Experiment 2.2

Student Name: Abhiroop Singh
UID: 22BCS10352
Branch: CSE
Section/Group: Kpit-902/B
Semester: 6
Date of Performance: 28/02/25
Subject Name: PBLJ
Subject Code: 22CSH-359

1. **Aim:** Develop Java programs using lambda expressions and stream operations for sorting, filtering, and processing large datasets efficiently.
   a Write a program to sort a list of Employee objects (name, age, salary) using lambda expressions.
   b Create a program to use lambda expressions and stream operations to filter students scoring above 75%, sort them by marks, and display their names.

   Write a Java program to process a large dataset of products using streams. Perform operations such as grouping products by category, finding the most expensive product in each category, and calculating the average price of all products.

2. **Objective:** To build and Develop Java programs using lambda expressions and stream operations for sorting, filtering, and processing large datasets efficiently.

3. **Implementation/Code:**
   1. Write a program to sort a list of Employee objects (name, age, salary) using lambda expressions:

```
import java.util.*;

class Worker {
    String fullName;
    int years;
    double earnings;
```

```
    public Worker(String fullName, int years, double earnings) {
        this.fullName = fullName;
        this.years = years;
        this.earnings = earnings;
    }

    @Override
    public String toString() {
        return "Worker{FullName='" + fullName + "', Years=" + years + ",
Earnings=" + earnings + "}";
    }
}

public class RefactoredEmployeeSorting {
    public static void main(String[] args) {
        List<Worker> staff = Arrays.asList(
                new Worker("Alice", 29, 45000),
                new Worker("Bob", 32, 52000),
                new Worker("Charlie", 26, 60000),
                new Worker("Diana", 38, 48000)
        );

        // Sorting by full name
        staff.sort(Comparator.comparing(worker -> worker.fullName));
        System.out.println("Sorted by Full Name:");
        staff.forEach(System.out::println);

        // Sorting by years
```

Name: Akshat dua
Uid: 22bcs10591

```
        staff.sort(Comparator.comparingInt(worker -> worker.years));
        System.out.println("\nSorted by Years:");
        staff.forEach(System.out::println);

        // Sorting by earnings
        staff.sort(Comparator.comparingDouble(worker -> worker.earnings));
        System.out.println("\nSorted by Earnings:");
        staff.forEach(System.out::println);
    }
}
```

2. Create a program to use lambda expressions and stream operations to filter students scoring above 75%, sort them by marks, and display their names.

```
import java.util.*;
import java.util.stream.Collectors;

class Learner {
    String fullName;
    double grade;

    public Learner(String fullName, double grade) {
        this.fullName = fullName;
        this.grade = grade;
    }

    @Override
    public String toString() {
        return fullName + " - " + grade;
    }
}

public class RefactoredStudentFiltering {
```

```
    public static void main(String[] args) {
        List<Learner> learners = Arrays.asList(
                new Learner("John", 78),
                new Learner("Mia", 65),
                new Learner("Leo", 90),
                new Learner("Zara", 82),
                new Learner("Ella", 95)
        );

        // Filtering learners scoring above 75% and sorting by grade (descending)
        List<Learner> highAchievers = learners.stream()
                .filter(l -> l.grade > 75)
                .sorted((l1, l2) -> Double.compare(l2.grade, l1.grade))
                .collect(Collectors.toList());

        System.out.println("High Achievers (Above 75% and Sorted by
Grades):");
        highAchievers.forEach(System.out::println);
    }
}
```

3. **Wr**ite a Java program to process a large dataset of products using streams. Perform operations such as grouping products by category, finding the most expensive product in each category, and calculating the average price of all products

```
import java.util.Scanner;
import java.util.concurrent.locks.*;

class BookingSystem {
    private int availableSeats;
    private final Lock bookingLock = new ReentrantLock();
```

Name: Akshat dua
Uid: 22bcs10591

```
    public BookingSystem(int totalSeats) {
        this.availableSeats = totalSeats;
    }

    public void bookSeats(String customerName, int seatsRequested) {
        bookingLock.lock();
        try {
            if (availableSeats >= seatsRequested) {
                System.out.println(customerName + " booked " + seatsRequested +
" seat(s). Remaining seats: " + (availableSeats - seatsRequested));
                availableSeats -= seatsRequested;
            } else {
                System.out.println("Sorry, " + customerName + "! Only " +
availableSeats + " seat(s) left.");
            }
        } finally {
            bookingLock.unlock();
        }
    }
}

class Customer extends Thread {
    private BookingSystem bookingSystem;
    private String customerName;
    private int seatsNeeded;

    public Customer(BookingSystem system, String name, int seats, int priority)
{
        this.bookingSystem = system;
        this.customerName = name;
        this.seatsNeeded = seats;
        setPriority(priority);
    }
```

Name: Akshat dua
Uid: 22bcs10591

```
    @Override
    public void run() {
        bookingSystem.bookSeats(customerName, seatsNeeded);
    }
}

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

        System.out.print("Total available seats: ");
        int totalSeats = scanner.nextInt();
        BookingSystem bookingSystem = new BookingSystem(totalSeats);

        System.out.print("Number of customers: ");
        int customerCount = scanner.nextInt();
        scanner.nextLine();

        Customer[] customers = new Customer[customerCount];

        for (int i = 0; i < customerCount; i++) {
            System.out.print("Customer name: ");
            String name = scanner.nextLine();

            System.out.print("Seats needed: ");
            int seats = scanner.nextInt();

            System.out.print("Priority (1-High, 2-Normal): ");
            int priority = scanner.nextInt();
            scanner.nextLine();
```

Name: Akshat dua
Uid: 22bcs10591

```
            int threadPriority = (priority == 1) ? Thread.MAX_PRIORITY :
Thread.NORM_PRIORITY;
            customers[i] = new Customer(bookingSystem, name, seats,
threadPriority);
        }

        for (Customer customer : customers) {
            customer.start();
        }

        scanner.close();
    }
}
```

4. Output:

```
Sorted by Full Name:
Worker{FullName='Alice', Years=29, Earnings=45000.0}
Worker{FullName='Bob', Years=32, Earnings=52000.0}
Worker{FullName='Charlie', Years=26, Earnings=60000.0}
Worker{FullName='Diana', Years=38, Earnings=48000.0}

Sorted by Years:
Worker{FullName='Charlie', Years=26, Earnings=60000.0}
Worker{FullName='Alice', Years=29, Earnings=45000.0}
Worker{FullName='Bob', Years=32, Earnings=52000.0}
Worker{FullName='Diana', Years=38, Earnings=48000.0}

Sorted by Earnings:
Worker{FullName='Alice', Years=29, Earnings=45000.0}
Worker{FullName='Diana', Years=38, Earnings=48000.0}
Worker{FullName='Bob', Years=32, Earnings=52000.0}
Worker{FullName='Charlie', Years=26, Earnings=60000.0}
```

Name: Akshat dua
Uid: 22bcs10591

```
High Achievers (Above 75% and Sorted by Grades):
Ella - 95.0
Leo - 90.0
Zara - 82.0
John - 78.0
```

```
Total available seats: 10
Number of customers: 3
Customer name: Alice
Seats needed: 4
Priority (1-High, 2-Normal): 1
Customer name: Bob
Seats needed: 6
Priority (1-High, 2-Normal): 2
Customer name: Charlie
Seats needed: 2
Priority (1-High, 2-Normal): 1
```

```
Alice booked 4 seat(s). Remaining seats: 6
Bob booked 6 seat(s). Remaining seats: 0
Sorry, Charlie! Only 0 seat(s) left.
```

5. Learning Outcome:

- **Concurrency Control:** Understand how to use locks (ReentrantLock) to manage concurrent seat bookings safely.
- **Thread Priority:** Learn how thread priority (Thread.MAX_PRIORITY, Thread.NORM_PRIORITY) influences execution order.
- **Synchronization:** Gain insights into handling shared resources (availableSeats) in a multi-threaded environment.
- **Dynamic Input Handling:** Develop skills to create interactive console applications using Scanner for user inputs.

Name: Akshat dua
Uid: 22bcs10591