



## Experiment 6

**Name:** Ayush Sharma

**Branch:** BE-CSE

**Semester:** 6<sup>th</sup>

**Subject Name:** Project Based Learning  
in Java with Lab

**UID:**22BCS12271

**Section/Group:**22BCS618-A

**Date of Performance:**28/02/2025

**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 :** The objective of Develop Java programs using lambda expressions and stream operations for sorting, filtering, and processing large datasets efficiently

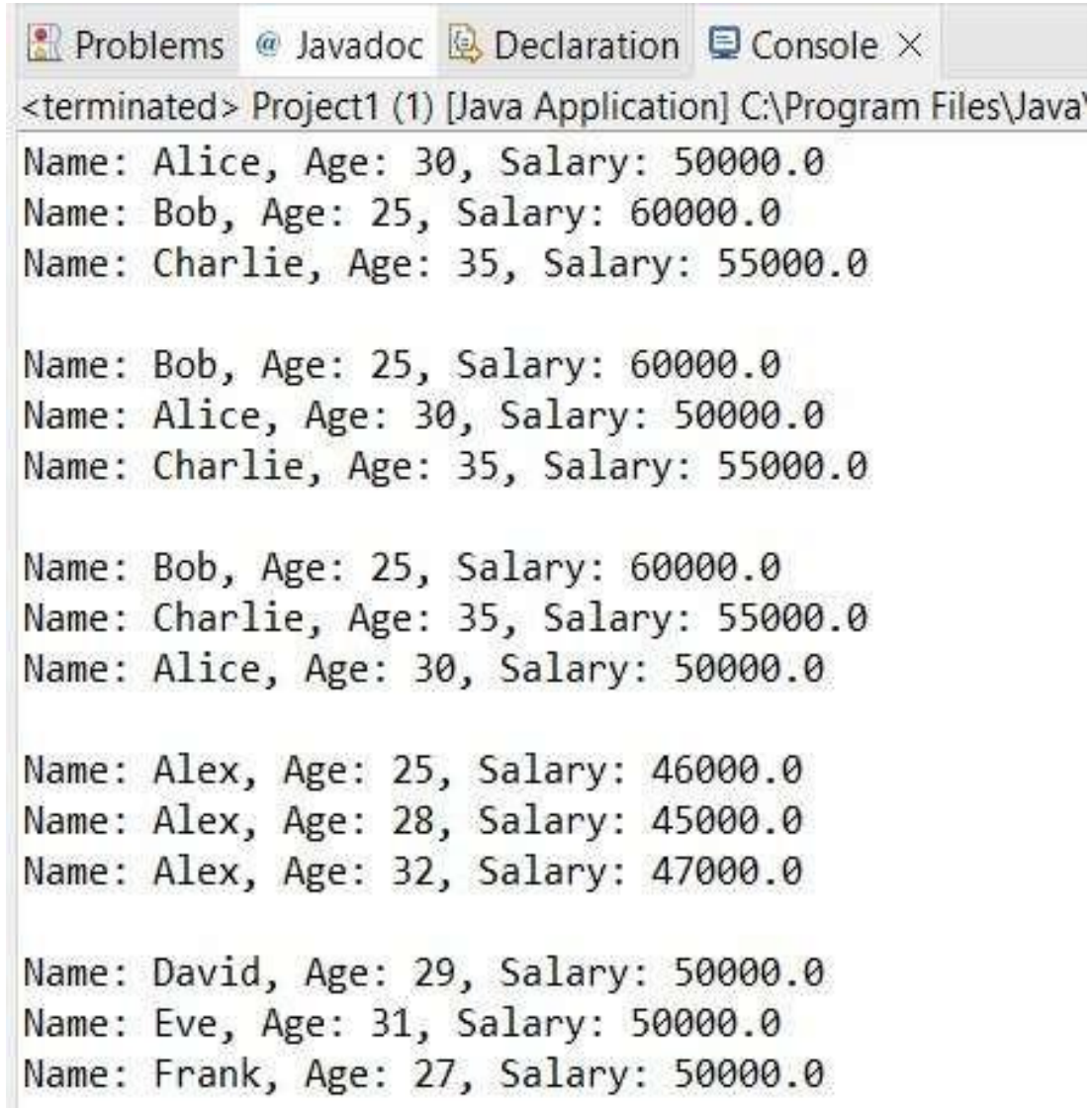
### **3. Implementation/Code:**

**3.1 Writing a Java program to implement /\*To implement a Java program that sorts a list of Employee objects (based on name, age, and salary) using lambda expressions and stream operations to demonstrate efficient data processing .**

```
import java.util.ArrayList;
import java.util.Comparator;
class Employee {
    private String name;
    private int age;
    private double salary;
    public Employee(String name, int age, double salary)
    { this.name = name;
      this.age = age;
      this.salary = salary;}
    public void display() {
        System.out.println("Name: " + name + ", Age: " + age + ", Salary: " + salary);}
    public String getName() {
        return name;}
    public int getAge() {
        return age;}
    public double getSalary()
    { return salary;}}
```

```
public class Main {
    public static void main(String[] args)
    {
        ArrayList<Employee> employees = new ArrayList<>();
        employees.add(new Employee("Alice", 30, 50000));
        employees.add(new Employee("Bob", 25, 60000));
        employees.add(new Employee("Charlie", 35, 55000));
        employees.stream()
            .sorted(Comparator.comparing(Employee::getName))
            .forEach(Employee::display);
        System.out.println();
        employees.stream()
            .sorted(Comparator.comparingInt(Employee::getAge))
            .forEach(Employee::display);
        System.out.println();
        employees.stream()
            .sorted(Comparator.comparingDouble(Employee::getSalary).reversed())
            .forEach(Employee::display);
        System.out.println();
        ArrayList<Employee> employees2 = new ArrayList<>();
        employees2.add(new Employee("Alex", 28, 45000));
        employees2.add(new Employee("Alex", 32, 47000));
        employees2.add(new Employee("Alex", 25, 46000));
        employees2.stream()
            .sorted(Comparator.comparingInt(Employee::getAge))
            .forEach(Employee::display);
        System.out.println();
        ArrayList<Employee> employees3 = new ArrayList<>();
        employees3.add(new Employee("David", 29, 50000));
        employees3.add(new Employee("Eve", 31, 50000));
        employees3.add(new Employee("Frank", 27, 50000));
        employees3.stream()
            .sorted(Comparator.comparing(Employee::getName))
            .forEach(Employee::display);
    }
}
```

## Output:



```
<terminated> Project1 (1) [Java Application] C:\Program Files\Java\
Name: Alice, Age: 30, Salary: 50000.0
Name: Bob, Age: 25, Salary: 60000.0
Name: Charlie, Age: 35, Salary: 55000.0

Name: Bob, Age: 25, Salary: 60000.0
Name: Alice, Age: 30, Salary: 50000.0
Name: Charlie, Age: 35, Salary: 55000.0

Name: Bob, Age: 25, Salary: 60000.0
Name: Charlie, Age: 35, Salary: 55000.0
Name: Alice, Age: 30, Salary: 50000.0

Name: Alex, Age: 25, Salary: 46000.0
Name: Alex, Age: 28, Salary: 45000.0
Name: Alex, Age: 32, Salary: 47000.0

Name: David, Age: 29, Salary: 50000.0
Name: Eve, Age: 31, Salary: 50000.0
Name: Frank, Age: 27, Salary: 50000.0
```

Fig. 1 (Output 3.1)

**3.2 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.io.*;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
import java.util.Comparator;
class Student {
    private String name;
    private double marks;
    public Student(String name, double marks)
        { this.name = name;
          this.marks = marks;}
    public void display() {
        System.out.println(name);}
    public String getName() {
        return name;
    }
    public double getMarks()
        { return marks;
    }
}
public class Project1 {
    public static void main(String[] args)
        { testCase1();
          testCase2();
          testCase3();
          testCase4();
        }
    private static void runTest(List<Student> students)
        { List<Student> filteredSortedStudents = students.stream()
            .filter(s -> s.getMarks() > 75)
            .sorted(Comparator.comparingDouble(Student::getMarks).reversed())
            .thenComparing(Student::getName))
            .collect(Collectors.toList());
        if (filteredSortedStudents.isEmpty())
            { System.out.println("No output (Empty List)");
          } else {
            filteredSortedStudents.forEach(Student::display);
          }
        System.out.println();
    }
    private static void testCase1() {
        List<Student> students = new ArrayList<>();
        students.add(new Student("Alice", 80));
```

```
students.add(new Student("Bob", 72));
students.add(new Student("Charlie", 90));
students.add(new Student("David", 65));
students.add(new Student("Eve", 85));
runTest(students);}

private static void testCase2() {
    List<Student> students = new ArrayList<>();
    students.add(new Student("Bob", 70));
    students.add(new Student("David", 60));
    students.add(new Student("Frank", 65));
    runTest(students);}

private static void testCase3()
{ List<Student> students = new
ArrayList<>(); students.add(new
Student("Alice", 80));
students.add(new Student("Bob", 80));
students.add(new Student("Charlie", 85));
runTest(students);}

private static void testCase4() {
    List<Student> students = new ArrayList<>();
    students.add(new Student("Alice", 60));
    students.add(new Student("Bob", 50));
    students.add(new Student("Charlie", 90));
    runTest(students);}}
```

## Output:



```
<terminated> Project1 (1) [Java Application] C:\Program Files\Java\jdk-21\
Charlie
Eve
Alice

No output (Empty List)

Charlie
Alice
Bob

Charlie
```

**Fig. 2 (Output 3.2)**

**3.3 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.**

```
import java.util.*;
import java.util.stream.Collectors;
import java.util.Optional;
class Product {
    private String name;
    private String category;
    private double price;
    public Product(String name, String category, double price)
    { this.name = name;
      this.category = category;
      this.price = price;}
    public String getName()
    { return name;}
    public String getCategory()
    { return category;}
    public double getPrice()
    { return price;}}
public class Project1 {
    public static void main(String[] args)
    { testCase1();
      testCase2();
      testCase3();
      testCase4();
      testCase5();}
    private static void runTest(List<Product> products) {
        Map<String, List<Product>> groupedByCategory = products.stream()
            .collect(Collectors.groupingBy(Product::getCategory));

        Map<String, Optional<Product>> mostExpensiveByCategory = products.stream()
            .collect(Collectors.groupingBy(Product::getCategory,
                Collectors.maxBy(Comparator.comparingDouble(Product::getPrice))));
        double averagePrice = products.stream()
            .collect(Collectors.averagingDouble(Product::getPrice));
        System.out.println("Grouped Products:");
        groupedByCategory.forEach((category, productList) -> {
            System.out.println(category + ": " + productList.stream()
                .map(Product::getName)
                .collect(Collectors.joining(", ")));
        });
    }
```

```

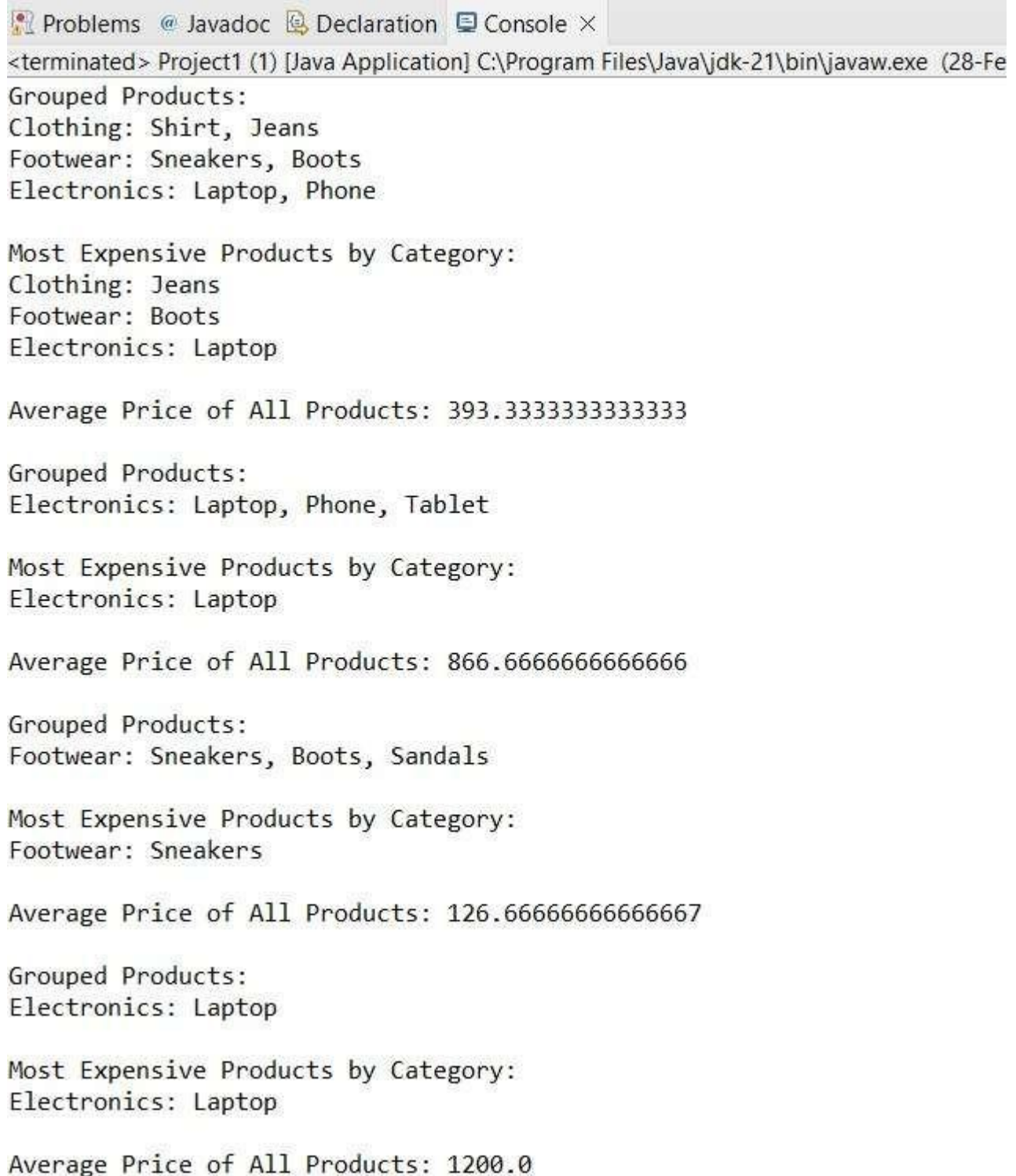
        System.out.println("\nMost Expensive Products by Category:");
        mostExpensiveByCategory.forEach((category, product) ->
            System.out.println(category + ": " + product.map(Product::getName).orElse("No product")));

        System.out.println("\nAverage Price of All Products: " + averagePrice);
        System.out.println();
    }
    private static void testCase1()
    { List<Product> products = Arrays.asList(
        new Product("Laptop", "Electronics", 1200),
        new Product("Phone", "Electronics", 800),
        new Product("Shirt", "Clothing", 50),
        new Product("Jeans", "Clothing", 60),
        new Product("Sneakers", "Footwear", 100),
        new Product("Boots", "Footwear", 150)
    );
    runTest(products);
}
    private static void testCase2()
    { List<Product> products = Arrays.asList(
        new Product("Laptop", "Electronics", 1200),
        new Product("Phone", "Electronics", 800),
        new Product("Tablet", "Electronics", 600)
    );
    runTest(products);
}
    private static void testCase3()
    { List<Product> products = Arrays.asList(
        new Product("Sneakers", "Footwear", 150),
        new Product("Boots", "Footwear", 150),
        new Product("Sandals", "Footwear", 80)
    );
    runTest(products);
}
    private static void testCase4()
    { List<Product> products = Arrays.asList(
        new Product("Laptop", "Electronics", 1200)
    );
    runTest(products);
}
    private static void testCase5() {
        List<Product> products = new ArrayList<>();
        runTest(products);
    }
}

```



## Output:



```
Problems @ Javadoc Declaration Console X
<terminated> Project1 (1) [Java Application] C:\Program Files\Java\jdk-21\bin\javaw.exe (28-Feb-2025 10:00:00 AM)
Grouped Products:
Clothing: Shirt, Jeans
Footwear: Sneakers, Boots
Electronics: Laptop, Phone

Most Expensive Products by Category:
Clothing: Jeans
Footwear: Boots
Electronics: Laptop

Average Price of All Products: 393.3333333333333

Grouped Products:
Electronics: Laptop, Phone, Tablet

Most Expensive Products by Category:
Electronics: Laptop

Average Price of All Products: 866.6666666666666

Grouped Products:
Footwear: Sneakers, Boots, Sandals

Most Expensive Products by Category:
Footwear: Sneakers

Average Price of All Products: 126.66666666666667

Grouped Products:
Electronics: Laptop

Most Expensive Products by Category:
Electronics: Laptop

Average Price of All Products: 1200.0
```

**Fig. 3 (Output 3.3)**



## 4. Learning Outcomes:

- **Wrapper Classes:** Understand autoboxing and unboxing for integrating primitive types with Java collections using wrapper classes.
- **Serialization:** Implement serialization and deserialization of custom objects, ensuring data persistence and effective file handling.
- **Cloning:** Explore shallow and deep cloning mechanisms for managing object copies in Java.
- **Lambda Expressions:** Grasp the use of lambda expressions and functional interfaces to simplify code and enhance reusability.
- **Method References:** Utilize method references and functional programming techniques for efficient data processing and improved code readability.