**Static class**

* A static class can have only static member and static methods.
* Static class can have only one static constructor.
* We cannot have instance constructor in static class.

**Static Constructor**

* In a class we can have only one static constructor.
* Static constructor must be parameter less.
* Access modifiers are not allowed to static constructor.

**Sealed class**

* We cannot inherit from the sealed class.
* A sealed class can have multiple constructor.
* We can create object of sealed class.

**Sealed class Constructor**

* Sealed class constructors can be parameterized or parameter less.

**Basic:**

Pillers of OOPs (E-API)

1. Encapsulation
2. Abstraction
3. Polymorphism
4. Inheritance

- By default a class is private, also we cannot explicitly define a class as private directly in namespace it can be a subclass and private class cannot be inherited.

- By default a method is private, private method can be used only inside the same class.

- If a class has private constructor, it cannot be inherited.

**Overloading:**

* Deals with the methods inside the same class.
* Method with Same Name but Different type or number of arguments.
* Overloading has nothing to do with the return type of method.

**Overriding:**

* Deals with the methods in parent and child class.
* Method in base class is virtual and in child class it is decorated with override keyword.
* Methods must have same signature and return type in both parent and child classes.

public class AA

{

public virtual void GetName()

{

Console.WriteLine("Hello Base");

}

public virtual void GetName(string name) // Overloading GetName()

{

Console.WriteLine("From Base :" + name);

}

}

public class BB : AA

{

//Overriding the base class method GetName()

public override void GetName() //public new void GetName()

{

Console.WriteLine("Hello Child");

}

}

public class mainClass

{

static void Main(string[] args)

{

AA ob = new AA();

ob.GetName();

AA obj1 = new BB();

obj1.GetName();

BB obj = new BB();

obj.GetName();

//BB obj2 = new AA(); //Child class cannot have the object of parent class.

Console.ReadLine();

}

}

**Output :**

1. **If GetName method in BB is decorated with override**

**Hello Base**

**Hello Child**

**Hello Child**

1. **If GetName method in BB is decorated with new**

**Hello Base**

**Hello Base**

**Hello Child**

**Static Constructor**

In c#, Static Constructor **is useful to perform a particular action only once throughout the application**. If we declare a constructor as static, then **it will be invoked only once** irrespective of the number of class instances and **it will be called automatically before the first instance is created**.

Generally, **in c# the static constructor will not accept any access modifiers and parameters**. In simple words, we can say it’s parameterless.

The following are the properties of static constructor in c# programming language.

* Static constructor in c# won’t accept any parameters and access modifiers.
* The static constructor will invoke automatically, whenever we create the first instance of a class.
* The static constructor will be invoked by CLR so we don’t have a control on static constructor execution order in c#.
* In c#, only one static constructor is allowed to create.

using System;

namespace Tutlane

{

    class User

    {

        // Static Constructor

        static User()

        {

            Console.WriteLine("I am Static Constructor");

        }

        // Default Constructor

        public User()

        {

            Console.WriteLine("I am Default Constructor");

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            // Both Static and Default constructors will invoke for first instance

            User user = new User();

            // Only Default constructor will invoke

            User user1 = new User();

            Console.WriteLine("\nPress Enter Key to Exit..");

            Console.ReadLine();

        }

    }

}

**Output:**

I am Static Constructor

I am Default Constructor

I am Default Constructor

Press Enter Key to Exit.

**Abstract Class:**

* It is best to use when we have some common methods/functionality for the child classes.
* Put all the common functionality in simple methods and all the methods whose implementation is different but name is same, make them Abstract method.
* An abstract class should be a base class.
* It prevents user to create object of base class.
* Can have abstract method and concrete methods.
  1. These methods can be public, protected or private.
  2. Private methods in abstract class can be called only from abstract class constructor.
  3. Public void method of abstract class can only be used by inherited class while public void method of normal class can be access directly by creating class object.
* Can have constructor, but cannot create object of abstract class.
  1. Abstract class constructor can be invoked from the child class.

**Interface:**

* It is bet to use when we have only unique methods/functionality for the child classes.
* Provides the feature of multiple inheritance and avoids the Diamond problem that we have in c++.
* All members are by default public, static and final.
* The class which implements the interface will be responsible for implementing all of it’s methods.
* Cannot create constructor of interface, so we cannot create object of it.