Лабораторная работа №2

Created with the Freeware Edition of HelpNDoc: Free Web Help generator

ХАРЬКОВСКИЙ НАЦИОНАЛЬНЫЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И МЕХАТРОНИКИ

Кафедра информационных технологий и мехатроники

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по проведению лабораторных работ по дисциплине «Объектно-ориентированное программирование» для студентов специальности 6.050101 “Компьютерные науки”

Разработчик - доцент кафедры информационных технологий и мехатроники
кандидат технических наук, старший научный сотрудник
Тимонин Владимир Алексеевич

Харків 2015

Created with the Freeware Edition of HelpNDoc: Free Web Help generator

Лабораторная работа №2
Исследование возможностей интегрированной среды разработки Visual C# для создания приложений, использующие методы.

Цель работы – исследовать возможности интегрированной среды разработки Visual Studio 2010 и получить практические навыки по созданию приложений, использующие методы.

1. Теоретические сведения

Класс создается с помощью ключевого слова **class**. Общая форма определения класса, который содержит только переменные экземпляров и методы, имеет следующий вид:

```
class имя_класса
    {
     // Объявление переменных экземпляров
     доступ тип переменная 1;
     доступ тип переменная 2;
                 . . .
     доступ тип переменная N;
     // Объявление методов
     доступ тип_возврата метод1 (параметры)
             {
               // тело метода
             }
     доступ тип_возврата метод2 (параметры)
             {
               // тело метода
             }
     доступ тип_возврата методN (параметры)
             {
               // тело метода
             }
    }
```

Объявление каждой переменной и каждого метода предваряется элементом **доступ**, означающий спецификатор доступа, который определяет, как к этому члену можно получить доступ. Управление доступом к членам класса достигается за счет использования четырех спецификаторов доступа: **public, private, protected** и **internal**.

Спецификатор доступа должен стоять первым в списке спецификаторов типа любого члена класса.

Спецификатор **public** разрешает доступ к соответствующему члену класса со стороны другого кода программы, включая методы, определенные внутри других классов.

Спецификатор **private** разрешает доступ к соответствующему члену класса только для методов, определенных внутри того же класса. Таким образом, методы других классов не могут получить доступ к private-члену не их класса. При отсутствии спецификатора доступа член класса является закрытым (private) по умолчанию. Следовательно, при создании закрытых членов класса спецификатор **private** необязателен.

Спецификатор **protected** применяется только при включении интерфейсов.

Спецификатор **internal** применяется в основном при использовании компоновочных файлов (assembly).

Created with the Freeware Edition of HelpNDoc: Free Web Help generator

1.1. Создание объектов

Пример 1. Для иллюстрации создадим класс, который инкапсулирует информацию о зданиях (домах, складских помещениях, офисах и пр.). В этом классе (назовем его Building) будут храниться три элемента информации о зданиях: количество этажей, общая площадь и количество жильцов.

```
class Building
    {
      public int floors; // количество этажей
      public int area;   // общая площадь основания здания
      public int count; // количество жильцов
    }
```

В классе Building определены три переменные экземпляра: **floors, area** и **count**. Класс Building не содержит ни одного метода. Поэтому пока его можно считать классом данных.

Определение **class** создает новый тип данных. В данном случае этот новый тип данных называется Building. Это имя можно использовать для объявления объектов типа Building. Объявление **class** — это лишь описание типа, оно не создает реальных объектов. Таким образом, предыдущий код не означает существования объектов типа Building.

Чтобы реально создать объект класса Building, используется инструкция:

Building house = new Building();

Это объявление выполняет две функции. Во-первых, оно объявляет переменную с именем **house** классового типа **Building**. Но эта переменная не определяет объект, а может лишь ссылаться на него. Во-вторых, рассматриваемое объявление создает реальную физическую копию объекта и присваивает переменной **house** ссылку на этот объект. И все это — "дело рук" оператора **new**. Таким образом, после выполнения приведенной выше строки кода переменная **house** будет ссылаться на объект типа **Building**.

Оператор **new** динамически (т.е. во время выполнения программы) выделяет память для объекта и возвращает ссылку на него. Эта ссылка (сохраненная в конкретной переменной) служит адресом объекта в памяти, выделенной для него оператором **new**. Таким образом, в C# для всех объектов классов должна динамически выделяться память.

При каждом создании экземпляра класса создается объект, который содержит собственную копию каждой переменной экземпляра, определенной этим классом. Таким образом, каждый объект класса Building будет содержать собственные копии переменных экземпляра: **floors, area** и **count**.

Для доступа к этим переменным используется оператор "точка" (.). Оператор "точка" связывает имя объекта с именем его члена.

Например, чтобы присвоить переменной **floors** значение **2**, используется инструкция:

house.floors = 2;

В общем случае оператор "точка" можно использовать для доступа как к переменным экземпляров, так и методам.

Пример 2. Рассмотрим полную программу, в которой используется класс Building.

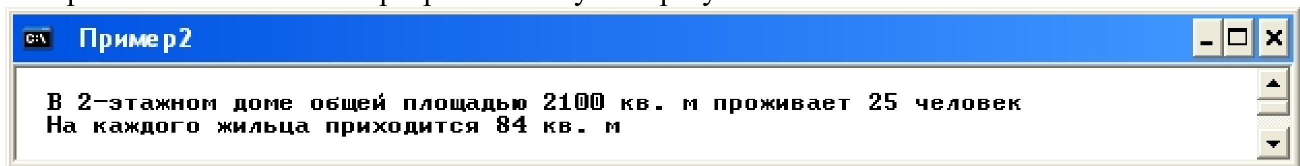
```
class Building
  {
     public int floors; // количество этажей
     public int area;   // общая площадь основания здания
     public int count; // количество жильцов
  }
class Program
  {
     public static void Main()
     {
```

Created with the Freeware Edition of HelpNDoc: Free Web Help generator

```
        int area1; // площадь, приходящаяся на одного жильца
        Building house = new Building();
        house.floors = 2;
        house.area = 2100;
        house.count = 25;
        // Вычисляем площадь, приходящуюся на одного жильца дома
        area1 = house.area / house.count;
        Console.WriteLine("\n  В {0}-этажном доме общей площадью {1} кв. м проживает
                                    {2} человек",  house.floors, house.area, house.count);
        Console.WriteLine("  На каждого жильца приходится {0} кв. м", area1);
    }
}
```

Эта программа состоит из двух классов: Building и Program. Внутри класса Program метод Main() сначала создает экземпляр класса Building с именем house, а затем получает доступ к переменным этого экземпляра house, присваивая им конкретные значения и используя эти значения в вычислениях. Важно понимать, что Building и Program — это два отдельных класса. Единственная связь между ними состоит в том, что один класс создает экземпляр другого. Хотя это отдельные классы, код класса Program может получать доступ к членам класса Building, поскольку они объявлены открытыми, т.е. public-членами. Если бы в их объявлении не было спецификатора доступа public, доступ к ним ограничивался бы рамками класса Building, а класс Program не имел бы возможности использовать их.

При выполнении этой программы получаем результат:



В действительности совсем не обязательно классам Building и Program находиться в одном исходном файле. Можно поместить каждый класс в отдельный файл и назвать эти файлы Building.cs и Program.cs, соответственно. После этого необходимо дать компилятору команду скомпилировать оба файла и скомпоновать их, для чего нужно поместить оба файла в проект и выполнить команду построения этого проекта.

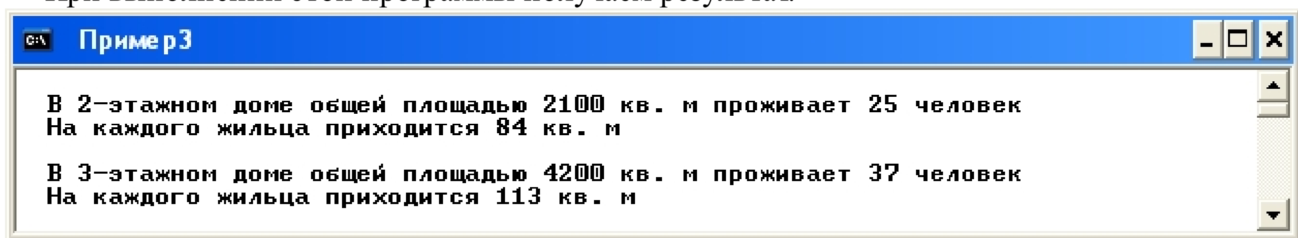
При создании нескольких объектов класса каждый объект класса имеет собственные копии переменных экземпляра, определенных в этом классе. Таким образом, содержимое переменных в одном объекте может отличаться от содержимого аналогичных переменных в другом Между двумя объектами нет связи, за исключением того, что они являются объектами одного и того же типа. Например, если у вас есть два объекта типа Building и каждый объект имеет свою копию переменных floors, area и count, то содержимое соответствующих (одноименных) переменных этих двух экземпляров может быть разным.

Пример 3. Следующая программа демонстрирует использование 2-х объектов класса Building.

```
class Program
  {
    public static void Main()
    {
        int area1; // площадь, приходящаяся на одного жильца
        Building house1 = new Building();
        Building house2 = new Building();
        // Присваиваем значения полям в объекте house1
        house1.floors = 2;
        house1.area = 2100;
        house1.count = 25;
```

```
        // Присваиваем значения полям в объекте house2
        house2.floors = 3;
        house2.area = 4200;
        house2.count = 37;
        // Вычисляем площадь, приходящуюся на одного жильца дома
        area1 = house1.area / house1.count;
        Console.WriteLine("\n  В {0}-этажном доме общей площадью {1} кв. м проживает
                               {2} человек",  house1.floors, house1.area, house1.count);
        Console.WriteLine("  На каждого жильца приходится {0} кв. м", area1);
        area1 = house2.area / house2.count;
        Console.WriteLine("\n  В {0}-этажном доме общей площадью {1} кв. м проживает
                               {2} человек", house2.floors, house2.area, house2.count);
        Console.WriteLine("  На каждого жильца приходится {0} кв. м", area1);
    }
```

При выполнении этой программы получаем результат:



В операции присвоения ссылочные переменные действуют не так, как переменные типа значений (например, типа int). Присваивая одной переменной (типа значения) значение другой, мы имеем довольно простую ситуацию. Переменная слева (от оператора присваивания) получает копию значения переменной справа. При выполнении аналогичной (казалось бы) операции присваивания между двумя переменными ссылочного типа ситуация усложняется, поскольку мы изменяем объект, на который ссылается ссылочная переменная, что может привести к неожиданным результатам. Например, рассмотрим следующий фрагмент программы:

```
Building housel = new Building();
Building house2 = housel;
```

На первый взгляд может показаться, что housel и house2 ссылаются на различные объекты, но это не так. Обе переменные, housel и house2, ссылаются на один и тот же объект. Присвоение значения переменной housel переменной house2 просто заставляет переменную house2 ссылаться на тот же объект, на который ссылается и переменная housel. В результате на этот объект можно воздействовать, используя либо имя housel, либо имя house2.

Например, присвоив

```
housel.area = 2600;
```

мы добьемся того, что обе инструкции

```
Console.WriteLine(housel.area);
Console.WriteLine(house2.area);
```

отобразят одно и то же значение — 2600.

Несмотря на то, что обе переменные, housel и house2, ссылаются на один и тот же объект, они никак не связаны между собой. Например, очередное присвоение переменной house2 просто заменяет объект, на который она ссылается. После выполнения последовательности инструкций:

```
Building housel = new Building();
Building house2 = housel;
Building house3 = new Building();
Building house2 = house3;
```

переменная house2 будет ссылаться на тот же объект, на который ссылается переменная house3. Объект, на который ссылается переменная housel, не меняется.

1.2. Методы

Как упоминалось выше, переменные экземпляров и методы — две основные составляющие классов. Пока наш класс Building содержит только данные. Хотя такие классы (без методов) вполне допустимы, большинство классов имеют методы. Методы — это процедуры (подпрограммы), которые манипулируют данными, определенными в классе, и во многих случаях обеспечивают доступ к этим данным. Обычно различные части программы взаимодействуют с классом посредством его методов.

1.2.1. Объявление методов

Любой метод содержит одну или несколько инструкций. В хорошей C#-программе один метод выполняет только одну задачу. Каждый метод имеет имя, и именно это имя используется для его вызова. В общем случае методу можно присвоить любое имя. Но помните, что имя Main() зарезервировано для метода, с которого начинается выполнение программы. Кроме того, в качестве имен методов нельзя использовать ключевые слова C#.

Имена методов сопровождаются парой круглых скобок, что позволяет отличать имена переменных от имен методов.

Формат записи метода такой:

```
доступ тип_возврата имя(список_параметров)
    {
      // тело метода
    }
```

где элемент **доступ** означает модификатор доступа, который определяет, какие части программы могут получить доступ к методу. Как упоминалось выше, модификатор доступа необязателен, и, если он не указан, подразумевается, что метод закрыт (private) в рамках класса, где он определен. Пока мы будем объявлять все методы как public-члены, чтобы их могли вызывать все остальные составные части программного кода, даже те, которые определены вне класса.

С помощью элемента **тип_возврата** указывается тип значения, возвращаемого методом. Это может быть любой допустимый тип, включая типы классов, создаваемые программистом. Если метод не возвращает никакого значения, необходимо указать тип **void**.

Имя метода задается элементом **имя**. В качестве имени метода можно использовать любой допустимый идентификатор, отличный от тех, которые уже использованы для других элементов программы в пределах текущей области видимости.

Элемент **список_параметров** представляет собой последовательность пар (состоящих из типа данных и идентификатора), разделенных запятыми. Параметры — это переменные, которые получают значения аргументов, передаваемых методу при вызове. Если метод не имеет параметров, **список_параметров** остается пустым.

1.2.2. Добавление методов в класс Building

Методы класса, как правило, манипулируют данными, определенными в классе, и обеспечивают доступ к этим данным. Зная это, вспомним, что метод Main() в предыдущей программе вычислял площадь, приходящуюся на одного человека, путем деления общей площади здания на количество жильцов.

Несмотря на формальную корректность, эти вычисления выполнены не самым удачным образом. Ведь с вычислением площади, приходящейся на одного человека, вполне может справиться сам класс Building, поскольку эта величина зависит только от значений переменных area и count, которые инкапсулированы в классе Building. Как говорится, сам Бог велел классу Building выполнить это арифметическое действие. Более того, если оно таки будет "закреплено" за этим классом, то другой программе, которая его использует, не придется делать это действие "вручную". Здесь налицо не просто удобство для "других" программ, а предотвращение неоправданного дублирования кода. Наконец, внося в класс Building метод, который вычисляет площадь, приходящуюся на одного человека, вы улучшаете его объектно-ориентированную структуру, инкапсулируя внутри рассматриваемого класса величины, связанные непосредственно со зданием.

Чтобы добавить в класс Building метод, необходимо определить его внутри объявления класса.

Created with the Freeware Edition of HelpNDoc: Free Web Help generator

Пример 4. Следующая версия класса Building содержит метод с именем areaPerson(), который отображает значение площади конкретного здания, приходящейся на одного человека.

```
class Building
   {
      public int floors; // количество этажей
      public int area;   // общая площадь основания здания
      public int count; // количество жильцов
      public void areaPerson()
      {
         Console.WriteLine("  На каждого жильца приходится {0} кв. м", area / count);
      }
   }
class Program
   {
      static void Main()
      {
         Building house1 = new Building();
         Building house2 = new Building();
         // Присваиваем значения полям в объекте house1
         house1.floors = 2;
         house1.area = 2100;
         house1.count = 25;
         // Присваиваем значения полям в объекте house2
         house2.floors = 3;
         house2.area = 4200;
         house2.count = 37;
         // Вычисляем площадь, приходящуюся на одного жильца дома
         Console.WriteLine("\n  В {0}-этажном доме общей площадью {1} кв. м проживает
                                   {2} человек", house1.floors, house1.area, house1.count);
         house1.areaPerson();
         Console.WriteLine("\n  В {0}-этажном доме общей площадью {1} кв. м проживает
                                   {2} человек", house2.floors, house2.area, house2.count);
         house2.areaPerson();
      }
   }
```

При выполнении этой программы получаем результат, который совпадает с результатом примера 3:

```
Пример4

В 2-этажном доме общей площадью 2100 кв. м проживает 25 человек
На каждого жильца приходится 84 кв. м

В 3-этажном доме общей площадью 4200 кв. м проживает 37 человек
На каждого жильца приходится 113 кв. м
```

Теперь рассмотрим ключевые элементы этой программы, начиная с самого метода areaPerson(). Первая строка этого метода выглядит так:

public void areaPerson()

В этой строке объявляется метод с именем areaPerson(), который не имеет параметров. Этот метод определен с использованием спецификатора доступа public, поэтому его могут использовать все остальные части программы. Метод areaPerson() возвращает значение типа void, т.е. не возвращает

Created with the Freeware Edition of HelpNDoc: Free Web Help generator

никакого значения.

Тело метода areaPerson() заключено в пару фигурных скобок ({ ... }) и состоит из единственной инструкции:

Console.WriteLine(" На каждого жильца приходится {0} кв. м", area / count);

Эта инструкция отображает площадь здания, которая приходится на одного человека, путем деления значения переменной area на значение переменной count.

Поскольку каждый объект типа Building имеет собственную копию значений area и count, то при вызове метода areaPerson() в вычислении площади здания, которая приходится на одного человека, будут использоваться копии этих переменных, принадлежащие конкретному вызывающему объекту.

Метод areaPerPerson() завершается закрывающей фигурной скобкой, т.е. при обнаружении закрывающей фигурной скобки управление программой передается вызывающему объекту.

Теперь рассмотрим внимательнее строку кода из метода Main():

house1.areaPerson();

Эта инструкция вызывает метод areaPerson() для объекта house1. Для этого используется имя объекта, за которым следует оператор "точка". При вызове метода управление выполнением программы передается телу метода, а после его завершения управление возвращается автору вызова, и выполнение программы возобновляется со строки кода, которая расположена сразу за вызовом метода.

В данном случае в результате вызова house1.areaPerson() отображается значение площади, которая приходится на одного человека, для здания, определенного объектом house1.

Точно так же в результате вызова house2.areaPerson() отображается значение площади, которая приходится на одного человека, для здания, определенного объектом house2.

Другими словами, каждый раз, когда вызывается метод areaPerson(), отображается значение площади, которая приходится на одного человека, для здания, описываемого заданным объектом.

Переменные экземпляра area и count используются внутри метода areaPerson() без каких бы то ни было атрибутов, т.е. им не предшествует ни имя объекта, ни оператор "точка". Это очень важный момент: если метод задействует переменную экземпляра, которая определена в его классе, он делает это напрямую, без явной ссылки на объект и без оператора "точка". И это логично. Ведь метод всегда вызывается для некоторого объекта конкретного класса. И если уж вызов состоялся, объект, стало быть, известен. Таким образом, нет необходимости указывать внутри метода объект во второй раз. Это значит, что значения area и count внутри метода areaPerson() неявно указывают на копии этих переменных, принадлежащих объекту, который вызывает метод areaPerson().

1.2.3. Возвращение из метода

В общем случае существует два варианта условий для возвращения из метода. Первый связан с обнаружением закрывающей фигурной скобки, обозначающей конец тела метода. Второй вариант состоит в выполнении инструкции return. Возможны две формы использования инструкции **return**: одна предназначена для void-методов (которые не возвращают значений), а другая — для возврата значений.

Немедленное завершение void-метода можно организовать с помощью следующей формы инструкции return:

return;

При выполнении этой инструкции управление программой передается автору вызова метода, а оставшийся код опускается. Рассмотрим, например, следующий метод:

```
public void myMeth()
    {
      int i ;
      for (i=0; i<10; i++)
        if ( i == 5) return; // прекращение выполнения метода при i = 5
      Console.WriteLine();
    }
```

Здесь цикл for будет работать при значениях i в диапазоне только от 0 до 5, так как только

Created with the Freeware Edition of HelpNDoc: Free Web Help generator

значение i станет равным 5, будет выполнен возврат из метода myMeth().

Метод может иметь несколько инструкций return. Например, выход из метода

```
public void myMeth ()
      {
        if (done) return;
        // . . .
        if (error) return;
       }
```

произойдет либо в случае его корректного завершения, либо при возникновении ошибки. Однако наличие слишком большого количества точек выхода из метода может деструктурировать код Поэтому, несмотря на допустимость их множественного применения, следует все же использовать эту возможность с большой осторожностью.

Хотя void-методы — не редкость, большинство методов все же возвращают значение. Способность возвращать значение — одно из самых полезных качеств метода.

Значения, возвращаемые методами, используются в программировании по-разному. В одних случаях (как в методе Math.Sqrt()) возвращаемое значение является результатом вычислений, в других — оно просто означает, успешно или нет выполнены действия, составляющие метод, а в третьих — оно может представлять собой код состояния. Однако независимо от цели применения, использование значений, возвращаемых методами, является неотъемлемой частью C#-программирования.

Методы возвращают значения вызывающим их процедурам, используя следующую форму инструкции return:

```
return значение;
```

где элемент значение и представляет значение, возвращаемое методом.

Способность методов возвращать значения можно использовать для улучшения реализации метода areaPerson(). Вместо того чтобы отображать значение площади, которая приходится на одного человека, метод areaPerson() будет теперь возвращать это значение, которое можно использовать в других вычислениях.

Пример 5. В следующем примере представлен модифицированный вариант метода areaPerson(), который возвращает значение площади, приходящейся на одного человека, а не отображает его (как в предыдущем варианте).

```
class Building
   {
      public int floors; // количество этажей
      public int area;   // общая площадь основания здания
      public int count; // количество жильцов
      public int areaPerson()
      {
         return area / count;
      }
   }
class Program
   {
      static void Main()
      {
         int area1; // площадь, приходящаяся на одного жильца
         Building house1 = new Building();
         Building house2 = new Building();
         // Присваиваем значения полям в объекте house1
         house1.floors = 2;
```

Created with the Freeware Edition of HelpNDoc: Free Web Help generator

```
        house1.area = 2100;
        house1.count = 25;
        // Присваиваем значения полям в объекте house2
        house2.floors = 3;
        house2.area = 4200;
        house2.count = 37;
        // Вычисляем площадь, приходящуюся на одного жильца дома
        Console.WriteLine("\n  В {0}-этажном доме общей площадью {1} кв. м проживает
                                    {2} человек", house1.floors, house1.area, house1.count);
        area1 = house1.areaPerson();
        Console.WriteLine("  На каждого жильца приходится {0} кв. м", area1);
        Console.WriteLine("\n  В {0}-этажном доме общей площадью {1} кв. м проживает
                                    {2} человек", house2.floors, house2.area, house2.count);
        area1 = house2.areaPerson();
        Console.WriteLine("  На каждого жильца приходится {0} кв. м", area1);
    }
  }
```

Результаты выполнения этого варианта программы аналогичны предыдущим.

Обратите внимание на вызов метода areaPerson(): его имя находится справа от оператора присваивания. В левой части стоит переменная, которая и получает значение, возвращаемое методом areaPerson(). Таким образом, после выполнения инструкции

```
area1 = house1.areaPerson();
```

значение площади, приходящейся на одного человека для объекта house1, будет сохранено в переменной area1.

Обратите также внимание на то, что метод areaPerson() имеет в этом примере другой тип возвращаемого значения, а именно тип int. Это означает, что метод возвращает автору вызова целое число. Тип значения, возвращаемого методом, — очень важная характеристика метода, поскольку тип данных, возвращаемых методом, должен быть совместимым с типом возвращаемого значения, указанного в заголовке определения метода. Следовательно, если вы хотите, чтобы метод возвращал данные типа double, при его определении в качестве типа возвращаемого значения следует указать double.

Несмотря на корректность предыдущей программы, ее эффективность оставляет желать лучшего. В частности, нет необходимости в использовании переменной area1. Вызов метода areaPerson() можно реализовать непосредственно в инструкции вызова метода WriteLine():

```
Console.WriteLine("  На каждого жильца приходится {0} кв. м", house1.areaPerson());
```

В этом случае при вызове метода WriteLine() автоматически вызывается метод house1.areaPerson(), и возвращаемое им значение передается методу WriteLine(). Более того, теперь обращение к методу house1.areaPerson() можно использовать везде, где необходимо значение площади для объекта класса Building, которая приходится на одного человека. Например, в следующей инструкции сравниваются такие значения площадей двух зданий:

```
if (A.areaPerson() > B.areaPerson())
 Console.WriteLine("В здании А больше места для каждого человека." );
```

1.2.4. Использование параметров

При вызове методу можно передать одно или несколько значений Значение, передаваемое методу, называется аргументом. Переменная внутри метода, которая принимает значение аргумента, называется параметром. Параметры объявляются внутри круглых скобок, которые следуют за именем метода. Синтаксис объявления параметров аналогичен синтаксису, применяемому для переменных.

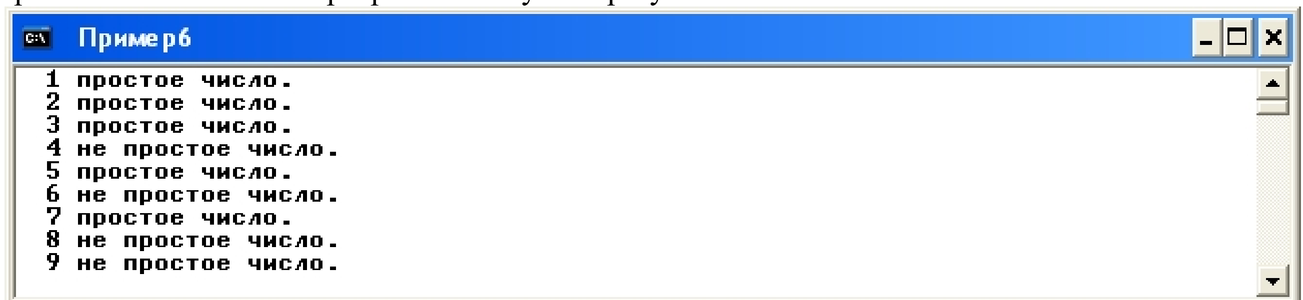
Параметр находится в области видимости своего метода, и, помимо специальной задачи получения аргумента, действует подобно любой локальной переменной.

Created with the Freeware Edition of HelpNDoc: Free Web Help generator

Пример 6. Следующий пример демонстрирует использования метода с параметром. В классе MyClass определен метод MyMeth(), который возвращает значение true, если переданное ему значение является простым, и значение false в противном случае. Следовательно, метод MyMeth() возвращает значение типа bool.

```
class MyClass
   {
       public bool MyMeth(int x)
        {
          for (int i=2; i < x/2 + 1; i++)
             if ((x %i) == 0) return false;
          return true;
        }
    }
class Program
   {
       static void Main()
       {
        MyClass ob = new MyClass();
        for (int i=1; i < 10; i++)
           if (ob. MyMeth(i))
             Console.WriteLine(i +" простое число.");
           else Console.WriteLine(i + " не простое число.");
       }
   }
```

При выполнении этой программы получаем результат:



В этой программе метод MyMeth() вызывается девять раз, и каждый раз ему передается новое значение. Передаваемый аргумент указывается между круглыми скобками. При первом вызове метода MyMeth() ему передается значение 1. Перед началом выполнения этого метода параметр **x** получит значение 1. При втором вызове аргумент будет равен числу 2, а значит, и параметр получит значение 2 и т.д. Важно то, что значение, переданное как аргумент при вызове функции MyMeth(), представляет собой значение, получаемое параметром x.

Метод может иметь более одного параметра. В этом случае достаточно объявить каждый параметр, отделив его от следующего запятой.

Пример 7. Следующий пример демонстрирует использования метода с двумя параметрами. В классе **MyClass** определен метод **MyMeth()**, который возвращает наименьший общий знаменатель для передаваемых ему значений. Обратите внимание на то, что при вызове метода **MyMeth()** аргументы также разделяется запятыми.

```
class MyClass
   {
```

Created with the Freeware Edition of HelpNDoc: Free Web Help generator

```
        public int MyMeth(int a, int b)
        {
            int max;
            max = a < b ? a : b;
            for (int i=2; i < max/2 + 1; i++)
              if (((a%i) == 0) & ((b%i) == 0)) return i;
            return 1;
        }
    }
class Program
   {
        static void Main()
        {
            MyClass ob = new MyClass();
            int a, b;
            a = 7; b = 8;
            Console.WriteLine("  Наименьший общий знаменатель для " + a +
                              " и " + b + " равен " + ob.MyMeth(a, b));
            a = 100; b = 8;
            Console.WriteLine("  Наименьший общий знаменатель для " + a +
                              " и " + b + " равен " + ob.MyMeth(a, b));
            a = 100; b = 75;
            Console.WriteLine("  Наименьший общий знаменатель для " + a +
                              " и " + b + " равен " + ob.MyMeth(a, b));
        }
   }
```

При выполнении этой программы получаем результат:

```
Пример7
Наименьший общий знаменатель для 7 и 8 равен 1
Наименьший общий знаменатель для 100 и 8 равен 2
Наименьший общий знаменатель для 100 и 75 равен 5
```

При передаче методу нескольких параметров каждый из них должен сопровождаться указанием собственного типа, причем типы параметров могут быть различными. Например, следующая запись вполне допустима:

```
int myMeth(int a, double b, float c)
   {
     // . . .
    }
```

1.2.5. Добавление параметризованного метода в класс Building

Для добавления в класс Building нового средства (вычисления максимально допустимого количества жильцов здания) можно использовать параметризованный метод. При этом предполагается, что площадь, приходящаяся на каждого человека, не должна быть меньше определенного минимального значения. Назовем этот новый метод **maxCount()** и приведем его определение. Метод возвращает максимальное количество человек, если на каждого должна приходиться заданная минимальная площадь.

```
public int maxCount(int minArea)
     {
         return area / minArea;
     }
```

При вызове метода maxCount() параметр minArea получает значение минимальной площади,

Created with the Freeware Edition of HelpNDoc: Free Web Help generator

необходимой для жизнедеятельности каждого человека. Результат, возвращаемый методом maxCount(), получается как частное от деления общей площади здания на это значение.

Пример 8. Следующий пример демонстрирует использования параметризованного метода **maxCount()**.

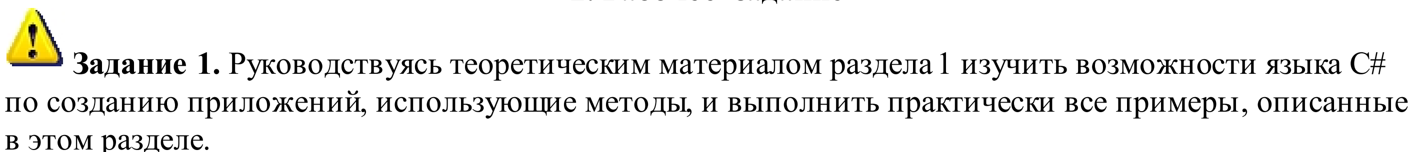
```
class Building
   {
      public int floors; // количество этажей
      public int area;   // общая площадь основания здания
      public int count; // количество жильцов
      // Метод возвращает площадь, которая приходится на одного человека
      public int areaPerson()
      {
         return area / count;
      }
      /* Метод возвращает максимальное возможное количество человек в здании,
         если на каждого должна приходиться заданная минимальная площадь */
      public int maxCount(int minArea)
      {
         return area / minArea;
      }
   }
class Program
   {
      static void Main()
      {
         int area1; // площадь, приходящаяся на одного жильца
         Building house1 = new Building();
         // Присваиваем значения полям в объекте house1
         house1.floors = 2;
         house1.area = 2100;
         house1.count = 25;
         Console.WriteLine("\n  В {0}-этажном доме общей площадью {1} кв. м проживает
                                    {2} человек", house1.floors, house1.area, house1.count);
         area1 = house1.areaPerson();
         Console.WriteLine("  На каждого жильца приходится {0} кв. м", area1);
         Console.WriteLine("  Макс. число человек для дома (на каждого должно
                              приходиться " + 115 + " кв. м): " + house1.maxCount(115));
      }
   }
```

При выполнении этой программы получаем результат:

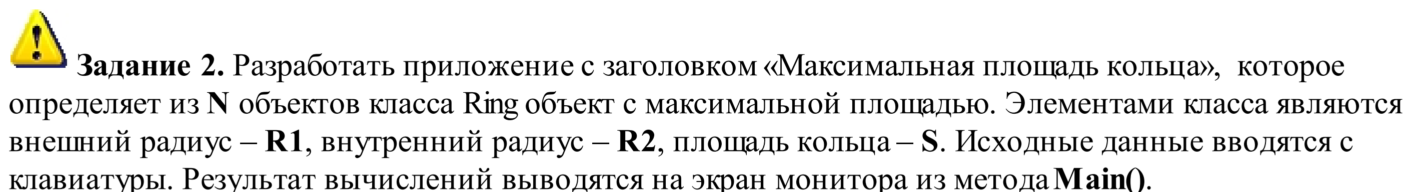
```
Пример8

 В 2-этажном доме общей площадью 2100 кв. м проживает 25 человек
 На каждого жильца приходится 84 кв. м
 Макс. число человек для дома (на каждого должно приходиться 115 кв. м): 18
```

Created with the Freeware Edition of HelpNDoc: Free Web Help generator

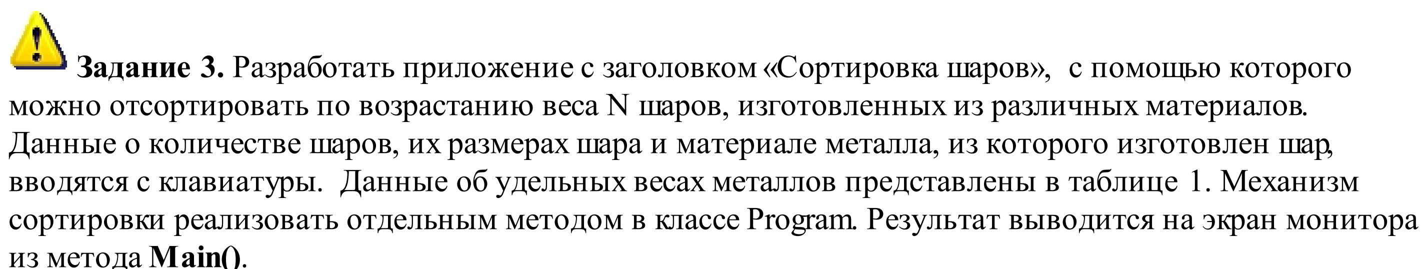
2. Рабочее задание



Задание 1. Руководствуясь теоретическим материалом раздела 1 изучить возможности языка C# по созданию приложений, использующие методы, и выполнить практически все примеры, описанные в этом разделе.



Задание 2. Разработать приложение с заголовком «Максимальная площадь кольца», которое определяет из **N** объектов класса Ring объект с максимальной площадью. Элементами класса являются внешний радиус – **R1**, внутренний радиус – **R2**, площадь кольца – **S**. Исходные данные вводятся с клавиатуры. Результат вычислений выводятся на экран монитора из метода **Main()**.



Задание 3. Разработать приложение с заголовком «Сортировка шаров», с помощью которого можно отсортировать по возрастанию веса N шаров, изготовленных из различных материалов. Данные о количестве шаров, их размерах шара и материале металла, из которого изготовлен шар, вводятся с клавиатуры. Данные об удельных весах металлов представлены в таблице 1. Механизм сортировки реализовать отдельным методом в классе Program. Результат выводится на экран монитора из метода **Main()**.

Таблица 1. Удельный вес цветных металлов

Наименование цветного металла	Удельный вес, г/см^3
Цинк	7,13
Алюминий	2,69808
Свинец	11,337
Олово	7,29
Медь	8,96
Титан	4,505
Никель	8,91
Магний	1,74
Ванадий	6,11
Вольфрам	19,3
Хром	7,19
Молибден	10,22
Серебро	10,5
Тантал	16,65
Золото	19,32
Платина	21,45

3. Контрольные вопросы

1. Что такое метод и его назначение.
2. Как объявляется метод?
3. Что такое аргумент?
4. Что такое параметр метода?
5. Как осуществляется вызов метода?
6. С помощью какого оператора осуществляется выход из метода?
7. Что такое параметризованный метод?

Created with the Freeware Edition of HelpNDoc: Free Web Help generator

8. Перечислите назначения оператора **return**.

Литература

1. Голощапов А.Л. Microsoft Visual Studio 2010. – СПб.:БХВ-Петербург, 2011. – 544 с.: ил.
2. Петцольд Ч. Программирование для Microsoft Windows на C#. В 2-х томах. Том 1. Пер. с англ. - М.: «Русская Редакция», 2002.- 576 с.: ил.
3. Петцольд Ч. Программирование для Microsoft Windows на C#. В 2-х томах. Том 2. Пер. с англ. - М.: «Русская Редакция», 2002.- 624 с.: ил.
4. Троелсен Э. Язык программирования C# 2010 и платформа .NET 4.0. Пер. с англ. - М.: Издательский дом "Вильямс", 2011. — 1392 с.: ил.
5. Фленов М.Е. Библия C#. - СПб.: БХВ-Петербург, 2011.– 560с.: ил.
6. Шилдт Г. C# Учебный курс. – СПб.: Питер, Издательская группа BHV, 2003. – 512 с.: ил.

Created with the Freeware Edition of HelpNDoc: Free Web Help generator