# Experiment 6

**Student Name: Sambhav** **UID: 22BCS11742**
**Branch: CSE** **Section/Group: 640-A**
**Semester: 6** **Date of Performance:03/03/25**
**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.

2. **Objective:**
   - Develop Java programs using lambda expressions and stream operations for sorting, filtering, and processing large datasets efficiently.
   - Implement easy, medium, and hard-level tasks involving sorting employees, filtering and sorting students, and processing products using streams.

3. **Implementation/Code:**

**a.**
```
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: " + age + ", Salary: " + salary;
   }  }
public class EmployeeSort {     public static void
main(String[] args) {         List<Employee> employees
= Arrays.asList(            new
Employee("Ayush", 20, 90000),            new
Employee("Vinay", 22, 100000),

            new Employee("Prakul", 23, 70000)
        );
        employees.sort(Comparator.comparing(emp  ->  emp.name));
System.out.println("Sorted   by   Name:   "   +   employees);
employees.sort(Comparator.comparingInt(emp    ->    emp.age));
System.out.println("Sorted   by   Age:   "   +   employees);
employees.sort(Comparator.comparingDouble(emp -> emp.salary));
        System.out.println("Sorted by Salary: " + employees);
    }
}
```

**b.** import java.util.*; import

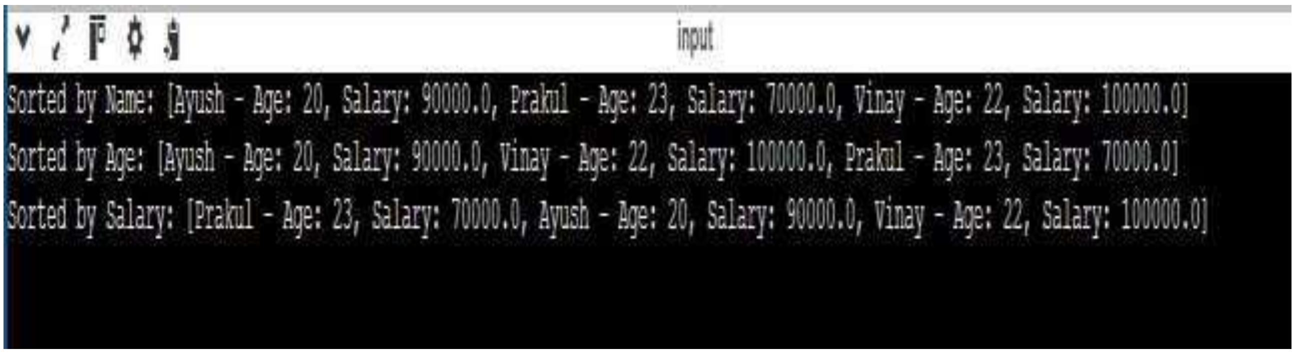
```
java.util.stream.Collectors; class Student {     private
String name;     private double marks;     public
Student(String name, double marks) {
this.name = name;         this.marks = marks;
    }
    public String getName() {
return name;
    }
    public double getMarks() {
return marks;
    }  }
public class StudentFilter {     public
static void main(String[] args) {
List<Student> students = List.of(            new
Student("Ayush", 85),           new
Student("Rajeev", 70),           new
Student("Vinay", 90),           new
Student("David", 60),           new Student("Prakul",
80)
        );

        List<String> topStudents = students.stream()
            .filter(s -> s.getMarks() > 75)
            .sorted(Comparator.comparingDouble(Student::getMarks).reversed())
            .map(Student::getName)
            .collect(Collectors.toList());
        System.out.println("Top Students: " + topStudents);
    }
}
```

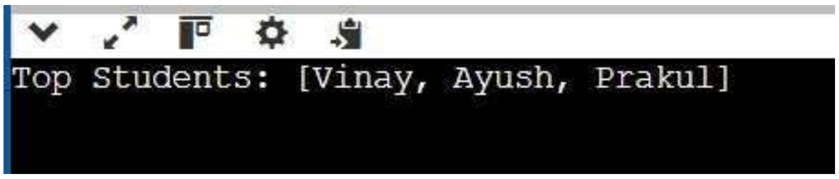
**c.** import java.util.*; import java.util.stream.Collectors; class

```
Product {     String name;     String category;     double price;
public Product(String name, String category, double price) {
this.name = name;        this.category = category;        this.price
= price;
    }
    @Override     public String toString()
{         return name + " ($"
+ price + ")";
    }  }
public class ProductProcessor {     public static void
main(String[] args) {         List<Product> products =
Arrays.asList(            new Product("Laptop",
"Electronics", 1200),            new Product("Phone",
"Electronics", 800),            new Product("TV",
"Electronics", 1500),           new Product("Shirt",
"Clothing", 50),           new Product("Jeans",
"Clothing", 70),           new Product("Blender",
"Appliances", 200),           new Product("Toaster",
"Appliances", 100)
        );
        Map<String, List<Product>> productsByCategory = products.stream()
.collect(Collectors.groupingBy(p -> p.category));        System.out.println("Products grouped by
category:");        productsByCategory.forEach((category, productList) ->
            System.out.println(category + ": " + productList));
        Map<String, Optional<Product>> mostExpensiveByCategory = products.stream()
            .collect(Collectors.groupingBy(                p
-> p.category,
                Collectors.maxBy(Comparator.comparingDouble(p -> p.price))
            ));
        System.out.println("\nMost expensive product in each category:");
mostExpensiveByCategory.forEach((category, product) ->
System.out.println(category + ": " + product.orElse(null)));        double averagePrice
= products.stream()
            .mapToDouble(p -> p.price)
            .average()
            .orElse(0);
        System.out.println("\nAverage price of all products: $" + averagePrice);
    }
}
```

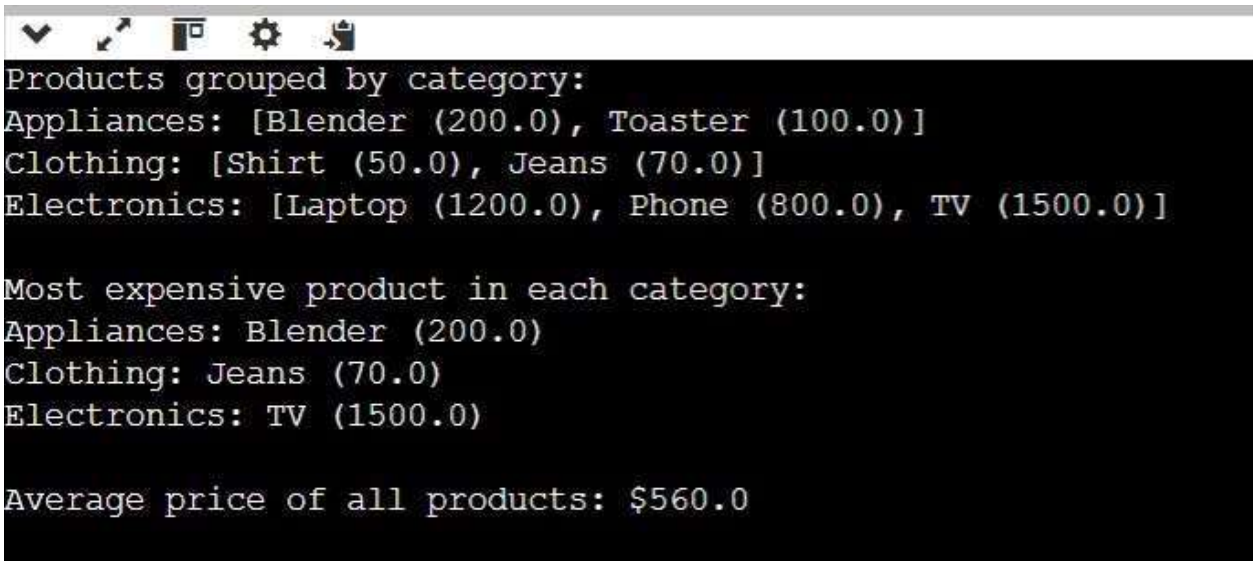
## 4. Output:



```
Sorted by Name: [Ayush - Age: 20, Salary: 90000.0, Prakul - Age: 23, Salary: 70000.0, Vinay - Age: 22, Salary: 100000.0]
Sorted by Age: [Ayush - Age: 20, Salary: 90000.0, Vinay - Age: 22, Salary: 100000.0, Prakul - Age: 23, Salary: 70000.0]
Sorted by Salary: [Prakul - Age: 23, Salary: 70000.0, Ayush - Age: 20, Salary: 90000.0, Vinay - Age: 22, Salary: 100000.0]
```



```
Top Students: [Vinay, Ayush, Prakul]
```



```
Products grouped by category:
Appliances: [Blender (200.0), Toaster (100.0)]
Clothing: [Shirt (50.0), Jeans (70.0)]
Electronics: [Laptop (1200.0), Phone (800.0), TV (1500.0)]

Most expensive product in each category:
Appliances: Blender (200.0)
Clothing: Jeans (70.0)
Electronics: TV (1500.0)

Average price of all products: $560.0
```

## 5. Learning Outcome:

- Understand and implement **lambda expressions** for sorting objects in a list based on different attributes.
- Utilize **Java Streams API** to perform operations like **filtering, sorting, and mapping** efficiently on large datasets.
- Learn **Comparator and method references** to simplify object comparisons for sorting.
- Apply **grouping and aggregation functions** using Collectors.groupingBy() and Collectors.maxBy() for processing categorized data.
- Gain hands-on experience in computing **statistical values** like the **average** from a dataset using mapToDouble() and average().
- Improve **code efficiency and readability** by using **functional programming** techniques in Java.