



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## Experiment 6

**Student Name:** Akshara Chauhan

**Branch:** CSE

**Semester:** 5<sup>th</sup>

**Subject Name:** PBLJ

**UID:** 23BCS11410

**Section/Group:** KRG\_2B

**Date of Performance:** 30/09/25

**Subject Code:** 23CSH-304

### 1. Aim:

To design and implement Java programs using lambda expressions and Stream API for efficient data sorting, filtering, and processing.

- To apply functional programming concepts, lambda expressions, and stream operations for simplifying data manipulation and improving performance.

#### **Part A – Easy Level:**

- To create a Java program that sorts a list of Employee objects (name, age, salary) using lambda expressions.
- To demonstrate how lambda expressions simplify sorting logic and enhance code readability.

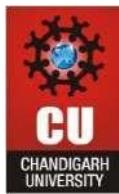
#### **Part B – Medium Level:**

- To develop a Java program that uses Stream API to filter students scoring above 75%, sort them by marks, and display their names.
- To apply filter(), sorted(), map(), and collect() operations for functional-style data processing.

#### **Part C – Hard Level:**

- To design a Java program that processes a large dataset of Product objects using advanced stream operations.
- To perform grouping of products by category, find the most expensive product in each category, and calculate the average price using Collectors.groupingBy(), maxBy(), and averagingDouble().

### 2. Objective:



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

- ✓ To understand and implement **lambda expressions** for concise and readable code.
- ✓ To use **Comparator** and **lambda syntax** for sorting based on multiple fields.
- ✓ To apply **Stream API** for filtering, mapping, and sorting data efficiently.
- ✓ To demonstrate **real-world use cases** of functional programming in Java.

## 3. JAVA script and output:

### EASY-LEVEL PROBLEM

```
package exp.pkg5;
```

```
import java.io.*;
```

```
import java.util.*;
```

```
class Employee {
```

```
    String name;
```

```
    int age;
```

```
    double salary;
```

```
Employee(String name, int age, double salary) {
```

```
    this.name = name;
```

```
    this.age = age;
```

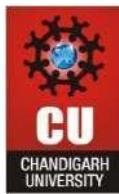
```
    this.salary = salary;
```

```
}
```

```
@Override
```

```
public String toString() {
```

```
    return name + " - " + age + " - " + salary;
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
}
```

```
}
```

```
public class Exp6 {
```

```
    public static void main(String[] args) {
```

```
        List<Employee> employees = new ArrayList<>();
```

```
        employees.add(new Employee("Akshara", 30, 50000));
```

```
        employees.add(new Employee("Ishika", 25, 60000));
```

```
        employees.add(new Employee("Varun", 28, 75000));
```

```
        employees.sort((e1, e2) -> Double.compare(e1.salary, e2.salary));
```

```
        System.out.println("Sorted by Salary:");
```

```
        employees.forEach(System.out::println);
```

```
}
```

```
}
```

## OUTPUT:

```
run:
```

```
Sorted by Salary:
```

```
Akshara - 30 - 50000.0
```

```
Ishika - 25 - 60000.0
```

```
Varun - 28 - 75000.0
```

```
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 1: Easy Level



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## **MEDIUM LEVEL PROBLEM:**

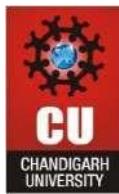
```
package exp.pkg5;

import java.io.*;
import java.util.*;
class Student {
    String name;
    int id;
    double marks;

    Student(String name, int id, double marks) {
        this.name = name;
        this.id = id;
        this.marks = marks;
    }
}

public class Exp6_Medium {
    public static void main(String[] args) {
        List<Student> students = Arrays.asList(
            new Student("Ravi", 101, 85.0),
            new Student("Akshara", 102, 92.0),
            new Student("Ishika", 103, 78.0),
            new Student("Nikhil", 104, 65.0)
        );
        System.out.println("Students scoring above 75%:");
        students.stream()
            .filter(s -> s.marks > 75)
            .sorted((s1, s2) -> Double.compare(s2.marks, s1.marks))
            .map(s -> s.name)
            .forEach(System.out::println);
    }
}
```

## **OUTPUT:**



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
Students scoring above 75%:  
Akshara  
Ravi  
Ishika  
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 2: Medium Level

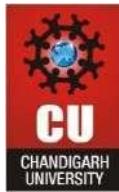
## HARD LEVEL PROBLEM

```
package exp.pkg5;
```

```
import java.io.*;  
import java.util.*;  
import java.util.stream.*;  
import java.util.Comparator;  
import java.util.Map;  
import java.util.Optional;
```

```
class Product {  
    int id;  
    String name;  
    double price;  
    String category;
```

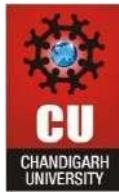
```
Product(int id, String name, double price, String category) {  
    this.id = id;
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
this.name = name;  
this.price = price;  
this.category = category;  
}  
}  
  
public class Exp6_Hard {  
    public static void main(String[] args) {  
        List<Product> products = Arrays.asList(  
            new Product(1, "Laptop", 80000, "Electronics"),  
            new Product(2, "Mobile", 30000, "Electronics"),  
            new Product(3, "Office Chair", 12000, "Furniture"),  
            new Product(4, "Table", 9000, "Furniture"),  
            new Product(5, "Earphones", 2000, "Electronics")  
        );  
  
        Map<String, List<Product>> grouped = products.stream()  
            .collect(Collectors.groupingBy(p -> p.category));  
  
        Map<String, Optional<Product>> maxPriceByCategory = products.stream()  
            .collect(Collectors.groupingBy(p -> p.category,  
                Collectors.maxBy(Comparator.comparingDouble(p -> p.price))));
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
double avgPrice = products.stream()
    .collect(Collectors.averagingDouble(p -> p.price));

System.out.println("Most Expensive Products by Category:");
maxPriceByCategory.forEach((category, product) ->
    System.out.println(category + " -> Most Expensive: " +
        product.get().name + " (Rs " + product.get().price + ")"));

System.out.println("\nAverage Price of All Products: Rs " + avgPrice);
}
```

## OUTPUT:

```
Most Expensive Products by Category:
Electronics -> Most Expensive: Laptop (Rs 80000.0)
Furniture -> Most Expensive: Office Chair (Rs 12000.0)

Average Price of All Products: Rs 26600.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 1: Hard Level