

# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## Experiment-4

**Name:** Aarukh

**Branch:** BE-CSE

**Semester:** 6<sup>th</sup>

**Subject Name:** Project Based Learning in  
**Java**

**UID:**22BCS50104

**Section/Group:** 643/B

**D.Performance:**27/02/2025

**Subject Code:** 22CSH-359

**1. Aim :** Develop Java programs using core concepts such as data structures, collections, and multithreading to manage and manipulate data.

**2. Easy Level:**

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.

**3. Implementation/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;
    }

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

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

    switch (choice) {
        case 1 -> addEmployee();
        case 2 -> updateEmployee();
        case 3 -> removeEmployee();
        case 4 -> searchEmployee();
        case 5 -> displayEmployees();
        case 6 -> {
            System.out.println("Exiting...");
            return;
        }
        default -> System.out.println("Invalid choice! Try again.");
    }
}

private static void addEmployee() {
    System.out.print("Enter ID: ");
    int id = scanner.nextInt();
    scanner.nextLine(); // Consume newline
    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.");
}

private static void updateEmployee() {
    System.out.print("Enter Employee ID to Update: ");
    int id = scanner.nextInt();
    for (Employee e : employees) {
        if (e.id == id) {
            scanner.nextLine(); // Consume newline
            System.out.print("Enter New Name: ");
            e.name = scanner.nextLine();
            System.out.print("Enter New Salary: ");
            e.salary = scanner.nextDouble();
        }
    }
}
```

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

private static void removeEmployee() {
    System.out.print("Enter Employee ID to Remove: ");
    int id = scanner.nextInt();
    employees.removeIf(e -> e.id == id);
    System.out.println("Employee removed successfully.");
}

private static void searchEmployee() {
    System.out.print("Enter Employee ID to Search: ");
    int id = scanner.nextInt();
    for (Employee e : employees) {
        if (e.id == id) {
            System.out.println(e);
            return;
        }
    }
    System.out.println("Employee not found!");
}

private static void displayEmployees() {
    if (employees.isEmpty()) {
        System.out.println("No employees to display.");
    } else {
        employees.forEach(System.out::println);
    }
}
}
```

## 4. OUTPUT:

```
input
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All
6. Exit
Choose an option: 1
Enter ID: 101
Enter Name: Aarukh khan
Enter Salary: 500000
Employee added successfully.

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All
6. Exit
Choose an option: 4
Enter Employee ID to Search: 101
ID: 101, Name: Aarukh khan, Salary: 500000.0

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

**Output for Easy Level:**

## 5. Medium Level:

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.

## 6. Code:

```
import java.util.*;

class CardCollection {

    private static Map<String, List<String>> cardMap = new HashMap<>();

    public static void main(String[] args) {
        addCard("SPADE", "Ace of Spades");
        addCard("SPADE", "King of Spades");
        addCard("HEART", "Queen of Hearts");
        addCard("DIAMOND", "Jack of Diamonds");
        addCard("CLUB", "10 of Clubs");

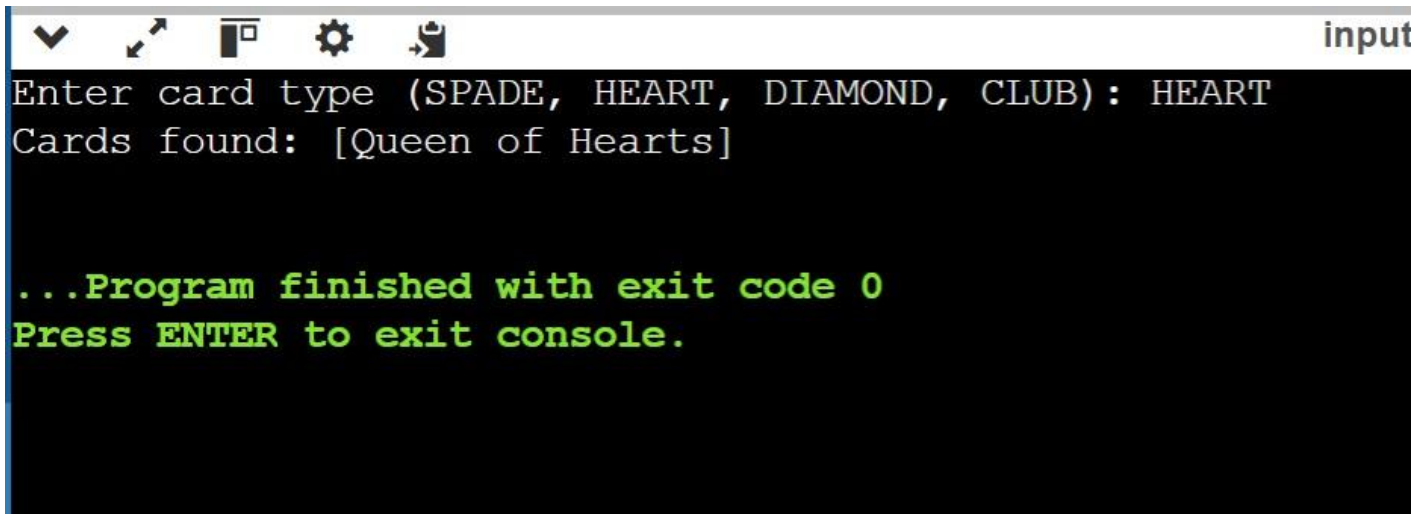
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter card type (SPADE, HEART, DIAMOND, CLUB): ");
        String type = scanner.next().toUpperCase(); // Convert input to uppercase

        List<String> cards = cardMap.get(type);
        if (cards != null) {
            System.out.println("Cards found: " + cards);
        } else {
            System.out.println("No cards found for this type.");
        }
        scanner.close();
    }

    private static void addCard(String type, String card) {
        cardMap.computeIfAbsent(type, k -> new ArrayList<>()).add(card);
    }
}
```

}

## 7. Output:



```
Enter card type (SPADE, HEART, DIAMOND, CLUB): HEART
Cards found: [Queen of Hearts]

...Program finished with exit code 0
Press ENTER to exit console.
```

## 8. Hard Level:

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.

## 9. Code:-

```
import java.util.concurrent.locks.ReentrantLock;

class TicketBookingSystem {
    private int availableSeats = 5;
    private final ReentrantLock lock = new ReentrantLock(true); // Fair lock

    public void bookTicket(String passenger, int seats) {
        lock.lock();
        try {
            if (seats <= availableSeats) {
                System.out.println(passenger + " successfully booked " + seats + " seat(s).");
                availableSeats -= seats;
            }
        }
    }
}
```

```
        } else {  
            System.out.println(passenger + " failed to book. Not enough seats.");  
        }  
    } finally {  
        lock.unlock();  
    }  
}  
}
```

```
class Passenger extends Thread {  
    private TicketBookingSystem system;  
    private String passengerName;  
    private int seats;  
  
    public Passenger(TicketBookingSystem system, String passengerName, int seats, int priority) {  
        this.system = system;  
        this.passengerName = passengerName;  
        this.seats = seats;  
        this.setPriority(priority);  
    }  
  
    public void run() {  
        system.bookTicket(passengerName, seats);  
    }  
}
```

```
public class TicketBookingApp {  
    public static void main(String[] args) {  
        TicketBookingSystem system = new TicketBookingSystem();  
  
        Passenger p1 = new Passenger(system, "VIP-1", 2, Thread.MAX_PRIORITY);  
        Passenger p2 = new Passenger(system, "User-1", 1, Thread.NORM_PRIORITY);
```

```
Passenger p3 = new Passenger(system, "VIP-2", 1, Thread.MAX_PRIORITY);
Passenger p4 = new Passenger(system, "User-2", 2, Thread.NORM_PRIORITY);
Passenger p5 = new Passenger(system, "User-3", 1, Thread.NORM_PRIORITY);

p1.start();
p2.start();
p3.start();
p4.start();
p5.start();
}
}
```

## 10. Output:-

```
VIP-1 successfully booked 2 seat(s).
User-1 successfully booked 1 seat(s).
VIP-2 successfully booked 1 seat(s).
User-2 failed to book. Not enough seats.
User-3 successfully booked 1 seat(s).
```

```
...Program finished with exit code 0
Press ENTER to exit console.
```

## 11. Learning outcomes:

- 1. Collections in Java:** Learn ArrayList, HashMap, and Collection interfaces for efficient data storage and retrieval.
- 2. CRUD Operations:** Implement basic operations like Add, Update, Remove, and Search using Java collections.
- 3. Multithreading & Synchronization:** Use synchronized and ReentrantLock to handle concurrent access and prevent race conditions.
- 4. Thread Priorities:** Assign priorities (MAX\_PRIORITY, NORM\_PRIORITY) to ensure important tasks (e.g., VIP bookings) execute first.