```
/*2. Write a Java program to create a base class Animal (Animal Family) with a method called Sound(). Create two subclasses Bird and Cat. Override the Sound() method in each subclass to make a specific sound for each animal.*/
class Animal
{
	void sound()
	{
		System.out.println("Animal Sound");
	}
}
class Bird extends Animal
{
	void sound()
	{
		System.out.println("parrot Bird Sound");
	}
}
class Cat extends Animal
{
	void sound()
	{
		System.out.println("cat mous");
	}
}
class Practical2
{
	public static void main(String args[])
	{
		Animal Bird=new Bird();
		Animal Cat=new Cat();
		Bird.sound();
		Cat.sound();
	}
}
```

```
/*
Write a Java program to create a class Vehicle with a method called speedUp().Create two subclasses Car and Bicycle. Override the speedUp() method in each subclass to increase the vehicle's speed differently
*/
class Vehicle
{
	void speedUp()
	{
		System.out.println("Vehicle Spped ");
	}
}
class Car extends Vehicle
{
	void speedUp()
	{
		System.out.println("Car Spped: 80km/hr");
	}
}
class Bicycle extends Vehicle
{
	void speedUp()
	{
		System.out.println("Bicycle Spped:5km/hr ");
	}
}

class Practical3
{
	public static void main(String args[])
	{
		Vehicle Car=new Car();
		Car.speedUp();
		Vehicle Bicycle=new Bicycle();
		Bicycle.speedUp();
	}
}
```

```
/*
Write a Java program to create a base class Shape with a method called calculateArea(). Create three subclasses: Circle, Rectangle, and Triangle. Override the calculateArea() method in each subclass to calculate and return the shape's area
*/
import java.util.*;

class Shape
{
	Scanner sc=new Scanner(System.in);


	void calculateArea()
	{
		System.out.println("calculateArea() Executed ");
	}
}
class Circle extends Shape
{
	float r, area;
	void calculateArea()

	{
		System.out.println("Enter Radius");
		r=sc.nextInt();

		area=(float)(3.14*r*r);
		System.out.println("Area of circle "+area);
	}
}
class Rectangle extends Shape
{
	int l,w;
	float area;
	void calculateArea()
	{
```

```
                System.out.println("Enter Length :");
                l=sc.nextInt();
                System.out.println("Enter Width :");
                w=sc.nextInt();
                area=l*w;
                System.out.println("Area of Rectangle: "+area);
        }
}
class Triangle extends Shape
{
        int b,h;
        float area;
        void calculateArea()
        {
                System.out.println("Enter Breadth :");
                b=sc.nextInt();
                System.out.println("Enter Height :");
                h=sc.nextInt();
                area=0.5f*b*h;
                System.out.println("Area of Triangle: "+area);
        }
}
class Practical4
{
        public static void main(String args[])
        {
                Shape Circle=new Circle();
                Shape Rectangle=new Rectangle();
                Shape Triangle=new Triangle();
                System.out.println("Circle: ");
                Circle.calculateArea();
                System.out.println("Rectangle: ");
                Rectangle.calculateArea();
                System.out.println("Triangle: ");
                Triangle.calculateArea();
        }
}
```

```
/*
6. Identify commonalities and differences between Publication, Book and Magazine classes. Title, Price, Copies are common instance variables and saleCopy is common method. The differences are, Bookclass has author and orderCopies(). Magazine Class has methods orderQty, Current issue, receiveissue().Write a program to find how many copies of the given books are ordered and display total sale of publication
*/
class Publication
{
   String title;
   double price;
   int copies;

   // Constructor for Publication
   Publication(String title, double price, int copies)
        {
      this.title = title;
      this.price = price;
      this.copies = copies;
   }

   // Common method to display total sale
   void saleCopy()
        {
      System.out.println("Total sale of " + title + ": " + (price * copies));
   }
}

// Book class extending Publication
class Book extends Publication {
   String author;

   // Constructor for Book
   Book(String title, double price, int copies, String author) {
      super(title, price, copies);
```

```
        this.author = author;
    }

    // Method to order additional copies for the book
    void orderCopies(int newCopies) {
        copies += newCopies;
        System.out.println("Ordered " + newCopies + " new copies of the book.");
    }
}

// Magazine class extending Publication
class Magazine extends Publication {
    String currentIssue;

    // Constructor for Magazine
    Magazine(String title, double price, int copies, String currentIssue) {
        super(title, price, copies);
        this.currentIssue = currentIssue;
    }

    // Method to order additional quantity of magazines
    void orderQty(int newQty) {
        copies += newQty;
        System.out.println("Ordered " + newQty + " new copies of the
magazine.");
    }

    // Method to display the current issue of the magazine
    void currentIssue() {
        System.out.println("Current issue of the magazine: " + currentIssue);
    }

    // Method to receive a new issue
    void receiveIssue() {
        System.out.println("Received the new issue of the magazine: " +
currentIssue);
    }
```

```
}

// Main class
public class Practical6 {
    public static void main(String[] args) {

        Book book1 = new Book("Java Programming", 500.0, 50, "James
Gosling");
        Magazine magazine1 = new Magazine("Tech Today", 100.0, 200, "January
2024");

        // Order more copies for both
        book1.orderCopies(20);
        magazine1.orderQty(50);

        // Display total sales for both
        book1.saleCopy();
        magazine1.saleCopy();

        // Display current issue of the magazine
        magazine1.currentIssue();

        // Receive new issue of the magazine
        magazine1.receiveIssue();
    }
}
```

/*Design and develop inheritance for a given case study, identify objects and relationships and implement inheritance wherever applicable. Employee class hasEmp_name, Emp_id, Address, Mail_id, and Mobile_noas members. Inherit the classes: Programmer, Team Lead, Assistant Project Manager and Project Manager from employee class. Add Basic Pay (BP) as the member of all the inherited classes with 97% of BP as DA, 10 % of BP as HRA, 12% of BP as PF, 0.1% of BP for staff club fund. Generate pay slips for the employees with their gross and net salary
*/

```
// Base class Employee
class Employee {
    String empName, empId;
    double basicPay;

    // Constructor to initialize employee details
    Employee(String empName, String empId, double basicPay) {
        this.empName = empName;
        this.empId = empId;
        this.basicPay = basicPay;
    }

    // Method to calculate and display pay slip
    void generatePaySlip() {
        double da = 0.97 * basicPay; // Dearness Allowance (97% of BP)
        double hra = 0.10 * basicPay; // House Rent Allowance (10% of BP)
        double pf = 0.12 * basicPay; // Provident Fund (12% of BP)
        double staffClubFund = 0.001 * basicPay; // Staff Club Fund (0.1% of BP)
        double grossSalary = basicPay + da + hra; // Gross salary
        double netSalary = grossSalary - (pf + staffClubFund); // Net salary

        System.out.println("\nPay Slip for " + empName + " (" + empId + ")");
        System.out.println("-----------------------------------------");
        System.out.printf("Basic Pay (BP): %.2f\n", basicPay);
        System.out.printf("Gross Salary: %.2f\n", grossSalary);
        System.out.printf("Net Salary: %.2f\n", netSalary);
        System.out.println("-----------------------------------------");
```

```
    }
}

// Derived class Programmer
class Programmer extends Employee {
    Programmer(String empName, String empId, double basicPay) {
        super(empName, empId, basicPay);
    }
}

// Derived class TeamLead
class TeamLead extends Employee {
    TeamLead(String empName, String empId, double basicPay) {
        super(empName, empId, basicPay);
    }
}

// Derived class AssistantProjectManager
class AssistantProjectManager extends Employee {
    AssistantProjectManager(String empName, String empId, double basicPay) {
        super(empName, empId, basicPay);
    }
}

// Derived class ProjectManager
class ProjectManager extends Employee {
    ProjectManager(String empName, String empId, double basicPay) {
        super(empName, empId, basicPay);
    }
}

// Main class
public class Practical7 {
    public static void main(String[] args) {
        // Create employees
        Programmer programmer = new Programmer("Alice", "P001", 50000);
        TeamLead teamLead = new TeamLead("Bob", "TL001", 70000);
```

```
        AssistantProjectManager apm = new AssistantProjectManager("Charlie",
"APM001", 90000);
        ProjectManager pm = new ProjectManager("David", "PM001", 100000);

        // Generate pay slips
        programmer.generatePaySlip();
        teamLead.generatePaySlip();
        apm.generatePaySlip();
        pm.generatePaySlip();
    }
}
```

```
/*
Write a Java program to create a class known as "BankAccount" with methods called deposit() and withdraw(). Create a subclass called SavingsAccount that overrides the withdraw() method to prevent withdrawals if the account balance falls below one hundred.

*/

// Base class: BankAccount
class BankAccount {
    private String accountHolderName;
    private double balance;

    // Constructor
    public BankAccount(String accountHolderName, double initialBalance) {
        this.accountHolderName = accountHolderName;
        this.balance = initialBalance;
    }

    // Method to deposit money
    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Deposited: ₹" + amount);
            System.out.println("Current Balance: ₹" + balance);
        } else {
            System.out.println("Invalid deposit amount.");
        }
    }

    // Method to withdraw money
    public void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            System.out.println("Withdrew: ₹" + amount);
            System.out.println("Current Balance: ₹" + balance);
        } else {
```

```
            System.out.println("Insufficient funds or invalid withdrawal amount.");
        }
    }

    // Getter for balance
    public double getBalance() {
        return balance;
    }
}

// Subclass: SavingsAccount
class SavingsAccount extends BankAccount {
    // Constructor
    public SavingsAccount(String accountHolderName, double initialBalance) {
        super(accountHolderName, initialBalance);
    }

    // Overriding the withdraw method
    @Override
    public void withdraw(double amount) {
        if (getBalance() - amount < 100) {
            System.out.println("Withdrawal denied. Balance cannot fall below
₹100.");
        } else {
            super.withdraw(amount);
        }
    }
}
public class Practical9 {
    public static void main(String[] args) {
        SavingsAccount account = new SavingsAccount("John Doe", 500);
        account.deposit(200);
        account.withdraw(550); // Withdrawal denied
        account.withdraw(50);  // Successful withdrawal
    }
}
```

```
/*
Write a Java program to create a class known as Person with methods called
getFirstName() and getLastName(). Create a subclass called Employee that adds
a new method named getEmployeeId() and overrides the getLastName() method
to include the employee's job title.*/
class Person
{
   private String firstName;
   private String lastName;

   public Person(String firstName, String lastName) {
      this.firstName = firstName;
      this.lastName = lastName;
   }

   public String getFirstName() {
      return firstName;
   }

   public String getLastName() {
      return lastName;
   }
}


class Employee extends Person {
   private String employeeId;
   private String jobTitle;


   public Employee(String firstName, String lastName, String employeeId,
String jobTitle) {
      super(firstName, lastName);
      this.employeeId = employeeId;
      this.jobTitle = jobTitle;
   }
```

```
    public String getEmployeeId() {
        return employeeId;
    }

    public String getLastName() {
        return super.getLastName() + " (" + jobTitle + ")";
    }
}

public class Practical10{
    public static void main(String[] args) {

        Person person = new Person("Alice", "Johnson");

        Employee employee = new Employee("Bob", "Smith", "E12345",
"Software Engineer");

        System.out.println("Person Details:");
        System.out.println("First Name: " + person.getFirstName());
        System.out.println("Last Name: " + person.getLastName());

        System.out.println("\nEmployee Details:");
        System.out.println("First Name: " + employee.getFirstName());
        System.out.println("Last Name: " + employee.getLastName());
        System.out.println("Employee ID: " + employee.getEmployeeId());
    }
}
```

```
/*
11. Design a base class shape with two double type values and member
functions to input the data and compute_area() for calculating area of shape.
Derive two classes: triangle and rectangle. Make compute_area() as abstract
function and redefine this function in the derived class to suit their
requirements. Write a program that accepts dimensions of triangle/rectangle and
display calculated area. Implement dynamic binding for given case study.
*/

import java.util.Scanner;

// Abstract base class Shape
abstract class Shape {
    double dimension1, dimension2;

    // Method to input data
    void inputData() {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the first dimension:");
        dimension1 = sc.nextDouble();
        System.out.println("Enter the second dimension:");
        dimension2 = sc.nextDouble();
    }

    abstract void computeArea();
}

class Triangle extends Shape {
    void computeArea() {
        double area = 0.5 * dimension1 * dimension2;
        System.out.println("Area of Triangle: " + area);
    }
}

class Rectangle extends Shape {
    void computeArea() {
        double area = dimension1 * dimension2;
```

```
        System.out.println("Area of Rectangle: " + area);
    }
}

// Main class
public class Practical11 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // Dynamic binding
        Shape shape;

        System.out.println("Choose the shape to calculate area:");
        System.out.println("1. Triangle");
        System.out.println("2. Rectangle");
        int choice = sc.nextInt();

        if (choice == 1) {
            shape = new Triangle();
        } else if (choice == 2) {
            shape = new Rectangle();
        } else {
            System.out.println("Invalid choice. Exiting.");
            return;
        }

        shape.inputData();
        shape.computeArea();
    }
}
```

```
/*
Design and develop a context for given case study and implement an interface for Vehicles Consider the example of vehicles like bicycle, car and bike. All Vehicles have common functionalities such as Gear Change, Speed up and apply breaks. Make an interface and put all these common functionalities. Bicycle, Bike, Car classes should be implemented for all these functionalities in their own class in their own way.
*/

interface Vehicle {
    void changeGear(int gear);
    void speedUp(int increment);
    void applyBrakes(int decrement);
}

// Class for Bicycle
class Bicycle implements Vehicle {
    private int gear;
    private int speed;

    @Override
    public void changeGear(int gear) {
        this.gear = gear;
        System.out.println("Bicycle gear changed to: " + this.gear);
    }

    @Override
    public void speedUp(int increment) {
        speed += increment;
        System.out.println("Bicycle speed increased to: " + speed + " km/h");
    }

    @Override
    public void applyBrakes(int decrement) {
        speed -= decrement;
        if (speed < 0) speed = 0;
        System.out.println("Bicycle speed reduced to: " + speed + " km/h");
```

```
    }
}

// Class for Bike
class Bike implements Vehicle {
    private int gear;
    private int speed;

    @Override
    public void changeGear(int gear) {
        this.gear = gear;
        System.out.println("Bike gear changed to: " + this.gear);
    }

    @Override
    public void speedUp(int increment) {
        speed += increment;
        System.out.println("Bike speed increased to: " + speed + " km/h");
    }

    @Override
    public void applyBrakes(int decrement) {
        speed -= decrement;
        if (speed < 0) speed = 0;
        System.out.println("Bike speed reduced to: " + speed + " km/h");
    }
}

// Class for Car
class Car implements Vehicle {
    private int gear;
    private int speed;

    @Override
    public void changeGear(int gear) {
        this.gear = gear;
        System.out.println("Car gear changed to: " + this.gear);
```

```
    }

    @Override
    public void speedUp(int increment) {
        speed += increment;
        System.out.println("Car speed increased to: " + speed + " km/h");
    }

    @Override
    public void applyBrakes(int decrement) {
        speed -= decrement;
        if (speed < 0) speed = 0;
        System.out.println("Car speed reduced to: " + speed + " km/h");
    }
}

public class Practical12
 {
    public static void main(String[] args) {
        Vehicle bicycle = new Bicycle();
        Vehicle bike = new Bike();
        Vehicle car = new Car();

        System.out.println("Testing Bicycle:");
        bicycle.changeGear(2);
        bicycle.speedUp(15);
        bicycle.applyBrakes(5);

        System.out.println("\nTesting Bike:");
        bike.changeGear(3);
        bike.speedUp(30);
        bike.applyBrakes(10);
        System.out.println("\nTesting Car:");
        car.changeGear(4);
        car.speedUp(60);
        car.applyBrakes(20);
    }  }
```

```
/*
13. Write a Java program to create a Animal interface with a method called bark() that
takes no arguments and returns void. Create a Dog class that implements Animal and
overrides speak() to print "Dog is barking".
*/
// Animal interface with a bark method
interface Animal {
    void bark();  // Abstract method that must be implemented by any class that implements Animal
}

// Dog class that implements the Animal interface
class Dog implements Animal {
    // Overriding the bark method to print "Dog is barking"
    @Override
    public void bark() {
        System.out.println("Dog is barking");
    }
}

// Main class to test the Dog class and the Animal interface
public class Practical13 {
    public static void main(String[] args) {
        // Create an instance of Dog
        Animal dog = new Dog();

        // Call the bark method
        dog.bark();  // Output: Dog is barking
    }
}
```

```
/*
14. Write a Java programming to create a banking system with three classes - Bank, Account, SavingsAccount, and CurrentAccount. The bank should have a list of accounts and methods for adding them. Accounts should be an interface with methods to deposit, withdraw, calculate interest, and view balances. SavingsAccount and CurrentAccount should implement the Account interface and have their own unique methods.
*/
import java.util.ArrayList;
import java.util.List;

// Account Interface with common methods for bank accounts
interface Account {
    void deposit(double amount);
    void withdraw(double amount);
    void calculateInterest();
    double getBalance();
}

// SavingsAccount class implementing the Account interface
class SavingsAccount implements Account {
    private double balance;
    private double interestRate = 0.04;  // 4% interest rate

    public SavingsAccount(double initialBalance) {
        this.balance = initialBalance;
    }

    @Override
    public void deposit(double amount) {
        balance += amount;
        System.out.println("Deposited " + amount + " into Savings Account.");
    }

    @Override
    public void withdraw(double amount) {
        if (amount > balance) {
```

```
            System.out.println("Insufficient funds in Savings Account.");
        } else {
            balance -= amount;
            System.out.println("Withdrew " + amount + " from Savings Account.");
        }
    }

    @Override
    public void calculateInterest() {
        double interest = balance * interestRate;
        balance += interest;
        System.out.println("Interest of " + interest + " added to Savings
Account.");
    }

    @Override
    public double getBalance() {
        return balance;
    }
}

// CurrentAccount class implementing the Account interface
class CurrentAccount implements Account {
    private double balance;
    private double overdraftLimit = 1000.00;  // Overdraft limit for Current
Account

    public CurrentAccount(double initialBalance) {
        this.balance = initialBalance;
    }

    @Override
    public void deposit(double amount) {
        balance += amount;
        System.out.println("Deposited " + amount + " into Current Account.");
    }
```

```
    @Override
    public void withdraw(double amount) {
        if (amount > (balance + overdraftLimit)) {
            System.out.println("Insufficient funds in Current Account, even with
overdraft.");
        } else {
            balance -= amount;
            System.out.println("Withdrew " + amount + " from Current Account.");
        }
    }

    @Override
    public void calculateInterest() {
        // Current Account does not have interest, so no interest is calculated
        System.out.println("Current Account does not earn interest.");
    }

    @Override
    public double getBalance() {
        return balance;
    }
}

// Bank class to manage multiple accounts
class Bank {
    private List<Account> accounts = new ArrayList<>();

    // Method to add an account to the bank
    public void addAccount(Account account) {
        accounts.add(account);
        System.out.println("Account added to the bank.");
    }

    // Method to display all account balances
    public void displayBalances() {
        for (Account account : accounts) {
            System.out.println("Account balance: " + account.getBalance());
```

```
        }
    }
}

// Main class to test the banking system
public class Practical14 {
    public static void main(String[] args) {
        // Create Bank instance
        Bank bank = new Bank();

        // Create different types of accounts
        SavingsAccount savingsAccount = new SavingsAccount(5000.00);
        CurrentAccount currentAccount = new CurrentAccount(2000.00);

        // Add accounts to the bank
        bank.addAccount(savingsAccount);
        bank.addAccount(currentAccount);

        // Perform operations on the Savings Account
        savingsAccount.deposit(1500.00);
        savingsAccount.withdraw(2000.00);
        savingsAccount.calculateInterest();
        System.out.println("Savings Account Balance: " +
savingsAccount.getBalance());

        // Perform operations on the Current Account
        currentAccount.deposit(1000.00);
        currentAccount.withdraw(3000.00);
        currentAccount.calculateInterest();
        System.out.println("Current Account Balance: " +
currentAccount.getBalance());

        // Display all account balances in the bank
        bank.displayBalances();
    }
}
```

```
/*
15. Write a Java program to create an interface Playable with a method play() that takes no arguments and returns void. Create three classes Football, Volleyball, and Basketball that implement the Playable interface and override the play() method to play the respective sports
*/
// Interface Playable with a method play()
interface Playable {
    void play();  // Abstract method to play a sport
}

// Class Football implementing the Playable interface
class Football implements Playable {
    @Override
    public void play() {
        System.out.println("Playing Football.");
    }
}

// Class Volleyball implementing the Playable interface
class Volleyball implements Playable {
    @Override
    public void play() {
        System.out.println("Playing Volleyball.");
    }
}

// Class Basketball implementing the Playable interface
class Basketball implements Playable {
    @Override
    public void play() {
        System.out.println("Playing Basketball.");
    }
}

// Main class to test the Playable interface and its implementations
public class Practical15 {
```

```
    public static void main(String[] args) {
        // Create objects of Football, Volleyball, and Basketball
        Playable football = new Football();
        Playable volleyball = new Volleyball();
        Playable basketball = new Basketball();

        // Call the play() method for each sport
        football.play();      // Output: Playing Football.
        volleyball.play();    // Output: Playing Volleyball.
        basketball.play();    // Output: Playing Basketball.
    }
}
```

```
/*
16. Write a Java program to create an interface Drawable with a method draw() that takes no arguments and returns void. Create three classes Circle, Rectangle, and Triangle that implement the Drawable interface and override the draw() method to draw their respective shapes
*/

interface Drawable {
    void draw();
}


class Circle implements Drawable {
    @Override
    public void draw() {
        System.out.println("Drawing a Circle.");
    }
}

// Class Rectangle implementing the Drawable interface
class Rectangle implements Drawable {
    @Override
    public void draw() {
        System.out.println("Drawing a Rectangle.");
    }
}

// Class Triangle implementing the Drawable interface
class Triangle implements Drawable {
    @Override
    public void draw() {
        System.out.println("Drawing a Triangle.");
    }
}


public class Practical16 {
```

```
    public static void main(String[] args) {
        // Create objects of Circle, Rectangle, and Triangle
        Drawable circle = new Circle();
        Drawable rectangle = new Rectangle();
        Drawable triangle = new Triangle();

        // Call the draw() method for each shape
        circle.draw();      // Output: Drawing a Circle.
        rectangle.draw();   // Output: Drawing a Rectangle.
        triangle.draw();    // Output: Drawing a Triangle.
    }
}
```

```
/* Implement a program to handle Arithmetic exception, Array Index Out of Bounds. The user enters two numbers Num1 and Num2. The division of Num1 and Num2 is displayed. If Num1 and Num2 are not integers, the program would throw a Number Format Exception. If Num2 were zero, the program would throw an ArithmeticException. Display the exception
*/
import java.util.Scanner;

public class Practical17 {

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

        try {
            // Input two numbers from the user
            System.out.print("Enter the first number (Num1): ");
            String input1 = sc.nextLine();
            System.out.print("Enter the second number (Num2): ");
            String input2 = sc.nextLine();

            // Try to parse the inputs into integers
            int num1 = Integer.parseInt(input1); // May throw NumberFormatException
            int num2 = Integer.parseInt(input2); // May throw NumberFormatException

            // Perform division (may throw ArithmeticException if num2 is zero)
            int result = num1 / num2;
            System.out.println("Result of division: " + result);

            // Example of ArrayIndexOutOfBoundsException handling
            int[] arr = new int[5];
            System.out.println("Accessing array element at index 6: " + arr[6]); // May throw ArrayIndexOutOfBoundsException

        } catch (NumberFormatException e) {
            // Handle the case when input is not a valid integer
```

```
            System.out.println("Exception: Invalid input, please enter integers
only.");
        } catch (ArithmeticException e) {
            // Handle the case when dividing by zero
            System.out.println("Exception: Cannot divide by zero.");
        } catch (ArrayIndexOutOfBoundsException e) {
            // Handle array index out of bounds exception
            System.out.println("Exception: Array index is out of bounds.");
        } catch (Exception e) {
            // Catch any other exceptions that might occur
            System.out.println("Exception: " + e.getMessage());
        } finally {
            sc.close(); // Close the scanner resource
            System.out.println("Execution completed.");
        }
    }
}
```

```
/*
18. Write a Java program that reads a list of integers from the user and throws
anexception if any numbers are duplicates
*/
import java.util.*;

class DuplicateNumberException extends Exception {
    public DuplicateNumberException(String message) {
        super(message);
    }
}

public class Practical18 {

    // Method to read integers and check for duplicates
    public static void checkForDuplicates(List<Integer> numbers) throws
DuplicateNumberException {
        Set<Integer> uniqueNumbers = new HashSet<>();

        // Check if any number is duplicated
        for (Integer number : numbers) {
            if (!uniqueNumbers.add(number)) {
                // If add() returns false, the number is already in the set, so it's a
duplicate
                throw new DuplicateNumberException("Duplicate number found: " +
number);
            }
        }

        // If no duplicates, print the list
        System.out.println("The list contains no duplicates.");
    }

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

        // List to store the user input
```

```
        List<Integer> numbers = new ArrayList<>();

        System.out.println("Enter the number of integers you want to input:");
        int n = sc.nextInt();

        System.out.println("Enter the integers:");

        // Read integers from the user
        for (int i = 0; i < n; i++) {
            int num = sc.nextInt();
            numbers.add(num);
        }

        // Check for duplicates and handle exception
        try {
            checkForDuplicates(numbers);
        } catch (DuplicateNumberException e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
```

```
/*19. Write a Java program that reads a file and throws an exception if the file is empty.
*/
import java.io.*;
import java.nio.file.Files;
import java.nio.file.Paths;

// Custom exception class to handle empty file scenario
class EmptyFileException extends Exception {
    public EmptyFileException(String message) {
        super(message);
    }
}

public class Practical19 {

    // Method to read the file and throw exception if it's empty
    public static void readFile(String filePath) throws EmptyFileException {
        try {
            // Read all lines from the file
            File file = new File(filePath);

            // Check if the file exists
            if (!file.exists()) {
                System.out.println("File not found!");
                return;
            }

            // Read the content of the file
            String content = new String(Files.readAllBytes(Paths.get(filePath)));

            // Check if the content is empty
            if (content.trim().isEmpty()) {
                throw new EmptyFileException("The file is empty!");
            } else {
                System.out.println("File content:");
```

```
            System.out.println(content);
        }
    } catch (IOException e) {
        System.out.println("An error occurred while reading the file: " +
e.getMessage());
    }
  }

  public static void main(String[] args) {
    // Specify the file path (change this path to an existing file in your system)
    String filePath = "test.txt"; // Change this to the path of the file you want to
read

    try {
      // Call the method to read the file
      readFile(filePath);
    } catch (EmptyFileException e) {
      System.out.println("Exception: " + e.getMessage());
    }
  }
}
```

```
/*
 Write a Java program to create a method that takes a string as input and throws an exception if the string does not contain vowels.
*/
// Custom Exception class
class NoVowelsException extends Exception {
    public NoVowelsException(String message) {
        super(message);
    }
}

public class Practical20 {

    // Method to check if the string contains vowels
    public static void checkVowels(String input) throws NoVowelsException {
        // Convert the string to lowercase to make it case insensitive
        input = input.toLowerCase();

        // Check if the string contains vowels
        boolean containsVowel = false;
        for (int i = 0; i < input.length(); i++) {
            char ch = input.charAt(i);
            if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {
                containsVowel = true;
                break;  // No need to check further once we find a vowel
            }
        }

        // If no vowel is found, throw the custom exception
        if (!containsVowel) {
            throw new NoVowelsException("The string does not contain any vowels.");
        } else {
            System.out.println("The string contains vowels.");
        }
    }
```

```
    public static void main(String[] args) {
        // Test the method
        String testString = "rhythm"; // Change this to test different inputs

        try {
            checkVowels(testString);
        } catch (NoVowelsException e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
```

```
/*
23. Using concepts of Object-Oriented programming develop solution for an application 1) Banking system having following operations : 1. Create an account 2. Deposit money 3. Withdraw money 4. Honor daily withdrawal limit 5. Check the balance 6. Display Account information.
*/

import java.util.Scanner;

// Class to represent a Bank Account
class BankAccount {
    private int accountNumber;
    private String accountHolderName;
    private double balance;
    private static final double DAILY_WITHDRAWAL_LIMIT = 50000; // Example withdrawal limit

    // Constructor to initialize the bank account
    public BankAccount(int accountNumber, String accountHolderName, double initialDeposit) {
        this.accountNumber = accountNumber;
        this.accountHolderName = accountHolderName;
        this.balance = initialDeposit;
    }

    // Method to deposit money
    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Deposited ₹" + amount + ". New balance: ₹" + balance);
        } else {
            System.out.println("Deposit amount must be positive.");
        }
    }

    // Method to withdraw money with honor for daily withdrawal limit
```

```
    public void withdraw(double amount) {
        if (amount <= 0) {
            System.out.println("Withdrawal amount must be positive.");
        } else if (amount > balance) {
            System.out.println("Insufficient funds for withdrawal.");
        } else if (amount > DAILY_WITHDRAWAL_LIMIT) {
            System.out.println("Cannot withdraw more than ₹" +
DAILY_WITHDRAWAL_LIMIT + " in a day.");
        } else {
            balance -= amount;
            System.out.println("Withdrawn ₹" + amount + ". New balance: ₹" +
balance);
        }
    }

    // Method to check the balance
    public void checkBalance() {
        System.out.println("Current balance: ₹" + balance);
    }

    // Method to display account information
    public void displayAccountInfo() {
        System.out.println("Account Number: " + accountNumber);
        System.out.println("Account Holder Name: " + accountHolderName);
        System.out.println("Current Balance: ₹" + balance);
    }
}

// Main Class
public class Practical23 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Creating an account
        System.out.print("Enter account number: ");
        int accountNumber = scanner.nextInt();
        scanner.nextLine(); // Consume newline character
```

```
        System.out.print("Enter account holder name: ");
        String accountHolderName = scanner.nextLine();
        System.out.print("Enter initial deposit amount: ₹");
        double initialDeposit = scanner.nextDouble();

        BankAccount account = new BankAccount(accountNumber,
accountHolderName, initialDeposit);

        // Menu-driven system for banking operations
        while (true) {
            System.out.println("\n--- Banking System ---");
            System.out.println("1. Create Account");
            System.out.println("2. Deposit Money");
            System.out.println("3. Withdraw Money");
            System.out.println("4. Check Balance");
            System.out.println("5. Display Account Information");
            System.out.println("6. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    // Account is created during object instantiation, so no need to
recreate it here.
                    System.out.println("Account already created.");
                    break;

                case 2:
                    System.out.print("Enter amount to deposit: ₹");
                    double depositAmount = scanner.nextDouble();
                    account.deposit(depositAmount);
                    break;

                case 3:
                    System.out.print("Enter amount to withdraw: ₹");
                    double withdrawalAmount = scanner.nextDouble();
                    account.withdraw(withdrawalAmount);
```

```
                break;

            case 4:
                account.checkBalance();
                break;

            case 5:
                account.displayAccountInfo();
                break;

            case 6:
                System.out.println("Exiting the system.");
                scanner.close();
                return;

            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }
  }
}
```

```
/*
Using concepts of Object-Oriented programming develop solution for any an application Inventory management system having following operations : 1. List of all products 2. Display individual product information 3. Purchase 4. Shipping 5. Balance stock 6. Loss and Profit calculation.
*/
import java.util.*;

// Class to represent a Product
class Product {
    private int productId;
    private String name;
    private double price;
    private int stock;
    private int sold;

    public Product(int productId, String name, double price, int stock) {
        this.productId = productId;
        this.name = name;
        this.price = price;
        this.stock = stock;
        this.sold = 0;
    }

    // Getters
    public int getProductId() {
        return productId;
    }

    public String getName() {
        return name;
    }

    public double getPrice() {
        return price;
    }
```

```
    public int getStock() {
        return stock;
    }

    public int getSold() {
        return sold;
    }

    // Purchase a product
    public boolean purchase(int quantity) {
        if (quantity <= stock) {
            stock -= quantity;
            sold += quantity;
            System.out.println(quantity + " units of " + name + " purchased
successfully!");
            return true;
        } else {
            System.out.println("Insufficient stock for " + name + ".");
            return false;
        }
    }

    // Display product information
    public void displayInfo() {
        System.out.println("Product ID: " + productId);
        System.out.println("Name: " + name);
        System.out.println("Price: ₹" + price);
        System.out.println("Stock: " + stock);
        System.out.println("Sold: " + sold);
    }

    // Calculate total revenue from sold items
    public double calculateRevenue() {
        return sold * price;
    }
}
```

```
// Class for Inventory Management
class Inventory {
    private List<Product> products;

    public Inventory() {
        products = new ArrayList<>();
    }

    // Add a product to inventory
    public void addProduct(Product product) {
        products.add(product);
    }

    // List all products
    public void listAllProducts() {
        System.out.println("All Products:");
        for (Product product : products) {
            System.out.println(product.getProductId() + ". " + product.getName());
        }
    }

    // Display individual product information
    public void displayProductInfo(int productId) {
        for (Product product : products) {
            if (product.getProductId() == productId) {
                product.displayInfo();
                return;
            }
        }
        System.out.println("Product with ID " + productId + " not found.");
    }

    // Handle purchase
    public void purchaseProduct(int productId, int quantity) {
        for (Product product : products) {
            if (product.getProductId() == productId) {
                product.purchase(quantity);
```

```
                return;
            }
        }
        System.out.println("Product with ID " + productId + " not found.");
    }

    // Calculate total balance stock
    public void calculateBalanceStock() {
        System.out.println("Balance Stock:");
        for (Product product : products) {
            System.out.println(product.getName() + ": " + product.getStock() + "
units");
        }
    }

    // Calculate profit
    public void calculateProfit() {
        double totalProfit = 0;
        for (Product product : products) {
            totalProfit += product.calculateRevenue();
        }
        System.out.println("Total Revenue: ₹" + totalProfit);
    }
}

// Main Class
public class Practical24 {
    public static void main(String[] args) {
        Inventory inventory = new Inventory();

        // Adding products to inventory
        inventory.addProduct(new Product(1, "Laptop", 50000, 10));
        inventory.addProduct(new Product(2, "Smartphone", 15000, 20));
        inventory.addProduct(new Product(3, "Tablet", 20000, 15));

        Scanner sc = new Scanner(System.in);
```

```
while (true) {
    System.out.println("\nInventory Management System:");
    System.out.println("1. List All Products");
    System.out.println("2. Display Product Information");
    System.out.println("3. Purchase Product");
    System.out.println("4. Display Balance Stock");
    System.out.println("5. Calculate Profit");
    System.out.println("6. Exit");
    System.out.print("Enter your choice: ");
    int choice = sc.nextInt();

    switch (choice) {
        case 1:
            inventory.listAllProducts();
            break;
        case 2:
            System.out.print("Enter Product ID: ");
            int productId = sc.nextInt();
            inventory.displayProductInfo(productId);
            break;
        case 3:
            System.out.print("Enter Product ID: ");
            productId = sc.nextInt();
            System.out.print("Enter Quantity: ");
            int quantity = sc.nextInt();
            inventory.purchaseProduct(productId, quantity);
            break;
        case 4:
            inventory.calculateBalanceStock();
            break;
        case 5:
            inventory.calculateProfit();
            break;
        case 6:
            System.out.println("Exiting...");
            sc.close();
            return;
```

```
                default:
                    System.out.println("Invalid choice! Please try again.");
            }
        }
    }
}
```

```
/*
Implement Factory design pattern for the given context. Consider Car building process, which requires many steps from allocating accessories to final makeup. These steps should be written as methods and should be called while creating an instance of a specific car type. Hatchback, Sedan, SUV could be the subclasses of Car class. Car class and its subclasses, CarFactory and Test Factory Pattern should be implemented.
*/
// Abstract base class: Car
abstract class Car {
    public Car() {
        prepareAccessories();
        buildBody();
        addEngine();
        paintCar();
    }

    abstract void prepareAccessories();
    abstract void buildBody();
    abstract void addEngine();
    abstract void paintCar();
}

// Subclass: Hatchback
class Hatchback extends Car {
    @Override
    void prepareAccessories() {
        System.out.println("Preparing accessories for Hatchback...");
    }

    @Override
    void buildBody() {
        System.out.println("Building body for Hatchback...");
    }

    @Override
    void addEngine() {
```

```
        System.out.println("Adding engine to Hatchback...");
    }

    @Override
    void paintCar() {
        System.out.println("Painting Hatchback...");
    }
}

// Subclass: Sedan
class Sedan extends Car {
    @Override
    void prepareAccessories() {
        System.out.println("Preparing accessories for Sedan...");
    }

    @Override
    void buildBody() {
        System.out.println("Building body for Sedan...");
    }

    @Override
    void addEngine() {
        System.out.println("Adding engine to Sedan...");
    }

    @Override
    void paintCar() {
        System.out.println("Painting Sedan...");
    }
}

// Subclass: SUV
class SUV extends Car {
    @Override
    void prepareAccessories() {
        System.out.println("Preparing accessories for SUV...");
```

```
    }

    @Override
    void buildBody() {
        System.out.println("Building body for SUV...");
    }

    @Override
    void addEngine() {
        System.out.println("Adding engine to SUV...");
    }

    @Override
    void paintCar() {
        System.out.println("Painting SUV...");
    }
}

// Factory class: CarFactory
class CarFactory {
    public static Car createCar(String type) {
        switch (type.toLowerCase()) {
            case "hatchback":
                return new Hatchback();
            case "sedan":
                return new Sedan();
            case "suv":
                return new SUV();
            default:
                throw new IllegalArgumentException("Unknown car type: " + type);
        }
    }
}

// Test Factory Pattern
public class Practical25 {
    public static void main(String[] args) {
```

```
        System.out.println("Building a Hatchback:");
        Car hatchback = CarFactory.createCar("hatchback");

        System.out.println("\nBuilding a Sedan:");
        Car sedan = CarFactory.createCar("sedan");

        System.out.println("\nBuilding an SUV:");
        Car suv = CarFactory.createCar("suv");
    }
}
```

```
/*
 Implement and apply Strategy Design pattern for simple Shopping Cart where three payment strategies are used such as Credit Card, PayPal, Bit Coin. Create an interface for strategy pattern and give concrete implementation for payment.
*/

// Payment Strategy Interface
interface PaymentStrategy {
    void pay(double amount);
}

// Concrete Implementation: CreditCardPayment
class CreditCardPayment implements PaymentStrategy {
    private String name;
    private String cardNumber;
    private String cvv;
    private String expiryDate;

    public CreditCardPayment(String name, String cardNumber, String cvv, String expiryDate) {
        this.name = name;
        this.cardNumber = cardNumber;
        this.cvv = cvv;
        this.expiryDate = expiryDate;
    }

    @Override
    public void pay(double amount) {
        System.out.println("Paid ₹" + amount + " using Credit Card (Card Number: " + cardNumber + ").");
    }
}

// Concrete Implementation: PayPalPayment
class PayPalPayment implements PaymentStrategy {
    private String email;
```

```
    public PayPalPayment(String email) {
        this.email = email;
    }

    @Override
    public void pay(double amount) {
        System.out.println("Paid ₹" + amount + " using PayPal (Email: " + email +
").");
    }
}

// Concrete Implementation: BitcoinPayment
class BitcoinPayment implements PaymentStrategy {
    private String walletAddress;

    public BitcoinPayment(String walletAddress) {
        this.walletAddress = walletAddress;
    }

    @Override
    public void pay(double amount) {
        System.out.println("Paid ₹" + amount + " using Bitcoin (Wallet: " +
walletAddress + ").");
    }
}

// ShoppingCart class
class ShoppingCart {
    private double totalAmount;

    public void addItem(double price) {
        totalAmount += price;
        System.out.println("Item added to cart. Price: ₹" + price);
    }

    public void checkout(PaymentStrategy paymentStrategy) {
        System.out.println("Total Amount: ₹" + totalAmount);
```

```
        paymentStrategy.pay(totalAmount);
    }
}

// Main Class
public class Practical26 {
    public static void main(String[] args) {
        ShoppingCart cart = new ShoppingCart();

        // Adding items to the shopping cart
        cart.addItem(500.00);
        cart.addItem(1200.00);
        cart.addItem(300.00);

        // Using Credit Card Payment Strategy
        System.out.println("\nPayment using Credit Card:");
        PaymentStrategy creditCardPayment = new CreditCardPayment("John
Doe", "1234-5678-9876-5432", "123", "12/25");
        cart.checkout(creditCardPayment);

        // Using PayPal Payment Strategy
        System.out.println("\nPayment using PayPal:");
        PaymentStrategy payPalPayment = new
PayPalPayment("john.doe@example.com");
        cart.checkout(payPalPayment);

        // Using Bitcoin Payment Strategy
        System.out.println("\nPayment using Bitcoin:");
        PaymentStrategy bitcoinPayment = new
BitcoinPayment("1A2B3C4D5E6F7G8H");
        cart.checkout(bitcoinPayment);
    }
}
```