Experiment 4

Student Name: Shivangi **UID: 22BCS16953**

Branch: BE-CSE **Date of Performance: 23-02-2025**

Semester: 6th **Section/Group: 22BCS_EPAM-801/ B**

Subject Name: Project based learning in java with lab **Subject Code: 22CSH-359**

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

2. **Objective:**

Easy Level:

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.

Medium Level:

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.

Hard Level:

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. **Implementation/Code:**

Easy Level:

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.

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

```
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 {
    static ArrayList<Employee> employees = new ArrayList<>();

    static void addEmployee(int id, String name, double salary) {
        employees.add(new Employee(id, name, salary));
    }
    static void updateEmployee(int id, String name, double salary) {
        for (int i = 0; i < employees.size(); i++) {
            Employee e = employees.get(i);
            if (e.id == id) {
                e.name = name;
                e.salary = salary;
                System.out.println("Employee updated successfully.");
                return;
            }
        }
        System.out.println("Employee not found!");
    }
    static void removeEmployee(int id) {
```

```
        for (int i = 0; i < employees.size(); i++) {
            if (employees.get(i).id == id) {
                employees.remove(i);
                System.out.println("Employee removed successfully.");
                return;
            }
        }
        System.out.println("Employee not found!");
    }
    static void searchEmployee(int id) {
        for (int i = 0; i < employees.size(); i++) {
            Employee e = employees.get(i);
            if (e.id == id) {
                e.display();
                return;
            }
        }
        System.out.println("Employee not found!");
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        addEmployee(1, "Shivangi", 50000);
        addEmployee(2, "Priyanka", 60000);
        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. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
            int id;
```

```
String name;
double salary;
switch (choice) {
    case 1:
        System.out.print("Enter ID: ");
        id = scanner.nextInt();
        scanner.nextLine();
        System.out.print("Enter Name: ");
        name = scanner.nextLine();
        System.out.print("Enter Salary: ");
        salary = scanner.nextDouble();
        addEmployee(id, name, salary);
        System.out.println("Employee added successfully.");
        break;
    case 2:
        System.out.print("Enter ID to update: ");
        id = scanner.nextInt();
        scanner.nextLine();
        System.out.print("Enter New Name: ");
        name = scanner.nextLine();
        System.out.print("Enter New Salary: ");
        salary = scanner.nextDouble();
        updateEmployee(id, name, salary);
        break;
    case 3:
        System.out.print("Enter ID to remove: ");
        id = scanner.nextInt();
        removeEmployee(id);
        break;
    case 4:
        System.out.print("Enter ID to search: ");
        id = scanner.nextInt();
```

```
                searchEmployee(id);
                break;
            case 5:
                System.out.println("Exiting...");
                scanner.close();
                return;
            default:
                System.out.println("Invalid option, please try again.");
        }
    }
  }
}
```

```
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
Enter your choice: 1
Enter ID: 3
Enter Name: Vivanshu
Enter Salary: 70000
Employee added successfully.

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
Enter your choice: 5
Exiting...
```

Medium Level:

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.

```
import java.util.*;
class Card {
    String symbol;
    String number;

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

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

public class CardCollection {
    private static Map<String, List<Card>> cardMap = new HashMap<>();

    public static void addCard(String symbol, String number) {
        cardMap.putIfAbsent(symbol, new ArrayList<>());
        cardMap.get(symbol).add(new Card(symbol, number));
    }

    public static void findCardsBySymbol(String symbol) {
        List<Card> cards = cardMap.get(symbol);
        if (cards == null || cards.isEmpty()) {
```

```
            System.out.println("No cards found for symbol: " + symbol);
        } else {
            for (Card card : cards) {
                System.out.println(card);
            }
        }
    }

    public static void main(String[] args) {
        addCard("Hearts", "2");
        addCard("Hearts", "3");
        addCard("Spades", "K");
        addCard("Clubs", "10");
        addCard("Hearts", "A");

        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter symbol to find cards: ");
        String symbol = scanner.nextLine();
        findCardsBySymbol(symbol);
        scanner.close();
    }
}
```

```
Enter symbol to find cards: Hearts
2 of Hearts
3 of Hearts
A of Hearts
```

Hard Level:

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.

```
class TicketBooking implements Runnable {
    private static int availableSeats = 10;

    @Override
    public synchronized void run() {
        if (availableSeats > 0) {
            System.out.println(Thread.currentThread().getName() + " booking seat...");
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            availableSeats--;
            System.out.println(Thread.currentThread().getName() + " booked a seat. Remaining
seats: " + availableSeats);
        } else {
            System.out.println("No seats available for " + Thread.currentThread().getName());
        }
    }

    public static void main(String[] args) {
        TicketBooking ticketBooking = new TicketBooking();
        Thread vip1 = new Thread(ticketBooking, "VIP1");
        Thread vip2 = new Thread(ticketBooking, "VIP2");
        Thread regular1 = new Thread(ticketBooking, "Regular1");
        Thread regular2 = new Thread(ticketBooking, "Regular2");

        vip1.setPriority(Thread.MAX_PRIORITY);
```

```
        vip2.setPriority(Thread.MAX_PRIORITY);
        regular1.setPriority(Thread.NORM_PRIORITY);
        regular2.setPriority(Thread.NORM_PRIORITY);

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

```
VIP1 booking seat...
VIP1 booked a seat. Remaining seats: 9
Regular2 booking seat...
Regular2 booked a seat. Remaining seats: 8
Regular1 booking seat...
Regular1 booked a seat. Remaining seats: 7
VIP2 booking seat...
VIP2 booked a seat. Remaining seats: 6
```

4. Learning Outcomes:

- Autoboxing & Unboxing: Efficiently convert between primitive types and their wrapper classes in Java.
- Serialization & Deserialization: Store and retrieve object states using file handling.
- Object-Oriented Design: Implement classes with attributes and methods, demonstrating encapsulation.
- File I/O Operations: Read from and write to files for persistent data storage.
- Menu-Driven Programming: Build interactive console applications with dynamic user input handling.