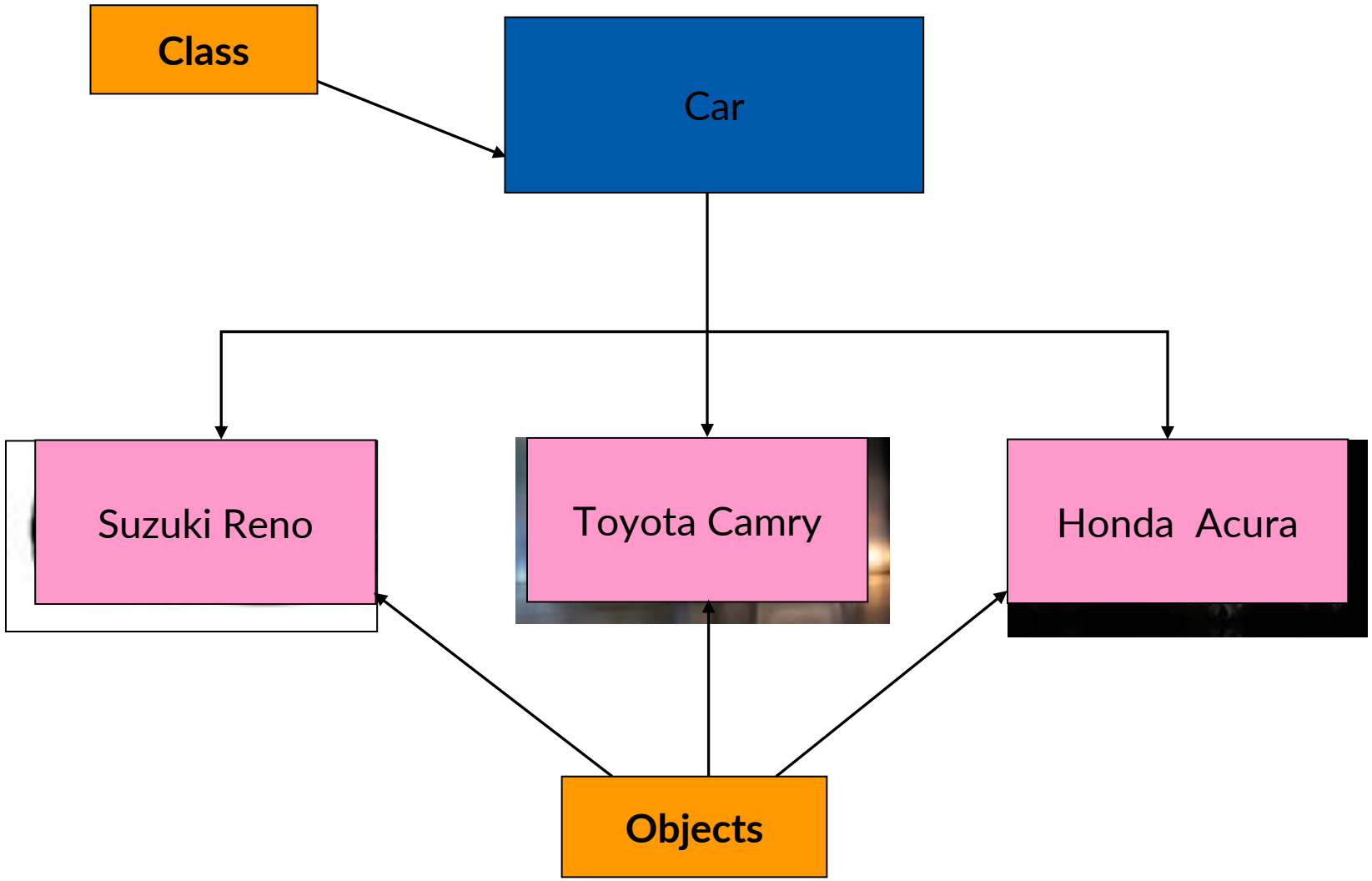
Programming Language - C#

OOPs

Object-Oriented Methodology

- Object orientation is a software development methodology that is based on modeling a real-world system.
- An object-oriented program consists of classes and objects.
- Below are the characteristics of object-oriented methodology
    - Realistic modeling
    - Reusability
    - Resilience to change
    - Existence as different forms

Object-Oriented Methodology (Contd.)

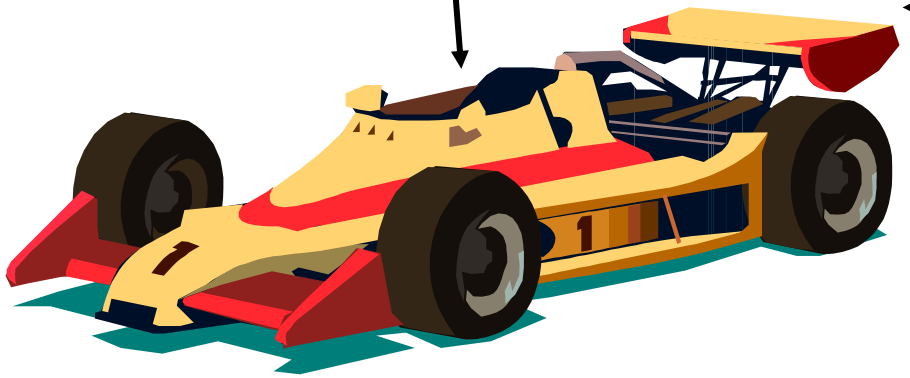


The Foundation of Object Orientation (Contd.)

Car positioned at one place defines it's State

Movement of car defines it's Behavior

Car number XX 4C 4546 shows the Identity of the car



The Foundation of Object Orientation (Contd.)

Car is flashing the lights to pass the message to the other car

Just a minute

- Identify the possible states of the following objects:
  - A cell phone
  - A stereo

Solution:

- States of a cell phone: Off, Ring, Vibrate, and Call
- States of a stereo: Play, Pause, Rewind, and Forward

Creating a Sample C# Program

A C# program can be written by using an editor like Notepad. Consider the following code, which declares a class Car and also creates an object MyCar of the same class:

```
using System;
  class Car
  {
    //Member variables
    string Engine;
    int NoOfWheels;
    //Member functions
     void AcceptDetails(){}
    }
```

The using keyword is used to include the namespaces in the program.

Comments are used to explain the code and are represented by // symbols.

Member variables are used to store the data for a class.

Member functions are declared inside the class that are used to perform a specific task.

Creating a Sample C# Program (Contd.)

```
        {
            Console.WriteLine("Enter the Engine Model");
            Engine = Console.ReadLine();
            Console.WriteLine("Enter the number of Wheels");
            NoOfWheels = Convert.ToInt32(Console.ReadLine());
        }
        public void DisplayDetails()
        {
            Console.WriteLine("The Engine Model is:{0}", Engine);
            Console.WriteLine("The number of wheels are:{0}", NoOfWheels);
        }
    }
```

Creating a Sample C# Program (Contd.)

```
//Class used to instantiate the Car class
class ExecuteClass
{
   public static void Main(string[] args)
   {
      Car MyCar = new Car();
      MyCar.AcceptDetails();
      MyCar.DisplayDetails();
   }
}
```

The Execute class is used as a class from where the Car class can be instantiated.

Assignment – Classes

- As a member of a team that is developing toys for JoyToys, Inc., you have been assigned the task of creating a bike module that accepts and displays bike details. Declare the Bike class and its member functions. The member function that accepts bike details should display the message “Accepting Bike Details”. Similarly, the member function to display bike details on the screen should display the message “Displaying Bike Details”.

Advantages

- Multiple developers can work simultaneously with a single class in separate files.
- Separate UI design code and business logic code
- Better maintenance of large classes by compacted them.
- Usage in LINQ to SQL or entity framework

Object class

Method	Description
GetType	Returns type of the object
Equals	Compares two object instances , returns true if they are equal otherwise false
RefereceEquals	Compares two object instances, returns true if both are same instances otherwise false
getToString	Converts an instance to a string type
GetHashCode	Returns hashcode for an object

Inheritance

- Syntax for inheritance in C#

```
Class A
{
Public void F(){

//code
}
}
```

```
Class B:A
{
Public void G(){
//code
//
}
}
```

```
Class test
{
static void main(){
B b=new B();
b.F();
b.G():
}}
```

- In C# you should declare the method as virtual in order to override

Contd..

```
Class A
{
Public virtual void F(){
//code
}
}
```

```
Class B:A
{
Public override void F(){
//code
}}
```

```
Class test
{
static void main(){
B b=new B();
b.F();

A a=new A():
a.F():
}}
```

Hiding methods

- In method hiding you can hide the implementation of the methods of a base class from the derived class
- This is done with the help of the new keyword.
- In method hiding you can redefine the method of the base class in the derived class

Using Virtual Functions

- When you have a function defined in a class which you want to allow to be implemented by the inherited classes, you can use virtual function.

- The virtual function could be implemented by the inherited classes in their own way and the call to the method is decided at the run time.

Contd...

```
class Hello
{
        public void sayHello()
        {
       console.writeline("hello");
        }
}

class FrenchHello:Hello
{       public void sayHello()
        {
        console.writeline("bonjour");
        }
}
```

WARNING : FrenchHello.sayHello hides inherited member Hello.sayHello

```
class Hello
{
        public void sayHello()
        {
        console.writeline("hello");
        }
}

class FrenchHello:Hello
{       public new void sayHello()
        {
        console.writeline("bonjour");
        }
}
```

Sealed classes

- C# allows classes and methods to be declared as sealed, this mean that you can never derive from it
- Are typically used when creating a framework of classes to be used by other unknown developers.
- Cannot be used as base or abstract classes
- If you want to declare a method as sealed it must be declared virtual in the base class.

Contd..

```
sealed class A
{

}

Class B:A
{

}
```

You cant extend A Class as it is sealed

```
class A
{
public virtual void X(){}
}

Class B:A
{
sealed public override void X(){}
}
```

```
Class C:A
{
public override void X(){}

}
```

You cant override x() because the method is sealed

ASSIGNMENT - INHERITANCE

Create classes employee and manager. An employee will have attributes such as id , name , salary, dob. A manager extends from an employee he will have additionally properties such as onsite allowance and bonus. Compute the salary of an employee and manager.

CONSTRUCTORS

- A constructor has the same name has that of the class
- A default constructor is created automatically if your class doesn’t have one
- A constructor can be declared as private constructor, and it is not possible for other classes to derive from this class – SINGLETON PATTERN
- A static constructor initializes static fields or data of the class and is executed only once.

INHERITANCE IN CONSTRUCTORS

```
class Person
{           protected string name;
            public Person(){
            name=null;
            }
            public Person(string n)
            {
            this.name=n;
            }}
```

```
class Employee:Person
{
            protect int empid

            public Employee(string name,int
empid):base(name){
            this.empid=empid
            }
}
```

- only the default constructor of the base class will be called automatically
- If you want to call the parameterized constructor, use the base keyword.

Static Variables & functions

- The keyword 'static' means that only one instance of a given variable exists for a class.

- Static variables can be initialized outside the member function or class definition.

- Static functions can access only static variables.

- Static functions exist even before the object is created.

- To manipulate and use the values of static variables, you can define a function as static function.

Assignment – Static

John a software developer in Zed Axis Technology needs to check how many times a function is called. For the same, he has been asked to create the function called “CountFunction”. Help John to create this function.

Anonymous types

- C# allows you to create an object with the new keyword without defining its class or type.
- The implicit typed variable – var is used to hold the reference of anonymous types
- The type of the type is decided by the compiler
- Predominantly used in LINQ and lambda expressions

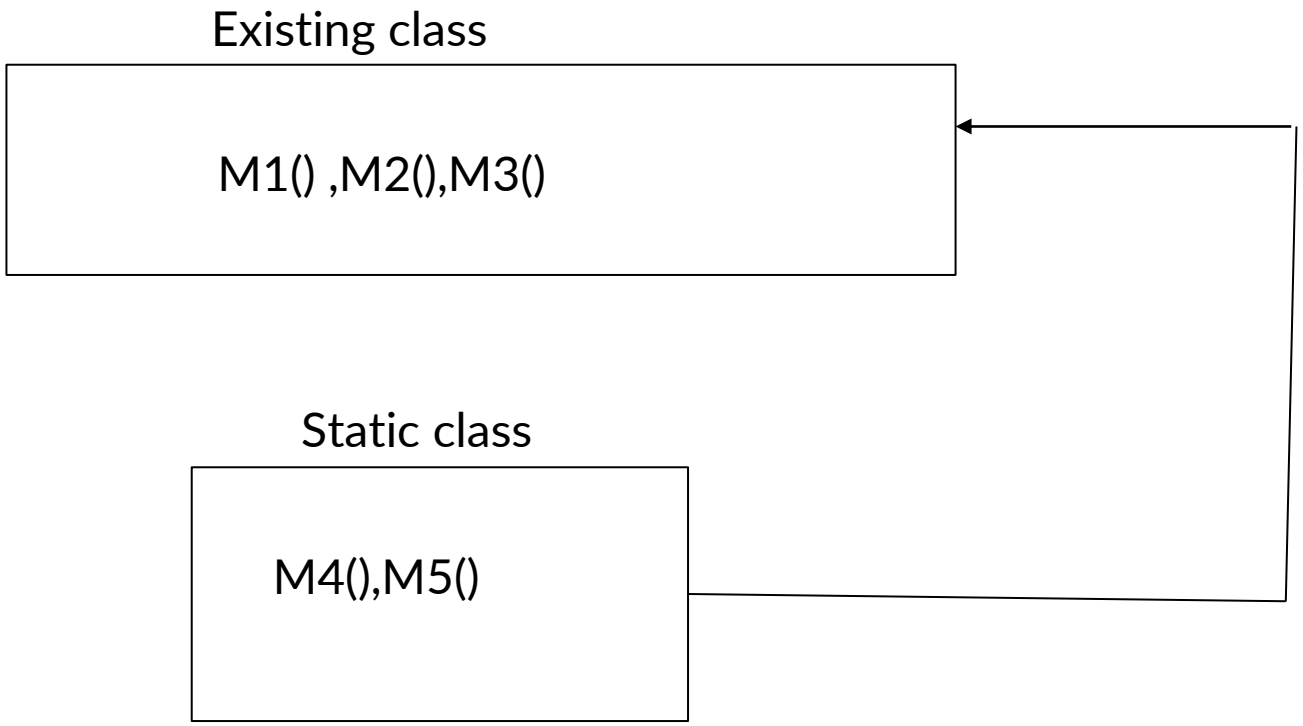
```
Var anonymousData = new {
                                FirstName ='Steve',
                                SecondName='Jobs'
                            }

Console.Writeline("firstname :"+anonymousData.FirstName);
```

Assignment – Anonymous & dynamic type

- Illustrate the difference between anonymous type and dynamic type with an example
- Daniel is working for a telecommunication company , he would like to collect the feed back from his customers on the quality of their companies' service, he has not decided the parameters that contribute to the feedback
- Which type of variable should he create. Illustrate with an example.

Extension methods

- In C# the extension method allows you to add new methods in the existing class or structure without modifying the source code .



- Existing class can use the static class methods.

Contd..

```
Public class SampleClass
{
        public string Display()
        {
        console.writeline("in display method"):
        }
        public stirng Print()
        {
        console.writeline("in print");
        }
}

Public static class ExtSampleclass
{
public static void NewMethod(this SampleClass
ob)
{
        console.writelin("hello I am in extension"):
        }
}


Class program
{
        static void main(string [] args)
        {
        SampleClass sc=new SampleClass():
        sc.Display():
        sc.Print();
        sc.NewMethod();
        }
}
```

Assignment -Extension methods

- Patrick works for an online shopping company , he wants to validate the customers email id and phone no , using string builder and extension method concepts develop a code to test the correctness of email id and phone no using regular expressions.

Nested classes

- In c# a user can create a class within another class.
- This features enables the user to logically group classes that only used in on place – thus increases encapsulation and create more readable and maintainable code.
- A nested class can be declared as private public protected internal protected internal
- Outer class is not allowed to access inner class members directly
- You are allowed to create objects of inner class in outer class
- Inner class can access static members of outer class

Contd...

```
public class Container
{
        public class Nested
        {
                private Container parent;
                public Nested(){}

                public Nested(Container parent)
                {
                this.parent=parent;
                }
        }
}
```

Attributes usage

- Attributes are used in C# to convey declarative information or meta data.
- Meta data such as assemblies, properties, types and methods.
- Attributes are added to the code by the [] on top of the required code element
- Types of attributes
    - Predefined
    - Custom

Contd..

- Attributes can have arguments just like methods, properties
- Attribues can have zero or more parameters
- Reflection can be used to obtain the meta data of the program by accessingthe attributes at run time
- Attributes are generally derived from **System.Attribute** class

Predefined attributes

- Predefined attributes are those attributes that are a part of the .NET framework class library and are supported by the compiler

- Examples : AttributeUsageAttribute
    - CLSCompliantAttribute
    - ContextStaticAttribute
    - FlagAttribute

```
class Sample{
        [Obsolete("method is obsolete',true)} static  void method()
        {
        C.W.L("obsolete method");
        }
}
```


Custom attribute

Steps for creating custom attribute

- Define a custom attribute class that is derived from System.Attribute class
- The custom attribute class name should have the suffix Attribute
- Use the attribute AttributeUsage to specify the uage of the custom attribute class created
- Create the constructor and the accessible properties of the custom attribute class

Defining Encapsulation

- Encapsulation involves packaging one or more components together.
- Encapsulation can be achieved by the following mechanisms

  Access specifiers (public, private ,protected, internal and protected internal)
  Properties

Properties

- A property is a member that provides a flexible mechanism to read, write, or compute the value of a private field.
- Properties can be used as if they are public data members, but they are special methods called accessors & mutators
- This enables data to be accessed easily and still helps promote the safety and flexibility of methods.
- For simple properties that require no custom accessor code, consider the option of using auto-implemented properties.

Demo

```
Public int Age{
        get
                { return this.age;}

        set
                { if(this.age>=65)
                { this.retired=true;
                this.age=value;
                }
                }
            }
```

Assignment - Properties

- Create a TimePeriod class that stores a time period. Internally the class stores the time in seconds, but a property named Hours enables a client to specify a time in hours. The accessors for the Hours property perform the conversion between hours and seconds.

- Explain the C# access specifiers scope in detail

Introducing Abstraction

- Abstraction involves extracting only the relevant information.

- Abstraction can be achieved by the following mechanisms

  100% abstraction - Interface

  Partial abstraction – Abstract classes

Using Abstract Classes

- It can have abstract and non abstract methods
- Its implementation must be provided by the derived class
- There are certain rules governing the use of an abstraction class:
    - Cannot create an instance of an abstract class.
    - Cannot declare an abstract method outside an abstract class.
    - Cannot be declared sealed.

Using Abstract Methods

- Abstract methods are methods without any body.
- The implementation of an abstract method is done by the derived class.
- When a derived class inherits the abstract method form the abstract class, it must override the abstract methods. This requirement is enforced at compile time and is also called dynamic polymorphism.

The syntax for using the abstract method is as follows:

- [access-modifiers] abstract return-type method name (parameters]);
- The abstract method is declared by adding the abstract modifier to the method.

Assignment – Abstract classes

Furniture and Fittings Company (FFC) manufactures domestic furniture. Customers provide their specifications to the company for the furniture they want. To cope with the received customer's orders, FFC decides to computerize the order-processing system. The system should accept the values of furniture items, such as a bookshelf and a chair. You need to develop the hierarchy of these items.

Using Interfaces

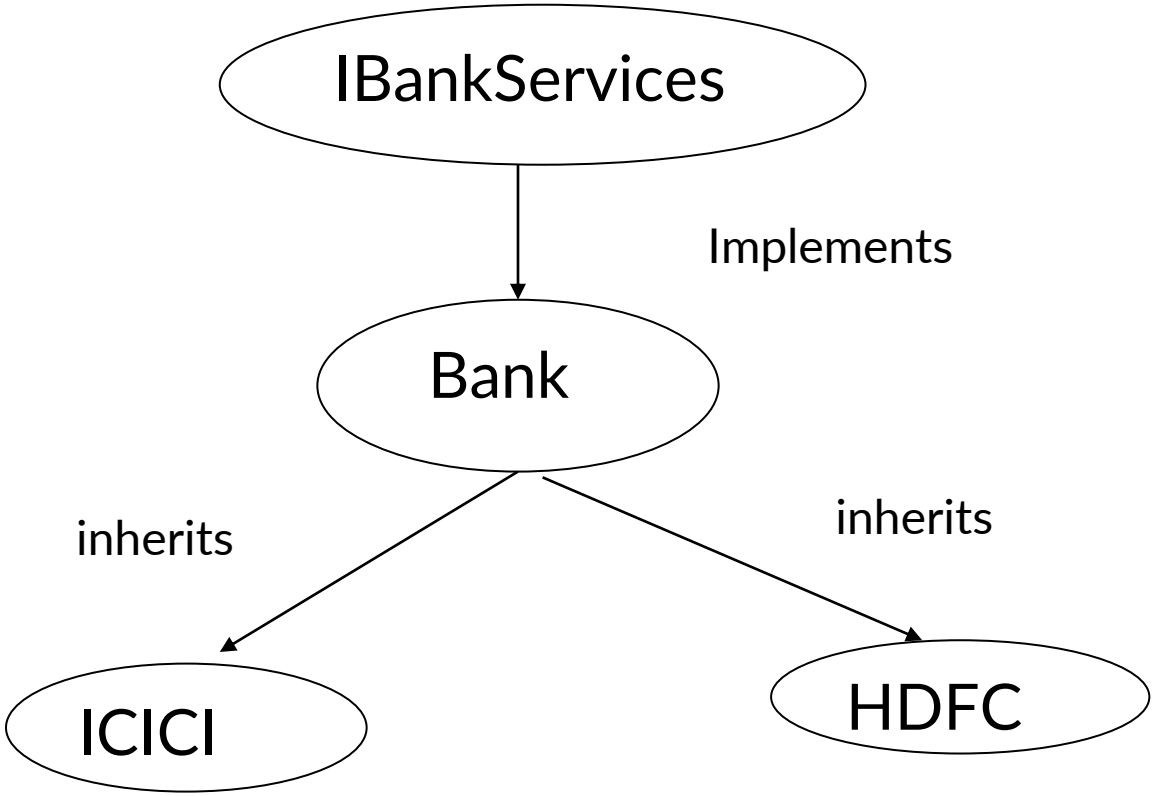
- Interfaces define properties, methods, and events, which are known as the members of the interface.
- Interfaces are fully supported by C#.
- Interfaces are used when a standard structure of methods is to be followed by the classes, and where classes will implement the functionality.
- Interfaces separate the definition of objects from their implementation so that the objects can evolve without the risk of introducing incompatibility in existing applications.

Working with Interfaces

- Working with interfaces includes interface declaration and implementation of interface by the classes.
- You can declare interfaces using the interface keyword.
- Interface statements are public, by default.
- You can declare only methods, functions, and properties in interfaces. You cannot declare a variable in interfaces.
- Interfaces declare methods, which are implemented by classes. A class can inherit from single class but can implement form multiple interfaces.

Inheriting Interfaces

- A class or a structure that implements interfaces also implements the base interfaces of the inherited interface.



Assignment – Abstraction

- Suppose C is a class that implements interfaces A & B. Which of the following will throw an error?

  A a= C;

  B b=(B)a

  C d=(C) a

- You are working in the accounts department of xyz company. You need to compute the price of commodities with respect to their retail and stock price. Design a solution for this requirement.

Polymorphism

- In C# compile time polymorphism mean defining multiple methods with the same name but with different parameters.
- By using compile time polymorphism, we can perform different tasks with the same method name by passing different parameters
- In C#, the run time polymorphism means overriding a base class method in the derived class by creating a similar function
- This can be achieved by using **virtual** & **override** keywords

Thank you

Innovative Services

Passionate Employees

Delighted Customers