



哈爾濱工業大學  
HARBIN INSTITUTE OF TECHNOLOGY

# Lab Manuals for Software Construction

## Lab-4 Debugging, Exception Handling, and Defensive Programming



School of Computer Science and Technology

Harbin Institute of Technology

Spring 2019

## 目录

1. 实验目标.....	1
2. 实验环境.....	1
3. 实验要求.....	2
3.1. Error and Exception Handling .....	2
3.2. Assertion and Defensive Programming .....	3
3.3. Logging .....	4
3.4. Testing for Robustness and Correctness.....	4
3.5. Using SpotBugs tool.....	5
3.6. Debugging .....	5
4. 实验报告.....	6
5. 提交方式.....	7
6. 评分方式.....	7

## 1. 实验目标

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

## 2. 实验环境

实验环境设置请参见 Lab-0 实验指南。

除此之外，本次实验需要你在 Eclipse IDE 中安装配置 SpotBugs（用于 Java 代码静态分析的工具）。请访问 <https://spotbugs.github.io>，了解它并学习其安装、配置和使用。

本次实验在 GitHub Classroom 中的 URL 地址为：

<https://classroom.github.com/a/mnFCZ30Y>

请访问该 URL，按照提示建立自己的 Lab4 仓库并关联至自己的学号。

由于本次实验是 Lab3 的继续，请首先将你 Lab3 的最终代码复制一份，推送到 Lab4 的仓库里，然后在其基础上完成本次实验任务。本次实验不能影响 Lab3 和 Lab3 仓库里的内容与 commit 历史。

本地开发时，本次实验只需建立一个项目，统一向 GitHub 仓库提交。实验包含的多项任务分别在不同的目录内开发，具体目录组织方式参见各任务最后一部分的说明。请务必遵循目录结构，以便于教师/TA 进行测试。

## 3. 实验要求

### 3.1. Error and Exception Handling

针对各种 `checked exception`，相信你在 Lab 3 的程序中均已经被“强制”考虑过了。本节通过自定义 `Exception` 类型，表示程序的一些常见错误类型，为代码可能发生的问题提供新的含义，区分代码运行时可能出现的相似问题，或给出程序中一组共性错误的特殊含义。

考虑以下各个场景，使用 `exception` 和常见的错误机制来处理它们，尽可能 `fail fast`。为每种错误类型建立新的 `checked` 异常类，在 Lab3 的代码中增加错误和异常处理代码，在特定函数上声明异常（`throws`）、抛出异常（`throw`）、捕获并处理异常（`try-catch-finally`、`try-with-resources`）。

- 输入文件中存在不符合语法规则的语句，例如某个元素定义的标签非法、元素定义的内容格式与语法规则不一致（例如超过 10000 的数字未使用科学计数法、需要两位小数但使用了 1 位或 3 位小数、要求三位大写字母但却使用了非大写字母等）、元素定义中分量的数目或次序与规范不符合（例如运动员信息需要 5 个分量但只提供了 4 个、运动员的姓名和号码颠倒次序等）等。这里不一一列举。针对你所开发的 3 个应用中的每一个应用，要能够考虑至少 5 种不同的“不符合语法规则”的情况并加以处理。
- 存在标签完全一样的元素。例如 `TrackGame` 中出现了多个同样名字的运动员、`StellarSystem` 中出现了多个同样名字的恒星或行星、`SocialNetworkCircle` 中出现两条社交关系但连接是同样的两个人（例如 `<A, B, 0.1>` 和 `<B, A, 0.2>`）、出现指向自己的社交关系（例如 `<A, A, 0.2>`）等。分别考虑你所开发的 3 个应用中的每一个应用。
- 文件中各元素之间的依赖关系不正确。例如在 `AtomStructure` 中，`NumberOfTracks` 定义的轨道数量和 `NumberOfElectron` 中使用的轨道数量不一致；`PersonalAppSystem` 中，没有使用 `App` 定义就在 `UsageLog`、`InstallLog`、`UninstallLog`、`Relation` 中使用 `App` 的标签；`App` 在某条 `UsageLog` 记录的时间点并未安装在手机上；`SocialNetworkCircle` 中没有用 `Friend` 定义就在 `SocialTie` 中使用；等。分别考虑你所开发的 3 个应用中的每一个应用。

在使用正则表达式进行文本解析的过程中，程序一旦发现这些非法情况，应

捕获异常，进行异常处理（提示错误），结束此文件的读取，将当前所遇到的不合法之处汇总起来提示给用户，并让用户选择其他文本文件。

**注 1：**上面只是给出了若干例子，并不代表你只需要处理这几个情况。请仔细阅读 Lab3 实验手册中的说明，在解析输入文件时考虑各种可能的不合法/不一致的情况，并自行设计错误和异常处理机制。

**注 2：**上面的要求中提到“程序一旦发现这些非法情况，应捕获异常，进行异常处理（提示错误），结束文件读取，将当前所遇到的不合法之处汇总起来提示给用户，并让用户选择其他文本文件”，这意味着你的程序不能简单抛出一个 `unchecked exception` 使应用直接退出，而是要使用 `checked exception` 进行处理。

**注 3：**请自行构造包含上述各种错误的输入文本，用于测试你的异常处理代码是否正常工作，并将这些文本文件随你的项目一起提交至 GitHub。

接下来，考虑 Lab3 手册中 3.11 节关于“判断多轨道系统的合法性”的部分。这些合法性规则，有些通过 ADT 的 `rep invariants` 和 `checkRep()` 加以保证，有些可在这里通过 `checked exception` 进行处理，帮助你发现数据中的潜在 bug。

## 3.2. Assertion and Defensive Programming

基于你在 Lab 3 里为各个 ADT 所撰写的 `rep invariants (RI)`、`Abstraction Function (AF)`，以及它们的每个方法的 `spec`（`pre-condition` 和 `post-condition`），对代码进行防御式改造。

针对每个类，它们有明确的 RI（如果你没有在 Lab 3 的 ADT 代码中阐述清楚这些 `invariants`，需对照 Lab 3 实验手册仔细补充），请据此撰写 `checkRep()` 并加入各方法。例如（但你的实现不应仅限于这两个例子）：

- 在 `TrackGame` 中，不管用户对比赛方案如何修改，每组比赛中的人数不超过跑道数；
- 在 `SocialNetworkCircle` 中，不管社交关系如何增加或删除，第 `i` 层轨道上的人与中心点的人之间的最短路径等于 `i`。

针对每个方法，它们有明确的 `spec`，请据此在各方法中通过 `assertion` 或异常机制分别处理 `pre-condition` 和 `post-condition`。

注意：上述只是在举例，并非全集，你需要对所有的类和方法进行防御式改造，并以注释的形式说清楚你的“防御策略”。

另外：如果你在 Lab3 中已经做了足够多的“防御工作”，该任务只需在实验

报告中说清楚你的每一个防御策略和相应的实现即可。

### 3.3. Logging

使用 `java logging`（或其他第三方 `java` 日志库如 `log4j`、`SLF4J`），为 3.1 和 3.2 节经过异常处理、错误处理、断言处理的程序增加日志功能。日志需要记录以下信息：

- 所有的异常/错误：发生的时间、异常/错误类型、异常/错误发生的类名和方法名，异常/错误的信息、异常/错误的处理结果；
  - 对多轨道系统的所有操作，包括：读取文件、增加/删除轨道、增加中心点物体、增加/删除轨道上的物体、增加/删除物体之间的关系、跃迁等。
- 具体参见 Lab3 手册的具体要求。

为应用添加日志查询功能，用户可输入过滤条件（例如按时间段、按类型、按类、按方法、按操作类型）进行日志查询。注意：查询结果不要直接显示原始日志记录，需要具有良好的可理解性。

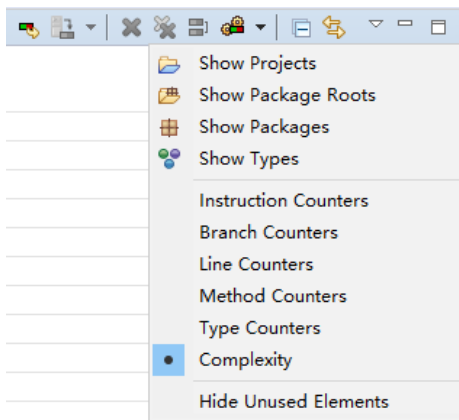
### 3.4. Testing for Robustness and Correctness

使用等价类和边界值的测试思想，为各 ADT 添加 `testing strategy`。

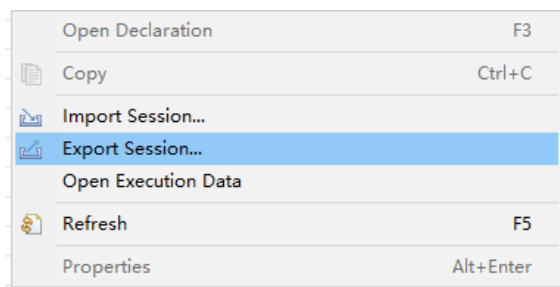
考虑 3.1 节中出现的多种非法情形，设计一组测试用例，人为制造非法输入的文件和非法输入的图操作指令，对程序进行健壮性和正确性测试，想方设法让程序崩溃（即验证程序是否有容错能力）。

使用 JUnit 为上述测试用例编写代码，并运行测试。

使用 `EclEmma` 查看测试的语句覆盖度（`Instruction Counters`）。切换为分支覆盖度（`Branch Counters`）和路径覆盖度（`Complexity`），对比一下三种覆盖度的差异。若覆盖度过低，继续增加测试用例，使覆盖度越高越好。最终，生成 `EclEmma` 的测试覆盖度报告。



EclEmma 的选项菜单



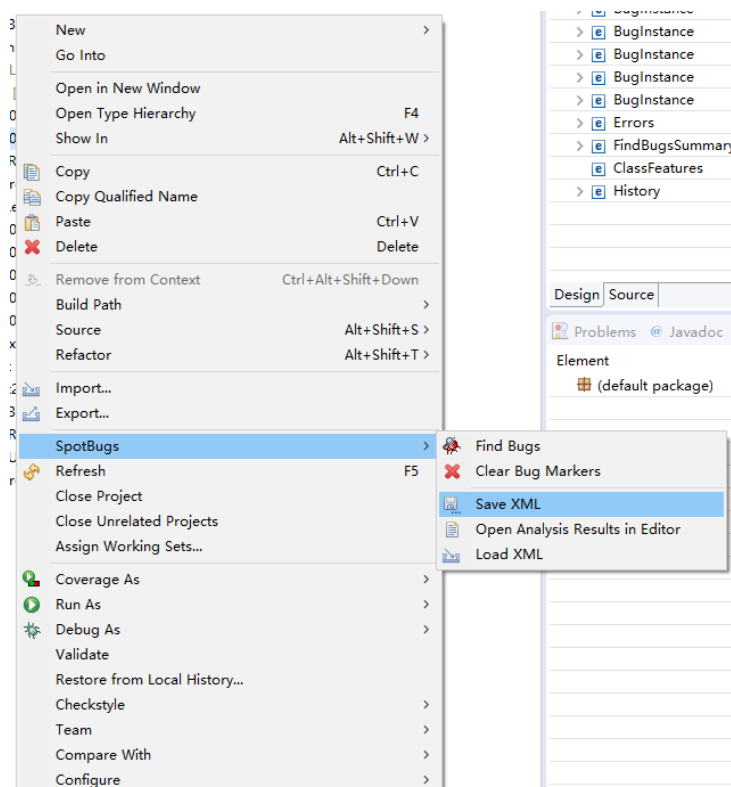
EclEmma 右键菜单导出覆盖度报告（HTML）

注意：对所有的方法进行测试！

另外：如果你在 Lab3 中已经做了足够多的“测试工作”，该任务只需在实验报告中说清楚你的测试策略和相应的实现即可。

### 3.5. Using SpotBugs tool

使用 SpotBugs 工具，检查你到目前为止的代码，根据工具所提示的潜在 bug 及其危害程度，逐项消除，并在后续编程中避免此类问题。



在项目上点击右键，选择 SpotBugs

### 3.6. Debugging

该节任务不针对 Lab 3 的背景，而是针对已有的程序，通过测试和调试发现其中存在的 bug 并将其修复。

请从 [https://github.com/rainywang/Spring2019\\_HITCS\\_SC\\_Lab4](https://github.com/rainywang/Spring2019_HITCS_SC_Lab4) 下载待修复的程序源文件，每个程序中蕴含错误，无法按期望执行得到结果。一共有三个独立的程序，利用你的经验和相关工具进行 debug，发现问题所在，将代码修改正确，给出测试用例，并提交。不要改动类名和文件名。

修复后的代码中需要增加必要的注释以标注出修改之处和原因，以帮助读者理解你所做的修改。

**注 1：**在 debug 之前，请利用各程序前面的 spec，根据等价类和边界值方法撰写充足的 JUnit 测试用例（以及注释形式的 testing strategy），根据测试结果进行逐步调试，发现错误并修复。提交结果中请务必在 test/debug 下包含你针对三个程序的所有测试用例。

**注 2：**不要试图另起炉灶重新写一个程序完成 spec 里的功能，而是要在遵循原程序的思路基础上进行修改。该任务不是编程竞赛，而是给你提供一个训练 debug 技巧的机会，无需用全新的代码展现你的编程水平。

提交目录：

```
项目名称: Lab4-学号
目录:      src
子目录:      debug
              (修复后的源文件).java
              test
                debug
                  ...Test.java
```

## 4. 实验报告

针对上述任务，请遵循 CMS 上 Lab4 页面给出的**报告模板**，撰写简明扼要的实验报告。

实验报告的目的是记录你的实验过程，尤其是遇到的困难与解决的途径。不需要长篇累牍，记录关键要点即可，但需确保报告覆盖了本次实验所有开发任务。

注意：

- 实验报告不需要包含所有源代码，请根据上述目的有选择的加入关键源代码，作为辅助说明。
- 请确保报告格式清晰、一致，故请遵循目前模板里设置的字体、字号、行间距、缩进；
- 实验报告提交前，请“目录”上右击，然后选择“更新域”，以确保你的目录标题/页码与正文相对应。



- 实验报告文件可采用 Word 或 PDF 格式，命名规则：Lab4-学号-Report。

## 5. 提交方式

**截止日期：**第 12 周周日夜间 23:55。截止时间之后通过 Email 等其他渠道提交实验报告和代码，均无效，教师和 TA 不接收，学生本次实验无资格。

**源代码：**从本地 Git 仓库推送至个人 GitHub 的 Lab4 仓库内。

**实验报告：**除了随代码仓库（doc）目录提交至 GitHub 之外，还需手工提交至 CMS 实验 4 页面下。

## 6. 评分方式

TA 在第 9 周和第 11 周实验课上现场验收：学生做完实验之后，向 TA 提出验收申请，TA 根据实验要求考核学生的程序运行结果并打分。现场验收并非必需，由学生主动向 TA 提出申请。

Deadline 之后，教师使用持续集成工具对学生在 GitHub 上的代码进行测试。教师和 TA 阅读实验报告，做出相应评分。