



SCHOOL OF  
COMPUTING

# LAB RECORD

23CSE111- Object Oriented Programming

*Submitted by*

CH.SC.U4CSE24020 -Kalathuru Varshitha

## BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING

AMRITA VISHWA VIDYAPEETHAM  
AMRITA SCHOOL OF COMPUTING

CHENNAI

March - 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.U4CSE24020 – Kalathuru Varshitha** 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    /    /2025.

Internal Examiner 1

Internal Examiner 2

# INDEX

S.NO	TITLE	PAGE.NO
	<b>UML DIAGRAM</b>	
<b>1.</b>	<b>TITLE OF UML DIAGRAM -1</b>	
	1.a) Use Case Diagram	05-06
	1.b) Class Diagram	06-07
	1.c) Sequence Diagram	07-08
	1.d) Object Diagram	08-09
	1.e) State Diagram	09-10
<b>2.</b>	<b>TITLE OF UML DIAGRAM -2</b>	
	2.a) Use Case Diagram	10-11
	2.b) Class Diagram	11-12
	2.c) Sequence Diagram	12-13
	2.d) Object diagram	13-14
	2.e) State Diagram	14-15
<b>3.</b>	<b>BASIC JAVA PROGRAMS</b>	
	3.a) to get the given number is even or odd.	15-16
	3.b) to reverse a number dynamic input.	16-17
	3.c) to convert meters into centimetres.	17-18
	3.d) to check the given number is palindrome	18-19
	3.e) to calculate the arithmetic operations	19-20
	3.f) java program on class and attributes.	21-22
	3.g) to find the area and volume of a room.	22-23
	3.h) Create a derived class Car that inherits from Vehicle and adds a method drive().	23-25
	3.i) To generate pay slips for the employees.	25-26
	3.j) Example for multilevel inheritance	26-27

	<b>INHERITANCE</b>	
4.	<b>SINGLE INHERITANCE PROGRAMS</b>	
	4.a) Single inheritance simple program	27-28
	4.b) Single inheritance dynamic input	28-29
5.	<b>MULTILEVEL INHERITANCE PROGRAMS</b>	
	5.a) Multilevel inheritance simple program	29-30
	5.b) Multilevel dynamic input	30-31
6.	<b>HIERARCHICAL INHERITANCE PROGRAMS</b>	
	6.a) Hierarchical inheritance dynamic input	31-33
	6.b) Hierarchical inheritance simple program	33-36
7.	<b>HYBRID INHERITANCE PROGRAMS</b>	
	7.a) Program on hybrid inheritance dynamic input	36-37
	7.b) Program on hybrid inheritance	38-39
	<b>POLYMORPHISM</b>	
8.	<b>CONSTRUCTOR PROGRAMS</b>	39-40
	8.a) Student Registration Using Constructor	
9.	<b>CONSTRUCTOR OVERLOADING PROGRAMS</b>	40-42
	9.a) Food order	
10.	<b>METHOD OVERLOADING PROGRAMS</b>	42-44
	10.a) Online Payment System – Method Overloading	
	10.b) Ticket Booking System – Method Overloading	
11.	<b>METHOD OVERRIDING PROGRAMS</b>	44-47
	11.a) Travel App – Method Overriding	
	11.b) Restaurant Ordering System – Method Overriding	
	<b>ABSTRACTION</b>	43-54
12.	<b>INTERFACE PROGRAMS</b>	
	12.a) Restaurant Ordering System	
	12.b) Smart Home System	
	12.c) Restaurant Ordering System	
	12.d) Flight Booking System	
13.	<b>ABSTRACT CLASS PROGRAMS</b>	54-58
	13.a) Online Payment System	
	13.b) Shopping Cart System	
	13.c) Vehicle License System	
	13.d) Vehicle License eligible System	
	<b>ENCAPSULATION</b>	
14.	<b>ENCAPSULATION PROGRAMS</b>	58-
	14.a) Encapsulation (Using Getter/Setter Methods)	
	14.b) Encapsulation Using Getter & Setter Validation	
	14.c) Encapsulation using dynamic input	

	14.d) Employee Leave Management System	
15.	<b>PACKAGES PROGRAMS</b>	
	15.a) User Defined Packages	
	15.b) User Defined Packages	
	15.c) Built – in Package (3 Packages)	
	15.d) Built – in Package (3 Packages)	
16.	<b>EXCEPTION HANDLING PROGRAMS</b>	73-79
	16.a) Online Exam System – Invalid Option Exception	
	16.b) Handling FileNotFoundException	
	16.c) ATM Withdrawal – InsufficientFundsException	
	16.d) Login – MaxLoginAttemptsExceededException	
17.	<b>FILE HANDLING PROGRAMS</b>	79-84
	17.a) Visitor Log System (Appending Names to a File)	
	17.b) Task Reminder Sys (Reading Tasks from a File)	
	17.c) Student Feedback Collector	
	17.d) Online Product Rating Saver	

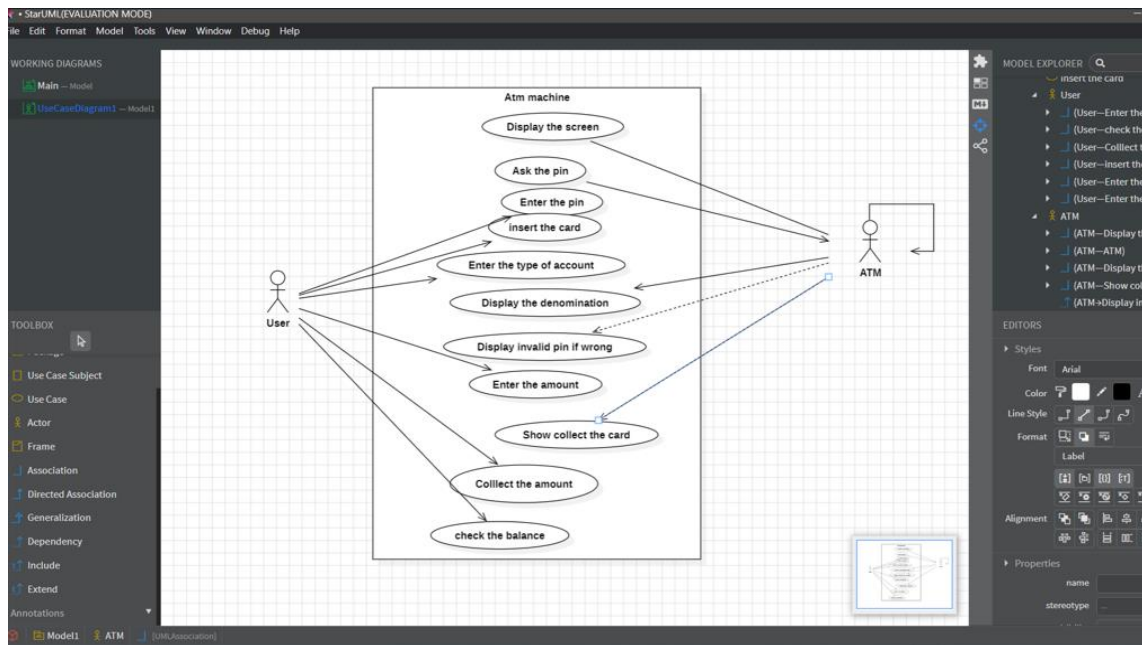
1)

## UML Diagrams - 1

1(a) How does a user interact with an ATM machine during a typical transaction?

### **Aim**

To analyze and understand the sequential process flow of actions performed by a user and corresponding system responses during an ATM transaction.



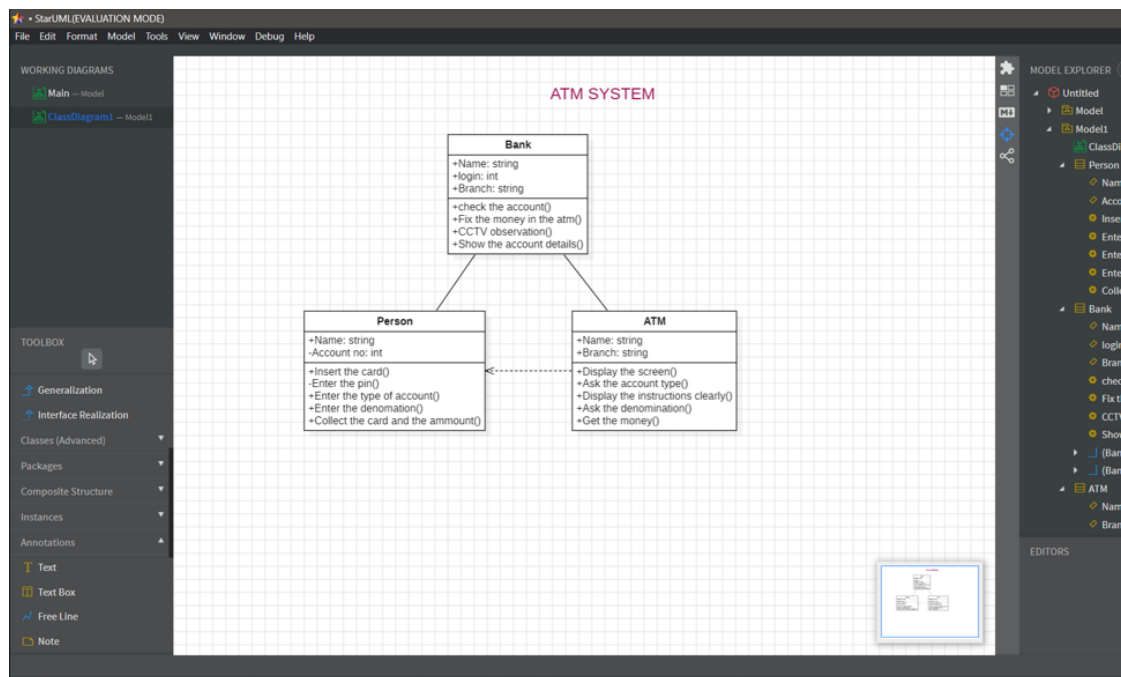
### **Result**

A detailed use case diagram visually representing the user's interaction with the ATM machine. It outlines steps like inserting the card, entering the PIN, selecting the type of account, inputting the amount, collecting the card, withdrawing cash, and checking balance. It also includes validation processes like displaying error messages for invalid PINs. This diagram helps clarify the entire transaction flow in an organized manner.

# 1(b) What are the attributes and methods involved in the interaction between a Bank, a Person, and an ATM system during a transaction?

## Aim

To analyze and represent the structure of interactions between the three entities—Bank, Person, and ATM—by listing their attributes and methods. This aims to understand how each entity contributes to a seamless ATM transaction.



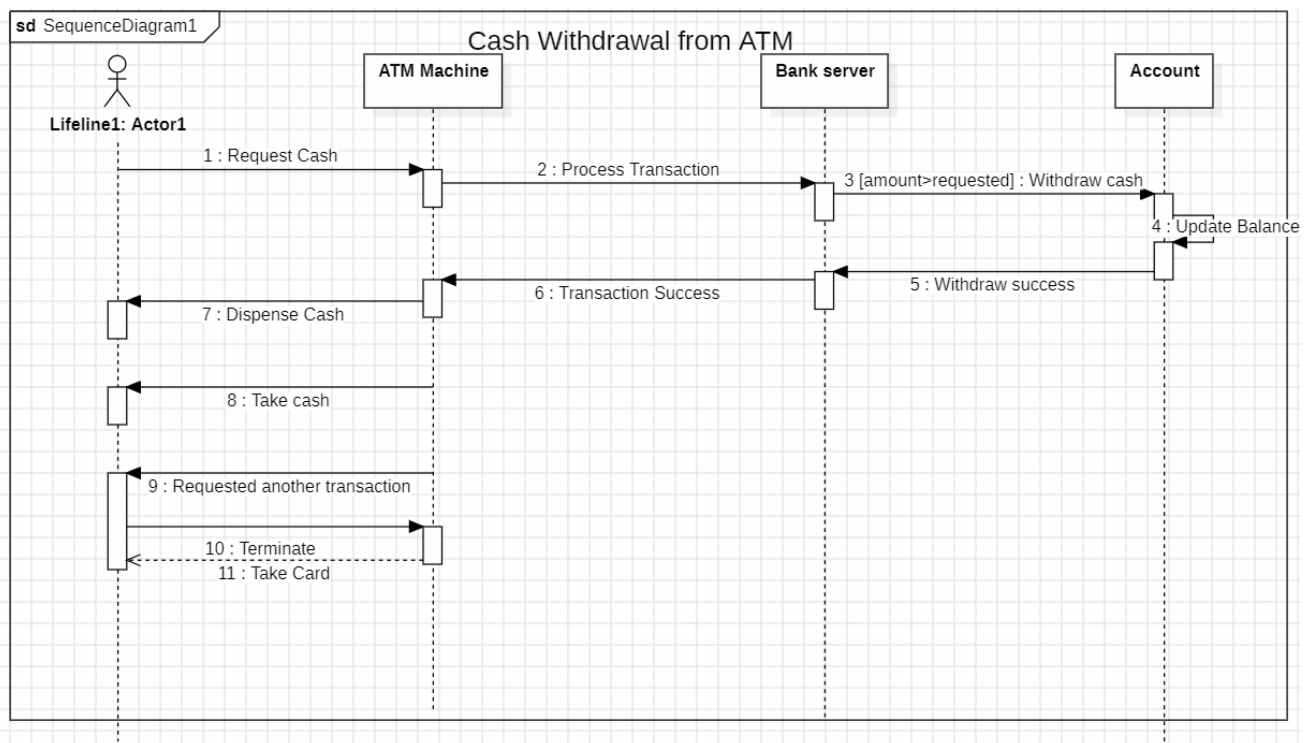
## Result

- Bank: Attributes like name and agent; methods .
- Person: Attributes like name and account number; methods
- ATM: Attributes like name and branch; methods such as displaying instructions, receiving card input, and dispensing cash.

1(c)What is the process for withdrawing cash from an ATM as represented in a sequence diagram?

### Aim

To analyze and understand the step-by-step interactions between the user, ATM machine, bank server, and account during the cash withdrawal process.



### Results:

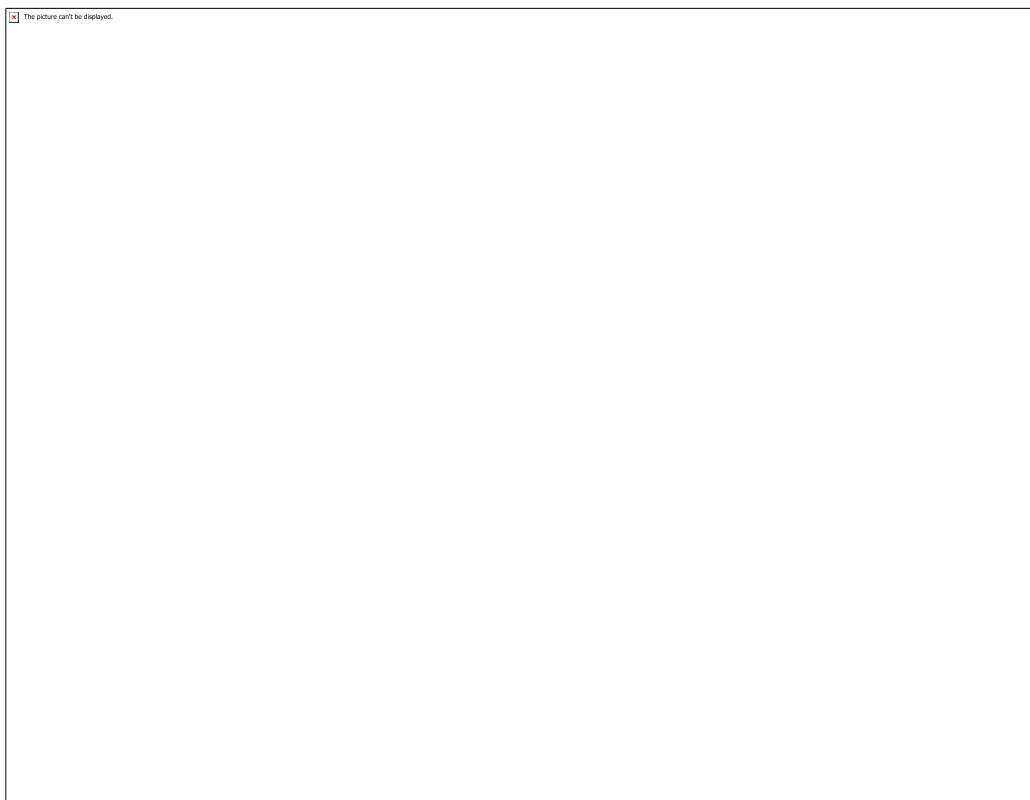
- User's Actions: Insert card, enter PIN, select transaction, collect cash/card.
- ATM's Role: Validate PIN, communicate with the bank, dispense cash.
- Bank's Response: Confirm account balance, authorize transaction, update balance.



**1(d)**What is the process flow for ATM withdrawal as illustrated in the flowchart?

**Aim**

To describe the sequential flow of an ATM withdrawal process, including decision-making points, based on the flowchart provided.



**Results:**

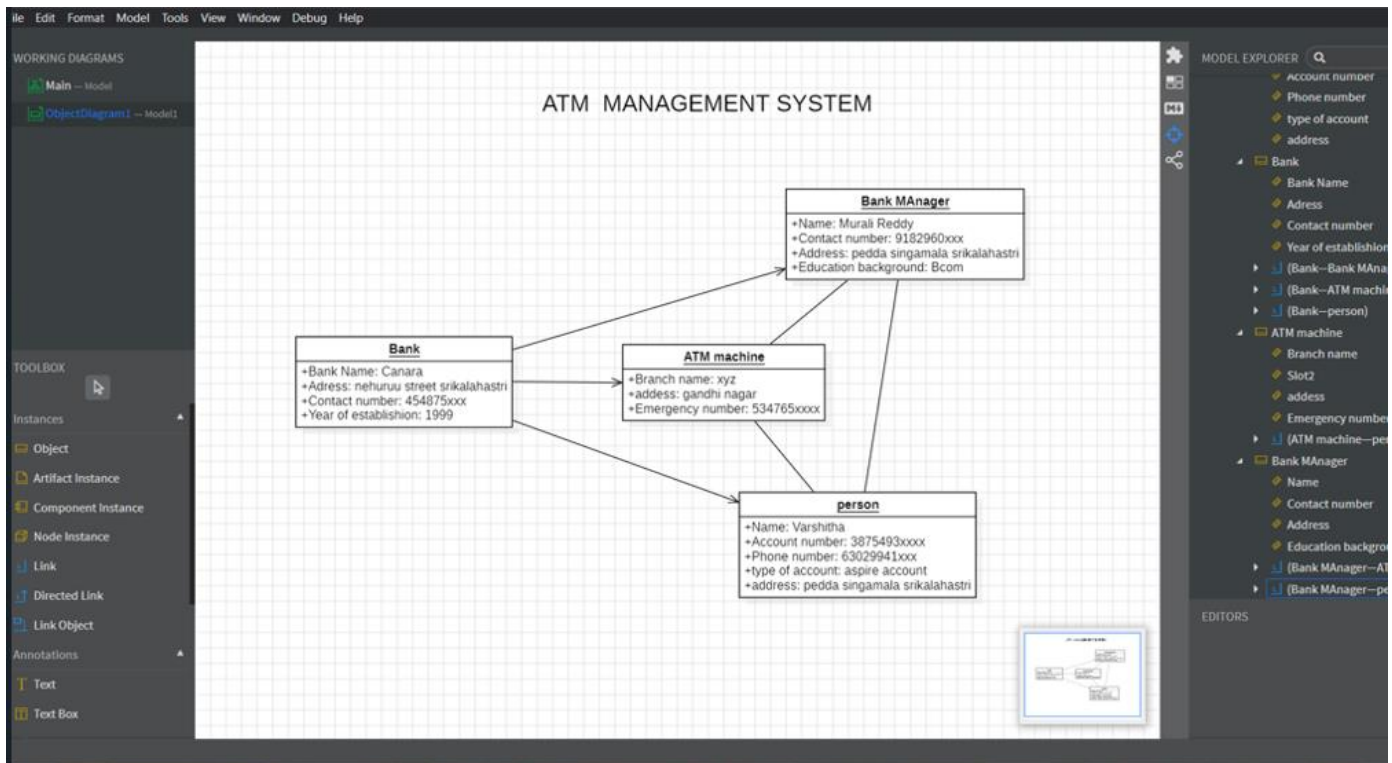
- Start: Card insertion initializes the process.
- PIN Validation: Confirms user authenticity.
- Transaction Input: User enters withdrawal amount.
- Decision Points: Checks PIN validity and account balance.
- Outcome: Dispenses cash and prints receipt, or denies transaction if conditions aren't met.

Let me know if you'd like more details!

**1(e)**What does the ATM Management System diagram depict?

## Aim

To identify the entities, attributes, and relationships represented in the ATM Management System diagram.



## Result

- Entities: Bank, ATM machine, Bank Manager, Person.
- Attributes: Names, contact numbers, addresses, and related details for each entity.
- Relationships: Connections between the Bank, ATM machine, Bank Manager, and Person.

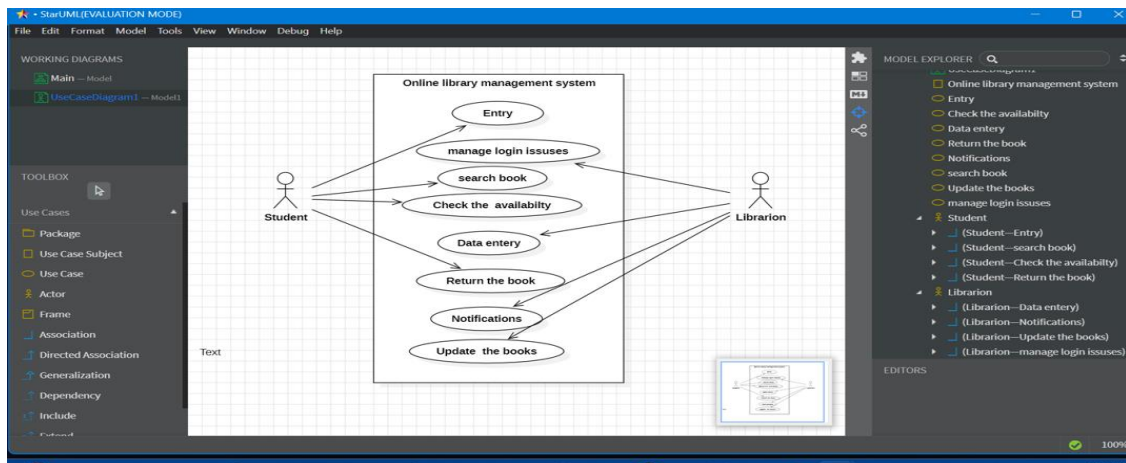
## 2

## UML Diagrams – 2

2(a) What interactions and functionalities does an online library management system provide to students and librarians?

### Aim

To analyze and illustrate the functionalities of an online library management system and how students and librarians interact with it.



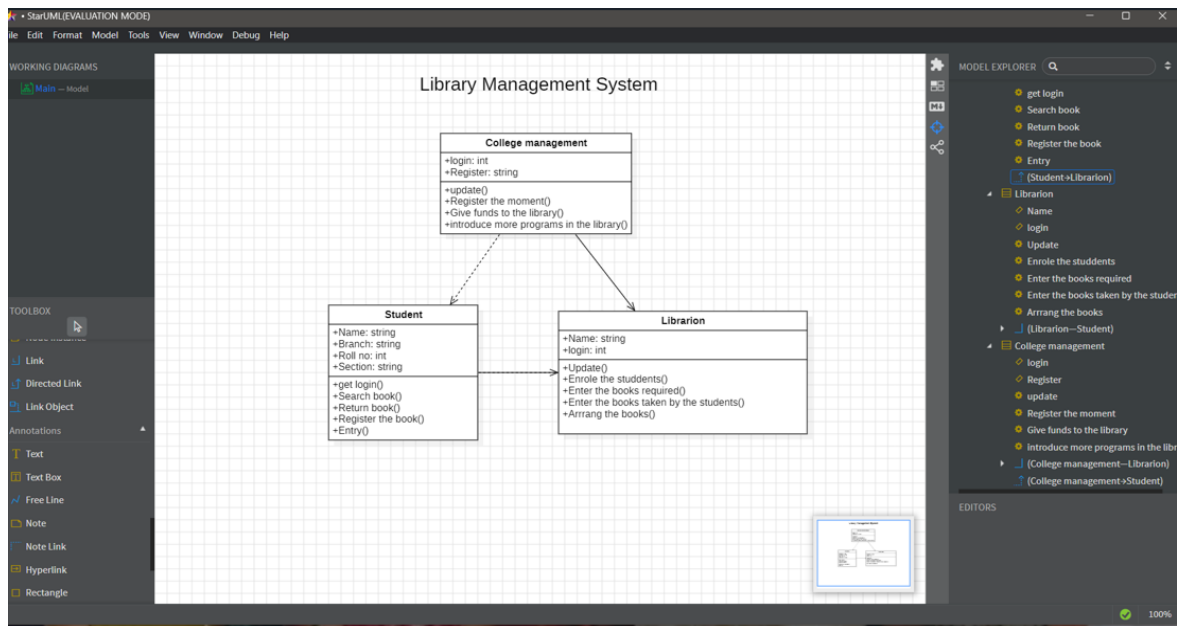
### Result:

- Students' Actions: Search books, check availability, return books, receive notifications.
- Librarians' Actions: Manage login issues, perform data entry, update books, handle returns, and send notifications.
- System Functionalities: Facilitates entry, book updates, and overall library management.

## 2(b) What functionalities and processes are represented in the Library Management System diagram?

### Aim

To analyze the attributes and methods of the College Management, Student, and Librarian entities as depicted in the Library Management System diagram.




### Result:

- College Management: Handles registration, member management, funding, and introducing programs for the library.
- Student: Can log in, search for, issue, return, and renew books with attributes like name, branch, and ID.
- Librarian: Manages book issuance, returns, renewals, late fines, and overall library operations.

## **2(c)What is the detailed sequence of actions involved in user registration and login validation processes within the system?**

### **Aim:**

To demonstrate the flow of information and interactions between the user, system, and database during registration and login processes, including handling valid, invalid, existing user, and new user scenarios.

The picture can't be displayed.

### **Result:**

The sequence diagram visually breaks down the registration and login process into discrete steps, validating registration details for new users, addressing invalid registrations, confirming successful registrations, and validating login credentials to ensure accurate and secure access.

2(d)How does the course registration system handle interactions between students, course details, registered courses, and external systems during the course registration process?

**Aim:**

To illustrate the flow of information and interactions between students and the systems involved, focusing on retrieving course information, selecting courses, and registering for courses successfully.

The picture can't be displayed.

**Result:**

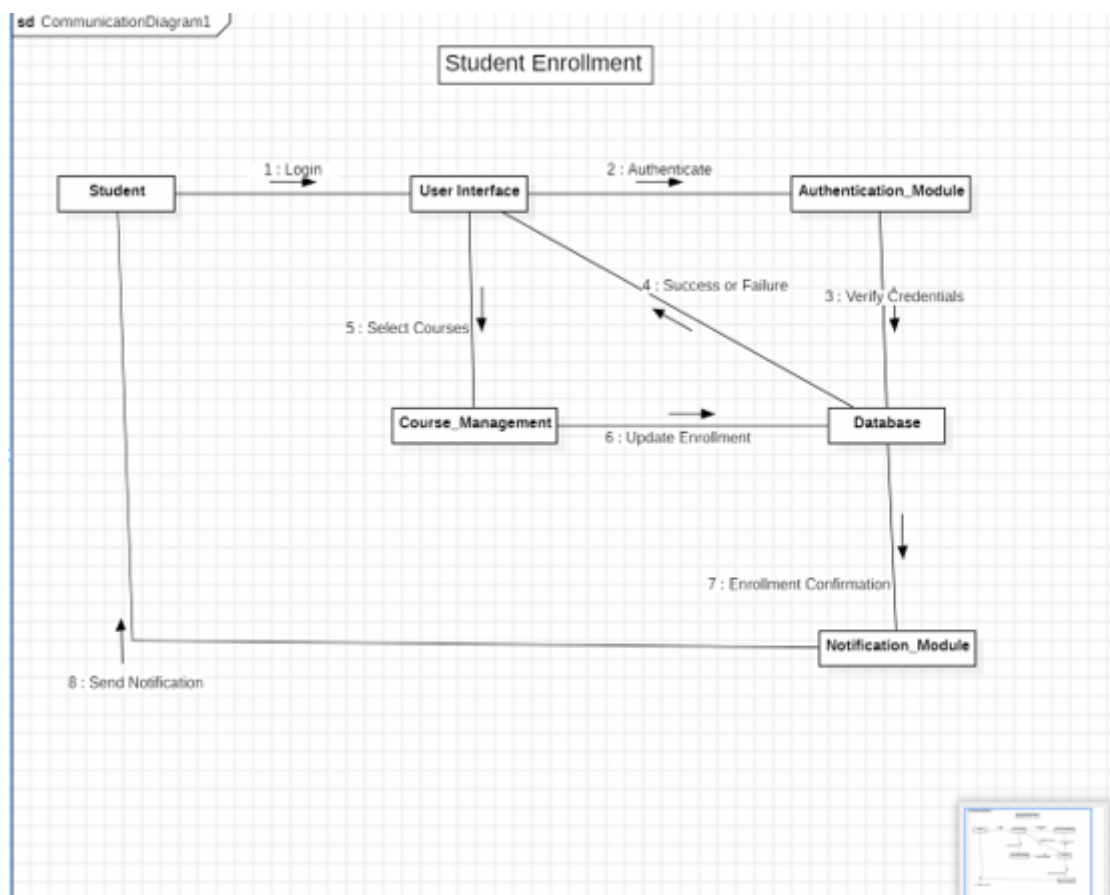
The communication diagram outlines the sequence of steps and messages exchanged between various entities, showcasing how students access the course catalog, retrieve course details from the legacy system, and register for courses

in the system. It highlights the dependencies and interactions that ensure the smooth functioning of the course registration process.

2(e)What does the Library Management System diagram reveal about the entities and their attributes?

### Aim

To identify the key entities, their attributes, and interactions as represented in the Library Management System diagram.



### Result:

- College: Includes attributes like name (e.g., Anna University), address (Chennai), branches, student and teacher count, and library proximity.
- Librarian: Attributes such as name, phone number, email, and address; handles tracking borrowed books.
- Students: Each student (e.g., Vishnu, Raushan, Pratiksha) is represented with attributes like name, branch, roll number, and email.

**1)** Write java program to get where the given number is even or odd.

**AIM:** To write java program to get where the given number is even or odd.

**Code:**

```
import java.util.Scanner;
public class evenorodd
{
    public static void main(String[] args)
    {
        int n;
        Scanner s=new Scanner(System.in);
        System.out.print("Enter the number:");
        n=s.nextInt();
        if(n%2==0)
        {
            System.out.println("is even number");
        }
        else
        {
            System.out.println("is odd number");
        }
    }
}
```

**Output:**





**2)Write java program to reverse a number and take the input from the user.**

**AIM:**to write java program to reverse a number and take the input from the user.

**Code:**

```
import java.util.Scanner;
public class reverse
{
    public static void main(String[] args)
    {
        int n ;
        int m ,reverse=0;
        Scanner s= new Scanner(System.in);
        System.out.println("Enter the number:");
        n=s.nextInt();
        while(n>0)
        {
            m=n%10;
            reverse=reverse*10+m;
            n=n/10;
        }
        System.out.println("The reverse is"+reverse);
    }
}
```

**Output:**



### 3)Write java to convert meters into centimeters.

**AIM:** Towrite java to convert meters into centimeters

**Code:**

```
import java.util.Scanner;
public class distance
{
    public static void main(String[] args)
    {

        int a;
        Scanner s= new Scanner(System.in);
        System.out.println("Enter the centimeter");
        a=s.nextInt();
        double centimeter= a*100;
        System.out.println( "centimeter is equal to" +centimeter);
    }
}
```

**Output:**

```
C:\Users\VARSHITHA\Desktop\java>java distance
Enter the centimeter
2
centimeter is equal to200.0
```

**4)Write java program to check wheather the given number is palindrom or not.**

**AIM:** To java program to check wheather the given number is palindrom or not.

**Code:**

```
import java.util.Scanner;
public class palindrome
{
    public static void main(String[] args)
    {
        int n,m,reverse=0,a;
        Scanner s=new Scanner(System.in);
        System.out.print("Enter a number:");
        n=s.nextInt();
        m=n;
        while(n>0)
        {
            a=n%10;
            reverse=reverse*10+a;
            n=n/10;
        }
        if(reverse==m)
        {
            System.out.println("Number is palindrome");
        }
        else
        {
            System.out.println("Number is not a palindrome");} }
}
```

**Output:**

```
Enter a number:123  
Number is not a palindrome
```

### 5)Write java program to calculate the arithmetic operations

**Aim:**To write java program to to calculate the arithmetic operations

**Code:**

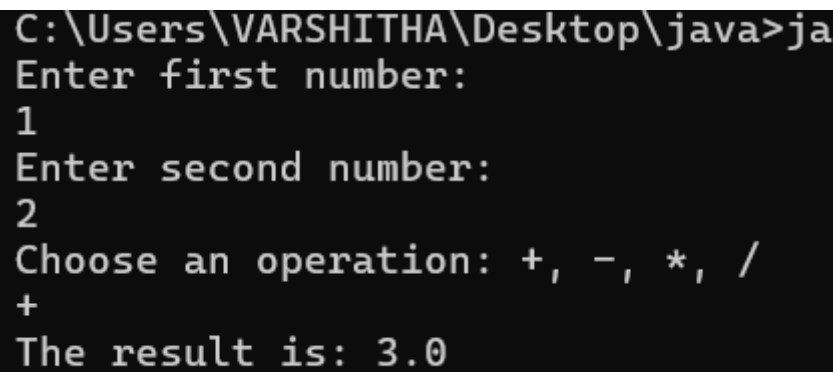
```
import java.util.Scanner;  
public class Calculator {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
        System.out.println("Enter first number: ");  
        double num1 = scanner.nextDouble();  
        System.out.println("Enter second number: ");  
        double num2 = scanner.nextDouble();  
        System.out.println("Choose an operation: +, -, *, /");  
        char operation = scanner.next().charAt(0);  
        double result;  
        switch (operation) {  
            case '+':  
                result = num1 + num2;  
                break;  
            case '-':  
                result = num1 - num2;  
                break;  
            case '*':  
                result = num1 * num2;  
                break;  
            case '/':  
                if (num2 != 0) {  
                    result = num1 / num2;  
                } else {  
                    System.out.println("Error! Division by zero.");  
                }  
            }  
    }  
}
```

```

        return;
    }
    break;
default:
    System.out.println("Invalid operation!");
    return;
}
System.out.println("The result is: " + result);
}
}

```

**Output:**



```

C:\Users\VARSHITHA\Desktop\java>ja
Enter first number:
1
Enter second number:
2
Choose an operation: +, -, *, /
+
The result is: 3.0

```

**6)Write java program on class and attributs and constructors.**

**Aim:** To write java program on class and attributs and constructors.

**Code:**

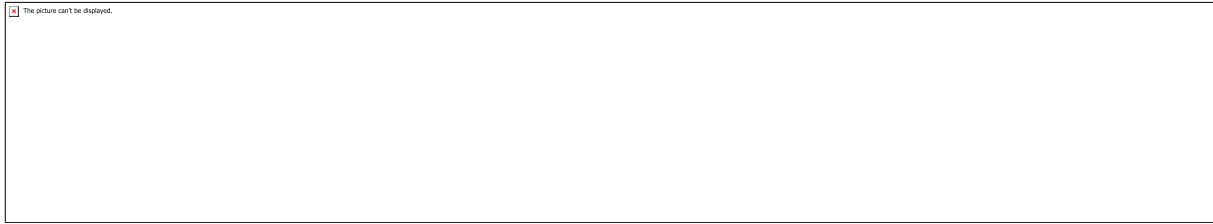
```

public class Book {
    String title; String author; int year;
    public Book(String title, String author) {
        this.title = title;    this.author = author;
    }
    public Book(String title, String author, int year) {
        this.title = title;    this.author = author;    this.year = year; }
    public void displayDetails() {
        System.out.println("Title: " + title + ", Author: " + author + ", Year: " + year);
    }
    public static void main(String[] args) {
        Book book1 = new Book("1984", "George Orwell");
        Book book2 = new Book("To Kill a Mockingbird", "Harper Lee", 1960);
    }
}

```

```
book1.displayDetails();    book2.displayDetails();  } }
```

### Output:



### 7) To write a java program to implement single inheritance to find the area and volume of a room.

**Aim:**To write a java program to implement single inheritance to find the area and volume of a room

```
import java.util.Scanner;
class Room
{
    int area,volume;
    void getarea()
    {
        System.out.println("The area of the room is:"+area);
    }
    void getvolume()
    {
        System.out.println("The volume of the room is:"+volume);
    }
}

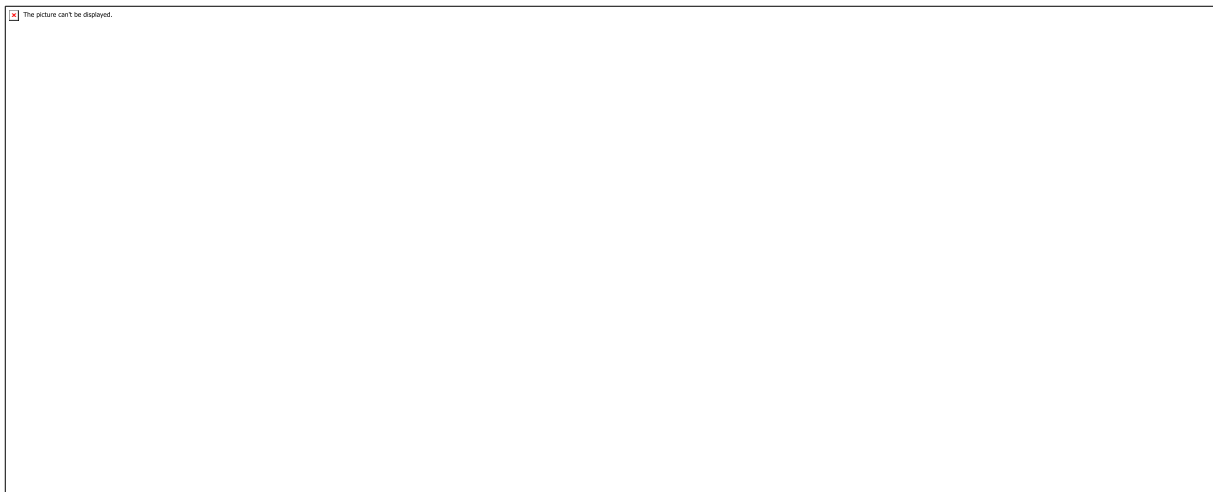
class areavolume extends Room
{
    public static void main(String[] args)
    {int l,b,h;
        Scanner s=new Scanner(System.in);
        System.out.println("Enter the length:");
        l=s.nextInt();
```

```

System.out.println("Enter the breadth:");
b=s.nextInt();
System.out.println("Enter the height:");
h=s.nextInt();
areavolume obj=new areavolume();
obj.area=l*b;
obj.volume=l*b*h;
obj.getarea();
obj.getvolume();
}}

```

### Output:



**8)Single Inheritance:** Create a base class called **Vehicle** with attributes like **make, model,** and methods like **start(), stop()**. Create a derived class **Car** that inherits from **Vehicle** and adds a method **drive()**.

**Aim:** To solve the problem based on the inheritance

### Code:

```

import java.util.Scanner;
class Room
{
    int area,volume;
    void getarea()
    {
        System.out.println("The area of the room is:"+area);
    }
    void getvolume()

```

```

    {
        System.out.println("The volume of the room is:"+volume);
    }
}
class areavolume extends Room
{
    public static void main(String[] args)
    {
        int l,b,h;
        Scanner s=new Scanner(System.in);
        System.out.println("Enter the length:");
        l=s.nextInt();
        System.out.println("Enter the breadth:");
        b=s.nextInt();
        System.out.println("Enter the height:");
        h=s.nextInt();
        areavolume obj=new areavolume();
        obj.area=l*b;
        obj.volume=l*b*h;
        obj.getarea();
        obj.getvolume();
    }
}

```

### Output:

```

C:\Users\VARSHITHA\Desktop>java areavolume
Enter the length:
2
Enter the breadth:
2
Enter the height:
2
The area of the room is:4
The volume of the room is:8

```



### **9)To write a java program to generate pay slips for the employees.**

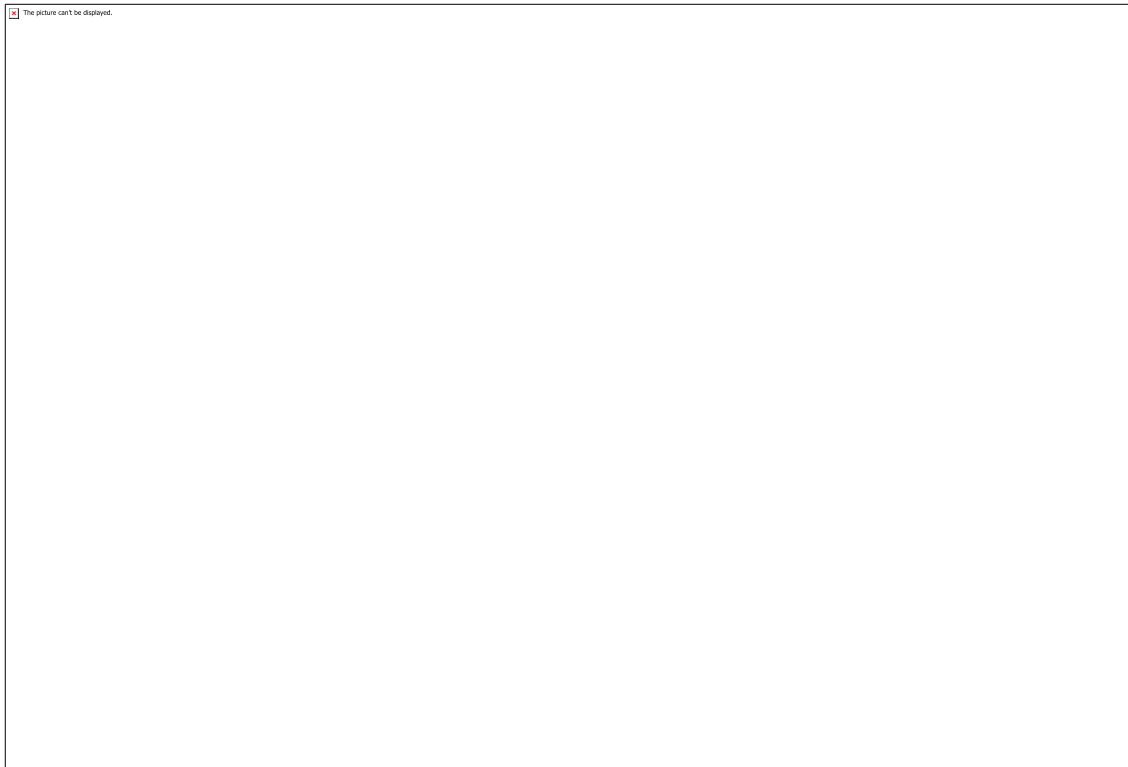
**Aim:**To write a java program to generate pay slips for the employees.

**Code:**

```
import java.util.Scanner;
class slip
{
    String name;
    String role;
    int bonus;
    int salary;
    int total;
    void manager()
    {
        System.out.println("Name: " + name);
        System.out.println("Role: " + role);
        System.out.println("Bonus: " + bonus);
        System.out.println("Salary: " + salary);
        System.out.println("Total salary: " + total);
    }
}
class slips extends slip
{
    public static void main(String[] args)
    {
        Scanner s= new Scanner(System.in);
        slip obj=new slip();
        System.out.println("Enter the name:");
        obj.name=s.nextLine();
        System.out.println("Enter the role:");
        obj. role=s.nextLine();
        System.out.println("Enter the bonus:");
```

```
obj.bonus=s.nextInt();
System.out.println("Enter the salary:");
obj.salary=s.nextInt();
obj.total=obj.bonus+obj.salary;
obj.manager();
}
}
```

### **Output:**



### **10)Example for multilevel inheritance.**

**Aim:**Example for multilevel inheritance.

#### **Code:**

```
class Shape
{
    public void display()
    {
        System.out.println("Inside display"); }
}
class Rectangle extends Shape
```

```
{
    public void area()
    { System.out.println("Inside area");
    }
}
class Cube extends Rectangle
{
    public void volume()
    {
        System.out.println("Inside volume");
    }
}
public class Tester { public static void main(String[] arguments)
{
    Cube cube = new Cube(); cube.display(); cube.area(); cube.volume(); }
}
```

**Output:**

```
Inside display
Inside area
Inside volume
```

## INHERITANCE PROGRAMS-4

### 4(a)single inhertance

**Aim:**is to demonstrate the concept of inheritance in Java.

```
class Teacher {
    String name;

    void teach()
    {
        System.out.println(name + " teaches.");
    }
}

class MathTeacher extends Teacher
{
    void subject()
    {
        System.out.println(name + " teaches Mathematics.");
    }
}

public class Teachers {
    public static void main(String[] args) {
        MathTeacher obj = new MathTeacher();
        obj.name = "Mr. Raj";
        obj.teach();
        obj.subject();
    }
}
```

**Output:**

```
C:\Users\VARSHITHA\Desktop>javac Teachers.java

C:\Users\VARSHITHA\Desktop>java Teachers
Mr. Raj teaches.
Mr. Raj teaches Mathematics.
```

#### 4(b)Single inheritance dynamic input (input from the user)

**Aim:**to demonstrate inheritance in Java by creating a Person class and an Employee class that extends Person.

The picture can't be displayed.

```
C:\Users\VARSHITHA\Desktop>javac Persons_job.java
```

```
C:\Users\VARSHITHA\Desktop>java Persons_job
```

```
Enter name: Varshitha Reddy
```

```
Enter age: 18
```

```
Enter designation: Manager
```

```
Enter salary: 2000000
```

```
--- Person Details ---
```

```
Name: Varshitha Reddy
```

```
Age: 18
```

```
--- Employee Details ---
```

```
Designation: Manager
```

```
Salary: 2000000.0
```

## MULTILEVEL INHERITANCE

### 5(a) MULTILEVEL INHERITANCE

**Aim:**to demonstrate multilevel inheritance in Java by creating an class, a class that extends , and a class that extends .

The picture can't be displayed.

**Output:**

```
Animals eat food.  
Mammals walk on land.  
Dogs bark.
```

### 5(b) MULTILEVEL INHERITANCE dynamic input

**Aim:**to demonstrate multilevel inheritance in Java with user input by using , , and classes.

```
class Animal {
    void eat() {
        System.out.println("Animals eat food.");
    }
}

class Mammal extends Animal {
    void walk() {
        System.out.println("Mammals walk on land.");
    }
}

class Dog extends Mammal {
    void bark() {
        System.out.println("Dogs bark.");
    }
}

public class MultilevelInheritanceExample1 {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.eat();
        dog.walk();
        dog.bark();
    }
}
```

**Output:**

```
--- Person Details ---
Name: Arjun
Age: 25

--- Student Details ---
Course: Computer Science

--- Graduate Details ---
CGPA: 8.5
```

## HIERARCHICAL INHERITANCE PROGRAMS-6

### 6(a) inheritance program dynamic input

**Aim:** To demonstrate hierarchical inheritance in Java, where multiple child classes inherit common methods from a single parent class.

**CODE:**

```
class HospitalMember {
    String name;
    HospitalMember(String name) {
        this.name = name;
    }
    void login() {
        System.out.println(name + " logged into the system.");
    }
}
class Doctor extends HospitalMember {
    Doctor(String name) {
        super(name);
    }
    void diagnose() {
        System.out.println(name + " is diagnosing a patient.");
    }
}
class Nurse extends HospitalMember {
    Nurse(String name) {
        super(name);
    }
    void assist() {
        System.out.println(name + " is assisting in surgery.");
    }
}
class Patient extends HospitalMember {
    Patient(String name) {
```



```

        super(name);
    }

    void takeMedicine() {
        System.out.println(name + " is taking prescribed medicine.");
    }
}

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

        Doctor doc = new Doctor("Dr. Ramesh");
        Nurse nurse = new Nurse("Nurse Mary");
        Patient patient = new Patient("John Doe");

        System.out.println("--- Doctor Actions ---");
        doc.login();
        doc.diagnose();

        System.out.println("\n--- Nurse Actions ---");
        nurse.login();
        nurse.assist();

        System.out.println("\n--- Patient Actions ---");
        patient.login();
        patient.takeMedicine();
    }
}

```

**OUTPUT:**

```
--- Doctor Actions ---  
Dr. Ramesh logged into the system.  
Dr. Ramesh is diagnosing a patient.  
  
--- Nurse Actions ---  
Nurse Mary logged into the system.  
Nurse Mary is assisting in surgery.  
  
--- Patient Actions ---  
John Doe logged into the system.  
John Doe is taking prescribed medicine.
```

### 6(b)products like clothing and electronics

To write a Java program to demonstrate Hierarchical Inheritance by creating a superclass Product and subclasses Electronics, Clothing, and Grocery, each having specific behaviors relevant to their category.

CODE:

```
class Product {  
    String name;  
    double price;  
  
    Product(String name, double price) {  
        this.name = name;  
        this.price = price;  
    }  
  
    void displayProduct() {  
        System.out.println("Product: " + name + ", Price: $" + price);  
    }  
}  
  
class Electronics extends Product {  
    Electronics(String name, double price) {  
        super(name, price);  
    }  
}
```

```

    }

    void warranty() {
        System.out.println("Electronics warranty: 1 year");
    }
}

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

    void fabricInfo() {
        System.out.println("Fabric: 100% cotton");
    }
}

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

    void expiryDate() {
        System.out.println("Check expiration date before use.");
    }
}

public class hari {
    public static void main(String[] args) {
        Electronics laptop = new Electronics("Laptop", 950.00);
        Clothing tshirt = new Clothing("T-Shirt", 19.99);
        Grocery rice = new Grocery("Basmati Rice", 25.50);
    }
}

```

```
System.out.println("--- Electronics ---");  
laptop.displayProduct();  
laptop.warranty();
```

```
System.out.println("\n--- Clothing ---");  
tshirt.displayProduct();  
tshirt.fabricInfo();
```

```
System.out.println("\n--- Grocery ---");  
rice.displayProduct();  
rice.expiryDate();
```

```
}
```

```
}
```

OUTPUT:

```
--- Electronics ---  
Product: Laptop, Price: $950.0  
Electronics warranty: 1 year  
  
--- Clothing ---  
Product: T-Shirt, Price: $19.99  
Fabric: 100% cotton  
  
--- Grocery ---  
Product: Basmati Rice, Price: $25.5  
Check expiration date before use.
```

## HYBRID INHERITANCE PROGRAMS-7

### 7(a) Simple program on hybrid inheritance.

**Aim:** To understand the usage of static variables, inheritance, and interface implementation in Java programming through the given example.

```
import java.util.Scanner;
class employee{
    static String name ;
    static int id;
    static String role;
    public void info(){
        System.out.println("Name : " + name);
        System.out.println("ID : " + id);
        System.out.println("Designation : " + role);
    }
}
interface lead{
    public void leadTeam();
}
interface supervise{
    public void superviseTeam();
}
class manager extends employee implements lead{
    public void leadTeam(){
        System.out.println(this.name + " leads the team.");}
}
class director extends manager implements supervise{
    public void superviseTeam(){
        System.out.println(this.name + " supervises the team.");}
}
class Main{
    public static void main (String[] args){
        director A = new director();
        Scanner sc = new Scanner(System.in);
        System.out.print("enter your name : ");
        A.name=sc.nextLine();
        System.out.print("enter your designation : ");
        A.role=sc.nextLine();
        System.out.print("enter your ID : ");
        A.id=sc.nextInt();
        A.info();
        A.leadTeam();
        A.superviseTeam();
    }
}
```

### Output:

```
enter your name : Lisa
enter your designation : director
enter your ID : 8
Name : Lisa
ID : 8
Designation : director
Lisa leads the team.
Lisa supervises the team.
```

## 7(b)Simple program on hybrid inheritance.

**Aim:** To understand the usage of static variables, inheritance, and interface implementation in Java programming through the given example.

```
interface Base {
    void displayBase();
}
interface Derived1 extends Base {
    void displayDerived1();
}
interface Another {
    void displayAnother();
}
class Derived2 implements Derived1, Another {
    @Override
    public void displayBase() {
        System.out.println("This is the Base interface method.");
    }

    @Override
    public void displayDerived1() {
        System.out.println("This is the Derived1 interface method.");
    }

    @Override
    public void displayAnother() {
        System.out.println("This is the Another interface method.");
    }

    public void displayDerived2() {
        System.out.println("This is the Derived2 class, implementing Derived1 and Another.");
    }
}
public class HybridInheritanceExample {
    public static void main(String[] args) {
        Derived2 obj = new Derived2();
        obj.displayBase();
        obj.displayDerived1();
        obj.displayAnother();
        obj.displayDerived2();
    }
}
```

### Output:

```
This is the Base interface method.
This is the Derived1 interface method.
This is the Another interface method.
This is the Derived2 class, implementing Derived1 and Another.
```

## CONSTRUCTOR PROGRAMS-8

### **8a) Student Registration Using Constructor**

**Aim :** To implement a Java program that demonstrates the use of constructors for initializing object data, using a real-time example of student registration.

**CODE:**

```
class Student {
    String name;
    int rollNumber;
    String course;
    Student(String studentName, int roll, String courseName) {
        name = studentName;
        rollNumber = roll;
        course = courseName;
    }
    void displayDetails() {
        System.out.println("Student Name : " + name);
        System.out.println("Roll Number : " + rollNumber);
        System.out.println("Course    : " + course);
    }
}

public class StudentRegistration {
    public static void main(String[] args) {
        // Creating object and passing values through constructor
        Student s1 = new Student("Riya Sharma", 101, "Computer Science");
        Student s2 = new Student("Karan Verma", 102, "Information
        Technology");
        s1.displayDetails();
        System.out.println("-----");
        s2.displayDetails();
    }
}
```

**OUTPUT:**

```
C:\Users\HP\OneDrive>javac F.java

C:\Users\HP\OneDrive>java F
Student Name : Riya Sharma
Roll Number  : 101
Course       : Computer Science
-----
Student Name : Karan Verma
Roll Number  : 102
Course       : Information Technology
```

## CONSTRUCTOR OVERLOADING PROGRAMS-9

### 9a)Food order

#### Aim:

To write a Java program that demonstrates constructor overloading as an example of compile-time polymorphism, using a real-time scenario of an online food ordering system where users can place orders with different levels of details.

#### CODE:

```
class FoodOrder {
    String customerName;
    String foodItem;
    String deliveryType;
    int quantity;
    FoodOrder(String foodItem) {
        this.customerName = "Guest";
        this.foodItem = foodItem;
        this.quantity = 1;
        this.deliveryType = "Standard";
    }
    FoodOrder(String foodItem, int quantity) {
        this.customerName = "Guest";
        this.foodItem = foodItem;
```



```

        this.quantity = quantity;
        this.deliveryType = "Standard";
    }
    FoodOrder(String customerName, String foodItem, int quantity, String
        deliveryType) {
        this.customerName = customerName;
        this.foodItem = foodItem;
        this.quantity = quantity;
        this.deliveryType = deliveryType;
    }
    void showOrder() {
        System.out.println("Customer Name : " + customerName);
        System.out.println("Food Item    : " + foodItem);
        System.out.println("Quantity    : " + quantity);
        System.out.println("Delivery Type : " + deliveryType);
        System.out.println("-----");
    }
}

public class F {
    public static void main(String[] args) {
        FoodOrder order1 = new FoodOrder("Burger");
        FoodOrder order2 = new FoodOrder("Pizza", 2);
        FoodOrder order3 = new FoodOrder("Aarav", "Pasta", 3, "Express");

        order1.showOrder();
        order2.showOrder();
        order3.showOrder();
    }
}

```

**OUTPUT:**

```
C:\Users\HP\OneDrive>javac F.java
```

```
C:\Users\HP\OneDrive>java F
```

```
Customer Name : Guest
```

```
Food Item      : Burger
```

```
Quantity       : 1
```

```
Delivery Type  : Standard
```

```
-----  
Customer Name : Guest
```

```
Food Item      : Pizza
```

```
Quantity       : 2
```

```
Delivery Type  : Standard
```

```
-----  
Customer Name : Aarav
```

```
Food Item      : Pasta
```

```
Quantity       : 3
```

```
Delivery Type  : Express  
-----
```

## METHOD OVERLOADING PROGRAMS-10

### 10.a) Online Payment System – Method Overloading

**Aim:**

To implement a Java program that demonstrates method overloading using a real-time example of an online payment system where payments are made through card, UPI, or wallet.

**CODE:**

```
class Payment {  
    void makePayment(long cardNumber) {  
        System.out.println("Payment done using Card: " + cardNumber);  
    }  
    void makePayment(String upiID) {  
        System.out.println("Payment done using UPI ID: " + upiID);  
    }  
    void makePayment(String walletName, double amount) {  
        System.out.println("Payment of ₹" + amount + " done using Wallet: " +
```

```

        walletName);
    }
}
public class F {
    public static void main(String[] args) {
        Payment pay = new Payment();
        pay.makePayment(1234567890123456L);
        pay.makePayment("user@upi");
        pay.makePayment("Paytm", 750.00);
    }
}

```

**OUTPUT:**

```

Payment done using Card: 1234567890123456
Payment done using UPI ID: user@upi
Payment of ?750.0 done using Wallet: Paytm

```

### 10.b) Ticket Booking System – Method Overloading

**Aim:**

To write a Java program that uses method overloading to simulate a ticket booking system where tickets are booked using destination, date, and number of passengers.

**CODE:**

```

class TicketBooking {
    void bookTicket(String destination) {
        System.out.println("Ticket booked to " + destination + " for 1 person.");
    }
    void bookTicket(String destination, String date) {
        System.out.println("Ticket booked to " + destination + " on " + date + " for 1 person.");
    }
    void bookTicket(String destination, String date, int passengers) {
        System.out.println("Ticket booked to " + destination + " on " + date + " for " + passengers + " passengers.");
    }
}

```

```

    }
}

public class F {
    public static void main(String[] args) {
        TicketBooking tb = new TicketBooking();

        tb.bookTicket("Delhi");
        tb.bookTicket("Mumbai", "2025-04-10");
        tb.bookTicket("Goa", "2025-05-01", 4);
    }
}

```

**OUTPUT:**

```

C:\Users\Anir\OneDrive> java F
Ticket booked to Delhi for 1 person.
Ticket booked to Mumbai on 2025-04-10 for 1 person.
Ticket booked to Goa on 2025-05-01 for 4 passengers.

```

## **METHOD OVERRIDING PROGRAMS-11**

### **11.a) Travel App – Method Overriding**

**Aim:**

To write a Java program that demonstrates method overriding using a real-time travel app with different transport modes (Bus, Train, Flight).

**CODE:**

```

class Travel {
    void travelDetails() {
        System.out.println("General travel details.");
    }
}

class Bus extends Travel {
    @Override

```

```

    void travelDetails() {
        System.out.println("Bus: AC Seater, ₹300 per ticket.");
    }
}

class Train extends Travel {
    @Override
    void travelDetails() {
        System.out.println("Train: Sleeper Class, ₹150 per ticket.");
    }
}

class Flight extends Travel {
    @Override
    void travelDetails() {
        System.out.println("Flight: Economy Class, ₹1500 per ticket.");
    }
}

public class F {
    public static void main(String[] args) {
        Travel t;

        t = new Bus();
        t.travelDetails();

        t = new Train();
        t.travelDetails();

        t = new Flight();
        t.travelDetails();
    }
}

```

## OUTPUT:

```
C:\Users\HP\OneDrive>java F
Bus: AC Seater, ?300 per ticket.
Train: Sleeper Class, ?150 per ticket.
Flight: Economy Class, ?1500 per ticket.
```

## 11.b) Restaurant Ordering System – Method Overriding

### Aim:

To implement method overriding in Java using a restaurant ordering system where different cuisines override the same `serveFood()` method.

### CODE:

```
class Restaurant {
    void serveFood() {
        System.out.println("Serving basic food.");
    }
}

class IndianCuisine extends Restaurant {
    @Override
    void serveFood() {
        System.out.println("Serving: Butter Chicken with Naan.");
    }
}

class ChineseCuisine extends Restaurant {
    @Override
    void serveFood() {
        System.out.println("Serving: Veg Manchurian with Fried Rice.");
    }
}

class ItalianCuisine extends Restaurant {
    @Override
```

```
void serveFood() {  
    System.out.println("Serving: Pasta Alfredo with Garlic Bread.");  
}  
}
```

```
public class F {  
    public static void main(String[] args) {  
        Restaurant r;  
  
        r = new IndianCuisine();  
        r.serveFood();  
  
        r = new ChineseCuisine();  
        r.serveFood();  
  
        r = new ItalianCuisine();  
        r.serveFood();  
    }  
}
```

**OUTPUT:**

```
Serving: Butter Chicken with Naan.  
Serving: Veg Manchurian with Fried Rice.  
Serving: Pasta Alfredo with Garlic Bread.
```

## **ABSTRACTION**

### **INTERFACE PROGRAMS-12**

#### **12.a) Online Shopping System**

**Aim:**

**To develop an Online Shopping System that allows users to buy Electronics or Clothing items using abstraction in Java.**

**CODE:**

```
import java.util.Scanner;
interface Shopping {
    void buyItem();
}
class Electronics implements Shopping {
    public void buyItem() {
        System.out.println("You bought an electronic gadget!");
    }
}
class Clothing implements Shopping {
    public void buyItem() {
        System.out.println("You bought a clothing item!");
    }
}
public class S {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Select Category: 1. Electronics 2. Clothing");
        int choice = scanner.nextInt();
        Shopping shopping;
        if (choice == 1) {
            shopping = new Electronics();
        } else {
            shopping = new Clothing();
        }
    }
}
```



```

    }
    shopping.buyItem();
    scanner.close();
}
}

```

**OUTPUT:**

```

Select Category: 1. Electronics  2. Clothing
2
You bought a clothing item!

```

## 12.b) Smart Home System

**Aim:**

To create a Smart Home System that controls different smart devices like Lights and Fans using abstraction.

**Code:**

```

import java.util.Scanner;
interface SmartDevice {
    void turnOn();
    void turnOff();
}
class SmartLight implements SmartDevice {
    public void turnOn() {
        System.out.println("Smart Light is turned ON.");
    }
    public void turnOff() {
        System.out.println("Smart Light is turned OFF.");
    }
}
class SmartFan implements SmartDevice {
    public void turnOn() {
        System.out.println("Smart Fan is turned ON.");
    }
    public void turnOff() {
        System.out.println("Smart Fan is turned OFF.");
    }
}

```

```

}
public class S {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Select Device: 1. Smart Light 2. Smart Fan");
        int choice = scanner.nextInt();
        SmartDevice device;
        if (choice == 1) {
            device = new SmartLight();
        } else {
            device = new SmartFan();
        }
        device.turnOn();
        device.turnOff();
        scanner.close();
    }
}

```

**Output:**

```

Select Device: 1. Smart Light 2. Smart Fan
1
Smart Light is turned ON.
Smart Light is turned OFF.

```

### **12.c) Restaurant Ordering System**

**Aim:**

**To develop a Restaurant Ordering System that allows users to order Vegetarian or Non-Vegetarian meals using abstraction.**

**CODE:**

```

import java.util.Scanner;
interface FoodOrder {
    void orderFood();
}

```

```

class VegFood implements FoodOrder {
    public void orderFood() {
        System.out.println("You ordered a Vegetarian Meal!");
    }
}
class NonVegFood implements FoodOrder {
    public void orderFood() {
        System.out.println("You ordered a Non-Vegetarian Meal!");
    }
}
public class S {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Select Food Type: 1. Vegetarian 2. Non-
        Vegetarian");
        int choice = scanner.nextInt();
        FoodOrder order;
        if (choice == 1) {
            order = new VegFood();
        } else {
            order = new NonVegFood();
        }
        order.orderFood();
        scanner.close();
    }
}

```

**OUTPUT:**

```

Select Food Type: 1. Vegetarian 2. Non-Vegetarian
2
You ordered a Non-Vegetarian Meal!

```

## **12.d) Flight Booking System**

**Aim:**

**To implement a Flight Booking System that allows users to book either Domestic or International flights using abstraction.**

**CODE:**

```
import java.util.Scanner;
interface Flight {
    void bookTicket(String passengerName, String destination);
}
class EconomyClass implements Flight {
    public void bookTicket(String passengerName, String destination) {
        int price = calculatePrice(destination, "Economy");
        System.out.println(" Booking Confirmed for " + passengerName);
        System.out.println("Class: Economy | Destination: " + destination);
        System.out.println("Total Fare: $" + price);
    }
    private int calculatePrice(String destination, String type) {
        return (destination.length() * 20) + (type.equals("Economy") ? 100 :
            200);
    }
}
class BusinessClass implements Flight {
    public void bookTicket(String passengerName, String destination) {
        int price = calculatePrice(destination, "Business");
        System.out.println(" Booking Confirmed for " + passengerName);
        System.out.println("Class: Business | Destination: " + destination);
        System.out.println("Total Fare: $" + price);
    }
    private int calculatePrice(String destination, String type) {
        return (destination.length() * 20) + (type.equals("Economy") ? 100 :
            200);
    }
}
public class FB {
```

```
public static void main(String[] args) {  
    Scanner scanner = new Scanner(System.in);  
    System.out.print("Enter Passenger Name: ");  
    String passengerName = scanner.nextLine();  
    System.out.print("Enter Destination: ");  
    String destination = scanner.nextLine();  
    System.out.println("Select Class: 1. Economy 2. Business");  
    int choice = scanner.nextInt();  
    Flight flight;  
    if (choice == 1) {  
        flight = new EconomyClass();  
    } else {  
        flight = new BusinessClass();  
    }  
    flight.bookTicket(passengerName, destination);  
    scanner.close();  
}  
}
```

**OUTPUT:**

```
Booking Confirmed for John Doe  
Class: Economy | Destination: Paris  
Total Fare: $200
```

## **ABSTRACT CLASS PROGRAMS-13**

### **13.a) Online Payment System**

**AIM:**

**To implement an Online Payment System using abstraction, demonstrating different payment methods like Credit Card and UPI transactions.**

**CODE:**

```
abstract class OnlinePayment {  
    abstract void makePayment(double amount);  
    public void transactionSuccessful() {  
        System.out.println("Transaction Completed Successfully!");  
    }  
}  
  
class CreditCardPayment extends OnlinePayment {  
    void makePayment(double amount) {  
        System.out.println("Paid Rs. " + amount + " using Credit Card.");  
        transactionSuccessful();  
    }  
}  
  
class UpiPayment extends OnlinePayment {  
    void makePayment(double amount) {  
        System.out.println("Paid Rs. " + amount + " using UPI.");  
        transactionSuccessful();  
    }  
}  
  
public class P{  
    public static void main(String[] args) {  
        OnlinePayment payment1 = new CreditCardPayment();  
        payment1.makePayment(1000.50);  
        OnlinePayment payment2 = new UpiPayment();  
        payment2.makePayment(500.75);  
    }  
}
```

## OUTPUT:

```
Paid Rs. 1000.5 using Credit Card.  
Transaction Completed Successfully!  
Paid Rs. 500.75 using UPI.  
Transaction Completed Successfully!
```

### 13.b) Shopping Cart System

#### AIM:

To implement a Shopping Cart System using abstraction where different products like Electronics and Clothing can be added to a cart.

#### CODE:

```
abstract class Product {  
    abstract void addToCart();  
    public void showDiscount() {  
        System.out.println("You get a 10% discount on this product.");  
    }  
}  
  
class Electronics extends Product {  
    void addToCart() {  
        System.out.println("Laptop added to cart.");  
        showDiscount();  
    }  
}  
  
class Clothing extends Product {  
    void addToCart() {  
        System.out.println("T-shirt added to cart.");  
        showDiscount();  
    }  
}  
  
public class S {  
    public static void main(String[] args) {  
        Product item1 = new Electronics();  
    }  
}
```

```
    item1.addToCart();  
    Product item2 = new Clothing();  
    item2.addToCart();  
}  
}
```

**OUTPUT:**

```
Laptop added to cart.  
You get a 10% discount on this product.  
T-shirt added to cart.  
You get a 10% discount on this product.
```

### **13.c) Vehicle License System**

**AIM:**

To create a Vehicle License System using abstraction, where users can apply for different types of licenses like Car License and Bike License.

**CODE:**

```
abstract class License {  
    abstract void issueLicense();  
    public void verifyDocuments() {  
        System.out.println("Documents verified successfully.");  
    }  
}  
  
class CarLicense extends License {  
    void issueLicense() {  
        System.out.println("Car Driving License Issued.");  
        verifyDocuments();  
    }  
}  
  
class BikeLicense extends License {  
    void issueLicense() {  
        System.out.println("Bike Driving License Issued.");  
        verifyDocuments();  
    }  
}
```



```

    }
}
public class S {
    public static void main(String[] args) {
        License license1 = new CarLicense();
        license1.issueLicense();
        License license2 = new BikeLicense();
        license2.issueLicense();
    }
}

```

**OUTPUT:**

```

Car Driving License Issued.
Documents verified successfully.
Bike Driving License Issued.
Documents verified successfully.

```

### **13.d) Hotel Booking System**

**AIM:**

To create a Hotel Booking System using abstraction, where users can book different types of rooms like Deluxe Room and Standard Room.

**CODE:**

```

abstract class HotelRoom {
    abstract void bookRoom();
    public void provideAmenities() {
        System.out.println("WiFi and breakfast included.");
    }
}

class DeluxeRoom extends HotelRoom {
    void bookRoom() {
        System.out.println("Deluxe Room booked with sea view.");
        provideAmenities();
    }
}

class StandardRoom extends HotelRoom {
    void bookRoom() {

```

```

        System.out.println("Standard Room booked with city view.");
        provideAmenities();
    }
}
public class S {
    public static void main(String[] args) {
        HotelRoom room1 = new DeluxeRoom();
        room1.bookRoom();
        HotelRoom room2 = new StandardRoom();
        room2.bookRoom();
    }
}

```

**OUTPUT:**

```

Deluxe Room booked with sea view.
WiFi and breakfast included.
Standard Room booked with city view.
WiFi and breakfast included.

```

## **ENCAPSULATION PROGRAMS -14**

### **14.a) Weather Report System**

**AIM:**

**To implement an Encapsulation-based Weather Report System that securely stores and retrieves temperature in Celsius and Fahrenheit.**

**CODE:**

```

class WeatherReport {
    private double temperatureCelsius;
    private double temperatureFahrenheit;
    public double getTemperatureCelsius() {
        return temperatureCelsius;
    }
    public double getTemperatureFahrenheit() {
        return temperatureFahrenheit;
    }
}

```

```

    public void updateTemperature(double tempCelsius) {
        this.temperatureCelsius = tempCelsius;
        this.temperatureFahrenheit = (tempCelsius * 9 / 5) + 32; // Auto-
        conversion
    }
}

public class W {
    public static void main(String[] args) {
        WeatherReport report = new WeatherReport();
        report.updateTemperature(25);
        System.out.println("Temperature in Celsius: " +
            report.getTemperatureCelsius());
        System.out.println("Temperature in Fahrenheit: " +
            report.getTemperatureFahrenheit());
    }
}

```

**OUTPUT:**

```

Temperature in Celsius: 25.0
Temperature in Fahrenheit: 77.0

```

#### **14.b) Employee Work Hours Tracker**

**AIM:**

**To develop an Encapsulation-based Employee Work Hours Tracker that records work hours and calculates salary dynamically.**

**CODE:**

```

class Employee {
    private int workHours;
    private double salary;
    private final double PAY_RATE = 500;
    public int getWorkHours() {
        return workHours;
    }
    public double getSalary() {
        return salary;
    }
}

```

```

    }
    public void addWorkHours(int hours) {
        this.workHours += hours;
        this.salary = this.workHours * PAY_RATE;
    }
}

public class E{
    public static void main(String[] args) {
        Employee emp = new Employee();
        emp.addWorkHours(8);
        System.out.println("Total Work Hours: " + emp.getWorkHours());
        System.out.println("Total Salary: Rs. " + emp.getSalary());
    }
}

```

**OUTPUT:**

```

Total Work Hours: 8
Total Salary: Rs. 4000.0

```

#### **14.c) Online Ticket Booking System**

**AIM:**

**To create an Encapsulation-based Online Ticket Booking System where users book tickets, and seat availability is updated accordingly.**

**CODE:**

```

import java.util.Scanner;

class Ticket {
    private String ticketId;
    private String customerName;
    private String destination;
    private double price;
    private boolean isBooked;

    public Ticket(String ticketId, String customerName, String destination,

```

```
        double price) {
    this.ticketId = ticketId;
    this.customerName = customerName;
    this.destination = destination;
    this.price = price;
    this.isBooked = false;
}
public void bookTicket() {
    if (!isBooked) {
        isBooked = true;
        System.out.println("Ticket booked successfully.");
    } else {
        System.out.println("Ticket already booked.");
    }
}
public void cancelTicket() {
    if (isBooked) {
        isBooked = false;
        System.out.println("Ticket canceled successfully.");
    } else {
        System.out.println("Ticket is not booked yet.");
    }
}
public void checkStatus() {
    if (isBooked) {
        System.out.println("Ticket is booked.");
    } else {
        System.out.println("Ticket is available.");
    }
}
public String getTicketId() {
    return ticketId;
}
```

```

    public double getPrice() {
        return price;
    }
}

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

        Ticket ticket1 = new Ticket("T123", "Alice", "New York", 150.00);

        // Main loop for ticket booking operations
        while (true) {
            System.out.println("\n--- Ticket Booking Menu ---");
            System.out.println("1. Check Ticket Status");
            System.out.println("2. Book Ticket");
            System.out.println("3. Cancel Ticket");
            System.out.println("4. Exit");
            System.out.print("Select an option: ");
            int option = scanner.nextInt();
            scanner.nextLine(); // Consume newline character after int input

            switch (option) {
                case 1:
                    ticket1.checkStatus();
                    break;

                case 2:
                    ticket1.bookTicket();
                    break;

                case 3:
                    ticket1.cancelTicket();

```

```

        break;

    case 4:
        System.out.println("Exiting Ticket Booking System. Goodbye!");
        scanner.close();
        return;

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

```

#### OUTPUT:

```

--- Ticket Booking Menu ---
1. Check Ticket Status
2. Book Ticket
3. Cancel Ticket
4. Exit
Select an option: 1
Ticket is available.

```

#### 14.d) Employee Leave Management System

##### AIM:

To implement an Encapsulation-based Employee Leave Management System where employees request leaves, and available leave balance is updated.

##### ALGORITHM:

1. Create an EmployeeLeave class with private attributes:
  - totalLeaves (total available leaves)
  - usedLeaves (number of used leaves)
2. Provide getter methods getAvailableLeaves() and getUsedLeaves() to check leave balance.
3. Define applyLeave(int days) method to update leave count.
  - Ensure the requested leave is within the available limit.

- Deduct leaves accordingly.
- 4. In main(), create an object of EmployeeLeave, apply leaves, and display the updated leave balance.

**CODE:**

```
import java.util.Scanner;

class Employee {
    private String name;
    private int availableLeaves;
    private int totalLeaves;
    private int usedLeaves;
    public Employee(String name, int totalLeaves) {
        this.name = name;
        this.totalLeaves = totalLeaves;
        this.availableLeaves = totalLeaves; // Initially, all leaves are available
        this.usedLeaves = 0;
    }
    public void requestLeave(int days) {
        if (days <= availableLeaves && days > 0) {
            availableLeaves -= days;
            usedLeaves += days;
            System.out.println(name + " has requested " + days + " days off.");
        } else {
            System.out.println("Insufficient leave balance or invalid number of days.");
        }
    }
    public void cancelLeave(int days) {
        if (days <= usedLeaves && days > 0) {
            availableLeaves += days;
            usedLeaves -= days;
            System.out.println(name + " has canceled " + days + " days of leave.");
        } else {
            System.out.println("Invalid number of days to cancel.");
        }
    }
}
```



```

    }
}
public void checkLeaveBalance() {
    System.out.println(name + " has " + availableLeaves + " leave days
        remaining.");
}
public String getName() {
    return name;
}

public int getUsedLeaves() {
    return usedLeaves;
}
}
public class L {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Employee emp1 = new Employee("John Doe", 20); // 20 total leaves
        while (true) {
            System.out.println("\n--- Leave Management Menu ---");
            System.out.println("1. Check Leave Balance");
            System.out.println("2. Request Leave");
            System.out.println("3. Cancel Leave");
            System.out.println("4. Exit");
            System.out.print("Select an option: ");
            int option = scanner.nextInt();

            switch (option) {
                case 1:
                    emp1.checkLeaveBalance();
                    break;
                case 2:
                    System.out.print("Enter number of leave days: ");

```

```

        int leaveDays = scanner.nextInt();
        emp1.requestLeave(leaveDays);
        break;
    case 3:
        System.out.print("Enter number of leave days to cancel: ");
        leaveDays = scanner.nextInt();
        emp1.cancelLeave(leaveDays);
        break;
    case 4:
        System.out.println("Exiting Leave Management System");
        scanner.close();
        return;
    default:
        System.out.println("Invalid option. Please try again.");
    }
}
}
}
}

```

#### OUTPUT:

```

--- Leave Management Menu ---
1. Check Leave Balance
2. Request Leave
3. Cancel Leave
4. Exit
Select an option: 2
Enter number of leave days: 5
John Doe has requested 5 days off.

```

## PACKAGES PROGRAMS-15

**15a) Create a package named mathoperations and define a class Addition with a method to add two numbers.**

**Aim: To create a package mathoperations and define a class Addition with a method to add two numbers.**

**CODE:**

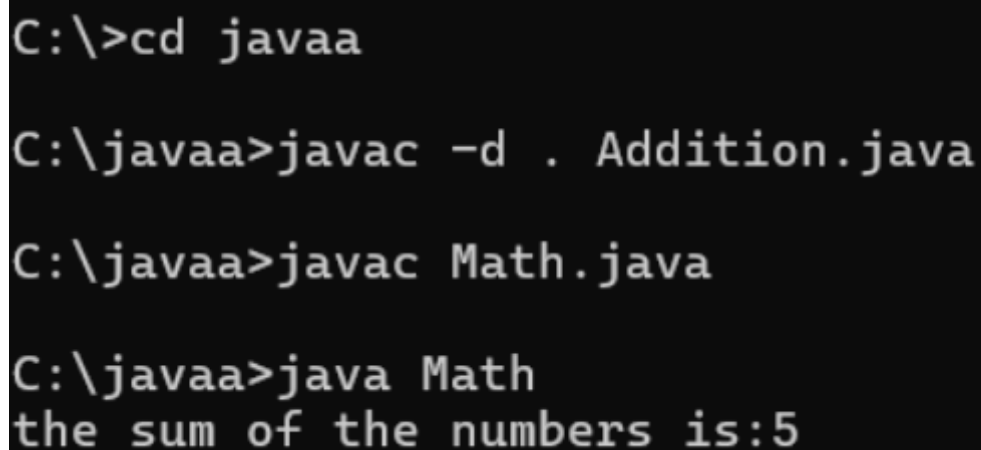
**Package code:**

```
package mathoperations;  
public class Addition{ public int add(int n1,int n2){  
return n1+n2;  
}  
}
```

**Main Code:**

```
import mathoperations.Addition;  
public class Math  
{  
public static void main(String[] args)  
{ Addition a=new Addition();  
int sum=a.add(2,3);  
System.out.println("the sum of the numbers is:"+sum);  
}  
}
```

**OUTPUT:**



```
C:\>cd javaa  
C:\javaa>javac -d . Addition.java  
C:\javaa>javac Math.java  
C:\javaa>java Math  
the sum of the numbers is:5
```

**15b) .Using import and static import i. Create a package utilities with a class MathUtils that has static methods for max, min, and square. ii. Use import and static import in a different package to demonstrate their usage.**

**Aim:**

**To demonstrate the use of import and static import in Java by:**

- 1. Creating a package utilities with static methods max, min, and square in a class MathUtils.**
- 2. Using these methods in another package using both normal and static import.**

**CODE:**

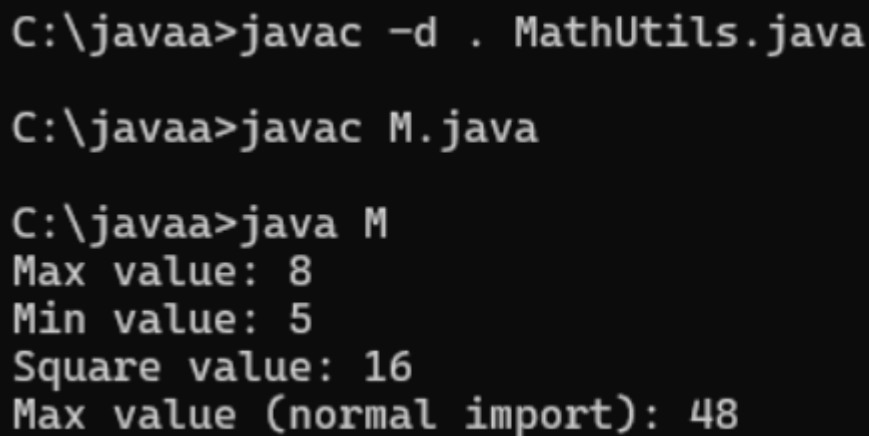
```
package utilities;  
public class MathUtils {  
    public static int max(int a, int b) {  
        return (a > b) ? a : b;  
    }  
    public static int min(int a, int b) {  
        return (a < b) ? a : b;  
    }  
    public static int square(int a) {  
        return a * a;  
    }  
}
```

**Main code:**

```
import utilities.MathUtils;  
import static utilities.MathUtils.max;  
import static utilities.MathUtils.min;  
import static utilities.MathUtils.square;  
public class M {  
    public static void main(String[] args) {  
        int maxValue = max(5, 8);  
        int minValue = min(5, 8);
```

```
int squareValue = square(4);
System.out.println("Max value: " + maxValue);
System.out.println("Min value: " + minValue);
System.out.println("Square value: " + squareValue);
int maxValueNormalImport = MathUtils.max(23, 48);
System.out.println("Max value (normal import): " +
    maxValueNormalImport);
}
}
```

**OUTPUT:**



```
C:\javaa>javac -d . MathUtils.java

C:\javaa>javac M.java

C:\javaa>java M
Max value: 8
Min value: 5
Square value: 16
Max value (normal import): 48
```

**15c) Sub-Packages** a. Create a package `company.hr` with a class `Employee` and another package `company.finance` with a class `Payroll`. b. Use the `Employee` class inside `Payroll` to calculate the salary of an employee.

**Aim:**

To demonstrate the concept of sub-packages in Java by:

1. Creating a package `company.hr` with a class `Employee`.
2. Creating another package `company.finance` with a class `Payroll`.
3. Using the `Employee` class in `Payroll` to calculate the salary.

## **CODE:**

```
Package company.hr;  
public class Employee {  
private String name;  
private double baseSalary;  
private int hoursWorked;  
public Employee(String name, double baseSalary, int hoursWorked) {  
this.name = name;  
this.baseSalary = baseSalary;  
this.hoursWorked = hoursWorked;  
}  
public String getName() {  
return name;  
}  
public double getBaseSalary() {  
return baseSalary;  
}  
public int getHoursWorked() {  
return hoursWorked;  
}  
}
```

## **Package code:**

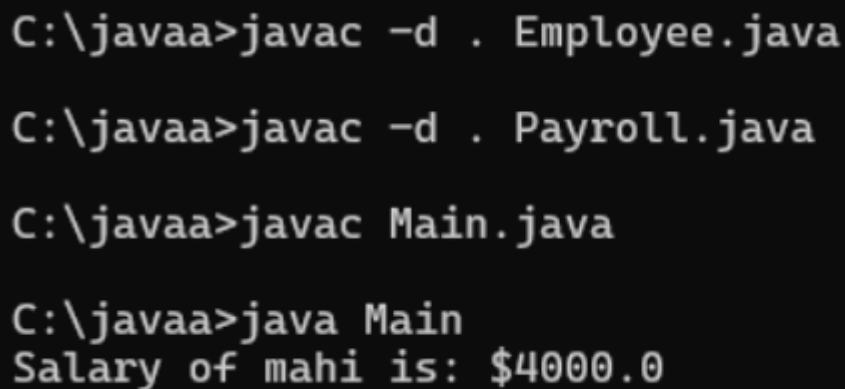
```
package company.finance;  
import company.hr.Employee;  
  
public class Payroll {  
public double calculateSalary(Employee employee) {  
double baseSalary = employee.getBaseSalary();  
int hoursWorked = employee.getHoursWorked();  
double hourlyRate = baseSalary / 160; // Assuming 160 working hours in  
a month
```

```
return hourlyRate * hoursWorked;
}
}
```

**Main Code:**

```
import company.hr.Employee;
import company.finance.Payroll;
public class Main {
    public static void main(String[] args) {
        Employee employee = new Employee("mahi", 4000.0, 160);
        Payroll payroll = new Payroll();
        double salary = payroll.calculateSalary(employee);
        System.out.println("Salary of " + employee.getName() + " is: $" +
            salary);}
}
```

**OUTPUT:**



```
C:\javaa>javac -d . Employee.java
C:\javaa>javac -d . Payroll.java
C:\javaa>javac Main.java
C:\javaa>java Main
Salary of mahi is: $4000.0
```

**15d) Custom Exception with Packages** a. Create a package exceptions and define a custom exception InvalidAgeException. b. Create another package validation with a class AgeValidator that throws the exception if age is below 18. c. Test the exception handling in a main class.

**Aim:**

To create a custom exception using Java packages:

1. Create a package exceptions with a custom exception class InvalidAgeException.
2. Create another package validation with a class AgeValidator that validates the age and throws InvalidAgeException if the age is below 18.

**3. Create a main class to test the custom exception handling.**

**CODE:**

**package code:**

**package exceptions;**

**public class InvalidAgeException extends Exception {**

**public InvalidAgeException(String message) {**

**super(message);**

**}**

**}**

**Package code:**

**package validation;**

**import exceptions.InvalidAgeException;**

**public class AgeValidator {**

**public void validateAge(int age) throws InvalidAgeException {**

**if (age < 18) {**

**throw new InvalidAgeException("Age must be 18 or above.");**

**}**

**System.out.println("Age is valid.");**

**}**

**}**

**Main code:**

**import validation.AgeValidator;**

**import exceptions.InvalidAgeException;**

**public class Main {**

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

**AgeValidator validator = new AgeValidator();**

**try {**

**validator.validateAge(15); // This will throw an exception**

**} catch (InvalidAgeException e) {**

**System.out.println("Error: " + e.getMessage());**

**} try { validator.validateAge(20);**

**} catch (InvalidAgeException e) { System.out.println("Error: " +**



```
        e.getMessage());  
    }  
}
```

**OUTPUT:**

```
C:\javaa>javac -d . InvalidAgeException.java  
C:\javaa>javac -d . AgeValidator.java  
C:\javaa>javac Main.java  
C:\javaa>java Main  
Error: Age must be 18 or above.  
Age is valid.
```

## **EXCEPTION HANDLING PROGRAMS-16**

### **16.a) Online Exam System – Invalid Option Exception**

**AIM:** To write a Java program that simulates an online exam system using exception handling to check for invalid answer choices.

**CODE:**

```
import java.util.Scanner;  
  
class InvalidOptionException extends Exception {  
    public InvalidOptionException(String message) {  
        super(message);  
    }  
}  
  
public class OnlineExam1 {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);
```

```

String[] options = {"A", "B", "C", "D"};
try {
    System.out.println("Q: What is the capital of France?");
    System.out.println("A) Berlin\nB) Madrid\nC) Paris\nD) Rome");
    System.out.print("Enter your option (A/B/C/D): ");
    String input = scanner.next().toUpperCase();
        boolean isValid = false;
    for (String opt : options) {
        if (opt.equals(input)) {
            isValid = true;
            break;
        }
    }

    if (!isValid) {
        throw new InvalidOptionException("You selected an invalid
option: " + input);
    }
    if (input.equals("C")) {
        System.out.println("Correct Answer!");
    } else {
        System.out.println("Wrong Answer.");
    }

} catch (InvalidOptionException e) {
    System.out.println("Error: " + e.getMessage());
} finally {
    scanner.close();
}
}
} OUTPUT:

```

```
C:\Users\HP\Desktop>java OnlineExam1
Q: What is the capital of France?
A) Berlin
B) Madrid
C) Paris
D) Rome
Enter your option (A/B/C/D): w
Error: You selected an invalid option: W

C:\Users\HP\Desktop>java OnlineExam1
Q: What is the capital of France?
A) Berlin
B) Madrid
C) Paris
D) Rome
Enter your option (A/B/C/D): c
Correct Answer!

C:\Users\HP\Desktop>java OnlineExam1
Q: What is the capital of France?
A) Berlin
B) Madrid
C) Paris
D) Rome
Enter your option (A/B/C/D): a
Wrong Answer.
```

## 16.b) Handling FileNotFoundException

### AIM:

To write a Java program that attempts to read a file and handles **FileNotFoundException** if the file is not found.

### CODE:

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class FH {
    public static void main(String[] args) {
        try {
            File file = new File("FH.java");
            Scanner scanner = new Scanner(file); // may throw
            FileNotFoundException
            while (scanner.hasNextLine()) {
                System.out.println(scanner.nextLine());
            }
            scanner.close();
        } catch (FileNotFoundException e) {
            System.out.println("File not found. Please check the file name or
```

```
        path.");  
    }  
}  
}
```

**OUTPUT:**

```
C:\Users\HP\Desktop>javac FH.java
```

```
C:\Users\HP\Desktop>java FH  
File not found. Please check the file name or path.
```

### **16.c) ATM Withdrawal – InsufficientFundsException**

**AIM:**

To simulate an ATM withdrawal system that checks for sufficient balance and throws a custom exception when funds are insufficient.

**CODE:**

```
import java.util.Scanner;  
  
class InsufficientFundsException extends Exception {  
    public InsufficientFundsException(String message) {  
        super(message);  
    }  
}  
  
public class ATM {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
        double balance = 2500.00;  
        try {  
            System.out.println("Current balance: ₹" + balance);  
            System.out.print("Enter withdrawal amount: ₹");  
            double amount = scanner.nextDouble();  
  
            if (amount <= 0) {  
                throw new IllegalArgumentException("Amount must be greater  
than zero.");  
            }  
        }  
    }  
}
```

```

    }
    if (amount > balance) {
        throw new InsufficientFundsException("Not enough balance to
withdraw ₹" + amount);
    }
    balance -= amount;
    System.out.println("Withdrawal successful! Remaining balance: ₹" +
balance);
} catch (InsufficientFundsException | IllegalArgumentException e) {
    System.out.println("Transaction Failed: " + e.getMessage());
} finally {
    scanner.close();
}
}
}

```

**OUTPUT:**

```

C:\Users\HP\Desktop>javac ATM.java

C:\Users\HP\Desktop>java ATM
Current balance: ?2500.0
Enter withdrawal amount: ?3000
Transaction Failed: Not enough balance to withdraw ?3000.0

C:\Users\HP\Desktop>java ATM
Current balance: ?2500.0
Enter withdrawal amount: ?600
Withdrawal successful! Remaining balance: ?1900.0

```

#### **16.d) Login – MaxLoginAttemptsExceededException**

**AIM:**

**To implement a secure login system in Java that locks a user out after a maximum number of failed attempts using a custom exception.**

```

CODE: import java.util.Scanner;
class MaxLoginAttemptsExceededException extends Exception {
    public MaxLoginAttemptsExceededException(String message) {
        super(message);
    }
}

public class Login {
    public static void main(String[] args) {
        final String USERNAME = "admin";
        final String PASSWORD = "123";
        int attempts = 0;
        final int MAX_ATTEMPTS = 3;
        Scanner scanner = new Scanner(System.in);
        try {
            while (attempts < MAX_ATTEMPTS) {
                System.out.print("Enter username: ");
                String user = scanner.nextLine();
                System.out.print("Enter password: ");
                String pass = scanner.nextLine();
                if (user.equals(USERNAME) && pass.equals(PASSWORD)) {
                    System.out.println("Login successful!");
                    return;
                } else {
                    attempts++;
                    System.out.println("Invalid credentials. Attempt " + attempts + "
of " + MAX_ATTEMPTS);
                }
            }
            throw new MaxLoginAttemptsExceededException("Too many failed
login attempts. Account locked.");
        } catch (MaxLoginAttemptsExceededException e) {
            System.out.println("Access Denied: " + e.getMessage());
        } finally {
            scanner.close();
        }
    }

```

```
}  
}  
}
```

**OUTPUT:**

```
C:\Users\HP\Desktop>javac Login.java  
  
C:\Users\HP\Desktop>java Login  
Enter username: admin  
Enter password: 778  
Invalid credentials. Attempt 1 of 3  
Enter username: 88  
Enter password: 99  
Invalid credentials. Attempt 2 of 3  
Enter username: h  
Enter password: 8  
Invalid credentials. Attempt 3 of 3  
Access Denied: Too many failed login attempts. Account locked.
```

## **FILE HANDLING PROGRAMS -17**

### **17.a) Visitor Log System (Appending Names to a File)**

**AIM:**

To develop a Java program that maintains a visitor log by appending visitor names to a file.

**ALGORITHM:**

1. Start the program.
2. Create a Scanner object to take input from the user.
3. Prompt the user to enter the visitor's name.
4. Open the file visitors.txt in append mode using FileWriter.
5. Write the visitor's name to the file.
6. Close the file after writing.
7. Display a success message.
8. Handle any exceptions that occur while writing to the file.
9. End the program.

**CODE:**

```

import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;
public class V {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter visitor name: ");
        String name = scanner.nextLine();
        try {
            FileWriter fw = new FileWriter("visitors.txt", true); // append mode
            fw.write(name + "\n");
            fw.close();
            System.out.println("Visitor logged successfully.");
        } catch (IOException e) {
            System.out.println("An error occurred while logging the visitor.");
        }
        scanner.close();
    }
}

```

**OUTPUT:**

```

C:\Users\HP\Desktop>javac V.java

C:\Users\HP\Desktop>java V
Enter visitor name: k.harichandana
Visitor logged successfully.

```

### **17.b) Task Reminder System (Reading Tasks from a File)**

**AIM:**

To create a Java program that reads and displays tasks from a file, acting as a task reminder system.

**CODE:**

```

import java.io.File;
import java.io.FileNotFoundException;

```



```
import java.util.Scanner;
public class T{
    public static void main(String[] args) throws FileNotFoundException {
        File file = new File("tasks.txt");
        Scanner scanner = new Scanner(file);
        System.out.println("Today's Tasks:");
        while (scanner.hasNextLine()) {
            System.out.println("- " + scanner.nextLine());
        }
        scanner.close();
    }
}
```

**OUTPUT:**

```
C:\Users\HP\Desktop>javac T.java

C:\Users\HP\Desktop>java T
Today's Tasks:
- 1. Wake up at 6:30 AM
- 2. Morning exercise for 30 minutes
- 3. Have breakfast at 7:30 AM
- 4. Attend college classes from 9:00 AM to 3:00 PM
- 5. Lunch break at 1:00 PM
- 6. Complete assignments from 4:00 PM to 6:00 PM
- 7. Evening walk at 6:30 PM
- 8. Revise class notes from 7:30 PM to 8:30 PM
- 9. Dinner at 9:00 PM
- 10. Relax and watch TV from 9:30 PM to 10:30 PM
- 11. Sleep at 11:00 PM
```

### **17.c) Student Feedback Collector (Write and Read Feedback)**

**AIM:**

**To implement a Java program that collects feedback from students, writes it to a file, and then reads all stored feedback.**

**CODE:**

```
import java.io.*;
import java.util.Scanner;

public class F {
    public static void main(String[] args) throws IOException {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter your feedback: ");
        String feedback = scanner.nextLine();

        FileWriter fw = new FileWriter("feedback.txt", true);
        fw.write(feedback + "\n");
        fw.close();
        System.out.println("Feedback submitted.");

        System.out.println("\nAll Feedbacks:");
        BufferedReader br = new BufferedReader(new
            FileReader("feedback.txt"));
        String line;
        while ((line = br.readLine()) != null) {
            System.out.println("- " + line);
        }
        br.close();

        scanner.close();
    }
}
```

**OUTPUT:**

```
C:\Users\HP\Desktop>javac F.java
```

```
C:\Users\HP\Desktop>java F  
Enter your feedback: It was good!  
Feedback submitted.
```

```
All Feedbacks:  
- It was good!
```

#### 17.d) Online Product Rating Saver

**AIM:**

To design a Java program that allows users to submit product ratings and saves them in a file.

**CODE:**

```
import java.io.FileWriter;  
import java.util.Scanner;  
public class P {  
    public static void main(String[] args) Exception {  
        Scanner scanner = new Scanner(System.in);  
        System.out.print("Enter product name: ");  
        String product = scanner.nextLine();  
        System.out.print("Enter your rating (1-5): ");  
        int rating = scanner.nextInt();  
        FileWriter fw = new FileWriter("ratings.txt", true);  
        fw.write(product + " - " + rating + " stars\n");  
        fw.close();  
        System.out.println("Thank you! Your rating has been saved.");  
        scanner.close();  
    }  
}
```

**OUTPUT:**

```
C:\Users\HP\Desktop>javac P.java
```

```
C:\Users\HP\Desktop>java P
```

```
Enter product name: Sony TV
```

```
Enter your rating (1-5): 4
```

```
Thank you! Your rating has been saved.
```

-----THANK YOU-----