单元测试

实验目标

- 针对Lab1完成的程序，设计黑盒测试用例和白盒测试用例。

- 在JUnit环境下撰写测试代码并执行测试。
- 使用EclEmma或IDE自带工具统计测试的覆盖度。

Step 1：设计黑盒测试用例

- **利用黑盒测试的等价类和边界值方法，为Lab1待测程序设计一组测试用例。**
- **测试对象(五选一)：**
  - `String queryBridgeWords(String word1, String word2)`：查询桥接词
  - `String generateNewText(String inputText)`：根据bridge word生成新文本
  - `String calcShortestPath(String word1, String word2)`：计算两个单词之间的最短路径
  - `Double calPageRank(String word)`：计算单词的PR值
  - `String randomWalk()`：随机游走
- **根据Lab1的要求，给出所选被测函数的需求规约描述；**
- **每个测试用例由输入数据和期望输出两部分组成。**

Step 2：使用JUnit编写黑盒测试用例并执行

- **在Lab1的Git仓库里，建立新的Git分支，命名为Lab3b (b代表black-box testing)；**
- **针对每个测试用例撰写testcase；**
- **执行测试用例：**
  - 执行，产生结果，记录实际输出；
  - 记录、分析结果；
  - 针对失败的测试用例，发现代码的问题，并修改代码；
- **重复上一步，直到所有的测试用例都完全通过为止。**
- **将Lab3b合并到master分支，并推送至GitHub。**

Step 3：设计白盒测试用例

- **针对Lab3b分支的当前代码，对以下函数(三选一)使用基本路径法设计白盒测试用例：**
  - `String queryBridgeWords(type G, String word1, String word2)` 查询桥接词
  - `String calcShortestPath(type G, String word1, String word2)`：计算两个单词之间的最短路径
  - `String randomWalk(type G)`：随机游走

- **每个测试用例由输入数据和期望输出两部分组成。**

Step 4：使用JUnit编写并执行白盒测试代码

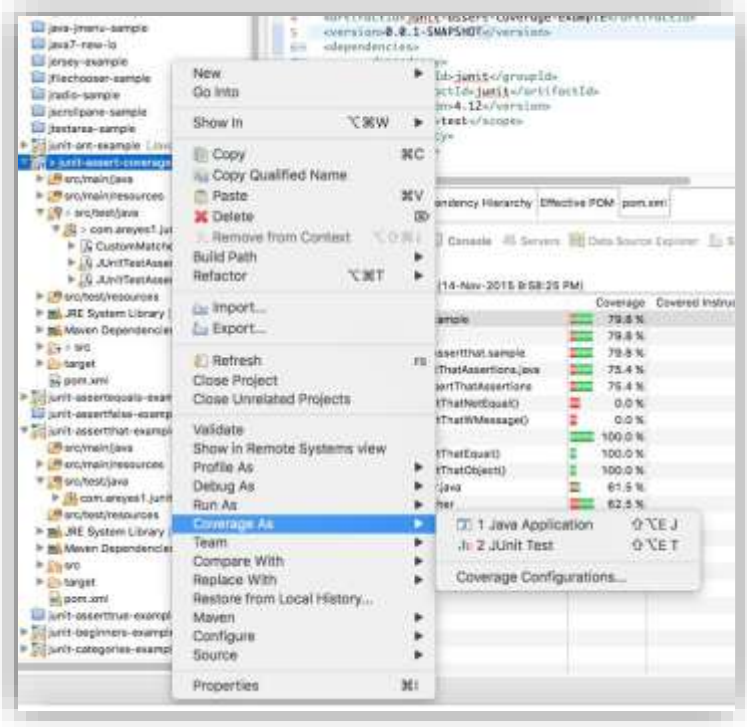
- **在Lab1的Git仓库里，建立新的Git分支，命名为Lab3w (w代表white-box testing)；**
- **针对每个测试用例撰写testcase；**
- **执行测试用例：**
  - 执行，产生结果，记录实际输出；
  - 记录、分析结果；
  - 针对失败的测试用例，发现代码的问题，并修改代码；
- **重复上一步，直到所有的测试用例都完全通过为止。**
- **将Lab3w合并到master分支，并推送至GitHub。**

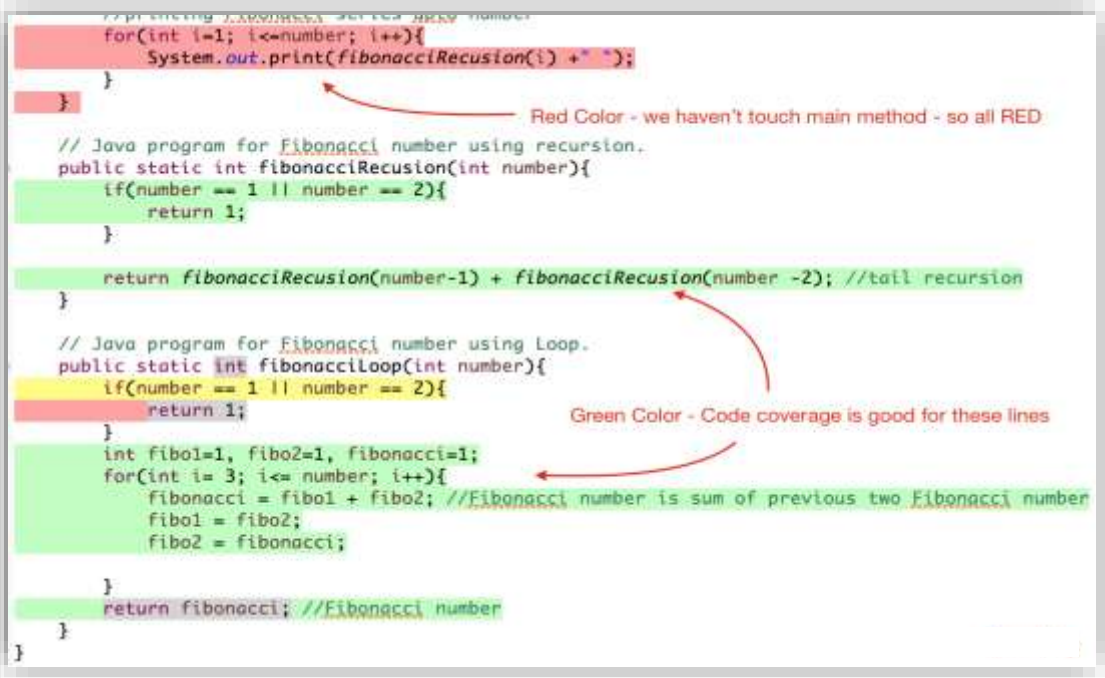
Step 5：统计测试覆盖度

- 测试的覆盖率是测试质量的一个重要指标。
  - 可使用EclEmma工具 https://www.eclemma.org/
  - 或Eclipse自带Coverage分析工具
  - 或IDEA自带Coverage分析工具
- 在测试过程中，当运行测试程序，覆盖度分析工具可自动分析出被测程序的各行代码被覆盖的情况；
- 代码被覆盖得越全面，测试质量就越好。
- 从工具导出覆盖度分析报告，观察语句覆盖度。

Step 5：统计测试覆盖度

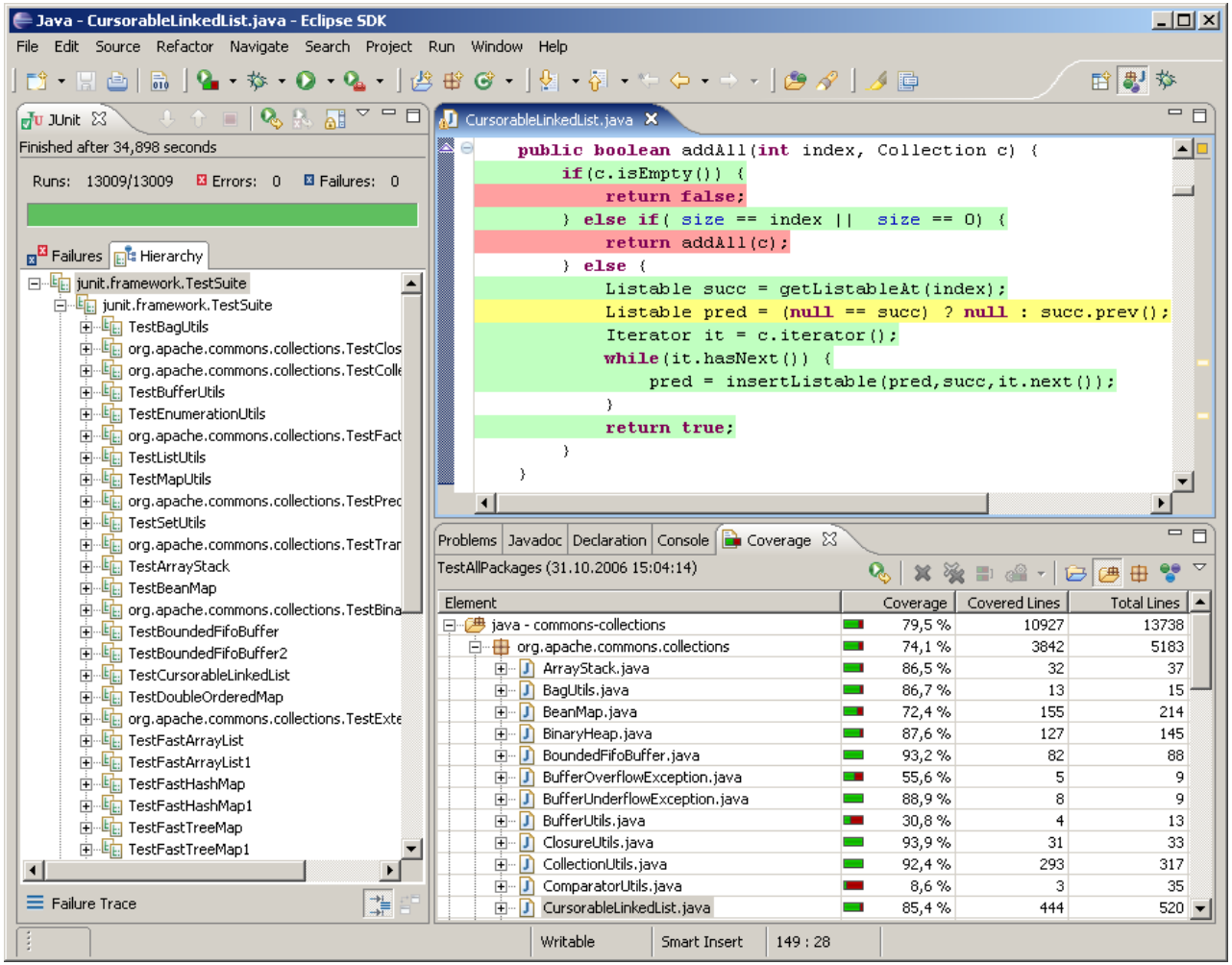
- **Eclispe下通过Marketplace安装EclEmma**





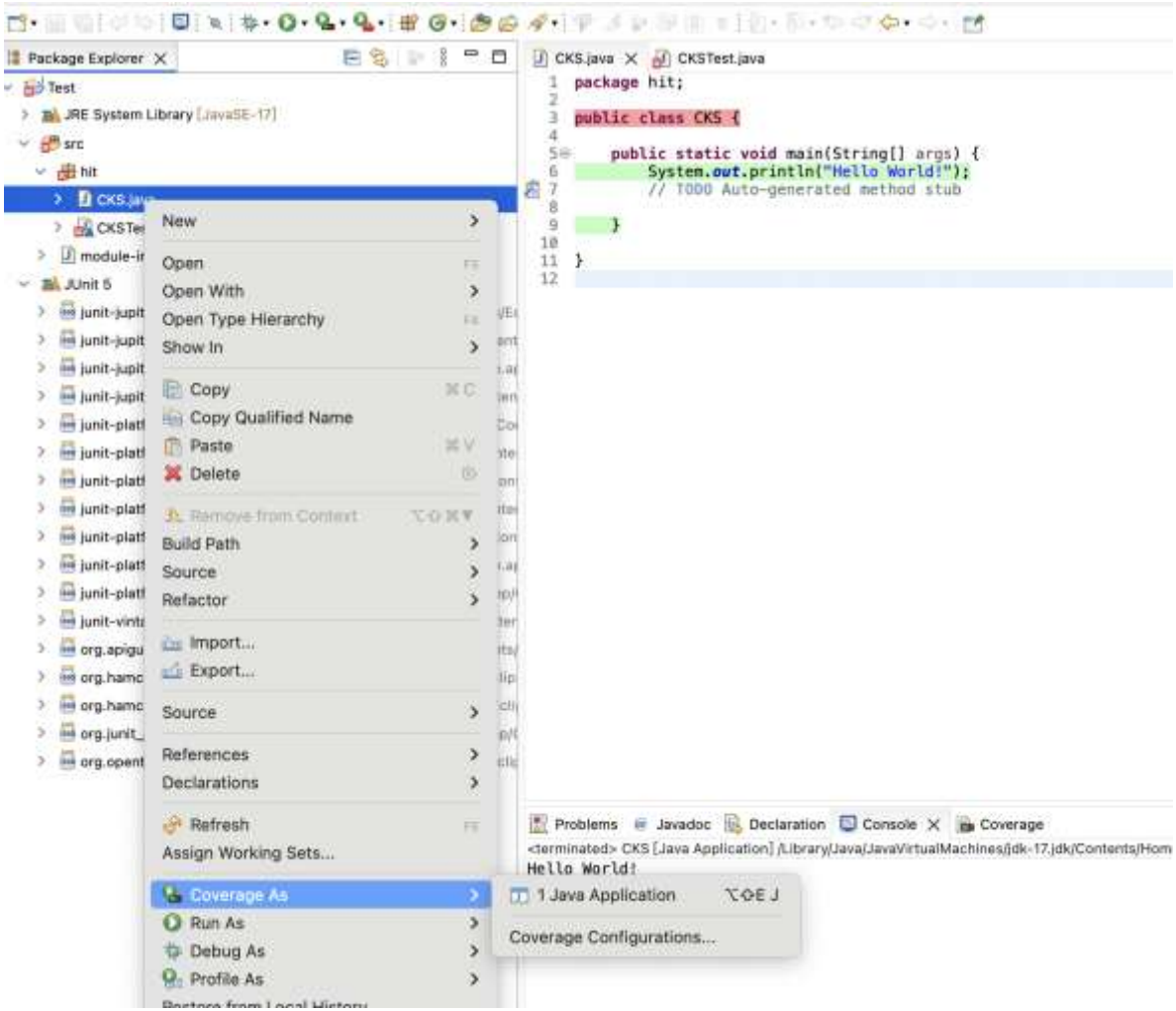
Step 5：统计测试覆盖度

- **Eclispe下通过Marketplace安装EclEmma**



Step 5：统计测试覆盖度

- 采用Eclispe下自带的覆盖度分析工具

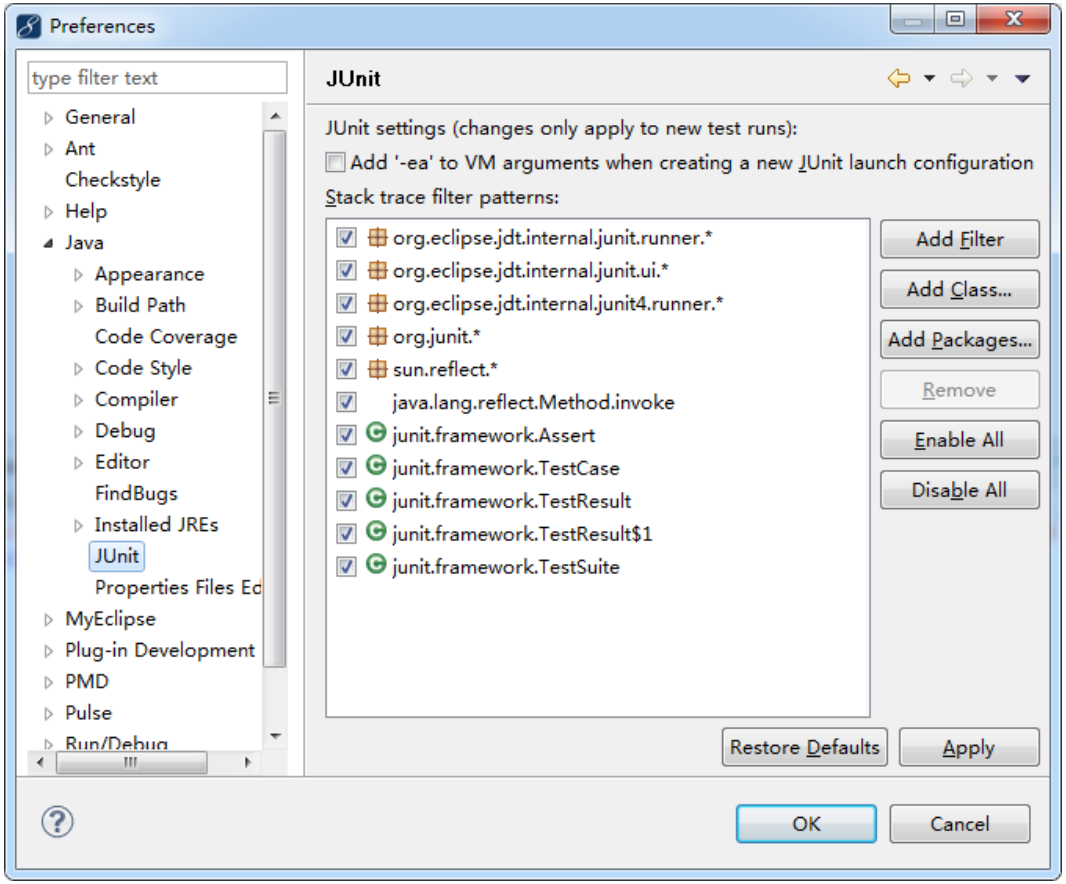


Step 5：统计测试覆盖度

- **IDEA下自带覆盖度分析工具**
  - https://www.jetbrains.com/help/idea/code-coverage.html

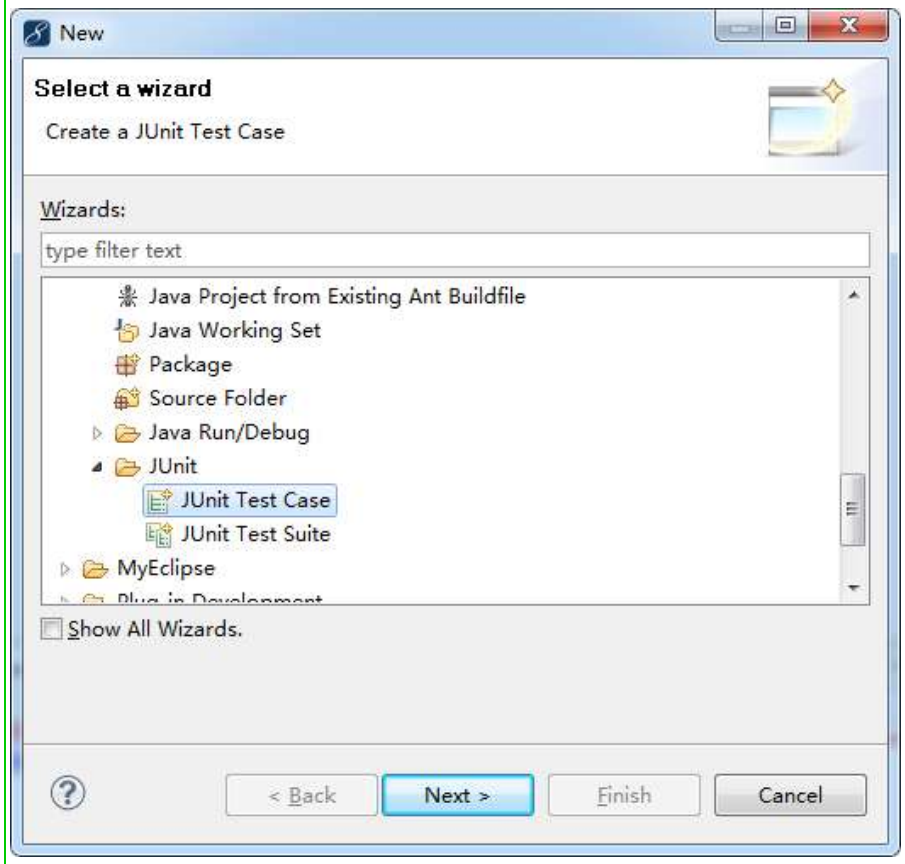
JUnit的安装与配置

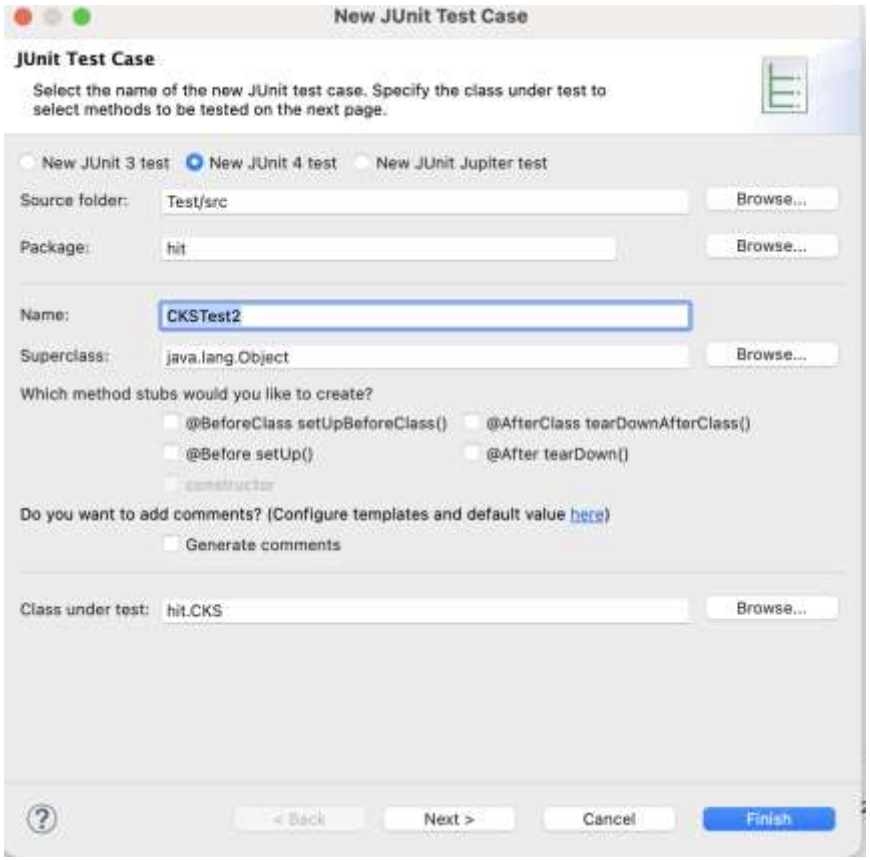
- https://junit.org/junit4/ https://junit.org/junit5/
- **Eclipse和IDEA当前版本已经集成了Junit。**



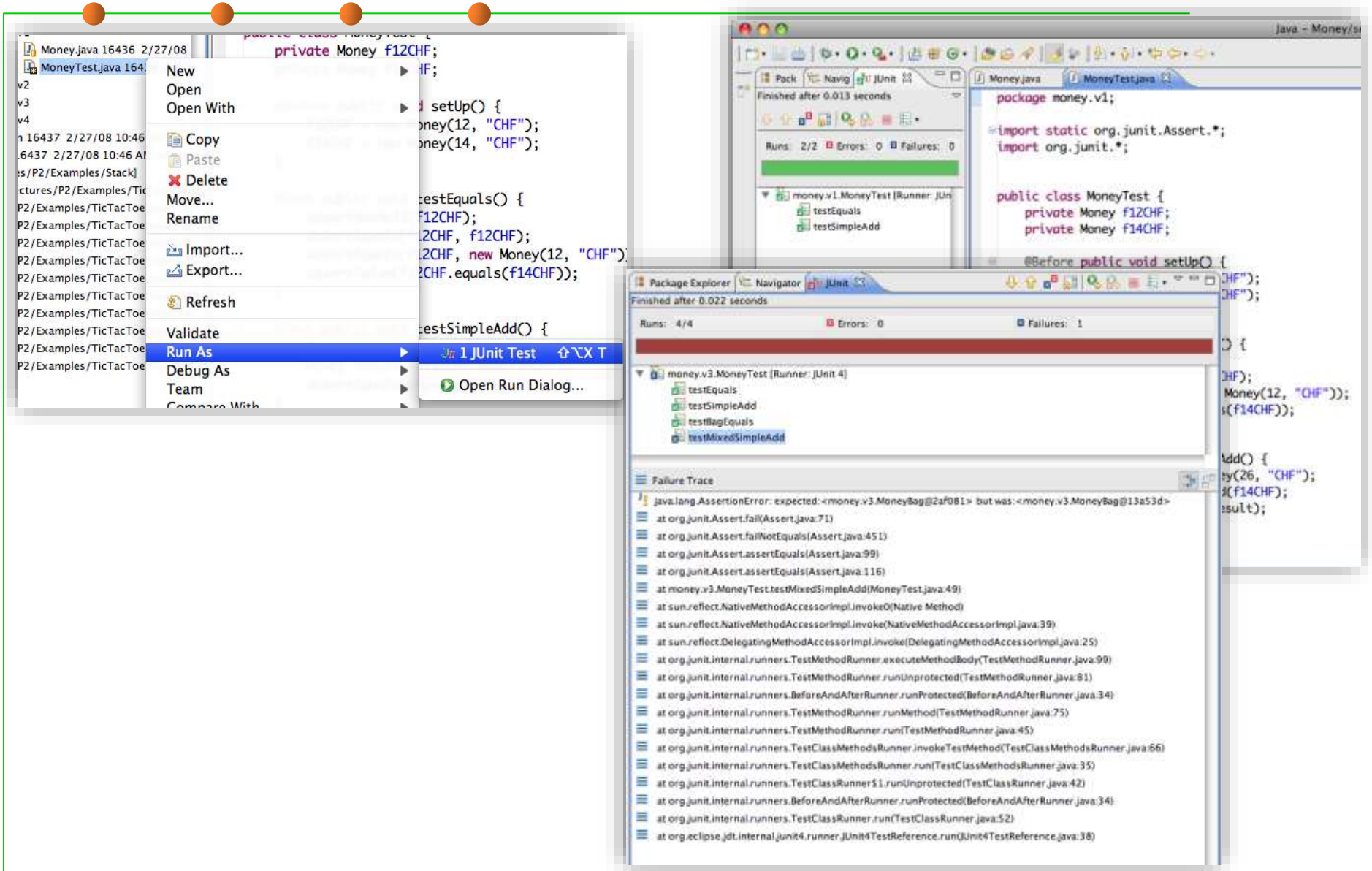
JUnit的安装与配置 Eclipse

- 针对一个存在的项目，在其中新建JUnit测试用例或测试套件；



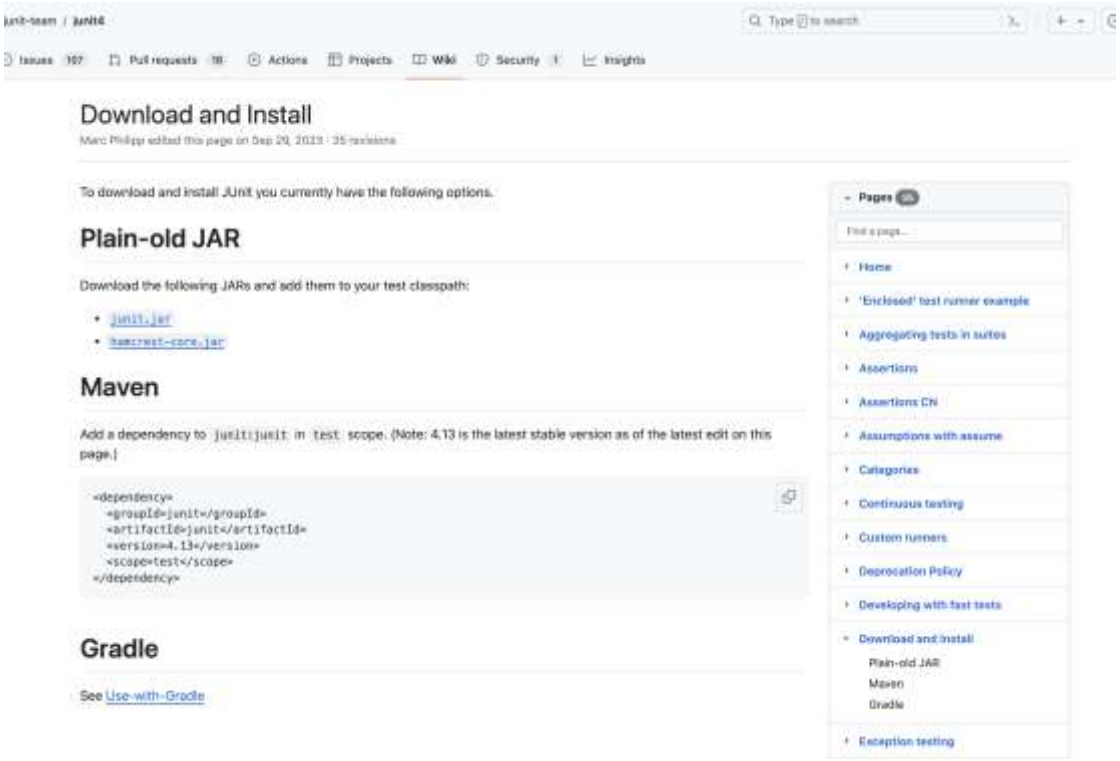


在Eclipse中运行JUnit测试用例



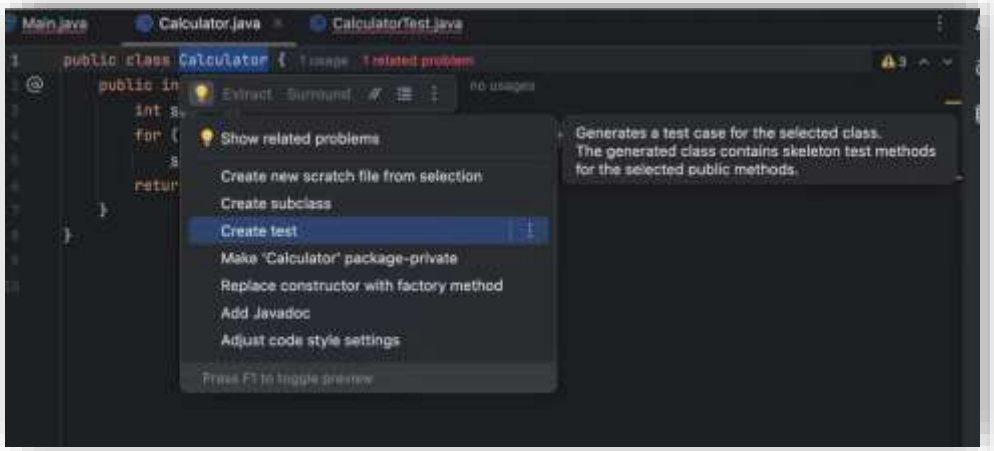
JUnit的安装与配置 IDEA

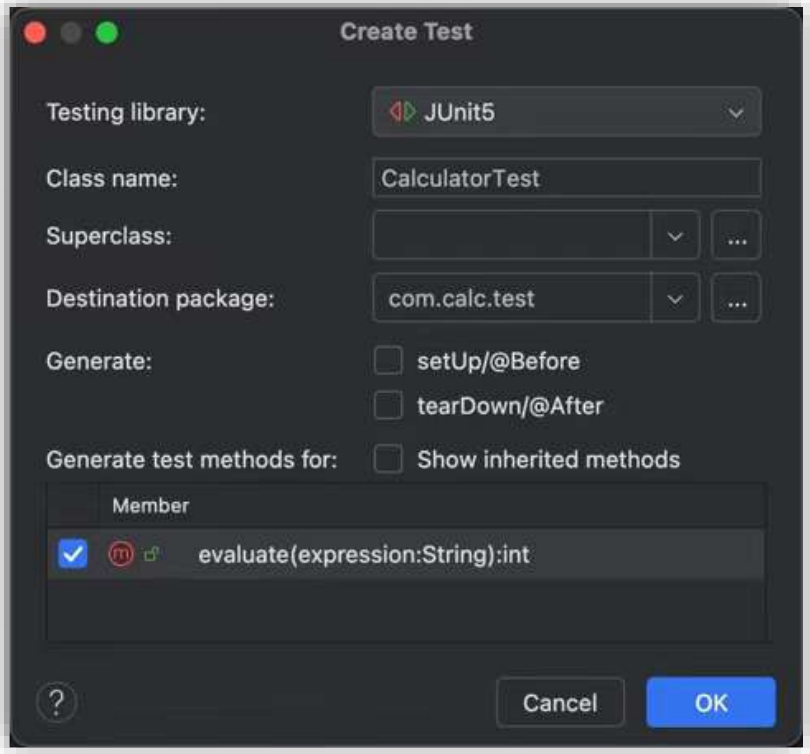
- **IDEA下使用JUnit https://junit.org/junit4/**
  - 方法1：下载 junit.jar 和 hamcrest-core.jar ，加入项目的Libraries中
  - 方法2：如果使用Maven，见网站说明



JUnit的安装与配置 IDEA

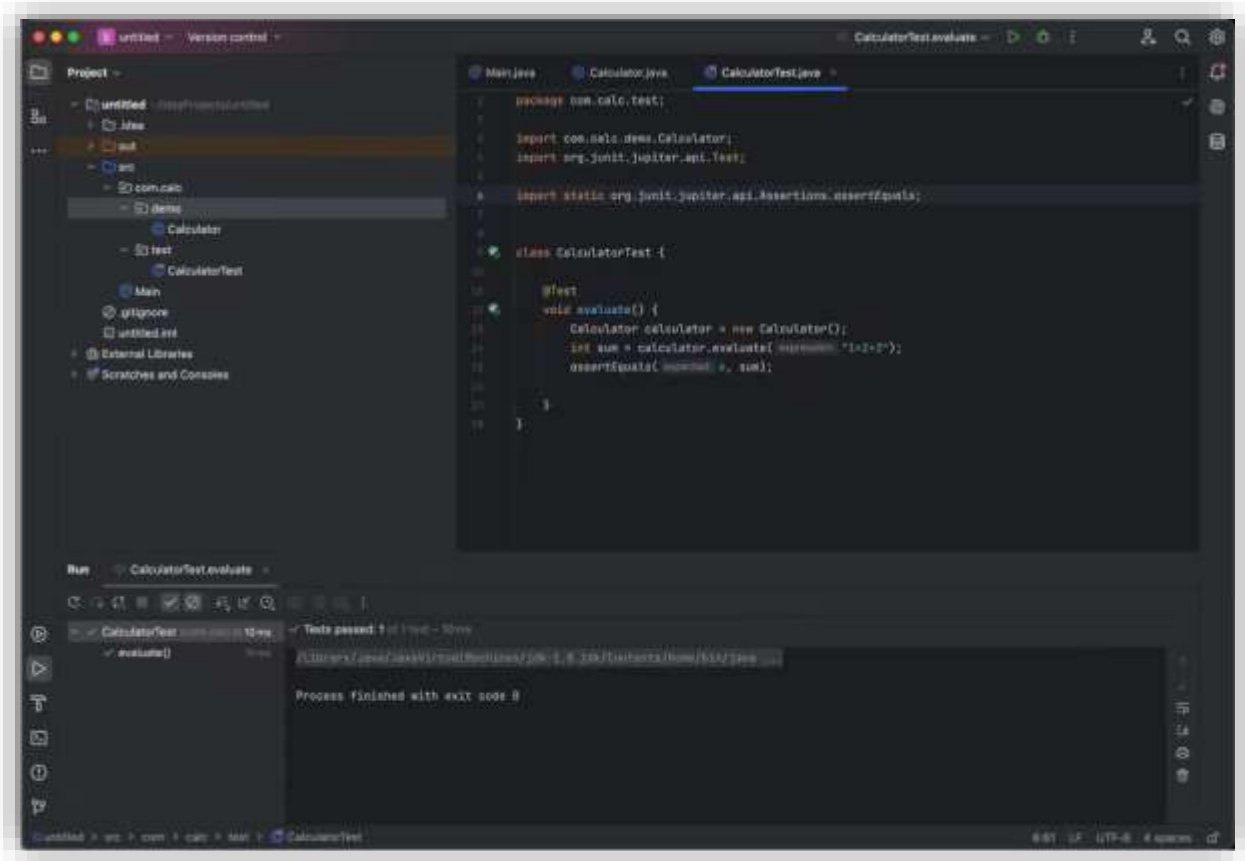
- **IDEA下使用JUnit https://junit.org/junit4/**
  - 方法3：在需要测试的类或接口名称上使用 Ctrl+Shift+t，然后选择创建测试。





JUnit的安装与配置 IDEA

- **IDEA下运行JUnit**
  - 在测试方法点击 IDEA 运行图标（或用 Ctrl+Shift+F10）运行单元测试。



JUnit4测试用例：小例子

```
public class Calculator {
  public int evaluate(String expression) {
    int sum = 0;
    for (String summand: expression.split("\\+"))
      sum += Integer.valueOf(summand);
    return sum;
  }
}

------

import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class CalculatorTest {
  @Test
  public void evaluatesExpression() {
    Calculator calculator = new Calculator();
    int sum = calculator.evaluate("1+2+3");
    assertEquals(6, sum);
  }
}
```

单元测试部分：实验评判标准

- 是否可正确配置JUnit/EclEmma；
- 所设计的测试用例是否满足要求；
- 是否正确书写了JUnit测试代码并执行，以获得测试结果；
- 所涉及的测试用例的覆盖度；
- 使用Git情况。

提交方式

- **请遵循实验报告模板撰写。**

- **提交日期：第14周周日晚(6月1日 23:55)**
- **提交两个个文件到头歌平台：**
  - 实验报告：命名规则“学号-Lab3-report.doc”，具体请参见模板
  - 增加了测试用例的源代码打包：命名规则“学号-Lab3-code.zip”
  - 同组的两人要分别提交
  - 确保GitHub上有本次实验之后的全部代码，包含所有测试用例

- **第13周和第14周的实验课上，请实验教师/TA现场检查测试执行情况。**

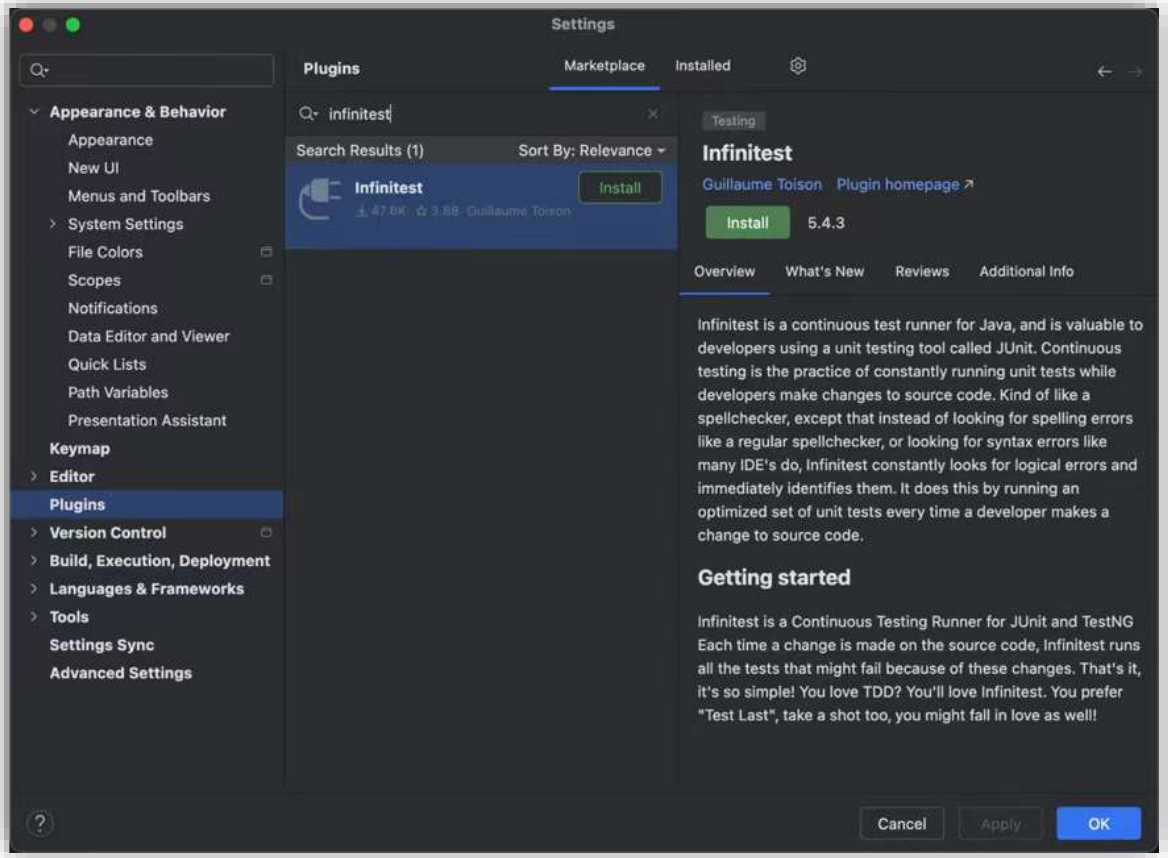
结束

Infinitest 持续测试

- **Infinitest：针对Eclipse的持续测试插件 (Continuous Testing plugin)；**
- **每当源代码发生变化后，所有受影响的测试用例都会被自动重新执行。**
- **https://infinitest.github.io**

在idea中安装infinitest

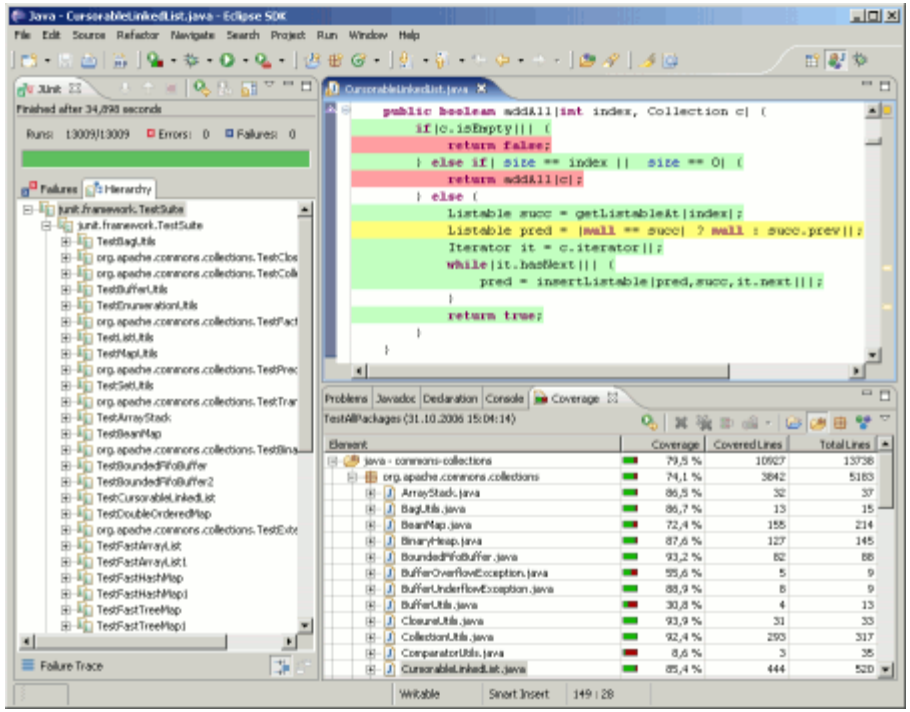
Settings -> Plugins -> Browse repositories -> 查找infinitest -> Install -> Restart idea



EclEmma

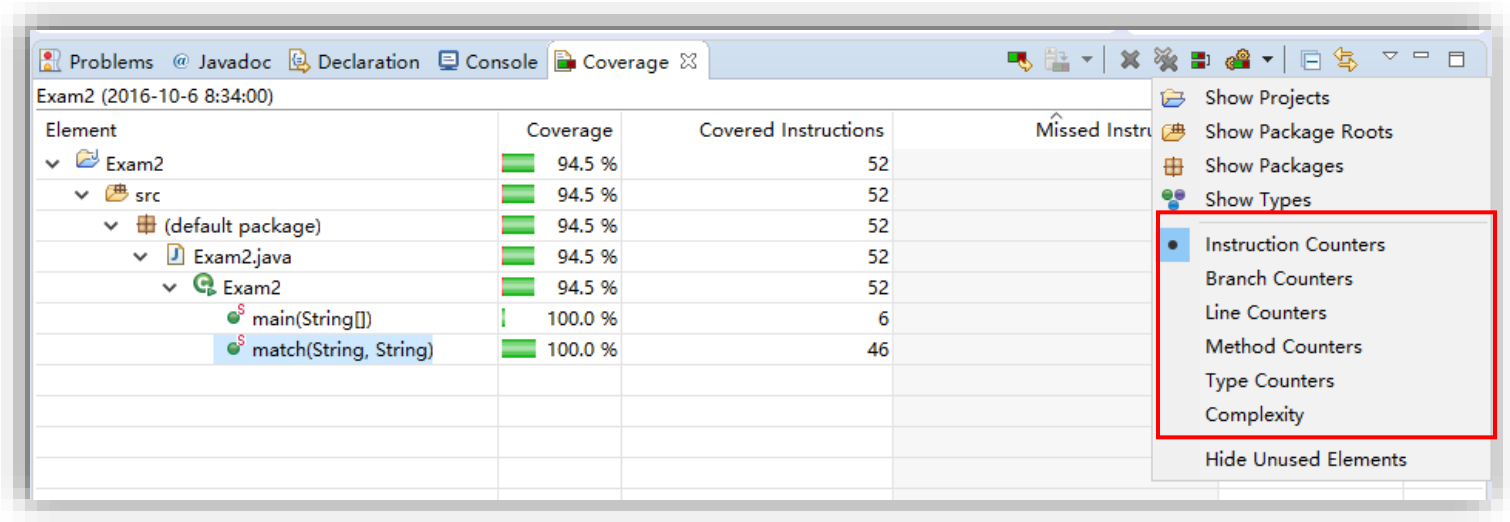
- 从http://www.eclemma.org/download.html 下载并配置到Eclipse中；
- 帮助教程：
  - http://eclemma.org/userdoc/index.html
  - http://www.ibm.com/developerworks/cn/java/j-lo-eclemma

- 针对前面你所设计的jUnit测试用例，使用EclEmma分析它们的代码覆盖度，并据此进一步完善测试用例，确保覆盖度尽可能高。



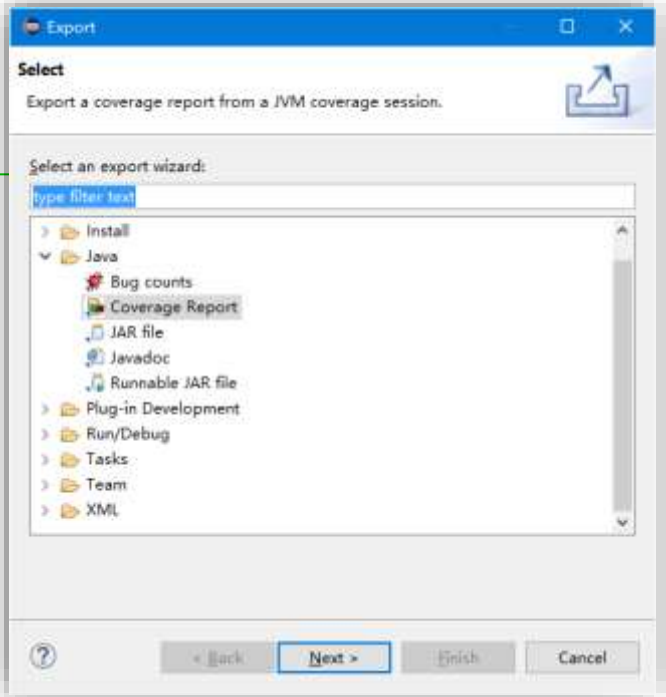
EclEmma

- **在Coverage视图的下拉菜单中选择不同的“覆盖度标准”**
  - Instruction counters：语句覆盖
  - Branch counters：判定覆盖
  - Complexity：基本路径覆盖



EclEmma

- **导出覆盖度分析报告：**
  - 在Coverage视图的特定Element上右键菜单上选择“export session”，打开以下对话框，选择“Coverage Report”；



JaCoCo > org.jacoco.examples > org.jacoco.examples

org.jacoco.examples

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
ReportGenerator		0%		0%	7	7	28	28	6	6	1	1
ExecutionDataServer.Handler		0%		0%	5	5	26	26	4	4	1	1
MBeanClient		0%		n/a	2	2	14	14	2	2	1	1
ExecutionDataClient		0%		n/a	2	2	14	14	2	2	1	1
ExecutionDataServer		0%		n/a	2	2	7	7	2	2	1	1
CoreTutorial.TestTarget		0%		0%	5	5	7	7	3	3	1	1
ClassInfo		95%		100%	1	5	2	17	1	4	0	1
ExecDump		93%		100%	1	8	2	23	1	5	0	1
CoreTutorial		97%		100%	1	11	2	44	1	6	0	1
ExecDump.new IExecutionDataVisitor() {...}		100%		n/a	0	2	0	3	0	2	0	1
CoreTutorial.MemoryClassLoader		100%		100%	0	4	0	8	0	3	0	1
ExecDump.new ISessionInfoVisitor() {...}		100%		n/a	0	2	0	3	0	2	0	1
Total	430 of 993	56%	8 of 26	69%	26	55	102	194	22	41	6	12

Code Coverage Report for JaCoCo 0.7.8-SNAPSHOT