

task1

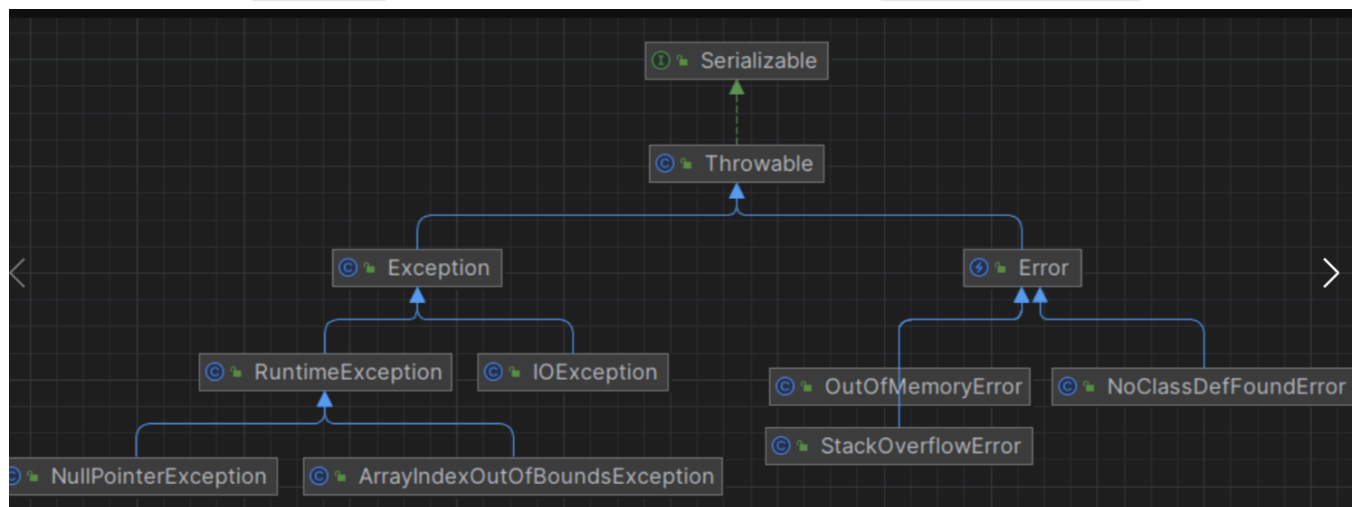
Q1

Error 通常无法通过代码来处理或恢复；而 Exception 表示可处理的问题，程序可以通过捕获和处理异常来进行恢复或者报告问题。

Errors 不应该被程序显式地处理，而是应该通过调整系统环境或者程序代码来解决；Exceptions 则是可以被捕获和处理的，通常通过异常处理机制来维护程序的健壮性和可靠性。

`RuntimeException` 是 `java.lang` 包中的一个异常类，它是所有未检查异常（unchecked exceptions）的父类。这些异常通常表示编程错误或逻辑错误，如空指针引用、数组下标越界、类型转换错误等，编译器不会强制要求你显式处理这些异常，因此你可以选择捕获它们，但并非必须

checked exception 是受检异常，这意味着在使用可能会抛出 `IOException` 的方法时，编译器会强制要求进行异常处理，要么通过 `try-catch` 块捕获异常，要么在方法上声明抛出异常（`throws IOException`）



Q2

Error 是由于系统内部错误、资源耗尽、虚拟机问题或其他不可恢复的环境问题引起的严重问题；Exception 是在程序运行过程中可能由于外部输入、错误的配置、不完整的代码逻辑或其他可预见的问题引起的问题的信号

Task2

想读取文件，无法运行

```
Path path = Paths.get("C:\\Users\\ZZF\\IdeaProjects\\Java02\\src\\com\\Example\\data\\data.text");
String data = Files.readString(path);
System.out.println(data);
```

问题： `Files.readString` 方法会抛出 `IOException`，需要用 `try-catch` 块进行捕获处理，防止程序因 IO 异常直接崩溃

调整后

```

public static void main(String[] args) {
    try {
        Path path =
Paths.get("C:\\Users\\ZZF\\IdeaProjects\\Java02\\src\\com\\Example\\data\\test.txt");
        String data = Files.readString(path);
        System.out.println(data);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

但无法读取数据

最终版

```

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Scanner;

// 自定义异常FileNotFoundException: 文件未找到
class FileNotFoundException extends Exception {
    public FileNotFoundException(String message) {
        super(message);
    }
}

// 自定义异常EmptyFileException: 文件为空
class EmptyFileException extends Exception {
    public EmptyFileException(String message) {
        super(message);
    }
}

public class DataAverage {
    public static void main(String[] args) throws EmptyFileException, FileNotFoundException
    {
        String data =
"C:\\Users\\ZZF\\IdeaProjects\\Java02\\src\\com\\Example\\data\\data.text";
        BufferedReader br = null;

        try {
            // 尝试创建文件读取流
            FileReader fr = new FileReader(data);
            br = new BufferedReader(fr); // 或者简化成 reader = new BufferedReader(new
FileReader(fileName));

            String line;

```

```
double sum = 0;
int count = 0;
boolean isEmpty = true;
```

```
while ((line = br.readLine()) != null) {//while 循环持续读取每行数据，直到文件末尾
    isEmpty = false;
    line = line.trim();
```

/*为什么必须先读行、再判断空？

只有先遍历到每一行，才能针对性判断该行是否需要处理，最终实现 “只解析非空行、跳过无效行”*/

```
    if (!line.isEmpty()) {
        try {
            int number = Integer.parseInt(line);
            sum += number;
            int i = count++;
            System.out.println("读取到整数: " + number);
        } catch (NumberFormatException e) {//如果解析失败，就会进入这里执行错误处理逻辑。
```

```
            System.err.println("格式错误: 无法将 \"" + line + "\" 转换为整数");
```

```
        }
```

```
    }
```

```
}
```

// 如果文件为空，抛出异常

```
if (isEmpty) {
    throw new EmptyFileException("文件为空");
}
```

```
double average = sum / count ;
System.out.println("文件读取完成，有效整数个数: " + count);
System.out.println("平均值为: " + average);
```

```
} catch (java.io.FileNotFoundException e) {
    // 捕获系统的FileNotFoundException并转换为自定义异常
    throw new FileNotFoundException("文件未找到: " + data);
} catch (IOException e) {
    System.err.println("文件读取错误: " + e.getMessage());
} finally {
    // 确保资源被关闭
    if (br != null) {
        try {
            br.close();
            System.out.println("文件资源已关闭");
        } catch (IOException e) {
            System.err.println("关闭文件时发生错误: " + e.getMessage());
        }
    }
}
}
```

```
va MockSongs.java data.text DataAverage.java x Repository.java MyRepository.java Test.java v
1 package com.Example.data;
2
3 import java.io.BufferedReader;
4 import java.io.FileReader;
5 import java.io.IOException;
6 import java.nio.file.Files;
7 import java.nio.file.Path;
8 import java.nio.file.Paths;
9 import java.util.Scanner;
10
11 // 自定义异常FileNotFoundException: 文件未找到
12 class FileNotFoundException extends Exception { 2个用法
13     public FileNotFoundException(String message) { 1个用法
14         super(message);
15     }
16 }
17
18 // 自定义异常EmptyFileException: 文件为空
19 class EmptyFileException extends Exception { 2个用法
20     public EmptyFileException(String message) { 1个用法
21         super(message);
22     }
23 }
24
25 public class DataAverage {
26     public static void main(String[] args) throws EmptyFileException, FileNotFoundException {
27         String data = "C:\\Users\\77F\\IdeaProjects\\Java02\\src\\com\\Example\\data\\data.text";
28     }
29 }

运行 DataAverage x
"C:\Program Files\Java\jdk-24\bin\java.exe" --enable-preview "-javaagent:D:\IntelliJ IDEA 2025.2.1\lib\idea_rt.jar"
读取到整数: 1
读取到整数: 3
读取到整数: 1
读取到整数: 4
文件读取完成, 有效整数个数: 4
平均值为: 2.25
文件资源已关闭
进程已结束, 退出代码为 0
```

Q4

会出现`limit = java.util.stream.SliceOps$1@xxxxxx`

出现这种结果是因为:

Stream 的惰性执行特性: `limit(4)` 是 Stream 的**中间操作**, 中间操作不会立即执行, 只会记录对 Stream 的处理逻辑, 不会生成最终结果。

直接打印 Stream 对象: `System.out.println(limit)` 本质是调用 Stream 对象的 `toString()` 方法, 输出的是该 Stream 实现类的“类名 + 哈希码”, 而非 Stream 中的元素内容。

//中间操作

`filter(Predicate<T>)`: 过滤元素

`map(Function<T, R>)`: 转换元素类型

`sorted()` / `sorted(Comparator<T>)`: 排序

`distinct()`: 去重

`limit(n)`: 取前 n 个元素

`skip(n)`: 跳过前 n 个元素

终端操作

- `forEach(Consumer<T>)`: 遍历
- `collect(Collectors.toList())` / `toSet()` / `toMap()`: 收集结果
- `count()`: 计数
- `min(Comparator<T>)` / `max(Comparator<T>)`: 求最小 / 最大值
- `anyMatch(Predicate<T>)` / `allMatch(...)` / `noneMatch(...)`: 匹配判断
- `findFirst()` / `findAny()`: 查找元素//

Q5//思路: 获得 — 筛选 — 排序 — 提取 — 收集

```
package com.Example.Student;

import java.util.List;
import java.util.function.Function;
import java.util.stream.Collectors;
import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        // 测试数据: 学生列表
        List<Student> students = Arrays.asList(
            new Student("Alice", 85),
            new Student("Bob", 58),
            new Student("Charlie", 90),
            new Student("David", 45),
            new Student("Eve", 72),
            new Student("Frank", 60),
            new Student("Grace", 55),
            new Student("Heidi", 95)
        );
        List<String> passingStudents = students.stream()
            .filter(student -> student.getScore() >= 60) // 1. 过滤分数≥60的学生

            .map(Student::getName)//对于对象不能直接调用String应先提取名字
            .map(String::toUpperCase)// 2. 姓名转换成大写

            .sorted()// 3. 按姓名字母顺序排序

            .collect(Collectors.toList());
        System.out.println(passingStudents); // 4. 收集成 List<String> 返回并打印
```

```
}  
}
```

The screenshot displays the IntelliJ IDEA IDE interface. The top toolbar includes icons for file operations, running, debugging, and search. The main editor window shows three files: Student.java, Main.java, and DataAverage.java. The Main.java file is open, displaying the following Java code:

```
10 public class Main {  
11     public static void main(String[] args) {  
19         new Student( name: "Frank", score: 60),  
20         new Student( name: "Grace", score: 55),  
21         new Student( name: "Heidi", score: 95)  
22     };  
23     List<String> passingStudents = students.stream()  
24         .filter( Student student -> student.getScore() >= 60) // 1. 过滤分数≥60的学生  
25  
26         .map(Student::getName)//对于对象不能直接调用String应先提取名字  
27         .map(String::toUpperCase)// 2. 姓名转换成大写  
28  
29         .sorted();// 3. 按姓名字母顺序排序  
30  
31         .collect(Collectors.toList());  
32     System.out.println(passingStudents); // 4. 收集成 List<String> 返回并打印  
33  
34 }  
35 }  
36 }  
37
```

Below the code editor, the '运行' (Run) tab is active, showing the execution of the program. The command executed is:

```
"C:\Program Files\Java\jdk-24\bin\java.exe" --enable-preview "-javaagent:D:\IntelliJ IDEA 20
```

The output of the program is:

```
[ALICE, CHARLIE, EVE, FRANK, HEIDI]
```

The status bar at the bottom indicates that the process has ended with an exit code of 0.