Experiment 4

Student Name: Mayank **UID: 22BCS13254**
Branch: CSE **Section/Group: 639-A**
Semester: 6th **Date of Performance:14/2/25**
Subject Name: PBLJ **Subject Code: 22CSH-359**

1. **Aim:** Develop a Java program using ArrayList to manage employee details with functionalities to add, update, remove, and search employees.

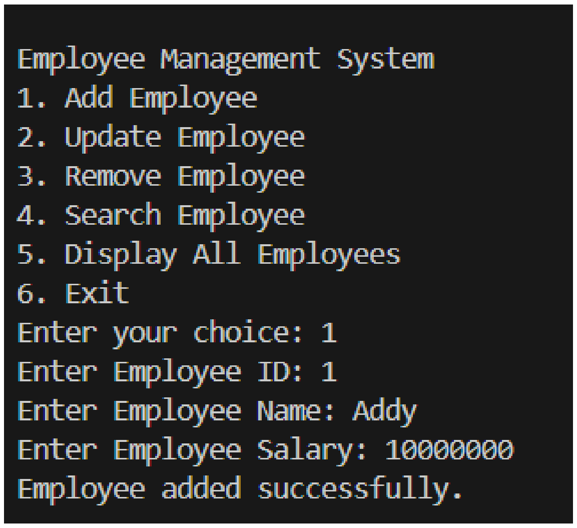
2. **Objective:** To efficiently manage employee records using ArrayList with CRUD operations for streamlined employee handling.

3. **Implementation/Code:**

```
import java.util.*;

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

    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 {
    private static List<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. Display All Employees");
        System.out.println("6. Exit");
        System.out.print("Enter your choice: ");
        int choice = scanner.nextInt();
        scanner.nextLine();
        switch (choice) {
            case 1: addEmployee(); break;
            case 2: updateEmployee(); break;
            case 3: removeEmployee(); break;
            case 4: searchEmployee(); break;
            case 5: displayEmployees(); break;
            case 6: 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();
    System.out.print("Enter Employee Name: ");
    String name = scanner.nextLine();
    System.out.print("Enter Employee Salary: ");
    double salary = scanner.nextDouble();
    scanner.nextLine();
    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();
    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();
            scanner.nextLine();
            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();
    scanner.nextLine();
    for (Employee emp : employees) {
        if (emp.id == 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();
    scanner.nextLine();
    for (Employee emp : employees) {
        if (emp.id == id) {
            System.out.println("Employee found: " + emp);
            return;
        }
        }
        System.out.println("Employee not found.");
    }
    private static void displayEmployees() {
        if (employees.isEmpty()) {
            System.out.println("No employees found.");
            return;
        }
        System.out.println("\nEmployee List:");
        for (Employee emp : employees) {
            System.out.println(emp);
        }
    }
}
```

4. Output:



```
Employee Management System
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: 1
Enter Employee Name: Addy
Enter Employee Salary: 10000000
Employee added successfully.
```

5. Learning Outcomes:

1. Understanding and implementing ArrayList for dynamic data storage and manipulation.
2. Developing CRUD (Create, Read, Update, Delete) functionalities in Java.
3. Enhancing problem-solving skills with search and update operations on employee records.

1. **Aim:** Implement a collection-based system to store and retrieve cards efficiently based on a given symbol using the Collection interface.

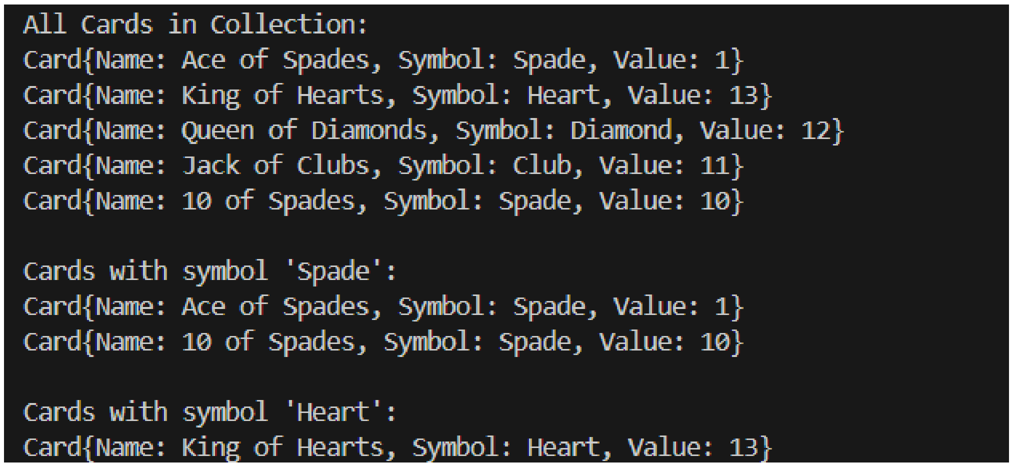
2. **Objective:** To organize and retrieve cards based on symbols using the Collection interface for easy accessibility.

3. **Implementation/Code:**

```
import java.util.*;

class Card {
    private String name;
    private String symbol;
    private int value;
    public Card(String name, String symbol, int value) {
        this.name = name;
        this.symbol = symbol;
        this.value = value;
    }
    public String getSymbol() {
        return symbol;
    }
    public String toString() {
        return "Card{Name: " + name + ", Symbol: " + symbol + ", Value: " + value + "}";
    }
}
public class CardCollectionSystem {
    private Collection<Card> cardCollection;
    public CardCollectionSystem() {
        cardCollection = new ArrayList<>();
    }
    public void addCard(String name, String symbol, int value) {
        cardCollection.add(new Card(name, symbol, value));
    }

    public void findCardsBySymbol(String symbol) {
        boolean found = false;
```

```
            System.out.println("Cards with symbol '" + symbol + "':");
            for (Card card : cardCollection) {
                if (card.getSymbol().equalsIgnoreCase(symbol)) {
                    System.out.println(card);
                    found = true;
                }
            }
            if (!found) {
                System.out.println("No cards found with the symbol '" + symbol + "'.");
            }
        }

        public void displayAllCards() {
            System.out.println("All Cards in Collection:");
            for (Card card : cardCollection) {
                System.out.println(card);
            }
        }

        public static void main(String[] args) {
            CardCollectionSystem cardSystem = new CardCollectionSystem();

            // Adding some sample cards
            cardSystem.addCard("Ace of Spades", "Spade", 1);
            cardSystem.addCard("King of Hearts", "Heart", 13);
            cardSystem.addCard("Queen of Diamonds", "Diamond", 12);
            cardSystem.addCard("Jack of Clubs", "Club", 11);
            cardSystem.addCard("10 of Spades", "Spade", 10);

            cardSystem.displayAllCards();

            System.out.println();
            cardSystem.findCardsBySymbol("Spade");

            System.out.println();
            cardSystem.findCardsBySymbol("Heart");
        }
    }
```

4. Output:



```
All Cards in Collection:
Card{Name: Ace of Spades, Symbol: Spade, Value: 1}
Card{Name: King of Hearts, Symbol: Heart, Value: 13}
Card{Name: Queen of Diamonds, Symbol: Diamond, Value: 12}
Card{Name: Jack of Clubs, Symbol: Club, Value: 11}
Card{Name: 10 of Spades, Symbol: Spade, Value: 10}

Cards with symbol 'Spade':
Card{Name: Ace of Spades, Symbol: Spade, Value: 1}
Card{Name: 10 of Spades, Symbol: Spade, Value: 10}

Cards with symbol 'Heart':
Card{Name: King of Hearts, Symbol: Heart, Value: 13}
```

5. Learning Outcomes:

1. Gaining proficiency in Java Collections Framework for managing and retrieving data efficiently.
2. Learning to organize and filter data using symbols and key attributes.
3. Implementing efficient search algorithms within a collection-based system.

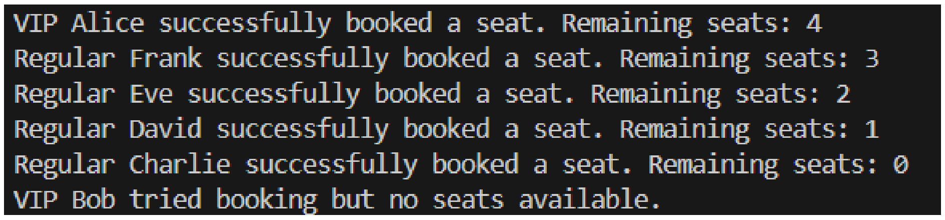
1. **Aim:** Create a synchronized multi-threaded ticket booking system that prevents double bookings and prioritizes VIP reservations.

2. **Objective:** To ensure fair and synchronized ticket booking, preventing double bookings while prioritizing VIP reservations.

3. **Implementation/Code:**

```
import java.util.*;
class TicketBookingSystem {
    private int availableSeats;
    public TicketBookingSystem(int seats) {
        this.availableSeats = seats;
    }
    public synchronized boolean bookTicket(String userType, String userName) {
        if (availableSeats > 0) {
            System.out.println(userType + " " + userName + " successfully booked a seat.
Remaining seats: " + (--availableSeats));
            return true;
        } else {
            System.out.println(userType + " " + userName + " tried booking but no seats
available.");
            return false;
        }
    }
}
class User extends Thread {
    private TicketBookingSystem system;
    private String userType;

    public User(TicketBookingSystem system, String userType, String name, int priority) {
        super(name);
        this.system = system;
        this.userType = userType;
        setPriority(priority);
    }
    @Override
    public void run() {
        system.bookTicket(userType, getName());
    }
}
public class TicketBookingApp {
    public static void main(String[] args) {
        int totalSeats = 5;
        TicketBookingSystem system = new TicketBookingSystem(totalSeats);
        User vip1 = new User(system, "VIP", "Alice", Thread.MAX_PRIORITY);
        User vip2 = new User(system, "VIP", "Bob", Thread.MAX_PRIORITY);
        User regular1 = new User(system, "Regular", "Charlie", Thread.NORM_PRIORITY);
        User regular2 = new User(system, "Regular", "David", Thread.NORM_PRIORITY);
        User regular3 = new User(system, "Regular", "Eve", Thread.NORM_PRIORITY);
        User regular4 = new User(system, "Regular", "Frank", Thread.MIN_PRIORITY);
        vip1.start();
        vip2.start();
        regular1.start();
        regular2.start();
        regular3.start();
        regular4.start();
    }
}
```

4. Output:



```
VIP Alice successfully booked a seat. Remaining seats: 4
Regular Frank successfully booked a seat. Remaining seats: 3
Regular Eve successfully booked a seat. Remaining seats: 2
Regular David successfully booked a seat. Remaining seats: 1
Regular Charlie successfully booked a seat. Remaining seats: 0
VIP Bob tried booking but no seats available.
```

5. Learning Outcomes:

1. Understanding multithreading and synchronization to prevent race conditions.
2. Implementing thread priorities to manage booking preferences, such as VIP reservations.
3. Ensuring data consistency and concurrency control in real-time booking systems.