Article By:- Durgesh Kumar Singh

(Enterpreneur, Researcher, Developer, Author, Trainer and Speaker)

Object Oriented Programming With Real-World Scenario

Normally every interviewer ask for a real world scenario explaining OOP and many of them fail to answer. This is the reason I am writing this article. This article is mainly intended for the audience who are knowing the Object Oriented Programming (OOP) concept theoretically but are unable to link with real world & programming world.

We write programs to solve our problem and get our work done.

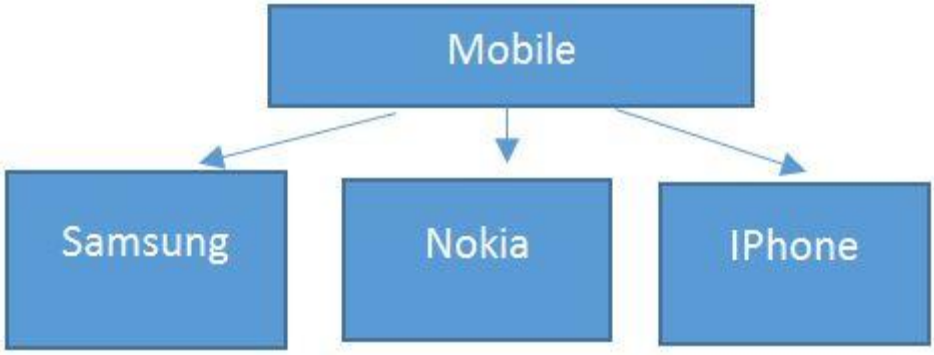
Object Oriented Programming is basically considered as design methodology for creating a non-rigid application. In OOPS every logic is written to get our work done, but this is done based on entity which we call it as Objects. OOP allow us to decompose our problem in to small unit of work which are accessed via Objects. We build function around this objects. There are mainly four pillars (features) of OOP.

If all this features fulfill our programming, then we can say it as perfect Object Oriented Programming.

1. Abstraction
2. Encapsulation
3. Inheritance
4. Polymorphism

Let’s consider an example for explaining each pillar, which at the end will make you understand & follow Object Oriented Programming. Before that, we need to know something

When we take a mobile as an object, its basic functionality for which it was invented were Calling & Receiving a call & Messaging. But now a days thousands of new features & models were added & the count is still increasing.



In above diagram, each brand (Samsung, Nokia, IPhone) have their own list of features along with basic functionality of dialing, receiving a call & messaging.

When we talk about OOP, as the word indicate it will talk about an object (a real world object)

Objects

Any real world entity which can have some characteristics or which can perform some work is called as Object. This object is also called as an instance i.e. - a copy of entity in programming language. If we consider the above example, a mobile manufacturing company, at a time manufactures lacs of pieces of each model which are actually an instance. This objects are differentiated from each other via some identity or its characteristics. This characteristics is given some unique name.

```
1.  Mobile mbl1 = new Mobile ();
2.  Mobile mbl2 = new Mobile ();
```

Class

A Class is a plan which describes the object. We call it as a blue print of how the object should be represented. Mainly a class would consist of a name, attributes & operations. Considering the above example, A Mobile can be a class which has some attributes like Profile Type, IMEI Number, Processor, and some more.) & operations like Dial, Receive & SendMessage.

There are some OOPS principle that need to be looked in while creating any of the class. This principle is called as SOLID where each letter has some specification. I won't be going into this points deeper. A single line of each explanation may clear you with some points.

SRP (The Single Responsibility Principle)
A class should have one, and only one responsibility

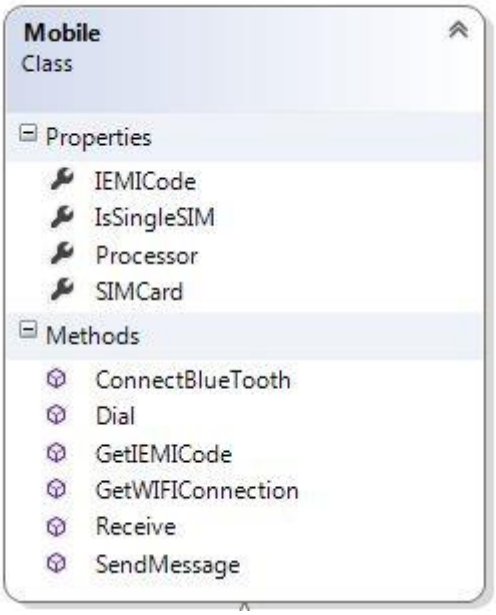
OCP (The Open Closed Principle)
You should be able to extend a classes behavior, without modifying it. (Inheritance)

LSP (The Liskov Substitution Principle)
Derived classes must be substitutable for their base classes. (Polymorphism)

ISP (The Interface Segregation Principle)
Make fine chopped interface instead of huge interface as client cannot be forced to implement an interface which they don't use.

DIP (The Dependency Inversion Principle)
Depend on abstractions, not on concretions. (Abstraction)

Now this class is represented as shown below




```
1.  public class Mobile
2.  {
3.      private string IEMICode { get; set; }
4.      public string SIMCard { get; set; }
5.      public string Processor { get; }
6.      public int InternalMemory { get; }
7.      public bool IsSingleSIM { get; set; }
8.
9.      public void GetIEMICode()
```

```
    {
        Console.WriteLine("IEMI Code - IEDF34343435235");
    }

    public void Dial()
    {
        Console.WriteLine("Dial a number");
    }
    public void Receive()
    {
        Console.WriteLine("Receive a call");
    }
    public virtual void SendMessage()
    {
        Console.WriteLine("Message Sent");
    }
}
```

Abstraction

Abstraction says, only show relevant details and rest all hide it. This is most important pillar in OOPS as it is providing us the technique to hide irrelevant details from User. If we consider an example of any mobile like Nokia, Samsung, IPhone.

Some features of mobiles

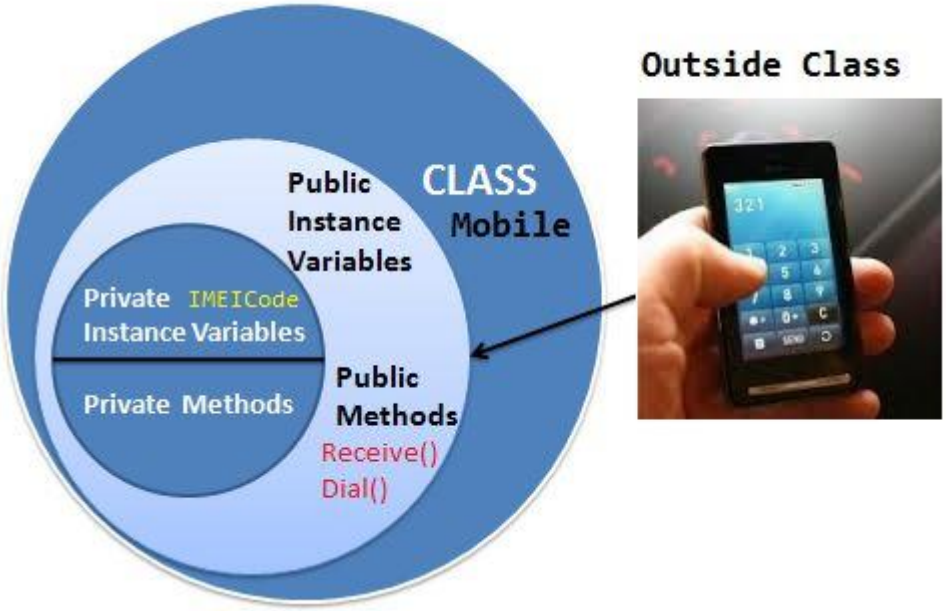
1. Dialing a number call some method internally which concatenate the numbers and displays it on screen but what is it doing we don't know.

2. Clicking on green button actual send signals to calling person's mobile but we are unaware of how it is doing.

This is called abstraction where creating method which is taking some parameter & returning some result after some logic execution without understating what is written within the method

```
public void Dial()
{
    //Write the logic
    Console.WriteLine("Dial a number");
}
```

Encapsulation



Encapsulation is defined as the process of enclosing one or more details from outside world through access right. It says how much access should be given to particular details. Both Abstraction & Encapsulation works hand in hand because Abstraction says what details to be made visible & Encapsulation provides the level of access right to that visible details. i.e. – It implements the desired level of abstraction.

Talking about Bluetooth which we usually have it in our mobile. When we switch on the Bluetooth I am able to connect another mobile but not able to access the other mobile features like dialing a number, accessing inbox etc. This is because, Bluetooth feature is given some level of abstraction.

Another point is when mobile A is connected with mobile B via Bluetooth whereas mobile B is already connected to mobile C then A is not allowed to connect C via B. This is because of accessibility restriction.
This is handled by access specifier like public, private, protected, and internal

```
1.  private string IMEICode = "76567556757656";
```

Polymorphism

Polymorphism can be defined as the ability of doing the same operation but with different type of input.

More precisely we say it as ‘many forms of single entity’. This play a vital role in the concept of OOPS.

Let’s say Samsung mobile have the 5MP camera available i.e. – it is having a functionality of CameraClick(). Now same mobile is having Panorama mode available in camera, so functionality would be same but with mode. This type is said to be Static polymorphism or Compile time polymorphism. See the example below:

```
1.  public class Samsumg : Mobile
2.  {
3.      public void GetWIFIConnection()
```

```
    {
        Console.WriteLine("WIFI connected");
    }

    //This is one mwthod which shows camera functionality
    public void CameraClick()
    {
        Console.WriteLine("Camera clicked");
    }

    //This is one overloaded method which shows camera functionality as well but wi
th its camera's different mode(panaroma)
    public void CameraClick(string CameraMode)
    {
        Console.WriteLine("Camera clicked in " + CameraMode + " Mode");
    }
}
```

Compile time polymorphism the compiler knows which overloaded method it is going to call.

Compiler checks the type and number of parameters passed to the method and decides which method to call and it will give an error if there are no methods that matches the method signature of the method that is called at compile time.

Another point where in SendMessage was intended to send message to single person at a time but suppose Nokia had given provision for sending message to a group at once. i.e. - Overriding the functionality to send message to a group. This type is called Dynamic polymorphism or Runtime polymorphism.

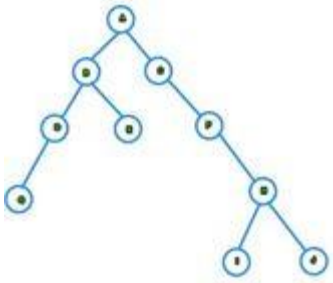
For overriding you need to set the method, which can be overridden to virtual & its new implementation should be decorated with override keyword.

```
public class Nokia : Mobile
{
     public void GetBlueToothConnection()
     {
         Console.WriteLine("Bluetooth connected");
     }

     //New implementation for this method which was available in Mobile Class
     //This is runtime polymorphism
     public override void SendMessage()
     {
         Console.WriteLine("Message Sent to a group");
     }
}
```

By runtime polymorphism, we can point to any derived class from the object of the base class at runtime that shows the ability of runtime binding.

Inheritance



Ability to extend the functionality from base entity in new entity belonging to same group. This will help us to reuse the functionality which is defined before.

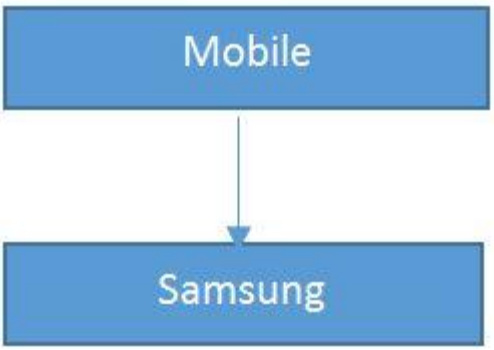
Considering the example, the above figure 1.1 itself shows what is inheritance. Basic Mobile functionality is to Send Message, dial & receive call. So the brands of mobile is using this basic functionality by extending the mobile class functionality and adding their own new features to their respective brand.

There are mainly 4 types of inheritance:

1. Single level inheritance
2. Multi-level inheritance
3. Hierarchical inheritance
4. Hybrid inheritance
5. Multiple inheritance

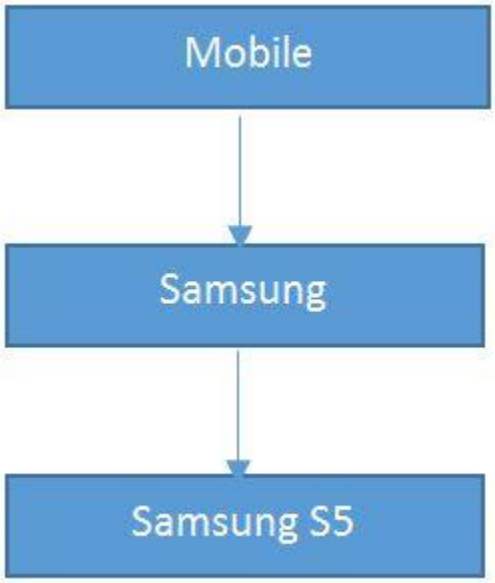
Single level inheritance

In Single level inheritance, there is single base class & a single derived class i.e. - A base mobile features is extended by Samsung brand.



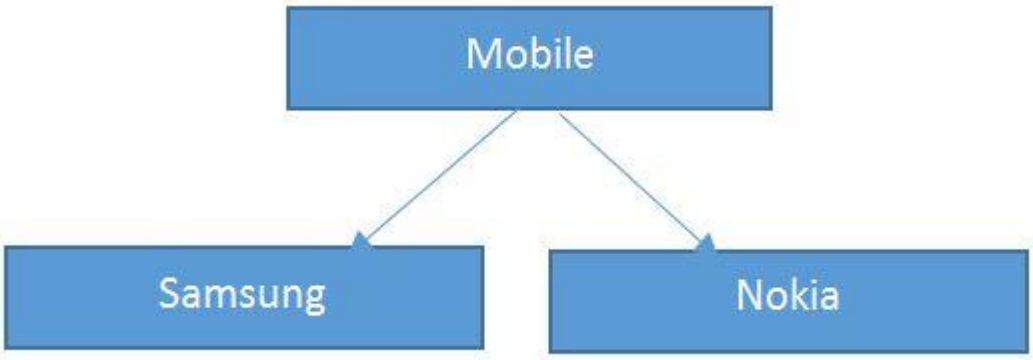
Multilevel inheritance

In Multilevel inheritance, there is more than one single level of derivation. i.e. - After base features are extended by Samsung brand. Now Samsung brand has manufactured its new model with new added features or advanced OS like Android OS, v4.4.2 (kitkat). From generalization, getting into more specification.



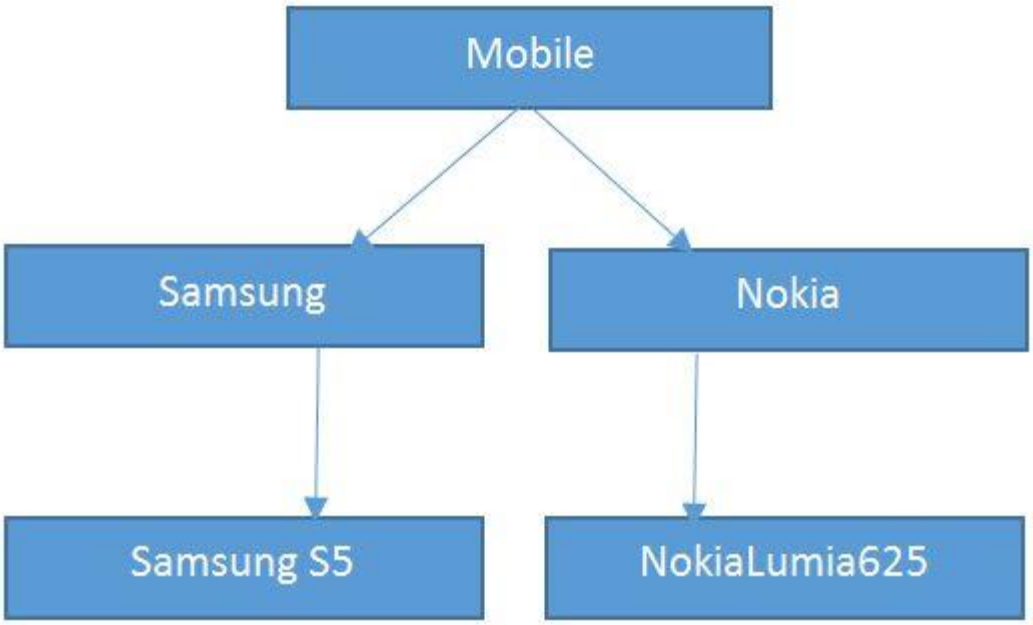
Hierarchal inheritance

In this type of inheritance, multiple derived class would be extended from base class, it's similar to single level inheritance but this time along with Samsung, Nokia is also taking part in inheritance.



Hybrid inheritance

Single, Multilevel, & hierarchal inheritance all together construct a hybrid inheritance.




```
public class Mobile
{
        //Properties
        //Methods
}

public class Samsumg : Mobile
{
        //Properties
        //Methods
}

public class Nokia : Mobile
{
    //Properties
   //Methods
}
```

Interface

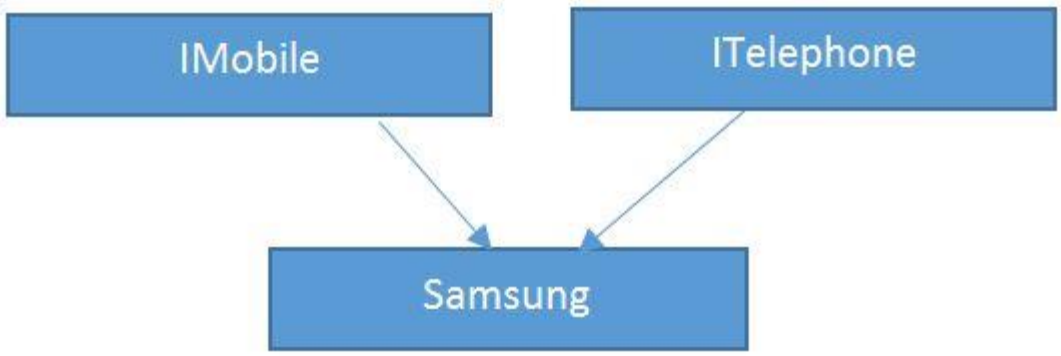
Multiple inheritance where derived class will extend from multiple base classes.

Samsung will use the function of multiple Phone (Mobile & Telephone). This would create a confusion for complier to understand which function to be called when any event in mobile is triggered like Dial () where Dial is available in both the Phone i.e. - (Mobile & Telephone). To avoid this confusion C# came with the concept of interface which is different from multiple inheritance actually.

If we take an interface it is similar to a class but without implementation & only declaration of properties, methods, delegates & events. Interface actually enforces the class to have a standard contract to provide all implementation to the interface members. Then what's is the use of interface when they do not have any implementation? Answer is, they are helpful for having readymade contracts, only we need to implement functionality over this contract.

I mean to say, Dial would remain Dial in case of Mobile or Telephone. It won't be fair if we give different name when its task is to Call the person.

Interface is defined with the keyword ‘interface’ .All properties & methods with in the interface should be implemented if it is been used. That’s the rule of interface.




```
interface IMobile
{
   Void Dial();
}

interface ITelephone
{
   void Dial();

}

public class Mobile : IMobile, ITelephone

{
    public void Dial()
    {
            Console.WriteLine("Dial a number");
    }
}
```

Conclusion

Following the above principle & keeping in mind the four pillars would lead you develop a good program & linking it with real world scenario will make you think more deeply. I hope you like this article. Don’t forget to share your comment whether it’s good or bad. Sharing is valuable no matter what.