المحاضرة الثانية

Passing Parameters to the method

Parameter Types:

C# allows four types of parameters in a parameter list:

- Input parameters
- Output parameters
- Reference parameters
- Parameter arrays

The params Keyword

You can create a method that displays any number of integers to the console by passing in an array of integers and then iterating over the array with a foreach loop. The ***params*** keyword allows you to pass in a variable number of parameters without necessarily explicitly creating the array.

The **params** keyword lets you specify a method parameter that takes an argument where the number of arguments is variable.

No additional parameters are permitted after the **params** keyword in a method declaration, and only one **params** keyword is permitted in a method declaration.

يسمح بوجود ب**params** واحد في أل method. ولا يسمح بإضافة أي متغيرات بعد **params** في تعريف ألmethod .

الكلمة المحجوزة params تَسْمحُ بإمرار عدد متغيّر مِنْ المدخلات بدون تَكوين مصفوفةً بشكل واضح.

Ex: Using the params keyword

```
using System;

namespace UsingParams
{
  public class Tester
  {
    public void DisplayVals(params int[] y)
```

```
    {
      foreach (int i in y)
      {
        Console.WriteLine("The Value {0}",i);
      }
    }

    static void Main( )
    {
      Tester t = new Tester( );
      t.DisplayVals(5,6,7,8);
      int [] x = new int[5] {1,2,3,4,5};
      t.DisplayVals(x);
    }
  }
}
```

Output:

```
The Value 5
The Value 6
The Value 7
The Value 8
The Value 1
The Value 2
The Value 3
The Value 4
The Value 5
```

Two-Dimensional array:

Rectangular array

```
     // int[,] mat = new int[3, 3];
        int[,] mat = new int[,] { { 0, 0, 0 }, { 1, 1, 1 },
    { 3, 3, 3 } };

                for (int i = 0; i < 3; i++)
                    for (int j = 0; j < 3; j++)
                        Console.WriteLine(mat[i,j]);
                Console.ReadKey();
```

Methods of array

No.of Dimension : Rank

total No.of elements in the array : Length

No.of D1 (Rows) : GetLength(0)

No.of D2(Column) : GetLength(1)

No.of D3 : GetLength(2)

lowerbound of D1 (Rows) : GetLowerBound(0)

lowerbound of D2(Column) : GetLowerBound(1)

upperBound of D1 (Rows) : GetUpperBound(0)

upperBound of D2(Column) : GetUpperBound(1)

Array (2D)

```
namespace array2[
{

  class array
  {
    int[,] x;

    public array(int n, int m)
      {
        x = new int[n, m];
      }
    public void Get_information()
     {
        Console.WriteLine("No.of Dimension = {0}\n ", x.Rank);
         Console.Write("total No.of elements in the array  ={0}\n\n",
x.Length);
         Console.Write(" No.of Rows  ={0}\n ", x.GetLength(0));
         Console.WriteLine("No.of Column={0}\n", x.GetLength(1));
         Console.Write("lowerbound of row: {0}\n", x.GetLowerBound(0));
         Console.WriteLine("upperBound of row: {0}\n",
x.GetUpperBound(0));
         Console.Write("lowerbound of column: {0}\n",
x.GetLowerBound(1));
         Console.WriteLine("upperBound of column: {0}\n",
x.GetUpperBound(1));
      }
```

```
        public void read_array()
        {
            for (int i = 0; i <= x.GetUpperBound(0); i++)
             {
               for (int j = 0; j <= x.GetUpperBound(1);j++)
                 {
                   Console.Write("Enter value at location {0},{1} In
Array: ", i, j);
                   x[i, j] = int.Parse(Console.ReadLine());
                 }
             }
        }

        public void write_Matrix()
           {
              Console.WriteLine();
              for (int i = 0; i <= x.GetUpperBound(0); i++)
                 {
                  for (int j = 0; j <= x.GetUpperBound(1);j++)
                   {
                       Console.Write("{0}\t",x[i, j]);
                   }
                   Console.WriteLine();
                 }
           }

           static void Main(string[] args)
           {
               array a=new array(3,5);
               a.Get_information();
               a.read_array();
               a.write_Matrix();
               Console.ReadKey();

           }
           }
       }
```

Jagged arrays

A jagged array is an array of arrays. It is called "jagged" because each row need not be the same size as all the others, and thus a graphical representation of the array would not be square.

تدْعَى "متعرج" لأن الصفّوف لَيْسَ مِنْ الضروري أَنْ يَكُونَ بنفس الحجمِ .وهكذا فان تمثيل المصفوفة لَنْ يَكُونَ مربّعَ.

When you create a jagged array, you declare the number of rows in your array. Each row will hold an array, which can be of any length.

عند تكوين مصفوفة غير منتظمة الصفوف (jagged) نعلن عدد الصفوف في المصفوفة. كل صف هو مصفوفة ويُمكنُ أَنْ تكُونَ بأيّ طول.

In a jagged array, each dimension is a one-dimensional array. To declare a jagged array, use the following syntax, where the number of brackets(الأقواس) indicates (يدل) the number of dimensions of the array:

***type* [] []...**

For example, you would declare a two-dimensional jagged array of integers named myJaggedArray as follows:

int [] [] x;

Access the ***fifth element*** of the ***third array*** by writing ***x[2][4].***

Example creates a jagged array named **x**, initializes its elements, and then prints their content. To save space, the program takes advantage of the fact that integer array elements are automatically initialized to 0, and it initializes the values of only some of the elements.

Ex: Working with a jagged array

```
using System;
using System.Collections.Generic;
using System.Text;

namespace JaggedArray
{
  public class Tester
  {
    static void Main( )
    {
      const int rows = 4;
      int[][] x = new int[rows][]; // declare the jagged array as 4 rows high

      x[0] = new int[5];    // the first row has 5 elements
      x[1] = new int[2];    // a row with 2 elements
      x[2] = new int[3];    // a row with 3 elements
      x[3] = new int[5];    // the last row has 5 elements
    ٥
```

```
      // Fill some (but not all) elements of the rows
        x[0][3] = 15;
        x[1][1] = 12;
        x[2][1] = 9;
        x[2][2] = 99;
        x[3][0] = 10;
        x[3][1] = 11;
        x[3][2] = 12;
        x[3][3] = 13;
        x[3][4] = 14;

        for ( int i = 0; i < 5; i++ )
        {
          Console.WriteLine( "x[0][{0}] = {1}",
            i, x[0][i] );
        }

        for ( int i = 0; i < 2; i++ )
        {
          Console.WriteLine( "x[1][{0}] = {1}",
            i, x[1][i] );
        }

        for ( int i = 0; i < 3; i++ )
        {
          Console.WriteLine( "x[2][{0}] = {1}",
            i, x[2][i] );
        }
        for ( int i = 0; i < 5; i++ )
        {
          Console.WriteLine( "x[3][{0}] = {1}",
            i, x[3][i] );
        }
      }
    }
}
```

Output:

```
x[0][0] = 0
x[0][1] = 0
x[0][2] = 0
x[0][3] = 15
x[0][4] = 0
```

x[1][0] = 0
x[1][1] = 12
x[2][0] = 0
x[2][1] = 9
x[2][2] = 99
x[3][0] = 10
x[3][1] = 11
x[3][2] = 12
x[3][3] = 13
x[3][4] = 14

The Lifecycle of an Object (دورة حياة الكائن)

The lifecycle includes two important stages: **Construction** and **Destruction**.

Construction:

When an object is first instantiated it needs to be initialized. This initialization is known as construction and is carried out by a constructor function.

عند تكوين الكائن مَنْ الضَّرُوري أَنْ يُهيئ بإعطائه قيمة ابتدائية، وهذه التهيئة تدعى بالبناءِ ويُنَفَّذ مِن قِبل دالة البناءِ.

Basic initialization of an object is automatic.

Table . Primitive types and their default values

Type	Default value
numeric (int, long, etc.)	**0**
Bool	**false**
Char	**'\0' (null)**
Enum	**0**
Reference	**null**

1. All objects have a **default** constructor, which is a **parameterless** method with the same name as the class itself.
   EX: Car object1=**new** Car ();
2. In addition, a class definition might include several constructor methods with **parameters**, known as **nondefault** constructors, that uses a parameter at instantiation:
   EX: Car object1=**new** Car (parameters);

To define a constructor, declare a method whose name is the same as the class in which it is declared. Constructors have no return type and are typically declared public. If there are arguments to pass, define an argument list just as you would for any other method.

Construction

EX1:

```
using System;
public class Test
{
   class xxx
   {
      public int x;  // int x
      public int y;

      public xxx(int p1, int p2)
      {
         x = p1;
         y = p2;
      }
   }

   static void Main()
   {
      xxx mC = new xxx(11, 22);
      Console.WriteLine("x = {0}, y = {1}", mC.x, mC.y);
   }
```

^

```
}
```

Output:

x = 11, y = 22

EX2:

```
using System;
class Circle
{
   const double PI = 3.141592;
    private double radius;

   public Circle()  // default constructor
   {
      radius = 5;
   }

    public double Area()
   {
      // return 3.141592 * radius * radius;
      return PI * radius * radius;
   }

   static void Main()
   {
      Circle c;
      c = new Circle();
      double areaOfCircle = c.Area();
      Console.WriteLine(areaOfCircle);
   }

}
```

اضف الـ method التالية إلى البرنامج السابق واستخدمها في الـ main.

```
public Circle(double initialRadius) // overloaded constructor
   {
      radius = initialRadius;
   }
```

Copy constructors

Copy constructors can be used for making copies of existing objects. A copy constructor can be recognized by the fact that it takes a parameter of the same type as the class to which it belongs.

It is sometimes useful to have a constructor that creates an identical copy of an existing object.

H.W.

Define a class **String** that include constructors that will enable us to create an uninitialized string String s1;//string with length 0
And also to initialize an object with a string value at the time of creation like: String s2("well done");
It also include copy constructor that copy one string to another.
Write a complete program to implement the above class that should perform the following tasks:

1- Create an uninitialized string object.
2- Create object with string value.
3- Copy one string object to another
4- Display a desired string objects.

Garbage Collection & Destructors:

C# garbage collection system reclaims objects automatically. When no references to an object exist, that object is assumed to be no longer needed & the memory occupied by the object is released.

Destructor:

It is possible to define a destructor method that will be called prior to an objects final destruction by the garbage collection for example; you might use a destructor to make sure that an open file is closed.

```
~ classname ( )
{
    // statement  ;
}
```

EX:

```
~ Circle ()
  {
       Console.WriteLine("The object is destroyed");

  }
```