**EXPERIMENT NO: 02**

**Title**: OOPS concepts in C#-Class, Implementation Inheritance, Extension methods.

**Aim:** Study of OOPS concepts.

**Theory:**

**C# | Class:**

1. A [class](https://www.geeksforgeeks.org/classes-objects-java/)is a user-defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type.
2. It will combine various types of data members such as fields, properties, and member functions.

Declaring a class :- Class are declared by using Class keywords.

public class users

{

public int a, b; // field or data members

// member function or method

public void display()

{

Console.WriteLine(“Class & Objects in C#”);

}

}

**Objects:**

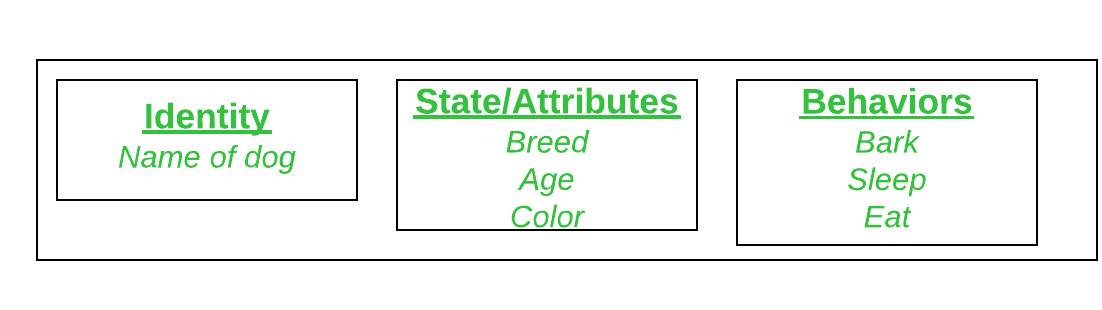
Object is an instance of a class & that can be used to access the data members & member function of a class.

**State:** It is represented by attributes of an object. It also reflects the properties of an object.

**Behavior:** It is represented by the methods of an object. It also reflects the response of an object with other objects.

**Identity:** It gives a unique name to an object and enables one object to interact with other objects.

Consider Dog as an object and see the below diagram for its identity, state, and behavior.



Objects correspond to things found in the real world. For example, a graphics program may have objects such as “circle”, “square”, “menu”. An online shopping system might have objects such as “shopping cart”, “customer”, and “product”.

Object is a runtime entity, it is created at runtime.

Object is an instance of a class. All the members of the class can be accessed through object.

We can create objects by using a “new” keyword followed by the name of the class.

**Users user=new Users();**

**Initializing an object:**

The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the class constructor.

**Example:**

using System;

public class BankAccount

{

private string accountNumber;

private decimal balance;

public string AccountNumber

{

get { return accountNumber; }

set { accountNumber = value; }

}

public decimal Balance

{

get { return balance; }

private set { balance = value; }

}

public void Deposit(decimal amount)

{

balance += amount;

}

public void Withdraw(decimal amount)

{

if (amount <= balance)

balance -= amount;

}

}

public class Program

{

public static void Main()

{

BankAccount myAccount = new BankAccount();

myAccount.AccountNumber = "123456789";

myAccount.Deposit(1000);

myAccount.Withdraw(500);

Console.WriteLine("Account Number: " + myAccount.AccountNumber);

Console.WriteLine("Balance: $" + myAccount.Balance);

}

}

**Output:**

**Add the output here**

**Inheritance:**

Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in C# by which one class is allowed to inherit the features(fields and methods) of another class.

**Important terminology:**

* **Super Class:** The class whose features are inherited is known as super class(or a base class or a parent class).
* **Sub Class:** The class that inherits the other class is known as subclass(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
* **Reusability:** Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

**Syntax:**

<access\_modifier> class <base\_class\_name>  
{  
// Base class Implementation  
}  
  
<access\_modifier> class <derived\_class\_name> : <base\_class\_name>  
{  
// Derived class implementation  
}

**Example:** using System;

namespace Inheritance {

// base class

class Vehicle {

public string name;

public void engine() {

Console.WriteLine("This is the engine of the vehicle");

}

}

// derived class of Vehicle

class Car : Vehicle {

public void display() {

Console.WriteLine("This car is" + name);

}

}

class Program {

static void Main(string[] args) {

// object of derived class

Car c = new Car();

c.name = " Toyota ";

// method of base class

c.engine();

// method from own class

c.display();

}

}

}

**Output:**

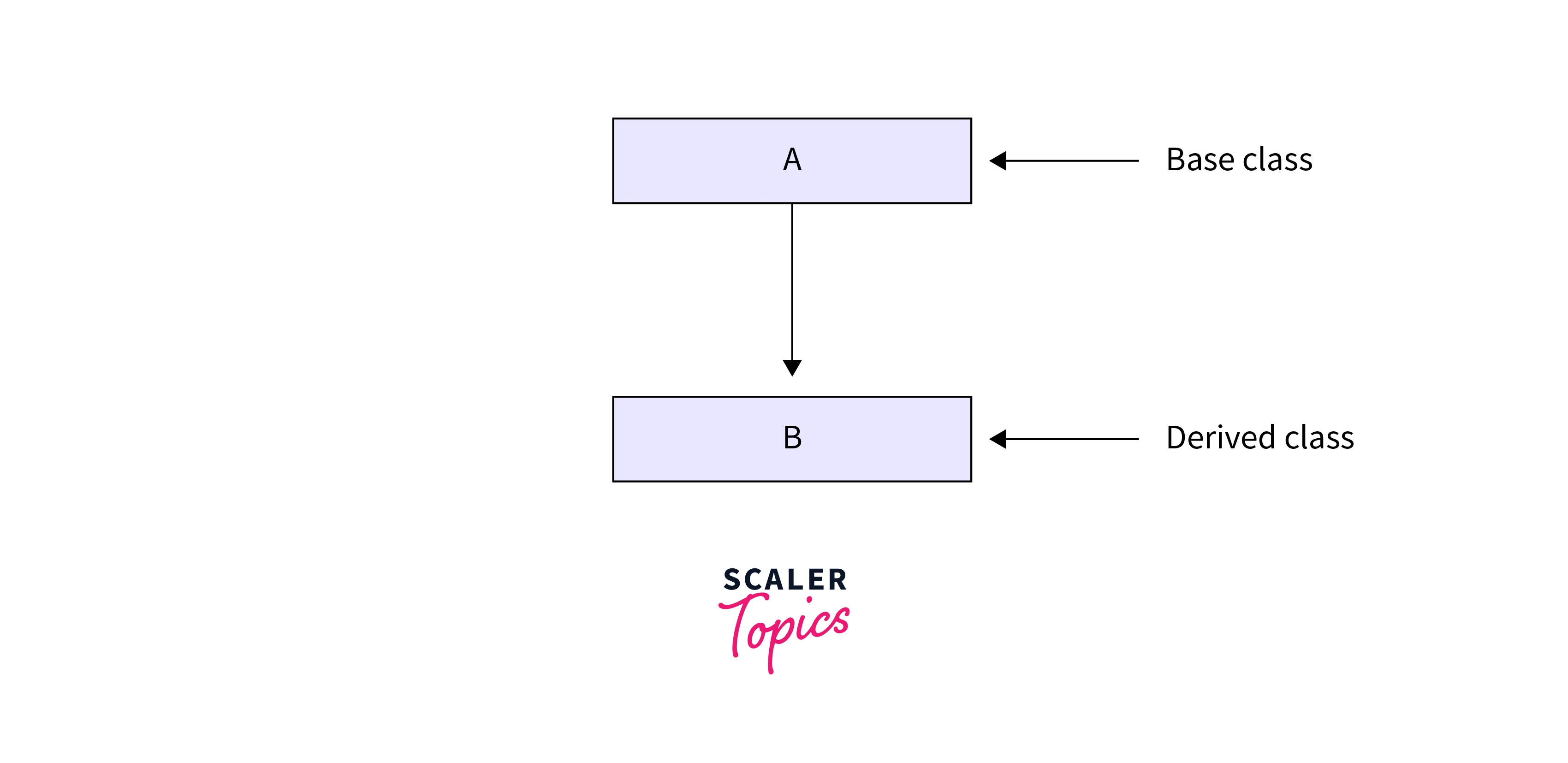
This is the engine of the vehicle

This car is Toyota.

**Types of Inheritance in C#**

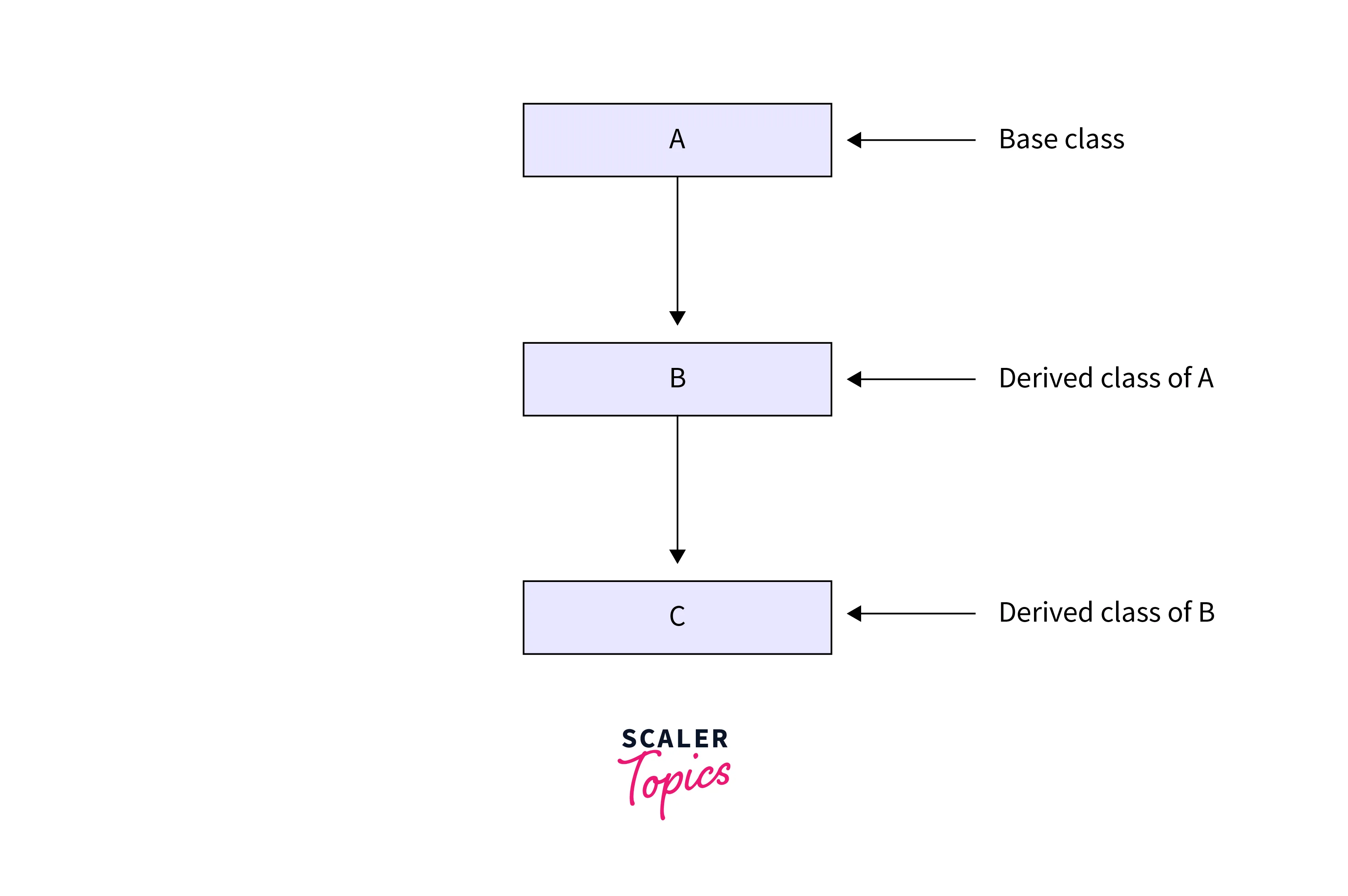
**Below are the different types of inheritance which is supported by C# in different combinations.**

**Single Inheritance:** In single inheritance, a derived class inherits the properties of only one base class.

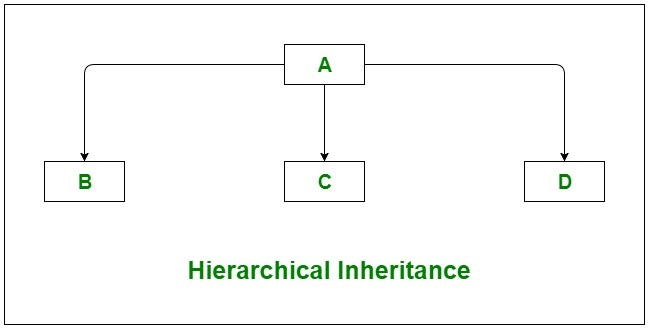


**Multilevel Inheritance:** When one class inherits another class which is further inherited by another class, it is known as multilevel inheritance in C#.

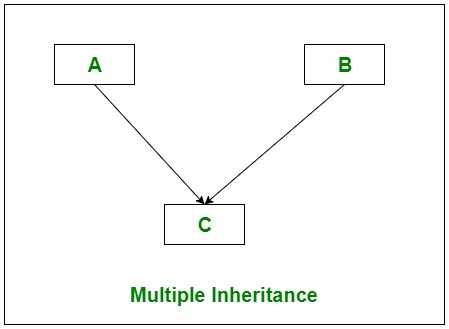
Inheritance is transitive so the last derived class acquires all the members of all its base classes



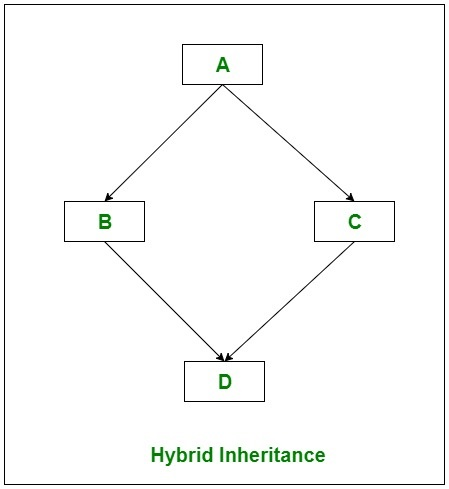
**Hierarchical Inheritance:** In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one subclass. In below image, class A serves as a base class for the derived class B, C, and D.



**Multiple Inheritance(Through Interfaces):**In Multiple inheritance, one class can have more than one superclass and inherit features from all parent classes. Please note that C# does not support multiple inheritance with classes. In C#, we can achieve multiple inheritance only through Interfaces. In the image below, Class C is derived from interface A and B.



**Hybrid Inheritance(Through Interfaces):** It is a mix of two or more of the above types of inheritance. Since C# doesn’t support multiple inheritance with classes, the hybrid inheritance is also not possible with classes. In C#, we can achieve hybrid inheritance only through Interfaces.



**Important facts about inheritance in C#:**

* **Default Superclass:** Except Object class, which has no superclass, every class has one and only one direct superclass(single inheritance). In the absence of any other explicit superclass, every class is implicitly a subclass of Object class.
* **Superclass can only be one:** A superclass can have any number of subclasses. But a subclass can have only one superclass. This is because C# does not support multiple inheritance with classes. Although with interfaces, multiple inheritance is supported by C#.
* **Inheriting Constructors:** A subclass inherits all the members (fields, methods) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.
* **Private member inheritance:** A subclass does not inherit the private members of its parent class. However, if the superclass has properties(get and set methods) for accessing its private fields, then a subclass can inherit.

**Constructors:**

A constructor is a special method of the class which gets automatically invoked whenever an instance of the class is created. Like methods, a constructor also contains the collection of instructions that are executed at the time of Object creation. It is used to assign initial values to the data members of the same class.

**Example :**

class Geek

{

.......

// Constructor

public Geek() {}

.......

}

// an object is created of Geek class,

// So above constructor is called

Geek obj = new Geek();

**Important points of Constructors:**

* Constructor of a class must have the same name as the class name in which it resides.
* A constructor can not be abstract, final, and Synchronized.
* Within a class, you can create only one static constructor.
* A constructor doesn’t have any return type, not even void.
* A static constructor cannot be a parameterized constructor.
* A class can have any number of constructors.
* Access modifiers can be used in constructor declaration to control its access i.e which other class can call the constructor.

**Types of Constructor:**

* Default Constructor
* Parameterized Constructor

**Default Constructor:**

A constructor with no parameters is called a default constructor. A default constructor has every instance of the class to be initialized to the same values.

**Example :**

using System;

public class Employee

{

public Employee()

{

Console.WriteLine("Default Constructor Invoked");

}

public static void Main(string[] args)

{

Employee e1 = new Employee();

Employee e2 = new Employee();

}

}

**Output :**

Default Constructor Invoked

Default Constructor Invoked

**Parameterized Constructor**

A constructor having at least one parameter is called as parameterized constructor. It can initialize each instance of the class to different values.

**Example :**

using System;

public class Employee

{

public int id;

public String name;

public float salary;

public Employee(int i, String n,float s)

{

id = i;

name = n;

salary = s;

}

**public** **void** display()

        {

            Console.WriteLine(id + " " + name+" "+salary);

        }

   }

class TestEmployee{

public static void Main(string[] args)

{

Employee e1 = new Employee(101, "Sonoo", 890000f);

Employee e2 = new Employee(102, "Mahesh", 490000f);

e1.display();

e2.display();

}

}

**Output :**

101 Sonoo 890000

102 Mahesh 490000

**Abstraction in C#:**

**-Abstraction** is basically showing the required feature to a user and hiding the implementation and detail. In C# we can achieve the abstraction with the help of abstract class

We can declare the abstract class with the keyword of abstract

It is not allowed to create abstract class objects in C #. Or in other words, you cannot use the abstract class directly with the new operator.

The class that contains the abstract keyword along with some of the methods (not all abstract methods) is known as the abstract base class.

The class that contains the abstract keyword with all its methods is called a pure abstract base class.

You cannot explain abstract methods outside the abstract class.

You cannot declare an abstract class as a sealed class.

Abstraction is an important part of object oriented programming. It means that only the required information is visible to the user and the rest of the information is hidden.

Abstraction can be implemented using abstract classes in C#. Abstract classes are base classes with partial implementation. These classes contain abstract methods that are inherited by other classes that provide more functionality.

**Some of the salient points about abstract classes are as follows**:

1. The abstract class is created using the keyword abstract and some of the methods of the abstract class also contain the keyword abstract.
2. No object can be created of the abstract class i.e.it cannot be instantiated.
3. The abstract methods in the abstract class are implemented actually only in the derived classes.

If all the methods in the abstract class contain the keyword abstract, then that class is known as pure Abstract class

Example:

using System;

namespace abstraction

{

//abstract class

abstract class Shape

{

//abstract methods

public abstract double calculateArea();

public abstract void displayDetails(double area);

}

//Rectangle class inheriting Shape class

class Rectangle : Shape

{

//private data members

private double length;

private double breadth;

public Rectangle(double length, double breadth)

{

this.length = length;

this.breadth = breadth;

}

//overriding abstract methods of Shape class using 'override’ keyword

public override double calculateArea()

{

return (length \* breadth);

}

public override void displayDetails(double area)

{

Console.Write("Length of rectangle: "+length);

Console.Write("\nBreadth of rectangle: "+breadth);

Console.Write("\nArea of rectangle: "+area);

}

}

//Square class inheriting Shape class

class Square : Shape{

//private data members

private double side;

public Square(double side)

{

this.side = side;

}

//overriding abstract methods of Shape class using 'override' keyword

public override double calculateArea()

{

return (side \* side);

} public override void displayDetails(double area)

{

Console.Write("Length of a side of square: "+side);

Console.Write("\nArea of square: "+area);

}

}

public class AbstractionDemo

{

public static void Main(string[] args)

{

double area;

//creating reference of Shape class using Rectangle class

Shape shapeRec = new Rectangle(5,6);

area = shapeRec.calculateArea();

shapeRec.displayDetails(area);

Console.WriteLine("\n");

//creating reference of Shape class using Square class

Shape shapeSquare = new Square(4);

area = shapeSquare.calculateArea();

shapeSquare.displayDetails(area);

}

}

}

**C# | Encapsulation**

Encapsulation is defined as the grouping of data into a single unit.

encapsulation is achieved by defining the class's fields as private or protected, which means that they are not directly accessible from outside the class.

Access to the class's fields is provided through methods, which are called "accessors" or "getters" and "mutators" or "setters".

Use properties:

In C#, you can use properties to encapsulate fields and provide access to their values.

Properties are special methods that allow you to get or set the value of a field while controlling access to it.

**Example:**

using System;

public class Person

{

private string name;

private int age;

public string Name

{

get { return name; }

set { name = value; }

}

public int Age

{

get { return age; }

set { age = value; }

}

}

class Program

{

static void Main(string[] args)

{

Person person = new Person();

person.Name = "John";

person.Age = 30;

Console.WriteLine("Name: " + person.Name);

Console.WriteLine("Age: " + person.Age);

}

}

Output:

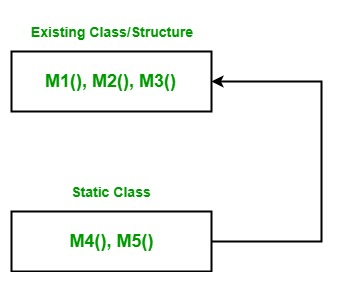
Name: John

Age: 30

**Extension Method in C#:**

In C#, the extension method concept allows you to add new methods in the existing class or in the structure without modifying the source code of the original type and you do not require any kind of special permission from the original type and there is no need to re-compile the original type. It is introduced in C# 3.0.

Let us discuss this concept with the help of an example. Suppose you have a class or a structure which contains three methods and you want to add two new methods in this class or structure, you did not have the source code of the class/structure, or do not have permissions from the class/structure, or the class is a sealed class, but you still want to add new methods in it, then you can use the concept extension method to add the new method in the existing class/structure. Now you create a new class which is static and contain the two methods which you want to add in the existing class, now bind this class with the existing class. After binding you will see the existing class can access the two new added methods. As shown in the below program.



**Example:** First we create a class named as Geek in Program1.cs file. It contains three methods that is M1(), M2(), and M3().

// C# program to illustrate the concept

// of the extension methods

using System;

namespace ExtensionMethod {

// Here Geek class contains three methods

// Now we want to add two more new methods in it

// Without re-compiling this class

class Geek {

// Method 1

public void M1()

{

Console.WriteLine("Method Name: M1");

}

// Method 2

public void M2()

{

Console.WriteLine("Method Name: M2");

}

// Method 3

public void M3()

{

Console.WriteLine("Method Name: M3");

}

}

}

Now we create a static class named as NewMethodClass in Program2.cs file. It contains two methods that are M4() and M5(). Now we want to add these methods in Geek class, so we use the binding parameter to bind these methods with Geek class. After that, we create another named as GFG in which Geek class access all the five methods.

// C# program to illustrate the concept

// of the extension methods

using System;

namespace ExtensionMethod {

// This class contains M4 and M5 method

// Which we want to add in Geek class.

// NewMethodClass is a static class

static class NewMethodClass {

// Method 4

public static void M4(this Geek g)

{

Console.WriteLine("Method Name: M4");

}

// Method 5

public static void M5(this Geek g, string str)

{

Console.WriteLine(str);

}

}

// Now we create a new class in which

// Geek class access all the five methods

public class GFG {

// Main Method

public static void Main(string[] args)

{

Geek g = new Geek();

g.M1();

g.M2();

g.M3();

g.M4();

g.M5("Method Name: M5");

}

}

}

Output:

Method Name: M1

Method Name: M2

Method Name: M3

Method Name: M4

Method Name: M5

**Important Points:**

* Here, Binding parameters are those parameters which are used to bind the new method with the existing class or structure. It does not take any value when you are calling the extension method because they are used only for binding not for any other use. In the parameter list of the extension method binding parameter is always present at the first place if you write binding parameter to second, or third, or any other place rather than first place then the compiler will give an error. The binding parameter is created using this keyword followed by the name of the class in which you want to add a new method and the parameter name. For example:

this Geek g

* Here, this keyword is used for binding, Geek is the class name in which you want to bind, and g is the parameter name.
* Extension methods are always defined as a static method, but when they are bound with any class or structure they will convert into non-static methods.
* When an extension method is defined with the same name and the signature of the existing method, then the compiler will print the existing method, not the extension method. Or in other words, the extension method does not support method overriding.
* You can also add new methods in the sealed class also using an extension method concept.
* It cannot apply to fields, properties, or events.
* It must be defined in top-level static class.
* Multiple binding parameters are not allowed means an extension method only contains a single binding parameter. But you can define one or more normal parameter in the extension method.

**Advantages:**

* The main advantage of the extension method is to add new methods in the existing class without using inheritance.
* You can add new methods in the existing class without modifying the source code of the existing class.
* It can also work with sealed class.

**Problem Statement:**

Create a program to model a basic library system. Implement a base class called Library Item with properties Title and Number Of Copies. Create two derived classes, Book and DVD. Implement methods to check out and return items, and display information about the library items.

**Conclusion:**

Here we studied oops concepts in c#, class, implementation inheritance & extension method.

**Sample Questions:**

1) What is OOPS features?

2) Explain class and objects?

3) Explain Inheritance?

4) What is difference between abstraction and encapsulation?

5) Explain Constructors?