



上海交通大学  
SHANGHAI JIAO TONG UNIVERSITY



# Chapter 4

*Unit Test environment and tools*  
*Junit and EMMA*





1. Java
  2. Eclipse
  3. JUNIT
  4. EMMA
-



## 4.1 Java

### Why is Java ?

- Easy to get source code and documentation
- Completed test environment
- More free test tools

### 安装JDK ( Java Development Kit )

- JDK 是整个Java的核心，包括了Java运行环境，Java工具和Java基础的类库。
- 没有JDK的话，无法安装或者运行java程序。



- ④ JDK共有3个版本，分别为：
    - SE(Java SE), standard edition, 标准版
    - EE(Java EE), enterprise edition, 企业版
    - ME(Java ME), micro edition, 主要用于移动设备、嵌入式设备上的java应用程序
  
  - ④ 推荐下载JDK版本：Java SE Development Kit 8u201
  - ④ 下载地址：
    - <https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>
  
  - ④ 目前JDK的最新版本为：Java SE Development Kit 12。
  - ④ 下载地址：
    - <https://www.oracle.com/technetwork/java/javase/downloads/jdk12-downloads-5295953.html>
-



## 安装与配置

- 下载与安装，假设安装到C:\Program Files\Java\jdk1.8.0\_60
- 配置PATH：相当于C++中的GCC配置
  - 我的电脑点右键，选择“属性”，选择“高级”标签，进入“环境变量设置，在系统变量里找到PATH变量，选择→编辑，里面已经有很多的变量值，变量值的最前面加上C:\Program Files\Java\jdk1.8.0\_60\bin;
- 配置CLASSPATH：寻找标准类库位置
  - 我的电脑点右键，选择“属性”，选择“高级”标签，进入环境变量设置，在系统变量那一栏中点新建→CLASSPATH，并且设置
  - classpath=.; C:\Program Files\Java\jdk1.8.0\_60\lib;C:\Program Files\Java\jdk1.8.0\_60\lib\tools.jar;



## 4.2 Eclipse

---

Eclipse is a kind of universal platform – an open extensible IDE for anything and nothing in particular. It provides a feature-rich development environment that allows the developer to efficiently create tools that integrate seamlessly into the Eclipse Platform.

[www.eclipse.org](http://www.eclipse.org)

---



## 使用Eclipse编写Java程序步骤：

1. 创建一个Java项目
2. 创建” source folder”命名为src，创建包
3. 创建Java源程序
4. 编译Java源程序
5. 运行程序



示例：HelloJava.java



[Example of Eclipse](#)

---



## 4.3 JUNIT

- JUnit是由 Erich Gamma 和 Kent Beck 编写的一个回归测试框架（regression testing framework）。
- JUnit测试是程序员开发测试工具，即所谓白盒测试，因为程序员知道被测试的软件如何（How）完成功能和完成什么样（What）的功能。JUnit是一套框架，继承TestCase类，就可以用JUnit进行自动测试了。
- <http://www.junit.org>





# JUNIT

---

- ④ For application developers, testing forms an integral part of the development life-cycle.
  - ④ As old code is modified for reasons ranging from adding new functionality to speed optimizations, the risk of dependent functionality getting broken increases.
  - ④ Tests that are written once, and can be run repeatedly is the only solution to avoid manually QA testing.
  - ④ To write these tests, for applications written in java, JUnit provides a solid framework for writing unit tests. Many JUnit integrations also exist to integrate your JUnit tests into build and development tools.
-



## JUNIT 安装 (省略)

---

- ① 1. 从<http://www.junit.org/>下载JUNIT包  
JUNIT.JAR
- ② 2. 将JAR文件拷贝到JDK的LIB目录,  
C:\Java\jdk1.8.0\_60\lib.。
- ③ 3. 在CLASSPATH 目录下增加一条 c:\Java\jdk  
1.8.0\_60\lib\junit.jar;
- ④ 新版eclipse 中已经集成了JUnit 3 和JUnit 4。



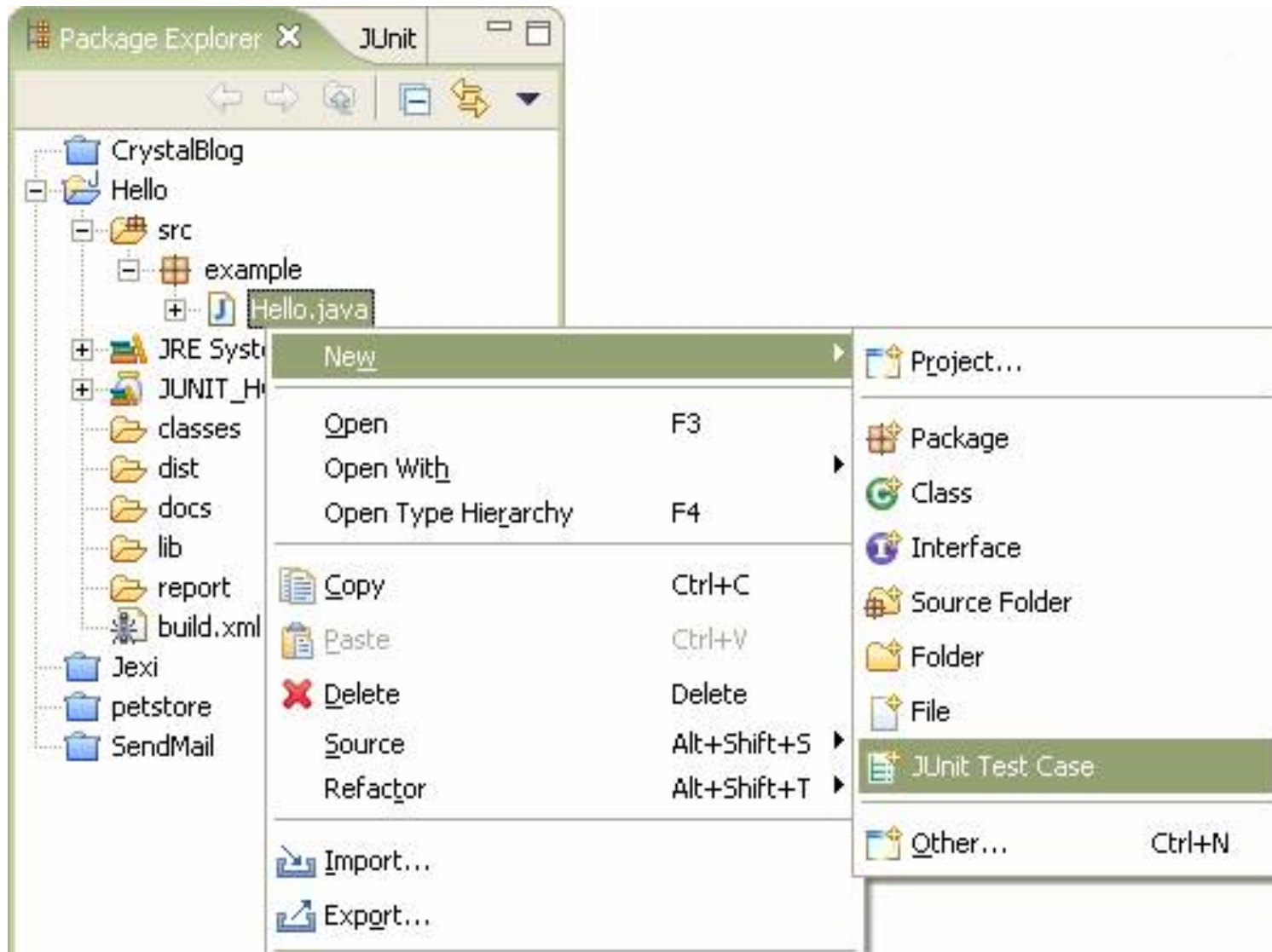
- 测试对于保证软件开发质量有着非常重要的作用，单元测试更是必不可少，JUnit是一个非常强大的单元测试包，可以对一个/多个类的单个/多个方法测试，还可以将不同的TestCase组合成TestSuit，使测试任务自动化。
- 我们创建一个Java工程，添加一个example.Hello类，首先我们给Hello类添加一个abs()方法，作用是返回绝对值：



```
Hello.java X
1 package example;
2
3 public class Hello {
4
5     public static void main(String[] args) {
6
7
8
9     public int abs(int n) {
10         return n >= 0 ? n : (-n);
11     }
12 }
```



下一步，我们准备对这个方法进行测试，确保功能正常。中Hello.java，右键点击，选择New->JUnit Test Case：





Eclipse会询问是否添加junit.jar包，确定后新建一个HelloTest类，用来测试Hello类。一般自己导入下载的junit包。

**New JUnit Test Case**

**JUnit Test Case**

Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.

Source Folder:

Package:

Name:

Superclass:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Add TestRunner statement for:

☒ setUp()

☒ tearDown()

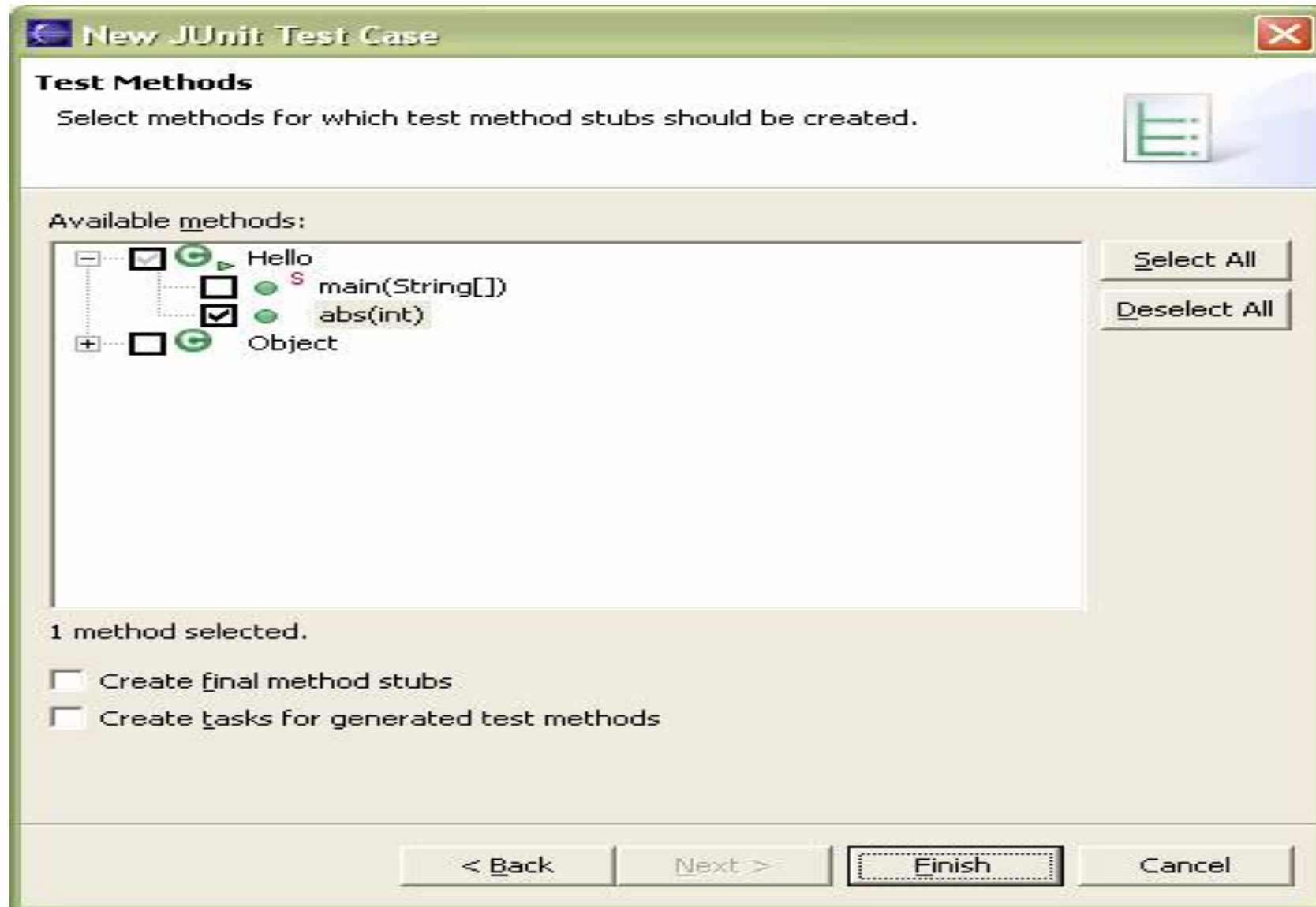
☐ constructor()

Class Under Test:

< Back   Next >   Finish   Cancel



选中setUp()和tearDown(), 然后点击 “Next”:







选择要测试的方法，我们选中abs(int)方法，完成后在HelloTest.java中输入：

```
1 package example;
2 import junit.framework.TestCase;
3
4 public class HelloTest extends TestCase {
5     private Hello hello;
6
7     protected void setUp() throws Exception {
8         super.setUp();
9         hello = new Hello();
10    }
11
12    protected void tearDown() throws Exception {
13        super.tearDown();
14    }
15
16    public void testAbs() {
17        assertEquals(hello.abs(14), 14);
18        assertEquals(hello.abs(-5), 5);
19        assertEquals(hello.abs(0), 0);
20    }
21 }
```





## JUnit会以以下顺序执行测试：（大致的代码）

```
try {  
    HelloTest test = new HelloTest(); // 建立测试类  
    实例  
    test.setUp(); // 初始化测试环境  
    test.testAbs(); // 测试某个方法  
    test.tearDown(); // 清理资源  
}  
catch...
```



- ④ setUp()是建立测试环境，这里创建一个Hello类的实例；tearDown()用于清理资源，如释放打开的文件等等。以test开头的方法被认为是测试方法，JUnit会依次执行testXxx()方法。在testAbs()方法中，我们对abs()的测试分别选择正数，负数和0，如果方法返回值与期待结果相同，则assertEquals不会产生异常。
- ④ 如果有多个testXxx方法，JUnit会创建多个XxxTest实例，每次运行一个testXxx方法，setUp()和tearDown()会在testXxx前后被调用，因此，不要在一个testA()中依赖testB()。



直接运行Run->Run As->JUnit Test, 就可以看到JUnit测试结果: 绿色表示测试通过, 只要有1个测试未通过, 就会显示红色并列出来未通过测试的方法。可以试图改变abs()的代码, 故意返回错误的结果 (比如return n+1;), 然后再运行JUnit就会报告错误。





JUnit通过单元测试，能在开发阶段就找出许多Bug，并且，多个Test Case可以组合成Test Suite，让整个测试自动完成，尤其适合于XP方法。每增加一个小的新功能或者对代码进行了小的修改，就立刻运行一遍Test Suite，确保新增和修改的代码不会破坏原有的功能，大大增强软件的可维护性，避免代码逐渐“腐烂”。

---



## 示例

---

1. 建立hello（）类
  2. 建立abs()方法；
  3. 建立单元测试 Test Case——helloTest（）
  4. 建立另一个math（）类
  5. 建立add（）和 times（）方法
  6. 建立单元测试 Test Case——mathTest（）
  7. 建立测试集 Test Suite
-



## 4.4 EMMA

- 测试覆盖率（Code Coverage）
- 测试覆盖率，简单的说，就是评价测试活动覆盖产品代码的指标。测试的目的，是确认产品代码按照预期一样工作，也可以看作是产品代码工作方式的说明文档。进一步考虑，测试覆盖率可以看作是产品代码质量的间接指标——之所以说是间接指标，因为测试覆盖率评价的是测试代码的质量，并不是产品代码的质量。
- 代码覆盖率是一种白盒测试，因为测试覆盖率是评价产品代码类内部的指标，而不是评价系统接口或规约。测试覆盖率尤其用于评价测试代码是否已经覆盖了产品代码所有的路径。



- ④ EMMA 是一种用来衡量java软件覆盖率的软件。它对于不可达代码的检测，以及验证测试集和交互式测试对程序的覆盖情况是很必要的。
  - ④ EMMA的设计目标有以下几个，
    - ④ 1. 在不增加创建或执行开销的情况下，提供丰富的覆盖分析数据；
    - ④ 2. 促进团队开发效率，同时，也能加快单元开发一测试周期；
    - ④ 3. 促进小型java软件的开发效率，同时，提高包含大量类的大型企业软件的开发。
  - ④ EMMA与其他覆盖测试工具的区别在于它面向极限编程，目的是促使循环的开发一测试软件开发模型。
-



- ① 衡量测试覆盖率的指标很多，常用的指标有：
- ② **Statement coverage**，也称作Line coverage，用于评价测试的代码语句覆盖率。
- ③ **Basic block coverage**，是Statement coverage的一个变种，它把没有一个分支的代码区域作为一个计量单位，而不是简单的代码行，用于一个if-else分支代码行数远远大于另一个的情况，在这种情况下，statement coverage指标并不适用。
- ④ **Decision coverage**（也称作Branch coverage），用于评价代码分支地测试覆盖率。
- ⑤ **Path coverage**，和Decision coverage相似，用于评价代码从开始到结束所有路径的测试覆盖率。
- ⑥ **Function coverage**，用于评价代码方法的测试覆盖率。
- ⑦ EMMA目前支持四种Coverage类型：
  - ⑧ class、method、line和basic block。





# 安装EclEmma插件

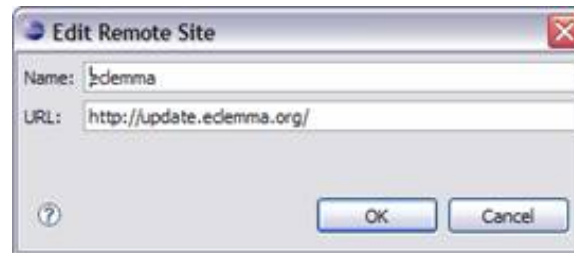
---

- ❶ EclEmma是一个基于EMMA的Java代码覆盖工具。它的目的是让你可以在Eclipse工作平台中使用强大的Java代码覆盖工具EMMA。EclEmma是非侵入式的不需要修改你的项目或执行其它任何安装，它能够在工作平台中启动像运行JUnit测试一样直接对代码覆盖进行分析。
  - ❷ 安装 EclEmma 插件的过程和大部分Eclipse 插件相同，既可以通过 Eclipse 标准的 Update 机制来远程安装 EclEmma 插件，也可以从站点下载 zip 文件并解压到 eclipse 所在的目录中。
-

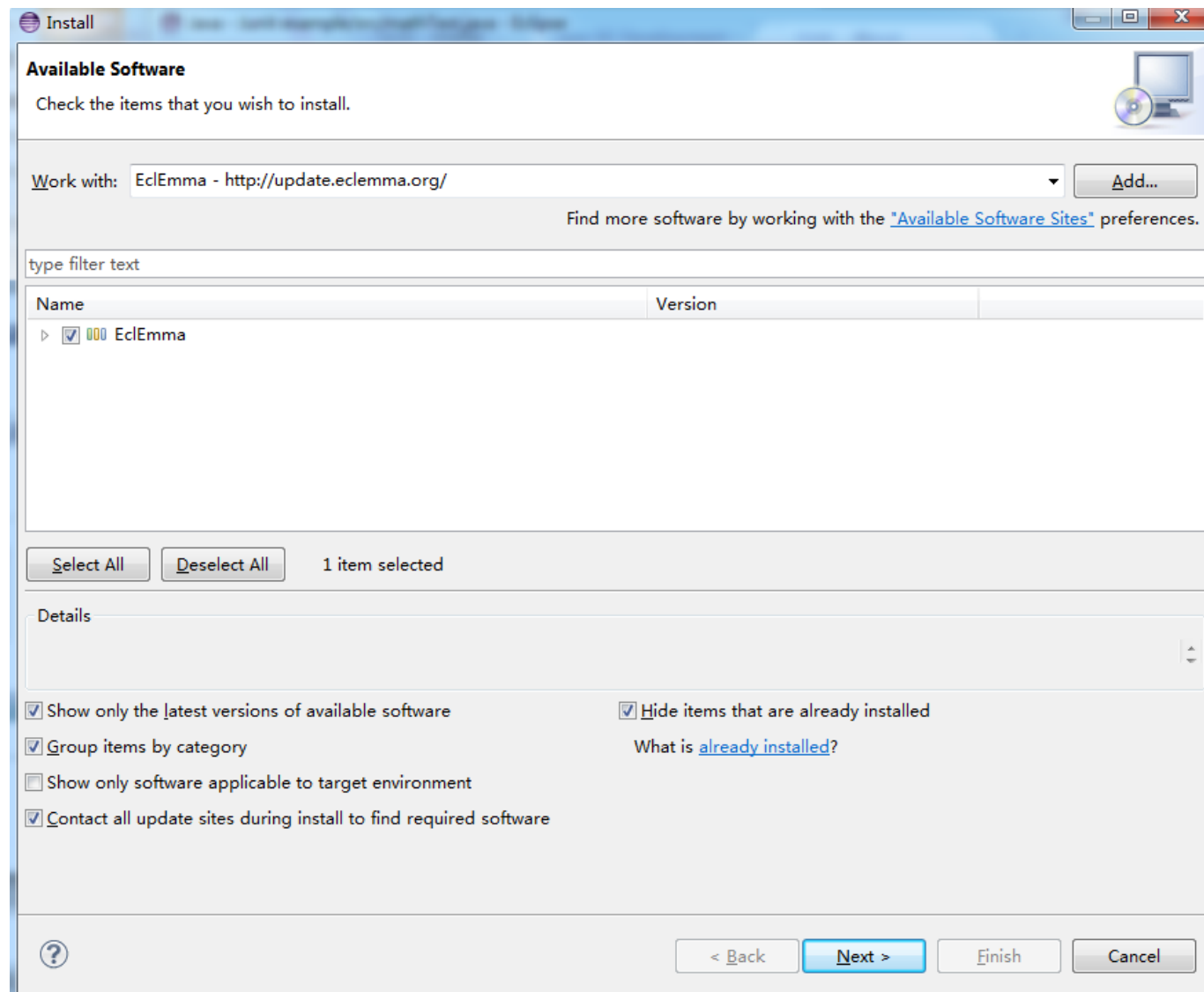
# 安装 EclEmma 插件

通过 Eclipse 的标准 Update 机制远程安装 EclEmma 插件  
添加 EclEmma 更新站点

- Help → Install New Software... → Add



- Name: EclEmma
- URL: <http://update.eclemma.org/>





# 使用EclEmma插件



安装完成并重新启动 Eclipse 之后，工具栏上应该出现一个新的按钮。





示例：

- 利用EclEmma测试程序运行覆盖率
- 生成EclEmma测试JUnit覆盖率报告
- 导出EclEmma覆盖率报告



## 4.5 IntelliJ IDEA

---

- Every aspect of IntelliJ IDEA is specifically designed to maximize developer productivity.
- Together, powerful static code analysis and ergonomic design make development not only productive but also an enjoyable experience.

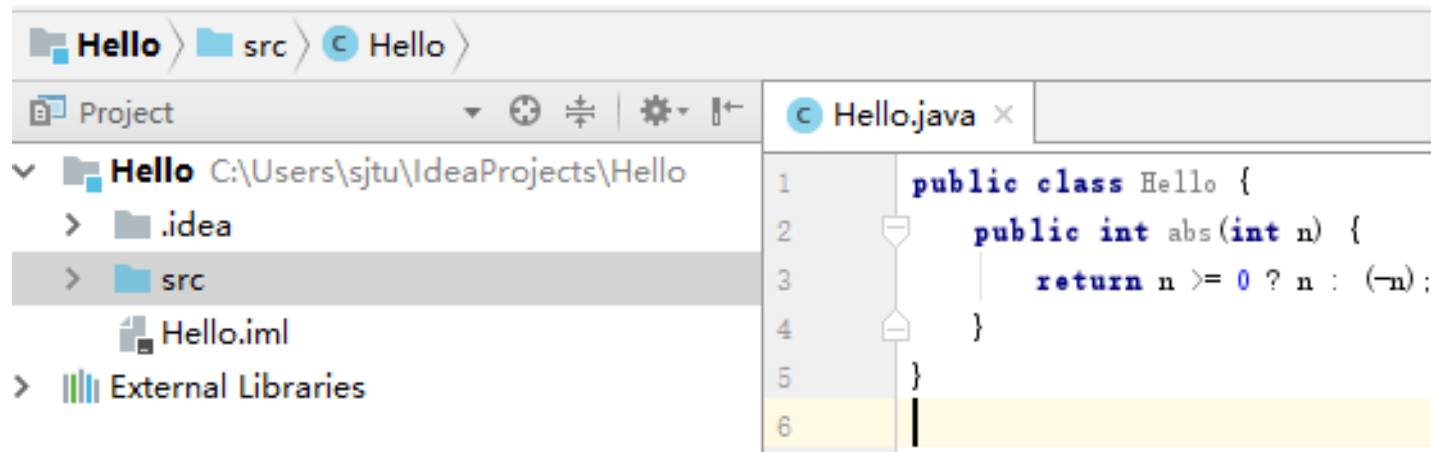




- ④ 内置JUnit4
- ④ 内置测试覆盖度工具
- ④ 测试类生成插件
- ④ 导出测试报告及测试覆盖率报告



# Junit on IntelliJ

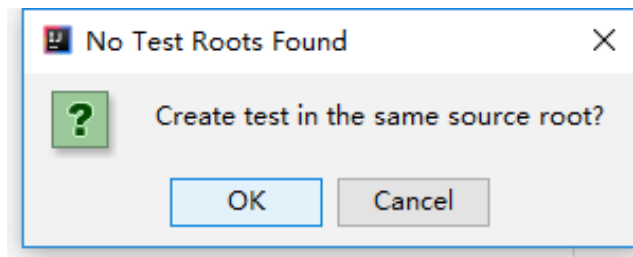






## 创建Test类

- 右键被测类.java文件，选择“Go To” - “Test”，打开测试类生成界面。弹出提示

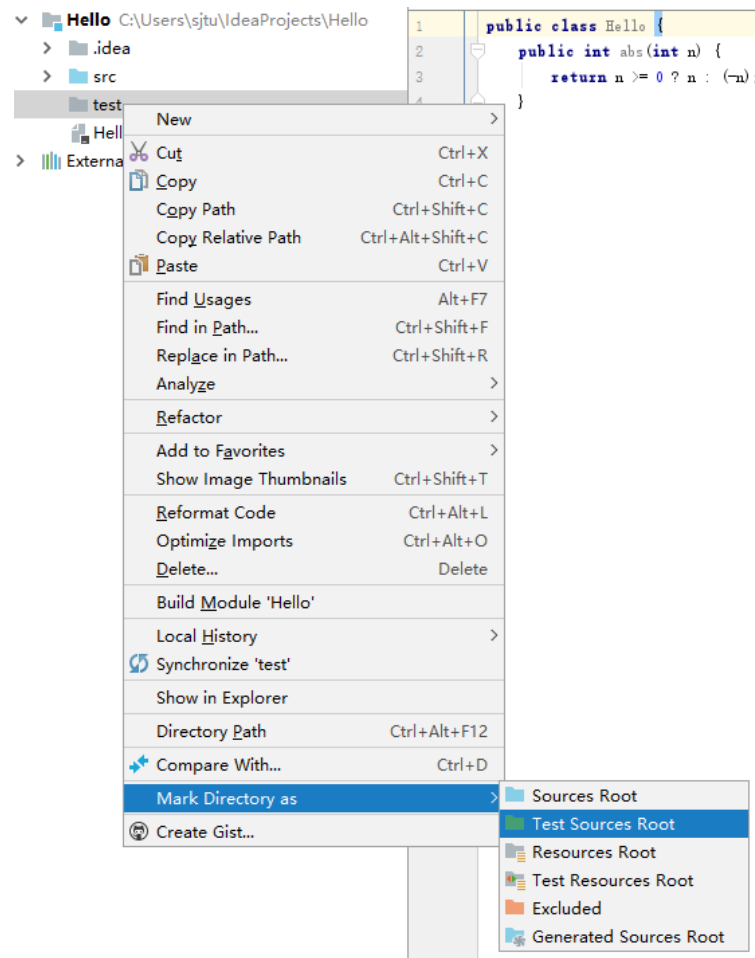


- 为了保持良好的代码结构，一般将测试类与主程序放在不同的根目录下



# 创建Test类

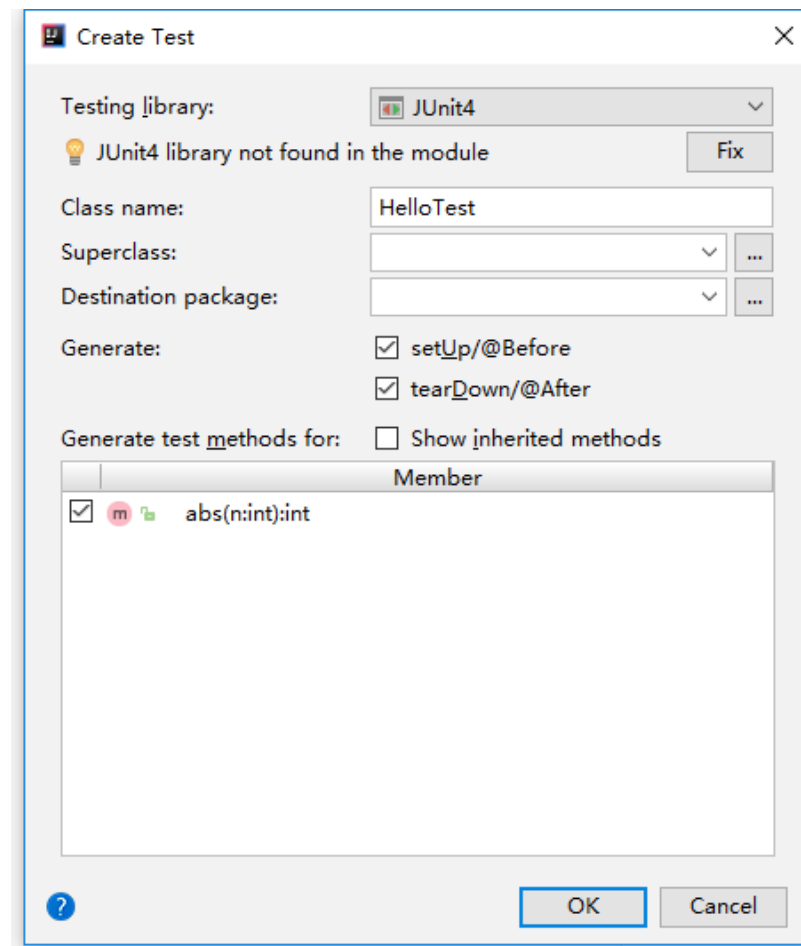
创建test文件夹，并将其设置为Test的根目录





## 创建Test类

进入创建测试类面板，注意“JUnit4 library not found in the module”的提示，点击“Fix”为工程加入JUnit包





# 创建Test类




```
1  import org.junit.After;
2  import org.junit.Before;
3  import org.junit.Test;
4
5  import static org.junit.Assert.*;
6
7  public class HelloTest {
8
9      @Before
10     public void setUp() throws Exception {
11     }
12
13     @After
14     public void tearDown() throws Exception {
15     }
16
17     @Test
18     public void abs() {
19     }
20 }
```



# 开始测试



右键test类或在test类文件中点击  图标，进行测试

-  Run 'HelloTest' Ctrl+Shift+F10
-  Debug 'HelloTest'
-  Run 'HelloTest' with Coverage

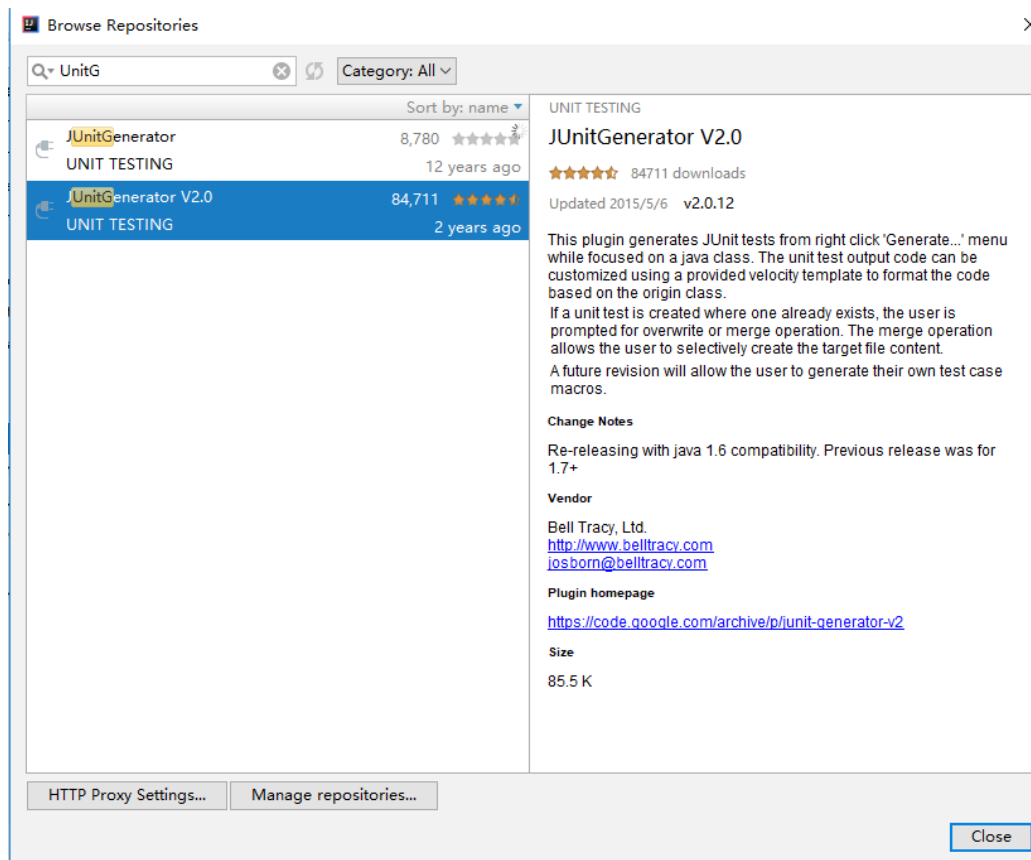
Coverage HelloTest <span>⚙️ →</span>				
↑ 100% classes, 100% lines covered in 'all classes in scope'				
	Element	Class, %	Method, %	Line, %
📁	com			
📁	java			
📁	javafx			
📁	javax			
📁	jdk			
✖	META-INF			
?	netscape			
	oracle			
	org			
	sun			
🔵	Hello	100% (1/1)	100% (1/1)	100% (2/2)



# 使用插件生成测试代码



打开File->Settings->Plugins->Browse Repositories，搜索JUnitGenerator并安装（可能要挂代理）。



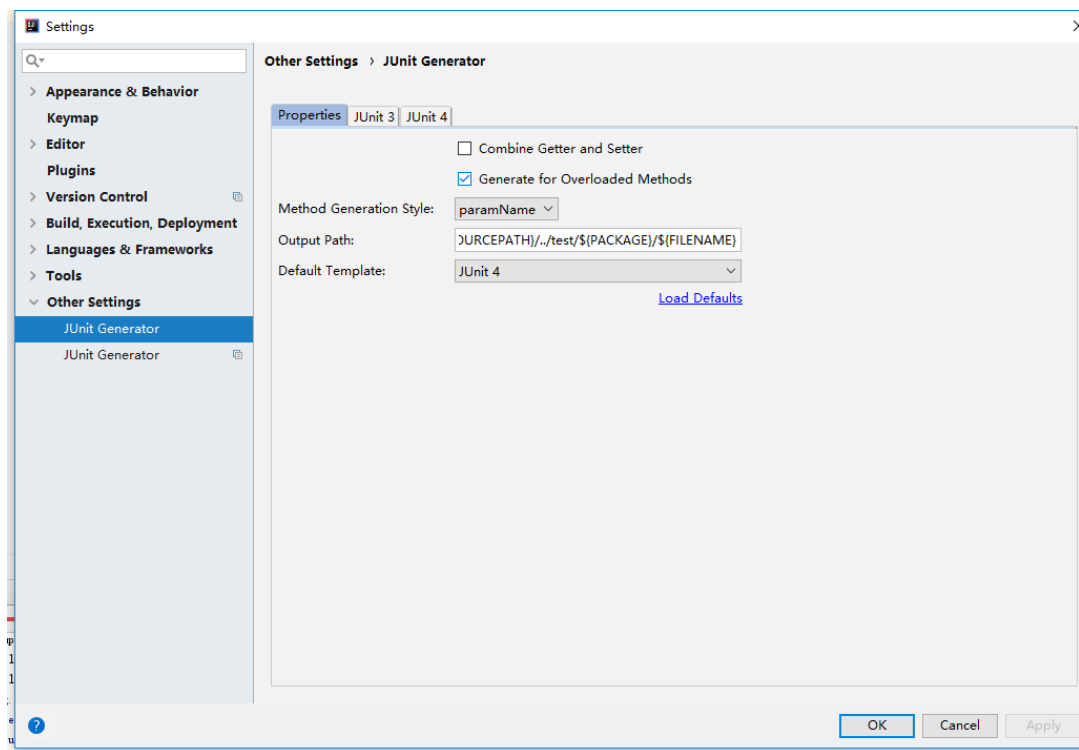


## 配置插件



在File->Settings->Other Settings里找到JUnit Generator，修改里面的设置以适应工程：

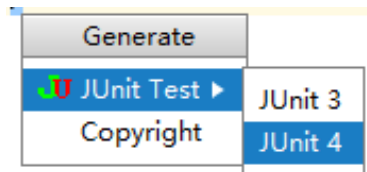
- Output Path：自动生成测试代码的输出路径
- JUnit 4：生成测试文件的模板





## 生成测试代码

在被测代码文件里右键选择Generate->JUnit Test->JUnit 3/4。或者使用快捷键Alt+Insert。



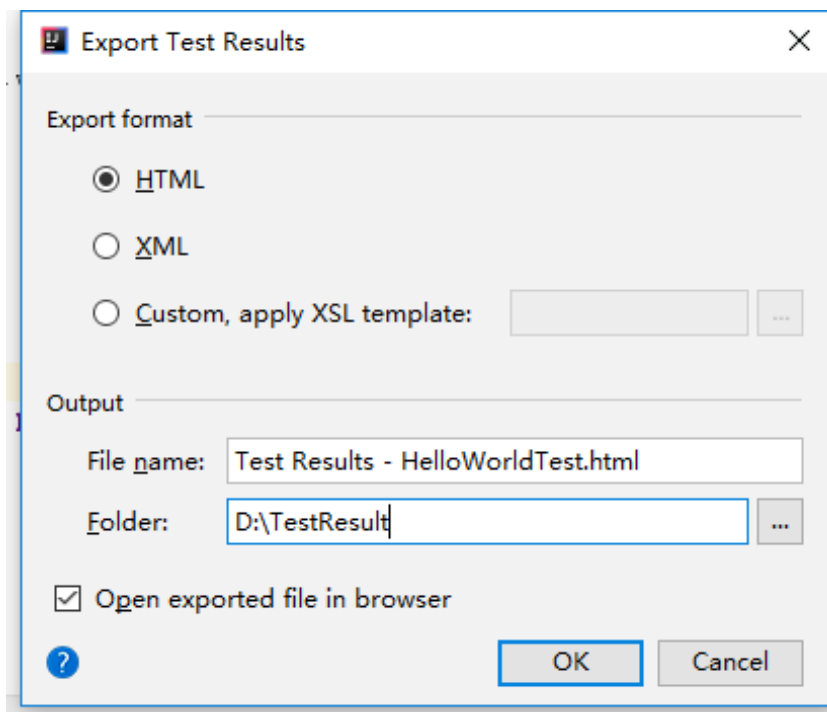
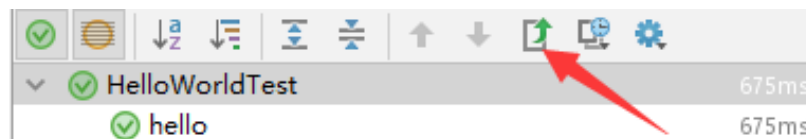
该插件会为测试类下的每个方法生成测试函数，可以使用IntelliJ自带的方式选择只为部分函数生成测试函数





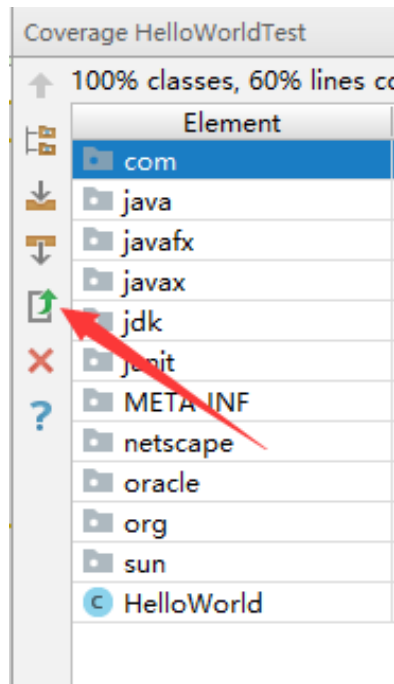
## 导出测试报告

在测试控制窗口界面点击Export Test Results，选择导出格式和路径。



# 导出测试覆盖率报告

在测试覆盖率窗口点击Generate Coverage Report，选择导出路径。



[ all classes ] [ <empty package name> ]

Coverage Summary for Class: HelloWorld (<empty package name>)

Class	Method, %	Line, %
HelloWorld	66.7% (2/ 3)	60% (3/ 5)
HelloWorld\$\$EnhancerByMockitoWithCGLIB\$\$78b37baf		
HelloWorld\$\$EnhancerByMockitoWithCGLIB\$\$78b37baf\$\$FastClassByMockitoWithCGLIB\$\$3553d01e		
HelloWorld\$\$FastClassByMockitoWithCGLIB\$\$1a2fa200		
<b>total</b>	66.7% (2/ 3)	60% (3/ 5)

```
1 public class HelloWorld {
2     public int count = 0;
3     public String hello() {
4         return this.getStr();
5     }
6
7     public String getStr() {
8         count++;
9         return "Hello world!";
10    }
11 }
```



- ④ **参数化测试**：测试用例与测试代码分离，提高测试代码效率，方便测试用例的替换。
    - JUnit4/5
    - TestNG
  - ④ **Mock（打桩）**：模拟对其他模块的调用，避免测试代码被外部代码干扰。
    - Mockito
    - PowerMock
  - ④ **变异测试**：修改源码使其产生变异体，观察变异体的测试结果，以评判测试用例的充分性。
    - Pitest
-



# 参数化测试

- JUnit4中已经包含了对参数化测试的支持，但是代码编写较为繁琐。  
JUnit5对参数化测试的接口进行了简化。

## JUnit4参数化测试代码示例

```
// annotate this test as parameterized test
@RunWith(Parameterized.class)
public class HelloWorldTest {

    // variables to be tested
    private int absInput;
    private int absResult;

    // parameterized constructor
    public HelloWorldTest(int absInput, int absResult) {
        super();
        this.absInput = absInput;
        this.absResult = absResult;
    }
}
```

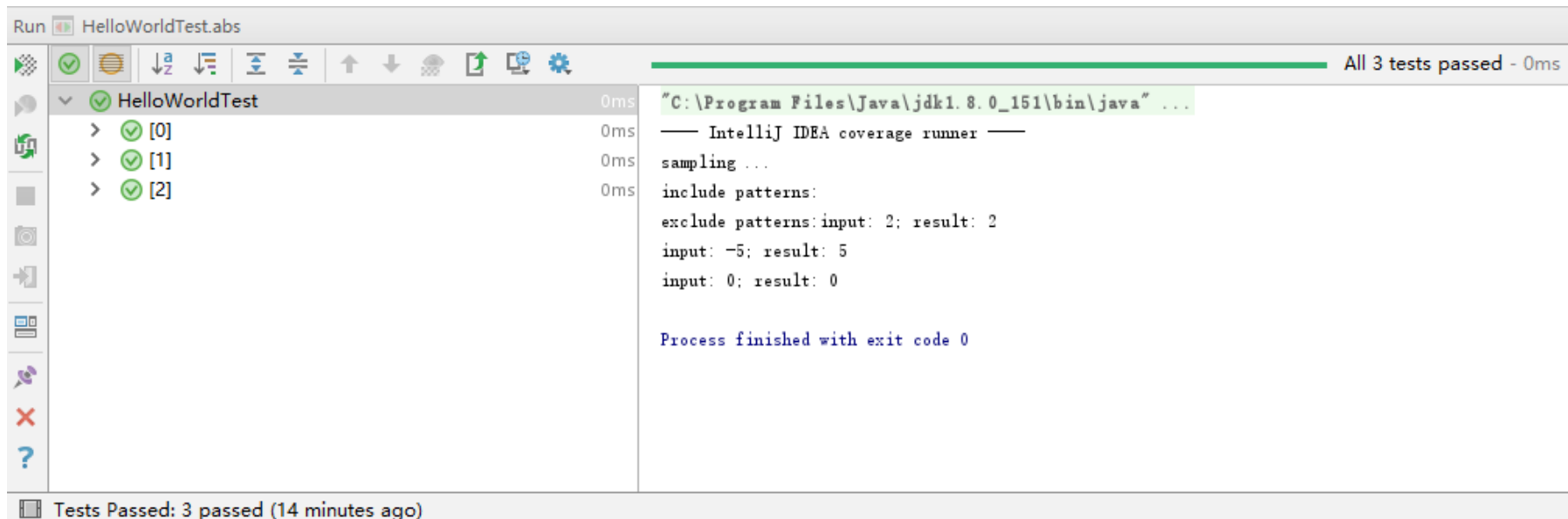
```
// prepare test data
@Parameterized.Parameters
public static Collection<Object[]> data() {
    Object[][] data = new Object[][] {
        { 2, 2 },
        { -5, 5 },
        { 0, 0 }
    };
    return Arrays.asList(data);
}

@org.junit.Test
public void abs() {
    System.out.printf("%dinput: ; result: %d%n", absInput, absResult);
    HelloWorld helloWorld = new HelloWorld();
    int actualResult = helloWorld.abs(absInput);
    assertEquals(absResult, actualResult);
}
```



# 参数化测试

- ① `@RunWith(Parameterized.class)` 注解告诉Junit这是一个参数化测试类
- ② 声明变量作为测试参数
- ③ 实现测试类的构造器，以测试变量为参数
- ④ 实现一个public static类型的函数，返回测试数据（固定写法）
- ⑤ 测试方法中使用测试参数进行测试





## JUnit5中的参数化测试

```
@ParameterizedTest
@ValueSource(ints = { 1, 2, 3 })
void testWithValueSource(int argument) {
    assertTrue(argument > 0 && argument < 4);
}
```

```
@ParameterizedTest
@CsvFileSource(resources = "/two-column.csv", numLinesToSkip = 1)
void testWithCsvFileSource(String first, int second) {
    assertNotNull(first);
    assertNotEquals(0, second);
}
```

*two-column.csv*

```
Country, reference
Sweden, 1
Poland, 2
"United States of America", 3
```



# Mock工具

- Mockito: Tasty mocking framework for unit tests in Java



- 主要接口

- mock()/@Mock: create mock
  - optionally specify how it should behave via Answer/ReturnValues/MockSettings
  - when()/given() to specify how a mock should behave
  - If the provided answers don't fit your needs, write one yourself extending the Answer interface
- spy()/@Spy: partial mocking, real methods are invoked but still can be verified and stubbed
- @InjectMocks: automatically inject mocks/spies fields annotated with @Spy or @Mock
- verify(): to check methods were called with given arguments
  - can use flexible argument matching, for example any expression via the any().
  - or capture what arguments were called using @Captor instead



## Maven导入:

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>RELEASE</version>
  </dependency>

  <dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-all</artifactId>
    <version>2.0.2-beta</version>
  </dependency>
</dependencies>
```





## ④ HelloWorld.java

```
public class HelloWorld {  
    public String hello() {  
        return this.getStr();  
    }  
  
    public String getStr() {  
        return "Test String";  
    }  
}
```

## ④ HelloWorldTest.java

```
private HelloWorld hw;  
  
@org.junit.Before  
public void setUp() throws Exception {  
    hw = spy(HelloWorld.class);  
    doReturn(toBeReturned: "Hello world").when(hw).getStr();  
}  
  
@org.junit.Test  
public void hello() {  
    assertEquals(expected: "Hello world", hw.hello());  
}
```



# 变异测试工具

PIT is a state of the art **mutation testing** system, providing **gold standard test coverage** for Java and the jvm. It's fast, scalable and integrates with modern test and build tooling



pox.xml

```
<build>
  <plugins>
    <plugin>
      <groupId>org.pitest</groupId>
      <artifactId>pitest-maven</artifactId>
      <version>1.3.2</version>
    </plugin>
  </plugins>
</build>
```

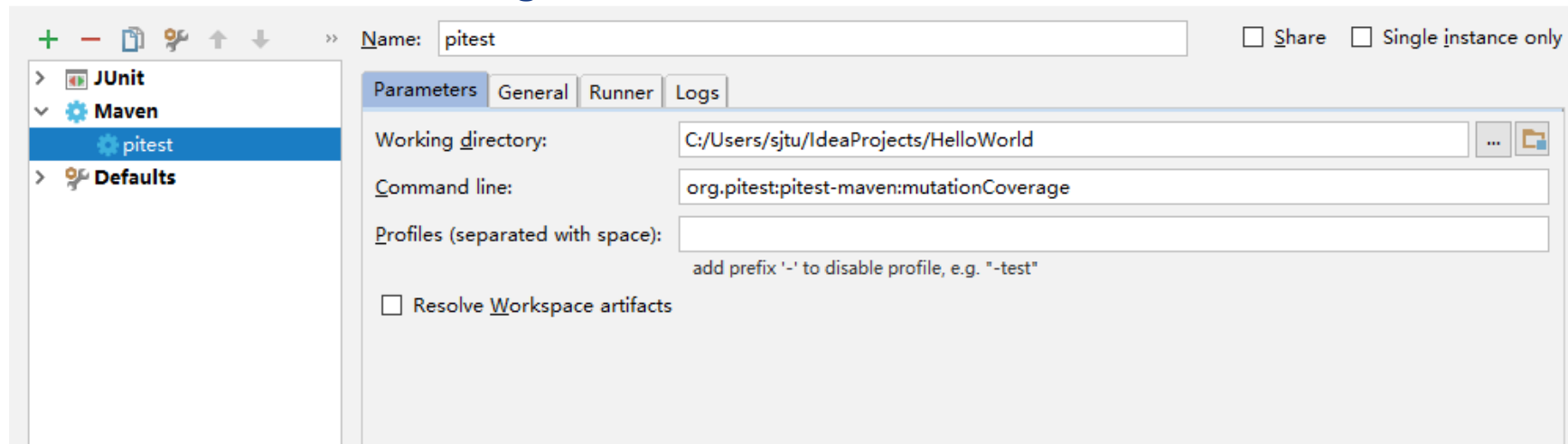


## Run default mutation coverage goal

```
mvn org.pitest:pitest-maven:mutationCoverage
```

## Configurations

- Run->Edit Configurations->Add->Maven->Add new Maven build



## Run





# PiTest Report



## Output / Report in target/pit-reports/YYYYMMDDHHMM

### - Statistics

```
>> Generated 6 mutations Killed 1 (17%)
>> Ran 1 tests (0.17 tests per mutation)
```

### - Mutators

```
> org.pitest.mutationtest.engine.gregor.mutators.ConditionalBoundaryMutator
>> Generated 1 Killed 0 (0%)
> KILLED 0 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 1
```

```
> org.pitest.mutationtest.engine.gregor.mutators.InvertNegsMutator
>> Generated 1 Killed 0 (0%)
> KILLED 0 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 1
```

```
> org.pitest.mutationtest.engine.gregor.mutators.ReturnValsMutator
>> Generated 3 Killed 1 (33%)
> KILLED 1 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 2
```

```
> org.pitest.mutationtest.engine.gregor.mutators.NegateConditionalsMutator
>> Generated 1 Killed 0 (0%)
> KILLED 0 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 1
```

## Pit Test Coverage Report

### Project Summary

Number of Classes	Line Coverage	Mutation Coverage
1	50% <div><div></div><div></div><div></div></div> 2/4	17% <div><div></div><div></div><div></div></div> 1/6

### Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage
<a href="#">com.sjtu.yanjiasen4.hello</a>	1	50% <div><div></div><div></div><div></div></div> 2/4	17% <div><div></div><div></div><div></div></div> 1/6

Report generated by [PIT](#) 1.3.2

## Pit Test Coverage Report

### Package Summary

[com.sjtu.yanjiasen4.hello](#)

Number of Classes	Line Coverage	Mutation Coverage
1	50% <div><div></div><div></div><div></div></div> 2/4	17% <div><div></div><div></div><div></div></div> 1/6

### Breakdown by Class

Name	Line Coverage	Mutation Coverage
<a href="#">HelloWorld.java</a>	50% <div><div></div><div></div><div></div></div> 2/4	17% <div><div></div><div></div><div></div></div> 1/6

Report generated by [PIT](#) 1.3.2



# PiTest Report



Report in target/pit-reports/YYYYMMDDHHMM

## HelloWorld.java

```
1 package com.sjtu.yanjiasen4.hello;
2
3 public class HelloWorld {
4     public String hello() {
5         return this.getStr();
6     }
7
8     public String getStr() {
9         return "Test String";
10    }
11
12    public int abs(int n) {
13        return n >= 0 ? n : (-n);
14    }
15 }
```

### Mutations

```
5 1. mutated return of Object value for com/sjtu/yanjiasen4/hello/HelloWorld::hello to ( if (x != null) null else throw new RuntimeException ) → KILLED
9 1. mutated return of Object value for com/sjtu/yanjiasen4/hello/HelloWorld::getStr to ( if (x != null) null else throw new RuntimeException ) → NO_COVERAGE
   1. changed conditional boundary → NO_COVERAGE
   2. removed negation → NO_COVERAGE
13 3. negated conditional → NO_COVERAGE
   4. replaced return of integer sized value with (x == 0 ? 1 : 0) → NO_COVERAGE
```

### Active mutators

- INCREMENTS\_MUTATOR
- VOID\_METHOD\_CALL\_MUTATOR
- RETURN\_VALS\_MUTATOR
- MATH\_MUTATOR
- NEGATE\_CONDITIONALS\_MUTATOR
- INVERT\_NEGS\_MUTATOR
- CONDITIONALS\_BOUNDARY\_MUTATOR

### Tests examined

- com.sjtu.yanjiasen4.hello.HelloWorldTest.hello(com.sjtu.yanjiasen4.hello.HelloWorldTest) (140 ms)



# Mutate Operator

---

- ④ Conditionals Boundary Mutator
    - $< / <=, > / >=$
  - ④ Negate Conditionals Mutator
    - $== / !=, <= / >, > / <=$
  - ④ Remove Conditionals Mutator
    - $a == b / \text{true}$
  - ④ Math Mutator
  - ④ Increments Mutator
  - ④ Return Values Mutator
-



## 4.5 实例

### XXXX支撑软件与可视化软件 测试计划



## 1、简介

### 1.1编写目的

### 1.2项目背景

### 1.3测试范围

### 1.4参考文档

## 2、测试参考文档和测试提交文档

### 2.1测试参考文档

### 2.2测试提交文档

## 3、测试进度

## 4、测试资源

### 4.1人力资源

### 4.2测试环境

### 4.3测试工具

## 5、测试策略

---





## 5.1功能测试

### 5.1.1 XXX系统管理软件功能测试

### 5.1.2 XXX系统实时通信软件功能测试

### 5.1.3目标三维动态显示模型及可视化软件功能测试

### 5.1.4XXX三维动态显示模型及可视化软件功能测试

### 5.1.5 XXX交会显示软件功能测试

### 5.1.6XXX系统数据管理软件功能测试

### 5.1.7 XXX文件与AutoCAD、Creator的转换软件功能测试

## 5.2性能测试

### 5.2.1 通讯性能测试

### 5.2.2 实验性能测试

### 5.2.3 数据库性能测试

### 5.2.4 显示性能测试

### 5.2.5 其它性能测试

## 5.3故障测试

### 5.3.1网络故障

### 5.3.2 服务器故障

### 5.3.3 管理计算机故障

### 5.3.4 计算节点机故障

### 5.3.5 图形工作站故障测试

## 5.4安全性测试

### 5.4.1 管理软件安全性测试

### 5.4.2 数据管理软件安全性测试



# 1、简介

---

- ① 1.1编写目的
- ① 为了全面、系统地对“XXX支撑软件与可视化软件”进行评估与测试，从而保证系统长期稳定的运行，组织对该软件进行系统的总体综合测试。
- ① 1.2项目背景
- ① “XXX验证平台”是中国兵器工业XXX研究所承担的国防基础科研项目。XXX所与XXX大学合作，对该项目的关键技术——XXX支撑技术和可视化技术展开研究。本项目就是在此背景下开展的，通过应用虚拟试验技术、计算机网络技术、可视化技术、数据库技术等，研制“XXX的支撑软件和可视化软件”，并最终集成在“XXX中。
- ① 本软件系统由以下7个二级软件组成：  
系统管理、实时通讯、三维显示、。。。。。



### 1.3测试范围

本次系统测试将采用开发相关测试软件和利用成熟的测试工具相结合的方式，现场对系统进行测试。按照系统需求任务书的要求，分别进行系统的功能测试、性能测试、故障测试以及安全性测试等相关内容。

本次测试的主要目标包括：

系统软件单元功能测试——根据软件详细设计文档的要求，完成各个软件模块的单元功能测试工作；

系统软件集成功能测试——根据软件需求规格说明的要求，完成软件功能测试工作；

系统软件性能测试——根据技术协议书和软件需求规格说明的要求，完成软件性能测试工作；

系统故障测试——对软件系统在网络故障、数据库故障等异常情况下进行测试工作；

安全性测试——对系统用户登陆、密码保护等安全管理进行测试工作。

---



## ④ 1.4 参考文档

④ 《GB 8566-88 计算机软件开发规范》

④ 《GJB 438A-97 武器系统软件开发文档》

④ XXXXXXXXX

# 2、测试参考文档和测试提交文档

## ④ 2.1 测试参考文档

④ 本测试计划参照GB8567——88标准编制，下表列出了制定测试计划时所使用的文档，并标明了各文档的可用性：

---



## 2.2测试提交文档

在测试完成后，需要提交给用户相关的测试评估报告与改进报告，并提交详细的测试报告。



### 3、测试进度

测试活动	计划开始日期	实际开始日期	结束日期
单元功能测试	2006年 3月16日	2006年3月16日	2004年3月26日
集成功能测试	2006年3月27日	2006年3月27日	2006年3月31日
性能测试	2006年4月2日	2006年4月3日	2006年4月5日
故障测试	2006年4月8日	2006年4月8日	2006年4月9日
安全测试	2006年4月10日	2006年4月10日	2006年4月11日
对测试进行评估	2006年4月15日	2006年4月16日	2006年4月18日
用户验收测试	2006年5月11日		



## 4.1 测试资源

### 4.1 人力资源

姓名	角色	具体职责或注释
胡 飞	总体策划人、方案设计人	策划总体规模，测试内容，规划测试方案
A	方案设计人、测试技术设计人	制定测试方案，确定测试深度，测试技术设计
B	计划人、记录人	计划测试进程，记录测试情况
C	计划人、测试人、记录人	计划整个进程以及各个阶段的进度安排、重点任务；测试并记录测试情况
D	计划人、测试人、记录人	测试并记录测试情况
E	测试人、记录人	测试并记录测试情况
F	测试人、记录人	测试并记录测试情况
G	测试人、记录人	测试并记录测试情况



## 4.2 测试环境

下表列出了测试的系统环境：

机器名 环境	数据库服务器	图形工作站	管理计算机
软件环境 (相关软件、 操作系统等)	Windows XP Oracle 9i		Windows XP
硬件环境 (设备、网 络等)	Xeon Mp $2.7 \times 2$ CPU/4smp/2GB/DVD/1 000M $\times 2$	HP Workstation xw6200	HP Workstation xw6200





## 4.3 测试工具

下表列出此项目测试中使用的工具：

用途	工具名称	生产厂商/自产	版本
压力与性能测试	Jmeter	Jakarta	1.9.1
压力测试网络检测	Webstress	国外软件	6.18
申请一定量的内存	Men	自产	1.0



## 五、测试策略

---

### 5.1 功能测试

由于对测试对象的功能测试应侧重于所有可直接追踪到用例或业务功能和业务规则的测试需求。这种测试的目标是核实数据的接受、计算、处理和输出是否正确，以及业务规则的实施是否恰当，所以此次的功能测试全部为黑盒测试，测试的主要目的是检测以下错误：

- 1) 是否有不正确或遗漏的功能？
  - 2) 在接口上，输入是否能正确的接受？能否输出正确的结果？
  - 3) 是否有数据错误或外部信息（例如数据文件）访问错误？
  - 4) 是否有初始化或终止性错误？
-



## 测试方法:

### 1) 等价类划分

XXXXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXXX

### 2) 边界值分析

Xxxxxxxxxxxxxxxxxxxx

Xxxxxxxxxxxxxxxxxxxx

### 3) 确立测试用例

---



5.1.1 具体功能模块1

5.1.2 具体功能模块2

XXXXXXXXXXXXXXXXXXXXX

5.1.n 具体功能模块n

---