哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

Lab 4: 代码评审与程序性能优化

实验要求

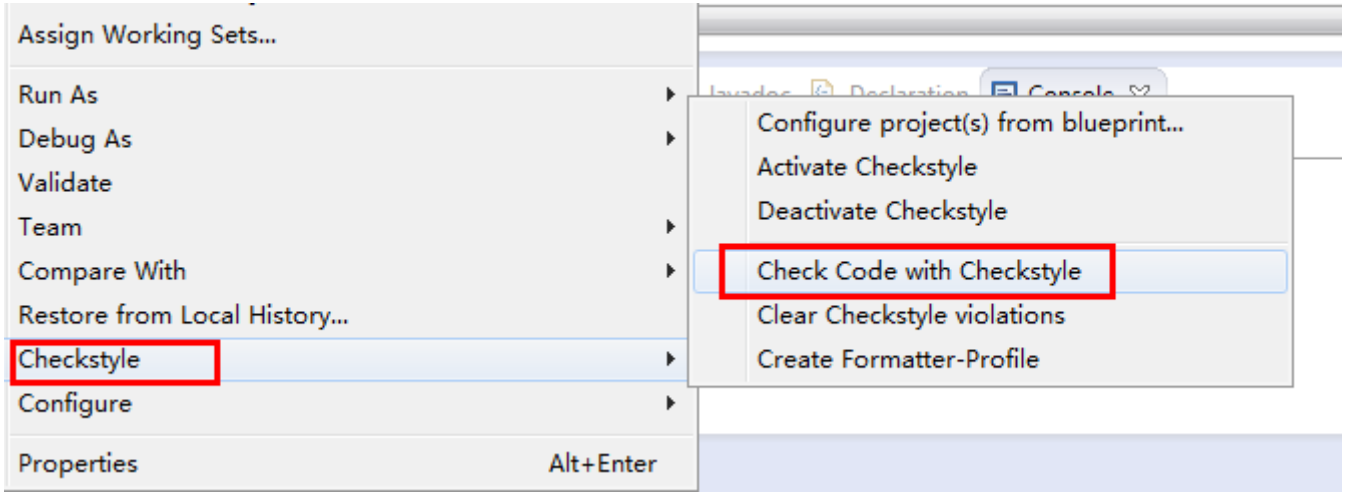
- 针对**Lab1**所完成的代码，进行代码评审**(**走查**)**和性能分析，从性能角度对代码进行优化；
- 练习代码评审的两个方面：静态分析、动态分析**(profiling)**；
- 使用以下四个工具完成实验：
  - Checkstyle
  - FindBugs
  - PMD
  - VisualVM
- 按**Lab1**的分组方式，两人一组，随机分配另一组的代码作为本组评审和分析的对象，实验期间不能与原作者进行沟通。

Checkstyle

- **CheckStyle帮助JAVA开发人员遵守某些编码规范，能自动化代码规范检查过程，从而使开发人员从这项重要但是枯燥的任务中解脱出来。**
- **CheckStyle检验的主要内容：**
  - Javadoc注释
  - 命名约定
  - 标题
  - Import语句
  - 体积大小
  - 空白
  - 修饰符
  - 块
  - 代码问题
  - 类设计

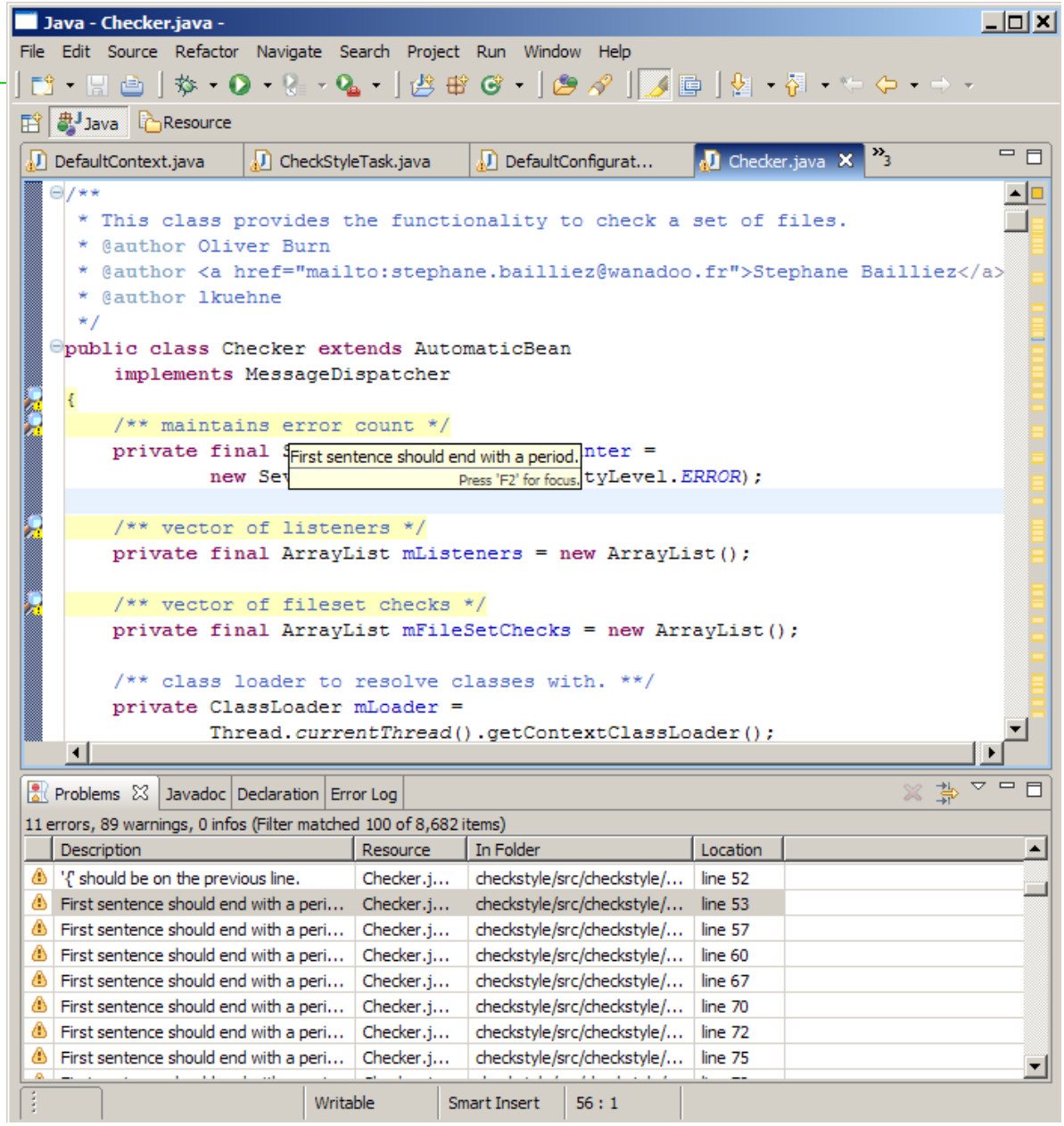
Checkstyle

- 从https://sourceforge.net/projects/eclipse-cs/files/latest/download下载Eclipse插件；
- 也可以直接从Eclipse的Update Center中更新 http://eclipse-cs.sf.net/update ；
- 安装成功后，选中工程，右键选择Checkstyle->check code with checkstyle，检查错误即可。

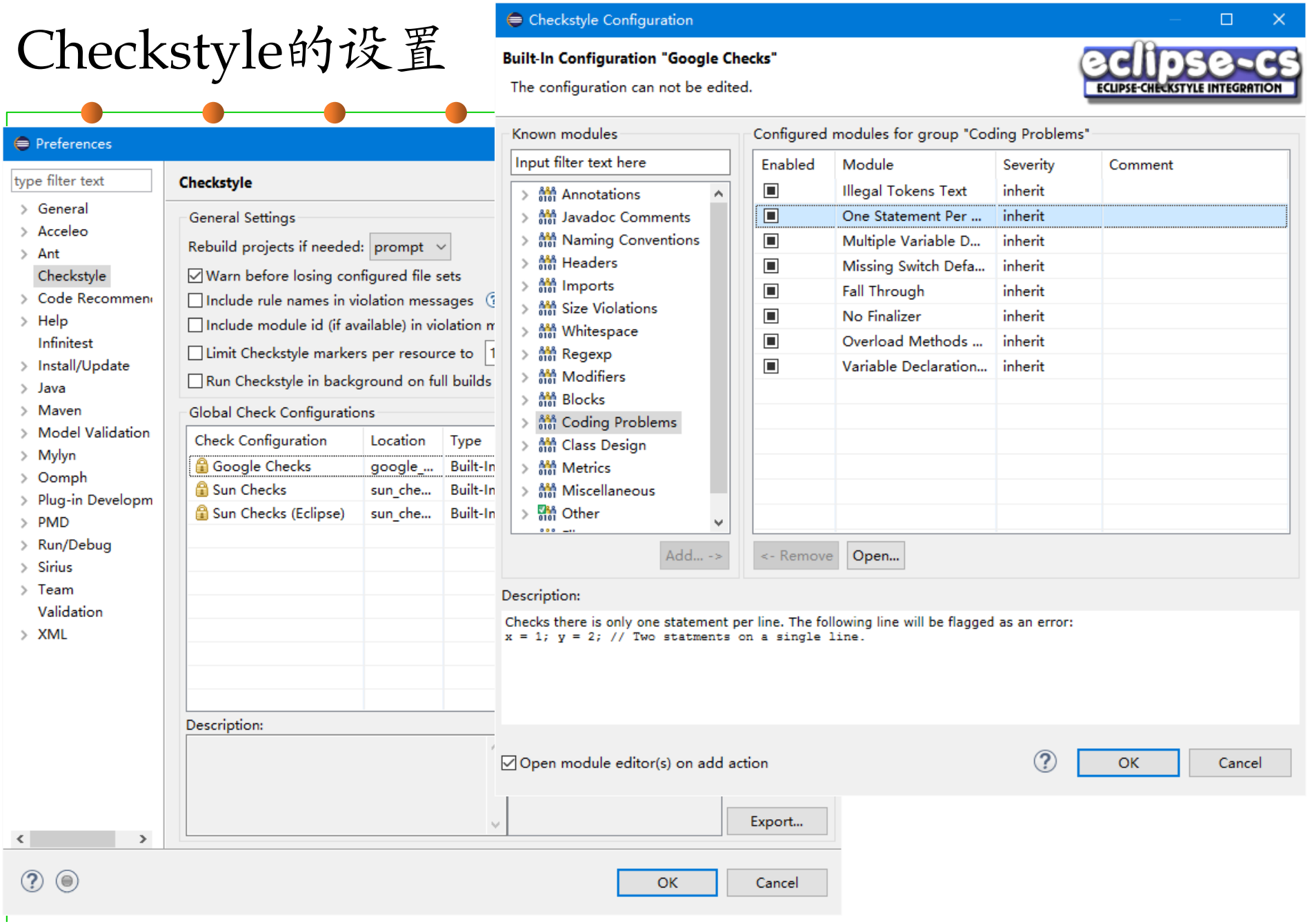


Checkstyle

- 检查结果在每行开始显示图标，鼠标移动至其上时显示详细问题；



Checkstyle的设置



CheckStyle的帮助

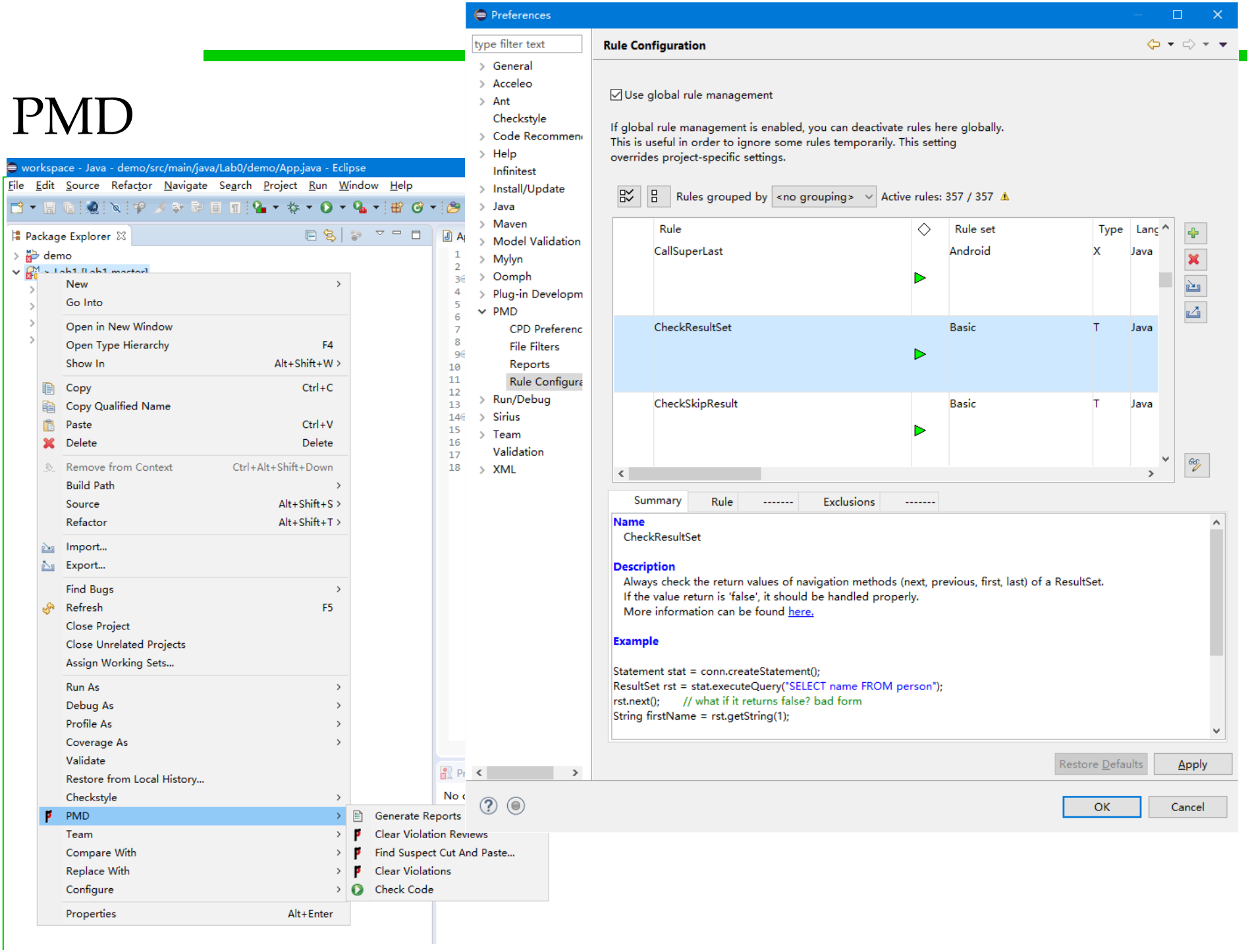
- 从http://eclipse-cs.sourceforge.net/basic_setup_project.html、http://eclipse-cs.sourceforge.net/basic_creating_config.html获得帮助：如何配置、如何自定义配置；

- 可从http://eclipse-cs.sourceforge.net/basic_creating_config.html 页面的最下方或者http://www.cnblogs.com/bluesky4485/archive/2011/11/30/2269198.html查阅每个问题的具体含义以及如何修正。
- 由此理解Java编码规范。

PMD

- **PMD：静态代码分析工具，自动检测各种潜在缺陷以及不安全或未优化的代码。**
- **从https://sourceforge.net/projects/pmd/files/pmd-eclipse/update-site 下载并配置至Eclipse，也可以从Eclipse的Update Center中更新 https://dl.bintray.com/pmd/pmd-eclipse-plugin/updates/**
- **PMD侧重于预先检测缺陷，它提供了高度可配置的丰富规则集，用户可以方便配置对待特定项目使用这些规则(从 https://pmd.github.io/pmd-5.5.1/pmd-java/rules/index.html 查阅各项规则集的具体含义)**
  - 右键选择项目，选择Find Suspect Cut And Paste (CPD)，用来检查重复代码(比如通过复制粘贴得到的代码)；
  - 右键选择项目，选择PMD->Check Code，用静态代码检查工具查看代码是否符合规范，它定义了一组检查规则。

PMD

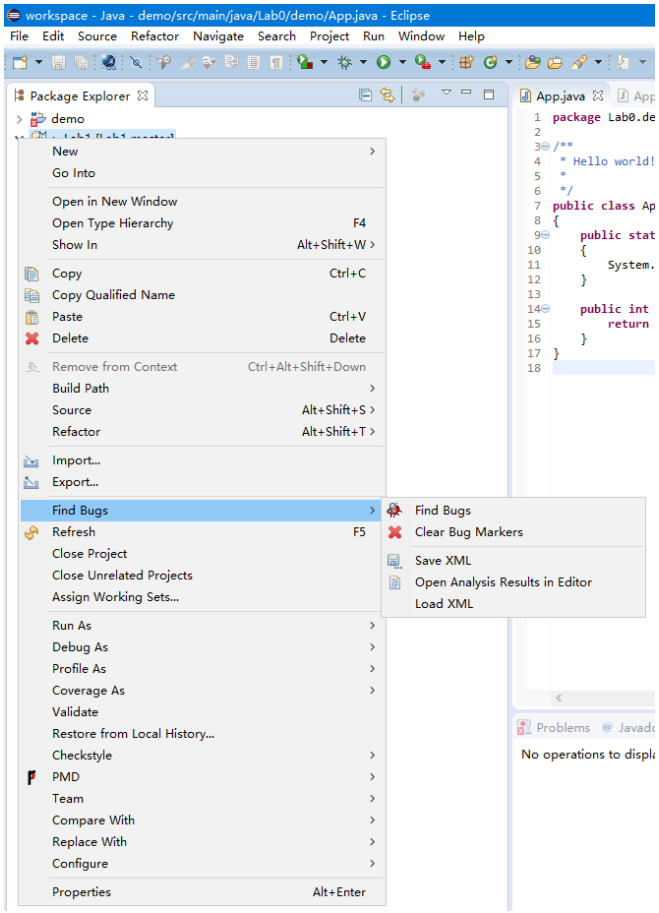

workspace - Java - demo/src/main/java/Lab0/demo/App.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
demo
New
Go Into
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Find Bugs
Refresh F5
Close Project
Close Unrelated Projects
Assign Working Sets...
Run As
Debug As
Profile As
Coverage As
Validate
Restore from Local History...
Checkstyle
PMD
Team
Compare With
Replace With
Configure
Properties Alt+Enter
Generate Reports
Clear Violation Reviews
Find Suspect Cut And Paste...
Clear Violations
Check Code
Preferences
type filter text
General
Acceleo
Ant
Checkstyle
Code Recommen
Help
Infinitest
Install/Update
Java
Maven
Model Validation
Mylyn
Oomph
Plug-in Developm
PMD
CPD Preferenc
File Filters
Reports
Rule Configura
Run/Debug
Sirius
Team
Validation
XML
Rule Configuration
Use global rule management
If global rule management is enabled, you can deactivate rules here globally.
This is useful in order to ignore some rules temporarily. This setting
overrides project-specific settings.
Rules grouped by <no grouping> Active rules: 357 / 357
Rule Rule set Type Lang
CallSuperLast Android X Java
CheckResultSet Basic T Java
CheckSkipResult Basic T Java
Summary Rule ------- Exclusions -------
Name
CheckResultSet
Description
Always check the return values of navigation methods (next, previous, first, last) of a ResultSet.
If the value return is 'false', it should be handled properly.
More information can be found here.
Example
Statement stat = conn.createStatement();
ResultSet rst = stat.executeQuery("SELECT name FROM person");
rst.next(); // what if it returns false? bad form
String firstName = rst.getString(1);
Restore Defaults Apply
OK Cancel

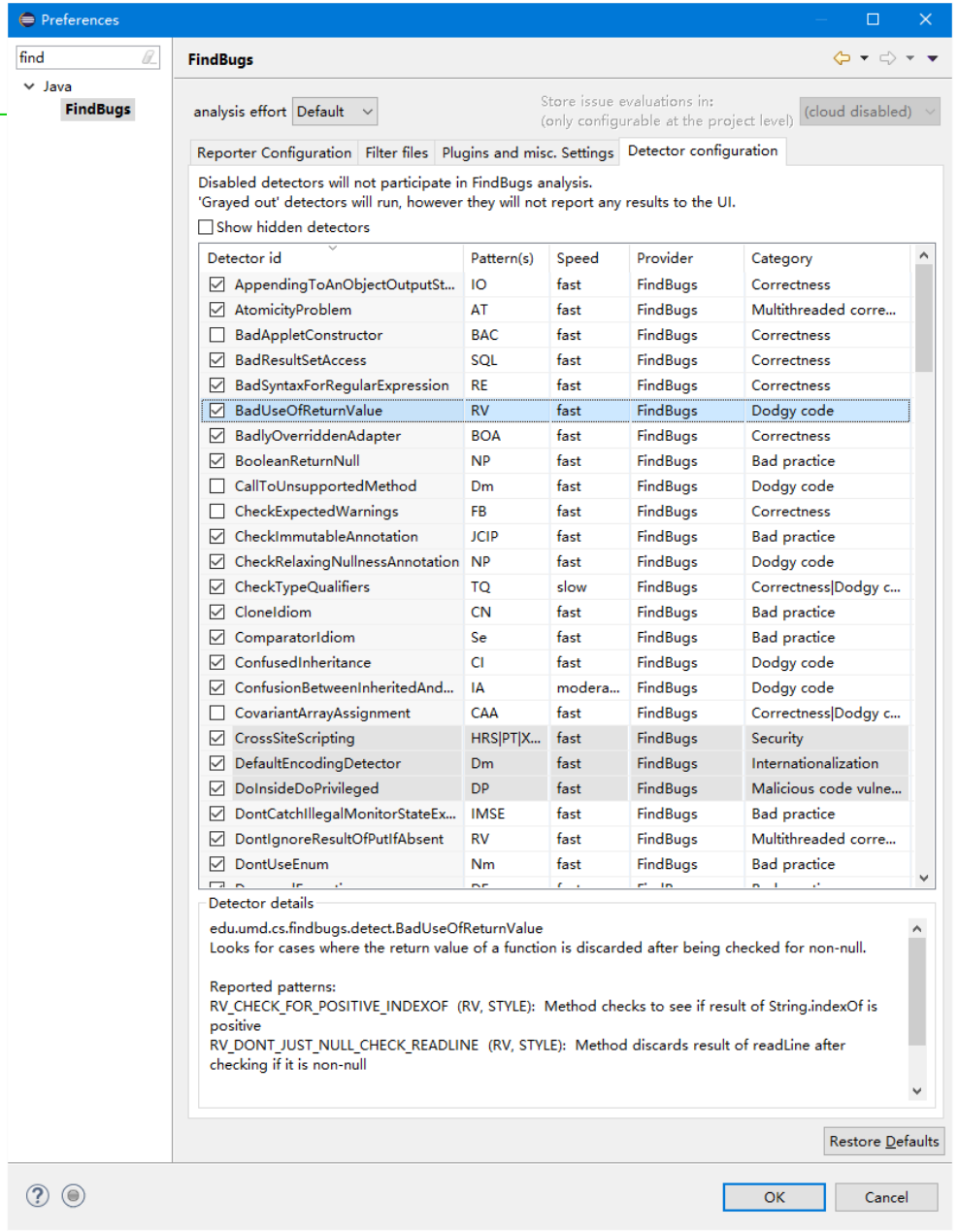
FindBugs

- **FindBugs：Java静态代码分析工具，不注重样式或者格式，专注于寻找真正的缺陷或者潜在的性能问题，帮助开发者提高代码质量以及排除隐含的缺陷，可以在不实际运行程序的情况对软件进行分析。**
- **从http://findbugs.sourceforge.net下载并配置至Eclipse，也可通过Eclipse Marketplace直接安装。**
- **常见的检测类型如下：**
  - 正确性(Correctness)：在某种情况下会导致bug，如错误的强制类型转换等。
  - 最佳实践反例(Bad practice)：这种类别下的代码违反了公认的最佳实践标准，比如某个类实现了equals方法但未实现hashCode方法等。
  - 多线程正确性(Multithreaded correctness)：关注于同步和多线程问题。
  - 性能(Performance)：潜在的性能问题。
  - 安全(Security)：安全相关。
  - 高危(Dodgy)：该类型下的问题代码导致bug的可能性很高。

FindBugs

- http://findbugs.sourceforge.net/bugDescriptions.html 获取FindBugs的详细bug描述清单。



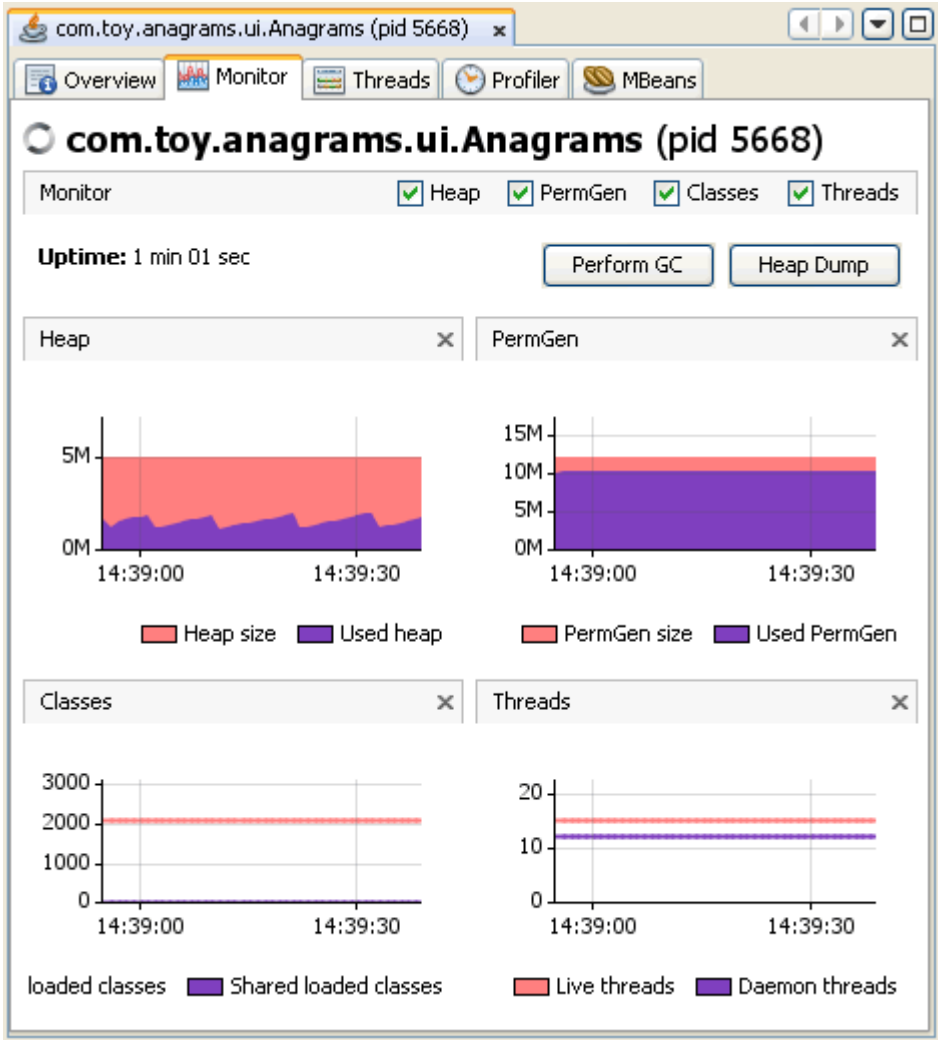


VisualVM

- **VisualVM是一个集成多个JDK命令行工具的可视化工具，是Java程序性能分析和运行监控的工具。**
  - 开发人员可以利用它来监控、分析线程信息，浏览内存堆数据。
  - 系统管理员可以利用它来监测、控制Java应用程序横跨整个网络的情况。
  - Java应用程序使用人员可以利用它来创建包含所有必要信息的Bug 报告。
- **两种使用方式：**
  - 从https://visualvm.github.io/ 下载客户端，安装在本机；注：自从 JDK 6 Update 7 以后已经作为 Oracle JDK 的一部分，位于 JDK 根目录的 bin 文件夹下。
  - 从http://visualvm.github.io/idesupport.html 下载Eclipse插件；
- **帮助文档：**
  - http://visualvm.github.io/documentation.html
  - http://visualvm.github.io/idesupport.html Eclipse中调用VisualVM

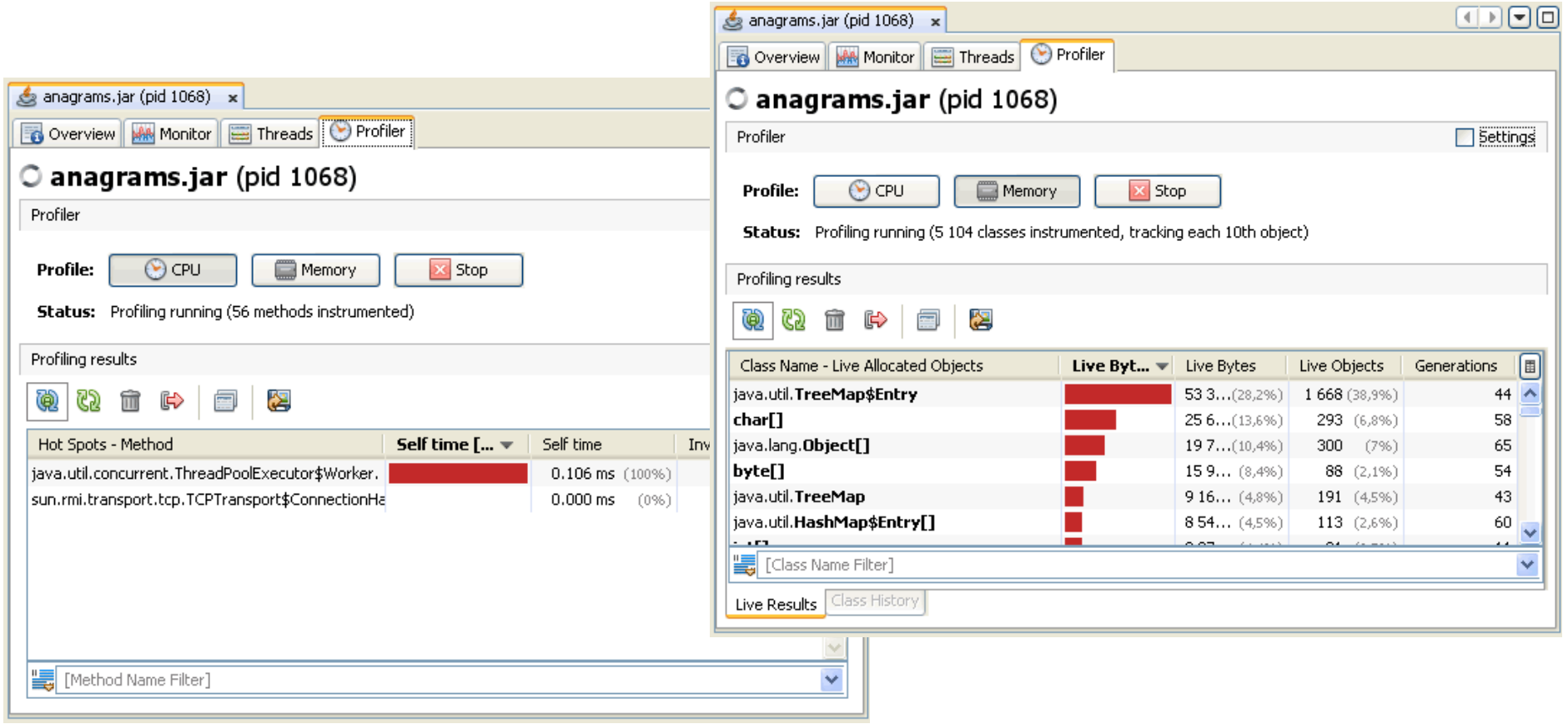
VisualVM

- 监控一个Java应用的整体情况



VisualVM

- **Profiling applications**
  - CPU Profiling：各方法的执行时间分布
  - Memory Profiling：内存使用情况



实验过程

- **Part 1：本组学生**
  - 在Eclipse环境下配置CheckStyle、FindBugs、PMD、VisualVM四个工具；
  - 在GitHub上找到待评审代码，fork至本组GitHub仓库内并重命名为Lab4；
  - 将fork得到的仓库Lab4 clone至本组的本地仓库Lab4；
  - 进行代码走查，记录所发现的问题，对代码进行修改，消除问题；
  - 提交代码至本地git仓库；
  - 分别使用四种工具对待评审代码进行评审和性能分析，记录所发现的问题：
    - 前三者：使用不同的规则集/配置来检测代码不符合规范之处；对它们的结果进行对比，看它们发现问题的能力差异；
    - VisualVM： CPU Profiling, Memory Profiling；
  - 修改代码，消除所发现的问题；
  - 提交代码至本地git仓库，并将所有修改历史push到GitHub上本组仓库Lab4；
  - 在Github上将最新的提交Pull Request到原作者的Lab1仓库；

实验过程

- **Part 2：原作者**
  - 原作者在自己的Lab1仓库建立新分支Lab4，将评审组pull request过来的代码合并进去，将Lab4分支fetch到本地Lab1仓库；
  - 原作者使用git命令列出评审组所做的所有修改，逐项确认并理解修改原因；
  - 对不认可的修改，自行加以去除，在Lab4分支上加以提交；
  - 将Lab4分支与原来的主分支merge起来，推送至本组GitHub的Lab1仓库。

评判标准

- 是否可在Eclipse中顺利配置四种工具并启动执行；
- 能够充分理解PMD、CheckStyles、FindBugs三种工具所检测的不符合规范的常见代码问题；
- 能够充分理解VisualVM产生的运行结果(时间信息、内存信息)；
- 根据工具报告的问题，对代码做了充分的改进；
- 使用Git/GitHub完成全部的提交和合并任务，并记录整个过程；

- 实验报告的规范性、完整性。

提交方式

- 请遵循实验报告模板撰写。

- **Part1 提交日期：第7周周五晚(10月20日 23:55)**
  - 评审者完成到步骤“在GitHub上将最新的提交Pull Request到原作者的仓库”，留出三天时间给原作者完成Part2；
- **实验报告提交日期：第8周周一晚(10月23日 23:55)**
- **提交两个文件到CMS：**
  - 实验报告：命名规则“学号-Lab4-report.doc”
  - 优化之后的代码(此为Part1里评审和改进的对方代码)：命名规则“学号-Lab4-code.rar/zip”
- 同组的两人需分别提交上述文件。

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

结束