Experiment-4

Student Name: Anna Agarwal **UID:** 22BCS16116
Branch: BE-CSE **Section/Group:** KPIT-901/A
Semester: 6th **Date of Performance:**21/02/25
Subject Name: PBLJ-Lab **Subject Code:** 22CSH-359

Easy Level:

1. Aim:

Write a Java program to implement an ArrayList that stores employee details (ID, Name, and Salary). Allow users to add, update, remove, and search employees.

2. Objective:

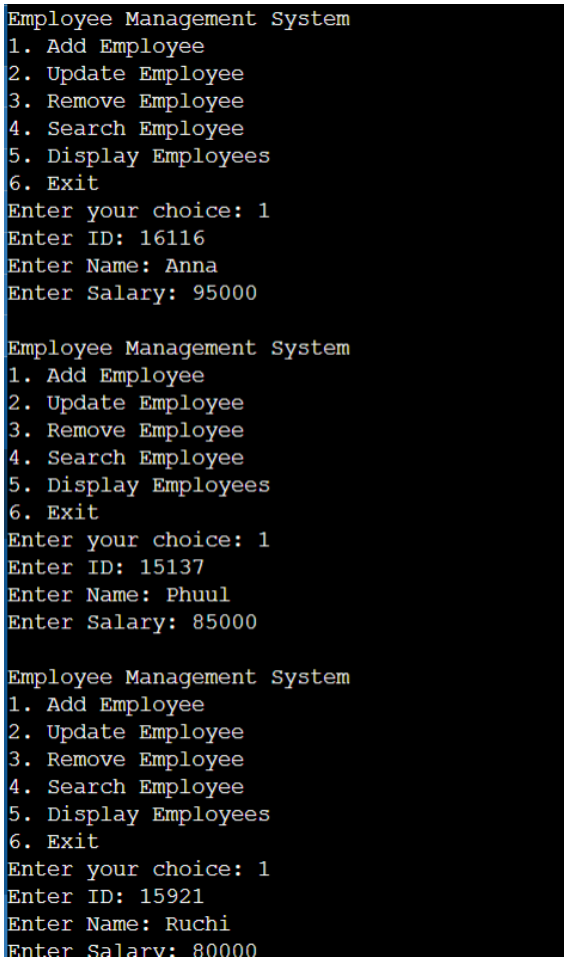
- **Store Employee Details**: Use an ArrayList to store employee objects containing ID, Name, and Salary.
- **Add Employees**: Allow users to add new employees to the list.
- **Update Employee Details**: Provide functionality to update employee information based on ID.
- **Remove Employees**: Enable users to remove an employee using their ID.
- **Search Employees**: Allow searching for employees by ID or Name.
- **Display Employee List**: Show all stored employee details in a formatted manner.

3. Implementation/Code:

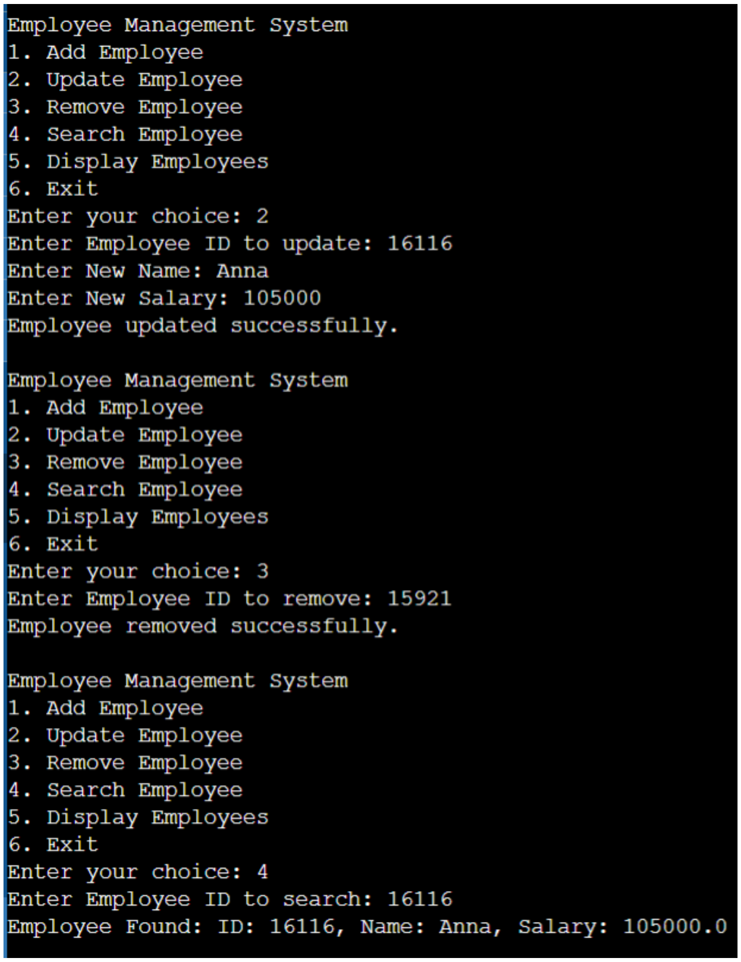
```
import java.util.*;

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;
    }

    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Salary: " + salary;
    }
}

public class EmployeeManager {
    private ArrayList<Employee> employees = new ArrayList<>();

    public void addEmployee(int id, String name, double salary) {
        employees.add(new Employee(id, name, salary));
    }

    public boolean updateEmployee(int id, String newName, double newSalary) {
        for (Employee emp : employees) {
            if (emp.getId() == id) {
                emp.setName(newName);
                emp.setSalary(newSalary);
                return true;
            }
        }
        return false;
    }

    public boolean removeEmployee(int id) {
        return employees.removeIf(emp -> emp.getId() == id);
    }

    public Employee searchEmployee(int id) {
        for (Employee emp : employees) {
            if (emp.getId() == id) {
                return emp;
            }
        }
        return null;
    }

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

    public static void main(String[] args) {
        EmployeeManager manager = new EmployeeManager();
        Scanner sc = new Scanner(System.in);
        int choice;

        do {
            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 Employees");
            System.out.println("6. Exit");
            System.out.print("Enter your choice: ");
            choice = sc.nextInt();

            switch (choice) {
                case 1:
                    System.out.print("Enter ID: ");
                    int id = sc.nextInt();
                    sc.nextLine();
                    System.out.print("Enter Name: ");
                    String name = sc.nextLine();
                    System.out.print("Enter Salary: ");
                    double salary = sc.nextDouble();
                    manager.addEmployee(id, name, salary);
                    break;
                case 2:
                    System.out.print("Enter Employee ID to update: ");
                    int updateId = sc.nextInt();
                    sc.nextLine();
                    System.out.print("Enter New Name: ");
                    String newName = sc.nextLine();
                    System.out.print("Enter New Salary: ");
                    double newSalary = sc.nextDouble();
                    if (manager.updateEmployee(updateId, newName, newSalary)) {
                        System.out.println("Employee updated successfully.");
                    } else {
                        System.out.println("Employee not found.");
                    }
                    break;
                case 3:
                    System.out.print("Enter Employee ID to remove: ");
                    int removeId = sc.nextInt();
                    if (manager.removeEmployee(removeId)) {
                        System.out.println("Employee removed successfully.");
                    } else {
                        System.out.println("Employee not found.");
                    }
                    break;
                case 4:
                    System.out.print("Enter Employee ID to search: ");
                    int searchId = sc.nextInt();
                    Employee emp = manager.searchEmployee(searchId);
                    if (emp != null) {
                        System.out.println("Employee Found: " + emp);
                    } else {
                        System.out.println("Employee not found.");
                    }
                    break;
                case 5:
                    manager.displayEmployees();
                    break;
                case 6:
                    System.out.println("Exiting...");
                    break;
                default:
                    System.out.println("Invalid choice. Please try again.");
            }
        } while (choice != 6);
        sc.close();
    }
}
```

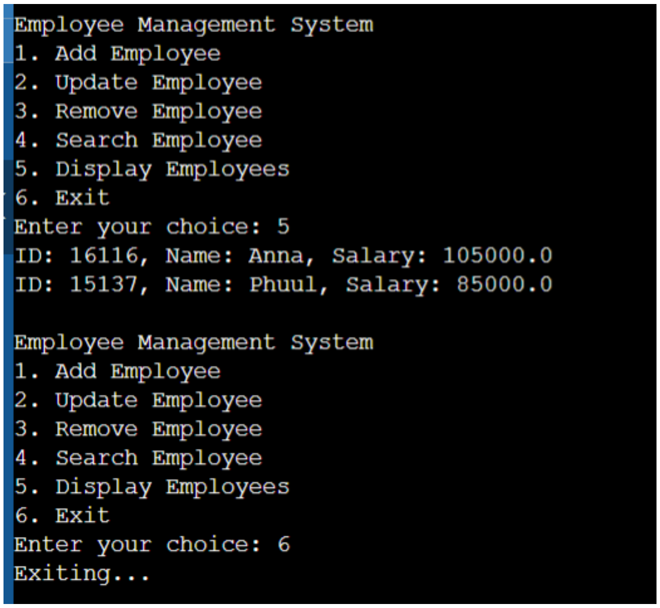
4. Output:



```
Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Enter your choice: 1
Enter ID: 16116
Enter Name: Anna
Enter Salary: 95000

Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Enter your choice: 1
Enter ID: 15137
Enter Name: Phuul
Enter Salary: 85000

Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Enter your choice: 1
Enter ID: 15921
Enter Name: Ruchi
Enter Salary: 80000
```



```
Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Enter your choice: 2
Enter Employee ID to update: 16116
Enter New Name: Anna
Enter New Salary: 105000
Employee updated successfully.

Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Enter your choice: 3
Enter Employee ID to remove: 15921
Employee removed successfully.

Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Enter your choice: 4
Enter Employee ID to search: 16116
Employee Found: ID: 16116, Name: Anna, Salary: 105000.0
```



```
Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Enter your choice: 5
ID: 16116, Name: Anna, Salary: 105000.0
ID: 15137, Name: Phuul, Salary: 85000.0

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

Medium Level:

1. **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.

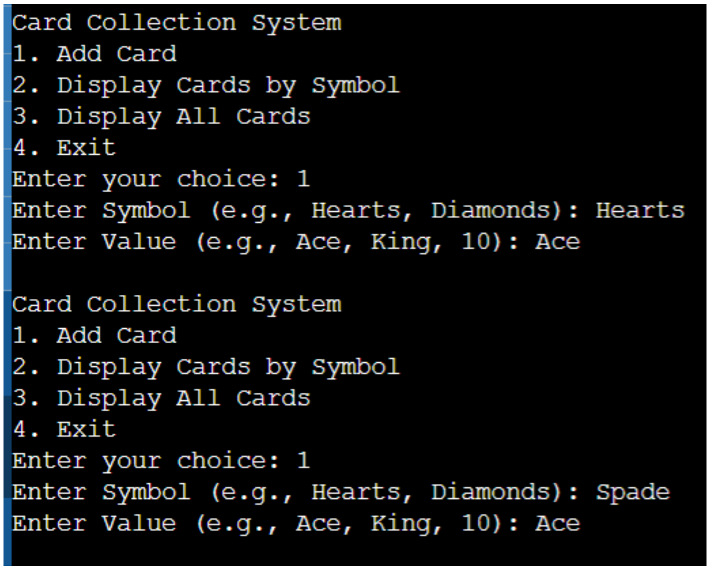
2. **Objective:**
   - **Understand Collections Framework**: Learn how to use the Collection interface and relevant implementations (ArrayList, HashSet, HashMap).
   - **Store and Manage Cards**: Use a suitable collection to store cards, ensuring easy retrieval.
   - **Search by Symbol**: Implement efficient searching to list all cards of a given symbol.
   - **Implement Basic Operations**: Learn how to add, remove, and display cards.
   - **Encapsulation & Object-Oriented Design**: Use classes and objects to structure data efficiently.

3. **Implementation/Code:**

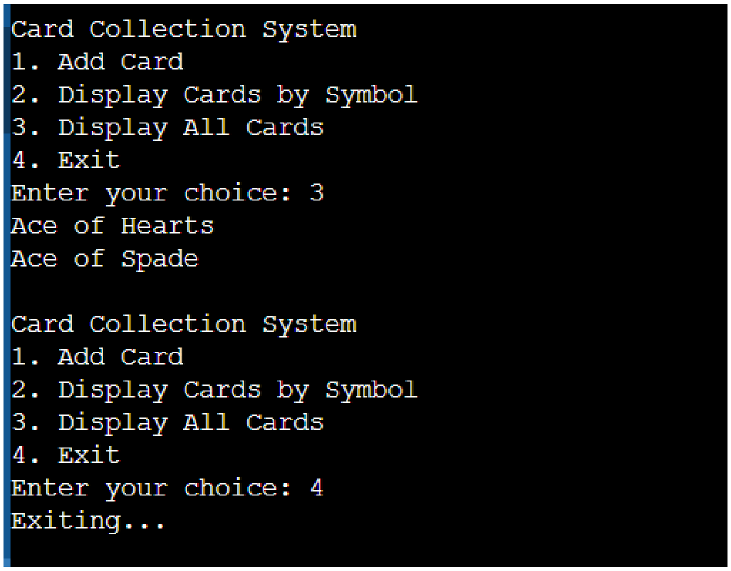
```
import java.util.*;

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;
    }

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

```
}

public class CardCollection {
    private Collection<Card> cards = new ArrayList<>();

    public void addCard(String symbol, String value) {
        cards.add(new Card(symbol, value));
    }

    public void displayCardsBySymbol(String symbol) {
        boolean found = false;
        for (Card card : cards) {
            if (card.getSymbol().equalsIgnoreCase(symbol)) {
                System.out.println(card);
                found = true;
            }
        }
        if (!found) {
            System.out.println("No cards found for symbol: " + symbol);
        }
    }

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

    public static void main(String[] args) {
        CardCollection collection = new CardCollection();
        Scanner sc = new Scanner(System.in);
        int choice;

        do {
            System.out.println("\nCard Collection System");
            System.out.println("1. Add Card");
            System.out.println("2. Display Cards by Symbol");
            System.out.println("3. Display All Cards");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");
            choice = sc.nextInt();
            sc.nextLine();

            switch (choice) {
                case 1:
                    System.out.print("Enter Symbol (e.g., Hearts, Diamonds): ");
                    String symbol = sc.nextLine();
                    System.out.print("Enter Value (e.g., Ace, King, 10): ");
                    String value = sc.nextLine();
                    collection.addCard(symbol, value);
                    break;
                case 2:
                    System.out.print("Enter Symbol to search: ");
                    String searchSymbol = sc.nextLine();
                    collection.displayCardsBySymbol(searchSymbol);
                    break;
                case 3:
                    collection.displayAllCards();
                    break;
                case 4:
                    System.out.println("Exiting...");
                    break;
                default:
                    System.out.println("Invalid choice. Please try again.");
            }
        } while (choice != 4);
        sc.close();
    }
}
}
```

4. Output:



```
Card Collection System
1. Add Card
2. Display Cards by Symbol
3. Display All Cards
4. Exit
Enter your choice: 1
Enter Symbol (e.g., Hearts, Diamonds): Hearts
Enter Value (e.g., Ace, King, 10): Ace

Card Collection System
1. Add Card
2. Display Cards by Symbol
3. Display All Cards
4. Exit
Enter your choice: 1
Enter Symbol (e.g., Hearts, Diamonds): Spade
Enter Value (e.g., Ace, King, 10): Ace
```



```
Card Collection System
1. Add Card
2. Display Cards by Symbol
3. Display All Cards
4. Exit
Enter your choice: 3
Ace of Hearts
Ace of Spade

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

Hard Level:

1. **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.

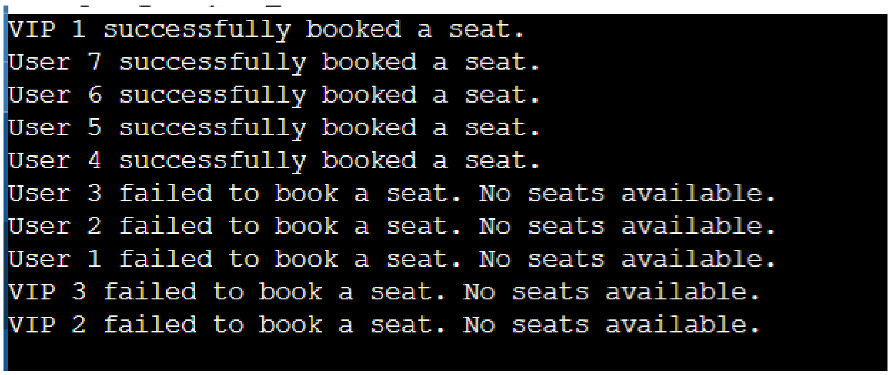
2. **Objective:**
   - **Thread Synchronization**: Learn how to prevent race conditions using synchronized methods/blocks.
   - **Thread Priorities**: Understand how to prioritize certain bookings (e.g., VIP bookings).
   - **Concurrency Management**: Ensure safe multi-threaded access to shared resources (seat availability).
   - **Efficient Ticket Allocation**: Implement a system where multiple users can book seats without conflicts.
   - **Practical Implementation of Threads**: Use Thread class and Runnable interface to simulate real-world scenarios.

3. **Implementation/Code:**

```
import java.util.*;

class TicketBookingSystem {
    private int availableSeats;
    private final Object lock = new Object();

    public TicketBookingSystem(int seats) {
        this.availableSeats = seats;
    }

    public void bookSeat(String name) {
        synchronized (lock) {
            if (availableSeats > 0) {
                System.out.println(name + " successfully booked a seat.");
                availableSeats--;
            } else {
                System.out.println(name + " failed to book a seat. No seats available.");
            }
        }
    }
}
```

```
class BookingThread extends Thread {
    private TicketBookingSystem system;
    private String name;

    public BookingThread(TicketBookingSystem system, String name, int priority) {
        this.system = system;
        this.name = name;
        setPriority(priority);
    }

    public void run() {
        system.bookSeat(name);
    }
}

public class TicketBooking {
    public static void main(String[] args) {
        TicketBookingSystem system = new TicketBookingSystem(5);
        Thread[] threads = new Thread[10];

        threads[0] = new BookingThread(system, "VIP 1", Thread.MAX_PRIORITY);
        threads[1] = new BookingThread(system, "VIP 2", Thread.MAX_PRIORITY);
        threads[2] = new BookingThread(system, "VIP 3", Thread.MAX_PRIORITY);
        threads[3] = new BookingThread(system, "User 1", Thread.NORM_PRIORITY);
        threads[4] = new BookingThread(system, "User 2", Thread.NORM_PRIORITY);
        threads[5] = new BookingThread(system, "User 3", Thread.NORM_PRIORITY);
        threads[6] = new BookingThread(system, "User 4", Thread.NORM_PRIORITY);
        threads[7] = new BookingThread(system, "User 5", Thread.NORM_PRIORITY);
        threads[8] = new BookingThread(system, "User 6", Thread.MIN_PRIORITY);
        threads[9] = new BookingThread(system, "User 7", Thread.MIN_PRIORITY);

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

4. Output:



```
VIP 1 successfully booked a seat.
User 7 successfully booked a seat.
User 6 successfully booked a seat.
User 5 successfully booked a seat.
User 4 successfully booked a seat.
User 3 failed to book a seat. No seats available.
User 2 failed to book a seat. No seats available.
User 1 failed to book a seat. No seats available.
VIP 3 failed to book a seat. No seats available.
VIP 2 failed to book a seat. No seats available.
```

5. Learning Outcome:

a) Understanding Java Collections Framework

- Learn to use ArrayList, HashMap, and Collection interface to store and manage dynamic data structures.
- Implement searching, updating, and removing elements efficiently.

b) Object-Oriented Programming (OOP) Concepts

- Apply encapsulation, inheritance, and method overriding to create well-structured and reusable code.
- Design modular classes for real-world applications like employee management, card storage, and ticket booking.

c) Thread Synchronization and Concurrency Management

- Use synchronized methods/blocks to prevent race conditions in multi-threaded applications.
- Ensure thread-safe operations for critical sections like seat booking.

d) Thread Prioritization and Performance Optimization

- Implement Thread priorities to handle VIP bookings first, simulating real-world scheduling.
- Understand how CPU scheduling affects multi-threaded performance.

e) Practical Implementation of Data Handling and Processing

- Store and retrieve structured data (employees, cards, tickets) efficiently.
- Implement searching and filtering techniques for optimized user experience.

f) Simulating Real-World Scenarios with Java

- Develop a **ticket booking system** with concurrency control.
- Create a **card storage system** for quick retrieval based on attributes.
- Build an **employee management system** with CRUD (Create, Read, Update, Delete) operations.