Java08

Task1

Q1

Error与Exception的区别

	Error（错误）	Exception（异常）
严重性	严重问题，是应用程序不应尝试捕获和处理的。	非严重问题，是应用程序应该并且可以捕获和处理的。
产生原因	通常源于虚拟机本身的问题，是系统内部错误或资源耗尽等底层问题。例如： `OutOfMemoryError`（内存耗尽） `StackOverflowError`（栈溢出） `VirtualMachineError`（虚拟机错误）	通常源于应用程序本身的问题，是程序代码可以预料到的、可修复的问题。例如： `FileNotFoundException`（文件找不到） `IOException`（网络连接中断） `NullPointerException`（空指针访问）

程序的处理态度

- `Error`：程序无法处理，通常会导致JVM终止运行。
- `Exception`：程序应该处理，使用 `try-catch` 块捕获，或者向上抛出。

Q2

unckecked异常与ckecked异常的区别

	CheckedException	UncheckedException（或RuntimeException）
是否在编译期检查	是。编译器会检查方法是否声明或处理了这些异常。如果不处理，编译失败。	否。编译器不强制要求方法捕获或声明它们。代码即使有潜在抛出风险也能通过编译。
处理强制性	必须处理。两种方式：1.使用 try-catch 块捕获并处理。2.在方法签名上用 throws 关键字声明抛出。	非必须处理。可以处理，但通常不推荐捕获（除非在最上层统一处理），而应该通过修正代码逻辑来避免。
设计理念	可恢复的异常。表示程序在正常操作下可以预料到并可能恢复的问题。	程序错误。表示程序逻辑上的错误，通常不应该被捕获后恢复，而应该修复代码。
代码可读性	方法签名明确指出了可能抛出的异常，调用者一目了然。	调用者需要查阅文档或代码才能知道可能抛出哪些运行时异常。

发生的原因

- **CheckedException**：通常由**外部因素**导致，这些因素不完全在程序员的直接控制之下，但在程序正常运行时它们有可能发生。**例如**：程序依赖于外部环境，而该环境可能不满足要求。当程序尝试打开一个不存在的文件时，会抛出`FileNotFoundException`。
- **UncheckedException**：通常由程序内部的逻辑错误引起，它们通常意味着代码本身有bug。**例如**：当访问数组时下标越界时，会发生`ArrayIndexOutOfBoundsException`。

Task2

Q3

```
import java.io.*;

class FileNotFoundException extends Exception{
    public FileNotFoundException(String message){
        super(message);
    }
}
class EmptyFileException extends Exception{
    public EmptyFileException(String message){
        super(message);
    }
}

public class java08_Q3 {
    public static void main(String[] args){
        File src=new File("data.txt");
        double average;
        try {
            average = getAverage(src);
            System.out.println("average=" + average);
        }catch (NumberFormatException e){
            System.out.println("文件"+src.getName()+"中包含非整数");
            System.out.println(e.getMessage());
        }catch (Exception e){
            System.out.println(e.getMessage());
        }

    }
    private static double getAverage(File src) throws FileNotFoundException,
EmptyFileException, NumberFormatException, IOException{
        if(!src.exists()){
            throw new FileNotFoundException("文件"+src.getName()+"未找到");
        }
        if(src.length()==0){
            throw new EmptyFileException("文件"+src.getName()+"为空");
        }
        try(BufferedReader br=new BufferedReader(new InputStreamReader(new
FileInputStream(src)))){
            //try-with-resources
            String line;
            int sum=0,count=0;
            while((line=br.readLine())!=null){
```

```
                //默认：文件中不包含空行与空格
                int num=Integer.parseInt(line);
//                public static int parseInt(String s) throws
NumberFormatException {
//                    return parseInt(s, 10);
//                }
                //若无法将数据解析为整数，该方法会抛出NumberFormatException
                sum+=num;
                count++;
            }
            return (double) sum/count;
        } catch (IOException e) {
            System.out.println("IOException");
            throw e;
        }
    }
}
```

Task3

Q4

运行代码后控制台输出的结果：

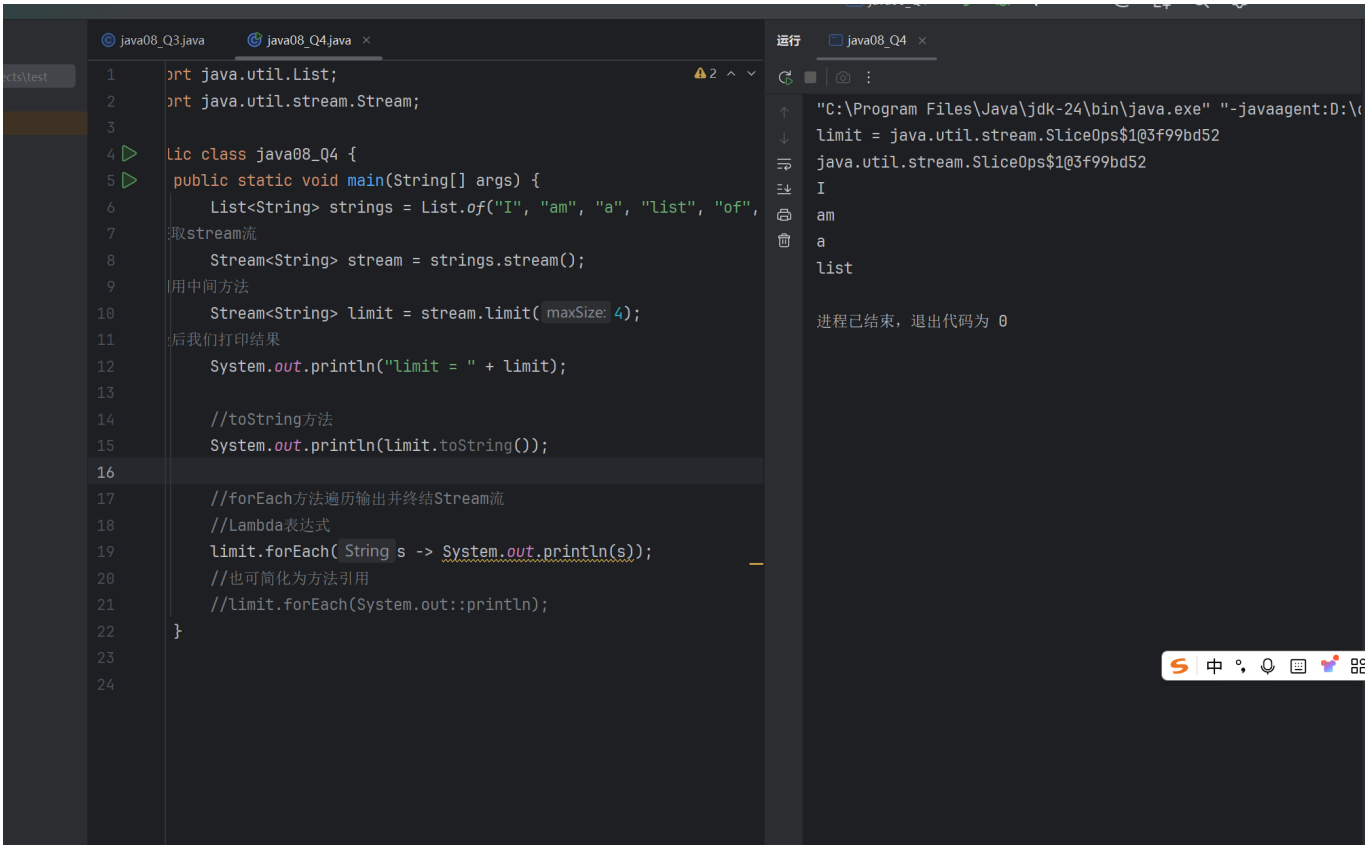
```
limit = java.util.stream.SliceOps$1@3f99bd52
```

出现这种结果的原因

```
//PrintStream.java
public void println(String x) {
        if (getClass() == PrintStream.class) {
            writeln(String.valueOf(x));
        } else {
            synchronized (this) {
                print(x);
                newLine();
            }
        }
    }
public void println(Object x) {
        String s = String.valueOf(x);
        if (getClass() == PrintStream.class) {
            // need to apply String.valueOf again since first invocation
            // might return null
            writeln(String.valueOf(s));
        } else {
            synchronized (this) {
```

```
                print(s);
                newLine();
            }
        }
    }

//String.java
public static String valueOf(Object obj) {
        return (obj == null) ? "null" : obj.toString();
    }
```

从println和valueOf的代码可以推测出：直接用System.out.println("limit = " + limit);输出limit时，会调用toString方法。要想输出limit中储存的信息，需要调用forEach方法进行遍历并输出。如图所示：



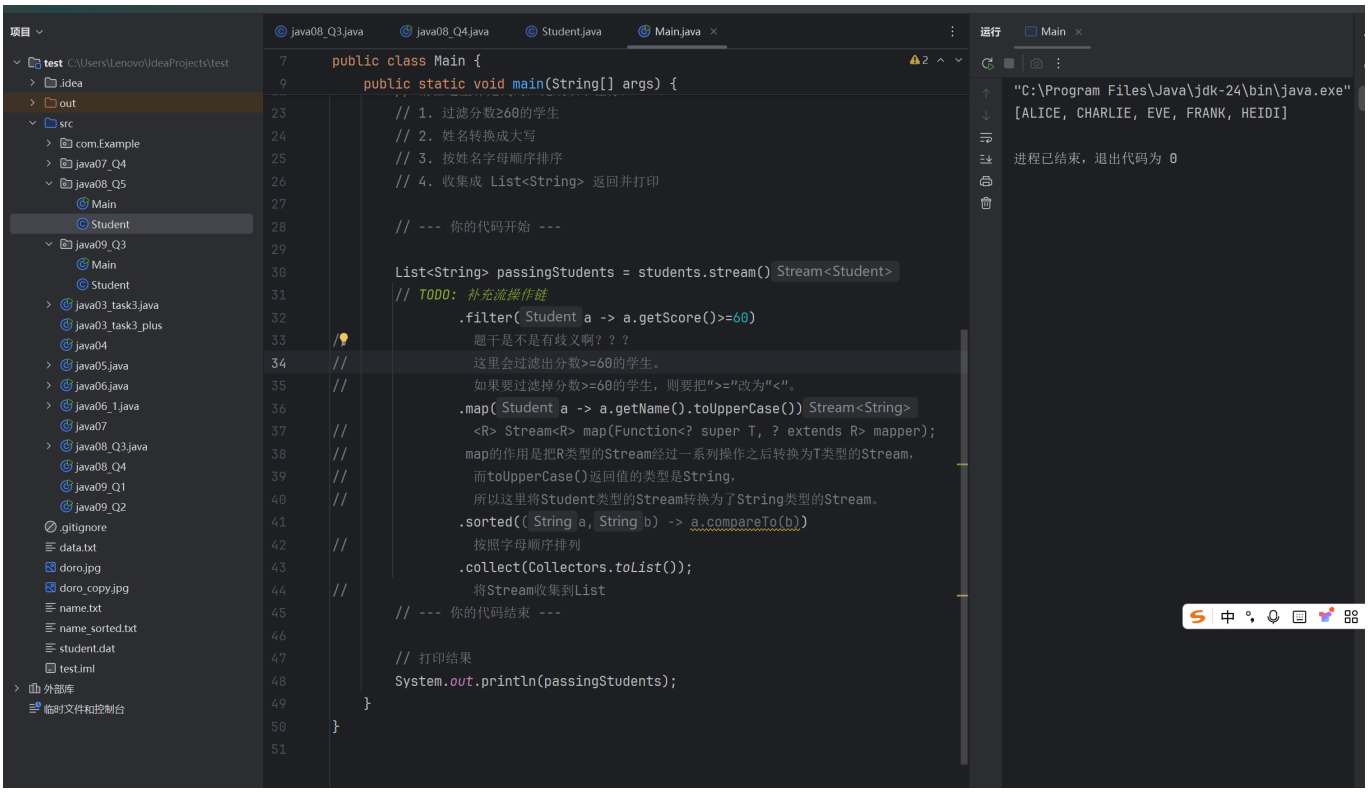
Q5

代码：

```
package java08_Q5;

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("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)
        );

        // 请在这里补充代码，完成以下任务：
        // 1. 过滤分数≥60的学生
        // 2. 姓名转换成大写
        // 3. 按姓名字母顺序排序
        // 4. 收集成 List<String> 返回并打印

        // --- 你的代码开始 ---

        List<String> passingStudents = students.stream()
        // TODO: 补充流操作链
                .filter(a -> a.getScore()>=60)
//                题干是不是有歧义啊？？？
//                这里会过滤出分数>=60的学生。
//                如果要过滤掉分数>=60的学生，则要把“>=”改为“<”。
                .map(a -> a.getName().toUpperCase())
//                <R> Stream<R> map(Function<? super T, ? extends R> mapper);
//               map的作用是把R类型的Stream经过一系列操作之后转换为T类型的Stream，
//                而toUpperCase()返回值的类型是String，
//                所以这里将Student类型的Stream转换为了String类型的Stream。
                .sorted((a,b) -> a.compareTo(b))
//                按照字母顺序排列
                .collect(Collectors.toList());
//                将Stream收集到List
        // --- 你的代码结束 ---

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

运行结果截图：


项目
test C:\Users\Lenovo\IdeaProjects\test
.idea
out
src
com.Example
java07_Q4
java08_Q5
Main
Student
java09_Q3
Main
Student
java03_task3.java
java03_task3_plus
java04
java05.java
java06.java
java06_1.java
java07
java08_Q3.java
java08_Q4
java09_Q1
java09_Q2
.gitignore
data.txt
doro.jpg
doro_copy.jpg
name.txt
name_sorted.txt
student.dat
test.iml
外部库
临时文件和控制台
java08_Q3.java
java08_Q4.java
Student.java
Main.java
public class Main {
public static void main(String[] args) {
// 1. 过滤分数≥60的学生
// 2. 姓名转换成大写
// 3. 按姓名字母顺序排序
// 4. 收集成 List<String> 返回并打印
// --- 你的代码开始 ---
List<String> passingStudents = students.stream() Stream<Student>
// TODO: 补充流操作链
.filter(Student a -> a.getScore()>=60)
题干是不是有歧义啊？？？
这里会过滤出分数>=60的学生。
如果要过滤掉分数>=60的学生，则要把“>=”改为“<”。
.map(Student a -> a.getName().toUpperCase()) Stream<String>
<R> Stream<R> map(Function<? super T, ? extends R> mapper);
map的作用是把R类型的Stream经过一系列操作之后转换为T类型的Stream，
而toUpperCase()返回值的类型是String，
所以这里将Student类型的Stream转换为了String类型的Stream。
.sorted((String a, String b) -> a.compareTo(b))
按照字母顺序排列
.collect(Collectors.toList());
将Stream收集到List
// --- 你的代码结束 ---
// 打印结果
System.out.println(passingStudents);
}
}
运行
Main
"C:\Program Files\Java\jdk-24\bin\java.exe"
[ALICE, CHARLIE, EVE, FRANK, HEIDI]
进程已结束，退出代码为 0