Object Oriented Programming

PROPERTIES

Why Encapsulation?

- Better control of class members (reduce the possibility of yourself (or others) to mess up the code)
- Fields can be made **read-only** (if you only use the get method), or **write-only** (if you only use the set method)
- Flexible: the programmer can change one part of the code without affecting other parts
- Increased security of data

PROPERTIES

private variables can only be accessed within the same class (an outside class has no access to it). However, sometimes we need to access them - and it can be done with properties.

A property is like a combination of a variable and a method, and it has two methods: a get and a set method:

```
class Person

{

  private string name; // field

  public string Name   // property

  {

    get { return name; }   // get method

    set { name = value; }  // set method

  }

}
```

The Name property is associated with the name field. It is a good practice to use the same name for both the property and the private field, but with an uppercase first letter.

The get method returns the value of the variable name.

The set method assigns a value to the name variable. The value keyword represents the value we assign to the property.

```
using System;
program abstraction0
{
  class Person
  {
     private string name; // field



  }


static void Main(string[] args)
      {
         Person myObj = new Person();
         myObj.name = "Ahmad";
         Console.WriteLine("NAME = " + myObj.name);
         Console.ReadKey();
      }
   }
```

The output will be:

???

```
using System;
program abstraction0
{
  class Person
  {
     private string name; // field
     public string Name   // property
     {
        get { return name; }
        set { name = value; }
     }
  }


static void Main(string[] args)
      {
         Person myObj = new Person();
         myObj.Name = "Ahmad";
         Console.WriteLine("NAME = " + myObj.Name);
         Console.ReadKey();
      }
   }
```

The output will be:

Ahmad

Automatic Properties (Short Hand)

C# also provides a way to use short-hand / automatic properties, where you do not have to define the field for the property, and you only have to write get; and set; inside the property.

```
using System;

program Abstraction1
{
  class Person
  {
    public string Name  // property
    { get; set; }
  }

static void Main(string[] args)
      {
        Person myObj = new Person();
        myObj.Name = "Ahmad";
        Console.WriteLine("NAME = " + myObj.Name);
        Console.ReadKey();
      }
}
```

= public string name

The output will be:

```
Ahmad
```

```
using System;
program Program1
{
   class student
   {
      public string Name
         { get; set; }

      public int D1
         { get; set; }

      public int D2
         { get; set; }

      public int D3
         { get; set; }

      public double Av
         { get; set; }
   }
```

```
static void Main(string[] args)
      {
         student stu = new student();

         stu.Name = "Ali";
         stu.D1=100; stu.D2= 70; stu.D3=88;
         stu.Av= (D1 + D2 + D3) / 3;

         Console.WriteLine("NAME = " + stu.Name );
         Console.WriteLine(“FIRST MARK = “ + stu.D1);
         Console.WriteLine(“SECOND MARK =  “ + stu.D2);
         Console.WriteLine(“THERID MARK = “ + stu.D3);
         Console.WriteLine(“AVERAGE = “ + stu.Av);

         Console.ReadKey();
      }
}
```

CONSTANTS

C# enables to create class constants.

These constants do not belong to a concrete object. They belong to the class.

constants are written in uppercase letters.

```
using System;

namespace ClassConstants
{
    class Math
    {
        public const double PI = 3.14159265359;
    }

    class Program
    {
        static void Main(string[] args)
        {
            Math mymath = New Math();
            Console.WriteLine(mymath.PI );
            area = r * PI  ;
        }
    }
}
```

```
3.14159265359
```

We have a **Math** class with a **PI** constant.

public const double PI = 3.14159265359;

The **const** keyword is used to define a constant.

The **public** keyword makes it accessible outside the body of the class.

ENUM

Enum is a set of integer constants and similar to a struct it is also a value type entity. It is mainly used to declare a list of integers by using the “enum” keyword inside a namespace, class or even struct. In enum, we provide a name to each of the integer constants, so that we can refer them using their respective names.

Enum can have a fixed number of constants. It helps in improving safety and can also be traversed.

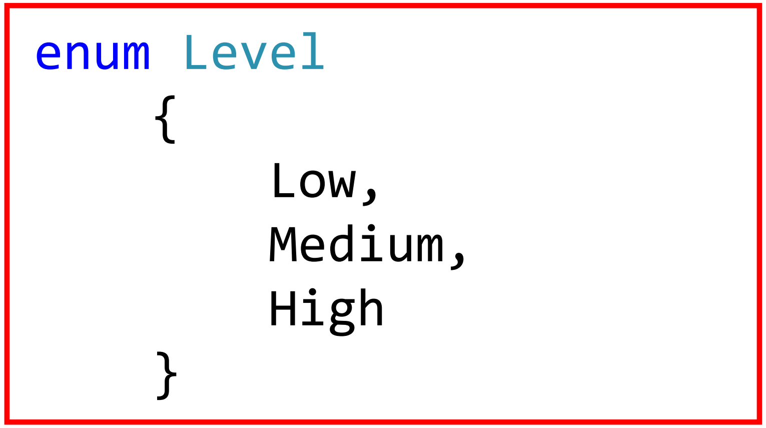
Enum is short for "enumerations", which means "specifically listed".

Features of Enum

- Enum improves the readability and maintainability of the code by providing meaningful names to the constants.
- Enum cannot be used with the string type constants.
- Enum can include constants such as int, long, short, byte, etc.
- By default, the value of enum constants starts with zero

Declaring an enum

The syntax for declaring enum is given below.



```
enum Level
    {
        Low,
        Medium,
        High
    }
```

All the enum constants have default values.

The value starts at 0 and moves its way up one by one.

Enum inside a Class

You can also have an enum inside a class:

```
using System;
program program7
{
    enum Level
    {
        Low,
        Medium,
        High
    }

   static void Main(string[] args)
        {
            Level myVar = Level.Medium;
            Console.WriteLine(myVar);
            Console.ReadKey();
        }
}
```

The output will be:
Medium

To get the integer value from an item, you must explicitly convert the item to an int:

```
using System;
program program8
{
    enum Months
    {
        January,       // 0
        February,      // 1
        March,         // 2
        April,         // 3
        May,           // 4
        June,          // 5
        July           // 6
    }

    static void Main(string[] args)
        {
            int myNum = (int)Months.April;
            Console.WriteLine(myNum);
            Console.ReadKey();
        }
    }
}
```

The output will be:

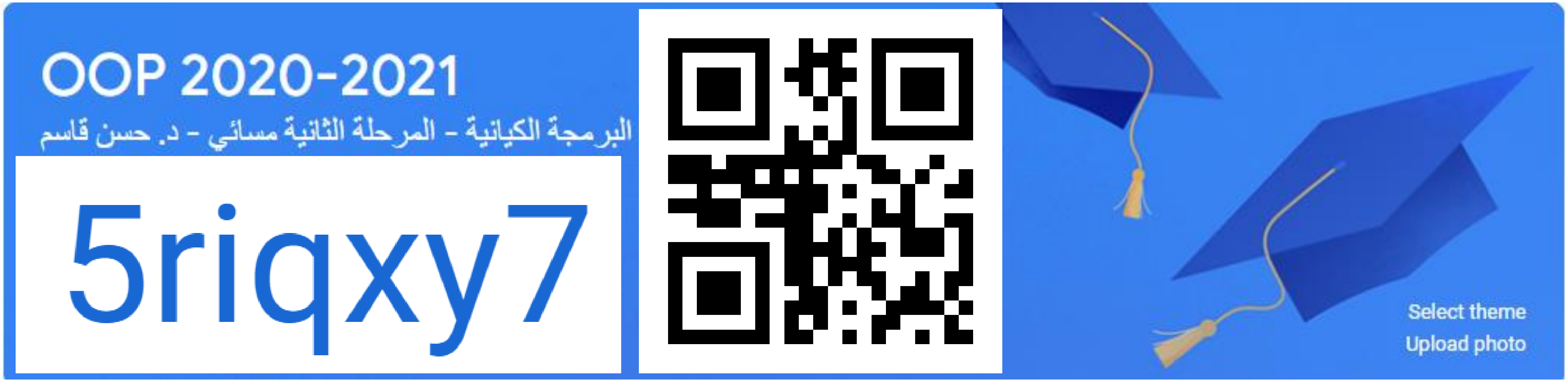
3

Why And When To Use Enums?

Use enums when you have values that you know aren't going to change, like month days, days, colors, etc.

QUESTION

Google Classroom :



OOP 2020-2021

البرمجة الكيانية - المرحلة الثانية مسائي - د. حسن قاسم

5riqxy7