Task1.异常

Q1. 请你简单概述一下Error与Exception的区别 当发生Exception和Error时 程序的处理态度分别应该是什么?

1.区别:

- **Error** 是严重错误，通常表示程序无法恢复的系统级问题（如虚拟机崩溃、内存溢出），应由 JVM 处理，程序一般不捕获。
- **Exception** 是可预期的异常情况（如文件不存在、除零），程序应主动处理或声明抛出。

  区别本质：Error 是“灾难性”事件，超出程序控制范围；Exception 是“可处理”的逻辑错误，程序员可通过 try-catch 或 throws 机制应对。

2.处理态度:

- 对于 Exception，应该**主动处理**（try-catch）或**声明抛出**（throws），保证程序健壮性。
- 对于 Error，通常**无需处理**，因为其发生时程序已处于不稳定状态，继续运行可能造成更大风险。

3.心得:

异常处理不仅是技术手段，更是对程序稳定性和用户体验的保障。区分 Error 和 Exception，能帮助我们更合理地分配精力去“预防”和“修复”。

Q2. 你知道这两种异常有什么区别吗 他们发生的原因分别是什么?

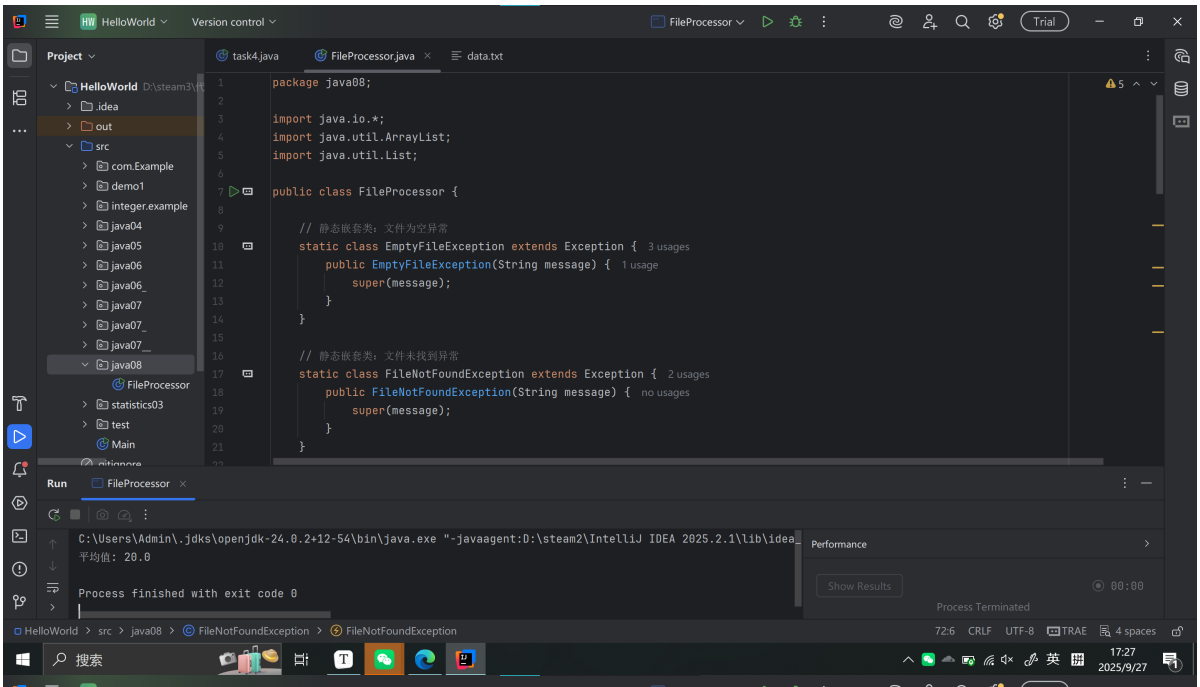
- **Checked 异常**：编译时检查，必须处理（try-catch 或 throws），如 `IOException`。原因：外部环境问题（如文件不存在、网络中断）。
- **Unchecked 异常**：运行时异常，不强制处理，如 `NullPointerException`、`ArithmeticException`。原因：程序逻辑错误（如空指针、除零）。

  心得：合理使用两种异常类型，能体现对“可控”与“不可控”风险的认知。开发者应主动处理 Checked 异常，避免 Unchecked 异常的发生，让程序既安全又高效。

Task2.处理

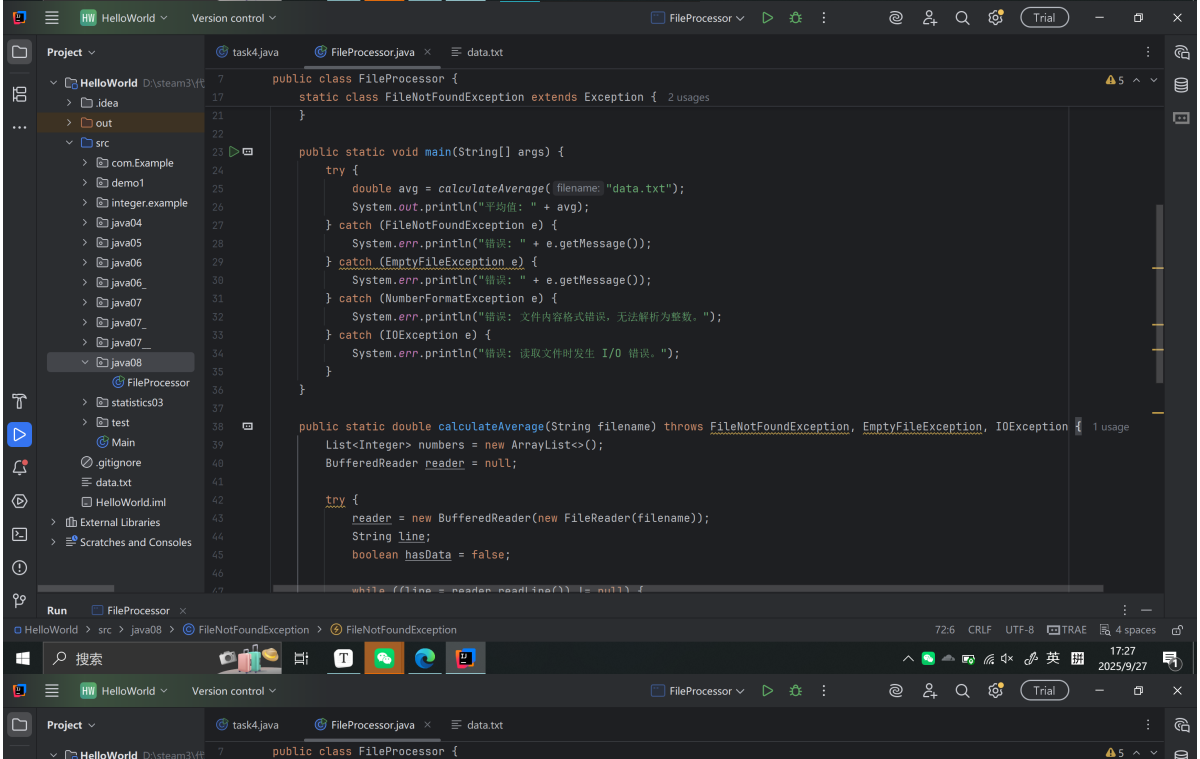
Q3. 题目: 文件读取和数据处理

编写一个Java程序，要求完成以下功能：1.读取文件；2.数据处理；3.异常处理；

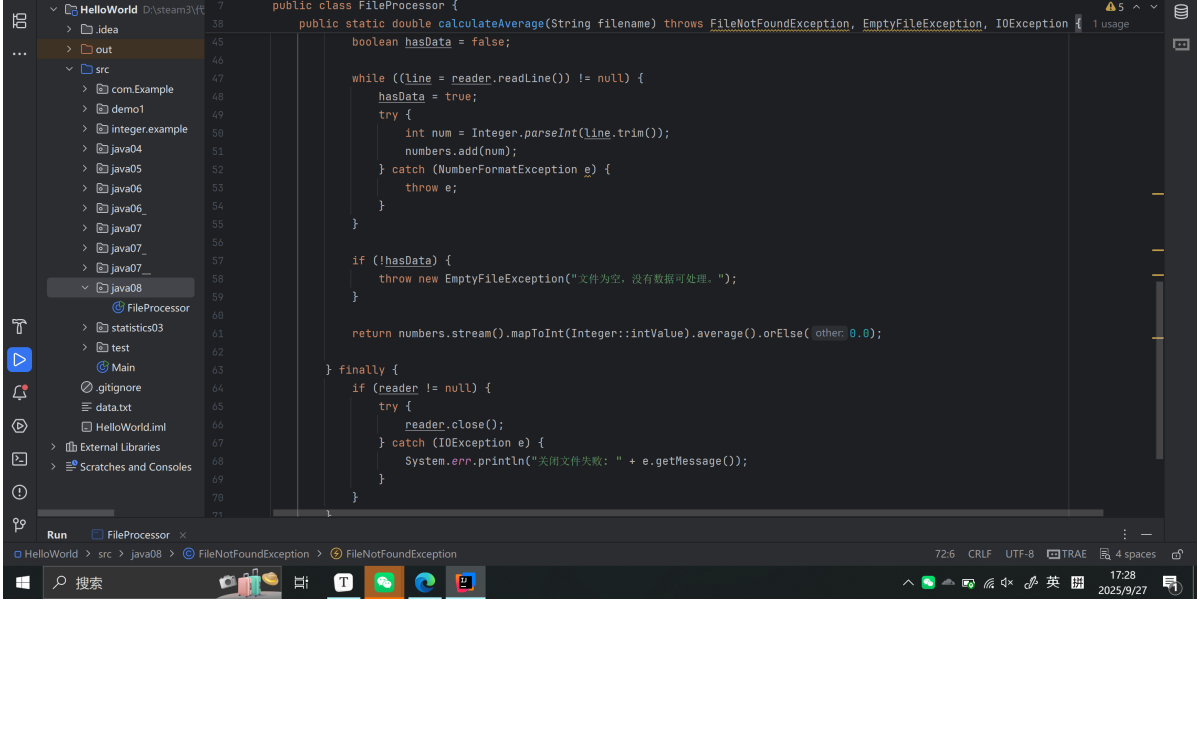


```
package java08;

import java.io.*;
import java.util.ArrayList;
import java.util.List;

public class FileProcessor {

    // 静态嵌套类：文件为空异常
    static class EmptyFileException extends Exception {  3 usages
        public EmptyFileException(String message) {  1 usage
            super(message);
        }
    }

    // 静态嵌套类：文件未找到异常
    static class FileNotFoundException extends Exception {  2 usages
        public FileNotFoundException(String message) {  no usages
            super(message);
        }
    }
```

```
C:\Users\Admin\.jdks\openjdk-24.0.2+12-54\bin\java.exe "-javaagent:D:\steam2\IntelliJ IDEA 2025.2.1\lib\idea
平均值: 20.0

Process finished with exit code 0
```



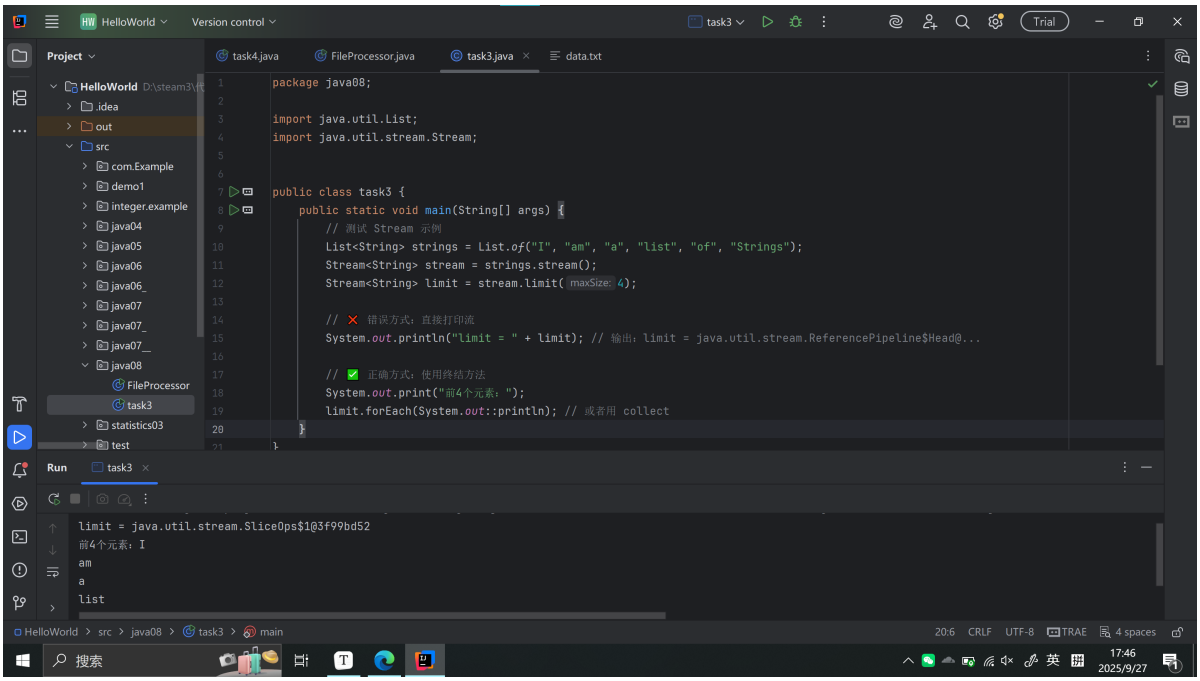
```
public class FileProcessor {
    static class FileNotFoundException extends Exception {  2 usages
    }

    public static void main(String[] args) {
        try {
            double avg = calculateAverage( filename: "data.txt");
            System.out.println("平均值: " + avg);
        } catch (FileNotFoundException e) {
            System.err.println("错误: " + e.getMessage());
        } catch (EmptyFileException e) {
            System.err.println("错误: " + e.getMessage());
        } catch (NumberFormatException e) {
            System.err.println("错误: 文件内容格式错误，无法解析为整数。");
        } catch (IOException e) {
            System.err.println("错误: 读取文件时发生 I/O 错误。");
        }
    }

    public static double calculateAverage(String filename) throws FileNotFoundException, EmptyFileException, IOException {  1 usage
        List<Integer> numbers = new ArrayList<>();
        BufferedReader reader = null;

        try {
            reader = new BufferedReader(new FileReader(filename));
            String line;
            boolean hasData = false;
```



```
public class FileProcessor {
    public static double calculateAverage(String filename) throws FileNotFoundException, EmptyFileException, IOException {  1 usage
            boolean hasData = false;

            while ((line = reader.readLine()) != null) {
                hasData = true;
                try {
                    int num = Integer.parseInt(line.trim());
                    numbers.add(num);
                } catch (NumberFormatException e) {
                    throw e;
                }
            }

            if (!hasData) {
                throw new EmptyFileException("文件为空，没有数据可处理。");
            }

            return numbers.stream().mapToInt(Integer::intValue).average().orElse( other: 0.0);

        } finally {
            if (reader != null) {
                try {
                    reader.close();
                } catch (IOException e) {
                    System.err.println("关闭文件失败: " + e.getMessage());
                }
            }
        }
```

Task3.Stream流

Q4. 尝试执行上面代码 会得到什么结果? 这和你预想的一样吗? 为什么会出现这种结果?



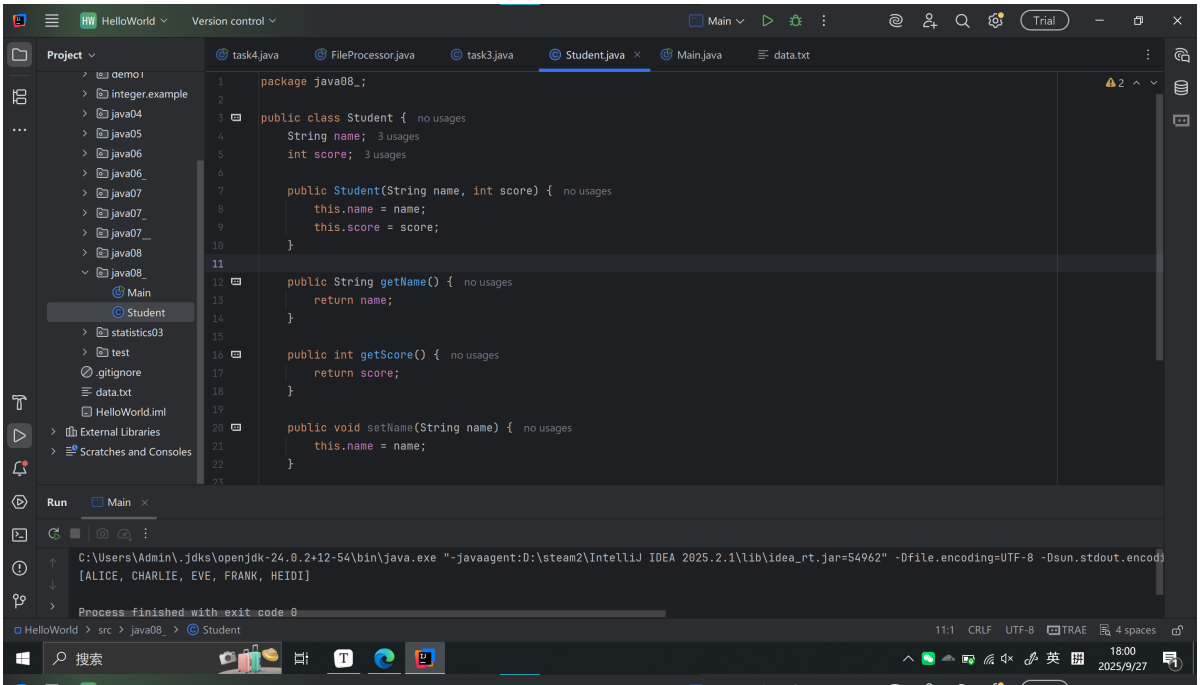
为什么出现这种结果?

- `System.out.println(limit)` 打印的是 `Stream` 对象本身，而不是它的内容。
- `limit(4)` 是**中间操作**，不会立即执行。
- 只有调用 `forEach`、`collect`、`count` 等**终结方法**时，才会触发流的真正处理。

  （心得：Stream 是"懒加载"的，只有终结方法才会真正执行。记住：**没有终结方法，流就不会动！**）

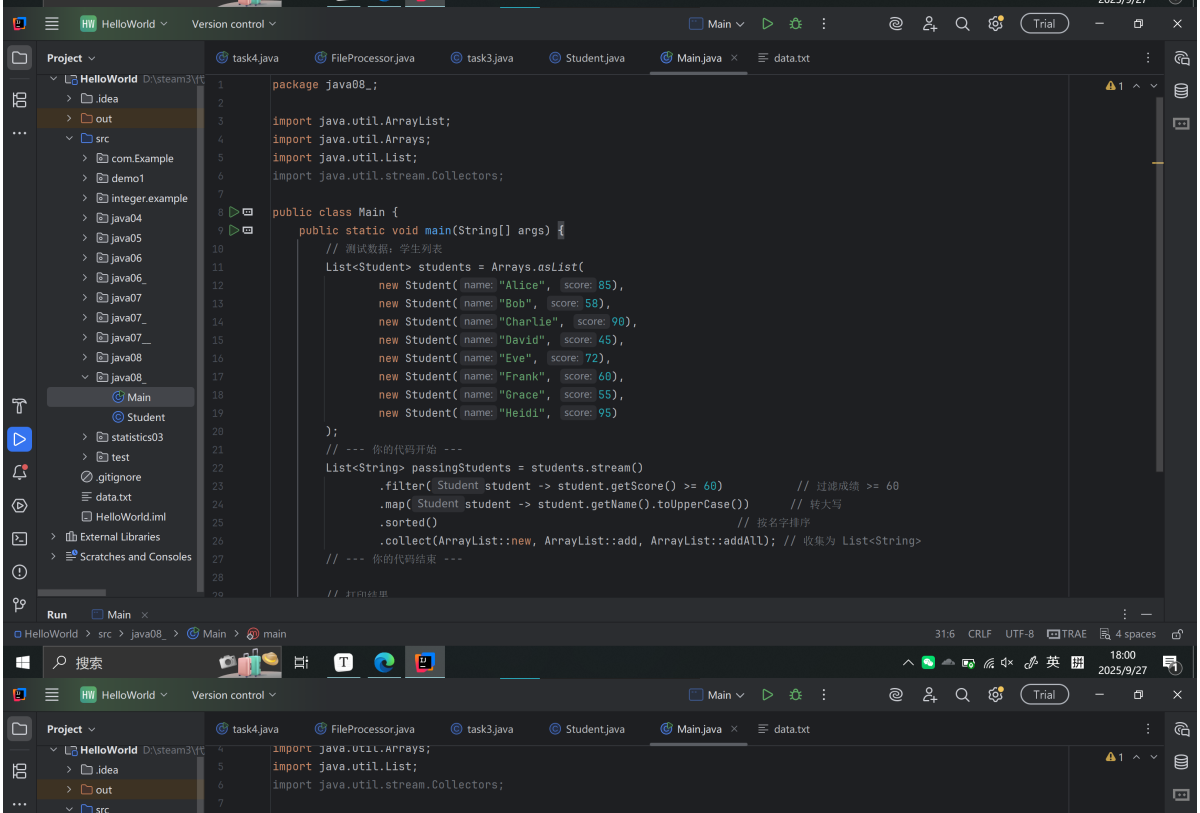
Q5. 完成下述题目

定义一个学生类、初始代码

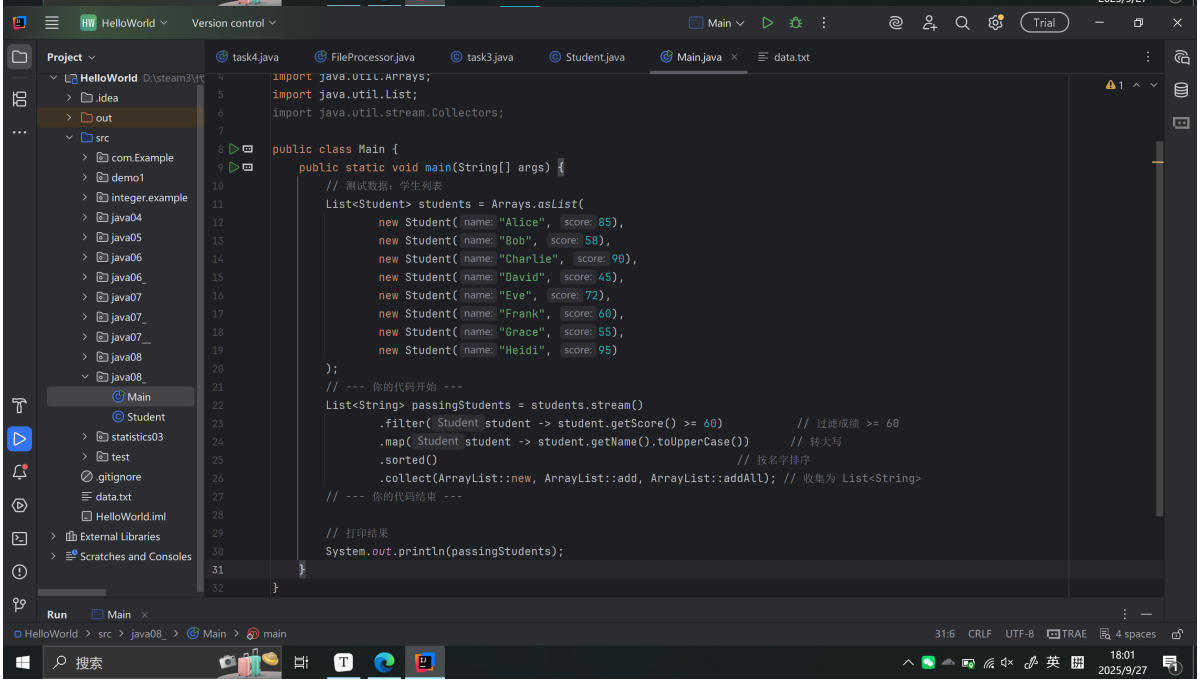


```java
package java08_;

public class Student {
    String name;
    int score;

    public Student(String name, int score) {
        this.name = name;
        this.score = score;
    }

    public String getName() {
        return name;
    }

    public int getScore() {
        return score;
    }

    public void setName(String name) {
        this.name = name;
    }
```

```
C:\Users\Admin\.jdks\openjdk-24.0.2+12-54\bin\java.exe "-javaagent:D:\steam2\IntelliJ IDEA 2025.2.1\lib\idea_rt.jar=54962" -Dfile.encoding=UTF-8 -Dsun.stdout.encodi
[ALICE, CHARLIE, EVE, FRANK, HEIDI]

Process finished with exit code 0
```



```java
package java08_;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class Main {
    public static void main(String[] args) {
        // 测试数据：学生列表
        List<Student> students = Arrays.asList(
                new Student( name: "Alice",  score: 85),
                new Student( name: "Bob",  score: 58),
                new Student( name: "Charlie",  score: 90),
                new Student( name: "David",  score: 45),
                new Student( name: "Eve",  score: 72),
                new Student( name: "Frank",  score: 60),
                new Student( name: "Grace",  score: 55),
                new Student( name: "Heidi",  score: 95)
        );
        // --- 你的代码开始 ---
        List<String> passingStudents = students.stream()
                .filter( Student student -> student.getScore() >= 60)          // 过滤成绩 >= 60
                .map( Student student -> student.getName().toUpperCase())      // 转大写
                .sorted()                                                      // 按名字排序
                .collect(ArrayList::new, ArrayList::add, ArrayList::addAll); // 收集为 List<String>
        // --- 你的代码结束 ---
```



```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class Main {
    public static void main(String[] args) {
        // 测试数据：学生列表
        List<Student> students = Arrays.asList(
                new Student( name: "Alice",  score: 85),
                new Student( name: "Bob",  score: 58),
                new Student( name: "Charlie",  score: 90),
                new Student( name: "David",  score: 45),
                new Student( name: "Eve",  score: 72),
                new Student( name: "Frank",  score: 60),
                new Student( name: "Grace",  score: 55),
                new Student( name: "Heidi",  score: 95)
        );
        // --- 你的代码开始 ---
        List<String> passingStudents = students.stream()
                .filter( Student student -> student.getScore() >= 60)          // 过滤成绩 >= 60
                .map( Student student -> student.getName().toUpperCase())      // 转大写
                .sorted()                                                      // 按名字排序
                .collect(ArrayList::new, ArrayList::add, ArrayList::addAll); // 收集为 List<String>
        // --- 你的代码结束 ---

        // 打印结果
        System.out.println(passingStudents);
    }
}
```