Overloading

Overloading a method

Overloading a method allows us to declare more than one method with the same name in the same class. However, it is required that all methods with the same name take a different number of parameters or have different parameter types. The methods with the same name are called overloaded methods.

The most common example of this is to have more than one constructor with the same name, which allows us to create the object with different types of parameters, or a different number of parameters.

Overloading can be done, however, if one method takes a **ref** or out argument and the other uses neither, like this:

Ex1:

```
class RefOutOverloadExample
{
   public void SampleMethod(int i) {  }
   public void SampleMethod(ref int i) {  }
 }
```

Ex2:

```
class RefOutOverloadExample
{
   public void SampleMethod(int i) {  }
   public void SampleMethod(out int i) {  }
}
```

Passing Arrays Using ref and out

Like all out parameters, an **out** parameter of an array type must be assigned before it is used; that is, it must be assigned by the callee. For example:

```
void TestMethod1(out int[] arr)
{
   arr = new int[10];   // definite assignment of arr
}
```

Like all ref parameters, a **ref** parameter of an array type must be definitely assigned by the caller. Therefore, there is no need to be definitely assigned by the callee. A **ref** parameter of an array type may be altered as a result of the call. For example, the array can be assigned the null value or can be initialized to a different array. For example:

```
void TestMethod2(ref int[] arr)
{
   arr = new int[10];   // arr initialized to a different array
}
```

Example 1:

In this example, the array **theArray** is declared in the caller (the Main method), and initialized in the **FillArray** method. Then, the array elements are returned to the caller and displayed.

```
class TestOut
{
   void FillArray(out int[] arr)
   {
      // Initialize the array:
      arr = new int[5] { 1, 2, 3, 4, 5 };
   }

   static void Main()
   {  TestOut ob=new TestOut();
      int[] theArray;       // Initialization is not required

      ob. FillArray(out theArray); // Pass the array to the callee using out:


      // Display the array elements:
      System.Console.WriteLine("Array elements are:");
      for (int i = 0; i < theArray.Length; i++)
      {
         System.Console.Write(theArray[i] + " ");
      }
}
```

Output 1

Array elements are:

1 2 3 4 5

H.W

Write complete program that contains the array (theArray) which is initialized in the caller (the Main method), and passed to the FillArray method by using the **ref** parameter. Some of the array demands are updated in the FillArray method. Then, the array elements are returned to the caller and displayed.

Object

يوجد نوع من البيانات إسمه **Object** وهو يأخذ جميع الأنواع . إذا قمنا بإسناد قيمة إليه تسمى هذه العملية **Boxing** وإذا قمنا باخذ قيمه منه من خلال متغير اخر تسمى العملية **UnBoxing** .

Boxing and unboxing are the processes that enable value types (e.g., integers) to be treated as reference types (objects). The value is "boxed" inside an Object, and subsequently "unboxed" back to a value type.

- Boxing Is Implicit (ضمنياً)

Boxing is an implicit conversion of a value type to the type Object. Boxing a value allocates an instance of the boxed type and copies the value into the new object instance.

- Unboxing Must Be Explicit (واضح)

To return the boxed object back to a value type, you must explicitly unbox it.

Example1 : Boxing and unboxing

Example1 creates an integer i and implicitly boxes it when it is assigned to the object o. The value is then explicitly unboxed and assigned to a new int whose value is displayed.

```
using System;
namespace boxing
{
  public class UnboxingTest
  {
    public static void Main( )
    {
```

```
        int i = 123;

        object o = i;         //Boxing

        int j = ( int ) o;    // unboxing (must be explicit)

        Console.WriteLine( "j: {0}", j );
      }
   }
}
```

Output:
j: 123

Example 2:

```
using System;
class xxx
{
   public int i = 10;
}

class MainClass
{
   static void Main()
   {
      object a;

      a = 1;   // an example of boxing
      Console.WriteLine(a);

      a = new xxx();
      xxx b;
      b = (xxx)a;
      Console.WriteLine(b.i);
   }
}
```

Output
1
10

EX:

```
using System;
public class MyClass
{

   public void UseParams(params int[] list)
   {
      for (int i = 0 ; i < list.Length; i++)
      {
         Console.WriteLine(list[i]);
      }
      Console.WriteLine();
   }

   public void UseParams2(params object[] list)
   {
      for (int i = 0 ; i < list.Length; i++)
      {
         Console.WriteLine(list[i]);
      }
      Console.WriteLine();
   }

   static void Main()
   {
     MyClass ob= new MyClass();

      ob.UseParams(1, 2, 3);
      ob.UseParams2(1, 'a', "test");

      // An array of objects can also be passed, as long as
      // the array type matches the method being called.

      int[] myarray = new int[3] {10,11,12};

      ob.UseParams(myarray);
   }
}
```

Output

```
1
2
3

1
a
test

10
11
12
```

Enumerations

Enumerations are data structures that store values with user-friendly names. This set of user-friendly named constants is called an *enumerator list*. The default data type of an enumerator list is an **integer.**(It is used to create a user-defined integer data type).

To declare the enumerator, use the **enum** keyword.

تتيح لنا لغة السي شارب صناعة أنواع جديدة غير المعروفة وإعطاء نطاق لها فمثلاً لو أنك مضطر لأن تستخدم متغير من نوع الأسبوع يعني أنك تريد إعطاء المتغير من هذا النوع قيمة أحد الأيام الموجودة في الأسبوع وقمت بالبحث عن نوع لفعل ذلك فلن تجده . لذلك سهلت علينا لغة السي شارب وقامت بمنحنا كلمة **Enum** والتى تقوم بعمل نوع جديد واليك الصيغة العامة لها :

<modifier> ***enum*** *<enumeration name>*
{ - - - - - - - - - - - - - - - };

Ex:

مثال على كيفية إستعمالها لصناعة نوع الأسبوع :

```
enum week { Sat , Sun , Mon , Tue , Wen , Thu , Fri } ;
```

وأنبه هنا أنه يجب عليك كتابتها قبل الـ **Class** يعني ليس داخل الدالة الرئيسية وإنما داخل الكلاس الحامل للدالة الرئيسية أو في منطقة بين كلمة `namespace` وكلمة `class` .

```
using System;

namespace First_Application_With_Console
{
      enum week { Sat , Sun , Mon , Tue , Wen , Thu , Fri } ;
      class Class1
      {

            static void Main(string[] args)
            {
                  week w1 = week.Tue ;
                  Console.WriteLine(w1);
            }
      }
}
```

في المثال السابق قمنا بإنشاء نوع جديد وقمنا بإنشاء كائن منه وإعطائه قيمة من القيم المتاحة له .
ولعلك تسأل لماذا لا نقوم بإعطائه القيمة مباشرة لماذا يجب علينا كتابة إسم النوع ؟
والجواب أن لغة السي شارب لا تعتبر هذا النوع من الأنواع الموجودة أصلاً في اللغة يعني ليست كلمة محجوزة لذلك يجب علينا كتابة إسم النوع الجديد ثم إتباعه بقيمته الجديدة .

The previous code creates an enumeration with the name week and declares all the possible values for week. The first element of the enumeration takes a default value of 0 and the successive elements take the previous value plus 1. You can also specify user-defined values for the elements of an enumeration. For example,

```
enum week { Sat=1,Sun , Mon , Tue , Wen , Thu , Fri } ;
```

In this case, the values of Sat ,Sun, Mon, and Tue are 1, 2, and 3, respectively.