Лабораторная работа №1

Created with the Freeware Edition of HelpNDoc: [Free HTML Help documentation 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 HTML Help documentation generator

Лабораторная работа №1
Исследование возможностей интегрированной среды разработки Visual C# для создания приложений, использующих классы и объекты.

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

1. Теоретические сведения

Класс — это шаблон, который определяет форму объекта. Он задает как данные, так и код, который оперирует этими данными. Методы и переменные, составляющие класс, называются членами класса. Класс – это определяемый пользователем тип, который состоит из данных полей (переменные-члены) и членов, оперирующих этими данными (конструкторов, свойств, методов, событий и т.п.). Все вместе поля данных класса представляют «состояние» экземпляра класса (иначе называемого объектом). Поддерживая инкапсуляцию, класс, во-первых, связывает данные с кодом, во-вторых, предоставляет средства управления доступом к членам класса.

C# использует спецификацию класса для создания объекта. Объекты — это экземпляры класса. Таким образом, класс — это множество намерений (планов), определяющих, как должен быть построен объект. Важно четко понимать следующее: класс — это логическая абстракция. О ее реализации нет смысла говорить до тех пор, пока не создан объект класса, и в памяти не появилось физическое его представление.

Класс является типом данных, экземпляром которого является объект.

1.1. Общая форма определения класса

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 1
             }
     доступ тип_возврата метод2 (параметры)
             {
               // тело метода 1
             }
     доступ тип_возврата методN (параметры)
             {
               // тело метода N
             }
    }
```

Объявление каждой переменной и каждого метода предваряется элементом доступ. Здесь элемент доступ означает спецификатор доступа (например, public), который определяет, как к этому

Created with the Freeware Edition of HelpNDoc: Free HTML Help documentation generator

члену можно получить доступ.

Управление доступом к членам класса достигается за счет использования четырех спецификаторов доступа: **public, private, protected** и **internal**.

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.

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

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

1.2. Методы класса

Любой метод содержит одну или несколько инструкций. В хорошей С#-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. 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**.

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);
- для возврата значений. Методы возвращают значения вызывающим их процедурам, используя следующую форму инструкции return:

return значение;

где элемент значение и представляет значение, возвращаемое методом.

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.

При вызове методу можно передать одно или несколько значений Существует несколько модификаторов, с помощью которых можно управлять способом передачи аргументов интересующему методу.

Значение, передаваемое методу, называется аргументом. Переменная внутри метода, которая принимает значение аргумента, называется параметром. Параметры объявляются внутри круглых

Created with the Freeware Edition of HelpNDoc: Free HTML Help documentation generator

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.

Так как только методы класса имеют доступ к private-элементам класса, то для полного доступа к закрытым членам класса необходимо применять методы доступа (метод get) и изменения (метод set). Например,

```
class Point
    {
     public int X;
     int Y;
     public int getY() { return Y;}   // описание метода доступа к Y
     public void setY(int y) { Y=y;} // описание метода изменения Y
    };
static void Main()
    {
     Point p1=new Point();
     p1.X=3;
     p1.Y=4; // Ошибка! p1.Y недоступен из-за его уровня защиты
     p1.setY(4);
     double R=Math.Sqrt(p1.X*p1.X + p1.Y*p1.Y); // Ошибка! p1.Y недоступен из-за
                                                 // его уровня защиты
     double R=Math.Sqrt(p1.X *p1.X + p1.getY()*p1.getY());
   }
```

1.2. Конструкторы класса

В C# поддерживается механизм конструкторов, которые позволяют ус танавливать состояние объекта в момент его создания. Конструктор — это специальный метод класса, который вызывается неявно при создании объекта с исполь зованием ключевого слова **new**. Однако в отличие от "нормального" метода, конструктор никогда не имеет возвращаемого значения(даже **void**) и всегда именуется идентично имени класса, который он конструирует. Конструктор предназначен для инициализации объекта и вызывается автоматически при его создании.

Конструкторы — это члены классов, используемые для создания объектов — экземпляров классов. Конструктор имеет то же имя, что и класс, членом которого он является, и не имеет воз вращаемого значения.

Конструктор класса вызывается вся кий раз, когда создается объект его класса, и инициализирует объект при его создании.

Каждый класс C# снабжается конструктором по умолчанию, который при необходимости может быть переопределен. Если в классе не определен конструктор, компилятор генери рует конструктор по умолчанию, не имеющий параметров.

По определению такой конструктор никогда не принимает аргументов После размещения нового объекта в памяти конструктор по умол чанию гарантирует установку всех полей в соответствующие стандартные значения, которые являются принятыми для них по умолчанию:

- значение false для переменных типа bool;
- значение 0 для переменных числовых типов (или 0.0 для типов с плавающей точкой);
- одиночный пустой символ для переменных типа string;
- значение 0 для переменных типа BigInteger;
- значение 1/1/0001 12:00:00 AM для переменных типа DateTime;
- значение null для переменных типа объектных ссылок (включая string).

Формат записи конструктора имеет вид:

```
доступ имя_класса()
        {
```

Created with the Freeware Edition of HelpNDoc: Free HTML Help documentation generator

// тело конструктора
}

Обычно конструктор используется, чтобы придать переменным экземпляра, определенным в классе, начальные значения или выполнить исходные действия, необходимые для создания полностью сформированного объекта. Кроме того, обычно в качестве элемента доступ используется модификатор доступа **public**, поскольку конструкторы, как правило, вызываются вне их класса.

В подавляющем большинстве случаев реализация конструктора класса по умолчанию намеренно оста ется пустой, поскольку все, что требуется — это создание объекта со значениями всех полей по умолчанию.

Обычно помимо конструкторов по умолчанию в классах определяются специальные конструкторы. При этом пользователь объекта обеспечивается простым и согласованным способом инициализации состояния объекта непосредственно в момент его создания.

Формат записи специального конструктора имеет вид:

доступ имя_класса(список_параметров)
{
// тело конструктора
}

где элемент **список_параметров** представляет собой последовательность пар (состоящих из типа данных и идентификатора), разделенных запятыми. Параметры — это переменные, которые получают значения аргументов, передаваемых конструктору для инициализации переменных класса.

Как только определен специальный конструктор, конструктор по умолчанию удаляется из класса и становится недоступным, то есть, если не определен специальный конструктор, компилятор C# снабжает класс конструктором по умолчанию, чтобы позволить пользователю объекта размещать его в памяти с набором данных, имеющих значения по умолчанию.

Таким образом, чтобы позволить пользователю объекта создавать экземпляр типа посредством конструктора по умолчанию, а также специального конструктора, необходимо явно переопределить конструктор по умолчанию.

1.3. Деструкторы

Средства языка C# позволяют определить метод, который должен вызываться непосредственно перед тем, как объект будет окончательно разрушен системой сбора мусора. Этот метод называется деструктором, и его можно использовать для обеспечения гарантии "чистоты" ликвидации объекта. Например, деструктор можно использовать для гарантированного закрытия файла, открытого некоторым объектом.

Деструкторы уничтожают объекты класса и освобождают занимаемую этими объектами память.

В классе не может быть более одного деструктора.

Синтаксис деструктора очень похож на синтаксис конструктора по умолча нию за исключением того, что его имени предшествует символ "тильда"(~). Деструктор не должен иметь ни параметров, ни типа возвращаемого значения.

Формат записи деструктора имеет вид:

~имя_класса()
{
// код деструктора
}

где элемент **имя_класса** здесь означает имя класса.

Чтобы добавить деструктор в класс, достаточно включить его как член. Он вызывается в момент, предшествующий процессу утилизации объекта. В теле деструктора указываются действия, которые должны быть выполнены перед разрушением объекта.

Деструктор вызывается только перед началом работы системы сбора мусора и не вызывается когда объект выходит за пределы области видимости. (Этим C#-деструкторы отличаются от C++-деструкторов, которые как раз вызываются, когда объект выходит за пределы области видимости.) Это означает, что невозможно точно знать, когда будет выполнен деструктор,

Created with the Freeware Edition of HelpNDoc: Free HTML Help documentation generator

следовательно, из-за недетерминированных условий вызова деструкторы не следует использовать для выполнения действий, которые должны быть привязаны к определенной точке программы.

1.4. Создание объектов

Объявления объектов и вызовы методов класса осуществляют в основной программе, что содержит функцию Main(). Для создания объекта любого класса используется оператор **new**, формат записи которого имеет вид

переменная_типа_класса = new имя_класса();

где элемент **переменная_типа_класса** означает имя создаваемой переменной типа класса, элемент **имя_класса** означает имя реализуемого в объекте класса. Имя класса вместе со следующей за ним парой круглых скобок — это конструктор реализуемого класса. Если в классе конструктор не определен явным образом, оператор new будет использовать конструктор по умолчанию, который предоставляется средствами языка C#.

Таким образом, оператор new можно использовать для создания объекта любого "классового" типа.

При создании объектов выделяется память только для сохранения атрибутов, или данных объекта. Память, необходимая для сохранения методов класса выделяется только раз, поэтому все объекты объявлены в программе, используют одни и те же методы, которые берегутся в памяти отдельно, т. е. один набор методов для всех объектов.

Пример 1. В качестве примера создадим приложение, с помощью которого можно рассчитать распределение учебного времени по дисциплинам. При реализации приложения будем использовать класс **Discipline**, который инкапсулирует информацию об учебной дисциплине (название, количество занятий (лекций, практических и самостоятельных работ) и часов, отводимых на каждый вид занятия). Результат выполнения приложения «Классы и объекты» приведен на рис.1.

```
class Discipline
  {
     public string nameDisc; // название дисциплины
     public int countLek;     // количество лекций в дисциплине
     public int countLR;      // количество лаб. работ в дисциплине
     public int countSR;      // количество сам. работ в дисциплине
     int TimeLek=2;           // количество часов на лекцию
     int TimeLR=4;            // количество часов на лаб. работу
     int TimeSR=2;            // количество часов на сам. работу
     public Discipline()
     {
        nameDisc = "";
        countLek = 0;
        countLR = 0;
        countSR = 0;
     }
     public Discipline(string name, int lek, int lr, int sr)
     {
        nameDisc = name;
        countLek = lek;
        countLR = lr;
        countSR = sr;
     }
     public int Sum()
     {
        return countLek * TimeLek + countLR * TimeLR + countSR * TimeSR;
     }
```

Created with the Freeware Edition of HelpNDoc: Free HTML Help documentation generator

}

В классе **Discipline** объявлены семь переменных, 2 конструктора, позволяющих инициализировать данные (название дисциплины, количество занятий), а также метод, с помощью которого можно рассчитать количество часов, отводимых на дисциплину.

Чтобы реально создать объект класса **Discipline**, используется оператор:

Discipline Disc1 = new Discipline();

Это объявление выполняет две функции. Во-первых, оно объявляет переменную с именем **Disc1** классового типа **Discipline**. Но эта переменная не определяет объект, а может лишь ссылаться на него. Во-вторых, рассматриваемое объявление создает реальную физическую копию объекта и присваивает переменной **Disc1** ссылку на этот объект. И все это — "дело рук" оператора **new**. Таким образом, после выполнения приведенной выше строки кода переменная **Disc1** будет ссылаться на объект типа **Discipline**.

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.

Предыдущую инструкцию, объединяющую в себе два действия, можно переписать в виде двух операторов:

Discipline Disc1; // объявление ссылки на объект.

Disc1 = new Discipline(); // выделение памяти для объекта типа Building.

В первой строке объявляется переменная **Disc1** как ссылка на объект типа **Discipline**. Поэтому **Disc1** — это переменная, которая может ссылаться на объект, но не сам объект. В этот момент (после выполнения первой инструкции) переменная **Disc1** содержит значение **null**, которое означает, что она не ссылается ни на какой объект. После выполнения второй инструкции будет создан новый объект класса **Discipline**, а ссылка на него будет присвоена переменной **Disc1**. Вот теперь ссылка **Disc1** связана с объектом.

Для создания второго объекта **Disc2** класса **Discipline** используется специальный конструктор:

```
public Discipline(string name, int lek, int lr, int sr)
    {
        nameDisc = name;
        countLek = lek;
        countLR = lr;
        countSR = sr;
    }
```

Кроме того, для создания и запуска объекта, в C# предлагается синтаксис инициализатора объекта. С помощью этого механизма можно создать новую объектную переменную и присвоить значения множеству общедоступных полей.

Формат записи конструктора с использованием синтаксиса инициализатора объекта имеет вид:

```
имя_класса переменная_типа_класса = new имя_класса {
                                        имя_переменной1=значение,
                                        имя_переменной2=значение,
                                              . . . ,
                                        имя_переменнойК=значение};
```

где имя_переменной означает имя общедоступного поля данного класса.

Для нашего примера оператор создания объекта **Disc3** будет иметь вид:

Discipline Disc3 = new Discipline { nameDisc="Математика", countLek=18, countLR=36, countSR=36};

При каждом создании экземпляра класса создается объект, который содержит собственную копию каждой переменной экземпляра, определенной этим классом. Таким образом, каждый объект класса **Discipline** будет содержать собственные копии переменных экземпляра: **nameDisc, countLek, countLR, countSR, TimeLek, TimeLR, TimeSR**.

Для доступа к членам класса используется оператор "точка" (.). Оператор "точка" связывает имя

Created with the Freeware Edition of HelpNDoc: Free HTML Help documentation generator

объекта с именем его члена. Общий формат этого оператора имеет вид **объект.член**, то есть объект указывается слева от оператора "точка", а его член — справа. Например, чтобы присвоить переменной **nameDisc** значение “**Информатика**”, используется оператор:

Disc1.nameDisc = "Информатика";

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

```
class Program
  {
     static void Main()
     {
        // Создание объекта с установкой каждого значения "вручную"
        Discipline Disc1 = new Discipline();
        Disc1.nameDisc = "Информатика";
        Disc1.countLek = 9;
        Disc1.countLR = 9;
        Disc1.countSR = 18;
        Console.WriteLine("  Общее число часов, отводимое на дисциплину <" + Disc1.nameDisc +
">, равно " + Disc1.Sum());
        // Создание объекта с использованием специального конструктора
        Discipline Disc2 = new Discipline("Программирование", 36, 18, 18);
        Console.WriteLine("  Общее число часов, отводимое на дисциплину <" + Disc2.nameDisc +
">, равно " + Disc2.Sum());
        // Создание объекта с использованием списка инициализации
        Discipline Disc3 = new Discipline { nameDisc="Математика", countLek=18, countLR=36,
countSR=36};
        Console.WriteLine("  Общее число часов, отводимое на дисциплину <" + Disc3.nameDisc +
">, равно " + Disc3.Sum());
        Console.WriteLine();
        Console.WriteLine("  Общее число учебных часов в семестре равно  " + (Disc1.Sum() +
Disc2.Sum() + Disc3.Sum()));
        Console.WriteLine();
     }
  }
```

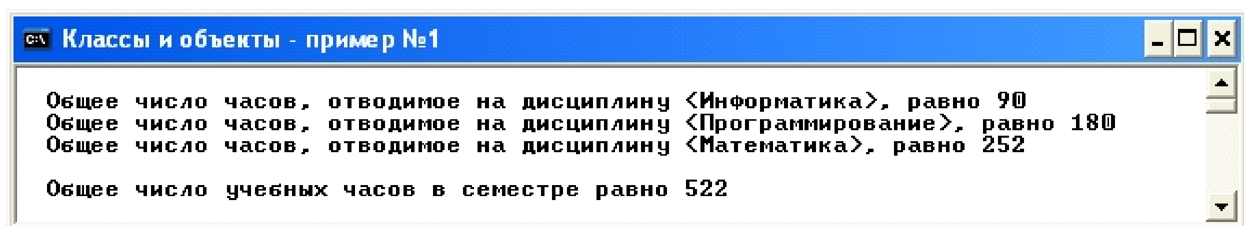

Рис.1. Результат выполнения приложения «Классы и объекты»

1.5. Массивы объектов

Так как массив в C# — это коллекция переменных одинакового типа, обращение к которым происходит с использованием общего для всех имени, то можно организовать работу с массивами объектов.

Для работы с массивами объектов сначала объявляется класс, например, **Discipline** (см. пример №1), после чего объявляется ссылочная переменная на массив, а затем для него выделяется память, и переменной массива присваивается ссылка на эту область памяти, например, с помощью оператора:

Discipline [] Disc = new Discipline[5];

Created with the Freeware Edition of HelpNDoc: Free HTML Help documentation generator

создается массив объектов **Disc** (состоящий из 5 объектов типа **Discipline**).

Доступ к отдельному элементу массива осуществляется посредством индекса (индекс описывает позицию элемента внутри массива, в C# первый элемент массива имеет нулевой индекс), а элементу объекта – посредством имени члена класса, например, оператор:

Disc[3].countLR = 18;

записывает количество лабораторных работ в 4-й объект.

2. Рабочее задание

Задание 1. Руководствуясь теоретическим материалом раздела 1 изучить возможности языка C# по созданию приложений, использующие классы и объекты, и выполнить пример №1, описанный в этом разделе.

Задание 2. Модифицировать приложение «Классы и объекты (модифицированный вариант)» таким образом, чтобы можно было изменять значения количество часов на лекцию, лабораторную и самостоятельную работы. Значения переменных nameDisc, countLek, countLR, countSR, TimeLek, TimeLR, TimeSR для каждой дисциплины вводятся с клавиатуры.

Задание 3. Модифицировать приложение «Классы и объекты (массивы объектов)» таким образом, чтобы количество дисциплин и значения переменных nameDisc, countLek, countLR, countSR, TimeLek, TimeLR, TimeSR для каждой дисциплины можно было вводить с клавиатуры.

Задание 4. Разработать приложение с заголовком «Решение квадратного уравнения», позволяющее получить значения корней квадратного уравнения.

Для чего необходимо разработать класс с именем Solution, который содержит три закрытых поля, предназначенных для хранения коэффициентов квадратного уравнения, и методы для хранения данных и решения квадратного уравнения.

Исходные данные вводятся с клавиатуры.

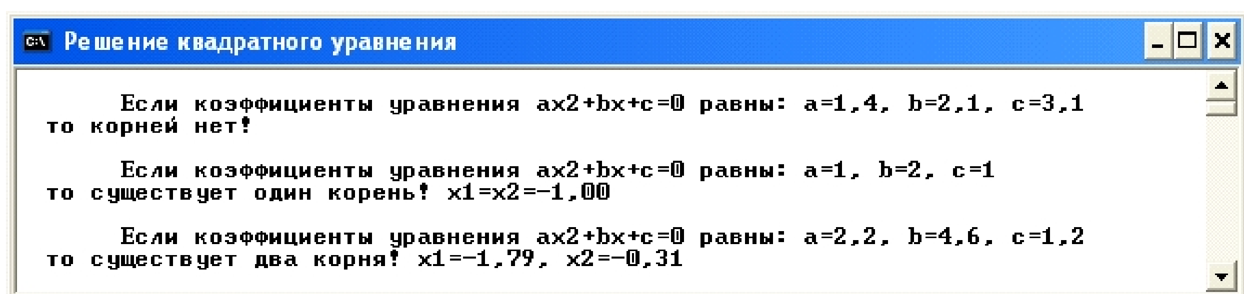

Рис. 2. Один из вариантов результата выполнения приложения

3. Контрольные вопросы

1. Что такое класс и его объявление?
2. Перечислите модификаторы доступа и их назначение.
3. Что такое объект и его создание?
4. Что такое конструктор и его назначение?
5. Какие значения принимают переменные разных типов данных при вызове конструктора по умолчанию?
6. Что такое деструктор и его назначение?
7. Как создать массив объектов?
8. Как осуществляется доступ к элементам объекта?

Created with the Freeware Edition of HelpNDoc: Free HTML Help documentation generator

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 HTML Help documentation generator