



Lab 6: 测试用例设计与jUnit单元测试



实验目标

- 按Lab1和Lab4的分组，两人共同完成实验，针对本组在Lab4之后git仓库中的最新版本，展开本次实验；
- 针对Lab4评审和优化过的程序，设计白盒测试用例；
- 针对Lab1中包含的需求，设计黑盒测试用例；
- 在jUnit环境下撰写测试代码并执行测试；
- 私用Infinittest进行持续测试；
- 使用EclEmma统计测试的覆盖度；
- 让自己的GitHub项目具备持续集成的能力(Travis CI) -可选

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(四选一):
 - `type createDirectedGraph(String filename)`: 生成有向图
 - `String queryBridgeWords(type G, String word1, String word2)`: 查询桥接词
 - `String generateNewText(type G, String inputText)`: 根据bridge word生成新文本（其内部要调用`queryBridgeWords`查询桥接词函数）
 - `String calcShortestPath(type G, String word1, String word2)`: 计算两个单词之间的最短路径
- 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

- 在本组Lab1的git仓库里，建立新的git分支，命名为lab6b (b代表black-box testing);
- 针对每个测试用例撰写testcase并放入testsuite;
- 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lab6b合并到master分支，并推送至GitHub。

Step 3: 设计白盒测试用例

- 针对lab6b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

- 在本组Lab1的git仓库里，建立新的git分支，命名为lab6w (w代表white-box testing);
- 针对每个测试用例撰写testcase并放入testsuite;
- 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lab6w合并到master分支，并推送至GitHub。

Step 5: Infinitest持续测试

- 在Eclipse中安装配置Infinitest插件;
- 启动Infinitest, 体验持续测试: 当代码发生变化时, 项目中包含的各测试用例被自动执行, 开发者可实时感知到代码修改是否影响之前的功能, 以便于及时调整刚刚修改的代码。

Step 6: 使用EclEmma统计测试覆盖度

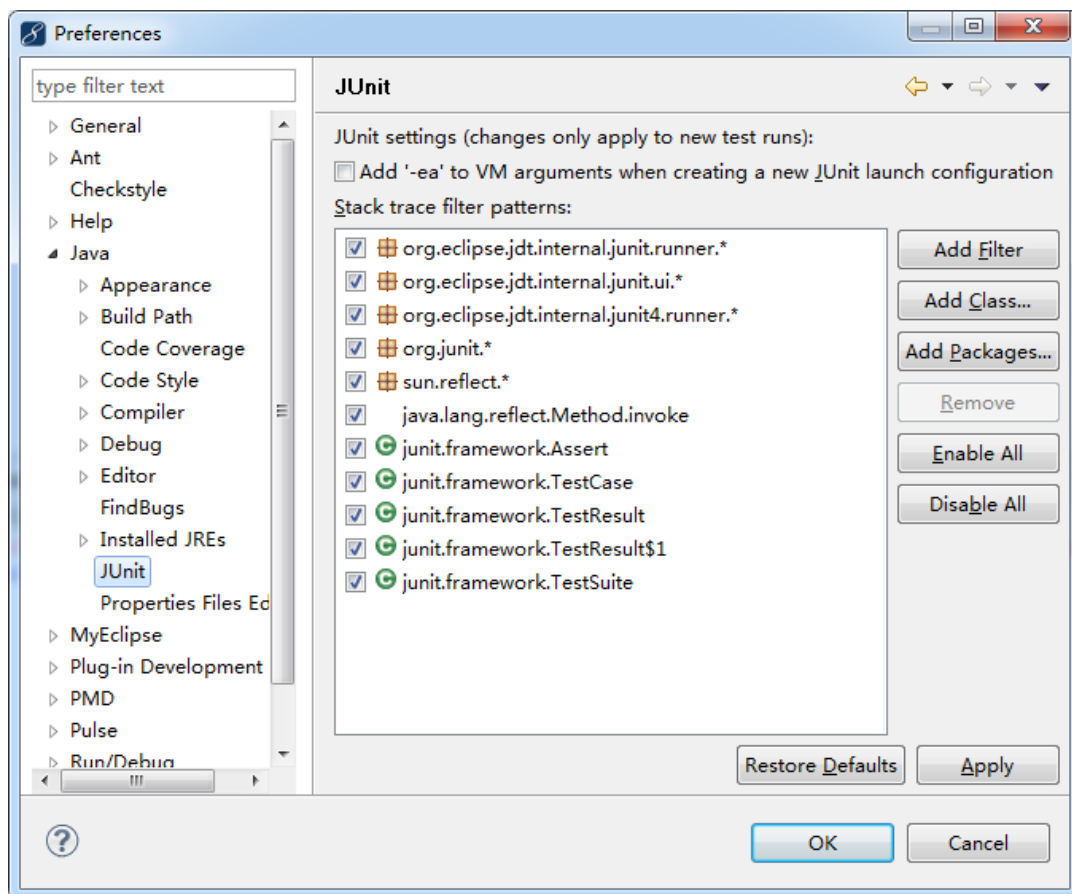
- 测试的覆盖率是测试质量的一个重要指标;
- EclEmma是一个帮助开发人员考察测试覆盖率的Eclipse开源插件。
- 在测试过程中，当运行测试程序，EclEmma可自动分析出被测程序的各行代码被覆盖的情况;
- 代码被覆盖得越全面，测试质量就越好。
- 从EclEmma导出覆盖度分析报告，观察语句覆盖度(instruction counters)、判定覆盖度(branch counters)、基本路径覆盖度(complexity)。

Step 7: 使用Travis CI/Jenkins实现持续集成(可选)

- 在自己的GitHub项目中配置Travis CI或Jenkins;
- 对代码做简单修改, 重新推送, 触发CI, 演示结果。
- 需要自学。

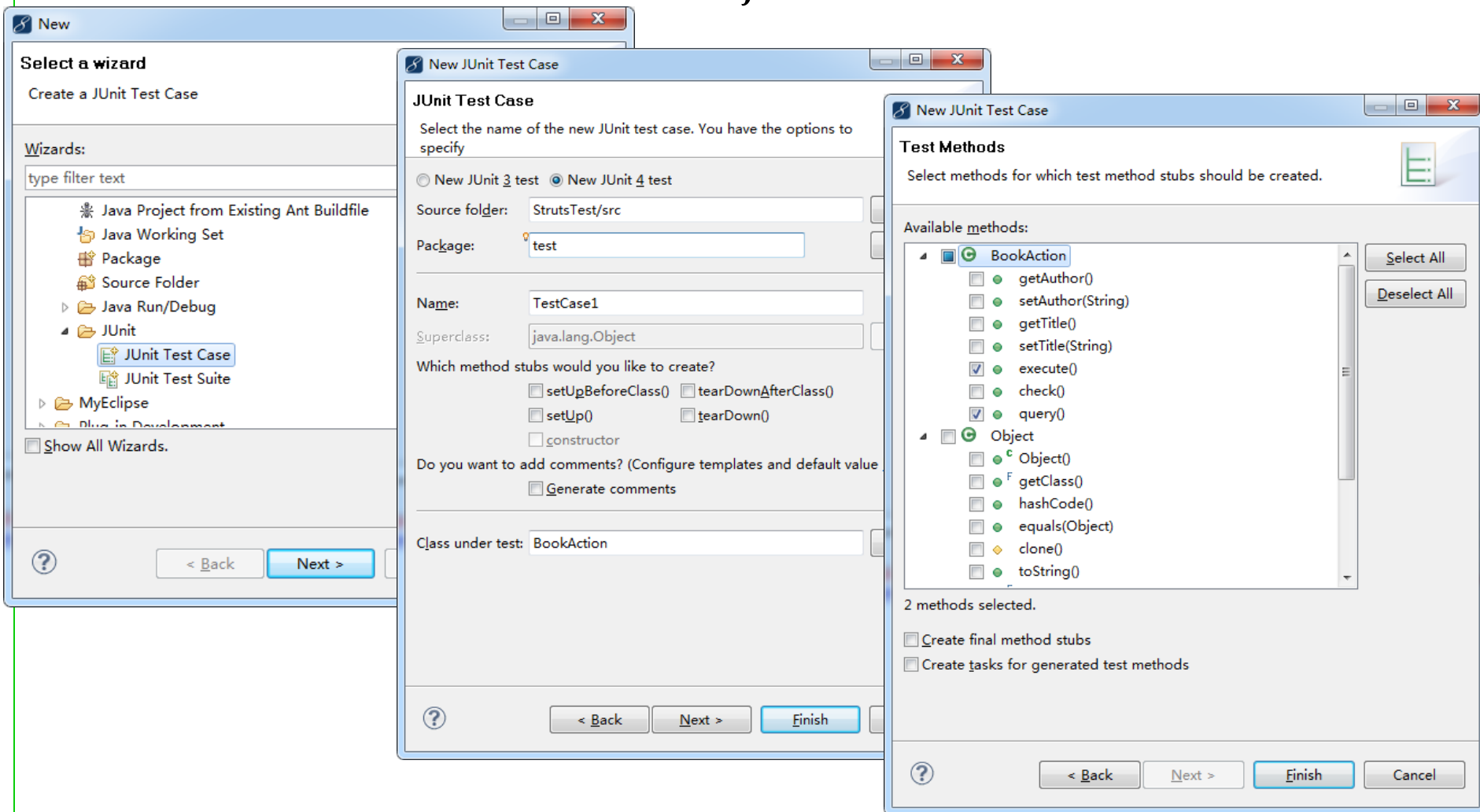
jUnit的安装与配置

- 大部分Eclipse版本中已经集成了jUnit;
- 也可自己下载jUnit包并配置到Eclipse中。
 - <http://www.junit.org>
 - 把相应的包加载到项目中



如何编写JUnit测试用例

- 针对一个存在的项目，在其中新建JUnit测试用例或测试套件；



注意JUnit的版本

■ JUnit3:

New JUnit Test Case

JUnit Test Case

Type name is empty.

☒ New JUnit 3 test ☐ New JUnit 4 test

Source folder: StrutsTest/src

Package: (default)

Name:

Superclass: junit.framework.TestCase

Which method stubs would you like to create?

☐ setUpBeforeClass() ☐ tearDownAfterClass()
☒ setUp() ☐ tearDown()
☐ constructor

Do you want to add comments? (Configure templates and default value [here](#))

☒ Generate comments

Class under test:

■ JUnit4:

New JUnit Test Case

JUnit Test Case

Type name is empty.

☐ New JUnit 3 test ☒ New JUnit 4 test

Source folder: StrutsTest/src

Package: (default)

Name:

Superclass: java.lang.Object

Which method stubs would you like to create?

☐ setUpBeforeClass() ☐ tearDownAfterClass()
☒ setUp() ☐ tearDown()
☐ constructor

Do you want to add comments? (Configure templates and default value [here](#))

☒ Generate comments

Class under test:

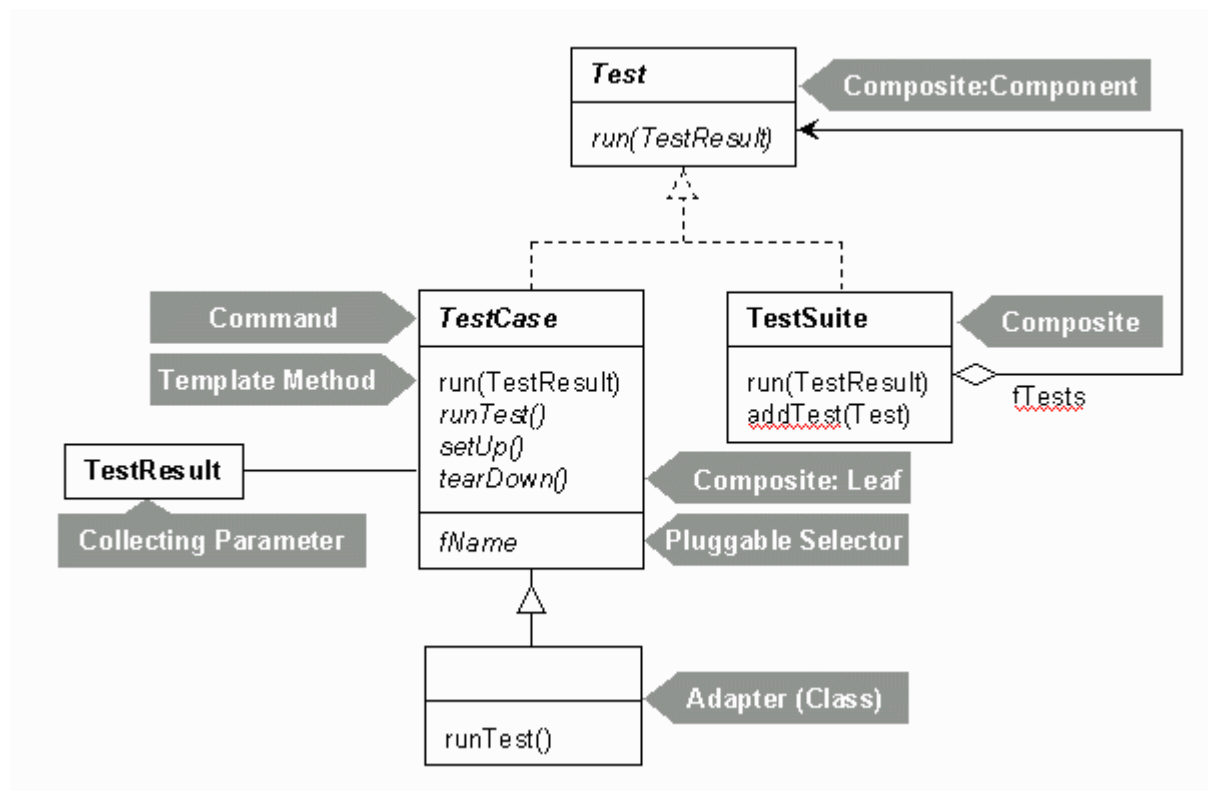
setUp(): 为测试做准备, 完成初始化;
tearDown(): 清理测试现场。

JUnit 3 vs JUnit 4

- JUnit4简化了测试用例的编写难度，不再使用TestCase等基础类库，而是充分利用Java5的Annotation特性。
- 在JUnit3中：单元测试类必须继承自TestCase，要测试的方法必须以test开头。
- 在JUnit4中：不再要求测试类必须继承自TestCase基类，测试方法也不必以test开头，只要以@Test元数据来描述即可。
- 请使用JUnit4完成本次实验
- <http://junit.org/junit4>

jUnit3 的类结构

- **TestCase**类：代表一个测试用例，编写相应的测试代码；
- **TestSuite**类：多个**TestCase**组成的套件，可以整体运行。



jUnit3测试用例：小例子

被测程序：

```
public class Calculuator {  
    public double add (double n1, double n2) {  
        return n1 + n2;  
    }  
}
```

测试用例：

```
import junit.framework.TestCase;  
  
public class TestCalculuator extends TestCase {  
    public void testAdd(){  
        Calculuator calculator=new Calculuator();  
        double result=calculator.add(1,2);  
        assertEquals(3,result,0);  
    }  
}
```

输入数据

期望结果

jUnit4测试用例：小例子

被测程序：

```
public class Calcuator {  
    public double add (double n1, double n2) {  
        return n1 + n2;  
    }  
}
```

测试用例：

```
import junit.framework.TestCase;  
import org.junit.Test;  
  
public class TestCalcuator extends TestCase{  
    @Test  
    public void testAdd(){  
        Calcuator calcuator=new Calcuator();  
        double result=calcuator.add(1,2);  
        assertEquals(3,result,0);  
    }  
}
```

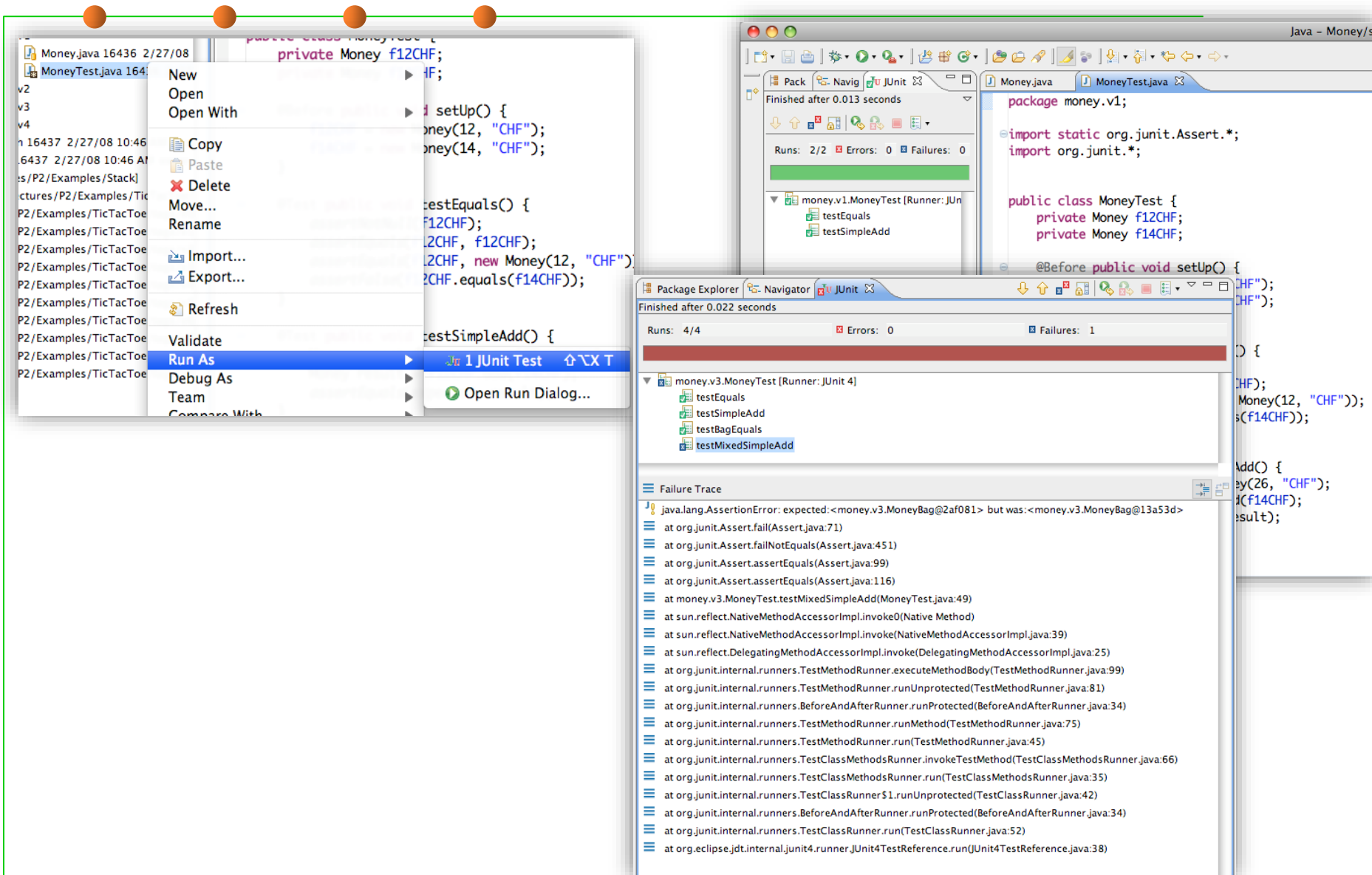

将多个测试用例放在同一个Suite中执行

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)
@Suite.SuiteClasses({
    TestFeatureLogin.class,
    TestFeatureLogout.class,
    TestFeatureNavigate.class,
    TestFeatureUpdate.class
})

public class FeatureTestSuite {
    // the class remains empty,
    // used only as a holder for the above annotations
}
```

在Eclipse中运行JUnit测试用例



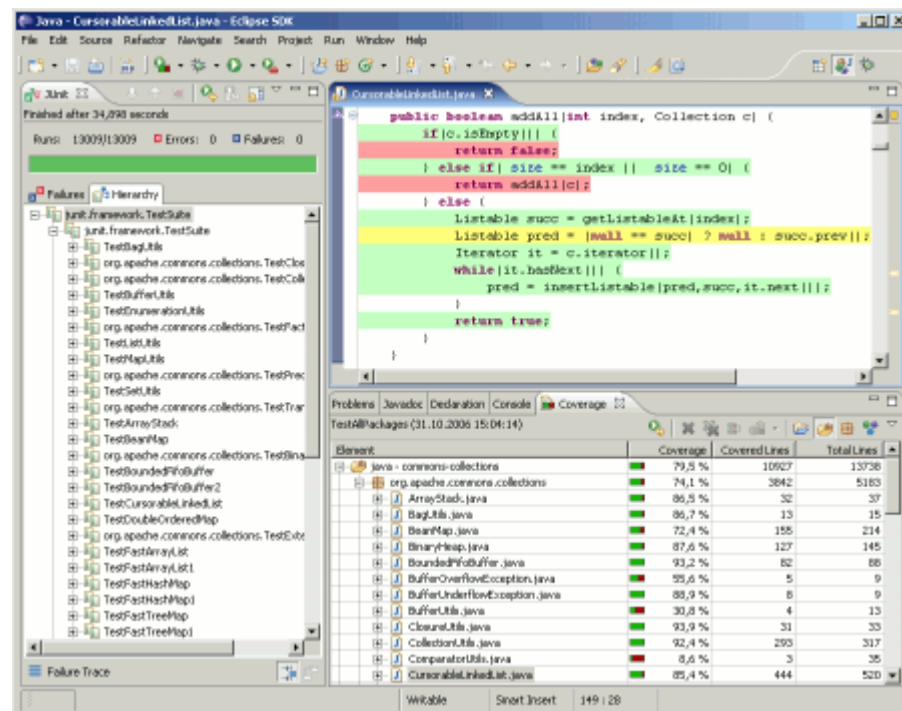
Infinitest 持续测试



- **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>

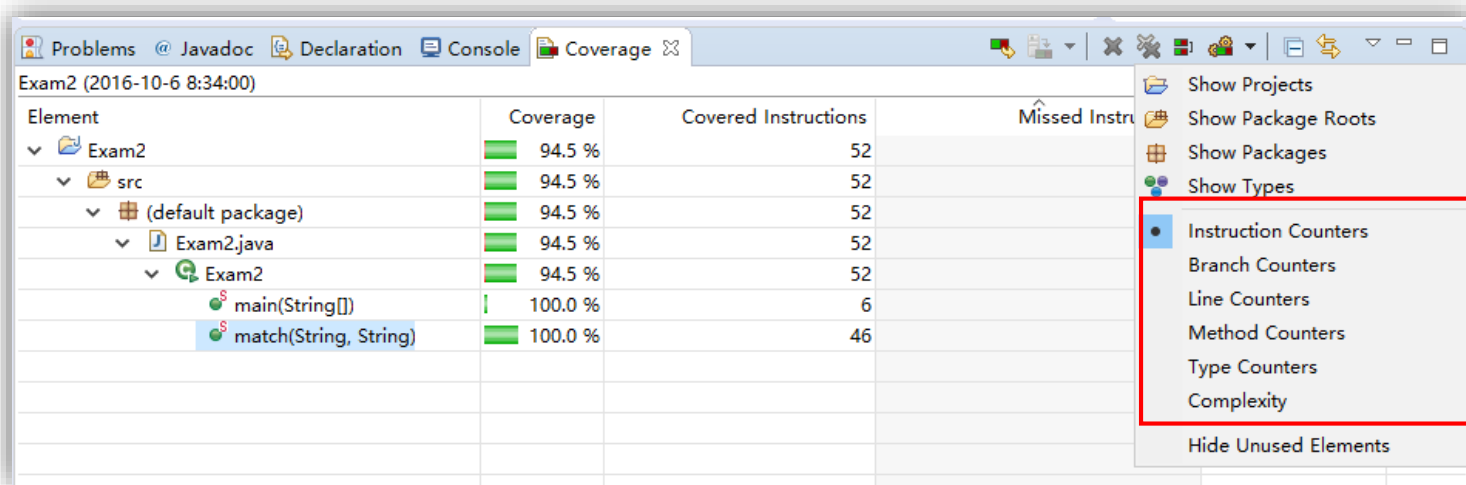
EclEmma

- 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

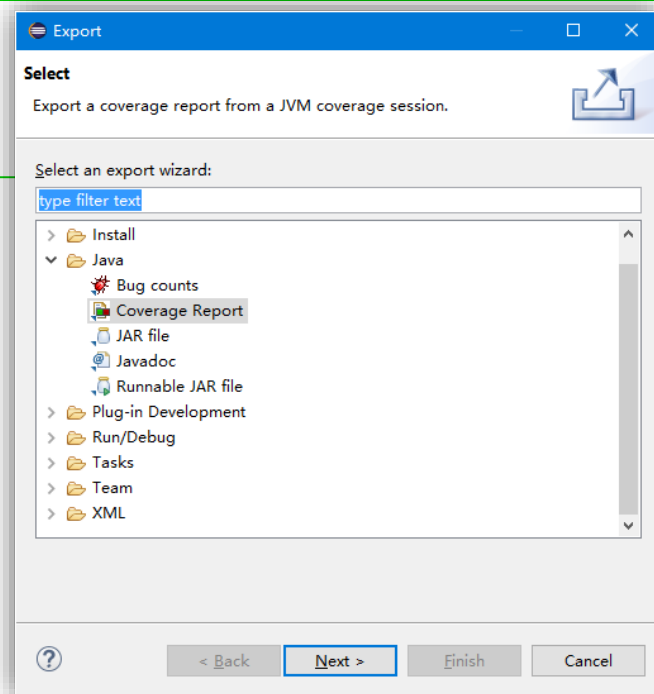
- 在Coverage视图的下拉菜单中选择不同的“覆盖度标准”
 - Instruction counters: 语句覆盖
 - Branch counters: 判定覆盖
 - Complexity: 基本路径覆盖



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	<div><div></div></div>	0%	<div><div></div></div>	0%	7	28	6	1
ExecutionDataServer.Handler	<div><div></div></div>	0%	<div><div></div></div>	0%	5	26	4	1
MBearClient	<div><div></div></div>	0%	<div><div></div></div>	n/a	2	14	2	1
ExecutionDataClient	<div><div></div></div>	0%	<div><div></div></div>	n/a	2	14	2	1
ExecutionDataServer	<div><div></div></div>	0%	<div><div></div></div>	n/a	2	7	2	1
CoreTutorial.TestTarget	<div><div></div></div>	0%	<div><div></div></div>	0%	5	7	3	1
ClassInfo	<div><div></div></div>	95%	<div><div></div></div>	100%	1	2	1	0
ExecDump	<div><div></div></div>	93%	<div><div></div></div>	100%	1	2	1	0
CoreTutorial	<div><div></div></div>	97%	<div><div></div></div>	100%	1	2	1	0
ExecDump.new IExecutionDataVisitor() {...}	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	0	0	0
CoreTutorial.MemoryClassLoader	<div><div></div></div>	100%	<div><div></div></div>	100%	0	0	0	0
ExecDump.new ISessionInfoVisitor() {...}	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	0	0	0
Total	430 of 993	56%	8 of 26	69%	26	102	22	6

Code Coverage Report for JaCoCo 0.7.8-SNAPSHOT

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(EclEmma);
- 使用git;

提交方式

- 请遵循实验报告模板撰写。
- 提交日期：第11周周一晚(11月13日 23:55)
- 提交两个文件到CMS：
 - 实验报告：命名规则“学号-Lab6-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-Lab6-report.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9周和第10周的实验课上，请TA现场检查测试执行情况。



哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

結束

