ASSIGNMENT DAY-6

Name: Vishnu Priya

Enrolment Number: SU625MR009

Batch/Class: MERN stack

Assignment: Day 6

Date of Submission:1 July 2025

Problem Statement:

Object Review: Create a class Smartphone with attributes and methods. Such that

- Attributes: brand, model, batteryLevel
- Methods: turnOn(), turnOff(), charge()

Algorithm:

```
Step 1: Start
Step 2: Define class ObjectReview with
    • brand (String)
    • model (String)
    • batteryLevel (int)
Step 3: Create a constructor that sets brand, model, and batteryLevel
    • Print smartphone details
Step 4: Define method turnOn()
    • If batteryLevel > 0, print "Smartphone is turning on"
    • Else, print "Phone is low on battery. Please plug in"
Step 5: Define method turnOff()
    • Print "Phone is turning off"
Step 6: Define method charge()
    • If batteryLevel < 100
        – Increase batteryLevel by 10
        – If batteryLevel > 100, set to 100
        – Print new battery level
    • Else, print "Battery is already fully charged"
Step 7: End
```

Pseudo Code:

```
Class ObjectReview:
```

```
Private Variables:
    brand: String
    model: String
    batteryLevel: Integer
Constructor(brand, model, batteryLevel):
    Set this.brand = brand
    Set this.model = model
    Set this.batteryLevel = batteryLevel
    Print brand, model, batteryLevel
Method turnOn():
    If batteryLevel > 0:
        Print "Smartphone is turning on"
    Else:
        Print "Phone is low on battery. Please plug in"
Method turnOff():
    Print "Phone is turning off"
Method charge():
    If batteryLevel < 100:
        batteryLevel = batteryLevel + 10
        If batteryLevel > 100:
            batteryLevel = 100
        Print "Charging... Battery level is now: " + batteryLevel + "%"
    Else:
        Print "Battery is already fully charged"
```

Code:

```
package OOPs;

public class ObjectReview {
	private String brand;
	private String model;
	private int batteryLevel;
	public ObjectReview(String brand,String model, int batteryLevel) {
		this.brand=brand;
		this.model=model;
		this.batteryLevel=batteryLevel;
		System.out.println("Brand: "+brand+"\n Model: "+model+" \n Battery 
Level:"+batteryLevel);
	}
	public void turnOn() {
		if(batteryLevel>0) {
			System.out.println("Smartphone is turning on");
		}else {
			System.out.println("Phone is low on battery. Please plug in");
		}
	}
	public void turnOff() {
		System.out.println("Phone is turning off");
	}
	public void charge() {
		if(batteryLevel<100) {
			batteryLevel+=10;
			if(batteryLevel>100) {
				batteryLevel=100;
			}System.out.println("charging.... Battery level is now: "+batteryLevel+"%");
		}else {
			System.out.println("Battery is already fully charged");
		}
	}

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		ObjectReview or= new ObjectReview("Apple","I16",23);
		or.turnOn();
		or.charge();
		or.turnOff();
		ObjectReview or1= new ObjectReview("Samsung","s16",89);
		or1.turnOn();
		or1.charge();
		or1.turnOff();
	}
}
```

Output:

```
Brand: Apple
 Model: I16
 Battery Level:23
Smartphone is turning on
charging.... Battery level is now: 33%
Phone is turning off
Brand: Samsung
```

```
 Model: s16
 Battery Level:89
Smartphone is turning on
charging.... Battery level is now: 99%
Phone is turning off
```

Problem Statement:

Conceptual need for Extension: Create base as employee with attributes such that

o Attributes: name, employeeId, salary

o Behaviors: work(), takeBreak()

And subclass manager, developer

Algorithm:

Step 1: Start the program.

Step 2: Define Employee class with attributes: name, employeeId, salary; constructor to initialize and print them; methods work() and takeBreak().

Step 3: Define Manager class extending Employee with extra attributes: department, teamSize; constructor calls super; method conductMeeting() prints meeting info.

Step 4: Define Developer class extending Employee with extra attributes: programmingLanguage, projects; constructor calls super; method writeCode() prints coding details and projects.

Step 5: In main(), create two Employee objects, call work() and takeBreak() on both.

Step 6: Create a Manager object, set department and team size, call work(), takeBreak(), and conductMeeting().

Step 7: Create a Developer object, set programming language and projects, call work(), takeBreak(), and writeCode().

Step 8: End the program

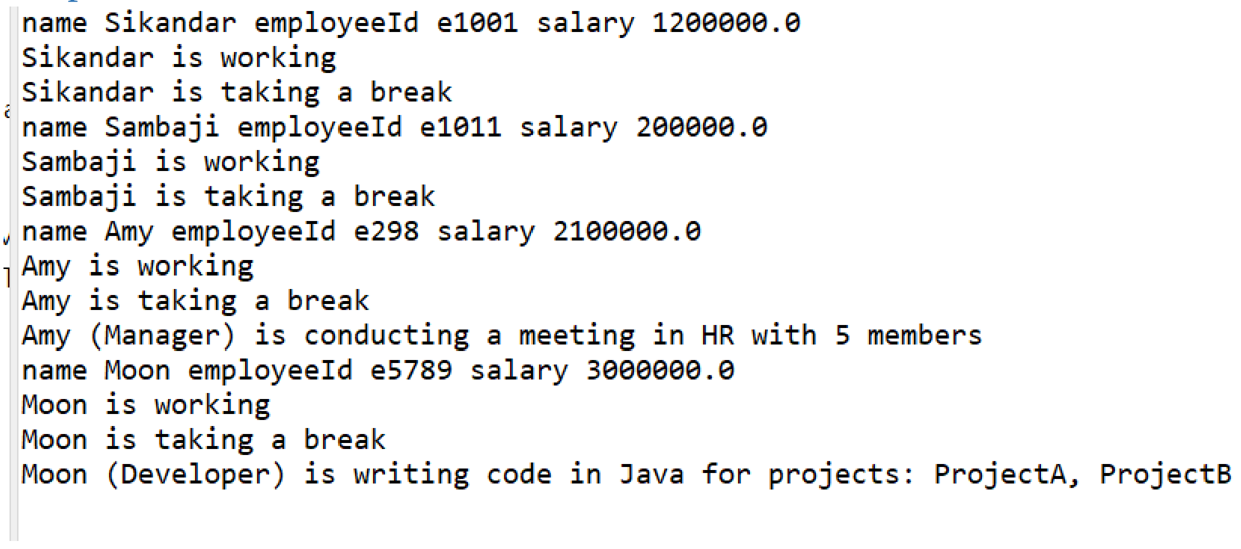
Pseudo Code:

```
CLASS Employee:
    ATTRIBUTES:
        name, employeeId, salary
    METHOD Constructor(name, employeeId, salary):
        SET this.name = name
        SET this.employeeId = employeeId
        SET this.salary = salary
        PRINT name, employeeId, salary
    METHOD work():
        PRINT name + " is working"
    METHOD takeBreak():
        PRINT name + " is taking a break"
CLASS Manager EXTENDS Employee:
    ATTRIBUTES:
        department, teamSize
    METHOD Constructor(name, employeeId, salary):
        CALL super Constructor(name, employeeId, salary)
    METHOD conductMeeting():
        PRINT name + " (Manager) is conducting a meeting in " + department + " with " + teamSize +
" members"
CLASS Developer EXTENDS Employee:
    ATTRIBUTES:
        programmingLanguage, projects
    METHOD Constructor(name, employeeId, salary):
        CALL super Constructor(name, employeeId, salary)
   METHOD writeCode():
        PRINT name + " (Developer) is writing code in " + programmingLanguage
        IF projects NOT EMPTY THEN
            PRINT " for projects: " + list of projects
MAIN PROGRAM:
    CREATE Employee E1("Sikandar", "e1001", 1200000)
    CALL E1.work()
    CALL E1.takeBreak()
    CREATE Employee E2("Sambaji", "e1011", 200000)
    CALL E2.work()
    CALL E2.takeBreak()
    CREATE Manager m("Amy", "e298", 2100000)
    SET m.department = "HR"
    SET m.teamSize = 5
    CALL m.work()
    CALL m.takeBreak()
    CALL m.conductMeeting()
    CREATE Developer D("Moon", "e5789", 3000000)
    SET D.programmingLanguage = "Java"
    SET D.projects = ["ProjectA", "ProjectB"]
    CALL D.work()
    CALL D.takeBreak()
    CALL D.writeCode()
```

Code:

```java
package OOPs;
class Employee {
    String name;
    String employeeId;
    double salary;

    Employee(String name, String employeeId, double salary) {
        this.name = name;
        this.employeeId = employeeId;
        this.salary = salary;
        System.out.println("Employee Name: " + name + "\nId: " + employeeId + "\nSalary: " + 
salary);
    }

    void work() {
        System.out.println(name + " is working.....");
    }

    void takeBreak() {
        System.out.println(name + " is on break!");
    }
}

class Manager extends Employee {
    String dept;
    int teamSize;

    Manager(String name, String employeeId, double salary, String dept, int teamSize) {
        super(name, employeeId, salary);
        this.dept = dept;
        this.teamSize = teamSize;
        System.out.println("Department: " + dept + " | Team Size: " + teamSize);
    }

    void conductMeeting() {
        System.out.println("Manager " + name + " is conducting a meeting for " + dept);
    }
}

class Developer extends Employee {
    private String programmingLanguage;
    private int projects;

    Developer(String name, String employeeId, double salary, String programmingLanguage, int
projects) {
        super(name, employeeId, salary);
        this.programmingLanguage = programmingLanguage;
        this.projects = projects;
    }

    void writeCode() {
        System.out.println(name + " is writing code in " + programmingLanguage + ".");
    }
}

class Company {
    public static void main(String[] args) {
        Employee emp = new Employee("Guru", "FD183", 30000);
        emp.work();
        emp.takeBreak();

        System.out.println();

        Manager mr = new Manager("Ashok", "TD283", 64000, "Technical", 6);
        mr.work();
        mr.takeBreak();
        mr.conductMeeting();

        System.out.println();

        Developer dl = new Developer("Vinny", "DL546", 56000, "Golang", 6);
        dl.work();
        dl.takeBreak();
        dl.writeCode();
    }
}
```

Output:



```
name Sikandar employeeId e1001 salary 1200000.0
Sikandar is working
Sikandar is taking a break
name Sambaji employeeId e1011 salary 200000.0
Sambaji is working
Sambaji is taking a break
name Amy employeeId e298 salary 2100000.0
Amy is working
Amy is taking a break
Amy (Manager) is conducting a meeting in HR with 5 members
name Moon employeeId e5789 salary 3000000.0
Moon is working
Moon is taking a break
Moon (Developer) is writing code in Java for projects: ProjectA, ProjectB
```

Problem Statement:

EmployeeHierarchy: Create a base class with name Employee include the attributes and methods and subclass:

Subclass Manager

Subclass Developer

Algorithm:

Step 1: Start the program.

Step 2: Define the Employee base class with attributes: name, employeeId, and salary.

Step 3: Create a constructor in Employee to initialize attributes.

Step 4: Define getDetails() method in Employee to print all attributes.

Step 5: Define Manager class that extends Employee and adds department attribute.

Step 6: Create constructor in Manager to initialize all attributes including those in Employee.

Step 7: Override getDetails() in Manager to include department details.

Step 8: Define Developer class that extends Employee and adds programmingLanguage.

Step 9: Create constructor in Developer to initialize all attributes including those in Employee.

Step 10: Override getDetails() in Developer to include programming language details.

Step 11: In main(), create a Manager object and call getDetails().

Step 12: Create a Developer object and call getDetails().

Step 13: End the program.

Pseudo Code:

```
CLASS Employee:
    ATTRIBUTES: name, employeeId, salary
   METHOD Constructor(name, employeeId, salary):
        SET this.name = name
        SET this.employeeId = employeeId
        SET this.salary = salary
 METHOD getDetails():
        PRINT "Name: " + name
        PRINT "EmployeeId: " + employeeId
        PRINT "Salary: " + salary
CLASS Manager EXTENDS Employee:
    ATTRIBUTE: department
 METHOD Constructor(name, employeeId, salary, department):
        CALL super(name, employeeId, salary)
        SET this.department = department
METHOD getDetails():
        CALL super.getDetails()
        PRINT "Department: " + department
CLASS Developer EXTENDS Employee:
    ATTRIBUTE: programmingLanguage
METHOD Constructor(name, employeeId, salary, programmingLanguage):
        CALL super(name, employeeId, salary)
        SET this.programmingLanguage = programmingLanguage
METHOD getDetails():
        CALL super.getDetails()
        PRINT "Programming Language: " + programmingLanguage
MAIN PROGRAM:
    CREATE Manager m1 = Manager("Alice", "M1001", 95000, "Sales")
    PRINT "Manager Details"
    CALL m1.getDetails()

    CREATE Developer d1 = Developer("Bob", "D2002", 85000, "Java")

    PRINT "Developer Details"

CALL d1.getDetails()
```

Code:

```java
package Inheritance;
class Employee {
    protected String name;
    protected String employeeId;
    protected double salary;

    public Employee(String name, String employeeId, double salary) {
        this.name = name;
        this.employeeId = employeeId;
        this.salary = salary;
    }

    public String getDetails() {
        return "Name: " + name + "\nID: " + employeeId + "\nSalary: " + salary;
    }
}

class Manager extends Employee {
    private String department;

    public Manager(String name, String employeeId, double salary, String department) {
        super(name, employeeId, salary);
        this.department = department;
    }

    @Override
    public String getDetails() {
        return super.getDetails() + "\nDepartment: " + department;
    }
}

class Developer extends Employee {
    private String programmingLanguage;

    public Developer(String name, String employeeId, double salary, String programmingLanguage) {
        super(name, employeeId, salary);
        this.programmingLanguage = programmingLanguage;
    }

    @Override
    public String getDetails() {
        return super.getDetails() + "\nProgramming Language: " + programmingLanguage;
    }
}

public class EmployeeHierarchy {
    public static void main(String[] args) {
        Employee e1 = new Employee("Vinny", "BE432", 50000);
        System.out.println("Employee Details:\n" + e1.getDetails());
        System.out.println();

        Manager m1 = new Manager("Dia", "RD102", 75000, "Sales");
        System.out.println("Manager Details:\n" + m1.getDetails());
        System.out.println();

        Developer d1 = new Developer("Charlie", "DB103", 65000, "Java");
        System.out.println("Developer Details:\n" + d1.getDetails());
    }
}
```

Output:

```
Employee Details:
Name: Vinny
ID: BE432
Salary: 50000.0

Manager Details:
Name: Dia
ID: RD102
Salary: 75000.0
Department: Sales

Developer Details:
Name: Charlie
ID: DB103
Salary: 65000.0
Programming Language: Java
```

Problem Statement:

Animal Kingdom: With animal as a base class, Create class that override when subclass is called. Subclass: Dog and Cat.

Algorithm:

Step 1: Start the program.

Step 2: Define a base class Animal with a method makeSound() that prints a generic sound.

Step 3: Define a Dog class that extends Animal and overrides makeSound() to print "Woof!".

Step 4: Define a Cat class that extends Animal and overrides makeSound() to print "Meow!".

Step 5: In main(), create an object of Animal and call makeSound().

Step 6: Create an object of Dog and call makeSound().

Step 7: Create an object of Cat and call makeSound().

Step 8: Create Animal references pointing to Dog and Cat objects (demonstrating polymorphism).

Step 9: Call makeSound() on polymorphic objects to show overridden methods are invoked.

Step 10: End the program.

Pseudo Code:

```
CLASS Animal:
   METHOD makeSound():
      PRINT "Some generic animal sound"
CLASS Dog EXTENDS Animal:
   METHOD makeSound():
      PRINT "Woof!"
CLASS Cat EXTENDS Animal:
   METHOD makeSound():
      PRINT "Meow!"
MAIN PROGRAM:
   CREATE Animal generic = new Animal()
   CREATE Dog dog = new Dog()
   CREATE Cat cat = new Cat()
   PRINT "Generic:"
   CALL generic.makeSound()
   PRINT "Dog:"
   CALL dog.makeSound()
   PRINT "Cat:"
   CALL cat.makeSound()
   CREATE Animal aDog = new Dog()
   CREATE Animal aCat = new Cat()
   PRINT "Polymorphic calls:"
```

```
    CALL aDog.makeSound()   // Outputs "Woof!"
    CALL aCat.makeSound()   // Outputs "Meow!"
```

Code:

```java
package Inheritance;
public class AnimalKingdom {

 static class Animal {
     public void makeSound() {
         System.out.println("Some generic animal sound");
     }
 }

 static class Dog extends Animal {
     public void makeSound() {
         System.out.println("Woof!");
     }
 }

 static class Cat extends Animal {
     public void makeSound() {
         System.out.println("Meow!");
     }
 }

 public static void main(String[] args) {
     Animal generic = new Animal();
     Dog dog = new Dog();
     Cat cat = new Cat();

     System.out.println("Generic:");
     generic.makeSound();

     System.out.println("\nDog:");
     dog.makeSound();

     System.out.println("\nCat:");
     cat.makeSound();

     System.out.println("\nPolymorphic calls:");
     dog.makeSound();
     cat.makeSound();
 }
}
```

Output:

```
Generic:
Some generic animal sound

Dog:
Woof!

Cat:
Meow!

Polymorphic calls:
Woof!
Meow!
```

Program Statement:

Payment Gateway: Write a program to pay money

Abstract class: PaymentGateway with abstract processPayment(double amount) Subclasses: CreditCardGateway, PayPalGateway Attempt to instantiate abstract class (should fail)

Algorithm:

Step 1: Start the program.

Step 2: Define an **abstract class** PaymentGateway with an abstract method processPayment(amount).

Step 3: Create a subclass CreditCardGateway that overrides processPayment() to print credit card payment info.

Step 4: Create another subclass PayPalGateway that overrides processPayment() to print PayPal payment info.

Step 5: In the main() method, create a PaymentGateway reference pointing to a CreditCardGateway object.

Step 6: Create another PaymentGateway reference pointing to a PayPalGateway object.

Step 7: Call processPayment(100.00) using the credit card gateway object.

Step 8: Call processPayment(50.25) using the PayPal gateway object.

Step 9: End the program.

Pseudo Code:

```
ABSTRACT CLASS PaymentGateway:
  ABSTRACT METHOD processPayment(amount)
```

```
    CLASS CreditCardGateway EXTENDS PaymentGateway:
    METHOD processPayment(amount):
        PRINT "Processing credit card payment of $" + amount (formatted)
CLASS PayPalGateway EXTENDS PaymentGateway:
    METHOD processPayment(amount):
        PRINT "Processing PayPal payment of $" + amount (formatted)
MAIN PROGRAM:
    DECLARE cc AS PaymentGateway = new CreditCardGateway()
    DECLARE pp AS PaymentGateway = new PayPalGateway()
    CALL cc.processPayment(100.00)
    CALL pp.processPayment(50.25)
```

Code:

```java
package Abstraction;

abstract class PaymentGateway {
 public abstract void processPayment(double amount);
}

class CreditCardGateway extends PaymentGateway {

 public void processPayment(double amount) {
     System.out.printf("Processing credit card payment of $%.2f%n", amount);
 }
}

class PayPalGateway extends PaymentGateway {
 public void processPayment(double amount) {
     System.out.printf("Processing PayPal payment of $%.2f%n", amount);
 }
}
public class PaymentGatewaytest{
 public static void main(String[] args) {

     PaymentGateway cc = new CreditCardGateway();
     PaymentGateway pp = new PayPalGateway();

     // Processing payments
     cc.processPayment(100.00);
     pp.processPayment(50.25);
 }
}
```

Output:

```
Processing credit card payment of $100.00
Processing PayPal payment of $50.25
```

Problem Statement:

Instrument Sound: Write a program creating main class as play( ) and subclasses as guitar and piano

Algorithm:

Step 1: Start the program.

Step 2: Define an **abstract class** Instrument with an abstract method play().

Step 3: Create a class Guitar that extends Instrument and overrides play() to print "Guitar is playing: tintintin".

Step 4: Create a class Piano that extends Instrument and overrides play() to print "Piano is playing: tan tan tan tan".

Step 5: In main(), create an array orchestra of type Instrument with size 2.

Step 6: Store a Guitar object in index 0 of the array.

Step 7: Store a Piano object in index 1 of the array.

Step 8: Loop through the orchestra array and call play() on each instrument.

Step 9: End the program.

Pseudo code:

```
ABSTRACT CLASS Instrument:
   ABSTRACT METHOD play()
CLASS Guitar EXTENDS Instrument:
   METHOD play():
      PRINT "Guitar is playing: tintintin"
CLASS Piano EXTENDS Instrument:
   METHOD play():
      PRINT "Piano is playing: tan tan tan tan"
MAIN PROGRAM:
   CREATE array orchestra[2] of type Instrument
   SET orchestra[0] = new Guitar()
   SET orchestra[1] = new Piano()
 FOR EACH instr IN orchestra:
    CALL instr.play()
```

Code:

```
package Abstraction;
abstract class Instrument {
 public abstract void play();
}

class Guitar extends Instrument {
 @Override
 public void play() {
    System.out.println("Guitar is playing: tintintin");
 }
}

class Piano extends Instrument {
 @Override
 public void play() {
    System.out.println("Piano is playing: tan tan tan tan");
 }
}

public class InstrumentSounds {
 public static void main(String[] args) {
    Instrument[] orchestra = new Instrument[2];
    orchestra[0] = new Guitar();
    orchestra[1] = new Piano();

    for (Instrument instr : orchestra) {
        instr.play();
    }
 }
}
```

Output:

```
Guitar is playing: tintintin
Piano is playing: tan tan tan tan
```

Problem Statement:

Employee Payroll: Write a program with the below

Base: Employee, abstract method calculatePayroll() Subclasses:

SalariedEmployee, HourlyEmployee Implement payroll logic and process

list of employees

Algorithm:

Step 1: Start the program.

Step 2: Define an abstract class Employee with attributes name and employeeId.

Step 3: Add an abstract method calculatePayroll() in Employee.

Step 4: Add a method getDetails() in Employee to return name and ID.

Step 5: Create SalariedEmployee class that extends Employee and adds monthlySalary.

Step 6: Implement calculatePayroll() in SalariedEmployee to return the monthly salary.

Step 7: Create HourlyEmployee class that extends Employee and adds hoursWorked and hourlyRate.

Step 8: Implement calculatePayroll() in HourlyEmployee to return hoursWorked × hourlyRate.

Step 9: In main(), create a list of employees.

Step 10: Add instances of SalariedEmployee and HourlyEmployee to the list.

Step 11: Loop through the list and for each employee, print their details and payroll amount.

Step 12: End the program

Pseudo Code:

```
ABSTRACT CLASS Employee:
    ATTRIBUTES: name, employeeId
    METHOD Constructor(name, employeeId)
    ABSTRACT METHOD calculatePayroll()
    METHOD getDetails():
        RETURN "Name: " + name + ", ID: " + employeeId
CLASS SalariedEmployee EXTENDS Employee:
    ATTRIBUTE: monthlySalary
    METHOD Constructor(name, employeeId, monthlySalary)
    METHOD calculatePayroll():
        RETURN monthlySalary
CLASS HourlyEmployee EXTENDS Employee:
    ATTRIBUTES: hoursWorked, hourlyRate
    METHOD Constructor(name, employeeId, hoursWorked, hourlyRate)
    METHOD calculatePayroll():
        RETURN hoursWorked * hourlyRate
```

```
}

MAIN PROGRAM:
    CREATE List employees
    ADD SalariedEmployee("Alice", "EMP001", 50000) to employees
    ADD HourlyEmployee("Bob", "EMP002", 160, 200) to employees
    ADD SalariedEmployee("Charlie", "EMP003", 60000) to employees
    ADD HourlyEmployee("Diana", "EMP004", 120, 250) to employees
    PRINT "Payroll Report:"
    FOR EACH emp IN employees:
        PRINT emp.getDetails()
        PRINT "Payroll: ₹" + emp.calculatePayroll()
        PRINT newline
```

Code:

```
package Polymorphism;
import java.util.ArrayList;
import java.util.List;
abstract class Employee {
    protected String name;
    protected String employeeId;
    public Employee(String name, String employeeId) {
        this.name = name;
        this.employeeId = employeeId;
    }
    public abstract double calculatePayroll();
    public String getDetails() {
        return "Name: " + name + ", ID: " + employeeId;
    }
}
class SalariedEmployee extends Employee {
    private double monthlySalary;
    public SalariedEmployee(String name, String employeeId, double monthlySalary) {
        super(name, employeeId);
        this.monthlySalary = monthlySalary;
    }
    @Override
    public double calculatePayroll() {
        return monthlySalary;
    }
}
class HourlyEmployee extends Employee {
    private int hoursWorked;
    private double hourlyRate;
    public HourlyEmployee(String name, String employeeId, int hoursWorked, double hourlyRate) {
        super(name, employeeId);
        this.hoursWorked = hoursWorked;
        this.hourlyRate = hourlyRate;
    }
    @Override
    public double calculatePayroll() {
        return hoursWorked * hourlyRate;
    }
}
public class PayrollSystem {
            public static void main(String[] args) {
        List<Employee> employees = new ArrayList<>();
        // Add salaried and hourly employees
        employees.add(new SalariedEmployee("Alice", "EMP001", 50000));
        employees.add(new HourlyEmployee("Bob", "EMP002", 160, 200));  // 160 hours * ₹200/hr
        employees.add(new SalariedEmployee("Charlie", "EMP003", 60000));
        employees.add(new HourlyEmployee("Diana", "EMP004", 120, 250)); // 120 hours * ₹250/hr
        // Process payroll
        System.out.println("Payroll Report:\n-----------------------");
        for (Employee emp : employees) {
            System.out.println(emp.getDetails());
            System.out.println("Payroll: ₹" + emp.calculatePayroll());
            System.out.println();
        }
    }
}
```

Output:

```
Payroll Report:
-----------------------
Name: Alice, ID: EMP001
Payroll: ₹50000.0

Name: Bob, ID: EMP002
Payroll: ₹32000.0

Name: Charlie, ID: EMP003
Payroll: ₹60000.0

Name: Diana, ID: EMP004
Payroll: ₹30000.0
```

Problem Statement:

Geometric Shapes: create a base shape class getArea() that can be called by subclass: Circle, Square that can print the area of shapes

Algorithm:

Pseudo Code:

Code:

```
package Polymorphism;
import java.util.ArrayList;
import java.util.List;
```

```
abstract class Shape {
 public abstract double getArea();
}
class Circle extends Shape {
 private final double radius;
 public Circle(double radius) {
     this.radius = radius;
 }
 public double getArea() {
     return Math.PI * radius * radius;
 }
}
class Square extends Shape {
 private final double side;
 public Square(double side) {
     this.side = side;
 }
 public double getArea() {
     return side * side;
 }
}
public class ShapesTest {
 public static void main(String[] args) {
     List<Shape> shapes = new ArrayList<>();
     shapes.add(new Circle(9));
     shapes.add(new Square(8.3));
     shapes.add(new Circle(7.2));
     double totalArea = 0;
     for (Shape s : shapes) {
         double area = s.getArea();
```

```
            System.out.printf("%s area = %.2f%n",
                s.getClass().getSimpleName(), area);
            totalArea += area;
        }
        System.out.printf("Total area = %.2f%n", totalArea);
    }
}
```

Output:

```
Circle area = 254.47
Square area = 68.89
Circle area = 162.86
Total area = 486.22
```