Object Oriented Programming Lab File

Name: Satvik Raj
SAP Id: 500119624
Batch: 1 (CCVT)
Submitted to: Satyam Tiwari

EXPERIMENT 5: Inheritance & Polymorpishm

1. Write a Java program to demonstrate that a private member of a superclass cannot be accessed directly from a derived class.

```
public class question1Private {
    public static void main(String[] args) {
        SubClass aSubClass = new SubClass();
        aSubClass.displayPrivateNumber();
    }
}

class SuperClass {
    private final int privateNumber = 42;
    public int getPrivateNumber() {
        return privateNumber;
    }
}

class SubClass extends SuperClass {
    public void displayPrivateNumber() {
        System.out.println(getPrivateNumber());
    }
}
```

```
42
```

2. Design a Java class Employee with attributes name, empid, and salary.

- Implement a default constructor, a parameterized constructor, and methods to return the employee's name and salary.
- Add a method increaseSalary(double percentage) to raise the salary by a userspecified percentage.
- Create a subclass Manager with an additional instance variable department.
- Develop a test program to validate these functionalities.

```
public class question2Employee {
    public static void main(String[] args) {
        Employee emp = new Employee("John Doe", 12345, 50000);
        System.out.println("Employee Name: " + emp.getName());
        System.out.println("Employee Salary: " + emp.getSalary());
        emp.increaseSalary(10);
        System.out.println("Employee Salary after increase: " + emp.getSalary());

        Manager mgr = new Manager("Jane Smith", 67890, 75000, "HR");
        System.out.println("Manager Name: " + mgr.getName());
        System.out.println("Manager Salary: " + mgr.getSalary());
        System.out.println("Manager Department: " + mgr.getDepartment());
        mgr.increaseSalary(15);
        System.out.println("Manager Salary after increase: " + mgr.getSalary());
    }
}

class Employee {
    private String name;
    public int empid;
    private double salary;

    public Employee() {
        this.name = "";
        this.empid = 0;
        this.salary = 0.0;
    }

    public Employee(String name, int empid, double salary) {
        this.name = name;
```

```
        this.empid = empid;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

    public double getSalary() {
        return salary;
    }

    public void increaseSalary(double percentage) {
        if (percentage > 0) {
            salary += salary * percentage / 100;
        }
    }
}

class Manager extends Employee {
    private final String department;
    public Manager() {
        super();
        this.department = "";
    }
    public Manager(String name, int empid, double salary, String department) {
        super(name, empid, salary);
        this.department = department;
    }
    public String getDepartment() {
        return department;
    }
}
```

```
Employee Name: John Doe
Employee Salary: 50000.0
Employee Salary after increase: 55000.0
Manager Name: Jane Smith
Manager Salary: 75000.0
Manager Department: HR
Manager Salary after increase: 86250.0
```

3. A university has different types of people associated with it, including staff members and students.

- The base class Person contains common attributes such as name, age, and address.
- The class Staff extends Person and adds attributes like staffId and department.
- Further, a subclass Professor extends Staff by introducing an additional attribute, specialization, and a method conductLecture().
- Similarly, the Student class extends Person and adds studentId and course. Finally, the subclassGraduateStudent extends Student, adding researchTopic and a method submitThesis().
- Implement this university management system in Java using multilevel inheritance andmethodoverriding.
- Demonstrate polymorphism by creating an array of Person objects containing instances of Professor and GraduateStudent, and call their respective methods.

```
class Person {
    String name;
    int age;
    String address;
    public Person(String name, int age, String address) {
        this.name = name;
        this.age = age;
        this.address = address;
    }
    public void displayInfo() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Address: " + address);
    }
}

class Staff extends Person {
    String staffId;
    String department;
  public Staff(String name, int age, String address, String staffId, String
department) {
        super(name, age, address);
        this.staffId = staffId;
        this.department = department;
    }
    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Staff ID: " + staffId);
        System.out.println("Department: " + department);
    }
}

class Professor extends Staff {
    String specialization;
    public Professor(String name, int age, String address, String staffId, String
department, String specialization) {
        super(name, age, address, staffId, department);
        this.specialization = specialization;
    }
  @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Specialization: " + specialization);
    }
    public void conductLecture() {
        System.out.println(name + " is conducting a lecture on " +
specialization);
    }
}
```

```
class Student extends Person {
    String studentId;
    String course;
    public Student(String name, int age, String address, String studentId, String
course) {
        super(name, age, address);
        this.studentId = studentId;
        this.course = course;
    }
    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Student ID: " + studentId);
        System.out.println("Course: " + course);
    }
}

class GraduateStudent extends Student {
    String researchTopic;
    public GraduateStudent(String name, int age, String address, String studentId,
String course, String researchTopic) {
        super(name, age, address, studentId, course);
        this.researchTopic = researchTopic;
    }
    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Research Topic: " + researchTopic);
    }
    public void submitThesis() {
        System.out.println(name + " has submitted a thesis on " + researchTopic);
    }
}

public class question3University {
    public static void main(String[] args) {
        Professor aProfessor = new Professor("Sumit", 45, "123", "P123", "Computer
Science", "Artificial Intelligence");
        GraduateStudent aStudent = new GraduateStudent("Abhishek", 28, "456",
"S456", "Biology", "Genetic Engineering");
        aProfessor.conductLecture();
        aProfessor.displayInfo();
        aStudent.submitThesis();
        aStudent.displayInfo();
    }
}
```

```
Sumit is conducting a lecture on Artificial Intelligence
Name: Sumit
Age: 45
Address: 123
Staff ID: P123
Department: Computer Science
Specialization: Artificial Intelligence
Abhishek has submitted a thesis on Genetic Engineering
Name: Abhishek
Age: 28
Address: 456
Student ID: S456
Course: Biology
Research Topic: Genetic Engineering
```

4. Design an abstract class named Vehicle with abstract methods such as startEngine() and stopEngine(), as well as a non-abstract method serviceInfo() that prints basic servicing instructions.

- Include a couple of protected or private fields (e.g., make, model) and a constructor for Vehicle that initializes those fields.
- Create a concrete class Car that extends Vehicle and provides implementations for startEngine() and stopEngine().
- Within these methods, include print statements or logic that simulates starting and stopping a car engine.
- Finally, instantiate a Car object in the main method, call all available methods (including the inherited non-abstract method), and verify that your abstraction works as intended.

```
abstract class Vehicle {
    private final String make;
    private final String model;

    public Vehicle(String make, String model) {
        this.make = make;
        this.model = model;
    }
    public abstract void startEngine();
    public abstract void stopEngine();
    public void serviceInfo() {
        System.out.println("Basic servicing instructions: Check oil, tire
pressure, and brakes.");
    }
    public String getMake() {
        return make;
    }

    public String getModel() {
        return model;
    }
}

class Car extends Vehicle {
    public Car(String make, String model) {
        super(make, model);}
    @Override
    public void startEngine() {
        System.out.println("The car engine is starting...");}
    @Override
    public void stopEngine() {
        System.out.println("The car engine is stopping...");}}

public class question4Car{
    public static void main(String[] args) {
        Car myCar = new Car("Tesla", "Model S");
        myCar.startEngine();
        myCar.stopEngine();
        myCar.serviceInfo();
        System.out.println("Car make: " + myCar.getMake());
        System.out.println("Car model: " + myCar.getModel());}}
```

```
The car engine is starting...
The car engine is stopping...
Basic servicing instructions: Check oil, tire pressure, and brakes.
Car make: Tesla
Car model: Model S
```

5. Imagine you're designing a restaurant management system where you have a base class called Chef with a method makeSpecialDish().

- This method simply prints a generic message indicating that the chef is making a special dish.
- Next, create three subclasses called ItalianChef, ChineseChef, and MexicanChef—each overrides makeSpecialDish() with a distinct print statement, such as "Making pasta", "Making dumplings", or "Making tacos".
- In your main method, declare an array or list of Chef references, but assign each element to a different subclass (ItalianChef, ChineseChef, MexicanChef).
- Use a loop to call the makeSpecialDish() method on each array element and observe how runtime polymorphism ensures that the correct subclass version of makeSpecialDish() is invoked.

```
class Chef {
    public void makeSpecialDish() {
        System.out.println("The chef is making a special dish.");
    }
}

class ItalianChef extends Chef {
    @Override
    public void makeSpecialDish() {
        System.out.println("Making pasta.");
    }
}

class ChineseChef extends Chef {
    @Override
    public void makeSpecialDish() {
        System.out.println("Making dumplings.");
    }
}

class MexicanChef extends Chef {
    @Override
    public void makeSpecialDish() {
        System.out.println("Making tacos.");
    }
}

public class question5Restaurant {
    public static void main(String[] args) {
        Chef[] chefs = {new ItalianChef(), new ChineseChef(), new MexicanChef()};
        for (Chef chef : chefs) {
            chef.makeSpecialDish();
        }
    }
}
```

```
Making pasta.
Making dumplings.
Making tacos.
```