**Learner Assignment Submission Format**

**Learner Details**

* **Name: Nithya B A**
* **Enrollment Number:SU625MR010**
* **Batch / Class:Mern stack**
* **Assignment: (Bridge Course Day 6)**
* **Date of Submission:1/7/2025**

**Problem Solving Activity 6.1**

1. **Program Statement**

To write a program with the following

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

And to print turnOn or turnOff by comparing the charge

1. **Algorithm**

**Step 1:** Start the program.

**Step 2:** Define class SmartP with attributes: brand, model, batteryLevel.

**Step 3:** Create a constructor to initialize brand, model, and batteryLevel, and print them.

**Step 4:** Define method turnOn() to check battery and print if phone can turn on.

**Step 5:** Define method turnoff()to print "Turned off" if battery is 0 or less.

**Step 6:** Define method charge() to print current battery

**Step 7:**End

1. **Pseudocode**

CLASS SmartP:

DECLARE brand AS String

DECLARE model AS String

DECLARE batteryLevel AS Integer

METHOD 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 THEN

PRINT "Turned on"

ELSE

PRINT "Please charge the phone"

METHOD turnOff():

IF batteryLevel <= 0 THEN

PRINT "Turned off"

METHOD charge():

PRINT "charge is " + batteryLevel

MAIN PROGRAM:

CREATE phone1 AS SmartP("Apple", "i15", 0)

CALL phone1.turnOn()

CALL phone1.turnOff()

CALL phone1.charge()

CREATE phone2 AS SmartP("Samsung", "S25 ultra", 95)

CALL phone2.turnOn()

CALL phone2.turnOff()

CALL phone2.charge()

**4.Program code**

package Day6;

public class SmartP {

String brand;

String model;

int batteryLevel;

SmartP(String brand,String model,int batteryLevel){

this.brand=brand;

this.model=model;

this.batteryLevel=batteryLevel;

System.*out*.println("brand: "+brand+" model: "+model+" batterylevel: "+batteryLevel);

}

void turnOn() {

if (batteryLevel>0){

System.*out*.println("Turned on");

}else {

System.*out*.println("Please charge the phone");

}

}

void turnOff() {

if(batteryLevel<=0)

System.*out*.println("Turned off");

}

void charge() {

System.*out*.println("charge is "+batteryLevel);

}

public static void main(String[]args) {

SmartP phone1=new SmartP("Apple","i15",0);

phone1.turnOn();

phone1.turnOff();

phone1.charge();

SmartP phone2=new SmartP("Samsung","S25 ultra",95);

phone2.turnOn();

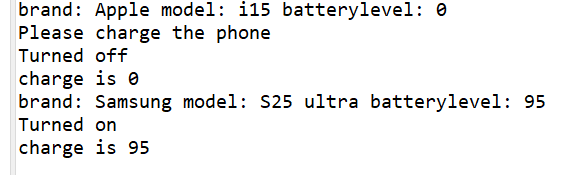
phone2.turnOff();

phone2.charge();

}

}

**5.Screenshots of Output**

****

**6. Observation / Reflection**

Used if loop with methods and got the output

**Problem Solving Activity 6.2**

1. **Program Statement**

**Write a program creating base class as employee and other classes as manager and developer, inherit the properties of base class.**

1. **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

1. **Pseudocode**

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()

1. **Program Code**

**package** Day6;

**public** **class** Company {

**static** **class** Employee {

String name;

String employeeId;

**float** salary;

**public** Employee(String name, String employeeId, **float** salary) {

**this**.name = name;

**this**.employeeId = employeeId;

**this**.salary = salary;

System.***out***.println("name " + name + " employeeId " + employeeId + " salary " + salary);

}

**public** **void** work() {

System.***out***.println(name + " is working");

}

**public** **void** takeBreak() {

System.***out***.println(name + " is taking a break");

}

}

**static** **class** Manager **extends** Employee {

String department;

**int** teamSize;

**public** Manager(String name, String employeeId, **float** salary) {

**super**(name, employeeId, salary);

}

**public** **void** conductMeeting() {

System.***out***.println(name + " (Manager) is conducting a meeting in " + department + " with " + teamSize + " members");

}

}

**static** **class** Developer **extends** Employee {

String programmingLanguage;

String[] projects;

**public** Developer(String name, String employeeId, **float** salary) {

**super**(name, employeeId, salary);

}

**public** **void** writeCode() {

System.***out***.print(name + " (Developer) is writing code in " + programmingLanguage);

**if** (projects != **null** && projects.length > 0) {

System.***out***.print(" for projects: ");

**for** (**int** i = 0; i < projects.length; i++) {

System.***out***.print(projects[i] + (i < projects.length - 1 ? ", " : ""));

}

}

System.***out***.println();

}

}

**public** **static** **void** main(String[] args) {

Employee E1 = **new** Employee("Sikandar", "e1001", 1\_200\_000f);

E1.work();

E1.takeBreak();

Employee E2 = **new** Employee("Sambaji", "e1011", 200\_000f);

E2.work();

E2.takeBreak();

Manager m = **new** Manager("Amy", "e298", 2\_100\_000f);

m.department = "HR";

m.teamSize = 5;

m.work();

m.takeBreak();

m.conductMeeting();

Developer D = **new** Developer("Moon", "e5789", 3\_000\_000f);

D.programmingLanguage = "Java";

D.projects = **new** String[] {"ProjectA", "ProjectB"};

D.work();

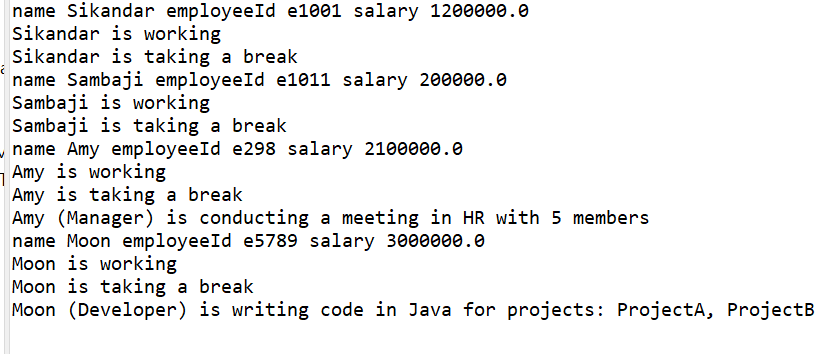
D.takeBreak();

D.writeCode();

}

}

**5.Screenshots of Output**

****

**7. Observation / Reflection**

Used inheritance concept and derived base class and inherited its properties to subclass

**Problem Solving Activity 1.1**

1. **Program Statement**

**Write a program creating a base class as Employee and subclasses as Manager and Developer and by using concept of inheritance acquire the base class attributes and print the details**

1. **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.

1. **Pseudocode**

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()

1. **Program Code**

package Day6;

public class Hierarchy {

static class Employee {

String name, employeeId;

float salary;

public Employee(String name, String employeeId, float salary) {

this.name = name;

this.employeeId = employeeId;

this.salary = salary;

}

public void getDetails() {

System.*out*.println("Name: " + name);

System.*out*.println("EmployeeId: " + employeeId);

System.*out*.println("Salary: " + salary);

}

}

static class Manager extends Employee {

String department;

public Manager(String name, String employeeId, float salary, String department) {

super(name, employeeId, salary);

this.department = department;

}

public void getDetails() {

super.getDetails();

System.*out*.println("Department: " + department);

}

}

static class Developer extends Employee {

String programmingLanguage;

public Developer(String name, String employeeId, float salary, String programmingLanguage) {

super(name, employeeId, salary);

this.programmingLanguage = programmingLanguage;

}

public void getDetails() {

super.getDetails();

System.*out*.println("Programming Language: " + programmingLanguage);

}

}

public static void main(String[] args) {

Manager m1 = new Manager("Alice", "M1001", 95000f, "Sales");

Developer d1 = new Developer("Bob", "D2002", 85000f, "Java");

System.*out*.println("Manager Details");

m1.getDetails();

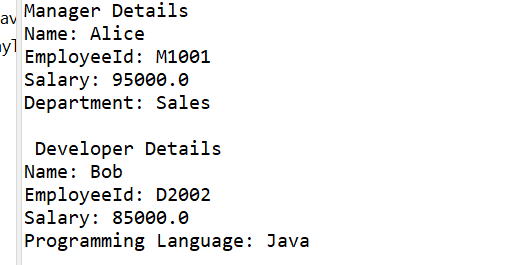
System.*out*.println("\n Developer Details");

d1.getDetails();

}

}

**5.Screenshots of Output**

****

**7. Observation / Reflection**

Here the child class inherits the properties of parent class and will be called by same method

**Problem Solving Activity 1.2**

1. **Program Statement**

**Write a program for Animal kingdom**

**Base class: Animal with method makeSound() Subclasses: Dog and Cat,**

**override the method Create and test objects**

1. **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.

1. **Pseudocode**

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!"

1. **Program Code**

**package** Day6;

**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("Dog:");

dog.makeSound();

System.***out***.println("Cat:");

cat.makeSound();

Animal aDog = **new** Dog();

Animal aCat = **new** Cat();

System.***out***.println("\nPolymorphic calls:");

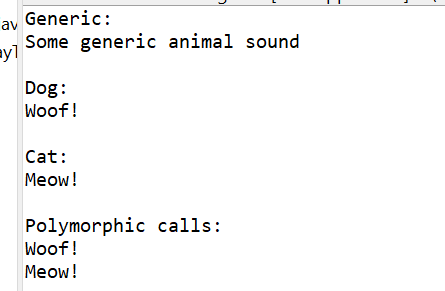
aDog.makeSound();

aCat.makeSound();

}

}

**5.Screenshots of Output**

****

**7. Observation / Reflection**

Adopted inheritance concept and also used polymorphism of method for function call

**Problem Solving Activity 2.2**

1. **Program Statement**

**Write a program to pay money**

**Abstract class: PaymentGateway with abstract processPayment(double**

**amount) Subclasses: CreditCardGateway, PayPalGateway Attempt to**

**instantiate abstract class (should fail)**

1. **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.

1. **Pseudocode**

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)

1. **Program Code**

**package** Day6;

**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();

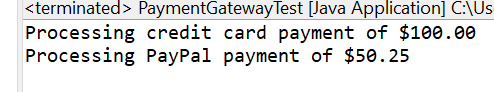
cc.processPayment(100.00);

pp.processPayment(50.25);

}

}

**5.Screenshots of Output**

****

**7. Observation / Reflection**

Written program using abstraction where the actual details are not visible but the required information will be shared

**Problem Solving Activity 3.1**

1. **Program Statement**

**Write a program creating main class as play( ) and subclasses as guitar and piano**

1. **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.

1. **Pseudocode**

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()

1. **Program Code**

**package** Day6;

**abstract** **class** Instrument {

**public** **abstract** **void** play();

}

**class** Guitar **extends** Instrument {

**public** **void** play() {

System.***out***.println("Guitar is playing: tintintin");

}

}

**class** Piano **extends** Instrument {

**public** **void** play() {

System.***out***.println("Piano is playing: tan tan tan tan");

}

}

**public** **class** InstrumentTest {

**public** **static** **void** main(String[] args) {

Instrument[] orchestra = **new** Instrument[2];

orchestra[0] = **new** Guitar();

orchestra[1] = **new** Piano();

**for** (Instrument instr : orchestra) {

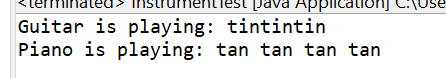
instr.play();

}

}

}

**5.Screenshots of Output**

****

**7. Observation / Reflection**

Understood the concept of polymorphism by using same methods

**Problem Solving Activity 3.2**

1. **Program Statement**

**Write a program with the below**

**Base: Employee, abstract method calculatePayroll() Subclasses:**

**SalariedEmployee, HourlyEmployee Implement payroll logic and process**

**list of employees**

1. **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

1. **Pseudocode**

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

1. **Program 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;

}

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;

}

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

System.out.println("Payroll Report:\n-----------------------");

for (Employee emp : employees) {

System.out.println(emp.getDetails());

System.out.println("Payroll: ₹" + emp.calculatePayroll());

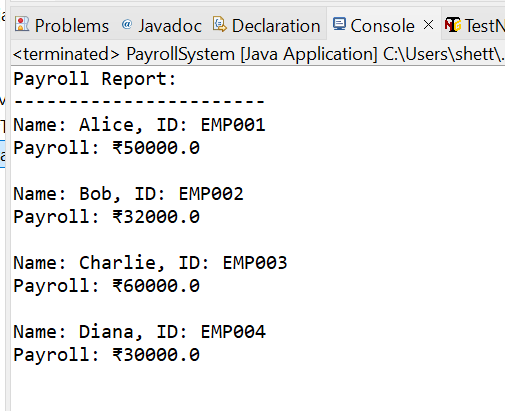
System.out.println();

       }

    }

}

**5.Screenshots of Output**

****

**6. Observation / Reflection:** Used function call to calculate the power of a number and understood the concept by using math built in operations

**Problem Solving Activity 2.3**

1. **Program Statement**

**Write a program to create a AutomatedTask and subclass to simplify the execution of tasks**

1. **Algorithm**

Step 1: Start the program.

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

Step 3: In execute(), call setup(), performTask(), teardown(), and print completion message.

Step 4: Define setup() to print setup message using getTaskName().

Step 5: Declare abstract methods performTask() and getTaskName() to be implemented by subclasses.

Step 6: Define teardown() to print teardown message using getTaskName().

Step 7: Create EmailSender class extending AutomatedTask and implement performTask() and getTaskName().

Step 8: Create FileArchiver class extending AutomatedTask and implement required methods.

Step 9: Create DatabaseBackup class extending AutomatedTask and implement required methods.

Step 10: In main(), create an array of tasks with EmailSender, FileArchiver, and DatabaseBackup.

Step 11: Loop through the array and call execute() on each task object.

Step 12: End the program.

**Pseudocode**

ABSTRACT CLASS AutomatedTask:

FINAL METHOD execute():

CALL setup()

CALL performTask()

CALL teardown()

PRINT "Task completed: " + getTaskName()

METHOD setup():

PRINT "Setting up " + getTaskName() + "..."

ABSTRACT METHOD performTask()

METHOD teardown():

PRINT "Tearing down " + getTaskName() + "..."

ABSTRACT METHOD getTaskName()

CLASS EmailSender EXTENDS AutomatedTask:

METHOD performTask():

PRINT "Sending email to recipients..."

METHOD getTaskName():

RETURN "EmailSender"

CLASS FileArchiver EXTENDS AutomatedTask:

METHOD performTask():

PRINT "Moving files to archive directory..."

METHOD getTaskName():

RETURN "FileArchiver"

CLASS DatabaseBackup EXTENDS AutomatedTask:

METHOD performTask():

PRINT "Backing up the database to backup server..."

METHOD getTaskName():

RETURN "DatabaseBackup"

MAIN PROGRAM:

CREATE array tasks[3] of type AutomatedTask

SET tasks[0] = new EmailSender()

SET tasks[1] = new FileArchiver()

SET tasks[2] = new DatabaseBackup()

FOR EACH task IN tasks:

CALL task.execute()

**Program Code**

package Day6;

abstract class AutomatedTask {

public final void execute() {

setup();

performTask();

teardown();

System.out.println("Task completed: " + getTaskName());

}

protected void setup() {

System.out.println("Setting up " + getTaskName() + "...");

}

protected abstract void performTask();

protected void teardown() {

System.out.println("Tearing down " + getTaskName() + "...\n");

}

protected abstract String getTaskName();

}

class EmailSender extends AutomatedTask {

protected void performTask() {

System.out.println("Sending email to recipients...");

}

protected String getTaskName() {

return "EmailSender";

}

}

class FileArchiver extends AutomatedTask {

protected void performTask() {

System.out.println("Moving files to archive directory...");

}

protected String getTaskName() {

return "FileArchiver";

}

}

class DatabaseBackup extends AutomatedTask {

protected void performTask() {

System.out.println("Backing up the database to backup server...");

}

protected String getTaskName() {

return "DatabaseBackup";

}

}

public class TaskTest {

public static void main(String[] args) {

AutomatedTask[] tasks = {

new EmailSender(),

new FileArchiver(),

new DatabaseBackup()

};

for (AutomatedTask task : tasks) {

task.execute();

}

}

}

**5.Screenshots of Output**

****

**6. Observation / Reflection**

Understood the flow of the program

**Problem Solving Activity 2.3**

1. **Program Statement**

**Write a program to create a AutomatedTask and subclass to simplify the execution of tasks**

1. **Algorithm**

Step 1: Start the program.

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

Step 3: In execute(), call setup(), performTask(), teardown(), and print completion message.

Step 4: Define setup() to print setup message using getTaskName().

Step 5: Declare abstract methods performTask() and getTaskName() to be implemented by subclasses.

Step 6: Define teardown() to print teardown message using getTaskName().

Step 7: Create EmailSender class extending AutomatedTask and implement performTask() and getTaskName().

Step 8: Create FileArchiver class extending AutomatedTask and implement required methods.

Step 9: Create DatabaseBackup class extending AutomatedTask and implement required methods.

Step 10: In main(), create an array of tasks with EmailSender, FileArchiver, and DatabaseBackup.

Step 11: Loop through the array and call execute() on each task object.

Step 12: End the program.

**3.Pseudocode**

ABSTRACT CLASS AutomatedTask:

FINAL METHOD execute():

CALL setup()

CALL performTask()

CALL teardown()

PRINT "Task completed: " + getTaskName()

METHOD setup():

PRINT "Setting up " + getTaskName() + "..."

ABSTRACT METHOD performTask()

METHOD teardown():

PRINT "Tearing down " + getTaskName() + "..."

ABSTRACT METHOD getTaskName()

CLASS EmailSender EXTENDS AutomatedTask:

METHOD performTask():

PRINT "Sending email to recipients..."

METHOD getTaskName():

RETURN "EmailSender"

CLASS FileArchiver EXTENDS AutomatedTask:

METHOD performTask():

PRINT "Moving files to archive directory..."

METHOD getTaskName():

RETURN "FileArchiver"

CLASS DatabaseBackup EXTENDS AutomatedTask:

METHOD performTask():

PRINT "Backing up the database to backup server..."

METHOD getTaskName():

RETURN "DatabaseBackup"

MAIN PROGRAM:

CREATE array tasks[3] of type AutomatedTask

SET tasks[0] = new EmailSender()

SET tasks[1] = new FileArchiver()

SET tasks[2] = new DatabaseBackup()

FOR EACH task IN tasks:

CALL task.execute()

**4.Program Code**

package Day6;

abstract class AutomatedTask {

public final void execute() {

setup();

performTask();

teardown();

System.out.println("Task completed: " + getTaskName());

}

protected void setup() {

System.out.println("Setting up " + getTaskName() + "...");

}

protected abstract void performTask();

protected void teardown() {

System.out.println("Tearing down " + getTaskName() + "...\n");

}

protected abstract String getTaskName();

}

class EmailSender extends AutomatedTask {

protected void performTask() {

System.out.println("Sending email to recipients...");

}

protected String getTaskName() {

return "EmailSender";

}

}

class FileArchiver extends AutomatedTask {

protected void performTask() {

System.out.println("Moving files to archive directory...");

}

protected String getTaskName() {

return "FileArchiver";

}

}

class DatabaseBackup extends AutomatedTask {

protected void performTask() {

System.out.println("Backing up the database to backup server...");

}

protected String getTaskName() {

return "DatabaseBackup";

}

}

public class TaskTest {

public static void main(String[] args) {

AutomatedTask[] tasks = {

new EmailSender(),

new FileArchiver(),

new DatabaseBackup()

};

for (AutomatedTask task : tasks) {

task.execute();

}

}

}

**5.Screenshots of Output**

****

**6. Observation / Reflection**

Understood the flow of the program

**Problem Solving Activity 3.2**

1. **Program Statement**

**Write a program where**

**Abstract base: Shape with getArea() Subclasses: Circle, Square Create**

**polymorphic list and calculate areas**

1. **Algorithm**

**Step 1:** Start the program.

**Step 2:** Define an abstract class Shape with an abstract method getArea().

**Step 3:** Create class Circle that extends Shape and implements getArea() using radius.

**Step 4:** Create class Square that extends Shape and implements getArea() using side.

**Step 5:** In main(), create a list to store Shape objects.

**Step 6:** Add Circle and Square objects to the list.

**Step 7:** Initialize totalArea as 0.

**Step 8:** Loop through each Shape in the list.

**Step 9:** Call getArea() for each shape and print its type and area.

**Step 10:** Add each shape’s area to totalArea.

**Step 11:** Print the final total area.

**Step 12:** End the program

**3.Pseudocode**

ABSTRACT CLASS Shape:

ABSTRACT METHOD getArea()

CLASS Circle EXTENDS Shape:

ATTRIBUTE radius

CONSTRUCTOR(radius)

SET this.radius = radius

METHOD getArea():

RETURN π \* radius \* radius

CLASS Square EXTENDS Shape:

ATTRIBUTE side

CONSTRUCTOR(side)

SET this.side = side

METHOD getArea():

RETURN side \* side

MAIN PROGRAM:

CREATE list shapes of type Shape

ADD new Circle(3.0) to shapes

ADD new Square(4.0) to shapes

ADD new Circle(1.5) to shapes

SET totalArea = 0

FOR EACH shape S IN shapes:

SET area = s.getArea()

PRINT S**4.Program Code**

package Day6;

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(3.0));

shapes.add(new Square(4.0));

shapes.add(new Circle(1.5));

double totalArea = 0;

for (Shape s : shapes) {

double area = s.getArea();

System.out.printf("%s area = %.2f%n",

s.getClass().getSimpleName(), area);

totalArea += area;

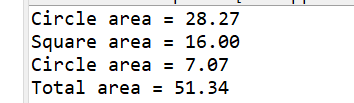
}

System.out.printf("Total area = %.2f%n", totalArea);

}

}

**5.Screenshots of Output**

****

**6. Observation / Reflection**

By using polymorphism the same method is accessed