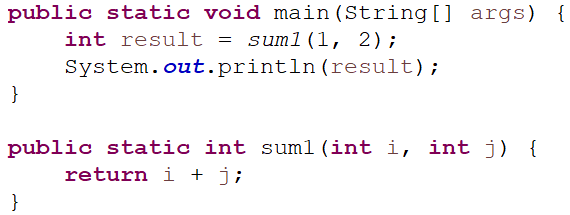
1. **入门**
2. **用main进行测试**

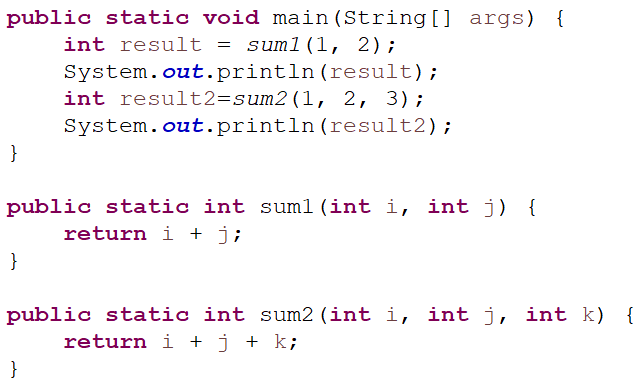
一开始写代码时都会用main函数进行运行，顺便测试一下代码是否按照期望运行。如下，测试sum方法是否按照预期进行执行：



1. **main的问题**

通过main方法进行测试存在一些问题。比如新开发了一个方法，是对3个数求和，那么问题至少有两个：

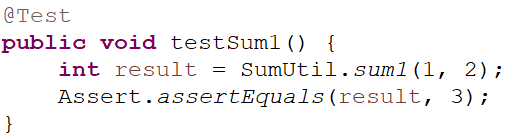
1. 要在原来测试的基础上修改，可能破坏原来的测试逻辑；
2. 测试成功了还是i测试失败了，都不知道。。。只能通过肉眼进行观察，如果测试量很大，是很难看的过来的



1. **junit**

为了应付各种测试的需求，需要使用junit测试框架来进行测试工作。

导入jar包，创建CaseTest1类，并创建一个方法：



它是由 @Test 进行了注解，表示这个方法是一个测试方法。

Assert.assertEquals(result, 3); 表示对 result 数值的期待是 3，如果是其他数值，就无法通过测试。

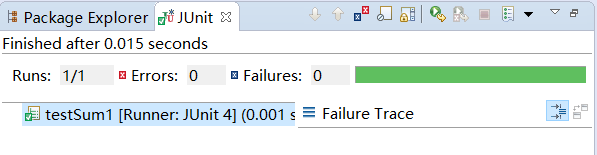
1. **运行方式**

与main方法的运行方式不一样，运行测试用例的时候，需要选择Run AS->JUnit Test方式。



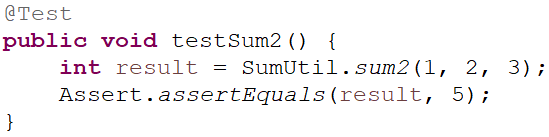
1. **运行成功**

运行成功之后，会显示如图所示的样子，是绿色的。



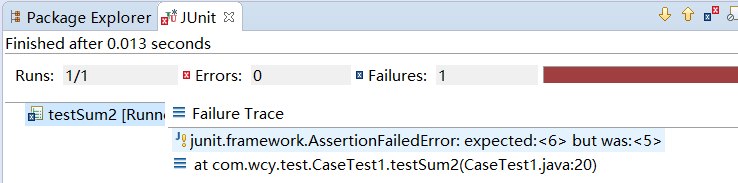
1. **新增加一个测试方法**

接着，新增一个testSum方法，用于测试sum2方法。



1. **运行失败**

运行之后，因为result的值是6，并不等于5，所以失败之后，看到如图所示的效果，表示没有通过测试。还给出来失败的行数：CaseTest1.java:20



1. **junit的好处**
2. 新增加的测试，对原来的测试没有影响；
3. 如果测试失败了，会立即得到通知。
4. **before & after**

@Before@After也是常见的测试框架注解，分别用来测试开始之前做的事情，和结束之后做的事情。

1. **Assert**

关于Assert的用法，除了举例里的判断数字是否相等，还可以判断boolean，是否是null，是否是相同对象等等常见逻辑。

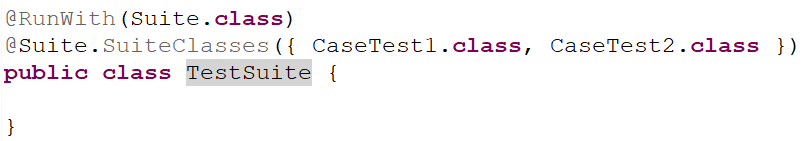
1. **junit TestSuite**
2. **多个测试**

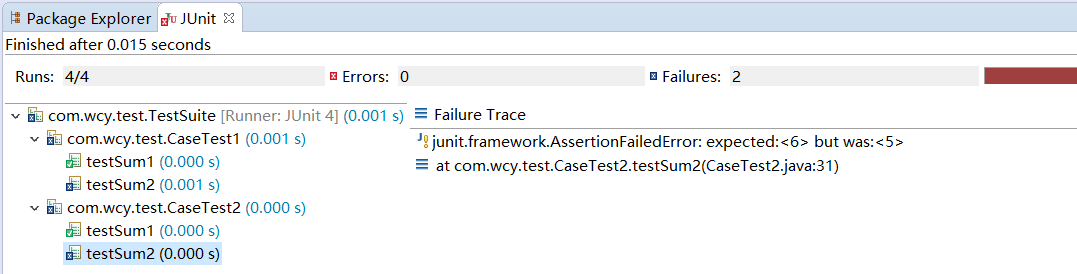
如果有很多工具类需要被测试，那么就会有CaseTest2，CaseTest3， CaseTest4。不得不去单独执行这些测试类，很麻烦，于是就有了**TestSuite的概念**。

1. **CaseTest2**

TestSuite其实就是一下执行多个测试类。首先来个CaseTest2

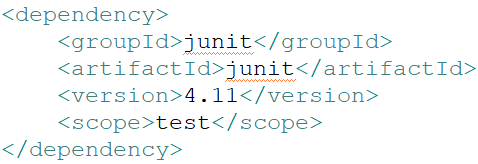
1. **TestSuite**





1. **junit maven**
2. **maven**

maven项目中在pom.xml中加入junit依赖。



1. **junit Spring下的测试**

Spring下的测试其实也是基于JUnit做的，不过做了一些调整。

1. **junit Springboot下的测试**

Springboot 下的测试其实也是基于 JUnit做的，不过做了一些调整。