C#, provide full support for object-oriented programming including encapsulation, inheritance, and polymorphism.

-------------------------------------------------------------------------------------------------------------

Encapsulation means that a group of related properties, methods, and other members are treated as a single unit or object.

Encapsulation is implemented by using access specifiers. An access specifier defines the scope and visibility of a class member. C# supports the following access specifiers:

Public

Private

Protected

Internal

Protected internal

-----------------------------------------------------------------------------------------------------------

Inheritance describes the ability to create new classes based on an existing class.

------------------------------------------------------------------------------------------------------------

Polymorphism means that you can have multiple classes that can be used interchangeably, even though each class implements the same properties or methods in different ways.

In polymorphism we will declare methods with same name and different parameters in same class or methods with same name and same parameters in different classes. Polymorphism has ability to provide different implementation of methods that are implemented with same name.

In Polymorphism we have 2 different types those are

- Compile Time Polymorphism (Called as Early Binding or Overloading or static binding)overloading.

- Run Time Polymorphism (Called as Late Binding or Overriding or dynamic binding)override”

--------------------------------------------------------------------------------------------------------------

In .NET, there are two categories of types, reference types and value types.

Different Between Structs are value types and classes are reference types.

The general difference is that a reference type lives on the heap, and a value type lives inline, that is, wherever it is your variable or field is defined.

A variable containing a value type contains the entire value type value. For a struct, that means that the variable contains the entire struct, with all its fields.

A variable containing a reference type contains a pointer, or a reference to somewhere else in memory where the actual value resides.

Classes Only:

Can support inheritance

Are reference (pointer) types

The reference can be null

Have memory overhead per new instance

Structs Only:

Cannot support inheritance

Are value types

Are passed by value (like integers)

Cannot have a null reference (unless Nullable is used)

Do not have a memory overhead per new instance - unless 'boxed'

-------------------------------------------------------------------------------------------------------------

Constructors

Constructors are class methods that are executed automatically when an object of a given type is created. A constructor can run only once when a class is created.

Destructors

Destructors are used to destruct instances of classes. In the .NET Framework, the garbage collector automatically manages the allocation and release of memory for the managed objects in your application. However, you may still need destructors to clean up any unmanaged resources that your application creates. There can be only one destructor for a class.

-----------------------------------------------------------------------------------------------------------------

Events

Events enable a class or object to notify other classes or objects when something of interest occurs. The class that sends (or raises) the event is called the publisher and the classes that receive (or handle) the event are called subscribers.

------------------------------------------------------------------------------------------------------------------

Delegates

A delegate is a type that holds a reference to a method. A delegate is declared with a signature that shows the return type and parameters for the methods it references, and can hold references only to methods that match its signature. A delegate is thus equivalent to a type-safe function pointer or a callback. A delegate declaration is sufficient to define a delegate class.

------------------------------------------------------------------------------------------------------------------

Access Modifiers and Access Levels

public

The type or member can be accessed by any other code in the same assembly or another assembly that references it.

private

The type or member can only be accessed by code in the same class.

protected

The type or member can only be accessed by code in the same class or in a derived class.

internal

The type or member can be accessed by any code in the same assembly, but not from another assembly.

protected internal

The type or member can be accessed by any code in the same assembly, or by any derived class in another assembly.

-----------------------------------------------------------------------------------------------------------------------------

Static Class

To access the static (shared) member, use the name of the class without creating an object of this class:

Static classes in C# and modules in Visual Basic have static (shared) members only and cannot be instantiated. Static (shared) members also cannot access non-static (non-shared) properties, fields or methods  
-------------------------------------------------------------------------------------------------------------------

Anonymous Types

Anonymous types enable you to create objects without writing a class definition for the data type. Instead, the compiler generates a class for you. The class has no usable name and contains the properties you specify in declaring the object. [ eg Var ]

-----------------------------------------------------------------------------------------------------------------

Inheritance

Inheritance enables you to create a new class that reuses, extends, and modifies the behavior that is defined in another class. The class whose members are inherited is called the base class, and the class that inherits those members is called the derived class. However, all classes in both C# and Visual Basic implicitly inherit from the Object class that supports .NET class hierarchy and provides low-level services to all classes.

-------------------------------------------------------------------------------------------------------------------

To specify that a class cannot be used as a base class:

public sealed class A { }

To specify that a class can be used as a base class only and cannot be instantiated:

public abstract class B { }

-------------------------------------------------------------------------------------------------------------------

is it possible to create virtual static method in my base class.

is it possible to create Access modifier for my Static Constructor

by Default Class is public and Constructor in Private

------------------------------------------------------------------------------------------------------------------------

overriding

virtual --> override

abstract --> override

Data Hiding using new key word

public class BaseC

{

public int x;

public void Invoke() { }

}

public class DerivedC : BaseC

{

new public void Invoke() { }

}

-----------------------------------------------------------------------------------------------------------------------------------

Interfaces

Interfaces, like classes, define a set of properties, methods, and events. But unlike classes, interfaces do not provide implementation. They are implemented by classes, and defined as separate entities from classes. An interface represents a contract, in that a class that implements an interface must implement every aspect of that interface exactly as it is defined.

Interface have no static and no Access Modifier

------------------------------------------------------------------------------------------------------------------------

Generics

Public class SampleGeneric<T>

{

public T Field;

}

To create an instance of a generic class:

SampleGeneric<string> sampleObject = new SampleGeneric<string>();

sampleObject.Field = "Sample string";

-------------------------------------------------------------------------------------------------------------------------------------

Method Overloading

Reflection (Type)

use reflection to dynamically create an instance of a type, bind the type to an existing object, or get the type from an existing object and invoke its methods or access its fields and properties.

Func

Action

try--> catch-->finally

using call IDispose after this function

--------------------------------------------------------------------------------------------------------------

What is the difference between arrays and collection?

Array:

You need to specify the size of an array at the time of its declaration. It cannot be resized dynamically.

The members of an array should be of the same data type.

Collection:

The size of a collection can be adjusted dynamically, as per the user's requirement. It does not have fixed size.

Collection can have elements of different types.

-----------------------------------------------------------------------------------------------------------------------

enum , Hashtable ,destructor,

IDispose, GarbageCollector,ArrayList