AMRITA VISHWA VIDYAPEETHAM DEEMED TO BE UNIVERSITY UNDER SECTION 3 OF UGC ACT, 1956 SCHOOL OF COMPUTING

LAB RECORD

23CSE111- OBJECT ORIENTED PROGRAMMING

SUBMITTED BY:

M.CHARULATHA

BACHELOR IN TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

AMRITA VISHWA VIDYAPEETHAM

AMRITA SCHOOL OF COMPUTING

CHENNAI

MARCH-2025

AMRITA VISHWA VIDYAPEETHAM AMRITA SCHOOL OF COMPUTING, CHENNAI

BONAFIDE CERTIFICATE

This is to certify that the Lab Record work for 23CSE111- Object Oriented Programming Subject submitted by ***CH.SC.U4CSE24029 – M. CHARULATHA*** in **“Computer Science and Engineering”** is a Bonafide record of the work carried out under my guidance and supervision at Amrita School of Computing, Chennai.

This Lab examination held on 11/03/2025

Internal Examiner 1 Internal Examiner2

INDEX

S.NO	TITLE	PAGE.NO
	UML DIAGRAM	
1.	**BANKING SYSTEM**	
	1.A) Use Case Diagram	
	1.B) Class Diagram	
	1.C) Sequence Diagram	
	1.D) Activity Diagram	
	1.E) Deployment Diagram	
2.	**LIBRARY MANAGEMENT**	
	2.A) Use Case Diagram	
	2.B) Class Diagram	
	2.C) Sequence Diagram	
	2.D) Activity Diagram	
	2.E) Deployment Diagram	
3.	**BASIC JAVA PROGRAMS**	
	1.Area of triangle	
	2. Average of 3 numbers	
	3.Grading System	
	4.Segregating age	
	5.Electricity Bill	
	6.Finding odd numbers	
	7.Checking for palindrome condition	
	8. Finding factorial of a number	
	9. Calculating even or odd number	
	10. Finding Fibonacci numbers	
	INHERITANCE	
4.	**SINGLE INHERITANCE PROGRAMS**	
	11. Student Management System	
	12. Library Management System	

JAVA-LAB MANUAL

UML DIAGRAMS

1. BANKING SYSTEM

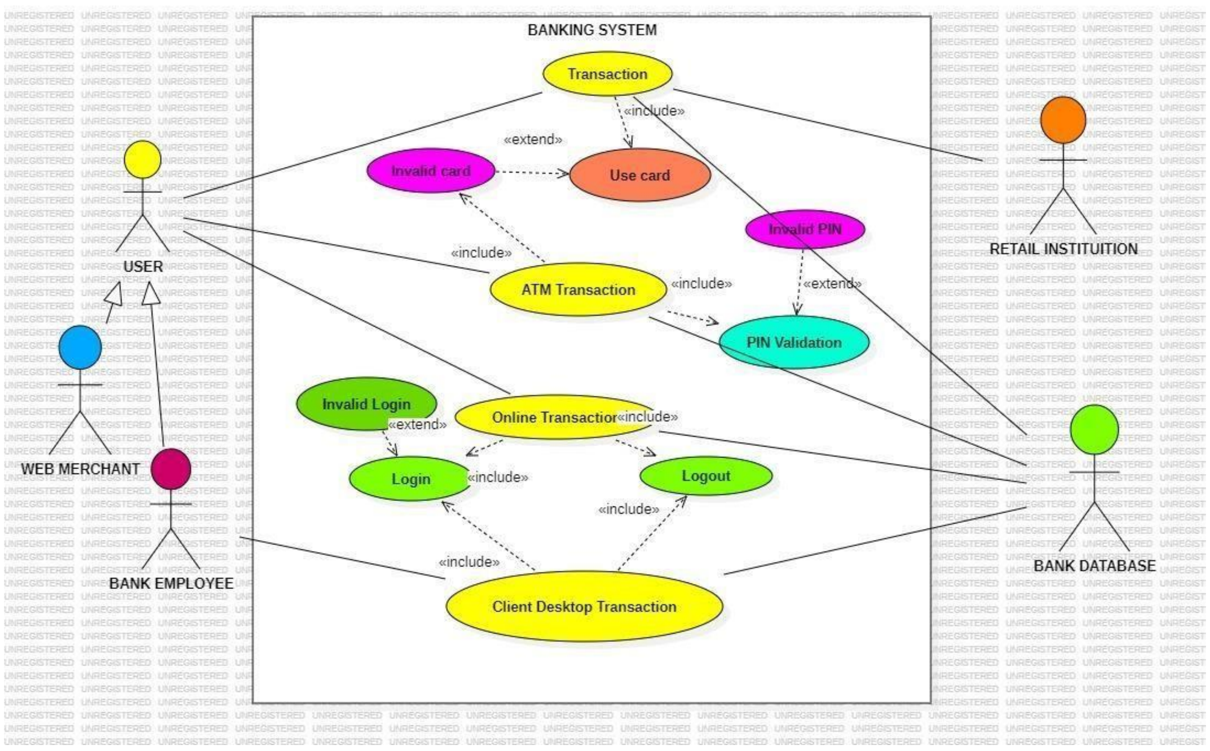
AIM:

To Demonstrate the working of the Banking System using different UML Diagrams.

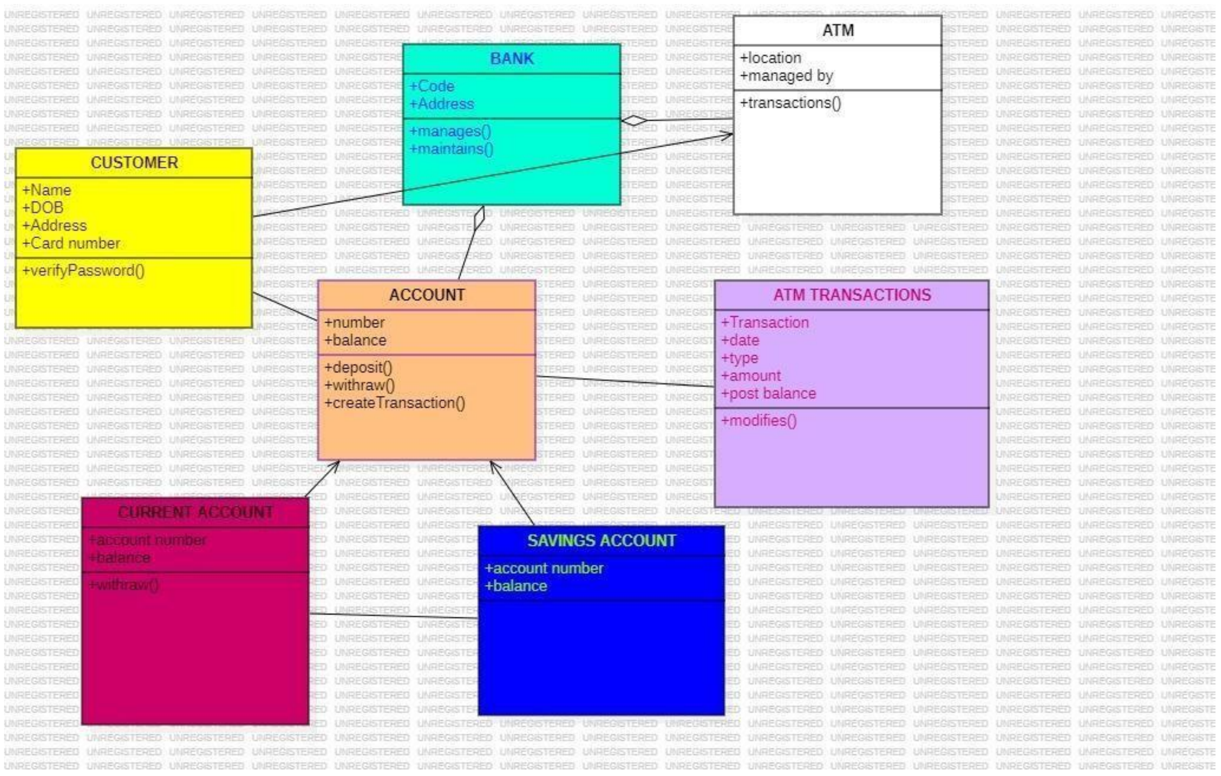
ALGORITHM:

1. Initialize the system
2. User Authentication
3. Main Menu
4. Account Creation
5. Deposit Money
6. Withdraw Money
7. Check Balance
8. Transfer Money
9. Exit System

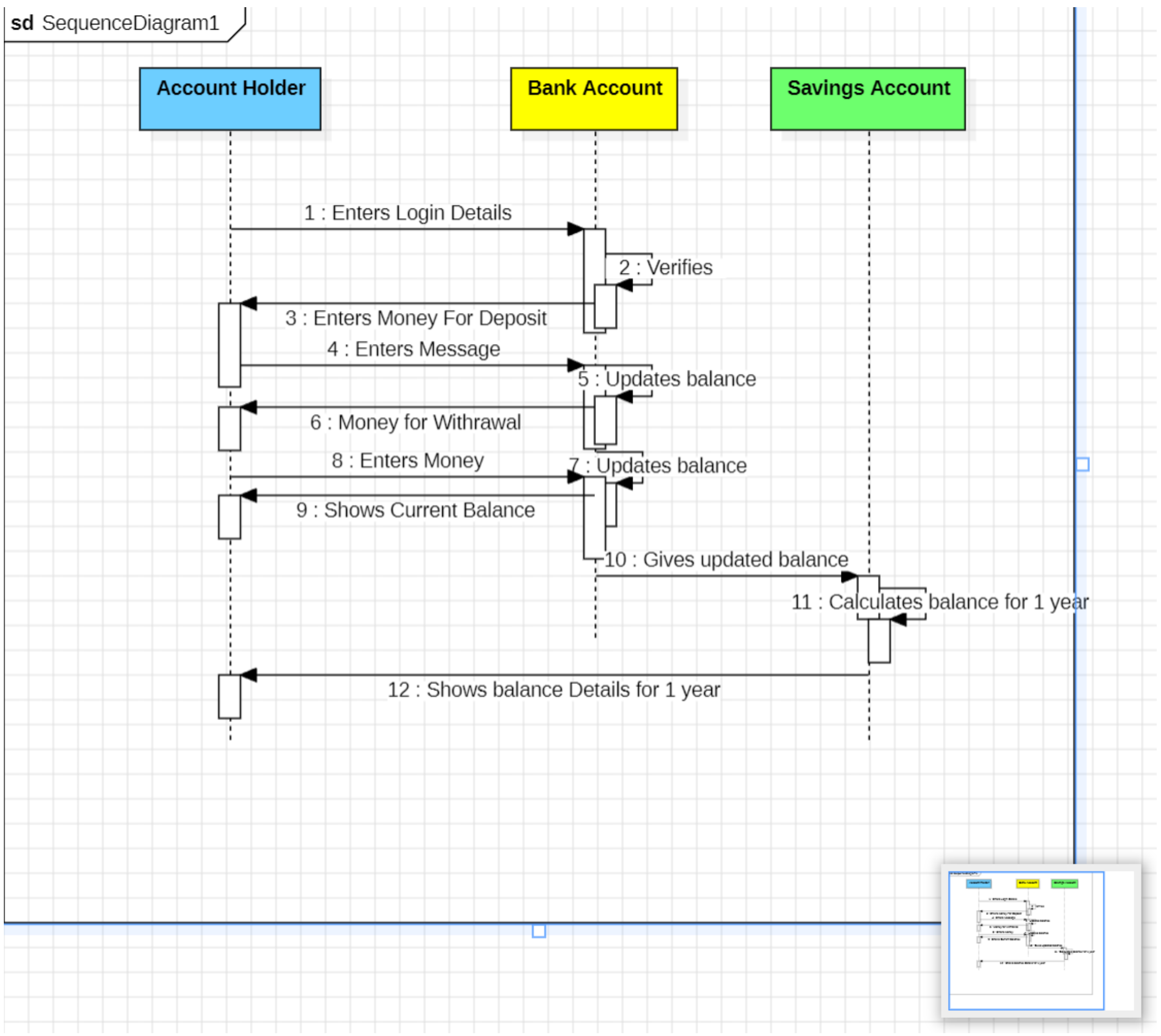
A) Use- Case Diagram



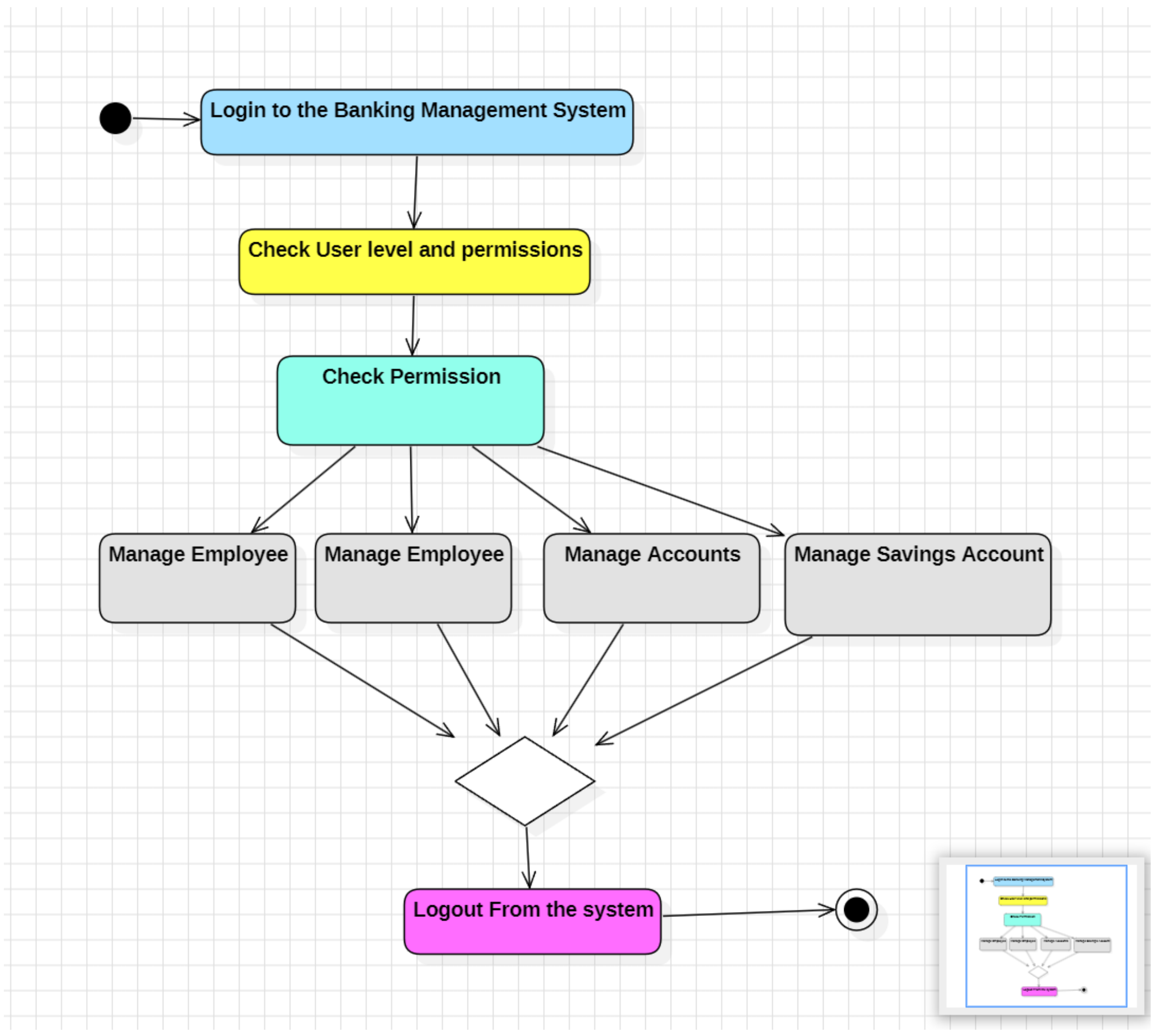
B) Class Diagram



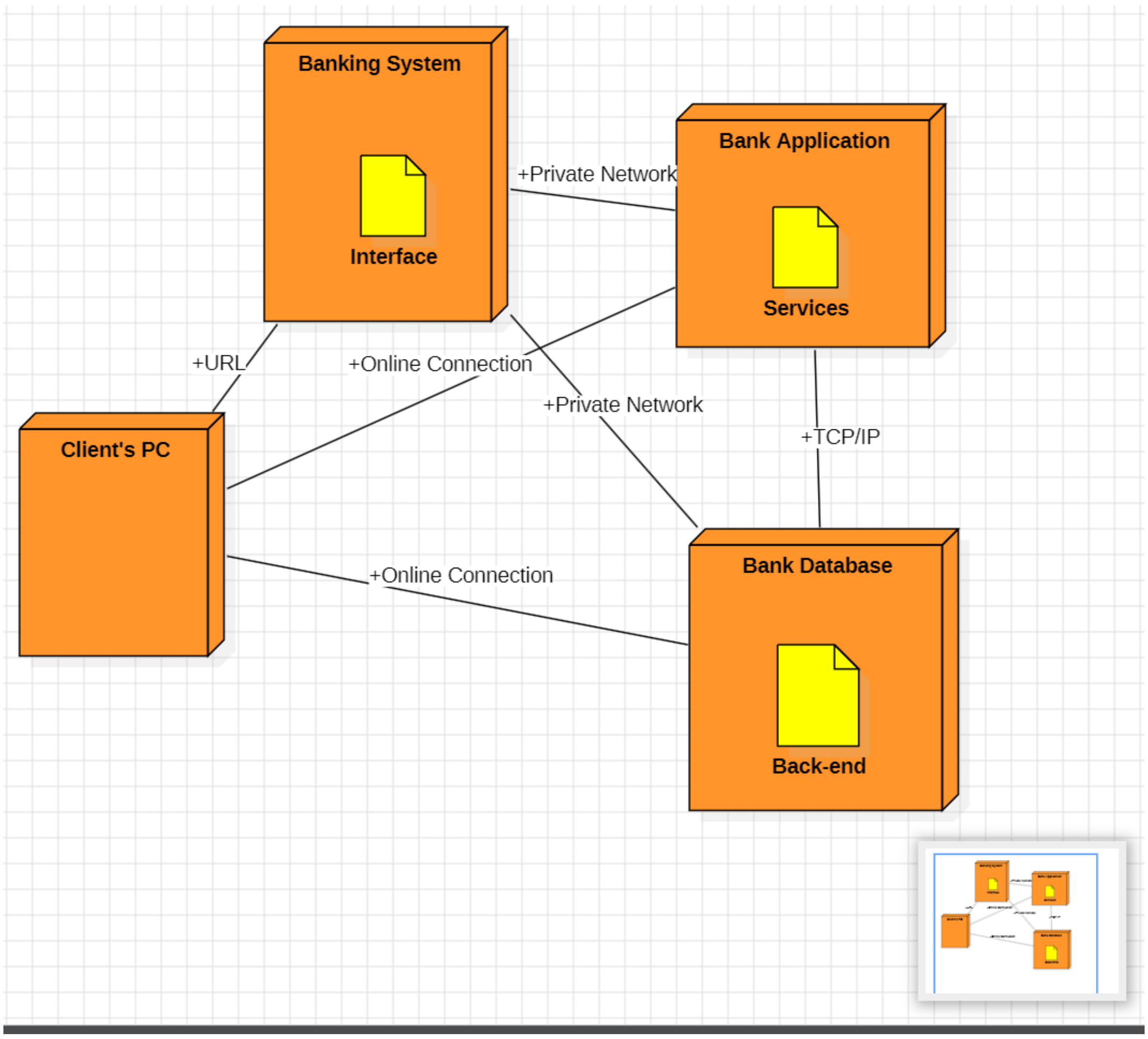
C) Sequence Diagram



D) Activity Diagram



E) Deployment Diagram


Banking System
Interface
Bank Application
Services
+Private Network
+URL
+Online Connection
+Private Network
+TCP/IP
Client's PC
Bank Database
Back-end
+Online Connection

2. LIBRARY MANAGEMENT

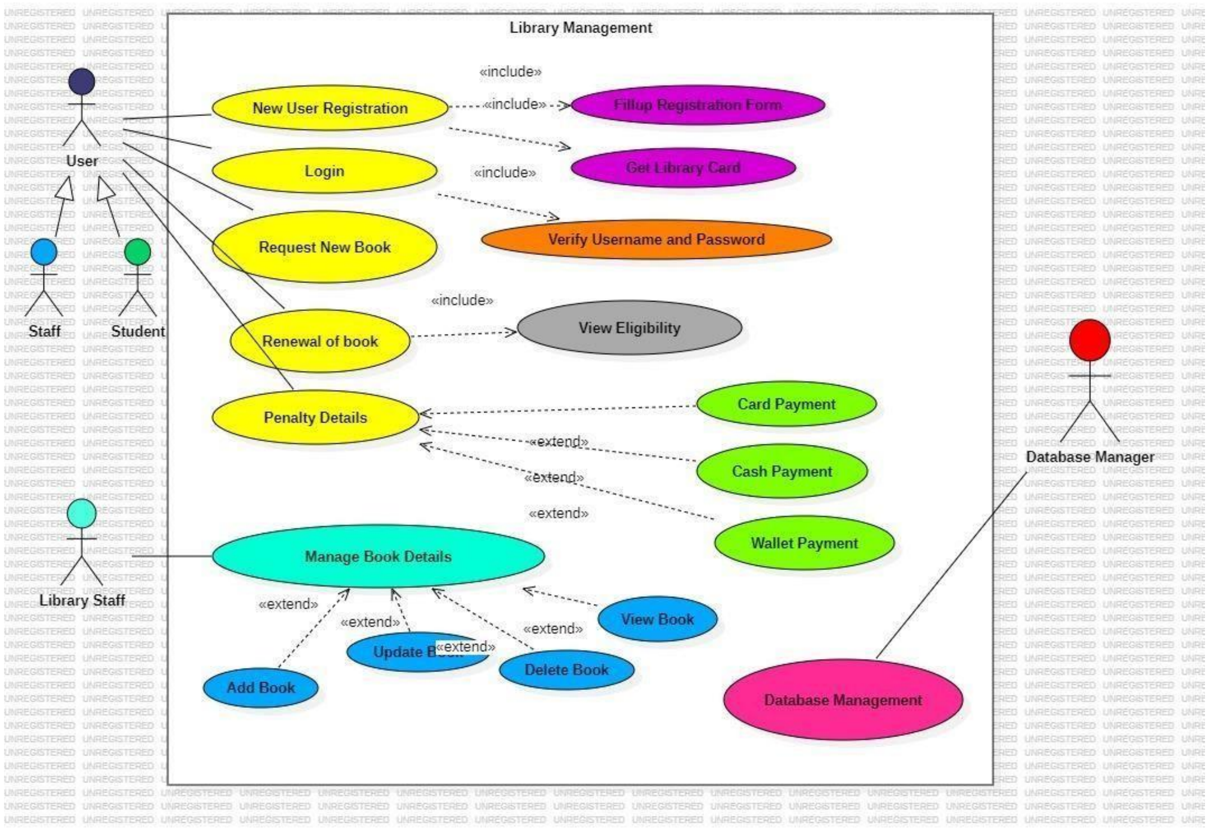
AIM:

To Demonstrate the working of the Library Management System using different UML Diagrams.

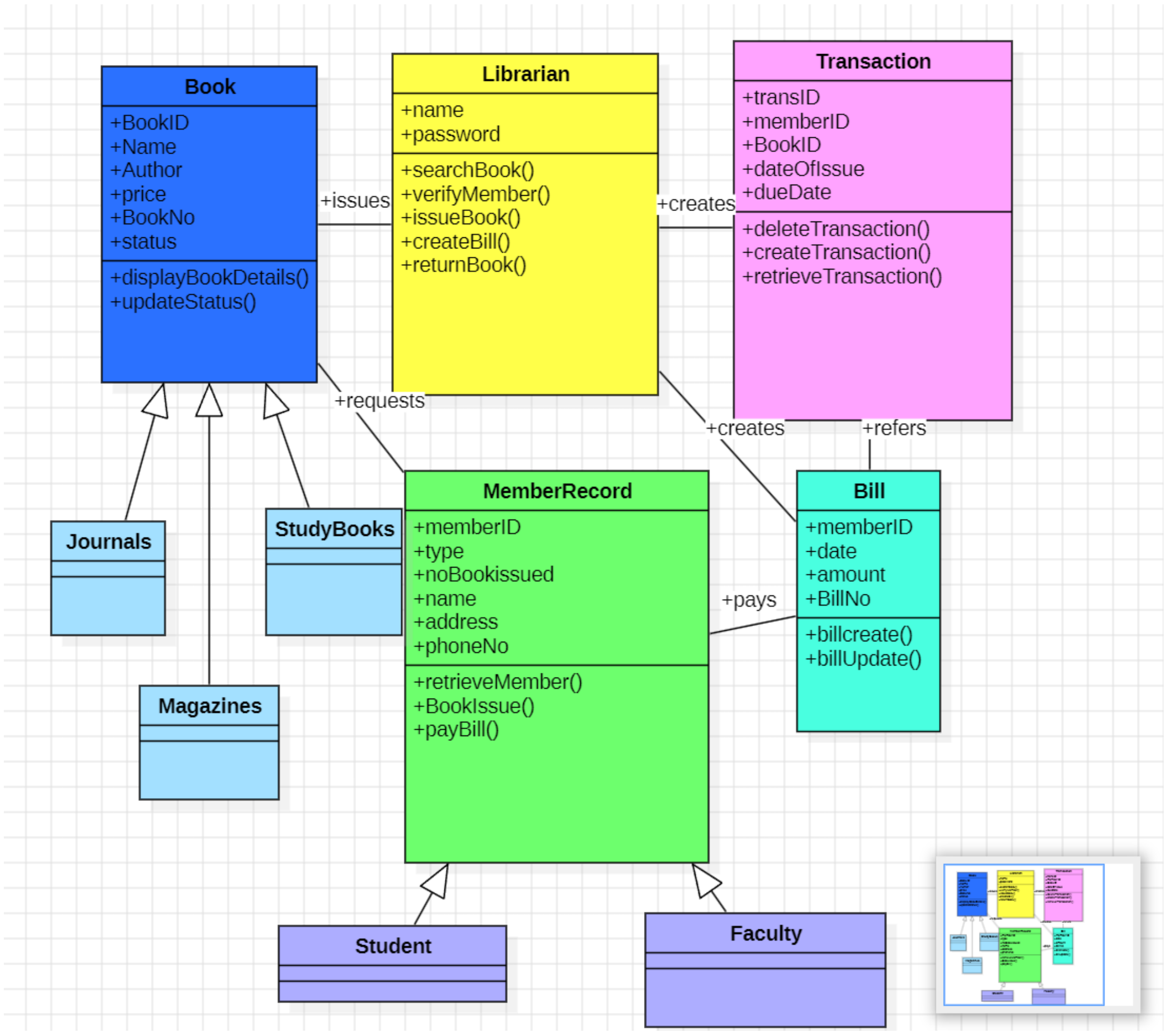
ALGORITHM:

1. Initialize System
2. User Authentication
3. Main Menu
4. Add New Book
5. Issue Book
6. Return Book
7. Search Book
8. Remove Book
9. View Borrowed Books
10. Exit System

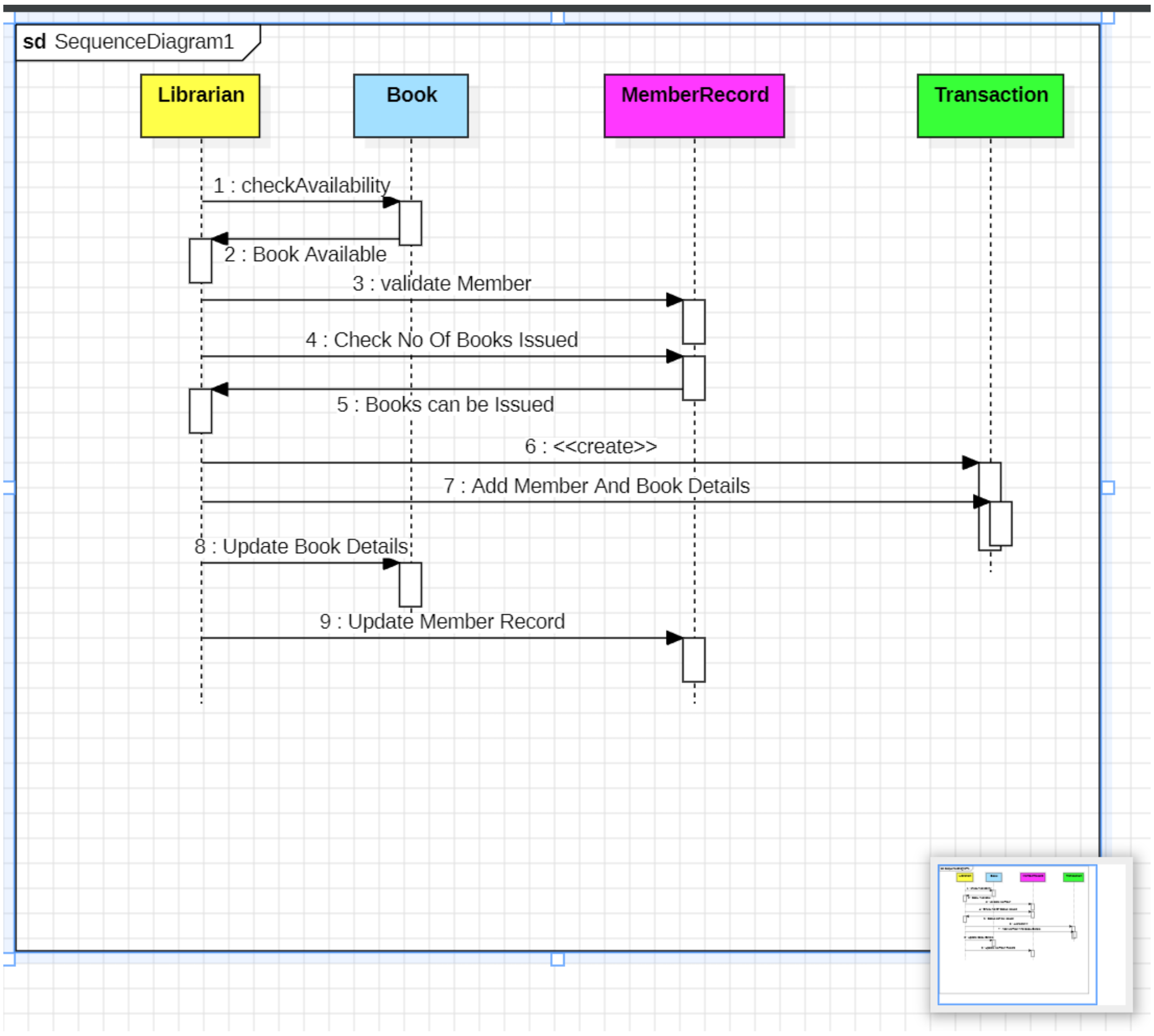
A) Use-Case Diagram



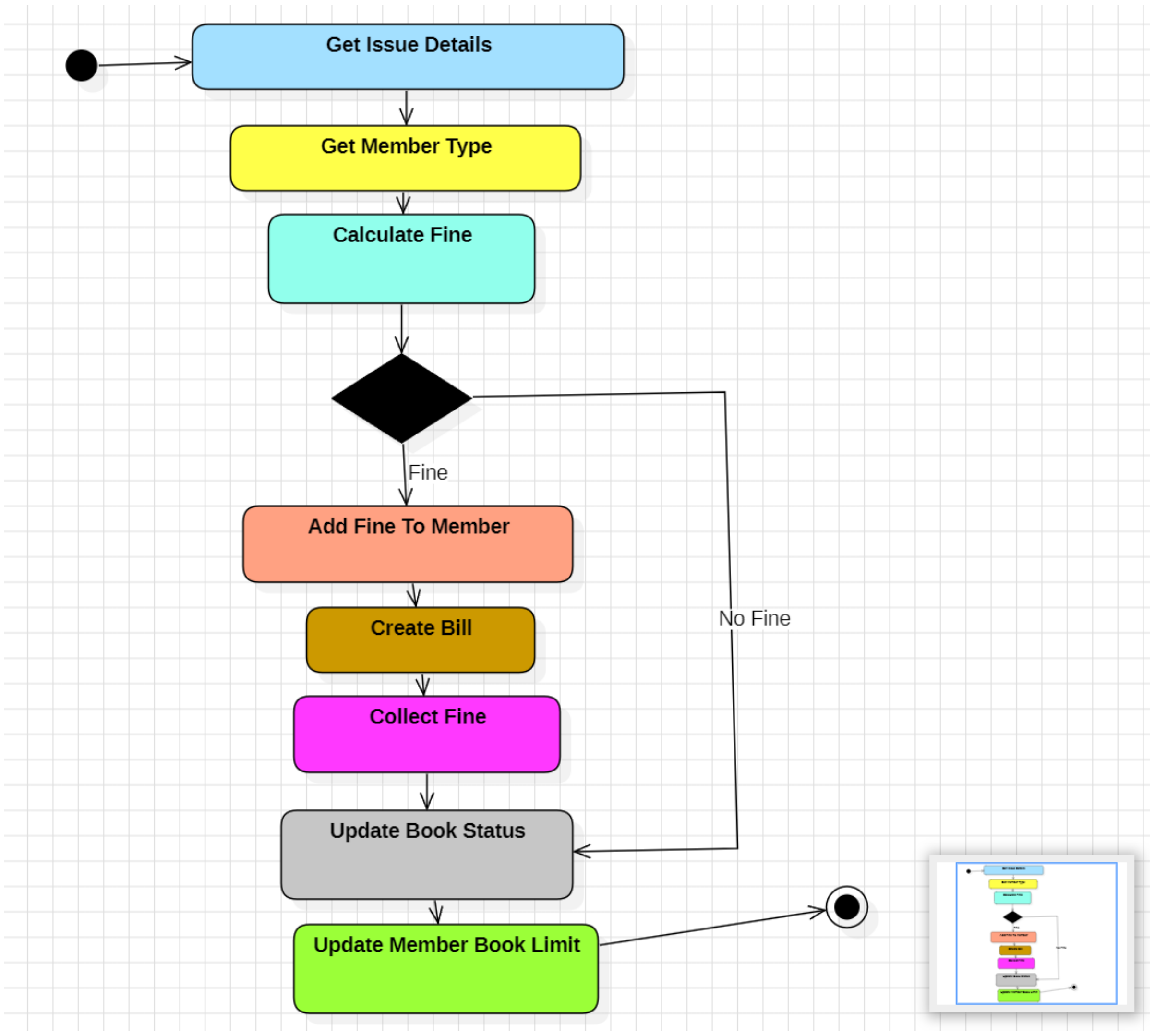
B) Class Diagram



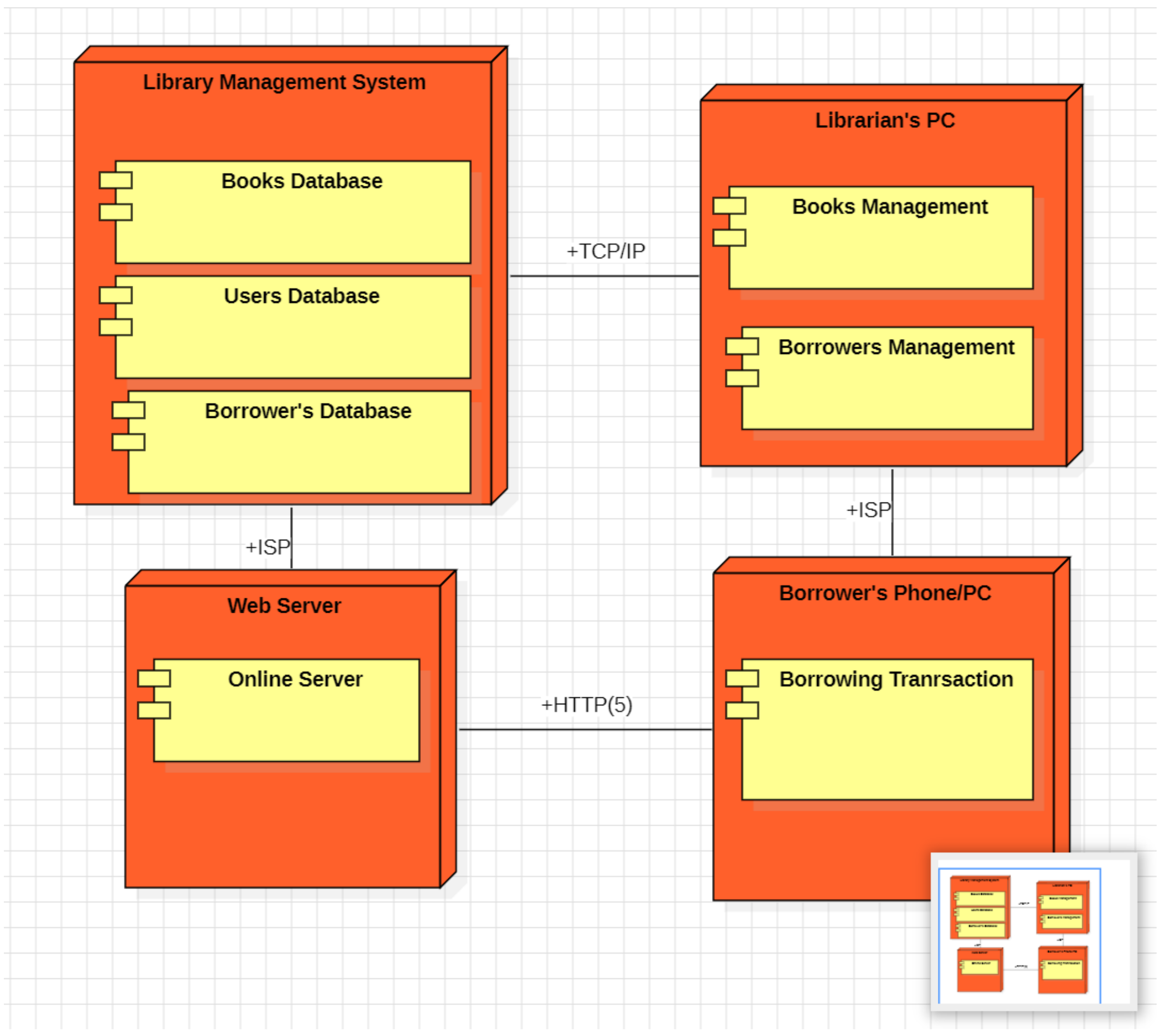
C) Sequence Diagram



D) Activity Diagram



E) DEPLOYMENT DIAGRAM:


Library Management System
Books Database
Users Database
Borrower's Database
+TCP/IP
Librarian's PC
Books Management
Borrowers Management
+ISP
+ISP
Web Server
Online Server
+HTTP(5)
Borrower's Phone/PC
Borrowing Tranrsaction

JAVA PROGRAMS

BASIC JAVA PROGRAMS (10QS)

1. **AIM:** To generate a program to print area of triangle

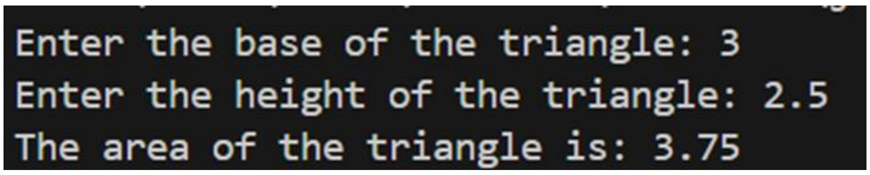
ALGORITHM:

1. Start.
2. Accept base and height values.
3. Compute the area using the formula: Area=1/2×base×height
4. Display the result.
5. End.

CODE :

```
 import java.util.Scanner;
class Triangle{

    public static void main(String[] args) { Scanner scan = new
        Scanner(System.in);
        System.out.print("Enter the base of the triangle: "); double base =
        scan.nextDouble(); System.out.print("Enter the height of the triangle: ");
        double height = scan.nextDouble();
        double area = 0.5 * base * height; System.out.println("The area of the
        triangle is: " +
 area);
        scan.close();
    }
 }
```

OUTPUT:



```
Enter the base of the triangle: 3
Enter the height of the triangle: 2.5
The area of the triangle is: 3.75
```

2.AIM: To Calculate average of 3 numbers

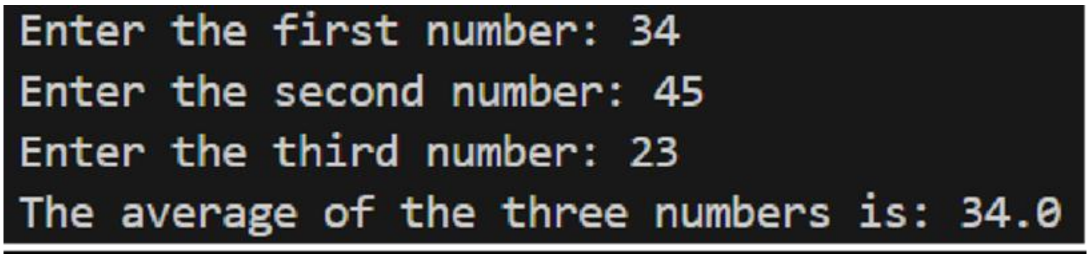
ALGORITHM:

```
1. Start.
2.  Accept three numbers from the user.
3.   Compute the average using the formula:
     average=num1+num2+num3/ 3
4.   Display the result.
5. End.
```

CODE:

```
import java.util.Scanner; class
average{
    public static void main(String[] args) { Scanner scan =
        new Scanner(System.in);
        System.out.print("Enter the first number: "); int num1 =
        scan.nextInt(); System.out.print("Enter the second number:
        "); int num2 = scan.nextInt(); System.out.print("Enter the
        third number: "); int num3 = scan.nextInt();
        double average = (num1 + num2 + num3) / 3; System.out.println("The
        average of the three numbers is: " +
average);
        scan.close();
    }
}
```

OUTPUT:



```
Enter the first number: 34
Enter the second number: 45
Enter the third number: 23
The average of the three numbers is: 34.0
```

3.

AIM: To determine a student’s grade based on marks using conditional statements.

ALGORITHM:

1. Start.
2. Accept the marks (out of 100) from the user.
3. Use if-else conditions:

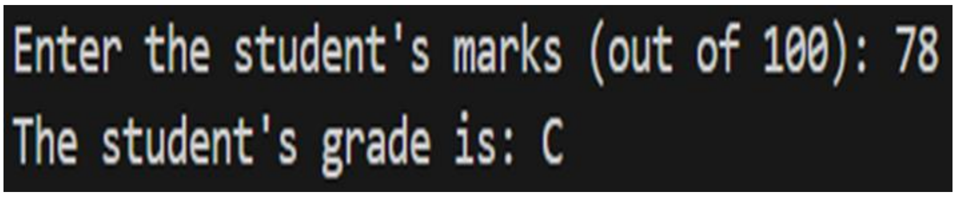
- 90–100 → Grade A
- 80–89 → Grade B
- 70–79 → Grade C
- 60–69 → Grade D
- 50–59 → Grade E
- Below 50 → Grade F

4. Display the grade.
5. End.

CODE:

```
 import java.util.Scanner; class grade{
public static void main(String[] args) { Scanner scan = new
    Scanner(System.in);
    System.out.print("Enter the student's marks (out of 100): ");
    int marks = scan.nextInt(); char grade=' ';
    if (marks >= 90 && marks <= 100) { grade = 'A';
    } else if (marks >= 80) { grade = 'B';
    } else if (marks >= 70) { grade = 'C';
    } else if (marks >= 60) { grade = 'D';
    } else if (marks >= 50) { grade = 'E';
    } else if (marks >= 0) { grade = 'F';
    }
     else {
        System.out.println("Invalid input");
        }
        System.out.println("The student's grade is: " + grade); scan.close(); }
    }
```

OUTPUT :



```
Enter the student's marks (out of 100): 78
The student's grade is: C
```

4.

AIM: To categorize a person as Child, Teenager, Adult, or Senior Citizen based on their age.

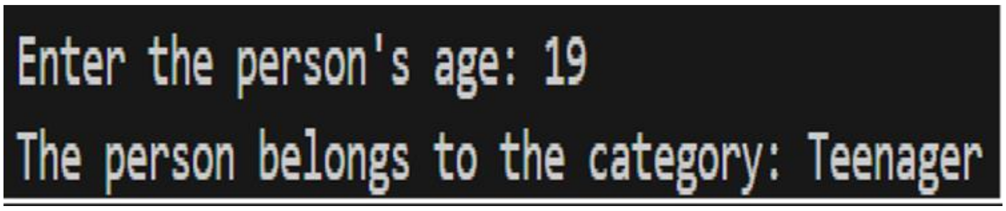
ALGORITHM :

1. Start.
2. Accept age as input.
3. Use if-else conditions to categorize:
   - 0–12 → Child
   - 13–19 → Teenager
   - 20–59 → Adult
   - 60+ → Senior Citizen
4. Display the category.
5. End.

CODE:

```
import java.util.Scanner; class category{
    public static void main(String[] args) { Scanner scan = new
        Scanner(System.in);
        System.out.print("Enter the person's age: "); int age = scan.nextInt();
        String category; if (age < 0) {
            category = "Invalid age entered.";

        } else if (age <= 12) { category = "Child";
        } else if (age <= 19) { category = "Teenager";
        } else if (age <= 59) { category = "Adult";
        } else {
            category = "Senior Citizen";
        }
        System.out.println("The person belongs to the category: " + category);
        scan.close();
    }
}
```

OUTPUT:



```
Enter the person's age: 19
The person belongs to the category: Teenager
```

5.

AIM : To generate Java program to calculate electricity bill

ALGORITHM:

1. Start.
2. Accept the number of units consumed.
3. Compute the bill using conditions:

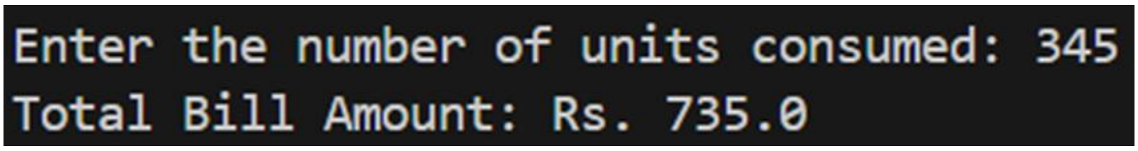
- Up to 100 units → ₹1.50 per unit.
- 101–300 units → ₹2.00 per unit (extra for units above 100).
- Above 300 units → ₹3.00 per unit (extra for units above 300).

4. Add a fixed service charge of ₹50.
5. Display the total bill amount.
6. End.

CODE:

```
import java.util.Scanner;
class bill{
    public static void main(String[] args) { Scanner scan = new
        Scanner(System.in);
        System.out.print("Enter the number of units consumed: ");
        double units = scan.nextDouble(); double billAmount;
        if (units <= 100) { billAmount = units *
            1.50;
        } else if (units <= 300) {
            billAmount = (100 * 1.50) + ((units - 100) * 2.00);
        } else {
            billAmount = (100 * 1.50) + (200 * 2.00) + ((units -
300) * 3.00);
        }
        double serviceCharge = 50.00; billAmount +=
        serviceCharge;
        System.out.println("Total Bill Amount: Rs. " + billAmount);
        scan.close();
    }
}
```

OUTPUT :



```
Enter the number of units consumed: 345
Total Bill Amount: Rs. 735.0
```

6.

AIM: To find the odd numbers in a range between a upper and a lower limit

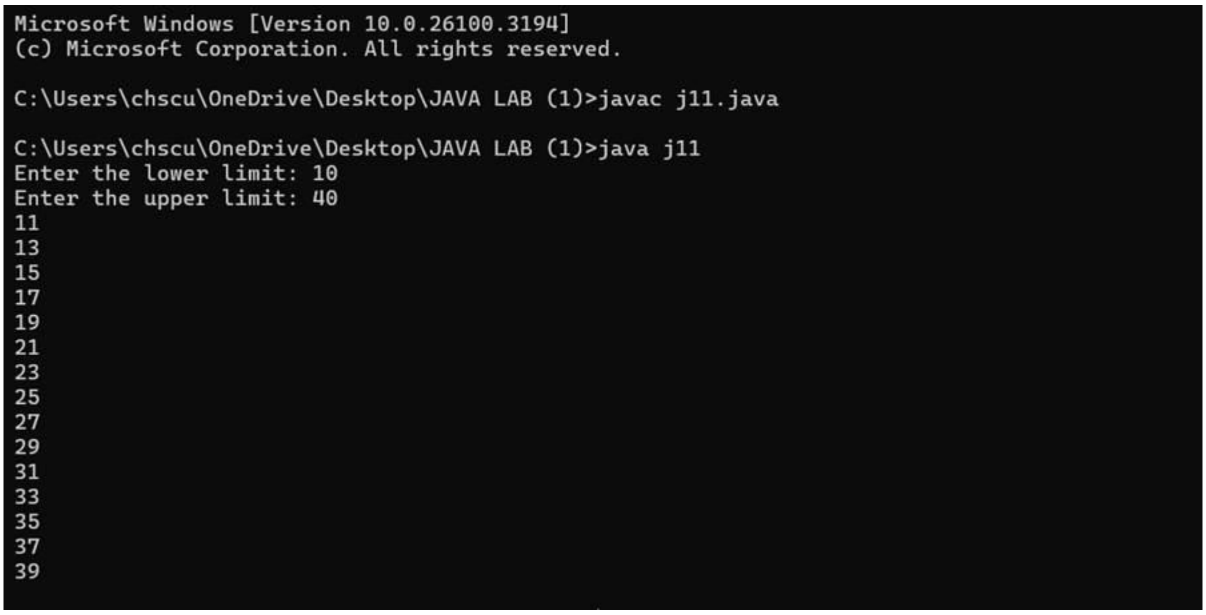
ALGORITHM:

```
Start
Initialize Scanner to take user input.
Prompt User to enter the lower limit.
Read the lower limit and store it in lower.
Prompt User to enter the upper limit.
Read the upper limit and store it in upper.
Loop from lower to upper:
If the current number is odd (number % 2 != 0), print it.
Close the Scanner to prevent resource leaks.
End
```

CODE:

```
import java.util.*;

public class OddNumbersInRange {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the lower limit: ");
        int lower = scanner.nextInt();
        System.out.print("Enter the upper limit: ");
        int upper = scanner.nextInt();

        for (int i = lower; i <= upper; i++) {
            if (i % 2 != 0) {
                System.out.println(i);
            }
        }
        scanner.close();
    }
}
```

OUTPUT:



```
Microsoft Windows [Version 10.0.26100.3194]
(c) Microsoft Corporation. All rights reserved.

C:\Users\chscu\OneDrive\Desktop\JAVA LAB (1)>javac j11.java

C:\Users\chscu\OneDrive\Desktop\JAVA LAB (1)>java j11
Enter the lower limit: 10
Enter the upper limit: 40
11
13
15
17
19
21
23
25
27
29
31
33
35
37
39
```

7.

AIM: To check wheather the number is a palindrome or not

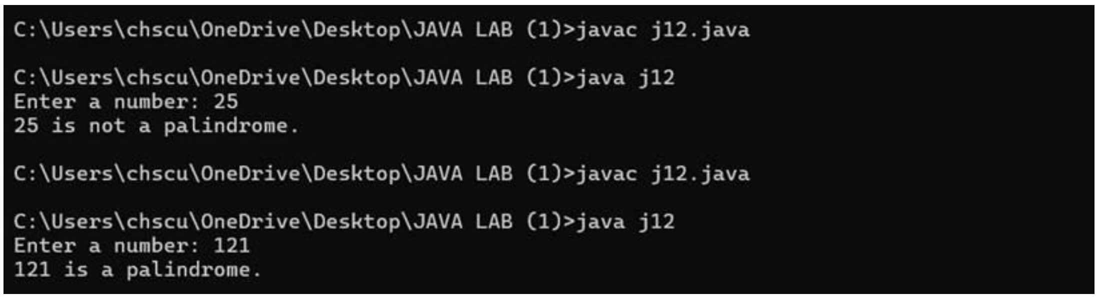
ALGORITHM:

```
1. Start
2. Initialize Scanner to take user input.
3. Prompt the user to enter a number.
4. Read the number and store it in number.
5. Store the original number in originalNumber.
6. Initialize reversedNumber to 0.
7. Reverse the number using a loop:
      o While number is not 0:
            ♣ Extract the last digit (digit = number % 10).
            ♣ Append digit to reversedNumber (reversedNumber =
              reversedNumber * 10 + digit).
            ♣ Remove the last digit (number = number / 10).
8. Compare originalNumber and reversedNumber:
      o If they are equal, print "originalNumber is a palindrome."
      o Otherwise, print "originalNumber is not a palindrome."
9. Close the Scanner.
10. End
```

CODE:

```
import java.util.*;
public class PalindromeCheck {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int number = scanner.nextInt();
        int originalNumber = number;
        int reversedNumber = 0;

        while (number != 0) {
            int digit = number % 10;
            reversedNumber = reversedNumber * 10 + digit;
            number /= 10;
        }
        if (originalNumber == reversedNumber) {
            System.out.println(originalNumber + " is a palindrome.");
        } else {
            System.out.println(originalNumber + " is not a palindrome.");
        }
        scanner.close();
    }
}
```

OUTPUT:



```
C:\Users\chscu\OneDrive\Desktop\JAVA LAB (1)>javac j12.java

C:\Users\chscu\OneDrive\Desktop\JAVA LAB (1)>java j12
Enter a number: 25
25 is not a palindrome.

C:\Users\chscu\OneDrive\Desktop\JAVA LAB (1)>javac j12.java

C:\Users\chscu\OneDrive\Desktop\JAVA LAB (1)>java j12
Enter a number: 121
121 is a palindrome.
```

8.

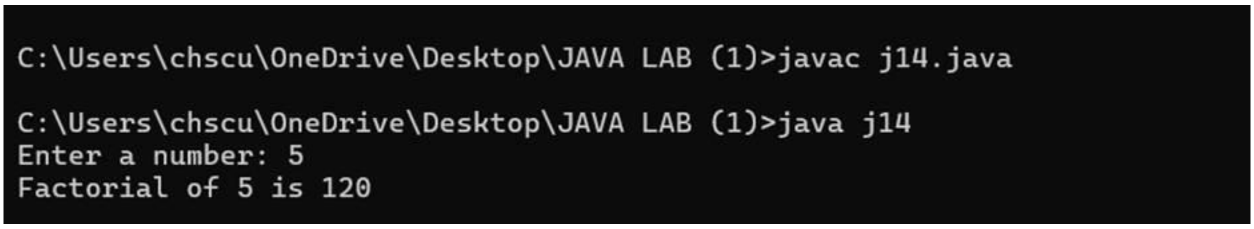
AIM: To find the factorial of a given number

ALGORITHM:

CODE:

```
import java.util.*;
public class Factorial {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int number = scanner.nextInt();
        int factorial = 0;
        for (int i = 1; i <= number; i++) {
            factorial *= i;
        }
        System.out.println("Factorial of " + number + " is " + factorial);
        scanner.close();
    }
}
```

OUTPUT:



```
C:\Users\chscu\OneDrive\Desktop\JAVA LAB (1)>javac j14.java

C:\Users\chscu\OneDrive\Desktop\JAVA LAB (1)>java j14
Enter a number: 5
Factorial of 5 is 120
```

9.AIM: to calculate the number is even or odd

ALGORITHM:

1. Start.
2. Import Scanner for user input.
3. Prompt the user to enter a number.
4. Use the modulus operator % to check if the number is divisible by 2.

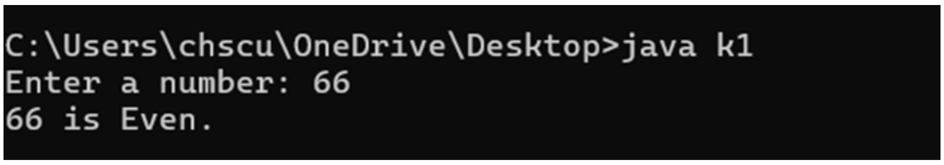
- If number % 2 == 0, print "Even".
- Otherwise, print "Odd".

5. End.

CODE:

```
import java.util.Scanner;

public class EvenOddCheck {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = sc.nextInt();
        if (num % 2 == 0)
            System.out.println(num + " is Even.");
        else
            System.out.println(num + " is Odd.");
        sc.close();
    }
}
```

OUTPUT:



```
C:\Users\chscu\OneDrive\Desktop>java k1
Enter a number: 66
66 is Even.
```

10. AIM: To print Fibonacci Series

ALGORITHM :

1. Start.
2. Import Scanner for user input.
3. Prompt the user to enter the number of terms.
4. Initialize first = 0, second = 1.
5. Use a loop to print Fibonacci numbers:
- Compute the next term by adding the last two terms.
- Print each term.

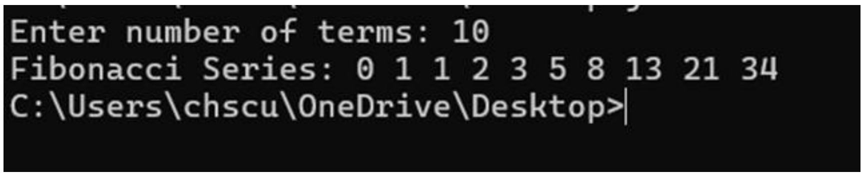
6. End.

CODE:

```
import java.util.Scanner;
public class Fibonacci {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of terms: ");
        int terms = sc.nextInt();
        int first = 0, second = 1, next;
        System.out.print("Fibonacci Series: " + first + " " + second);
        for (int i = 2; i < terms; i++) {
            next = first + second;
            System.out.print(" " + next);
            first = second;
            second = next;
        }
        sc.close();
    }
```

}

OUTPUT:



```
Enter number of terms: 10
Fibonacci Series: 0 1 1 2 3 5 8 13 21 34
C:\Users\chscu\OneDrive\Desktop>
```

POLYMORPHISM
OVERLOADING AND OVERRIDING (8Qs)

11.STUDENT MANAGEMENT SYSTEM

AIM:

- Use inheritance for different types of students (e.g., Undergraduate, Graduate).
- Implement polymorphism for displaying student details.
- Use loops for managing student records.

ALGORITHM:

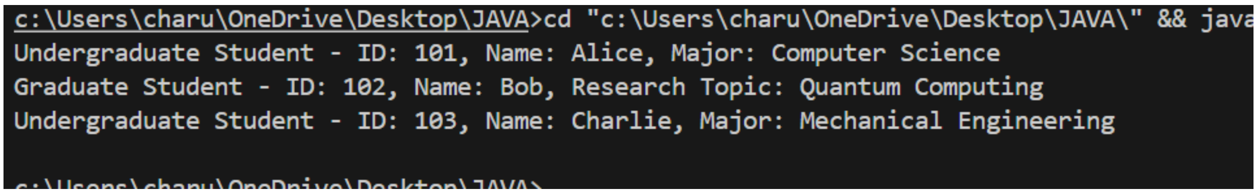
1. Define base class `Student`.
2. Create derived classes `Undergraduate` and `Graduate`.
3. Override `displayDetails()` method in subclasses.
4. Initialize an array of students.
5. Loop through the array and display details.

CODING:

```
class Student {
   String name;
   int id;

   Student(String name, int id) {
      this.name = name;
      this.id = id;
   }

   void displayDetails() {
      System.out.println("Student ID: " + id + ", Name: " + name);
   }
```

```
}


// Derived class for Undergraduate students
class Undergraduate extends Student {
    String major;


    Undergraduate(String name, int id, String major) {
        super(name, id);
        this.major = major;
    }


    @Override
    void displayDetails() {
        System.out.println("Undergraduate Student - ID: " + id + ", Name: "
+ name + ", Major: " + major);
    }
}
 class Graduate extends Student {
    String researchTopic;


    Graduate(String name, int id, String researchTopic) {
        super(name, id);
        this.researchTopic = researchTopic;
    }


    @Override
    void displayDetails() {
        System.out.println("Graduate Student - ID: " + id + ", Name: " +
name + ", Research Topic: " + researchTopic);
    }
}

public class StudentManagement {
    public static void main(String[] args) {
        Student students[] = new Student[3];

        students[0] = new Undergraduate("Alice", 101, "Computer
Science");
        students[1] = new Graduate("Bob", 102, "Quantum Computing");
        students[2] = new Undergraduate("Charlie", 103, "Mechanical
Engineering");
        for (Student s : students) {
            s.displayDetails();
        }
    }
}
```

OUTPUT:



```
c:\Users\charu\OneDrive\Desktop\JAVA>cd "c:\Users\charu\OneDrive\Desktop\JAVA\" && java
Undergraduate Student - ID: 101, Name: Alice, Major: Computer Science
Graduate Student - ID: 102, Name: Bob, Research Topic: Quantum Computing
Undergraduate Student - ID: 103, Name: Charlie, Major: Mechanical Engineering

c:\Users\charu\OneDrive\Desktop\JAVA>
```

12.LIBRARY MANAGEMENT SYSTEM

AIM:

- Base class: `Book`, Derived classes: `Ebook`, `PrintedBook`.
- Use polymorphism for book details display.
- Loop through books to manage borrowing and returning.

ALGORITHM:

1. Define base class `Book`.
2. Create derived classes `Ebook` and `PrintedBook`.
3. Override `displayInfo()` method in subclasses.
4. Initialize an array of books.
5. Loop through the array and display book details.

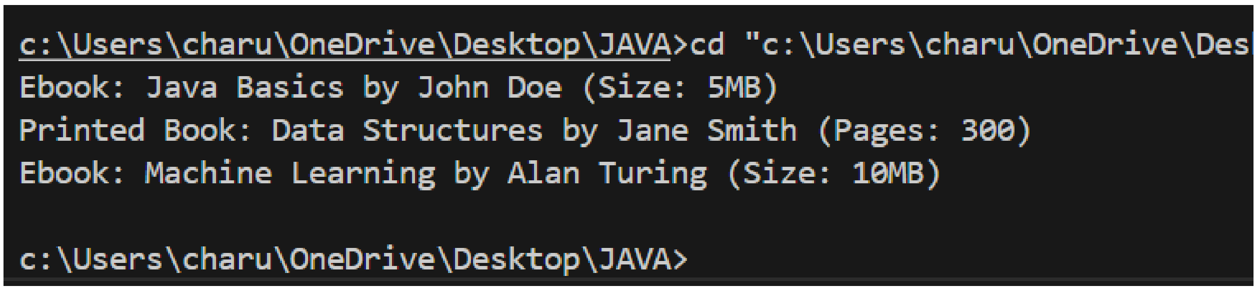
CODING:

```
class Ebook extends Book {
    int fileSize;

    Ebook(String title, String author, int fileSize) {
        super(title, author);
        this.fileSize = fileSize;
    }

    @Override
    void displayInfo() {
        System.out.println("Ebook: " + title + " by " + author + " (Size: " +
fileSize + "MB)");
    }
```

```
}
class PrintedBook extends Book {
    int pages;

    PrintedBook(String title, String author, int pages) {
        super(title, author);
        this.pages = pages;
    }

    @Override
    void displayInfo() {
        System.out.println("Printed Book: " + title + " by " + author + "
(Pages: " + pages + ")");
    }
}
public class LibraryManagement {
    public static void main(String[] args) {
        // Creating an array of Book objects (looping used)
        Book books[] = new Book[3];

        books[0] = new Ebook("Java Basics", "John Doe", 5);
        books[1] = new PrintedBook("Data Structures", "Jane Smith", 300);
        books[2] = new Ebook("Machine Learning", "Alan Turing", 10);
        for (Book b : books) {
            b.displayInfo();
        }
    }
```

}

OUTPUT:



```
c:\Users\charu\OneDrive\Desktop\JAVA>cd "c:\Users\charu\OneDrive\Des
Ebook: Java Basics by John Doe (Size: 5MB)
Printed Book: Data Structures by Jane Smith (Pages: 300)
Ebook: Machine Learning by Alan Turing (Size: 10MB)

c:\Users\charu\OneDrive\Desktop\JAVA>
```

13.BANKING SYSTEM

AIM:

- Base class: `Account`, Derived classes: `SavingsAccount`, `CurrentAccount`.
- Override methods for withdrawal and deposit behavior.
- Use loops to handle multiple transactions.

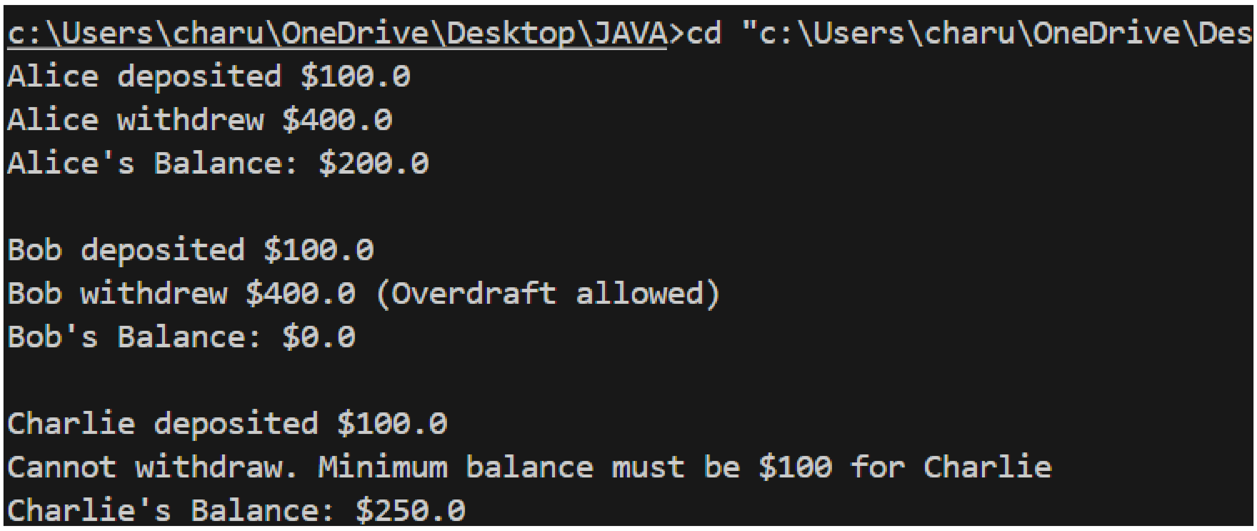
ALGORITHM:

1. Define base class `Account`.
2. Create derived classes `SavingsAccount` and `CheckingAccount`.
3. Override `withdraw()` method in subclasses.
4. Initialize an array of accounts.
5. Loop through accounts and perform transactions.

CODING:

```
class Account {
    String accountHolder;
    double balance;

    Account(String accountHolder, double balance) {
        this.accountHolder = accountHolder;
        this.balance = balance;
    }
    void deposit(double amount) {
        balance += amount;
        System.out.println(accountHolder + " deposited $" + amount);
    }

    void withdraw(double amount) { // Polymorphic method
        if (balance >= amount) {
            balance -= amount;
            System.out.println(accountHolder + " withdrew $" + amount);
        } else {
            System.out.println("Insufficient funds for " + accountHolder);
        }
    }

    void displayBalance() {
        System.out.println(accountHolder + "'s Balance: $" + balance);
    }
}
class SavingsAccount extends Account {
    double interestRate;

    SavingsAccount(String accountHolder, double balance, double
interestRate) {
        super(accountHolder, balance);
        this.interestRate = interestRate;
    }
    @Override
    void withdraw(double amount) {
        if (balance - amount >= 100) { // Maintain minimum balance of $100
            super.withdraw(amount);
        } else {
            System.out.println("Cannot withdraw. Minimum balance must be
$100 for " + accountHolder);
        }
    }
}
class CheckingAccount extends Account {
    double overdraftLimit;

    CheckingAccount(String accountHolder, double balance, double
overdraftLimit) {
        super(accountHolder, balance);
        this.overdraftLimit = overdraftLimit;
    }

    @Override
    void withdraw(double amount) {
        if (balance + overdraftLimit >= amount) {
            balance -= amount;
            System.out.println(accountHolder + " withdrew $" + amount + "
(Overdraft allowed)");
        } else {
            System.out.println("Overdraft limit exceeded for " +
accountHolder);
        }
    }
}
 public class BankingSystem {
    public static void main(String[] args) {
        // Creating an array of Account objects (looping used)
        Account accounts[] = new Account[3];

        accounts[0] = new SavingsAccount("Alice", 500, 2.5);
        accounts[1] = new CheckingAccount("Bob", 300, 200);
        accounts[2] = new SavingsAccount("Charlie", 150, 3.0);
                for (Account acc : accounts) {
            acc.deposit(100);
            acc.withdraw(400);
            acc.displayBalance();
            System.out.println();
        }
    }
}
```

OUTPUT:



```
c:\Users\charu\OneDrive\Desktop\JAVA>cd "c:\Users\charu\OneDrive\Des
Alice deposited $100.0
Alice withdrew $400.0
Alice's Balance: $200.0

Bob deposited $100.0
Bob withdrew $400.0 (Overdraft allowed)
Bob's Balance: $0.0

Charlie deposited $100.0
Cannot withdraw. Minimum balance must be $100 for Charlie
Charlie's Balance: $250.0
```

14.EMPLOYEE PAYROLL SYSTEM

AIM:

- Parent class: `Employee`, Child classes: `FullTimeEmployee`, `PartTimeEmployee`.
- Polymorphic method to calculate salary based on type.
- Loop through employees to process salaries.

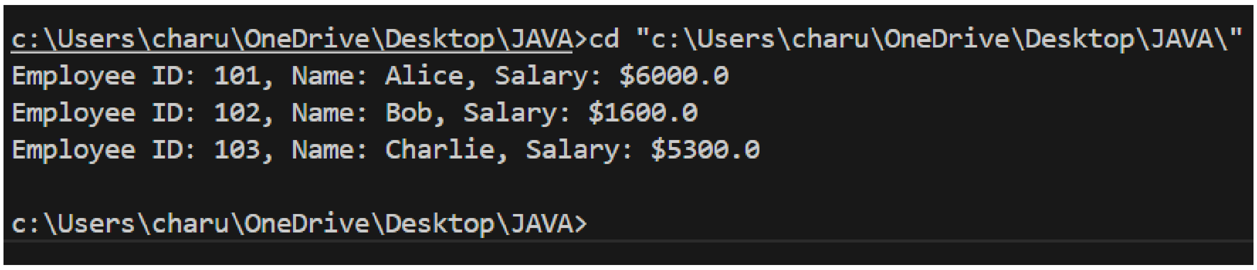
ALGORITHM:

1. Define base class `Employee`.
2. Create derived classes `FullTimeEmployee` and `PartTimeEmployee`.
3. Override `calculateSalary()` method in subclasses.
4. Initialize an array of employees.
5. Loop through employees and display salaries.

CODING:

```
class Employee {
    String name;
    int id;
    double baseSalary;

    Employee(String name, int id, double baseSalary) {
        this.name = name;
        this.id = id;
        this.baseSalary = baseSalary;
    }

    double calculateSalary() { // Polymorphic method
        return baseSalary;
    }

    void displaySalary() {
        System.out.println("Employee ID: " + id + ", Name: " + name + ",
Salary: $" + calculateSalary());
    }
}
class FullTimeEmployee extends Employee {
    double bonus;

    FullTimeEmployee(String name, int id, double baseSalary, double
bonus) {
        super(name, id, baseSalary);
        this.bonus = bonus;
    }
    @Override
    double calculateSalary() {
        return baseSalary + bonus;
    }
}

class PartTimeEmployee extends Employee {
    int hoursWorked;
    double hourlyRate;

    PartTimeEmployee(String name, int id, double hourlyRate, int
hoursWorked) {
        super(name, id, 0);
        this.hourlyRate = hourlyRate;
        this.hoursWorked = hoursWorked;
    }

    @Override
    double calculateSalary() {
        return hourlyRate * hoursWorked;
    }
}
public class EmployeePayroll {
    public static void main(String[] args) {
        // Creating an array of Employee objects (looping used)
        Employee employees[] = new Employee[3];
        employees[0] = new FullTimeEmployee("Alice", 101, 5000, 1000);
        employees[1] = new PartTimeEmployee("Bob", 102, 20, 80);
        employees[2] = new FullTimeEmployee("Charlie", 103, 4500, 800);

        // Loop through and display employee salaries
        for (Employee emp : employees) {
            emp.displaySalary();
        }
    }
}
```

OUTPUT:



```
c:\Users\charu\OneDrive\Desktop\JAVA>cd "c:\Users\charu\OneDrive\Desktop\JAVA\"
Employee ID: 101, Name: Alice, Salary: $6000.0
Employee ID: 102, Name: Bob, Salary: $1600.0
Employee ID: 103, Name: Charlie, Salary: $5300.0

c:\Users\charu\OneDrive\Desktop\JAVA>
```

15.VEHICLE RENTAL SYSTEM

AIM:

- Base class: `Vehicle`, Derived classes: `Car`, `Bike`, `Truck`.
- Polymorphism for rental price calculation.
- Use loops for booking and returning vehicles.

ALGORITHM:

1. Define base class `Vehicle`.
2. Create derived classes `Car`, `Bike`, and `Truck`.
3. Override `calculateRental()` method in subclasses.
4. Initialize an array of vehicles.

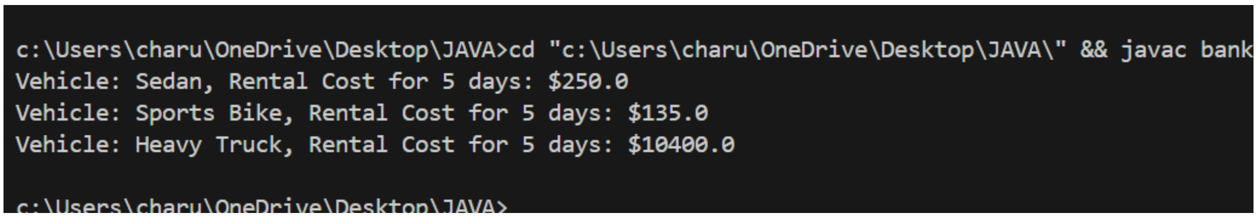
5. Loop through vehicles and display rental details.

CODING:

```
Vehicle(String model, double rentalPrice) { this.model =
        model;
        this.rentalPrice = rentalPrice;
}

double calculateRental(int days) { // Polymorphic method return rentalPrice
        * days;
}

void displayDetails(int days) { System.out.println("Vehicle: " + model
        + ", Rental
Cost for " + days + " days: $" + calculateRental(days));
}


}

@Override
double calculateRental(int days) {
        return super.calculateRental(days) * 0.9; // 10% discount for
bikes
}


} Truck(String model, double rentalPrice, double weightLimit) {
        super(model, rentalPrice); this.weightLimit =
        weightLimit;
}

@Override
double calculateRental(int days) {
        return super.calculateRental(days) + (weightLimit * 2
* days);
```

```
}


}

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

 objects (looping used) Vehicle vehicles[] = new Vehicle[3];

      vehicles[0] = new Car("Sedan", 50); vehicles[1] = new
        Bike("Sports Bike", 30);
        vehicles[2] = new Truck("Heavy Truck", 80, 1000); int

        rentalDays = 5;

        for (Vehicle v : vehicles) { v.displayDetails(rentalDays);
        }
}


}
```

OUTPUT:



```
c:\Users\charu\OneDrive\Desktop\JAVA>cd "c:\Users\charu\OneDrive\Desktop\JAVA\" && javac bank
Vehicle: Sedan, Rental Cost for 5 days: $250.0
Vehicle: Sports Bike, Rental Cost for 5 days: $135.0
Vehicle: Heavy Truck, Rental Cost for 5 days: $10400.0

c:\Users\charu\OneDrive\Desktop\JAVA>
```

16.HOSPITAL MANAGEMENT SYSTEM:

AIM:

- Parent class: `Person`, Derived classes: `Doctor`, `Patient`, `Staff`.
- Override methods for role-based actions.
- Loop to manage patient check-in and check-out.

ALGORITHM:

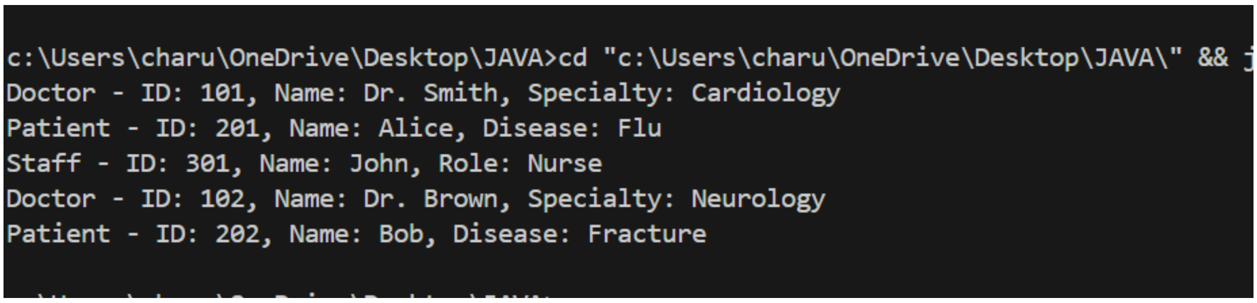
1. Define base class `Person`.
2. Create derived classes `Doctor`, `Patient`, and `Staff`.
3. Override `displayDetails()` method in subclasses.
4. Initialize an array of hospital personnel.
5. Loop through the array and display details.

CODING:

```
class Person {
    String name;
    int id;

    public Person(String name, int id) {
        this.name = name;
        this.id = id;
    }

    public void displayDetails() {
        System.out.println("ID: " + id + ", Name: " + name);
    }
}
 class Doctor extends Person {
    String specialty;

    public Doctor(String name, int id, String specialty) {
        super(name, id);

        this.specialty = specialty;

    }



    @Override

    public void displayDetails() {

        System.out.println("Doctor - ID: " + id + ", Name: " + name + ",
Specialty: " + specialty);

    }

}

class Patient extends Person {

    String disease;



    public Patient(String name, int id, String disease) {

        super(name, id);

        this.disease = disease;

    }



    @Override

    public void displayDetails() {

        System.out.println("Patient - ID: " + id + ", Name: " + name + ",
Disease: " + disease);

    }
```

```
}
class Staff extends Person {
    String role;

    public Staff(String name, int id, String role) {
        super(name, id);
        this.role = role;
    }

    @Override
    public void displayDetails() {
        System.out.println("Staff - ID: " + id + ", Name: " + name + ", Role: "
+ role);
    }
}

public class HospitalManagement {
    public static void main(String[] args) {
        Person[] people = {
            new Doctor("Dr. Smith", 101, "Cardiology"),
            new Patient("Alice", 201, "Flu"),
            new Staff("John", 301, "Nurse"),
            new Doctor("Dr. Brown", 102, "Neurology"),
            new Patient("Bob", 202, "Fracture")
        };

        for (Person p : people) {
            p.displayDetails();
        }
    }
}
```

OUTPUT:



```
c:\Users\charu\OneDrive\Desktop\JAVA>cd "c:\Users\charu\OneDrive\Desktop\JAVA\" && j
Doctor - ID: 101, Name: Dr. Smith, Specialty: Cardiology
Patient - ID: 201, Name: Alice, Disease: Flu
Staff - ID: 301, Name: John, Role: Nurse
Doctor - ID: 102, Name: Dr. Brown, Specialty: Neurology
Patient - ID: 202, Name: Bob, Disease: Fracture
```

17.SHOPPING CART SYSTEM:

AIM:

- Base class: `Product`, Derived classes: `Electronics`, `Clothing`.
- Polymorphism for discount calculation.
- Loop through cart items to display the bill.

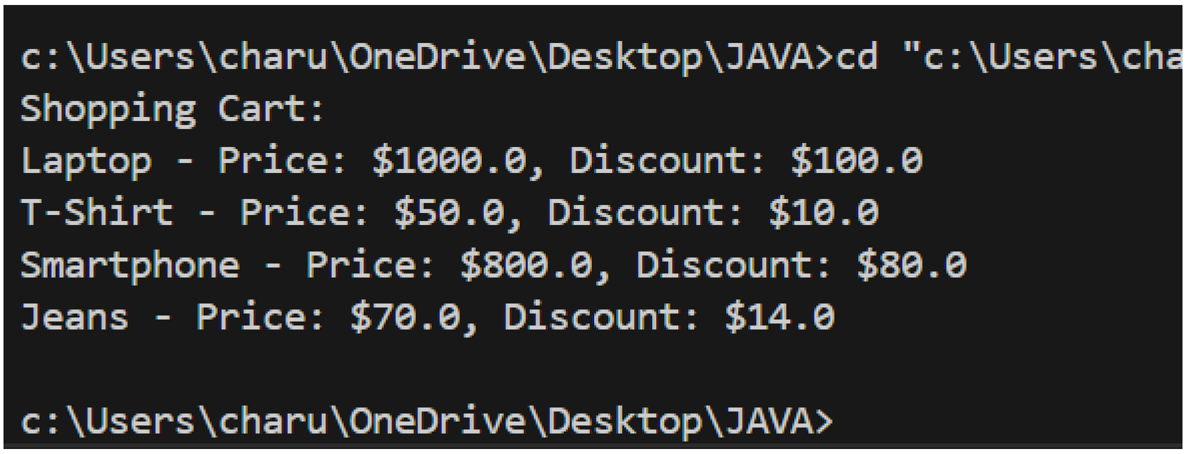
ALGORITHM:

1. Define base class `Product`.
2. Create derived classes `Electronics` and `Clothing`.
3. Override `calculateDiscount()` method in subclasses.
4. Initialize an array of cart items.
5. Loop through the cart and display product details.

CODING:

```
class Product {
    String name;
    double price;

    public Product(String name, double price) {
        this.name = name;
        this.price = price;
    }
     public double calculateDiscount() {
        return 0;
    }

    public void displayProduct() {
        System.out.println(name + " - Price: $" + price + ", Discount: $" +
calculateDiscount());
    }
}
class Electronics extends Product {
    public Electronics(String name, double price) {
        super(name, price);
    }
    @Override
    public double calculateDiscount() {
        return price * 0.10; // 10% discount
    }
}
 class Clothing extends Product {
    public Clothing(String name, double price) {
        super(name, price);
    }

    @Override
    public double calculateDiscount() {
        return price * 0.20; // 20% discount
    }
}
public class ShoppingCart {
    public static void main(String[] args) {
        Product[] cart = {
            new Electronics("Laptop", 1000),
            new Clothing("T-Shirt", 50),
            new Electronics("Smartphone", 800),
            new Clothing("Jeans", 70)
        };

        System.out.println("Shopping Cart:");

        for (Product p : cart) {

            p.displayProduct();

        }

    }

}
```

OUTPUT:



```
c:\Users\charu\OneDrive\Desktop\JAVA>cd "c:\Users\cha
Shopping Cart:
Laptop - Price: $1000.0, Discount: $100.0
T-Shirt - Price: $50.0, Discount: $10.0
Smartphone - Price: $800.0, Discount: $80.0
Jeans - Price: $70.0, Discount: $14.0

c:\Users\charu\OneDrive\Desktop\JAVA>
```

18.ONLINE EXAMINATION

SYSTEM AIM:

- Base class: `Question`, Derived classes: `MCQ`, `Descriptive Question`.
- Override method for evaluating answers.
- Loop to present questions and take user input.

ALGORITHM:

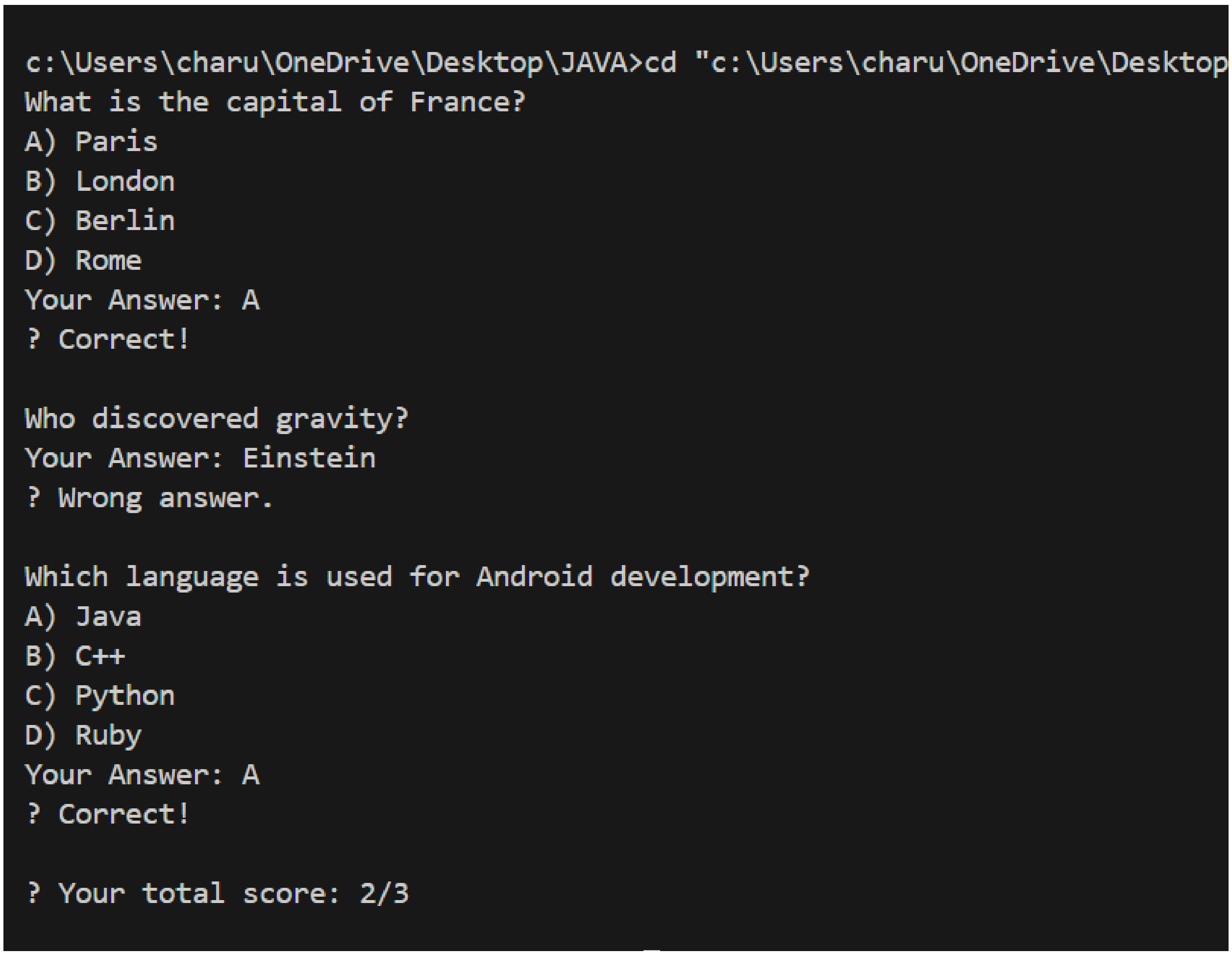
1. Define base class `Question`.
2. Create derived classes `MCQ` and `DescriptiveQuestion`.
3. Override `askQuestion()` and `checkAnswer()` methods.
4. Initialize an array of questions.

5. Loop through questions, get user input, and evaluate answers.

CODING:

```
import java.util.Scanner;
 class Question {
    String questionText;

    public Question(String questionText) {
        this.questionText = questionText;
    }
    public void askQuestion() {
        System.out.println(questionText);
    }

    public boolean checkAnswer(String answer) {
        return false;
    }
}

class MCQ extends Question {
    String[] options;
    String correctAnswer;
    public MCQ(String questionText, String[] options, String
correctAnswer) {

        super(questionText);

        this.options = options;

        this.correctAnswer = correctAnswer;

    }


    @Override

    public void askQuestion() {

        System.out.println(questionText);

        for (String option : options) {

            System.out.println(option);

        }

    }


    @Override

    public boolean checkAnswer(String answer) {

        return answer.equalsIgnoreCase(correctAnswer);

    }

}

class DescriptiveQuestion extends Question {

    String correctAnswer
    public DescriptiveQuestion(String questionText, String correctAnswer)
{
        super(questionText);
        this.correctAnswer = correctAnswer;
    }

    @Override
    public boolean checkAnswer(String answer) {
        return answer.equalsIgnoreCase(correctAnswer);
    }
}
public class OnlineExam {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
         Question[] questions = {
            new MCQ("What is the capital of France?", new String[]{"A)
Paris", "B) London", "C) Berlin", "D) Rome"}, "A"),
            new DescriptiveQuestion("Who discovered gravity?", "Newton"),
            new MCQ("Which language is used for Android development?",
new String[]{"A) Java", "B) C++", "C) Python", "D) Ruby"}, "A")
        };

        int score = 0;
         for (Question q : questions) {
            q.askQuestion();

            System.out.print("Your Answer: ");

            String userAnswer = scanner.nextLine();

            if (q.checkAnswer(userAnswer)) {

                System.out.println("Correct!\n");

                score++;

            } else {

                System.out.println("Wrong answer.\n");

            }

        }

         System.out.println("Your total score: " + score + "/" +
questions.length);

        scanner.close();

    }

}
```

OUTPUT:



```
c:\Users\charu\OneDrive\Desktop\JAVA>cd "c:\Users\charu\OneDrive\Desktop
What is the capital of France?
A) Paris
B) London
C) Berlin
D) Rome
Your Answer: A
? Correct!

Who discovered gravity?
Your Answer: Einstein
? Wrong answer.

Which language is used for Android development?
A) Java
B) C++
C) Python
D) Ruby
Your Answer: A
? Correct!

? Your total score: 2/3
```

INHERITANCE
SINGLE, HIERHARCHIAL, MULTI-LEVEL, HYBRID (8Qs)

19.ZOO MANAGEMENT

SYSTEM AIM:

- Base class: `Animal`, Derived classes: `Mammal`, `Bird`, `Reptile`.
- Use polymorphism for different sounds and behaviors.
- Loop to display animal details.

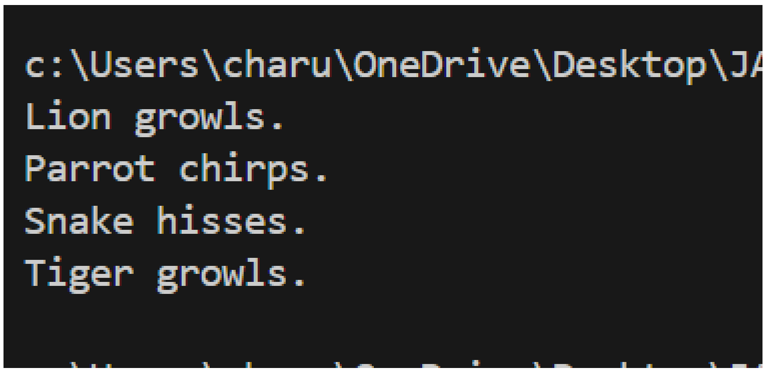
ALGORITHM:

1. Define base class `Animal`.
2. Create derived classes `Mammal`, `Bird`, and `Reptile`.
3. Override `makeSound()` method in subclasses.
4. Initialize an array of animals.
5. Loop through the array and display animal sounds.

CODING:

```
class Animal
    String name;

    public Animal(String name) {
        this.name = name;
    }

    public void makeSound() {
        System.out.println(name + " makes a sound.");
    }
}
class Mammal extends Animal {
    public Mammal(String name) {
        super(name);
    }

    @Override
    public void makeSound() {
        System.out.println(name + " growls.");
    }
}
class Bird extends Animal {
    public Bird(String name) {
        super(name);
    }

    @Override
    public void makeSound() {
        System.out.println(name + " chirps.");
    }
}
class Reptile extends Animal {
    public Reptile(String name) {
        super(name);
    }

    @Override
    public void makeSound() {
        System.out.println(name + " hisses.");
    }
}
public class ZooManagement {
    public static void main(String[] args) {
        Animal[] zoo = {
            new Mammal("Lion"),
            new Bird("Parrot"),
            new Reptile("Snake"),
            new Mammal("Tiger")
        };
        for (Animal a : zoo) {
            a.makeSound();
        }
    }
}
```

OUTPUT:



```
c:\Users\charu\OneDrive\Desktop\JA
Lion growls.
Parrot chirps.
Snake hisses.
Tiger growls.
```

20.GAME LEADERBOARD SYSTEM

AIM:

- Base class: `Player`, Derived classes: `CasualPlayer`, `ProPlayer`.
- Override method for score calculation.
- Loop to rank players based on scores.

ALGORITHM:

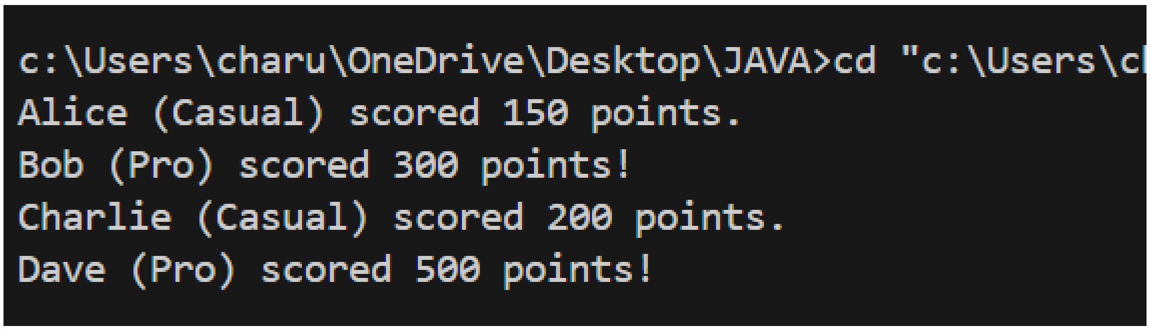
1. Define base class `Player`.
2. Create derived classes `CasualPlayer` and `ProPlayer`.
3. Override `displayScore()` method in subclasses.
4. Initialize an array of players.
5. Loop through players and display scores.

CODING:

```
class Player {
    String
    name; int
    score;

    public Player(String name, int score) {
        this.name = name;
        this.score = score;
    }

    public void displayScore() {
        System.out.println(name + " scored " + score + " points.");
    }
}
class CasualPlayer extends Player {
    public CasualPlayer(String name, int score) {
        super(name, score);
    }

    @Override
    public void displayScore() {
        System.out.println(name + " (Casual) scored " + score + " points.");
    }
}
```

```
class ProPlayer extends Player {
    public ProPlayer(String name, int score) {
        super(name, score);
    }


    @Override
    public void displayScore(){
        System.out.println(name + " (Pro) scored " + score + " points!");
    }
}
public class GameLeaderboard {
    public static void main(String[] args) {
        Player[] leaderboard = {
            new CasualPlayer("Alice", 150),
            new ProPlayer("Bob", 300),
            new CasualPlayer("Charlie", 200),
            new ProPlayer("Dave", 500)
        };
         for (Player p : leaderboard) {
            p.displayScore();

         }

         }

         }
```

OUTPUT:



```
c:\Users\charu\OneDrive\Desktop\JAVA>cd "c:\Users\c
Alice (Casual) scored 150 points.
Bob (Pro) scored 300 points!
Charlie (Casual) scored 200 points.
Dave (Pro) scored 500 points!
```

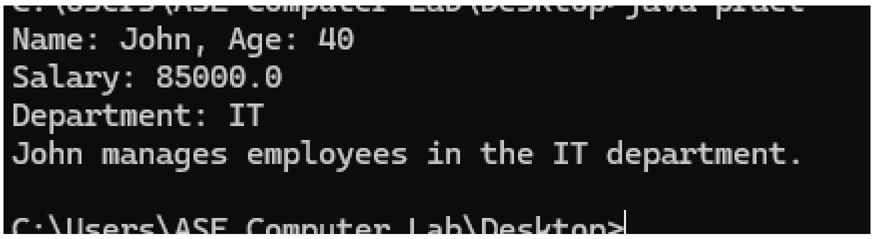
21.SALARY DISTRIBUTION

CODING:

```
class Person {
    String name;
    int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public void display() {
        System.out.println("Name: " + name + ", Age: " + age);
    }
}
```

```
class Employee extends Person {
    double salary;

    public Employee(String name, int age, double salary) {
        super(name, age);
        this.salary = salary;
    }

    @Override
    public void display() {
        super.display();
        System.out.println("Salary: " + salary);
    }
}

class Manager extends Employee {
    String department;

    public Manager(String name, int age, double salary, String
department) {
        super(name, age, salary);
        this.department = department;
    }
    public void displayManagedEmployees() {
        System.out.println(name + " manages employees in the " +
department + " department.");
    }

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

public class pract{
    public static void main(String[] args) {
        Manager manager = new Manager("John", 40, 85000, "IT");
        manager.display();
        manager.displayManagedEmployees();
    }
}
```

OUTPUT:



```
Name: John, Age: 40
Salary: 85000.0
Department: IT
John manages employees in the IT department.

C:\Users\ASE Computer Lab\Desktop>
```

22.AREA AND PERIMETER OF RECTANGLE AND CIRCLE

ALGORITHM:

1. Define Rectangle and Circle classes with methods for area and perimeter.

2.Use formulas: Rectangle → l×b, 2×(l+b); Circle → π×r², 2×π×r.

3.In main(), create objects, call methods, and print results.

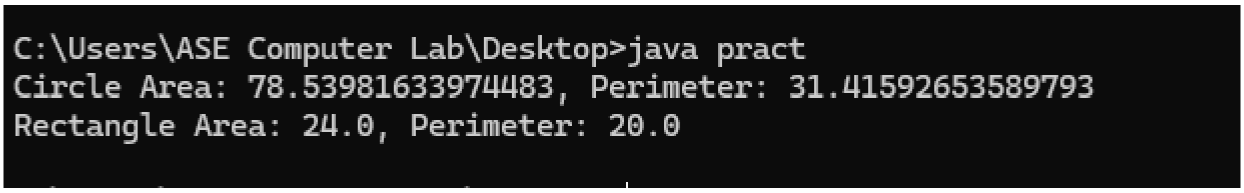
CODING:

```
abstract class Shape { abstract double calculateArea(); abstract double calculatePerimeter(); }

class Circle extends Shape { double radius;

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

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

@Override
double calculatePerimeter() {
    return 2 * Math.PI * radius;
}


}

class Rectangle extends Shape { double width, height;

public Rectangle(double width, double height) {
    this.width = width;
    this.height = height;
}

@Override
double calculateArea() {
    return width * height;
}

@Override
double calculatePerimeter() {
    return 2 * (width + height);
}
}
```

public class pract { public static void main(String[] args) { Shape circle = new Circle(5); Shape rectangle = new Rectangle(4, 6);

```
   System.out.println("Circle Area: " + circle.calculateArea() + ", Perimeter: " + circle.calculatePerimeter());

    System.out.println("Rectangle Area: " + rectangle.calculateArea() + ", Perimeter: " + rectangle.calculatePerimeter());
}


}
```

OUTPUT:



```
C:\Users\ASE Computer Lab\Desktop>java pract
Circle Area: 78.53981633974483, Perimeter: 31.41592653589793
Rectangle Area: 24.0, Perimeter: 20.0
```

23. BANK ACCOUNT MANAGEMENT

ALGORITHM:

1.Create a BankAccount class with variables like accountHolder, balance and methods for deposit(), withdraw(), and displayBalance().

2.Use conditions in withdraw() to check for sufficient balance before deduction.

3.In main(), create account objects, perform transactions, and show account details.

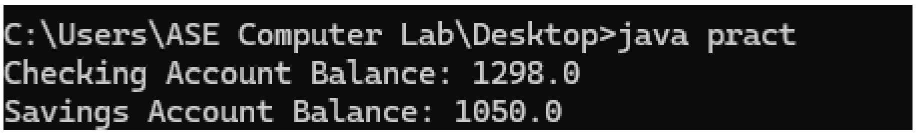
CODING:

```
class Account {

   String accountNumber;

   double balance;


   public Account(String accountNumber, double balance) {
       this.accountNumber = accountNumber;
       this.balance = balance;
   }

   public void deposit(double amount) {

       balance += amount;
   }

   public void withdraw(double amount) {
       if (balance >= amount) {
           balance -= amount;
       } else {
           System.out.println("Insufficient funds");
       }
   }

   public double getBalance() {
       return balance;
   }
}

class CheckingAccount extends Account {
   double transactionFee;
   public CheckingAccount(String accountNumber, double balance,
    double transactionFee) {

       super(accountNumber, balance);

       this.transactionFee = transactionFee;

   }


   @Override

   public void withdraw(double amount) {

       super.withdraw(amount);

       balance -= transactionFee;

   }

}


class SavingsAccount extends Account {

   double interestRate;


   public SavingsAccount(String accountNumber, double balance,
    double interestRate) {

       super(accountNumber, balance);

       this.interestRate = interestRate;

   }


   public void applyInterest() {

       balance += balance * interestRate;
```

```
    }
```

```
}


public class Main {

   public static void main(String[] args) {

      CheckingAccount checking = new CheckingAccount("CA123",
    1000, 2);

      SavingsAccount savings = new SavingsAccount("SA123", 1000,

0.05);



      checking.deposit(500);

      checking.withdraw(200);

      System.out.println("Checking Account Balance: " +
    checking.getBalance());



      savings.applyInterest();

      System.out.println("Savings Account Balance: " +
    savings.getBalance());

   }

}
```

OUTPUT:



```
C:\Users\ASE Computer Lab\Desktop>java pract
Checking Account Balance: 1298.0
Savings Account Balance: 1050.0
```

24. DRAWABLE INTERFACE

ALGORITHM:

1. Define a Drawable interface with a draw() method to visualize account info.
2. Create a BankAccount class that implements Drawable, with methods: deposit(), withdraw(), displayBalance(), and draw().

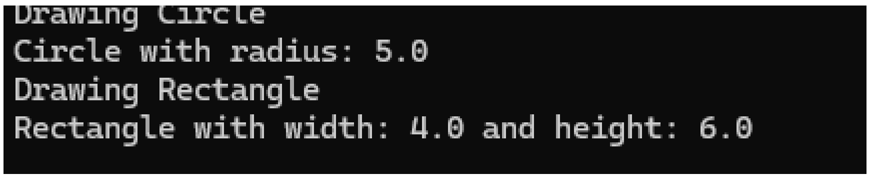
3. In draw(), display a visual-like representation of account status.
4. In main(), create account objects, perform transactions, and call draw() to show account summary.

CODING:

```
interface Drawable {
   void draw();
}

abstract class Shape implements Drawable {
   abstract void displayInfo();
}

class Circle extends Shape {
   double radius;

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

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

   @Override
   void displayInfo() {
      System.out.println("Circle with radius: " + radius);
   }
}

class Rectangle extends Shape {
   double width, height;

   public Rectangle(double width, double height) {
      this.width = width;
      this.height = height;
   }

   @Override
   public void draw() {
      System.out.println("Drawing Rectangle");
   }
```

```
    @Override
    void displayInfo() {
        System.out.println("Rectangle with width: " + width + " and
height: " + height);
    }
}

public class pract {
    public static void main(String[] args) {
        Shape circle = new Circle(5);
        Shape rectangle = new Rectangle(4, 6);

        circle.draw();
        circle.displayInfo();

        rectangle.draw();
        rectangle.displayInfo();
    }
}
```

OUTPUT:



```
Drawing Circle
Circle with radius: 5.0
Drawing Rectangle
Rectangle with width: 4.0 and height: 6.0
```

25. VEHICLE EFFICIENCY

ALGORITHM:

1. Define a base class Vehicle with attributes brand, fuelEfficiency, and a method calculateFuelEfficiency().
2. Create subclasses Car and Truck that extend Vehicle.
3. Override calculateFuelEfficiency() in both subclasses to provide specific logic:

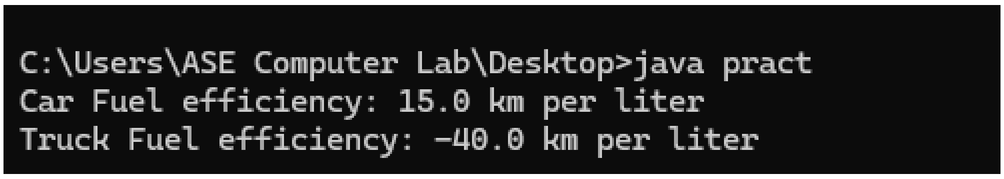
- Car → displays base efficiency.
- Truck → reduces efficiency based on load.

4. In main(), create objects of Car and Truck, and call their calculateFuelEfficiency() methods.

CODING:

```
class Vehicle {
    String brand;
    double fuelEfficiency;

    public Vehicle(String brand, double fuelEfficiency) {
        this.brand = brand;
        this.fuelEfficiency = fuelEfficiency;
    }

    public void calculateFuelEfficiency() {
        System.out.println("Fuel efficiency: " + fuelEfficiency + " km
per liter");
    }
}

class Car extends Vehicle {
    public Car(String brand, double fuelEfficiency) {
        super(brand, fuelEfficiency);

    }

    @Override
    public void calculateFuelEfficiency() {
        System.out.println("Car Fuel efficiency: " + fuelEfficiency + "

km per liter");
    }
}

class Truck extends Vehicle {
    double load;

    public Truck(String brand, double fuelEfficiency, double load) {
        super(brand, fuelEfficiency);
        this.load = load;
    }


    @Override
    public void calculateFuelEfficiency() {
        double reducedEfficiency = fuelEfficiency - (load * 0.1); //
Load decreases efficiency
        System.out.println("Truck Fuel efficiency: " +
reducedEfficiency + " km per liter");
    }
}

public class pract {
    public static void main(String[] args) {
            Vehicle car = new Car("Toyota", 15);
            Vehicle truck = new Truck("Ford", 10, 500);

            car.calculateFuelEfficiency();
            truck.calculateFuelEfficiency();
        }
    }
```

OUTPUT:



```
C:\Users\ASE Computer Lab\Desktop>java pract
Car Fuel efficiency: 15.0 km per liter
Truck Fuel efficiency: -40.0 km per liter
```

26. VOTING SYSTEM:

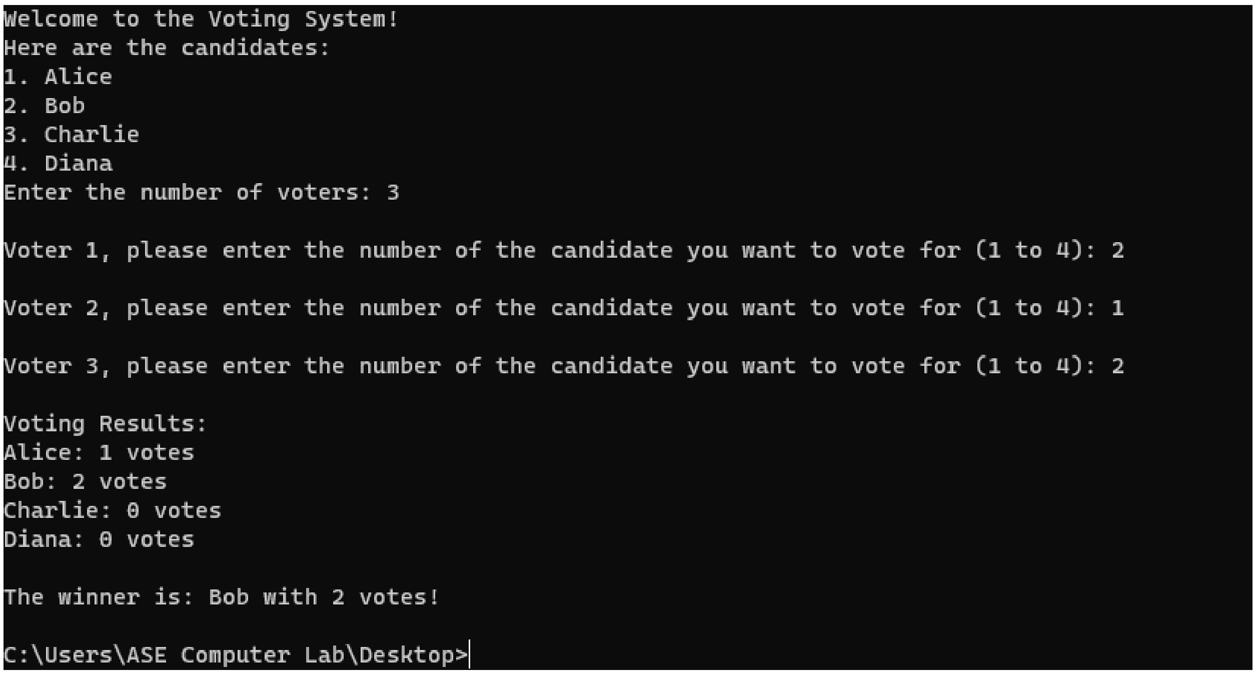
ALGORITHM:

- Display a list of candidates.
- Prompt voters to vote for their chosen candidate by entering the candidate's number.
- Store each vote in a separate counter for each candidate.
-
- Display the total number of votes each candidate received after all voting is complete

CODING:

```
import java.util.Scanner;


public class VotingSystem {


    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);



        String[] candidates = {

            "Alice",
            "Bob",
            "Charlie",
            "Diana"
        };
        int[] votes = new int[candidates.length];

        System.out.println("Welcome to the Voting System!");
        System.out.println("Here are the candidates:");
        for (int i = 0; i < candidates.length; i++) {
            System.out.println((i + 1) + ". " + candidates[i]);
        }
        System.out.print("Enter the number of voters: ");
        int numVoters = scanner.nextInt();
        for (int i = 0; i < numVoters; i++) {
            System.out.print("\nVoter " + (i + 1) + ", please enter the number of the candidate
    you want to vote for (1 to " + candidates.length + "): ");
            int vote = scanner.nextInt();
            if (vote >= 1 && vote <= candidates.length) {
                votes[vote - 1]++; // Increment vote count for the chosen candidate
            } else {
                System.out.println("Invalid vote! Please enter a valid candidate number.");
                i--; // Retry for this voter
            }
        }



        System.out.println("\nVoting Results:");
        for (int i = 0; i < candidates.length; i++) {
            System.out.println(candidates[i] + ": " + votes[i] + " votes");
        }
        int maxVotes = 0;

        String winner = "";

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

            if (votes[i] > maxVotes) {

                maxVotes = votes[i];

                winner = candidates[i];

            }

        }


        System.out.println("\nThe winner is: " + winner + " with " + maxVotes + " votes!");

    }

}
```

OUTPUT:



```
Welcome to the Voting System!
Here are the candidates:
1. Alice
2. Bob
3. Charlie
4. Diana
Enter the number of voters: 3

Voter 1, please enter the number of the candidate you want to vote for (1 to 4): 2

Voter 2, please enter the number of the candidate you want to vote for (1 to 4): 1

Voter 3, please enter the number of the candidate you want to vote for (1 to 4): 2

Voting Results:
Alice: 1 votes
Bob: 2 votes
Charlie: 0 votes
Diana: 0 votes

The winner is: Bob with 2 votes!

C:\Users\ASE Computer Lab\Desktop>
```

ABSTRACTION (ABSTRACT CLASS & INTERFERENCE) (8QS)

27.ONLINE EXAM

ALGORITHM:

- Question Management

- User Authentication
- Exam Delivery
- Answer Processing
- Scoring & Result Generation

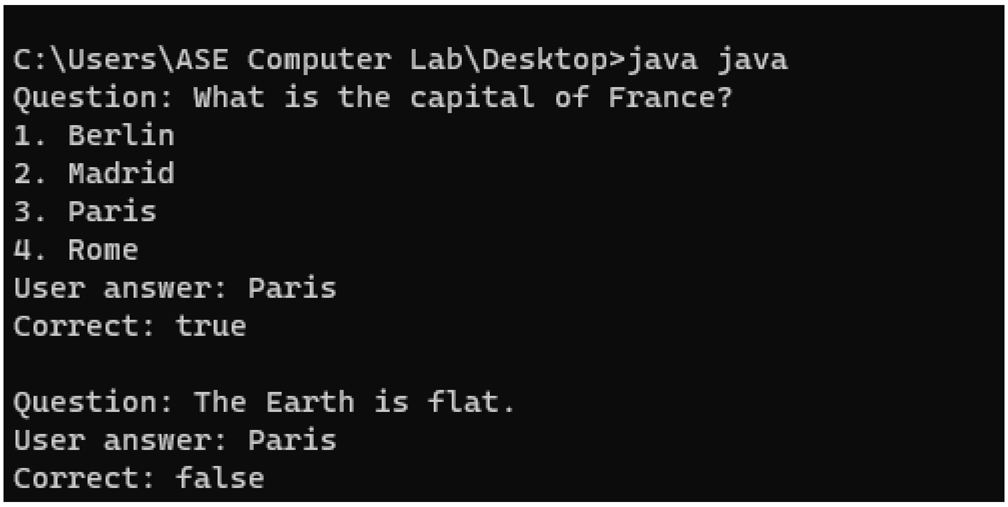
CODING:

```
abstract class Question {

    protected String questionText;

    public Question(String questionText) {

        this.questionText = questionText;

    }

    public abstract boolean checkAnswer(String answer);

     public void displayQuestion() {

        System.out.println("Question: " + questionText);

    }

}
```

```
class MultipleChoiceQuestion extends Question {
    private String[] options;

    private String correctAnswer;

    public MultipleChoiceQuestion(String questionText, String[] options, String
correctAnswer) {

        super(questionText);

        this.options = options;

        this.correctAnswer = correctAnswer;

    }

     @Override

    public boolean checkAnswer(String answer) {

        return correctAnswer.equalsIgnoreCase(answer);

    }

    public void displayOptions() {

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

            System.out.println((i + 1) + ". " + options[i]);
        }

    }

}

class TrueFalseQuestion extends Question {

    private boolean correctAnswer;

    public TrueFalseQuestion(String questionText, boolean correctAnswer) {

        super(questionText);

        this.correctAnswer = correctAnswer;

    }

    @Override

    public boolean checkAnswer(String answer) {

        return (correctAnswer && answer.equalsIgnoreCase("true")) || (!correctAnswer && 
answer.equalsIgnoreCase("false"));

    }

}
```

```
public class java {

    public static void main(String[] args) {

            Question[] questions = new Question[2];

        questions[0] = new MultipleChoiceQuestion("What is the capital of France?",

                                            new String[]{"Berlin", "Madrid", "Paris", "Rome"},

                                            "Paris");

        questions[1] = new TrueFalseQuestion("The Earth is flat.", false);

        for (Question question : questions) {

            question.displayQuestion();



            if (question instanceof MultipleChoiceQuestion) {

                ((MultipleChoiceQuestion) question).displayOptions();

            }

            String userAnswer = "Paris";

            System.out.println("User answer: " + userAnswer);
            System.out.println("Correct: " + question.checkAnswer(userAnswer));

            System.out.println();  // Blank line between questions

        }

    }

}
```

OUTPUT:



```
C:\Users\ASE Computer Lab\Desktop>java java
Question: What is the capital of France?
1. Berlin
2. Madrid
3. Paris
4. Rome
User answer: Paris
Correct: true

Question: The Earth is flat.
User answer: Paris
Correct: false
```

28. VOTING SYSTEM:

ALGORITHM:

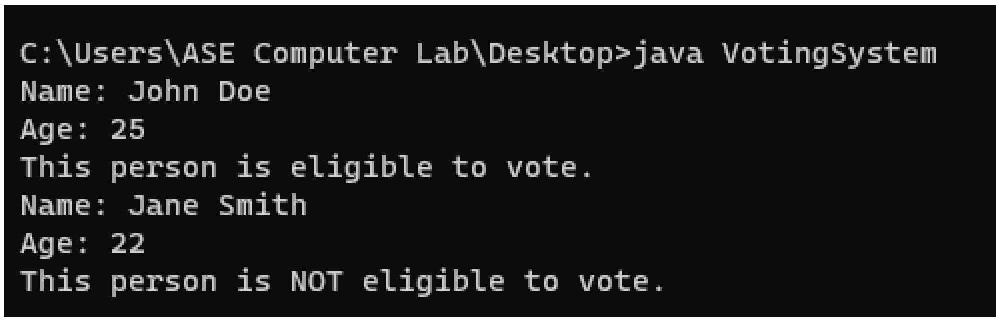
1. Initialize an array of candidate names and a corresponding vote counter array.
2. Ask for number of voters.
3. Loop through each voter:
    - Prompt for candidate number.
    - Validate input and increment respective candidate’s vote.
4. Display total votes for each candidate.
5. Find and display the candidate with the highest votes.

CODING:

```
abstract class Voter {
    protected String name;
    protected int age;
    public Voter(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public abstract boolean isEligibleToVote();
      public void displayVoterInfo() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}
 class CitizenVoter extends Voter {
    public CitizenVoter(String name, int age) {
        super(name, age);
    }


    @Override
    public boolean isEligibleToVote() {
        return age >= 18; // Eligible to vote if 18 or older
    }
}

class NonCitizenVoter extends Voter {
    public NonCitizenVoter(String name, int age) {
        super(name, age);
    }

    @Override
    public boolean isEligibleToVote() {
        return false; // Non-citizens are not eligible to vote
    }
}

public class VotingSystem {
    public static void main(String[] args) {
        Voter voter1 = new CitizenVoter("John Doe", 25);
        Voter voter2 = new NonCitizenVoter("Jane Smith", 22);

        voter1.displayVoterInfo();
        if (voter1.isEligibleToVote()) {
            System.out.println("This person is eligible to vote.");
        } else {
            System.out.println("This person is NOT eligible to vote.");
        }

        voter2.displayVoterInfo();
        if (voter2.isEligibleToVote()) {
            System.out.println("This person is eligible to vote.");
        } else {
```

```
        System.out.println("This person is NOT eligible to vote.");
      }
    }
}
```

OUTPUT:



```
C:\Users\ASE Computer Lab\Desktop>java VotingSystem
Name: John Doe
Age: 25
This person is eligible to vote.
Name: Jane Smith
Age: 22
This person is NOT eligible to vote.
```

29. BANK MANAGEMENT SYSTEM

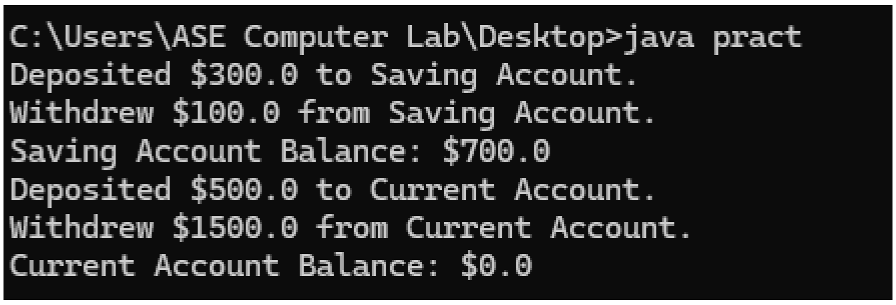
ALGORITHM:

1. Define an abstract class BankAccount with deposit, withdraw, and displayBalance methods.
2. Create SavingAccount and CurrentAccount subclasses.
3. Handle deposits/withdrawals with validations (overdraft for current).
4. In main, create instances, perform operations, and show balances.

CODING:

```
 abstract class BankAccount {

 protected double balance;


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

public abstract void deposit(double amount);
public abstract void withdraw(double amount);
public abstract void displayBalance();


}

class SavingAccount extends BankAccount {

public SavingAccount(double initialBalance) {
    super(initialBalance);
}

@Override
public void deposit(double amount) {
    if (amount > 0) {
        balance += amount;
        System.out.println("Deposited $" + amount + " to Saving
Account.");
    } else {
        System.out.println("Deposit amount should be greater than
zero.");
    }
}

@Override
public void withdraw(double amount) {
    if (amount > 0 && amount <= balance) {
        balance -= amount;
        System.out.println("Withdrew $" + amount + " from Saving
Account.");
    } else {
        System.out.println("Insufficient balance or invalid amount.");
    }
}


@Override
public void displayBalance() {
    System.out.println("Saving Account Balance: $" + balance);
}


}

 class CurrentAccount extends BankAccount {

private double overdraftLimit = 1000;

public CurrentAccount(double initialBalance) {
    super(initialBalance);
}

@Override
public void deposit(double amount) {
    if (amount > 0) {
        balance += amount;
        System.out.println("Deposited $" + amount + " to Current
Account.");
    } else {
        System.out.println("Deposit amount should be greater than
zero.");
    }
}

@Override
p
    public void withdraw(double amount) {
if (amount > 0 && balance - amount >= -overdraftLimit) {
        balance -= amount;
        System.out.println("Withdrew $" + amount + " from Current
Account.");
    } else {
        System.out.println("Insufficient balance or exceeds overdraft
limit.");
    }
}

@Override
public void displayBalance() {
    System.out.println("Current Account Balance: $" + balance);
}


}

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

 savingAccount = new SavingAccount(500);

 BankAccount currentAccount = new CurrentAccount(1000);
    savingAccount.deposit(300);
    savingAccount.withdraw(100);
    savingAccount.displayBalance();

    currentAccount.deposit(500);
    currentAccount.withdraw(1500);
    currentAccount.displayBalance();
}


}
```

OUTPUT:



```
C:\Users\ASE Computer Lab\Desktop>java pract
Deposited $300.0 to Saving Account.
Withdrew $100.0 from Saving Account.
Saving Account Balance: $700.0
Deposited $500.0 to Current Account.
Withdrew $1500.0 from Current Account.
Current Account Balance: $0.0
```

30.EMPLOYEE MANAGEMENT SYSTEM

ALGORITHM:

1. Create abstract Employee class with name and hoursWorked.

2.Subclass it into FullTimeEmployee and PartTimeEmployee with different rates.

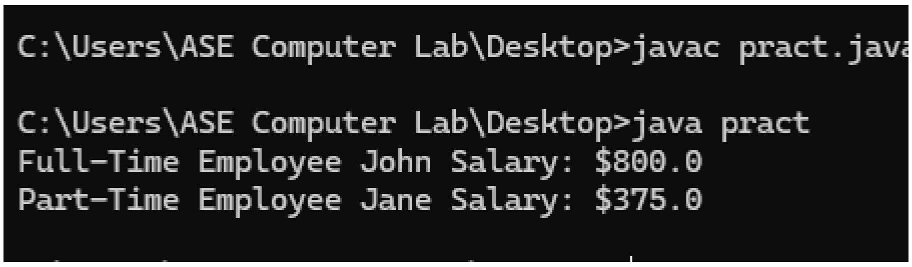
3. Override method to calculate salary.
4. In main, create objects and invoke calculateSalary.

CODING:

```
abstract class Employee {
   protected String name;
   protected int hoursWorked;

   public Employee(String name, int hoursWorked) {
      this.name = name;
      this.hoursWorked = hoursWorked;
   }    public abstract void calculateSalary();
}

class FullTimeEmployee extends Employee {
   private double hourlyRate = 20;

   public FullTimeEmployee(String name, int hoursWorked) {
      super(name, hoursWorked);
   }

   @Override
   public void calculateSalary() {
      double salary = hourlyRate * hoursWorked;
      System.out.println("Full-Time Employee " + name + " Salary: $" + salary);
   }
}
```

```
class PartTimeEmployee extends Employee {
    private double hourlyRate = 15;

    public PartTimeEmployee(String name, int hoursWorked) {
        super(name, hoursWorked);
    }

    @Override
    public void calculateSalary() {
        double salary = hourlyRate * hoursWorked;
        System.out.println("Part-Time Employee " + name + " Salary: $" + salary);
    }
}


public class EmployeeManagement {
    public static void main(String[] args) {
        Employee fullTimeEmp = new FullTimeEmployee("John", 40);
        Employee partTimeEmp = new PartTimeEmployee("Jane", 25);

        fullTimeEmp.calculateSalary();
        partTimeEmp.calculateSalary();
    }
}
```

OUTPUT:



```
C:\Users\ASE Computer Lab\Desktop>javac pract.java

C:\Users\ASE Computer Lab\Desktop>java pract
Full-Time Employee John Salary: $800.0
Part-Time Employee Jane Salary: $375.0
```

31.SHAPE DRAWING SYSTEM

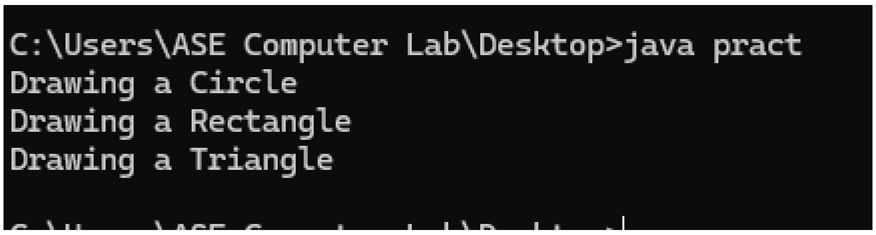
ALGORITHM:

1. Create abstract class Shape with draw() method.
2. Extend it into Circle, Rectangle, and Triangle.
3. Override draw() to print corresponding shape name.
4. In main, instantiate and call draw method for each shape.

CODING:

```
abstract class Shape {
   public abstract void draw();
}

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

class Rectangle extends Shape {
   @Override
   public void draw() {
      System.out.println("Drawing a Rectangle");
   }
}

class Triangle extends Shape {
   @Override
   public void draw() {
        System.out.println("Drawing a Triangle");
    }
}

public class ShapeDrawingSystem {
    public static void main(String[] args) {
        Shape circle = new Circle();
        Shape rectangle = new Rectangle();
        Shape triangle = new Triangle();

        circle.draw();
        rectangle.draw();
        triangle.draw();
    }
}
```

OUTPUT:



```
C:\Users\ASE Computer Lab\Desktop>java pract
Drawing a Circle
Drawing a Rectangle
Drawing a Triangle
```

32.VEHICLE MANAGEMENT SYSTEM:

ALGORITHM:

1. Define abstract Vehicle class with startEngine() method.

2.Create Car and Truck subclasses implementing the method.

3. Instantiate vehicle objects.

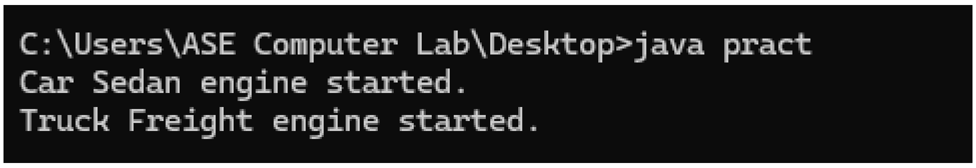
4. Call the startEngine() method to simulate behavior.

CODING:

```
abstract class Vehicle {
    protected String model;

    public Vehicle(String model) {
        this.model = model;
    }

    public abstract void startEngine();
}

class Car extends Vehicle {
    public Car(String model) {
        super(model);
    }

    @Override
    public void startEngine() {
        System.out.println("Car " + model + " engine started.");
    }
}

class Truck extends Vehicle {
    public Truck(String model) {
        super(model);
    }
```

```
    @Override
    public void startEngine() {
        System.out.println("Truck " + model + " engine started.");
    }
}

public class VehicleManagement {
    public static void main(String[] args) {
        Vehicle car = new Car("Sedan");
        Vehicle truck = new Truck("Freight");

        car.startEngine();
        truck.startEngine();
    }
}
```

OUTPUT:



```
C:\Users\ASE Computer Lab\Desktop>java pract
Car Sedan engine started.
Truck Freight engine started.
```

33.ACCOUNT MANAGEMENT SYSTEM

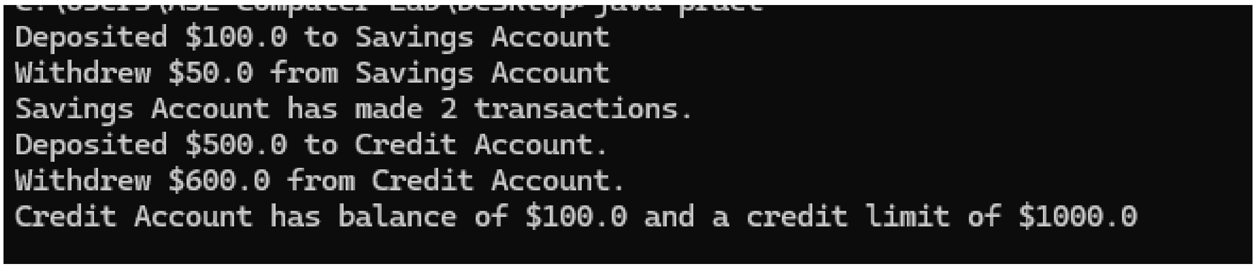
ALGORITHM:

1. Create abstract Account class with deposit, withdraw, and transactionHistory.
2. Extend into SavingsAccount and CreditAccount.
3. Implement methods with proper logic (including credit limit).
4. Perform operations in main and track history.

CODING:

```
abstract class Account {
    protected double balance;
    public Account(double balance) {
        this.balance = balance;
    }

    public abstract void deposit(double amount);
    public abstract void withdraw(double amount);
    public abstract void transactionHistory();
}

class SavingsAccount extends Account {
    private int transactionCount = 0;

    public SavingsAccount(double balance) {
        super(balance);
    }

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

    @Override
    public void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
            transactionCount++;
            System.out.println("Withdrew $" + amount + " from Savings Account");
        } else {
            System.out.println("Insufficient funds in Savings Account.");
        }
    }

    @Override
    public void transactionHistory() {
        System.out.println("Savings Account has made " + transactionCount + "
transactions.");
    }
}

class CreditAccount extends Account {
    private double creditLimit = 1000;

    public CreditAccount(double balance) {
        super(balance);
    }

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

    @Override
    public void withdraw(double amount) {
        if (balance + creditLimit >= amount) {
            balance -= amount;
            System.out.println("Withdrew $" + amount + " from Credit Account.");
        } else {
            System.out.println("Exceeds credit limit.");
        }
    }

    @Override
    public void transactionHistory() {
        System.out.println("Credit Account has balance of $" + balance + " and a credit
limit of $" + creditLimit);
    }
}

public class AccountManagementSystem {
    public static void main(String[] args) {
        Account savings = new SavingsAccount(500);
        Account credit = new CreditAccount(200);

        savings.deposit(100);
        savings.withdraw(50);
        savings.transactionHistory();

        credit.deposit(500);
        credit.withdraw(600);
        credit.transactionHistory();
    }
}
```

OUTPUT:



```
Deposited $100.0 to Savings Account
Withdrew $50.0 from Savings Account
Savings Account has made 2 transactions.
Deposited $500.0 to Credit Account.
Withdrew $600.0 from Credit Account.
Credit Account has balance of $100.0 and a credit limit of $1000.0
```

34. PRODUCT MANAGEMENT SYSTEM:

ALGORITHM:

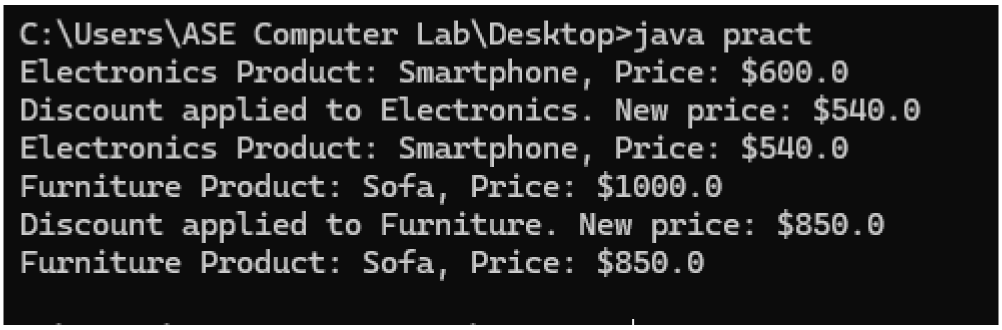
1. Define abstract Product class with applyDiscount() and displayProductDetails().
2. Subclass into Electronics and Furniture.
3. Implement discount logic (10% for electronics, 15% for furniture).
4. In main, call display and applyDiscount methods.

CODING:

```
abstract class Product {
   protected String name;
   protected double price;

   public Product(String name, double price) {
      this.name = name;
      this.price = price;
   }

   public abstract void applyDiscount();
   public abstract void displayProductDetails();
}

class Electronics extends Product {
   public Electronics(String name, double price) {
      super(name, price);
```

```
    }

    @Override
    public void applyDiscount() {
        price *= 0.9; // 10% discount
        System.out.println("Discount applied to Electronics. New price: $" + price);
    }

    @Override
    public void displayProductDetails() {
        System.out.println("Electronics Product: " + name + ", Price: $" + price);
    }
}

class Furniture extends Product {
    public Furniture(String name, double price) {
        super(name, price);
    }

    @Override
    public void applyDiscount() {

        price *= 0.85; // 15% discount
        System.out.println("Discount applied to Furniture. New price: $" + price);
    }

    @Override
    public void displayProductDetails() {
        System.out.println("Furniture Product: " + name + ", Price: $" + price);
    }
}


public class pract {
    public static void main(String[] args) {
        Product electronics = new Electronics("Smartphone", 600);
        Product furniture = new Furniture("Sofa", 1000);


        electronics.displayProductDetails();
        electronics.applyDiscount();
        electronics.displayProductDetails();


        furniture.displayProductDetails();
        furniture.applyDiscount();
        furniture.displayProductDetails();
    }
}
```

OUTPUT:



```
C:\Users\ASE Computer Lab\Desktop>java pract
Electronics Product: Smartphone, Price: $600.0
Discount applied to Electronics. New price: $540.0
Electronics Product: Smartphone, Price: $540.0
Furniture Product: Sofa, Price: $1000.0
Discount applied to Furniture. New price: $850.0
Furniture Product: Sofa, Price: $850.0
```

ENCAPSULATION (4QS)

35.

AIM: To display the employee details privately

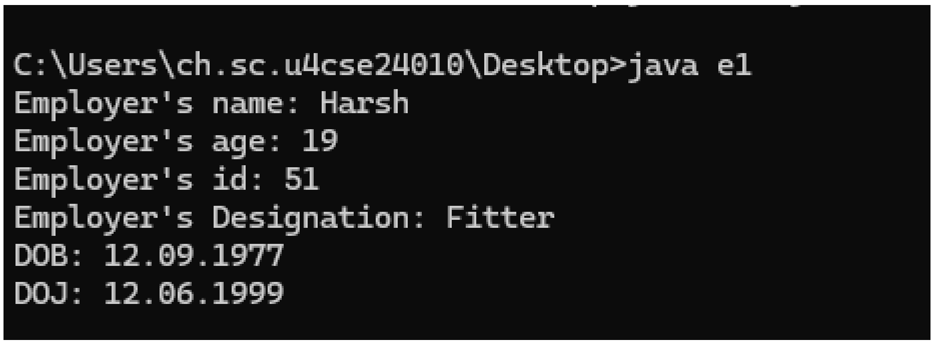
ALGORITHM:

- Start
- Create an `Employee` class with attributes
- Define a constructor to initialize values
- Define a `displayDetails()` method to show employee details
- In the `Main` class, create an `Employee` object with details
- Call `displayDetails()` method to print details
- Stop

CODE:

```
class Employee {
   private String name;
   private int age;
   private int id;
   private String designation;
   private String dob;
   private String doj;
   // Setter methods
   void setName(String n) {
      name = n;
   }
   void setAge(int a) {
      age = a;
   }
   void setId(int i) {
      id = i;
   }
   void setDesignation(String d) {
      designation = d;
   }


   void setDob(String d) {
      dob = d;
   }

   void setDoj(String d) {
      doj = d;
   }
   String getName() {
      return name;
   }
   int getAge() {
      return age;
   }
   int getId() {
      return id;
   }
   String getDesignation() {
      return designation;
   }
   String getDob() {
      return dob;
   }
   String getDoj() {
      return doj;
   }
}
public class Main {
   public static void main(String[] args) {
      Employee emp = new Employee();

      // Setting values without using 'this' or constructor
      emp.setName("Harsh");
      emp.setAge(19);
      emp.setId(51);
      emp.setDesignation("Fitter");
      emp.setDob("12.09.1977");
      emp.setDoj("12.06.1999");
      System.out.println("Employer's name: " + emp.getName());
      System.out.println("Employer's age: " + emp.getAge());
      System.out.println("Employer's id: " + emp.getId());
      System.out.println("Employer's Designation: " + emp.getDesignation());
      System.out.println("DOB: " + emp.getDob());
      System.out.println("DOJ: " + emp.getDoj());
   }
}
```

OUTPUT:



```
C:\Users\ch.sc.u4cse24010\Desktop>java e1
Employer's name: Harsh
Employer's age: 19
Employer's id: 51
Employer's Designation: Fitter
DOB: 12.09.1977
DOJ: 12.06.1999
```

36.

AIM: Car renting System

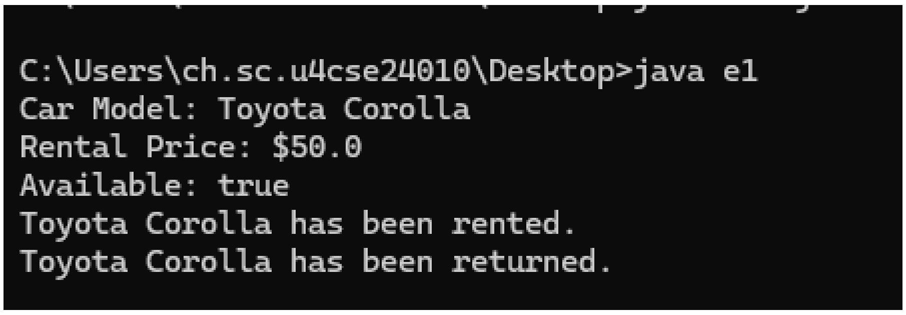
ALGORITHM:

- **Start**
- **Create `CarRental` class**
    - Define private attributes
    - Create setter methods for `carModel`, `rentalPrice`, and `availability`
    - Create getter methods for `carModel`, `rentalPrice`, and `availability`
    - Create `rentCar()` method
    - Create `returnCar()` method
- **Create `Main` class**
    - Create an object of `CarRental`
    - Set car details
    - Display car details
    - Rent the car
    - Return the car
- **Stop**

CODE:

```
class CarRental {
    private String carModel;
    private double rentalPrice;
    private boolean isAvailable;
    // Setter methods
    void setCarModel(String model) {
        carModel = model;
    }

    void setRentalPrice(double price) {
        if (price > 0) {
            rentalPrice = price;
        } else {
            System.out.println("Rental price must be greater than zero.");
        }
    }
    void setAvailability(boolean available) {
        isAvailable = available;
    }
    // Getter methods
    String getCarModel() {
        return carModel;
    }
    double getRentalPrice() {
        return rentalPrice;
    }
    boolean getAvailability() {
        return isAvailable;
    }

    // Rent a car
    void rentCar() {
        if (isAvailable) {
            isAvailable = false;
            System.out.println(carModel + " has been rented.");
        } else {
            System.out.println(carModel + " is not available.");
        }
    }

    // Return a car
    void returnCar() {
        isAvailable = true;
        System.out.println(carModel + " has been returned.");
    }
}

// Main class
public class Main {
    public static void main(String[] args) {
        CarRental car = new CarRental();
        car.setCarModel("Toyota Corolla");
        car.setRentalPrice(50);
        car.setAvailability(true);

        System.out.println("Car Model: " + car.getCarModel());
        System.out.println("Rental Price: $" + car.getRentalPrice());
        System.out.println("Available: " + car.getAvailability());

        car.rentCar();
        car.returnCar();
    }
}
```

OUTPUT:



```
C:\Users\ch.sc.u4cse24010\Desktop>java e1
Car Model: Toyota Corolla
Rental Price: $50.0
Available: true
Toyota Corolla has been rented.
Toyota Corolla has been returned.
```

37.

AIM: Online course platform

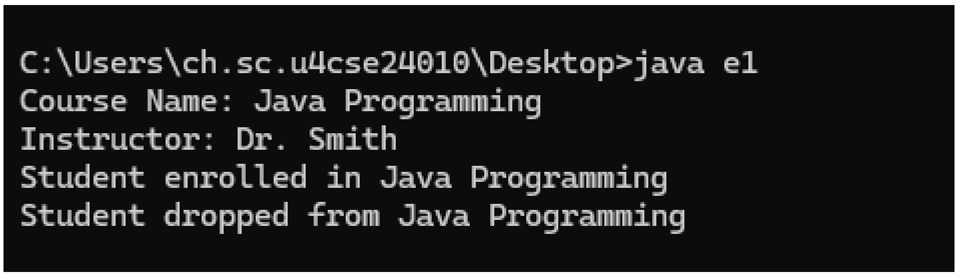
ALGORITHM:

1. Start
2. Create Course class
    a. Define private attributes
    b. Create setter methods for course details
    c. Create getter methods for course details
    d. Create enrollStudent() method
    e. Create dropStudent() method
3. Create Main class
    a. Create an object of Course
    b. Set course details
    c. Display course details
    d. Enroll a student
    e. Drop a student
4. Stop

CODE:

```
class Course {
    private String courseName;
    private String courseId;
    private String instructor;
    private int maxStudents;
    private int currentStudents;
    // Setter methods
    void setCourseName(String name) {
        courseName = name;
    }
    void setCourseId(String id) {
        courseId = id;
    }
    void setInstructor(String name) {
        instructor = name;
    }
    void setMaxStudents(int max) {
        if (max > 0) {
            maxStudents = max;
        }
    }
    void setCurrentStudents(int current) {
        if (current >= 0 && current <= maxStudents) {
            currentStudents = current;
        }
    }
    // Getter methods
    String getCourseName() {
        return courseName;
    }

    String getCourseId() {
        return courseId;
    }

    String getInstructor() {
        return instructor;
    }
    int getMaxStudents() {
        return maxStudents;
    }
    int getCurrentStudents() {
        return currentStudents;
    }
    // Method to enroll a student
    void enrollStudent() {
        if (currentStudents < maxStudents) {
            currentStudents++;
            System.out.println("Student enrolled in " + courseName);
        } else {
            System.out.println("Course is full.");
        }
    }

    // Method to drop a student
    void dropStudent() {
        if (currentStudents > 0) {
            currentStudents--;
            System.out.println("Student dropped from " + courseName);
        }
    }
}
public class e1 {
    public static void main(String[] args) {
        Course javaCourse = new Course();
        javaCourse.setCourseName("Java Programming");
        javaCourse.setCourseId("CS101");
        javaCourse.setInstructor("Dr. Smith");
        javaCourse.setMaxStudents(50);
        javaCourse.setCurrentStudents(45);

        System.out.println("Course Name: " + javaCourse.getCourseName());
        System.out.println("Instructor: " + javaCourse.getInstructor());

        javaCourse.enrollStudent();
        javaCourse.dropStudent();
    }
}
```

OUTPUT:



```
C:\Users\ch.sc.u4cse24010\Desktop>java e1
Course Name: Java Programming
Instructor: Dr. Smith
Student enrolled in Java Programming
Student dropped from Java Programming
```

38.

AIM: Library management system

ALGORITHM:

```
• Start
• Create LibraryBook class
    • Define private attributes
    • Create setter methods for book details
    • Create getter methods for book details
    • Create borrowBook() method
    • Create returnBook() method
• Create Main class
    • Create an object of LibraryBook
    • Set book details
    • Display book details
    • Borrow a book
    • Return a book
• Stop
```

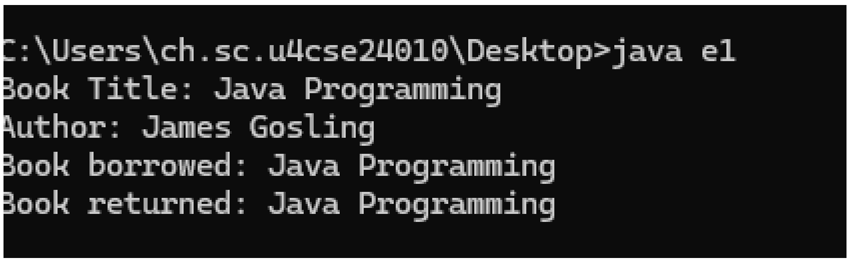
CODE:

```
class LibraryBook {
    private String bookId;
    private String title;
    private String author;
    private int availableCopies;

    // Setter methods
    void setBookId(String id) {
        bookId = id;
    }

    void setTitle(String bookTitle) {
        title = bookTitle;
    }
```

```
void setAuthor(String bookAuthor) {
    author = bookAuthor;
}

void setAvailableCopies(int copies) {
    if (copies >= 0) {
        availableCopies = copies;
    }
}

// Getter methods
String getBookId() {
    return bookId;
}

String getTitle() {
    return title;
}

String getAuthor() {
    return author;
}

int getAvailableCopies() {
    return availableCopies;
}

// Borrow a book
void borrowBook() {
    if (availableCopies > 0) {
        availableCopies--;

        System.out.println("Book borrowed: " + title);
    } else {
        System.out.println("Book is not available.");
    }
}

// Return a book
```

```
    void returnBook() {
        availableCopies++;
        System.out.println("Book returned: " + title);
    }
}

// Main class
public class Main {
    public static void main(String[] args) {
        LibraryBook book = new LibraryBook();
        book.setBookId("B101");
        book.setTitle("Java Programming");
        book.setAuthor("James Gosling");
        book.setAvailableCopies(5);

        System.out.println("Book Title: " + book.getTitle());
        System.out.println("Author: " + book.getAuthor());

        book.borrowBook();
        book.returnBook();
    }
}
```

OUTPUT:



```
C:\Users\ch.sc.u4cse24010\Desktop>java e1
Book Title: Java Programming
Author: James Gosling
Book borrowed: Java Programming
Book returned: Java Programming
```

PACKAGES (4QS)

39.

AIM: To find the median in an array list

ALGORITHM:

Step 1: Initialize Variables

1. Create an ArrayList<Integer> to store the numbers.
2. Create a Scanner object to take user input.

Step 2: Take Input from User

3. Prompt the user to enter the number of elements (n).
4. Read n from the user.
5. Prompt the user to enter n numbers.
6. Read each number and store it in the ArrayList.

Step 3: Sort the List

7. Use Collections.sort(numbers) to sort the list in ascending order.

Step 4: Find the Median

8. If n is even:
   a. Compute the median as the average of the two middle numbers:

median=numbers[n2−1]+numbers[n2]2.0\text{median} = \frac{\text{numbers}[\frac{n}{2} - 1] + \text{numbers}[\frac{n}{2}]}{2.0}median=2.0numbers[2n −1]+numbers[2n]

9. If n is odd:
   a. Median is the middle element in the sorted list:

median=numbers[n2]\text{median} = \text{numbers}[\frac{n}{2}]median=numbers[2n]

Step 5: Display Results

10. Print the sorted list.
11. Print the calculated median.

Step 6: Close Scanner

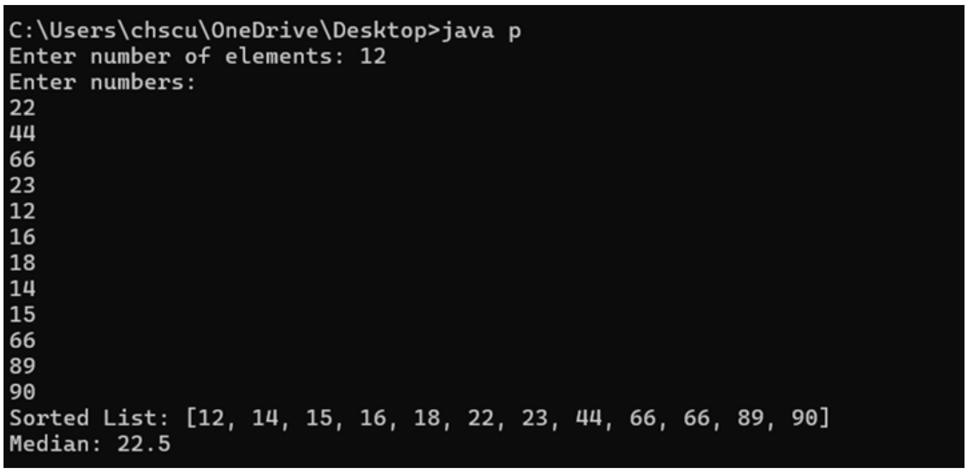
12. Close the Scanner to prevent resource leaks.

CODE:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Scanner;
public class MedianFinder {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      ArrayList<Integer> numbers = new ArrayList<>();

      System.out.print("Enter number of elements: ");
      int n = scanner.nextInt();

      System.out.println("Enter numbers:");
      for (int i = 0; i < n; i++) {
         numbers.add(scanner.nextInt());
      }

      Collections.sort(numbers);
      double median;
```

```
        if (n % 2 == 0) {
            median = (numbers.get(n / 2 - 1) + numbers.get(n / 2)) / 2.0;
        } else {
            median = numbers.get(n / 2);
        }
        System.out.println("Sorted List: " + numbers);
        System.out.println("Median: " + median);
        scanner.close();
    }
}
```

OUTPUT:



```
C:\Users\chscu\OneDrive\Desktop>java p
Enter number of elements: 12
Enter numbers:
22
44
66
23
12
16
18
14
15
66
89
90
Sorted List: [12, 14, 15, 16, 18, 22, 23, 44, 66, 66, 89, 90]
Median: 22.5
```

40.

AIM: To count the number of words, lines and characters

ALGORITHM:

Step 1: Initialize Counters

1. Declare and initialize three counters:
    a. charCount = 0 (to store the number of characters)
    b. wordCount = 0 (to store the number of words)
    c. lineCount = 0 (to store the number of lines)
2. Define the filename as "sample.txt".

Step 2: Open the File for Reading

3. Use a BufferedReader with a FileReader to read the file line by line.
4. If the file does not exist or an error occurs, handle it using a try-catch block.

Step 3: Read the File Line by Line

5. Start a loop to read each line from the file.
6. If the line is not null (i.e., there is still content in the file):
    a. Increment lineCount by 1.
    b. Add the length of the line to charCount (to count characters).
    c. Split the line into words using split("\\s+") (to count words).

   d. Increment wordCount by the number of words in the line.

Step 4: Display the Results

7. Print the total number of lines.
8. Print the total number of words.
9. Print the total number of characters.

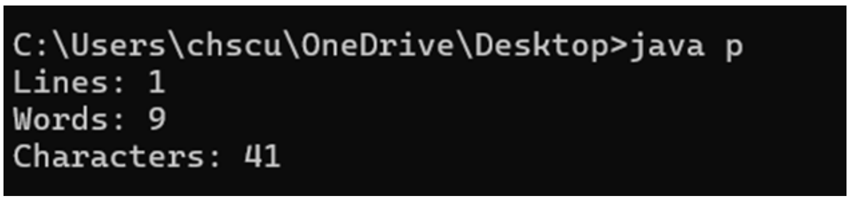
Step 5: Handle Exceptions

10. If an IOException occurs, print an error message.

CODE:

```
public class p {
   public static void main(String[] args) {
      String filename = "sample.txt";
      int charCount = 0, wordCount = 0, lineCount = 0;

      try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
         String line;
         while ((line = reader.readLine()) != null) {
            lineCount++;
            charCount += line.length();
            wordCount += line.split("\\s+").length;
         }
         System.out.println("Lines: " + lineCount);
         System.out.println("Words: " + wordCount);
         System.out.println("Characters: " + charCount);
      } catch (IOException e) {
         System.out.println("Error reading file: " + e.getMessage());
      }
   }
}
```

OUTPUT:



```
C:\Users\chscu\OneDrive\Desktop>java p
Lines: 1
Words: 9
Characters: 41
```

41.

AIM: To reverse set of words

ALGORITHM:

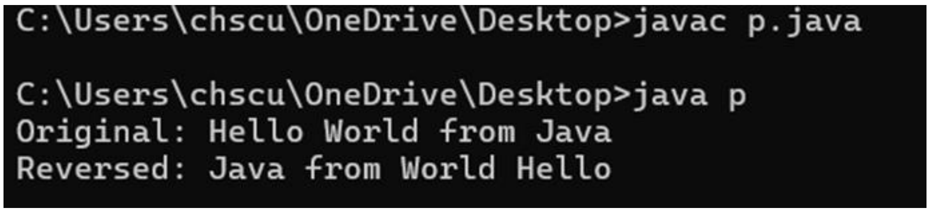
Step 1: Define a Function to Reverse Words

```
1. Input: A sentence (string).
2. Split the sentence into words using split("\\s+") (splitting based on spaces).
3. Initialize an empty StringBuilder to store the reversed sentence.
Step 2: Reverse the Order of Words
4. Iterate through the words array in reverse order (from last to first).
    a. Append each word to the StringBuilder.
    b. Add a space after each word.
5. Trim the final reversed sentence to remove the trailing space.
Step 3: Main Function Execution
6. Define a test sentence (e.g., "Hello World from Java").
7. Call the function reverseWords(sentence) to get the reversed sentence.
8. Print the original sentence.
9. Print the reversed sentence.
```

CODE:

```java
public class p {
   public static String reverseWords(String sentence) {
      String[] words = sentence.split("\\s+");
      StringBuilder reversed = new StringBuilder();
      for (int i = words.length - 1; i >= 0; i--) {
         reversed.append(words[i]).append(" ");
      }
      return reversed.toString().trim();
   }
   public static void main(String[] args) {
      String sentence = "Hello World from Java";
      System.out.println("Original: " + sentence);
      System.out.println("Reversed: " + reverseWords(sentence));
   }
}
```

OUTPUT:



```
C:\Users\chscu\OneDrive\Desktop>javac p.java

C:\Users\chscu\OneDrive\Desktop>java p
Original: Hello World from Java
Reversed: Java from World Hello
```

42.

AIM: To read and write a file

ALGORITHM:

Step 1: Define the File Name

1. Set filename = "data.txt".

Step 2: Write Data to the File

2. Open the file using FileWriter in write mode.
3. Write the following text to the file:
   a. "Hello, this is a test file!" (followed by a newline).
   b. "Java File Handling Example."
4. Close the FileWriter automatically using a try-with-resources block.
5. If an error occurs, print "Error writing file" along with the error message.

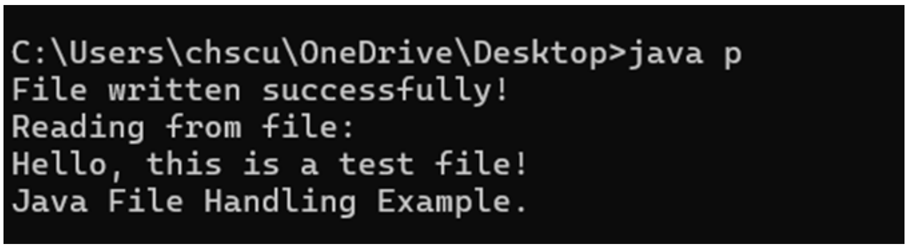
Step 3: Read Data from the File

6. Open the file using BufferedReader with FileReader.
7. Read the file line by line:
   a. Print each line to the console.
8. Close the BufferedReader automatically using a try-with-resources block.
9. If an error occurs, print "Error reading file" along with the error message.

CODE:

```
import java.io.*;
public class p {
    public static void main(String[] args) {
        String filename = "data.txt";
        try (FileWriter writer = new FileWriter(filename)) {
            writer.write("Hello, this is a test file!\n");
            writer.write("Java File Handling Example.");
            System.out.println("File written successfully!");
        } catch (IOException e) {

            System.out.println("Error writing file: " + e.getMessage());
        }
        try (BufferedReader reader = new BufferedReader(new
         FileReader(filename))) {
            String line;
            System.out.println("Reading from file:");
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        }
    }
}
```

OUTPUT:



```
C:\Users\chscu\OneDrive\Desktop>java p
File written successfully!
Reading from file:
Hello, this is a test file!
Java File Handling Example.
```

EXCEPTION HANDLING (4QS)

43. AIM:

To handle division by zero when calculating the average monthly balance in a banking application.

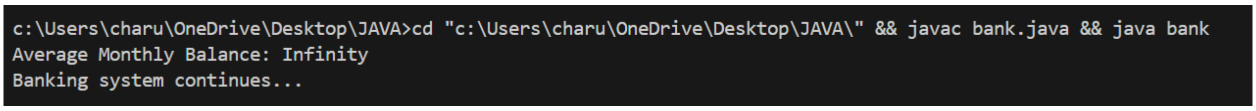
ALGORITHM:

1. Start
2. Declare `totalBalance` and `months` variables.
3. Use a `try` block to calculate `totalBalance / months`.
4. If `months` is zero, an `ArithmeticException` occurs.
5. Catch the exception and display an error message.
6. Continue program execution.
7. End

CODING:

```
public class BankingApp {
    public static void main(String[] args) {
        double totalBalance = 5000;
        int months = 0; // This will cause a division by zero


        try {
            double avgBalance = totalBalance / months;
            System.out.println("Average Monthly Balance: " + avgBalance);
        } catch (ArithmeticException e) {
            System.out.println("Error: Cannot divide by zero. Please enter a valid number of
months.");
        }
        System.out.println("Banking system continues...");
    }
}
```

OUTPUT:



```
c:\Users\charu\OneDrive\Desktop\JAVA>cd "c:\Users\charu\OneDrive\Desktop\JAVA\" && javac bank.java && java bank
Average Monthly Balance: Infinity
Banking system continues...
```

44. ONLINE SHOPPING SYSTEM

Aim:

To handle multiple exceptions such as **ArrayIndexOutOfBoundsException** and **NullPointerException** in an online shopping system.

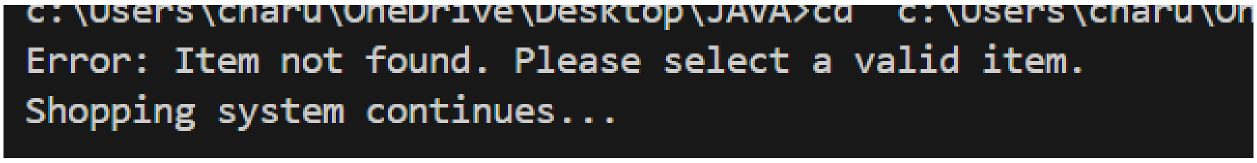
Algorithm:

1. Start
2. Declare an array of items.
3. Assign an invalid index and a null value for testing.
4. Use a try block to access the array and get the length of the null item.
5. If an index is invalid, catch ArrayIndexOutOfBoundsException.
6. If an item is null, catch NullPointerException.
7. Continue program execution.
8. End

CODING:

```
public class OnlineShopping {

    public static void main(String[] args) {

        String[] items = {"Laptop", "Phone", "Headphones"};

        int itemIndex = 5; // Invalid index
        String selectedItem = null; // Simulating a null selection


        try {
            System.out.println("Selected Item: " + items[itemIndex]); //
      ArrayIndexOutOfBoundsException
            System.out.println("Item Length: " + selectedItem.length()); //
      NullPointerException
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Error: Item not found. Please select a valid item.");
        } catch (NullPointerException e) {
            System.out.println("Error: No item selected. Please choose an item.");
        }
        System.out.println("Shopping system continues...");
    }
      }
```

OUTPUT:



```
c:\Users\charu\OneDrive\Desktop\JAVA>cd  c:\Users\charu\On
Error: Item not found. Please select a valid item.
Shopping system continues...
```

45. FLIGHT BOOKING SYSTEM

Aim:

To use throws and finally for handling invalid passenger numbers in a flight booking system.

Algorithm:

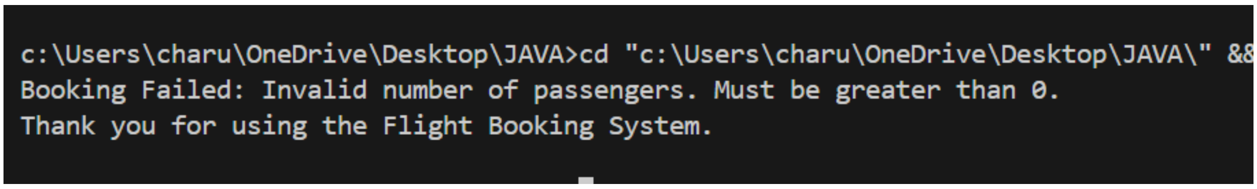
1. Start
2. Define a method bookFlight(int passengers).
3. If passengers is less than or equal to 0, throw an exception.

4. Use a try block to call bookFlight().
5. Catch the exception and display an error message.
6. Use finally to ensure cleanup.
7. End

CODING:

```
public class bank {


   public static void bookFlight(int passengers) throws IllegalArgumentException {

      if (passengers <= 0) {

         throw new IllegalArgumentException("Invalid number of passengers. Must be
greater than 0.");

      } else {

         System.out.println("Flight booked successfully for " + passengers + "
passengers.");

      }

   }


   public static void main(String[] args) {

      try {

         // Step 4: Call the method inside try block

         bookFlight(0);  // Change this value to test valid/invalid cases

      }

      catch (IllegalArgumentException e) {

         System.out.println("Booking Failed: " + e.getMessage());


      }

      finally {

         System.out.println("Thank you for using the Flight Booking System.");

      }

   }
}
```

OUTPUT:



```
c:\Users\charu\OneDrive\Desktop\JAVA>cd "c:\Users\charu\OneDrive\Desktop\JAVA\" &&
Booking Failed: Invalid number of passengers. Must be greater than 0.
Thank you for using the Flight Booking System.
```

46. ATM WITHRAWAL SYSTEM

Aim:

To implement an ATM withdrawal system with **custom exception handling** for invalid withdrawal amounts and insufficient balance.

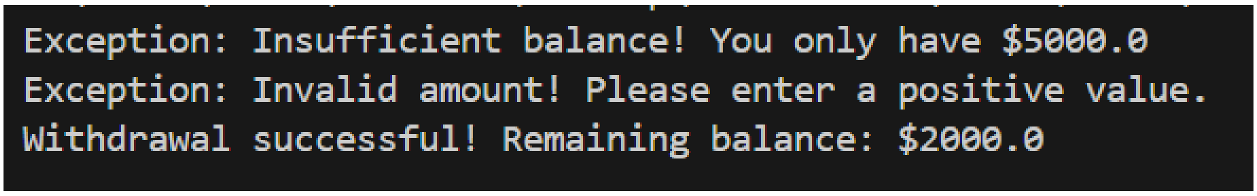
Algorithm:

1. Start
2. Define a class InsufficientBalanceException (custom exception).
3. Define a class InvalidAmountException (custom exception).
4. Create a class ATM with a withdraw() method:
    - If the withdrawal amount is **negative**, throw InvalidAmountException.
    - If the withdrawal amount exceeds **balance**, throw InsufficientBalanceException.
    - Otherwise, deduct the amount and display the remaining balance.
5. Use a try-catch block in main() to handle exceptions.
6. End

CODING:

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

class InvalidAmountException extends Exception {
    public InvalidAmountException(String message) {
        super(message);
    }
}
class ATM {
    private double balance;

    public ATM(double balance) {
        this.balance = balance;
    }

    public void withdraw(double amount) throws InsufficientBalanceException,
InvalidAmountException {
        if (amount <= 0) {
            throw new InvalidAmountException("Invalid amount! Please enter a positive
value.");
        }
        if (amount > balance) {
            throw new InsufficientBalanceException("Insufficient balance! You only have $"
+ balance);

        }
        balance -= amount;
        System.out.println("Withdrawal successful! Remaining balance: $" + balance);
    }
}
public class ATMSystem {
    public static void main(String[] args) {
        ATM atm = new ATM(5000); // Initial balance = $5000


        try {
            atm.withdraw(6000);  // Exceeds balance → InsufficientBalanceException
        } catch (InsufficientBalanceException | InvalidAmountException e) {
            System.out.println("Exception: " + e.getMessage());
        }


        try {
            atm.withdraw(-100); // Negative amount → InvalidAmountException
        } catch (InsufficientBalanceException | InvalidAmountException e) {
            System.out.println("Exception: " + e.getMessage());
        }


        try {
            atm.withdraw(3000); // Valid withdrawal → Success
        } catch (InsufficientBalanceException | InvalidAmountException e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
```

OUTPUT:



```
Exception: Insufficient balance! You only have $5000.0
Exception: Invalid amount! Please enter a positive value.
Withdrawal successful! Remaining balance: $2000.0
```

FILE HANDLING (4QS)

47.

AIM: To write a file

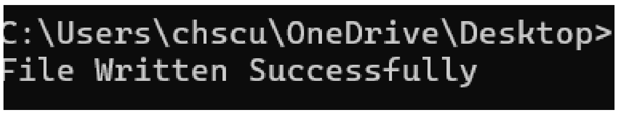
ALGORITHM:

```
Step 1: Start
Step 2: Try to execute the following steps (use try-catch for exception handling)
Step 3: Create a FileWriter object and associate it with a file named "example.txt"
Step 4: Write the text "hello everyone your safety our responsibility" into the file
Step 5: Close the FileWriter to save the content and release resources
Step 6: Print "File Written Successfully" if no error occurs
Step 7: If an IOException occurs, catch the exception and print "An error
occurred"
Step 8: Print the stack trace of the exception for debugging
Step 9: End
```

CODE:

```
import java.io.FileWriter;

import java.io.IOException;

public class e{

public static void main(String[] args){

try{

FileWriter writer = new FileWriter("example.txt");

writer.write("hello everyone your safety our responsibility");
writer.close();

System.out.println("File Written Successfully");

}

catch (IOException e){

System.out.println("An error occured");

e.printStackTrace();


}

}

}
```

OUTPUT:



```
C:\Users\chscu\OneDrive\Desktop>
File Written Successfully
```

48.

AIM: To read a file

ALGORITHM:

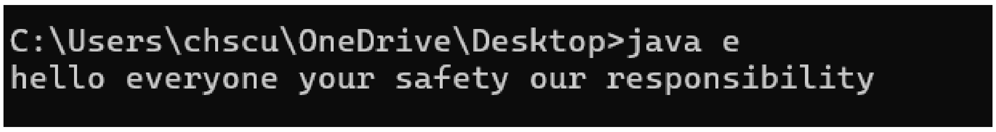
```
Step 1: Start
Step 2: Use a try-catch block to handle potential exceptions
Step 3: Create a File object pointing to "example.txt"
Step 4: Create a Scanner object to read from the file
Step 5: While there is another line to read in the file:
```

CODE:

```
import java.io.FileNotFoundException;

import java.util.Scanner;

public class e {

   public static void main(String[] args) {

      try {

         File file = new File("example.txt");
```

```
            Scanner reader = new Scanner(file);
            while (reader.hasNextLine()) {
                String data = reader.nextLine();
                System.out.println(data);
            }
            reader.close();
        } catch (FileNotFoundException e) {

            System.out.println("File not found.");
            e.printStackTrace();
        }
    }
}
```

OUTPUT:



```
C:\Users\chscu\OneDrive\Desktop>java e
hello everyone your safety our responsibility
```

49.

AIM: **T**o append the contents in a file

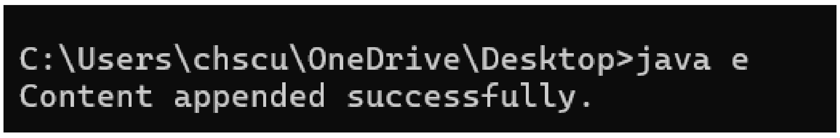
ALGORITHM:

Step 1: Start
Step 2: Use a try-catch block to handle exceptions
Step 3: Create a FileWriter object for "example.txt" in append mode (i.e., true)
Step 4: Use the write() method to append the text "\nThis line is appended." to the file
Step 5: Close the FileWriter to save changes and release resources
Step 6: Print "Content appended successfully." if no error occurs
Step 7: If an IOException occurs:

CODE:

```
import java.io.FileWriter;

import java.io.IOException;

public class e {

    public static void main(String[] args) {

        try {

            FileWriter writer = new FileWriter("example.txt", true); // true =
append mode

            writer.write("\nThis line is appended.");

            writer.close();

            System.out.println("Content appended successfully.");

        } catch (IOException e) {

            System.out.println("An error occurred.");

            e.printStackTrace();

        }

    }

}
```

OUTPUT:



```
C:\Users\chscu\OneDrive\Desktop>java e
Content appended successfully.
```

50.

AIM: To delete a file

ALGORITHM:

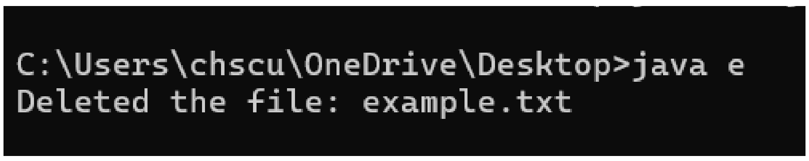
```
Step 1: Start
Step 2: Create a File object and associate it with "example.txt"
Step 3: Use the delete() method to attempt file deletion
Step 4: If the file is successfully deleted:
```

- Print "Deleted the file: example.txt"
  Step 5: Otherwise:
- Print "Failed to delete the file."
  Step 6: End

CODE:

```
import java.io.File;

public class DeleteFile {

    public static void main(String[] args) {

        File file = new File("example.txt");

        if (file.delete()) {

            System.out.println("Deleted the file: " + file.getName());

        } else {

            System.out.println("Failed to delete the file.");

        }

    }

}
```

OUTPUT:



```
C:\Users\chscu\OneDrive\Desktop>java e
Deleted the file: example.txt
```