



**SCHOOL OF
COMPUTING**

LAB RECORD

23CSE111-Object Oriented Programming

Submitted by

HASINI BALABOMMU

CH.SC.U4CSE24010

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND

ENGINEERING

AMRITA VISHWA VIDYAPEETHAM

AMRITA SCHOOL OF COMPUTING

CHENNAI

APRIL-2025



**SCHOOL OF
COMPUTING**

**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.U4CSE24010 – HASINI BALABOMMU** 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: 08/04/2025

Course Incharge

INDEX

S.NO	TITLE	PAGE.NO
	UML DIAGRAM	
1.	LIBRARY MANAGEMENT SYSTEM	1
	1.a)Use Case Diagram	
	1.b)Class Diagram	
	1.c) Sequence Diagram	
	1.d)Activity diagram	
	1.e)State diagram	
2.	HOSPITAL MANAGEMENT SYSTEM	5
	2.a) Use Case Diagram	
	2.b) Class Diagram	
	2.c) Sequence Diagram	
	2.d) Activity diagram	
	2.e) Object Diagram	
3.	BASIC JAVA PROGRAMS	9
	3.a) AREA OF TRIANGLE	
	3.b) AVERAGE OF 3 NUMBERS	
	3.c) STUDENT GRADE DETECTION	
	3.d) CATEGORIZE PEOPLE BASED ON AGE	
	3.e)CALCULATE ELECTRICITY BILL	
	3.f)ODD NUMBERS WITHIN A RANGE	
	3.g)PALINDROME CHECK	
	3.h)FACTORIAL OF A GIVEN NUMBER	
	3.i)EVEN OR ODD NUMBER	
	3.j)FIBONACCO SERIES	
4.	INHERITANCE	20
	SINGLE INHERITANCE PROGRAMS	
	4.a) EMPLOYEE PAY SLIP	
	4.b)DISPLAY EMPLOYEE DETAILS	

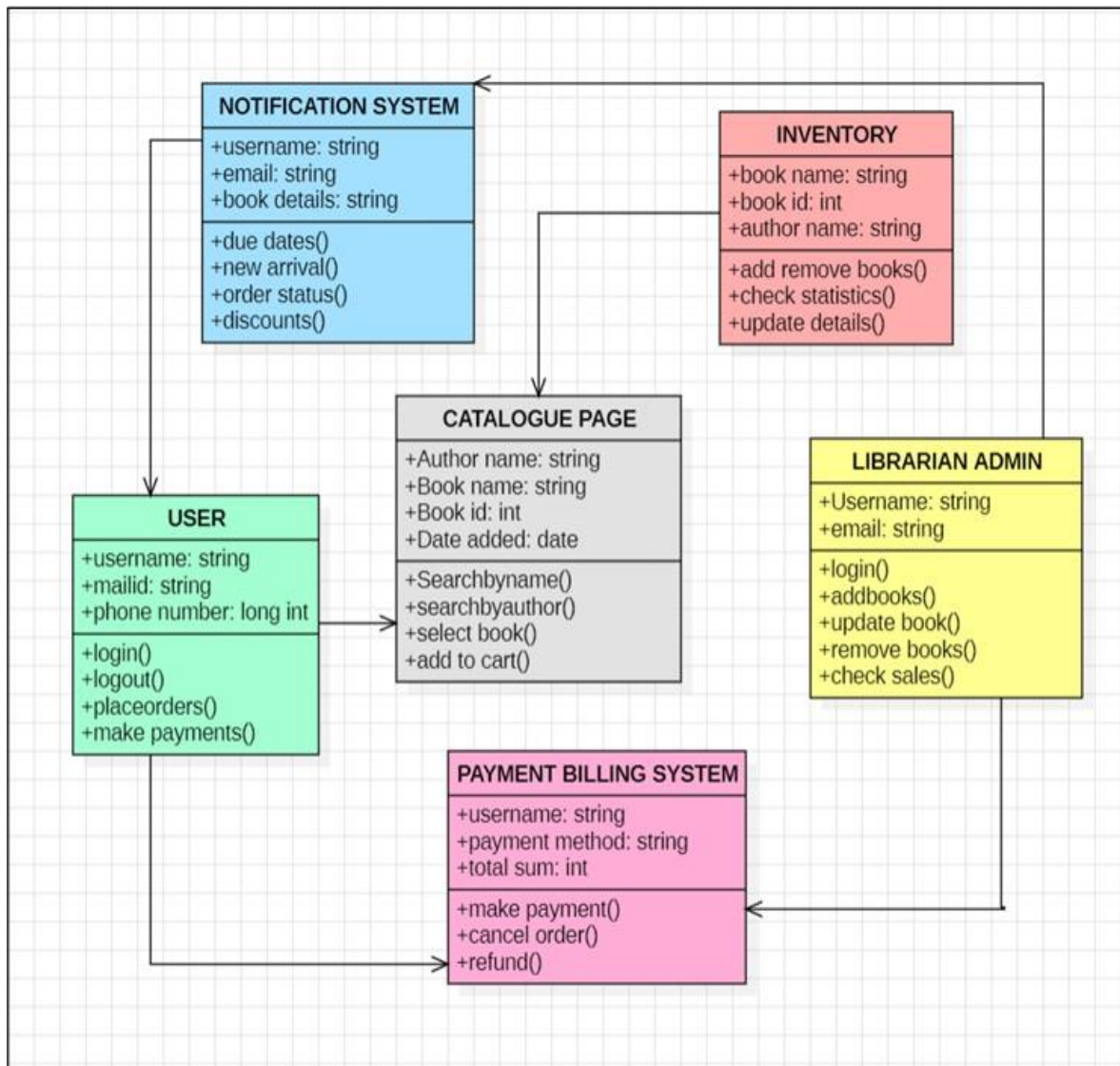
	MULTILEVEL INHERITANCE PROGRAMS	
	5.a)DISPLAY PERSONAL DETAILS	
	5.b)BANK MANGEMENT	
	HIERARCHICAL INHERITANCE PROGRAMS	
	6.a)AREA CALCULATOR	
	6.b)SCHOOL TASK MANAGEMENT	
	MULTIPLE INHERITANCE PROGRAMS	
	7.a)IMPLEMENTING MULTIPLE METHOD	
	7.b)IMPLEMENT ROLES OF VARIOUS JOB ROLE S	
5.	POLYMORPHISM	39
5A)	OVERRIDING PROGRAMS	40
	1. OVERRIDING SALARY METHODS	
	2. OVERRIDING NUMBER OF PLAYERS IN EACH GAME	
	3. OVERRIDING SPEEDS OF A VEHICLE	
	4. INTEREST RATE OF BANK	
5B)	METHOD OVERRIDING PROGRAMS	48
	1. E – COMMERCE APPLICATION	
	2. AREA CALCULATOR	
	3. ONLINE SHOPPING	
	4. FOOD DELIVERY TIME CALCULATOR	
6.	ABSTRACTION	60
6A)	INTERFACE PROGRAMS	70
	1. AREA CALCULATOR	
	2. VEHICLE INTERFACE	
	3. PAYMENT INTERFACE	
	4. MULTIPLE INTERFACE IMPLEMENTATION	
6B)	ABSTRACT CLASS PROGRAMS	61
	1. ONLINE TICKETING SYSTEM	
	2. PEN ABSTRACT CLASS	
	3. ONLINE EXAM	
	4. VOTING EXAM	
7.	ENCAPSULATION	76
.	ENCAPSULATION PROGRAMS	
	1. DISPLAY EMPLOYEE DETAILS PRIVATELY	
	2. CAR RENTING SYSTEM	
	3. ONLINE COURSE	
	4. LIBRARY MANAGEMENT SYSTEM	
8.	PACKAGES PROGRAMS	87
	1. MEDIAN IN ARRAYLIST	
	2. COUNT NUMBER OF CHARACTERS	
	3. REVERSE SET OF WORDS	
	4. READ AND WRITE A FILE	

9.	FILE HANDLING PROGRAMS	94
	1. WRITE A FILE	
	2. READ A FILE	
	3. APPEND A FILE	
	4. DELETE A FILE	
10.	EXCEPTION HANDLING PROGRAMS	98
	1. DIVISION BY ZERO	
	2. NULL POINTER EXCEPTION	
	3. HANDLING PASSENGERS	
	4. ATM WITHDRAWAL SYSTEM	

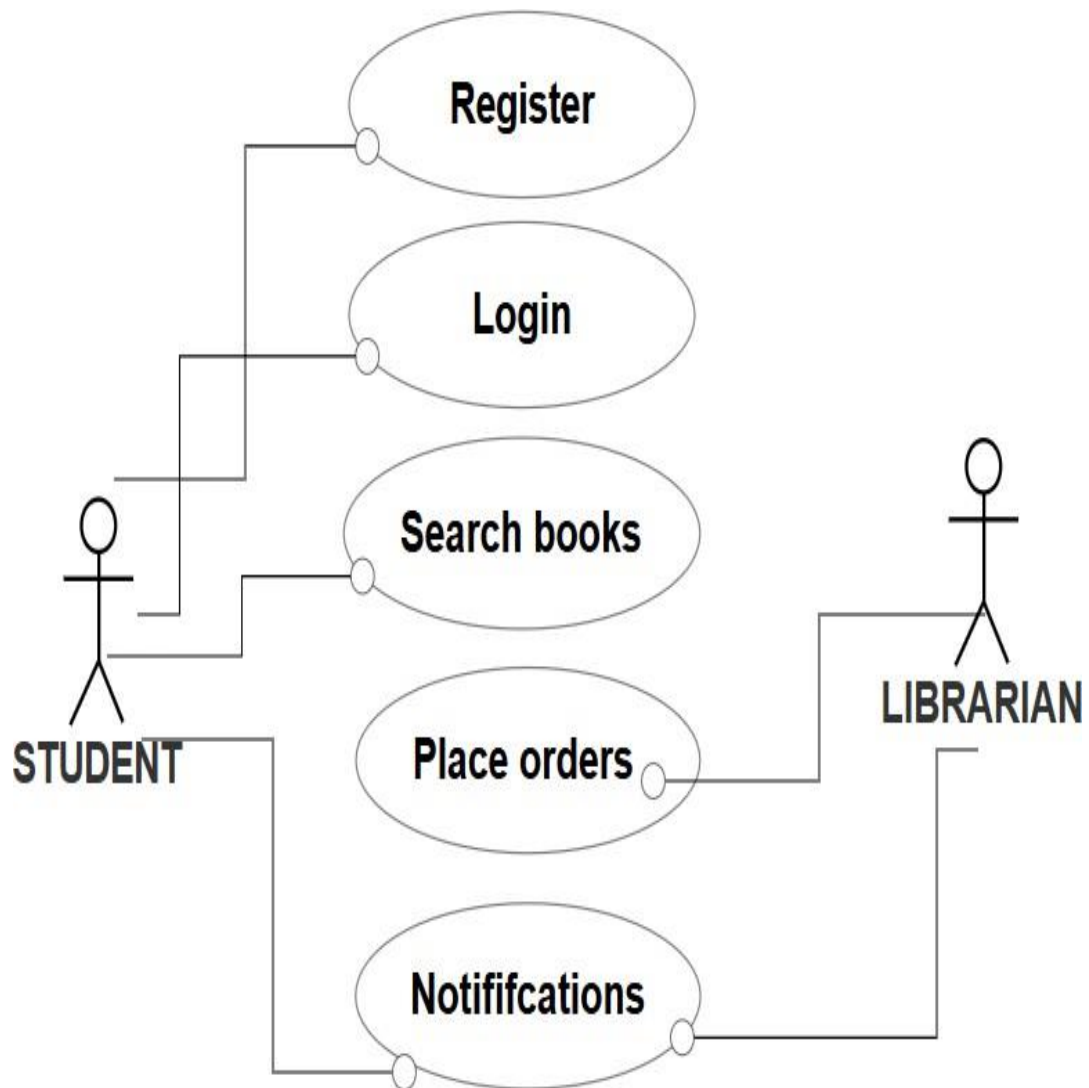
UML DIAGRAMS

1. LIBRARY MANAGEMENT SYSTEM

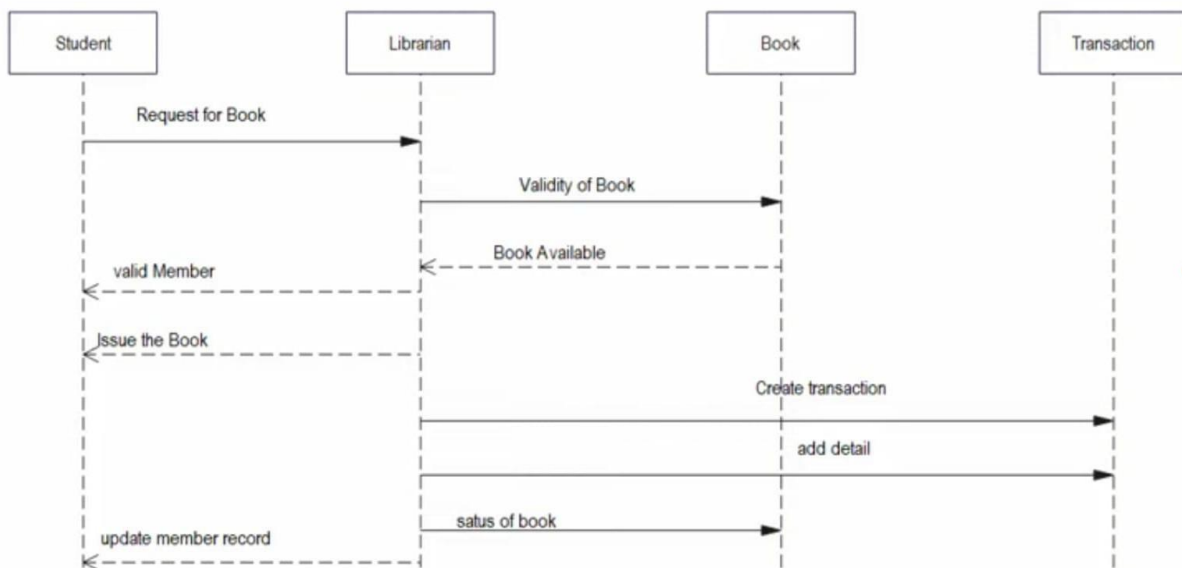
1A) CLASS DIAGRAM:



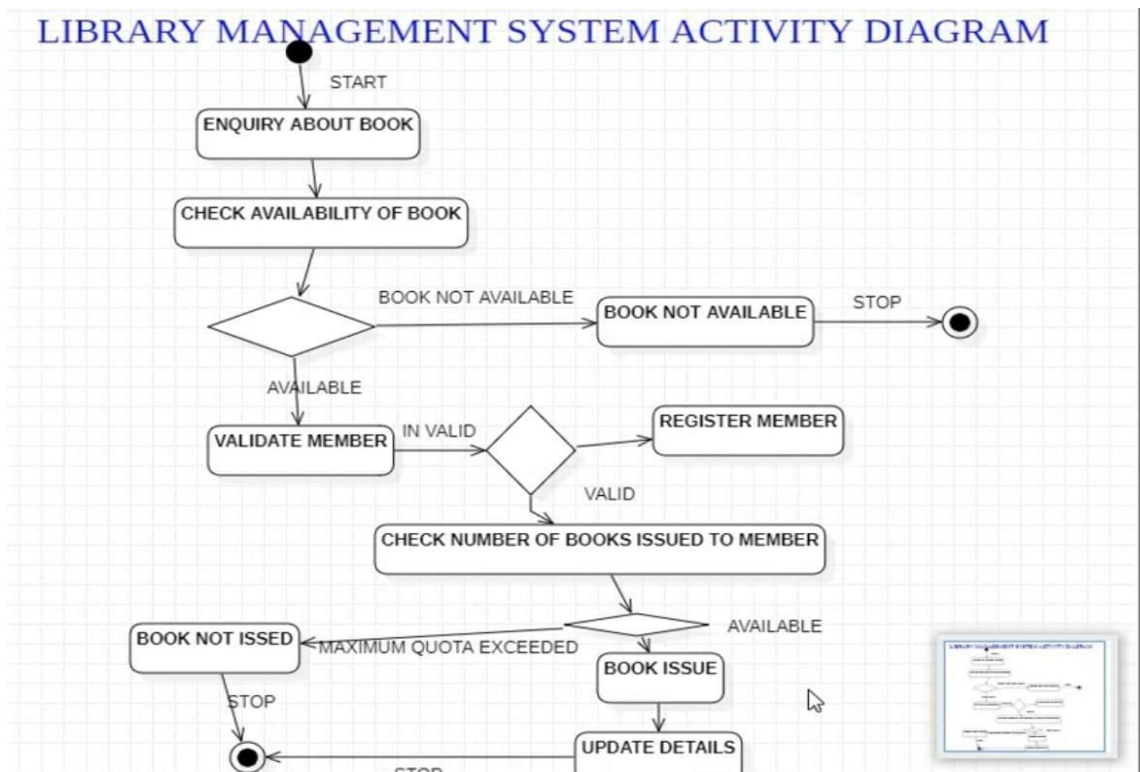
1B) USE CASE DIAGRAM



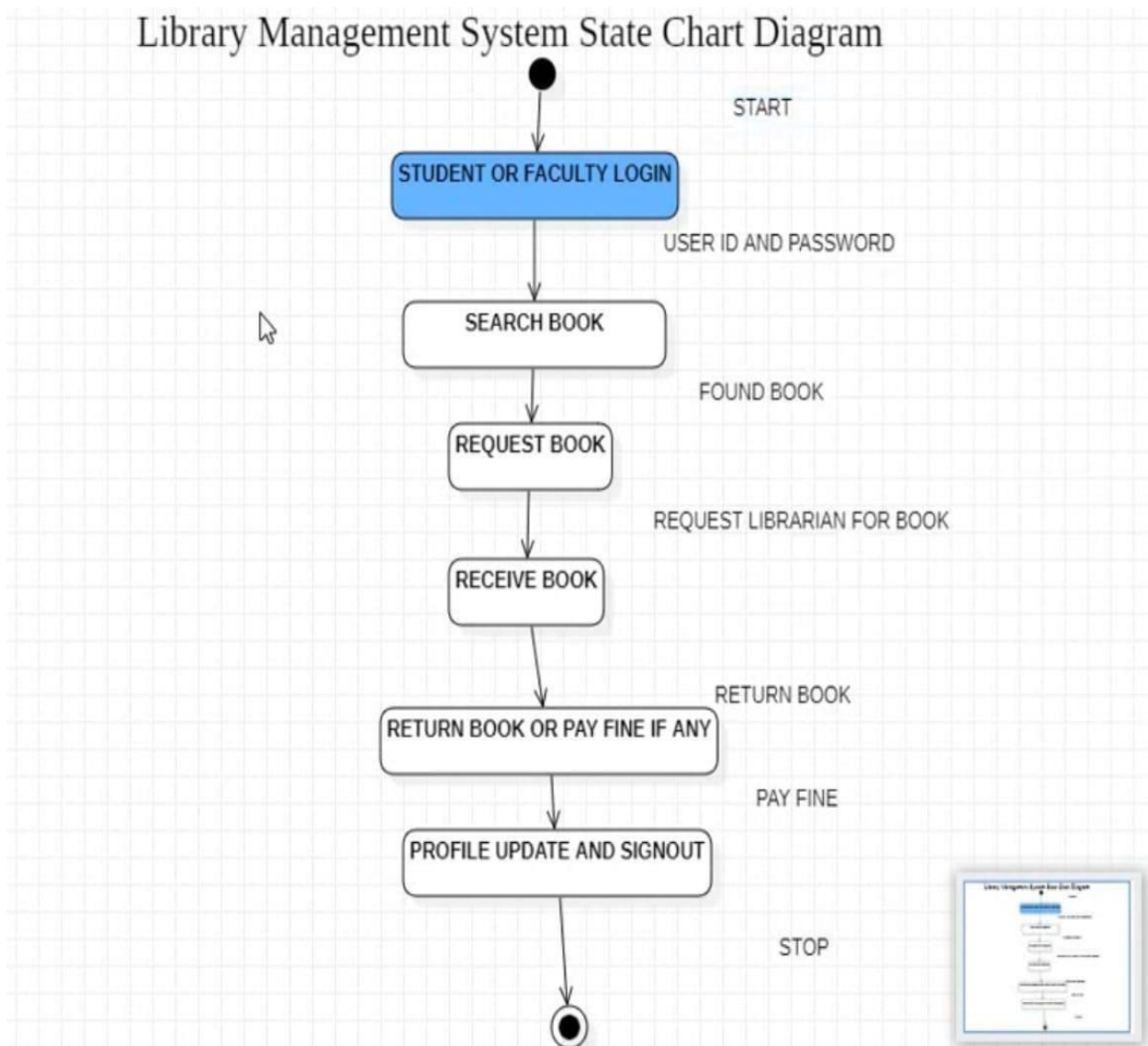
1C) SEQUENCE DIAGRAM



1D) STATE ACTIVITY DIAGRAM:

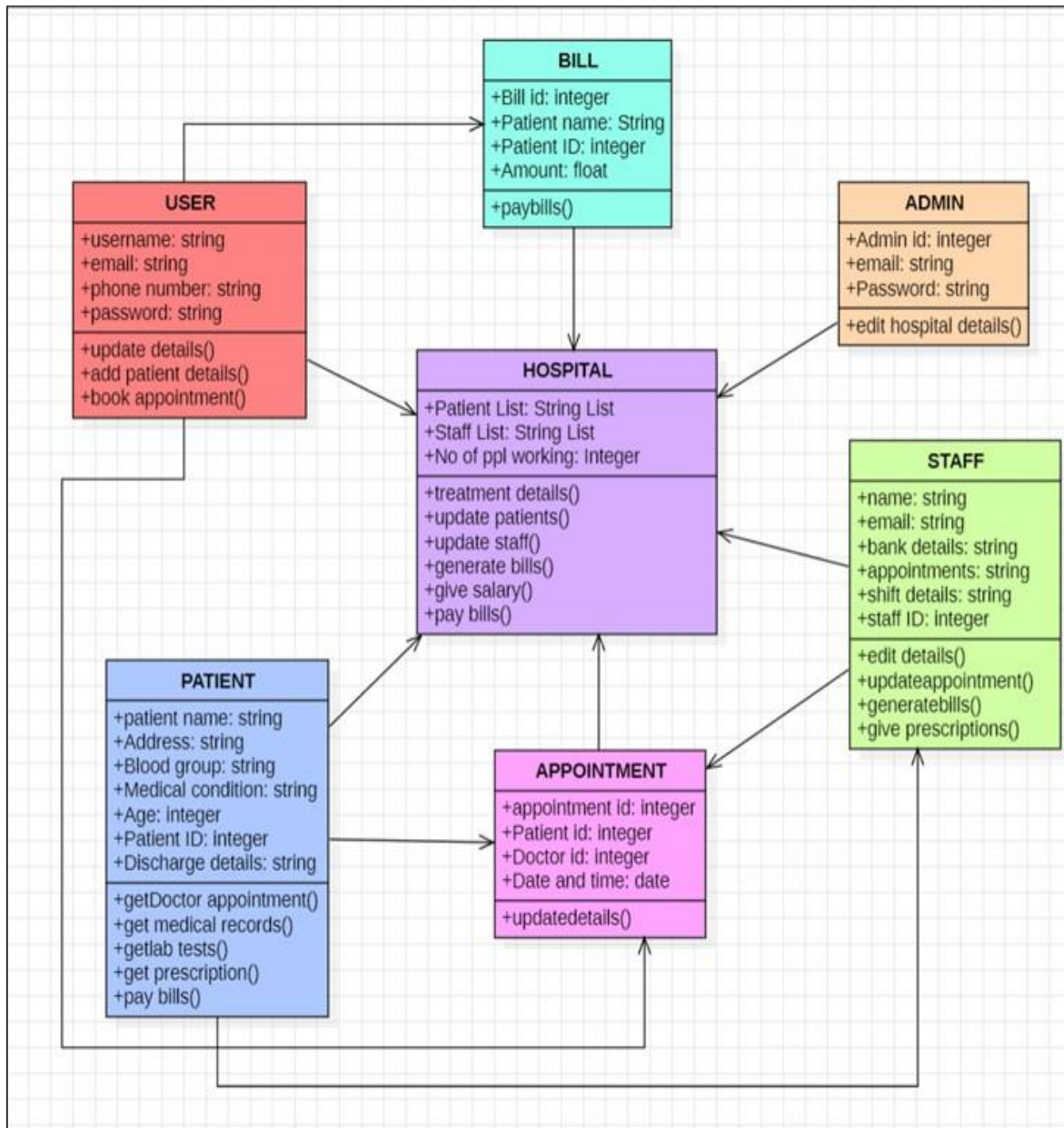


1E) STATE CHART DIAGRAM:

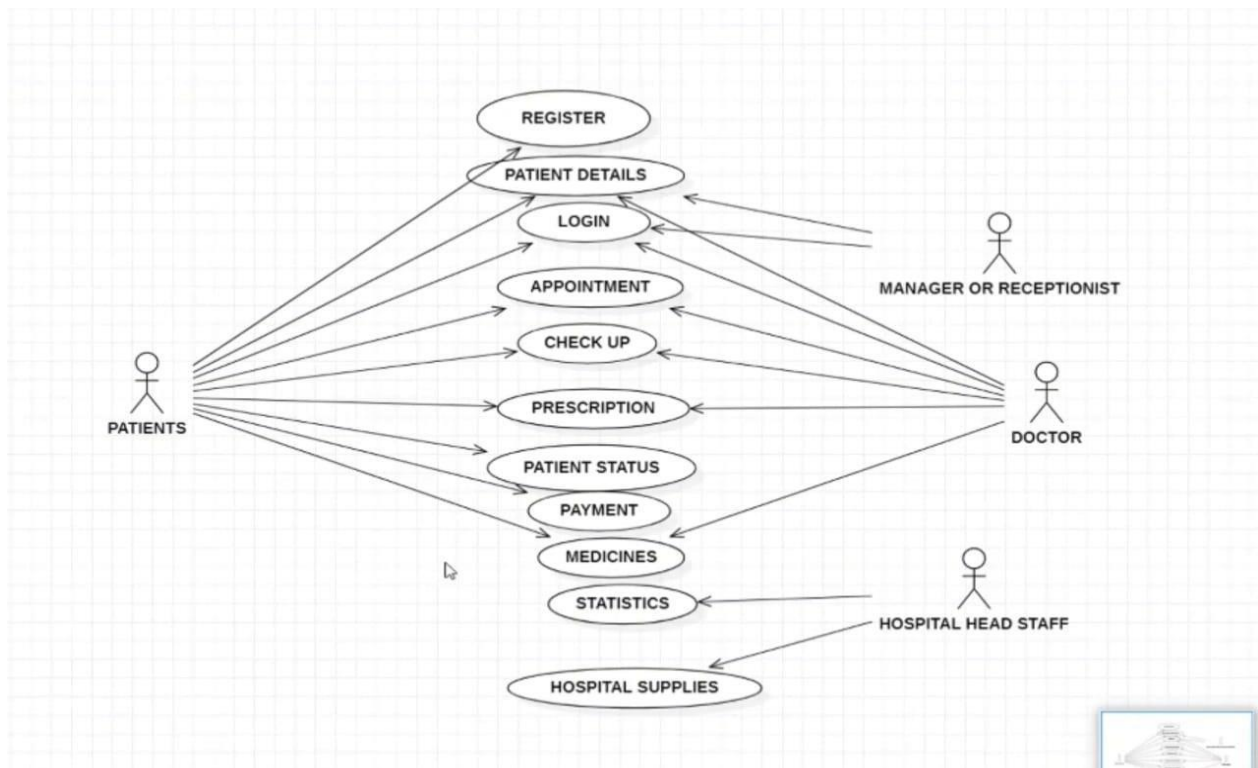


2. ONLINE HOSPITAL MANAGEMENT SYSTEM

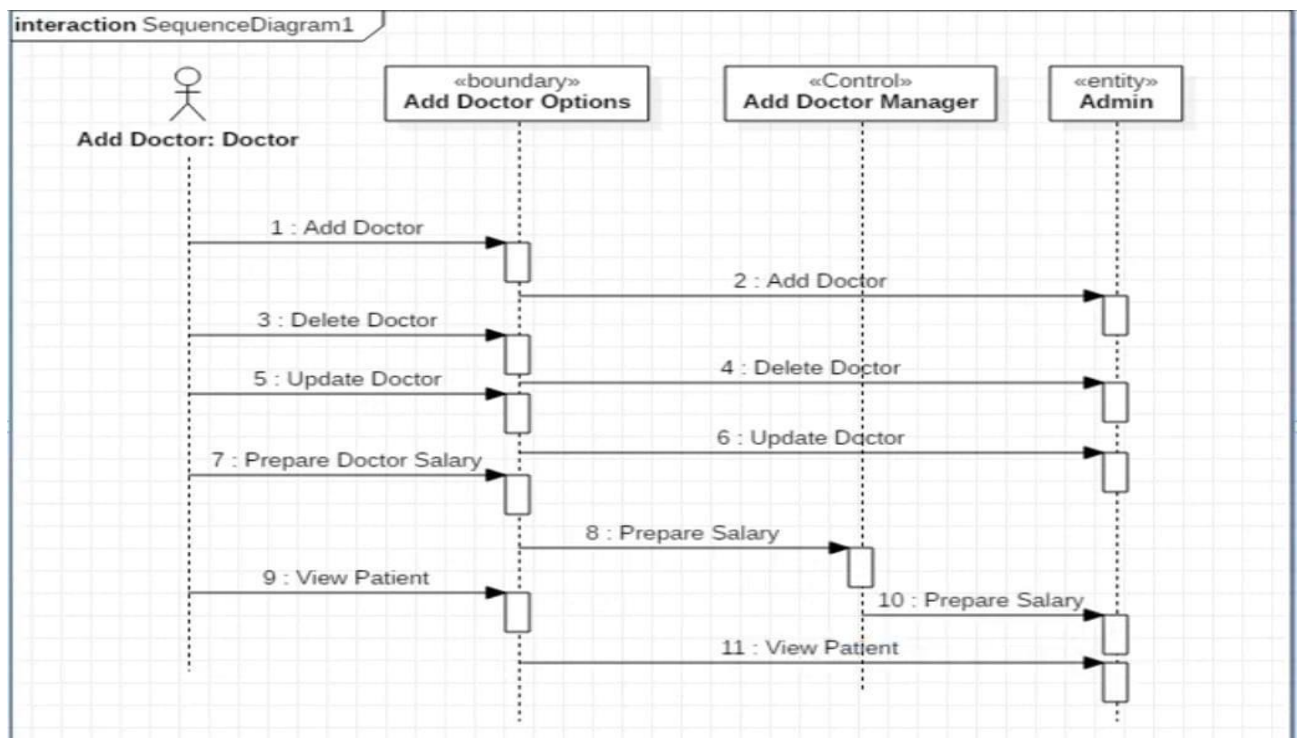
2A) CLASS DIAGRAM:



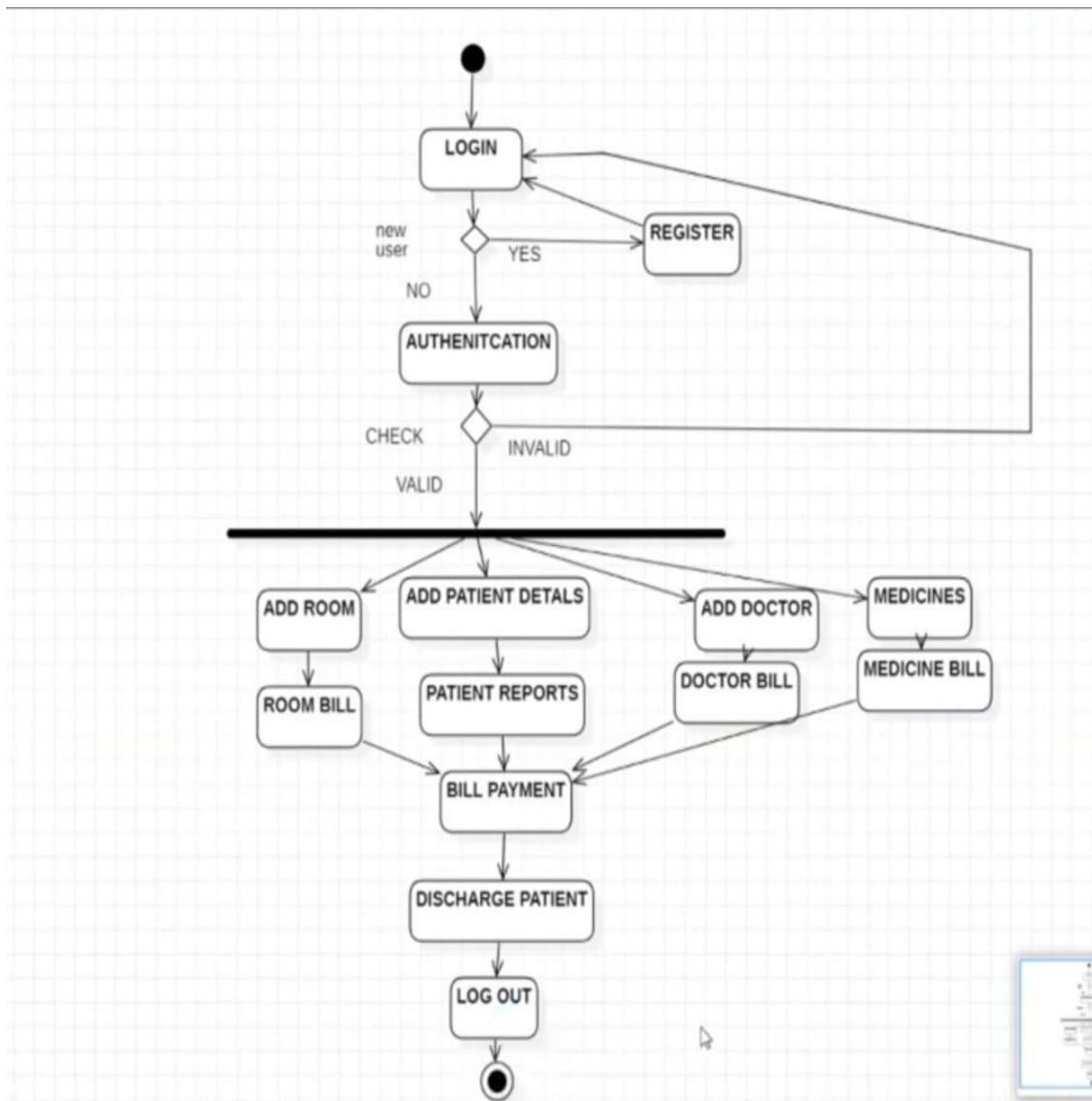
2B) USE CASE DIAGRAM:



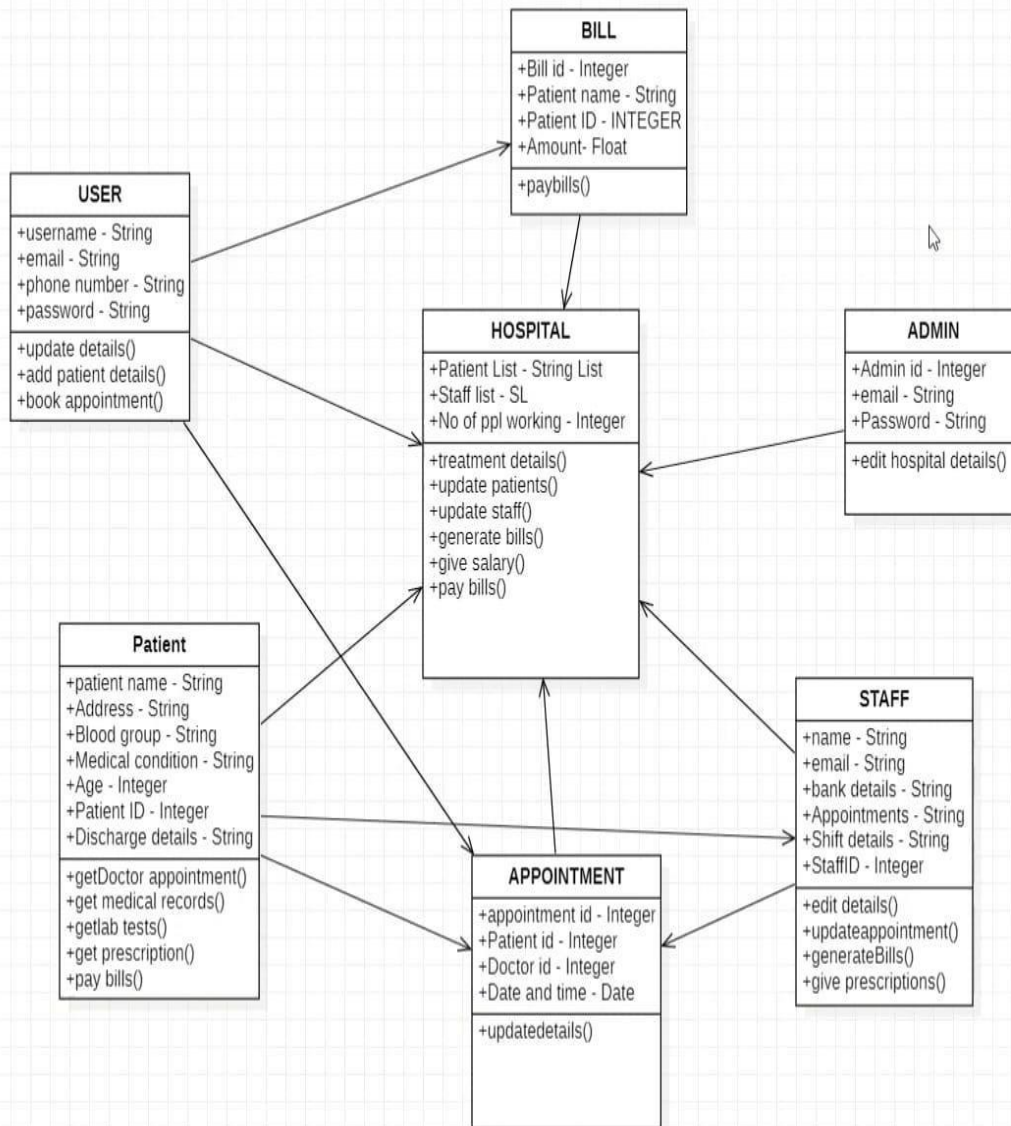
1C) SEQUENCE DIAGRAM



1D) ACTIVITY DIAGRAM:



1E) OBJECT DIAGRAM:



3. JAVA BASIC PROGRAMS

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: $\text{Area} = 1/2 \times \text{base} \times \text{height}$
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:
$$\text{average} = \text{num1} + \text{num2} + \text{num3} / 3$$
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:

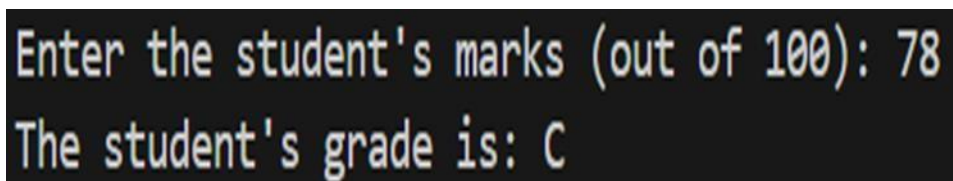
1. Start.
2. Accept the marks (out of 100) from the user.
3. Use if-else conditions:
 - 90-100 → Grade A
 - 80-89 → Grade B
 - 70-79 → Grade C
 - 60-69 → Grade D
 - 50-59 → Grade E
 - Below 50 → Grade F
4. Display the grade.
5. End.

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 :

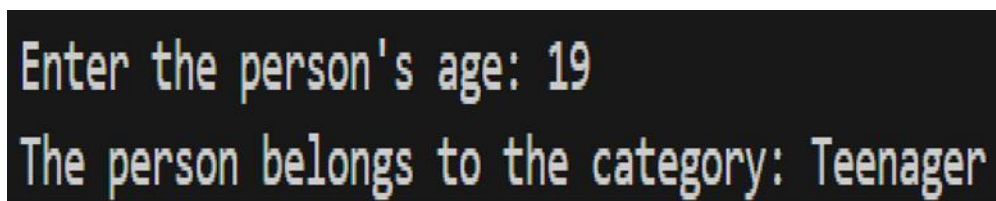
1. Start.
2. Accept age as input.
3. Use if-else conditions to categorize:
 - 0-12 → Child
 - 13-19 → Teenager
 - 20-59 → Adult
 - 60+ → Senior Citizen
4. Display the category.
5. End.

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:A screenshot of a terminal window with a black background and light blue text. The first line shows the prompt "Enter the person's age: 19" where "19" is the user input. The second line shows the output "The person belongs to the category: Teenager".

```
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:
 - 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:
 - If they are equal, print "originalNumber is a palindrome."
 - 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();
    }
}

```

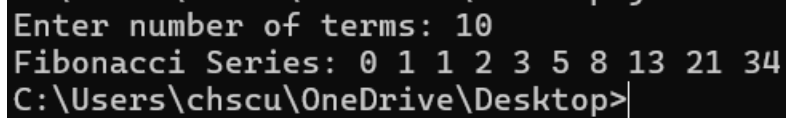


```
int first = 0, second = 1, next;

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>
```

4. INHERITANCE

4a) Single inheritance

- 1) To generate pay slips for calculating salary details of the employess working along with it displaying their details .

AIM : To develop a Java program that calculates the salary of employees based on the number of days worked and displays their details.

ALGORITHM:

1. Start.
2. Create a class salary with a method cal(int n) to calculate the salary.
3. Create a class details that extends salary and defines the method display(...) to show employee details.
4. In the main method:
 - Accept employee name, ID, designation, and number of days worked.
 - Compute the salary using cal(n).
 - Display all details using the display() method.
5. End

CODE :

```
import java.util.Scanner;

class salary{
int n; double sal;
double cal(int n){
sal = n*2500;
return sal;
}}

class details extends salary{
String na ; int eid; String d; int n;
void display(String na,int eid,String d,int n,double sal){
System.out.println("Name"+na);
System.out.println("Employee id:"+eid);
System.out.println("Designation:"+d);
```

```
System.out.println("Number of days worked:"+n);
System.out.println("SALARY:"+sal);
}}
public class j3{
public static void main(String[] args)
{
Scanner obj = new Scanner(System.in);
System.out.println("Enter your name");
String na = obj.nextLine();
System.out.println("Enter employee id");
int eid = obj.nextInt();
System.out.println("Enter designation");
obj.nextLine();
String d = obj.nextLine();
System.out.println("Enter Number of days worked:");
int n = obj.nextInt();
details de = new details();
double sal = de.cal(n);
de.display(na,eid,d,n,sal);
obj.close();
}
}
```

OUTPUT:

```

C:\Users\chscu\OneDrive\Desktop\JAVA LAB (1)>javac j3.java
C:\Users\chscu\OneDrive\Desktop\JAVA LAB (1)>java j3
Enter your name
Hasini Balabommu
Enter employee id
1006
Enter designation
Manger
Enter Number of days worked:
25
NameHasini Balabommu
Employee id:1006
Designation:Manger
Number of days worked:25
SALARY:62500.0

```

- 2) To Write a Java program that demonstrates the use of classes and objects. This program defines a class called Person with attributes like name, age, and a method displayInfo() to display the information of a person.

AIM: To create a Java program that demonstrates the use of classes and objects using a Person class.

ALGORITHM :

1. Start.
2. Define a class person with a method displayinfo(String name, int age).
3. In the main method:
 - Accept user input for name and age.
 - Create an object of person class.
 - Call displayinfo() to display user details.
4. End.

CODE :

```

import java.util.Scanner; class
person{

```

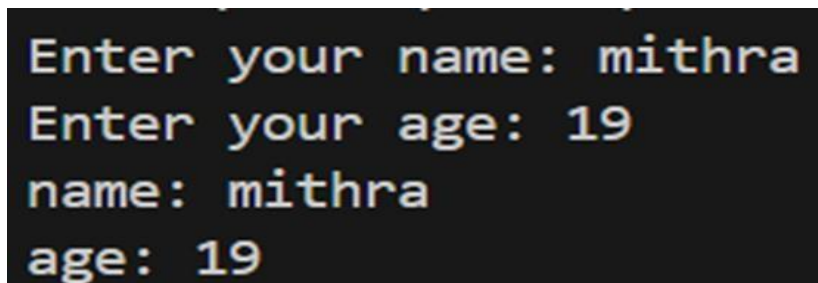
```

        void displayinfo (String name,int
        age){    System.out.println("name: "+name);
        System.out.println("age: "+age);
        }
    }
}

public class human{
    public static void main(String []args){ Scanner
        scan=new Scanner(System.in);
        System.out.print("Enter your name: "); String
        name=scan.nextLine(); System.out.print("Enter
        your age: "); int age=scan.nextInt();
        person p=new person();
        p.displayinfo(name,age);
    }
}

```

OUTPUT:



```

Enter your name: mithra
Enter your age: 19
name: mithra
age: 19

```

4b) Multilevel Inheritance

Q no 1) AIM: To display the persons details using multilevel inheritance

ALGORITHM:

1. Start
2. Define Person class
 - Declare attributes: name (String) and age (int).

- Define method display(String name, int age) to print employee name and age.
- 3. Define Employee class (Inherits from Person)
 - Declare attribute: sal (double).
 - Define method display(String name, int age, double sal) to print employee name, age, and salary.
- 4. Define Manager class (Inherits from Employee)
 - Declare attribute: dept (String).
 - Define method display(String name, int age, double sal, String dept) to print employee name, age, salary, and department.
- 5. Define m1 class with main method
 - Create a Scanner object to read user input.
 - Prompt the user to enter:
 - Name (String)
 - Age (int)
 - Salary (double)
 - Department (String)
 - Store the input values.
- 6. Create an object of Person class
 - Call display(name, age) method to print name and age.
- 7. Create an object of Manager class
 - Call display(name, age, sal, dept) method to print name, age, salary, and department.
- 8. End

CODE:

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

```
String name;
int age;
void display(String name,int age){
    System.out.println("Employee name:"+name);
    System.out.println("AGE:"+age);
}
}
class Employee extends Person{
    double sal;
    void display(String name,int age,double sal){
        System.out.println("Employee name:"+name);
        System.out.println("AGE:"+age);
        System.out.println("Salary:"+sal);
    }
}
class Manager extends Employee{
    String dept;
    void display(String name,int age,double sal,String dept){
        System.out.println("Employee name:"+name);
        System.out.println("AGE:"+age);
        System.out.println("Salary:"+sal);
        System.out.println("Department:"+dept);
    }
}
public class ml1{
    public static void main(String[] args){
        Scanner obj = new Scanner(System.in);
```

```
System.out.println("Enter your name");
String name = obj.nextLine();
System.out.println("Enter your age");
int age = obj.nextInt();
System.out.println("Enter your Salary");
double sal = obj.nextDouble();
System.out.println("Enter your department");
obj.nextLine();
String dept = obj.nextLine();
Person p = new Person();
p.display(name,age);
Manager m = new Manager();
m.display(name,age,sal,dept);
}
}
```

OUTPUT:


```
C:\Users\ch.sc.u4cse24010\Desktop>javac ml1.java

C:\Users\ch.sc.u4cse24010\Desktop>java ml1
Enter your name
Hasini
Enter your age
18
Enter your Salary
120000
Enter your department
AI
Employee name:Hasini
AGE:18
Employee name:Hasini
AGE:18
Salary:120000.0
Department:AI
```

Q no 2) To display various bank details such as balance rate of interest using methods such as withdraw , deposit etc... using multiple inheritance

AIM:

ALGORITHM:

1. Start
2. Define Bankaccount class
3. Declare attributes:
 - o ano (int) → Account number
 - o b (double) → Balance
 - o roi (float) → Rate of interest
 - o amt (double) → Deposit amount
4. Define method deposit(double b, double amt)
5. Add amt to b (balance).
6. Print the updated balance.
7. Define Savingsaccount class (Inherits from Bankaccount)

- Declare attribute: w (double) → Withdrawal amount
- Define method withdraw(double b, double w)
 8. Check if $w > b$ (withdrawal amount is greater than balance)
 - 1) If true, print "balance not sufficient".
 - 2) Else, subtract w from b and print the updated balance.
 9. Define fixeddeposit class (Inherits from Savingsaccount)
- Define method getinterest(float r, double b)
 - Calculate interest as $r = (\text{float})(b * 0.77)$.
 - Return the interest amount.
- 10. Define ml2 class with main method
- Create a Scanner object to read user input.
- Prompt the user to enter:
 - Balance in account (b).
 - Amount to deposit (amt).
 - Amount to withdraw (w).
- Store the input values.
 11. Create an object of Bankaccount
- Call deposit(b, amt) to add deposit amount to balance and print it.
 12. Create an object of Savingsaccount
- Call withdraw(b, w) to perform withdrawal and print the new **balance**.
- 13. End

CODE:

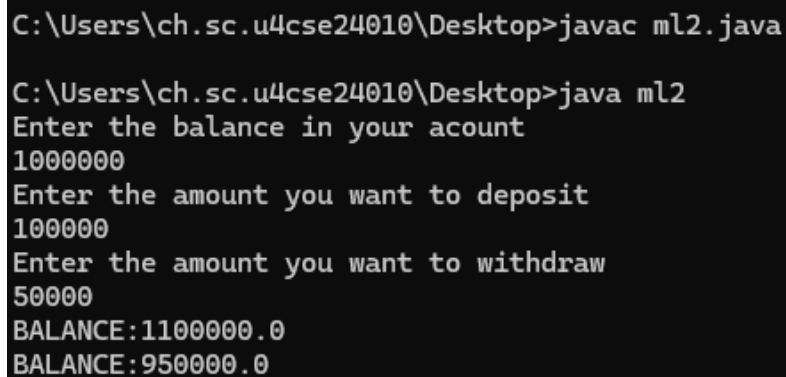
```
import java.util.Scanner;

class Bankaccount{
    int ano;
```

```
double b;
float roi;
double amt;
void deposit(double b,double amt){
    b=b+amt;
    System.out.println("BALANCE:"+b);
}
}
class Savingsaccount extends Bankaccount{
    double w;
    void withdraw(double b,double w){
        if(w>b){
            System.out.println("balance not sufficient");
        }
        else{
            b=b-w;
            System.out.println("BALANCE:"+b);
        }
    }
}
class fixeddeposit extends Savingsaccount{
    float getinterest(float r,double b){
        r=(float)(b*0.77);
        return r;
    }
}
public class ml2{
```

```
public static void main(String[] args){
Scanner obj = new Scanner(System.in);
System.out.println("Enter the balance in your account");
double b = obj.nextDouble();
System.out.println("Enter the amount you want to deposit");
double amt = obj.nextDouble();
System.out.println("Enter the amount you want to withdraw");
double w = obj.nextDouble();
Bankaccount ba = new Bankaccount();
ba.deposit(b,amt);
Savingsaccount s = new Savingsaccount();
s.withdraw(b,w);
}
}
```

OUTPUT:



```
C:\Users\ch.sc.u4cse24010\Desktop>javac ml2.java
C:\Users\ch.sc.u4cse24010\Desktop>java ml2
Enter the balance in your account
1000000
Enter the amount you want to deposit
100000
Enter the amount you want to withdraw
50000
BALANCE:1100000.0
BALANCE:950000.0
```

4c) Hierarchical Inheritance

Q no 1)To create a base class shape and subclasses rectangle and circle and applying the methods such as calculating area and perimeter

AIM : To create a base class Shape and subclasses Rectangle and Circle, implementing methods to calculate area and perimeter.

ALGORITHM :

1. Start.
2. Create a class Shape with default methods for calculateArea() and calculatePerimeter().
3. Define a subclass Rectangle with length and width attributes:
 - Override methods to compute area and perimeter.
4. Define a subclass Circle with radius:
 - Override methods for area and perimeter.
5. In main:
 - Take user input for dimensions.
 - Create objects of Rectangle and Circle.
 - Display computed area and perimeter.
6. End.

CODE :

```
import java.util.Scanner;
class Shape {
double calculateArea() {
return 0;
}
double calculatePerimeter() {
```

```
    return 0; } }  
class Rectangle extends Shape {  
    double length, width;  
    Rectangle(double l, double w) {  
        length = l;  
        width = w;  
    }  
  
    double calculateArea() {  
        return length * width;  
    }  
    double calculatePerimeter() {  
        return 2 * (length + width  
    } }  
class Circle extends Shape {  
    double radius; double pi = 3.14159;  
    Circle(double r) {  
        radius = r;  
    }  
  
    double calculateArea() {  
        return pi * radius * radius;  
    }  
    double calculatePerimeter() {  
        2 * pi * radius;  
    } }  
class j4 {  
    public static void main(String[] args) {  
        Scanner obj = new Scanner(System.in);  
        System.out.println("Enter the length");  
        double l = obj.nextDouble();  
        System.out.println("Enter breadth");  
        double w = obj.nextDouble();  
        System.out.println("Enter radius");  
        double r = obj.nextDouble();
```

```
Shape rect = new Rectangle(l,w);
System.out.println("Rectangle Area: " + rect.calculateArea());
System.out.println("Rectangle Perimeter: " + rect.calculatePerimeter());
Shape circ = new Circle(r); System.out.println("\nCircle Area: " +
circ.calculateArea()); System.out.println("Circle Perimeter: " +
circ.calculatePerimeter());
}
}
```

OUTPUT:

```
C:\Users\chscu\OneDrive\Desktop\JAVA LAB (1)>java j4
Enter the length
12
Enter breadth
10
Enter radius
7
Rectangle Area: 120.0
Rectangle Perimeter: 44.0

Circle Area: 153.93791
Circle Perimeter: 43.98226

C:\Users\chscu\OneDrive\Desktop\JAVA LAB (1)>|
```

4d) Multiple inheritance

Q no 1)

AIM: Define two interfaces Flyable and Swimmable. Create a class Bird implementing Flyable and a class Duck implementing both Flyable and Swimmable.

ALGORITHM:

1. Define the Interfaces:
 - Create an interface Flyable with a method fly().
 - Create an interface Swimmable with a method swim().
2. Create the Bird class:
 - Implement the Flyable interface.
 - Define the fly() method to print "Bird is flying."
3. Create the Duck class:
 - Implement both Flyable and Swimmable interfaces.
 - Define the fly() method to print "Duck is flying."
 - Define the swim() method to print "Duck is swimming."
4. Create the Main Class (MultipleInheritanceExample):
 - Instantiate an object of Bird and call the fly() method.
 - Instantiate an object of Duck and call both fly() and swim() methods.
5. End the program.

CODE:

```
interface Flyable {  
    void fly();  
}  
  
interface Swimmable {  
    void swim();  
}  
  
class Bird implements Flyable {  
    public void fly() {  
        System.out.println("Bird is flying.");  
    }  
}
```



```
    }  
}  
class Duck implements Flyable, Swimmable {  
    public void fly() {  
        System.out.println("Duck is flying.");  
    }  
  
    public void swim() {  
        System.out.println("Duck is swimming.");  
    }  
}  
public class m1 {  
    public static void main(String[] args) {  
        Bird bird = new Bird();  
        bird.fly();  
  
        Duck duck = new Duck();  
        duck.fly();  
        duck.swim();  
    }  
}
```

OUTPUT:

```
C:\Users\chscu\OneDrive\Desktop\LAB>java m1  
Bird is flying.  
Duck is flying.  
Duck is swimming.
```

Qno 2)

AIM: Implement a scenario where you have a class Employee, Manager and Director, showcasing multiple inheritance with interfaces for roles like Lead, Supervise, etc

ALGORITHM:

1. Define the Employee class:
 - Create a constructor to initialize name.
 - Define a work() method to print "<name> is working."
2. Define the Lead interface:
 - Declare the method leadTeam().
3. Define the Supervise interface:
 - Declare the method superviseWork().
4. Create the Manager class:
 - Inherit from Employee.
 - Implement the Lead interface.
 - Define the leadTeam() method to print "<name> is leading the team."
5. Create the Director class:
 - Inherit from Manager.
 - Implement the Supervise interface.
 - Define the superviseWork() method to print "<name> is supervising the work."
6. Create the Main Class (HybridInheritanceExample):
 - Instantiate an object of Director with a name.
 - Call the work(), leadTeam(), and superviseWork() methods.
7. End the program.

CODE:

```
// Base Employee class
class Employee {
    String name;
    Employee(String name) {
        this.name = name;
    }
    void work() {
        System.out.println(name + " is working.");
    }
}
```

```
    }  
    }  
interface Lead {  
    void leadTeam();  
}  
interface Supervise {  
    void superviseWork();  
}  
  
class Manager extends Employee implements Lead {  
    Manager(String name) {  
        super(name);  
    }  
  
    public void leadTeam() {  
        System.out.println(name + " is leading the team.");  
    }  
}  
  
// Director class inheriting Manager and implementing Supervise  
class Director extends Manager implements Supervise {  
    Director(String name) {  
        super(name);  
    }  
  
    public void superviseWork() {  
        System.out.println(name + " is supervising the work.");  
    }  
}  
  
// Main class to test the implementation  
public class HybridInheritanceExample {
```

```

public static void main(String[] args) {
    Director director = new Director("Alice");

    director.work();          // Inherited from Employee
    director.leadTeam();      // Implemented from Lead
    director.superviseWork(); // Implemented from Supervise
}
}

```

OUTPUT:

```

C:\Users\chscu\OneDrive\Desktop\LAB>java m1
Alice is working.
Alice is leading the team.
Alice is supervising the work.

```

5.POLYMORPHYISM

5a) Overriding

Q no 1)

AIM: to override the salary methods for employees based on their position

ALGORITHM:

8. Start
9. Define a base class Employee with:
 - A double variable sal to store salary.
 - An int variable n for the number of days worked.
 - A method calculateSalary(double sal, int n) (to be overridden).
10. Create a subclass programmer that:
 - Overrides calculateSalary() to calculate salary as $sal = n * 2500$.
 - Prints "Programmers Salary: <calculated_salary>".

11. Create another subclass manager that:

- Overrides calculateSalary() to calculate salary as $sal = n * 1500$.
- Prints "Manager Salary: <calculated_salary>".

12. In the main method of class r1:

- Create a Scanner object to take user input.
- Create objects of programmer and manager.
- Ask the user to enter the number of days worked.
- Read the input value n.
- Initialize sal to 0.
- Call calculateSalary(sal, n) for both programmer and manager.

13. Display the calculated salaries.

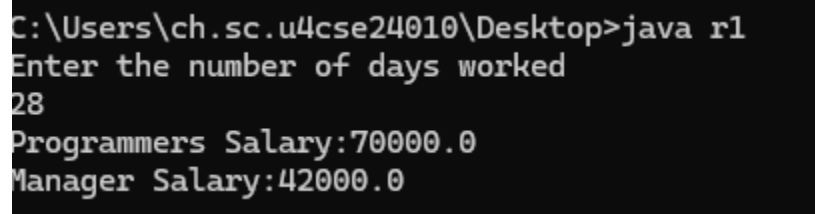
14. End.

CODE:

```
import java.util.Scanner;
class Employee {
double sal;
int n;
void calculateSalary(double sal,int n){
}
}
class programmer extends Employee{
@Override
void calculateSalary(double sal,int n){
sal = n*2500;
System.out.println("Programmers Salary:"+sal);
}
}
class manager extends Employee{
@Override
void calculateSalary(double sal,int n){
sal=n*1500;
```

```
System.out.println("Manager Salary:"+sal);
}
}
public class r1{
public static void main(String[] args){
Scanner obj = new Scanner(System.in);
programmer p = new programmer();
manager m = new manager();
System.out.println("Enter the number of days worked");
int n = obj.nextInt();
double sal = 0;
p.calculateSalary(sal,n);
m.calculateSalary(sal,n);
}
}
```

OUTPUT:



```
C:\Users\ch.sc.u4cse24010\Desktop>java r1
Enter the number of days worked
28
Programmers Salary:70000.0
Manager Salary:42000.0
```

Q no 2)

AIM: to override to display the number of players in each game

ALGORITHM:

15. Start
16. Define a base class games with a method show()
17. Create three subclasses:
 - cricket

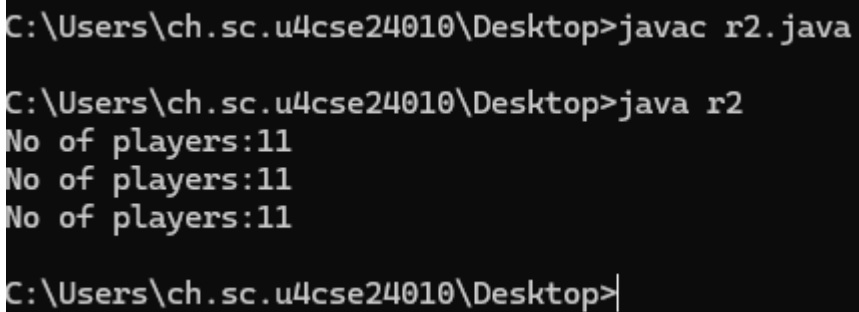
- hockey
- football

18. Override the show() method in each subclass to print "No of players: 11".
19. Define a main method inside the r2 class.
20. Create objects of cricket, hockey, and football classes.
21. Call the show() method for each object.
22. Print the number of players for each sport.
23. End

CODE:

```
import java.util.Scanner;
class games{
void show(){
}
}
class cricket extends games{
@Override
void show(){
System.out.println("No of players:11");
}
}
class hockey extends games{
@Override
void show(){
System.out.println("No of players:11");
}
}
class football extends games{
@Override
void show(){
System.out.println("No of players:11");
```

```
}  
}  
public class r2{  
    public static void main(String[] args){  
        cricket c = new cricket();  
        hockey h = new hockey();  
        football f = new football();  
        c.show();  
        h.show();  
        f.show();  
    }  
}
```

OUTPUT:

```
C:\Users\ch.sc.u4cse24010\Desktop>javac r2.java  
  
C:\Users\ch.sc.u4cse24010\Desktop>java r2  
No of players:11  
No of players:11  
No of players:11  
  
C:\Users\ch.sc.u4cse24010\Desktop>|
```

Q no 3)**AIM:****ALGORITHM:****CODE:**

```
import java.util.Scanner;  
class vehicle{  
    void speedup(int s){  
    }  
}
```



```
class car extends vehicle{
@Override
void speedup(int s){
s=s+20;
System.out.println("Car Speed:"+s);
}
}
class bike extends vehicle{
@Override
void speedup(int s){
s=s+10;
System.out.println("bike speed"+s);
}
}
public class r3{
public static void main(String[] args){
Scanner obj = new Scanner(System.in);
System.out.println("Enter your current speed");
int s = obj.nextInt();
car c = new car();
bike b = new bike();
c.speedup(s);
b.speedup(s);
}
}
```

OUTPUT:

```
C:\Users\ch.sc.u4cse24010\Desktop>javac r3.java

C:\Users\ch.sc.u4cse24010\Desktop>java r3
Enter your current speed
40
Car Speed:60
bike speed50
```

Q no 4)

AIM: To display the interest rate of an bank on the request of user

ALGORITHM:

Step 1: Start the program.

Step 2: Import the Scanner class from java.util to read user input.

Step 3: Define a base class Bank with a method getInterestrates() that returns 0.0.

Step 4: Create three subclasses SBI, ICICI, and RBI that extend Bank and override getInterestrates() to return their respective interest rates:

- SBI → 8.77%
- ICICI → 8.22%
- RBI → 8.00%

Step 5: In the main method:

- Create a Scanner object to take user input.
- Prompt the user to enter a bank name (SBI, ICICI, RBI).
- Read the bank name, convert it to uppercase to avoid case sensitivity.

Step 6: Use a switch-case to check the entered bank name:

- If the bank name is "SBI", create an instance of SBI.
- If the bank name is "ICICI", create an instance of ICICI.
- If the bank name is "RBI", create an instance of RBI.
- If the bank name is invalid, display "Invalid bank name" and exit the program.

Step 7: Call the `getInterestrates()` method on the selected bank object and display the interest rate.

Step 8: Close the Scanner object to free resources.

Step 9: End the program.

CODE:

```
import java.util.Scanner;
class Bank{
double getInterestrates(){
return 0;
}
}
class SBI extends Bank{
@Override
double getInterestrates()
{
return 8.77;
}
}
class ICICI extends Bank{
@Override
double getInterestrates(){
return 8.22;
}
}
class RBI extends Bank{
@Override
double getInterestrates(){
return 8.0;
}
}
```

```
public class m1{
    public static void main(String[] args){
        Scanner obj = new Scanner(System.in);
        System.out.println("Enter your bank name");
        String bankname = obj.nextLine().trim().toUpperCase();
        Bank bank;
        switch(bankname){
            case "SBI":
                bank = new SBI();
                break;
            case "ICICI":
                bank = new ICICI();
                break;
            case "RBI":
                bank = new RBI();
                break;
            default:
                System.out.println("Invalid bank name");
                obj.close();
                return;
        }
        System.out.println("Interest Rate is:"+bank.getInterestrates()+"%");
        obj.close();
    }
}
```

OUTPUT:

```
C:\Users\chscu\OneDrive\Desktop\LAB>java m1
Enter your bank name
sbi
Interest Rate is:8.77%
```

5b) Overloading

Q no1)

AIM: An Ecommerce application for payment

ALGORITHM:

Step 1: Start the program.

Step 2: Import the Scanner class from java.util to read user input.

Step 3: Define the Payment class with multiple makePayment() methods (Method Overloading):

- Cash Payment: Accepts a double amount and prints a cash payment message.
- Credit Card Payment: Accepts a String cardNumber and double amount, masks the card number, and prints a payment confirmation.
- UPI Payment: Accepts a String upid, double amount, and String remarks, then prints a payment confirmation.
- UPI QR Payment: Accepts a double amount and String qrCodeData, then prints a payment confirmation.

Step 4: In the main method:

- Create a Scanner object to take user input.
- Create a Payment object to process payments.
- Display a menu for selecting the payment method:

markdown

CopyEdit

1. Cash
2. Credit Card
3. UPI

4. UPI via QR Code

- Prompt the user to enter an option (1-4).

Step 5: Use a switch-case to process the selected payment method:

- Case 1 (Cash Payment):
 - Ask for the amount.
 - Call `makePayment(amount)`.
- Case 2 (Credit Card Payment):
 - Consume the newline left by `nextInt()`.
 - Ask for the card number.
 - Ask for the amount.
 - Call `makePayment(cardNumber, amount)`.
- Case 3 (UPI Payment):
 - Consume the newline left by `nextInt()`.
 - Ask for the UPI ID.
 - Ask for the amount.
 - Consume newline after `nextDouble()`.
 - Ask for remarks.
 - Call `makePayment(upid, amount, remarks)`.
- Case 4 (UPI QR Payment):
 - Consume the newline left by `nextInt()`.
 - Ask for QR code data.
 - Ask for the amount.
 - Call `makePayment(amount, qrCodeData)`.
- Default Case:
 - Print "Invalid option!" if the user enters an invalid number.

Step 6: Close the Scanner object.

Step 7: End the program.

CODE:

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

```
class Payment {

    // Cash Payment
    public void makePayment(double amount) {
        System.out.println("Processing cash payment of ₹" + amount);
    }

    // Credit Card Payment
    public void makePayment(String cardNumber, double amount) {
        System.out.println("Processing credit card payment...");
        System.out.println("Card Number: **** *  **** " +
cardNumber.substring(cardNumber.length() - 4));
        System.out.println("Amount: ₹" + amount);
    }

    // UPI Payment
    public void makePayment(String upid, double amount, String remarks) {
        System.out.println("Processing UPI payment...");
        System.out.println("UPI ID: " + upid);
        System.out.println("Amount: ₹" + amount);
        System.out.println("Remarks: " + remarks);
    }

    // UPI QR Payment
    public void makePayment(double amount, String qrCodeData) {
        System.out.println("Processing UPI payment via QR scan...");
        System.out.println("QR Code: " + qrCodeData);
        System.out.println("Amount: ₹" + amount);
    }
}
```

```
}
```

```
public class I1 {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
        Payment payment = new Payment();  
  
        System.out.println("Choose payment method:");  
        System.out.println("1. Cash");  
        System.out.println("2. Credit Card");  
        System.out.println("3. UPI");  
        System.out.println("4. UPI via QR Code");  
        System.out.print("Enter option (1-4): ");  
        int option = scanner.nextInt();  
  
        switch (option) {  
            case 1:  
                System.out.print("Enter amount: ");  
                double cashAmount = scanner.nextDouble();  
                payment.makePayment(cashAmount);  
                break;  
  
            case 2:  
                scanner.nextLine(); // consume leftover newline  
                System.out.print("Enter card number: ");  
                String cardNumber = scanner.nextLine();  
                System.out.print("Enter amount: ₹");  
                double cardAmount = scanner.nextDouble();  
                payment.makePayment(cardNumber, cardAmount);  
                break;
```


case 3:

```
    scanner.nextLine(); // consume leftover newline
    System.out.print("Enter UPI ID: ");
    String upild = scanner.nextLine();
    System.out.print("Enter amount: ₹");
    double upiAmount = scanner.nextDouble();
    scanner.nextLine(); // consume newline
    System.out.print("Enter remarks: ");
    String remarks = scanner.nextLine();
    payment.makePayment(upild, upiAmount, remarks);
    break;
```

case 4:

```
    scanner.nextLine(); // consume leftover newline
    System.out.print("Enter QR code data: ");
    String qrCode = scanner.nextLine();
    System.out.print("Enter amount: ₹");
    double qrAmount = scanner.nextDouble();
    payment.makePayment(qrAmount, qrCode);
    break;
```

default:

```
    System.out.println("Invalid option!");
```

```
}
```

```
scanner.close();
```

```
}
```

```
}
```

OUTPUT:

```

C:\Users\ch.sc.u4cse24010\Desktop>javac l1.java

C:\Users\ch.sc.u4cse24010\Desktop>java l1
Choose payment method:
1. Cash
2. Credit Card
3. UPI
4. UPI via QR Code
Enter option (1-4): 3
Enter UPI ID: hasini1607#ekjhiw
Enter amount: ?200
Enter remarks: canteen
Processing UPI payment...
UPI ID: hasini1607#ekjhiw
Amount: ?200.0
Remarks: canteen

```

Q no 2)**AIM:** To build a area calculator**ALGORITHM:**

Step 1: Start the program.

Step 2: Import the Scanner class from java.util to read user input.

Step 3: Define the ShapeAreaCalculator class with method overloading for calculating the area of different shapes:

- Method 1 (Square): Accepts double side, calculates area = side × side, and prints the area.
- Method 2 (Rectangle): Accepts double length, double breadth, calculates area = length × breadth, and prints the area.
- Method 3 (Circle): Accepts double radius, boolean isCircle, checks if isCircle == true, calculates area = $\pi \times \text{radius}^2$, and prints the area.

Step 4: In the main method:

- Create a Scanner object for user input.
- Create a ShapeAreaCalculator object for performing area calculations.
- Display a menu for selecting a shape:

markdown

1. Square

2. Rectangle

3. Circle

- Prompt the user to enter an option (1-3).

Step 5: Use a switch-case to process the selected shape:

- Case 1 (Square):
 - Ask the user to enter the side length.
 - Call `calculateArea(side)`.
- Case 2 (Rectangle):
 - Ask the user to enter the length and breadth.
 - Call `calculateArea(length, breadth)`.
- Case 3 (Circle):
 - Ask the user to enter the radius.
 - Call `calculateArea(radius, true)`.
- Default Case:
 - Print "Invalid option!" if the user enters an invalid number.

Step 6: Close the Scanner object to free resources.

Step 7: End the program.

CODE:

```
import java.util.Scanner;
class ShapeAreaCalculator {
    // Area of square
    public void calculateArea(double side) {
        double area = side * side;
        System.out.println("Area of square: " + area + " sq. units");
    }

    // Area of rectangle
    public void calculateArea(double length, double breadth) {
        double area = length * breadth;
        System.out.println("Area of rectangle: " + area + " sq. units");
    }

    // Area of circle
```

```
public void calculateArea(double radius, boolean isCircle) {
    if (isCircle) {
        double area = Math.PI * radius * radius;
        System.out.println("Area of circle: " + area + " sq. units");
    }
}

public class I2 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        ShapeAreaCalculator calculator = new ShapeAreaCalculator();

        System.out.println("Choose shape to calculate area:");
        System.out.println("1. Square");
        System.out.println("2. Rectangle");
        System.out.println("3. Circle");
        System.out.print("Enter option (1-3): ");
        int option = scanner.nextInt();

        switch (option) {
            case 1:
                System.out.print("Enter side length of square: ");
                double side = scanner.nextDouble();
                calculator.calculateArea(side);
                break;
            case 2:
                System.out.print("Enter length of rectangle: ");
                double length = scanner.nextDouble();
                System.out.print("Enter breadth of rectangle: ");
                double breadth = scanner.nextDouble();
                calculator.calculateArea(length, breadth);
                break;
            case 3:
                System.out.print("Enter radius of circle: ");
```

```

        double radius = scanner.nextDouble();
        calculator.calculateArea(radius, true);
        break;

        default:
            System.out.println("Invalid option!");
    }
    scanner.close();
}
}

```

OUTPUT:

```

C:\Users\ch.sc.u4cse24010\Desktop>javac l2.java

C:\Users\ch.sc.u4cse24010\Desktop>java l2
Choose shape to calculate area:
1. Square
2. Rectangle
3. Circle
Enter option (1-3): 1
Enter side length of square: 20
Area of square: 400.0 sq. units

```

Q no 3)

AIM: Online shopping to calculate bill

ALGORITHM:

Step 1: Start

Step 2: Define the ShoppingCart class with three overloaded methods:

- Method 1: calculateBill(double price, int quantity) → Calculates total bill for a single item.
- Method 2: calculateBill(double[] prices) → Calculates total bill for multiple items.
- Method 3: calculateBill(double price, int quantity, double discount) → Calculates total bill after applying a discount.

Step 3: Implement calculateBill methods:

- For a single item:

- Multiply price by quantity.
 - Print total bill.
- For multiple items:
 - Initialize total to 0.
 - Loop through the array of prices, adding each price to total.
 - Print total bill.
- For an item with a discount:
 - Multiply price by quantity to get total.
 - Calculate discount amount using $(\text{total} * \text{discount} / 100)$.
 - Subtract discount amount from total.
 - Print final bill after discount.

Step 4: Define the ShoppingCart class with the main method:

- Create an instance of ShoppingCart.
- Call methods with different types of data:
 - Call calculateBill(price, quantity) for a single item.
 - Call calculateBill(prices[]) for multiple items.
 - Call calculateBill(price, quantity, discount) for an item with a discount.

Step 5: End

CODE:

```
class ShoppingCart {
    // Total bill for a single item
    public void calculateBill(double price, int quantity) {
        double total = price * quantity;
        System.out.println("Total bill: ₹" + total);
    }

    // Total bill for multiple items
    public void calculateBill(double[] prices) {
        double total = 0;
        for (double price : prices) {
            total += price;
        }
        System.out.println("Total bill for multiple items: ₹" + total);
    }
}
```

```

    }

    // Total bill with discount
    public void calculateBill(double price, int quantity, double discount) {
        double total = price * quantity;
        double discountAmount = total * (discount / 100);
        double finalAmount = total - discountAmount;
        System.out.println("Total bill after " + discount + "% discount: ₹" + finalAmount);
    }
}

public class ShoppingTest {
    public static void main(String[] args) {
        ShoppingCart cart = new ShoppingCart();

        cart.calculateBill(500, 2); // Single item
        cart.calculateBill(new double[]{200, 300, 150}); // Multiple items
        cart.calculateBill(1000, 2, 10); // Discounted bill
    }
}

```

OUTPUT:

```

C:\Users\chscu\OneDrive\Desktop\LAB>java m1
Total bill: ?1000.0
Total bill for multiple items: ?650.0
Total bill after 10.0% discount: ?1800.0

```

Q no 4)**AIM:** Food delivery time calculator**ALGORITHM:**

Step 1: Define the estimateTime method for distance-based delivery

1. Input: distance (double)
2. Compute time: $\text{time} = 10 + (2 * \text{distance})$

3. Print: "Estimated Delivery Time: " + time + " minutes"

Step 2: Define the estimateTime method for distance with traffic condition

1. Input: distance (double), heavyTraffic (boolean)
2. Compute time: $\text{time} = 10 + (2 * \text{distance})$
3. If heavyTraffic is true, add 5 minutes to time
4. Print: "Estimated Delivery Time (with traffic): " + time + " minutes"

Step 3: Define the estimateTime method for order type-based delivery

1. Input: distance (double), orderType (String)
2. Compute time: $\text{time} = 10 + (2 * \text{distance})$
3. If orderType is "Fast", reduce time by 5 minutes
4. Print: "Estimated Delivery Time (" + orderType + " Delivery): " + time + " minutes"

Step 4: Execute the DeliveryTest class (main method)

1. Create an instance of DeliveryEstimator
2. Call estimateTime(5) → Distance-based estimation
3. Call estimateTime(5, true) → Distance with heavy traffic estimation
4. Call estimateTime(5, "Fast") → Distance with fast delivery estimation

CODE:

```
class DeliveryEstimator {
    // Delivery time based on distance
    public void estimateTime(double distance) {
        double time = 10 + (2 * distance);
        System.out.println("Estimated Delivery Time: " + time + " minutes");
    }

    // Delivery time with traffic conditions
    public void estimateTime(double distance, boolean heavyTraffic) {
        double time = 10 + (2 * distance);
        if (heavyTraffic) {
            time += 5;
        }
        System.out.println("Estimated Delivery Time (with traffic): " + time + " minutes");
    }
}
```



```
// Delivery time based on order type (Fast or Normal)
public void estimateTime(double distance, String orderType) {
    double time = 10 + (2 * distance);
    if (orderType.equalsIgnoreCase("Fast")) {
        time -= 5; // 5 min faster for fast delivery
    }
    System.out.println("Estimated Delivery Time (" + orderType + " Delivery): " + time
+ " minutes");
}
}
```

```
public class DeliveryTest {
    public static void main(String[] args) {
        DeliveryEstimator delivery = new DeliveryEstimator();

        delivery.estimateTime(5); // Distance-based
        delivery.estimateTime(5, true); // Distance with traffic
        delivery.estimateTime(5, "Fast"); // Distance with fast delivery
    }
}
```

OUTPUT:

```
C:\Users\chscu\OneDrive\Desktop\LAB>java m1
Estimated Delivery Time: 20.0 minutes
Estimated Delivery Time (with traffic): 25.0 minutes
Estimated Delivery Time (Fast Delivery): 15.0 minutes
```

6) ABSTRACTION

6a) Using abstract classes

Q no 1)

AIM: To display online ticketing system

ALGORITHM:

1. Define an Abstract Class Ticket
 - Declare a double price variable to store the base ticket price.
 - Define an abstract method calculatePrice() to calculate the final price.
 - Implement a method printTicket() to print the ticket price after calculation.
2. Create a BusTicket Class (inherits Ticket)
 - Define a constructor that initializes price using super(price).
 - Implement the calculatePrice() method to return price * 1.05 (including a 5% tax).
3. Create a TrainTicket Class (inherits Ticket)
 - Define a constructor that initializes price using super(price).
 - Implement the calculatePrice() method to return price * 1.10 (including a 10% tax).
4. Create the Main Class (m1)
 - Instantiate BusTicket with a base price of 100.
 - Instantiate TrainTicket with a base price of 200.
 - Call printTicket() for both objects to display final prices.
5. Output the Calculated Ticket Prices
 - Display the calculated ticket prices for both bus and train tickets with respective tax rates applied.

CODE:

```
abstract class Ticket {  
    double price;  
    Ticket(double price) {  
        this.price = price;  
    }  
  
    // Abstract method for price calculation  
    abstract double calculatePrice();  
  
    void printTicket() {
```

```
        System.out.println("Ticket Price: $" + calculatePrice());
    }
}
```

// Bus Ticket class

```
class BusTicket extends Ticket {
    BusTicket(double price) {
        super(price);
    }
}
```

@Override

```
double calculatePrice() {
    return price * 1.05; // 5% tax
}
```

```
}
```

// Train Ticket class

```
class TrainTicket extends Ticket {
    TrainTicket(double price) {
        super(price);
    }
}
```

@Override

```
double calculatePrice() {
    return price * 1.10; // 10% tax
}
```

```
}
```

// Main class

```
public class TicketBookingSystem {
    public static void main(String[] args) {
        Ticket busTicket = new BusTicket(100);
        Ticket trainTicket = new TrainTicket(200);
    }
}
```

```
        System.out.println("Bus Ticket:");
        busTicket.printTicket();
    }
}
```

```

        System.out.println("\nTrain Ticket:");
        trainTicket.printTicket();
    }
}

```

OUTPUT:

```

C:\Users\chscu\OneDrive\Desktop\LAB>java m1
Bus Ticket:
Ticket Price: $105.0

Train Ticket:
Ticket Price: $220.000000000000003

```

Q no 2)

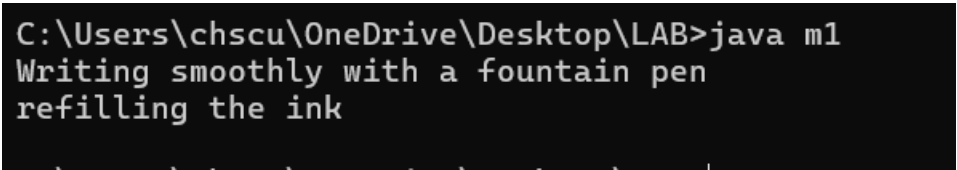
AIM: To understand the abstract methods using pen abstract class and methods like

ALGORITHM:

1. Define an Abstract Class pen
 - Declare two abstract methods:
 - write(): To define the writing functionality.
 - refill(): To define the refilling functionality.
2. Create a pics Class (inherits pen)
 - Implement the write() method:
 - Print "Writing smoothly with a fountain pen".
 - Implement the refill() method:
 - Print "refilling the ink".
3. Create the Main Class (m1)
 - Instantiate the pics class as an object p.
 - Call the write() method to simulate writing with the pen.
 - Call the refill() method to simulate refilling the ink.
4. Output the Results
 - Display messages indicating writing and refilling actions

CODE:

```
abstract class pen{
    abstract void write();
    abstract void refill();
}
class pics extends pen{
    void write(){
        System.out.println("Writing smoothly with a fountain pen");
    }
    void refill(){
        System.out.println("refilling the ink");
    }
}
public class m1{
    public static void main(String []args){
        pics p = new pics();
        p.write();
        p.refill();
    }
}
```

OUTPUT:

```
C:\Users\chscu\OneDrive\Desktop\LAB>java m1
Writing smoothly with a fountain pen
refilling the ink
```

Q no 3)**AIM: 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
```

Q no 4)

AIM: 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.
```

6b) Using interface

Q no 1)

AIM: shape area calculator

CODE:

// Shape.java

```
interface Shape {
    double getArea();
}
```

// Circle.java

```
class Circle implements Shape {
    double radius;

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

```
    public double getArea() {  
        return Math.PI * radius * radius;  
    }  
}
```

// Rectangle.java

```
class Rectangle implements Shape {  
    double length, width;
```

```
    Rectangle(double length, double width) {  
        this.length = length;  
        this.width = width;  
    }
```

```
    public double getArea() {  
        return length * width;  
    }  
}
```

// Main.java

```
public class Main {  
    public static void main(String[] args) {  
        Shape circle = new Circle(5);  
        Shape rectangle = new Rectangle(4, 6);  
  
        System.out.println("Circle Area: " + circle.getArea());  
        System.out.println("Rectangle Area: " + rectangle.getArea());  
    }  
}
```

OUTPUT:

```
C:\Users\chscu\OneDrive\Desktop>java e  
Circle Area: 78.53981633974483  
Rectangle Area: 24.0
```

Qno 2)**AIM:** applying the basic concepts through vehicle interface**CODE:**

```
interface Vehicle {
    void start();
    void stop();
    int getSpeed();
}

class Car implements Vehicle {
    public void start() {
        System.out.println("Car started.");
    }
    public void stop() {
        System.out.println("Car stopped.");
    }

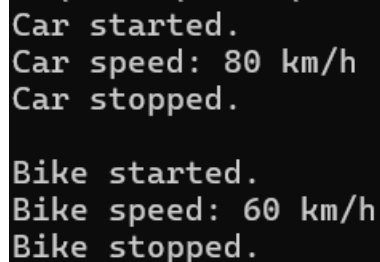
    public int getSpeed() {
        return 80;
    }
}

class Bike implements Vehicle {
    public void start() {
        System.out.println("Bike started.");
    }

    public void stop() {
        System.out.println("Bike stopped.");
    }

    public int getSpeed() {
        return 60;
    }
}
```

```
public class VehicleTest {  
    public static void main(String[] args) {  
        Vehicle car = new Car();  
        Vehicle bike = new Bike();  
  
        car.start();  
        System.out.println("Car speed: " + car.getSpeed() + " km/h");  
        car.stop();  
  
        System.out.println();  
  
        bike.start();  
        System.out.println("Bike speed: " + bike.getSpeed() + " km/h");  
        bike.stop();  
    }  
}
```

OUTPUT:A screenshot of a terminal window showing the output of the Java program. The text is as follows:
Car started.
Car speed: 80 km/h
Car stopped.

Bike started.
Bike speed: 60 km/h
Bike stopped.**Q no 3)****AIM:** Payment interface**CODE:**

```
interface Payment {  
    void pay(double amount);  
}  
  
class CreditCard implements Payment {  
    public void pay(double amount) {
```

```

        System.out.println("Paid ₹" + amount + " using Credit Card.");
    }
}

class DebitCard implements Payment {
    public void pay(double amount) {
        System.out.println("Paid ₹" + amount + " using Debit Card.");
    }
}

class UPIPayment implements Payment {
    public void pay(double amount) {
        System.out.println("Paid ₹" + amount + " using UPI.");
    }
}

public class PaymentTest {
    public static void main(String[] args) {
        Payment p1 = new CreditCard();
        Payment p2 = new DebitCard();
        Payment p3 = new UPIPayment();
        p1.pay(1000);
        p2.pay(500);
        p3.pay(250);
    }
}

```

OUTPUT:

```

C:\Users\chscu\OneDrive\Desktop>java e
Paid ?1000.0 using Credit Card.
Paid ?500.0 using Debit Card.
Paid ?250.0 using UPI.

```

Q no 4)

AIM: multiple interface implementation

CODE:

```
// Printable.java
interface Printable {
    void print();
}

// Scannable.java
interface Scannable {
    void scan();
}

// PrinterScanner.java
class PrinterScanner implements Printable, Scannable {
    public void print() {
        System.out.println("Printing document...");
    }

    public void scan() {
        System.out.println("Scanning document...");
    }
}

public class e {
    public static void main(String[] args) {
        PrinterScanner device = new PrinterScanner();

        device.print();
        device.scan();
    }
}
```

OUTPUT:

```
C:\Users\chscu\OneDrive\Desktop>java e
Printing document...
Scanning document...
```


7) ENCAPSULATION

Qno 1)

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.setld(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

```

Qno 2)

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.
```

Qno 3)

AIM: Online course platform

ALGORITHM:

1. Start
2. Create Course class
 - Define private attributes
 - Create setter methods for course details
 - Create getter methods for course details
 - Create enrollStudent() method
 - Create dropStudent() method
3. Create Main class
 - Create an object of Course
 - Set course details
 - Display course details
 - Enroll a student
 - 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
```

Qno 4)

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.");
    }
}

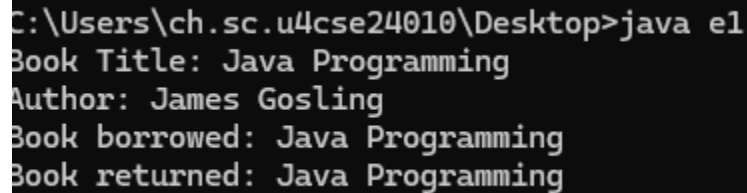
// Return a book
void returnBook() {
    availableCopies++;
    System.out.println("Book returned: " + title);
}

// 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
```

8) PACKAGES

Q no 1)

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:

- Compute the median as the average of the two middle numbers:

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

9. If n is odd:

- Median is the middle element in the sorted list:

$$\text{median} = \text{numbers}[n/2]$$

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());
        }
        // Sorting the list
        Collections.sort(numbers);
        // Finding median
        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

```

Q no 2)

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

ALGORITHM:

Step 1: Initialize Counters

1. Declare and initialize three counters:
 - charCount = 0 (to store the number of characters)
 - wordCount = 0 (to store the number of words)
 - 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):
 - Increment lineCount by 1.
 - Add the length of the line to charCount (to count characters).
 - Split the line into words using split("\\s+") (to count words).
 - 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
```

Q no 3)

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).
 - Append each word to the `StringBuilder`.
 - 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:

```
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

```

Q no 4)

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:
 - "Hello, this is a test file!" (followed by a newline).
 - "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:
 - 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";
        // Writing to the file
        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());
        }
        // Reading from the file
        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.
```

9.FILE HANDLING

Q no 1)

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 occurred");
```

```
e.printStackTrace();
}
}
}
```

OUTPUT:

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

Q no 2)

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
```

Q no 3)

AIM: To 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.
```

Q no 4)**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
```

10. EXCEPTION HANDLING

Q no 1)

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...
```

Q no 2)

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.");
        }
    }
}
```

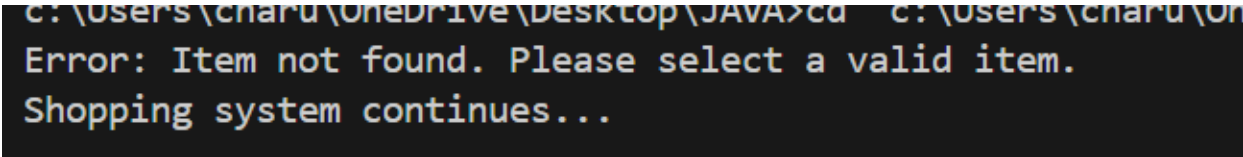


```

    } 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\OneDrive\Desktop\JAVA
Error: Item not found. Please select a valid item.
Shopping system continues...

```

Q no 3)

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

CODE:

```

public class bank {

    // Step 2: Method that may throw an exception
    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) {
        // Step 5: Catch and handle the exception
        System.out.println("Booking Failed: " + e.getMessage());
    }
    finally {
        // Step 6: Cleanup or final message
        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.

```

Q no 4)

Aim: ATM WITHDRAWAL SYSTEM

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

```