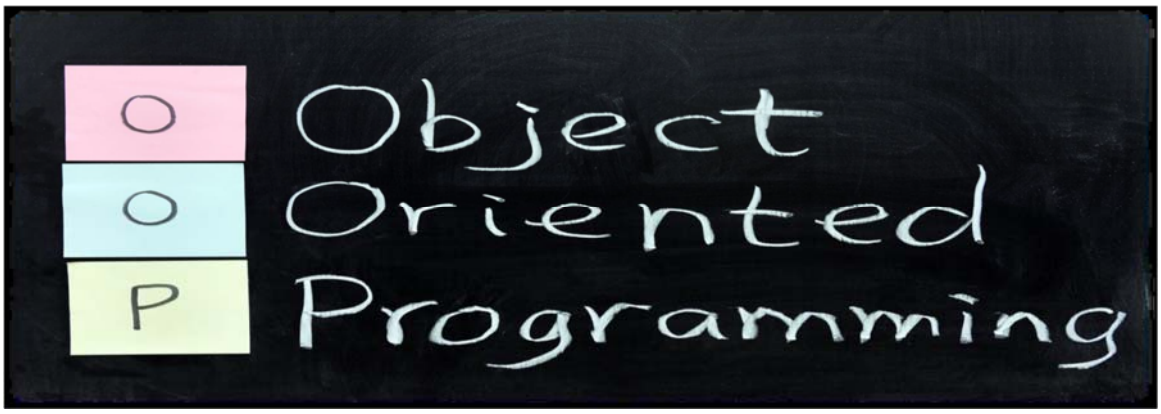
Object Oriented Programming
OOP



Advantages of OOP:

- Objects help in task partitioning in the project.
- Secure programs can be built using data hiding.
- It can potentially map objects.
- Enables the categorization of the objects into various classes.
- Object-oriented systems can be upgraded effortlessly.
- Redundant codes can be eliminated using inheritance.
- Codes can be extended using reusability.
- Greater modularity can be achieved.
- Data abstraction increases reliability.
- Flexible due to the dynamic binding concept.

Disadvantages of OOP:

- It requires more resources.
- Dynamic behavior of objects requires RAM storage.
- Detection and debugging are harder in complex applications when the message passing is performed.
- Inheritance makes their classes tightly coupled, which affects the reusability of objects.

Class

Class: It is the basic unit of the OOP which is used to wrap up(تغليف) the data and the codes (encapsulation).
- The code and data that form the class are called the members of the class.
- A class is defined as a data structure used to create object.

Declaring and Defining Classes

To define a new type or class, first declare it, and then define its methods and fields.
Declare a class using the **class** keyword.

The syntax is as follows:

```
[access-modifiers] class <classname>  [:base-class [,interface(s)]]
   {
    class-body              //It contents  field, properties, methods,
event(members)
   }
```

Member Definitions

All members have their own accessibility level, defined in all cases by one of the following keywords:
public: Member is accessible (يمكن الوصول اليه)from any code
private: Member is accessible only from code that is part of the class (the default if no keyword is used)
protected: Member accessible only from code that is part of either the class or a derived class.

Example 1 :

```
using System;

class Car
{

// car properties (class Fields)
private int Number,Model;
public string factory_Design, Color;

// car functions (class Methods)
public void show(int N,string made,int M,string C)
{
Number=N;
Model=M;
factory_Design=made;
Color=C;
Console. Write("Car Number is : "+Number);
Console. Write("\nCar factory Design is : " + factory_Design);
Console. Write("\nCar Model is : " + Model);
Console. Write("\nCar Color is : " + Color);
}
}
class Program
{
static void Main(string[] rags)
{
Car MyCar = new Car();
MyCar.show(75341, "TOYOTA", 2013, "Red");
Console.ReadKey();
}
}
```

Object

- An instances of a class are called objects (instantiated).
- Objects are created in memory when your program executes.

Creating Objects

Car MyCar = new Car ( );

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 (CurrentTime);

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;
class xxx
{
      public int x;    // int x; or private int x;
      public int y;

      public xxx(int p1, int p2)
      {
         x = p1;
         y = p2;
      }
  class Program
 {
  static void Main()
  {
      xxx mC = new xxx(1, 2);
      Console.WriteLine("x = {0}, y = {1}", mC.x, mC.y);
      Console.ReadKey();

  }
  }
}
```

EX2:

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

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

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

Output:

x = 11, y = 22

EX3

```
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;
    }
```

Methods

- A method is a named sequence of statements. A method is very similar to a function or a subroutine. Each method has a name and a body. Most methods can be given some data (input paramerters) for processing and can return information (output), which is usually the result of the processing.
-The method is a function contained within (إحتوتْ ضمن) a class. A function is a general term (مصطلح عام) that is not contained within (الذي لَمْ يُحتَوى ضمن) a class.
-All C# methods are defined inline .

The syntax of a method is as follows:

```
access-modifiers  returnType methodName ( parameterList )
{
   // method body statements
}
```

EX.1:

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

The syntax of a method call is as follows:

```
objectname.methodName ( argumentList )
```

EX.1:

```
Circle c = new Circle();
double areaOfCircle = c.Area();
```

Calling the method

1. Using a methode name by itself to call method of the same class.
2. Using a variable that contains a refrence to an object followed by (.) & method name to call non-static method.
3. Using the class name followed by (.) & method name to call static method.

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

Input parameters are passed *by value* into methods. It is not allowed to change the value supplied by the caller.

المتغيرات التي تمرر قيمتها إلى الـــــ method. ولا تســـمح بتغيير القيمـة المجهزة(خارج الـ method).

Output parameters are parameters whose values are not set when the method is called. The ***out*** keyword must precede the parameter's type in the parameter list.

المتغيرات التي قِيَمَتها غير مهيأة عند استدعاء الدالة (أي تتم التهيئة أو إعطاء القيمة داخل الدالة). ويسبق المتغير بالكلمة المحجوزة out.

Reference parameters are passed *by* reference into methods. The ***ref*** keyword must precede the parameter's type in the parameter list.

المتغيرات التي تمرر عنوانها إلى الـــــ method. ويسبق المتغير بالكلمة المحجوزة ref.

Parameter arrays enable you to specify that your method accept a variable number of arguments. using the C# keyword ***params***,
You can use one parameter array in your method's parameter list. You can combine a parameter array with other parameters in your method's parameter list. If you use a parameter array in a method parameter list, however, it must be specified as the last parameter in the list. ***You cannot use the out or ref keywords on a parameter array.***

تَسْمحُ بإمرار عدد متغيّر مِنْ المدخلات ، بالإمكان استخدام params واحد في أخر الـ . method's parameter list
ولا يمكن استخدام الكلمات المحجوزة out or ref معها.

ref

The **ref** keyword causes arguments to be passed by reference. The effect is that any changes made to the parameter in the method will be reflected in that variable when control passes back to the calling method. To use a

ref parameter, both the method definition and the calling method must **explicitly** use the **ref** keyword. For example:

إنّ تأثيرَ استخدام *ref* هو ان أيّ تغييرات تجري على البارامترِ في الـــ method سَتنعكس خارج الـ method .
لإسْتِعْمال *ref*، يجب كتابتها بشكل صريح في كلا تعريف الـــ method واستدعاء الـــ method. على سبيل المثال:

```
class RefExample
{
    void Method(ref int i)
    {
       i = 44;
    }

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

       int val = 0;
       ob.Method(ref val);
       Console.WriteLine(val);              // val is now 44
    }
 }
```

- An argument passed to a **ref** parameter must first be initialized.
يجب إعطاء المتغير الذي يمرر باستخدام **ref قيمة ابتدائية قبل استدعاء** الـ method **.**

out

The **out** keyword causes arguments to be passed by reference. This is similar to the ref keyword, ***except that ref requires that the variable be initialized before being passed***. To use an **out** parameter, both the method definition and the calling method must explicitly use the **out** keyword. For example:

إنّ تأثيرَ استخدام ***out*** هو نفس ***ref*** وهو ان أيّ تغييرات تجري على البارامترِ في الـــ method سَتنعكس خارج الدالة. وبنفس الطريقة أي لاستعمال ***out*** ، يجب كتابتها بشكل صريح في كلا تعريف الـ method واستدعاء الـ method . على سبيل المثال:

```
class OutExample
{
   void Method(out int i)
   {
      i = 44;
   }
   static void Main()
   {
      OutExample ob=new OutExample();
       int value;
       ob.Method(out value);            // value is now 44
   }
}
```

Although variables passed as an **out** arguments need not be initialized prior to being passed, the calling method is required to assign a value before the method returns.

لا تحتاج أن تعطي قيمة ابتدائية قبل استدعاء الmethod **ولكن تحتاج عملية** assign **للقيمة قبل الخروج من الـ**method

- Although **ref** and **out** are treated differently at run-time, they are treated the same at compile time. Therefore methods cannot be ***overloaded*** if one method takes a **ref** argument and the other takes an out argument. These two methods, for example, are identical in terms of compilation, so this code will not compile:

```
class CS0663_Example
{
   // compiler error CS0663: "cannot define overloaded
   // methods that differ only on ref and out"
   public void SampleMethod(ref int i) {  }
   public void SampleMethod(out int i) {  }
}
```

Example: method to return multiple values

```
class OutReturnExample
{
   void Method(out int i, out string s1, out string s2)
   {
      i = 44;
      s1 = "I've been returned";
      s2 = null;
   }
   static void Main()
   {   OutReturnExample ob= new OutReturnExample();
      int value;
      string str1, str2;
      ob.Method(out value, out str1, out str2);

       // value is now 44
      // str1 is now "I've been returned"
      // str2 is (still) null;
   }
}
```