Experiment-4

StudentName: Mohammad Farhan Alam UID:22BCS13460
Branch: BE-CSE Section/Group: DL-902-A
Semester: 6th Date of Performance:07/02/25
Subject Name: PBLJ Lab Subject Code: 22CSH-359

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. Algorithm:
   1. Initialize an ArrayList to store employees.
   2. Display a menu for adding, updating, removing, searching, and displaying employees.
   3. Perform operations based on user input using loops and conditions.
   4. Exit when the user chooses to quit.

3. Implementation/Code:

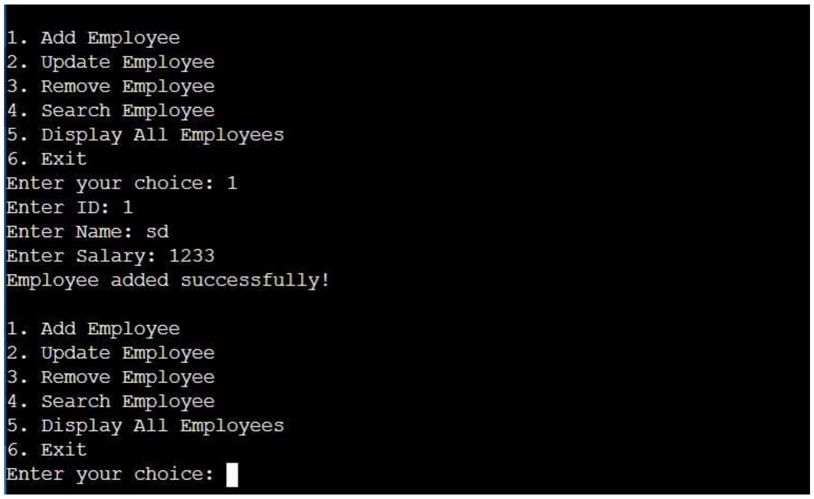
```
import java.util.ArrayList;
import   java.util.Scanner;
class  Employee  {  int  id;
String     name;     double
salary;
   Employee(int id, String name, double salary)
      { this.id = id; this.name = name; this.salary
      = salary;
   }
   @Override public String
   toString() {
      return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

```
public static void main(String[] args) {
    ArrayList<Employee> employees = new ArrayList<>();
    Scanner scanner = new Scanner(System.in); int choice;
    do {
        System.out.println("\n1. 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:
        "); choice = scanner.nextInt(); switch
        (choice) {
            case 1:
                System.out.print("Enter 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!");
                break;
            case 2:
                System.out.print("Enter ID to update: ");
                int updateId = scanner.nextInt();
                for (Employee emp : employees) {
                    if (emp.id == updateId) {
```

```
            scanner.nextLine(); // consume newline
            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!");
            break;
        } }
    break;
case 3:
    System.out.print("Enter ID to remove: "); int removeId
    = scanner.nextInt(); employees.removeIf(emp ->
    emp.id == removeId); System.out.println("Employee
    removed successfully!"); break;
case 4:
    System.out.print("Enter ID to search: ");
    int searchId = scanner.nextInt(); for
    (Employee emp : employees) {
        if (emp.id == searchId) {
            System.out.println(emp);
            break;
        } }
    break;
case 5:
    System.out.println("All Employees:");
    for (Employee emp : employees) {
        System.out.println(emp);
    }
    break;
case 6:
```

```
                System.out.println("Exiting program...");
                break;
            default:
                System.out.println("Invalid choice! Please try again.");
        }
    } while (choice != 6);
    scanner.close();
  }
}
```

4. OUTPUT:



```
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Enter your choice: 1
Enter ID: 1
Enter Name: sd
Enter Salary: 1233
Employee added successfully!

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

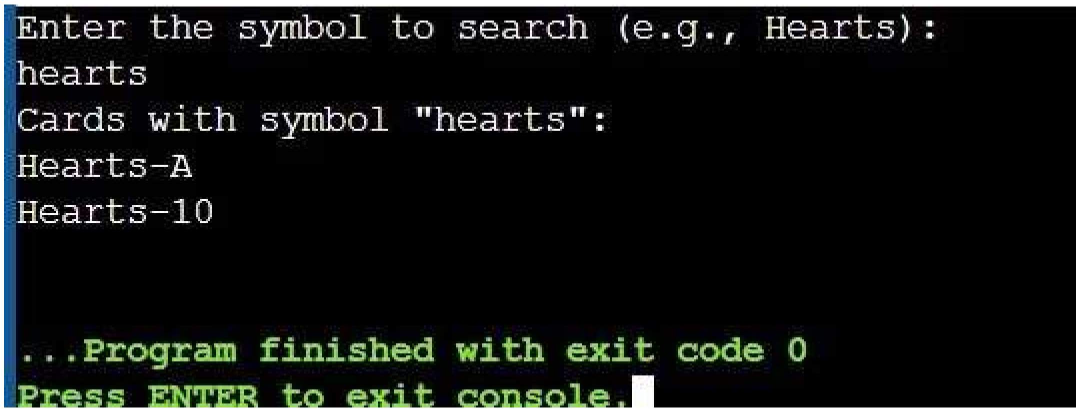
<u>Question2:</u>

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. Algorithm:
   - Create a List of cards with symbols and values.
   - Accept user input for the symbol to search.
   - Loop through the list and display cards matching the symbol.

3. Implementation/Code:

```
import java.util.ArrayList;

import java.util.Collections;

import java.util.List; import

java.util.Scanner; class

Card {

    String symbol;

    String value;

    Card(String symbol, String value)

        { this.symbol = symbol;

        this.value = value;

    } public String toString() {

    return symbol + "-" + value;

    }

}public class CardCollection { public

    static void main(String[] args) {

        List<Card> cards = new ArrayList<>();
        Scanner scanner = new Scanner(System.in);

        cards.add(new Card("Hearts", "A"));

        cards.add(new Card("Spades", "K"));

        cards.add(new Card("Hearts", "10"));

        cards.add(new Card("Diamonds", "Q"));

        cards.add(new Card("Clubs", "J"));

        System.out.println("Enter the symbol to search (e.g., Hearts): ");
        String symbol = scanner.nextLine();

        System.out.println("Cards with symbol \"" + symbol + "\":");

        for (Card card : cards) { if

        (card.symbol.equalsIgnoreCase(symbol)) {

                System.out.println(card);

          }

        }

        scanner.close();

    }

}
```

4. Output:



```
Enter the symbol to search (e.g., Hearts):
hearts
Cards with symbol "hearts":
Hearts-A
Hearts-10


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

<u>Question3:</u>

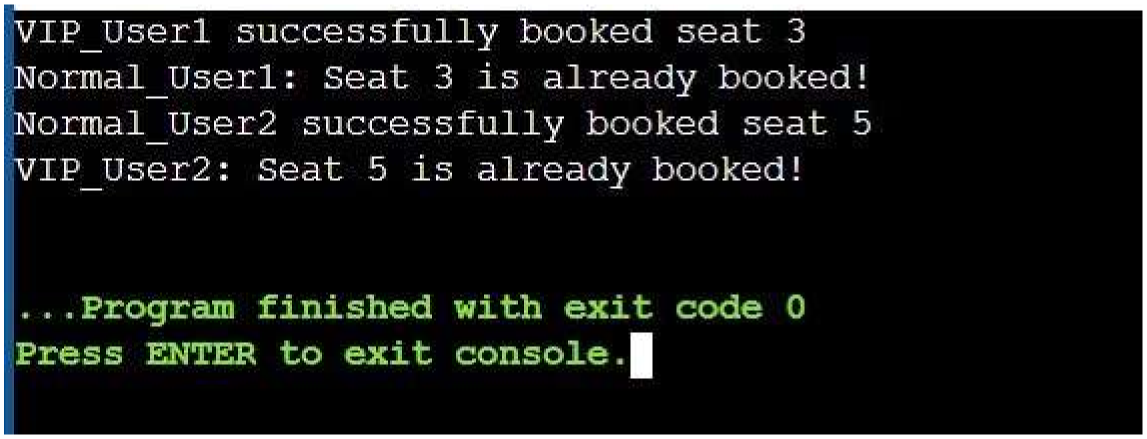
1. Aim:Develop a ticket booking system with synchronized threads to ensure no double booking of seats. Use thread priorities to simulate VIP bookings being processed first.

2. Algorithm:
   - Initialize a boolean[] for seats and a ReentrantLock for synchronization.
   - Create threads with priorities representing users booking seats.
   - Lock the seat array while booking to prevent double bookings.
   - Run threads; higher priority threads book first.

3. Implementation/Code:

```
import java.util.concurrent.locks.ReentrantLock;
class TicketBookingSystem { private final
boolean[] seats; private final ReentrantLock
lock;
  TicketBookingSystem(int totalSeats) {
    this.seats = new boolean[totalSeats];
    this.lock = new ReentrantLock();
  } public void bookSeat(String user, int seatNumber)
  { lock.lock(); try {
      if (seatNumber < 0 || seatNumber >= seats.length) {
      System.out.println(user + ": Invalid seat number!");
      return; }
      if (!seats[seatNumber]) {
        seats[seatNumber] = true;
        System.out.println(user + " successfully booked seat " + seatNumber);
      } else {
        System.out.println(user + ": Seat " + seatNumber + " is already
booked!");
      }
    } finally {
      lock.unlock();
    }
  } } class User extends
Thread {
  private final TicketBookingSystem bookingSystem;
  private final int seatNumber;
  User(String name, TicketBookingSystem bookingSystem, int seatNumber, int
priority) { super(name); this.bookingSystem = bookingSystem; this.seatNumber =
seatNumber;
    setPriority(priority);
  } @Override
  public void run()
  {
    bookingSystem.bookSeat(getName(), seatNumber);
} }
public class TicketBookingDemo {
  public static void main(String[] args) {
        TicketBookingSystem bookingSystem = new TicketBookingSystem(10);
        User user1 = new User("VIP_User1", bookingSystem, 3, Thread.MAX_PRIORITY);
        User user2 = new User("Normal_User1", bookingSystem, 3, Thread.MIN_PRIORITY);
        User user3 = new User("Normal_User2", bookingSystem, 5, Thread.NORM_PRIORITY);
        User user4 = new User("VIP_User2", bookingSystem, 5, Thread.MAX_PRIORITY);
        user1.start();
        user2.start();
        user3.start();
        user4.start();
    }
}
```

4. OUTPUT:



```
VIP_User1 successfully booked seat 3
Normal_User1: Seat 3 is already booked!
Normal_User2 successfully booked seat 5
VIP_User2: Seat 5 is already booked!


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

5. Learning Outcome:

- Object-Oriented Programming (OOP): Applied encapsulation, inheritance, and polymorphism to design modular and reusable code (e.g., Employee, Card, User classes).

- Collections Framework: Utilized ArrayList and Collection for data storage, retrieval, and filtering operations, showcasing dynamic data management.

- Multithreading and Synchronization: Designed a thread-safe system using ReentrantLock and thread priorities to handle concurrency and ensure data consistency (e.g., ticket booking).

- User Interaction: Built interactive, menu-driven programs for CRUD operations, validating user inputs for robust functionality.

- Real-world Problem Solving: Implemented practical systems like employee management, card searching, and seat booking, reflecting real-world scenarios and scalable design.