Experiment-4

Student Name: Anshika kumari
UID: 22BCS10074
Branch: BE-CSE
Section/Group: 22BCS-IOT-640-A
Semester: 6th
Date of Performance: 17/02/2025
Subject Name: PBLJ with Lab
Subject Code: 22CSH-359

1. **Aim:** Develop Java programs using core concepts such as data structures, collections, and multithreading to manage and manipulate data.

2. **Problem Statements:**
   Problem 1.1: 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.
   Problem 1.2: 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.
   Problem 1.3: 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.

3. **Implementation/Code:**

Problem 1.1

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

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

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

    public static void main(String[] args)
        { while (true) {
            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: ");

            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline
```

```
            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.exit(0);
                default:
                    System.out.println("Invalid choice. Please try again.");
            }
        }
    }
```

```
private static void addEmployee()
    { System.out.print("Enter employee ID: ");
    int id = scanner.nextInt();
    scanner.nextLine(); // Consume newline
    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.");
}

private static void updateEmployee()
    { System.out.print("Enter employee ID to update: ");
    int id = scanner.nextInt();
    scanner.nextLine(); // Consume newline

    for (Employee emp : employees)
        { if (emp.getId() == id) {
            System.out.print("Enter new name (press enter to skip): ");
            String name = scanner.nextLine();
            if (!name.isEmpty())
                { emp.setName(name)
                ;
```

```
            }

            System.out.print("Enter new salary (enter 0 to skip): ");
            double salary = scanner.nextDouble();
            if (salary != 0)
               { emp.setSalary(salary);
            }

            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();

    for (int i = 0; i < employees.size(); i++)
        { if (employees.get(i).getId() == id) {
            employees.remove(i);
            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();

    for (Employee emp : employees)
        { if (emp.getId() == 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 to display.");
    } else {
        for (Employee emp : employees)
            { System.out.println(emp);
```

```
            }
        }
    }
}
```

Problem 1.2:

```
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;
    }

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

```
    }
}

public class CardCollection {
    private static Collection<Card> deck = new ArrayList<>();
    private static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args)
        { initializeDeck();

        while (true) {
            System.out.println("\n1. Find cards by symbol");
            System.out.println("2. Display all cards");
            System.out.println("3. Exit");
            System.out.print("Enter your choice: ");

            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch (choice)
                { case 1:
                    findCardsBySymbol();
                    break;
                case 2:
```

```
                displayAllCards();
                break;
            case 3:
                System.exit(0);
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }
}

private static void initializeDeck() {
    String[] symbols = { "Hearts", "Diamonds", "Clubs", "Spades" };
    String[] values = { "Ace", "2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack",
"Queen", "King" };

     for (String symbol : symbols)
     { for (String value : values) {
         deck.add(new Card(symbol, value));
       }
    }
}

private static void findCardsBySymbol() {
    System.out.print("Enter the symbol (Hearts/Diamonds/Clubs/Spades): ");
```

```
    String symbol = scanner.nextLine();

    List<Card> foundCards = new ArrayList<>();
    for (Card card : deck) {
        if (card.getSymbol().equalsIgnoreCase(symbol))
            { foundCards.add(card);
        }
    }

    if (foundCards.isEmpty()) {
        System.out.println("No cards found for the given symbol.");
    } else {
        System.out.println("Cards found:");
        for (Card card : foundCards) {
            System.out.println(card);
        }
    }
}

private static void displayAllCards()
    { for (Card card : deck) {
        System.out.println(card);
    }
}
```

```
}
```

Problem 1.3:

```
import java.util.Scanner;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

class TicketBookingSystem
    { private int availableSeats;
    private int totalBookings;


import java.util.Scanner;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

class TicketBookingSystem
    { private int availableSeats;
    private int totalBookings;

    public TicketBookingSystem(int totalSeats)
        { this.availableSeats = totalSeats;
        this.totalBookings = 0;
    }
```

```
    public synchronized boolean bookTicket(String customerName, int
seatsToBook, boolean isVIP) {
        if (availableSeats >= seatsToBook) {
            System.out.printf("%s %s is booking %d seat(s).\n", isVIP ? "[VIP]" : "",
customerName, seatsToBook);
            try {
                Thread.sleep(500); // Reduced processing time for demonstration
```

```
            } catch (InterruptedException e)
                { Thread.currentThread().interrupt();
            }
            availableSeats -= seatsToBook;
            totalBookings++;
            System.out.printf("%s %s booked %d seat(s) successfully. Remaining
seats: %d\n",
                    isVIP ? "[VIP]" : "", customerName, seatsToBook, availableSeats);
            return true;
        } else {
            System.out.printf("Sorry %s, %d seats are not available. Current available
seats: %d\n",
                    customerName, seatsToBook, availableSeats);
            return false;
        }
    }

    public synchronized void displayStatus() {
        System.out.printf("\nCurrent Status:\nTotal Seats: %d\nAvailable Seats:
%d\nTotal Bookings: %d\n\n",
                availableSeats + totalBookings, availableSeats, totalBookings);
    }
}

class BookingThread implements Runnable {
```

```
    private TicketBookingSystem bookingSystem;
    private String customerName;
    private int seatsToBook;
    private boolean isVIP;

    public BookingThread(TicketBookingSystem bookingSystem, String
customerName, int seatsToBook, boolean isVIP) {
        this.bookingSystem = bookingSystem;
        this.customerName = customerName;
        this.seatsToBook = seatsToBook;
        this.isVIP = isVIP;
    }

    @Override
    public void run() {
        bookingSystem.bookTicket(customerName, seatsToBook, isVIP);
    }
}

public class TicketBookingDemo {
    private static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        TicketBookingSystem bookingSystem = initializeSystem();
```

```
    ExecutorService executor = Executors.newFixedThreadPool(5);

    while (true)
        { displayMenu();
        int choice = getUserChoice();

        switch (choice)
            { case 1:
                processBooking(bookingSystem, executor);
                break;
            case 2:
                bookingSystem.displayStatus();
                break;
            case 3:
                exitSystem(executor);
                return;
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }
}

private static TicketBookingSystem initializeSystem()
    { System.out.print("Enter the total number of seats available: ");
```

```
        int totalSeats = getValidIntInput();
        return new TicketBookingSystem(totalSeats);
    }

    private static void displayMenu()
        { System.out.println("\n1. Book Tickets");
        System.out.println("2. Display Booking Status");
        System.out.println("3. Exit");
        System.out.print("Enter your choice: ");
    }

    private static int getUserChoice()
        { return getValidIntInput();
    }

    private static void processBooking(TicketBookingSystem bookingSystem,
ExecutorService executor) {
        System.out.print("Enter customer name: ");
        String customerName = scanner.next();

        System.out.print("Enter number of seats to book: ");
        int seatsToBook = getValidIntInput();

        System.out.print("Is this a VIP customer? (true/false): ");
```

```
        boolean isVIP = getValidBooleanInput();


        executor.submit(new BookingThread(bookingSystem, customerName,
seatsToBook, isVIP));
    }


    private static void exitSystem(ExecutorService executor)
        { executor.shutdown();
        try {
            if (!executor.awaitTermination(800, TimeUnit.MILLISECONDS))
                { executor.shutdownNow();
            }
        } catch (InterruptedException e)
            { executor.shutdownNow();
        }
        System.out.println("Thank you for using the Ticket Booking System.
Goodbye!");
        scanner.close();
    }


    private static int getValidIntInput()
        { while (!scanner.hasNextInt()) {
            System.out.print("Invalid input. Please enter a number: ");
            scanner.next();
```

```
        }
        return scanner.nextInt();
    }

    private static boolean getValidBooleanInput()
        { while (!scanner.hasNextBoolean()) {
            System.out.print("Invalid input. Please enter true or false: ");
            scanner.next();
        }
        return scanner.nextBoolean();
    }
}
    { this.availableSeats = totalSeats;
        this.totalBookings = 0;
    }

    public synchronized boolean bookTicket(String customerName, int
seatsToBook, boolean isVIP) {
        if (availableSeats >= seatsToBook) {
            System.out.printf("%s %s is booking %d seat(s).\n", isVIP ? "[VIP]" : "",
customerName, seatsToBook);
            try {
                Thread.sleep(500); // Reduced processing time for demonstration
```

```
            } catch (InterruptedException e)
                { Thread.currentThread().interrupt();
            }
            availableSeats -= seatsToBook;
            totalBookings++;
            System.out.printf("%s %s booked %d seat(s) successfully. Remaining
seats: %d\n",
                    isVIP ? "[VIP]" : "", customerName, seatsToBook, availableSeats);
            return true;
        } else {
            System.out.printf("Sorry %s, %d seats are not available. Current available
seats: %d\n",
                    customerName, seatsToBook, availableSeats);
            return false;
        }
    }

    public synchronized void displayStatus() {
        System.out.printf("\nCurrent Status:\nTotal Seats: %d\nAvailable Seats:
%d\nTotal Bookings: %d\n\n",
                availableSeats + totalBookings, availableSeats, totalBookings);
    }
}

class BookingThread implements Runnable {
```

```
    private TicketBookingSystem bookingSystem;
    private String customerName;
    private int seatsToBook;
    private boolean isVIP;

    public BookingThread(TicketBookingSystem bookingSystem, String
customerName, int seatsToBook, boolean isVIP) {
        this.bookingSystem = bookingSystem;
        this.customerName = customerName;
        this.seatsToBook = seatsToBook;
        this.isVIP = isVIP;
    }

    @Override
    public void run() {
        bookingSystem.bookTicket(customerName, seatsToBook, isVIP);
    }
}

public class TicketBookingDemo {
    private static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        TicketBookingSystem bookingSystem = initializeSystem();
```

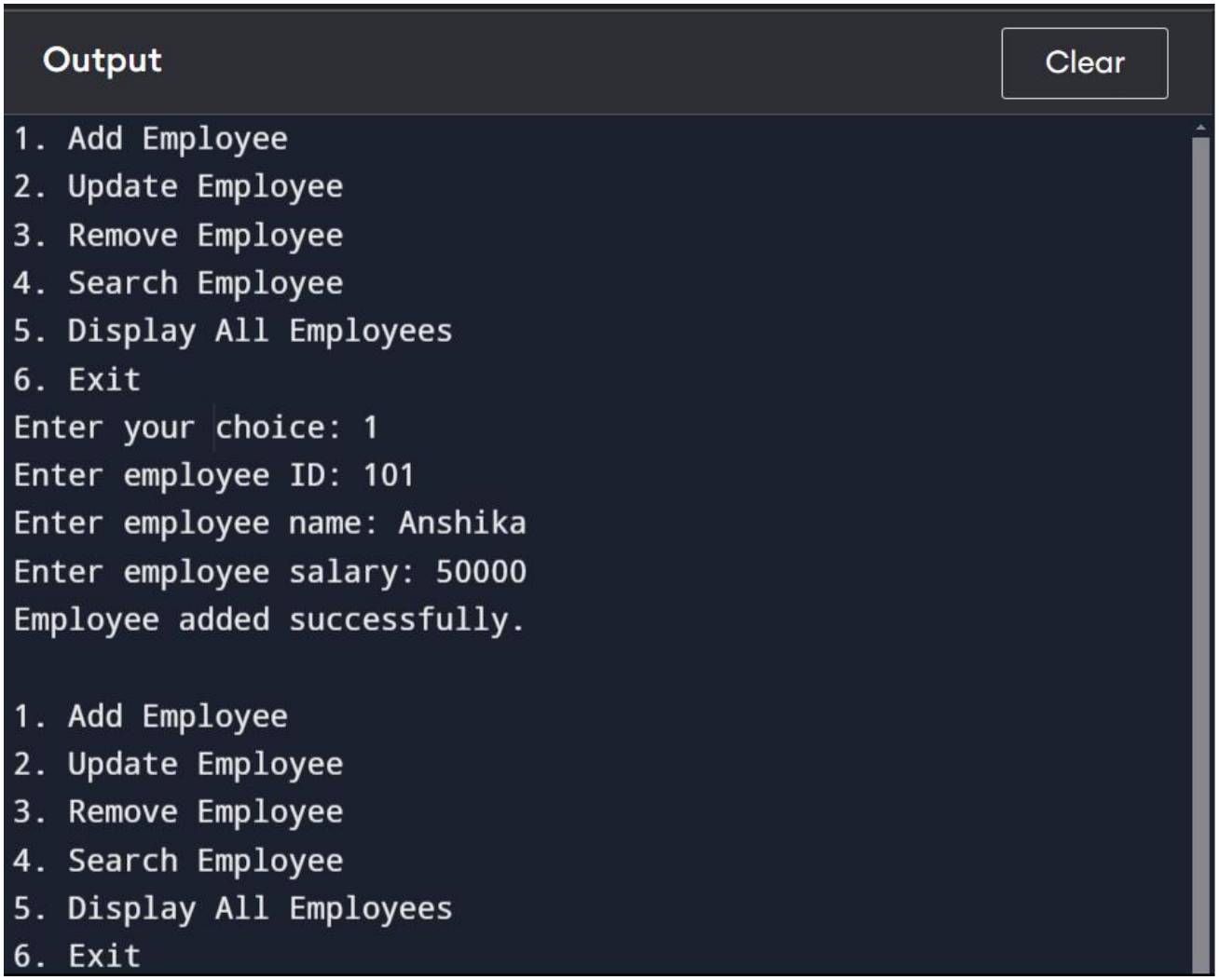
```
    ExecutorService executor = Executors.newFixedThreadPool(5);

    while (true)
        { displayMenu();
        int choice = getUserChoice();

        switch (choice)
            { case 1:
                processBooking(bookingSystem, executor);
                break;
            case 2:
                bookingSystem.displayStatus();
                break;
            case 3:
                exitSystem(executor);
                return;
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }
}

private static TicketBookingSystem initializeSystem()
    { System.out.print("Enter the total number of seats available: ");
```

```
        int totalSeats = getValidIntInput();
        return new TicketBookingSystem(totalSeats);
    }

    private static void displayMenu()
        { System.out.println("\n1. Book Tickets");
        System.out.println("2. Display Booking Status");
        System.out.println("3. Exit");
        System.out.print("Enter your choice: ");
    }

    private static int getUserChoice()
        { return getValidIntInput();
    }

    private static void processBooking(TicketBookingSystem bookingSystem,
ExecutorService executor) {
        System.out.print("Enter customer name: ");
        String customerName = scanner.next();

        System.out.print("Enter number of seats to book: ");
        int seatsToBook = getValidIntInput();

        System.out.print("Is this a VIP customer? (true/false): ");
```

```
        boolean isVIP = getValidBooleanInput();


        executor.submit(new BookingThread(bookingSystem, customerName,
seatsToBook, isVIP));
    }


    private static void exitSystem(ExecutorService executor)
        { executor.shutdown();
        try {
            if (!executor.awaitTermination(800, TimeUnit.MILLISECONDS))
                { executor.shutdownNow();
            }
        } catch (InterruptedException e)
            { executor.shutdownNow();
        }
        System.out.println("Thank you for using the Ticket Booking System.
Goodbye!");
        scanner.close();
    }


    private static int getValidIntInput()
        { while (!scanner.hasNextInt()) {
            System.out.print("Invalid input. Please enter a number: ");
            scanner.next();
```

```
        }
        return scanner.nextInt();
    }

    private static boolean getValidBooleanInput()
        { while (!scanner.hasNextBoolean()) {
            System.out.print("Invalid input. Please enter true or false: ");
            scanner.next();
        }
        return scanner.nextBoolean();
    }
}
```

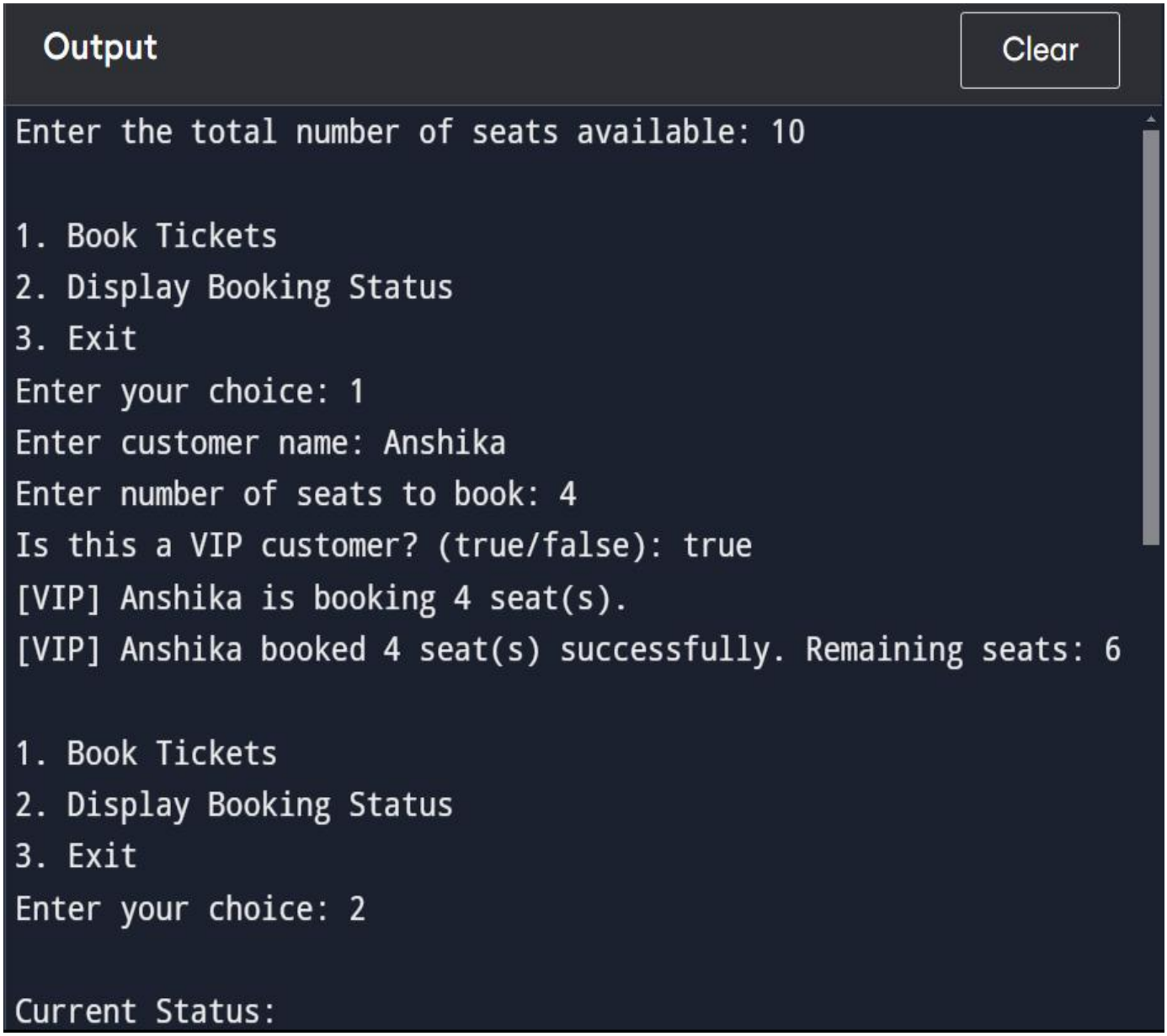
4. Output:



(Fig. 1- Problem 1.1 Output)

```
                                                                    > cd "d:\Semest
" ; if ($?) { javac CardCollection.java } ; if ($?) { java CardCollection }

1. Find cards by symbol
2. Display all cards
3. Exit
Enter your choice: 1
Enter the symbol (Hearts/Diamonds/Clubs/Spades): Hearts
Cards found:
Ace of Hearts
2 of Hearts
3 of Hearts
4 of Hearts
5 of Hearts
6 of Hearts
7 of Hearts
8 of Hearts
9 of Hearts
10 of Hearts
Jack of Hearts
Queen of Hearts
King of Hearts

1. Find cards by symbol
2. Display all cards
3. Exit
```

(Fig. 2- Problem 1.2 Output)



(Fig. 3- Problem 1.3 Output)

5. Learning Outcome:

1. Learn how to use ArrayList to store and manage objects dynamically.
2. Learn how to use the Collection interface to store and retrieve custom objects efficiently.

3. Understand thread synchronization to prevent data inconsistency in concurrent environments.
4. Learn how to prioritize VIP bookings using thread priorities and ExecutorService for better concurrency control.