Contents

[Encapsulation 2](#_Toc519719268)

[Inheritance 3](#_Toc519719269)

[Abstraction 4](#_Toc519719270)

[Polymorphism 6](#_Toc519719271)

**Exercise 1**

**OOP Principles**

# 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. Two classes can’t have changed each other unless they private. Bundling similar methods and fields together in one class and preventing access outside the class.

package oopsconcept;

public class Mobile {

private String manufacturer;

private String operating\_system;

public String model;

private int cost;

//Constructor to set properties/characteristics of object

Mobile(String man, String o,String m, int c){

this.manufacturer = man;

this.operating\_system=o;

this.model=m;

this.cost=c;

}

//Method to get access Model property of Object

public String getModel(){

return this.model;

}

public String setModel(){

this.model = model;

}

// We can add other method to get access to other properties

}

# Inheritance

Inheritance means a class extends functionality from another class. i.e. dog extends animal.

Inheritance - deriving characteristics from the parent class as a subclass as well as extending these characteristics within its own class.

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. i.e. dog extends animal.

package oopsconcept;

public class Android extends Mobile{

//Constructor to set properties/characteristics of object

Android(String man, String o,String m, int c){

super(man, o, m, c);

}

//Method to get access Model property of Object

public String getModel(){

return "This is Android Mobile- " + model;

}

}

Continue……

package oopsconcept;

public class Blackberry extends Mobile{

//Constructor to set properties/characteristics of object

Blackberry(String man, String o,String m, int c){

super(man, o, m, c);

}

public String getModel(){

return "This is Blackberry-"+ model;

}

}

# Abstraction

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.

Abstraction can take by having abstract classes and interfaces.

Hiding implementation from user while only showing the functionality.

An abstract class is something which is incomplete, and you cannot create an instance of the abstract class.

**Difference between abstract class and interface.**

**Interface**: An interface is a contract: The person writing the interface says, "hey, I accept things looking that way", and the person using the interface says "OK, the class I write looks that way". **An, interface is an empty shell**. There are only the signatures of the methods, which implies that the methods do not have a body. The interface can't do anything. It's just a pattern.

Example as follows:

// I say all motor vehicles should look like this:

interface MotorVehicle

{

void run();

int getFuel();

}

// My team mate complies and writes vehicle looking that way

class Car implements MotorVehicle

{

int fuel;

void run()

{

print("Wrroooooooom");

}

int getFuel()

{

return this.fuel;

}

}

Let's 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 a kick or some can be by pressing buttons.

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

package oopsconcept;

public abstract class VehicleAbstract {

public abstract void start();

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");

}

}

Continue……

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();

}

}

**Second example of Abstraction as follows………….**

Making coffee with a coffee machine is a good example of abstraction. You need to know how to use your coffee machine to make coffee. You need to provide water and coffee beans, switch it on and select the kind of coffee you want to get. The things which you don’t need to know is how the coffee machine is working internally to brew a fresh cup of delicious coffee. You don’t need to know the ideal temperature of the water or the amount of ground coffee you need to use.

# Polymorphism

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 the same name works differently in different context. Java provides two ways to implement polymorphism.

Overloaded methods. The word “run” can be used in many contexts, i.e. to run a business, to run a marathon. This int urn makes the word polymorphic.

Is the ability of an object to take on many forms? Changing the signature of the method. Many forms of data where objects can share the same interface.

**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. Example as follows

package oopsconcept;

class Overloadsample {

public void print(String s){

System.out.println("First Method with only String- "+ s);

}

public void print (int i){

System.out.println("Second Method with only int- "+ i);

}

public void print (String s, int i){

System.out.println("Third Method with both- "+ s + "--" + i);

}

}

public class PolymDemo {

public static void main(String[] args) {

Overloadsample obj = new Overloadsample();

obj.print(10);

obj.print("Amit");

obj.print("Hello", 100);

}

}

**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 the example used in inheritance explanation.

package oopsconcept;

public class OverridingDemo {

public static void main(String[] args) {

//Creating Object of SuperClass and calling getModel Method

Mobile m = new Mobile("Nokia", "Win8", "Lumia",10000);

System.out.println(m.getModel());

//Creating Object of Sublcass and calling getModel Method

Android a = new Android("Samsung", "Android", "Grand",30000);

System.out.println(a.getModel());

//Creating Object of Sublcass and calling getModel Method

Blackberry b = new Blackberry("BlackB", "RIM", "Curve",20000);

System.out.println(b.getModel());

}

}