Experiment 4

Student Name: Sumit Sharma **UID: 22BCS12980**
Branch: CSE **Section/Group:IOT-641-B**
Semester: 6th **DOP:26/02/2025**
Subject: Java Lab **Subject Code: 22CSH-359**

Aim: To develop a Java program to implement an ArrayList that stores employee details (ID, Name, and Salary). Allow users to add, update, remove, and searchemployees.

Objective: To develop a Java program that uses an ArrayList to store and manage employee details (ID, Name, and Salary). The program allows users to:

Add a new employee.
Update an existing employee's details.
Remove an employee from the list.
Search for an employee by ID. Display
all employees in the list.

Algorithm:

1. Start
2. Create an Employee class with attributes:
3. ID (int), Name (String), Salary (double).
4. Use an ArrayList to store multiple employees.
5. Display a menu with options:
6. Add an Employee
7. Update Employee Details
8. Remove an Employee
9. Search for an Employee
10. Display All Employees
11. Exit
12. Based on user input, perform the respective operation.
13. If updating or removing, search for the employee by ID.
14. Display confirmation messages after each operation.
15. Loop the menu until the user chooses to exit.
16. End

Code:

```
import java.util.ArrayList;
import java.util.Scanner;
class Employee {
    int id;
    String name;
    double salary;
    public Employee(int id, String name, double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }
    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Salary: " + salary;
    }
}
public class EmployeeManagement {
    static ArrayList<Employee> employees = new ArrayList<>();
    static Scanner scanner = new Scanner(System.in);

    public static void addEmployee() {
        System.out.print("Enter Employee ID: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // Consume newline
        System.out.print("Enter Name: ");
```

```
        String name = scanner.nextLine();
        System.out.print("Enter Salary: ");
        double salary = scanner.nextDouble();
        employees.add(new Employee(id, name, salary));
        System.out.println("Employee added successfully!");
    }
    public static void updateEmployee() {
        System.out.print("Enter Employee ID to update: ");
        int id = scanner.nextInt();
        scanner.nextLine();
        for (Employee emp : employees) {
            if (emp.id == id) {
                System.out.print("Enter new Name: ");
                emp.name = scanner.nextLine();
                System.out.print("Enter new Salary: ");
                emp.salary = scanner.nextDouble();
                System.out.println("Employee updated successfully!");
                return;
            }
        }
        System.out.println("Employee not found!");
    }
    public static void removeEmployee() {
        System.out.print("Enter Employee ID to remove: ");
        int id = scanner.nextInt();
        for (Employee emp : employees) {
            if (emp.id == id) {
```

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

public static void displayAllEmployees() {
    if (employees.isEmpty()) {
        System.out.println("No employees found!");
    } else {
        for (Employee emp : employees) {
            System.out.println(emp);
        }
    }
```

```
    }

    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. Display All Employees");

            System.out.println("6. Exit");

            System.out.print("Enter your choice: ");

            int choice = scanner.nextInt();

            switch (choice) {

                case 1: addEmployee(); break;

                case 2: updateEmployee(); break;

                case 3: removeEmployee(); break;

                case 4: searchEmployee(); break;

                case 5: displayAllEmployees(); break;

                case 6: System.out.println("Exiting..."); return;

                default: System.out.println("Invalid choice! Try again.");

            }

        }

    }

}
```

Output:

```
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Enter your choice: 1
Enter Employee ID: 34
Enter Name: Rajat Sharma
Enter Salary: 50000
Employee added successfully!

Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Enter your choice: 2
Enter Employee ID to update: 23
Enter new Name: Arpit Thakur
Enter new Salary: 50000
Employee updated successfully!

Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Enter your choice: 6
Exiting...


...Program finished with exit code 0
Press ENTER to exit console.
```

Question 2

Aim: Create a program to collect and store all the cards to assist the users in finding all the cards in a given symbol using Collection interface.

Objective: To develop a Java program using the Collection interface to store and manage playing cards. The program will help users:

Store cards in a collection.

Search for cards by a given symbol (e.g., Hearts, Spades). Display

all available cards in the collection.

Algorithm:

- Start
- Create a Card class with attributes:
- Symbol (String), Number (String).
- Use a Collection (ArrayList) to store multiple card objects.
- Display a menu with options:
- Add a card.
- Find all cards by symbol.
- Display all stored cards.
- Exit the program.
- Based on user input, perform the respective operation.
- If searching, iterate through the list and find all matching symbols.
- Display confirmation messages after each operation.
- Loop the menu until the user chooses to exit.
- End

Code:

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

class Card {
    String symbol;
    String rank;

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

    @Override
    public String toString() {
        return "Symbol: " + symbol + ", Rank: " + rank;
    }
}

public class CardCollectionSystem {
    static ArrayList<Card> cards = new ArrayList<>();
    static Scanner scanner = new Scanner(System.in);

    public static void addCard() {
        System.out.print("Enter Symbol (Hearts, Diamonds, etc.): ");
```

```
        String symbol = scanner.nextLine();
        System.out.print("Enter Rank (Ace, 2, King, etc.): ");
        String rank = scanner.nextLine();

        cards.add(new Card(symbol, rank));
        System.out.println("Card added successfully!");
    }

    public static void searchCard() {
        System.out.print("Enter Symbol to search: ");
        String symbol = scanner.nextLine();
        boolean found = false;
        for (Card card : cards) {
            if (card.symbol.equalsIgnoreCase(symbol)) {
                System.out.println(card);
                found = true;
            }
        }
        if (!found) {
            System.out.println("No cards found with the symbol " + symbol);
        }
    }

    public static void displayAllCards() {
        if (cards.isEmpty()) {
            System.out.println("No cards found!");
        } else {
            for (Card card : cards) {
                System.out.println(card);
            }
        }
    }

    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 by Symbol");
            System.out.println("3. Display All Cards");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline
            switch (choice) {
                case 1: addCard(); break;
                case 2: searchCard(); break;
                case 3: displayAllCards(); break;
                case 4: System.out.println("Exiting..."); return;
                default: System.out.println("Invalid choice! Try again.");
            }
        }
    }
}
```

Output:

```
Card Collection System
1. Add Card
2. Search by Symbol
3. Display All Cards
4. Exit
Enter your choice: 1
Enter Symbol (Hearts, Diamonds, etc.): Diamonds
Enter Rank (Ace, 2, King, etc.): 3
Card added successfully!

Card Collection System
1. Add Card
2. Search by Symbol
3. Display All Cards
4. Exit
Enter your choice: 1
Enter Symbol (Hearts, Diamonds, etc.): Hearts
Enter Rank (Ace, 2, King, etc.): king
Card added successfully!

Card Collection System
1. Add Card
2. Search by Symbol
3. Display All Cards
4. Exit
Enter your choice: 2
Enter Symbol to search: 3
No cards found with the symbol 3

Card Collection System
1. Add Card
2. Search by Symbol
3. Display All Cards
4. Exit
Enter your choice: 4
Exiting...
```

Question 3:

Aim: 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.

Objective:

Prevent Double Booking: Utilize thread synchronization mechanisms (such as locks or monitors) to ensure that no two threads can book the same seat simultaneously, maintaining seat availability consistency.

Simulate Priority-Based Booking: Implement thread priorities to process VIP seat bookings before regular bookings, mimicking real-world scenarios where higher-priority customers receive service first.

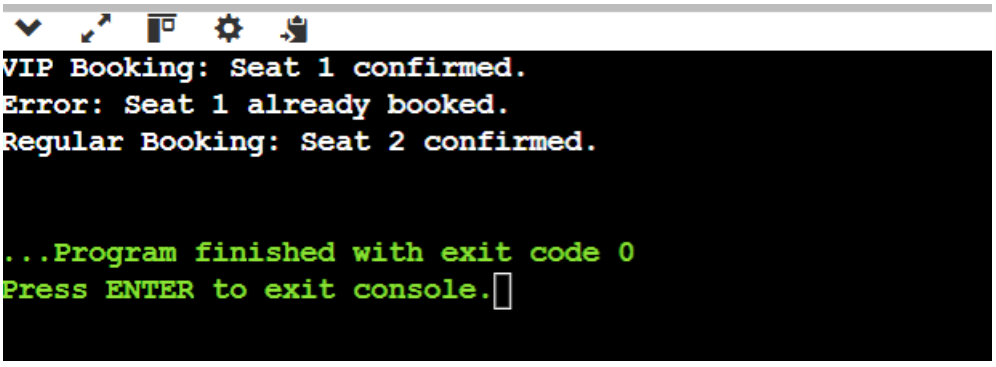
Efficient Resource Management: Optimize system performance by coordinating concurrent seat reservations while minimizing the risk of race conditions and ensuring fair access to resources.

Code:

```
import java.util.*;

class TicketBookingSystem {
    private static final int TOTAL_SEATS = 10;
    private static boolean[] seats = new boolean[TOTAL_SEATS];
    private static final Object lock = new Object();

    static class BookingThread extends Thread {
        private int seatNumber;
        private boolean isVIP;

        public BookingThread(int seatNumber, boolean isVIP) {
            this.seatNumber = seatNumber;
            this.isVIP = isVIP;
            if (isVIP) {
                setPriority(Thread.MAX_PRIORITY);
            } else {
                setPriority(Thread.MIN_PRIORITY);
            }
        }

        @Override
        public void run() {
            synchronized (lock) {
                if (!seats[seatNumber]) {
                    seats[seatNumber] = true;
                    System.out.println((isVIP ? "VIP Booking" : "Regular Booking") + ": Seat " + (seatNumber + 1)
+ " confirmed.");
                } else {
                    System.out.println("Error: Seat " + (seatNumber + 1) + " already booked.");
                }
            }
        }
    }

    public static void main(String[] args) {
        List<Thread> threads = new ArrayList<>();
        threads.add(new BookingThread(0, true));
        threads.add(new BookingThread(1, false));
        threads.add(new BookingThread(0, false));

        for (Thread t : threads) {
            t.start();
        }

        for (Thread t : threads) {
            try {
                t.join();
            } catch (InterruptedException e) {
```

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

Output:



Learning Outcomes:

1. Learn how to perform basic CRUD (Create, Read, Update, Delete) operations on a List of String objects in Java.
2. Understand how to use the ArrayList class for dynamically storing and manipulating a collection of items.
3. Practice handling user input using the Scanner class for interaction with the program.
4. Implement methods for searching, deleting, and displaying items in a list efficiently.
5. Gain familiarity with control flow and loops to allow for continuous user interaction until the program is exited.