NAME : Agrima Sharma **UID :** 22BCS15314 **SECTION :** 607-A

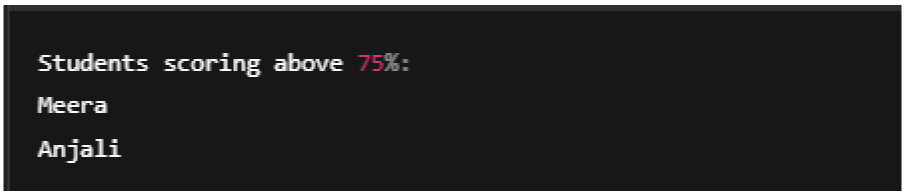
//Problem 1: Sort a list of Employee objects based on name, age, and salary.

```
import java.util.*;

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 String getName() { return name; }
    public int getAge() { return age; }
    public double getSalary() { return salary; }

    @Override
    public String toString() {
        return "Employee{name='" + name + "', age=" + age + ", salary=" + salary +
"}";
    }
}

public class EasyLevel {
    public static void main(String[] args) {
        List<Employee> employees = Arrays.asList(
            new Employee("Alice", 30, 60000),
            new Employee("Bob", 25, 50000),
            new Employee("Charlie", 28, 70000)
        );

        System.out.println("Sorted by Name:");

employees.stream().sorted(Comparator.comparing(Employee::getName)).forEa
ch(System.out::println);

        System.out.println("\nSorted by Age:");

employees.stream().sorted(Comparator.comparing(Employee::getAge)).forEach
(System.out::println);

        System.out.println("\nSorted by Salary:");

employees.stream().sorted(Comparator.comparing(Employee::getSalary)).forEa
ch(System.out::println);
    }
}
```

```
Sorted by Name:
Employee{name='Alice', age=30, salary=60000.0}
Employee{name='Bob', age=25, salary=50000.0}
Employee{name='Charlie', age=28, salary=70000.0}

Sorted by Age:
Employee{name='Bob', age=25, salary=50000.0}
Employee{name='Charlie', age=28, salary=70000.0}
Employee{name='Alice', age=30, salary=60000.0}

Sorted by Salary:
Employee{name='Bob', age=25, salary=50000.0}
Employee{name='Alice', age=30, salary=60000.0}
Employee{name='Charlie', age=28, salary=70000.0}
```

//Problem 2: Filter students scoring above 75%, sort by marks, and display names.

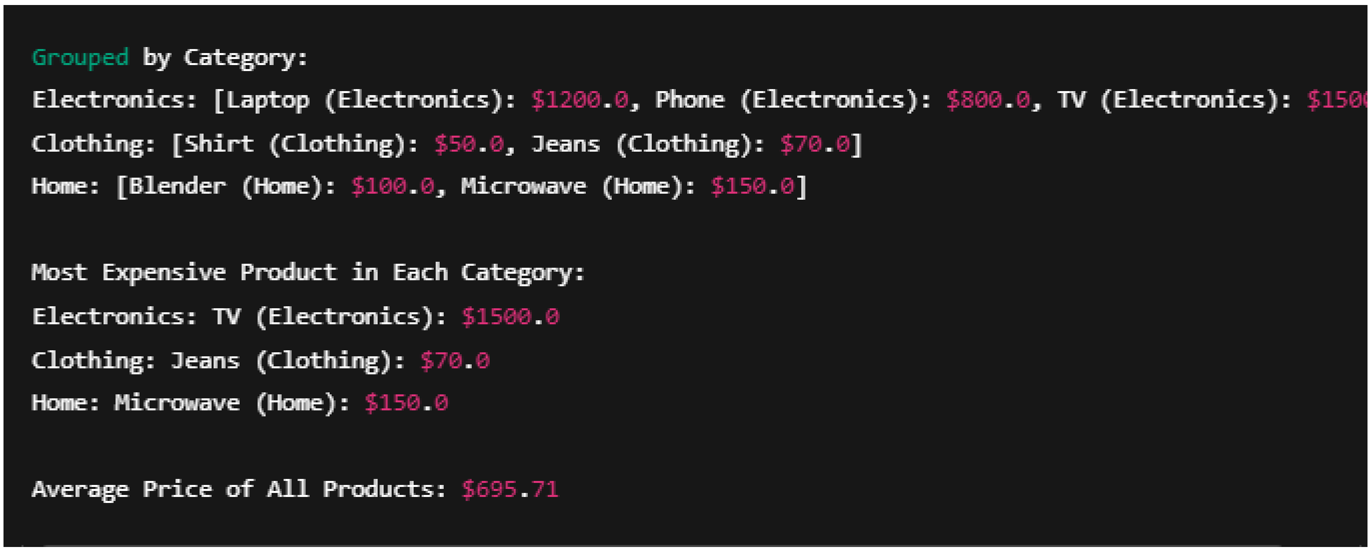
```
import java.util.*;
import java.util.stream.*;

class Student {
    String name;
    double marks;

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

    public String getName() { return name; }
    public double getMarks() { return marks; }
}

public class MediumLevel {
    public static void main(String[] args) {
        List<Student> students = Arrays.asList(
            new Student("Anjali", 82.5),
            new Student("Ravi", 74.0),
            new Student("Meera", 91.0),
            new Student("Arjun", 67.5)
        );

        List<String> topStudents = students.stream()
            .filter(s -> s.getMarks() > 75)
            .sorted(Comparator.comparing(Student::getMarks).reversed())
            .map(Student::getName)
            .collect(Collectors.toList());

        System.out.println("Students scoring above 75%:");
        topStudents.forEach(System.out::println);
    }
}
```



```
Students scoring above 75%:
Meera
Anjali
```

Problem 3: Process a large dataset of products using streams.

```
import java.util.*;
import java.util.stream.*;
import java.util.function.*;
import java.util.Map.Entry;

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

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

    public String getCategory() { return category; }
    public double getPrice() { return price; }
    public String getName() { return name; }

    @Override
    public String toString() {
        return name + " (" + category + "): $" + price;
    }
}

public class HardLevel {
    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", "Home", 100),
            new Product("Microwave", "Home", 150)
        );

        System.out.println("Grouped by Category:");
        Map<String, List<Product>> grouped = products.stream()
            .collect(Collectors.groupingBy(Product::getCategory));
        grouped.forEach((category, prodList) -> {
            System.out.println(category + ": " + prodList);
        });

        System.out.println("\nMost Expensive Product in Each Category:");
        Map<String, Optional<Product>> maxPrice = products.stream()
            .collect(Collectors.groupingBy(Product::getCategory,
                Collectors.maxBy(Comparator.comparing(Product::getPrice))));
        maxPrice.forEach((category, product) ->
            System.out.println(category + ": " + product.orElse(null)));

        double avgPrice =
                products.stream().collect(Collectors.averagingDouble(Product::getPrice));
                    System.out.println("\nAverage Price of All Products: $" + avgPrice);
                }
}
```



```
Grouped by Category:
Electronics: [Laptop (Electronics): $1200.0, Phone (Electronics): $800.0, TV (Electronics): $150
Clothing: [Shirt (Clothing): $50.0, Jeans (Clothing): $70.0]
Home: [Blender (Home): $100.0, Microwave (Home): $150.0]

Most Expensive Product in Each Category:
Electronics: TV (Electronics): $1500.0
Clothing: Jeans (Clothing): $70.0
Home: Microwave (Home): $150.0

Average Price of All Products: $695.71
```