Assignment-4

Student Name: Unnati Srivastava **UID: 22BCS13475**

Branch: BE-CSE **Section/Group: IOT-610/B**

Semester: 6th **Date Of Performance: 23/2/25**

Subject Name: Project Based Learning in Java **Subject Code: 22CSH-359**

Easy:

QUES: Employee Management System

Write a Java program to implement an ArrayList that stores employee details (ID, Name, and Salary). Allow users to:

Add employees

Update employee details

Remove employees

Search for employees

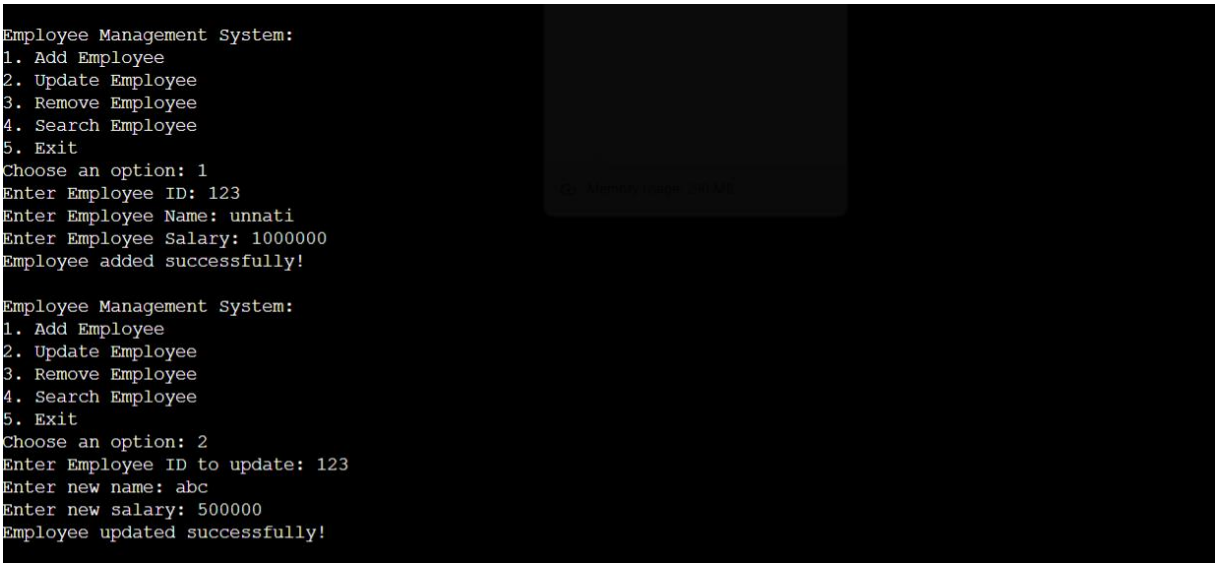
Solution:

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

class Employee {
    private int id;
    private String name;
    private double salary;

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

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }
```

```
    public double getSalary() {
        return salary;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

    @Override
    public String toString() {
        return "Employee [ID=" + id + ", Name=" + name + ", Salary=" + salary + "]";
    }
}

public class EmployeeManagement {
    private static ArrayList<Employee> employees = new ArrayList<>();
    private static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        while (true) {
            System.out.println("\nEmployee Management System:");
            System.out.println("1. Add Employee");
            System.out.println("2. Update Employee");
            System.out.println("3. Remove Employee");
            System.out.println("4. Search Employee");
            System.out.println("5. Exit");
            System.out.print("Choose an option: ");

            int choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    addEmployee();
                    break;
                case 2:
                    updateEmployee();
                    break;
                case 3:
                    removeEmployee();
                    break;
                case 4:
                    searchEmployee();
                    break;
```

```
            case 5:
                System.out.println("Exiting...");
                return;
            default:
                System.out.println("Invalid choice. Try again.");
        }
    }
}

private static void addEmployee() {
    System.out.print("Enter Employee ID: ");
    int id = scanner.nextInt();
    scanner.nextLine(); // Consume newline
    System.out.print("Enter Employee Name: ");
    String name = scanner.nextLine();
    System.out.print("Enter Employee Salary: ");
    double salary = scanner.nextDouble();

    employees.add(new Employee(id, name, salary));
    System.out.println("Employee added successfully!");
}

private static void updateEmployee() {
    System.out.print("Enter Employee ID to update: ");
    int id = scanner.nextInt();
    for (Employee emp : employees) {
        if (emp.getId() == id) {
            scanner.nextLine(); // Consume newline
            System.out.print("Enter new name: ");
            String name = scanner.nextLine();
            System.out.print("Enter new salary: ");
            double salary = scanner.nextDouble();
            emp.setName(name);
            emp.setSalary(salary);
            System.out.println("Employee updated successfully!");
            return;
        }
    }
    System.out.println("Employee not found.");
}

private static void removeEmployee() {
    System.out.print("Enter Employee ID to remove: ");
    int id = scanner.nextInt();
    for (Employee emp : employees) {
        if (emp.getId() == id) {
```

```
                employees.remove(emp);
                System.out.println("Employee removed successfully!");
                return;
            }
        }
        System.out.println("Employee not found.");
    }

    private static void searchEmployee() {
        System.out.print("Enter Employee ID to search: ");
        int id = scanner.nextInt();
        for (Employee emp : employees) {
            if (emp.getId() == id) {
                System.out.println(emp);
                return;
            }
        }
        System.out.println("Employee not found.");
    }
}
```



Medium:

QUES: Card Collection System

Create a program to collect and store all the cards (e.g., playing cards) and assist users in finding all the cards of a given symbol using the Collection interface.

Solution:

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Scanner;

class Card {
    private String symbol;
    private String value;

    public Card(String symbol, String value) {
        this.symbol = symbol;
        this.value = value;
    }

    public String getSymbol() {
        return symbol;
    }

    public String getValue() {
        return value;
    }

    @Override
    public String toString() {
        return value + " of " + symbol;
    }
}

public class CardCollection {
    private static Map<String, List<Card>> cardMap = new HashMap<>();
    private static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        while (true) {
            System.out.println("\nCard Collection System:");
            System.out.println("1. Add Card");
            System.out.println("2. Search Cards by Symbol");
            System.out.println("3. Exit");
```

```
        System.out.print("Choose an option: ");

        int choice = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        switch (choice) {
            case 1:
                addCard();
                break;
            case 2:
                searchCardsBySymbol();
                break;
            case 3:
                System.out.println("Exiting...");
                return;
            default:
                System.out.println("Invalid choice. Try again.");
        }
    }
}

private static void addCard() {
    System.out.print("Enter card symbol (e.g., Hearts, Spades): ");
    String symbol = scanner.nextLine();
    System.out.print("Enter card value (e.g., Ace, 2, King): ");
    String value = scanner.nextLine();

    Card card = new Card(symbol, value);
    cardMap.computeIfAbsent(symbol, k -> new ArrayList<>()).add(card);
    System.out.println("Card added successfully!");
}

private static void searchCardsBySymbol() {
    System.out.print("Enter symbol to search for: ");
    String symbol = scanner.nextLine();

    List<Card> cards = cardMap.get(symbol);
    if (cards != null && !cards.isEmpty()) {
        System.out.println("Cards of symbol " + symbol + ":");
        for (Card card : cards) {
            System.out.println(card);
        }
    } else {
        System.out.println("No cards found for symbol: " + symbol);
    }
}
```

```
}
```

```
Card Collection System:
1. Add Card
2. Search Cards by Symbol
3. Exit
Choose an option: 1
Enter card symbol (e.g., Hearts, Spades): Spades
Enter card value (e.g., Ace, 2, King): Ace
Card added successfully!

Card Collection System:
1. Add Card
2. Search Cards by Symbol
3. Exit
Choose an option: 2
Enter symbol to search for: Spades
Cards of symbol Spades:
Ace of Spades

Card Collection System:
1. Add Card
2. Search Cards by Symbol
3. Exit
Choose an option: 3
Exiting...
```

Hard:

QUES: Ticket Booking System with Multithreading

Develop a ticket booking system with synchronized threads to ensure no double booking of seats. Use thread priorities to simulate VIP bookings being processed first.

Solution:

```
import java.util.HashSet;
import java.util.Set;


class TicketBookingSystem {
    private final Set<Integer> bookedSeats = new HashSet<>();


    public synchronized void bookSeat(int seatNumber, String customerType) {
        if (bookedSeats.add(seatNumber)) {
            System.out.println(customerType + " booked seat number: " + seatNumber);
        } else {
            System.out.println("Seat " + seatNumber + " is already booked.");
        }
    }
}
```

```
class BookingThread extends Thread {
    private final TicketBookingSystem system;
    private final int seatNumber;
    private final String customerType;

    public BookingThread(TicketBookingSystem system, int seatNumber, String
customerType) {
        this.system = system;
        this.seatNumber = seatNumber;
        this.customerType = customerType;
    }

    @Override
    public void run() {
        system.bookSeat(seatNumber, customerType);
    }
}

public class TicketBookingSystemMain {
    public static void main(String[] args) {
        TicketBookingSystem system = new TicketBookingSystem();

        new BookingThread(system, 1, "VIP").start();
        new BookingThread(system, 1, "Regular").start();
        new BookingThread(system, 2, "VIP").start();
        new BookingThread(system, 2, "Regular").start();
    }
}
```

```
VIP booked seat number: 1
Regular booked seat number: 2
Seat 2 is already booked.
Seat 1 is already booked.
```