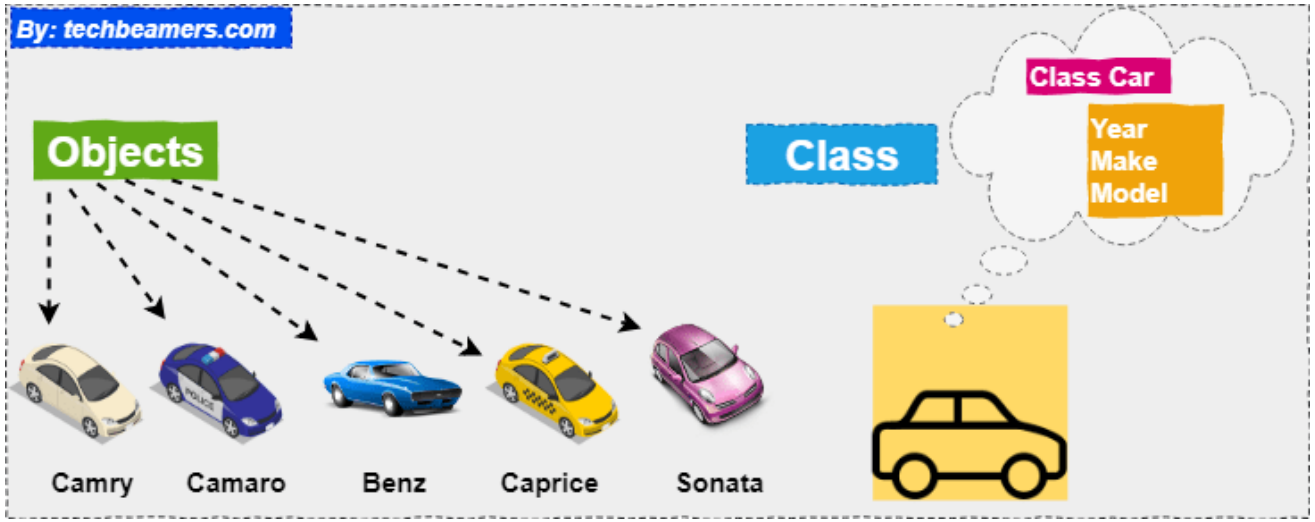
Class and Objects

Class:
A class is simply a user-defined data type that represents both state and behavior. The state represents the properties and **behavior** is the action that objects can perform.

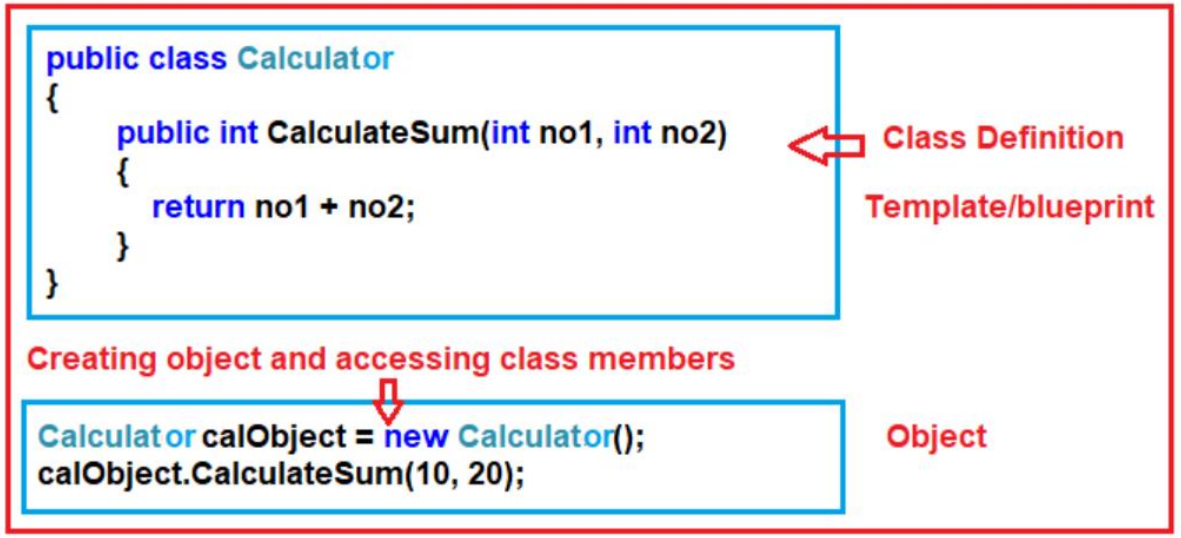
In other words, we can say that a class is the blueprint/plan/template that describes the details of an object. A class is a blueprint from which the individual objects are created. In C#, a Class is composed of three things i.e. a name, attributes, and operations.



Objects:
It is an instance of a class. A class is brought live by creating objects. An object can be considered as a thing that can perform activities. The set of activities that the object performs defines the object's behavior.

All the members of a class can be accessed through the object. To access the class members, we need to use the dot (.) operator. The dot operator links the name of an object with the name of a member of a class.

How can we create a Class and Object in C#?




```
class Program
{
    static void Main(string[] args)
    {
        //Creating object
        Calculator calObject = new Calculator();

        //Accessing Calculator class member using Calculator class object
        int result = calObject.CalculateSum(10, 20);

        Console.WriteLine(result);
        Console.ReadKey();
    }
}

//Defining class or blueprint or template
public class Calculator
{
    public int CalculateSum(int no1, int no2)
    {
        return no1 + no2;
    }
}
```

Types of classes in C#:

- Abstract class
- Concrete class
- Sealed class
- Partial Class
- Static class

What is a Constructor in C#?

We can define the constructors in C# 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

Rules to follow while creating the C# Constructors:

- ✓ The constructor name should be the same as the class name.
- ✓ It should not contain return type even void also.
- ✓ The constructor should not contain modifiers.
- ✓ As part of the constructor body return statement with value is not allowed.

```
class Example
{
        Example()
        {
                ---------- // all .net legal statements are allowed
                ---------- //except return statement with value
        }
}
```

How many types of constructors are there in C#.net?

- Default Constructor
- Parameterized Constructor
- Copy Constructor
- Static Constructor
- Private Constructor

Default Constructor in C#:

The Constructor without parameter is called a default constructor. Again the default constructor is classified into two types.

1.System-defined default constructor

2.User-defined default constructor

```
class Employee
{
    int eid, eage;
    String eaddress, ename;
    public void Display()
    {
        Console.WriteLine("\nemployee id is:  " + eid);
        Console.WriteLine("employee name is:  " + this.ename);
        Console.WriteLine("employee age is:  " + this.eage);
        Console.WriteLine("employee address is:  " + eaddress);
    }
}
class Test
{
    static void Main(string[] args)
    {
        Employee e1 = new Employee();
        Employee e2 = new Employee();
        e1.Display();
        e2.Display();
        Console.ReadKey();
    }
}
```

What is Parameterized Constructor in C#?

The developer given constructor with parameters is called the parameterized constructor in C#. With the help of a Parameterized constructor, we can initialize each instance of the class with different values. That means using parameterized constructor we can store a different set of values into different objects created to the class.

```
class Employee
{
    int eid, eage;
    String eaddress, ename;
    public Employee(int id, int age, string name, string address)
    {
        this.eid = id;
        this.eage = age;
        this.ename = name;
        this.eaddress = address;
    }
    public void Display()
    {
        Console.WriteLine("employee id is:  " + eid);
        Console.WriteLine("employee name is:  " + this.ename);
        Console.WriteLine("employee age is:  " + this.eage);
        Console.WriteLine("employee address is:  " + eaddress);
    }
}
class Test
{
    static void Main(string[] args)
    {
        Employee e1 = new Employee(101, 30, "Pranaya", "Mumbai");
        Employee e2 = new Employee(101, 28, "Rout", "BBSR");
        e1.Display();
        e2.Display();
        Console.ReadKey();
    }
}
```

What is Copy Constructor in C#?

The constructor which takes a parameter of the 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).

```
Employee e1 = new Employee();
Employee e2 = new Employee(e1);
```

What is Private Constructor in C#?

In C#, it is also possible to create a constructor as private. The 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 creating an object for the class within the same class. Generally, private constructors are used in the Remoting concept.

```
class Program
{
    private Program()
    {
        Console.WriteLine("this is private constructor");
    }
    static void Main(string[] args)
    {
        Program p = new Program();
        Console.WriteLine("main method");
        Console.ReadKey();
    }
}
```

What is constructor overloading?

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

```
class ConstructorOverloading
{
    int x;
    public ConstructorOverloading()
    {
        this.x = 10;
    }
    public ConstructorOverloading(int x)
    {
        this.x = x;
    }
    public void Display()
    {
        Console.WriteLine("the value of x:{0}", x);
    }
}
class Test
{
    static void Main(string[] args)
    {
        ConstructorOverloading obj1 = new ConstructorOverloading();
        ConstructorOverloading obj2 = new ConstructorOverloading(20);
        obj1.Display();
        obj2.Display();
        Console.ReadKey();
    }
}
```

Understanding Static Constructor in C#:

In C#, it is also possible to create a constructor as static and when we do so, it is called Static Constructor. The static Constructor in C# will be invoked only once. There is no matter how many numbers instances (objects) of the class are created, it is going to be invoked only once and that is when the class is load for the first time.

The 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 in C# are created only once even though we created any number of objects.

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.

Can we initialize non-static data members within a static constructor in C#?

It is not possible to initialize non-static data members within a static constructor, it raises a compilation error. Have a look at the following example.

Can we initialize static data fields within a non-static constructor in C#?

Yes, you can initialize static data members within a non-static constructor but after then they lose their static nature. Consider the following example:

What is Destructor in C#?

The 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 prefix with ~ tilde. The constructor in C# is called when the object of the class is created. On the other hand, the destructor in C# is gets executed when the object of the class is destroyed.

The Constructor and destructor methods will exactly have the same name as the class to which they belong. So to differentiate between these two a tilde (~) operator is used before the destructor method.

When will be the object of a class get destroyed in C#?

The object of a class in C# will be destroyed by the garbage collector in any of the following cases

1.At the end of program execution, each and every object that is associated with the program will be destroyed by the garbage collector.

2.The Implicit calling of the garbage collector occurs sometime in the middle of the program execution provided the memory is full so that the garbage collector will identify unused objects of the program and destroys them.

3.The Explicit calling of the garbage collector can be done in the middle of program execution with the help of the "**GC.Collect()**" statement so that if there are any unused objects associated with the program will be destroyed in the middle of the program execution.

```
class DestructorDemo
{
    public DestructorDemo()
    {
        Console.WriteLine("constructor object created");
    }
    ~DestructorDemo()
    {
        Console.WriteLine("object is destroyed");
    }
}
```

Raw Copy

```
namespace DestructorExample
{
    class DestructorDemo
    {
        public DestructorDemo()
        {
            Console.WriteLine("constructor object created");
        }
        ~DestructorDemo()
        {
            Console.WriteLine("object is destroyed");
        }
    }
    class Test
    {
        static void Main(string[] args)
        {
            DestructorDemo obj1 = new DestructorDemo();
            DestructorDemo obj2 = new DestructorDemo();
            obj1 = null;
            obj2 = null;
            GC.Collect();
            Console.ReadKey();
        }
    }
}
```

What are Access Specifiers in C#?

The Access Specifiers in C# are also called access modifiers which are used to define the scope of the type (class and interface) as well as the scope of their members (variables, properties, and methods). That is who can access them and who cannot access them are defined by the Access Specifiers.

Private | Public | Protected | Internal | Protected Internal

Access Modifiers	Inside Assembly		Outside Assembly	
	With Inheritance	With Type	With Inheritance	With Type
Public	✓	✓	✓	✓
Private	X	X	X	X
Protected	✓	X	✓	X
Internal	✓	✓	X	X
Protected Internal	✓	✓	✓	X