



Experiment-4

Student Name: Goutam

UID: 22BCS15338

Branch: CSE

Section/Group: 22BCS_IOT-640/A

Semester: 6th

Date of Performance: 17/02/2025

Subject Name: Project Based Learning in Java with 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.
- 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.
- 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:

- For the Employee Management System:** To implement an ArrayList for storing and managing employee details dynamically.
- For the Card Collection System:** To use the Collection interface for organizing and retrieving cards based on symbols.
- For the Ticket Booking System:** To develop a synchronized, thread-safe ticket booking system with VIP priority handling.

3. Procedure:

a) For the Employee Management System:

- Take user input for employee details and store them in an ArrayList.
- Provide options to add, update, remove, and search employees.
- Implement a menu-driven system for user interaction.
- Use loops and conditions to manage and modify employee records. □ Display appropriate messages for successful or failed operations.

b) For the Card Collection System:

- Create a collection to store different types of cards.
- Allow users to add and retrieve cards based on their symbols.
- Implement search functionality to filter cards by symbol.
- Use Java's Collection framework for efficient data handling.
- Display matching cards or an appropriate message if none are found.

c) For the Ticket Booking System:

- Initialize seat booking with synchronized threads to prevent double booking.
- Assign higher priority to VIP bookings for faster processing.
- Allow multiple users to attempt bookings concurrently.
- Ensure only one thread can book a seat at a time using synchronization.
- Display booking status after each transaction.

4. Code:



```
public class EmployeeApp { public
    static void main(String[] args) {
        ArrayList<Employee> employees = new ArrayList<>();
        Scanner sc = new Scanner(System.in);

        while (true) {
            System.out.println("\nMenu: 1. Add 2. Update 3. Remove 4. Search 5. List 6. Exit");
            System.out.print("Enter choice:
"); int choice = sc.nextInt(); if
(choice == 6) break; switch
(choice) {
    case 1:
        System.out.print("Enter id, name, and salary:
"); int id = sc.nextInt(); String name = sc.next();
        double salary = sc.nextDouble();
        employees.add(new Employee(id, name, salary));
        break;
    case 2:
        System.out.print("Enter id to update:
"); id = sc.nextInt(); boolean updated =
false; for (Employee e : employees) {
            if (e.id == id) {
                System.out.print("New name: ");
                e.name = sc.next();
                System.out.print("New salary: ");
```

```
        e.salary =  
        sc.nextDouble(); updated  
        = true; break;  
    }  
}  
if (!updated) System.out.println("Employee not found!");  
break;  
case 3:  
    System.out.print("Enter id to remove: ");  
    id = sc.nextInt(); boolean removed =  
    false; for (int i = 0; i < employees.size();  
    i++) {  
        if (employees.get(i).id == id) {  
            employees.remove(i);  
            removed = true;  
            break;  
        }  
    }  
    if (!removed) System.out.println("Employee not found!");  
    break;  
case 4:  
    System.out.print("Enter id to search:  
    "); id = sc.nextInt(); boolean found =  
    false; for (Employee e : employees) {  
        if (e.id == id) {  
            System.out.println(e);  
            found = true;  
        }  
    }  
    if (!found) System.out.println("Employee not found!");  
    break;  
case 5:  
    System.out.println("Employee List:");  
    for (Employee e : employees) {  
        System.out.println(e);  
    }  
    break;  
default:  
    System.out.println("Invalid choice!");  
}  
}  
sc.close();  
}  
}
```

b) Card Collection Using a Collection:

```
import java.util.*; public class
CardCollection { public static void
main(String[] args) {
    // Map where the key is a card symbol and the value is a list of card names.
    Map<String, List<String>> cards = new HashMap<>();
    Scanner sc = new Scanner(System.in);

    while (true) {
        System.out.println("\nMenu: 1. Add Card 2. Find Cards by Symbol 3. Exit");
        System.out.print("Choice: "); int
        choice = sc.nextInt();
        sc.nextLine(); // consume
        newline if (choice == 3) break;

        if (choice == 1) {
            System.out.print("Enter card symbol: ");
            String symbol = sc.nextLine();
            System.out.print("Enter card name: ");
            String cardName = sc.nextLine(); // Create a
            new list if symbol doesn't exist
            cards.putIfAbsent(symbol, new ArrayList<>());
            cards.get(symbol).add(cardName);
            System.out.println("Card added!");
        } else if (choice == 2) {
            System.out.print("Enter card symbol to search: ");
            String symbol = sc.nextLine();
            List<String> list = cards.get(symbol);
            if (list == null || list.isEmpty()) {
                System.out.println("No cards found for symbol: " + symbol);
            } else {
                System.out.println("Cards with symbol " + symbol + ": " + list);
            }
        } else {
            System.out.println("Invalid choice!");
        }
    }
    sc.close();
}
```

c) Ticket Booking System with Synchronized Threads:

```
class TicketBookingSystem { int availableSeats = 5;
```

```
// Synchronized method to ensure one thread at a time can book a
seat. public synchronized void bookSeat(String passenger) { if
(availableSeats > 0) {
    System.out.println(passenger + " booked seat number " + availableSeats); availableSeats--;
    ;
} else {
    System.out.println(passenger + " found no seats available.");
}
}
}
}
}
class BookingThread extends Thread {
    TicketBookingSystem system;
    BookingThread(TicketBookingSystem system, String name, int priority) {
        super(name);
        this.system = system;
        setPriority(priority);
    }
    public void run() {
        system.bookSeat(getName());
    }
}
}
public class TicketBooking { public
    static void main(String[] args) {
        TicketBookingSystem system = new TicketBookingSystem();
        BookingThread t1 = new BookingThread(system, "VIP 1", Thread.MAX_PRIORITY);
        BookingThread t2 = new BookingThread(system, "VIP 2", Thread.MAX_PRIORITY);
        BookingThread t3 = new BookingThread(system, "Normal 1", Thread.NORM_PRIORITY);
        BookingThread t4 = new BookingThread(system, "Normal 2", Thread.NORM_PRIORITY);
        BookingThread t5 = new BookingThread(system, "Normal 3", Thread.NORM_PRIORITY);
        BookingThread t6 = new BookingThread(system, "Normal 4", Thread.NORM_PRIORITY);
        t1.start(); t2.start(); t3.start(); t4.start(); t5.start(); t6.start();
    }
}
}
```

5. Outputs:

a) Output for Employee Management System: -

```

PS D:\6th Sem Content\Java Lab\Java codes> cd "d:\6th Sem Content\
mployeeApp }

Menu: 1. Add  2. Update  3. Remove  4. Search  5. List  6. Exit
Enter choice: 1
Enter id, name, and salary: 17253 Mohan 201200

Menu: 1. Add  2. Update  3. Remove  4. Search  5. List  6. Exit
Enter choice: 1
Enter id, name, and salary: 15845 Mohit 500000

Menu: 1. Add  2. Update  3. Remove  4. Search  5. List  6. Exit
Enter choice: 2
Enter id to update: 17253
New name: Manoj
New salary: 200000

Menu: 1. Add  2. Update  3. Remove  4. Search  5. List  6. Exit
Enter choice: 4
Enter id to search: 17253
17253: Manoj ($200000.0)

Menu: 1. Add  2. Update  3. Remove  4. Search  5. List  6. Exit
Enter choice: 5
Employee List:
17253: Manoj ($200000.0)
15845: Mohit ($500000.0)

Menu: 1. Add  2. Update  3. Remove  4. Search  5. List  6. Exit
Enter choice: 6
PS D:\6th Sem Content\Java Lab\Java codes\Experiment 4>

```

b) Output for Card Collection Using a Collection: -

```

PS D:\6th Sem Content\Java Lab\Java codes> cd "d:\6th S
dCollection }

Menu: 1. Add Card  2. Find Cards by Symbol  3. Exit
Choice: 1
Enter card symbol: )(
Enter card name: Parenthesis
Card added!

Menu: 1. Add Card  2. Find Cards by Symbol  3. Exit
Choice: 1
Enter card symbol: !
Enter card name: Exclamation
Card added!

Menu: 1. Add Card  2. Find Cards by Symbol  3. Exit
Choice: 2
Enter card symbol to search: )(
Cards with symbol )( : [Parenthesis]

Menu: 1. Add Card  2. Find Cards by Symbol  3. Exit
Choice: 2
Enter card symbol to search: !
Cards with symbol ! : [Exclamation]

Menu: 1. Add Card  2. Find Cards by Symbol  3. Exit
Choice: 3
PS D:\6th Sem Content\Java Lab\Java codes\Exp 4.2>

```

c) Output for Ticket Booking System with Synchronized Threads: -

```
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\6th Sem Content\Java Lab\Java codes> cd "d:\6th Sem Content\Java Lab\Experiment 4\Exp 4.3"
) { java TicketBooking }
VIP 1 booked seat number 5
Normal 2 booked seat number 4
Normal 3 booked seat number 3
Normal 1 booked seat number 2
VIP 2 booked seat number 1
Normal 4 found no seats available.
PS D:\6th Sem Content\Java Lab\Java codes\Experiment 4\Exp 4.3>
```

6. Learning Outcomes:

- Understand and implement Java Collections for efficient data storage and retrieval.
- Develop user-friendly programs with menu-driven interactions for dynamic data management.
- Apply exception handling to ensure robustness and prevent runtime errors.
- Utilize multithreading and synchronization to manage concurrent processes safely.
- Enhance problem-solving skills by designing real-world applications with Java.