Repeatable Unit Testing with JUnit

Ozgur Yilmazel

Overview

- Benefits of Repeatable Unit Tests
- Principles
- JUnit 101
- JUnit Observations
- JUnit Tactics
- JUnit Extensions

Benefits of Repeatable Unit Tests

- Decreased Integration Time & Costs
- Clarify Scope of Work
- Supports Ruthless Design Innovation (Refactoring)
- Ongoing Documentation/Tutorial on Component Usage
- Long term benefits far outweigh short term development costs

Benefits of Repeatable Unit Tests (cont.)

- Regression Test New Changes
- Higher Quality Code == Reduced Maintenance Costs
- Higher Quality Code == Increased Business Agility

Principles

- Code is Not “Done” until Tests are Correctly Written and Executed
- No Software Promotion until All Tests Pass
- Code a Little, Test a Little
  - Keeps the Developer on Track
  - Isolate & Localize Problems
  - Validates Requirements & Design Earlier rather than Later

Principles (cont.)

- Test First Design
- You Can't Afford *Not* To Write Unit Tests
  - Move Integration "Pain" to Earlier in Project - Less Expensive
  - Cost of Software Change Increases Exponentially towards End of Project

JUnit 101

HowTo - Install

- Download from junit.org
- Add jar file to classpath
- Online Documentation Available
- Many Extensions Exist
  - Load Testing
  - Servlet/Struts Testing

HowTo - Create Test Class

```
import junit.framework.*;
public class TestFoo extends
  TestCase {
}
```

Write `setUp/tearDown` Methods for Class

```
import junit.framework.*;
public class TestSquare extends
  TestCase {
  private Square shape;
  public void setUp() {
     shape = new Square();
  }
  public void tearDown() {
     shape = null;
  }
}
```

Add Methods for each Test

```
import junit.framework.*;
public class TestFoo extends TestCase {
   // other methods omitted…
   public void testSmall() {
       assertNotNull(“shape is null”,shape);
       shape.setX(10);
       shape.setY(10);
       assertEquals(“area
   incorrect”,100,shape.getArea();
   }
}
```

Code `main` Method

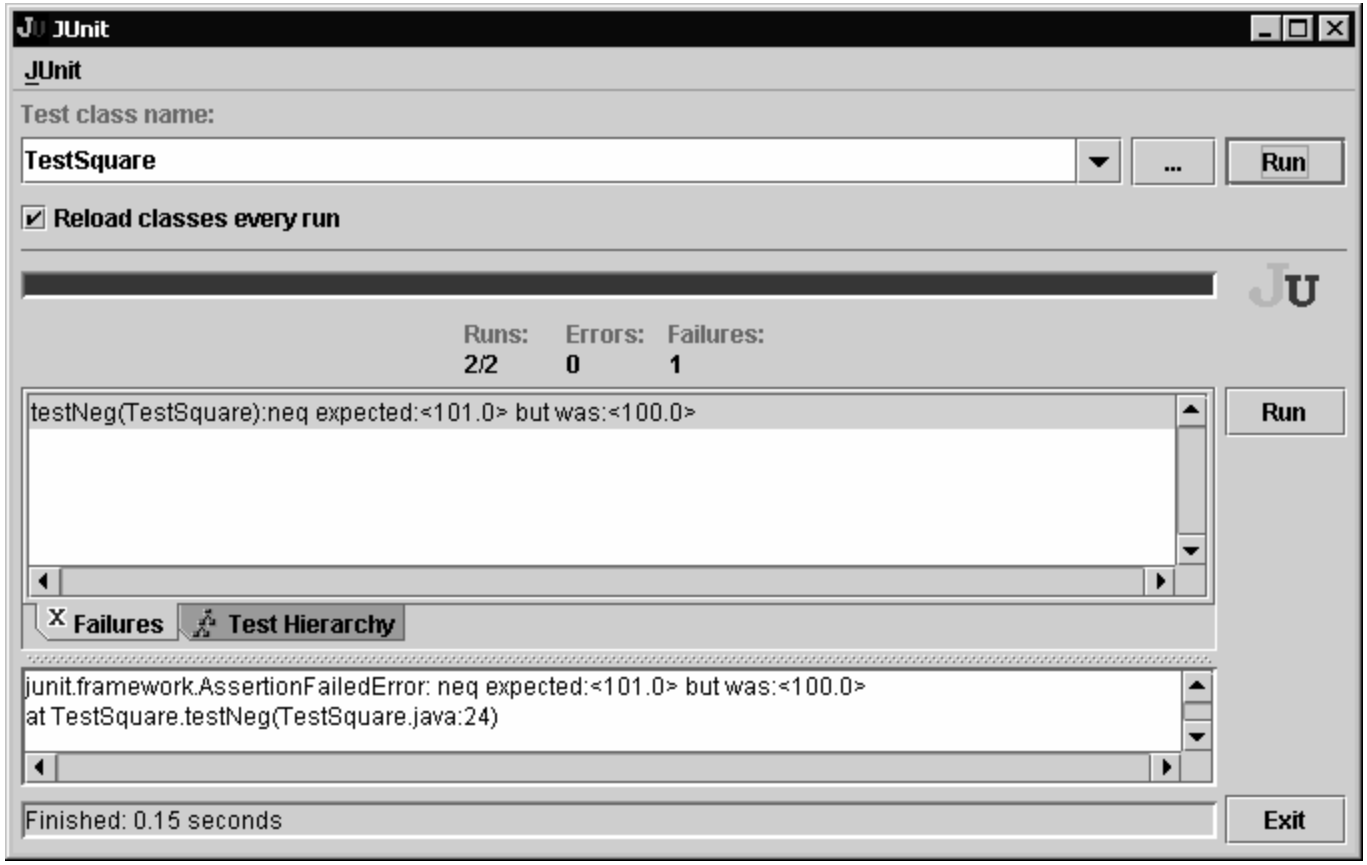
```
import junit.framework.*;
public class TestFoo extends TestCase {
   // other methods omitted…
   public static void main(String[] argv) {
       junit.textui.TestRunner.run(TestFoo.class)
   }
}
```

Run It!

```
..F
Time: 0
There was 1 failure:
1) testNeg(TestSquare)junit.framework.AssertionFailedError: neq
expected:<101.0> but was:<100.0>
at TestSquare.testNeg(TestSquare.java:23)
at TestSquare.main(TestSquare.java:16)

FAILURES!!!
Tests run: 2,  Failures: 1,  Errors: 0
```

Run It! (GUI)


JUnit
JUnit
Test class name:
TestSquare
...
Run
Reload classes every run
Runs:
2/2
Errors:
0
Failures:
1
testNeg(TestSquare):neq expected:<101.0> but was:<100.0>
Run
Failures
Test Hierarchy
junit.framework.AssertionFailedError: neq expected:<101.0> but was:<100.0>
at TestSquare.testNeg(TestSquare.java:24)
Finished: 0.15 seconds
Exit

ANT Integration

- ANT has tasks for working w/ JUnit (part of optional package)
- Sample ANT task:

```
<target name="test-all" depends="clean,jar,test-jar" if="junit.present">
  <!-- Import JUnit task -->
  <taskdef
   name="junit"
   classname="org.apache.tools.ant.taskdefs.optional.junit.JUnitTask"
  />

  <junit printsummary="yes" fork="yes" filtertrace="on" haltonfailure="no" >
   <classpath refid="test.classpath"/>
   <formatter type="plain" />
   <batchtest fork="yes" todir="${junit.reports}">
    <fileset dir="${java.src.dir}" >
      <include name="**/*Test.java"/>
    </fileset>
   </batchtest>
  </junit>

</target>
```

JUnit Observations

- Many assertXXX Methods Provide Objective Evaluation of Code
- Visual TestRunner is Available
  - Keep it green to keep it clean
- Failure - Failed Assertion
- Error - Unexpected Exception

Tactics

- Create One Test Class for each Production Class
  - ProviderDAOJDBC & ProviderDAOJDBCTest
- Keep Test Classes in line w/ file
  - Use ANT filesets to include or exclude

Tactics (cont.)

- Bug Report == Write a Test
- Run Tests at Least Daily (Smoke Test)
  - Can use GUMP (jakarta.apache.org/gump)
- Write Tests for Code with Highest Probability of Breakage
- Write Test Code First
  - Or at least very early in development

Tactics (cont.)

- Tether Test Cases into Master Test Case
  - Run One Test Case to Verify Integrity before Migration to next Phase
- Limit Testing of Model/Entity/Bean Classes
- Within Test Cases, Leverage assertXXX Methods
  - Use assertXXX Methods Instead of Output Statements
- Use JUnit

Extensions

- StrutsTestCase - Test Struts Actions (http://strutstestcase.sourceforge.net/)
- junitperf - Beat the Code Senseless! (http://www.clarkware.com/software/JUnitPerf.html)
- JDepend - Test the Design (http://www.clarkware.com/software/JDepend.html)
- www.junit.org
- Perl Unit: http://search.cpan.org/dist/Test-Unit/

Summary

- Benefits of Repeatable Unit Tests
- Principles
- JUnit 101
- JUnit Observations
- JUnit Tactics
- JUnit Extensions

References

- Original JUnit Paper: http://junit.sourceforge.net/doc/testinfected/testing.htm
- Another JUnit Primer: http://www.clarkware.com/articles/JUnitPrimer.html
- eXtreme Programming: http://extremeprogramming.org