Object Oriented Programming

Encapsulation 2

Static Class, Methods, Constructors, Fields

static means something which cannot be instantiated. You cannot create an object of a static class and cannot access static members using an object.

classes, variables, methods, properties, operators, events, and constructors can be defined as static using the static modifier keyword.

Static Class

Apply the static modifier before the class name and after the access modifier to make a class static. The following defines a static class with static fields and methods.

Example: C# Static Class

```
public static class Calculator
{
  private static int      resultStorage = 0;
  public  static string rtype = "Arithmetic";

  public  static int      Sum(int num1, int num2)
   {   return num1 + num2;
    }
  public static void Store(int result)
   {     resultStorage = result;
   }
}
```

the Calculator class is a static. All the members of it are also static.

You cannot create an object of the static class; therefore the members of the static class can be accessed directly using a class name like ClassName.MemberName, as shown below.

Accessing Static Members

```
class Program
{
  static void Main(string[] args)
  {
    var result = Calculator.Sum(10, 25); // calling static method
    Calculator.Store(result);
    var calcType = Calculator.Type;    // accessing static variable
    Calculator.Type = "Scientific";//assign value to static variable
  }
}
```

Rules for Static Class

- Static classes cannot be instantiated.
- All the members of a static class must be static; otherwise the compiler will give an error.
- A static class can contain static variables, static methods, static properties, static operators, static events, and static constructors.
- A static class cannot contain instance members and constructors.
- Indexers and destructors cannot be static
- var cannot be used to define static members. You must specify a type of member explicitly after the static keyword.
- Static classes are sealed class and therefore, cannot be inherited.
- A static class cannot inherit from other classes.
- Static class members can be accessed using ClassName.MemberName.
- A static class remains in memory for the lifetime of the application domain in which your program resides.

Static Members in Non-static Class

The normal class (non-static class) can contain one or more static methods, fields, properties, events and other non-static members.

Static Fields

Static fields in a non-static class can be defined using the static keyword.

Static fields of a non-static class is shared across all the instances. So, changes done by one instance would reflect in others.

Example: Shared Static Fields

```
public class StopWatch
{
  public static int InstanceCounter = 0;    // instance constructor
  public StopWatch()
   {
    }
}
class Program
 {
  static void Main(string[] args)
  {
   StopWatch sw1 = new StopWatch();
   StopWatch sw2 = new StopWatch();
   Console.WriteLine(StopWatch.NoOfInstances); //2
   StopWatch sw3 = new StopWatch();
   StopWatch sw4 = new StopWatch();
   Console.WriteLine(StopWatch.NoOfInstances);//4
   }
 }
```

Static Methods

You can define one or more static methods in a non-static class. Static methods can be called without creating an object. You cannot call static methods using an object of the non-static class. The static methods can only call other static methods and access static members. You cannot access non-static members of the class in the static methods.

```
class Program
{  static int counter = 0;
   string name = "Demo Program";
   static void Main(string[] args)
   {   counter++; // can access static fields
       Display("Hello World!"); // can call static methods
       name = "New Demo Program"; //Error: cannot access non-
static members
       SetRootFolder("C:\MyProgram");  //Error:  cannot  call  non-
static method
    }
static void Display(string text)
    {  Console.WriteLine(text);
    }
 public void SetRootFolder(string path)
  {    }
}
```

Rules for Static Methods

- Static methods can be defined using the static keyword before a return type and after an access modifier.
- Static methods can be overloaded but cannot be overridden.
- Static methods can contain local static variables.
- Static methods cannot access or call non-static variables unless they are explicitly passed as parameters.

Static Constructors

A non-static class can contain a parameterless static constructor. It can be defined with the static keyword and without access modifiers like public, private, and protected.

The following example demonstrates the difference between static constructor and instance constructor.

Example: Static Constructor vs Instance Constructor

```
public class StopWatch
 {    // static constructor
      static StopWatch()
      {  Console.WriteLine("Static constructor called");
      }    // instance constructor
 public StopWatch()
  {  Console.WriteLine("Instance constructor called");
  }    // static method
  public static void DisplayInfo()
  {     Console.WriteLine("DisplayInfo called");
   }    // instance method
  public void Start()
   { }    // instance method
  public void Stop()
   {  }
}
```

Above, the non-static class StopWatch contains a static constructor and also a non-static constructor.
The static constructor is called only once whenever the static method is used or creating an instance for the first time. The following example shows that the static constructor gets called when the static method called for the first time. Calling the static method second time onwards won't call a static constructor.

Example: Static Constructor Execution

```
StopWatch.DisplayInfo(); // static constructor called
hereStopWatch.DisplayInfo(); // none of the constructors called here
```

Output:

```
Static constructor called.
DisplayInfo called
DisplayInfo called
```

The following example shows that the static constructor gets called when you create an instance for the first time.

```
StopWatch sw1 = new StopWatch(); // First static constructor and then instance constructor called StopWatch sw2 = new StopWatch(); // only instance constructor called StopWatch.DisplayInfo();
```

Output:

```
Static constructor called
instance constructor called
instance constructor called
DisplayInfo called
```

Rules for Static Constructors

- The static constructor is defined using the static keyword and without using access modifiers public, private, or protected.
- A non-static class can contain one parameterless static constructor. Parameterized static constructors are not allowed.
- Static constructor will be executed only once in the lifetime. So, you cannot determine when it will get called in an application if a class is being used at multiple places.
- A static constructor can only access static members. It cannot contain or access instance members.

Static members are stored in a special area in the memory called High-Frequency Heap. Static members of non-static classes are shared across all the instances of the class. So, the changes done by one instance will be reflected in all the other instances.

Static Members In A Class

The members of the class can be declared static using the static keyword. When an object is declared as static, irrespective of the number of objects created there will be only one copy of the static object.

Being static implies that there will be a single instance of the member that will exist for a given class. It means that the value of the static function or variables inside the class can be invoked without creating an object for them.

Static variables are used for declaring constants as their values can be obtained directly by invoking the class rather than creating an instance of it.

```
1 public class Details
2  {
3          public static void stat(){
4          Console.WriteLine("Static method invoked");
5          }
6  }
7 public class Program
8 {
9           public static void Main(string[] args)
10         {
11                Details.stat();
12         }
13   }
```

The output of the following program will be:

Static method invoked

In the example, we have created a class “Details” that contains a static method “stat”. We have another class “Program” that contains the main method. In our previous topics, we saw, how we can initialize a class to access methods. But as we discussed, the static members of the class can be accessed with class object initialization.

Thus, in the main method, we have just invoked the method using the class directly without creating any object. The output of the program executed the code written inside the static method. In this case, we have printed a message to the console.

Static Class

A static class is similar to a normal class in C#. The static class can have only static members and it cannot be instantiated. A static class is used to make sure that the class is not instantiated. A static class is declared by using the keyword static before the keyword class during the declaration.

```
1 public static class Details
2 {
3  public static void multiply(int a, int b){
4             int c = a*b;
5                   Console.WriteLine("Multiplication result is: "+c);
6                 }
7   }
8 public class Program
9 {
10            public static void Main(string[] args)
11          {
12                    Details.multiply(2,8);
13          }
14  }
```

The output of the following program will be:

Multiplication result is: 16

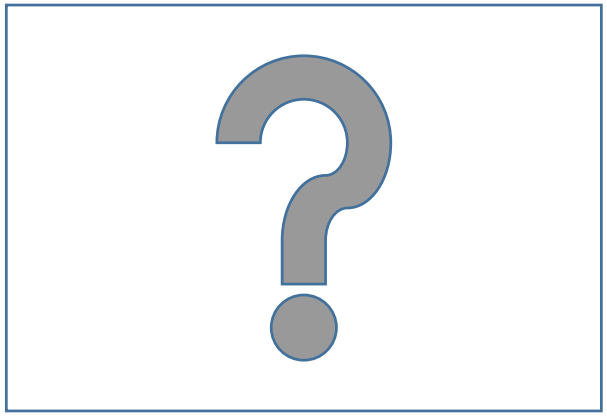
In the example, we have created a static class “Details” and inside the static class we have created another static method “multiply”. Inside the method, we have some code snippets that we want to execute. We also have another class “Program” with the main method.

Inside the main method, we have invoked the multiply method present inside the static class. If you look at our main method you will see that we have not initialized or created an object for the static class instead we have directly invoked the class from the main method.

Thus, when we directly invoke the multiply method using the class name and by providing parameters, it executes the code and prints the output.

Program static

QUESTION



Google Classroom :