Experiment 4

Student Name: Khushi **UID: 22BCS15811**
Branch:BE-CSE **Section/Group: 618_B**
Semester: 6 **Date of Performance:19/2/25**
Subject Name: PBLJ **Subject Code: 22CSH 359**

Question-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.

Objective: To implement **ArrayList** in Java for managing employee records using CRUD operations (Create, Read, Update, Delete). It reinforces **Object-Oriented Programming (OOP)** principles like encapsulation and object manipulation.

Implementation/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;
   }}
```

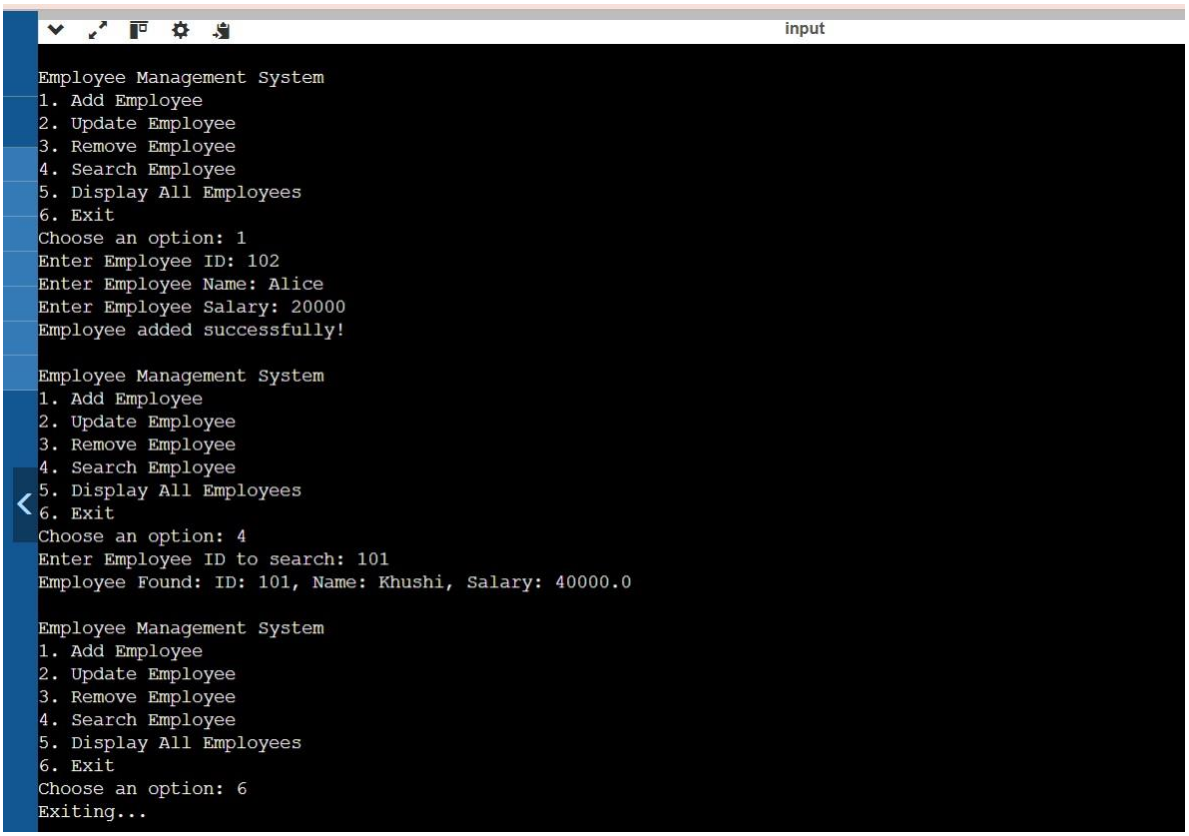
```
public class EmployeeManagement {
    static ArrayList<Employee> employees = new ArrayList<>();
static Scanner scanner = new Scanner(System.in);

    public static void addEmployee() {
System.out.print("Enter Employee ID: ");         int
id = scanner.nextInt();         scanner.nextLine();
        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!\n");
    }
    public static void updateEmployee() {
        System.out.print("Enter Employee ID to update: ");
int id = scanner.nextInt();         for (Employee emp :
employees) {            if (emp.id == id) {
scanner.nextLine();
                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!\n");
return;
            }
```

```
        }
        System.out.println("Employee not found!\n");
    }

    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!\n");
    }

    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("Employee Found: " + emp + "\n");
return;
  }
        }
        System.out.println("Employee not found!\n");
    }

    public static void displayEmployees() {
        if (employees.isEmpty()) {
            System.out.println("No employees found!\n");
return;
        }
```

```
        System.out.println("Employee List:");
for (Employee emp : employees) {
            System.out.println(emp);
        }
        System.out.println();
    }

    public static void main(String[] args) {
while (true) {
            System.out.println("Employee 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("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! Try again.\n");
        }}}}
```

Output:



Learning Outcome:

- Learn how to use ArrayList for dynamic data storage and manipulation.
- Implement encapsulation, object creation, and object manipulation using Java classes.
- Gain hands-on experience with adding, updating, removing, and searching records.

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.

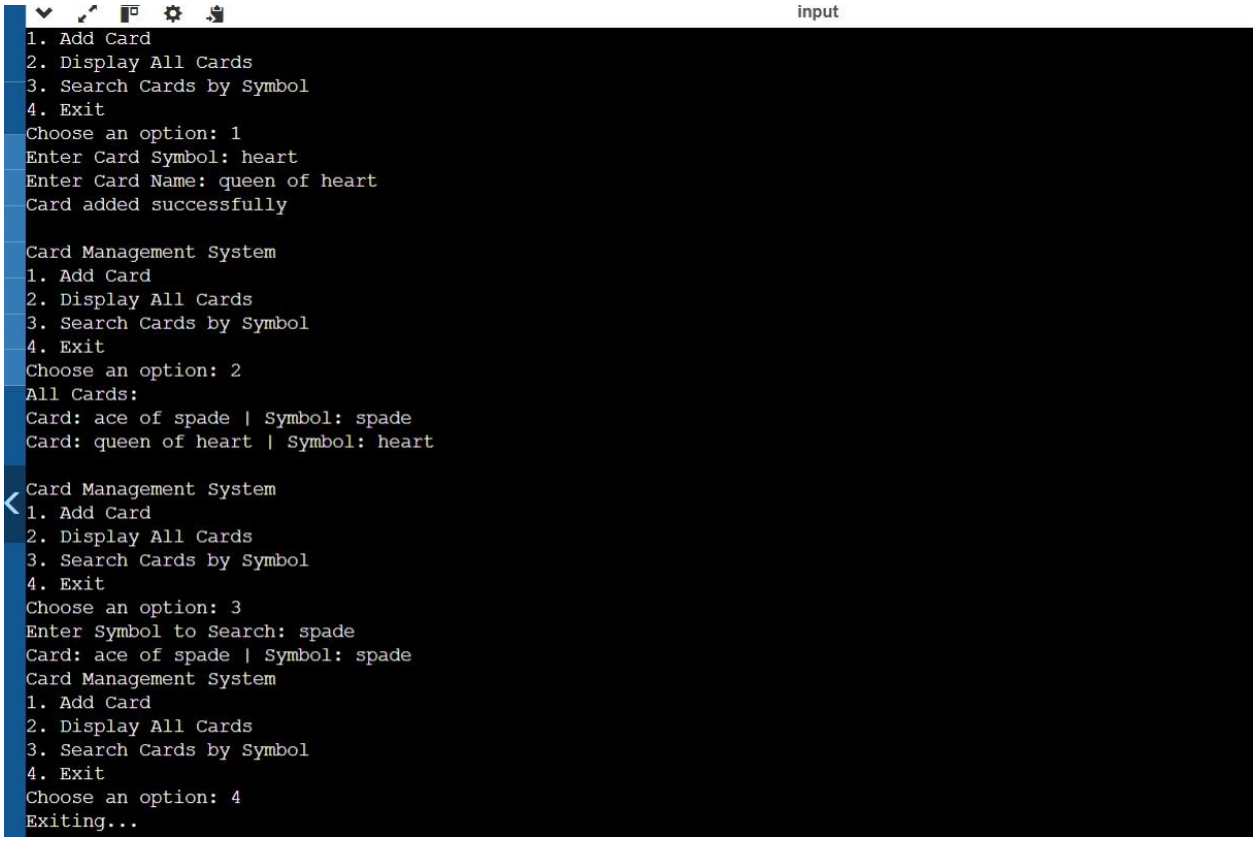
Code:

```
import java.util.*; class
Card {
   String symbol;
String name;
   public Card(String symbol, String name) {
this.symbol = symbol;        this.name =
name;
   }
   @Override
   public String toString() {
      return "Card: " + name + " | Symbol: " + symbol;
   }
}
public class CardCollection {
   static Collection<Card> cards = new ArrayList<>();
static Scanner scanner = new Scanner(System.in);


   public static void addCard() {
      System.out.print("Enter Card Symbol: ");
      String symbol = scanner.next();
scanner.nextLine();
```

```
        System.out.print("Enter Card Name: ");
String name = scanner.nextLine();
cards.add(new Card(symbol, name));
        System.out.println("Card added successfully \n");
    }

    public static void displayCards() {
if (cards.isEmpty()) {
            System.out.println("No cards available\n");
return;
        }
        System.out.println("All Cards:");
for (Card card : cards) {
            System.out.println(card);
        }
        System.out.println();
    }

    public static void searchCardsBySymbol() {
        System.out.print("Enter Symbol to Search: ");
String searchSymbol = scanner.next();
boolean found = false;          for (Card card : cards)
{
            if (card.symbol.equalsIgnoreCase(searchSymbol)) {                    System.out.println(card);
found = true;
            }}          if
(!found) {
```

```
            System.out.println("No cards found with this symbol!\n");
        }}

    public static void main(String[] args) {
while (true) {
            System.out.println("Card Management System");
            System.out.println("1. Add Card");
            System.out.println("2. Display All Cards");
            System.out.println("3. Search Cards by Symbol");
            System.out.println("4. Exit");
System.out.print("Choose an option: ");            int
choice = scanner.nextInt();

            switch (choice) {
case 1:
addCard();
break;              case 2:
displayCards();
break;              case 3:
                searchCardsBySymbol();
                break;
case 4:
                System.out.println("Exiting...");
scanner.close();                return;
default:
                System.out.println("Invalid choice.\n");
            }}}}
```

OUTPUT:



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.

Code:

```
import java.util.concurrent.*; class
TicketBookingSystem {     private
int availableSeats;
   public TicketBookingSystem(int seats) {
this.availableSeats = seats;
```

```
    }
    public synchronized boolean bookSeat(String customer) {
if (availableSeats > 0) {
            System.out.println(customer + " booked a seat. Remaining: " + (--availableSeats));
return true;
        } else {
            System.out.println(customer + " failed to book a seat. No seats left!");
return false;
        }}}}
class CustomerThread extends Thread {
private TicketBookingSystem system;
private String customerName;
    public CustomerThread(TicketBookingSystem system, String customerName, int priority) {
this.system = system;
        this.customerName = customerName;
this.setPriority(priority); // Set thread priority
    }

    @Override
public void run() {
        system.bookSeat(customerName);
    }}
public class TicketBooking {     public
static void main(String[] args) {
        TicketBookingSystem system = new TicketBookingSystem(5);
        CustomerThread vip1 = new CustomerThread(system, "VIP_John",
Thread.MAX_PRIORITY);
```

```
        CustomerThread vip2 = new CustomerThread(system, "VIP_Alice",
Thread.MAX_PRIORITY);

        CustomerThread normal1 = new CustomerThread(system, "User_Bob",
Thread.NORM_PRIORITY);

        CustomerThread normal2 = new CustomerThread(system, "User_Emma",
Thread.NORM_PRIORITY);

        CustomerThread normal3 = new CustomerThread(system, "User_Mike",
Thread.NORM_PRIORITY);

        CustomerThread normal4 = new CustomerThread(system, "User_Sam",
Thread.NORM_PRIORITY);

        vip1.start();

vip2.start();

normal1.start();

normal2.start();

normal3.start();

normal4.start();

    }

}
```

OUTPUT:

Learning Outcomes:

- Learn how to prevent race conditions and ensure data integrity using synchronized methods.
- Gain practical experience in managing multiple threads accessing shared resources.
- Implement logic to prevent multiple users from booking the same seat simultaneously.