CODSE232P-015

P.K. Ravindu Theja Rupasinghe

01.What is Static, Sealed and Abstract Class in C#

Static

This Static class contains only static members. It cannot be created as an object that meaning is It cannot be instantiated.

It is not allowed to create objects of the static class and since it does not allow to create objects it means it does not allow instance constructor.

Purpose – If we want to provide general purpose functions, we can use this Static class

Special Notes

This class cannot be inherited from, and they cannot inherit from any class.

This static class does not have object (Instance Constructors) but they have only static constructor

All methods , attributes (Properties) must be static

Example

```
Public static class Student
{
}
```

Sealed

This sealed class is used for restrict the users from inheriting the class. That meaning is this class cannot be inherited. When we create a class as a sealed, other class cannot derive from it. Actually, this is a Security feature. This sealed keyword tells to the Compiler this class is Sealed and therefore cannot be extended.

example

```
public sealed class Student
{
}
```

Abstract Class

Abstract class is a restricted class that meaning is when we declare a class as a abstract class that class cannot be used to create objects. If we want to access this class this class must be inherited from another class.

Note – Abstract Method can only be used in a abstract class. And it does not have a code block because of that the body is provided by the derived class

02.Inheritence

Inheritance is a fundamental concept that allows a class to inherit properties and methods from another class. When we use inheritance we can reduce our code Lines. Because of that we can inherit our declared variables and methods for the another class which can perfume same class.

There are Base class and Derived class or we can say Perent class and Child class.

In OOP Concept inheritance there are few inheritance types.

Those are,

- Single Inheritance
- Multiple Inheritance (Through interfaces)
- Multilevel inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

Let's talk a about one by one

Single inheritance

The child class inherit from only one single Parent class.(That meaning is A derived class inherits from a single base class.)

```
public class Animal // (This is a )
{
   public void Eat()
   {
      Console.WriteLine("Eating...");
   }
}

public class Dog : Animal  //(This is a Child class)
{
   public void Bark()
   {
      Console.WriteLine("Barking...");
   }
}
```

Note – in C# we use **:** for inherit class

Multiple Inheritance (Using interface)

If we want to inherit more than one class, we should have to use a interface, After you declare the interface you can inherit to the necessary classes. But there is one important thing. If we define something in the interface we definitely call and mention it on inherited class.

```
public interface IAnimal
{
    void Eat();
}

public interface IPet
{
    void Play();
}

public class Dog : IAnimal, IPet
{
    public void Eat()
    {
        Console.WriteLine("Eating...");
    }

    public void Play()
    {
        Console.WriteLine("Playing...");
    }
}
```

Multilevel Inheritnce

A class inherits from a base class, and then another class inherits from that derived class, like a chain.

This is a example for that

```
public class Animal  //This is a Parent class
{
  public void Eat()
  {
    Console.WriteLine("Eating...");
  }
}

public class Mammal : Animal   //Mamal class inherited from Animal
{
  public void Walk()
  {
    Console.WriteLine("Walking...");
  }
}

public class Dog : Mammal // Dog class inherited from Mammal Class
{
  public void Bark()
  {
    Console.WriteLine("Barking...");
  }
}
```

Hierarchical Inheritance

This meaning is there is only one Parent class and Many Childs class. Those child class inherited from only one parent class. (Multiple classes inherit from a single base class)

```
public class Animal // Animal is a Base class
{
   public void Eat()
   {
      Console.WriteLine("Eating...");
   }
}

public class Dog : Animal // Dog class is derived  class it inherited from Animal
{
   public void Bark()
   {
      Console.WriteLine("Barking...");
   }
}

public class Cat : Animal  //cat is also derived class
{
   public void Meow()
   {
      Console.WriteLine("Meowing...");
   }
}
```

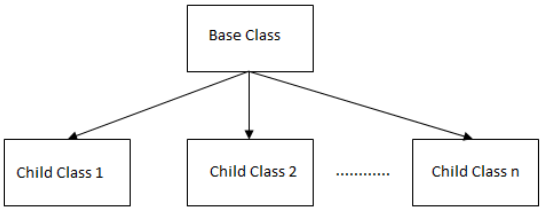

Fig: Hierarchical Inheritance

Hybrid Inheritance

This meaning is there is combinations of two or more types of inheritance. That meaning is there can be Multilevel inheritance , multiple inheritance like wise.

03. Interfaces

Actually if we want to inherit attribute and methods in to a another class we can use single inheritance. But after if we want to again inherit attribute and methods to the same class we cannot inherit like below

```
class Student
{
        String name;
        int age;

        public void getStudentDetails ()
        {
                Console.Write("Enter Your Name : ");
                Name = Console.ReadLine();
                Console.Write("Enter Your age");
                age = Convert.ToInt32(Console.ReadLine());
        }
}
```

```
Class Subject
{
        double subjectFee;

        public void getSubjectFee()
        {
                // Code Block
        }
}

Class MathStudent : Student  :  Subject                  // Note – You cannot inherit like this , This example should correct like below
{
        String mathStudentID;

        Public void getMathsID()
        {
                Console.Write("Enter Your Maths ID : ");
                mathStudentID = Console.ReadLine();
        }

}
```

This is a Correct way

```
class Student
{
        String name;
        int age;

        public void getStudentDetails ()
        {
                Console.Write("Enter Your Name : ");
                Name = Console.ReadLine();
                Console.Write("Enter Your age");
                age = Convert.ToInt32(Console.ReadLine());
        }
}

interface Subject       //➔ class subject should change to interface Subject
{
        double subjectFee;

        void getSubjectFee();           //➔ Do not use access modifier

}
Class MathStudent : Student  :  Subject                  //➔ use comma instead foer :  ➔ like this Class MathStudent : Student  ,  Subject
{
        String mathStudentID;
        Double subjectFee;              // ➔Implement This one also

        Public void getMathsID()
        {
                Console.Write("Enter Your Maths ID : ");
                mathStudentID = Console.ReadLine();
        }

        Public void getSubjectFee()             //➔Implement interface methods with the access modifier
        {
                Console.Write("Enter Subject Fees :")
                subjectFee = Convert.ToDouble(Console.ReadLine());
        }

}
```

An interface is a completely "abstract class", which can only contain abstract methods and properties (with empty bodies)

To access the interface methods, the interface must be "implemented" (kind like inherited) by another class. To implement an interface, use the : symbol (just like with inheritance). The body of the interface method is provided by the "implement" class. Note that you do not have to use the override keyword when implementing an interface:

As an example

```
// Interface
interface IAnimal
{
  void animalSound(); // interface method (does not have a body)
}

// Pig "implements" the IAnimal interface
class Pig : IAnimal
{
  public void animalSound()
  {
    // The body of animalSound() is provided here
    Console.WriteLine("The pig says: wee wee");
  }
}

class Program
{
  static void Main(string[] args)
  {
    Pig myPig = new Pig();  // Create a Pig object
    myPig.animalSound();
  }
}
```

04. Polymorphism

Actually polymorphism meaning is many Forms. And it occurs when we have many classes that are related with each other by inheritance , we can able to override the inherited methods, and there are two types of that , those are Method Overloading (In Compile Time) , and Method Overriding (In Runtime)

in polymorphism there are two types

i. Static Polymorphism
   a. Function Overloading
   b. Operator Overloading

ii. Dynamic Polymorphism

As a Short note

Types of Polymorphism

01. Compile-time Polymorphism (**Method Overloading and Operator Overloading**)
    a. Method Overloading ➔
       Multiple methods in the same class have the same name but different parameters.

    b. Operator Overloading ➔
       Customizing the behavior of an operator (like +, -, etc.) for user-defined types.

02. Run-time Polymorphism (**Method Overriding**)

    Achieved through inheritance and interfaces. It allows a method to do different things based on the object it is acting upon

Let's talk a about one by one

Method Overloading

Method overloading allows us a class to have multiple methods with the same name, but with different signatures. (**That meaning is different parameters, but method name is same**)

As an example

```
public class MathOperations
{
   public int Add(int a, int b)                // Add is a method name. in C# we can use Uppercase for Starting a method name, but java cannot
   {                                           // do that. In java method name should start from Lowercase
      return a + b;
   }

   public double Add(double a, double b)       // Same Method name → Add
   {
      return a + b;
   }
}
```

Operator Overloading

Operator overloading gives us the ability to use the same operator to do various operations. This is also a OOP Concept of overloading for the functions.

Dynamic Polymorphism

C# allows us to create abstract classes that are used to provide partial class implementation of an interface. Implementation is completed when a derived class inherits from it. Abstract classes contain abstract methods, which are implemented by the derived class. (Derived class meaning is Parent Class)

Actually this is also known as runtime polymorphism. That is resolved during runtime rather than at compile time

Benefits of Dynamic Polymorphism

Flexibility

Extensibility

Maintainability

As an example

```
class Animal                    //This is a Derrived class
{
    public virtual void MakeSound()
  {
    Console.WriteLine("Some generic animal sound");
  }
}
class Dog : Animal              //Dog Class inherited from Animal Parent Class
{
  public override void MakeSound()     //This method inherited from the Animal Class. But here it's Method Overiding
  {
    Console.WriteLine("Barking");
  }
}

class Cat : Animal
{
  public override void MakeSound()
  {
    Console.WriteLine("Meowing");
  }
}
```

05.File Handling

In .NET Framework provide System.IO library for handle the files. In here we can able to handle File Creating , reading , writing and Deleting , Copying and Moving , Opening.

A file is a collection of data stored in a disk with a specific name and a directory path. When a file is opened for reading or writing, it becomes a stream.

Let's look how to Write to a File and Read It

In the following example, we use the WriteAllText() method to create a file named "filename.txt" and write some content to it. Then we use the ReadAllText() method to read the contents of the file

```
using System;
using System.IO;     //if we use file handling we should import it to the namespace. Library nema is  System.IO

namespace MyApplication
{
 class Program
 {
   static void Main(string[] args)
   {
     string writeText = "Hello World!";
     File.WriteAllText("filename.txt", writeText);                    // Create a file and write the contents of writeText to it

     string readText = File.ReadAllText("filename.txt");              // Read the contents of the file
     Console.WriteLine(readText); // Output the content
   }
 }
}
```

06.Exception Handling

Actually this Exception Handling use for manages the runtime errors.

When executing C# code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things.

When an error occurs, C# will normally stop and generate an error message. The technical term for this is: C# will throw an exception (throw an error).

Ensuring that the program can handle unexpected conditions gracefully without crashing.

Thera are some key words for do this.

i. Try Block
ii. Catch Block
iii. Finally Block
iv. Throw Keyword

Let's talk a about one by one

01. Try Block

- The try statement allows you to define a block of code to be tested for errors while it is being executed.

- The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

- The try and catch keywords come in pairs:

This is a Syntax for Try and Catch'

```
try
{
  // Block of code to try
}
catch (Exception e)
{
  // Block of code to handle errors
}
```

This is an example for that

```
Try
{
        int[] myNumbers = {1, 2, 3};
        Console.WriteLine(myNumbers[10]);     // This will generate an error, because myNumbers[10] does not exist.
}
catch (Exception e)
{
  Console.WriteLine(e.Message);          // Catch the Error and we can show a Error Message for the user
}
```

02. Finally Block

Finally block is also a part of the exception handling mechanism. This is use for execute a block of code regardless of whether an exception is thrown or not

the purpose of the finally block is to provide a place to clean up any resources that are no longer needed, such as closing files, releasing network connections, or disposing of objects. This provide guaranteed Execution , Resource Clean up and Optional block

as an example

```
try
{
  int[] myNumbers = {1, 2, 3};
  Console.WriteLine(myNumbers[10]);
}
catch (Exception e)
{
  Console.WriteLine("Something went wrong.");
}
finally
{
  Console.WriteLine("The 'try catch' is finished.");
}
```

03.Throw Block

This throw block is use for create customs error.

The throw statement is used together with an exception class. There are many exception classes available in
C#: ArithmeticException, FileNotFoundException, IndexOutOfRangeException, TimeOutException, etc:

```
using System;

namespace MyApplication
{
  class Program
  {
    static void checkAge(int age)
    {
      if (age < 18)
      {
        throw new ArithmeticException("Access denied - You must be at least 18 years old.");
      }
      else
      {
        Console.WriteLine("Access granted - You are old enough!");
      }
    }

    static void Main(string[] args)
    {
      checkAge(20);
    }
  }
}
```

Throw keyword can be used in two primary contexts.

Throwing an Exception: This is used to create and throw a new exception.

Rethrowing an Exception: This is used to rethrow an exception that was caught in a catch block, preserving the original stack trace.

07. Delegates

In C#, a delegate is a type that represents references to methods with a specific parameter list and return type. Delegates are used to pass methods as arguments to other methods, define callback methods, and implement event handling.

Delegate are used to pass methods as arguments to other methods

The most important thing is Delegates allow methods to be passed as a Parameter. And also delegate can be used to define callback methods.

Delegates can be chained together as an example multiple methods can be called on a Single event.