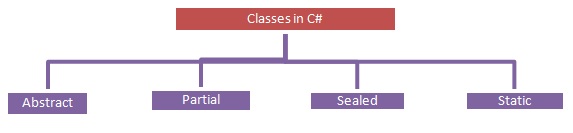
**2. Advance C#**

**2.1 Types class**

* Types of classes in c# :



* Abstract Class :
  + An abstract class is a class that provides a common definition to the subclasses, and this is the type of class whose object is not created.
  + We cannot create an object of an abstract class.
  + It must be inherited in a subclass if you want to use it.
  + An Abstract class contains both abstract and non-abstract methods.
  + The methods inside the abstract class can either have an or no implementation.
  + We can inherit two abstract classes; in this case, implementation of the base class method is optional.
  + An Abstract class has only one subclass.
  + Methods inside the abstract class cannot be private.
  + If there is at least one method abstract in a class, then the class must be abstract.
  + For example

abstract class Accounts

{

}

* Partial Class :
  + It is a type of class that allows dividing their properties, methods, and events into multiple source files, and at compile time, these files are combined into a single class.
  + All the parts of the partial class must be prefixed with the partial keyword.
  + If you seal a specific part of a partial class, the entire class is sealed, the same as for an abstract class.
  + Inheritance cannot be applied to partial classes.
  + The classes written in two class files are combined at run time.
  + For example

partial class Accounts

{

}

* Sealed Class :
  + A Sealed class is a class that cannot be inherited and used to restrict the properties.
  + A Sealed class is created using the sealed keyword.
  + Access modifiers are not applied to a sealed class.
  + To access the sealed members, we must create an object of the class.
  + For example

sealed class Accounts

{

}

* Static Class :
  + It is the type of class that cannot be instantiated. In other words, we cannot create an object of that class using the new keyword, such that class members can be called directly using their name.
  + It was created using the static keyword.
  + Only static members are allowed; in other words, everything inside the class must be static.
  + We cannot create an object of the static class.
  + A Static class cannot be inherited.
  + It allows only a static constructor to be declared.
  + The static class methods can be called using the class name without creating the instance.
  + For example

static class Accounts

{

}

Bottom of Form

**2.2 Generics**

* Generics in C# provide a way to create classes, interfaces, and methods with placeholders for the data types they work with. This allows you to write code that can work with any data type, providing flexibility and type safety.
* Generics are extensively used in collections (such as List, Dictionary, etc.) and other scenarios where a common functionality is needed for different types.

Generic Classes:

* Syntax :

public class MyClass<T>

{

private T value;

public MyClass(T val)

{

value = val;

}

public T GetValue()

{

return value;

}

}

Generic Methods :

* You can create generic methods inside non-generic classes as follows:

public class MyUtility

{

public T Add<T>(T a, T b)

{

dynamic dynamicA = a;

dynamic dynamicB = b;

return dynamicA + dynamicB;

}

}

Generic Interfaces :

* Syntax :

public interface IRepository<T>

{

void Add(T item);

T GetById(int id);

}

Constraints :

* You can use constraints to specify requirements on the generic type:

public class MyClass<T> where T : IComparable { // Code here }

* This example ensures that T must implement the IComparable interface.

Covariance and Contravariance :

* Generics in C# support covariance and contravariance, allowing more flexibility when working with generic types. This is achieved using the out and in keywords.
* Syntax :

// Covariant interface

public interface IMyCovariant<out T>

{

T GetItem();

}

// Contravariant interface

public interface IMyContravariant<in T>

{

void SetItem(T item);

}

**2.3 File system in Depth**