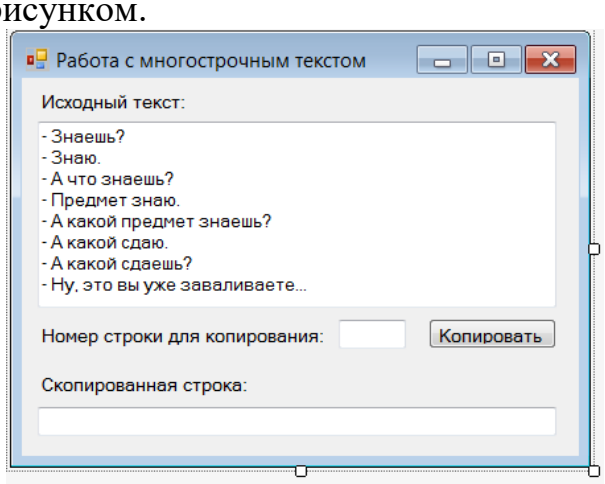




Задание 5.

Разработать программу, которая считывает строку под определенным номером и помещает её в текстовое поле.

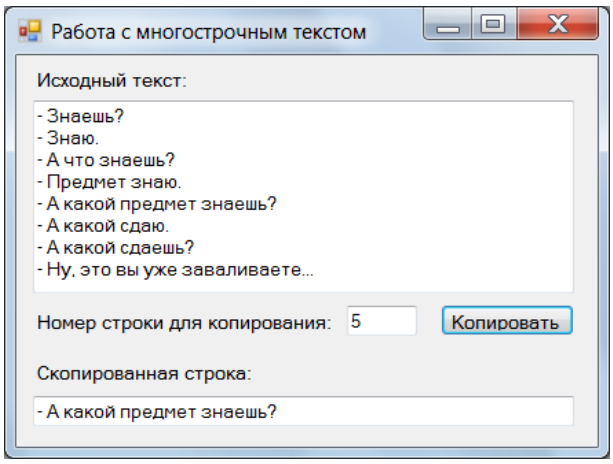
1. Создайте новое **Приложение WindowsForms**. Имя проекта и приложения **textBox4**. Папка для размещения проекта **Текст**.
2. Для ввода или вывода многострочного текста, воспользуемся компонентом **textBox**. строк. Для возможности доступа к строкам вместо свойства **Text** у него имеется свойство **Lines**, при выборе которого во время проектирования задается начальное значение строк.
   Для доступа к строкам во время выполнения программы также используется свойство **Lines**. Разместите на форме компоненты **textBox,** компоненты **label**, кнопку **button**.
3. Используя панель **Свойства,** задайте значения свойств компонентов формы в соответствии с рисунком.



4. Для реализации решения задачи процедура обработки события **Click** кнопки может быть записана в виде:

```
private void button1_Click(object sender, EventArgs e)
{
    textBox3.Text = textBox1.Lines[Convert.ToInt32(textBox2.Text) - 1];
}
```

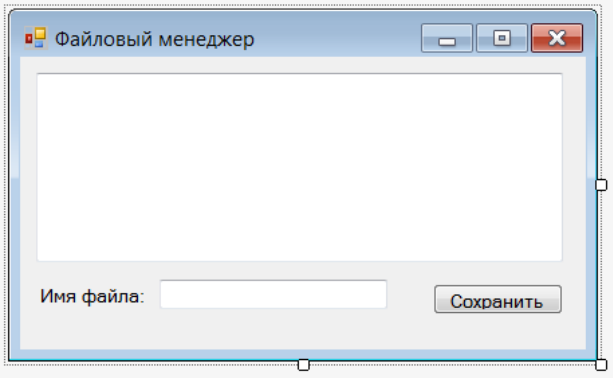
5. Сохраните проект и протестируйте работу приложения.



Задание 6.

Разработать программу, которая сохраняет текст, набранный в поле **textBox1** в файл, имя которого задано в текстовом поле **textBox2**.

1. Создайте новое **Приложение WindowsForms**. Имя проекта и приложения **textBox5**. Папка для размещения проекта **Текст**.
2. Разместите на форме компоненты **textBox,** компонент l**Label**, кнопку **button**.
3. Используя панель **Свойства,** задайте значения свойств компонентов формы в соответствии с рисунком.



4. Поскольку мы будем работать с файлами, то необходимо выполнить подключение пространства имен для работы с файлами:

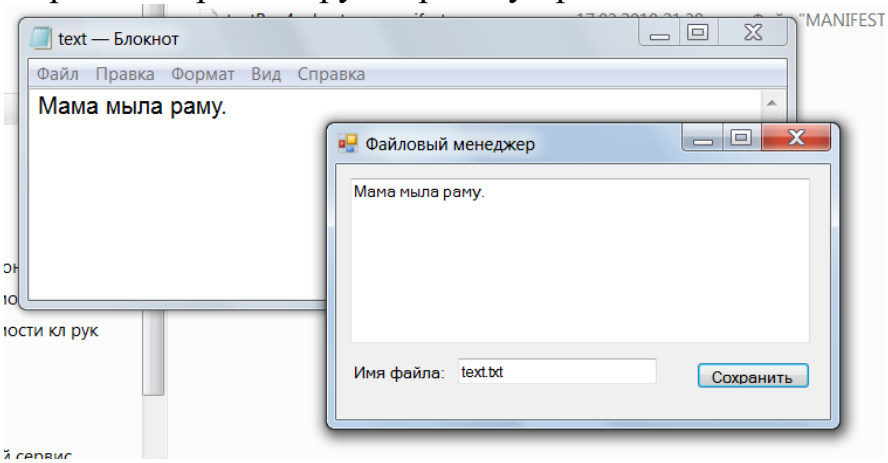
```
using System.Windows.Forms;
using System.IO;

namespace textBox4
{
```

5. Процедура обработки события **Click** кнопки **button** будет иметь следующий вид:

```
private void button1_Click(object sender, EventArgs e)
{
    StreamWriter write_text;
    FileInfo file = new FileInfo(textBox3.Text);
    write_text = file.AppendText();
    write_text.WriteLine(textBox1.Text);
    write_text.Close();
}
```

Первой строкой в процедуре определяется класс для записи в файл. Далее запись информации в файл с именем заданным в **textBox2**. Если такого файла не существует, то он будет создан на диске.

6. Сохраните проект и протестируйте работу приложения.



Задания для внеаудиторной самостоятельной работы

Составьте опорный конспект по свойствам и методам основных компонентов для работы с текстом.

Критерии оценивания на практическом занятии

Балл	Уровень освоения компетенций	Критерии оценивания уровня освоения компетенций
5	Максимальный	Практическая работа выполнена в полном соответствии с требованиями, студент представил все задания практической работы без погрешностей и замечаний, на все вопросы при защите практической работы дал правильные ответы.
4	Высокий	Практическая работа выполнена в полном соответствии с требованиями, студент представил все задания практической работы с небольшими погрешностями в выполнении на персональном компьютере, на защите практической работы затруднялся при ответах на некоторые вопросы, нуждался в уточняющих вопросах и подсказках со стороны преподавателя.
3	Средний	Практическая работа выполнена в соответствии с требованиями, студент представил все задания практической работы с существенными погрешностями в выполнении на персональном компьютере, неспособен правильно интерпретировать полученные результаты, на защите затруднялся и/или не ответил на большинство вопросов, нуждался в уточняющих вопросах и подсказках со стороны преподавателя.
2	Ниже среднего	Студент несамостоятельно выполнил практическую работу, неспособен пояснить выполнение ни одного задания, не ответил ни на один контрольный вопрос на защите.

ПРАКТИЧЕСКАЯ РАБОТА №4
СОЗДАНИЕ ПРОЕКТА С КОНТЕЙНЕРНЫМИ ЭЛЕМЕНТАМИ УПРАВЛЕНИЯ

Цель работы: сформировать умения использования контейнерных элементов управления при создании проекта **VisualStudio**, изучить их основные свойства; сформировать умения по созданию процедур на основе событий компонентов.

Формируемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.

В результате освоения практических заданий обучающийся должен

- уметь: использовать языки программирования, строить логически правильные и эффективные программы
- знать: объектно-ориентированную модель программирования, основные алгоритмические конструкции.

Материально-техническое обеспечение занятия: персональный компьютер, среда программирования **VisualStudio**.

Практические задания и методические указания

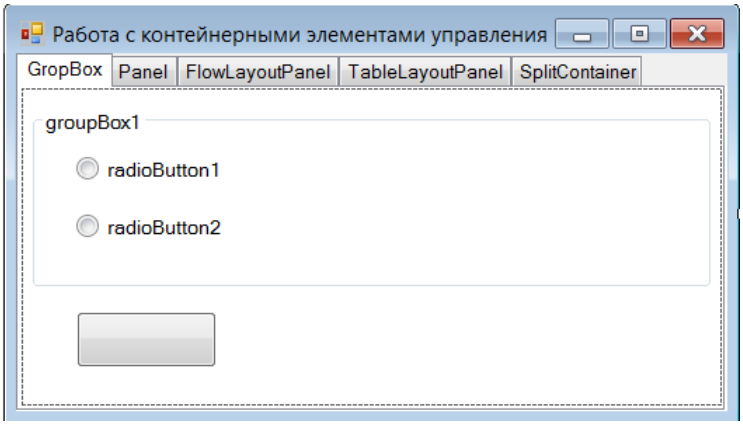
Задание 1.

Создать проект с возможностью группировки элементов на вкладках.

1. Запустите среду программирования **VisualStudio**. Создайте новое **Приложение WindowsForms**. Имя проекта и приложения **Zad1**. Папка для размещения проекта **КонтейнерЭлементы**.
2. Контейнерные элементы управления – это специализированные элементы управления, выступающие в роли настраиваемого вместилища для других элементов управления. К контейнерным элементам управления относятся **Panel** и **GroupBox**. Они представляют форме логические и физические подразделы, которые могут группировать другие элементы управления в единообразные подгруппы пользовательского интерфейса. Перетащите на форму элемент управления **TabControl** с вкладки **Контейнеры** панели элементов. На панели **Свойства** задайте свойству **Dock** значение **Fill**.
3. На панели **Свойства** выберите свойство **TabPages**, чтобы открыть **Редактор коллекцииTabPage**. Добавьте вкладки так, чтобы их стало всего пять. За-

дайте свойствам **Text** этих пяти элементов управления **TabPage** значения **GropBox**, **Panel**, **FlowLayoutPanel,** **TableLayoutPanel** и **SplitContainer**. Щелкните **ОК**.

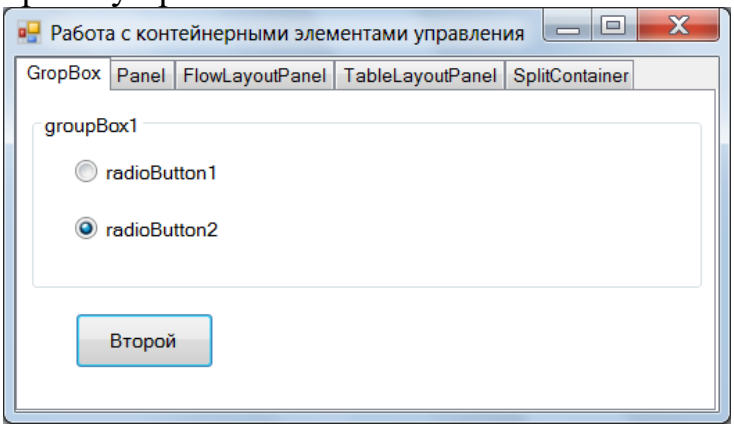
4. В форме выберите вкладку **GroupBox**. Перетащите элемент управления **groupBox** с вкладки **Контейнеры** панели элементов в элемент управления **TabPage**.
5. Перетащите в **GroupBox** два элемента управления **radioButton** с вкладки **Стандартные элементы управления** панели элементов.
6. Добавьте на вкладку **GroupBox** вне элемента управления **groupBox** кнопку **button**. Свойство **Text** кнопки задайте пустым, а свойству **Name** задайте значение **but**.



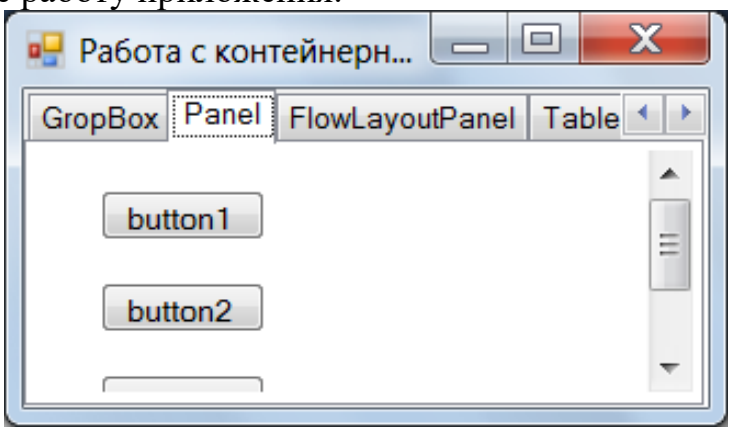
7. Дважды щелкните мышью по кнопке и добавьте код обработчика события **Click** установки надписи на кнопке в зависимости от выбранного переключателя:

```
private void button1_Click(object sender, EventArgs e)
{
    if (radioButton1.Checked==true)
    {
        this.but.Text = "Первый";
    }
    else
    {
        if (radioButton2.Checked==true)
        {
            this.but.Text = "Второй";
        }
    }
}
```

8. Протестируйте работу приложения.



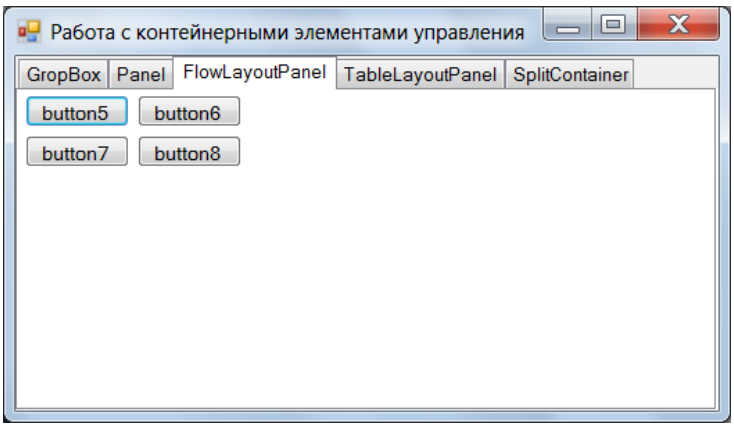
9. Выберите на форме вкладку **Panel**. Перетащите элемент управления **Panel** с вкладки **Контейнеры** панели элементов в элемент управления **TabPage**. Для элемента **Panel** задайте свойству **Dock** значение **Fill**.
10. Перетащите в элемент управления **Panel** четыре элемента управления **button** с вкладки **Стандартные элементы управления** панели элементов.
11. Выделите элемент **Panel** на панели **Свойства** найдите свойство **AutoScroll** и установите значение **True**. в этом случае элемент управления **Panel** будет отображать полосы прокрутки, если элементы находятся за пределами видимых границ.
12. Протестируйте работу приложения.



13. Выберите на форме вкладку **FlowLayoutPanel.** Перетащите элемент управления **FlowLayoutPanel**с вкладки **Контейнеры** панели элементов в элемент управления **TabPage**. Для элемента **FlowLayoutPanel** задайте свойству **Dock** значение **Fill**.
14. Перетащите в элемент управления **FlowLayoutPanel** четыре элемента управления **button** с вкладки **Стандартные элементы управления** панели элементов. Обратите внимание на размещение добавляемых элементов: по умолчанию порядок следования элементов управления в **FlowLayoutPanel** – слева направо. Это значит, что элементы управления, расположенные в **FlowLayoutPanel**, будут находиться в левом верхнем углу и размещаться вправо до тех пор, пока не достигнут края панели. Такое поведение контролируется свойством **FlowDirection**, которому может быть задано четыре значения заполнения в **FlowLayoutPanel: LeftToRight –** по умолчанию, **RightToLeft** – справа налево, **TopDown** – сверху вниз, **BottomUp** – снизу вверх.
15. Дважды щелкните кнопку **button5** и добавьте в обработчик события **Click** следующий код:

```
private void button5_Click(object sender, EventArgs e)
{
    flowLayoutPanel1.SetFlowBreak(button6, true);
}
```

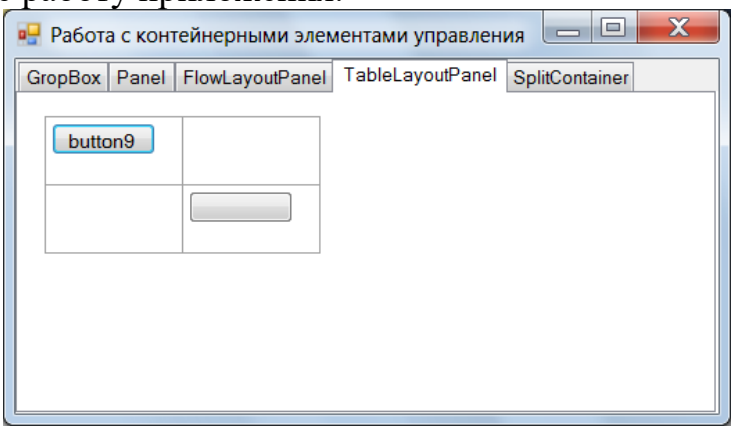
16. Протестируйте работу приложения.



17. Выберите на форме вкладку **TableLayoutPanel.** Перетащите элемент управления **TableLayoutPanel**с вкладки **Контейнеры** панели элементов в элемент управления **TabPage**. Задайте свойству **CellBorderStyle** значение **Inset**, а свойству **AutoScroll** значение **True**.
18. Перетащите в левую верхнюю ячейку элемента управления **TableLayoutPanel**элемента управления **button** с вкладки **Стандартные элементы управления** панели элементов.
19. Дважды щелкните **button9** и добавьте в обработчик события **Click** следующий код:

```
private void button9_Click(object sender, EventArgs e)
{
    Button aButton = new Button();
    tableLayoutPanel1.Controls.Add(aButton, 1, 1);
}
```

20. Протестируйте работу приложения.



21. Выберите на форме вкладку **SplitContainer.** Перетащите элемент управления **SplitContainer**с вкладки **Контейнеры** панели элементов в элемент управления **TabPage**. Задайте свойству **BorderStyle** значение **Fixed3D**.
22. Перетащите два элемента управления **button** с вкладки **Стандартные элементы управления** панели элементов в **Panel1**. Задайте свойствам **Text** этих кнопок значения **Fix/UnfixPanel1** и **Fix/UnfixSplitter.** Измените размеры кнопок так, чтобы отображался текст.
23. Добавьте кнопку в **Panel2** и задайте свойству **Text** значение **Collapse/UncollapsePanel1**. Измените размеры кнопки так, чтобы отображался текст.

24. Дважды щелкните кнопку **Fix/UnfixPanel1** и добавьте в обработчик события **Click** следующий код:

```
private void button10_Click(object sender, EventArgs e)
{
    if (splitContainer1.FixedPanel==FixedPanel.Panel1)
    {
        splitContainer1.FixedPanel = FixedPanel.None;
    }
    else
    {
        splitContainer1.FixedPanel = FixedPanel.Panel1;
    }
}
```

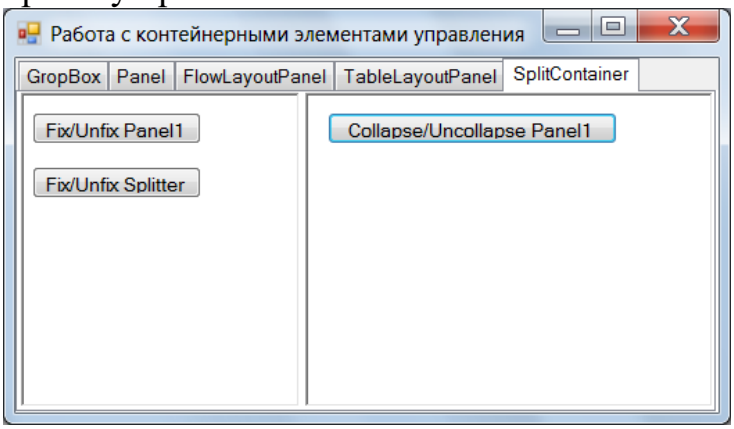
25. Дважды щелкните кнопку **Fix/UnfixSplitter**и добавьте в обработчик события **Click** следующий код:

```
private void button11_Click(object sender, EventArgs e)
{
    splitContainer1.IsSplitterFixed = !(splitContainer1.IsSplitterFixed);
}
```

26. Дважды щелкните кнопку **Collapse/UncollapsePanel1** и добавьте в обработчик события **Click** следующий код:

```
private void button12_Click(object sender, EventArgs e)
{
    splitContainer1.Panel1Collapsed = !(splitContainer1.Panel1Collapsed);
}
```

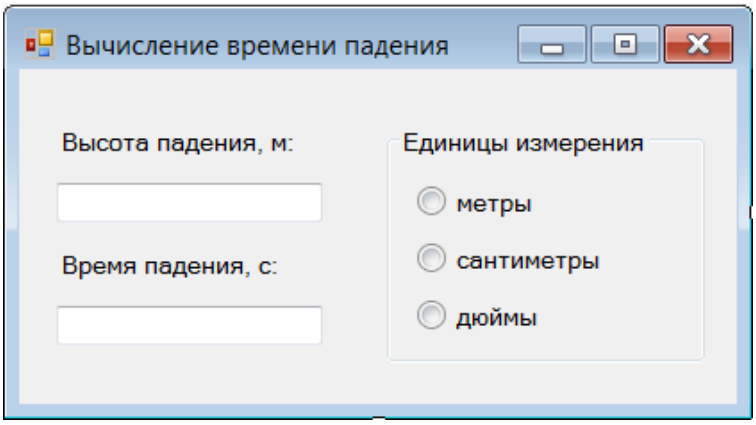
27. Протестируйте работу приложения.



Задание 2.

Разработать приложение, с помощью которого можно вычислить время падения тела с некоторой высоты при условии, что высота может задаваться в метрах, сантиметрах и дюймах.

1. Создайте новое **Приложение WindowsForms**. Имя проекта и приложения **Zad2**. Папка для размещения проекта **КонтейнерЭлементы**.
2. Разместите на форме компоненты в соответствии с рисунком.



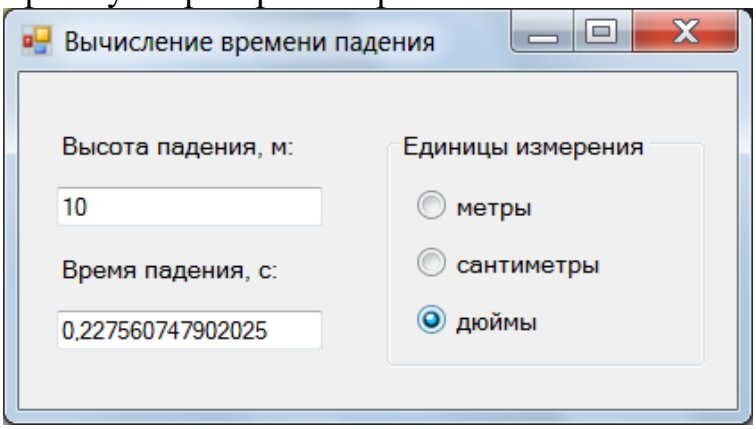
3. Для группы переключателей создайте свой собственный обработчик события **CheckedChanged**. Это событие возникает, когда пользователь изменяет состояние флажка, устанавливая или снимая отметку. Для того чтобы создать обработчик событий **rbChanged** для группы переключателей, отвечающих за выбор единицы измерения в расчете, необходимо выделить первый переключатель в группе. Затем на панели **Свойства** открыть вкладку **События**, в поле **CheckedChanged** ввести строку **rbChanged** и нажать клавишу **Enter** на клавиатуре. В результате будет создано тело обработчика событий **rbChanged**.
4. Введите следующий код в построенную заготовку обработчика:

```
{
    double g = 9.81;
    double h;
    double t;
    h = Convert.ToDouble(textBox1.Text);
    RadioButton rb = (RadioButton)sender;
    switch (rb.TabIndex)
    {
        case 0:
            {
                t = Math.Sqrt(2 * h / g);
                textBox2.Text = Convert.ToString(t);
                break;
            }
        case 1:
            {
                t = Math.Sqrt(2 * h /100/ g);
                textBox2.Text = Convert.ToString(t);
                break;
            }
        case 2:
            {
                t = Math.Sqrt(2 * h*2.54/100 / g);
                textBox2.Text = Convert.ToString(t);
                break;
            }
    }
}
```

Данный код работает следующим образом: вначале он сохраняет в переменной **rb** идентификатор переключателя, состояние которого было изменено.

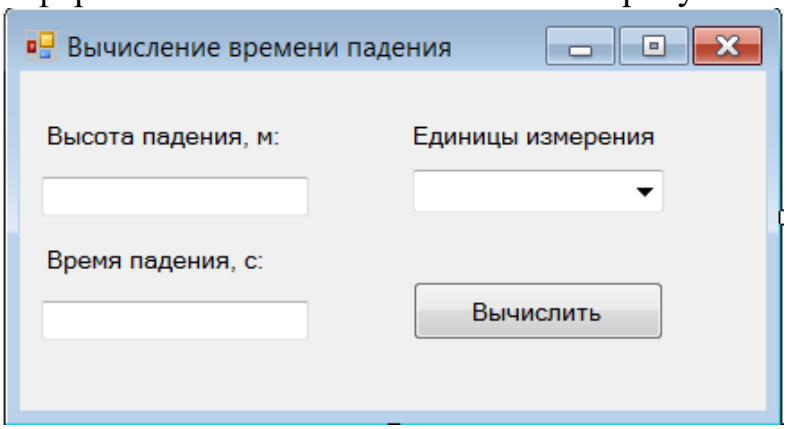
Затем обработчики извлекает номер переключателя в группе. В зависимости от этого номера, обработчик выполняет расчет времени падения.

5. Выделите по очереди остальные переключатели и на вкладке События панели Свойства для события **CheckedChanged** выберите из списка событие **rbChanged.**
6. Запустите программу и проверьте её работоспособность.



Недостаток компонента **RadioButton** заключается в том, что имеется возможность определить только номер выбранной альтернативы, а не ее текстовое содержание. Для того чтобы определить и текстовое содержание альтернативы, можно использовать комбинированную строку, компонент **ComboBox**. Комбинированная строка ввода объединяет в себе свойство строки и списка. В обычном состоянии она имеет вид строки **TextBox** со стоящей рядом кнопкой с изображением направленной вниз стрелки. Если нажать эту кнопку, то появится список строк, где можно выбрать произвольную.

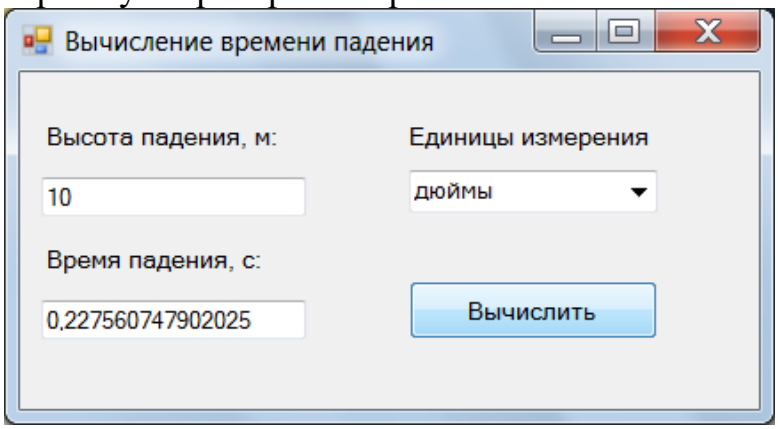
7. Создайте новое **Приложение WindowsForms**. Имя проекта и приложения **Zad3**. Папка для размещения проекта **КонтейнерЭлементы**.
8. Разместите на форме компоненты в соответствии с рисунком:



9. Выделите элемент управления **comboBox1** на панели **Свойства** найдите свойство **Items**, щелкните по кнопке ... и в диалоговом окне **Редактор коллекции строк** введите строки с единицами измерения: **метры, сантиметры, дюймы**.
10. Создайте обработчик события **Click** кнопки **Вычислить** и вставьте следующий код:

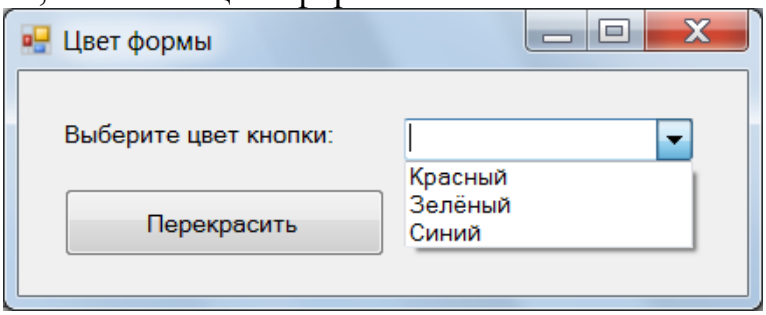
```
private void button1_Click(object sender, EventArgs e)
{
    double g = 9.81;
    double h;
    double t;
    h = Convert.ToDouble(textBox1.Text);
    if (comboBox1.Text=="метры")
    {
        t = Math.Sqrt(2 * h / g);
        textBox2.Text = Convert.ToString(t);
    }
    if (comboBox1.Text == "сантиметры")
    {
        t = Math.Sqrt(2 * h /100/ g);
        textBox2.Text = Convert.ToString(t);
    }
    if (comboBox1.Text == "дюймы")
    {
        t = Math.Sqrt(2 * h *2.54/100/ g);
        textBox2.Text = Convert.ToString(t);
    }
}
```

11. Запустите программу и проверьте её работоспособность.

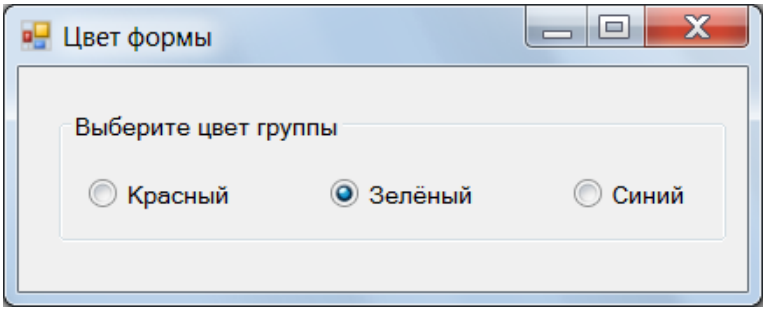


Контрольные задания

1. Разработайте приложение, которое при выборе определенного цвета в выпадающем списке, изменяет цвет формы.



2. Разработайте приложение, которое при выборе определенного цвета в группе переключателей, изменяет цвет формы.



Задания для внеаудиторной самостоятельной работы

Составьте письменно в тетради опорный конспект по основным командам для работы с контейнерными компонентами.

Критерии оценивания на практическом занятии

Балл	Уровень освоения компетенций	Критерии оценивания уровня освоения компетенций
5	Максимальный	Практическая работа выполнена в полном соответствии с требованиями, студент представил все задания практической работы и контрольные задания без погрешностей и замечаний, на все вопросы при защите практической работы дал правильные ответы.
4	Высокий	Практическая работа выполнена в полном соответствии с требованиями, студент представил все задания практической работы и контрольные задания с небольшими погрешностями в выполнении на персональном компьютере, на защите практической работы затруднялся при ответах на некоторые вопросы, нуждался в уточняющих вопросах и подсказках со стороны преподавателя.
3	Средний	Практическая работа выполнена в соответствии с требованиями, студент представил все задания практической работы и контрольные задания с существенными погрешностями в выполнении на персональном компьютере, неспособен правильно интерпретировать полученные результаты, на защите затруднялся и/или не ответил на большинство вопросов, нуждался в уточняющих вопросах и подсказках со стороны преподавателя.
2	Ниже среднего	Студент несамостоятельно выполнил практическую работу, неспособен пояснить выполнение ни одного задания, не ответил ни на один контрольный вопрос на защите.

ПРАКТИЧЕСКАЯ РАБОТА №5
СОЗДАНИЕ ПРОЕКТА С ИСПОЛЬЗОВАНИЕМ ПОЛОС ПРОКРУТКИ ДЛЯ ВВОДА ИНФОРМАЦИИ

Цель работы: сформировать умения использования полос прокрутки для ввода информации, изучить основные свойства; сформировать умения по созданию процедур на основе событий компонента.

Формируемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.

В результате освоения практических заданий обучающийся должен

- уметь: использовать языки программирования, строить логически правильные и эффективные программы
- знать: объектно-ориентированную модель программирования, основные алгоритмические конструкции.

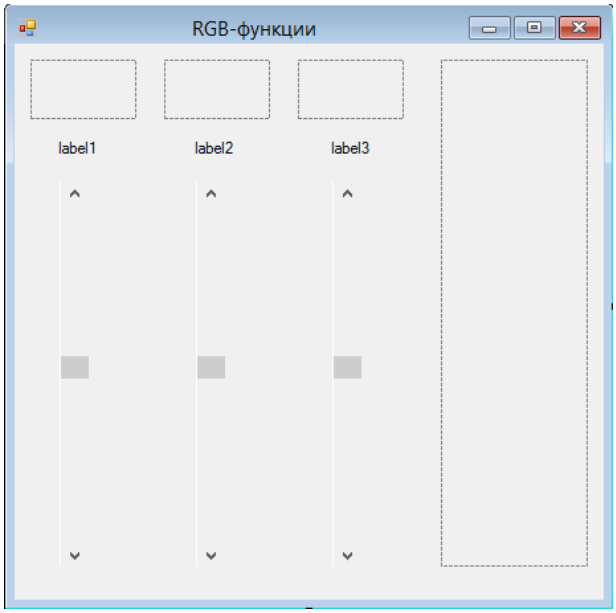
Материально-техническое обеспечение занятия: персональный компьютер, среда программирования **Visual Studio**.

Практические задания и методические указания

Задание 1.

Разработать проект демонстрации работы RGB – функций (установок цвета по трем составляющим) с помощью полос прокрутки. Каждый бегунок полос прокрутки должен будет менять вклад RGB – компонента, отображающийся на панели как цвет, а на метке как число. Результирующий цвет должен отображаться на панели.

1. Запустите среду программирования **VisualStudio**. Создайте новое **Приложение WindowsForms**. Имя проекта и приложения **ScrollBox1**. Папка для размещения проекта **ScrollBox**.
2. Разместите на форме компоненты в соответствии с рисунком.



Элементы управления **HScrollBar** и **VScrollBar** представляют собой обычные полосы прокрутки, отображаемые на границах окон редактирования и просмотра приложений **MicrosoftWindows**. **Полоса прокрутки** – классический элемент оконного интерфейса, использующийся для скроллинга информации, которая не помещается целиком в область вывода. В объектах просмотра и редактирования **Windows** этот элемент появляется при необходимости автоматически.

Окно элемента управления **HScrollBar** располагается горизонтально, а элемента управления **VScrollBar**— вертикально.

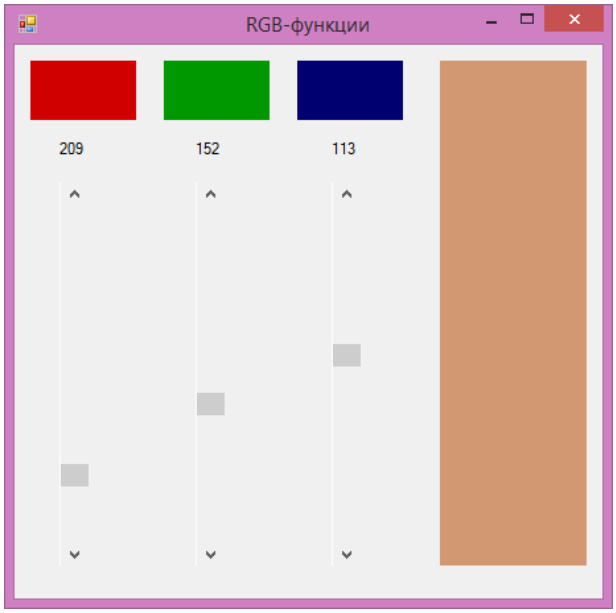
5. Задайте для элемента управления **VScrollBar1** значение свойства **Name=RedBar** и установить значение свойств **Max=255**, **Value=122**, **LargeChange=1**.
6. Задайте для элемента управления **VScrollbar2** значение свойства **Name=GreenBar** и установите значение свойств **Max=255**, **Value=122**, **LargeChange=1**.
7. Задайте для элемента управления **VScrollbar3** значение свойства **Name=BlueBar** и установите значение свойств **Max=255**, **Value=122**, **LargeChange=1**.
8. Выделите элемент **RedBar,** на панели **Свойства** и перейдите на вкладку **События**. Найдите событие **Scroll,** справа от него в поле сделайте двойной щелчок левой кнопкой мыши. Оказавшись в коде программы, введите следующий код:

```
private void RedBar_Scroll(object sender, ScrollEventArgs e)
{
    panel1.BackColor = Color.FromArgb(Convert.ToInt32(RedBar.Value), 0, 0);
    label1.Text = Convert.ToString(RedBar.Value);
    panel4.BackColor = Color.FromArgb(Convert.ToInt32(RedBar.Value), Convert.ToInt32(GreenBar.Value), Convert.ToInt32(BlueBar.Value));
}
```

Метод **FromArgb**создает структуру **Color** из указанных 8-разрядных значений цветов (красный, зеленый, синий).

9. Аналогично запишите процедуры обработки события **Scroll** для элементов **GreenBar** и **BlueBar**.
10. Задайте имя формы проекта **RGB-функции**.

11. Свойство **Text** компонентов **Label** сделайте пустым.
12. Сохраните изменения и запустите проект, убедитесь в его работоспособности.



Задание 2.

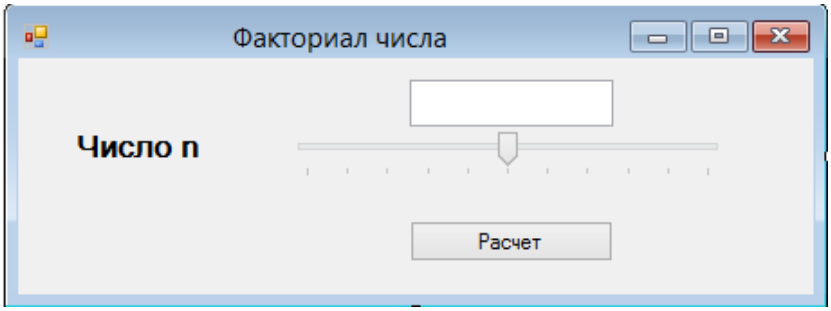
Разработать проект, который позволяет пользователю вычислить факториал числа. Число, для которого рассчитывается факториал, выбирается с помощью элемента управления **TrackBar**. При щелчке по кнопке «Расчет», меняется надпись «Число n» на «N!» и в строке ввода выводится значение факториала числа.

1. Создайте новое **Приложение WindowsForms**. Имя проекта и приложения **ScrollBox2**. Папка для размещения проекта **ScrollBox**.
2. Элемент управления **TrackBar** представляет собой шкалу с движком, с помощью которого пользователь может изменять (регулировать) численное значение. Это может быть уровень громкости, баланс звуковых каналов, насыщенность отдельных компонентов цвета, яркость и пр. Движок элемента управления **TrackBar** можно передвигать мышью, клавишами перемещения курсора, а также клавишами **Home**, **End**, **PgUp** и **PgDn**. При перемещении движка создаются события **Scroll**.

   Помимо движка, в окне элемента управления **TrackBar** есть деления. Они отображаются в виде коротких штрихов, расположенных на равном расстоянии друг от друга.

   Вы можете выбрать горизонтальное или вертикальное расположение окна **TrackBar**. Деления могут находиться с любой стороны, с обеих сторон или их может не быть совсем.

   При создании элемента управления **TrackBar** приложение должно определить диапазон значений, соответствующих положению движка, шаг делений, а также шаг изменения этих значений.
3. Разместите на форме компоненты в соответствии с рисунком.



4. Для элемента управления **TrackBar1** установите значение свойств **Max=10**, **Value=5**.
5. Выделите элемент **TrackBar1**, на панели **Свойства** перейдите на вкладку **События**. Найдите событие **Scroll**, справа от него в поле сделайте двойной щелчок левой кнопкой мыши. Оказавшись в коде программы, введите следующий код:

```
private void trackBar1_Scroll(object sender, EventArgs e)
{
    textBox1.Text = Convert.ToString(trackBar1.Value);
}
```

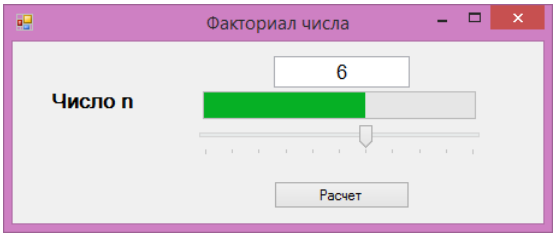
6. Запишите процедуру обработки события **Click** кнопки «**Расчет**». При расчете факториала числа необходимо воспользоваться конструкцией цикл.

```
private void button1_Click(object sender, EventArgs e)
{
    int n = Convert.ToInt32(textBox1.Text);
    int p = 1;
    for (int i=1; i<=n; i++)
    {
        p = p * i;
    }
    textBox1.Text = Convert.ToString(p);
}
```

7. Задайте имя формы проекта **Факториал числа**.
8. Сохраните изменения в проекте и запустите проект, убедитесь в его работоспособности.
9. Измените код процедуры обработки события **Click**кнопки **«Расчет»** так, чтобы надпись**«Число n»** была заменена на **«N!»**.
10. В то время как элемент управления **TrackBar** предназначен для регулирования каких-либо числовых значений, элемент управления **ProgressBar** поможет Вам графически отобразить значения. Он часто применяется, например, для отображения процента завершения какого-либо длительного процесса. Добавьте в проект элемент управления **ProgressBar.**
11. Для элемента управления **ProgressBar1** установите значение свойств **Max=10**, **Value=5**.
12. Перейдите в окно обработчика события **Scroll** элемента управления **TrackBar1** и измените код следующим образом:

```
private void trackBar1_Scroll(object sender, EventArgs e)
{
    textBox1.Text = Convert.ToString(trackBar1.Value);
    progressBar1.Value = trackBar1.Value;
}
```

13. Сохраните изменения в проекте и запустите проект, убедитесь в его работоспособности.



Задания для внеаудиторной самостоятельной работы

Составьте опорный конспект по теме: запишите основные методы, которые используются при создании проекта; запишите в тетрадь основные процедуры для обработки событий проекта.

Критерии оценивания на практическом занятии

Балл	Уровень освоения компетенций	Критерии оценивания уровня освоения компетенций
5	Максимальный	Практическая работа выполнена в полном соответствии с требованиями, студент представил все задания практической работы без погрешностей и замечаний, на все вопросы при защите практической работы дал правильные ответы.
4	Высокий	Практическая работа выполнена в полном соответствии с требованиями, студент представил все задания практической работы с небольшими погрешностями в выполнении на персональном компьютере, на защите практической работы затруднялся при ответах на некоторые вопросы, нуждался в уточняющих вопросах и подсказках со стороны преподавателя.
3	Средний	Практическая работа выполнена в соответствии с требованиями, студент представил все задания практической работы с существенными погрешностями в выполнении на персональном компьютере, неспособен правильно интерпретировать полученные результаты, на защите затруднялся и/или не ответил на большинство вопросов, нуждался в уточняющих вопросах и подсказках со стороны преподавателя.
2	Ниже среднего	Студент несамостоятельно выполнил практическую работу, неспособен пояснить выполнение ни одного задания, не ответил ни на один контрольный вопрос на защите.

ПРАКТИЧЕСКАЯ РАБОТА №6,7
СОЗДАНИЕ ПРОЕКТА С ИСПОЛЬЗОВАНИЕМ КОМПОНЕНТОВ СТАНДАРТНЫХ ДИАЛОГОВ И СИСТЕМЫ МЕНЮ

Цель работы: сформировать умения использования компонентов стандартных диалогов и системы меню, изучить основные свойства; сформировать умения по созданию процедур на основе событий компонентов.

Формируемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.

В результате освоения практических заданий обучающийся должен

- уметь: использовать языки программирования, строить логически правильные и эффективные программы
- знать: объектно-ориентированную модель программирования, основные алгоритмические конструкции.

Материально-техническое обеспечение занятия: персональный компьютер, среда программирования **Visual Studio**.

Практические задания и методические указания

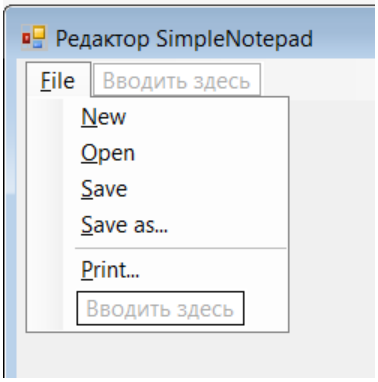
Задание:

Разработать приложение **SimpleNotepad**, представляющее собой простейший текстовый редактор. Использовать в приложении компонент для работы с меню и стандартные окна диалога.

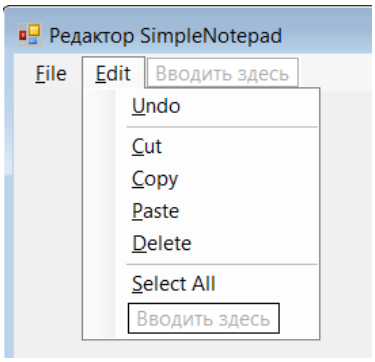
1. Запустите среду программирования **VisualStudio**. Создайте новое **Приложение WindowsForms**. Имя проекта и приложения **SimpleNotepad**.
2. Измените свойство **Name** формы на **SimpleNotepadForm.**
3. Перейдите в окно **Обозреватель решений** и переименуйте файл **Form1.cs** на **SimpleNotepadForm.cs.**
4. Измените размеры окно формы. Измените заголовок окна на **РедакторSimpleNotepad**.
5. Добавьте в окно приложения элемент управления **MenuStrip**.
6. В поле **Вводить здесь** меню введите строку **&File**. В результате в окне приложения появится меню **File**. Обратите внимание, что первая буква в названии подчеркнута. Это получилось потому, что перед ней стоит префикс &.

Этим префиксом отмечается буква, предназначенная для ускоренного выбора меню при помощи клавиатуры.

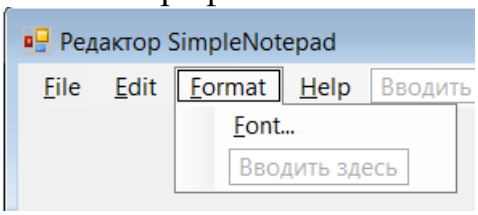
7. Создайте меню **File** в соответствии с рисунком. Для вставки разделительной черты необходимо вызвать контекстное меню и выбрать команду **Separator**.



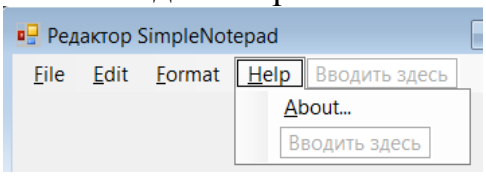
8. Создайте меню **Edit** в соответствии с рисунком. В меню **Edit** реализованы стандартные функции приложения **Блокнот**.



9. Меню **Format** состоит только из одной строки **Font**, с помощью которой пользователь может изменить шрифт текста.



10. Меню **Help** состоит только из одной строк **About**.



11. Переименуйте идентификаторы меню и строк меню таким образом, чтобы с ними было удобнее работать в программе. Для этого в окне дизайнера формы щелкните правой кнопкой мыши главное меню приложения и затем выберите из контекстного меню строку **Правка DropDownItems**. В окне **Редактор коллекций элементов** отредактируйте имена меню и строк меню, изменив значение свойства **Name** соответствующего элемента. При этом меню верхнего уровня должны называться **menuFile**, **menuEdit**, **menuFormat**, **menuHelp**. Имена строк формируются путем добавления к имени меню текста, отображаемого в строке меню. Например, строка **New** называется **menuFileNew**.

12. Запустите приложение. Убедитесь, что меню корректно отображается, и вы можете выбирать его строки.
13. Редактирование текста в приложении будет выполнять компонент **RichTextBox**. Перетащите компонент **RichTextBox**с панели элементов в окно приложения. Задайте значение свойства **Dock** данного компонента равным **Fill**, для того чтобы он занимал все окно приложения.
14. Создайте обработчики событий, необходимые для выполнения таких функций как создание нового документа, открытие и сохранение документа.
15. Добавьте на форму компонент **OpenFileDialog** с вкладки **Диалоговые окна** панели элементов. Выделите пункт меню **Open...** и дважды щелкните по нему левой кнопкой мыши, для того чтобы добавить обработчик события. В обработчик события добавьте следующий код для отображения диалогового окна открытия файла.

```
private void menuFileOpen_Click(object sender, EventArgs e)
{
    if (openFileDialog1.ShowDialog()==System.Windows.Forms.DialogResult.OK && openFileDialog1.FileName.Length>0)
    {
        try
        {
            richTextBox1.LoadFile(openFileDialog1.FileName, RichTextBoxStreamType.RichText);
        }
        catch (System.ArgumentException ex)
        {
            richTextBox1.LoadFile(openFileDialog1.FileName, RichTextBoxStreamType.PlainText);
        }
        this.Text = "Файл[" + openFileDialog1.FileName + "]";
    }
}
```

16. Выделите компонент **openFileDialog1**, на панели **Свойства** найдите свойство **Filter** и введите следующую строку **RTF files|*.rtf|Text files|*.txt|All files|*.***
17. Добавьте на форму компонент **SaveFileDialog** с вкладки **Диалоговые окна** панели элементов. Выделите пункт меню **SaveAs…** и дважды щелкните по нему левой кнопкой мыши, для того чтобы добавить обработчик события. В обработчик события добавьте следующий код для отображения диалогового окна **Сохранение файла**. Этот код сохраняет текст введенный в элемент управления **richTextBox**, в текстовый файл в указанной папке. Метод **ShowDialog** отображает диалоговое окно, а затем с помощью поля **DialogResult**.**OK** осуществляется проверка нажата ли пользователем кнопка **ОК**.

```
private void menuFileSave_Click(object sender, EventArgs e)
{
    if (saveFileDialog1.ShowDialog()==System.Windows.Forms.DialogResult.OK && saveFileDialog1.FileName.Length>0)
    {
        richTextBox1.SaveFile(saveFileDialog1.FileName);
        this.Text = "Файл [" + saveFileDialog1 + "]";
    }
}
```

18. Выделите компонент **saveFileDialog1**, на панели **Свойства** найдите свойство **Filter**и введите следующую строку **RTFfiles|*.rtf**, найдите свойство **FileName** и задайте значение **doc1.rtf**
19. Добавьте в окно дизайнера форм компоненты **PrintDocumet**, **PrintDialog** вкладки **Печать** панели элементов.

20. Компонент **PrintDocumet** предназначен для вывода данных документа на принтер. Свойства компонента **PrintDocumet** описывают, как именно нужно распечатывать документ. В свойство **DocumentName** данного компонента запишите строку **SimpleNotepadDocument**. Эта строка будет идентифицировать документ при отображении состояния очереди печати. Значения остальных свойств оставьте без изменения.
21. С помощью компонента **PrintDialog** приложение выведет на экран стандартное диалоговое окно печати документа. Задайте для данного компонента значение свойства **Document** равным **printDocument1**. Этим обеспечивается связь компонента **printDialog** с компонентом **PrintDocumet**.
22. Для работы с классами, предназначенными для выполнения операций с потоками и печати, добавьте в начало программы следующие строки:

```
using System.IO;
using System.Drawing.Printing;
```

23. Добавьте в класса**SimpleNotepadForm** поля **m_myReader** (для печати содержимого редактора текста) и **m_PrintPageNumber** (номер текущей распечатываемой страницы документа).

```
public partial class SimpleNotepadForm : Form
{
    private StringReader m_myReader;
    private uint m_PrintPageNumber;

    public SimpleNotepadForm()
    {
```

24. Создайте обработчик события печати документа.

```
private void menuFilePrint_Click(object sender, EventArgs e)
{
    m_PrintPageNumber = 1;
    string strText = this.richTextBox1.Text;
    m_myReader = new StringReader(strText);
    Margins margins = new Margins(100, 50, 50, 50);
    printDocument1.DefaultPageSettings.Margins = margins;
    if (printDialog1.ShowDialog()==DialogResult.OK)
    {
        this.printDocument1.Print();
    }
    m_myReader.Close();
}
```

Печать документа будет начинаться с первой страницы, поэтому в поле **m_PrintPageNumber** записывается значение 1. Далее выполняется чтение текущего содержимого окна редактирования текста в поток **m_myReader** класса **StringReader**. Далее задаются границы отступов на распечатываемой странице и отображается диалоговое окно печати документа. Если пользователь щелкает в этом окне кнопку **OK**, документ **printDocument1** отправляется на печать методом **Print**. Далее ненужный более поток **m_myReader** закрывается методом **Close**.

25. На данном этапе приложение еще не в состоянии распечатать документ. Причина этого в том, что приложение пока еще не знает, каким именно образом нужно печатать документ.

26. Чтобы в нашем приложении заработала функция печати, необходимо создать обработчик события **PrintPage**. Для этого нужно дважды щелкнуть левой клавишей мыши значок компонента **printDocument1**. В тело обработчика событий вставьте программный код из текстового файла **Print.txt**. Комментарии в тексте обработчика событий **PrintPage** поясняют назначение отдельных программных строк. Заметим, что для полного понимания действий, выполняемых нашим обработчиком событий, требуется предварительное знакомство с графической подсистемой **Graphics Device Interface Plus (GDI+)**, реализованной компанией **Microsoft** в рамках библиотеки классов **.NET Framework**. Пока же нужно отметить, что приложение распечатывает текст построчно в цикле. После завершения печати всех строк текущей страницы обработчик событий **PrintPage** печатает верхний и нижний колонтитулы, а также рисует горизонтальные линии, отделяющие текст колонтитулов от текста документа.
27. Измените меню **File**, добавив пункт **Exit,** при выборе которого окно редактора текста должно быть закрыто. Добавьте обработчик события для данного пункта меню:

```
private void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.Close();
}
```

28. Однако при закрытии окно приложения возникает проблема: окно редактора текста будет закрыто и в том случае, если пользователь не сохранил сделанные им изменения. Чтобы решить эту проблему, нам нужно каким-то образом отслеживать наличие изменений в окне редактирования текста. Определите в классе **SimpleNotepadForm** поле **m_DocumentChanged**, в котором будет храниться флаг, отмечающий изменения, сделанные пользователем в документе. В новом или только что загруженном документе изменений нет, поэтому начальное значение этого флага равно **false**.

```
private bool m_DocumentChanged = false;
```

29. Создайте обработчик события **richTextBox1_TextChanged**, выполнив двойной щелчок по компоненту **richTextBox**. Этот обработчик получит управление, как только пользователь внесет любые изменения в содержимое редактируемого документа.

```
private void richTextBox1_TextChanged(object sender, EventArgs e)
{
    m_DocumentChanged = true;
}
```

30. Если пользователь редактировал документ, а потом решил создать новый, выбрав из меню **File** строку **New**, изменения, внесенные в старый документ, могут быть потеряны.
31. Чтобы избежать этого, проверьте флаг **m_DocumentChanged** перед тем как очищать содержимое редактора текста. Если в редакторе есть не сохраненные изменения, необходимо выполнить сохранение документа. Отредактируйте код обработчика события пункта меню **New** следующим образом:

```
private void menuFileNew_Click(object sender, EventArgs e)
{
    if (m_DocumentChanged)
    {
        if (saveFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK && saveFileDialog1.FileName.Length > 0)
        {
            richTextBox1.SaveFile(saveFileDialog1.FileName);
            this.Text = "Файл [" + saveFileDialog1 + "]";
            m_DocumentChanged = false;
        }
    }
    richTextBox1.Clear();
}
```

32. Отредактируйте код обработчика события пунктов меню **Save** и**Save As**, добавив строку кода **m_DocumentChanged=false**
33. Измените обработчик события пункта меню **Exit** следующим образом:

```
private void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (saveFileDialog1.ShowDialog()==System.Windows.Forms.DialogResult.OK && saveFileDialog1.FileName.Length>0)
    {
        richTextBox1.SaveFile(saveFileDialog1.FileName);
        this.Text = "Файл [" + saveFileDialog1 + "]";
        m_DocumentChanged = false;
    }
    this.Close();
}
```

34. Надо выполнить еще одну проверку флага **m_DocumentChanged** в методе **Dispose**, который вызывается при закрытии окна приложения. Для этого на панели Обозреватель решений выделите **SimpleNotepadForm.Designer.cs**, найдите метод **Dispose** и внесите необходимые изменения.

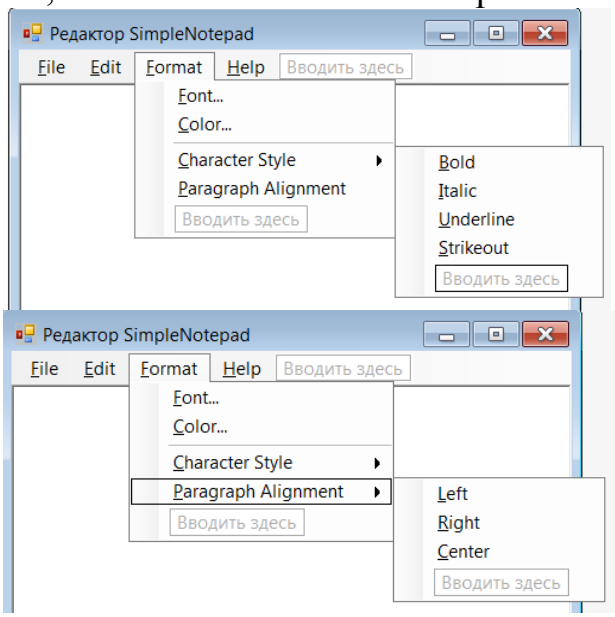
```
protected override void Dispose(bool disposing)
{
    if (saveFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK && saveFileDialog1.FileName.Length > 0)
    {
        richTextBox1.SaveFile(saveFileDialog1.FileName);
        this.Text = "Файл [" + saveFileDialog1 + "]";
        m_DocumentChanged = false;
    }
    if (disposing && (components != null))
    {
        components.Dispose();
    }
    base.Dispose(disposing);
}
```

Теперь, когда пользователь попытается закрыть программу с помощью соответствующей кнопки заголовка окна, не сохранив сделанные изменения, на экране появится стандартное диалоговое окно, предлагающее ему сохранить документ.

35. Создайте обработчики событий пунктов меню **Edit**. Реализация данных функций является простой, поскольку элемент управления **RichTextBox** имеет все необходимые методы. Добавьте в пункт меню **Edit** строку **Redo**. Подготовьте обработчики событий для всех строк меню **Edit**.

```
private void menuEditUndo_Click(object sender, EventArgs e)
{
    richTextBox1.Undo();
}

private void menuEditRedo_Click(object sender, EventArgs e)
{
    richTextBox1.Redo();
}

private void menuEditCut_Click(object sender, EventArgs e)
{
    richTextBox1.Cut();
}

private void menuEditCopy_Click(object sender, EventArgs e)
{
    richTextBox1.Copy();
}

private void menuEditPaste_Click(object sender, EventArgs e)
{
    richTextBox1.Paste();
}

private void menuEditDelete_Click(object sender, EventArgs e)
{
    richTextBox1.Cut();
}

private void menuEditSelectAll_Click(object sender, EventArgs e)
{
    richTextBox1.SelectAll();
}
```

36. Измените меню **Format**, добавив в него строку **Color**, а также два меню второго уровня – **CharacterStyle** и **ParagraphAlignment**. Измените имена строк меню таким образом, чтобы с ними было легче работать в программе.



37. Добавьте на форму компонент **FontDialog** с вкладки **Диалоговые окна** панели элементов. Этот компонент отображает на экране стандартное диалоговое окно выбора шрифта. Создайте обработчик события пункта меню **Font** и вставьте следующий код:

```
private void menuFormatFont_Click(object sender, EventArgs e)
{
    if (fontDialog1.ShowDialog()==DialogResult.OK)
    {
        richTextBox1.SelectionFont = fontDialog1.Font;
    }
}
```

После того как пользователь выбрал нужный ему шрифт, обработчик события переписывает этот шрифт из свойства **fontDialog1.Font** в свойство **richTextBox1.SelectionFont**. Свойство **SelectionFont** позволяет изменить шрифт фрагмента текста, выделенного пользователем (или программой) в окне редактирования.

38. Добавьте на форму компонент **ColorDialog** с вкладки **Диалоговые окна** панели элементов. Этот компонент отображает на экране стандартное диалоговое окно выбора шрифта. Создайте обработчик события пункта меню **Color** и вставьте следующий код:

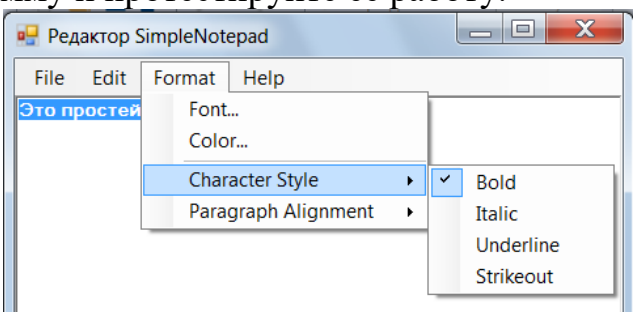
```
private void colorToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (colorDialog1.ShowDialog()==DialogResult.OK)
    {
        richTextBox1.SelectionColor = colorDialog1.Color;
    }
}
```

39. Создайте обработчик события для пункта меню **Bold** и вставьте следующий код:

```
private void menuFormatFontCharacterStyleBold_Click(object sender, EventArgs e)
{
    if (richTextBox1.SelectionFont!=null)
    {
        System.Drawing.Font currentFont = richTextBox1.SelectionFont;
        System.Drawing.FontStyle newFontStyle;
        if (richTextBox1.SelectionFont.Bold==true)
        {
            newFontStyle = FontStyle.Regular;
        }
        else
        {
            newFontStyle = FontStyle.Bold;
        }
        richTextBox1.SelectionFont = new Font(currentFont.FontFamily, currentFont.Size, newFontStyle);
        CheckMenuFontCharacterStyle();
    }
}
```

Получив управление данный метод прежде всего, определяет шрифт фрагмента текста, выделенного пользователем, анализируя свойство **richTextBox1.SelectionFont**. Если шрифт определить не удалось, и это свойство содержит значение **null**, программа не делает никаких изменений. В противном случае программа сохраняет текущий шрифт в переменной **currentFont** класса **System.Drawing.Font**. Далее метод проверяет, был ли выделен фрагмент текста жирным шрифтом, анализируя свойство **richTextBox1.SelectionFont.Bold**. Если это свойство содержит значение **true**, то метод снимает выделение, если нет, то устанавливает его. Для снятия выделения программа записывает в переменную **newFontStyle** значение **FontStyle.Regular**, а для установки— значение **FontStyle.Bold**.

40. При выделении фрагмента текста жирным шрифтом в меню одновременно отмечается «галочкой» строка **Bold**. Для этого необходимо создать **CheckMenuFontCharacterStyle**. Для этого перейдите в окно кода и вставьте программный код процедуры из текстового документа **CheckMenu.txt.** Метод **CheckMenuFontCharacterStyle** по очереди проверяет стилевое оформление выделенных фрагментов, отмечая «галочками» соответствующие строки меню **CharacterStyle** или снимая со строк этого меню отметки.
41. Запустите программу и протестируйте её работу.



42. Аналогично создайте обработчики для пунктов меню **Italic, Underline, Strikeout**.

43. Подготовьте обработчики событий для строк меню **Paragraph Alignment** следующим образом:

```
private void menuFormatParagraphAlignmentLeft_Click(object sender, EventArgs e)
{
    richTextBox1.SelectionAlignment = HorizontalAlignment.Left;
}

private void menuFormatParagraphAlignmenRight_Click(object sender, EventArgs e)
{
    richTextBox1.SelectionAlignment = HorizontalAlignment.Right;
}

private void menuFormatParagraphAlignmentCenter_Click(object sender, EventArgs e)
{
    richTextBox1.SelectionAlignment = HorizontalAlignment.Center;
}
```

Все эти обработчики событий строк меню **ParagraphAlignment** изменяют значение свойства **richTextBox1.SelectionAlignment**, задающего выравнивание параграфа текста, выделенного пользователем (для выделения параграфа с целью изменения выравнивания достаточно установить в него текстовый курсор).

44. Сохраните изменения и протестируйте работу приложения.

Задание 2. Создание панели инструментов приложения

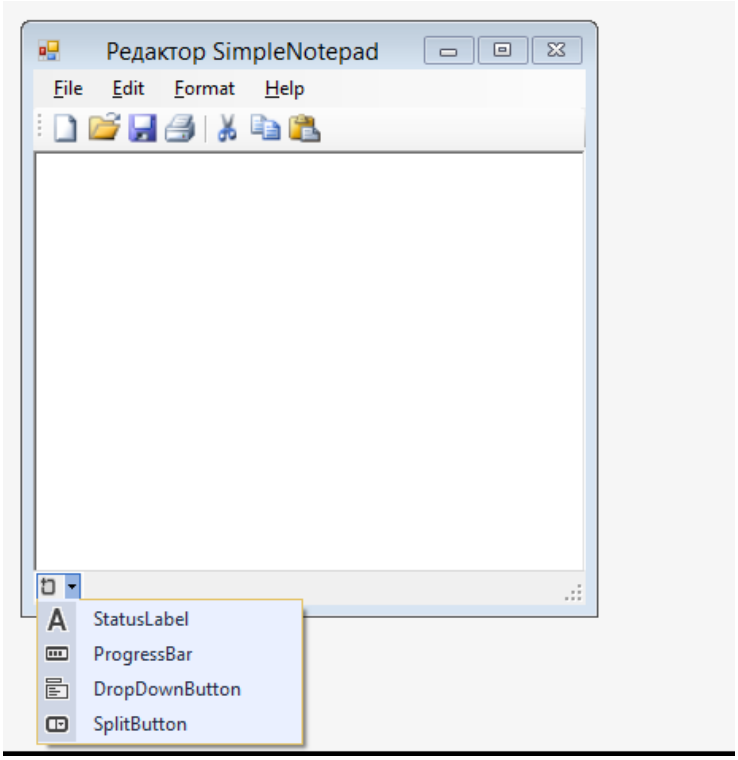
1. Во многих приложениях, созданных для операционной системы **Windows**, наиболее часто использующиеся строки меню дублируются кнопками, расположенными на инструментальных панелях. Чтобы добавить инструментальную панель в окно приложения, перетащите мышью элемент управления **ToolStrip** вкладки **Все формы Windows Forms** панели элементов в окно проектирования формы нашего приложения. По умолчанию окно инстру-

ментальной панели появится в верхней части формы. В только что добавленной панели нет ни одной кнопки.

2. Сразу после добавления окно инструментальной панели будет расположено над окном редактора текста. Чтобы исправить это положение, щелкните правой кнопкой мыши окно редактора текста, а затем выберите из контекстного меню строку **На передний план**. В результате окна примут правильное взаимное расположение.
3. Создайте в папке проекта отдельную папку для хранения изображений инструментальной панели, назвав ее, например, **ImageList**. Далее скопируйте в созданную папку изображения, подготовленные для кнопок панели.
4. Выделите компонент **ToolStrip** и на панели свойств найдите свойство **Items,** щелкните по кнопке  . Откроется окно редактора коллекций.
5. В диалоговом окне **Редактор коллекции элементов** в списке элементов выберите **Button** и нажмите кнопку **Добавить**. На панели свойств измените значение свойства **Name** данной кнопки на **создатьToolStripButton**, а размер кнопки установите равным **24**.
6. В окне свойств найдите свойство **Image**. Нажмите на кнопку  , в диалоговом окре**Выбор ресурса** нажмите на кнопку **Импорт** и выберите соответствующее изображение из сохранённых ранее в папке **ImageList**.
7. Задайте комментарий-подсказку для кнопки **создатьToolStripButton**, установив свойство **ToolTripText** равным **Создать**.
8. Аналогично создайте кнопки **открытьToolStripButton, сохранитьToolStripButton, печатьToolStripButton.**
9. Далее необходимо создать разделитель между группами кнопок, ответственных за выполнение различных групп операций. Для этого в списке элементов выберите **Separator** и нажмите кнопку **Добавить**.
10. Далее аналогично создайте кнопки **вырезатьToolStripButton, копироватьToolStripButton, вставитьToolStripButton**.
11. Теперь необходимо связать созданные ранее функции-обработчики события команд пункт меню с соответствующими кнопками панели инструментов. Выделите кнопку **создатьToolStripButton,** на панели **Свойства** перейдите на вкладку **События**  , найдите событие **Click** и в списке выберите функцию **menuFileNew_Click**. Аналогично добавьте функции для остальных кнопок панели инструментов.
12. Сохраните приложение. Запустите его и протестируйте работу кнопок панели инструментов.

Задание 3. Создание строки состояния

1. Разместите на форме компонент **StatusStrip** со вкладки **Меню и панели инструментов** панели элементов.
2. В меню компонента **StatusStrip** выберите опцию **StatusLabel**.



3. Далее на панели **Свойства** найдите свойство **Text** и сделайте его пустым.
4. Для того чтобы отслеживать выбор строк меню необходимо предусмотреть специальный обработчик события **DropDownItemClicked**. Это событие создается строками меню, когда пользователь выбирает их при помощи мыши или клавиатуры. Создайте обработчик событий. Для этого выделите мышью меню **File** в окне дизайнера формы, а затем перейдите на вкладку **События** панели **Свойства**. В списке событий найдите событие **DropDownItemClicked** и дважды щелкните по строке рядом с ним. После этого будет создано пустое тело обработчика события. Вставьте следующий код в обработчик события:

```
private void menuFile_DropDownItemClicked(object sender, ToolStripItemClickedEventArgs e)
{
    this.UpdateStatus(e.ClickedItem);
}
```

5. Создайте пустую строку перед обработчиком данного события и опишите метод **UpdateStatus** следующим образом:

```
private void UpdateStatus(ToolStripItem item)
{
    if (item != null)
    {
        string msg = String.Format("{0} selected", item.Text);
        this.statusStrip1.Items[0].Text = msg;
    }
}
```

6. Выделите мышью меню **Edit** в окне дизайнера формы, а затем перейдите на вкладку **События** панели **Свойства**. В списке событий найдите событие **DropDownItemClicked** и в списке рядом с ним выберите событие **menuFile_DropDownItemClicked**. Аналогично выполните для всех остальных пунктов меню.

7. Сохраните приложение. Запустите его и протестируйте работу строки состояния.

Задания для внеаудиторной самостоятельной работы

Составьте опорный конспект по теме: запишите основные методы, которые используются при создании проекта.

Критерии оценивания на практическом занятии

Балл	Уровень освоения компетенций	Критерии оценивания уровня освоения компетенций
5	Максимальный	Практическая работа выполнена в полном соответствии с требованиями, студент представил все задания практической работы без погрешностей и замечаний, на все вопросы при защите практической работы дал правильные ответы.
4	Высокий	Практическая работа выполнена в полном соответствии с требованиями, студент представил все задания практической работы с небольшими погрешностями в выполнении на персональном компьютере, на защите практической работы затруднялся при ответах на некоторые вопросы, нуждался в уточняющих вопросах и подсказках со стороны преподавателя.
3	Средний	Практическая работа выполнена в соответствии с требованиями, студент представил все задания практической работы с существенными погрешностями в выполнении на персональном компьютере, неспособен правильно интерпретировать полученные результаты, на защите затруднялся и/или не ответил на большинство вопросов, нуждался в уточняющих вопросах и подсказках со стороны преподавателя.
2	Ниже среднего	Студент несамостоятельно выполнил практическую работу, неспособен пояснить выполнение ни одного задания, не ответил ни на один контрольный вопрос на защите.

ПРАКТИЧЕСКАЯ РАБОТА №8,9
РАЗРАБОТКА МНОГООКОННОГО ПРИЛОЖЕНИЯ

Цель работы: сформировать практические умения создания многооконного приложения сс фреймами средствами среды программирования **VisualStudio**.

Формируемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.

– ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
– ПК 2.2. Программировать в соответствии с требованиями технического задания.
– ПК 2.3. Применять методики тестирования разрабатываемых приложений.

В результате освоения практических заданий обучающийся должен

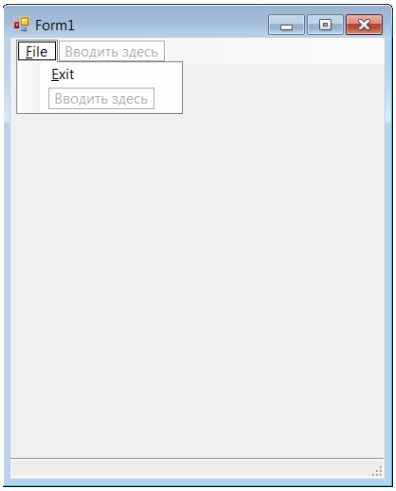
– уметь: использовать языки программирования, строить логически правильные и эффективные программы
– знать: объектно-ориентированную модель программирования, основные алгоритмические конструкции.

Материально-техническое обеспечение занятия: персональный компьютер, среда программирования **Visual Studio**.

Практические задания и методические указания

Задание 1. Создание главного окна приложения

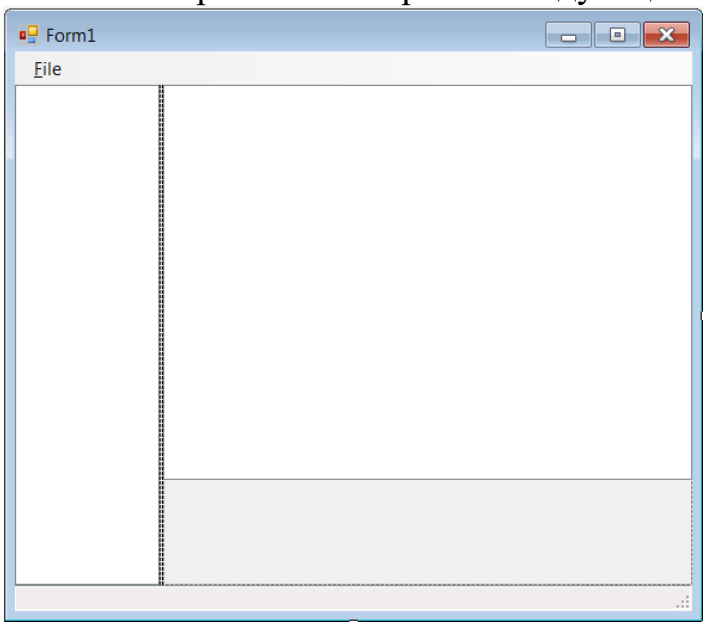
1. Запустите среду программирования **VisualStudio**. Создайте новое **Приложение WindowsForms**. Имя проекта и приложения **FramesApp**.
2. Добавьте в форму главного окна приложения меню (**MenuStrip**) и строку состояния (**StatusStrip**). В меню **File** создайте строку **Exit**, предназначенную для завершения работы приложения. Напишите обработчик данного события.



3. Добавьте в окно нашего приложения элемент управления **TreeView**, перетащив мышью его значок из панели элементов вкладки **Все формы WindowsForms** в окно формы. Установите значение свойства **Dock** данного элемента управления равным **Left**. В результате элемент управления **TreeView** будет выровнен по левой границе главного окна приложения. В нашем приложении окно элемента управления **TreeView** будет использовано для отображения списка дисков и каталогов.
4. Для элемента управления **TreeView** создайте разделитель. Для этого перетащите значок элемента управления **Splitter**из панели элементов вкладки

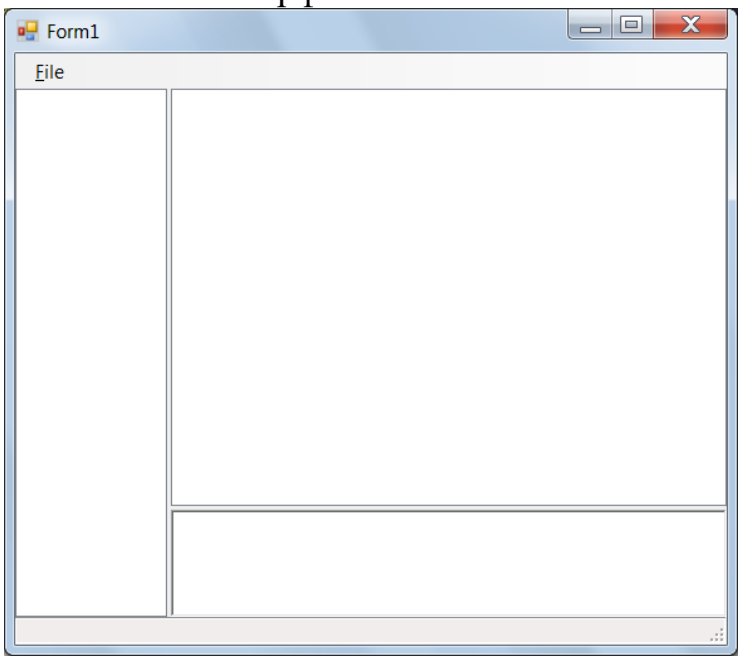
Все формы WindowsForms в окно формы. Разделитель будет автоматически расположен справа от окна элемента управления **TreeView**.

5. Элемент управления **Splitter** имеет свойства, которые можно менять в процессе разработки приложения, а также динамически во время его работы. Самое важное свойство разделителя **Splitter** — это свойство **Dock**, задающее его расположение. По умолчанию значение этого свойства равно **Left**, благодаря чему разделитель разместился слева от окна дерева **TreeView**. При необходимости можно изменять расположение разделителя с помощью окна редактирования, аналогичного окну.
6. Перетащите значок элемента управления **Panel**из панели элементов вкладки **Все формы WindowsForms** в окно формы, а затем установите значение свойства **Dock** этой панели равным **Fill**. В результате наша панель будет расположена справа от разделителя и займет всю правую часть окна приложения.
7. В верхнем окне будет находиться элемент управления **ListView**, показывающий список файлов и каталогов. Добавьте в окно приложения элемент управления **ListView**, перетащив его значок из панели элементов вкладки **Все формы WindowsForms** в окно формы. После установки значения свойства **Dock**, равным **Top**, и увеличения высоты окна этого элемента управления, главное окно нашего приложения примет следующий вид.



8. Добавьте в правую часть главного окна горизонтальный разделитель **Splitter**. Перетащите его пиктограмму из панели элементов вкладки **Все формы WindowsForms** в окно формы, а затем установите значение свойства **Dock** добавленного разделителя, равным **Top**. В результате разделитель будет расположен непосредственно под окном элемента управления **ListView**.
9. В нижней части окна приложения будет находиться элемент управления **RichTextBox**, предназначенный для отображения дополнительной информации. Перетащите значок элемента управления **RichTextBox** из панели элементов вкладки **Все формы WindowsForms** в окно формы и установите значение свойства **Dock** равным **Fill**.

10. Протестируйте работу приложения. Сохраните полученный проект. Не написав ни единой строчки кода, вы создали простейшее приложение с действующим многооконным интерфейсом пользователя.



Задание 2. Создание методов для элемента управления TreeView

1. Инициализация дерева **TreeView** осуществляется в конструкторе класса **Form1**. Для этого необходимо подготовить метод **DriveTreeInit**.

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();

        DriveTreeInit();
    }
```

2. Для обращения к этому методу, а также к другим методам, работающим с дисками, каталогами и файлами, подключите пространство имен **System.IO**.

```
using System.IO;
```

3. Перейдите в окно редактирования кода и создайте метод **DriveTreeInit**:

```
public void DriveTreeInit()
{
    string[] drivesArray = Directory.GetLogicalDrives();

    treeView1.BeginUpdate();
    treeView1.Nodes.Clear();

    foreach(string s in drivesArray)
    {
        TreeNode drive = new TreeNode(s, 0, 0);
        treeView1.Nodes.Add(drive);

        GetDirs(drive);
    }
    treeView1.EndUpdate();
}
```

В самом начале своей работы этот метод получает список логических дисковых устройств, установленных в системе, и сохраняет его в массиве **drivesArray**. Далее метод **DriveTreeInit** вызывает метод **treeView1.BeginUpdate**. Этот метод временно блокирует перерисовку окна дерева до тех пор, пока не будет вызван метод **treeView1.EndUpdate**. Пара этих методов используется в том случае, когда нужно добавить, удалить или изменить большое количество элементов дерева. Если не заблокировать перерисовку окна, на обновление дерева уйдет слишком много времени.

В классе **TreeView** определено свойство **Nodes**, хранящее все узлы дерева. Перед тем как приступить к заполнению дерева, метод **DriveTreeInit** удаляет все узлы, вызывая для этого метод **treeView1.Nodes.Clear**.

Заполнение дерева происходит в цикле. Массив **drivesArray** содержит массив текстовых строк с обозначениями логических устройств, установленных в системе. Для каждого такого устройства в цикле создается объект класса **TreeNode** — узел дерева. В качестве первого параметра конструктору передается текстовая строка названия устройства. Созданный узел добавляется в набор узлов дерева с помощью метода **treeView1.Nodes.Add**. В качестве параметра этому методу передается ссылка на объект **TreeNode**, т.е. на добавляемый узел. Далее вызывается метод **GetDirs**, добавляющий в дерево список содержимого корневого каталога текущего дискового устройства.

4. Метод **GetDirs** получает в качестве параметра ссылку на узел дерева, который требуется наполнить списком имен каталогов и файлов. Создайте метод **GetDirs.**

```
public void GetDirs(TreeNode node)
{
    DirectoryInfo[] diArray;
    node.Nodes.Clear();

    string fullPath = node.FullPath;
    DirectoryInfo di = new DirectoryInfo(fullPath);

    try
    {
        diArray = di.GetDirectories();
    }
    catch
    {
        return;
    }

    foreach(DirectoryInfo dirinfo in diArray)
    {
        TreeNode dir = new TreeNode(dirinfo.Name, 0, 0);
        node.Nodes.Add(dir);
    }
}
```

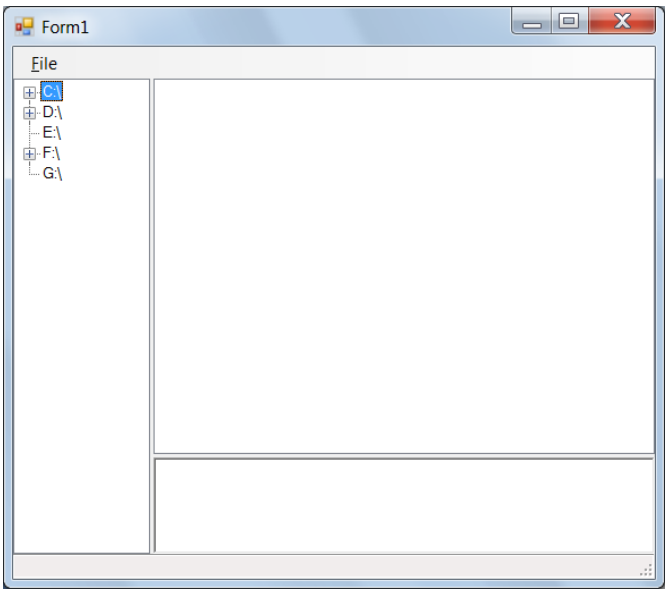
Первым делом метод **GetDirs** удаляет все элементы из текущего узла, вызывая для этого метод **node.Nodes.Clear**.

Далее метод переписывает в переменную **fullPath** типа **string** полный путь **node.FullPath** к узлу дерева. Эта строка получается объединением текстовых строк всех родительских узлов. Если корневой узел хранит текстовую строку обозначения логического диска, то после выполнения этой процедуры в переменной **fullPath** будет храниться полный путь к файлу или каталогу.

Далее для получения содержимого каталога, на который ссылается переменная **fullPath**, используется метод **GetDirectories**. При возникновении исключения программа просто возвращает управление, не выполняя над узлом дерева никаких функций.

В том случае если содержимое каталога было успешно получено, оно сохраняется в массиве **diArray**. Далее содержимое массива **diArray** используется для заполнения узла дерева содержимым каталога.

5. Запустите программу на исполнение. В окне дерева появится список дисковых устройств. Раскрываются ли узлы дерева при щелчке мышью?

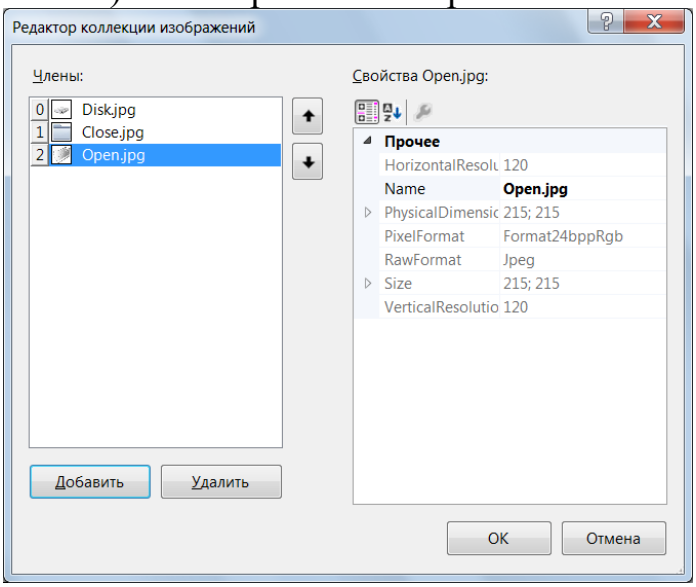


6. Чтобы узлы дерева раскрывались, когда пользователь щелкает их мышью или пытается раскрыть при помощи клавиатуры, нужно обрабатывать событие **BeforeExpand**. Для создания обработчика этого события выделите дерево в окне дизайнера формы, а затем откройте вкладку событий. Далее в поле **BeforeExpand** ведите имя обработчика событий **treeView1_OnBeforeExpand**. После этого добавьте в этот обработчик событий следующий код:

```
private void treeView1_OnBeforeExpand(object sender, TreeViewCancelEventArgs e)
{
    treeView1.BeginUpdate();

    foreach (TreeNode node in e.Node.Nodes)
    {
        GetDirs(node);
    }

    treeView1.EndUpdate();

}
```

   Событие **BeforeExpand** возникает при попытке пользователя раскрыть узел дерева. В этом случае наш обработчик заполняет открываемый узел при помощи рассмотренного ранее метода **GetDirs**. Ссылка на узел извлекается из поля **e.Node.Nodes**, передаваемого обработчику событий в качестве параметра.
7. Запустите приложение. Раскрываются ли узлы дерева при щелчке мышью теперь?
8. Вы можете улучшить внешний вид дерева, если добавите к его узлам значки дисковых устройств и каталогов (папок). В папке проекта создайте папку **ImageList**. Подготовьте эти значки, используя любой графический редактор. Скопируйте эти значки в папку **ImageList**. Чтобы подключить флажки к проекту, выберите из меню проект системы **MicrosoftVisualStudio** строку **Добавить существующий элемент**, а затем добавьте файлы значков при помощи появившегося на экране диалогового окна.

9. Чтобы использовать изображения в дереве, их необходимо объединить в список класса **ImageList**. Добавьте этот список в приложение, перетащив значок компонента **ImageList** в окно проектирования формы из панели элементов вкладки **Все формы WindowsForms**. Выделив элемент **imageList1** левой клавишей мыши, отредактируйте его свойство **Images**. Для редактирования будет открыто окно **Редактор коллекций изображений**. Воспользуйтесь кнопкой **Добавить** для добавления в список файлов изображений, скопированных ранее в каталог нашего приложения. Изображения следует разместить в следующем порядке: первым (с индексом 0) должно идти изображение для диска, вторым (с индексом 1) — изображение закрытой папки, и третьим (с индексом 2) — изображение открытой папки.



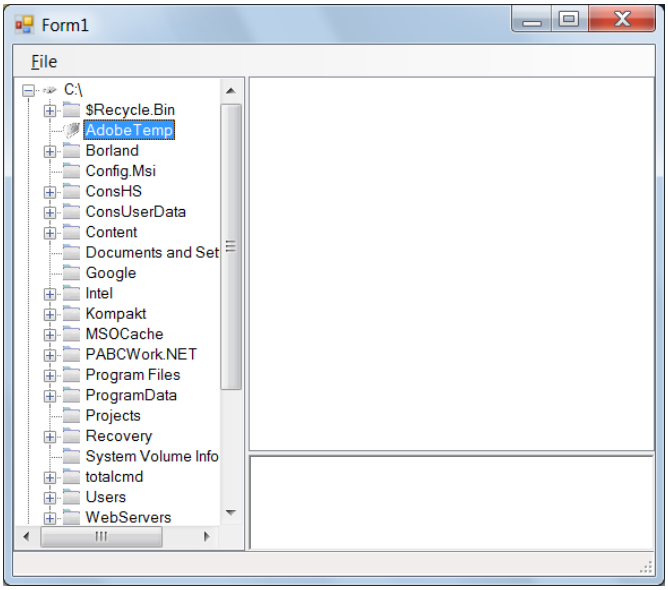
10. Создав и заполнив список изображений, подключите его к дереву просмотра дисков и каталогов. Для этого отредактируйте свойство **ImageList** элемента управления **treeView1**, присвоив ему ссылку на список изображений **imageList1**.
11. Для отображения значков в узлах дерева необходимо изменить исходный текст методов **DriveTreeInit** и **GetDirs**. Первый из этих методов инициализирует дерево, а второй— добавляет к узлу дерева список каталогов. Обратите внимание на конструктор класса **TreeNode**, создающий узлы дерева внутри тела цикла **foreach**. Первый параметр задает текст надписи для узла дерева. Если к элементу управления **TreeView** подключен список изображений, то второй и третий параметры конструктора класса **TreeNode** задают индексы изображений для узла дерева. При этом второй параметр определяет изображение невыделенного узла дерева, а третий — выделенного. Что касается метода **DriveTreeInit**, то расположенный в нем конструктор создает узлы, отображающий только дисковые устройства. В любом состоянии (как выделенном, так и невыделенном) нам необходимо отображать один и тот же значок дискового устройства, имеющий в нашем случае индекс 0. Поэтому второй и третий параметры конструктора передают нулевые значения.

```
public void DriveTreeInit()
{
    string[] drivesArray = Directory.GetLogicalDrives();

    treeView1.BeginUpdate();
    treeView1.Nodes.Clear();

    foreach(string s in drivesArray)
    {
        TreeNode drive = new TreeNode(s, 0, 0);
        treeView1.Nodes.Add(drive);

        GetDirs(drive);
    }
    treeView1.EndUpdate();
}
```

12. В методе **GetDirs** конструктору класса **TreeNode**, размещенному внутри оператора цикла **foreach**, через второй и третий параметры передаются индексы значков закрытой и открытой папки, соответственно.

```
public void GetDirs(TreeNode node)
{
    DirectoryInfo[] diArray;
    node.Nodes.Clear();

    string fullPath = node.FullPath;
    DirectoryInfo di = new DirectoryInfo(fullPath);

    try
    {
        diArray = di.GetDirectories();
    }
    catch
    {
        return;
    }

    foreach(DirectoryInfo dirinfo in diArray)
    {
        TreeNode dir = new TreeNode(dirinfo.Name, 1, 2);
        node.Nodes.Add(dir);
    }
}
```

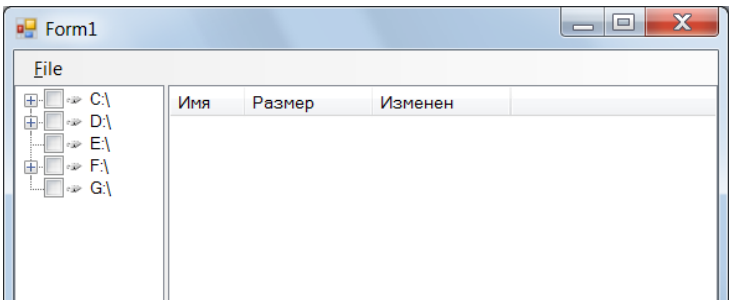
13. Протестируйте работу приложения.



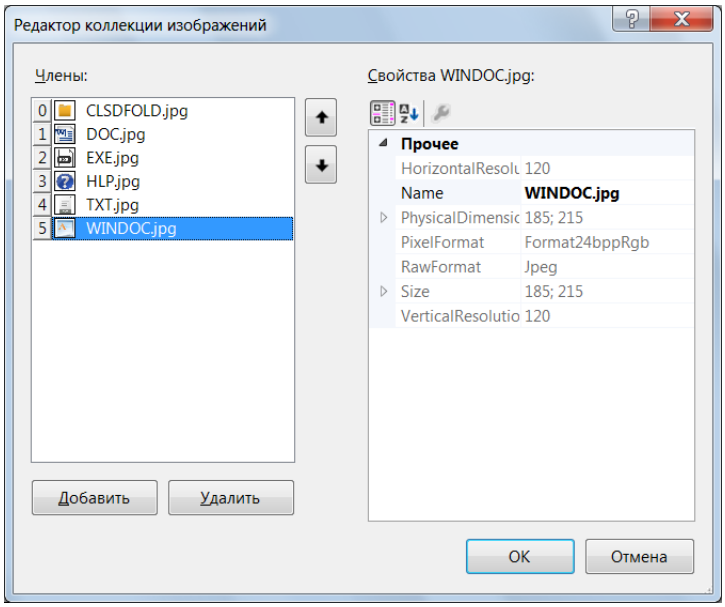
14. Выделите элемент управления **TreeView** на панели Свойства найдите свойство **CheckBoxes** и установите его значение **True**. Теперь узлы дерева будут иметь индивидуальные флажки. Отметив все или некоторые узлы дерева флажками, можно выполнять над соответствующими объектами групповые операции.
15. Выделите элемент управления **TreeView** на панели Свойства найдите свойство **LabelEdit** и установите его значение **True**. Теперь пользователь может редактировать текстовые надписи, расположенные около узлов дерева.

Задание 3. Создание методов для элемента управления ListView

1. Выделите список **ListView** в окне дизайнера форм и установите для свойства **View** значение **Details**. При необходимости программа сможет динамически переключать режимы отображения, изменяя значение свойства **View** во время своей работы.
2. Поскольку в приложении будет отображаться список в виде таблицы, то необходимо создать столбцы таблицы и определить их атрибуты. Для этого необходимо отредактировать свойство **Columns**. Это делается при помощи **Редактор коллекции ColumnHeader**.
3. С помощью кнопки **Добавить** добавьте в таблицу три столбца, а затем настройте свойства **Text**, **TextAlign** и **Width** этих столбцов. Свойство **Text** задает название столбца, отображаемого в верхней части списка **ListView**. Первый столбец (с индексом 0) должен называться **Имя**, второй — **Размер**, а третий — **Изменен**. Свойство **TextAlign** задает выравнивание заголовка столбца. По умолчанию это свойство имеет значение **Left**, задающее выравнивание по левой границе. С помощью свойства **Width** можно указать начальную ширину столбца в пикселях. После отображения списка пользователь сможет изменять ширину столбцов при помощи мыши.
4. Запустите приложение на исполнение. Теперь в правом верхнем фрейме приложения появится пустой список с заголовками.



5. Элемент управления **ListView** может отображать содержимое своего окна в четырех различных режимах, задаваемых при помощи свойства **View**. В зависимости от выбранного режима, элементы списка могут снабжаться значками маленького или большого размера. В приложении **FramesApp** будут использованы шесть значков для отображения папок и файлов различных типов (файлы исполнимых программ, файлов справочной системы, текстовых файлов, файлов документов и пр.). Скопируйте файлы значков в папку **ImageList** созданную ранее.
6. Для работы с этими значками потребуется еще один список изображений **ImageList**. Перетащите его значок с панели элементов в окно дизайнера форм. Новый список будет иметь идентификатор **imageList2**. Отредактируйте данный список.



7. В нашем приложении список **ListView** отображается только в одном режиме, а именно, как детализированная таблица. Такая таблица снабжается значками только маленького размера. Если же Ваше приложение будет использовать режим **LargeIcon**, необходимо подготовить дополнительный список **ImageList**, добавив в него файлы значков большого размера. Подключите подготовленный список к элементу управления **ListView**. Для этого присвойте свойствам **SmallImageList**, **LargeImageList** значение **imageList2**.
8. В общем случае список **ListView** содержит элементы, каждый из которых имеет дополнительные элементы более низкого уровня, которые называются атрибутами. Сам элемент представляется текстовой строкой в первом столбце списка, отображаемого в виде детализированной таблицы. Значения остальных атрибутов отображаются в остальных столбцах таблицы. Теперь

необходимо наполнить список **ListView** функциональностью. Когда пользователь выделяет диск или каталог в дереве просмотра **TreeView**, создается событие **AfterSelect**. Обработчик этого события должен определить, какой диск или какой каталог был выделен, а затем наполнить окно списка **ListView** именами каталогов и файлов, расположенных на этом диске или в этом каталоге. Создайте обработчик события **AfterSelect**. Предварительно создайте поле **fullPath** в классе **Form1**, вставив следующую строку кода:

```
publicstring fullPath;
```

Код обработчика события скопируйте из файла **AfterSelect.txt**.

Прежде всего, данный метод извлекает ссылку на узел дерева, выделенный пользователем, из свойства **Node** параметра обработчика событий **treeView1_OnAfterSelect**. Далее, полный путь к выделенному узлу записывается в поле **fullPath** класса **string**.

Так как наше дерево содержит только обозначения дисков и каталогов, то это будет путь либо к корневому каталогу диска, либо к одному из подкаталогов. Далее мы создаем объект класса **DirectoryInfo**и получаем списки всех файлов и каталогов, располагающихся в каталоге, выделенном пользователем в дереве. Для выполнения этих операций применяются методы **GetFiles** и **GetDirectories**. Перечень файлов обработчик события сохраняет в массиве **fiArray**, а перечень каталогов — в массиве **diArray**.

Наполнение списка именами каталогов выполняется в цикле **foreach**. С помощью конструктора класса **ListViewItem** мы создаем элемент списка, а затем задаем значения атрибутов этого элемента. Длина каталогов считается равной нулю, а время последнего изменения каталога извлекается при помощи метода **LastWriteTime** и преобразуется в текстовую строку методом **ToString**.

В свойство **lvi.ImageIndex** записывается нулевое значение — индекс значка в списке изображений **imageList2** с изображением закрытой папки.

После заполнения всех атрибутов элемента списка этот элемент добавляется в список методом **listView1.Items.Add**.

На следующем этапе в список добавляются имена файлов, расположенных в выделенном каталоге. Эти имена тоже добавляются в цикле

Размер очередного файла мы получаем с помощью свойства **Length**, а дату последнего изменения — с помощью свойства **LastWriteTime** (как и для каталогов).

Расширение имени файла извлекается из полного имени методом **Path.GetExtension**, а затем все его буквы преобразуется в прописные методом**ToLower**. Непосредственный выбор значка выполняется при помощи оператора **switch**. Подготовленный элемент списка, описывающий текущий файл, добавляется в список методом **listView1.Items.Add.**

9. Протестируйте работу приложения. Приложение может отображать в левом фрейме дерево каталогов, а в правом верхнем фрейме — содержимое выбранных каталогов.
10. Разработайте обработчик события **ItemActivate** к элементу управления **ListView**, для того чтобы было можно отображать в окне текстового редак-

тора содержимое текстовых файлов, отображаемых в правом верхнем фрейме. Это событие возникает, когда пользователь дважды щелкает строку в окне списка **ListView**. Для того, чтобы в строке состояния отображалась информация об имени файла добавьте на строку состояния **StatusLabel**. Исходный текст обработчика события **ItemActivate** имеет следующий вид:

```
private void OnItemActivate(object sender, EventArgs e)
{
    foreach(ListViewItem lvi in listView1.SelectedItems)
    {
        string ext = Path.GetExtension(lvi.Text).ToLower();
        if (ext==".txt"||ext==".htm"||ext==".html")
        {
            try
            {
                richTextBox1.LoadFile(Path.Combine(fullPath, lvi.Text),
                    RichTextBoxStreamType.PlainText);

                StatusLabel1.Text = lvi.Text;
            }
            catch
            {
                return;
            }
        }
        else if (ext==".rtf")
        {
            try
            {
                richTextBox1.LoadFile(Path.Combine(fullPath, lvi.Text),
                    RichTextBoxStreamType.RichText);

                StatusLabel1.Text = lvi.Text;
            }
            catch
            {
                return;
            }
        }
}
```

В качестве первого параметра методу передается полный путь к файлу, полученный комбинированием полного пути каталога **fullPath**, выделенного в дереве, и имени файла **lvi.Text**, выделенного в списке **ListView**.

Через второй параметр методу **LoadFile** передается тип загружаемого документа, заданный как **RichTextBoxStreamType.PlainText** (т.е. текстовый документ). Дополнительно имя документа отображается в строке состояния главного окна приложения, для чего это имя переписывается из свойства **lvi.Text** в свойство **statusLabel1.Text**.

11. Протестируйте работу приложения. Сохраните изменения.

Задания для внеаудиторной самостоятельной работы

Составьте опорный конспект по теме: запишите основные методы, которые используются при создании проекта.

Критерии оценивания на практическом занятии

Балл	Уровень освоения компетенций	Критерии оценивания уровня освоения компетенций
5	Максимальный	Практическая работа выполнена в полном соответствии с требованиями, студент представил все задания практической работы без погрешностей и замечаний, на все вопросы при защите практической работы дал правильные ответы.
4	Высокий	Практическая работа выполнена в полном соответствии с требованиями, студент представил все задания практической работы с небольшими погрешностями в выполнении на персональном компьютере, на защите практической работы затруднялся при ответах на некоторые вопросы, нуждался в уточняющих вопросах и подсказках со стороны преподавателя.
3	Средний	Практическая работа выполнена в соответствии с требованиями, студент представил все задания практической работы с существенными погрешностями в выполнении на персональном компьютере, неспособен правильно интерпретировать полученные результаты, на защите затруднялся и/или не ответил на большинство вопросов, нуждался в уточняющих вопросах и подсказках со стороны преподавателя.
2	Ниже среднего	Студент несамостоятельно выполнил практическую работу, неспособен пояснить выполнение ни одного задания, не ответил ни на один контрольный вопрос на защите.

ПРАКТИЧЕСКАЯ РАБОТА №10
РАЗРАБОТКА ПРИЛОЖЕНИЯ MDI

Цель работы: сформировать практические умения создания многооконного приложения MDI средствами среды программирования **VisualStudio**.

Формируемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.

В результате освоения практических заданий обучающийся должен

- уметь: использовать языки программирования, строить логически правильные и эффективные программы
- знать: объектно-ориентированную модель программирования, основные алгоритмические конструкции.

Материально-техническое обеспечение занятия: персональный компьютер, среда программирования **Visual Studio**.

Практические задания и методические указания

Задание:

Создайте приложение, демонстрирующее особенности работы с несколькими окнами.

1. Запустите среду программирования **VisualStudio**. Создайте новое **Приложение WindowsForms**. Имя проекта и приложения **MDIApp**.
2. Многооконный интерфейс **MDI** (Multiple Document Interface) позволяет в одном приложении работать одновременно с несколькими документами или с разными представлениями одного и того же документа. Окна, отображающие содержимое документов внутри главного окна **MDI-приложения**, называются **MDI-окнами**.

   Любое MDI-приложение содержит **меню Windows**, предназначенное для управления окнами, отображающими различные документы или различные представления одного и того же документа. Как правило, в меню Windows есть строки **Cascade** и **Tile**, с помощью которых пользователь может упорядочить MDI-окна, расположив их с перекрытием (друг за другом) или рядом друг с другом. В этом меню могут быть и другие строки, управляющие расположением MDI-окон.

   Ниже разделительной черты в меню Windows обычно находятся строки, обозначающие отдельные MDI-окна. Если выбрать одну из этих строк, указанное окно будет активизировано и показано на первом плане.

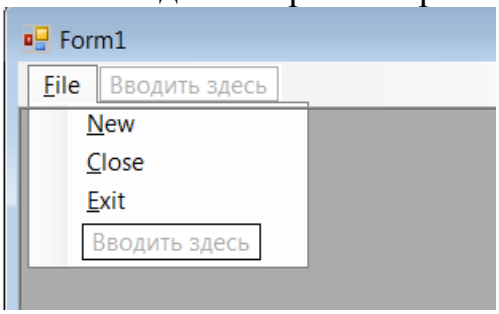
   Меню Windows может быть разным в различных MDI-приложениях. Однако в любом случае это меню позволяет пользователю автоматически упорядочить расположение MDI-окон и выбрать нужное окно из списка для активизации.

   Двигая MDI-окна при помощи заголовка, Вы не сможете переместить их за пределы главного окна приложения. В то же время MDI-окна можно делать активными, при этом заголовок активного MDI-окна выделяется цветом.

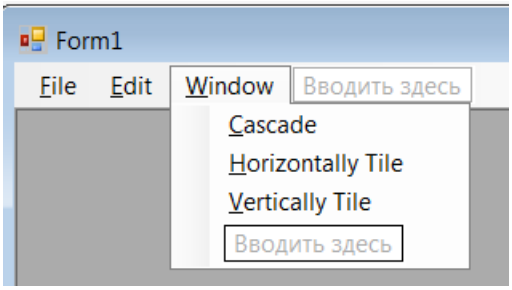
   Каждое MDI-окно имеет, как правило, **системное меню** и **кнопки изменения размера**. С помощью системного меню пользователь может изменять размеры или перемещать MDI-окно (строки Restore, Move, Size, Maximize, Minimize), закрывать окно (строка Close), передавать фокус ввода от одного MDI-окна другому (строка Next Window).

   Если MDI-окно сворачивается в значок (минимизируется), то этот значок располагается в нижней части главного окна приложения.

3. Выделите форму, на панели **Свойства** найдите свойство **IsMDIContainer** и установите значение этого свойства равным **True**. В результате главное окно приложения превратится в контейнер для дочерних MDI-окон.
4. Добавьте в приложение главное меню. Для этого перетащите из панели элементов системы **Visual Studio** значок главного меню **MenuStrip**.
5. Создайте меню **File** со строками **New**, **Close** и **Exit**. С помощью строки **New** этого меню мы будем создавать новые MDI-окна с редактором текста на базе элемента управления **RichTextBox**. Строка **Close** предназначена для закрытия MDI-окон, а строка **Exit** — для завершения работы приложения.



6. Создайте меню **Edit** со строками **Copy** и **Paste.**
7. Создайте меню **Window**, предназначенное для управления MDI-окнами. Строка **Cascade** предназначена для каскадного расположения открытых MDI-окон, а строки **HorizontallyTile** и **VerticallyTile** — для размещения окон рядом друг с другом в горизонтальном и вертикальном направлении, соответственно.



8. После создания меню **Window** выделите его, на панели **Свойства** установите значение свойства **MdiWindowListItem** равным **windowToolStripMenuItem**. При этом во время работы приложения в меню **Window** будут автоматически добавлены строки списка открытых MDI-окон. В этом меню строки активных окон будут отмечены.
9. MDI-окна обычно используются для отображения различных документов или различных представлений одного и того же документа. В приложении **MDIApp**MDI-окна будут содержать внутри себя элемент управления **RichTextBox**, т.е. редактор текста.
10. Добавьте в проект приложения новую форму, которая будет играть роль шаблона для создания дочерних MDI-окон. Для добавления формы щелкните правой клавишей мыши в окне **Обозреватель решений** по имени проекта **MDIApp** и выберите в контекстном меню **Добавить** строку **Форму Windows**. После этого на экране появится диалоговое окно **Добавление нового элемента**, в котором нужно выбрать значок **ФормаWindows Form** и

затем нажать кнопку **Открыть**. В результате в окне дизайнера форм появится новая форма **Form2**.

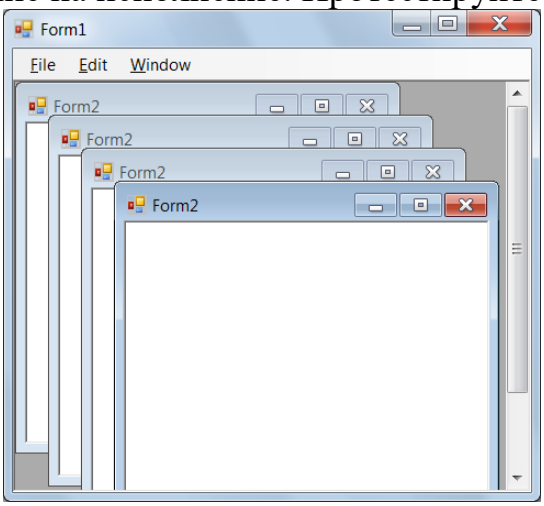
11. Так как это окно будет предназначено для редактирования текста, перетащите в него из панели элементов значок элемента управления **RichTextBox**. Чтобы окно редактора текста **RichTextBox** заполнило собой клиентскую часть MDI-окна, установите значение свойства **Dock** равным **Fill**. Кроме того, проследите, чтобы свойство **Anchor** было равно **Top**, **Left**. Окно элемента управления **RichTextBox** будет заполнять клиентскую область MDI-окна при любом изменении размеров последнего.
12. Напишите программный код, создающий MDI-окна. Этот код должен получать управление, когда пользователь выбирает строку **New** из меню **File**. Добавьте следующий обработчик события **Click** для строки **New** меню **File**:

```
private void newToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form2 mdiChild = new Form2();
    mdiChild.MdiParent = this;
    mdiChild.Show();
}
```

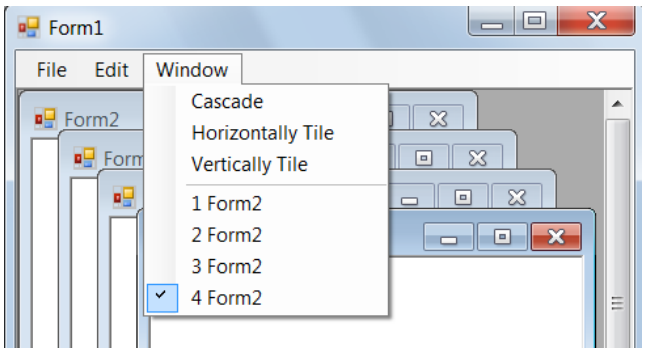
Сначала создается новая форма как объект класса **Form2**, а затем выполняется сохранение ссылки на эту форму в переменной **mdiChild**. Свойство **MdiParent** этого окна должно содержать ссылку на родительское окно приложения MDI, поэтому в него записывается ссылка на объект класса **Form1**, используя ключевое свойство **this**.

Для того чтобы MDI-окно появилось на экране, его необходимо отобразить явным образом при помощи метода **Show**.

13. Запустите приложение на исполнение. Протестируйте его работу.



14. Создав с помощью меню **File** несколько дочерних MDI-окон, раскройте меню **Window** и убедитесь, что в нем появилось несколько новых строк. С помощью этих строк можно выдвинуть на передний план любое нужное MDI-окно.



15. Чтобы пользователь мог упорядочивать дочерние MDI-окна, в приложении предусмотрены строки **Cascade**, **HorizontallyTile**и **VerticallyTile** в меню **Window**. Добавить в исходный текст приложения обработчики событий данных пунктов меню:

```
private void cascadeToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.LayoutMdi(MdiLayout.Cascade);
}

private void horizontallyTileToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.LayoutMdi(MdiLayout.TileHorizontal);
}

private void verticallyTileToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.LayoutMdi(MdiLayout.TileVertical);
}
```

Все они вызывают метод **LayoutMdi** родительской формы, передавая ему в качестве параметра одну из констант, определяющих нужный тип упорядочивания. Это константы **MdiLayout.Cascade**, **MdiLayout.TileHorizontal**и **MdiLayout.TileVertical**.

16. Реализуйте операцию копирования выделенного фрагмента текста из активного MDI-окна в **Clipboard** (буфер обмена). Для этого добавьте следующий обработчик событий к строке **Copy** меню **Edit:**

```
private void copyToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form activeChild = this.ActiveMdiChild;

    if (activeChild != null)
    {
        RichTextBox editBox = (RichTextBox)activeChild.ActiveControl;

        if (editBox != null)
        {
            Clipboard.SetDataObject(editBox.SelectedText);
        }
    }
}
```

Здесь вначале определяется идентификатор активного MDI-окна, извлекая его из свойства **ActiveMdiChild** родительского окна приложения MDI. Этот идентификатор сохраняется в переменной **activeChild**.

Если в приложении нет активного окна, то приведенный выше обработчик событий завершает свою работу, не выполняя никаких других действий.

Определив идентификатор активного MDI-окна, обработчик события получает идентификатор активного элемента управления, расположенного внутри этого окна. Для этого используется свойство **activeChild.ActiveControl**, хранящее ссылку на активный элемент управления.

MDI-окно содержит только один элемент управления— редактор **RichTextBox**. Поэтому полученная ссылка преобразуется явным образом к типу **RichTextBox** и сохраняется в переменной **editBox** для дальнейшего использования.

Теперь идентификатор редактора текста получен и можно переписать выделенный в его окне фрагмент текста в универсальный буфер обмена **Clipboard**. Запись в **Clipboard** осуществляется методом **Clipboard.SetDataObject**. В качестве параметра методу **Clipboard.SetDataObject** передается выделенный текст, извлеченный из редактора текста с помощью свойства **editBox.SelectedText**.

17. Дополните приложение кодом, необходимым для вставки данных из универсального буфера обмена **Clipboard** в активное MDI-окно. Добавьте следующий обработчик события для строки **Paste** меню **Edit:**

```
private void pasteToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form activeChild = this.ActiveMdiChild;

    if (activeChild != null)
    {
        RichTextBox editBox = (RichTextBox)activeChild.ActiveControl;

        if (editBox != null)
        {
            IDataObject data = Clipboard.GetDataObject();

            if (data.GetDataPresent(DataFormats.Text))
            {
                editBox.SelectedText =
                  data.GetData(DataFormats.Text).ToString();
            }
        }
    }
}
```

Этот обработчик вначале определяет идентификатор активного MDI-окна, а затем, пользуясь этим идентификатором и свойством **ActiveControl**, получает идентификатор редактора текста **RichTextBox**.

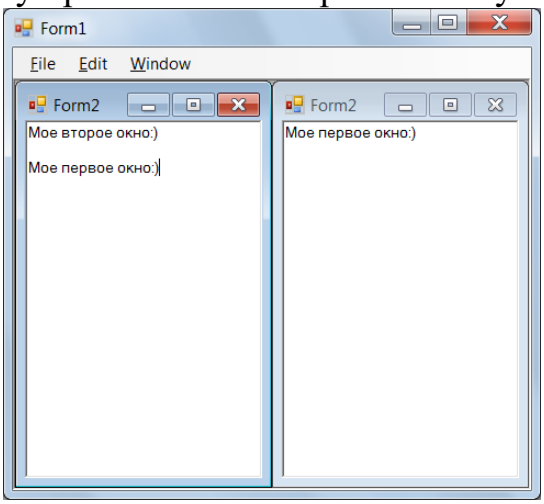
Далее для вставки данных из буфера **Clipboard** метод получает ссылку на интерфейс **IDataObject**, вызывая для этого метод **Clipboard.GetDataObject**. Эта ссылка сохраняется в переменной **data**.

Пользуясь полученной ссылкой на интерфейс **IDataObject**, обработчик события определяет, имеется ли в буфере **Clipboard** текст, который можно было бы вставить в редактор текста. Для этого используется метод **GetDataPresent**. В качестве параметра этому методу передается идентификатор формата текстовых данных **DataFormats.Text**.

Если в буфере **Clipboard** имеются текстовые данные, программа извлекает их при помощи метода **GetData**, а затем преобразует в текстовую строку при помощи метода **ToString**. Далее эта текстовая строка записывается в свой-

ство **SelectedText** нашего редактора текста, благодаря чему и происходит вставка данных из буфера **Clipboard**.

18. Протестируйте работу приложения. Сохраните полученное приложение.



Задания для внеаудиторной самостоятельной работы

Составьте опорный конспект по теме: запишите основные методы, которые используются при создании проекта.

Критерии оценивания на практическом занятии

Балл	Уровень освоения компетенций	Критерии оценивания уровня освоения компетенций
5	Максимальный	Практическая работа выполнена в полном соответствии с требованиями, студент представил все задания практической работы без погрешностей и замечаний, на все вопросы при защите практической работы дал правильные ответы.
4	Высокий	Практическая работа выполнена в полном соответствии с требованиями, студент представил все задания практической работы с небольшими погрешностями в выполнении на персональном компьютере, на защите практической работы затруднялся при ответах на некоторые вопросы, нуждался в уточняющих вопросах и подсказках со стороны преподавателя.
3	Средний	Практическая работа выполнена в соответствии с требованиями, студент представил все задания практической работы с существенными погрешностями в выполнении на персональном компьютере, неспособен правильно интерпретировать полученные результаты, на защите затруднялся и/или не ответил на большинство вопросов, нуждался в уточняющих вопросах и подсказках со стороны преподавателя.
2	Ниже среднего	Студент несамостоятельно выполнил практическую работу, неспособен пояснить выполнение ни одного задания, не ответил ни на один контрольный вопрос на защите.

ПРАКТИЧЕСКАЯ РАБОТА №11
СОЗДАНИЕ ПРОЕКТА С ИСПОЛЬЗОВАНИЕМ КЛАССА

Цель работы:сформировать практические умения создания проекта с использованием классов в среде программирования **VisualStudio**.

Формируемые компетенции:

- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ПК 2.2. Программировать в соответствии с требованиями технического задания.
- ПК 2.3. Применять методики тестирования разрабатываемых приложений.

В результате освоения практических заданий обучающийся должен

- уметь: использовать языки программирования, строить логически правильные и эффективные программы
- знать: объектно-ориентированную модель программирования, понятие классов и объектов, их свойств и методов.

Материально-техническое обеспечение занятия: персональный компьютер, среда программирования **Visual Studio**.

Практические задания и методические указания

Задание 1.

Создайте проект, для решения задачи с использованием классов:

класс «Worker»

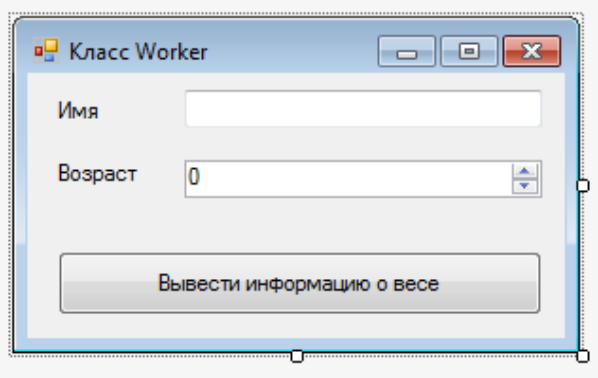
поля: имя; возраст; вес (начальное значение 60).

методы:

«GetEat» - отображается вес.

«SetEat» - если человек что-то съедает, то его вес увеличивается на количество съеденного.

1. Запустите среду программирования **VisualStudio**. Создайте новое **Приложение WindowsForms**. Имя проекта и приложения **ClassApp1**.
2. Создайте интерфейс приложения в соответствии с образцом:



3. Добавьте в проект новый класс **Worker**, для этого перейдите в **Обозреватель решений** и в контекстном меню проекта выполните команду **Добавить – Класс**. В окне **Добавление нового элемента** введите имя **Worker.cs**и нажмите кнопку **Добавить**.
   Класс представляет собой шаблон, по которому определяется форма объекта. В нем указываются данные и код, который будет оперировать этими данными. В C# используется спецификация класса для построения объектов, которые являются экземплярами класса. При этом очень важно подчеркнуть, что класс является логической абстракцией. Физическое представление класса появится в оперативной памяти лишь после того, как будет создан объект этого класса.
   При определении класса объявляются данные, которые он содержит, а также код, оперирующий этими данными. Если самые простые классы могут содержать только код или только данные, то большинство настоящих классов содержит и то, и другое.
   Объект можно рассматривать как совокупность свойств, методов и событий, работающий при этом как единое целое.
4. В класс добавьте два общедоступных поля: **Имя** и **Возраст**, а также одно скрытое поле: **Вес**.

```
class Worker
{
    public string имя;
    public int возраст;
    double вес = 60.00;
}
```

5. Для записи и чтения данных из скрытых полей используйте методы. **Метод** – это фрагмент программного кода, имеющий собственное имя и позволяющий значительно облегчить процесс программирования. Добавьте в класс **Worker** метод **SetEat**, который будет отвечать за еду: если человек что-то съест, то его вес должен будет увеличиться на количество съеденного. Если поле вес скрытое, то запись в него возможна и чтение из него тоже не возможно. Для чтения данных из скрытого поля необходимо использовать еще один метод **GetEat**.

```
class Worker
{
    public string имя;
    public int возраст;
    double вес = 60.00;
    public void SetEat(double eda)
    {
        вес += eda;
    }

    public double GetEat()
    {
        return вес;
    }
}
```

6. Далее используйте данные два метода в программе. Заставьте рабочего съесть 2 кг, а затем 3 кг пищи, а потом проверьте его вес. Для этого перейдите к окну формы и выполните двойной щелчок по кнопке **Вывести информацию о весе**. В обработчик события вставьте следующий код:

```
private void button1_Click(object sender, EventArgs e)
{
    Worker worker = new Worker();

    worker.имя = textBox1.Text;
    worker.возраст = (int)numericUpDown1.Value;

    MessageBox.Show(String.Format("Рабочий: {0} \nВозраст: {1} \nВес: {2}",
    worker.имя, worker.возраст, worker.GetEat()));
    worker.SetEat(2);
    worker.SetEat(3);

    MessageBox.Show(String.Format("Рабочий: {0} \nВозраст: {1} \nВес: {2}",
    worker.имя, worker.возраст, worker.GetEat()));

}
```

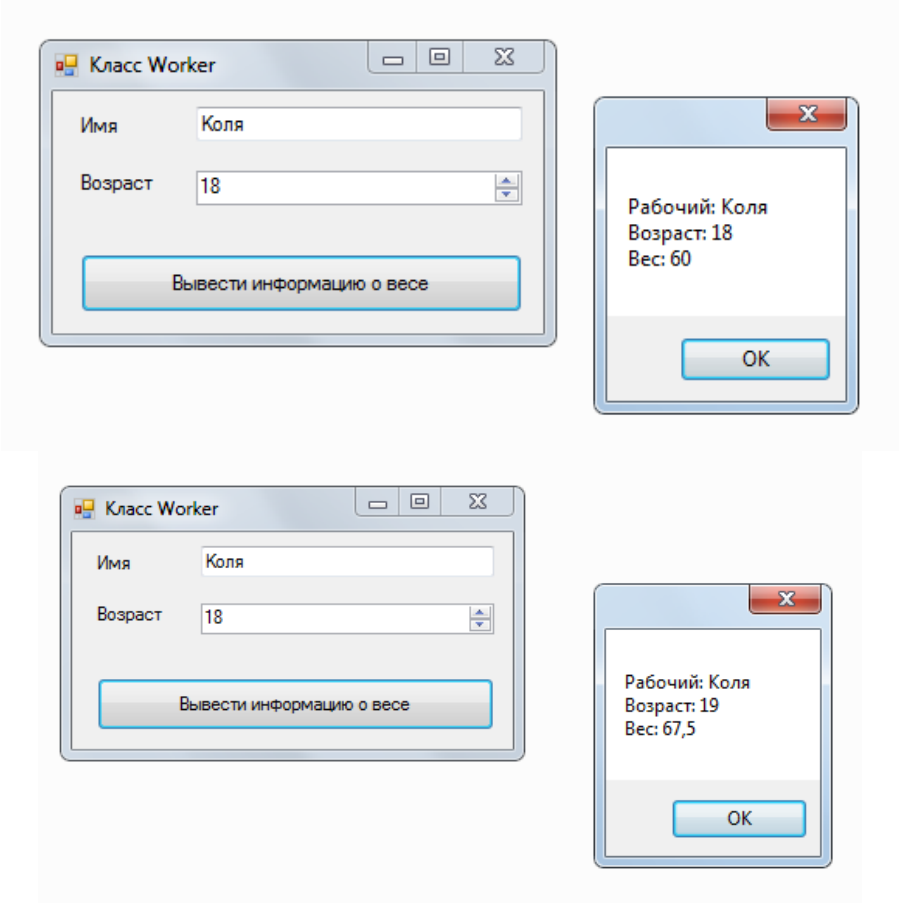
7. Запустите программу на выполнение. Проверьте её работоспособность.
8. Усовершенствуйте метод **SetEat** таким образом, что если рабочий за раз съедает более, чем 10 кг, то его возраст увеличивается на год, а вес увеличивается только наполовину от съеденного. Для этого измените код модуля следующим образом:

```
public void SetEat(double eda)
{
    if (eda > 10)
    {
        возраст++;
        вес += eda / 2;
    }
    else
        вес += eda;
}
```

9. Попросите рабочего съесть 15 кг. Для этого измените программный код кнопки следующим образом:

```
private void button1_Click(object sender, EventArgs e)
{
    Worker worker = new Worker();

    worker.имя = textBox1.Text;
    worker.возраст = (int)numericUpDown1.Value;

    MessageBox.Show(String.Format("Рабочий: {0} \nВозраст: {1} \nВес: {2}",
    worker.имя, worker.возраст, worker.GetEat()));
    worker.SetEat(15);

    MessageBox.Show(String.Format("Рабочий: {0} \nВозраст: {1} \nВес: {2}",
    worker.имя, worker.возраст, worker.GetEat()));

}
```

10. Запустите программу на выполнение. Проверьте её работоспособность.



11. Создайте новый проект **ClassApp2** для решения следующей задачи с использованием классов.

Класс «Человек»

Поля: имя; возраст; вес, настроение (начальное значение=10).

Методы:

- «Информация о человеке» - отображаются имя, возраст и вес.
- «Есть» - если человек что-то съедает, то его вес увеличивается на количество съеденного.
- «Гулять» - настроение увеличивается на 1.
- «Танцевать» - настроение увеличивается на 2.
- «Работать» - настроение уменьшается на 2.

12. Усовершенствуйте метод «Есть» так, чтобы при съедании за один раз более 5 кг, его возраст увеличивался бы на год, а вес увеличивался бы только на половину от съеденного.
13. Усовершенствуйте метод так, чтобы настроение не могло становиться отрицательным числом.
14. Дополните основную программу так, чтобы человек после еды четыре раза погулял и два раза потанцевал.
15. Добавьте в класс метод, который будет возвращать текущее настроение человека.
16. Добавьте в основную программу метод работать 9 раз и выведите настроение пользователя на экран. Измените метод работать таким образом, чтобы настроение никогда не было меньше нуля (т.е. если настроение было 1 и человек поработал, то оно должно стать не меньше 0).
17. Запустите программу на выполнение. Проверьте её работоспособность.

Задание 2.

Создайте класс с использованием свойств.

1. Откройте проект **ClassApp1**.
2. Измените класс так, чтобы изменим код так, чтобы вместо двух методов **SetEat** и **GetEat** появилось одно свойство **Ves. Свойства** – неотъемлемая часть класса. Именно с помощью свойств реализовывается один из «китов» объектно-ориентированного программирования – инкапсуляция.

```
class Worker
{
    public string имя;
    public int возраст;
    double вес = 60.00;

    public double Ves
    {
        get { return вес; }
        set
        {
            if (value > 10)
            {
                возраст++;
                вес += value / 2;
            }
            else
                вес += value;
        }
    }
}
```

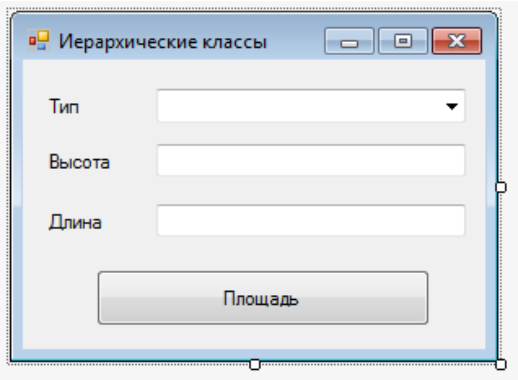
3. Запустите программу. Возможен ли запуск?
4. Измените код кнопки следующим образом:

```
private void button1_Click(object sender, EventArgs e)
{
    Worker worker = new Worker();

    worker.имя = textBox1.Text;
    worker.возраст = (int)numericUpDown1.Value;

    MessageBox.Show(String.Format("Рабочий: {0} \nВозраст: {1} \nВес: {2}",
    worker.имя, worker.возраст, worker.Ves));
    worker.Ves=15;

    MessageBox.Show(String.Format("Рабочий: {0} \nВозраст: {1} \nВес: {2}",
    worker.имя, worker.возраст, worker.Ves));

}
```

5. Запустите программу на выполнение. Проверьте её работоспособность.
6. Создайте новый проект **ClassApp3** работы с приведенным ниже классом. Свойства определите самостоятельно.
   Класс «Покупатель»
   Поля класса: название покупаемого продукта, цена, количество, кошелек, настроение (начальное значение=10).
   Методы класса:
   - «Информация о покупке» - отображается название купленного продукта, его количество и стоимость покупки.
   - «Купить» - если человек что-то покупает, то стоимость покупки увеличивается, а наличность кошелька уменьшается.
   - «Посмотреть настроение» - отслеживайте само настроение: если настроение больше 15, то оно «бодрое», если от 5 до 15, то – нормальное, если от -10 до 5, то – плохое, если меньше -10, то – депрессивное.
7. Усовершенствуйте метод «Купить» так, чтобы выполнялась проверка: хватит ли денег в кошельке для совершения покупки.
8. Добавьте связь с настроением: если денег хватило на совершение покупки, то настроение увеличивается на 50% от суммы покупки, а если не хватило – то настроение уменьшается на количество единиц, равное количеству денег, которых не хватило для совершения покупки.
9. Запустите программу на выполнение. Проверьте её работоспособность.

Задание 3.
Создайте классы, иерархически связанные друг с другом.

1. Создайте новое **Приложение WindowsForms**. Имя проекта и приложения **ClassApp4**.
2. Создайте интерфейс приложения в соответствии с образцом:



3. Создайте классы «Фигура» и «Четырехугольник» так, чтобы «Фигура» был «предком», а «Четырехугольник» - потомком. Добавьте класс «Фигура»:

```
class Figura
{
    protected double visota;
    protected double dlina;
    protected Figura(double v, double d)
    {
        visota = v;
        dlina = d;
    }
    public string pl()
    {
        return Convert.ToString(visota * dlina);
    }
}
```

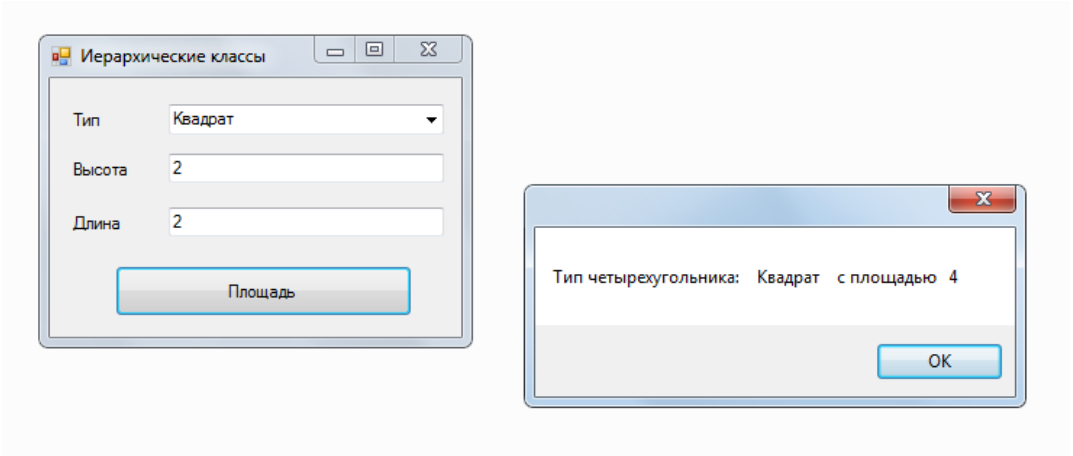
4. **Наследование** – одна из основных характеристик объектно-ориентированного программирования. Когда новый класса становится потомком существующего, он наследует все незакрытые методы, свойства и поля своего предка. Новый класса затем можно расширять, добавляя или заменяя эти методы, поля и свойства. Все это, в конечном счете, позволяет значительно упростить программирование путем многократного использования кода. Добавьте класс «Четырехугольник».

```
class Chetug:Figura
{
    public string tip;
    public Chetug(string t, double v, double d): base (v,d)
    {
        tip = t;
    }
    public string inf()
    {
        return "Тип четырехугольника:    " + tip;
    }
}
```

5. Используйте данные классы в программе. Напишите программный код для кнопки **Площадь**.

```
private void button1_Click(object sender, EventArgs e)
{
    Chetug ct = new Chetug(comboBox1.Text, double.Parse(textBox1.Text), double.Parse(textBox2.Text));
    MessageBox.Show(ct.inf() + "    с площадью    " + ct.pl());
}
```

6. Запустите программу на выполнение. Проверьте её работоспособность.



Задания для внеаудиторной самостоятельной работы

Ответьте на вопросы письменно в тетради:

- Что такое класс?
- Что такое объект?
- Назовите основные принципы ООП и дайте каждому из них характеристику.
- Для чего нужны свойства класса?

Критерии оценивания на практическом занятии

Балл	Уровень освоения компетенций	Критерии оценивания уровня освоения компетенций
5	Максимальный	Практическая работа выполнена в полном соответствии с требованиями, студент представил все задания практической работы без погрешностей и замечаний, на все вопросы при защите практической работы дал правильные ответы.
4	Высокий	Практическая работа выполнена в полном соответствии с требованиями, студент представил все задания практической работы с небольшими погрешностями в выполнении на персональном компьютере, на защите практической работы затруднялся при ответах на некоторые вопросы, нуждался в уточняющих вопросах и подсказках со стороны преподавателя.
3	Средний	Практическая работа выполнена в соответствии с требованиями, студент представил все задания практической работы с существенными погрешностями в выполнении на персональном компьютере, неспособен правильно интерпретировать полученные результаты, на защите затруднялся и/или не ответил на большинство вопросов, нуждался в уточняющих вопросах и подсказках со стороны преподавателя.
2	Ниже среднего	Студент несамостоятельно выполнил практическую работу, неспособен пояснить выполнение ни одного задания, не ответил ни на один контрольный вопрос на защите.

ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ОБУЧЕНИЯ

Основная литература:

1. Голицына, О.Л. Программирование на языках высокого уровня: учебное пособие для СПО / О. Л. Голицына, И. И. Попов. - М: ФОРУМ, 2014. - 495с.
2. Семакин, И.Г. Основы алгоритмизации и программирования: учебник для СПО / И. Г. Семакин, А. П. Шестаков. - М.: Академия, 2013. - 300с.
3. Казанский, А. А. Программирование на visual c# 2013: учебное пособие для СПО / А. А. Казанский. — М.: Издательство Юрайт, 2017. — 191с. - Режим доступа: www.biblio-online.ru/book/A12DB344-78CA-4224-99E4-EDEB728A5578.
4. Черпаков, И. В. Основы программирования: учебник и практикум для СПО / И. В. Черпаков. — М.: Издательство Юрайт, 2017. — 219с. —Режим доступа: www.biblio-online.ru/book/F79BE55A-C6F1-439D-9ED5-0D78A50B403F.

Дополнительная литература:

1. Затонский, А.В. Программирование и основы алгоритмизации: теоретические основы и примеры реализации численных методов: учебное пособие для вузов / А. В. Затонский, Н. В. Бильфельд. - М.: РИОР: ИНФРА-М, 2014. - 165с.
2. Ишкова, Э.А. C#. Начала программирования: учебник дл вузов / Э. А. Ишкова. - М.: БИНОМ, 2013. - 333 с.
3. Огнева, М. В. Программирование на языке c++: практический курс: учебное пособие для СПО / М. В. Огнева, Е. В. Кудрина. — М.: Издательство Юрайт, 2017. — 335 с. — Режим доступа: www.biblio-online.ru/book/B76AB4A4-7623-4842-9136-B6ADC57B90BC.
4. Павловская, Т.А. C#. Программирование на языке высокого уровня: учебник для вузов / Т. А. Павловская. - М. и др.: Питер, 2014. - 432с.
5. Трофимов, В. В. Алгоритмизация и программирование: учебник для академического бакалавриата / В. В. Трофимов, Т. А. Павловская; под ред. В. В. Трофимова. — М.: Издательство Юрайт, 2017. — 137с. - Режим доступа: www.biblio-online.ru/book/B08DB966-3F96-4B5A-B030-E3CD9085CED4.

Internet – ресурсы

№ п/п	Наименование ресурса	Электронный адрес
1	Интернет-Университет Информационных технологий	http://www.intuit.ru/
2	Каталог библиотеки учебных курсов MSDM	http://msdn.microsoft.com/ru-ru/gg638594
3	Академия мобильных приложений	http://appinvent.ru/

А.А.Дроздова

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ
Основы объектно-ориентированного программирования

Методические рекомендации
к практическим занятиям
для студентов очной формы обучения
по специальности

09.02.04 Информационные системы (по отраслям)

ЧПОУ Вологодский кооперативный колледж
160014,г.Вологда,ул.Горького,93