Name : Chandrakanth HV
Assignment : Bridge Course Day 6
Date : 01-07-2025

SECTION 1 :
Problem Statement 1 :Create a class named Employee with the following:

- Attributes: name, employeeId, salary
- Method: getDetails() – This should return the employee's details.

Then, create two subclasses:

1. Manager : Add an extra attribute: department
   Change the getDetails() method so it also shows the department.
2. Developer : Add an extra attribute: programmingLanguage
   Change the getDetails() method to include the programming language.

Pseudo code :

- START
- DISPLAY "Choose Employee Type: 1. Manager 2. Developer"
- READ choice
- IF choice = 1 THEN
- CREATE Manager object
- CALL processDetails() to input name, ID, salary, department
- ELSE IF choice = 2 THEN
- CREATE Developer object
- CALL processDetails() to input name, ID, salary, language
- ELSE
- PRINT "Invalid choice"
- END IF
- CLOSE Scanner
- ENd

Algorithm: steps

1. Start
2. Create a Scanner object for user input
3. Display employee type options (Manager or Developer)
4. Read user's choice
5. If choice is Manager:
6. Create Manager object
7. Call processDetails() to input and display info

8. Else if choice is Developer:
9. Create Developer object
10. Call processDetails() to input and display info
11. Else:
12. Show invalid choice message
13. Close Scanner
14. Stop

Code :

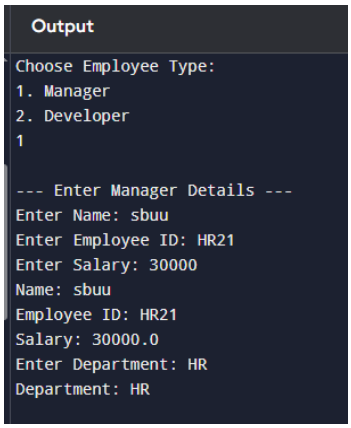
```
import java.util.Scanner;

class Employee {
    String name;
    String employeeId;
    double salary;

    public void processDetails(Scanner scanner) {
        System.out.print("Enter Name: ");
        name = scanner.nextLine();

        System.out.print("Enter Employee ID: ");
        employeeId = scanner.nextLine();

        System.out.print("Enter Salary: ");
        salary = Double.parseDouble(scanner.nextLine());

        System.out.println("Name: " + name);
        System.out.println("Employee ID: " + employeeId);
        System.out.println("Salary: " + salary);
    }
}

class Manager extends Employee {
    String department;

    @Override
    public void processDetails(Scanner scanner) {
        super.processDetails(scanner);
        System.out.print("Enter Department: ");
        department = scanner.nextLine();
        System.out.println("Department: " + department);
    }
}
```

```
class Developer extends Employee {
    String programmingLanguage;

    @Override
    public void processDetails(Scanner scanner) {
        super.processDetails(scanner);
        System.out.print("Enter Programming Language: ");
        programmingLanguage = scanner.nextLine();
        System.out.println("Programming Language: " + programmingLanguage);
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Choose Employee Type:\n1. Manager\n2. Developer");
        int choice = Integer.parseInt(scanner.nextLine());

        if (choice == 1) {
            Manager manager = new Manager();
            System.out.println("\n--- Enter Manager Details ---");
            manager.processDetails(scanner);
        } else if (choice == 2) {
            Developer developer = new Developer();
            System.out.println("\n--- Enter Developer Details ---");
            developer.processDetails(scanner);
        } else {
            System.out.println("Invalid choice.");
        }

        scanner.close();
    }
}
```

Test cases	Input	Expected Output	Actual Output	Status

TC1	Choose Employee Type: 1. Manager 2. Developer	--- Enter Manager Details --- Enter Name: sbuu Enter Employee ID: HR21 Enter Salary: 30000 Name: sbuu Employee ID: HR21 Salary: 30000.0 Enter Department: HR Department: HR	--- Enter Manager Details --- Enter Name: sbuu Enter Employee ID: HR21 Enter Salary: 30000 Name: sbuu Employee ID: HR21 Salary: 30000.0 Enter Department: HR Department: HR	Pass
TC2	Choose Employee Type: 1. Manager 2. Developer	--- Enter Developer Details --- Enter Name: stu Enter Employee ID: HRS20 Enter Salary: 15000 Name: stu Employee ID: HRS20 Salary: 15000.0 Enter Programming Language: java Programming Language: java	--- Enter Developer Details --- Enter Name: stu Enter Employee ID: HRS20 Enter Salary: 15000 Name: stu Employee ID: HRS20 Salary: 15000.0 Enter Programming Language: java Programming Language: java	Pass
TC3	Choose Employee Type: 1. Manager 2. Developer 3	Invalid choice.	Invalid choice.	Pass

TC1 :



TC2 :

```
Output
Choose Employee Type:
1. Manager
2. Developer
2

--- Enter Developer Details ---
Enter Name: stu
Enter Employee ID: HRS20
Enter Salary: 15000
Name: stu
Employee ID: HRS20
Salary: 15000.0
Enter Programming Language: java
Programming Language: java
```

TC3 :

```
Output
Choose Employee Type:
1. Manager
2. Developer
3
Invalid choice.
```

Observation:

The code uses inheritance to share common functionality between Manager and Developer.Scanner is used to read all inputs.The processDetails() method helps to combine input and output, simplifying the structure.Method overriding is applied to extend functionality in subclasses.This design promotes code reusability and clean structure.

Problem Statement 2 : Create a class structure using inheritance:**

1. Create a base class called Animal,It should have a method called makeSound().
2. Create two subclasses:Dog,Cat,In each subclass, override the makeSound() method to give the correct sound

Pseudo code :

- START
- Display "Choose an animal: 1. Dog 2. Cat"
- READ user input as choice
- IF choice is 1 THEN
- CREATE Dog object
- ELSE IF choice is 2 THEN
- CREATE Cat object
- ELSE
- CREATE Animal object
- PRINT "Invalid choice"
- CALL makeSound() method using animal object
- CLOSE scanner
- END

Algorithm: steps

1. Start the program.
2. Create a Scanner object to read user input.
3. Display the animal selection menu (1. Dog, 2. Cat).
4. Read the user's choice as an integer.
5. Use an if-else condition:
6. If choice is 1, create a Dog object.
7. If choice is 2, create a Cat object.
8. If invalid, create a generic Animal object and print "Invalid choice."
9. Call the makeSound() method using the Animal reference.
10. Close the Scanner.
11. End the program.

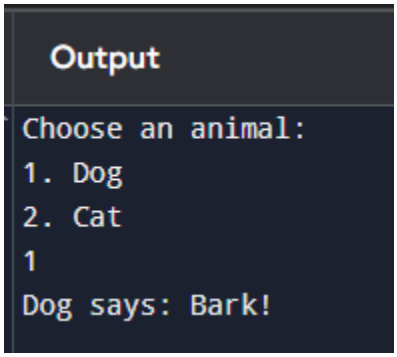
Code :

```
import java.util.Scanner;
class Animal {
   public void makeSound()  {
   }
}
```

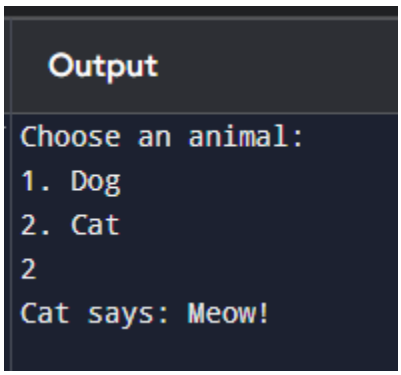
```
class Dog extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Dog says: Bark!");
    }
}


class Cat extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Cat says: Meow!");
    }
}


public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Choose an animal:\n1. Dog\n2. Cat");
        int choice = Integer.parseInt(scanner.nextLine());

        Animal animal;

        if (choice == 1) {
            animal = new Dog();
        } else if (choice == 2) {
            animal = new Cat();
        } else {
            animal = new Animal();
            System.out.println("Invalid choice.");
        }

        animal.makeSound();
        scanner.close();
    }
}
```

Test cases	Input	Expected Output	Actual Output	Status
TC1	Choose an animal: 1. Dog 2. Cat 1	Dog says: Bark!	Dog says: Bark!	Pass
TC2	Choose an animal: 1. Dog 2. Cat 2	Cat says: Meow!	Cat says: Meow!	Pass
TC3	Choose an animal: 1. Dog 2. Cat 9	Invalid choice.	Invalid choice.	Pass

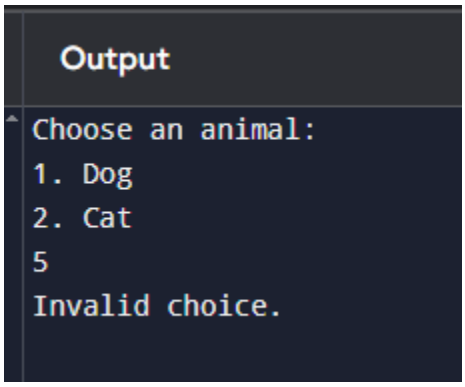
TC1 :



TC2 :



TC3 :



Observation:

This program uses inheritance and method overriding.The method makeSound() is overridden in Dog and Cat classes.The base class reference Animal holds different subclass objects (Dog or Cat), demonstrating polymorphism .If the user enters an invalid choice, a base class object is created, and no sound is printed

.

Problem Statement 3 : Design an Inheritance Base: ElectronicDevice
Subclasses: Television, Laptop, Smartphone, List attributes and methods per subclass

Pseudo code :

- Start
- Create scanner
- Display "Choose device"
- Read user choice
- If choice is 1:
- Create Television object
- Get brand and screen size from user
- Display TV details
- Else if choice is 2:
- Create Smartphone object
- Get brand and battery from user
- Display Smartphone details
- Else:
- Show "Invalid choice"
- Close scanner
- End

Algorithm: steps

1. Start the program.
2. Create a Scanner object to take user input.
3. Ask the user to choose a device type: Television or Smartphone
4. Based on the choice:
5. If user chooses Television, ask for brand and screen size.
6. If user chooses Smartphone, ask for brand and battery capacity.
7. Store the input in respective class objects.
8. Display the details of the selected device.
9. End the program.
10. Call drive() to show the car is driving.
11. End the program

Code :

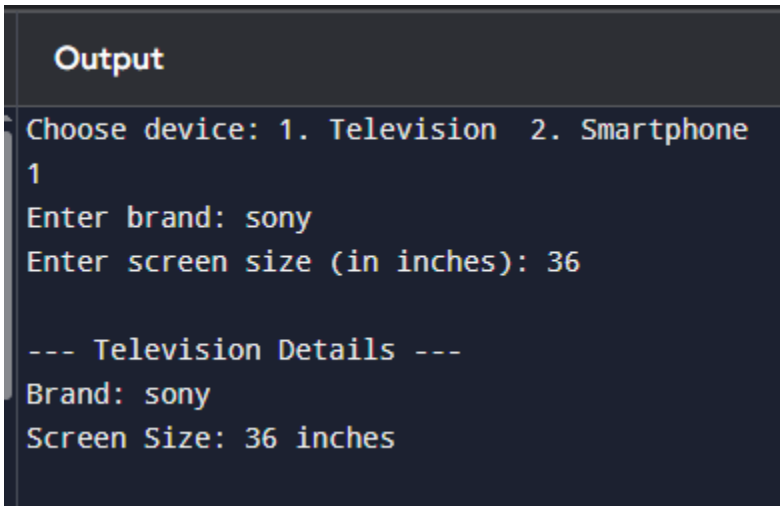
```
import java.util.Scanner;

class ElectronicDevice {
    String brand;
    double price;

    void getDetails(Scanner scanner) {
        System.out.print("Enter brand: ");
        brand = scanner.nextLine();
        System.out.print("Enter price: ");
        price = Double.parseDouble(scanner.nextLine());
    }

    void displayDetails() {
        System.out.println("Brand: " + brand);
        System.out.println("Price: " + price);
    }
}

class Television extends ElectronicDevice {
    int screenSize;

    @Override
    void getDetails(Scanner scanner) {
        super.getDetails(scanner);
        System.out.print("Enter screen size (in inches): ");
        screenSize = Integer.parseInt(scanner.nextLine());
    }

    void displayTelevisionDetails() {
        super.displayDetails();
        System.out.println("Screen Size: " + screenSize + " inches");
    }
}

class Smartphone extends ElectronicDevice {
    int cameraMP;

    @Override
    void getDetails(Scanner scanner) {
        super.getDetails(scanner);
        System.out.print("Enter camera resolution (in MP): ");
        cameraMP = Integer.parseInt(scanner.nextLine());
    }
```

```
    void displaySmartphoneDetails() {
        super.displayDetails();
        System.out.println("Camera: " + cameraMP + " MP");
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("\n--- Enter Television Details ---");
        Television tv = new Television();
        tv.getDetails(scanner);
        System.out.println("\nTelevision Details:");
        tv.displayTelevisionDetails();

        System.out.println("\n--- Enter Smartphone Details ---");
        Smartphone phone = new Smartphone();
        phone.getDetails(scanner);
        System.out.println("\nSmartphone Details:");
        phone.displaySmartphoneDetails();

        scanner.close();
    }
}
```

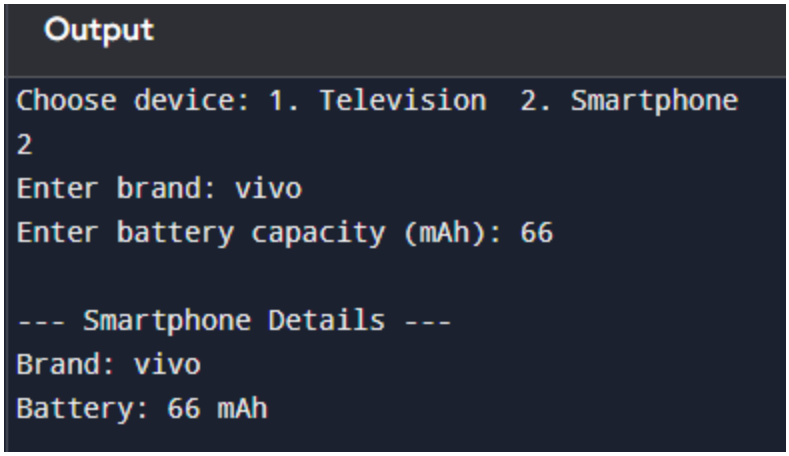
Test cases	Input	Expected Output	Actual Output	Status
TC1	Choose device: 1. Television 2. Smartphone 1 Enter brand: sony Enter screen size (in inches): 36	Brand: sony Screen Size: 36 inches	Brand: sony Screen Size: 36 inches	Pass

TC2	Choose device: 1. Television 2. Smartphone 2 Enter brand: vivo Enter battery capacity (mAh): 66	Brand: vivo Battery: 66 mAh	Brand: vivo Battery: 66 mAh	Pass
TC3	Choose device: 1. Television 2. Smartphone 5	Invalid choice.	Invalid choice.	Pass

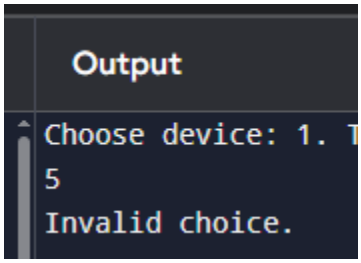
TC1 :



TC2 :



TC2 :



Observation:

The program demonstrates single-level inheritance where Television and Smartphone classes inherit from the base class ElectronicDevice.The user is prompted to enter details such as brand, price, screen size , and camera megapixels.Method overriding is used to extend the getDetails() method in each subclass for device-specific input.The program successfully accepts and displays the input values for both television and smartphone objects.

Section 2 :

Problem Statement 1 : Create an abstract class PaymentGateway with an abstract method processPayment(double amount).
Then create two subclasses:,CreditCardGateway,PayPalGateway

Each subclass should implement the processPayment method.Also, try to create an object of the PaymentGateway class

Pseudo code :

- Start
- Display payment method options
- Read user's choice
- Ask user to enter amount
- Read amount
- If choice is 1
- Create CreditCardGateway object
- Else if choice is 2
- Create PayPalGateway object
- Else
- Display "Invalid choice" and stop
- Call processPayment(amount) using the selected object
- End
-

Algorithm: steps

1. Start
2. Create a Scanner object to read input
3. Display payment method options:Credit Card,PayPal
4. Read user’s choice and store in choice
5. Ask the user to enter the payment amount
6. Read amount and store in amount
7. Declare a reference variable of type PaymentGateway
8. If choice == 1, assign it to a new CreditCardGateway
9. Else if choice == 2, assign it to a new PayPalGateway
10. Else print "Invalid choice" and exit
11. Call processPayment(amount) on the selected object
12. End

Code :

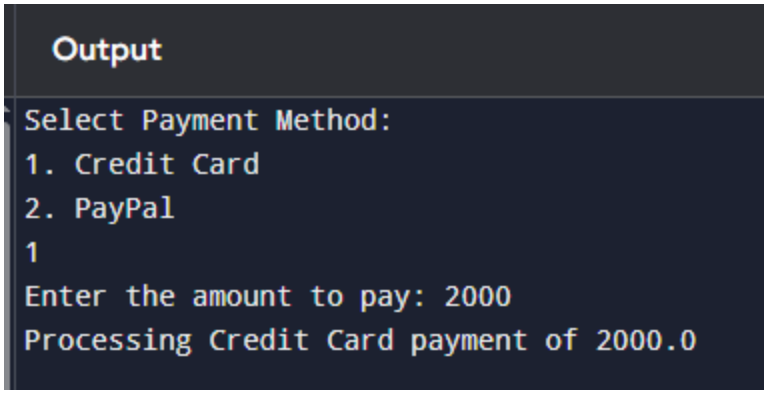
```
import java.util.Scanner;
```

```
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 " + amount);
    }
}

class PayPalGateway extends PaymentGateway {
    @Override
    void processPayment(double amount) {
        System.out.println("Processing PayPal payment of " + amount);
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Select Payment Method:");
        System.out.println("1. Credit Card");
        System.out.println("2. PayPal");
        int choice = scanner.nextInt();

        System.out.print("Enter the amount to pay: ");
        double amount = scanner.nextDouble();

        PaymentGateway gateway;

        if (choice == 1) {
            gateway = new CreditCardGateway();
        } else if (choice == 2) {
            gateway = new PayPalGateway();
        } else {
            System.out.println("Invalid choice.");
            return;
        }

        gateway.processPayment(amount);
```

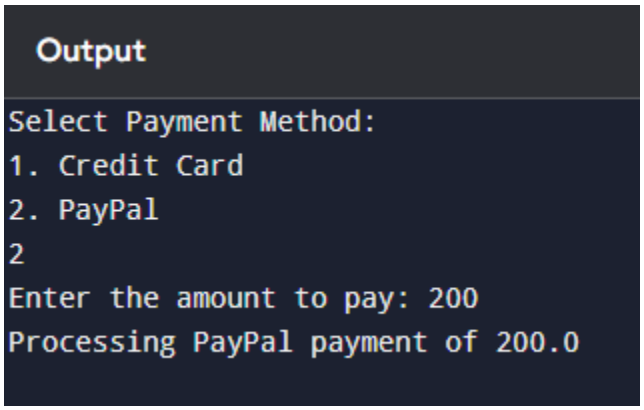
```
    }
}
```

Test cases	Input	Expected Output	Actual Output	Status
TC1	Select Payment Method: 1. Credit Card 2. PayPal 1	Select Payment Method: 1. Credit Card 2. PayPal 1 Enter the amount to pay: 2000 Processing Credit Card payment of 2000.0	Select Payment Method: 1. Credit Card 2. PayPal 1 Enter the amount to pay: 2000 Processing Credit Card payment of 2000.0	Pass
TC2	Select Payment Method: 1. Credit Card 2. PayPal 2	Select Payment Method: 1. Credit Card 2. PayPal 2 Enter the amount to pay: 200 Processing PayPal payment of 200.0	Select Payment Method: 1. Credit Card 2. PayPal 2 Enter the amount to pay: 200 Processing PayPal payment of 200.0	Pass
TC3	Select Payment Method: 1. Credit Card 2. PayPal 5	Select Payment Method: 1. Credit Card 2. PayPal 5 Enter the amount to pay: 200 Invalid choice.	Select Payment Method: 1. Credit Card 2. PayPal 5 Enter the amount to pay: 200 Invalid choice.	Pass

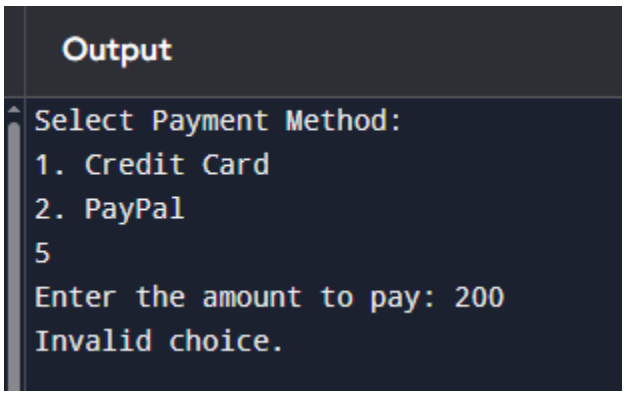
TC1 :



TC2 :



TC3 :



Observation:

The program demonstrates abstraction using an abstract class PaymentGateway.It uses runtime polymorphism to call the processPayment() method based on user input.Trying to create an object of the abstract class directly will give a compile-time error.The program is extendable — new payment types can be added easily by creating more subclasses.

Problem Statement 2 : Create an abstract class Instrument with an abstract method play(). Then create two subclasses:Guitar,Piano.

Each subclass should implement the play() method to print its own sound.
Test the classes by calling the play() method for both instruments.

Pseudo code :

- Start
- Show instrument options to user
- Read user choice
- If choice is 1 → create guitar
- else if choice is 2 → create piano
- else → print "invalid choice" and exit
- Call play() on selected instrument
- End

Algorithm: steps

1. Start
2. Display menu (1. Guitar, 2. Piano)
3. Take user input → choice
4. If choice = 1 → create Guitar object
5. If choice = 2 → create Piano object
6. Else → print invalid and exit
7. Call play() method
8. End

Code :

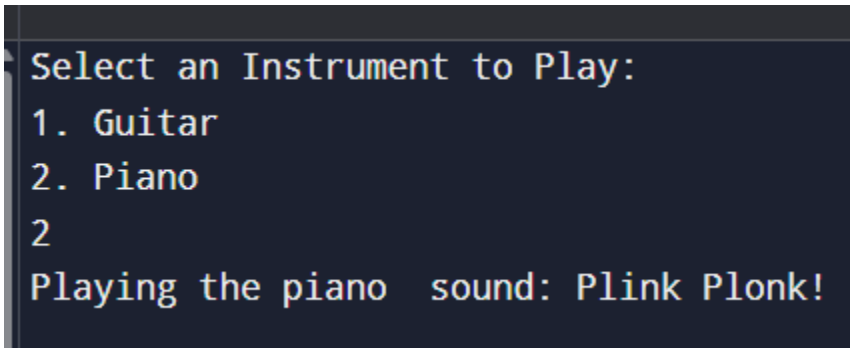
```
import java.util.Scanner;

abstract class Instrument {
   abstract void play();
}

class Guitar extends Instrument {
   @Override
   void play() {
      System.out.println("Strumming the guitar  sound: Twing Twing!");
   }
}

class Piano extends Instrument {
   @Override
```

```
    void play() {
        System.out.println("Playing the piano  sound: Plink Plonk!");
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Select an Instrument to Play:");
        System.out.println("1. Guitar");
        System.out.println("2. Piano");
        int choice = scanner.nextInt();

        Instrument instrument;

        if (choice == 1) {
            instrument = new Guitar();
        } else if (choice == 2) {
            instrument = new Piano();
        } else {
            System.out.println("Invalid choice.");
            return;
        }

        instrument.play();
    }
}
```

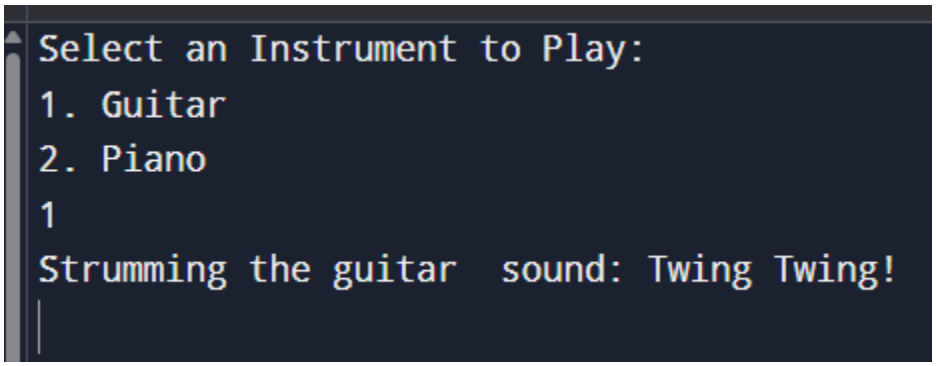
Test cases	Input	Expected Output	Actual Output	Status
TC1	Select an Instrument to Play: 1. Guitar 2. Piano 2	Playing the piano sound: Plink Plonk!	Playing the piano sound: Plink Plonk!	Pass
TC2	Select an Instrument to Play: 1. Guitar 2. Piano 1	Strumming the guitar sound: Twing Twing!	Strumming the guitar sound: Twing Twing!	Pass

TC3	Select an Instrument to Play: 1. Guitar 2. Piano piano	ERROR!	ERROR!	Pass

TC1 :



TC2 :



TC3 :

```
Select an Instrument to Play:
1. Guitar
2. Piano
piano
ERROR!
Exception in thread "main" java.util.InputMismatchException
    at java.base/java.util.Scanner.throwFor(Scanner.java:947)
    at java.base/java.util.Scanner.next(Scanner.java:1602)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2267)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2221)
    at Main.main(Main.java:28)
```

Observation: The program uses abstraction to handle different automated tasks with a common extence() method. It simplifies task execution and improves code structure by treating all tasks uniformly.

Problem Statement 3 :

Create an abstract class `AutomatedTask` with an abstract method `execute()`.
Then create subclasses: `EmailSender`, `FileArchiver`, `DatabaseBackup`
Each subclass should implement the `execute()` method to display a message related to its task. Use abstraction to simplify how these tasks are executed.

Pseudo code :

- Start
- Create empty task list
- Input number of tasks
- Repeat for each task:
- input task type
- add corresponding task object to list
- if invalid, skip
- For each task in list: call execute()
- End

Algorithm: steps

1. Start
2. Create a list to store tasks
3. Read total number of tasks
4. For each task:
   a. Read task type
   b. If type is email → add EmailSender
   c. If archive → add FileArchiver
   d. If backup → add DatabaseBackup
   e. Else → show invalid message
5. Loop through list and execute all tasks
6. End

Code :

```
import java.util.ArrayList;
import java.util.Scanner;

public class Main {

   abstract static class AutomatedTask {
      abstract void execute();
```

```
}

static class EmailSender extends AutomatedTask {
    @Override
    void execute() {
        System.out.println("Sending email.");
    }
}

static class FileArchiver extends AutomatedTask {
    @Override
    void execute() {
        System.out.println("Archiving files.");
    }
}

static class DatabaseBackup extends AutomatedTask {
    @Override
    void execute() {
        System.out.println("Backing up database.");
    }
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    ArrayList<AutomatedTask> tasks = new ArrayList<>();

    System.out.print("Enter number of tasks to execute: ");
    int count = scanner.nextInt();
    scanner.nextLine();

    for (int i = 1; i <= count; i++) {
        System.out.println("\nTask " + i + ":");
        System.out.print("Enter task type (email/archive/backup): ");
        String type = scanner.nextLine().toLowerCase();

        switch (type) {
            case "email":
                tasks.add(new EmailSender());
                break;
            case "archive":
                tasks.add(new FileArchiver());
                break;
            case "backup":
```

```
                tasks.add(new DatabaseBackup());
                break;
            default:
                System.out.println("Invalid task type. Skipping.");
        }
    }

    System.out.println("\n--- Executing Tasks ---");
    for (AutomatedTask task : tasks) {
        task.execute();
    }

    scanner.close();
  }
}
```

Test cases	Input	Expected Output	Actual Output	Status
TC1	Enter number of tasks to execute: 2 Task 1: Enter task type (email/archive/backup): email Task 2: Enter task type (email/archive/backup): archive	--- Executing Tasks --- Sending email... Archiving files...	--- Executing Tasks --- Sending email... Archiving files...	Pass
TC2	Enter number of tasks to execute: 1 Task 1: Enter task type (email/archive/backup): backup	--- Executing Tasks --- Backing up database...	--- Executing Tasks --- Backing up database...	Pass
TC3	Enter number of tasks to execute: 1 Task 1: Enter task type (email/archive/backup): abc	Invalid task type. Skipping.	Invalid task type. Skipping.	Pass

TC1 :

```
Enter number of tasks to execute: 2

Task 1:
Enter task type (email/archive/backup): email

Task 2:
Enter task type (email/archive/backup): archive

--- Executing Tasks ---
Sending email...
Archiving files...
```

TC2 :

```
Enter number of tasks to execute: 1

Task 1:
Enter task type (email/archive/backup): backup

--- Executing Tasks ---
Backing up database...
```

TC3 :

```
Enter number of tasks to execute: 1

Task 1:
Enter task type (email/archive/backup): abc
Invalid task type. Skipping.

--- Executing Tasks ---
```

Observation: The program uses abstraction to perform different tasks (email, archive, backup) through a single interface, making the code clean, simple, and easy to manage.

SECTION 3 :(Polymorphism - Many Forms)

Problem Statement 1 :

Create an abstract class `Employee` with an abstract method `calculatePayroll()`.

Create two subclasses: `SalariedEmployee`, `HourlyEmployee`

Each subclass should implement the payroll calculation.

Process a list of employees and print their payrolls.

Pseudo code :

- Start
- Create empty employee list
- Read number of employees → count
- Repeat for each employee (1 to count):
- Read name and type
- If type is salaried:
- read salary
- add salaried employee to list
- else if type is hourly:
- read rate and hours
- add hourlyemployee to list
- else:
- print "invalid type"
- Print "payroll report"
- For each employee in list:
- print name and payroll
- End

Algorithm: steps

1. Start
2. Input number of employees
3. For each employee:
    a. Input name and type
    b. If salaried → input salary → add salaried employee
    c. If hourly → input rate & hours → add hourly employee
    d. Else → show error
4. Print payroll report
5. For each employee → display name and payroll
6. End

Code :

```
import java.util.ArrayList;
import java.util.Scanner;
public class Main {
    abstract static class Employee {
        String name;

        Employee(String name) {
            this.name = name;
        }

        abstract double calculatePayroll();
    }

    static class SalariedEmployee extends Employee {
        double monthlySalary;

        SalariedEmployee(String name, double monthlySalary) {
            super(name);
            this.monthlySalary = monthlySalary;
        }

        @Override
        double calculatePayroll() {
            return monthlySalary;
        }
    }

    static class HourlyEmployee extends Employee {
        double hourlyRate;
        int hoursWorked;

        HourlyEmployee(String name, double hourlyRate, int hoursWorked) {
            super(name);
            this.hourlyRate = hourlyRate;
            this.hoursWorked = hoursWorked;
        }

        @Override
        double calculatePayroll() {
            return hourlyRate * hoursWorked;
        }
    }

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

```
        Scanner scanner = new Scanner(System.in);
        ArrayList<Employee> employees = new ArrayList<>();

        System.out.print("Enter number of employees: ");
        int count = scanner.nextInt();
        scanner.nextLine();

        for (int i = 1; i <= count; i++) {
            System.out.println("\nEnter details for Employee " + i);

            System.out.print("Enter name: ");
            String name = scanner.nextLine();

            System.out.print("Enter type (salaried/hourly): ");
            String type = scanner.nextLine().toLowerCase();

            if (type.equals("salaried")) {
                System.out.print("Enter monthly salary: ");
                double salary = scanner.nextDouble();
                scanner.nextLine();
                employees.add(new SalariedEmployee(name, salary));
            } else if (type.equals("hourly")) {
                System.out.print("Enter hourly rate: ");
                double rate = scanner.nextDouble();

                System.out.print("Enter hours worked: ");
                int hours = scanner.nextInt();
                scanner.nextLine();

                employees.add(new HourlyEmployee(name, rate, hours));
            } else {
                System.out.println("Invalid employee type. Skipping.");
            }
        }

       System.out.println("\n--- Payroll Report ---");
for (Employee emp : employees) {
    System.out.println("Name: " + emp.name);

    if (emp instanceof SalariedEmployee) {
        SalariedEmployee se = (SalariedEmployee) emp;
        System.out.println("Type: Salaried");
        System.out.println("Monthly Salary: " + se.monthlySalary);
        System.out.println("Total Payroll: " + se.calculatePayroll());
```

```
    } else if (emp instanceof HourlyEmployee) {
        HourlyEmployee he = (HourlyEmployee) emp;
        System.out.println("Type: Hourly");
        System.out.println("Hourly Rate: " + he.hourlyRate);
        System.out.println("Hours Worked: " + he.hoursWorked);
        System.out.println("Total Payroll: " + he.calculatePayroll());
    }
}
        scanner.close();
    }
}
```

Test cases	**Input**	**Expected Output**	**Actual Output**	**Status**
TC1	Enter number of employees: 1 Enter details for Employee 1 Enter name: abc Enter type (salaried/hourly): salaried Enter monthly salary: 30000	--- Payroll Report --- abc: 30000.00	--- Payroll Report --- abc: 30000.00	Pass
TC2	Enter number of employees: 1 Enter details for Employee 1 Enter name: abc Enter type (salaried/hourly): hourly Enter hourly rate: 5000 Enter hours worked: 7	--- Payroll Report --- abc: 35000.00	--- Payroll Report --- abc: 35000.00	Pass
TC3	Enter number of employees: 1	Invalid employee type. Skipping.	Invalid employee type. Skipping.	Pass

	Enter details for Employee 1 Enter name: anu Enter type (salaried/hourly): 123			

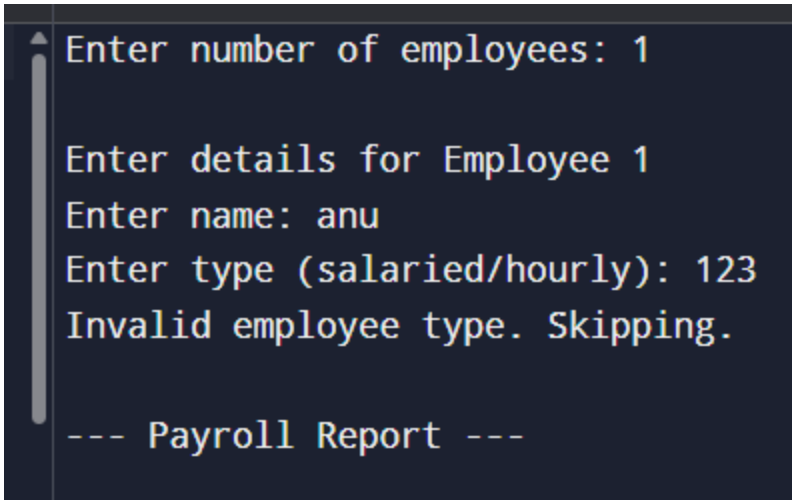
TC1 :

```
Enter number of employees: 1

Enter details for Employee 1
Enter name: abc
Enter type (salaried/hourly): salaried
Enter monthly salary: 30000

--- Payroll Report ---
abc: 30000.00
```

TC2 :

```
Enter number of employees: 1

Enter details for Employee 1
Enter name: abc
Enter type (salaried/hourly): hourly
Enter hours worked: 7

--- Payroll Report ---
abc: 35000.00

--- Code Execution Successful ---
```

TC3 :



Observation: This program builds a simple Employee Payroll System using Java and object-oriented concepts. It collects employee details, calculates payroll based on type (salaried or hourly), and displays the results. Key challenges included handling user input correctly and managing different employee types using inheritance.

Problem Statement 2 :

Create an abstract class `Shape` with an abstract method `getArea()`.
Create subclasses:`Circle`, `Square`
Use a list to store different shapes and calculate their areas using polymorphism.

Pseudo code :

- Start
- Create empty list for shapes
- Input number of shapes → count
- Repeat count times:
    - Input shape type (circle/square)
    - If circle → input radius → add circle to list
    - If square → input side → add square to list
    - Else → print "invalid type"
- For each shape → print area
- End

Algorithm: steps

1. Start
2. Input number of shapes
3. For each shape:
4. Input type
    - If circle → input radius → add Circle
    - If square → input side → add Square
    - Else → print invalid type
5. For each shape → print area
6. End

Code :

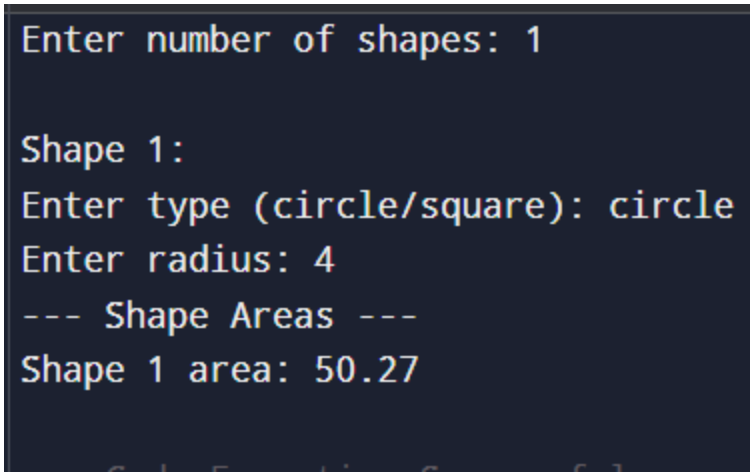
```
import java.util.ArrayList;
import java.util.Scanner;

public class Main {

   abstract static class Shape {
      abstract double getArea();
   }

   static class Circle extends Shape {
```

```
    double radius;

    Circle(double radius) {
        this.radius = radius;
    }

    @Override
    double getArea() {
        return Math.PI * radius * radius;
    }
}

static class Square extends Shape {
    double side;

    Square(double side) {
        this.side = side;
    }

    @Override
    double getArea() {
        return side * side;
    }
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    ArrayList<Shape> shapes = new ArrayList<>();

    System.out.print("Enter number of shapes: ");
    int count = scanner.nextInt();
    scanner.nextLine();

    for (int i = 1; i <= count; i++) {
        System.out.println("\nShape " + i + ":");
        System.out.print("Enter type (circle/square): ");
        String type = scanner.nextLine().toLowerCase();

        if (type.equals("circle")) {
            System.out.print("Enter radius: ");
            double r = scanner.nextDouble();
            scanner.nextLine();
            shapes.add(new Circle(r));
        } else if (type.equals("square")) {
```

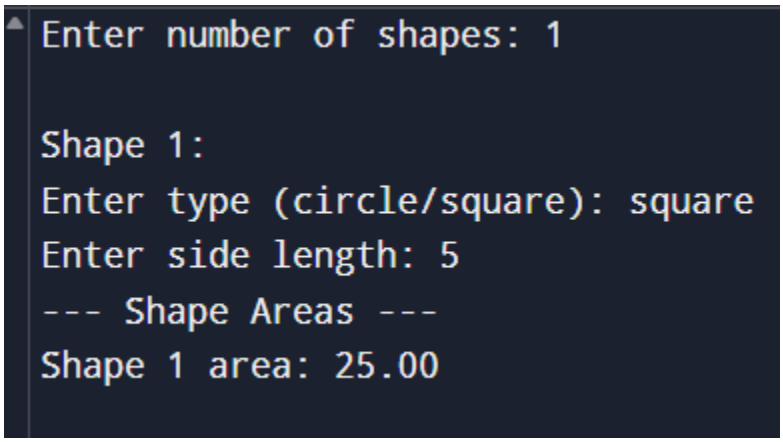
```
            System.out.print("Enter side length: ");
            double s = scanner.nextDouble();
            scanner.nextLine();
            shapes.add(new Square(s));
        } else {
            System.out.println("Invalid shape type. Skipping.");
        }
    }

    System.out.println("--- Shape Areas ---");
    for (int i = 0; i < shapes.size(); i++) {
        System.out.printf("Shape %d area: %.2f\n", i + 1, shapes.get(i).getArea());
    }

    scanner.close();
  }
}
```

Test cases	Input	Expected Output	Actual Output	Status
TC1	Enter number of shapes: 1 Shape 1: Enter type (circle/square): circle Enter radius: 4	--- Shape Areas --- Shape 1 area: 50.27	--- Shape Areas --- Shape 1 area: 50.27	Pass
TC2	Enter number of shapes: 1 Shape 1: Enter type (circle/square): square Enter side length: 5	--- Shape Areas --- Shape 1 area: 25.00	--- Shape Areas --- Shape 1 area: 25.00	Pass
TC3	Enter number of shapes: 1 Shape 1: Enter type (circle/square): sdf	Invalid shape type. Skipping.	Invalid shape type. Skipping.	Pass

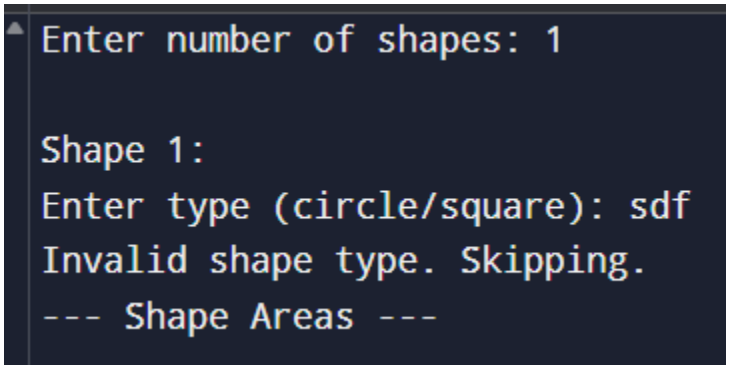
TC1 :



TC2 :



TC3 :



Observation: This program calculates the area of different shapes (circle and square) using inheritance and abstraction in Java. It takes user input for shape type and dimensions, stores objects in a polymorphic list, and prints each area. The main challenge was handling user input and ensuring correct object creation based on shape type.

Problem Statement 3 :

Create a base class `Tool` with a method `draw()`.
Create subclasses:`PenTool`, `EraserTool`, `LineTool`
Use a collection to store the tools and demonstrate polymorphism by calling `draw()` on each.

Pseudo code :

- Start
- Create empty list for tools
- Input number of tools → count
- Repeat for count times:
    - Input tool type (pen/eraser/line)
    - If pen → add pentool to list
- else if eraser → add erasertool
- else if line → add linetool
- else → print invalid
- For each tool in list:
    - Call draw() method
- End

Algorithm: steps

1. Start
2. Create list to store Tool objects
3. Read number of tools
4. Loop through tool entries:
    - Read tool type
    - Add corresponding object (PenTool, EraserTool, LineTool)
    - If invalid, skip
5. Loop through list and call draw()
6. End

Code :

```
import java.util.ArrayList;
import java.util.Scanner;

public class Main {

    abstract static class Tool {
```

```
    abstract void draw();
}

static class PenTool extends Tool {
    @Override
    void draw() {
        System.out.println("Drawing with Pen Tool.");
    }
}

static class EraserTool extends Tool {
    @Override
    void draw() {
        System.out.println("Erasing with Eraser Tool.");
    }
}

static class LineTool extends Tool {
    @Override
    void draw() {
        System.out.println("Drawing a line with Line Tool.");
    }
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    ArrayList<Tool> tools = new ArrayList<>();

    System.out.print("Enter number of tools to use: ");
    int count = scanner.nextInt();
    scanner.nextLine();

    for (int i = 1; i <= count; i++) {
        System.out.println("\nTool " + i + ":");
        System.out.print("Enter tool type (pen/eraser/line): ");
        String type = scanner.nextLine().toLowerCase();

        switch (type) {
            case "pen":
                tools.add(new PenTool());
                break;
            case "eraser":
                tools.add(new EraserTool());
                break;
```

```
                case "line":
                    tools.add(new LineTool());
                    break;
                default:
                    System.out.println("Invalid tool type. Skipping.");
            }
        }

        System.out.println("\n--- Executing Tools ---");
        for (Tool tool : tools) {
            tool.draw();
        }

        scanner.close();
    }
}
```

Test cases	Input	Expected Output	Actual Output	Status
TC1	Enter number of tools to use: 2 Tool 1: Enter tool type (pen/eraser/line): pen Tool 2: Enter tool type (pen/eraser/line): line	--- Executing Tools --- Drawing with Pen Tool. Drawing a line with Line Tool.	--- Executing Tools --- Drawing with Pen Tool. Drawing a line with Line Tool.	Pass
TC2	Enter number of tools to use: 1 Tool 1: Enter tool type (pen/eraser/line): eraser	--- Executing Tools --- Erasing with Eraser Tool.	--- Executing Tools --- Erasing with Eraser Tool.	Pass
TC3	Enter number of tools to use: 1 Tool 1: Enter tool type (pen/eraser/line): asf	Invalid tool type. Skipping.	Invalid tool type. Skipping.	Pass

TC1 :

```
Enter number of tools to use: 2

Tool 1:
Enter tool type (pen/eraser/line): pen

Tool 2:
Enter tool type (pen/eraser/line): line

--- Executing Tools ---
Drawing with Pen Tool.
Drawing a line with Line Tool.
```

TC2 :

```
Enter number of tools to use: 1

Tool 1:
Enter tool type (pen/eraser/line): eraser

--- Executing Tools ---
Erasing with Eraser Tool.
```

TC3 :

```
Enter number of tools to use: 1

Tool 1:
Enter tool type (pen/eraser/line): asf
Invalid tool type. Skipping.

--- Executing Tools ---
```

Observation: The program shows polymorphism by calling the draw() method on different tool types using a common base class. Tools are chosen by user input and executed through a single interface.