ФГБОУ ВПО «Воронежский государственный технический университет»
Кафедра компьютерных интеллектуальных технологий проектирования

112-2015

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторным работам № 1-4 по дисциплине «Среды визуального программирования» для студентов направления 09.03.02 «Информационные системы и технологии» (профиль «Информационные системы и технологии в машиностроении») очной формы обучения

Воронеж 2015

Составители: канд. техн. наук А.Н. Юров,
канд. техн. наук А.В. Бредихин

УДК 004.9

Методические указания к лабораторным работам № 1-4 по дисциплине «Среды визуального программирования» для студентов направления 09.03.02 «Информационные системы и технологии» (профиль «Информационные системы и технологии в машиностроении») очной формы обучения / ФГБОУ ВПО «Воронежский государственный технический университет»; сост. А.Н. Юров, А.В. Бредихин. Воронеж, 2015. 31 с.

Методические указания содержат материал по созданию приложений на языке программирования C# в среде визуальной разработки Visual Studio, а также практические задачи и перечень заданий для выполнения лабораторных работ по дисциплине «Среды визуального программирования».

Предназначены для студентов 2 курса.

Методические указания подготовлены в электронном виде в текстовом редакторе MS Word 2013 и содержатся в файле lPart1.docx.

Табл. 1. Ил. 16. Библиогр.: 9 назв.

Рецензент канд. физ.-мат. наук, доц. Н.А. Тюкачев

Ответственный за выпуск зав. кафедрой д-р техн. наук, проф. М.И. Чижов

Издается по решению редакционно-издательского совета Воронежского государственного технического университета

ВВЕДЕНИЕ

В последнее время становится значимым разработка приложений в средах визуального программирования в связи с ростом производительности вычислительных средств ЭВМ как стационарных, так и мобильных сенсорных устройств (КПК, планшеты). Способ создания программ для ЭВМ путем манипулирования графическими объектами вместо написания кода вручную является достаточно доступным и простым.

Такие визуальные средства являются средами программирования, в которые интегрированы библиотеки готовых объектов и методов для выполнения общих для большинства приложений задач, в частности задачи наглядного и стандартизованного получения и отображения информации.

Одной из таких многочисленных сред разработки программного обеспечения является Visual Studio – комплексное решение компании Microsoft, включающее набор инструментальных средств, проектных шаблонов, инструментов для тестирования и т.д. Продукт поддерживает ряд языков программирования, которые могут быть объединены в рамках .NET концепции, кроме того доступен базовый фундамент для создания приложений на основе Windows Forms, WPF и ряд других.

В данных методических указаниях представлен материал по созданию консольных и графических приложений на объектно-ориентированном языке C# в среде Visual Studio 2012. Все примеры могут быть использованы в иных средах разработки, поддерживающие написание программ на C# в операционных системах, включая Windows, Linux и другие.

ЛАБОРАТОРНАЯ РАБОТА № 1
СРЕДЫ ВИЗУАЛЬНОГО ПРОГРАММИРОВАНИЯ

Цель работы: разработка консольного приложения в среде визуального программирования Visual Studio.

Задачи и требования к выполнению:

1.Изучить среду разработки Visual Studio, знать особенности установки среды и настройки проектов.

2.Изучить структуру консольного проекта.

3.Собрать консольный проект в Visual Studio, ознакомиться с отладочными средствами интегрированной среды при разработке приложений.

Теоретические сведения

Среда визуальной разработки программного обеспечения представляет собой решение по созданию кода, в которой наиболее распространенные блоки фрагментов текста на том или ином языке программирования представлены в виде графических объектов. Применяются в основном для создания прикладных программ и разработки графического интерфейса пользователя (GUI).

Преимущества:

-быстрота разработки;

-лёгкость освоения;

-стандартизация внешнего вида программ.

Недостатки:

-как правило, привязка к конкретной среде разработки связанное с проблематичность перехода на другую среду разработки;

-затруднённое использование нестандартных компонентов;

-наличие недокументированных особенностей компонент.

На рис. 1-4 представлены визуальные редакторы интерфейсов в работе для некоторых интегрированных сред разработки программного обеспечения.

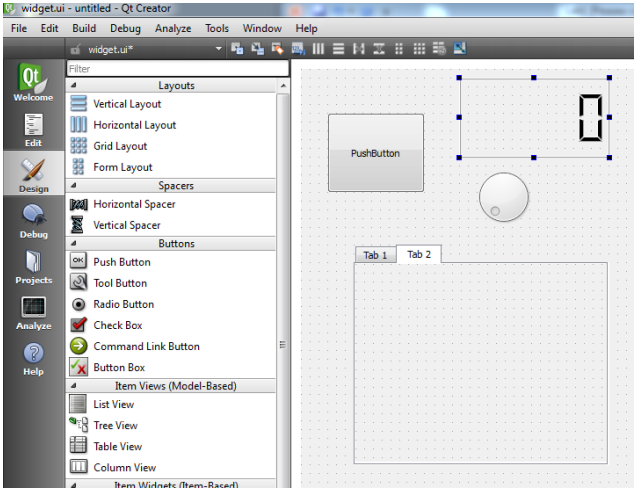


Рис. 1. QT Creator (Модуль Designer)

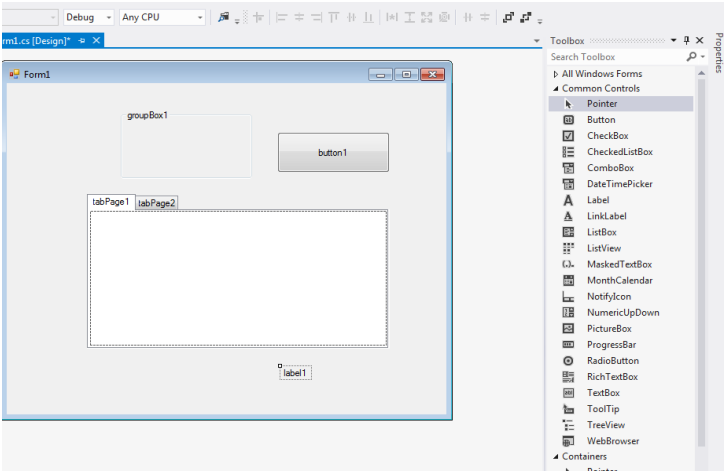


Рис. 2. Visual Studio 2012(Модуль Designer Windows Form)

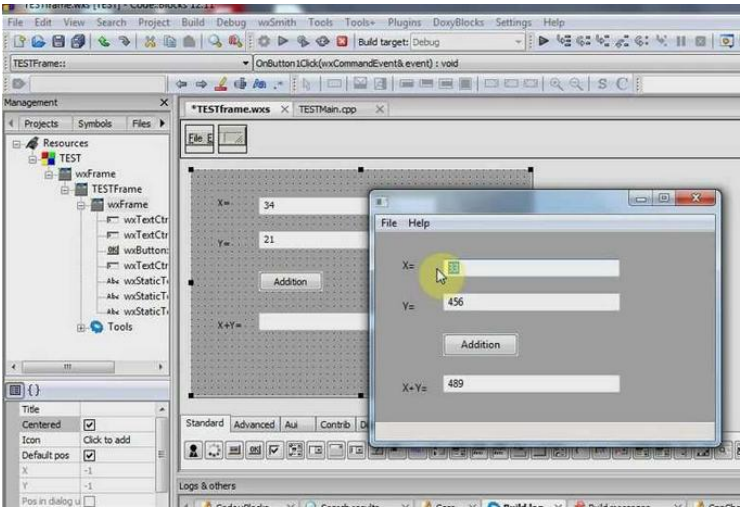


Рис. 3. wxWidgets интерфейс Code::Blocks

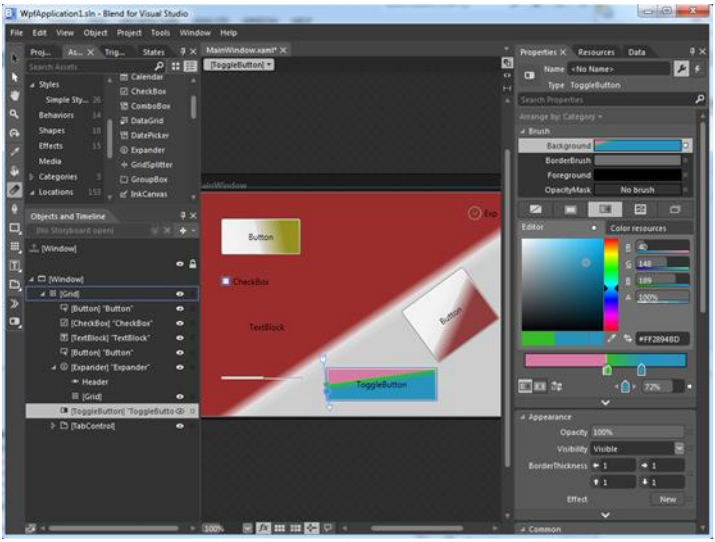


Рис. 4. Visual Studio 2012(Модуль Blend)

Интегрированная среда разработки Visual Studio позволяет создавать как консольные приложения, так и

приложения с графическим интерфейсом. Основными преимуществами данного IDE решения являются:

-работа в программной среде .NET Framework;

-быстрота разработки графического интерфейса пользователя (GUI);

-логичная структура построения классов;

-подробное документирование и обширная база примеров Online;

-межязыковое взаимодействие в виртуальной среде .Net.

Установить программный продукт можно по следующей ссылке с указанного Интернет-ресурса (актуально на момент публикации методических указаний): http://www.visualstudio.com/downloads/download-visual-studio-vs

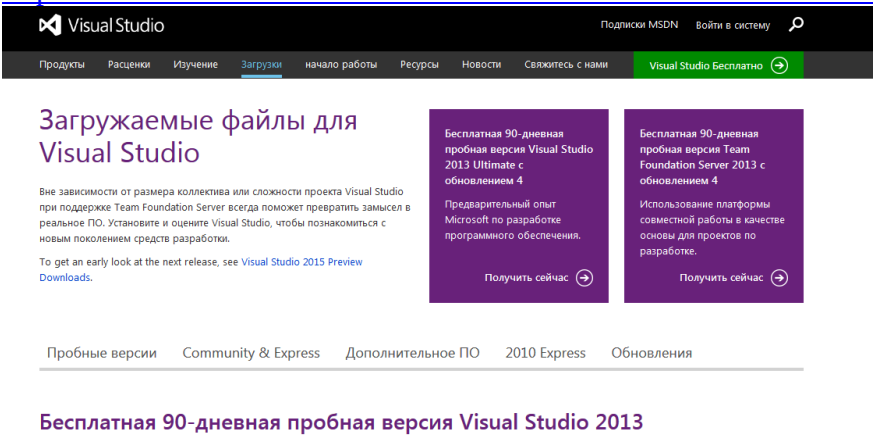


Рис. 5. Интернет-ресурс

В случае недоступности ресурса рекомендуется воспользоваться поиском в любом браузере, получить доступ к онлайн-установщику и установить все компоненты среды разработки ПО.

На рис. 6-7 представлены окна инсталлятора Visual Studio 2012.

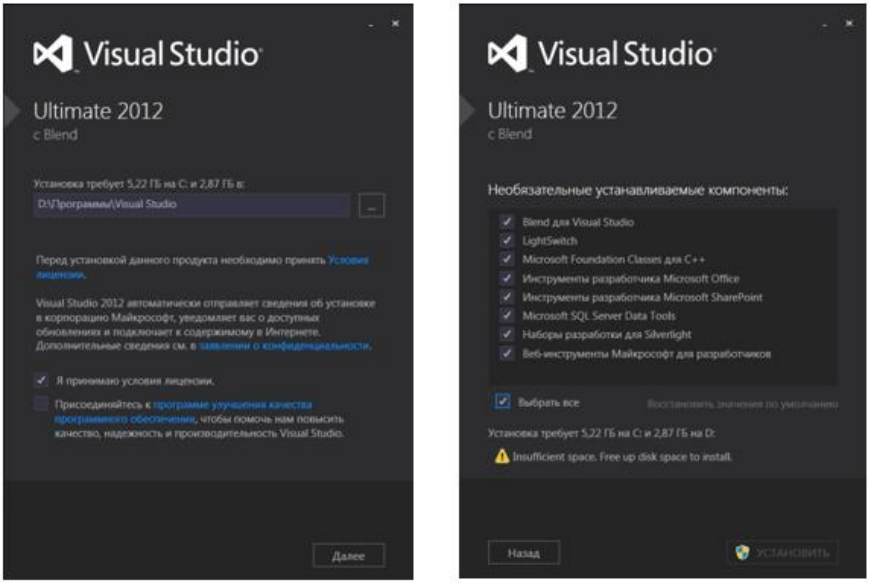


Рис. 6. Выбор раздела для установки среды, а также необходимых инструментов разработчика

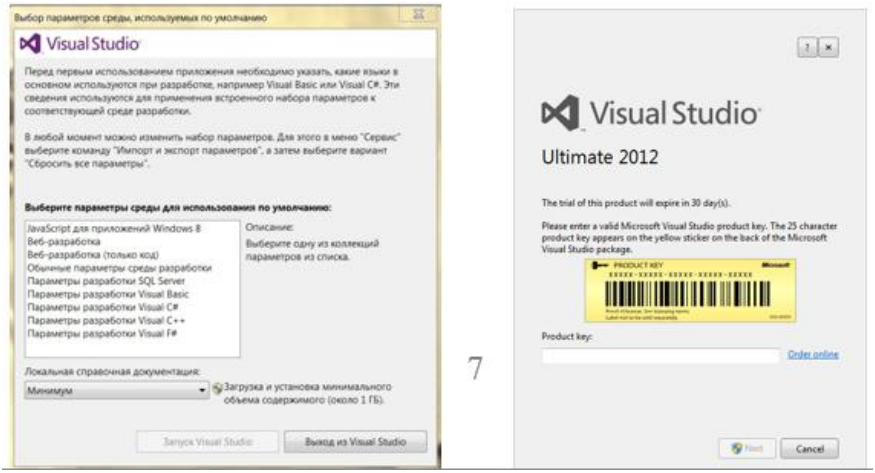
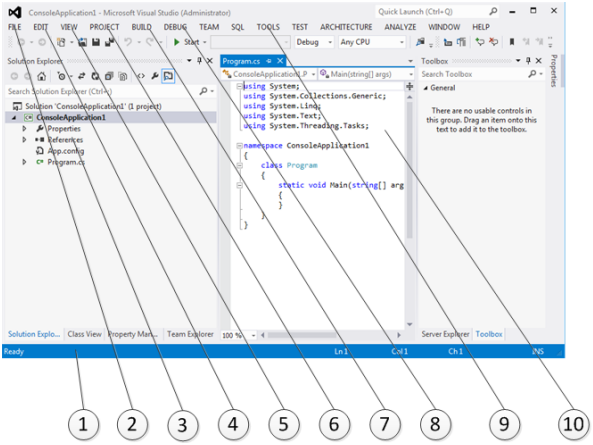


Рис.7. Настройка среды под выбранные языковые средства VS и регистрация продукта

После того, как все операции по установке будут выполнены, можно переходить к созданию проектов, однако предварительно требуется изучить возможности и сервисы, которые есть в IDE Visual Studio. На рис.8 приведено описание элементов среды, как для текстовых элементов управления, так и графических.



1-строка состояния;
2-Создание и запись проектов;
3-Навигация по структуре проекта;
4-Редактирование;
5-Отображение инструментальных окон;
6-Внесение настроек в проекты;
7-Сборка проекта;
8-Отладочные средства VS;
9-Сервисы и инструменты разработчика;
10-Текстовый редактор VS

Рис.8. Элементы управления Visual Studio

Перейдем к построению первого консольного приложения в Visual Studio на языке C#. В таблице представлены команды или решения для выполнения работы.

Языковые конструкции и выражения C#

Команда (инструкция)	Назначение
Console.ReadLine()	Ввод данных
Console.WriteLine ()	Вывод данных
if (), switch()/case, ?	Условные конструкции
for(), while, do…while	Операторы циклов
int[] myArr = new int[5]	Объявления массивов на 5 элементов
&&, ||	Логическое И, ИЛИ

Продолжение таблицы

Команда (инструкция)	Назначение
using	Разрешает использование типов в пространстве имен, поэтому уточнение использования типа в этом пространстве имен не требуется
namespace ConsoleApplication1	Определяется пространство имен приложения

В консольном примере (листинг 1) печатаются целые числа из некоторого массива. На рис. 9 показан результат работы приложения.

Листинг 1. Пример консольного приложения C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
   class Program { static void Main(string[] args)
         {
           // Объявляем массив
            int[] myArr = new int[5];
            // Инициализируем каждый элемент
            //массива вручную
            myArr[0] = 100;
            myArr[1] = 23;
            myArr[2] = 25;
 myArr[3] = 31;
 myArr[4] = 1;
 foreach (int i in myArr)
Console.WriteLine(i);
Console.ReadLine();
           }
        }
   }
```

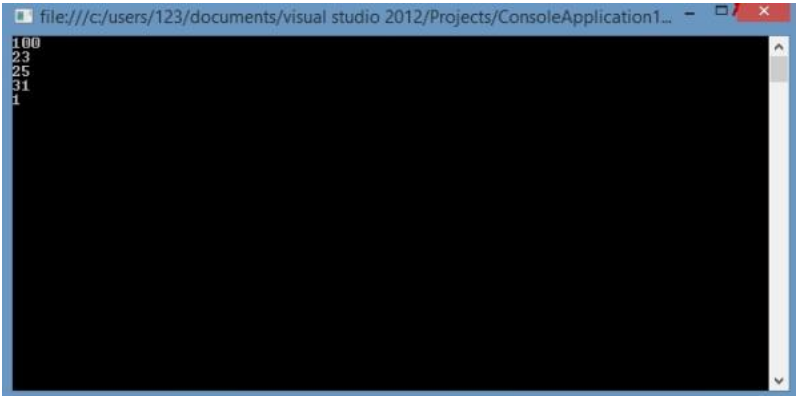


Рис. 9. Результат выполнения программы

Задания на самостоятельную работу:

1.Разработать консольное приложение, в котором производится вычисление скорости, времени и расстоянию. Для недостающих данных предусмотреть ввод с клавиатуры.

2.Разработать консольное приложение для расчета квадратного уравнения по введенным значениям.

ЛАБОРАТОРНАЯ РАБОТА № 2
ОБЪЕКТЫ И КЛАССЫ C#. НАСЛЕДОВАНИЕ

Цель работы: разработать консольное приложение в среде визуального программирования (Visual Studio) согласно заданию.

Задачи и требования к выполнению:

1.Освоить приемы работы с объектами и классами языка C#.

2.Изучить механизм наследования в C#.

3.Рассмотреть работу конструкторов и деструкторов в классах C#.

4.Изучить конструкции по созданию исключений в программном коде на C#.

Теоретические сведения

Класс является основой для создания объектов. В классе определяются данные и код, который работает с этими данными. Объекты являются экземплярами класса.

Методы и переменные, составляющие класс, называются членами класса. При определении класса объявляются данные, которые он содержит, и код, работающий с этими данными. Данные содержатся в переменных экземпляра, которые определены классом, а код содержится в методах.

Определение значений переменных в объекте (переменных экземпляра) происходит в конструкторе. В классе могут быть определены несколько конструкторов.

Члены класса с областью видимости данных public доступны везде за пределами данного класса, с типом доступа protected – внутри членов данного класса и производных, с типом доступа private - только для других членов данного класса. Тип доступа internal применяется для типов, доступных в пределах одной сборки. В качестве примера создания класса на листинге 2 приведен код по описанию некоторой модели детали, а на рис.10 разработанный класс представлен схематически.

Листинг 2. Пример реализации класса C#

```
class CadModel
    {
        public string Name;
        private int Material;
        protected int Type;
        public CadModel(int M, int T, string N)
        {
            Material = M;
            Type = T;
            Name = N;
        }
        public int GetMaterial() { return Material; }}
```

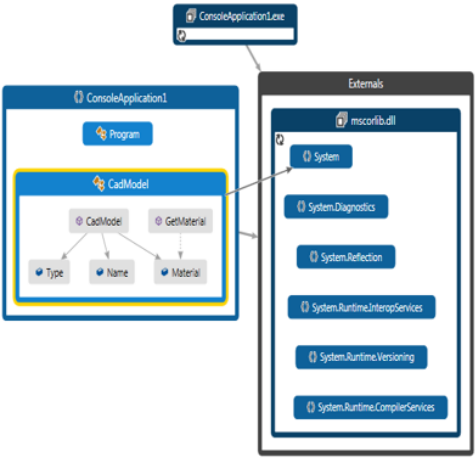


Рис. 10. Схематическое представление класса

Иногда в классе требуется не только объявить данные, но и присвоить им начальные значения. В некоторых случаях перед завершением работы приложения, наоборот, необходимо очистить память от имеющихся данных. Для этих целей в C# есть специальные методы.

Конструктор класса – метод для инициализации объекта при его создании. Он имеет то же имя, что и его класс. В конструкторах тип возвращаемого значения не указывается явно. Конструкторы используются для присваивания начальных значений переменным экземпляра, определенным классом, и для выполнения любых других процедур инициализации, необходимых для создания объекта.

Все классы имеют конструкторы независимо от того, определен он или нет. По умолчанию в C# предусмотрено наличие конструктора, который присваивает нулевые значения всем переменным экземпляра (для переменных обычных типов) и значения null (для переменных ссылочного типа).

```
имя_класса(список_параметров){тело_конструктора}
```

Деструктор – метод, вызывающийся автоматически при уничтожении объекта класса (непосредственно перед “сборкой мусора”). Деструктор не имеет параметров и возвращаемого значения.

```
~имя_класса() {тело_деструктора}
```

Наследование - это свойство, с помощью которого один объект может приобретать свойства другого. При этом поддерживается концепция иерархической классификации, имеющей направление сверху вниз.

Используя наследование, объект должен определить только те качества, которые делают его уникальным в пределах своего класса. Он может наследовать общие атрибуты от своих родительских классов. На листинге 3 приведен фрагмент кода, в котором реализовано наследование классов, а на рис.11 показано взаимодействие классов схематически.

Листинг 3. Пример наследования в C#

```
//Базовый класс
    class CadModel
    {
        public string Name;
        private int Material;
        protected int Type;
        //public CadModel(){}
        public CadModel(int M, int T, string N)
        {
            Material = M;
            Type = T;
            Name = N;
        }
      public int GetMaterial() {
      return Material; }
    }
    //Класс-наследник
    class Block :  CadModel
    {
```

```
        public Block():base(5, 5, "balka")
        {
        }
        private int Size;
    }
```

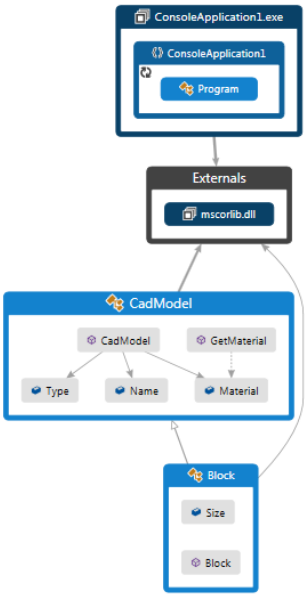


Рис. 11. Схема наследования классов

Обработка исключений и проверки на ввод данных осуществляется с помощью инструкций try и catch. В .NET Runtime фундаментальным средством обработки ошибок являются исключения. Этот способ надежнее, чем возвращаемое значение, поскольку исключение невозможно проигнорировать. Для работы с исключениями код должен быть организован особым образом. Часть кода, в которой может быть сгенерировано исключение, должна быть помещена в блок try, а в блок catch следует поместить код, обрабатывающий исключения. В следующем листинге приведена работа с исключениями.

Листинг 4. Работа с исключениями в C#

```
    using System;
```

```
class Test {
   static int Zero = 0;
   public static void Main() {
      try {
         int j = 22 / Zero;
      }
     catch (Exception e) {
         Console.WriteLine("Exception " +
e.Message);
      }
      Console.WriteLine("After catch");
   }
}
```

Задания на самостоятельную работу:

1.Разработать приложение, которое считывает целое число, являющее днем года (1 до 365), и сохраняет его в целочисленной переменной. Предусмотреть в программе ошибочный ввод (символы вместо чисел). Далее выполнить преобразование числа в название дня и месяца. Пример 40 ->9 февраля.

2.Разработать класс, который описывает простейшие геометрические фигуры, линии и точки, а также учитывает их атрибуты (цвет, размер точки, толщина линии и т.д.)

ЛАБОРАТОРНАЯ РАБОТА № 3
ИНТЕРФЕЙСЫ C#

Цель работы: разработать консольное приложение в среде визуального программирования (Visual Studio) согласно заданию.

Задачи и требования к выполнению:

1.Изучить работу с интерфейсами,

2.Изучить наследование интерфейсов в языке C#.

3.Познакомиться с приемами создания структур в C#.

Теоретические сведения

Интерфейс (interface) представляет собой не более чем просто именованный набор абстрактных членов. Абстрактные методы являются чистым протоколом, поскольку не имеют никакой стандартной реализации. Конкретные члены, определяемые интерфейсом, зависят от того, какое поведение моделируется с его помощью.

В интерфейсе ни у одного из методов не должно быть тела. Это означает, что в интерфейсе вообще не предоставляется никакой реализации. В нем указывается только, что именно следует делать, но не как это делать. Как только интерфейс будет определен, он может быть реализован в любом количестве классов. Кроме того, в одном классе может быть реализовано любое количество интерфейсов.

Помимо методов, в интерфейсах можно также указывать свойства, индексаторы и события. Интерфейсы не могут содержать члены данных. В них нельзя также определить конструкторы, деструкторы или операторные методы. Кроме того, ни один из членов интерфейса не может быть объявлен как static. На листинге 5 приведен пример консольного приложения по работе с интерфейсами в C#, в котором рассматриваются математические операции с 2-мя переменными.

Листинг 5. Пример использования интерфейса

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
    // Создаем два интерфейса
    public interface IMathOperation
    {
        // Определяем набор абстрактных методов
        int Sum();
        int Sub();
    }
```

```
    public interface ISqrSqrt
    {
        int Sqr(int x);
        int Sqrt(int x);
    }
    class A : IMathOperation
    {
        int My_x, My_y;
        public int x
        {
            set { My_x = value; }
            get { return My_x; }
        }
        public int y
        {
            set { My_y = value; }
            get { return My_y; }
        }
        public A() { }
        public A(int x, int y)
        {
            this.x = x;
            this.y = y;
        }
        // Реализуем методы интерфейса
        public virtual int Sum()
        {
            return x + y;
        }
        public int Sub()
        {
            return x - y;
        }
    }
    // Данный класс унаследован от класса A, но при
этом в нем не нужно
    // заново реализовывать интерфейс, но при этом
можно переопределить
    // некоторые его методы
    class Aa : A
    {
        public int z;
        public Aa(int z, int x, int y)
            : base(x, y)
```

```
        {
            this.z = z;
        }
        // Переопределим метод Sum
        public override int Sum()
        {
            return base.x + base.y + z;
        }
    }
    // Данный класс унаследован от класса A, и при
этом
    // реализует интерфейс ISqrSqrt
    class Ab : A, ISqrSqrt
    {
        public int Sqr(int x)
        {
            return x * x;
        }
        public int Sqrt(int x)
        {
            return (int)Math.Sqrt((double)(x));
        }
    }
    class Program
    {
        static void Main()
        {
            A obj1 = new A(x: 10, y: 12);
            Console.WriteLine("obj1: ");
            Console.WriteLine("{0} + {1} = {2}",
obj1.x, obj1.y, obj1.Sum());
            Console.WriteLine("{0} - {1} = {2}",
obj1.x, obj1.y, obj1.Sub());
            Aa obj2 = new Aa(z: -3, x: 10, y: 14);
            Console.WriteLine("\nobj2: ");
            Console.WriteLine("{0} + {1} + {3} = {2}",
obj2.x, obj2.y, obj2.Sum(), obj2.z);
            Console.ReadLine();
        }
    }
}
```

Результаты работы приведены на рис.12.

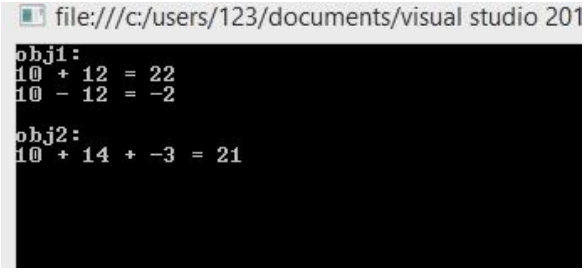


Рис. 12. Результат расчетов в программе

В C# допускается объявлять переменные ссылочного интерфейсного типа, т.е. переменные ссылки на интерфейс. Такая переменная может ссылаться на любой объект, реализующий ее интерфейс. При вызове метода для объекта посредством интерфейсной ссылки выполняется его вариант, реализованный в классе данного объекта. Этот процесс аналогичен применению ссылки на базовый класс для доступа к объекту производного класса.

Переменной ссылки на интерфейс доступны только методы, объявленные в ее интерфейсе. Поэтому интерфейсную ссылку нельзя использовать для доступа к любым другим переменным и методам, которые не поддерживаются объектом класса, реализующего данный интерфейс.

Листинг 6. Использование переменных ссылочного интерфейсного типа.

```
using System;
namespace ConsoleApplication1
{
    public interface IInfo
    {
        void uiName();
        void uiFamily();
        void uiAge();
    }
    class UI : IInfo
    {
        string Name, Family;
        int Age;
```

```
        public UI(string Name, string Family, int Age)
        {
            this.Name = Name;
            this.Family = Family;
            this.Age = Age;
        }
        // Реализуем интерфейс
        public void uiName()
        {
            Console.WriteLine("Имя пользователя: " +
Name);
        }
        public void uiFamily()
        {
            Console.WriteLine("Фамилия: " + Family);
        }
        public void uiAge()
        {
            Console.WriteLine("Возраст: " + Age);
        }
        // Собственный метод класса UI
        public void allInfo()
        {
            Console.WriteLine(Name + " " + Family + "
" + Age);
        }
    }
    class Program
    {
        static void Main()
        {
            UI ui1 = new UI(Name: "Ivan", Family:
"Ivanov", Age: 18);
            // Создадим ссылку на интерфейс
            IInfo obj;
            //Используем ссылку на объект ui1
            obj = ui1;
            obj.uiName();
            obj.uiFamily();
            obj.uiAge();
            // Вызов собственного метода не
разрешается:
            // obj.allInfo();
            Console.ReadLine();
```

```
            }
        }
    }
```
Результаты работы приведены на рис.13.

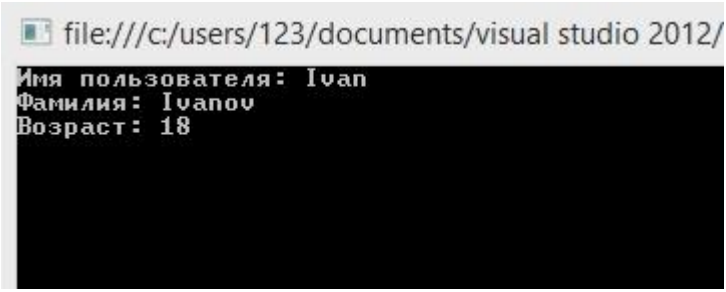


Рис. 13. Результат выполнения программы

Один интерфейс может наследовать другой. Синтаксис наследования интерфейсов такой же, как и у классов. Когда в классе реализуется один интерфейс, наследующий другой, в нем должны быть реализованы все члены, определенные в цепочке наследования интерфейсов.

Таким образом, интерфейсы могут быть организованы в иерархии. Как и в иерархии классов, в иерархии интерфейсов, когда какой-то интерфейс расширяет существующий, он наследует все абстрактные члены своего родителя (или родителей). В приведенном ниже фрагменте кода показана возможная реализация по наследованию интерфейсов.

Листинг 7. Наследование интерфейсов

```
using System;
namespace ConsoleApplication1
{
    public interface A
    {
        int Sum();
    }
    // Унаследованный интерфейс
    public interface B : A
    {
        int Del();
```

```
    }
    class MyOperation : B
    {
        int x = 10, y = 5;
        public int Sum()
        {
            return x + y;
        }
        public int Del()
        {
            return x / y;
        }
    }
    class Program
    {
        static void Main()
        {
        }
    }
}
```

На рис.14 показана схема наследования интерфейсов.

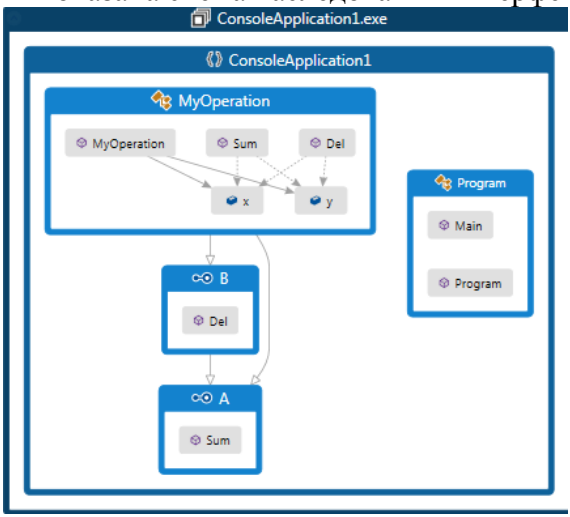


Рис. 14. Наследование интерфейсов

Структура является типом значения. При создании структуры переменная, к которой она назначается, сохраняет

фактические данные структуры. При назначении структуры новой переменной выполняется ее копирование. Поэтому новая переменная и исходная переменная содержат две отдельных копии одних данных. Изменения, внесенные в одну копию, не влияют на другую копию.

Листинг 8. Пример структуры данных в C#

```
struct UserInfo
{
    public string Name;
    public byte Age;
    public UserInfo(string Name, byte Age)
    {
        this.Name = Name;
        this.Age = Age;
    }
    public void WriteUserInfo()
    {
        Console.WriteLine("Имя:  {0},  возраст:
{1}", Name, Age);
    }
}
```

Задание на самостоятельную работу:

1. Разработать консольное приложение, где будет дан массив, в нем определить:

а) максимальный элемент;

б) минимальный элемент;

в) насколько максимальный элемент больше минимального;

г) индексы максимального и минимального элементов.

ЛАБОРАТОРНАЯ РАБОТА № 4
ДЕЛЕГАТЫ В C#

Цель работы: разработать консольное приложение в среде визуального программирования (Visual Studio) согласно заданию.

Задачи и требования к выполнению:

1.Изучить работу с делегатами в C#.

2.Изучить лямбда-выражения, рассмотреть примеры их использования.

3.Изучить обработку событий в .NET Framework.

Теоретические сведения

Делегат представляет собой объект, который может ссылаться на метод. Следовательно, когда создается делегат, то в итоге получается объект, содержащий ссылку на метод. Более того, метод можно вызывать по этой ссылке. Иными словами, делегат позволяет вызывать метод, на который он ссылается.

По сути, делегат - это безопасный в отношении типов объект, указывающий на другой метод (или, возможно, список методов) приложения, который может быть вызван позднее. В частности, объект делегата поддерживает три важных фрагмента информации:

-адрес метода, на котором он вызывается;

-аргументы (если есть) этого метода;

-возвращаемое значение (если есть) этого метода

На листинге 9 приводится пример с использованием делегатов.

Листинг 9. Пример использования

```
using System;
namespace ConsoleApplication1
{
    // делегат
    delegate void Autorization(string str1,string
str2);
    class Program
    {
        static void Connect1(string login,string
password)
        {
```

```
Console.WriteLine("Login:{0}\nPassword:  {1}",  login,
password);
            }
            static void Connect2(string login, string
password)
            {

Console.WriteLine("Login:{0}\nPassword:  {1}",  login,
password);
            }
            static void Main()
            {
                // Сконструируем делегат
                Autorization      obj      =      new
Autorization(Connect1);
                Console.WriteLine("Доступ:");
                obj("Андрей","12345");
                // Изменим ссылку на метод
                obj = new Autorization(Connect2);
                obj("Татьяна", "54321");
                Console.ReadKey();
            }
        }
    }
```

Результаты работы показаны на рис. 15.

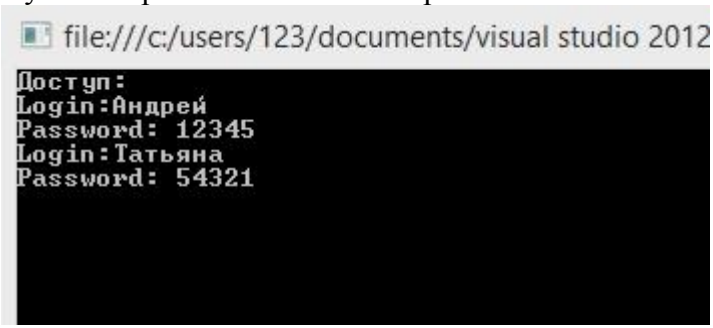


Рис. 15. Результат выполнения программы

В C# имеются возможности, которые существенно упрощают синтаксис присваивания метода делегату. Это так называемое групповое преобразование методов, позволяющее

присвоить имя метода делегату, не прибегая к оператору new или явному вызову конструктора делегата.

Например,

```
 static void Main()
        {
            // Сконструируем делегат
            Autorization obj = new
Autorization(Connect1);

Console.WriteLine("Доступ:");
            obj = Connect1;
            obj("Андрей", "12345");
            obj = Connect2;
            obj("Татьяна", "54321");
            Console.ReadKey();
        }
```

Далее рассматривается работа с лямбда-выражениями. В одиночном лямбда-выражении часть, находящаяся справа от оператора =>, воздействует на параметр (или ряд параметров), указываемый слева. Возвращаемым результатом вычисления такого выражения является результат выполнения лямбда-оператора. Ниже приведена общая форма одиночного лямбда-выражения, принимающего единственный параметр:

параметр => выражение

Если же требуется указать несколько параметров, то используется следующая форма:

(список_параметров) => выражение

Таким образом, когда требуется указать два параметра или более, их следует заключить в скобки. Если же выражение не требует параметров, то следует использовать пустые скобки.

Листинг 10. Пример использования лямбда-выражений

```
using System;
namespace ConsoleApplication1
{
```

```
        // Создадим несколько делегатов имитирующих
        // простейшую форму регистрации
        delegate int LengthLogin(string s);
        delegate bool BoolPassword(string s1, string
s2);
        class Program
        {
            static void Main()
            {
                //link1:
                bool NextGo = false;
                while (!NextGo)
                {
                    Console.Write("Введите логин: ");
                    string          login          =
Console.ReadLine();
                    // Используем лямбда-выражение
                    LengthLogin ll = ss => ss.Length;
                    int longlogin = ll(login);
                    if (longlogin >= 25)
                    {
                        Console.WriteLine("Слишком
длинное имя\n");
                        //goto link1;
                    }
                    else NextGo = true;
                }
                Console.Write("Введите пароль: ");
                string          password1          =
Console.ReadLine();
                Console.Write("Повторите пароль: ");
                string          password2          =
Console.ReadLine();
                // Используем лямбда выражение
                BoolPassword bp = (s1, s2) => s1 ==
s2;
                if (bp(password1, password2))
                    Console.WriteLine("Регистрация
удалась!");
                else
                    Console.WriteLine("Регистрация
провалилась. Пароли не совпадают");
                Console.ReadLine();
            }}}
```

Результаты работы показаны на рисунке 16.

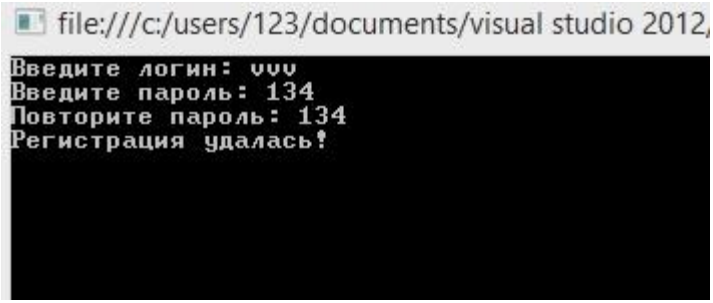


Рис. 16. Результат выполнения программы

Обработка событий в среде .NET Framework. В C# разрешается формировать какие угодно разновидности событий. Но ради совместимости программных компонентов со средой .NET Framework следует придерживаться рекомендаций, установленных для этой цели корпорацией Microsoft. Эти рекомендации, по существу, сводятся к следующему требованию: у обработчиков событий должны быть два параметра. Первый из них — ссылка на объект, формирующий событие, второй — параметр типа EventArgs, содержащий любую дополнительную информацию о событии, которая требуется обработчику. Таким образом, .NET-совместимые обработчики событий должны иметь следующую общую форму:

```
        void обработчик(object отправитель, EventArgs e)
{
          //...
        }
```

В следующем фрагменте кода показано, как использовать событие по нажатию клавиши в приложении.

Листинг 11. Пример обработки событий

```
// Производный класс от EventArgs
   class MyEventArgs : EventArgs
```

```
        {
            public char ch;
        }
        class KeyEvent
        {
            //    Создадим    событие,    используя
обобщенный делегат
            public  event  EventHandler<MyEventArgs>
KeyDown;
            public void OnKeyDown(char ch)
            {
                MyEventArgs c = new MyEventArgs();
                if (KeyDown != null)
                {
                    c.ch = ch;
                    KeyDown(this, c);
                }
            }
        }
```

Задания на самостоятельную работу:

1.Разработать приложение, в котором пользователь может перемещать некоторый символ, используя курсорные клавиши клавиатуры.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Прата С. Язык программирования С++. Лекции и упражнения / С. Прата. 5-е изд. – М.: ООО ”И.Д. Вильямс”, 2007. – 1184 с.
2. Страуструп Б Язык программирования С++ / Б. Страуструп. - М.: Бином, 2011. – 1136 с.
3. Шилдт Г. С++ Базовый курс / Г. Шилдт. 3-е изд. – М.: ООО ”И.Д. Вильямс”, 2010. – 624 с.
4. Шилдт Г. Полный справочник по C# / Г. Шилдт. – 4-е изд. – М.: Вильямс, 2009. – 800 с.
5. Шилдт Г. Самоучитель C#/ Г. Шилдт. – 3-е изд. – 3-е изд. – СПб.: БХВ-Петербург, 2002. – 688 с.
6. Дейтел Х. C# / Х. Дейтел, П. Дейтел, Дж. Листфилд, Т.Нието, Ш. Йегер и др. – СПб.: БХВ-Петербург, 2006. – 1056 с.
7. Агупов П.В. C#. Разработка компонентов в MS Visual Studio 2005/2008 / П.В. Агупов - СПб.: БХВ-Петербург, 2008. – 480 с.
8. Бишоп Дж. C# в кратком изложении / Дж. Бишоп, Н. Хорспул. - М.: Бином, 2005. – 472 с.
9. London J. Modeling Derivatives in C++ / London J. Wiley, 2005. - 841p.

СОДЕРЖАНИЕ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторным работам № 1-4 по дисциплине
«Среды визуального программирования» для студентов
направления 09.03.02 «Информационные системы
и технологии» (профиль «Информационные системы
и технологии в машиностроении») очной формы обучения

Составители:
Юров Алексей Николаевич
Бредихин Алексей Вячеславович

В авторской редакции

Компьютерный набор А.Н. Юрова

Подписано к изданию 20.03.2015.
Уч.-изд. л. 1,9. «С»

ФГБОУ ВПО «Воронежский государственный технический
университет»
394026 Воронеж, Московский просп., 14