## Practical 04

## Exercise 01

package com.mycompany.p4;

public class emp

{

private int empid;

private String empname;

private String empdes;

public void setempid(int empid)

{

this.empid=empid;

}

public int getempid()

{

return empid;

}

public void setempname(String empname)

{

this.empname=empname;

}

public String getempname()

{

return empname;

}

public void setempdes(String empdes)

{

this.empdes=empdes;

}

public String getempdes()

{

return empdes;

}

}

package com.mycompany.p4;

public class P4

{

public static void main(String[] args)

{

emp bogdan = new emp();

emp bird = new emp();

bogdan.setempid(123);

bogdan.setempname("Bogdan");

bogdan.setempdes("colombo");

bird.setempid(456);

bird.setempname("Bird");

bird.setempdes("kandy");

System.out.println("ID"+bogdan.getempid());

System.out.println("Name"+bogdan.getempname());

System.out.println("Designation"+bogdan.getempdes());

System.out.println("ID"+bird.getempid());

System.out.println("Name"+bird.getempname());

System.out.println("Designation"+bird.getempdes());

}

}

## Exercise 02

1. Class Super B:

• This class contains an instance variable x of type int and four methods: setIt, increase, triple, and returnIt.

• The setIt method sets the value of x to the provided integer n.

• The increase method increments the value of x by 1.

• The triple method multiplies the value of x by 3.

• The returnIt method returns the value of x.

2. Class Sub:

• This class extends superb, meaning it inherits all the members (fields and methods) from the superb class.

• The triple method in Sub overrides the triple method in superb. Instead of multiplying x by 3, it adds 3 to the current value of x.

• The quadruple method is a new method introduced in the Sub class, which multiplies the value of x by 4.

3. Class TestInheritance:

• This class contains the main method where we test the functionality of both superb and Sub.

• First, we create an instance b of superb, set its value to 2, increase it by 1, and then triple it (resulting in x = 9).

• We print the value of b using System.out.println(b.returnIt()), which prints 9.

• Then, we create an instance c of Sub, set its value to 2, increase it by 1, and then triple it (resulting in x = 9, which is different from what we might expect based on the triple method in superb).

• We print the value of c using System.out.println(c.returnIt()), which prints 9.

4. Method Overriding:

• In Java, method overriding allows a subclass to provide a specific implementation of a method that is already defined in its superclass.

• In this example, the triple method is overridden in the Sub class. When we call c.triple(), it executes the triple method from Sub, not the one from superb.

5. Method Hiding:

• The behaviour we observe in the TestInheritance class is not method hiding. Instead, it's a classic example of method overriding, where the subclass's method implementation takes precedence over the superclass's method implementation.

## Exercise 03

public class Person {

private String name;

private int id;

public void setName(String name) {

this.name = name;

}

public String getName() {

return name;

}

public void setID(int id) {

this.id = id;

}

public int getID() {

return id;

}

}

public class Student extends Person {

private String course;

public void setCourse(String course) {

this.course = course;

}

public String getCourse() {

return course;

}

}

public class Lecturer extends Person {

private String programme;

public void setProg(String programme) {

this.programme = programme;

}

public String getProg() {

return programme;

}

}

public class TestPerson {

public static void main(String[] args) {

// Creating a Student object

Student student = new Student();

student.setName("John Doe");

student.setID(1001);

student.setCourse("Computer Science");

// Creating a Lecturer object

Lecturer lecturer = new Lecturer();

lecturer.setName("Jane Smith");

lecturer.setID(2001);

lecturer.setProg("Software Engineering");

// Displaying student and lecturer information

System.out.println("Student:");

System.out.println("Name: " + student.getName());

System.out.println("ID: " + student.getID());

System.out.println("Course: " + student.getCourse());

System.out.println("\nLecturer:");

System.out.println("Name: " + lecturer.getName());

System.out.println("ID: " + lecturer.getID());

System.out.println("Programme: " + lecturer.getProg());

}

}

## Exercise 04

Animal Class:

* This class is a basic class representing an animal. It doesn't have any specific attributes or behaviours defined in this example.

Mammal Class:

* This class extends the Animal class, making it a subclass of Animal. It inherits all the attributes and behaviours of the Animal class.

Reptile Class:

* This class also extends the Animal class, making it a subclass of Animal. Like Mammal, it inherits all the attributes and behaviours of the Animal class.

Dog Class:

* This class extends the Mammal class, making it a subclass of Mammal. As a result, Dog is also indirectly a subclass of Animal due to the inheritance chain (Dog > Mammal > Animal).
* The main method in the Dog class will be used to test instances and their relationships.

main Method in Dog Class:

* Inside the main method, three objects are created:

1. a: An instance of Animal.
2. m: An instance of Mammal.
3. d: An instance of Dog.

* The first System.out.println statement checks whether m is an instance of Animal. Since Mammal extends Animal, the result will be true.
* The second System.out.println statement checks whether d is an instance of Mammal. As Dog directly extends Mammal, the result will be true.
* The third System.out.println statement checks whether d is an instance of Animal. As mentioned earlier, Dog is indirectly a subclass of Animal through the inheritance chain, so the result will be true.