Experiment 5

Student Name: Namandeep Jain **UID: 22BCS14946**
Branch: CSE **Section/Group: 618/A**
Semester: 6 **Date of Performance: 21/02/25**
Subject Name: Project Based Learning in Java **Subject Code: 22CSH-359**
Problem 1

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:

The Objective is to implement an ArrayList that stores employee details (ID, Name, and Salary) and allow users to add, update, remove, and search employees.

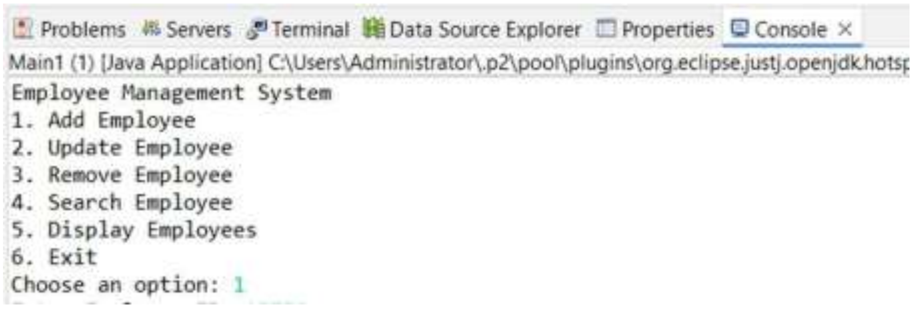
3. Implementation/Code:

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

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 Main {
    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();
       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!");
    }

    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 details 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();
    employees.removeIf(emp -> emp.id == id);
    System.out.println("Employee removed successfully!");
}

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 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)
    { 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 Employees");
        System.out.println("6.          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:
                    displayEmployees();
                    break;
                case 6:
                    System.out.println("Exiting...")
                    ; scanner.close(); return;
                default:
                    System.out.println("Invalid choice! Please try again.");
            }
        }
    }
}
```

4. Output



```
Problems  Servers  Terminal  Data Source Explorer  Properties  Console
Main1 (1) [Java Application] C:\Users\Administrator\.p2\pool\plugins\org.eclipse.justj.openjdk.hotsp
Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Choose an option: 1
```

5. Learning Outcomes

- Learn how to use ArrayList to store and manage employee details dynamically.
- Implement adding, updating, removing, and searching records efficiently.
- Use Java classes and objects to encapsulate employee details.

Problem 2

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:**

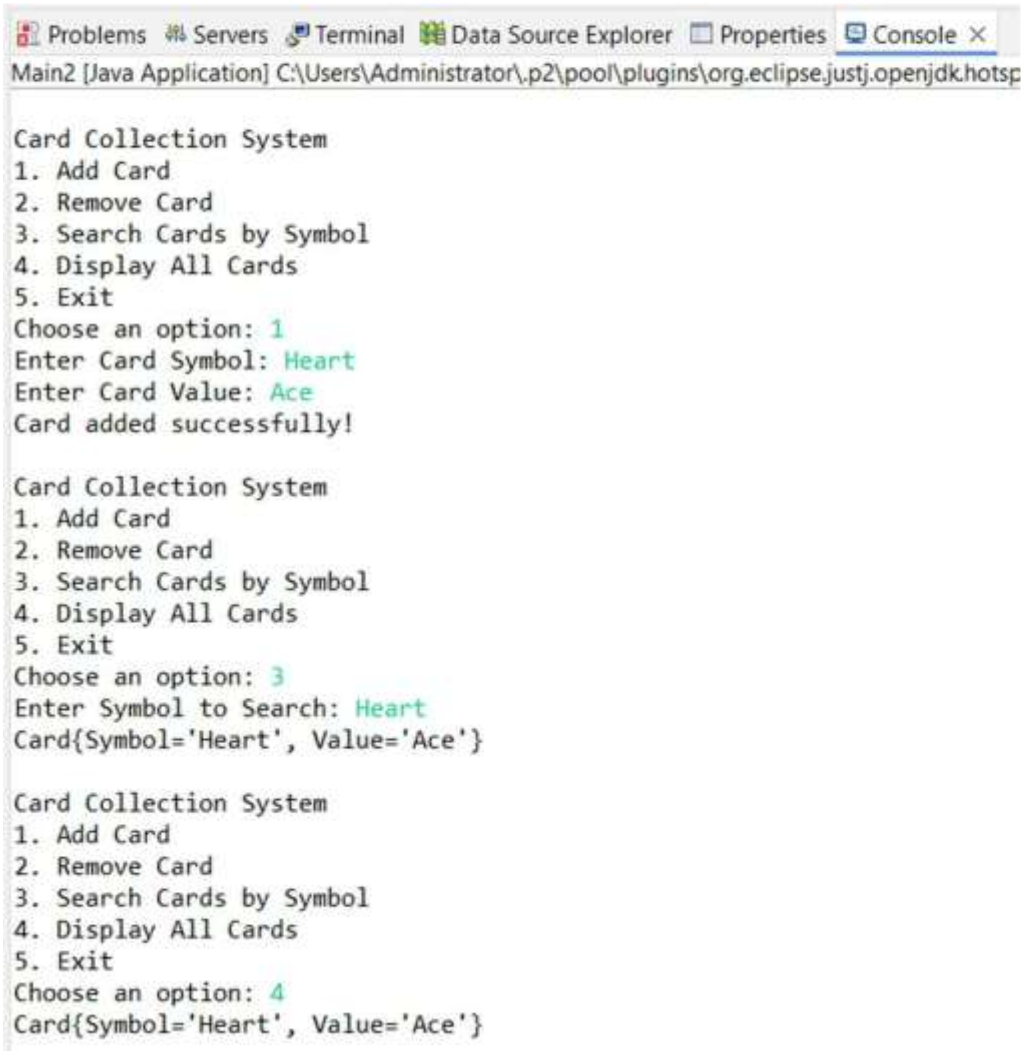
The Objective is to collect and store all the cards to assist the users in finding all the cards in a given symbol using Collection interface.

3. **Implementation/Code:**

```
package java1;
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;

   }

   @Override public String
   toString() {
        return "Card{Symbol='" + symbol + "', Value='" + value + "'}";
    }
} public class Main2 { private
Collection<Card> cards;

    public Main2() {
        cards = new ArrayList<>();
    }
    public void addCard(String symbol, String value)
        { cards.add(new Card(symbol, value));
        System.out.println("Card added successfully!");
    }

    public void removeCard(String symbol, String value)
        { cards.removeIf(card -> card.getSymbol().equals(symbol) &&
card.getValue().equals(value));
        System.out.println("Card removed successfully!");
    }

    public void searchCardsBySymbol(String symbol)
        { boolean found = false;
        for (Card card : cards) {
            if (card.getSymbol().equals(symbol))
                {  System.out.println(card);  found
                = true;
            }
        } if (!found)
        {
            System.out.println("No cards found for the 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)
    { Scanner scanner = new
    Scanner(System.in); Main2 collection = new
    Main2();
     while (true) {
            System.out.println("\nCard Collection System");
            System.out.println("1. Add Card");
            System.out.println("2. Remove Card");
            System.out.println("3. Search Cards by Symbol");
            System.out.println("4. Display All Cards");
            System.out.println("5.                Exit");
            System.out.print("Choose an option: ");
            int      choice      =      scanner.nextInt();
            scanner.nextLine();

            switch (choice)
                { case 1:
                    System.out.print("Enter Card Symbol: ");
                    String symbol = scanner.nextLine();
                    System.out.print("Enter Card Value: ");
                    String    value    =    scanner.nextLine();
                    collection.addCard(symbol,       value);
                    break;
                case 2:
                    System.out.print("Enter Card Symbol to Remove: ");
                    String removeSymbol = scanner.nextLine();
```

```
                        System.out.print("Enter Card Value to Remove: ");
                        String removeValue = scanner.nextLine();
                        collection.removeCard(removeSymbol, removeValue);
                        break;
                    case 3:
                        System.out.print("Enter Symbol to Search: ");
                        String searchSymbol = scanner.nextLine();
                        collection.searchCardsBySymbol(searchSymbol);
                        break;
                    case 4:
                        collection.displayAllCards();
                        break;
                    case 5:
                        System.out.println("Exiting...")
                        ; scanner.close(); return;
                    default:
                        System.out.println("Invalid choice! Please try again.");
                }
            }
        }
    }
```

4. Output



```
Main2 [Java Application] C:\Users\Administrator\.p2\pool\plugins\org.eclipse.justj.openjdk.hotsp

Card Collection System
1. Add Card
2. Remove Card
3. Search Cards by Symbol
4. Display All Cards
5. Exit
Choose an option: 1
Enter Card Symbol: Heart
Enter Card Value: Ace
Card added successfully!

Card Collection System
1. Add Card
2. Remove Card
3. Search Cards by Symbol
4. Display All Cards
5. Exit
Choose an option: 3
Enter Symbol to Search: Heart
Card{Symbol='Heart', Value='Ace'}

Card Collection System
1. Add Card
2. Remove Card
3. Search Cards by Symbol
4. Display All Cards
5. Exit
Choose an option: 4
Card{Symbol='Heart', Value='Ace'}
```

5. Learning Outcomes

- Implement ArrayList for dynamic storage of card objects.
- Custom Class Implementation: Learn how to create and use custom classes (Card).
- Object-Oriented Programming (OOP): Apply encapsulation and class design principles.

Problem 3

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:

The Objective is to use thread priorities to simulate VIP bookings being processed first.

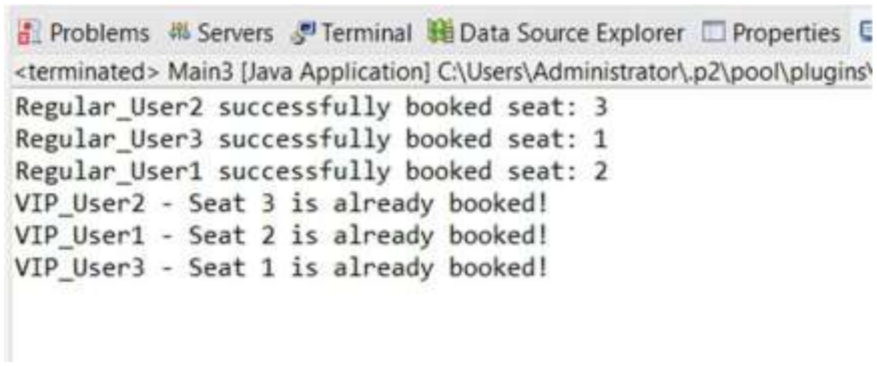
3. Implementation/Code:

```
package          java1;
import java.util.*;

class TicketBookingSystem {
   private final int totalSeats;
   private final boolean[] seats;

   public TicketBookingSystem(int totalSeats)
      { this.totalSeats = totalSeats; this.seats
      = new boolean[totalSeats];
   }

   public synchronized boolean bookSeat(int seatNumber, String user)
      { if (seatNumber < 0 || seatNumber >= totalSeats) {
         System.out.println(user + " - Invalid seat number: " + seatNumber); return
         false;
      }
      if (!seats[seatNumber])
         { seats[seatNumber] = true;
         System.out.println(user + " successfully booked seat: " + seatNumber); return
         true;
      } else {
         System.out.println(user + " - Seat " + seatNumber + " is already booked!");
         return false;
      }
   }
}
```

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

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

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

public class Main3 {
   public static void main(String[] args) {
      TicketBookingSystem system = new TicketBookingSystem(5);

      List<BookingThread> threads = new ArrayList<>(); threads.add(new
      BookingThread(system, 2, "VIP_User1",
Thread.MAX_PRIORITY));                              threads.add(new
      BookingThread(system, 2, "Regular_User1",
Thread.NORM_PRIORITY));                       threads.add(new
      BookingThread(system, 3, "VIP_User2",
Thread.MAX_PRIORITY));                              threads.add(new
      BookingThread(system, 3, "Regular_User2",
Thread.NORM_PRIORITY));                       threads.add(new
      BookingThread(system, 1, "VIP_User3",
Thread.MAX_PRIORITY)); threads.add(new
BookingThread(system, 1, "Regular_User3",
Thread.NORM_PRIORITY));

      Collections.shuffle(threads);  //  Simulate  concurrent  requests
      for (BookingThread thread : threads) {
         thread.start();
```

```
        }
    }
}
```

4. Output



```
<terminated> Main3 [Java Application] C:\Users\Administrator\.p2\pool\plugins\
Regular_User2 successfully booked seat: 3
Regular_User3 successfully booked seat: 1
Regular_User1 successfully booked seat: 2
VIP_User2 - Seat 3 is already booked!
VIP_User1 - Seat 2 is already booked!
VIP_User3 - Seat 1 is already booked!
```

5. Learning Outcomes

i. Use synchronized methods to prevent race conditions and ensure seat bookings are not duplicated. **ii.** Assign priorities to threads (Thread.MAX_PRIORITY for VIP users) to control execution order.

iii. Learn how multiple threads can compete for shared resources.