# Experiment 4

**Student Name:** Sukhleen Kaur **UID:** 22BCS14011

**Branch:** BE CSE **Section/Group:**22BCSKRG_IOT3B

**Semester:** 6th **Date of Performance:**19/02/2025

**Subject Name:** Programming in java lab **Subject Code:** 22ITH-359

## 1. Array List Implementation for Employee Details

### (Easy) Aim :

To implement an Array List to store employee details such as ID, Name, and Salary and allow users to perform CRUD operations like adding, updating, removing, and searching employees.

### Objective :

- To understand the use of the ArrayList class in Java.
- To perform basic operations (add, update, remove, search) on an ArrayList. • To learn how to create a class representing an employee and use it in an ArrayList.

### Implementation/Code :

```
package employee;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
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 void setName(String name) {
this.name= name;
}
public double getSalary() {
return salary;
}
public void setSalary(double salary) {
this.salary = salary;
}
@Override
public String toString() { return "ID: " + id + ", Name: " +
name + ", Salary: " + salary;
}
}
public class EmployeeManagementSystem { private static final
Scanner scanner = new Scanner(System.in);
private static final List<Employee> employees = new ArrayList<>();
public static void main(String[] args) {
while (true) {
System.out.println("\nEmployee 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");
```

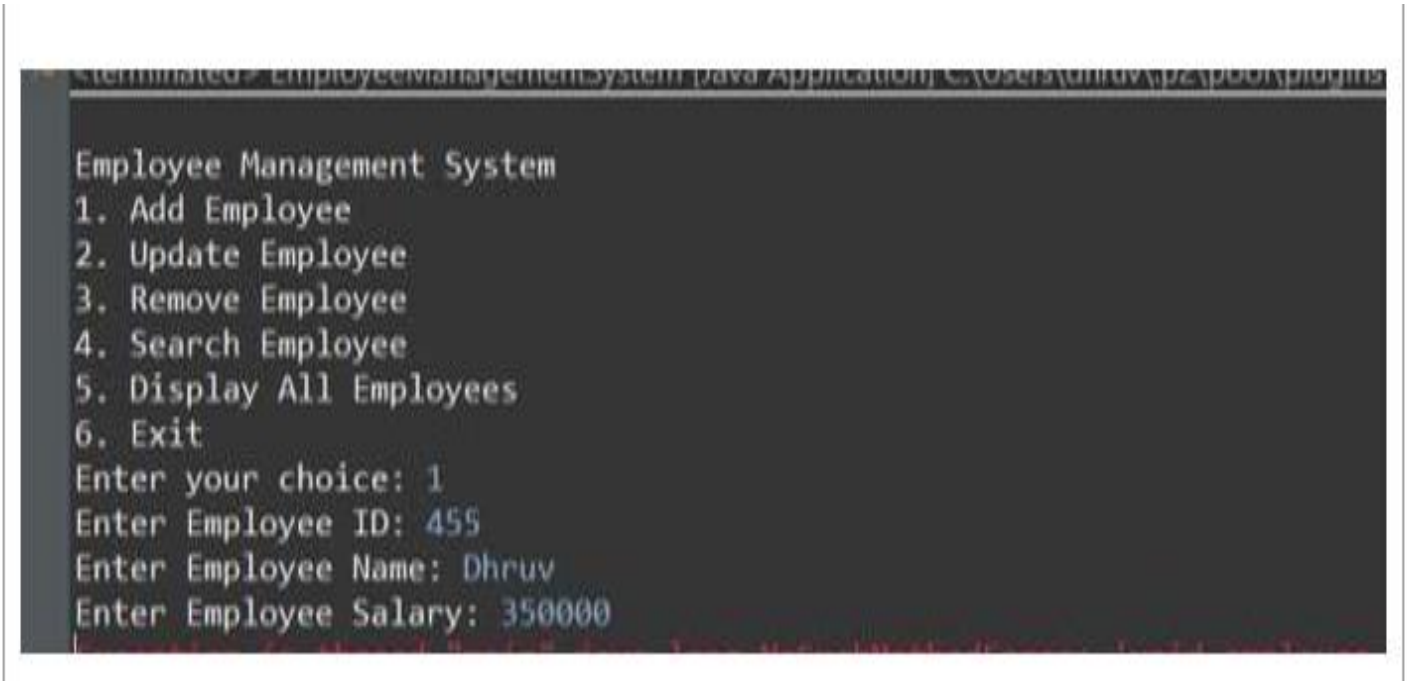
```
System.out.println("5. Display All Employees");
System.out.println("6. Exit"); System.out.print("Enter
your choice: "); int choice = scanner.nextInt();
scanner.nextLine();
switch (choice) { case 1 ->
addEmployee(); case 2 ->
updateEmployee(); case 3 ->
removeEmployee(); case 4 ->
searchEmployee(); case 5 ->
displayEmployees(); case 6 -
> exitProgram();
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();
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();
for (Employee emp : employees) {
if (emp.getId()
== id) {
System.out.print("Enter new Name: ");
emp.setName(scanner.nextLine());
System.out.print("Enter new Salary: ");
emp.setSalary(scanner.nextDouble());
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();
Iterator<Employee> iterator = employees.iterator();
while (iterator.hasNext()) {
 if (iterator.next().getId() == id) {
iterator.remove();
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(emp);
return;
}
}
System.out.println("Employee not found.");
}
private static void displayEmployees() { if
(employees.isEmpty()) {
System.out.println("No employees to display.");
} else {
employees.forEach(System.out::println);
}
}
```

```
private static void exitProgram() {
System.out.println("Exiting program.");
scanner.close();
System.exit(0);
}
}
```

**Output :**



## 2. Card Collection System Using Collection Interface (Medium)

### Aim :

To create a program that collects and stores all cards in a collection to assist users in finding all the cards of a given symbol using the Collection interface.

### Objective :

- To learn how to use the Collection interface in Java.
- To understand how to manage a collection of objects and filter based on certain attributes. • To practice the use of iteration and filtering in Java collections.

### Implementation/Code :

```
package exp4; import
java.util.*;
import java.util.stream.Collectors;
class Card
{ private final String suit;
private final String rank;
 public Card(String suit, String rank)
{
this.suit = suit;
this.rank
= rank;
}
public String getSuit() {
return suit; }
@Override
public String toString() {
 return rank + " of " + suit;
} }
public class CardCollectionSystem {
public static void main(String[] args)
{ List<Card> deck = Arrays.asList(
new Card("Hearts", "Ace"),
new Card("Diamonds", "King"),
new Card("Clubs", "Queen"),
```

```
new Card("Spades", "Jack"),
new Card("Hearts", "10"),
 new Card("Diamonds", "2"),
new Card("Spades", "Ace")

);
try (Scanner scanner = new Scanner(System.in)) {
System.out.print("Enter the suit (Hearts, Diamonds, Clubs, Spades) to find cards: "); String
suit = scanner.nextLine().trim();

List<Card> filteredCards = deck.stream()
.filter(card -> card.getSuit().equalsIgnoreCase(suit)) .collect(Collectors.toList());

if (filteredCards.isEmpty()) {
System.out.println("No cards found for the suit: " + suit);
} else {
System.out.println("Cards of suit " + suit + ":"); filteredCards.forEach(System.out::println);
}
}
}
}
```

**Output :**

## 3. Ticket Booking System with Synchronized Threads(Hard)

### Aim :

To develop a ticket booking system where multiple threads ensure no double booking occurs, using synchronized methods, and simulate VIP booking by using thread priorities.

### Objective :

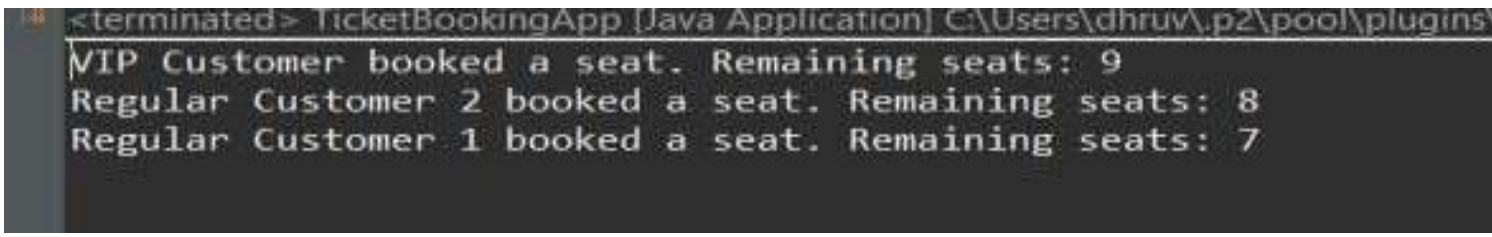
- To understand thread synchronization in Java to prevent data inconsistency.
- To implement thread priorities for VIP bookings.
- To learn how to manage concurrent access to shared resources in a multithreaded environment.

### Implementation/Code :

```
package exp4;
import java.util.concurrent.locks.ReentrantLock;
class TicketBookingSystem { private
int availableSeats = 10;
private final ReentrantLock lock = new ReentrantLock();
public void bookTicket(String customerName) {
lock.lock();
try {
if (availableSeats > 0) {
System.out.println(customerName + " booked a seat. Remaining seats: " + (availableSeats - 1));
availableSeats--;
} else {
System.out.println(customerName + " attempted to book, but no seats available."); }
} finally { lock.unlock();
} }
public int getAvailableSeats() { return
availableSeats;
}
}
class BookingThread extends Thread {
private final TicketBookingSystem system;
private final String customerName;
```

```
public BookingThread(TicketBookingSystem system, String customerName) { this.system
= system;
this.customerName = customerName;
}
@Override
public void run() {
system.bookTicket(customerName);
} }
public class TicketBookingApp {
public static void main(String[] args) {
TicketBookingSystem system = new TicketBookingSystem();
Thread vipThread = new BookingThread(system, "VIP Customer");
vipThread.setPriority(Thread.MAX_PRIORITY);
Thread regularThread1 = new BookingThread(system, "Regular Customer 1");
Thread regularThread2 = new BookingThread(system, "Regular Customer 2");
vipThread.start(); regularThread1.start();
regularThread2.start();
try { vipThread.join();
regularThread1.join();
regularThread2.join();
} catch (InterruptedException e) {
System.err.println("Thread interrupted: " + e.getMessage());
}  }
}
```

**Output :**



## Learning Outcomes :

- Understanding synchronization in Java and its importance in multithreaded applications.
- Gaining knowledge of thread priorities and how to manage them.
- Learning how to safely share and modify data across multiple threads.