Experiment 4

Student Name: Harshit Mishra **UID: 22BCS11592**

Branch: BE/CSE **Section/Group: IOT_618/B**

Semester: 6th **Date of Performance: 21/02/25**

Subject Name: Project based learning in Java

Subject Code: 22CSH- 359

Easy Level

1. **Aim:** Write a Java program to implement an Array List that stores employee details (ID, Name, and Salary). Allow users to add, update, remove, and search employees.

2. **Objective:** To implement an Employee Management System using Array List in java that allows users to add, update, remove, and search employee details (ID, Name, and Salary).

3. **Implementation/Code:**

```
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;
    }
    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Salary: " + salary;
    }
}
public class EmployeeManagement {
    static List<Employee> employees = new ArrayList<>();
    static Scanner sc = new Scanner(System.in);
    static void addEmployee() {
        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();
        employees.add(new Employee(id, name, salary));
        System.out.println("Employee added successfully!");
    }
    static void updateEmployee() {
        System.out.print("Enter Employee ID to update: ");
        int id = sc.nextInt();
        for (Employee e : employees) {
            if (e.id == id) {
```

```
            sc.nextLine();
            System.out.print("Enter new Name: ");
            e.name = sc.nextLine();
            System.out.print("Enter new Salary: ");
            e.salary = sc.nextDouble();
            System.out.println("Employee updated successfully!");
            return;
        }
    }
    System.out.println("Employee not found!");
}
static void removeEmployee() {
    System.out.print("Enter Employee ID to remove: ");
    int id = sc.nextInt();
    employees.removeIf(e -> e.id == id);
    System.out.println("Employee removed successfully!");
}
static void searchEmployee() {
    System.out.print("Enter Employee ID to search: ");
    int id = sc.nextInt();
    for (Employee e : employees) {
        if (e.id == id) {
            System.out.println(e);
            return;
        }
    }
```

```
        System.out.println("Employee not found!");
    }
    public static void main(String[] args) {
        while (true) {
            System.out.println("\n1. Add Employee\n2. Update Employee\n3. Remove
Employee\n4. Search Employee\n5. Exit");
            System.out.print("Choose an option: ");
            int choice = sc.nextInt();
            switch (choice) {
                case 1 -> addEmployee();
                case 2 -> updateEmployee();
                case 3 -> removeEmployee();
                case 4 -> searchEmployee();
                case 5 -> {
                    System.out.println("Exiting...");
                    return;
                }
                default -> System.out.println("Invalid choice! Try again.");
            }
        }
    }
}
```

4. Output:

Add Employee:

```
PROBLEMS 19   OUTPUT   DEBUG CONSOLE   TASK MO

PS C:\Users\harsh\OneDrive\Documents\Java Sem
nagement }

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
Choose an option: 1
Enter ID: 11592
Enter Name: Harshit Mishra
Enter Salary: 150000
Employee added successfully!
```

Update Employee:

```
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
Choose an option: 2
Enter Employee ID to update: 11592
Enter new Name: Harshit
Enter new Salary: 100000
Employee updated successfully!
```

Remove & Search Employee:

```
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
Choose an option: 3
Enter Employee ID to remove: 11592
Employee removed successfully!

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
Choose an option: 4
Enter Employee ID to search: 11592
Employee not found!
```

5. Learning Outcomes:

- Implementing CRUD operations.
- Understanding Java Collections.
- Learn to handle user input from the command line.
- Understand how to store and manage Employee information using arrays.
- Learn how to handle the concept of Multithreading.

Medium Level

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:** The objective of this program is to store and retrieve playing cards based on their symbols (e.g., Hearts, Spades, Diamonds, Clubs) using the Collection framework in Java.

3. **Implementation/Code:**

```
import java.util.*;
class CardCollection {
    private final Map<String, Set<String>> cardMap = new HashMap<>();
    public void addCard(String symbol, String cardName) {
        cardMap.putIfAbsent(symbol, new HashSet<>());
        cardMap.get(symbol).add(cardName);
    }
    public void showCards(String symbol) {
            System.out.println(cardMap.getOrDefault(symbol, Set.of("No cards
found.")));
    }
    public static void main(String[] args) {
        CardCollection collection = new CardCollection();
        collection.addCard("Hearts", "Ace");
        collection.addCard("Hearts", "King");
```

```
        collection.addCard("Spades", "Queen");
        collection.addCard("Diamonds", "Jack");
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter symbol: ");
        collection.showCards(scanner.nextLine());
        scanner.close();
    }
}
```

4. Output:

```
PS C:\Users\harsh\OneDrive\Documents\Java Sem 6> cd "c
 }
Enter symbol: Hearts
[Ace, King]
PS C:\Users\harsh\OneDrive\Documents\Java Sem 6> cd "c
 }
Enter symbol: King
[No cards found.]
PS C:\Users\harsh\OneDrive\Documents\Java Sem 6>
```

5. Learning Outcomes:

- Understanding Java collections.
- Implement key-value storage for categorizing playing cards.
- Add and retrieve elements dynamically without predefined limits.
- Use Scanner to take user input and process it efficiently.

Hard Level

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. **Objective:** The objective of this Java program is to simulate a ticket booking system where multiple users (threads) attempt to book seats concurrently.

3. **Implementation/Code:**

```
import java.util.Scanner;
import java.util.concurrent.locks.*;
class TicketBookingSystem {
    private final boolean[] seats;
    private final Lock lock = new ReentrantLock();
    public TicketBookingSystem(int totalSeats) {
        seats = new boolean[totalSeats];
    }
    public void bookSeat(int seatNumber, String customer) {
        lock.lock();
        try {
            if (!seats[seatNumber]) {
                seats[seatNumber] = true;
                System.out.println(customer + " successfully booked seat " +
seatNumber);
```

```
            } else {
                System.out.println(customer + " failed to book seat " + seatNumber + "
(Already booked)");
            }
        } finally {
            lock.unlock();
        }
    }
}
class BookingThread extends Thread {
    private final TicketBookingSystem system;
    private final int seatNumber;
    private final String customer;
    public BookingThread(TicketBookingSystem system, int seatNumber, String
customer, int priority) {
        this.system = system;
        this.seatNumber = seatNumber;
        this.customer = customer;
        setPriority(priority);
    }
    @Override
    public void run() {
        system.bookSeat(seatNumber, customer);
    }
}
```

```
public class TicketBookingApp {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter total number of seats: ");
        int totalSeats = scanner.nextInt();
        TicketBookingSystem system = new TicketBookingSystem(totalSeats);
        System.out.print("Enter number of users: ");
        int userCount = scanner.nextInt();
        scanner.nextLine(); // Consume newline
        Thread[] users = new Thread[userCount];
        for (int i = 0; i < userCount; i++) {
            System.out.print("Enter customer name: ");
            String customer = scanner.nextLine();
            System.out.print("Enter seat number to book: ");
            int seatNumber = scanner.nextInt();
            System.out.print("Enter priority (1-10, 10 is highest): ");
            int priority = scanner.nextInt();
            scanner.nextLine(); // Consume newline
            users[i] = new BookingThread(system, seatNumber, customer, priority);
        }
        for (Thread user : users) {
            user.start();
        }
        scanner.close();
    }}
```

4. Output:

```
gApp }
Enter total number of seats: 5
Enter number of users: 2
Enter customer name: Harshit
Enter seat number to book: 2
Enter priority (1-10, 10 is highest): 1
Enter customer name: Rajan
Enter seat number to book: 4
Enter priority (1-10, 10 is highest): 1
Harshit successfully booked seat 2
Rajan successfully booked seat 4
PS C:\Users\harsh\OneDrive\Documents\Java Sem 6>
```

5. Learning Outcomes:

- How multiple threads interact when accessing shared data.
- Handling race conditions in a multi-threaded environment.
- Ensuring that only one thread modifies the shared resource (seats) at a time.
- Taking user input for dynamic seat selection and priority assignment.