**Java -** Object Oriented Programming (OOP)

1. **Encapsulation**

Encapsulation is the concept of hiding the internal representation of an object through the use of access modifiers such as public and private. Public modifiers allows visibility anywhere whether it’s in the class or not, whereas private modifiers only allows visibility within its own class. This provides additional security to the data within a class.

**An example of encapsulation would be:**

public class Employee {

private String emp.Name;

private int empAge;

private String empDepartment;

private int empNo;

}

public String getEmpName () {

return empName;

}  
public int getEmpAge () {

return empAge;

public String getEmpDepartment () {  
 return empDepartment;

}

public String getEmpNo () {

return empNo;

}

public void setEmpNo (String newValue) {

empNo = newValue;

}

public void setEmpName (String newValue) {

empName = newValue;

}

public class Encapsulation {

public static void main (String args []) {

Encapsulation obj = new Encapsulation ();

obj.setEmpName (“Jonas”);

obj.setEmpAge (26);

obj.setEmpDepartment (HR);

obj.setEmpNo (62925);

System.out.println (“Employee Name:” + obj.setEmpName ());

System.out.println (“Employee Age:” + obj.setEmpAge ());

System.out.println (“Employee Department:” +obj.setDepartment ());

System.out.println (“Employee Number:” + obj.setDepartment ());

}

}

**Output:**

Employee Name: Jonas

Employee Age: 26

Employee Department: HR

Employee Number: 62925

1. **Inheritance**

Inheritance enables a user to operate the properties of one class with another, either directly or indirectly, by declaring that the class “extends” a class. For example:

public class animal {

//enter code//

class dog extends Animal {

The extension of a base class increases the functionality of object classes whilst allowing base class methods to be overridden.

**An example of Inheritance would be:**

class vehicle {

String model;

String colour;

int speed;

double size;

void characteristics () {

System.out.println (“Model:” + model);

System.out.println (“Colour:” + colour);

System.out.println (“Speed:” + speed);

System.out.println (“Size:” + size);

}

}

class van extends vehicle {

int doors;

int gears;

String fuel;

Void characteristics van () {

System.out.println (“Colour of the van:” + colour);

System.outprintln (“Speed of the van:” + speed);

System.out.println (“Size of the van:” + size);

System.out.println (“No of Doors:” + doors);

System.out.println (“No of Gears:” + gears);

System.out.println (“Type of fuel:” + fuel);

}

}

public class Inheritance {

public static void main (String args []) {

van a1 = new van ();

a1.model = “Volkswagen”;

a1.colour = “White”;

a1.speed = 165;

a1.size = 1.8;

a1.doors = 3;

a1.gears = 6;

a1.fuel = “diesel”;

a1.characteristics van ();

}

}

**Output:**

Model of the van: Volkswagen

Colour of the van: White

Speed of the van: 165

Size of the van: 1.8

No of doors: 3

No of Gears: 6

Type of fuel: Diesel

1. **Polymorphism**

Polymorphism is a concept where an object can be performed in numerous ways. For example any Java object that has the capability to pass more than one IS-A test would be classed as polymorphic. The most common use of polymorphism is when a parent class references a child class object. In Java there are two different types of polymorphism; compile time and real time. Compile time polymorphism is static binding and includes overloading which helps improve the readability to the user, whereas runtime polymorphism is dynamic binding and includes overriding.

**An example of Runtime Polymorphism:**

class animal {

public void whatAmI () {

System,out.println (“I am an animal”);

}

}

class giraffe extends animal {

Public void whatAmI () {

System.out.println (“I am a giraffe”)

}

}

class tiger extends animal {

public void whoAmI () {

System.out.println (“I am a tiger”)

}

}

class panda extends animal

public void whoAmI () {

System.out.println (“I am a Panda”)

}

}

Public static void main (String [] args) {  
animal1 ref1 = new animal ();

animal1 ref2 = new giraffe ();

animal1 ref3 = new tiger ();

animal ref4 = new panda ();

**Output:**

I am an Animal

I am a Giraffe

I am a Tiger

I am a Panda

**An example of Compile time polymorphism:**

class calculation {

void sum (double a, double b) {

System.out.println (a + b)

}

Void sum (int a, int b, int c) {

System.out.println (a + b +c);

}

Public static void main (String [] args) {

Calculation obj = new calculation ();

obj.sum (25, 25, 25)

obj.sum (10.5, 10.5)

}

}

**Output:**

75

21

1. **Abstraction**

Abstraction in Java is the concept of hiding the internal detail i.e. the implementation details of a class, providing visibility only the functionality of that class to the user. The two ways to use abstraction within Java are through the use of “abstract” or “interface. Abstraction can hide anywhere between 0 to 100% abstraction, whereas interface provides 100% abstraction. Abstraction techniques are declared within the parent class and if a method is declared as abstract then the class must be too. Also, abstract classes cannot be instantiated meaning that a user cannot create an object of an abstract class.

**An Example of Abstraction:**

abstract class animal {

abstract void whatAmI ();

}

Class dog extends animal {

void whatAmI () System.out.prinln (“I am a Dog”);

}

Public static void main (String [] args) {

Animal obj = new dog ();

Obj.whatAmI ();

**Output:**

I am a Dog