Experiment 4

Student Name: Sujit Kumar Bhagat **UID: 22BCS10505**
Branch: BE-CSE **Section/Group: IOT-643/B**
Semester:6th **Date of Performance: 20/02/2025**
Subject Name: Project Based Learning in Java with Lab **Subject Code: 22CSH-359**

Problem 1

1. **Aim: Develop Java programs using core concepts such as data structures, collections, and multithreading to manage and manipulate data.**
2. **Objective: 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.**

3. **Implementation/Code:**

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

// Employee class representing an employee
class Employee {
    private int id;
    private String name;
    private double salary;

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

    // Getters & Setters
    public int getId() {
        return id;
```

```
    }

    public String getName() {
        return name;
    }

    public double getSalary() {
        return salary;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

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

// EmployeeManager class to handle CRUD operations
class EmployeeManager {
    private ArrayList<Employee> employees = new ArrayList<>();

    // Add an employee
    public void addEmployee(int id, String name, double salary) {
        employees.add(new Employee(id, name, salary));
        System.out.println("Employee added successfully!");
```

```
    }

    // Update an employee's details
    public boolean updateEmployee(int id, String newName, double newSalary) {
        for (Employee emp : employees) {
            if (emp.getId() == id) {
                emp.setName(newName);
                emp.setSalary(newSalary);
                System.out.println("Employee details updated!");
                return true;
            }
        }
        System.out.println("Employee not found!");
        return false;
    }

    // Remove an employee
    public boolean removeEmployee(int id) {
        return employees.removeIf(emp -> emp.getId() == id);
    }

    // Search an employee by ID
    public Employee searchById(int id) {
        for (Employee emp : employees) {
            if (emp.getId() == id) {
                return emp;
            }
        }
        return null;
    }

    // Search an employee by Name
    public ArrayList<Employee> searchByName(String name) {
```

```
        ArrayList<Employee> result = new ArrayList<>();
        for (Employee emp : employees) {
            if (emp.getName().equalsIgnoreCase(name)) {
                result.add(emp);
            }
        }
        return result;
    }

    // Display all employees
    public void displayEmployees() {
        if (employees.isEmpty()) {
            System.out.println("No employees found.");
        } else {
            for (Employee emp : employees) {
                System.out.println(emp);
            }
        }
    }
}

// Main class with a menu-driven program
public class EmployeeManagementSystem {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        EmployeeManager manager = new EmployeeManager();

        while (true) {
            System.out.println("\n--- Employee 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 by ID");
```

```
System.out.println("5. Search Employee by Name");
System.out.println("6. Display All Employees");
System.out.println("7. Exit");
System.out.print("Choose an option: ");
int choice = scanner.nextInt();
scanner.nextLine(); // Consume newline

switch (choice) {
    case 1:
        System.out.print("Enter Employee ID: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // Consume newline
        System.out.print("Enter Employee Name: ");
        String name = scanner.nextLine();
        System.out.print("Enter Employee Salary: ");
        double salary = scanner.nextDouble();
        manager.addEmployee(id, name, salary);
        break;

    case 2:
        System.out.print("Enter Employee ID to Update: ");
        int updateId = scanner.nextInt();
        scanner.nextLine(); // Consume newline
        System.out.print("Enter New Name: ");
        String newName = scanner.nextLine();
        System.out.print("Enter New Salary: ");
        double newSalary = scanner.nextDouble();
        manager.updateEmployee(updateId, newName, newSalary);
        break;

    case 3:
        System.out.print("Enter Employee ID to Remove: ");
        int removeId = scanner.nextInt();
```

```
            if (manager.removeEmployee(removeId)) {
                System.out.println("Employee removed successfully!");
            } else {
                System.out.println("Employee not found!");
            }
            break;

        case 4:
            System.out.print("Enter Employee ID to Search: ");
            int searchId = scanner.nextInt();
            Employee foundById = manager.searchById(searchId);
            if (foundById != null) {
                System.out.println("Employee Found: " + foundById);
            } else {
                System.out.println("Employee not found!");
            }
            break;

        case 5:
            System.out.print("Enter Employee Name to Search: ");
            String searchName = scanner.nextLine();
            ArrayList<Employee> foundByName = manager.searchByName(searchName);
            if (foundByName.isEmpty()) {
                System.out.println("No employees found with the name: " + searchName);
            } else {
                System.out.println("Employees Found:");
                for (Employee emp : foundByName) {
                    System.out.println(emp);
                }
            }
            break;

        case 6:
```

```
                System.out.println("All Employees:");
                manager.displayEmployees();
                break;

            case 7:
                System.out.println("Exiting program...");
                scanner.close();
                return;

            default:
                System.out.println("Invalid choice! Please try again.");
        }
    }
}
```

4. **Output:**

```
--- Employee Management System ---
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee by ID
5. Search Employee by Name
6. Display All Employees
7. Exit
Choose an option: 1
Enter Employee ID: 123
Enter Employee Name: Sujit
Enter Employee Salary: 180000
Employee added successfully!
```

Problem 2

1. **Aim: Develop Java programs using core concepts such as data structures, collections, and multithreading to manage and manipulate data.**

2. **Objective: 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.**

3. **Code:**

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

// Card class representing a playing card
class Card {
    private String symbol;
    private String value;

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

    // Getters
    public String getSymbol() {
        return symbol;
    }

    public String getValue() {
        return value;
    }

    // Display card details
    @Override
    public String toString() {
        return value + " of " + symbol;
    }
}

// CardCollection class using Collection framework (ArrayList)
class CardCollection {
    private List<Card> cards;
```

```
    // Constructor initializes the list
    public CardCollection() {
        this.cards = new ArrayList<>();
    }

    // Add a new card to the collection
    public void addCard(String symbol, String value) {
        cards.add(new Card(symbol, value));
    }

    // Find all cards with a specific symbol
    public List<Card> findCardsBySymbol(String symbol) {
        List<Card> result = new ArrayList<>();
        for (Card card : cards) {
            if (card.getSymbol().equalsIgnoreCase(symbol)) {
                result.add(card);
            }
        }
        return result;
    }

    // Display all stored cards
    public void displayCards() {
        if (cards.isEmpty()) {
            System.out.println("No cards in the collection.");
        } else {
            for (Card card : cards) {
                System.out.println(card);
            }
        }
    }
}

// Main class
public class CardManager {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        CardCollection collection = new CardCollection();

        // Sample data
        collection.addCard("Hearts", "Ace");
        collection.addCard("Diamonds", "King");
        collection.addCard("Spades", "Queen");
        collection.addCard("Hearts", "10");
        collection.addCard("Clubs", "Jack");
```

```
        while (true) {
            System.out.println("\n--- Card Collection Manager ---");
            System.out.println("1. Add Card");
            System.out.println("2. Find Cards by Symbol");
            System.out.println("3. Display All Cards");
            System.out.println("4. Exit");
            System.out.print("Choose an option: ");
            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1:
                    System.out.print("Enter card symbol (Hearts, Diamonds, Clubs, Spades):
");
                    String symbol = scanner.nextLine();
                    System.out.print("Enter card value (Ace, 2, 3, ... King, Queen, Jack): ");
                    String value = scanner.nextLine();
                    collection.addCard(symbol, value);
                    System.out.println("Card added successfully!");
                    break;

                case 2:
                    System.out.print("Enter symbol to search: ");
                    String searchSymbol = scanner.nextLine();
                    List<Card> foundCards =
collection.findCardsBySymbol(searchSymbol);
                    if (foundCards.isEmpty()) {
                        System.out.println("No cards found for the symbol: " +
searchSymbol);
                    } else {
                        System.out.println("Cards found:");
                        for (Card card : foundCards) {
                            System.out.println(card);
                        }
                    }
                    break;

                case 3:
                    System.out.println("All Cards in Collection:");
                    collection.displayCards();
                    break;

                case 4:
                    System.out.println("Exiting program...");
                    scanner.close();
                    return;
```

```
                default:
                    System.out.println("Invalid choice! Try again.");
            }
        }
    }
}
```

4. **Output**

```
--- Card Collection Manager ---
1. Add Card
2. Find Cards by Symbol
3. Display All Cards
4. Exit
Choose an option: 1
Enter card symbol (Hearts, Diamonds, Clubs, Spades): Hearts
Enter card value (Ace, 2, 3, ... King, Queen, Jack): 1
Card added successfully!
```

Problem 3

1. **Aim: Develop Java programs using core concepts such as data structures, collections, and multithreading to manage and manipulate data.**
2. **Objective: 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.**
3. **Code:**

```
import java.util.ArrayList;

import java.util.List;

import java.util.Random;
```

```
// TicketBookingSystem class
class TicketBookingSystem {
    private final int totalSeats;
    private final boolean[] seats;  // Seat availability
    private final Object lock = new Object();  // Lock for synchronization

    public TicketBookingSystem(int totalSeats) {
        this.totalSeats = totalSeats;
        this.seats = new boolean[totalSeats]; // false means available
    }

    // Synchronized method to book a seat
    public boolean bookSeat(int seatNumber, String customerType) {
        synchronized (lock) {
            if (seatNumber < 0 || seatNumber >= totalSeats) {
                System.out.println(customerType + " Invalid seat number!");
                return false;
            }
            if (!seats[seatNumber]) {
                seats[seatNumber] = true; // Book seat
                System.out.println(customerType + " successfully booked seat: " +
seatNumber);
                return true;
            } else {
```

```
                System.out.println(customerType + " Seat " + seatNumber + " is already
booked.");
                return false;
            }
        }
    }
}

// BookingThread class to simulate users booking seats
class BookingThread extends Thread {
    private final TicketBookingSystem system;
    private final String customerType;
    private final Random random = new Random();

    public BookingThread(TicketBookingSystem system, String customerType, int
priority) {
        this.system = system;
        this.customerType = customerType;
        setPriority(priority);  // Set thread priority (higher for VIPs)
    }

    @Override
    public void run() {
        for (int i = 0; i < 3; i++) {  // Each user tries to book 3 seats
            int seatNumber = random.nextInt(10);  // Randomly selecting a seat
```

```
            system.bookSeat(seatNumber, customerType);
            try {
                Thread.sleep(100);  // Simulate processing time
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

// Main class
public class TicketBookingApp {
    public static void main(String[] args) {
        TicketBookingSystem system = new TicketBookingSystem(10);  // 10 seats
available
        List<Thread> threads = new ArrayList<>();

        // Creating VIP Customers (Higher Priority)
        for (int i = 0; i < 3; i++) {
            BookingThread vip = new BookingThread(system, "VIP Customer " + (i +
1), Thread.MAX_PRIORITY);
            threads.add(vip);
        }

        // Creating Regular Customers (Normal Priority)
```

```
        for (int i = 0; i < 5; i++) {
            BookingThread regular = new BookingThread(system, "Regular Customer "
+ (i + 1), Thread.NORM_PRIORITY);
            threads.add(regular);
        }

        // Start all threads
        for (Thread t : threads) {
            t.start();
        }

        // Wait for all threads to complete
        for (Thread t : threads) {
            try {
                t.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        System.out.println("All bookings completed.");
    }
}
```

5. **Output:**

```
VIP Customer 1 successfully booked seat: 1
Regular Customer 5 successfully booked seat: 7
Regular Customer 4 successfully booked seat: 4
Regular Customer 3 successfully booked seat: 2
Regular Customer 2 Seat 4 is already booked.
Regular Customer 1 Seat 4 is already booked.
VIP Customer 3 Seat 2 is already booked.
VIP Customer 2 successfully booked seat: 9
Regular Customer 5 Seat 2 is already booked.
VIP Customer 1 successfully booked seat: 5
Regular Customer 3 successfully booked seat: 0
Regular Customer 4 Seat 1 is already booked.
Regular Customer 2 Seat 7 is already booked.
Regular Customer 1 successfully booked seat: 3
VIP Customer 3 Seat 1 is already booked.
VIP Customer 2 Seat 1 is already booked.
Regular Customer 5 Seat 0 is already booked.
VIP Customer 1 Seat 7 is already booked.
Regular Customer 4 Seat 0 is already booked.
Regular Customer 3 Seat 3 is already booked.
Regular Customer 2 Seat 0 is already booked.
Regular Customer 1 Seat 3 is already booked.
VIP Customer 3 Seat 3 is already booked.
VIP Customer 2 Seat 0 is already booked.
All bookings completed.
```

6. **Learning Outcomes:**

OOP Mastery – Applied **Encapsulation, Classes, and Objects** in Java.
Collections Usage – Implemented **ArrayList** for **dynamic data storage and management**.
CRUD Operations – Developed **Create, Read, Update, Delete** functionalities.
User Interaction – Built a **menu-driven system** using **Scanner** for input handling.
Problem-Solving – Improved **debugging skills** and handled **edge cases efficiently**.