Exp-4

1. 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.(Easy)
   Code –

```
import java.util.*;

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

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

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

        while (true) {
            System.out.println("\n1. Add Employee\n2. Update Employee\n3. Remove
Employee\n4. Search Employee\n5. Display Employees\n6. Exit");
            System.out.print("Choose an option: ");
            int choice = scanner.nextInt();
            scanner.nextLine();
```

```
            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.getId() == updateId) {
                            System.out.print("Enter new Salary: ");
                            emp.setSalary(scanner.nextDouble());
                            System.out.println("Employee salary updated.");
                            break;
                        }
                    }
                    break;

                case 3:
                    System.out.print("Enter Employee ID to remove: ");
                    int removeId = scanner.nextInt();
                    employees.removeIf(emp -> emp.getId() == removeId);
                    System.out.println("Employee removed.");
                    break;

                case 4:
                    System.out.print("Enter Employee ID to search: ");
                    int searchId = scanner.nextInt();
                    for (Employee emp : employees) {
                        if (emp.getId() == searchId) {
                            System.out.println(emp);
                            break;
                        }
                    }
                    break;

                case 5:
                    System.out.println("\nEmployee List:");
                    for (Employee emp : employees) {
                        System.out.println(emp);
                    }
                    break;

                case 6:
                    System.out.println("Exiting...");
                    scanner.close();
                    return;

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

```
            }
        }
    }
}
```

Output—

```
Output    Generated files


1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Choose an option: 1
Enter ID: 123
Enter Name: Tarun Seth
Enter Salary: 1000000
Employee added successfully.

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Choose an option: 2
Enter Employee ID to update: 321

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Choose an option: 3
Enter Employee ID to remove: 123
Employee removed.
```

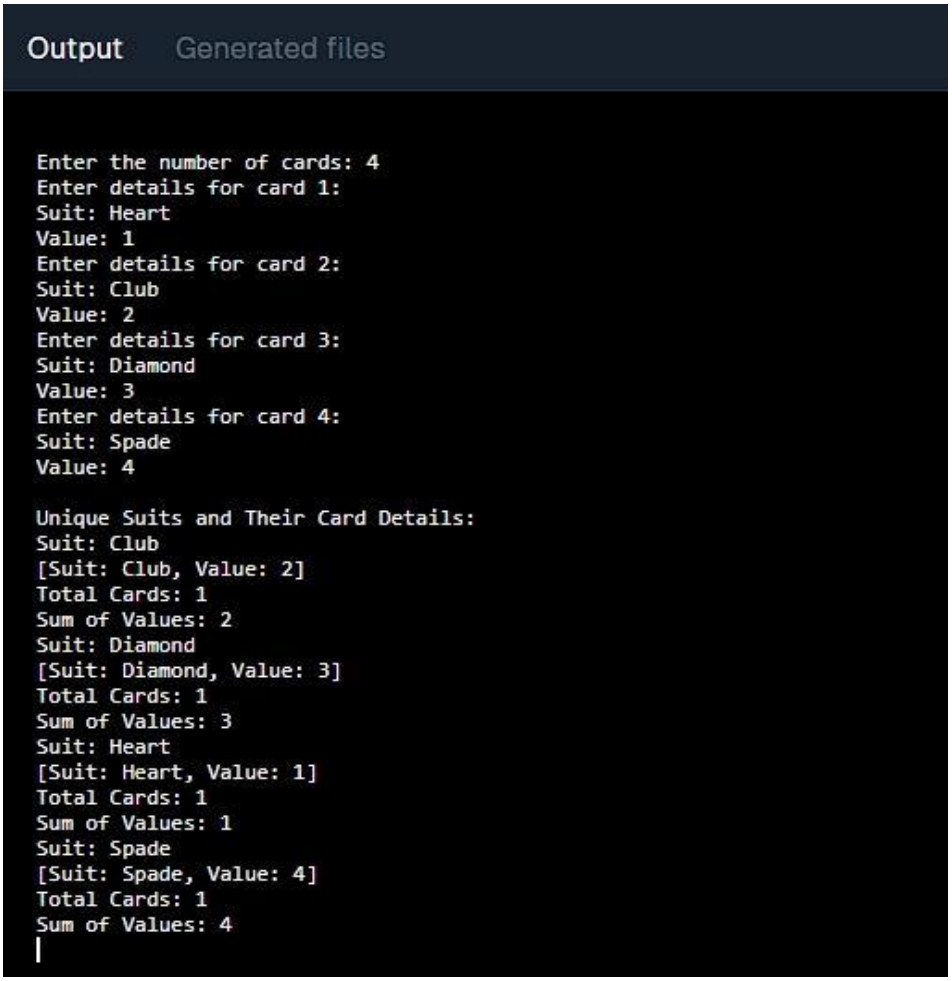
2. Create a program to collect and store all the cards to assist the users in finding all the cards in a given symbol using Collection interface.(Medium)

   Code –

```java
import java.util.*;

class PlayingCard {
    private String suit;
    private int value;

    public PlayingCard(String suit, int value) {
        this.suit = suit;
        this.value = value;
    }

    public String getSuit() {
        return suit;
    }

    public int getValue() {
        return value;
    }

    @Override
```

```
    public String toString() {
        return "[Suit: " + suit + ", Value: " + value + "]";
    }
}

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

        System.out.print("Enter the number of cards: ");
        int cardCount = scanner.nextInt();
        scanner.nextLine();

        for (int i = 1; i <= cardCount; i++) {
            System.out.println("Enter details for card " + i + ":");
            System.out.print("Suit: ");
            String suit = scanner.nextLine();
            System.out.print("Value: ");
            int value = scanner.nextInt();
            scanner.nextLine();

            cardGroups.computeIfAbsent(suit, k -> new ArrayList<>()).add(new
PlayingCard(suit, value));
        }

        displayCardDetails(cardGroups);
        scanner.close();
    }

    private static void displayCardDetails(Map<String, List<PlayingCard>> cardGroups) {
        System.out.println("\nUnique Suits and Their Card Details:");
        for (Map.Entry<String, List<PlayingCard>> entry : cardGroups.entrySet()) {
            String suit = entry.getKey();
            List<PlayingCard> cards = entry.getValue();
            int totalValue = cards.stream().mapToInt(PlayingCard::getValue).sum();

            System.out.println("Suit: " + suit);
            cards.forEach(System.out::println);
            System.out.println("Total Cards: " + cards.size());
            System.out.println("Sum of Values: " + totalValue);
        }
    }
}
```

Output –



3. 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.(Hard)
   Code –

```
import java.util.*;

class SeatBookingSystem {
    private final boolean[] seats;

    public SeatBookingSystem(int totalSeats) {
        this.seats = new boolean[totalSeats];
    }

    public synchronized boolean bookSeat(int seatNumber) {
        if (seatNumber < 0 || seatNumber >= seats.length || seats[seatNumber]) {
```

```
            return false; // Seat already booked or invalid
        }
        seats[seatNumber] = true;
        return true;
    }
}

class BookingThread extends Thread {
    private SeatBookingSystem system;
    private int seatNumber;

    public BookingThread(SeatBookingSystem system, int seatNumber, int priority) {
        this.system = system;
        this.seatNumber = seatNumber;
        setPriority(priority);
    }

    @Override
    public void run() {
        if (system.bookSeat(seatNumber)) {
            System.out.println(Thread.currentThread().getName() + " successfully booked
seat " + seatNumber);
        } else {
            System.out.println(Thread.currentThread().getName() + " failed to book seat " +
seatNumber);
        }
    }
}

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

        // Creating threads with different priorities
        for (int i = 0; i < 10; i++) {
            int priority = (i % 3 == 0) ? Thread.MAX_PRIORITY : Thread.MIN_PRIORITY; //
VIPs get highest priority
            Thread thread = new BookingThread(system, i, priority);
            threads.add(thread);
        }

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

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

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

```
}
```

Output—

```
Thread-2 successfully booked seat 2
Thread-1 successfully booked seat 1
Thread-6 successfully booked seat 6
Thread-8 successfully booked seat 8
Thread-0 successfully booked seat 0
Thread-7 successfully booked seat 7
Thread-3 successfully booked seat 3
Thread-4 successfully booked seat 4
Thread-5 successfully booked seat 5
Thread-9 successfully booked seat 9
All bookings processed.
```