WORKSHEET 4

Student Name: VIVEK KUMAR **UID: 22BCS16136**

Branch: CSE **Section/Group: 903/B**

Semester: 6th **Date of Performance: 18/02/2025**

Subject Name: Project Based Learning in Java

(A) 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.

1. Source Code:

```
import java.util.*;

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;
    }

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

public class EmployeeManagement {
    private static final List<Employee> employees = new ArrayList<>();
    private static final Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        while (true) {
            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. Display All Employees");
        System.out.println("6. Exit");
        System.out.print("Enter your choice: ");
        int choice = scanner.nextInt();
        scanner.nextLine();

        switch (choice) {
            case 1:
                addEmployee();
                break;
            case 2:
                updateEmployee();
                break;
            case 3:
                removeEmployee();
                break;
            case 4:
                searchEmployee();
                break;
            case 5:
                displayEmployees();
                break;
            case 6:
                System.out.println("Exiting...");
                return;
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }
}

private static void addEmployee() {
    System.out.print("Enter Employee ID: ");
    int id = scanner.nextInt();
    scanner.nextLine();
    System.out.print("Enter Employee Name: ");
    String name = scanner.nextLine();
    System.out.print("Enter Employee Salary: ");
    double salary = scanner.nextDouble();
    scanner.nextLine();
```

```
        employees.add(new Employee(id, name, salary));
        System.out.println("Employee added successfully.");
    }

    private static void updateEmployee() {
        System.out.print("Enter Employee ID to update: ");
        int id = scanner.nextInt();
        scanner.nextLine();

        for (Employee emp : employees) {
            if (emp.id == id) {
                System.out.print("Enter New Name: ");
                emp.name = scanner.nextLine();
                System.out.print("Enter New Salary: ");
                emp.salary = scanner.nextDouble();
                scanner.nextLine();
                System.out.println("Employee updated successfully.");
                return;
            }
        }
        System.out.println("Employee not found.");
    }

    private static void removeEmployee() {
        System.out.print("Enter Employee ID to remove: ");
        int id = scanner.nextInt();
        scanner.nextLine();

        Iterator<Employee> iterator = employees.iterator();
        while (iterator.hasNext()) {
            if (iterator.next().id == id) {
                iterator.remove();
                System.out.println("Employee removed successfully.");
                return;
            }
        }
        System.out.println("Employee not found.");
    }

    private static void searchEmployee() {
        System.out.print("Enter Employee ID to search: ");
        int id = scanner.nextInt();
        scanner.nextLine();

        for (Employee emp : employees) {
```

```
            if (emp.id == id) {
                System.out.println("Employee Found: " + emp);
                return;
            }
        }
        System.out.println("Employee not found.");
    }

    private static void displayEmployees() {
        if (employees.isEmpty()) {
            System.out.println("No employees found.");
        } else {
            System.out.println("\nEmployee List:");
            for (Employee emp : employees) {
                System.out.println(emp);
            }
        }
    }
}
```

(B) 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.

Source Code –

```
import java.util.*;

class Card {
    private String symbol;
    private String value;

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

    public String getSymbol() {
        return symbol;
    }

    public String getValue() {
        return value;
    }
```

```
    @Override
    public String toString() {
        return "Card{Symbol='" + symbol + "', Value='" + value + "'}";
    }
}

public class CardCollection {
    private static final Collection<Card> cards = new ArrayList<>();
    private static final Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        while (true) {
            System.out.println("\nCard Collection System");
            System.out.println("1. Add Card");
            System.out.println("2. Remove Card");
            System.out.println("3. Search Cards by Symbol");
            System.out.println("4. Display All Cards");
            System.out.println("5. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
            scanner.nextLine();

            switch (choice) {
                case 1:
                    addCard();
                    break;
                case 2:
                    removeCard();
                    break;
                case 3:
                    searchBySymbol();
                    break;
                case 4:
                    displayCards();
                    break;
                case 5:
                    System.out.println("Exiting...");
                    return;
                default:
                    System.out.println("Invalid choice. Please try again.");
            }
        }
    }
```

```
private static void addCard() {
    System.out.print("Enter Card Symbol: ");
    String symbol = scanner.nextLine();
    System.out.print("Enter Card Value: ");
    String value = scanner.nextLine();
    cards.add(new Card(symbol, value));
    System.out.println("Card added successfully.");
}

private static void removeCard() {
    System.out.print("Enter Card Symbol to remove: ");
    String symbol = scanner.nextLine();

    Iterator<Card> iterator = cards.iterator();
    while (iterator.hasNext()) {
        if (iterator.next().getSymbol().equalsIgnoreCase(symbol)) {
            iterator.remove();
            System.out.println("Card removed successfully.");
            return;
        }
    }
    System.out.println("Card not found.");
}

private static void searchBySymbol() {
    System.out.print("Enter Symbol to search: ");
    String symbol = scanner.nextLine();
    boolean found = false;

    for (Card card : cards) {
        if (card.getSymbol().equalsIgnoreCase(symbol)) {
            System.out.println(card);
            found = true;
        }
    }

    if (!found) {
        System.out.println("No cards found with the given symbol.");
    }
}

private static void displayCards() {
    if (cards.isEmpty()) {
        System.out.println("No cards in the collection.");
    } else {
```

```
            System.out.println("\nCard Collection:");
            for (Card card : cards) {
                System.out.println(card);
            }
        }
    }
}
```

(C) 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.

Source code –

```
import java.util.*;

class TicketBookingSystem {
    private final boolean[] seats;

    public TicketBookingSystem(int numberOfSeats) {
        this.seats = new boolean[numberOfSeats]; // False means available, true means booked
    }

    public synchronized boolean bookSeat(int seatNumber, String customer) {
        if (seatNumber < 0 || seatNumber >= seats.length) {
            System.out.println(customer + " - Invalid seat number.");
            return false;
        }
        if (!seats[seatNumber]) {
            seats[seatNumber] = true;
            System.out.println(customer + " successfully booked seat " + seatNumber);
            return true;
        } else {
            System.out.println(customer + " - Seat " + seatNumber + " is already booked.");
            return false;
        }
    }
}

class Customer extends Thread {
    private final TicketBookingSystem system;
```

```
    private final int seatNumber;
    private final String customerName;

    public Customer(TicketBookingSystem system, int seatNumber, String customerName, int priority) {
        this.system = system;
        this.seatNumber = seatNumber;
        this.customerName = customerName;
        setPriority(priority); // Higher priority for VIPs
    }

    @Override
    public void run() {
        system.bookSeat(seatNumber, customerName);
    }
}

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

        // Creating normal and VIP customers
        customers.add(new Customer(system, 3, "VIP_Alice", Thread.MAX_PRIORITY));
        customers.add(new Customer(system, 3, "Bob", Thread.NORM_PRIORITY));
        customers.add(new Customer(system, 5, "VIP_Charlie", Thread.MAX_PRIORITY));
        customers.add(new Customer(system, 5, "David", Thread.NORM_PRIORITY));
        customers.add(new Customer(system, 1, "Eve", Thread.MIN_PRIORITY));

        // Shuffle to simulate real-world randomness
        Collections.shuffle(customers);

        // Start booking threads
        for (Thread customer : customers) {
            customer.start();
        }

        // Wait for all threads to finish
        for (Thread customer : customers) {
            try {
                customer.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
```

```
}
```