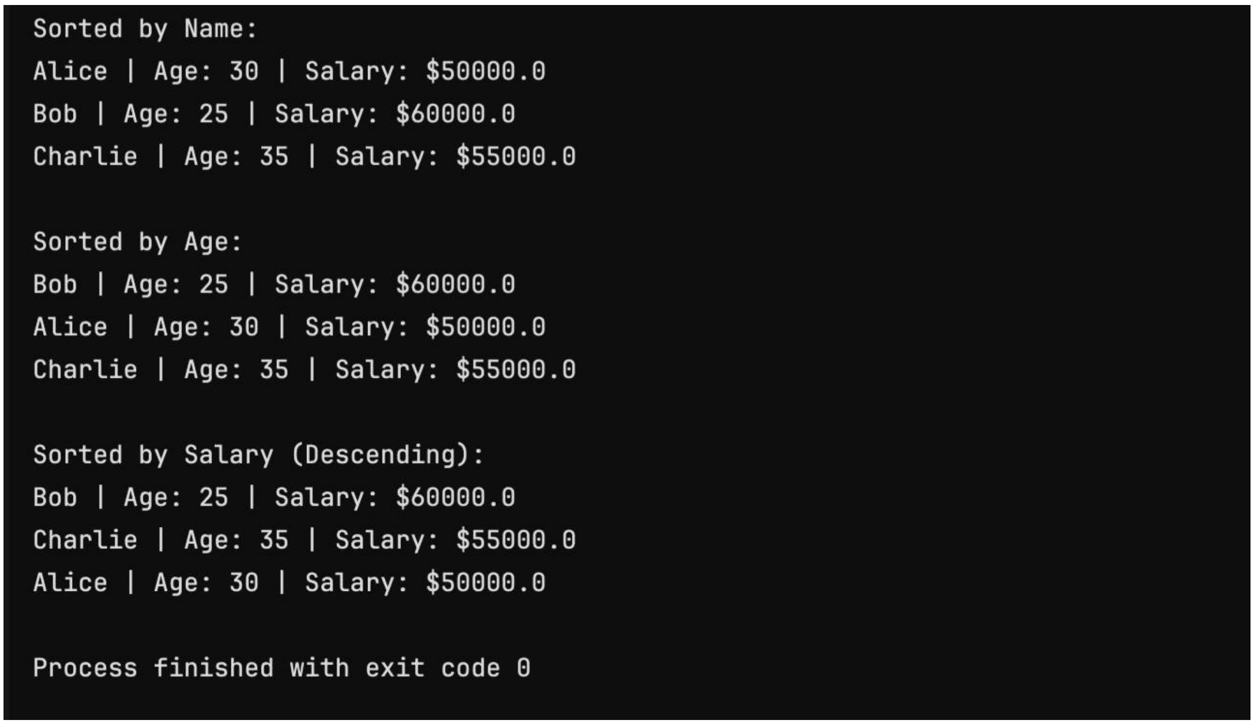
a) Write a program to sort a list of Employee objects (name, age, salary) using lambda expressions.

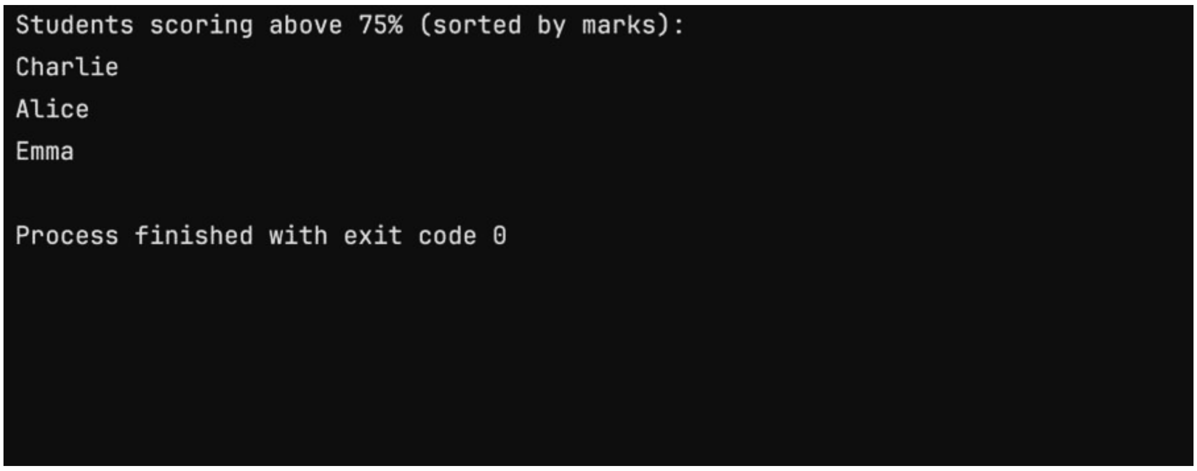
```java
import java.util.*;
class Employee {
    String name;
    int age;
    double salary;
    public Employee(String name, int age, double salary) {
        this.name = name;
        this.age = age;
        this.salary = salary;
    }
    @Override
    public String toString() {
        return name + " | Age: " + age + " | Salary: $" + salary;
    }
}
public class EmployeeSorter {
    public static void main(String[] args) {
        List<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.sort((e1, e2) -> e1.name.compareTo(e2.name));
        System.out.println("Sorted by Name:");
        employees.forEach(System.out::println);
        employees.sort(Comparator.comparingInt(e -> e.age));
        System.out.println("\nSorted by Age:");
        employees.forEach(System.out::println);
        employees.sort((e1, e2) -> Double.compare(e2.salary, e1.salary));
        System.out.println("\nSorted by Salary (Descending):");
        employees.forEach(System.out::println);
    }
}
```



```
Sorted by Name:
Alice | Age: 30 | Salary: $50000.0
Bob | Age: 25 | Salary: $60000.0
Charlie | Age: 35 | Salary: $55000.0

Sorted by Age:
Bob | Age: 25 | Salary: $60000.0
Alice | Age: 30 | Salary: $50000.0
Charlie | Age: 35 | Salary: $55000.0

Sorted by Salary (Descending):
Bob | Age: 25 | Salary: $60000.0
Charlie | Age: 35 | Salary: $55000.0
Alice | Age: 30 | Salary: $50000.0

Process finished with exit code 0
```

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

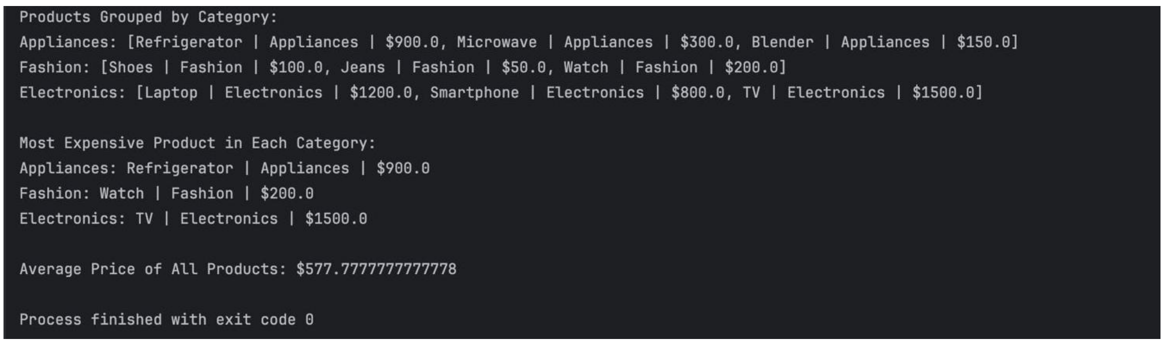
```
import java.util.*;
import java.util.stream.Collectors;
class Student {
    String name;
    double marks;
    public Student(String name, double marks) {
        this.name = name;
        this.marks = marks;
    }
    @Override
    public String toString() {
        return name + " | Marks: " + marks;
    }
}
public class StudentFilter {
    public static void main(String[] args) {
        List<Student> students = Arrays.asList(
            new Student("Alice", 85),
            new Student("Bob", 65),
            new Student("Charlie", 90),
            new Student("David", 70),
            new Student("Emma", 80)
        );
        List<String> topStudents = students.stream()
            .filter(s -> s.marks > 75)
            .sorted((s1, s2) -> Double.compare(s2.marks, s1.marks))
            .map(s -> s.name)
            .collect(Collectors.toList());
        System.out.println("Students scoring above 75% (sorted by 
marks):");
        topStudents.forEach(System.out::println);
    }
}
```



```
Students scoring above 75% (sorted by marks):
Charlie
Alice
Emma

Process finished with exit code 0
```

c) 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.

```java
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("Smartphone", "Electronics", 800),
            new Product("Refrigerator", "Home Appliances", 1500),
            new Product("Microwave", "Home Appliances", 300),
            new Product("Table", "Furniture", 200),
            new Product("Chair", "Furniture", 100)
        );
        Map<String, List<Product>> groupedByCategory = products.stream()
            .collect(Collectors.groupingBy(p -> p.category));
        System.out.println("Products Grouped by Category:");
        groupedByCategory.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.get()));
        double averagePrice = products.stream()
            .mapToDouble(p -> p.price)
            .average()
            .orElse(0.0);
        System.out.println("\nAverage Price of All Products: $" + averagePrice);
```



```
Products Grouped by Category:
Appliances: [Refrigerator | Appliances | $900.0, Microwave | Appliances | $300.0, Blender | Appliances | $150.0]
Fashion: [Shoes | Fashion | $100.0, Jeans | Fashion | $50.0, Watch | Fashion | $200.0]
Electronics: [Laptop | Electronics | $1200.0, Smartphone | Electronics | $800.0, TV | Electronics | $1500.0]

Most Expensive Product in Each Category:
Appliances: Refrigerator | Appliances | $900.0
Fashion: Watch | Fashion | $200.0
Electronics: TV | Electronics | $1500.0

Average Price of All Products: $577.7777777777778

Process finished with exit code 0
```