Experiment-4

Student Name: Kamalpreet Singh **UID: 22BCS11720**
Branch: B.E- CSE **Section/Group: IOT_643-A**
Semester: 6th **DateofPerformance:17-02-2025**
Subject Name: Project Based Learning Java with Lab **Subject Code: 22CSH-359**

Aim: Write a Program to perform the basic operations like insert, delete, display and search in list. List contains String object items where these operations are to be performed.

Objective: - To implement basic operations on a list that stores string objects.
- To develop a card collection system using the Collection framework.
- To implement a synchronized ticket booking system using multithreading.

Implementation/Code:

Easy Level: Employee Management System using `ArrayList`

```
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;
    }

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

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

```
do {
    System.out.println("\nMenu:");
    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 choice: ");
    choice = scanner.nextInt();

    switch (choice) {
        case 1:
            System.out.print("Enter ID: ");
            int id = scanner.nextInt();
            scanner.nextLine();
            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));
            System.out.println("Employee added successfully.");
            break;
        case 2:
            System.out.print("Enter Employee ID to update: ");
            int updateId = scanner.nextInt();
            for (Employee emp : employees) {
                if (emp.id == updateId) {
                    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("Employee updated successfully.");
                }
            }
            break;
        case 3:
            System.out.print("Enter Employee ID to remove: ");
            int removeId = scanner.nextInt();
            employees.removeIf(emp -> emp.id == removeId);
            System.out.println("Employee removed successfully.");
            break;
        case 4:
            System.out.print("Enter Employee ID to search: ");
            int searchId = scanner.nextInt();
            boolean found = false;
            for (Employee emp : employees) {
                if (emp.id == searchId) {
                    System.out.println(emp);
```

```
                    found = true;
                }
            }
            if (!found) System.out.println("Employee not found.");
            break;
        case 5:
            System.out.println("\nEmployee List:");
            for (Employee emp : employees) {
                System.out.println(emp);
            }
            break;
        case 6:
            System.out.println("Exiting...");
            break;
        default:
            System.out.println("Invalid choice. Try again.");
      }
    } while (choice != 6);

    scanner.close();
  }
}
```

Medium Level: Card Collection System using `Collection` Interface

```
import java.util.*;

public class CardCollection {
  public static void main(String[] args) {
    Map<String, List<String>> cardDeck = new HashMap<>();
    Scanner scanner = new Scanner(System.in);

    cardDeck.put("Hearts", Arrays.asList("A", "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K"));
    cardDeck.put("Diamonds", Arrays.asList("A", "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K"));
    cardDeck.put("Clubs", Arrays.asList("A", "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K"));
    cardDeck.put("Spades", Arrays.asList("A", "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K"));

    System.out.println("Available symbols: " + cardDeck.keySet());
    System.out.print("Enter the card symbol (Hearts, Diamonds, Clubs, Spades) to find its cards: ");
    String symbol = scanner.nextLine();

    if (cardDeck.containsKey(symbol)) {
      System.out.println("Cards available in " + symbol + ": " + cardDeck.get(symbol));
    } else {
      System.out.println("Invalid symbol entered.");
    }
```

```
        scanner.close();
    }
}
```

Hard Level: Ticket Booking System with Synchronized Threads

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

class TicketBookingSystem {
    private int availableSeats = 5;
    private final ReentrantLock lock = new ReentrantLock();

    public void bookTicket(String name) {
        lock.lock();
        try {
            if (availableSeats > 0) {
                System.out.println(name + " booked a seat. Seats left: " + (--availableSeats));
                Thread.sleep(100); // Simulating processing delay
            } else {
                System.out.println(name + " tried to book, but no seats left.");
            }
        } catch (InterruptedException e) {
            System.out.println("Booking interrupted for " + name);
        } finally {
            lock.unlock();
        }
    }
}

class Passenger extends Thread {
    private final TicketBookingSystem bookingSystem;
    private final String passengerName;

    public Passenger(TicketBookingSystem system, String name) {
        this.bookingSystem = system;
        this.passengerName = name;
    }

    public void run() {
        bookingSystem.bookTicket(passengerName);
    }
}

public class TicketBookingApp {
    public static void main(String[] args) {
        TicketBookingSystem bookingSystem = new TicketBookingSystem();

        Passenger p1 = new Passenger(bookingSystem, "VIP Passenger 1");
```

```
        Passenger p2 = new Passenger(bookingSystem, "VIP Passenger 2");
        Passenger p3 = new Passenger(bookingSystem, "Regular Passenger 1");
        Passenger p4 = new Passenger(bookingSystem, "Regular Passenger 2");
        Passenger p5 = new Passenger(bookingSystem, "Regular Passenger 3");
        Passenger p6 = new Passenger(bookingSystem, "Regular Passenger 4");

        p1.setPriority(Thread.MAX_PRIORITY);
        p2.setPriority(Thread.MAX_PRIORITY);
        p3.setPriority(Thread.NORM_PRIORITY);
        p4.setPriority(Thread.NORM_PRIORITY);
        p5.setPriority(Thread.NORM_PRIORITY);
        p6.setPriority(Thread.NORM_PRIORITY);

        p1.start();
        p2.start();
        p3.start();
        p4.start();
        p5.start();
        p6.start();
    }
}
```

Outputs:

Easy-

```
Menu:
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Enter choice: 1
Enter ID: 11720
Enter Name: Kamalpreet Singh
Enter Salary: 80000
Employee added successfully.

Menu:
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Enter choice: 2
Enter Employee ID to update: 11720
Enter New Name: Kamalpreet Singh
Enter New Salary: 100000
Employee updated successfully.

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

Employee List:
ID: 11720, Name: Kamalpreet Singh, Salary: $100000.0
```

Medium-

```
Available symbols: [Spades, Hearts, Diamonds, Clubs]
Enter the card symbol (Hearts, Diamonds, Clubs, Spades) to find its cards: Diamonds
Cards available in Diamonds: [A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K]
```

Hard-

```
VIP Passenger 1 booked a seat. Seats left: 4
VIP Passenger 2 booked a seat. Seats left: 3
Regular Passenger 1 booked a seat. Seats left: 2
Regular Passenger 2 booked a seat. Seats left: 1
Regular Passenger 3 booked a seat. Seats left: 0
Regular Passenger 4 tried to book, but no seats left.
```

Learning Outcomes:

- Understand and implement basic list operations.
- Learn to use Java's Collection framework for organizing data.
- Gain knowledge of thread synchronization in real-world applications.
- Develop a menu-driven Java application.