Experiment 4

Student Name: Sahil **UID: 23BCS10928**
Branch: CSE **Section:901/B**
Semester: 6th **DOP:20-02-25**
Subject: Java **Subject Code: 22CSH-359**

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.

Objective: The objective of this program is to implement an Employee Management System using an ArrayList. It allows users to add, update, remove, search, and display employee details efficiently.

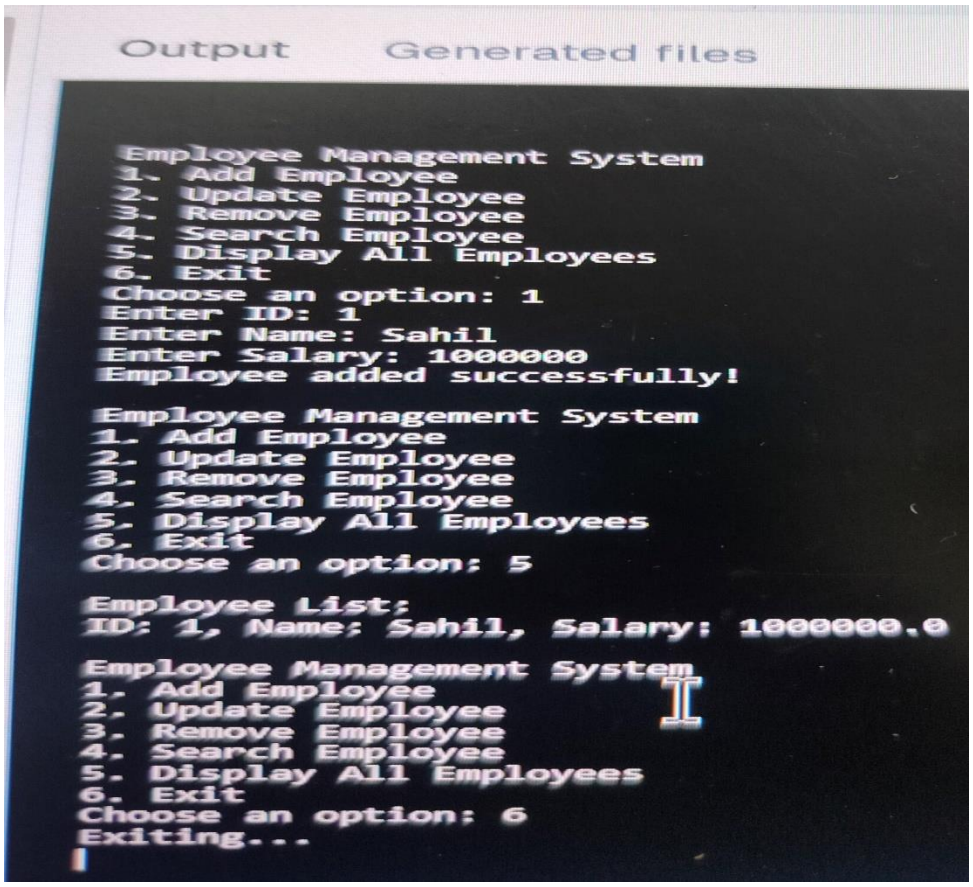
Code:

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

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

        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("Choose an option: ");
int choice = scanner.nextInt();
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: ");
        id = scanner.nextInt();
        for (Employee emp : employees) {
            if (emp.id == id) {
                scanner.nextLine();
                System.out.print("Enter New Name: ");
                emp.name = scanner.nextLine();
                System.out.print("Enter New Salary: ");
                emp.salary = scanner.nextDouble();
                System.out.println("Employee updated successfully!");
                break;
            }
        }
        break;
    case 3:
        System.out.print("Enter Employee ID to remove: ");
        id = scanner.nextInt();
        employees.removeIf(emp -> emp.id == id);
        System.out.println("Employee removed successfully!");
        break;
    case 4:
        System.out.print("Enter Employee ID to search: ");
        id = scanner.nextInt();
        for (Employee emp : employees) {
            if (emp.id == id) {
                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 option! Please try again.");
        }
    }
  }
}
```

Output:



Program:B

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.

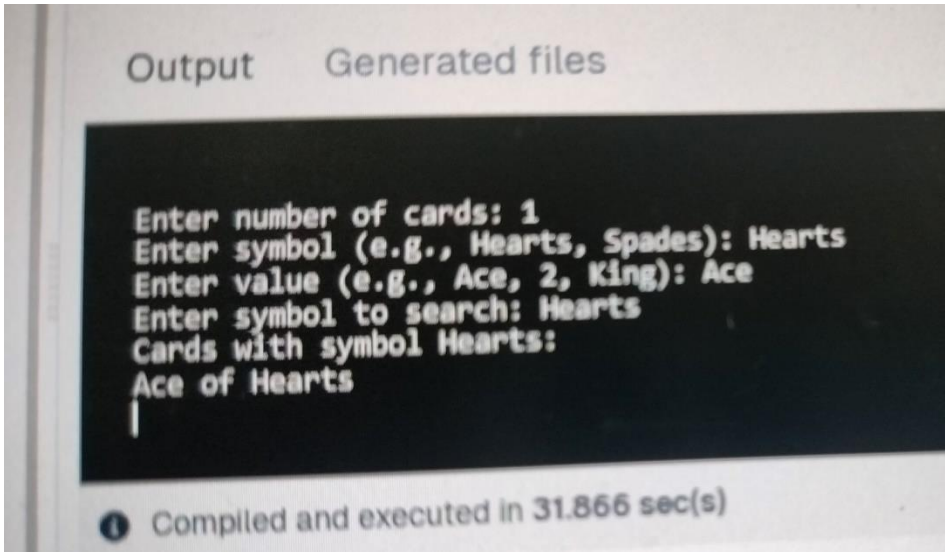
Objective: The objective of this program is to store and manage a collection of playing cards using the Collection interface. It allows users to add cards and search for all cards of a given symbol efficiently.

Code:

```
import java.util.*;

class Card {
    String symbol;
    String value;

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

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

public class CardCollection {
    public static void main(String[] args) {
        Collection<Card> cards = new ArrayList<>();
        Scanner scanner = new Scanner(System.in);

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

        for (int i = 0; i < n; i++) {
            System.out.print("Enter symbol (e.g., Hearts, Spades): ");
            String symbol = scanner.nextLine();
            System.out.print("Enter value (e.g., Ace, 2, King): ");
            String value = scanner.nextLine();
            cards.add(new Card(symbol, value));
        }

        System.out.print("Enter symbol to search: ");
        String searchSymbol = scanner.nextLine();
```

```
        System.out.println("Cards with symbol " + searchSymbol + ":");
        for (Card card : cards)

{
            if (card.symbol.equalsIgnoreCase(searchSymbol)) {
                System.out.println(card);
            }
        }

        scanner.close();
    }
}
```

Output:



Program:C

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.

Objective: The objective of this program is to develop a synchronized ticket booking system that prevents double booking of seats. It uses multithreading to handle multiple booking requests simultaneously. Thread priorities are used to ensure VIP bookings are processed first.

Code:

```
import java.util.*;

class TicketBookingSystem {
    private final Set<Integer> bookedSeats = new HashSet<>();
    private final int totalSeats;

    public TicketBookingSystem(int totalSeats) {
        this.totalSeats = totalSeats;
    }

    public synchronized boolean bookSeat(int seatNumber, String user) {
        if (bookedSeats.contains(seatNumber)) {
            System.out.println(user + " tried to book Seat " + seatNumber + " but it is already
booked.");
            return false;
        }

        bookedSeats.add(seatNumber);
        System.out.println(user + " successfully booked Seat " + seatNumber);
        return true;
    }
}

class BookingThread extends Thread {
    private final TicketBookingSystem system;
    private final int seatNumber;
    private final String user;

    public BookingThread(TicketBookingSystem system, int seatNumber, String user, int priority) {
        this.system = system;
        this.seatNumber = seatNumber;
        this.user = user;
        setPriority(priority);
    }
```

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

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

        // Creating VIP and normal users
        threads.add(new BookingThread(system, 3, "VIP User 1", Thread.MAX_PRIORITY));
        threads.add(new BookingThread(system, 3, "Normal User 1", Thread.NORM_PRIORITY));
        threads.add(new BookingThread(system, 5, "VIP User 2", Thread.MAX_PRIORITY));
        threads.add(new BookingThread(system, 5, "Normal User 2", Thread.NORM_PRIORITY));
        threads.add(new BookingThread(system, 2, "Normal User 3", Thread.NORM_PRIORITY));

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

        for (Thread t : threads) {
            try {
                t.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Output: