Experiment-4

Student Name: Gagnesh Kakkar
UID: 23BCS11196
Branch: B.E-C.S.E
Section/Group: 23KRG-2B
Semester: 5th
Date of Performance: 18/08/2025
Subject Name: PBLJ
Subject Code: 23CSH-304

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. **Objective:** To understand how to use Java Collections, specifically ArrayList, to manage dynamic data efficiently.

3. **Input/Apparatus Used:** Java ArrayList, Scanner, for-each loop, object-oriented programming.

4. **Procedure:**
   1. Create an Employee class with id, name, and salary.
   2. Use an ArrayList<Employee> to store the employee objects.
   3. Provide menu-driven options for:
   4. Add employee
   5. Update employee by ID
   6. Remove employee by ID
   7. Search employee by ID
   8. Loop over the list for searching/updating/removing.

5. **Sample Input:**
   Add Employee: ID=101, Name=John, Salary=50000
   Add Employee: ID=102, Name=Alice, Salary=60000
   Search Employee by ID: 101

   Sample Output:

Employee Found: ID=101, Name=John, Salary=50000Employee Found: ID=101, Name=John, Salary=50000

6. Code:

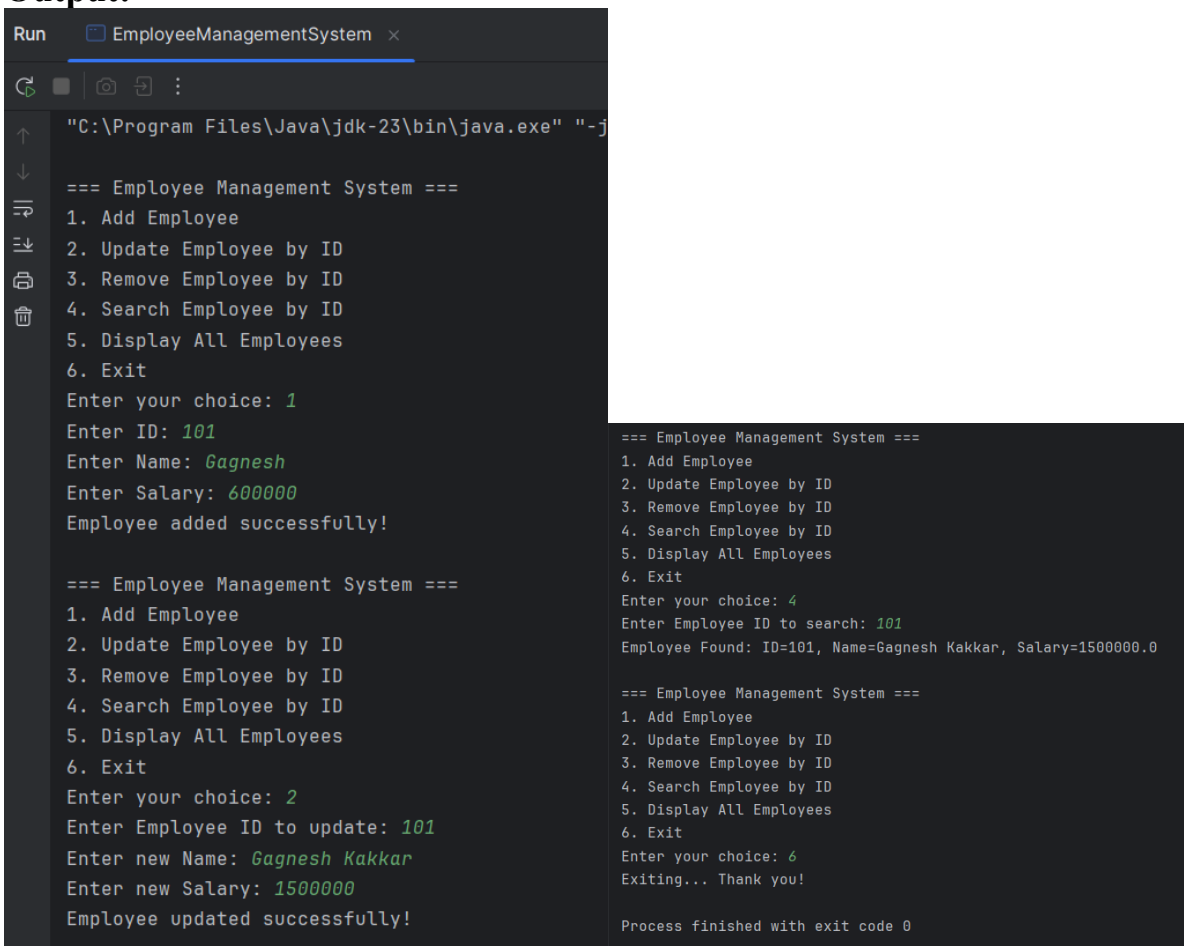
EXPERIMENT-4.java

```java
package PBLJ.Experiments;


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

class Employee {  5 usages
    int id;  5 usages
    String name;  3 usages
    double salary;  3 usages

    public Employee(int id, String name, double salary) {  1 usage
        this.id = id;
        this.name = name;
        this.salary = salary;
    }

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

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

        int choice;
        do {
            System.out.println("\n=== Employee Management System ===");
            System.out.println("1. Add Employee");
            System.out.println("2. Update Employee by ID");
            System.out.println("3. Remove Employee by ID");
            System.out.println("4. Search Employee by ID");
            System.out.println("5. Display All Employees");
            System.out.println("6. Exit");
            System.out.print("Enter your choice: ");
            choice = sc.nextInt();
```

```java
        switch (choice) {
            case 1:
                System.out.print("Enter ID: ");
                int id = sc.nextInt();
                sc.nextLine();
                System.out.print("Enter Name: ");
                String name = sc.nextLine();
                System.out.print("Enter Salary: ");
                double salary = sc.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 = sc.nextInt();
                boolean foundUpdate = false;
                for (Employee emp : employees) {
                    if (emp.id == updateId) {
                        sc.nextLine();
                        System.out.print("Enter new Name: ");
                        emp.name = sc.nextLine();
                        System.out.print("Enter new Salary: ");
                        emp.salary = sc.nextDouble();
                        System.out.println("Employee updated successfully!");
                        foundUpdate = true;
                        break;
                    }
                }
                if (!foundUpdate) {
                    System.out.println("Employee with ID " + updateId + " not found.");
                }
                break;
```

```
            case 3:
                System.out.print("Enter Employee ID to remove: ");
                int removeId = sc.nextInt();
                boolean removed = employees.removeIf( Employee emp -> emp.id == removeId);
                if (removed) {
                    System.out.println("Employee removed successfully!");
                } else {
                    System.out.println("Employee with ID " + removeId + " not found.");
                }
                break;

            case 4:
                System.out.print("Enter Employee ID to search: ");
                int searchId = sc.nextInt();
                boolean foundSearch = false;
                for (Employee emp : employees) {
                    if (emp.id == searchId) {
                        System.out.println("Employee Found: " + emp);
                        foundSearch = true;
                        break;
                    }
                }
                if (!foundSearch) {
                    System.out.println("Employee with ID " + searchId + " not found.");
                }
                break;

            case 5:
                System.out.println("\n=== Employee List ===");
                if (employees.isEmpty()) {
                    System.out.println("No employees found.");
                } else {
                    for (Employee emp : employees) {
                        System.out.println(emp);
                    }
                }
                break;

            case 6:
                System.out.println("Exiting... Thank you!");
                break;

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

7. Output:



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. Objective: To understand collection interfaces like Map, List, and how to store and retrieve grouped data.

3. Input/Apparatus Used: To understand collection interfaces like Map, List, and how to store and retrieve grouped data.

4. Procedure:

1. Define a Card class with attributes like symbol, number.
2. Use a HashMap<String, ArrayList<Card>> where the key is the symbol.
3. Populate the map by grouping cards with the same symbol.
4. Allow users to input a symbol to retrieve all matching cards.

5.

Sample Input:

Enter symbol: Spade

Sample Output :

Cards with symbol 'Spade':

Spade - 1

Spade - 3

Spade - 10

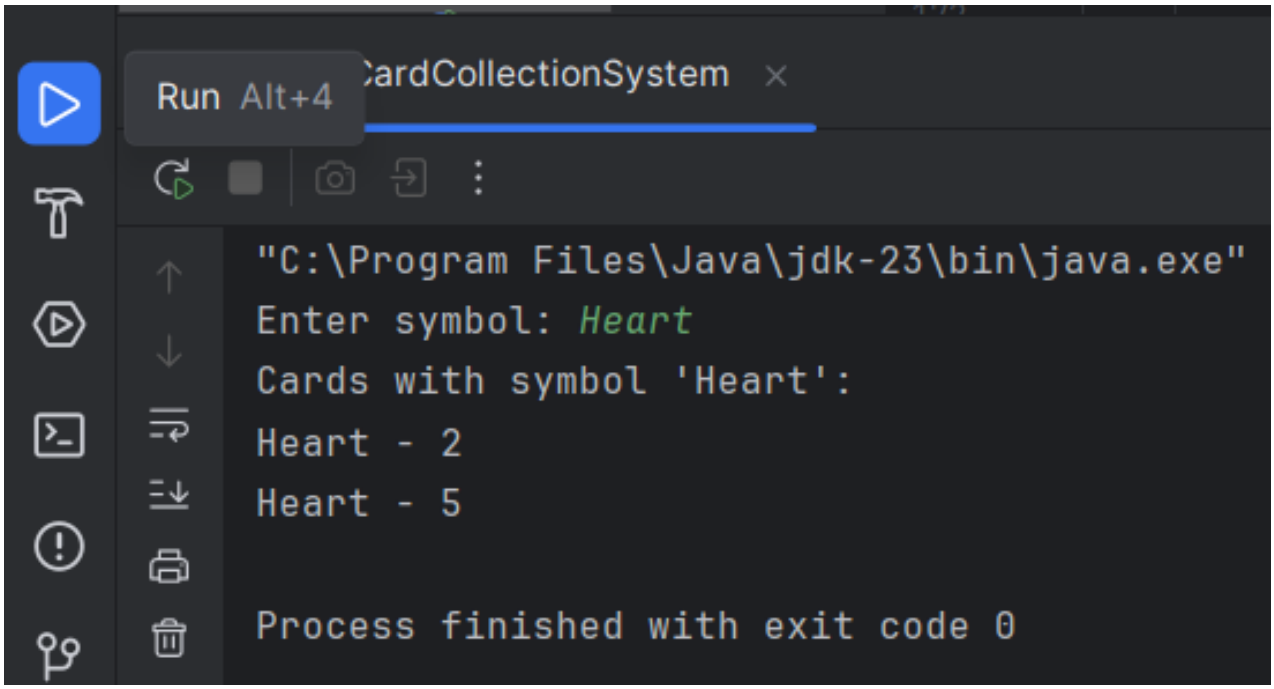
6. Code:

EXPERIMENT-4.java

```java
package PBLJ.Experiments;

import java.util.*;

class Card {  10 usages
    String symbol;  4 usages
    int number;  2 usages

    public Card(String symbol, int number) {  6 usages
        this.symbol = symbol;
        this.number = number;
    }

    @Override
    public String toString() {
        return symbol + " - " + number;
    }
}

class CardCollectionSystem {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        HashMap<String, ArrayList<Card>> cardMap = new HashMap<>();

        addCard(cardMap, new Card( symbol: "Spade",  number: 1));
        addCard(cardMap, new Card( symbol: "Spade",  number: 3));
        addCard(cardMap, new Card( symbol: "Spade",  number: 10));
        addCard(cardMap, new Card( symbol: "Heart",  number: 2));
        addCard(cardMap, new Card( symbol: "Heart",  number: 5));
        addCard(cardMap, new Card( symbol: "Diamond",  number: 7));
```

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

        if (cardMap.containsKey(searchSymbol)) {
            System.out.println("Cards with symbol '" + searchSymbol + "':");
            for (Card c : cardMap.get(searchSymbol)) {
                System.out.println(c);
            }
        } else {
            System.out.println("No cards found with symbol '" + searchSymbol + "'");
        }

        sc.close();
    }

    private static void addCard(HashMap<String, ArrayList<Card>> cardMap, Card card) {  6 usages
        cardMap.putIfAbsent(card.symbol, new ArrayList<>());
        cardMap.get(card.symbol).add(card);
    }
}
```

7. Output:



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. **Objective:** To understand multithreading, thread synchronization, and thread priorities in Java.

3. **Input/Apparatus Used:** Thread, synchronized method, setPriority(), ticket counter simulation.

4. **Procedure:**

   1. Create a TicketBooking class with synchronized bookTicket() method.
   2. Use a Thread class to simulate customers (normal and VIP).
   3. Create threads with different priorities.
   4. Start threads and observe how VIPs are handled first due to higher priority.
   5. Ensure no two threads can book the same seat using synchronized.

Sample Input:
Thread 1: Normal User - Booking Seat 1
Thread 2: VIP User - Booking Seat 1

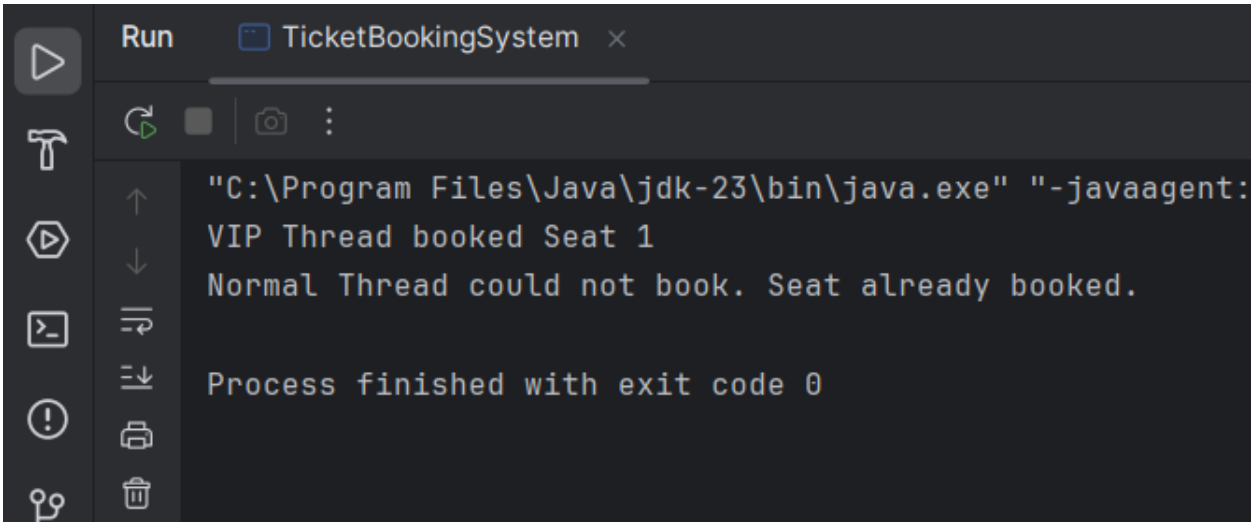
Sample Output:
VIP Thread booked Seat 1
Normal Thread could not book. Seat already booked.

5. Code:

EXPERIMENT-4.java

```java
package PBLJ.Experiments;

class TicketBooking {  4 usages
    private boolean isBooked = false;  2 usages

    public synchronized void bookTicket(String userType) {  1 usage
        if (!isBooked) {
            System.out.println(userType + " booked Seat 1");
            isBooked = true;
        } else {
            System.out.println(userType + " could not book. Seat already booked.");
        }
    }
}

class Customer extends Thread {  4 usages
    private TicketBooking ticketBooking;  2 usages
    private String userType;  2 usages

    public Customer(TicketBooking ticketBooking, String userType) {  2 usages
        this.ticketBooking = ticketBooking;
        this.userType = userType;
    }

    @Override
    public void run() {
        ticketBooking.bookTicket(userType);
    }
}

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


        Customer normalUser = new Customer(ticketBooking, userType: "Normal Thread");

        Customer vipUser = new Customer(ticketBooking, userType: "VIP Thread");

        normalUser.setPriority(Thread.MIN_PRIORITY);
        vipUser.setPriority(Thread.MAX_PRIORITY);

        normalUser.start();
        vipUser.start();
    }
}
```

6. Output:


Run
TicketBookingSystem
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:
VIP Thread booked Seat 1
Normal Thread could not book. Seat already booked.
Process finished with exit code 0