Experiment-4

Student Name: Anshika Goel **UID: 22BCS11678**
Branch: BE-CSE **Section/Group: 22BCS_IOT-643/B**
Semester: 6th **Date of Performance: 27/02/25**
Subject: Project-Based Learning with Java **Subject Code: 22CSH-359**

Easy -Level

1. **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.

2. **Algorithm:**

- **Employee Class**: Represents an employee with ID, name, and salary. It includes a toString method to display the employee details.
- **EmployeeModification Class**: Handles the main logic of the program. It includes a menu and methods for employee operations such as add, update, remove, search, and list employees.
- **Menu Handling**: Continuously displays the menu and prompts the user for a choice. Each choice corresponds to a specific method: addEmployee, updateEmployee, removeEmployee, searchEmployee, and listEmployees.
- **Methods**:
  - addEmployee: Adds a new employee to the list.
  - updateEmployee: Updates the details of an employee based on their ID.
  - removeEmployee: Removes an employee from the list by ID.
  - searchEmployee: Displays the details of an employee by their ID.
  - listEmployees: Lists all employees.
- **Exit**: Selecting option 6 exits the program, displays an exit message.

3. **Implementation/Code:**

```
import java.util.ArrayList;
import java.util.Scanner;
class Employee {
    private int id;
    private String name;
    private double salary;
    public Employee(int id, String name, double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }
    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;
    }
   public String toString() {
        return "Employee ID: " + id + ", Name: " + name + ", Salary: " + salary;
    }
}
public class EmployeeModification {
    private ArrayList<Employee> employees;
    public EmployeeModification() {
        employees = new ArrayList<>();
    }
    public void addEmployee(int id, String name, double salary) {
        Employee employee = new Employee(id, name, salary);
        employees.add(employee);
        System.out.println("Employee added: " + employee);
    }
    public void updateEmployee(int id, String name, double salary) {
        for (Employee employee : employees) {
            if (employee.getId() == id) {
                employee.setName(name);
                employee.setSalary(salary);
                System.out.println("Employee updated: " + employee);
                return;
            }
        }
        System.out.println("Employee with ID " + id + " not found.");
    }
    public void removeEmployee(int id) {
        for (Employee employee : employees) {
            if (employee.getId() == id) {
                employees.remove(employee);
                System.out.println("Employee removed: " + employee);
                return;
            }
        }
        System.out.println("Employee with ID " + id + " not found.");
    }
    public void searchEmployee(int id) {
```

```
    for (Employee employee : employees) {
        if (employee.getId() == id) {
            System.out.println("Employee found: " + employee);
            return;
        }
    }
    System.out.println("Employee with ID " + id + " not found.");
}

public void listEmployees() {
    if (employees.isEmpty()) {
        System.out.println("No employees found.");
    }
    else {
        System.out.println("Employee List:");
        for (Employee employee : employees) {
            System.out.println(employee);
        }
    }
}
public static void main(String[] args) {
    EmployeeModification emp = new EmployeeModification();
    Scanner scanner = new Scanner(System.in);
    int choice=0;
    while(choice!=6){
        System.out.println("\nEmployee 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");
        System.out.println("5. List Employees");
        System.out.println("6. Exit");
        System.out.print("Enter your choice: ");
        choice = scanner.nextInt();
        switch (choice) {
            case 1:
                System.out.print("Enter Employee ID: ");
                int add_id = scanner.nextInt();
                scanner.nextLine(); // Consume newline
                System.out.print("Enter Employee Name: ");
                String add_name = scanner.nextLine();
                System.out.print("Enter Employee Salary: ");
                double add_salary = scanner.nextDouble();
                emp.addEmployee(add_id, add_name, add_salary);
                break;
            case 2:
                System.out.print("Enter Employee ID to update: ");
                int update_id = scanner.nextInt();
```

```
                scanner.nextLine();
                System.out.print("Enter new Employee Name: ");
                String update_name = scanner.nextLine();
                System.out.print("Enter new Employee Salary: ");
                double update_salary = scanner.nextDouble();
                emp.updateEmployee(update_id, update_name, update_salary);
                break;
            case 3:
                System.out.print("Enter Employee ID to remove: ");
                int remove_id = scanner.nextInt();
                emp.removeEmployee(remove_id);
                break;
            case 4:
                System.out.print("Enter Employee ID to search: ");
                int search_id = scanner.nextInt();
                emp.searchEmployee(search_id);
                break;
            case 5:
                emp.listEmployees();
                break;
            case 6:
                System.out.println("Exiting...");
                break;
            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
5. List Employees
6. Exit
Enter your choice: 1
Enter Employee ID: 11678
Enter Employee Name: Anshika Goel
Enter Employee Salary: 50000
Employee added: Employee ID: 11678, Name: Anshika Goel, Salary: 50000.0
```

```
Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. List Employees
6. Exit
Enter your choice: 1
Enter Employee ID: 14382
Enter Employee Name: Ananya Goel
Enter Employee Salary: 690000
Employee added: Employee ID: 14382, Name: Ananya Goel, Salary: 690000.0

Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. List Employees
6. Exit
Enter your choice: 1
Enter Employee ID: 456
Enter Employee Name: Shruti
Enter Employee Salary: 89800
Employee added: Employee ID: 456, Name: Shruti, Salary: 89800.0

Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. List Employees
6. Exit
Enter your choice: 2
Enter Employee ID to update: 456
Enter new Employee Name: Naman
Enter new Employee Salary: 687899
Employee updated: Employee ID: 456, Name: Naman, Salary: 687899.0

Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. List Employees
6. Exit
Enter your choice: 5
Employee List:
Employee ID: 11678, Name: Anshika Goel, Salary: 500000.0
Employee ID: 14382, Name: Ananya Goel, Salary: 690000.0
Employee ID: 456, Name: Naman, Salary: 687899.0
```

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. **Algorithm:**

- Input the number of cards, N.
- Initialize a TreeMap> to store cards by their symbol.
- For each card from 1 to N, input the symbol and number, create a Card object, and add it to the list for that symbol in the map.
- Print the distinct symbols in alphabetical order by iterating over the keys of the map.
- For each symbol, print the card details, the number of cards, and the sum of their numbers by iterating over the list of cards for that symbol.
- End.

3. **Implementation/Code:**

```
import java.util.*;
class Card {
    String symbol;
    int number;
    public Card(String symbol, int number) {
        this.symbol = symbol;
        this.number = number;
    }
    public String getSymbol() {
        return symbol;
    }
    public int getNumber() {
        return number;
    }
    public String toString() {
        return symbol + " " + number;
    }
}
public class CardGame {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter Number of Cards : ");
        int N = scanner.nextInt();
        Map<String, List<Card>> cardMap = new TreeMap<>();
        for (int i = 1; i <= N; i++) {
            System.out.println("Enter card " + i + ": ");
            System.out.print("Symbol: ");
            String symbol = scanner.next();
            System.out.print("Number: ");
```

```
            int number = scanner.nextInt();
            Card card = new Card(symbol, number);
            cardMap.putIfAbsent(symbol, new ArrayList<>());
            cardMap.get(symbol).add(card);
        }
        System.out.println("Distinct Symbols are : ");
        for (String symbol : cardMap.keySet()) {
            System.out.print(symbol + " ");
        }
        System.out.println();
        for (String symbol : cardMap.keySet()) {
            List<Card> cards = cardMap.get(symbol);
            System.out.println("Cards in " + symbol + " Symbol");
            int sum = 0;
            for (Card card : cards) {
                System.out.println(card);
                sum += card.getNumber();
            }
            System.out.println("Number of cards : " + cards.size());
            System.out.println("Sum of Numbers : " + sum);
        }
    }
     }
```

4. **Output:**

```
Enter Number of Cards : 13
Enter card 1:
Symbol: s
Number: 1
Enter card 2:
Symbol: s
Number: 12
Enter card 3:
Symbol: s
Number: 13
Enter card 4:
Symbol: d
Number: 4
Enter card 5:
Symbol: c
Number: 5
Enter card 6:
Symbol: h
Number: 5
Enter card 7:
Symbol: h
Number: 7
```

```
Enter card 8:
Symbol: c
Number: 3
Enter card 9:
Symbol: c
Number: 2
Enter card 10:
Symbol: h
Number: 9
Enter card 11:
Symbol: s
Number: 7
Enter card 12:
Symbol: d
Number: 4
Enter card 13:
Symbol: d
Number: 3
```

```
Distinct Symbols are :
c d h s
Cards in c Symbol
c 5
c 3
c 2
Number of cards : 3
Sum of Numbers : 10
Cards in d Symbol
d 4
d 4
d 3
Number of cards : 3
Sum of Numbers : 11
Cards in h Symbol
h 5
h 7
h 9
Number of cards : 3
Sum of Numbers : 21
Cards in s Symbol
s 1
s 12
s 13
s 7
Number of cards : 4
Sum of Numbers : 33
```

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. **Algorithm:**

- **Initialize**: Create a TicketBookingSystem with available seats (e.g., an array of booleans).
- **Create Threads**: For each booking, create a BookingThread representing a customer (VIP or Regular) attempting to book a specific seat.
- **Assign Priority**: Set VIP threads' priority to Thread.MAX_PRIORITY and regular threads' priority to Thread.NORM_PRIORITY.
- Start all threads simultaneously.
- **Booking Process (in each thread)**:
  - Each thread calls a synchronized bookSeat() method.
  - If the seat is available, mark it as booked and print success.
  - if the seat is already booked, print failure.
- VIP threads are processed first due to their higher priority.
- Use synchronization to prevent multiple threads from booking the same seat.
- The program ends when all threads complete, and seats are successfully booked with no double bookings.

3. **Implementation/Code:**

```
class TicketBookingSystem {
    private boolean[] seats;
    public TicketBookingSystem(int numberOfSeats) {
        seats = new boolean[numberOfSeats];
    }
    public synchronized boolean bookSeat(int seatNumber, String customerType) {
        if (seatNumber >= 0 && seatNumber < seats.length && !seats[seatNumber]) {
            seats[seatNumber] = true;
            System.out.println(customerType + " booked seat " + seatNumber);
            return true;
        }
        System.out.println(customerType + " failed to book seat " + seatNumber + " (Already booked or
invalid seat number).");
        return false;
    }
}
class BookingThread extends Thread {
    private TicketBookingSystem bookingSystem;
```

```
    private int seatNumber;
    private String customerType;
    public BookingThread(TicketBookingSystem bookingSystem, int seatNumber, String customerType){
        this.bookingSystem = bookingSystem;
        this.seatNumber = seatNumber;
        this.customerType = customerType;
    }
    public void run() {
        boolean success = bookingSystem.bookSeat(seatNumber, customerType);
        if (success) {
            System.out.println(customerType + " successfully booked seat " + seatNumber);
        }
    }
}
public class Main {
    public static void main(String[] args) {
        TicketBookingSystem bookingSystem = new TicketBookingSystem(5);

        BookingThread thread1 = new BookingThread(bookingSystem, 1, "Regular");
        BookingThread thread2 = new BookingThread(bookingSystem, 2, "VIP");
        BookingThread thread3 = new BookingThread(bookingSystem, 1, "VIP");
        BookingThread thread4 = new BookingThread(bookingSystem, 3, "Regular");
        BookingThread thread5 = new BookingThread(bookingSystem, 2, "Regular");
        BookingThread thread6 = new BookingThread(bookingSystem, 0, "VIP");

        thread2.setPriority(Thread.MAX_PRIORITY);
        thread3.setPriority(Thread.MAX_PRIORITY);
        thread6.setPriority(Thread.MAX_PRIORITY);
        thread1.setPriority(Thread.NORM_PRIORITY);
        thread4.setPriority(Thread.NORM_PRIORITY);
        thread5.setPriority(Thread.NORM_PRIORITY);

        thread1.start();
        thread2.start();
        thread3.start();
        thread4.start();
        thread5.start();
        thread6.start();
    }
}
```

4. **Output:**

```
Regular booked seat 1
Regular successfully booked seat 1
VIP booked seat 0
VIP successfully booked seat 0
Regular booked seat 2
Regular successfully booked seat 2
Regular booked seat 3
Regular successfully booked seat 3
VIP failed to book seat 1 (Already booked or invalid seat number).
VIP failed to book seat 2 (Already booked or invalid seat number).
```