

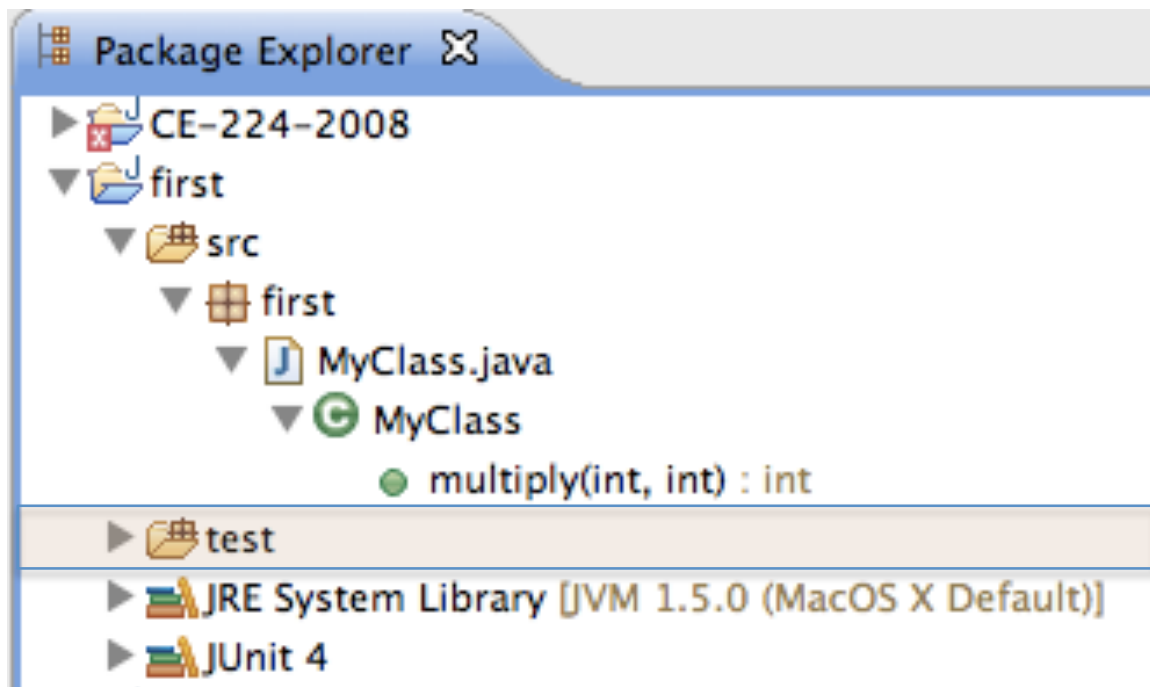
Introdução ao JUnit

Preparação do Projeto

Criar um novo projeto chamado "primeiro".

Criar um novo source folder chamado "teste". Clique no botão direito do mouse no set projeto, selecione *Properties* e escolha o *Java Build Path* . Selecione a tab *Source*. Pressione o botão *Add Folder*, logo em seguida clique no botão *Create New Folder*. Crie o folder "teste".

Uma maneira alternativa mais rápida para adicionar um novo source folder é clicar no botão direito do mouse sobre um projeto e selecionar *New* → *Source Folder* e criar o folder "teste" diretamente. O resultado é o seguinte:



Criando uma class java

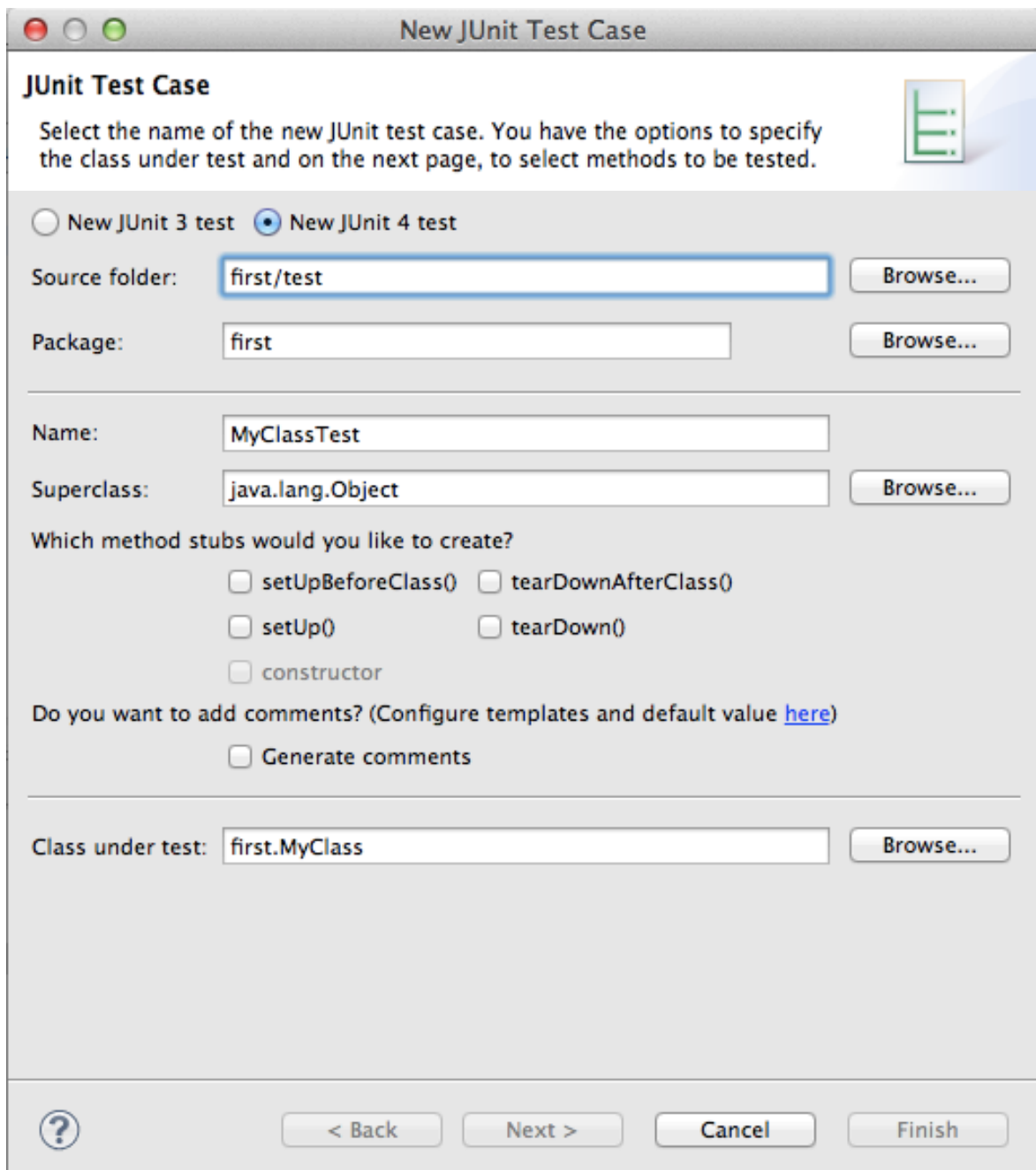
No folder src (código fonte de produção), crie a seguinte classe:

```
public class MyClass {
    public int multiply(int x, int y) {
        // the following is just an example
        if (x > 999) {
            throw new IllegalArgumentException("X should be less than 1000");
        }
        return x / y;
    }
}
```

Criando um teste JUnit

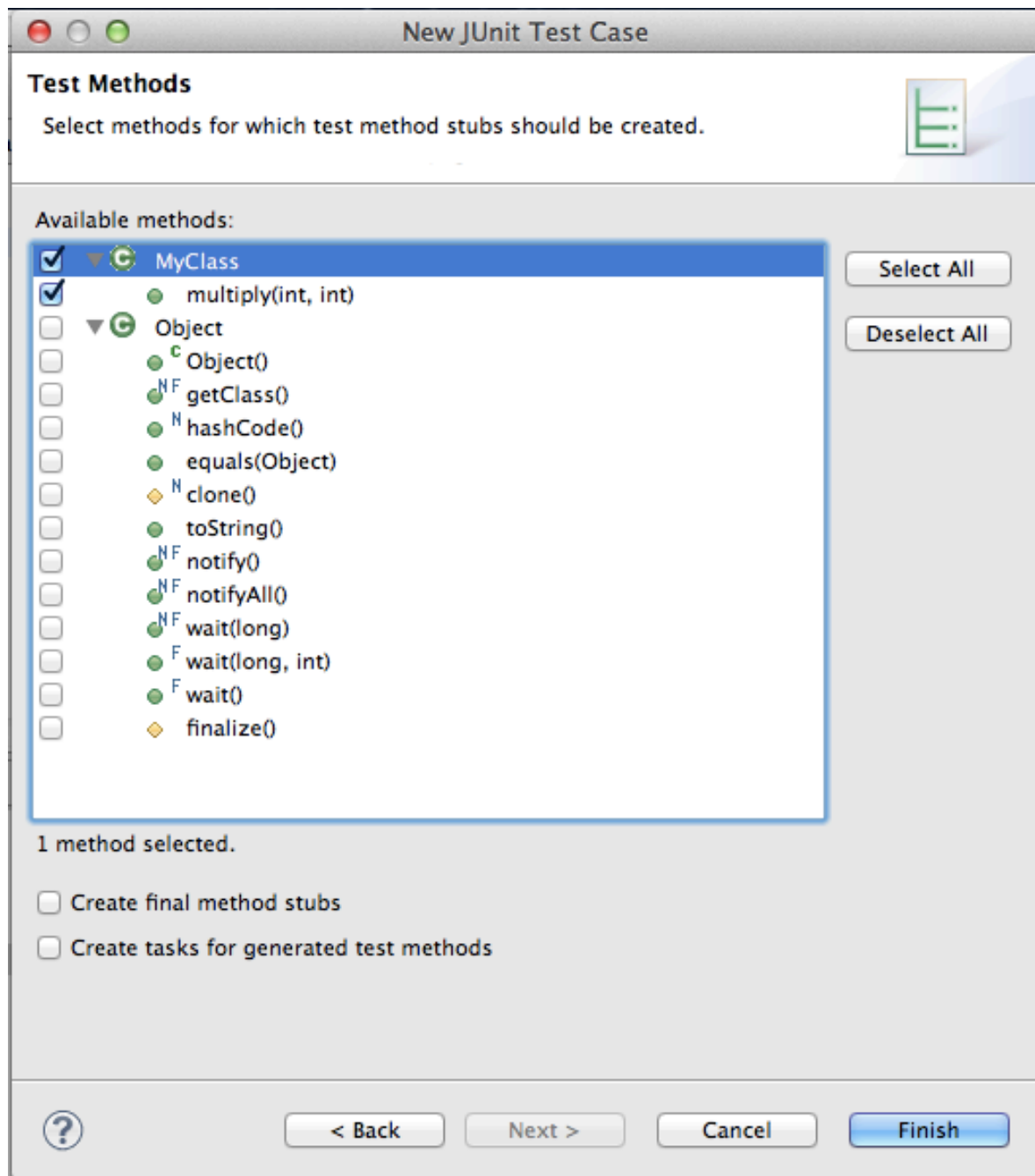
Clique no botão direito do mouse sobre a nova class no *Package Explorer* view e selecione *New* → *JUnit Test Case*. Ou seja, quero criar um teste de unidade para a classe *MyClass*, método *multiply(,)*.

No wizard que segue, assegure-se de que o source folder seja o "teste", de modo que a classe de teste seja criada nesse folder.

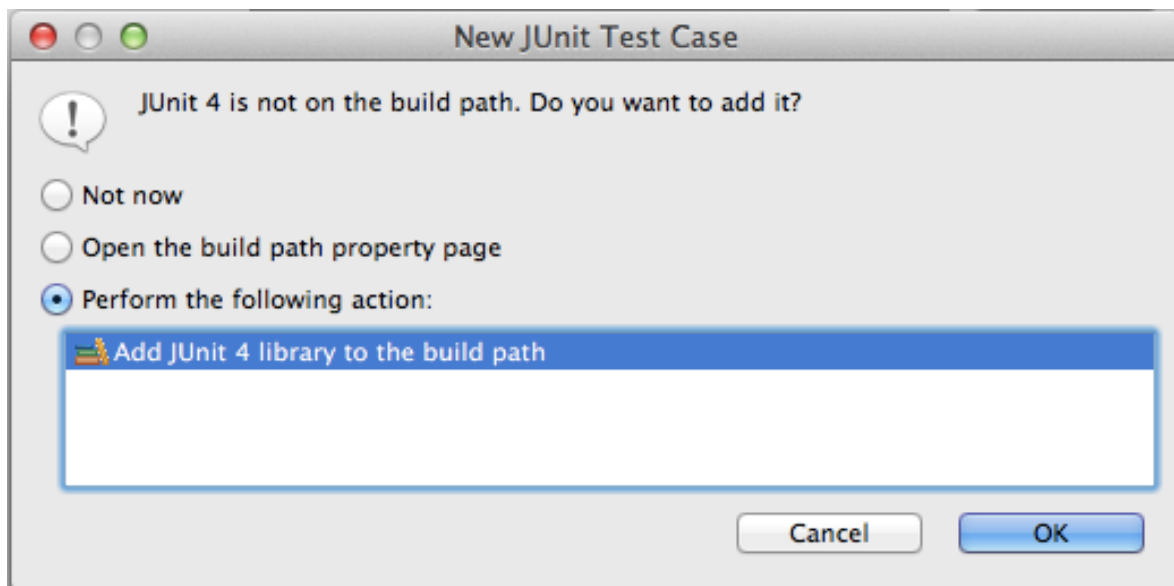


The screenshot shows the 'New JUnit Test Case' dialog box. At the top, it says 'JUnit Test Case' and provides instructions: 'Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.' There are two radio buttons: 'New JUnit 3 test' (unselected) and 'New JUnit 4 test' (selected). Below these are fields for 'Source folder:' (containing 'first/test') and 'Package:' (containing 'first'), each with a 'Browse...' button. The 'Name:' field contains 'MyClassTest'. The 'Superclass:' field contains 'java.lang.Object' with a 'Browse...' button. A section titled 'Which method stubs would you like to create?' contains five checkboxes: 'setUpBeforeClass()' (unselected), 'tearDownAfterClass()' (unselected), 'setUp()' (unselected), 'tearDown()' (unselected), and 'constructor' (unselected). Below this is a question 'Do you want to add comments? (Configure templates and default value [here](#))' with a 'Generate comments' checkbox (unselected). At the bottom, the 'Class under test:' field contains 'first.MyClass' with a 'Browse...' button. The bottom of the dialog has a help icon, and four buttons: '< Back', 'Next >', 'Cancel', and 'Finish'.

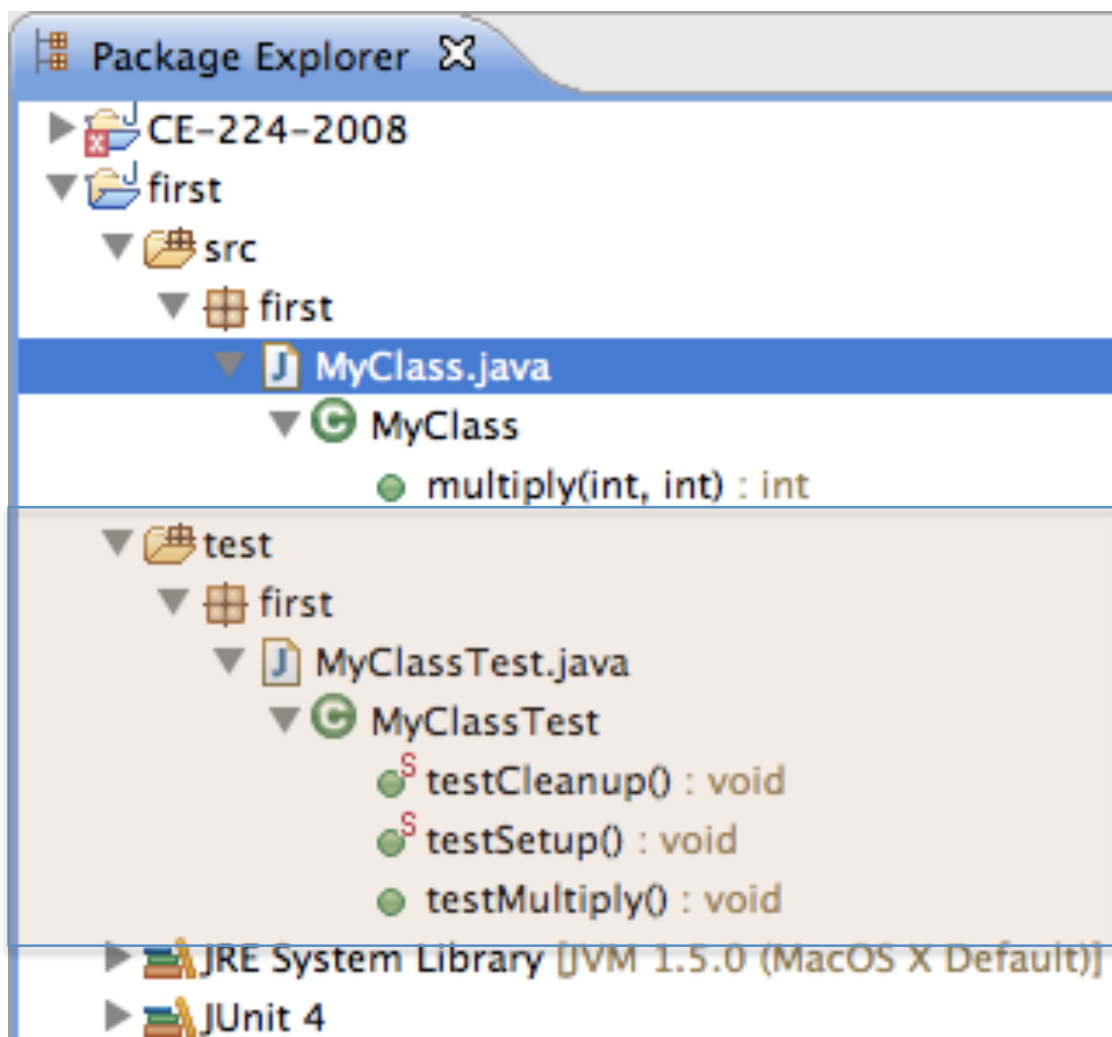
Pressione o botão *Next* e selecione os métodos que você quer testar. Neste caso é o método *multiply(,)*.



Se a biblioteca JUnit não faz parte ainda do classpath do seu projeto, o Eclipse aproveitará esta oportunidade para pedir para adicioná-la ao seu projeto.



O resultado final é o seguinte:



Crie um teste com o seguinte código:

```

import static org.junit.Assert.assertEquals;

import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;

public class MyClassTest {

    @BeforeClass
    public static void testSetup() {
    }

    @AfterClass
    public static void testCleanup() {
        // Teardown for data used by the unit tests
    }

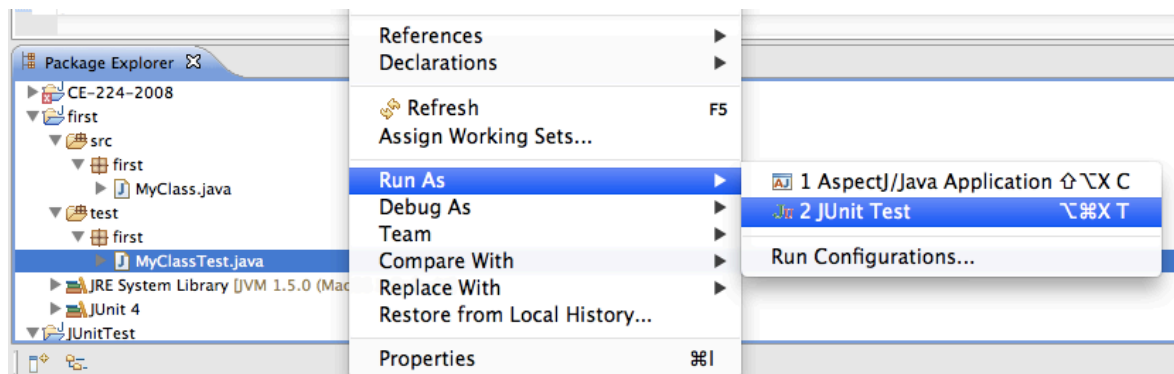
    @Test(expected = IllegalArgumentException.class)
    public void testExceptionIsThrown() {
        MyClass tester = new MyClass();
        tester.multiply(1000, 5);
    }

    @Test
    public void testMultiply() {
        MyClass tester = new MyClass();
        assertEquals("10 x 5 must be 50", 50, tester.multiply(10, 5));
    }
}

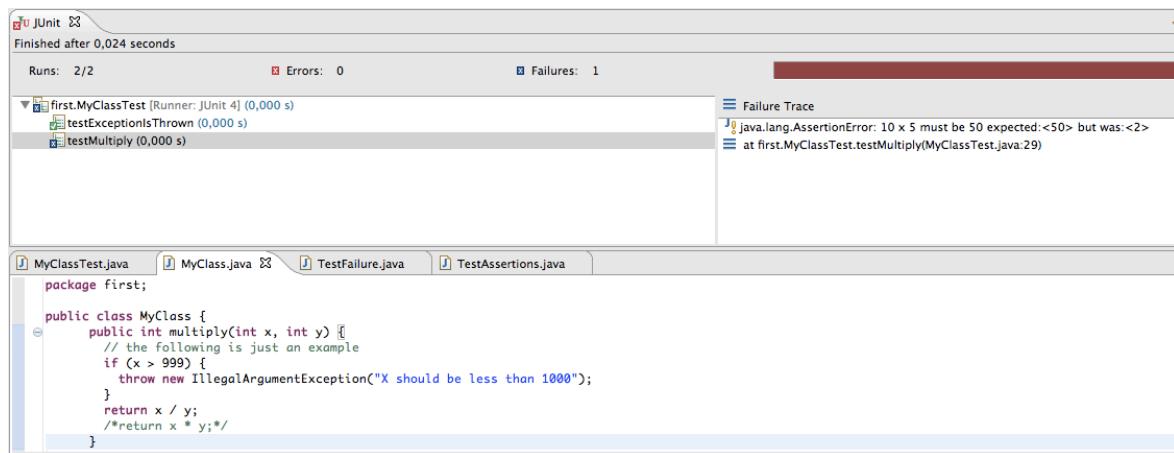
```

Rodando seu teste no Eclipse

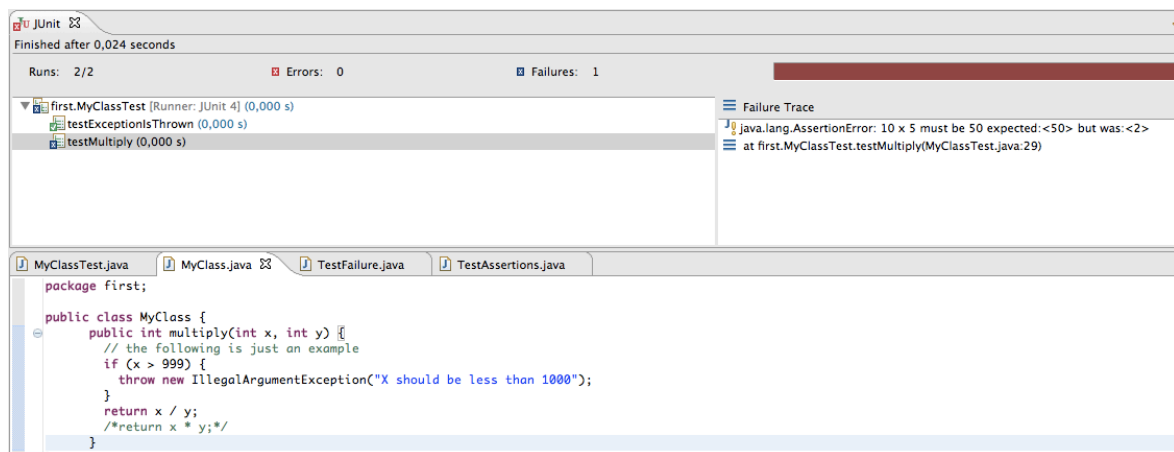
Clique com o botão direito do mouse na sua classe de teste e selecione *Run-As* → *JUnit Test* ou clique direito no botão de execução do Eclipse.



O resultado dos testes serão exibidos no *JUnit view*. Nesse exemplo, um teste deveria ser bem sucedido, que é indicado pela barra verde, e o outro deveria mostrar um erro, que é indicado pela barra vermelha.



O teste está falhando porque o método multiply(~,~) tem um defeito: em vez de multiplicação, ele faz divisão. Conserte o bug e rode novamente o teste para obter uma barra verde.



Algumas anotações:

Annotation	Description
@Test public void method()	The annotation @Test identifies that a method is a test method.
@Before public void method()	This method is executed before each test. This method can prepare the test environment (e.g. read input data, initialize the class).
@After public void method()	This method is executed after each test. This method can cleanup the test environment (e.g. delete temporary data, restore defaults). It can also save memory by cleaning up expensive memory structures.

A anotação @Test diz ao JUnit que o método testMultiply() em que ele está atrelado pode ser rodado como um caso de teste. Para rodar o método, o JUnit primeiro constrói uma instância da classe de teste e então invoca o método anotado. Quaisquer exceções lançadas pelo teste será reportado pelo JUnit como uma falha. Se nenhuma exceção for lançada, assume-se que o teste foi bem sucedido.

Alguns tipos de asserções (fonte do conjunto de asserções: <http://junit.sourceforge.net/javadoc/org/junit/Assert.html>):

Statement	Description
<code>fail(String)</code>	Let the method fail. Might be used to check that a certain part of the code is not reached. Or to have a failing test before the test code is implemented.
<code>assertTrue([message], boolean condition)</code>	Checks that the boolean condition is true.
<code>assertEquals([String message], expected, actual)</code>	Tests that two values are the same. Note: for arrays the reference is checked not the content of the arrays.
<code>assertNull([message], object)</code>	Checks that the object is null.
<code>assertNotNull([message], object)</code>	Checks that the object is not null.
<code>assertSame([String], expected, actual)</code>	Checks that both variables refer to the same object.
<code>assertNotSame([String], expected, actual)</code>	Checks that both variables refer to different objects.

Como fazer para um teste falhar incondicionalmente mesmo antes de código tanto de produção quanto de teste existir? Basta usar o `fail(String)`:

```
package junitExample;

import static org.junit.Assert.*;

import org.junit.Test;

public class TestFailure {

    @Test
    public void testFailure() throws Exception {
        fail("Not yet implemented");
    }

}
```

O `fail(String)` pode ser usado também para verificar se uma parte do código é inalcançável ou não!

A seguir um método de teste para vários tipos de asserções:

```
import org.junit.Test;
import static org.junit.Assert.*;
public class TestAssertions {
    @Test
    public void testAssertions() {
        //test data
        String str1 = new String ("abc");
        String str2 = new String ("abc");
```

```

String str3 = null;
String str4 = "abc";
String str5 = "abc";
String str6 = str4;
int val1 = 5;
int val2 = 6;
String[] expectedArray = {"one", "two", "three"};
String[] resultArray = {"one", "two", "three"};

//Check that two objects are equal
assertEquals(str1, str2);

//Check that a condition is true
assertTrue (val1 < val2);

//Check that a condition is false
assertFalse(val1 > val2);

//Check that an object isn't null
assertNotNull(str1);

//Check that an object is null
assertNull(str3);

//Check if two object references not point to the same object
assertNotSame(str1,str2);

//Check if two object are equal
assertEquals(str1,str4);

//Check if two object references not point to the same object
assertNotSame(str1,str4);

//Check if two object references point to the same object
assertSame(str4,str5);

//Check if two object references point to the same object
assertSame(str4,str6);

//Check if two object references point to the same object
assertSame(str5,str6);

//Check if two object references not point to the same object
assertNotSame(str1,str3);

//Check whether two arrays are equal to each other.
assertArrayEquals(expectedArray, resultArray);
}
}

```