JAVA OBJECT ORIENTED PROGRAMMING CONCEPTS

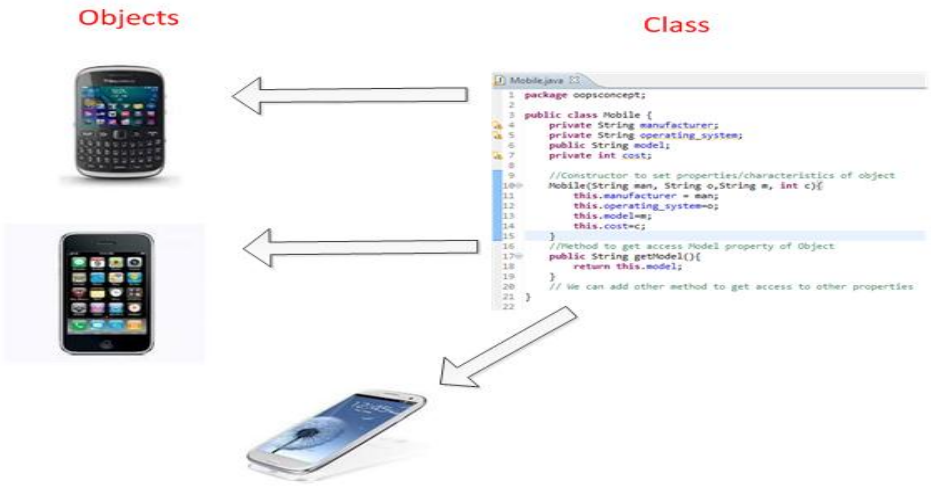
Introduction

This article will help you to understand about Java OOP'S concepts with examples. Let's discuss about what are the features of Object Oriented Programming.

Writing object-oriented programs involves creating classes, creating objects from those classes, and creating applications, which are stand-alone executable programs that use those objects.

A class is a template, blueprint, or contract that defines what an object's data fields and methods will be. An object is an instance of a class. You can create many instances of a class. A Java class uses variables to define data fields and methods to define actions.

Additionally, a class provides methods of a special type, known as constructors, which are invoked to create a new object. A constructor can perform any action, but constructors are designed to perform initializing actions, such as initializing the data fields of objects.



Objects are made up of attributes and methods. Attributes are the characteristics that define an object; the values contained in attributes differentiate objects of the same class from one another. To understand this better let's take example of Mobile as object. Mobile has characteristics like model, manufacturer, cost, operating system etc. So if we create "Samsung" mobile object and "IPhone" mobile object we can distinguish them from characteristics. The values of the attributes of an object are also referred to as the object's state.

There are three main features of OOPS.

1) Encapsulation

2) Inheritance

3) Polymorphism

Let's we discuss about the about features in details.

Encapsulation

Encapsulation means putting together all the variables (instance variables) and the methods into a single unit called Class. It also means hiding data and methods within an Object. Encapsulation provides the security that keeps data and methods safe from inadvertent changes. Programmers sometimes refer to encapsulation as using a “black box,” or a device that you can use without regard to the internal mechanisms. A programmer can access and use the methods and data contained in the black box but cannot change them. Below example shows Mobile class with properties, which can be set once while creating object using constructor arguments. Properties can be accessed using getXXX() methods which are having public access modifiers.

```
1.  package oopsconcept;
2.  public class Mobile {
3.      private String manufacturer;
4.      private String operating_system;
5.      public String model;
6.      private int cost;
7.      //Constructor to set properties/characteristics of object
8.      Mobile(String man, String o,String m, int c){
9.          this.manufacturer = man;
10.         this.operating_system=o;
```

```
11.         this.model=m;
12.         this.cost=c;
13.     }
14.     //Method to get access Model property of Object
15.     public String getModel(){
16.         return this.model;
17.     }
18.     // We can add other method to get access to other properties
19. }
```

Inheritance

An important feature of object-oriented programs is inheritance—the ability to create classes that share the attributes and methods of existing classes, but with more specific features. Inheritance is mainly used for code reusability. So you are making use of already written class and further extending on that. That why we discussed about the code reusability the concept. In general one line definition we can tell that deriving a new class from existing class, it's called as Inheritance. You can look into the following example for inheritance concept. Here we have Mobile class extended by other specific class like Android and Blackberry.

```
1.  package oopsconcept;
2.  public class Android extends Mobile{
3.          //Constructor to set properties/characteristics of object
```

```
4.          Android(String man, String o,String m, int c){
5.                  super(man, o, m, c);
6.              }
7.          //Method to get access Model property of Object
8.          public String getModel(){
9.              return "This is Android Mobile- " + model;
10.         }
11. }
```

```
1.  package oopsconcept;
2.  public class Blackberry extends Mobile{
3.      //Constructor to set properties/characteristics of object
4.      Blackberry(String man, String o,String m, int c){
5.                  super(man, o, m, c);
6.              }
7.      public String getModel(){
8.          return "This is Blackberry-"+ model;
9.      }
10. }
```

Polymorphism

In Core Java Polymorphism is one of easy concept to understand. Polymorphism definition is that Poly means many and morphos means forms. It describes the feature of languages that allows the same word or symbol to be interpreted

correctly in different situations based on the context. There are two types of Polymorphism available in Java. For example, in English the verb “run” means different things if you use it with “a footrace,” a “business,” or “a computer.” You understand the meaning of “run” based on the other words used with it. Object-oriented programs are written so that the methods having same name works differently in different context. Java provides two ways to implement polymorphism.

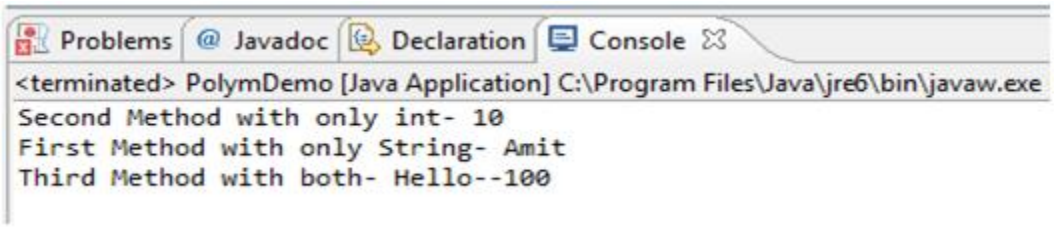
Static Polymorphism (compile time polymorphism/ Method overloading):

The ability to execute different method implementations by altering the argument used with the method name is known as method overloading. In below program we have three print methods each with different arguments. When you properly overload a method, you can call it providing different argument lists, and the appropriate version of the method executes.

```
1.  package oopsconcept;
2.  class Overloadsample {
3.      public void print(String s){
4.          System.out.println("First Method with only String- "+ s);
5.      }
```

```
6.      public void print (int i){
7.          System.out.println("Second Method with only int- "+ i);
8.      }
9.      public void print (String s, int i){
10.         System.out.println("Third Method with both- "+ s + "--" + i);
11.     }
12. }
13. public class PolymDemo {
14.     public static void main(String[] args) {
15.         Overloadsample obj = new Overloadsample();
16.         obj.print(10);
17.         obj.print("Amit");
18.         obj.print("Hello", 100);
19.     }
20. }
```

Output:



Dynamic Polymorphism (run time polymorphism/ Method Overriding)

When you create a subclass by extending an existing class, the new subclass contains data and methods that were defined in the original superclass. In other words, any child class object has all the attributes of its parent. Sometimes, however, the superclass data fields and methods are not entirely appropriate for the subclass objects; in these cases, you want to override the parent class members. Let's take example used in inheritance explanation.

```
1.  package oopsconcept;
2.  public class OverridingDemo {
3.      public static void main(String[] args) {
4.          //Creating Object of SuperClass and calling getModel Method
5.          Mobile m = new Mobile("Nokia", "Win8", "Lumia",10000);
6.          System.out.println(m.getModel());
7.          //Creating Object of Sublcass and calling getModel Method
8.          Android a = new Android("Samsung", "Android", "Grand",30000);
9.          System.out.println(a.getModel());
10.         //Creating Object of Sublcass and calling getModel Method
11.         Blackberry b = new Blackberry("BlackB", "RIM", "Curve",20000);
12.         System.out.println(b.getModel());
13.     }
14. }
```

Abstraction

All programming languages provide abstractions. It can be argued that the complexity of the problems you're able to solve is directly related to the kind and quality of abstraction. An essential element of object-oriented programming is abstraction. Humans manage complexity through abstraction. When you drive your car you do not have to be concerned with the exact internal working of your car (unless you are a mechanic). What you are concerned with is interacting with your car via its interfaces like steering wheel, brake pedal, accelerator pedal etc. Various manufacturers of car has different implementation of car working but its basic interface has not changed (i.e. you still use steering wheel, brake pedal, accelerator pedal etc to interact with your car). Hence the knowledge you have of your car is abstract.

A powerful way to manage abstraction is through the use of hierarchical classifications. This allows you to layer the semantics of complex systems, breaking them into more manageable pieces. From the outside, the car is a single object. Once inside, you see that the car consists of several subsystems: steering, brakes, sound system, seat belts, heating, cellular phone, and so on. In turn, each of these subsystems is made up of more specialized units. For instance, the sound system consists of a radio, a CD player, and/or a tape player. The point is that you

manage the complexity of the car (or any other complex system) through the use of hierarchical abstractions.

An abstract class is something which is incomplete and you can not create instance of abstract class. If you want to use it you need to make it complete or concrete by extending it. A class is called concrete if it does not contain any abstract method and implements all abstract method inherited from abstract class or interface it has implemented or extended. By the way Java has concept of abstract classes, abstract method but a variable can not be abstract in Java.

Lets take an example of Java Abstract Class called Vehicle. When I am creating a class called Vehicle, I know there should be methods like start() and Stop() but don't know start and stop mechanism of every vehicle since they could have different start and stop mechanism e.g some can be started by kick or some can be by pressing buttons.

Advantage of Abstraction is if there is new type of vehicle introduced we might just need to add one class which extends Vehicle Abstract class and implement specific methods. Interface of start and stop method would be same.

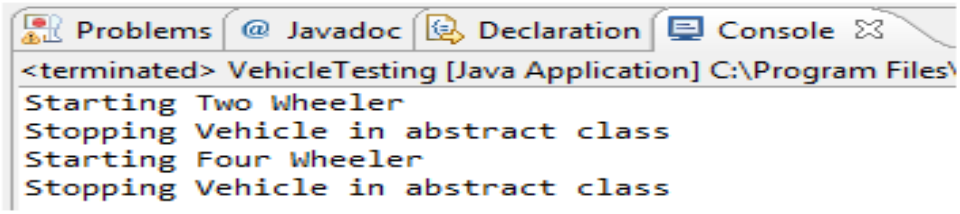
```
1. package oopsconcept;
2. public abstract class VehicleAbstract {
3.     public abstract void start();
4.     public void stop(){
```

```
        System.out.println("Stopping Vehicle in abstract class");
    }
}
class TwoWheeler extends VehicleAbstract{
    @Override
    public void start() {
        System.out.println("Starting Two Wheeler");
    }
}
class FourWheeler extends VehicleAbstract{
    @Override
    public void start() {
        System.out.println("Starting Four Wheeler");
    }
}
```

```
package oopsconcept;
public class VehicleTesting {
    public static void main(String[] args) {
        VehicleAbstract my2Wheeler = new TwoWheeler();
        VehicleAbstract my4Wheeler = new FourWheeler();
        my2Wheeler.start();
        my2Wheeler.stop();
        my4Wheeler.start();
        my4Wheeler.stop();
    }
```

```
11. }
```

Output :



Summary

- An object is an instance of a class.
- Encapsulation provides the security that keeps data and methods safe from inadvertent changes.
- Inheritance is parent-child relationship of class which is mainly used for code reusability.
- Polymorphism definition is that Poly means many and morphos means forms.
- Using abstraction one can simulate real world objects.
- Abstraction provides advantage of code reuse
- Abstraction enables program open for extension.

Source: http://www.w3resource.com/java-tutorial/java-object-oriented-programming.php