



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment-4

Student Name: Riya Mehta

Branch: B.E-C.S.E

Semester: 5th

Subject Name: PBLJ

UID: 23BCS14042

Section/Group: 23KRG-2B

Date of Performance: 18/08/2025

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

6. Code:

```

EXPERIMENT-4.java x
1 package PBLJ.Experiments;
2
3
4 import java.util.ArrayList;
5 import java.util.Scanner;
6
7 class Employee { 5 usages
8     int id; 5 usages
9     String name; 3 usages
10    double salary; 3 usages
11
12    public Employee(int id, String name, double salary) { 1 usage
13        this.id = id;
14        this.name = name;
15        this.salary = salary;
16    }
17
18    public String toString() {
19        return "ID=" + id + ", Name=" + name + ", Salary=" + salary;
20    }
21 }
22
23 class EmployeeManagementSystem {
24     public static void main(String[] args) {
25         Scanner sc = new Scanner(System.in);
26         ArrayList<Employee> employees = new ArrayList<>();
27
28         int choice;
29         do {
30             System.out.println("\n=== Employee Management System ===");
31             System.out.println("1. Add Employee");
32             System.out.println("2. Update Employee by ID");
33             System.out.println("3. Remove Employee by ID");
34             System.out.println("4. Search Employee by ID");
35             System.out.println("5. Display All Employees");
36             System.out.println("6. Exit");
37             System.out.print("Enter your choice: ");
38             choice = sc.nextInt();
39
40             switch (choice) {
41                 case 1:
42                     System.out.print("Enter ID: ");
43                     int id = sc.nextInt();
44                     sc.nextLine();
45                     System.out.print("Enter Name: ");
46                     String name = sc.nextLine();
47                     System.out.print("Enter Salary: ");
48                     double salary = sc.nextDouble();
49
50                     employees.add(new Employee(id, name, salary));
51                     System.out.println("Employee added successfully!");
52                     break;
53
54                 case 2:
55                     System.out.print("Enter Employee ID to update: ");
56                     int updateId = sc.nextInt();
57                     boolean foundUpdate = false;
58                     for (Employee emp : employees) {
59                         if (emp.id == updateId) {
60                             sc.nextLine();
61                             System.out.print("Enter new Name: ");
62                             emp.name = sc.nextLine();
63                             System.out.print("Enter new Salary: ");
64                             emp.salary = sc.nextDouble();
65                             System.out.println("Employee updated successfully!");
66                             foundUpdate = true;
67                             break;
68                         }
69                     }
70                     if (!foundUpdate) {
71                         System.out.println("Employee with ID " + updateId + " not found.");
72                     }
73                     break;

```

```
75         case 3:
76             System.out.print("Enter Employee ID to remove: ");
77             int removeId = sc.nextInt();
78             boolean removed = employees.removeIf( Employee emp -> emp.id == removeId);
79             if (removed) {
80                 System.out.println("Employee removed successfully!");
81             } else {
82                 System.out.println("Employee with ID " + removeId + " not found.");
83             }
84             break;
85
86         case 4:
87             System.out.print("Enter Employee ID to search: ");
88             int searchId = sc.nextInt();
89             boolean foundSearch = false;
90             for (Employee emp : employees) {
91                 if (emp.id == searchId) {
92                     System.out.println("Employee Found: " + emp);
93                     foundSearch = true;
94                     break;
95                 }
96             }
97             if (!foundSearch) {
98                 System.out.println("Employee with ID " + searchId + " not found.");
99             }
100             break;
101
102         case 5:
103             System.out.println("\n=== Employee List ===");
104             if (employees.isEmpty()) {
105                 System.out.println("No employees found.");
106             } else {
107                 for (Employee emp : employees) {
108                     System.out.println(emp);
109                 }
110             }
111             break;
112
113         case 6:
114             System.out.println("Exiting... Thank you!");
115             break;
116
117         default:
118             System.out.println("Invalid choice! Please try again.");
```

7. Output:

```
Run EmployeeManagementSystem x
"C:\Program Files\Java\jdk-23\bin\java.exe" "-j

=== Employee Management System ===
1. Add Employee
2. Update Employee by ID
3. Remove Employee by ID
4. Search Employee by ID
5. Display All Employees
6. Exit
Enter your choice: 1
Enter ID: 101
Enter Name: Gagnesh
Enter Salary: 600000
Employee added successfully!

=== Employee Management System ===
1. Add Employee
2. Update Employee by ID
3. Remove Employee by ID
4. Search Employee by ID
5. Display All Employees
6. Exit
Enter your choice: 2
Enter Employee ID to update: 101
Enter new Name: Gagnesh Kakkar
Enter new Salary: 1500000
Employee updated successfully!

=== Employee Management System ===
1. Add Employee
2. Update Employee by ID
3. Remove Employee by ID
4. Search Employee by ID
5. Display All Employees
6. Exit
Enter your choice: 4
Enter Employee ID to search: 101
Employee Found: ID=101, Name=Gagnesh Kakkar, Salary=1500000.0

=== Employee Management System ===
1. Add Employee
2. Update Employee by ID
3. Remove Employee by ID
4. Search Employee by ID
5. Display All Employees
6. Exit
Enter your choice: 6
Exiting... Thank you!

Process finished with exit code 0
```

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

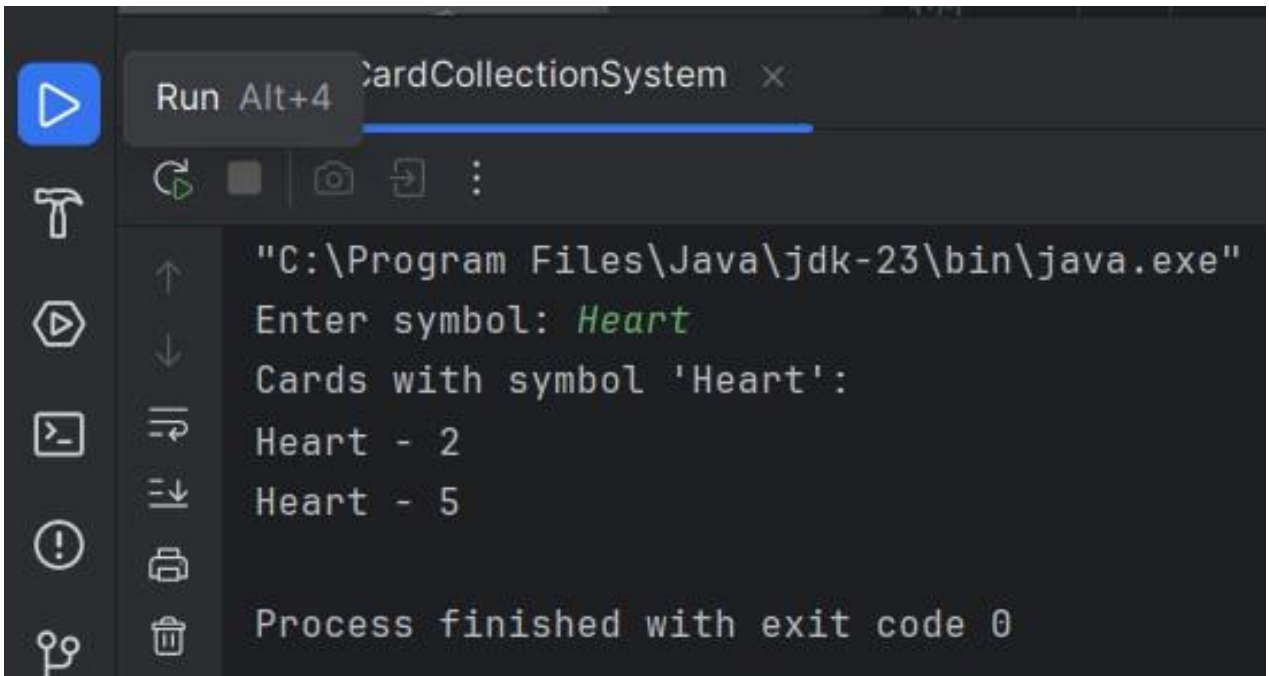
Spade - 10

6. Code:

```
EXPERIMENT-4.java x
1 package PBLJ.Experiments;
2
3 import java.util.*;
4
5 class Card { 10 usages
6     String symbol; 4 usages
7     int number; 2 usages
8
9     public Card(String symbol, int number) { 6 usages
10         this.symbol = symbol;
11         this.number = number;
12     }
13
14     @Override
15     public String toString() {
16         return symbol + " - " + number;
17     }
18 }
19
20 class CardCollectionSystem {
21     public static void main(String[] args) {
22         Scanner sc = new Scanner(System.in);
23
24         HashMap<String, ArrayList<Card>> cardMap = new HashMap<>();
25
26         addCard(cardMap, new Card( symbol: "Spade", number: 1));
27         addCard(cardMap, new Card( symbol: "Spade", number: 3));
28         addCard(cardMap, new Card( symbol: "Spade", number: 10));
29         addCard(cardMap, new Card( symbol: "Heart", number: 2));
30         addCard(cardMap, new Card( symbol: "Heart", number: 5));
31         addCard(cardMap, new Card( symbol: "Diamond", number: 7));
32     }
33 }
```

```
33     System.out.print("Enter symbol: ");
34     String searchSymbol = sc.nextLine();
35
36     if (cardMap.containsKey(searchSymbol)) {
37         System.out.println("Cards with symbol '" + searchSymbol + "':");
38         for (Card c : cardMap.get(searchSymbol)) {
39             System.out.println(c);
40         }
41     } else {
42         System.out.println("No cards found with symbol '" + searchSymbol + "');");
43     }
44
45     sc.close();
46 }
47
48 @private static void addCard(HashMap<String, ArrayList<Card>> cardMap, Card card) { 6 usages
49     cardMap.putIfAbsent(card.symbol, new ArrayList<>());
50     cardMap.get(card.symbol).add(card);
51 }
52 }
```

7. Output:



```
Run Alt+4 CardCollectionSystem x
"C:\Program Files\Java\jdk-23\bin\java.exe"
Enter symbol: Heart
Cards with symbol 'Heart':
Heart - 2
Heart - 5
Process finished with exit code 0
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

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:

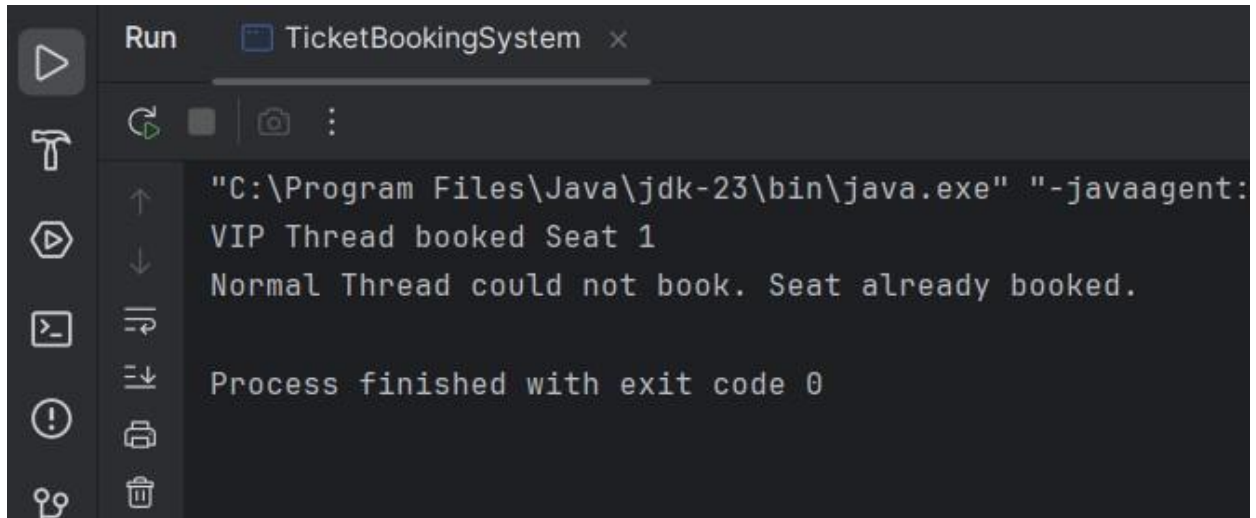
VIP Thread booked Seat 1

Normal Thread could not book. Seat already booked.

5. Code:

```
EXPERIMENT-4.java x
1  package PBLJ.Experiments;|
2
3  class TicketBooking { 4 usages
4      private boolean isBooked = false; 2 usages
5
6      public synchronized void bookTicket(String userType) { 1 usage
7          if (!isBooked) {
8              System.out.println(userType + " booked Seat 1");
9              isBooked = true;
10         } else {
11             System.out.println(userType + " could not book. Seat already booked.");
12         }
13     }
14 }
15
16 class Customer extends Thread { 4 usages
17     private TicketBooking ticketBooking; 2 usages
18     private String userType; 2 usages
19
20     public Customer(TicketBooking ticketBooking, String userType) { 2 usages
21         this.ticketBooking = ticketBooking;
22         this.userType = userType;
23     }
24
25     @Override
26     public void run() {
27         ticketBooking.bookTicket(userType);
28     }
29 }
30
31 class TicketBookingSystem {
32     public static void main(String[] args) {
33         TicketBooking ticketBooking = new TicketBooking();
34
35
36         Customer normalUser = new Customer(ticketBooking, userType: "Normal Thread");
37
38         Customer vipUser = new Customer(ticketBooking, userType: "VIP Thread");
39
40         normalUser.setPriority(Thread.MIN_PRIORITY);
41         vipUser.setPriority(Thread.MAX_PRIORITY);
42
43         normalUser.start();
44         vipUser.start();
45     }
46 }
```


6. Output:



```
Run TicketBookingSystem x
"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
```