Experiment 4

Student Name: Aman Pratap Singh **UID:**22BCS16078
Branch: CSE **Section/Group:** 619-B

Problem 1: Employee Management System (Easy Level)

Description:

Write a Java program to implement an ArrayList that stores employee details (ID, Name, and Salary). The program should allow users to:

- **Add a new employee**
- **Update an existing employee**
- **Remove an employee**
- **Search for an employee by ID**

```
import java.util.*;
class Employee {
int id;
String name;
double salary;
Employee(int id, String name, double salary) {
this.id = id;
this.name = name;
this.salary = salary;
}
void display() {
System.out.println("ID: " + id + ", Name: " + name + ", Salary: " + salary);
}
}
public class EmployeeManager {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
List<Employee> employeeList = new ArrayList<>();
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("Choose an option: ");
            int choice = sc.nextInt();
sc.nextLine();

            if (choice == 1) {
                System.out.print("Enter ID: ");
                int id = sc.nextInt();
sc.nextLine();
                System.out.print("Enter Name: ");
                String name = sc.nextLine();
System.out.print("Enter Salary: ");
double salary = sc.nextDouble();
employeeList.add(new Employee(id, name, salary));
System.out.println("Employee added successfully.");
            } else if (choice == 2) {

                System.out.print("Enter Employee ID to update: ");
                int idToUpdate = sc.nextInt();
                sc.nextLine();
boolean found = false;
for (Employee emp : employeeList) {
if (emp.id == idToUpdate) {
                        System.out.print("Enter new Name: ");
emp.name = sc.nextLine();
                        System.out.print("Enter new Salary: ");
emp.salary = sc.nextDouble();
                        System.out.println("Employee updated successfully.");
found = true;
break;
                    }
}
                if (!found) {
                    System.out.println("Employee not found.");
```

```
            }
        } else if (choice == 3) {

System.out.print("Enter Employee ID to remove: ");
int idToRemove = sc.nextInt();
boolean removed = false;
            Iterator<Employee> iterator = employeeList.iterator();
            while (iterator.hasNext()) {
Employee emp = iterator.next();
if (emp.id == idToRemove) {
iterator.remove();
System.out.println("Employee removed successfully.");
 removed = true;
break;
    }
 }
 if (!removed) {
 System.out.println("Employee not found.");
      }
}
else if (choice == 4) {

 System.out.print("Enter Employee ID to search: ");
 int idToSearch = sc.nextInt();
boolean found = false;
for (Employee emp : employeeList) {
if (emp.id == idToSearch) {
emp.display();
found = true;
break;
  }
}
if (!found) {
System.out.println("Employee not found.");
}
} else if (choice == 5) {

  if (employeeList.isEmpty()) {
```

```
 System.out.println("No employees to display.");
  } else {
                System.out.println("\nEmployee Details:");
for (Employee emp : employeeList) {
                    emp.display();
                }
            }
        } else if (choice == 6) {
System.out.println("Exiting the program.");
break;
        } else {
            System.out.println("Invalid choice. Please try again.");
        }
    }
    sc.close();
  }
}

}
}
```

Output:

```
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: 16458
Enter Name: Masumi Gupta
Enter Salary: 50000
Employee added successfully!

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Enter your choice:
```

Problem 2: Card Collection System (Medium Level)
Description:
Create a Java program that uses the **Collection interface** to collect and store **cards**. The program should assist users in:

- **Adding cards** (Rank and Symbol)
- **Searching cards by symbol**
- **Displaying all stored cards**

Code:

```
import java.util.*;

// Card class to represent a playing card
class Card {
    private String symbol;
    private String rank;

    public Card(String symbol, String rank) {
        this.symbol = symbol;
        this.rank = rank;
    }

    public String getSymbol() {
        return symbol;
    }

    public String getRank() {
        return rank;
    }

    @Override
    public String toString() {
        return rank + " of " + symbol;
    }
}

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

```
public void addCard() {
    System.out.print("Enter Symbol (Hearts, Diamonds, etc.): ");
    String symbol = scanner.nextLine();
    System.out.print("Enter Rank (Ace, 2, King, etc.): ");
    String rank = scanner.nextLine();

    cards.add(new Card(symbol, rank));
    System.out.println("Card added successfully!");
}
public void searchBySymbol() {
    System.out.print("Enter Symbol to search: ");
    String symbol = scanner.nextLine();

    boolean found = false;
    for (Card card : cards) {
        if (card.getSymbol().equalsIgnoreCase(symbol)) {
            System.out.println(card);
            found = true;
        }
    }

    if (!found) {
        System.out.println("No cards found with symbol: " + symbol);
    }
}

public void displayAllCards() {
    if (cards.isEmpty()) {
        System.out.println("No cards in the collection.");
    } else {
        System.out.println("Stored Cards:");
        for (Card card : cards) {
            System.out.println(card);
        }
    }
}
public void menu() {
    while (true) {
        System.out.println("\n1. Add Card");
```

```
            System.out.println("2. Search by Symbol");
            System.out.println("3. Display All Cards");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");

            int choice = scanner.nextInt();
            scanner.nextLine();

            switch (choice) {
                case 1:
                    addCard();
                    break;
                case 2:
                    searchBySymbol();
                    break;
                case 3:
                    displayAllCards();
                    break;
                case 4:
                    System.out.println("Exiting...");
                    return;
                default:
                    System.out.println("Invalid choice! Please try again.");
            }
        }
    }
    public static void main(String[] args) {
        CardCollection cardCollection = new CardCollection();
        cardCollection.menu();
    }
}
```

Output:

```
1. Add Card
2. Search by Symbol
3. Display All Cards
4. Exit
Enter your choice: 1
Enter Symbol (Hearts, Diamonds, etc.): hearts
Enter Rank (Ace, 2, King, etc.): ace
Card added successfully!

1. Add Card
2. Search by Symbol
3. Display All Cards
4. Exit
Enter your choice: 1
Enter Symbol (Hearts, Diamonds, etc.): diamond
Enter Rank (Ace, 2, King, etc.): king
Card added successfully!

1. Add Card
2. Search by Symbol
3. Display All Cards
4. Exit
Enter your choice: 3
Stored Cards:
ace of hearts
king of diamond
```

Problem 3: Ticket Booking System (Hard Level)
Description:
Develop a **multi-threaded ticket booking system** that ensures **synchronized seat booking** to prevent double booking. Use **thread priorities** to give preference to **VIP bookings**.
Features:

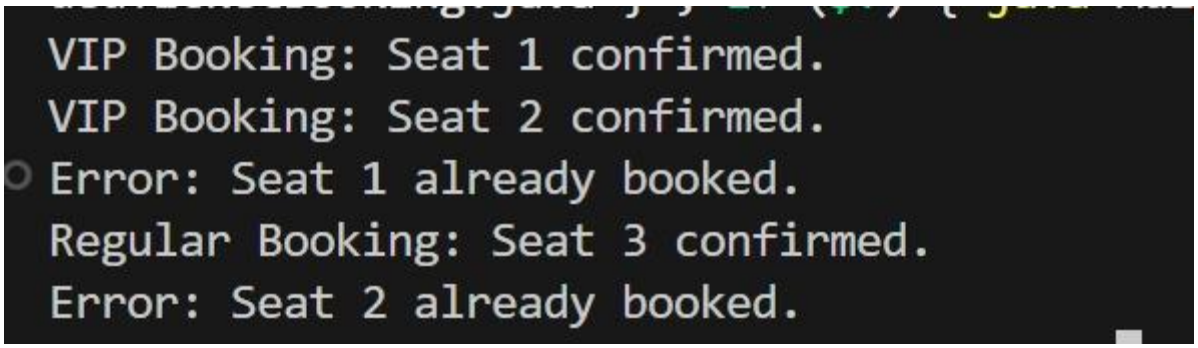
- Multiple users booking tickets simultaneously
- Synchronization to prevent double booking
- VIP customers have **higher priority**

Code:

```
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

class TicketBookingSystem {
    private boolean[] seats;
    private final Lock lock = new ReentrantLock();

    public TicketBookingSystem(int totalSeats) {
        seats = new boolean[totalSeats];
    }

    public void bookSeat(int seatNumber, String customerType) {
        lock.lock();
        try {
            if (seatNumber < 0 || seatNumber >= seats.length) {
                System.out.println(customerType + " Booking: Invalid seat number.");
                return;
            }
            if (!seats[seatNumber]) {
                seats[seatNumber] = true;
                System.out.println(customerType + " Booking: Seat " + (seatNumber + 1) + " confirmed.");
            } else {
                System.out.println("Error: Seat " + (seatNumber + 1) + " already booked.");
            }
        } finally {
            lock.unlock();
```

```
        }
    }
}
class BookingThread extends Thread {
    private TicketBookingSystem system;
    private int seatNumber;
    private String customerType;

    public BookingThread(TicketBookingSystem system, int seatNumber, String customerType, int
priority) {
        this.system = system;
        this.seatNumber = seatNumber;
        this.customerType = customerType;
        this.setPriority(priority);
    }

    public void run() {
        system.bookSeat(seatNumber, customerType);
    }
}
public class MultiThreadedTicketBooking {
    public static void main(String[] args) {
        TicketBookingSystem system = new TicketBookingSystem(10);

        BookingThread vip1 = new BookingThread(system, 0, "VIP", Thread.MAX_PRIORITY);
        BookingThread vip2 = new BookingThread(system, 1, "VIP", Thread.MAX_PRIORITY);
        BookingThread regular1 = new BookingThread(system, 0, "Regular",
Thread.MIN_PRIORITY);
        BookingThread regular2 = new BookingThread(system, 1, "Regular",
Thread.MIN_PRIORITY);
        BookingThread regular3 = new BookingThread(system, 2, "Regular",
Thread.NORM_PRIORITY);

        vip1.start();
        regular1.start();
        vip2.start();
        regular2.start();
        regular3.start();
    }
}
```

Output:


VIP Booking: Seat 1 confirmed.
VIP Booking: Seat 2 confirmed.
Error: Seat 1 already booked.
Regular Booking: Seat 3 confirmed.
Error: Seat 2 already booked.