Experiment-6

Student Name: Gagnesh Kakkar **UID:** 23BCS11196
Branch: B.E-C.S.E **Section/Group:** 23KRG-2B
Semester: 5th **Date of Performance:** 06/10/2025
Subject Name: PBLJ **Subject Code:** 23CSH-304

Easy Level

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

2. **Objective:** To understand how lambda expressions simplify sorting logic and enhance code readability.

3. **Input/Apparatus Used:** Comparator, Lambda syntax. Comparator, Lambda syntax.

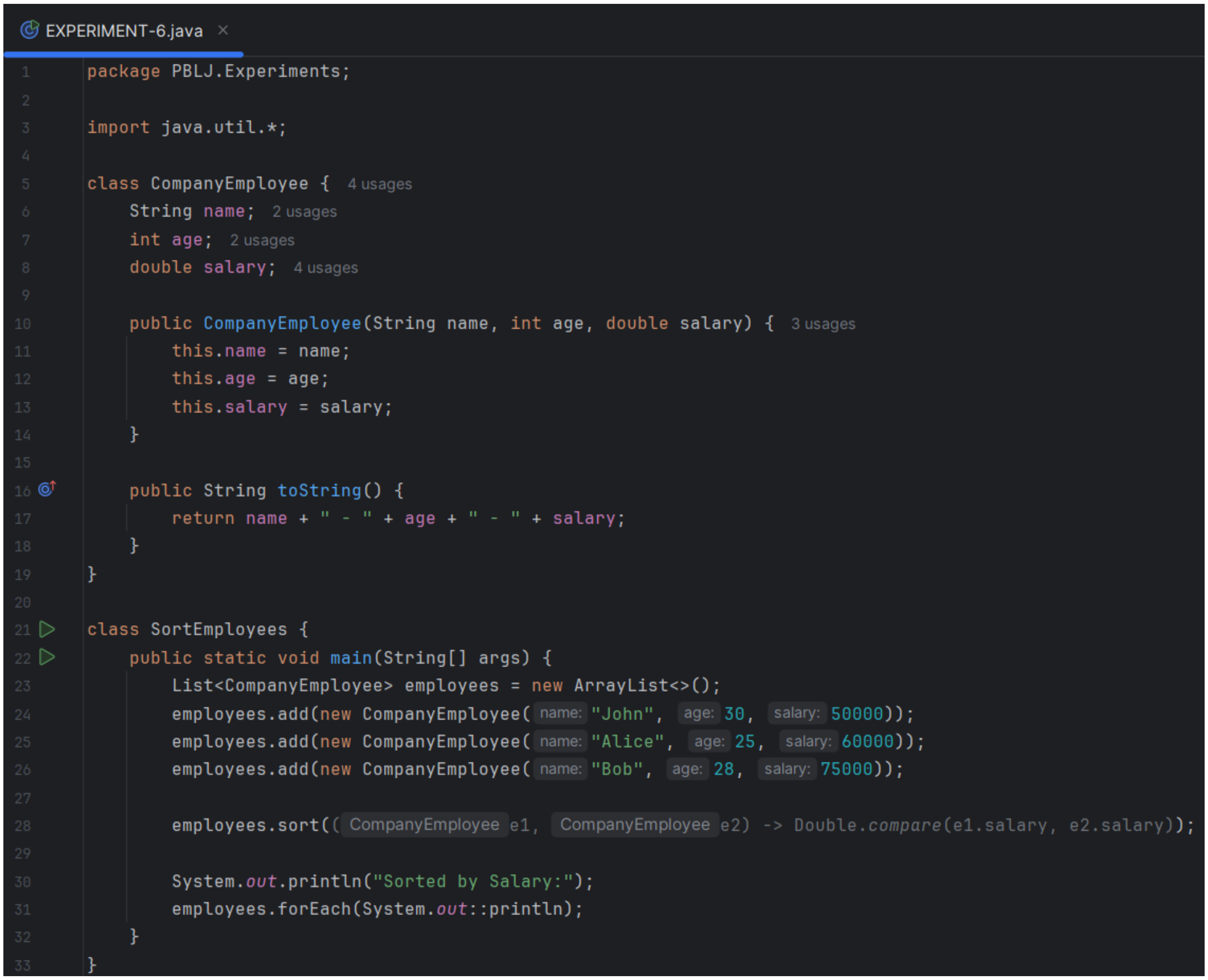
4. **Procedure:**
   1. Define an Employee class with name, age, and salary.
   2. Create a list of Employee objects.
   3. Use Collections.sort() or List.sort() with lambda expressions to sort by name, age, or salary.
   4. Display the sorted employee list.

5.

Sample Output:

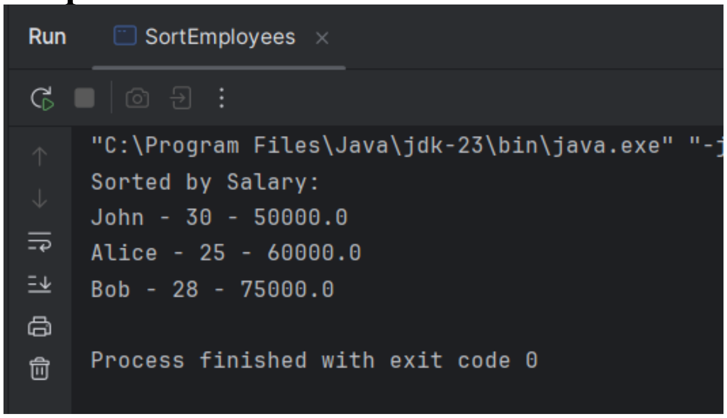
```
Sorted by Salary:
John - 30 - 50000
Alice - 25 - 60000
Bob - 28 - 75000
```

6. Code:



```java
package PBLJ.Experiments;

import java.util.*;

class CompanyEmployee {  4 usages
    String name;  2 usages
    int age;  2 usages
    double salary;  4 usages

    public CompanyEmployee(String name, int age, double salary) {  3 usages
        this.name = name;
        this.age = age;
        this.salary = salary;
    }

    public String toString() {
        return name + " - " + age + " - " + salary;
    }
}

class SortEmployees {
    public static void main(String[] args) {
        List<CompanyEmployee> employees = new ArrayList<>();
        employees.add(new CompanyEmployee( name: "John",  age: 30,  salary: 50000));
        employees.add(new CompanyEmployee( name: "Alice",  age: 25,  salary: 60000));
        employees.add(new CompanyEmployee( name: "Bob",  age: 28,  salary: 75000));

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

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

7. Output:



```
"C:\Program Files\Java\jdk-23\bin\java.exe" "-j
Sorted by Salary:
John - 30 - 50000.0
Alice - 25 - 60000.0
Bob - 28 - 75000.0

Process finished with exit code 0
```

Medium Level

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

2. **Objective:** To apply filtering, sorting, and transformation operations using the Stream API.

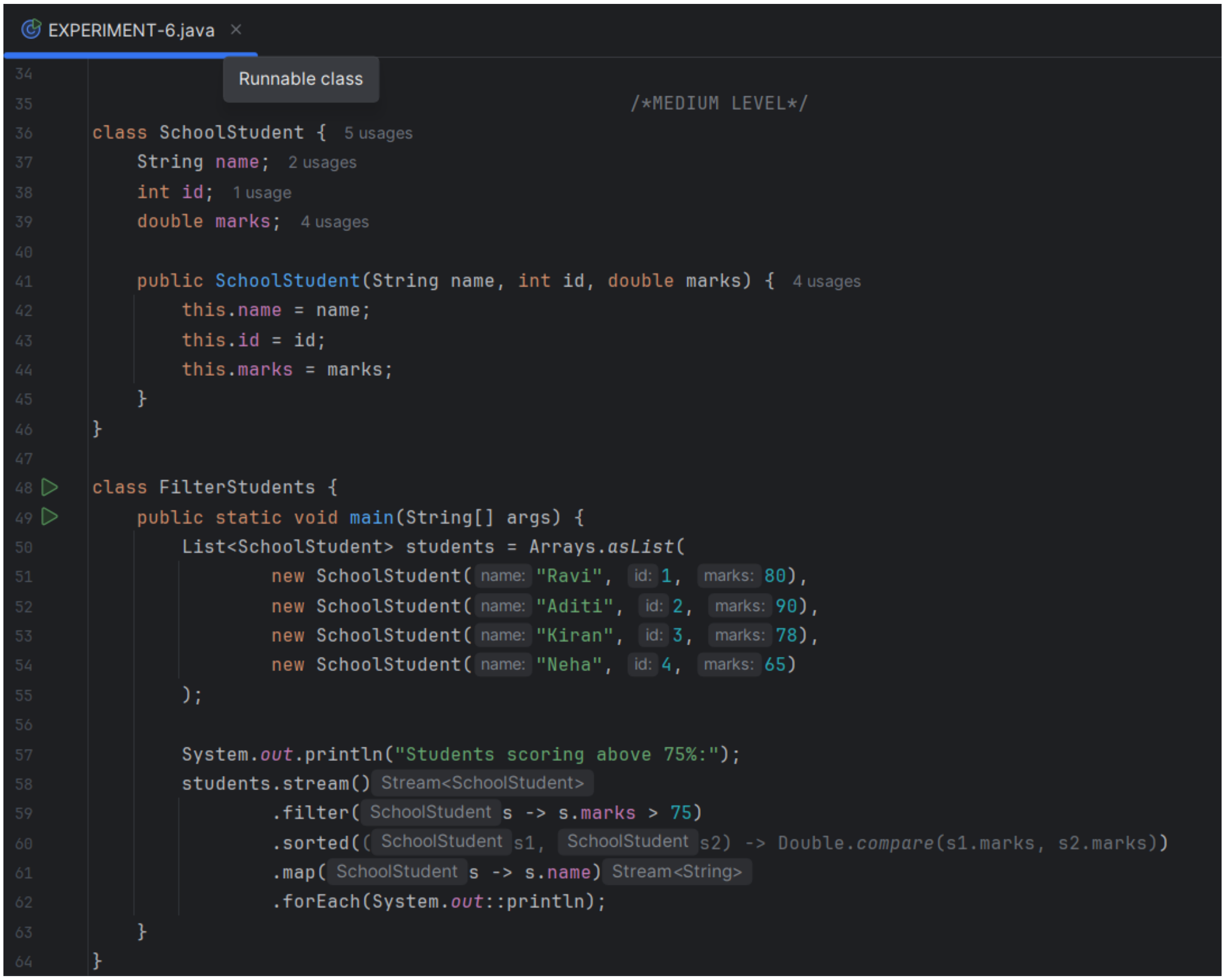
3. **Input/Apparatus Used:** Used: Stream, filter(), sorted(), map(), collect().

4. **Procedure:**
   1. Define a Student class with name, id, and marks.
   2. Create a list of students.
   3. Use Stream API to:
      - Filter students with marks > 75
      - Sort them by marks
      - Extract and display their names

5. **Sample Output :**

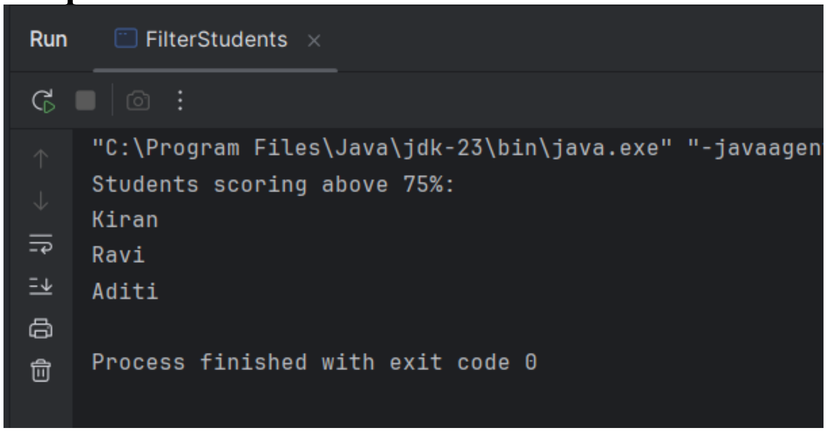
```
Students scoring above 75%:
Ravi
Aditi
Kiran
```

6. Code:



```java
                                                   /*MEDIUM LEVEL*/
class SchoolStudent {
    String name;
    int id;
    double marks;

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

class FilterStudents {
    public static void main(String[] args) {
        List<SchoolStudent> students = Arrays.asList(
                new SchoolStudent("Ravi", 1, 80),
                new SchoolStudent("Aditi", 2, 90),
                new SchoolStudent("Kiran", 3, 78),
                new SchoolStudent("Neha", 4, 65)
        );

        System.out.println("Students scoring above 75%:");
        students.stream()
                .filter(s -> s.marks > 75)
                .sorted((s1, s2) -> Double.compare(s1.marks, s2.marks))
                .map(s -> s.name)
                .forEach(System.out::println);
    }
}
```

7. Output:



```
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagen
Students scoring above 75%:
Kiran
Ravi
Aditi

Process finished with exit code 0
```

Hard Level

1. **Aim:** 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:** Demonstrate advanced stream operations including groupingBy, maxBy, and averagingDouble.

3. **Input/Apparatus Used:** Stream, Collectors.groupingBy(), Collectors.maxBy(), Collectors.averagingDouble().

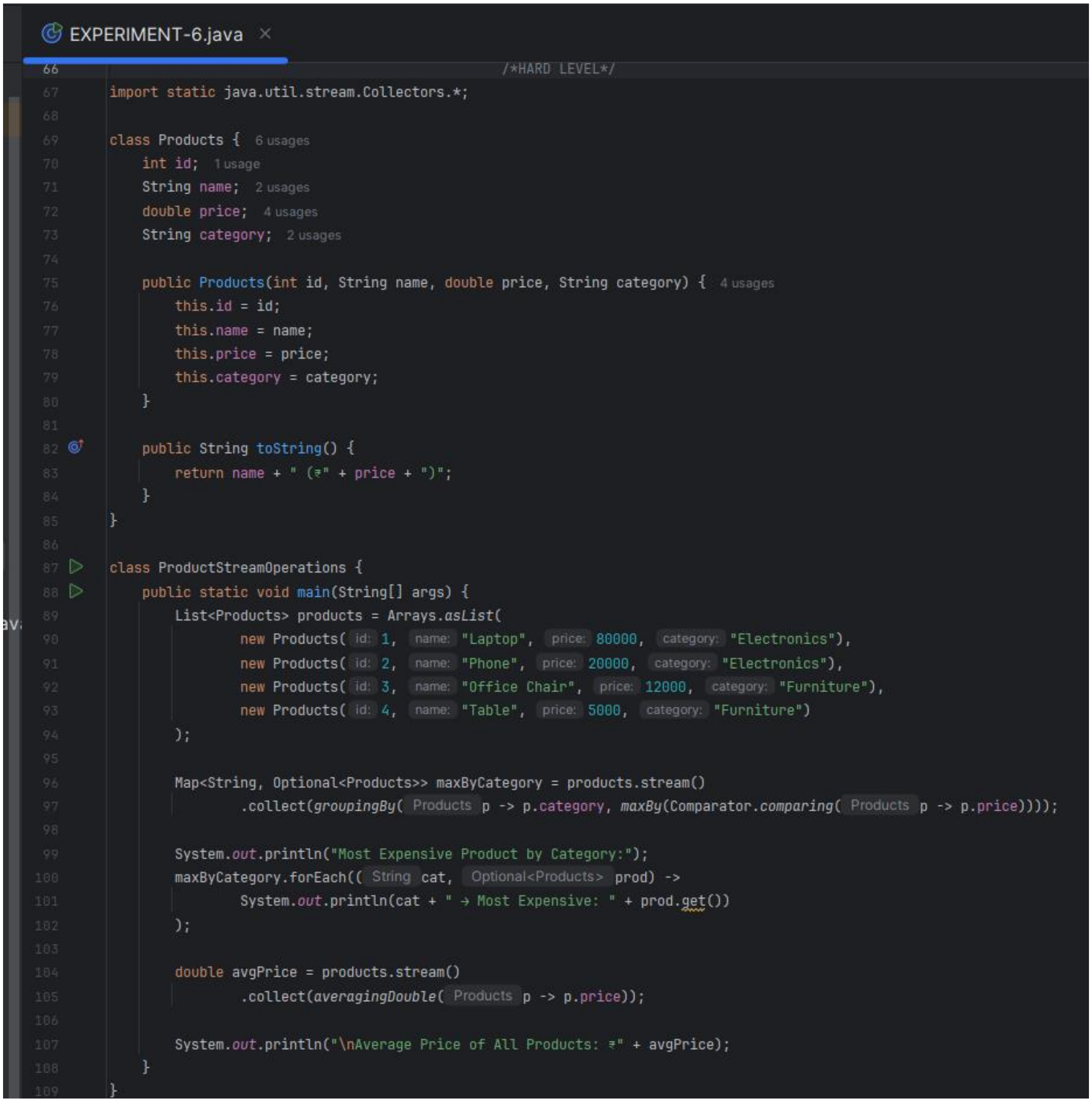
4. **Procedure:**

1. Define a Product class with id, name, price, and category.
2. Create a list of Product objects.
3. Use Stream API to:
   - Group products by category
   - Find most expensive product per category using maxBy()
   - Compute average price using averagingDouble()

Sample Output:

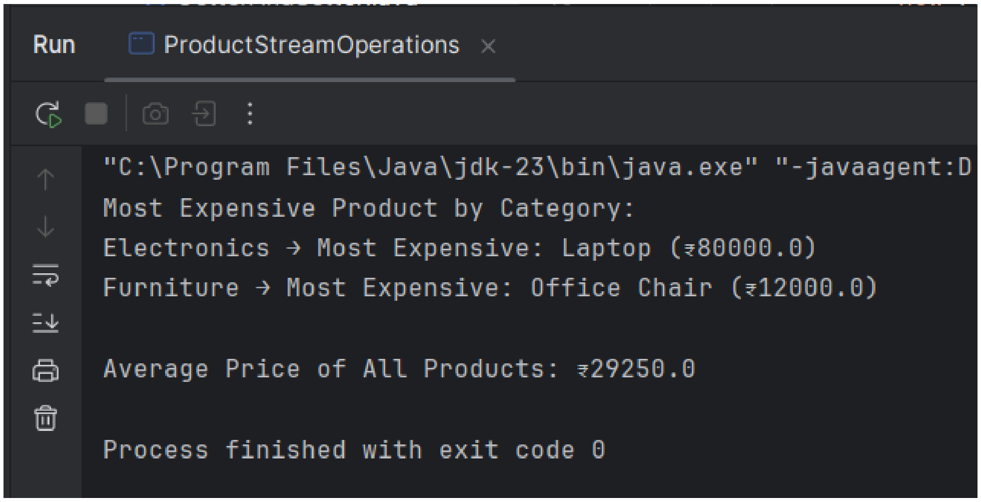
```
Electronics → Most Expensive: Laptop (₹80,000)
Furniture → Most Expensive: Office Chair (₹12,000)
Average Price of All Products: ₹15,200
```

5. Code:



```
                                                      /*HARD LEVEL*/
import static java.util.stream.Collectors.*;

class Products {  6 usages
    int id;  1 usage
    String name;  2 usages
    double price;  4 usages
    String category;  2 usages

    public Products(int id, String name, double price, String category) {  4 usages
        this.id = id;
        this.name = name;
        this.price = price;
        this.category = category;
    }

    public String toString() {
        return name + " (₹" + price + ")";
    }
}

class ProductStreamOperations {
    public static void main(String[] args) {
        List<Products> products = Arrays.asList(
                new Products( id: 1,  name: "Laptop",  price: 80000,  category: "Electronics"),
                new Products( id: 2,  name: "Phone",  price: 20000,  category: "Electronics"),
                new Products( id: 3,  name: "Office Chair",  price: 12000,  category: "Furniture"),
                new Products( id: 4,  name: "Table",  price: 5000,  category: "Furniture")
        );

        Map<String, Optional<Products>> maxByCategory = products.stream()
                .collect(groupingBy( Products p -> p.category, maxBy(Comparator.comparing( Products p -> p.price))));

        System.out.println("Most Expensive Product by Category:");
        maxByCategory.forEach(( String cat,  Optional<Products> prod) ->
                System.out.println(cat + " → Most Expensive: " + prod.get())
        );

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

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

6. Output:



```
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:D
Most Expensive Product by Category:
Electronics → Most Expensive: Laptop (₹80000.0)
Furniture → Most Expensive: Office Chair (₹12000.0)

Average Price of All Products: ₹29250.0

Process finished with exit code 0
```