Классы.
Свойства классов.
Методы классов

Классы в языке C#

- **Класс** – описание множества схожих объектов.

  Класс определяет свойства описываемых им объектов, методы (способы) получения информации об объектах, изменения свойств и поведения объектов. Описание класса в языке C#:

- ```
  namespace MyNameSpace // Пространство имён класса
  {

      class MyClass             // Объявление класса
      {
        public static void Main()
        {
            /* Описание метода Main() класса MyClass */
        }
      }

  }
  ```

Метод Main() при запуске приложения выполняется первым.
  ```

Классы в языке C#

- Создание объекта, представляющего рассматриваемый класс, осуществляется при помощи ключевого слова **new** :

  MyClass MyObject = new MyClass() ;

- На базе одного класса может быть создано неограниченное количество объектов-представителей.

- В .NET для классов, их методов и свойств доступны следующие модификаторы доступа:
  – **public** – доступен всем;
  – **protected** – доступен только самому объекту и его потомкам;
  – **private** – доступен только самому объекту, но не его потомкам;
  – **internal** – доступен в пределах текущей сборки;
  – **protected internal** – доступен всем из текущей сборки.

- Модификаторы доступа применимы к свойствам, методам и переменным класса. Однако если требуется открыть доступ к переменной класса, её лучше превратить в свойство.

Классы в языке C#

- Модификаторы доступа также могут быть применены к классам.

  Модификатор **public** делает класс доступным для всех объектов.

  Модификатор **internal** делает класс доступным только внутри определенной сборки.

  Модификаторы **private** и **protected** могут использоваться только для вложенных классов.

- Если для переменной, свойства, метода или класса не был указан модификатор доступа, то по умолчанию для них используется модификатор **private**.

- Пример «опубликования» метода **M1()** класса **MyClass** :

```
class MyClass
{
      public static void M1()
  { /* Описание метода M1() */ }
}
```

Свойства

- Свойства – это атрибуты объектов класса.

  Свойства, как и переменные описываются в рамках определенного класса. Тем не менее, в языке C# свойства отличаются от переменных. В частности, переменные принято именовать с маленькой буквы, а свойства – с большой.

- Пример описания переменных и методов класса:

```
class MyClass
{

    int width;  // Описание private-переменной класса
    int height; // Описание private-переменной класса

    public int Width { /* Описание свойства класса */ }
    public int Height { /* Описание свойства класса */ }

}
```

Свойства

- Фигурные скобки в описании свойства позволяют задать блок действий, обеспечивающих корректную работу с данным свойством (например, задание значения свойства).
- Доступ к свойству осуществляется при помощи двух методов-аксессоров (от англ. *access* – доступ) **get** (для чтения значения) и **set** (для записи значения).

```
class MyClass
{
    int width;
    int height;

    public int Width
    {
        get { return width; } // Описание аксессора get
        set { width = value; } // Описание аксессора set
    }
}
```

Свойства

- Переменная **value** является виртуальной и имеет тот же тип, что и свойство.
- Ключевым отличием свойства от переменной является возможность ограничения операций чтения и записи его значений.

```
class MyClass
{
    int width;
    int height;

    public int Width // Свойство доступно только для чтения
    {
        get { return width; }
    }
}
```

Свойства

```
class MyClass
{
    int width;
    int height;

    public int Width   // Свойству может быть задано
                       // только положительное значение
    {
        get { return width; }

        set
        {
            if (value > 0) width = value;
        }
    }
}
```

Свойства

- Обращение к свойству объекта, представляющего описанный класс:

  ИмяОбъект.Свойство

- Пример:

```
// Создаем переменную MyObject и записываем в неё
// ссылку на созданный объект класса MyClass
MyClass MyObject = new MyClass() ;

// Устанавливаем значение свойства Width объекта MyObject
MyObject . Width = 100;

// Выводим на консоль значение свойства Width (100)
Console.WriteLine( MyObject . Width );
```

Свойства

- Аксессоры могут быть описаны с использованием модификаторов доступа.

- ```
  class MyClass
  {
      int width;
      int height;

      public int Width   // Значение свойства
                         // нельзя изменить извне
      {
          get { return width; }
          private set { width = value; }
      }
  }
  ```
  ```

Методы

- Описание метода осуществляется по следующей схеме:
  МодификаторДоступа Статичность ТипВозвращаемогоЗначения ИмяМетода(СписокПараметров)
  { *ОписаниеМетода* **}**

- ```
  class MyClass
  {
   double width;

        public void MyMethod (double width) // Описание метода
        {
              Console.WriteLine( width ); // Код метода
        }
  }
  ```
  ```

Методы

- При помощи ключевого слова **return** работу метода можно прервать в любой момент, вернув некоторое значение:

```
public double Sum (double a, double b) // Сумма двух чисел
{
 return a + b;
}
```

- Ключевое слово **return** также может быть использовано для метода, не предполагающего возврата значения (тип – **void**):

```
public void MyMethod () // Метод, не возвращающий значение
{
 return ;
}
```

Параметры методов

- По умолчанию переменные передаются в метод по значению. Это означает, что внутри себя метод создает локальные переменные с именами параметров, присваивает им указанные значения, а после работы – удаляет локальные переменные.
- Таким образом, значения самих исходных переменных по умолчанию не могут быть изменены методом.

```
static  void Product (double a,  double b) // Произведение чисел
{
 Console.WriteLine( a  *  b ); // Вывод результата на консоль

 a  +=  2; // Попытка изменить переменную a
 b  -=  2;  // Попытка изменить переменную b

 Console.WriteLine( a  *  b );  // Вывод нового результата
}
```

Параметры методов

- Проверим, изменятся ли переменные после вызова метода.

```
static  void  Main ()
{
 double  a = 3,  b = 4;  // Объявляем переменные

 Product(a,  b);  //  На консоль выводятся числа 12 и 10

 Console.WriteLine( a );  //  3
 Console.WriteLine( b );  //  4
}
```

- Примечание. Вместо имен переменных в качестве параметров метода при его вызове могут быть указаны конкретные числа.

```
Product(3,  4);  //  Ошибки не возникнет
```

Параметры методов

- Чтобы метод использовал не значение переданной переменной, а непосредственно саму переменную, следует перед указанием типа и имени параметра использовать ключевое слово **ref** (от английского *Reference* – ссылка).

```
static void Product (ref double a, ref double b)
{
  Console.WriteLine( a * b ); // Вывод результата на консоль

  a += 2; // Попытка изменить переменную a
  b -= 2; // Попытка изменить переменную b

  Console.WriteLine( a * b ); // Вывод нового результата
}
```

- **Примечание.** При вызове метода с параметром-ссылкой перед именем переменной также следует указывать **ref**. Переменная перед этим обязательно должна быть проинициализирована.

Параметры методов

- Проверим, изменятся ли переменные после вызова метода.

```
static void Main ()
{
 double a = 3, b = 4; // Объявляем переменные

 Product(ref a, ref b); // Выводятся числа 12 и 10

 Console.WriteLine( a ); // 5
 Console.WriteLine( b ); // 2
}
```

- Примечание. Если при вызове метода вместо имени переменной в качестве параметра-ссылки будет указано конкретное число, возникнет ошибка.

```
Product(3, 4); // Ошибка
```

Параметры методов

- Если переменная, указываемая в качестве параметра метода, предназначена только для возврата через неё значения бз использования внутри метода, то её можно сделать «выходной», указав перед ней ключевое слово **out**. В этом случае её не требуется инициализировать в обязательном порядке.

```
static  void Product (double a,  double b ,  out  double res)
{
 res  =  a  *  b; // Результат запишется в переменную res

  Console.WriteLine( a  *  b );  // И будет показан в консоли
}
```

- **Примечание.** При вызове метода с выходным параметром перед именем переменной также следует указывать **out**. Саму переменную инициализировать не обязательно, однако её значение обязательно должно измениться, иначе компилятор выдаст ошибку.

Параметры методов

- Проверим, изменятся ли переменные после вызова метода.

```
static  void  Main ()
{
 double  a = 3,  b = 4,  res;  // Объявляем переменные

 Product(a, b , out res);  //  Выводится число 12

 Console.WriteLine( a );  //  3
 Console.WriteLine( b );  //  4
 Console.WriteLine( res );  //  12
}
```

- Примечание. Если при вызове метода вместо имени переменной в качестве параметра-ссылки будет указано конкретное число, возникнет ошибка.

```
Product(3,  4,  12);  //  Ошибка
```

Параметры методов

- В C# имеется возможность описывать методы с произвольным количеством **однотипных** параметров. Для этого всё множество однотипных параметров указывается как один параметр-массив, перед которым используется ключевое слово **params**. Далее в методе осуществляется работа с каждым элементом массива параметров.

```
static  double Sum (params  double[] parameters)
{
 double  s = 0; //  Инициализируем переменную суммы

 foreach (double par  in parameters)  s += par; // Суммируем

 return  s;  // Возврат суммы переданных параметров
}
```

- **Пример вызова:** Console.WriteLine(**Sum**(**1, 2, 4, 6, 8**)); // 21

Параметры методов

- **Примечание.** У метода может быть только один параметр с модификатором **params**, и он должен быть указан последним в списке параметров. Без этих двух условий компилятор не смог бы определить окончание произвольного списка параметров.
- Примеры **ошибочного** использования модификатора **params**:

```
static double Sum(params string[] str, params double[] pars)
{/* Код метода */ }

static double Sum(params double[] pars, string str)
{/* Код метода */ }
```

- Правильное описание:

```
static double Sum(string str, params double[] pars)
{/* Код метода */ }
```