JUnit - Unit Testing in Java

CS 4354
Summer II 2014
Jill Seaman

Software Testing

• Executing the system with simulated test data and checking the results for errors, anomalies, and unexpected performance.

• **Failure**: Deviation between the specification and the actual behavior of the system.

• **Fault** (aka “bug” or “defect”): A design or coding mistake that may cause abnormal behavior (with respect to specifications)

• **Test case**: set of inputs and expected results that exercises a system (or part) with the purpose of detecting faults

• **Testing**: the systematic attempt to find faults in a planned way in the implemented software.

Testing

Test cases should contain the following:

• **Name**: Explains what is being tested

• **Input**: Set of input data and/or commands and/or actions

• **Expected results**: Output or state or behavior that is correct for the given input.

Testing

• **Who performs testing?**
  ✦ Developers
  ✦ Testing staff
  ✦ Users/Customers

• **What kind of testing do developers do?**
  ✦ **Unit testing**: individual program units (i.e. classes) are tested
  ✦ **Component testing**: system components (composed of individual units) are tested to make sure the contained units interact correctly.
  ✦ **System testing**: the system components are integrated and the system is tested as a whole.
Testing in Agile Methods

• Test-first Development
  ✦ Tests are written before the task is implemented.
  ✦ Forces developer to clarify the interface and the behavior of the implementation.
• Test automation is crucial
  ✦ Testing is developer's responsibility (no external test team)
  ✦ No interaction required: results checked automatically and reported.
  ✦ Automatic regression testing ensures no existing functionality gets broken by a new increment or refactoring.

Requirements for Automatic Testing

• The framework must use the programming language to write the test (developer tests)
• It must allow the separation of application code from test code.
• It must enable tests to run independently of each other (one failure cannot cause others to fail).
• It must allow developers to organize test cases into a suite
• The success or failure of a test should be visible at a glance.
• It must support unit testing at the following levels:
  ✦ testing a single method.
  ✦ testing an entire class (interaction of methods).
  ✦ testing the interaction of two or more objects.

JUnit

• Open source framework for the automation of unit testing in Java.
• It meets the requirements in the previous slide.
• It is used widely in the industry.
• It can be downloaded from junit.org
• I will be using version 4.11

JUnit Tutorial (based on vogella.com)

• First we will consider the code to be tested:

```java
package mine;

class MyClass {
    public int multiply (int x, int y) {
        return x*y;
    }
}
```

• How can I use JUnit to test it?
  ✦ Create a Test class: a class which is used only for testing.
  ✦ Add a method that will implement the test case.
  ✦ Annotate the method with the @Test annotation.
  ✦ In this method you use a method provided by the JUnit framework to check the expected result of the code execution versus the actual result.
JUnit Tutorial: the test class

• The test class:

```java
package mine;
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class MyClassTest {
    @Test
    public void multiplicationOfZeroIntegersShouldReturnZero() {
        // MyClass is tested
        MyClass tester = new MyClass();
        // Tests
        assertEquals("10 x 0 must be 0", 0, tester.multiply(10, 0));
        assertEquals("0 x 10 must be 0", 0, tester.multiply(0, 10));
        assertEquals("0 x 0 must be 0", 0, tester.multiply(0, 0));
    }
}
```

JUnit Tutorial: How to compile and run the test?
Part I: From the command line

• Download the jar files from junit.org:
  - junit.jar
  - hamcrest-core.jar

• The downloaded filenames may include version numbers.
• Put these in a directory.
• I use src as my root directory. I put these in src/lib.
• I also made a src/bin file to store my *.class files.
• The *.java files from the last slides go in src/mine.

```java
package mine;
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class MyTestRunner {
    public static void main(String[] args) {
        Result result = JUnitCore.runClasses(MyClassTest.class);
        for (Failure failure : result.getFailures()) {
            System.out.println(failure.toString());
        }
    }
}
```

JUnit Tutorial: How to compile and run the test?
Part I: From the command line

• Now I need a driver class to execute the test(s):

```java
package mine;
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class MyTestRunner {
    public static void main(String[] args) {
        Result result = JUnitCore.runClasses(MyClassTest.class);
        for (Failure failure : result.getFailures()) {
            System.out.println(failure.toString());
        }
    }
}
```

• I passed the name of my Test class to the runClasses method.

Here is the compile and execute process ($ is the prompt):

```
$ javac -d bin -cp lib/junit-4.11.jar:lib/hamcrest-core-1.3.jar mine/*.java
$ java -cp bin:lib/junit-4.11.jar:lib/hamcrest-core-1.3.jar mine.MyTestRunner
```

• No output means the test(s) passed
• The -d bin option tells the compiler to store the *.class files in the bin directory.
• The -cp option tells the compiler and JVM where to look for the required class files.
• Note the “:” to separate the directory names and jar files in the -cp option.
JUnit Tutorial: How to compile and run the test?  
Part I: From the command line

• Now I will change the last test to expect 1 instead of 0, so that it fails:

```java
// Tests
assertEquals("10 x 0 must be 0", 0, tester.multiply(10, 0));
assertEquals("0 x 10 must be 0", 0, tester.multiply(0, 10));
assertEquals("0 x 0 must be 1", 1, tester.multiply(0, 0));
```

• Now I recompile and run again, and I get this (see below).

• Note the error message on the last 2 lines:

```
$ javac -d bin -cp lib/junit-4.11.jar:lib/hamcrest-core-1.3.jar mine/*.java
$ java -cp bin:lib/junit-4.11.jar:lib/hamcrest-core-1.3.jar mine.MyTestRunner
multiplicationOfZeroIntegersShouldReturnZero(mine.MyClassTest):
  0 x 0 must be 1 expected:<1> but was:<0>
```

JUnit Annotations (@tags)

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@Test</td>
<td>identifies a public void method as a test method.</td>
</tr>
<tr>
<td>@Test (expected = Exception.class)</td>
<td>Fails if the method does not throw the named exception</td>
</tr>
<tr>
<td>@Before</td>
<td>identifies a method that is to be executed before each test.</td>
</tr>
<tr>
<td>@BeforeClass</td>
<td>identifies a method that is to be executed once, before the start of all tests. It must be public static void.</td>
</tr>
<tr>
<td>@After</td>
<td>Analogous to Before/BeforeClass</td>
</tr>
<tr>
<td>@AfterClass</td>
<td></td>
</tr>
<tr>
<td>@Ignore</td>
<td>identifies a method to be skipped (it’s broken, or not ready)</td>
</tr>
</tbody>
</table>

JUnit Tutorial: How to compile and run the test?  
Part II: From within Eclipse

• Eclipse has built-in support for creating and running JUnit tests.
  ✦ you do not need to download and install the junit.jar files, at least not for the more recent versions of eclipse.

• For example, to create a JUnit test or a test class for an existing class,
  ✦ select this class in the Package Explorer view,
  ✦ right-click on it and select New → JUnit Test Case.

• To run a test,
  ✦ select the class which contains the tests,
  ✦ right-click on it and select Run-as → JUnit Test. This starts JUnit and executes all test methods in this class.
JUnit Tutorial: How to compile and run the test?
Part II: From within Eclipse

- I will do the following demo in class.
- Make a project for Assignment2, put the classes in the src folder, inside the assign2 package.
- Make a new src folder called test (right-click on the project, select New → Source Folder)
- Right-click on Movie.java and select New → JUnit Test Case. Call it MovieTest and put it in the test package.
  ✦ if you get “Warning JUnit 4 is not on the BuildPath…” say yes to add it.

To run the test,

✦ select the MovieTest class (in the package explorer)
✦ right-click on it and select Run-as → JUnit Test. This starts JUnit and executes all test methods in this class.
✦ Eclipse uses the JUnit view which shows the results of the tests

Green is good

To make the test fail,

✦ In the Movie.java class, change the value of the shipping credit to 2.88.
✦ Run the test again.
JUnit Tutorial: How to compile and run the test?  
Part II: From within Eclipse

• Add another method testShippingCreditToy:

```java
@Test
public void testShippingCreditToy() {
    Toy t = new Toy(3344,2,12.55,"Monopoly",34);
    assertEquals("toy shipping credit should be ????, t.shippingCredit()
```