Experiment-4

Student Name: Aniya kumari
UID: 22BCS16932
Branch: BE-CSE
Section/Group: 618-B
Semester: 6th
Date of Performance: 21/02/25
Subject Name: Project Based Learning in Java
Subject Code: 22CSH-359

EASY:

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.

Objective: The Employee Management System aims to implement an ArrayList to store and manage employee details, including ID, Name, and Salary. The program allows users to efficiently add, update, remove, and search for employees through a menu-driven interface. This ensures easy management and retrieval of employee records in a structured manner.

Implementation/Code:

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


class Employee {
int id;
   String name;
double salary;


   public Employee(int id, String name, double salary) {
this.id = id;        this.name = name;
```

```
        this.salary = salary;
    }

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

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

    public static void addEmployee() {
System.out.print("Enter ID: ");         int id =
scanner.nextInt();         scanner.nextLine(); //
Consume newline
System.out.print("Enter Name: ");
        String name = scanner.nextLine();
System.out.print("Enter Salary: ");         double salary
= scanner.nextDouble();         employees.add(new
Employee(id, name, salary));
    }

    public static void updateEmployee() {
        System.out.print("Enter Employee ID to update: ");
```

```
        int id = scanner.nextInt();         for
(Employee emp : employees) {
if (emp.id == id) {
scanner.nextLine();
                System.out.print("Enter New Name: ");
emp.name = scanner.nextLine();
System.out.print("Enter New Salary: ");
emp.salary = scanner.nextDouble();
System.out.println("Updated Successfully!");
                return;
            }
        }
        System.out.println("Employee Not Found!");
    }

    public static void removeEmployee() {
        System.out.print("Enter Employee ID to remove: ");
int id = scanner.nextInt();         employees.removeIf(emp
-> emp.id == id);
        System.out.println("Removed Successfully!");
    }

    public static void searchEmployee() {
        System.out.print("Enter Employee ID to search: ");
int id = scanner.nextInt();         for (Employee emp :
employees) {
```

```
            if (emp.id == id) {
System.out.println(emp);                return;
            }
        }
        System.out.println("Employee Not Found!");
    }

    public static void main(String[] args) {
while (true) {
            System.out.println("\n1. Add  2. Update  3. Remove  4. Search  5. Exit");
int choice = scanner.nextInt();             switch (choice) {                case 1 ->
addEmployee();                case 2 -> updateEmployee();                case 3 ->
removeEmployee();                case 4 -> searchEmployee();                case 5 ->
System.exit(0);
                default -> System.out.println("Invalid Choice!");
            }
        }
    }
}
```

Output

```
1. Add  2. Update  3. Remove  4. Search  5. Exit
1
Enter ID: 101
Enter Name: Bob
Enter Salary: 30000

1. Add  2. Update  3. Remove  4. Search  5. Exit
1
Enter ID: 102
Enter Name: Alice
Enter Salary: 40000
```

```
1. Add  2. Update  3. Remove  4. Search  5. Exit
2
Enter Employee ID to update: 101
Enter New Name: Bob
Enter New Salary: 60000
Updated Successfully!

1. Add  2. Update  3. Remove  4. Search  5. Exit
3
Enter Employee ID to remove: 102
Removed Successfully!
```

```
1. Add  2. Update  3. Remove  4. Search  5. Exit
4
Enter Employee ID to search: 101
ID: 101, Name: Bob, Salary: 60000.0
```

MEDIUM:

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 the Collection interface.

Objective: The Card Collection System is designed to store and manage a collection of playing cards using the Collection interface. The program allows users to add new cards by specifying their symbol and value. Additionally, it provides functionality to

search for all cards belonging to a specific symbol, helping users efficiently categorize and retrieve card information.

Code/Implementation:

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

class Card {
    String symbol;
    String value;

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

    @Override
    public String toString() {
return value + " of " + symbol;
    }
}
public class CardCollection {
    private static List<Card> deck = new ArrayList<>();
private static Scanner scanner = new Scanner(System.in);
public static void main(String[] args) {        while (true) {
```

```
            System.out.println("\n1. Add Card  2. Find Cards  3. Exit");
System.out.print("Enter your choice: ");            int choice =
scanner.nextInt();            scanner.nextLine(); // Consume newline


            switch (choice) {
case 1:
addCard();
break;                case 2:
findCards();
break;                case 3:
                    System.out.println("Exiting...");
return;                default:
                    System.out.println("Invalid choice! Please enter 1, 2, or 3.");
            }
        }
    }
    public static void addCard() {
        System.out.print("Enter Symbol (Hearts, Diamonds, etc.): ");
        String symbol = scanner.nextLine().trim();

        System.out.print("Enter Value (2, 3, King, etc.): ");
        String value = scanner.nextLine().trim();


        deck.add(new Card(symbol, value));
        System.out.println("Card Added!");
    }
    public static void findCards() {
```

```
        System.out.print("Enter Symbol to find: ");
        String symbolToFind = scanner.nextLine().trim();

        boolean found = false;
        System.out.println("\nCards with symbol " + symbolToFind + ":");
for (Card card : deck) {
            if (card.symbol.equalsIgnoreCase(symbolToFind)) { // Case-insensitive comparison
System.out.println(card);                found = true;
            }
        }

        if (!found) {
            System.out.println("No cards found for this symbol!");
        }
    }
}
```

Output:

```
1. Add Card  2. Find Cards  3. Exit
Enter your choice: 1
Enter Symbol (Hearts, Diamonds, etc.): Hearts
Enter Value (2, 3, King, etc.): 3
Card Added!

1. Add Card  2. Find Cards  3. Exit
Enter your choice: 2
Enter Symbol to find: Hearts

Cards with symbol Hearts:
3 of Hearts
```

HARD:

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.

Objective: The Ticket Booking System with Threads focuses on developing a multi-threaded application to handle concurrent seat bookings while preventing double booking. The program uses synchronized threads to ensure only one user can book a particular seat at a time. It also incorporates thread priorities, ensuring that VIP bookings are processed first. By taking user input for customer names, seat numbers, and priority levels, the system simulates a real-world ticket booking scenario while maintaining fairness and efficiency.

Code/Implementation:

```
import java.util.Scanner;

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

java.util.concurrent.locks.ReentrantLock; class

TicketBooking extends Thread {
```

```
    private static final Lock lock = new ReentrantLock();
    private static boolean[] seats = new boolean[5]; // 5 seats for simplicity
private int seatNumber;     private String customerName;
    public TicketBooking(String customerName, int seatNumber, int priority) {
this.customerName = customerName;         this.seatNumber = seatNumber;
this.setPriority(priority);
    }
    @Override
public void run() {
lock.lock();         try
{
            if (!seats[seatNumber]) {
seats[seatNumber] = true;
                System.out.println(customerName + " successfully booked seat " + seatNumber);
            } else {
                System.out.println(customerName + " failed to book seat " + seatNumber + " (Already
booked)");
            }         }
finally {
lock.unlock();
        }
    }
}
public class TicketBookingSystem {
public static void main(String[] args) {
Scanner scanner = new
Scanner(System.in);
```

```
        TicketBooking[] bookings = new TicketBooking[5];
        System.out.println("Welcome to the Ticket Booking System (Seats: 0-4)");
for (int i = 0; i < 5; i++) {
            System.out.print("\nEnter Customer Name: ");
            String name = scanner.next();
            System.out.print("Enter Seat Number (0-4): ");
int seatNumber = scanner.nextInt();              if
(seatNumber < 0 || seatNumber >= 5) {
                System.out.println("Invalid seat number! Try again.");
i--;                  continue;
            }
            System.out.print("Enter Priority (1 for VIP, 2 for Regular): ");
int priority = scanner.nextInt();              int threadPriority = (priority ==
1) ? Thread.MAX_PRIORITY : Thread.MIN_PRIORITY;
            bookings[i] = new TicketBooking(name, seatNumber, threadPriority);
        }
        System.out.println("\nProcessing Bookings...\n");
for (TicketBooking booking : bookings) {              if
(booking != null) {                  booking.start();
            }
        }
        scanner.close();
    }
}
```

Output:

```
Welcome to the Ticket Booking System (Seats: 0-4)

Enter Customer Name: Elena
Enter Seat Number (0-4): 1
Enter Priority (1 for VIP, 2 for Regular): 1

Enter Customer Name: Damon
Enter Seat Number (0-4): 2
Enter Priority (1 for VIP, 2 for Regular): 1

Enter Customer Name: Matt
Enter Seat Number (0-4): 2
Enter Priority (1 for VIP, 2 for Regular): 2

Enter Customer Name: Katherine
Enter Seat Number (0-4): 0
Enter Priority (1 for VIP, 2 for Regular): 2
```

```
Enter Customer Name: Bonny
Enter Seat Number (0-4): 3
Enter Priority (1 for VIP, 2 for Regular): 1

Processing Bookings...

Elena successfully booked seat 1
Damon successfully booked seat 2
Matt failed to book seat 2 (Already booked)
Katherine successfully booked seat 0
Bonny successfully booked seat 3
```

Learning Outcome:

1. Learned to use ArrayList for storing, updating, and searching employee records efficiently.
2. Gained experience in handling collections and implementing caseinsensitive search in Java.
3. Understood multi-threading, synchronization, and thread priorities to prevent race conditions in concurrent booking.