

**2019年春季学期  
计算机学院《软件构造》课程**

**Lab 4实验报告**

|  |  |
| --- | --- |
| 姓名 | 郭子阳 |
| 学号 | 1170300520 |
| 班号 | 1703005 |
| 电子邮件 | [guoziyang0033@gmail.com](mailto:guoziyang0033@gmail.com) |
| 手机号码 | 18800420598 |

**目录**

[1 实验目标概述 1](#_Toc3922818)

[2 实验环境配置 1](#_Toc3922819)

[3 实验过程 1](#_Toc3922820)

[3.1 Error and Exception Handling 2](#_Toc3922821)

[3.2 Assertion and Defensive Programming 3](#_Toc3922822)

[3.2.1 checkRep()检查invariants 3](#_Toc3922823)

[3.2.2 Assertion保障pre-/post-condition 3](#_Toc3922824)

[3.3 Logging 3](#_Toc3922825)

[3.3.1 写日志 3](#_Toc3922826)

[3.3.2 日志查询 4](#_Toc3922827)

[3.4 Testing for Robustness and Correctness 5](#_Toc3922828)

[3.4.1 Testing strategy 5](#_Toc3922829)

[3.4.2 测试用例设计 5](#_Toc3922830)

[3.4.3 测试运行结果与EclEmma覆盖度报告 5](#_Toc3922831)

[3.5 SpotBugs tool 5](#_Toc3922832)

[3.6 Debugging 6](#_Toc3922833)

[3.6.1 理解待调试程序的代码思想 6](#_Toc3922834)

[3.6.2 发现并定位错误的过程 6](#_Toc3922835)

[3.6.3 如何修正错误 6](#_Toc3922836)

[3.6.4 结果 7](#_Toc3922837)

[4 实验进度记录 7](#_Toc3922838)

[5 实验过程中遇到的困难与解决途径 7](#_Toc3922839)

[6 实验过程中收获的经验、教训、感想 8](#_Toc3922840)

[6.1 实验过程中收获的经验和教训 8](#_Toc3922841)

[6.2 针对以下方面的感受 8](#_Toc3922842)

# 实验目标概述

本次实验重点训练学生面向健壮性和正确性的编程技能，利用错误和异常处理、断言与防御式编程技术、日志/断点等调试技术、黑盒测试编程技术，使程序可在不同的健壮性/正确性需求下能恰当的处理各种例外与错误情况，在出错后可优雅的退出或继续执行，发现错误之后可有效的定位错误并做出修改。

实验针对 Lab 3 中写好的 ADT 代码和基于该 ADT 的三个应用的代码，使用以下技术进行改造，提高其健壮性和正确性：

* 错误处理
* 异常处理
* Assertion和防御式编程
* 日志
* 调试技术
* 黑盒测试及代码覆盖度

# 实验环境配置

实验环境：

系统：macOS Mojave 10.14.3，Oracle JDK 1.8.0\_211

ide：Eclipse IDE for Eclipse Committers 2018-12 (4.10.0), IntelliJ IDEA ULTIMATE 2019.1

编辑器：Visual Studio Code 1.31.1

构建工具：Gradle，Travis CI，Drone CI

版本管理：git

代码托管：Github, Gogs

配置过程：

本次实验基于Lab3，实验环境无需配置，安装少量插件即可

GitHub Lab4仓库的URL地址

https://github.com/ComputerScienceHIT/Lab4-1170300520

# 实验过程

## Error and Exception Handling

创建Exception包，并创建ConfigurationException异常基类继承RuntimeException。建立了三个异常类继承了ConfigurationException：ConfigurationSyntaxException, ConfigurationLabelException, ConfigurationDependencyException，分别代表语法异常、标签异常和依赖异常。

ConfigurationSyntaxException，语法异常，指的是配置文件中存在不符合语法规范的情况。具体的分类有：应使用科学记数法时（>10000）却未使用，不应使用科学记数法时（<= 10000）却使用了，文件中存在无效的标签，日期格式不符合规定，AtomicStructure的名称不符合规范，角度大于360度或小于等于0度以及科学记数法使用不规范等情况。

ConfigurationLabelException，标签异常，指的是在配置文件中存在多个同类标签代表同一个物体的情况，如StellarSystem中存在多个相同名称的Planet，StellarSystem中定义了多个Stellar，AtomicStructure中定义了多个Name，PersonalAppEcosystem中存在多个相同名称的App等。

ConfigurationDependencyException，依赖异常，指的是配置文件中存在依赖关系错误的情况，如AtomicStructure中轨道个数和实际轨道个数不一致，PersonalAppEcosystem中App还未安装就已使用或卸载等情况。

所有的Exception都在捕捉后输出到日志，并向上抛出，最终在application中，使用基类ConfigurationException捕捉处理，并要求用户重新输入一个文件路径。

所有输出的错误信息都包含了错误发生的文件名以及配置文件行数。原本的配置文件解析方式是将整个配置文件组合成单个String，并且用正则表达式去匹配整个文件。这种方式不利于错误的发现（如无效的标签就无法发现），已经修改为对每行配置进行匹配，及时发现异常。

## Assertion and Defensive Programming

### checkRep()检查invariants

对每一个类书写checkRep()方法，对一些不变量进行检查，如判空等，并在每次方法调用前后调用checkRep()方法检查。

### Assertion保障pre-/post-condition

在方法调用前后使用assert断言保证调用前后环境正确即可。

## Logging

### 写日志

日志系统使用的是slf4j+log4j2组合，slf4j框架用于统一日志系统接口。直接在src下书写log4j2的配置文件log4j2.xml：

<?xml version="1.0" encoding="UTF-8"?>

<Configuration status="OFF">

<Appenders>

<Console name="Console" target="SYSTEM\_OUT">

<PatternLayout pattern="%highlight{%d{yyyy-MM-dd} %d{HH:mm:ss} %-5level %-27C{1} - [%M] %msg%n}"/>

</Console>

</Appenders>

<Loggers>

<Root level="ALL">

<AppenderRef ref="Console" level="ALL"/>

</Root>

</Loggers>

</Configuration>

其中，PatternLayout标签的pattern属性指定类日志的输出格式，包括日期、时间、日志等级、日志产生类、日志产生方法和日志信息，highlignt是为了在支持ANSI字符的控制台输出彩色日志。配置文件只定义了一个日志Appender，输出到控制台，输出级别为ALL，并关闭slf4j的日志输出。

在需要输出日志的类中新建对象：Logger logger = LoggerFactory.getLogger(类名.class)，获取当前类的日志句柄。并使用logger.info()等方法输出不同等级的日志。

### 日志查询

日志查询显然不是简单地输出到文件，关闭JVM后再手动去查询，而是需要在运行时读取输出的日志。由于日志系统只支持输出到控制台或文件，而控制台的日志无法读取，于是将日志输出到一个文件中，并读取即可，**可能**会造成类似数据库的不可重复读的问题（日志系统与主线程是两个线程），但是问题**应该不大**。

希望做到每次启动的日志文件输出在单独的文件中，而不是所有的输出都在同一个文件里（网上大多数教程），我将log加上了启动时的时间作为日志文件名，由于时间是动态产生的，无法在配置文件中配置，于是在运行时动态配置这个日志Appender。

LoggerContext ctx = (LoggerContext) LogManager.getContext(false);

        org.apache.logging.log4j.core.config.Configuration config = ctx.getConfiguration();

        PatternLayout layout = PatternLayout.newBuilder().withCharset(Charset.forName("UTF-8"))

                .withPattern("%d{yyyy-MM-dd} %d{HH:mm:ss} %-5level %-27C{1} - [%M] %msg%n").withConfiguration(config)

                .build();

        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd\_HH-mm-ss");

        logFileName = "logs/log\_" + sdf.format(new Date()) + ".log";

        Appender appender = FileAppender.newBuilder().setName("MyRollingFile").withImmediateFlush(true)

                .setLayout(layout).withFileName(logFileName).build();

        appender.start();

        config.addAppender(appender);

        config.getRootLogger().addAppender(appender, Level.ALL, null);

        ctx.updateLoggers(config);

这段代码读取配置文件并动态添加Appender，在每次启动时添加的Appender的fileName都不同，即可输出到不同的文件。

在运行时读取日志文件，考虑到配置文件可能比较大的问题，如果每次读取都从头读取，效率太低，于是使用RandomAccessFile类读取，该类可以从文件的指定位置开始读取，只需要在每次读取完成后记录一下读取的进度，即可在下次读取时从该位置继续读取。

过滤时只需要通过正则表达式简单匹配就可以了。

## Testing for Robustness and Correctness

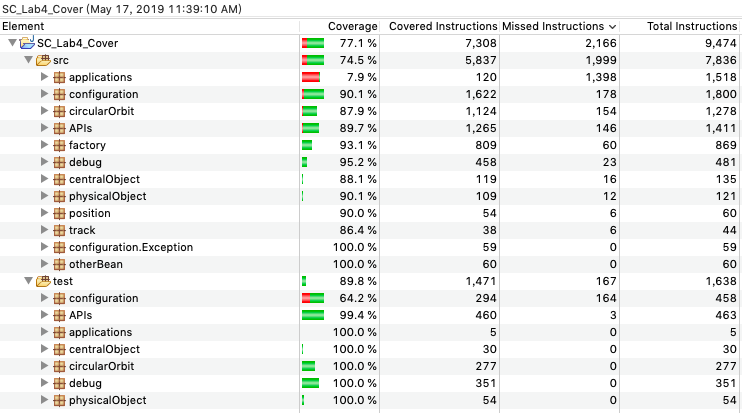
### Testing strategy

使用各种可能出现的输入序列对方法进行测试，以检验其健壮性。使用合法和非法数据的边界值测试方法，判断是否可以对边界情况进行检查。

### 测试用例设计

对主要的自定义异常都书写了相应的输入文件用于测试，并输入程序检测程序是否可以妥善处理。

### 测试运行结果与EclEmma覆盖度报告



由于GUI、application以及日志读取无法测试，导致覆盖度偏低，为77.1%。

EclEmma覆盖度报告保存在与本报告同目录下。

## SpotBugs tool

由于Java的自动装箱拆箱的特性，比较两个包装类型的数字的大小不需要手动拆箱，直接使用比较运算即可。

可使用try-with-resource方式直接释放流资源，而不需要try-catch后手动释放。

switch中嵌套switch不可以忘记break。

## Debugging

### 理解待调试程序的代码思想

FindMedianSortedArrays，寻找两个整型数组的中位数，使用二分法查找。

RemoveComments，用于在一段代码字符串数组中，删去所有的行间注释和块注释。

TopVotedCandidate，用于计算投票时的某一时刻的领先选手，使用了二维链表。

### 发现并定位错误的过程

理解题意，阅读代码，断点调试

### 如何修正错误

FindMedianSortedArrays：第31行处，计算halfLen时，应当向上取整，而现在的计算方法为向下取整，故改为

int iMin = 0, iMax = m, halfLen = (m + n + 1) / 2;

第49行处，if判断m+n是否为奇数，而不是判断偶数，应改为

if ((m + n) % 2 == 1) {

RemoveComments：while循环判断每一行时，缺少了行间注释（//）和当前处于块注释中的情况，添上即可。

TopVotedCandidate：主要产生了一些边界错误，如加一不加一等错误。

### 结果

# 实验进度记录

请使用表格方式记录你的进度情况，以超过半小时的连续编程时间为一行。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 2019-5-6 | 15:00-20:00 | 完成Exception并重构reader | 完成 |
| 2019-5-12 | 14:00-17:00 | 完成了异常的拦截 | 完成 |
| 2019-5-12 | 17:00-22:00 | 优化了AppEcosystem的速度 | 完成 |
| 2019-5-15 | 13:30-15:00 | 完成了日志的读取过滤 | 完成 |
| 2019-5-15 | 19:00-20:00 | 完成debug | 完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| PersonalAppEcosystem读取速度太慢（最大文件15s） | 优化了PersonalAppEcosystem的构建逻辑，减少读取时间（9s） |
| 日志框架的配置文件无法动态生成日志文件名 | 在运行时动态配置，而不在配置文件中配置 |

# 实验过程中收获的经验、教训、感想

## 实验过程中收获的经验和教训

学会了日志框架的使用，正在写入的文件读取，异常的定义与处理等

## 针对以下方面的感受

过分的防御并不是很必要，甚至有可能得不偿失，在显著提升程序性能的情况下，应当给予用户一定的信任，相应用户可以按照规约输入而不是恶意破坏。

实验较上次简单，时间宽松。