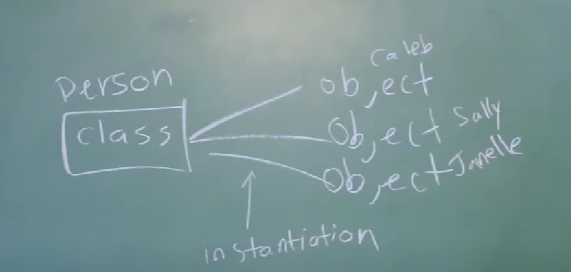
C# OOP

Class and objects



Class and Object are the basic concepts of Object-Oriented Programming which revolve around the real-life entities.

A class is a blueprint that defines the variables and the methods common to all objects of a certain kind. It helps us to bind data and methods together, making the code reusable.

For example, a mobile phone has attributes like a brand name , RAM, and functions like texting and calling. Thus, the mobile phone is a class of various phones (the objects).

An object is a single instance of a class, which contains data and methods working on that data. So an object consists of three things:

* Name: This is a variable name that represents the object.
* Member data: The data that describes the object.
* Member methods: Behavior that describes the object.

For example, Samsung Galaxy is an object with the brand name Samsung, 2GB RAM as properties, and calling and texting as behaviors.

Fields:-

* Fields are normal variable members of a class.
* Generally, you should declare your fields as private, then use Properties to get and set their values.
* By this way you won’t affect their values them directly



**Properties:-**

* They are actually special methods called “accessors”
* Properties are called accessors because they offer a way to get and set a field if you have a private field.
* They have two codes inside; set{}; and get{}; called “property accessors”.
* “value” is a keyword, It refers to the assigned value, It’s like a parameter for the set method
* Properties can be used to read only or write only other fields. This could be done by declaring only either get{} or set{}. Also they can have access modifiers, like private, so you can only get or set their values inside their class.

Method:-

* A **method** is a block of code which only runs when it is called.
* You can pass data, known as parameters, into a method.
* Methods are used to perform certain actions, and they are also known as **functions**.

Encapsulation:-

* Encapsulation in object-oriented programming (OOP) is a mechanism of hiding the internal details (implementation) of an object from other objects and the outside world
* In OOP, encapsulation is typically achieved through the use of access modifiers, such as "private" and "protected," which restrict access to certain members of a class.
* In C#, encapsulation can be implemented using properties, which are a special type of class member that provides a way to access and modify the value of a private field. Properties allow you to control access to the internal data of a class while providing a convenient and consistent way for other objects to access and modify that data.



* In this example, the private field "\_speed" is encapsulated by the public property "Speed." The "get" accessor of the property allows other objects to retrieve the value of the private field, while the "set" accessor allows them to modify it.

Static

* In C#, a static class is a class that cannot be instantiated.
* A static class is created using the static keyword in C# and .NET.
* A static class can contain static members only. You can‘t create an object for the static class.



## **Static Members :-**

There are two types of C# static class members, static and non-static.

***Non-static members***

* This is the default type for all the members.
* If you do not use the "static" keyword to declare a field/property or a method, then it can be called a "Non-static member."
* The main feature of a non-static member is it will be bound to the object only.

Non-static Fields / Properties

* The memory is allocated when the object is created.

Non-static Methods

* These methods can implement operations on non-static fields and properties

***Static Members***

If you use the "static" keyword for the declaration of a field/property or a method, it is called a "Static member".

The main feature of a non-static member is that it will not be bound to any object. Instead, it is individually accessible with the class name.

In other words, the static members are accessible directly without creating one object.

Static Fields / Properties

The memory will be allocated individually, without any relation to the object.

Static Methods

These methods can only implement operations on static fields and properties and can‘t access non-static members.

Inheritance:

* Acquiring (taking) the properties of one class into another is called inheritance. Code reusability is one of the key features of OOPs, and it is achieved via inheritance.
* Using inheritance, one or more classes can derive from an existing class. The existing class is called a base class, and the inherited class is called a derived or inherited class.

Overriding:

* Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes.
* When a method in a subclass has the same name, same parameters or signature and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.



Abstract Class:

An abstract class is an incomplete class or special class we can't be instantiated.

The purpose of an abstract class is to provide a blueprint for derived classes and set some rules what the derived classes must implement when they inherit an abstract class.

We can use an abstract class as a base class and all derived classes must implement abstract definitions. An abstract method must be implemented in all non-abstract classes using the override keyword.



Polymorphism:

* In c#, polymorphism provides an ability for the [classes](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) to implement different [methods](https://www.tutlane.com/tutorial/csharp/csharp-methods-functions-with-examples) called through the same name.
* It also provides an ability to invoke a derived class's methods through base class reference during runtime based on our requirements .
* Compile Time Polymorphism -> method overloading
* Run Time Polymorphism -> method overriding

Constructor:

* A constructor in C# is a member of a class. It is a method in the class which gets executed when a class object is created.
* Usually we put the initialization code in the constructor.
* The name of the constructor is always is the same name as the class.
* A C# constructor can be public or private.
* A class can have multiple overloaded constructors.



Interface:

An interface defines a contract. Any [class](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/class) or [struct](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/struct) that implements that contract must provide an implementation of the members defined in the interface.

