



Experiment 4

Student Name: Vinod Kumar Saini

UID: 22BCS14967

Branch: CSE

Section/Group: 641/B

Semester: 6th

Date of Performance: 5/03/25

Subject Code: 22CSH-359

Subject Name: PBLJ

A}

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 :** Java program that uses an ArrayList to store employee details (ID, Name, and Salary). The program allows users to add, update, remove, and search for employees.
3. **Algorithms:**
 1. Start
 2. Create an Employee class with attributes:
 3. ID (int), Name (String), Salary (double).
 4. Use an ArrayList to store multiple employees.
 5. Display a menu with options:
 6. Add an Employee
 7. Update Employee Details
 8. Remove an Employee
 9. Search for an Employee
 10. Display All Employees
 11. Exit

4. Code :

```
Main.java
1 import java.util.ArrayList;
2 import java.util.Scanner;
3
4 class Employee {
5     int id;
6     String name;
7     double salary;
8
9     public Employee(int id, String name, double salary) {
10         this.id = id;
11         this.name = name;
12         this.salary = salary;
13     }
14 } public class Main {
15     private static ArrayList<Employee> employees = new ArrayList<>();
16
17     public static void addEmployee(int id, String name, double salary) {
18         Employee employee = new Employee(id, name, salary);
19         employees.add(employee);
20         System.out.println("Employee added successfully.");
21     }
22
23     public static void updateEmployee(int id, String name, double salary) {
24         for (Employee emp : employees) {
25             if (emp.id == id) {
26                 emp.name = name;
27                 emp.salary = salary;
28                 System.out.println("Employee updated successfully.");
29                 return;
30             }
31         }
32         System.out.println("Employee not found.");
33     }
34
35     public static void removeEmployee(int id) {
36         for (Employee emp : employees) {
37             if (emp.id == id) {
```

```
31     }
32     System.out.println("Employee not found.");
33 }
34
35 public static void removeEmployee(int id) {
36     for (Employee emp : employees) {
37         if (emp.id == id) {
38             employees.remove(emp);
39             System.out.println("Employee removed successfully.");
40             return;
41         }
42     }
43     System.out.println("Employee not found.");
44 }
45
46 public static void searchEmployee(int id) {
47     for (Employee emp : employees) {
48         if (emp.id == id) {
49             System.out.println(emp);
50             return;
51         }
52     }
53     System.out.println("Employee not found.");
54 }
55 public static void displayAllEmployees() {
56     if (employees.isEmpty()) {
57         System.out.println("No employees to display.");
58         return;
59     }
60     for (Employee emp : employees) {
61         System.out.println(emp);
62     }
63 }
64
65 public static void main(String[] args) {
66     Scanner scanner = new Scanner(System.in);
67     while (true) {
68         System.out.println("\nEmployee Management System");
69         System.out.println("1. Add Employee");
70         System.out.println("2. Update Employee");
71         System.out.println("3. Remove Employee");
72         System.out.println("4. Search Employee");
73         System.out.println("5. Display All Employees");
74         System.out.println("6. Exit");
75         System.out.print("Enter your choice: ");
76         int choice = scanner.nextInt();
77         scanner.nextLine(); // Consume the newline
```

```
76         int choice = scanner.nextInt();
77         scanner.nextLine(); // Consume the newline
78
79         switch (choice) {
80             case 1:
81                 System.out.print("Enter Employee ID: ");
82                 int id = scanner.nextInt();
83                 scanner.nextLine(); // Consume the newline
84                 String name = scanner.nextLine();
85                 System.out.print("Enter Employee Salary: ");
86                 double salary = scanner.nextDouble();
87                 addEmployee(id, name, salary);
88                 break;
89             case 2:
90                 System.out.print("Enter Employee ID to update: ");
91                 id = scanner.nextInt();
92                 scanner.nextLine();
93                 System.out.print("Enter new Name: ");
94                 name = scanner.nextLine();
95                 System.out.print("Enter new Salary: ");
96                 salary = scanner.nextDouble();
97                 updateEmployee(id, name, salary);
98                 break;
99             case 3:
100                 System.out.print("Enter Employee ID to remove: ");
101                 id = scanner.nextInt();
102                 removeEmployee(id);
103                 break;
104             case 4:
105                 System.out.print("Enter Employee ID to search: ");
106                 id = scanner.nextInt();
107                 searchEmployee(id);
108                 break;
109             case 5:
110                 displayAllEmployees();
111                 break;
112             case 6:
113                 System.out.println("Exiting system...");
114                 scanner.close();
115                 return;
116             default:
117                 System.out.println("Invalid choice. Please try again.");
118         }
119     }
120 }
121 }
122 }
```

5. Output

```
Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Enter your choice: 1
Enter Employee ID: 1001
Enter Employee Name: Vinod
Enter Employee Salary: 50000
Employee added successfully.
```



B}

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:** To develop a Java program using the Collection interface to store and manage playing cards. The program will help users:

Store cards in a collection.

Search for cards by a given symbol (e.g., Hearts, Spades).

Display all available cards in the collection.

3. **Algorithm:**

- Start
- Create a Card class with attributes:
- Symbol (String), Number (String).
- Use a Collection (ArrayList) to store multiple card objects.
- Display a menu with options:
- Add a card.
- Find all cards by symbol.
- Display all stored cards.
- Exit the program

4. Code

```
Main.java
1 import java.util.ArrayList;
2 import java.util.List;
3 import java.util.Scanner;
4
5 class Card {
6     private String rank;
7     private String symbol;
8
9     public Card(String rank, String symbol) {
10         this.rank = rank;
11         this.symbol = symbol;
12     }
13
14     public String getRank() {
15         return rank;
16     }
17
18     public String getSymbol() {
19         return symbol;
20     }
21     @Override public String toString() {
22         return rank + " of " + symbol;
23     }
24 }
25
26 public class Main {
27     private List<Card> cards;
28     public Main() {
29         cards = new ArrayList<>();
30         createDeck(); // Populate the collection with a deck of cards
31     }
32     private void createDeck() {
33         String[] ranks = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King", "Ace"};
34         String[] symbols = {"Hearts", "Diamonds", "Clubs", "Spades"};
35
36         for (String symbol : symbols) {
37             for (String rank : ranks) {
38                 cards.add(new Card(rank, symbol));
39             }
40         }
41     }
42     public void findCardsBySymbol(String symbol) {
43         boolean found = false;
44         for (Card card : cards) {
45             if (card.getSymbol().equalsIgnoreCase(symbol)) {
46                 System.out.println(card); // Print the card
47                 found = true;
48             }
49         }
50     }
51 }
```

```

44-         for (Card card : cards) {
45-             if (card.getSymbol().equalsIgnoreCase(symbol)) {
46-                 System.out.println(card); // Print the card
47-                 found = true;
48-             }
49-         }
50-         if (!found) {
51-             System.out.println("No cards found with the symbol: " + symbol);
52-         }
53-     }
54-     public void displayAllCards() {
55-         for (Card card : cards) {
56-             System.out.println(card);
57-         }
58-     }
59-
60-     public static void main(String[] args) {
61-         Scanner scanner = new Scanner(System.in);
62-         Main cardCollection = new Main();
63-
64-         while (true) {
65-             System.out.println("\nCard Collection Menu");
66-             System.out.println("1. Display all cards");
67-             System.out.println("2. Find cards by symbol");
68-             System.out.println("3. Exit");
69-             System.out.print("Enter your choice: ");
70-             int choice = scanner.nextInt();
71-             scanner.nextLine();
72-
73-             switch (choice) {
74-                 case 1:
75-                     cardCollection.displayAllCards();
76-                     break;
77-                 case 2:
78-                     System.out.print("Enter the symbol to search for (Hearts, Diamonds, Clubs, Spades): ");
79-                     String symbol = scanner.nextLine();
80-                     cardCollection.findCardsBySymbol(symbol);
81-                     break;
82-                 case 3:
83-                     System.out.println("Exiting the program...");
84-                     scanner.close();
85-                     return;
86-                 default:
87-                     System.out.println("Invalid choice. Please try again.");
88-             }
89-         }
90-     }

```

4. Output

```

Card Collection Menu
1. Display all cards
2. Find cards by symbol
3. Exit
Enter your choice: 2
Enter the symbol to search for (Hearts, Diamonds, Clubs, Spades): Clubs
2 of Clubs
3 of Clubs
4 of Clubs
5 of Clubs
6 of Clubs
7 of Clubs
8 of Clubs
9 of Clubs
10 of Clubs
Jack of Clubs
Queen of Clubs
King of Clubs

```

C}

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. Code :

```
Main.java
1 class TicketBookingSystem {
2     private boolean[] seats; // Array to store seat availability
3
4     // Constants for seat availability
5     private static final boolean BOOKED = true;
6     private static final boolean AVAILABLE = false;
7
8     public TicketBookingSystem(int totalSeats) {
9         seats = new boolean[totalSeats]; // Initialize all seats to available (false)
10    }
11
12    // Synchronized method to book a seat
13    public synchronized boolean bookSeat(int seatNumber, String customerType) {
14        if (seatNumber < 0 || seatNumber >= seats.length) {
15            System.out.println("Invalid seat number.");
16            return false;
17        }
18
19        if (seats[seatNumber] == BOOKED) {
20            System.out.println(customerType + " failed to book Seat " + seatNumber + ". Seat already booked.");
21            return false;
22        } else {
23            // Book the seat
24            seats[seatNumber] = BOOKED;
25            System.out.println(customerType + " successfully booked Seat " + seatNumber);
26            return true;
27        }
28    }
29
30    public void displayAvailableSeats() {
31        System.out.println("Available Seats:");
32        for (int i = 0; i < seats.length; i++) {
33            if (seats[i] == AVAILABLE) {
34                System.out.println("Seat " + i + " is available.");
35            }
36        }
37    }
38 }
39
40 class BookingThread extends Thread {
41     private TicketBookingSystem system;
42     private int seatNumber;
43     private String customerType;
44
45     public BookingThread(TicketBookingSystem system, int seatNumber, String customerType) {
46         this.system = system;
47         this.seatNumber = seatNumber;
```



```
Main.java
45 public BookingThread(TicketBookingSystem system, int seatNumber, String customerType) {
46     this.system = system;
47     this.seatNumber = seatNumber;
48     this.customerType = customerType;
49 }
50
51 @Override
52 public void run() {
53     // Try to book the seat, if unsuccessful, retry
54     while (!system.bookSeat(seatNumber, customerType)) {
55         try {
56             Thread.sleep(500); // Simulate some waiting time before retrying
57         } catch (InterruptedException e) {
58             e.printStackTrace();
59         }
60     }
61 }
62 }
63
64 public class Main {
65     public static void main(String[] args) {
66         // Create a ticket booking system with 10 seats
67         TicketBookingSystem system = new TicketBookingSystem(10);
68
69         // Display available seats
70         system.displayAvailableSeats();
71
72         // Create normal and VIP customers (Threads)
73         BookingThread normalCustomer1 = new BookingThread(system, 5, "Normal Customer 1");
74         BookingThread normalCustomer2 = new BookingThread(system, 5, "Normal Customer 2");
75         BookingThread vipCustomer1 = new BookingThread(system, 2, "VIP Customer 1");
76         BookingThread normalCustomer3 = new BookingThread(system, 3, "Normal Customer 3");
77         BookingThread vipCustomer2 = new BookingThread(system, 4, "VIP Customer 2");
78
79         // Set priority: VIP customers should have a higher chance of booking
80         vipCustomer1.setPriority(Thread.MAX_PRIORITY);
81         vipCustomer2.setPriority(Thread.MAX_PRIORITY);
82         normalCustomer1.setPriority(Thread.NORM_PRIORITY);
83         normalCustomer2.setPriority(Thread.NORM_PRIORITY);
84         normalCustomer3.setPriority(Thread.NORM_PRIORITY);
85
86         // Start the threads (customers trying to book seats)
87         vipCustomer1.start();
88         vipCustomer2.start();
89         normalCustomer1.start();
90         normalCustomer2.start();
91         normalCustomer3.start();
92     }
93 }
```

```
85
86 // Start the threads (customers trying to book seats)
87 vipCustomer1.start();
88 vipCustomer2.start();
89 normalCustomer1.start();
90 normalCustomer2.start();
91 normalCustomer3.start();
92
93 try {
94     // Wait for all threads to finish
95     vipCustomer1.join();
96     vipCustomer2.join();
97     normalCustomer1.join();
98     normalCustomer2.join();
99     normalCustomer3.join();
100 } catch (InterruptedException e) {
101     e.printStackTrace();
102 }
103
104 // Display available seats after booking
105 system.displayAvailableSeats();
106 }
107 }
108
```

3. Screenshot of Outputs:

```
Available Seats:
Seat 0 is available.
Seat 1 is available.
Seat 2 is available.
Seat 3 is available.
Seat 4 is available.
Seat 5 is available.
Seat 6 is available.
Seat 7 is available.
Seat 8 is available.
Seat 9 is available.
VIP Customer 1 successfully booked Seat 2
Normal Customer 3 successfully booked Seat 3
Normal Customer 2 successfully booked Seat 5
Normal Customer 1 failed to book Seat 5. Seat already booked.
VIP Customer 2 successfully booked Seat 4
Normal Customer 1 failed to book Seat 5. Seat already booked.
Normal Customer 1 failed to book Seat 5. Seat already booked.
```

4. Learning Outcomes:

- Understanding of Object-Oriented Programming (OOP)
- Real-world Application of Software Design Principles.
- Threading in java.