OOP- oop is a programming approach which provides solutions to problems with the help of algorithms based on the real world. It uses a real world approach to solve problems. So object oriented techniques offer a better and easy way to write programs.

Source- https://dotnettutorials.net/lesson/abstraction-csharp-realtime-example/
Delegate - is reference type entity that is used to store reference of method. So whenever we need to call a method, we can easily use the delegate by initializing pointer to that method, thereby helping to implement encapsulation.

Abstraction- is a process of defining a class by providing the necessary details to call the object operations (i.e., methods) by hiding its implementation details. It means we need to expose what is necessary and compulsory, and we need to hide unnecessary things from the outside world.
Abstraction lets you focus more on what an object does rather than how it does. That means what are the services available as part of the class that we need to expose, but how are the services implemented that we need to hide.

To take a real-time example, when we log in to any social networking site like Facebook, Twitter, LinkedIn, etc., we enter our user ID and password, and then we get logged in. Here, we don't know how they are processing the data or what logic or algorithm they are using for login. This information is abstracted/hidden from us since they are not essential to us. This is basically what abstraction is.

We can implement the abstraction OOPs principle in two ways. They are as follows:

1- Using Interface 2- Using Abstract Classes and Abstract Methods

Both interface and abstract classes and abstract methods provide some mechanism to hide the implementation details by only exposing the services. The user only knows what services or methods are available, but the user will not know how these services or methods are implemented.

Note- Using abstract class, we can achieve 0 to 100% abstraction. The reason is that you can also provide implementation to the methods inside the abstract class. It does not matter whether you implement all methods or none of the methods inside the abstract class. This is allowed, which is not possible with an interface.

Source-
https://dotnettutorials.net/lesson/abstract-class-and-abstract-methods-interview-questions-in-csharp/

Abstract Methods are methods declared/initialized within an abstract class or an interface that do not have a method body or implementation in the declaring class or interface. Instead, the responsibility for implementing the method is delegated to any concrete (non-abstract) class that derives from the abstract class or implements the interface.

A method without the body is known as the Abstract Method. What the method contains is only the declaration of the method. That means the abstract method contains only the declaration, no implementation. You should explicitly use the abstract modifier as follows. And once you use the abstract modifier, automatically, the method will be called an abstract method. If we end the method with a semicolon as follows, it is called an Abstract Method.

public abstract void Add(int num1, int num2);

If a method is declared abstract under any class, then the child class of that abstract class is responsible for implementing the abstract method without fail. Abstract class imposes some restrictions on the Child classes. And children or Child classes have to be followed or fulfill those restrictions.

Note: Every abstract method declared within an abstract class must and should be implemented by the Child classes without fail; otherwise, we will get a compile-time error.

Abstract class is a class that serves as a blueprint for other classes. Abstract classes cannot be instantiated directly, but they can be used as base classes for other classes that derive from them. Abstract classes are declared using the abstract keyword. They often define a common set of characteristics or behaviors that should be shared among multiple derived classes. **We need to define the abstract method inside an abstract class only.** abstract class constructor is used to initialize fields of the abstract class.

Abstract class contains both abstract and non-abstract methods.

Why Abstract Class Cannot Be Instantiated in C#? Its abstract methods cannot be executed because it is not a fully implemented class. But we can create a reference for the abstract class. But we cannot create an instance of an abstract class.

We must Implement the abstract methods using the override modifier as follows.

```
//Creating Child class instance

AbsChild absChild = new AbsChild();

 //Creating abstract class reference pointing to child class object

 AbsParent absParent = absChild;
```

Who will Provide the Implementation of Abstract Methods? The Answer is Child Class. If you have a child class of an abstract class, then it is the responsibility of the child class to

provide the implementation for all the abstract methods of the parent class. You cannot escape. Every method should be implemented. If you implement all the abstract methods, you can only consume the non-abstract method of the Parent class. So, the point that you need to remember is that in the child class, you need to implement each and every abstract method of the parent class, and then only you can consume the non-abstract methods of the parent class.

When to use Abstract classes and Methods

Ensure Code Consistency, Enable Code Reusability,Provide Default Implementations

Encapsulation vs Abstraction

1- The Encapsulation Principle is all about data hiding (or information hiding). On the other hand, the Abstraction Principle is all about detailed hiding (implementation hiding).

2- Using the Encapsulation principle, we can protect our data, i.e., from outside the class, nobody can access the data directly. We are exposing the data through publicly exposed methods and properties. The advantage is that we can validate the data before storing and returning it. On the other hand, using the Abstraction Principle, we are exposing only the services so that the user can consume the services, but how the services/methods are implemented is hidden from the user. The user will never know how the method is implemented.

3- With the Encapsulation Principle, we group data members and member functions into a single unit called class, interface, enum, etc. On the other hand, with the Abstraction Principle, we are exposing the interface or abstract class to the user and hiding implementation details, i.e., hiding the child class information.

4- We can implement Encapsulation by declaring the data members as private and exposing the data members only through publicly exposed methods and properties with proper validation. On the other hand, we can implement abstraction through abstract classes and interfaces.

5- Abstraction in C# is used to hide unwanted data and shows only the required properties and methods to the user. Encapsulation in C# is used to bind data members and member functions into a single unit to prevent outsiders from accessing it directly.

Advantages of Abstraction Principle in C#

The Abstraction Principle reduces the complexity of viewing things. It only provides the method signature by hiding how the method is actually implemented.

The Abstraction Principle helps to increase the security of an application or program as we are only providing the necessary details to call the method by hiding how the methods are actually implemented.

With the Abstraction Principle, the enhancement will become very easy because without affecting end-users, we are able to perform any type of changes in our internal system.

Without the Abstraction principle, maintaining application code is very complex. Abstraction gives one structure to program code.

Encapsulation- The process of defining a class by hiding its internal data members from outside the class and accessing those internal data members only through publicly exposed methods (setter and getter methods) or properties with proper validations is called Encapsulation.

The process of binding or grouping the **State (i.e., Data Members)** and **Behaviour (i.e., Member Functions)** together into a single unit (i.e., **class, interface, struct, etc.**) is called Encapsulation in C#. The Encapsulation Principle ensures that the state and behavior of a unit (i.e., **class, interface, struct, etc.**) cannot be accessed directly from other units (i.e., **class, interface, struct, etc.**).

Data Encapsulation is implemented.

1. By declaring the variables as private (to restrict their direct access from outside the class)
2. By defining one pair of public setter and getter methods or properties to access private variables from outside the class.

The biggest advantage of Encapsulation is Data Hiding.

Real-World Example of Encapsulation: The Encapsulation Principle in C# is very similar to a Capsule. As a capsule binds its medicine within it, in the same way in C#, the Encapsulation Principle binds the State (Variables) and Behaviour (Methods) into a single unit called class, enum, interface, etc. So, you can think of Encapsulation as a cover or layer that binds related states and behavior together in a single unit.

Data hiding or Information Hiding is a Process in which we hide internal data from outside the world. The purpose of data hiding is to protect the data from misuse by the outside world. Data hiding is also known as Data Encapsulation. Without the Encapsulation Principle, we cannot achieve data hiding.

Data Encapsulation or Data Hiding is implemented by using the **Access Specifiers**. An access specifier defines the scope and visibility of the class member, and we have already discussed

the different types of Access Specifiers Supported in C# in our previous article. C# supports the following six access specifiers:

1- **public**: The public members can be accessed by any other code in the same assembly or another assembly that references it.
2- **private**: The private members can be accessed only by code in the same class.
3- **protected**: The protected Members are available within the same class as well as to the classes that are derived from that class.
4- **internal**: The internal members can be accessed by any code in the same assembly but not from another assembly.
5- **protected internal**: The protected internal members can be accessed by any code in the assembly in which it's declared or from within a derived class in another assembly.
6- **private protected:** The private protected members can be accessed by types derived from the class that is declared within its containing assembly.

Advantages of Providing Variable Access via Setter and Getter Methods - we can validate the user-given data before storing the value in the variable

Advantages of Encapsulation in C#:

1- **Data protection**: You can validate the data before storing it in the variable.
2- **Achieving Data Hiding**: The user will have no idea about the inner implementation of the class.
3- **Security**: The encapsulation Principle helps to secure our code since it ensures that other units(classes, interfaces, etc) can not access the data directly.
4- **Flexibility**: The encapsulation Principle in C# makes our code more flexible, allowing the programmer to easily change or update the code.
5- **Control:** The encapsulation Principle gives more control over the data stored in the variables. For example, we can control the data by validating whether the data is good enough to store in the variable.

Inheritance- It is a way to define the relationship between two classes. Inheritance defines parent and child relationship between 2 classes.

The process of creating a new class from an existing class such that the new class acquires all the properties and behaviors of the existing class is called inheritance. The properties (or behaviors) are transferred from which class is called the superclass or parent class or base class whereas the class which derives the properties or behaviors from the superclass is known as a subclass or child class or derived class.

Inheritance is the concept that is used for code reusability and changeability purposes.

IS-A relationship is implemented using Inheritance and the **HAS-A relationship** is implemented using Composition i.e. declaring a variable. So, whenever we declare a variable of one class inside another class, we call it a Composition or HAS-A relationship.

HAS-A relationship (aggregation) is a process in which one class defines another class as any entity reference. It is another way to reuse the class. It is a form of association that represents the HAS-A relationship.

Polymorphism *provides the ability for a class to have multiple implementations with the same name.* and it occurs when we have many classes that are related to each other by inheritance. Polymorphism uses those methods to perform different tasks. This allows us **to perform a single action in different ways.**
There are two types of polymorphism

1. Static polymorphism/compile-time polymorphism/Early binding (CSE-overload)
2. Dynamic polymorphism/Run time polymorphism/Late binding (DRL-override)

Compile-Time Polymorphism- The function call bound to the class at the time of compilation, if the function is going to be executed from the same bounded class at run-time, then it is called Compile-Time Polymorphism. Happens in the case of **Method Overloading**

Run-Time Polymorphism- The function call bound to the class at the time of compilation, if the function is going to be executed from a different class (Parent Class) at run-time rather than the class bound at compilation-time, then it is called Run-Time Polymorphism. This happens in the case of **Method Overriding**

Early Binding- means target methods, properties, functions are detected and checked during compile time. If method/function/property does not exist or has data type issues then the compiler throws an exception during compile time.

Late Binding- methods, properties, functions are detected during runtime rather than compile time, If method/properties does not exist during runtime application will throw an exception.

Interface is a pure abstract class, which allows us to define only abstract methods. The abstract method means a method without a body or implementation. It is used to achieve multiple inheritances, which the class can't achieve. It is used to achieve full abstraction because it cannot have a method body.

Interfaces allow you to define a common set of functionality that multiple classes can share, promoting code reusability and ensuring a consistent structure for related classes.

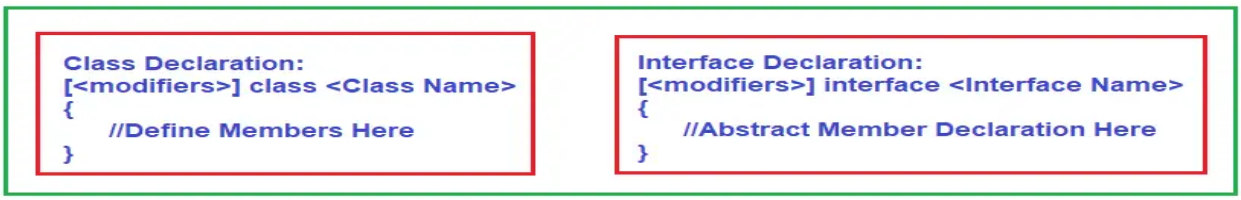
When to use Interface- Implementing Multiple Inheritance, Implementing Polymorphism, Testing and Mocking, Code Reusability ,Dependency Injection.

Difference-

1. **Class:** Contains only the Non-Abstract Methods (Methods with Method Body).
2. **Abstract Class:** Contains both Non-Abstract Methods (Methods with Method Body) and Abstract Methods (Methods without Method Body).
3. **Interface:** Contain only Abstract Methods (Methods without Method Body).

Note:

- Every abstract method of an interface should be implemented by the child class of the interface without fail.
- Default scope for an interface's members is public, whereas it is private in the case of a class.
- By default, every member of an interface is abstract, so we aren't required to use the abstract modifier. **void Add(int num1, int num2);**
- we cannot declare fields/variables, constructors, and destructors in an interface. An interface can contain Abstract methods, Properties, Indexes, Events.
- An interface cannot contain Non-abstract functions, Data fields, Constructors, Destructors.
- Interfaces can inherit from another interface. We cannot create an instance of an interface, but we can create a reference of an interface



Explicit Interface Implementation: When each interface method is implemented separately under the child class by providing the method name and the interface name, it is called Explicit Interface Implementation. But in this case, while calling the method, we should compulsorily use the interface reference created using the object of a class or typecast the object to the appropriate interface type.

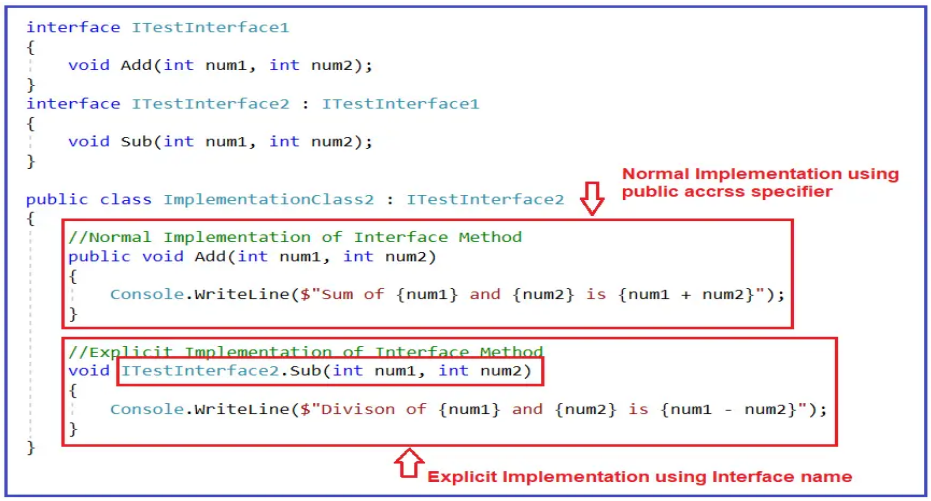
Method Overloading- Method Overloading allows a class to have multiple methods with the same name but with a different signature. The functions or methods can be overloaded based on the number, type (int, float, etc), order, and kind (Value, Ref or Out) of parameters.

The point that you need to keep in mind is that the signature of a method does not include the return type and the params modifiers. So it is not possible to overload a method just based on the return type and params modifier.

Methods can be overloaded in the same or in super and sub classes because overloaded methods are different methods. But we can't override a method in the same class it leads to Compile Time Error: "method is already defined"

example - system-defined "WriteLine()" method. It is an overloaded method, not a single method taking different types of values.

Ways to overload method- (**NTO)** - change **N**umber of parameters in method, use different data **T**ype for parameters, change **O**rder of parameters.



Method Overriding -process of re-implementing the superclass non-static and non-private method in the subclass with the same signature is called Function Overriding or Method Overriding.

The same signature means the name and the parameters should be the same. The implementation of the subclass overrides (i.e. replaces) the implementation of the superclass method.

Reason to use- If the superclass method logic is not fulfilling the sub-class business requirements, then the subclass needs to override that method with the required business logic.

The parent child method which is overridden is called the overridden method.

When a method in a subclass has the same name, same no. of arguments and the same type signature as a method in its super class then the method is known as an overridden method.

The child class method which is overriding is called the overriding method.

Extension Methods- Extension methods can be used as an approach to extending the functionality of a class in the future if the source code of the class is not available or we don't have any permission in making changes to the class.

Points to Remember while working with C# Extension methods

1- Extension methods must be defined only under the static class. If you check our NewClass, then you will see that the NewClass is a static class.
2- We already discussed that Static Class contains only Static Members. As an extension method is defined under a static class, it means the extension method should be created as a static method whereas once the method is bound with another class, the method changes into non-static. Now, if you check the methods in NewClass, then you will see that all three methods are declared as static only.
3- The first parameter of an extension method is known as the binding parameter which should be the name of the class to which the method has to be bound and the binding parameter should be prefixed with this As here we are creating these extension methods to extend the functionality of OldClass, so, you can check the first parameter of all these methods are going to be OldClass which is also prefixed with this keyword.
4- An extension method can have only one binding parameter and that should be defined in the first place on the parameter list.
5- If required, an extension method can be defined with normal parameters also starting from the second place of the parameter list. If you check the Test3 method, we have passed the second parameter as int and while calling this method we also need to pass one integer value.

```
using System;
namespace ExtensionMethods
{
    public class OldClass
    {
        public int x = 100;
        public void Test1()
        {
            Console.WriteLine("Method one: " + this.x);
        }
        public void Test2()
        {
            Console.WriteLine("Method two: " + this.x);
        }
    }
}
```

```
using System;
namespace ExtensionMethods
{
    public static class NewClass
    {
        public static void Test3(this OldClass O)
        {
            Console.WriteLine("Method Three");
        }
        public static void Test4(this OldClass O, int x)
        {
            Console.WriteLine("Method Four: " + x);
        }
        public static void Test5(this OldClass O)
        {
            Console.WriteLine("Method Five:" + O.x);
        }
    }
}
```

```
using System;
namespace ExtensionMethods
{
    class Program
    {
        static void Main(string[] args)
        {
            OldClass obj = new OldClass();
            obj.Test1();
            obj.Test2();
            //Calling Extension Methods
            obj.Test3();
            obj.Test4(10);
            obj.Test5();
            Console.ReadLine();
        }
    }
}
```

Constructors are the special types of methods of a class that are automatically executed whenever we create an instance (object) of that class. The Constructors are responsible for two things. One is the **object initialization** and the other one is **memory allocation**. The role of the new keyword is to create the object.

Constructor can be defined in abstract classes.

1. The constructor name should be the same as the class name. (Overriding not possible)
2. It should not contain return type even void also.
3. The constructor should not contain modifiers.
4. As part of the constructor body return statement with value is not allowed

Note- Base constructor is always called first, followed by Derived class constructor.

five types of constructors-

1- Default Constructor/Parameterless 2- Parameterized Constructor 3- Copy Constructor

4- Static Constructor 5- Private Constructor

Default Constructor- Implicitly Defined Constructors are parameter less and these constructors are also known as Default Constructors. This is because they are used to initialize the variables with default values. Implicitly Defined Constructors are public.The default constructor will assign default values to the data members (non-static variables).

How many constructors can be defined in a class -> ***within a class, we can define any number of constructors***. But the most important point that you need to remember is that ***each and every constructor must have a different signature***. A different signature means the number, type, and parameter order should be different. So in a class, we can define one no-argument constructor plus 'n' number of parameterized constructors.

Copy Constructor which takes a parameter of the same class type is called a ***copy constructor***. **This constructor is used to copy one object's data into another object**. The main purpose of the copy constructor is to initialize a new object (instance) with the values of an existing object (instance).

```
class Program {
    static void Main(string[] args){
        CopyConstructor obj1 = new CopyConstructor(10);
        obj1.Display();
        CopyConstructor obj2 = new CopyConstructor(obj1);
        obj2.Display();
        Console.ReadKey();  }    }

public class CopyConstructor{
    int x;
    //Parameterized Constructor
    public CopyConstructor(int i){
        x = i;  }
```

```
//Copy Constructor
public CopyConstructor(CopyConstructor obj) {
    x = obj.x; }

public void Display() {
    Console.WriteLine($"Value of X = {x}");  }
```

static constructor is used to initialize the static fields of the class. You can also write some code inside the static constructor which is going to be executed only once. The static data members are created only once even though we created any number of objects. **Static constructors can not be overloaded because static constructors are always parameterless.**

It is not possible to initialize non-static data members within a static constructor, it raises a compilation error.

Points to Remember while creating Static Constructor in C#:

1. There can be only one static constructor in a class.
2. The static constructor should be without any parameters.
3. It can only access the static members of the class.
4. There should not be any access modifier in the static constructor definition.
5. If a class is static then we cannot create the object for the static class.
6. Static constructor will be invoked only once i.e. at the time of first object creation of the class, from 2nd object creation onwards static constructor will not be called.

Private Constructor whose accessibility is private is known as a private constructor. When a class contains a private constructor then we cannot create an object for the class outside of the class. So, private constructors are used to create an object for the class within the same class. Generally, private constructors are used in the Remoting concept.

Points To Remember about C# Private Constructor:

1. Using Private Constructor in C# we can implement the singleton design pattern.
2. We need to use the private constructor in C# when the class contains only static members.
3. Using a private constructor is not possible to create an instance from outside the class.

constructor overloading When we define multiple constructors within a class with different parameter types, numbers and orders then it is called constructor overloading. **Base**

constructor is always called first,followed by derived class constructor. ***Constructor overriding is not possible.***

Destructor is also a special type of method present in a class, just like a constructor, having the same name as the class name but prefixed with ~ tilde. The constructor in C# is called when the object of the class is created. On the other hand, the destructor gets executed when the object of the class is destroyed.
A destructor method gets called when the object of the class is destroyed.

Static classes and static class members

static class is similar to a class that is both abstract and sealed. Static class cannot be instantiated or inherited and that all of the members of the static class are static in nature. To declare a class as static, you should mark it with the static keyword in the class declaration.
A static class can only have static members — you cannot declare instance members (methods, variables) in a static class. You can have a static constructor in a static class but you cannot have an instance constructor inside a static class.

When to use a static class- implement helper or utility classes as static classes since they don't need to be instantiated or inherited and generally contain a collection of some reusable methods and properties.

Static method- A static method can call only other static methods; It cannot call a non-static method. A static method can be called directly from the class, without having to create an instance of the class. A static method can only access static variables; it cannot access instance variables. static methods are a bit faster in execution than non-static methods, i.e., instance methods. Static objects are stored in the heap.
A non-static method can access any static method without creating an instance of the class. A non-static method can access any static variable without creating an instance of the class because the static variable belongs to the class.

Constants- are evaluated at compile time. Constants should be used when value is not changing during runtime. Constants support only value type variables.

Readonly- evaluated at runtime. Readonly variables used mostly when actual value is i=unknown before runtime. Readonly variables hold reference type variables. Initialized at time of declaration and in constructor.

Ref Keywords- Pass arguments by reference and not value. To use 'ref' keyword need to explicitly mention 'ref'
Void method (ref int refArg) { refArg = refArg+10;}

Out Keyword- pass arguments within methods and functions.

Call by value- in call by value, the value of the original parameter is copied into parameters of the function. It will not allow you to change the actual variables using function calls.

Call by reference- arguments and given parameters both point to the same address. It allows you to change the actual variables using function calls.

Struct- not support inheritance, are value type, can not have null reference, are passed by value.
Class- support inheritance, are reference type, reference can be null

Var- variables are declared using var keywords are statically typed. The type of the variable is decided by the compiler at compile time. var variable should be initialized at the time of declaration. So that the compiler will decide the type of the variable according to the value it initialized. If the variable is not initialized it throws an error. It cannot be used for properties or returning values from the function. It can only be used as a local variable in function.

Dynamic- variables declared using dynamic keywords are dynamically typed. Dynamic variable is decided by the compiler at run time. Dynamic variables need not be initialized at the time of declaration. Because the compiler does not know the type of the variable at compile time. If the variable is not initialized it will not throw an error. It can be used for properties or returning values from the function.

CTS- Common type system- ensures that datatypes defined in 2 different prog. Languages gets compiled to common data type.

CLS- Common lang. Specifications- set of rules, when any .net prog. Lang. adheres to these set of rules it can be consumed by any lang. Following specifications.