Experiment 4 .1

Student Name: Aashna deep **UID:22BCS10833**
Branch: CSE **Section/Group:643/B**
Semester: 6th **Date of Performance:24/02/25**
Subject Name: PBLJ **Subject Code: 22CSH-359**

1. Aim: Write a Java program to implement an Array List that stores employee details (ID, Name, and Salary). Allow users to add, update, remove, and search employees.

2. Procedure :

Create a Java Class (Employee)

- Define attributes: id, name, and salary.
- Implement a constructor and getter/setter methods.
- Override toString() for proper display of employee details.

Create a Main Class (EmployeeManagement)

- Declare an ArrayList<Employee> to store employee details.
- Implement methods for:
  - Adding an employee (taking input from the user).
  - Updating employee details (search by ID and modify details).
  - Removing an employee (search by ID and delete).
  - Searching for an employee (retrieve employee details using ID).
  - Displaying all employees.

Implement a Menu-driven System

- Use a Scanner to take user input.
- Provide options to add, update, remove, search, and display employees.
- Use a while loop to keep the menu running until the user exits.

3. Code:

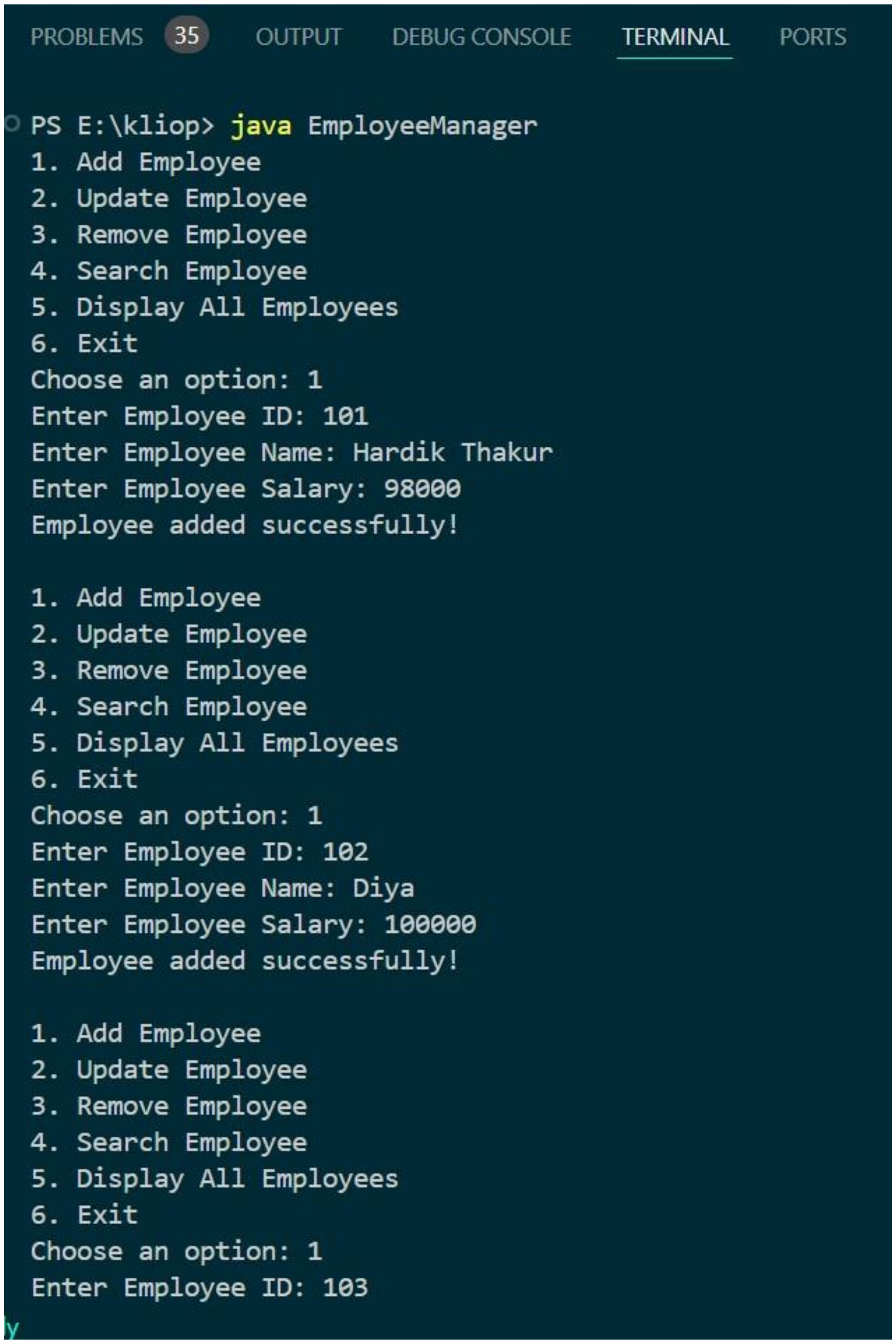
```
import java.util.ArrayList;
import java.util.Scanner;

// Employee class 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;
    }

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

// Employee Management System public
class EmployeeManagement {
    private static ArrayList<Employee> employees = new ArrayList<>();
private static 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(); // Consume newline

            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 Employee Management System...");
return;                 default:
                    System.out.println("Invalid choice! Please enter a valid option.");
            }
        }
    }

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

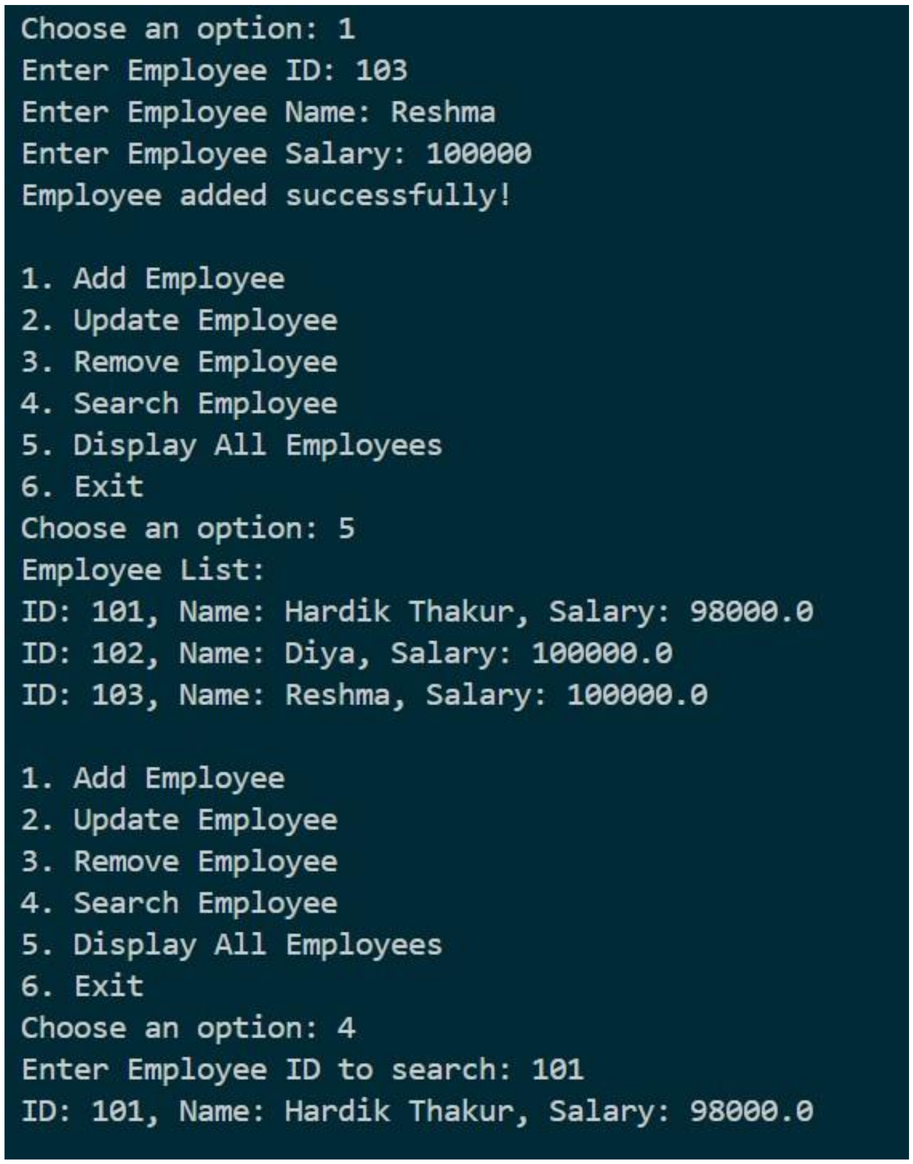
```
  double salary = scanner.nextDouble();

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

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

        for (Employee emp : employees) {
if (emp.getId() == id) {
                System.out.print("Enter new name: ");
                String newName = scanner.nextLine();
System.out.print("Enter new salary: ");
                double newSalary = scanner.nextDouble();

                emp.setName(newName);
                emp.setSalary(newSalary);

                System.out.println("Employee updated successfully!");
return;
            }
        }
        System.out.println("Employee ID not found.");
    }

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

        for (Employee emp : employees) {
if (emp.getId() == id) {
employees.remove(emp);
                System.out.println("Employee removed successfully!");
return;
            }
        }
        System.out.println("Employee ID not found.");
```

```
}

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

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

    // Display All Employees      private
static void displayEmployees() {          if
(employees.isEmpty()) {
            System.out.println("No employees to display.");
        } else {
            System.out.println("\nList of Employees:");
for (Employee emp : employees) {
                System.out.println(emp);
            }
        }
    }
}
```

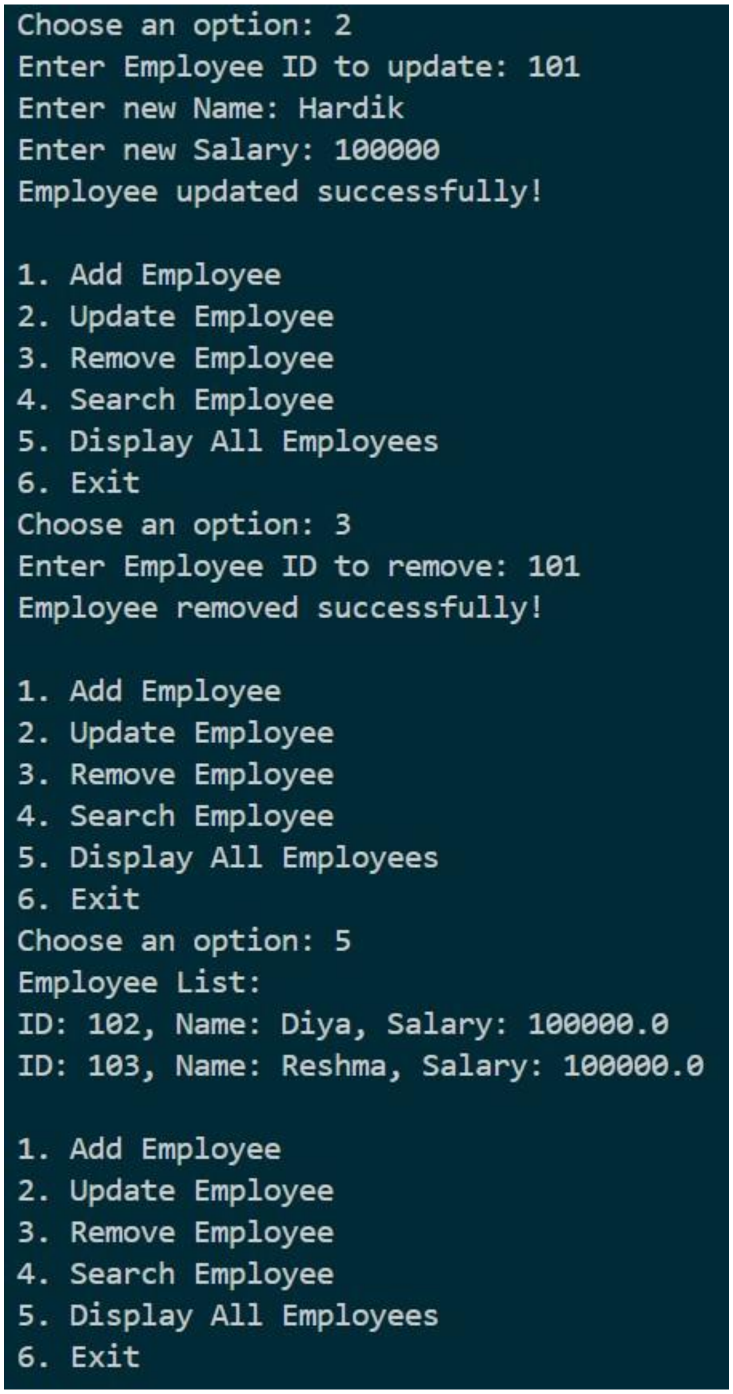
4. Output



```
PS E:\kliop> java EmployeeManager
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Choose an option: 1
Enter Employee ID: 101
Enter Employee Name: Hardik Thakur
Enter Employee Salary: 98000
Employee added successfully!

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Choose an option: 1
Enter Employee ID: 102
Enter Employee Name: Diya
Enter Employee Salary: 100000
Employee added successfully!

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Choose an option: 1
Enter Employee ID: 103
```



```
Choose an option: 1
Enter Employee ID: 103
Enter Employee Name: Reshma
Enter Employee Salary: 100000
Employee added successfully!

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Choose an option: 5
Employee List:
ID: 101, Name: Hardik Thakur, Salary: 98000.0
ID: 102, Name: Diya, Salary: 100000.0
ID: 103, Name: Reshma, Salary: 100000.0

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Choose an option: 4
Enter Employee ID to search: 101
ID: 101, Name: Hardik Thakur, Salary: 98000.0
```



```
Choose an option: 2
Enter Employee ID to update: 101
Enter new Name: Hardik
Enter new Salary: 100000
Employee updated successfully!

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Choose an option: 3
Enter Employee ID to remove: 101
Employee removed successfully!

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Choose an option: 5
Employee List:
ID: 102, Name: Diya, Salary: 100000.0
ID: 103, Name: Reshma, Salary: 100000.0

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
```

Learning Outcomes:

1. Understanding ArrayList in Java ○Learn how to use ArrayList for dynamic storage of objects.

2. Object-Oriented Programming (OOP) Concepts ○Implement Encapsulation using getter and setter methods. ○Use Constructors to initialize object data.
   - ○ Understand toString() method to format object output.

3. Handling User Input Efficiently ○Learn to use Scanner to take and process user input.

4. Implementing CRUD Operations ○Create (Add Employee)
    - ○ Read (Search and Display Employee Details) ○Update (Modify Employee Details)
    - ○ Delete (Remove Employee from List)

5. Implementing a Menu-Driven Program ○Use loops and switch-case to create an interactive console-based system.

6. Exception Handling Considerations ○Understand how to handle user input validation and avoid errors.

Experiment 4 .2

Student Name: Aashna Deep	**UID:22BCS10833**
Branch: CSE	**Section/Group:643/B**
Semester: 6th	**Date of Performance:24/02/25**
Subject Name: PBLJ	**Subject Code: 22CSH-359**

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 the Collection interface.**

2. **Procedure:**

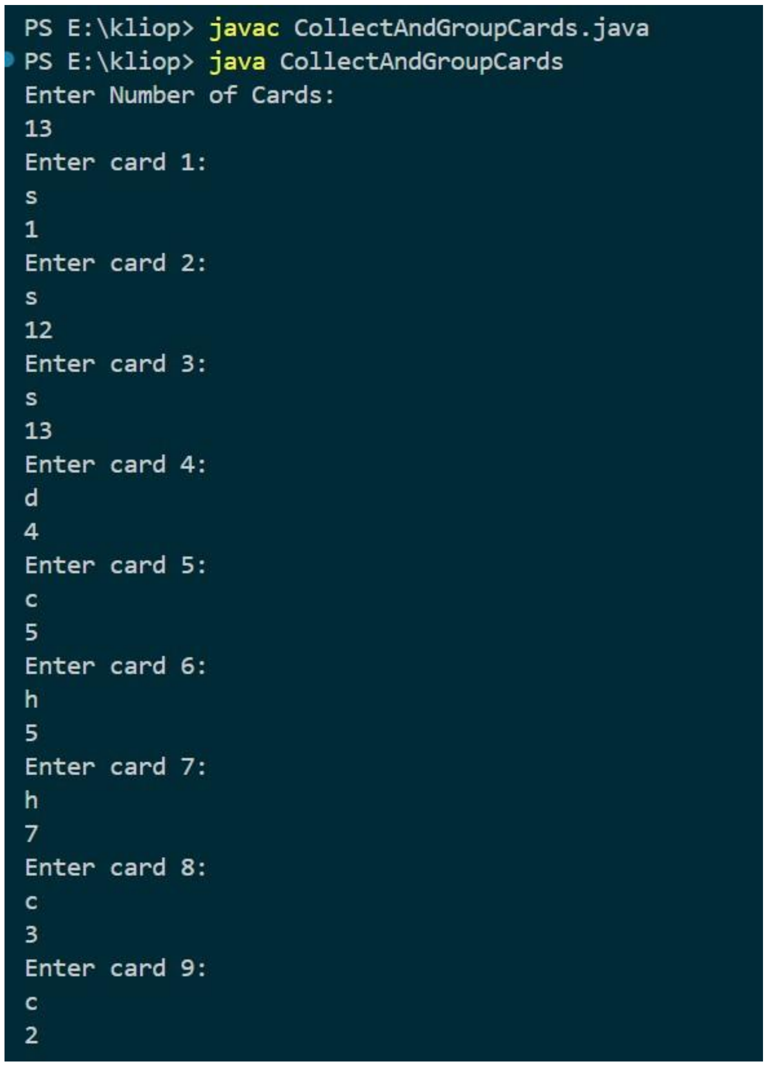
- Create a Card class with attributes symbol and rank.
- Use an ArrayList to store card objects.
- Implement functions:
- Add Card → Take user input and store the card.
- Display Cards → Show all stored cards.
- Search by Symbol → Find and display matching cards.
- Use a menu-driven approach with a while loop to let users add, search, display, or exit.
- Exit the program when the user selects the exit option.

3. **Code**

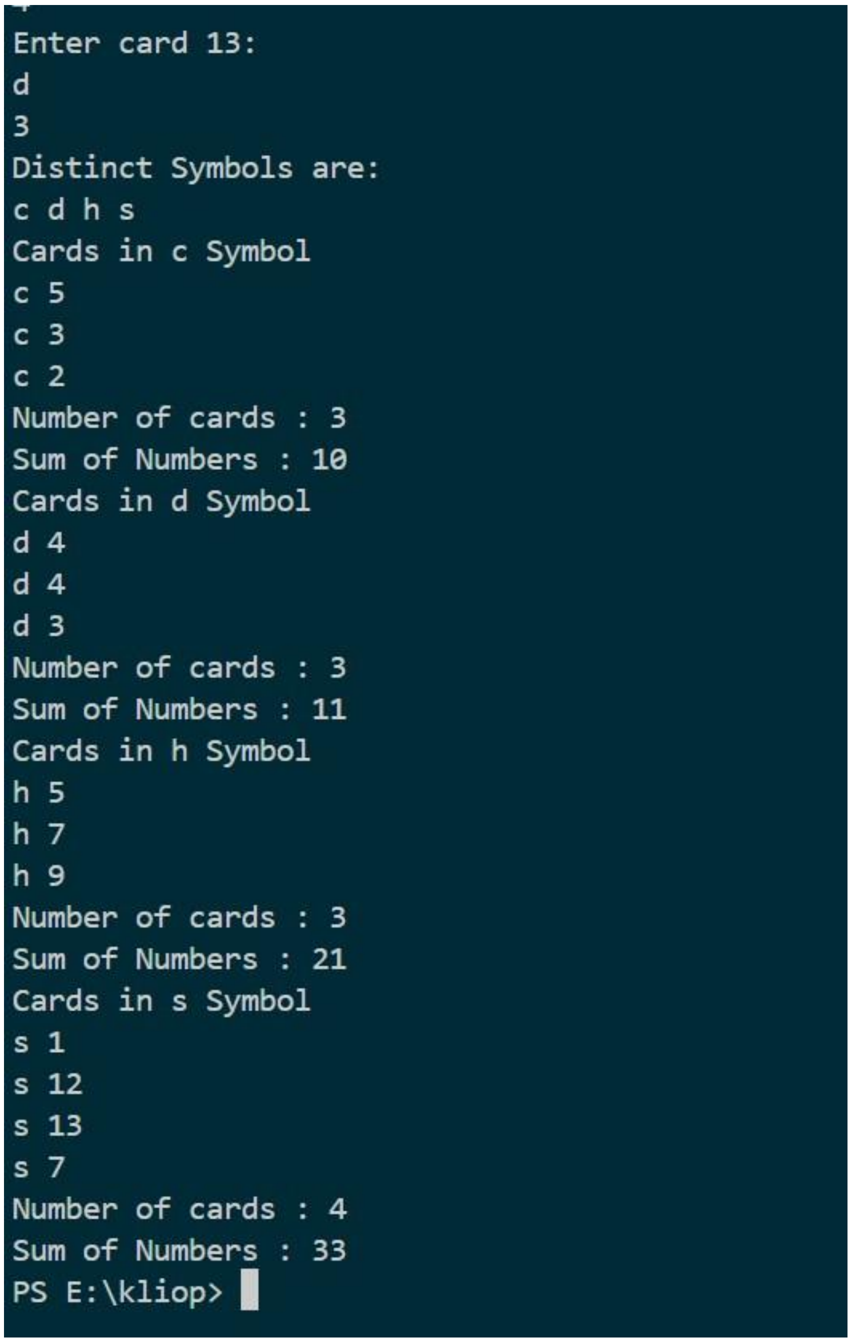
```java
import java.util.*;

    classCard {
    String symbol;
int number;

    publicCard(Stringsymbol, intnumber) {
    this.symbol= symbol;
    this.number= number;
    }
}

         publicclassCollectAndGroupCards {
         publicstaticvoidmain(String[] args) {
         Scanner scanner = new Scanner(System.in);
         Map<String, List<Integer>> cardMap = new TreeMap<>();

System.out.println("Enter Number of Cards: ");
int
n =scanner.nextInt();

              for (int i =1; i <= n; i++) {
              System.out.println("Enter card "+ i +":");
              String symbol =scanner.next();
              int number =scanner.nextInt();

    cardMap.putIfAbsent(symbol, new ArrayList<>());
    cardMap.get(symbol).add(number);
          }

    System.out.println("Distinct Symbols are:");
    for (String symbol : cardMap.keySet()) {
    System.out.print(symbol +"");
         }
System.out.println();
```

```

    for (String symbol : cardMap.keySet()) {
    List<Integer> numbers =cardMap.get(symbol);
    System.out.println("Cards in "+ symbol +" Symbol");
    for (int num : numbers) {
    System.out.println(symbol +""+ num);            }
    System.out.println("Number of cards : "+numbers.size());
    System.out.println("Sum of Numbers : "+numbers.stream().mapToInt(Integer::intValue).sum());        }

    scanner.close();
    } }

```

4. Output



```
PS E:\kliop> javac CollectAndGroupCards.java
PS E:\kliop> java CollectAndGroupCards
Enter Number of Cards:
13
Enter card 1:
s
1
Enter card 2:
s
12
Enter card 3:
s
13
Enter card 4:
d
4
Enter card 5:
c
5
Enter card 6:
h
5
Enter card 7:
h
7
Enter card 8:
c
3
Enter card 9:
c
2
```



```
Enter card 13:
d
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
PS E:\kliop>
```

5. **Learning Outcomes:**

   1. Use of Maps and Lists – Store and group data efficiently.
   2. OOP Concepts – Create and use classes (Card class).
   3. Sorting & Grouping – Automatically sort symbols using TreeMap.
   4. Iteration & Aggregation – Loop through data, count cards, and sum numbers.
   5. User Input Handling – Read and process multiple inputs efficiently.

Experiment 4 .3

Student Name: Aashna Deep **UID:22BCS10833**
Branch: CSE **Section/Group:643/B**
Semester: 6th **Date of Performance:24/02/25**
Subject Name: PBLJ **Subject Code: 22CSH-359**

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—java code of it.

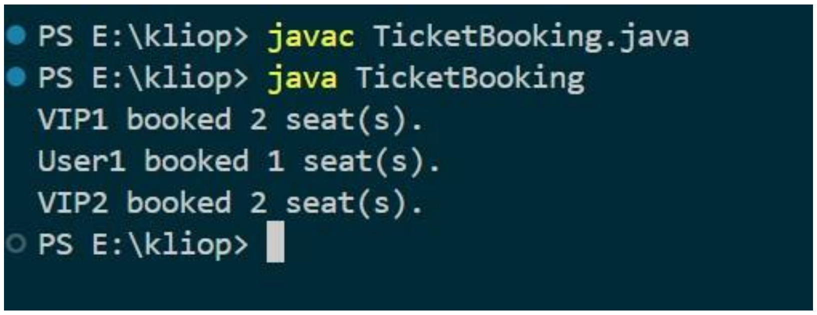
2. Procedure:

1. Initialize System: Create a TicketBookingSystem with available seats.
2. Create Threads: Instantiate Customer threads with different priorities.
3. Start Threads: Run threads to attempt ticket booking.
4. Synchronization: bookTicket ensures no double booking.
5. Process Completion: Threads execute based on priority and availability.

3. Code:

```
class TicketBookingSystem {
    private int availableSeats;

    public TicketBookingSystem(int seats) {
        this.availableSeats = seats;
    }

    public synchronized void bookTicket(String user, int seats) {
if (availableSeats >= seats) {
            System.out.println(user + " booked " + seats + " seat(s).");
availableSeats -= seats;
        } else {
            System.out.println(user + " booking failed. Not enough seats.");
        }
    }
}

class Customer extends Thread {
    private TicketBookingSystem system;
private int seats;

    public Customer(TicketBookingSystem system, String name, int seats)
{        super(name);        this.system = system;        this.seats = seats;
    }

    public void run() {
        system.bookTicket(getName(), seats);
    }
}

public class TicketBooking {
    public static void main(String[] args) {
        TicketBookingSystem system = new TicketBookingSystem(5);

        Customer c1 = new Customer(system, "VIP1", 2);
        Customer c2 = new Customer(system, "VIP2", 2);
        Customer c3 = new Customer(system, "User1", 1);

        c1.setPriority(Thread.MAX_PRIORITY);
c2.setPriority(Thread.MAX_PRIORITY);
        c3.setPriority(Thread.NORM_PRIORITY);

        c1.start();
c2.start();
c3.start();
    }
}
```

4. Output:



```
PS E:\kliop> javac TicketBooking.java
PS E:\kliop> java TicketBooking
VIP1 booked 2 seat(s).
User1 booked 1 seat(s).
VIP2 booked 2 seat(s).
PS E:\kliop>
```

5. **Learning Outcomes:**

- Understanding **thread synchronization** in Java.
- Implementing **thread priorities** for VIP bookings.
- Ensuring **safe concurrent access** to shared resources.
- Practical experience with **multithreading** concepts.