Bridge course Assignment-6

Name:Balam Indira Priyadarsini

Date:01/07/25

Problem Solving Activity 1.1 : Employee Hierarchy

1. Program Statement: we need to Create the base class Employee with the Attributes: name, employeeid, salary and Method: getDetail()

Subclass Manager: Attribute: department().Override getDetails() to include department

Subclass Developer: Attribute :programmingLanguage. Override getDetail().

2. Algorithm:

Step 1: start program

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

Step 3: Create subclass Manager extending Employee.

Step 4:Add attribute department.Override the getDetail() method to include department info.

Step 5: Create subclass Developer extending Employee.

Step 6: Add attribute programmingLanguage .Override the getDetail() method to include programming language info.

Step 7: In the main method, create instances of Manager and Developer

Step8: Call getDetail() on each instance to display all information.

Step 9:End the program

3. Pseudocode:

Start

Class Employee:

Attributes: name, employeeId, salary

Method: getDetail()

Display name, employeeId, salary

Class Manager inherits Employee:

Attribute: department

Method: getDetail()

Call super.getDetail()

Display department

Class Developer inherits Employee:

Attribute: programmingLanguage

Method: getDetail()

Call super.getDetail()

Display programmingLanguage

Main:

Create Manager object with all attributes

Call getDetail()

Create Developer object with all attributes

Call getDetail()

End

4. Program Code:

Main.java

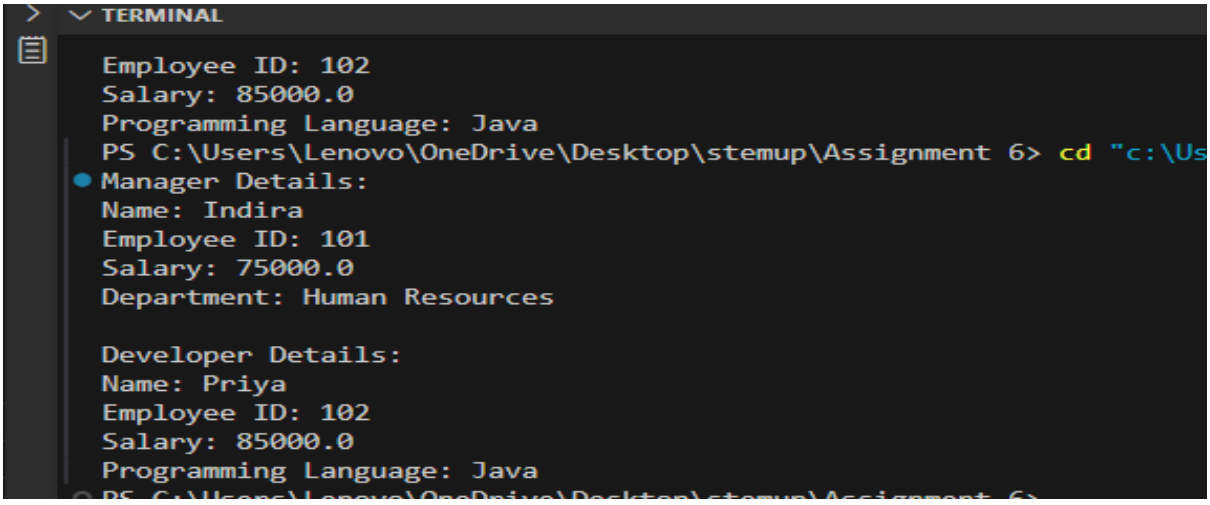
Assignment 6 > Main.java > Employee

```java
class Employee {
    String name;
    int employeeId;
    double salary;
    Employee(String name, int employeeId, double salary) {
        this.name = name;
        this.employeeId = employeeId;
        this.salary = salary;
    }
    void getDetails() {
        System.out.println("Name: " + name);
        System.out.println("Employee ID: " + employeeId);
        System.out.println("Salary: " + salary);
    }
}

class Manager extends Employee {
    String department;
    Manager(String name, int employeeId, double salary, String department) {
        super(name, employeeId, salary);
        this.department = department;
    }

    @Override
    void getDetails() {
        super.getDetails();
        System.out.println("Department: " + department);
    }
}

class Developer extends Employee {
    String programmingLanguage;
    Developer(String name, int employeeId, double salary, String programmingLanguage) {
        super(name, employeeId, salary);
        this.programmingLanguage = programmingLanguage;
    }
    @Override
    void getDetails() {
        super.getDetails();
        System.out.println("Programming Language: " + programmingLanguage);
    }
}

public class Main {
    Run | Debug
    public static void main(String[] args) {
        Manager m = new Manager(name:"Alice", employeeId:101, salary:75000, department:"Human Resources");
        System.out.println(x:"Manager Details:");
        m.getDetails();

        System.out.println();

        Developer d = new Developer(name:"Bob", employeeId:102, salary:85000, programmingLanguage:"Java");
        System.out.println(x:"Developer Details:");
        d.getDetails();
    }
}
```

5. Test Cases

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	Manager: Alice	Manager Details: Name: Alice Employee ID: 101 Salary: 75000.0 Department: Human Resources	Manager Details: Name: Alice Employee ID: 101 Salary: 75000.0 Department: Human Resources	Pass
2	Developer: Bob	Developer Details: Name: Bob Employee ID: 102 Salary: 85000.0 Programming Language: Java	Developer Details: Name: Bob Employee ID: 102 Salary: 85000.0 Programming Language: Java	Pass
3	Manager: Indira	Manager Details: Name: Indira Employee ID: 101 Salary: 75000.0 Department: Human Resources	Manager Details: Name: Indira Employee ID: 101 Salary: 75000.0 Department: Human Resources	Pass

6. Output



```
PS C:\Users\Lenovo\OneDrive\Desktop\ste
Manager Details:
Name: Alice
Employee ID: 101
Salary: 75000.0
Department: Human Resources

Developer Details:
Name: Bob
Employee ID: 102
Salary: 85000.0
Programming Language: Java
```

7. Observation / Reflection

This program demonstrates the concept of inheritance and method overriding in Java using a real-world employee system. The base class Employee contains shared attributes and methods, which are extended by Manager and Developer are the subclasses to include additional role-specific information. By overriding the getDetail() method, each subclass provides its own implementation while still reusing the base functionality via super.getDetail().This structure promotes code reusability**,** readability, and supports the object-oriented principle of polymorphism**.**

Problem Solving Activity 1.2 : Animal Kingdom

1. Program Statement: Need to Create Base class as Animal with methodSound(). And Subclasses as Dog and cat , override the method and need to Create and test objects

2. Algorithm:

Step 1:start program

Step 2: Create a base class Animal with a method sound() that gives a general message.

Step 3: Create subclass Dog and override the sound() method to print "Dog barks".

Step 4: Create subclass Cat and override the sound() method to print "Cat meows".

Step 5 : In the main method, create objects of Dog and Cat .

Step 6: Call the sound() method on each object and observe the output.

Step 7:End the program

3. Pseudocode

```
Start

Class Animal:
    Method sound()
        Print "Animal makes a sound"

Class Dog inherits Animal:
    Method sound()
        Print "Dog barks"

Class Cat inherits Animal:
    Method sound()
        Print "Cat meows"

Main:
    Create object d of Dog
    Create object c of Cat
    d.sound()
    c.sound()

End
```

4. Program Code

Animal.java

Assignment 6 > Animal.java > Animal > main(String[])

```java
class Animal1{
    void sound(){
        System.out.println(x:"Animal makes a sound");
    }
}
class Dog extends Animal1{
    void sound(){
        System.out.println(x:"Dog barks");
    }
}
class Cat extends Animal1{
    @Override
    void sound(){
        System.out.println(x:"Cat meow");
    }
}
public class Animal {
    Run | Debug
    public static void main(String[] args) {
        Dog d=new Dog();
        Cat c=new Cat();
        d.sound();
        c.sound();
    }
}
```

5. Test Cases

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	Dog	Dog barks	Dog barks	Pass
2	Cat	Cat meow	Cat meow	Pass
3	duck	duck quack	duck quack	Pass

6. Output

```
PS C:\Users\Lenovo\OneDrive\De
Dog barks
Cat meow
```

7. Observation / Reflection

This program clearly demonstrates the concept of polymorphism through method overriding in object-oriented programming. The Animal class defines a general sound() method, which is then overridden in its subclasses Dog and Cat to provide specific behavior. When the method is called on objects of Dog and Cat, their own implementations are executed, showcasing dynamic method dispatch.This helps in designing flexible and maintainable code where behavior varies depending on the object, even when accessed through a common interface or base class. It improves understanding of class hierarchy and real-world modeling using inheritance.

Problem Solving Activity 1.3 : Design an Inheritance Tree

1. Program Statement: Create a Base class ElectronicDevice and Subclasses as Television, Laptop, Smartphone List attributes and methods per subclass.

2. Algorithm

Step 1: Start program

Step 2: Create a base class as ElectronicDevice .Inside ElectronicDevice, add attributes: brand, model, price, and powerStatus.

Step 3: Add methods in ElectronicDevice:

- turnOn() -set powerStatus to true.
- turnOff() -set powerStatus to false.
- getDetails() -print brand, model, and price.

Step 4 :Create a subclass as Television that inherits from ElectronicDevice. In Television, add attributes: screenSize, resolution, and isSmart.

Step 5 : Add methods in Television: changeChannel() and adjustVolume()

Step 6 : Create another subclass Laptop that inherits from ElectronicDevice. In Laptop, add attributes as ramSize, storageCapacity, and processor.

Step 7: Add methods in Laptop: compileCode() and runApplication().

Step 8: Create a third subclass Smartphone that inherits from ElectronicDevice. In Smartphone, add attributes: cameraResolution, batteryCapacity, and is5GEnabled.

Step 9: Add methods in Smartphone: makeCall() and installApp(). In the main() method of class DeviceTest1, create an object of Television.

Step 10: Assign values to its attributes and call its methods**.**Create an object of Laptop, assign values, and call its methods.Create an object of Smartphone, assign values, and call its methods.

Step 11: End the program.

3. Pseudocode

Start

Class ElectronicDevice:

Attributes: brand, model, price, powerStatus

Methods: turnOn(), turnOff(), getDetails()

Class Television inherits ElectronicDevice:

Attributes: screenSize, resolution, isSmart

Methods: changeChannel(), adjustVolume()

Class Laptop inherits ElectronicDevice:

Attributes: ramSize, storageCapacity, processor

Methods: compileCode(), runApplication()

Class Smartphone inherits ElectronicDevice:

Attributes: cameraResolution, batteryCapacity, is5GEnabled

Methods: makeCall(), installApp()

End

4. Program Code

DeviceTest1.java

Assignment 6 > DeviceTest1.java > DeviceTest1 > main(String[])

```java
class ElectronicDevice {
    String brand;
    String model;
    double price;
    boolean powerStatus;
    void turnOn() {
        powerStatus=true;
    }
    void turnOff(){
        powerStatus=false;
    }
    void getDetails(){
        System.out.println("Brand:"+brand+",Model:"+model+",Price:"+price);
    }
}
    class Television extends ElectronicDevice{
        int screenSize;
        String resolution;
        Boolean isSmart;
        void changeChannel(int Channel){
            System.out.println("Changing to channel"+Channel);
        }
        void adjustVolume(int level){
            System.out.println("Volume set to "+level);
        }
    }
    class Laptop extends ElectronicDevice{
        int ramSize;
        int storageCapacity;
        String processor;
        void compileCode(){
            System.out.println(x:"Compiling code..");
        }
        void runApplication(String appName){
            System.out.println("Running"+appName);
        }
    }
```

DeviceTest1.java

Assignment 6 > DeviceTest1.java > Laptop > compileCode()

```
37      }
38      class Smartphone extends ElectronicDevice{
39          String cameraResolution;
40          int  batteryCapacity;
41          boolean is5GEnabled;
42          void makeCall(String number){
43              System.out.println("Calling"+number);
44          }
45          void installApp(String appName){
46              System.out.println("Installing"+appName);
47          }
48      }
49  public class DeviceTest1 {
        Run | Debug
50      public static void main(String[] args) {
51          Television tv =new Television();
52          tv.brand = "Samsung";
53          tv.model = "SmartLED";
54          tv.price = 45000;
55          tv.screenSize = 55;
56          tv.resolution = "4K";
57          tv.isSmart = true;
58
59          System.out.println(x:"=== Television Details ===");
60          tv.getDetails();
61          tv.turnOn();
62          tv.changeChannel(Channel:101);
63          tv.adjustVolume(level:20);
64          tv.turnOff();
65
66          Laptop laptop = new Laptop();
67          laptop.brand = "Dell";
68          laptop.model = "XPS 15";
69          laptop.price = 98000;
70          laptop.ramSize = 16;
71          laptop.storageCapacity = 512;
72          laptop.processor = "Intel i7";
```

```
        System.out.println(x:"\n=== Laptop Details ===");
        laptop.getDetails();
        laptop.turnOn();
        laptop.compileCode();
        laptop.runApplication(appName:"Eclipse IDE");
        laptop.turnOff();

        // Smartphone object
        Smartphone phone = new Smartphone();
        phone.brand = "Apple";
        phone.model = "iPhone 15";
        phone.price = 120000;
        phone.cameraResolution = "48MP";
        phone.batteryCapacity = 5000;
        phone.is5GEnabled = true;

        System.out.println(x:"\n=== Smartphone Details ===");
        phone.getDetails();
        phone.turnOn();
        phone.makeCall(number:"9876543210");
        phone.installApp(appName:"WhatsApp");
        phone.turnOff();
    }
}
```

5. Test Cases

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/ Fail)
1	Television	Brand:Samsung,Model:SmartLED,Price:45000.0 Changing to channel101 Volume set to 20	Brand:Samsung,Model:SmartLED,Price:45000.0 Changing to channel101 Volume set to 20	pass
2	Laptop	Brand:Dell,Model:XPS 15,Price:98000.0 Compiling code.. RunningEclipse IDE	Brand:Dell,Model:XPS 15,Price:98000.0 Compiling code.. RunningEclipse IDE	Pass
3	Smartphones	Brand:Apple, Model:iPhone 15,Price:120000.0	Brand: Apple, Model:iPhone 15,Pric	pass

		Calling987654 3210 InstallingWhat sApp	e:1200 00.0 Calling 987654 3210 Installi ngWhat sApp	

6. Output

```
PS C:\Users\Lenovo\OneDrive\Desktop\stemup\Assignment 6>
=== Television Details ===
Brand:Samsung,Model:SmartLED,Price:45000.0
Changing to channel101
Volume set to 20

=== Laptop Details ===
Brand:Dell,Model:XPS 15,Price:98000.0
Compiling code..
RunningEclipse IDE

=== Smartphone Details ===
Brand:Apple,Model:iPhone 15,Price:120000.0
Calling9876543210
InstallingWhatsApp
```

7. Observation / Reflection

This inheritance tree design reflects a real-world hierarchy of electronic devices, showcasing how common properties and behaviors are encapsulated in a base class (ElectronicDevice) and extended with specialized features in each subclass (Television, Laptop, and Smartphone). This approach promotes code reusability, modularity, and polymorphism, making it easier to manage and scale code in object-oriented systems. Each subclass adds unique functionality while reusing the common device behavior from the base class, providing a clean and extensible model for representing electronic

Problem Solving Activity 2.1 : Payment Gateway

1. Program Statement: Abstract class as PaymentGateway with abstract process Payment(double amount) and Subclasses as CreditCardGateway , PayPalGateway Attempt to instantiate abstract class(should fail).

2. Algorithm

Step 1:start program

Step 2: Define an abstract class PaymentGateway. Inside it, declare an abstract method processPayment(double amount).

Step 3: Create a subclass CreditCardGateway that extends PaymentGateway. Override the processPayment() method to print a credit card payment message.

Step 4: Create another subclass PayPalGateway that also extends PaymentGateway. Override processPayment() in PayPalGateway to print a PayPal payment message.

Step 5 :In the main() method:

- create an object of PaymentGateway (it will show compile-time error).
- Create objects of CreditCardGateway and PayPalGateway.
- Call process Payment() on both with example amounts.

Step 6:End the program.

3. Pseudocode

Start

Abstract Class PaymentGateway:

Abstract Method processPayment(amount)

Class CreditCardGateway extends PaymentGateway:

Override processPayment(amount):

Print "Processing credit card payment of ₹<amount>"

Class PayPalGateway extends PaymentGateway:

Override processPayment(amount):

Print "Processing PayPal payment of ₹<amount>"

Main:

Attempt to create object of PaymentGateway → should give error

Create object of CreditCardGateway

Call processPayment(1000)

Create object of PayPalGateway

Call processPayment(1500)

End.

4. Program Code

PaymentTest.java

Assignment 6 > PaymentTest.java > PaymentTest

```java
abstract class PaymentGateway{
    abstract void processPayment(double amount);
}
class CreditCardGateway extends PaymentGateway {
    @Override
    void processPayment(double amount){
        System.out.println("Processing credit card payment of Rs."+amount);
    }
}
class PayPalGateway extends PaymentGateway{
    @Override
    void processPayment(double amount){
        System.out.println("Processing PayPal payment of Rs"+amount);
    }
}

public class PaymentTest {
    Run | Debug
    public static void main(String[] args) {
        PaymentGateway cc=new CreditCardGateway();
        cc.processPayment(amount:1000.00);
        PaymentGateway pp=new PayPalGateway();
        pp.processPayment(amount:1500.00);
    }
}

```

5. Test Cases

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	1000.00	Processing credit card payment of Rs.1000.0	Processing credit card payment of Rs.1000.0	pass
2	1500.00	Processing PayPal payment of Rs1500.0	Processing PayPal payment of Rs1500.0	Pass

6. Output

7. Observation / Reflection

The concept of abstraction in Java using an abstract class PaymentGateway which defines a common interface (processPayment) for all payment types. Abstract classes cannot be instantiated, as shown by the compile-time error when trying to create an object of PaymentGateway. Instead, subclasses like CreditCardGateway and PayPalGateway provide their own specific implementation of processPayment(). This approach follows the OOP principle of abstraction and polymorphism, allowing different payment systems to be treated through a common interface while providing customized behavior.

Problem Solving Activity 2. 2 : Instrument Sounds

1. Program Statement:

Create a Abstract class as Instrument with abstract play() and for the Subclasses as Guitar, Piano Implement and test

2. Algorithm

Step 1:start program

Step 2: Create an abstract class called Instrument. Inside the class, define an abstract method play().

Step 3:Create a class Guitar that extends Instrument. Override the play() method to display: "Strumming the guitar...".

Step 4: Create another class Piano that extends Instrument. Override the play() method to display: "Playing the piano...".

Step 5 :In main(). Create a Guitar object and call its Play() method and also create a Piano object and call its play() method.

Step 6: End the program

3. Pseudocode

Start

Abstract class Instrument:

  Abstract method: play()

Class Guitar extends Instrument:

  Override play()

    Print "Strumming the guitar..."

Class Piano extends Instrument:

  Override play()

    Print "Playing the piano..."

Main method:

  Create object g of Guitar

  Call g.play()

  Create object p of Piano

  Call p.play()

End

4. Program Code

InstrumentTest.java

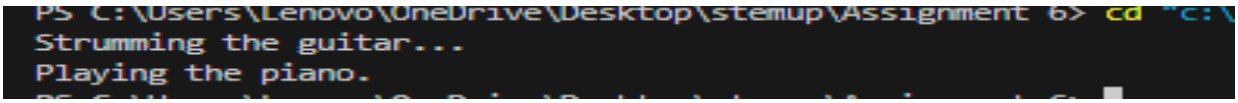
Assignment 6 > InstrumentTest.java > InstrumentTest > main(String[])

```java
abstract class Instrument {
    abstract void play();
}
class Guitar extends Instrument {
    @Override
    void play(){
    System.out.println(x:"Strumming the guitar...");
}
}
class Piano extends Instrument{
    @Override
    void play(){
        System.out.println(x:"Playing the piano.");
    }
}
public class InstrumentTest {
    Run | Debug
    public static void main(String[] args) {
    Instrument g=new  Guitar();
    g.play();
    Instrument p=new Piano();
    p.play();
    }
}

```

5. Test Cases

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	Guitar-class Play()-method	Strumming the guitar...	Strumming the guitar...	Pass
2	Piano-class Play()-method	Playing the piano...	Playing the piano...	Pass

6. Output



7. Observation / Reflection

This program illustrates the use of **abstraction** in Java by defining an abstract class Instrument that provides a common method play() which must be implemented by all instrument types. Both Guitar and Piano override the play() method with behavior specific to each instrument. When we call play() using their respective objects, the output shows the correct message, demonstrating **polymorphism**. Abstract classes help in defining a general template for all subclasses while allowing them to provide their unique implementations.

Problem Solving Activity 2.3 : Abstracting a Task

1. Program Statement:

Create a Base class as AutomatedTask and method to execute() Subclasses: EmailSender , FileArchiver, DatabaseBackup and to Using abstraction to simplify the execution of tasks.

2. Algorithm

Step 1:start program.

Step 2: Define an abstract class AutomatedTask with an abstract method execute().

Step 3 Create a class EmailSender that inherits from AutomatedTask.Override the execute() method to print a message for sending emails.

Step 4: Create a class FileArchiver that inherits from AutomatedTask.Override the execute() method to print a message for archiving files.

Step 5:Create a class DatabaseBackup that inherits from AutomatedTask.Override the execute() method to print a message for backing up databases.

Step 6:In main() need to Create one object each of the three subclasses and call the execute() method for each task.

Step 7:End the program.

3. Pseudocode

```
Start

Abstract class AutomatedTask:
    Abstract method execute()

Class EmailSender extends AutomatedTask:
    Override execute()
        Print "Sending an email..."

Class FileArchiver extends AutomatedTask:
    Override execute()
        Print "Archiving files..."

Class DatabaseBackup extends AutomatedTask:
    Override execute()
        Print "Backing up database..."

Main:
    Create object e of EmailSender
    e.execute()
    Create object f of FileArchiver
    f.execute()
    Create object d of DatabaseBackup
    d.execute()

End
```

4. Program Code

TaskExecutor.java

Assignment 6 > TaskExecutor.java > DatabaseBackup > execute()

```java
abstract class AutomatedTask{
    abstract void execute();
}
class EmailSender extends AutomatedTask{
    @Override
    void execute(){
        System.out.println(x:"Sending an email.");
    }
}
class FileArchiver extends AutomatedTask{
    @Override
    void execute(){
        System.out.println(x:"Archiving files...");
    }
}
class DatabaseBackup extends AutomatedTask {
    @Override
    void execute() {
        System.out.println(x:"Backing up database...");
    }
}
public class TaskExecutor {
    Run | Debug
    public static void main(String[] args) {
        AutomatedTask e=new EmailSender();
        AutomatedTask f=new FileArchiver();
        AutomatedTask d=new DatabaseBackup();
        e.execute();
        f.execute();
        d.execute();
    }
}
```

5. Test Cases

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	EmailSender	Sending an email...	Sending an email...	Pass

2	FileArchiver	Archiving files...0	Archiving files...	Pass
3	DatabaseBackup	Backing up database..	Backing up database..	Pass

6. Output

7. Observation / Reflection

This program uses abstraction and polymorphism to simplify task execution by defining a general abstract class AutomatedTask with a method execute() that each specific task implements differently. Instead of writing different logic in the main program, we treat each task (like sending emails, archiving files, and backing up data) as an object with a common behavior.

Problem Solving Activity 3.1 : Employee Payroll

1. Program Statement:

Create a Base class as Employee ,abstract method as calculatePayroll().Subclasses as

SalariedEmployee , HourlyEmployee and to Implement payroll for the logic and process list of employees.

2. Algorithm

Step 1:start program

Step 2:Create an abstract class Employee with attributes as name, id, and an abstract method as calculatePayroll().

Step 3: Create a class SalariedEmployee that extends Employee. Add attribute as monthlySalary. Override calculatePayroll() to return monthlySalary.

Step 4: Create a class HourlyEmployee that extends Employee. Add attributes as hoursWorked, hourlyRate. Override calculatePayroll() to return hoursWorked × hourlyRate..

Step 5: In main(), create a list of employees. Loop through each employee and display their name and payroll.

Step 6:End the program

3. Pseudocode

start

Abstract class Employee:

attributes: name, id

abstract method: calculatePayroll()

Class SalariedEmployee extends Employee:

attribute: monthlySalary

method calculatePayroll() returns monthlySalary

Class HourlyEmployee extends Employee:

attributes: hoursWorked, hourlyRate

method calculatePayroll() returns hoursWorked * hourlyRate

Main method:

Create 1 salaried employee

Create 2 hourly employees

Store them in a list

Loop and print payroll for each

END

4. Program Code

Payroll.java

Assignment 6 > Payroll.java > SalariedEmployee > monthlySalary

```java
abstract class Employee {
    String name;
    int id;
    Employee(String name,int id){
        this.name=name;
        this.id=id;
    }
    abstract double calculatePayroll();
}
class SalariedEmployee extends Employee {
    double monthlySalary;
    SalariedEmployee(String name,int id,double salary){
        super(name,id);
        this.monthlySalary=salary;
    }
        @Override
        double calculatePayroll(){
        return monthlySalary;
        }
}
class HourlyEmployee extends Employee{
    int hourlyworked;
    double hourlyRate;
    HourlyEmployee(String name,int id,int hours,double rate){
        super(name,id);
        this.hourlyworked=hours;
        this.hourlyRate=rate;
}
    @Override
    double calculatePayroll(){
    return hourlyworked*hourlyRate;
    }
}
```

```java
public class Payroll {
    Run | Debug
    public static void main(String[] args) {
        Employee emp1 = new SalariedEmployee(name:"Alice", id:1, salary:50000);
        Employee emp2 = new HourlyEmployee(name:"Bob", id:2, hours:160, rate:300);
        Employee emp3 = new HourlyEmployee(name:"Charlie", id:3, hours:100, rate:200);

        System.out.println(x:"=== Payroll Report ===");
        System.out.println(emp1.name + " - Rs." + emp1.calculatePayroll());
        System.out.println(emp2.name + " - Rs." + emp2.calculatePayroll());
        System.out.println(emp3.name + " - Rs." + emp3.calculatePayroll());
    }
}
```

5. Test Cases

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	Alice, salary = ₹50000	₹50000.0	₹50000.0	Pass
2	Bob, 160 hrs, ₹300/hr	₹48000.0	₹48000.0	Pass
3	Charlie, 100 hrs, ₹200/hr	₹20000.0	₹20000.0	Pass

6. Output

```
PS C:\Users\Lenovo\OneDrive\Desktop\stemup\Assignm
=== Payroll Report ===
Alice - Rs.50000.0
Bob - Rs.48000.0
Charlie - Rs.20000.0
PS C:\Users\Lenovo\OneDrive\Desktop\stemup\Assignm
```

7. Observation / Reflection

This program shows how abstraction and inheritance help us design a flexible payroll system. The Employee abstract class defines a common method calculatePayroll() which is implemented differently in each subclass. SalariedEmployee returns a fixed salary, while HourlyEmployee calculates pay based on hours. By using polymorphism (common Employee type), we can easily manage a list of different employees.

Problem Solving Activity 3.2 :Geometric Shapes

1. Program Statement:

Create a Abstract base class as Shape with getArea() and Subclasses as Circle ,Square Create polymorphic list and calculate areas

2. Algorithm

Step 1:start program

Step 2:Create an abstract class Shape with abstract method getArea().Create Circle class for the Attributes as radius and formula is getArea() = 3.14 × radius × radius

Step 3: Create Square class and Attribute as side. For the formula getArea() = side × side

Step 4: In main() method:

- Create a list of shapes using base class Shape
- Add Circle and Square objects to the list
- Loop through the list and print the area of each shape

Step 5: End the program

3. Pseudocode

Start

Abstract class Shape:

abstract method getArea()

Class Circle extends Shape:

attribute: radius

method getArea() returns 3.14 * radius * radius

Class Square extends Shape:

attribute: side

method getArea() returns side * side

Main:

Create list of Shape objects

Add Circle and Square to list

For each shape in list:

Print area

End

4. Program Code

ShapeTest.java

Assignment 6 > ShapeTest.java > Square

```
1   abstract class Shape{
2       abstract double getArea();
3   }
4   class Circle extends Shape{
5       double radius;
6       Circle(double radius){
7           this.radius=radius;
8       }
9   @Override
10  double getArea(){
11      return 3.14*radius*radius;
12  }
13  }
14  class Square extends Shape{
15      double side;
16      Square(double side){
17          this.side=side;
18      }
19      @Override
20      double getArea(){
21          return side*side;
22      }
23  }
```

```java
}
public class ShapeTest {
    Run | Debug
    public static void main(String[] args) {
        Shape[] shapes=new Shape[2];
        shapes[0]=new Circle(radius:5);
        shapes[1]=new Square(side:4);
        System.out.println(x:"===shape Areas ===");
        for(Shape s:shapes){
            System.out.println("Area:"+s.getArea());
        }
    }
}
```

5. Test Cases

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	Circle=5,	Area:78.5	Area:78.5	Pass
2	square=4	Area:16.0	Area:16.0	Pass
3	Circle=6	Area:113.03999999999999	Area:113.03999999999999	pass

6. Output

```
===shape Areas ===
Area:78.5
Area:16.0
```

7. Observation / Reflection

This program demonstrates polymorphism by calling the same method draw() on different tool types. Even though all tools are stored in a common type (Tool), each subclass (PenTool, EraserTool, LineTool) overrides draw() to provide specific behavior.

Problem Solving Activity 3.3 :Polymorphism in UI

1. Program Statement:

Create a Base class as Tool and method as draw() for the Subclasses as PenTool , EraserTool , LineTool and need to Demonstrate polymorphism using a collection.

2. Algorithm

Step 1: start program

Step 2 : Create a base class Tool with a method draw().Create subclasses PenTool, EraserTool, and LineTool.

Step 3: Each subclass should override the draw() method to print its own action.

Step 4: In main . Create an array of type Tool .Add different tool objects to the array. Loop through the array and call the draw() method for each object.

Step 5: End the program

3. Pseudocode

Start

Class Tool:

Method: draw()

Class PenTool extends Tool:

For draw() method print "Drawing with Pen Tool"

Class EraserTool extends Tool:

For draw() method print "Erasing with Eraser Tool"

Class LineTool extends Tool:

for draw() method print "Drawing a Line with Line Tool"

Main:

Create array of Tool

Add PenTool, EraserTool, LineTool to array

For each tool in array:

Call draw()

End

4. Program Code

UIToolsExample.java

Assignment 6 > UIToolsExample.java > ...

```java
class Tool{
    void draw(){
        System.out.println(x:"Using a tool");
    }
}
class PenTool extends Tool{
    @Override
    void draw(){
        System.out.println(x:"Drawing with Pen Tool");
    }
}
class EraserTool extends Tool{
    @Override
    void draw(){
        System.out.println(x:"Erasing with Eraser Tool");
    }
}
class LineTool extends Tool{
    @Override
    void draw(){
        System.out.println(x:"Drawing a Line with Line Tool");
    }
}
```

```
24
25  public class UIToolsExample {
        Run | Debug
26      public static void main(String[] args) {
27          Tool[] tools=new Tool[3];
28          tools[0]=new PenTool();
29          tools[1]=new EraserTool();
30          tools[2]=new LineTool();
31          System.out.println(x:"===Drawing Actions===");
32          for (Tool tool:tools){
33           tool.draw();
34          }
35      }
36  }
```

5. Test Cases

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	PenTool	Drawing with Pen Tool	Drawing with Pen Tool	Pass
2	EraserTool	Erasing with Eraser Tool	Erasing with Eraser Tool	Pass
3	LineTool	Drawing a Line with Line Tool	Drawing a Line with Line Tool	pass

6. Output

```
===Drawing Actions===
Drawing with Pen Tool
Erasing with Eraser Tool
Drawing a Line with Line Tool
```

7. Observation / Reflection

This program is a clear example of polymorphism in Java. We used a base class Tool and stored different tool objects (PenTool, EraserTool, LineTool) in a common array. When we called the draw() method, each object printed its own message and this is called as runtime polymorphism.