



## Experiment 6

**Student Name:** Khushi Kumari

**UID:** 22BCS14714

**Branch:** CSE

**Section/Group:** 640/B

**Semester:** 6th

**Date of Performance:** 12/03/25

**Subject:** Project Based Learning with JAVA

**Subject Code:** 22CSH-359

### 1. Aim:

**Easy Level:** Write a program to sort a list of Employee objects (name, age, salary) using lambda expressions.

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

**Hard Level:** 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. Implementation/Code:

#### Easy Level:

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

```
class Employee {  
    String name;  
    int age;  
    double salary;  
  
    public Employee(String name, int age, double salary) {  
        this.name = name;  
        this.age = age;  
        this.salary = salary;  
    }  
}
```

```
public String toString() {  
    return "Name: " + name + ", Age: " + age + ", Salary: " + salary;  
}  
}  
  
public class EmployeeSort {  
    public static void main(String[] args) {  
        List<Employee> employees = new ArrayList<>();  
        employees.add(new Employee("John", 30, 50000));  
        employees.add(new Employee("Alice", 25, 60000));  
        employees.add(new Employee("Bob", 35, 45000));  
  
        System.out.println("Sorted by Name:");  
        List<Employee> sortedByName = employees.stream()  
            .sorted((e1, e2) -> e1.name.compareTo(e2.name))  
            .collect(Collectors.toList());  
        sortedByName.forEach(System.out::println);  
  
        System.out.println("\nSorted by Age:");  
        List<Employee> sortedByAge = employees.stream()  
            .sorted((e1, e2) -> Integer.compare(e1.age, e2.age))  
            .collect(Collectors.toList());  
        sortedByAge.forEach(System.out::println);  
  
        System.out.println("\nSorted by Salary:");  
        List<Employee> sortedBySalary = employees.stream()  
            .sorted((e1, e2) -> Double.compare(e1.salary, e2.salary))  
            .collect(Collectors.toList());  
        sortedBySalary.forEach(System.out::println);  
    }  
}
```

**OUTPUT :**

```
Sorted by Name:
Name: Aditi, Age: 25, Salary: 6000.0
Name: Shreya, Age: 30, Salary: 5000.0
Name: ansh, Age: 35, Salary: 45000.0

Sorted by Age:
Name: Aditi, Age: 25, Salary: 6000.0
Name: Shreya, Age: 30, Salary: 5000.0
Name: ansh, Age: 35, Salary: 45000.0

Sorted by Salary:
Name: Shreya, Age: 30, Salary: 5000.0
Name: Aditi, Age: 25, Salary: 6000.0
Name: ansh, Age: 35, Salary: 45000.0
```

## Medium Level:

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

class Student {
    String name;
    double percentage;

    public Student(String name, double percentage) {
        this.name = name;
        this.percentage = percentage;
    }

    public String toString() {
        return "Name: " + name + ", Percentage: " + percentage;
    }
}

public class StudentFilterSort {
    public static void main(String[] args) {
        List<Student> students = new ArrayList<>();
        students.add(new Student("Shreya", 92.5));
        students.add(new Student("Aditi", 85.0));
        students.add(new Student("Ansh", 90.0));
        students.add(new Student("Raju", 78.5));
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
System.out.println("Students scoring above 75%, sorted by marks:");

students.stream()
    .filter(student -> student.percentage > 75)
    .sorted((s1, s2) -> Double.compare(s2.percentage, s1.percentage))
    .map(student -> student.name)
    .forEach(System.out::println);
}
```

## **OUTPUT:**

```
Students scoring above 75%, sorted by marks:
Shreya
Ansh
Aditi
Raju
```

## **Hard Level:**

```
import java.util.*;

import java.util.stream.Collectors;

class Product {
    String name;
    String category;
    double price;

    public Product(String name, String category, double price) {
        this.name = name;
        this.category = category;
        this.price = price;
    }
}
```

```
public String toString() {  
    return "Name: " + name + ", Category: " + category + ", Price: " + price;  
}  
}
```

```
public class ProductProcessor {  
    public static void main(String[] args) {  
        List<Product> products = new ArrayList<>();  
        products.add(new Product("Laptop", "Electronics", 999.99));  
        products.add(new Product("Phone", "Electronics", 599.99));  
        products.add(new Product("Shirt", "Clothing", 29.99));  
        products.add(new Product("Jacket", "Clothing", 89.99));  
        products.add(new Product("Book", "Stationery", 15.99));  
        products.add(new Product("Pen", "Stationery", 2.99));  
  
        System.out.println("Products grouped by category:");  
        Map<String, List<Product>> byCategory = products.stream()  
            .collect(Collectors.groupingBy(product -> product.category));  
  
        byCategory.forEach((category, productList) -> {  
            System.out.println(category + ":");  
            productList.forEach(System.out::println);  
        });  
  
        System.out.println("\nMost expensive product in each category:");
```



```
Map<String, Optional<Product>> mostExpensive = products.stream()
    .collect(Collectors.groupingBy(
        product -> product.category,
        Collectors.maxBy((p1, p2) -> Double.compare(p1.price, p2.price))
    ));
```

```
mostExpensive.forEach((category, product) ->
    System.out.println(category + ": " + product.get()));
```

```
double averagePrice = products.stream()
    .mapToDouble(product -> product.price)
    .average()
    .orElse(0.0);
```

```
System.out.println("\nAverage price of all products: $" + String.format("%.2f", averagePrice));
```

```
}
```

```
}
```

**Output:**

```
Products grouped by category:
```

```
Clothing:
```

```
Name: Shirt, Category: Clothing, Price: 29.99
```

```
Name: Jacket, Category: Clothing, Price: 89.99
```

```
Electronics:
```

```
Name: Laptop, Category: Electronics, Price: 999.99
```

```
Name: Phone, Category: Electronics, Price: 599.99
```

```
Stationery:
```

```
Name: Book, Category: Stationery, Price: 15.99
```

```
Name: Pen, Category: Stationery, Price: 2.99
```

```
Most expensive product in each category:
```

```
Clothing: Name: Jacket, Category: Clothing, Price: 89.99
```

```
Electronics: Name: Laptop, Category: Electronics, Price: 999.99
```

```
Stationery: Name: Book, Category: Stationery, Price: 15.99
```

```
Average price of all products: $289.82
```

### 3. Learning Outcome:

- Understand how to sort by different data types (String, int, double) using lambda expressions
- Gain knowledge of the sorted() method in streams
- Understand how to handle more complex data processing tasks with streams
- Understand how to replace traditional loops with stream-based operations
- Using how lambda expression works in java.