Formalizing the unit testing process with JUnit

Jason Chambers
AJUG December 2002
www.ajug.org

Introduction

- Advisory developer - delta.com
- 12 years experience
- Sun Certified Programmer & Web Component Developer for the Java 2 platform
- Interests include Java, Linux, web development
- jason_chambers@yahoo.com
- The opinions expressed during this presentation do not necessarily reflect those of Delta Technology or Delta Air Lines... it's just me talking

Contents

- What is unit testing anyway?
- History of Java unit testing
- JUnit to the rescue
- A quick tour of JUnit
- Demo
- Best practices
- Q & A

What is unit testing?

- Verification of observable behaviour of a programmatic unit
- Each unit is tested in isolation
- Apply a set of one or more inputs to the unit and observe the outputs in each case to verify
- In Java: unit=class, input=method invocation, output=return/out parameters
- Not a replacement for any other types of testing

What is the value?

- Increases confidence in change
- Increases quality
- Key aspect of Extreme Programming and other agile methodologies
- Eases diagnosis of problems
- Code becomes less resistant to change (less expensive)
- Refactor with confidence
- Increases productivity

History of unit testing in Java

- Embedded test driver
- Huge improvement over C++ unit testing (can only have one main)
- Used during development
- Is it used during maintenance phase?
- Prone to becoming stale
- Poor cohesion - test code intermingled

```java
package mypackage;
class SomeClass {
  public int doSomething() {return 1;

  /** Unit test driver
      Please run me often
      Please keep me updated */
  public static void
          main(String args[]) {
    SomeClass o = new SomeClass();
    if (o.doSomething() == 1)
      System.out.println("Passed");
    else
      System.out.println("Failed");
  }
}
```

JUnit to the rescue

- Open source Java testing framework used to write and run repeatable tests
- Released using IBM's CPL 0.5 license
- Developed by Erich Gamma and Kent Beck
- Elegant design (rich in design patterns)
- Mature
- Easy to use

JUnit features

- Assertions for testing expected results
- Test fixtures for sharing common test data
- Test suites for easily organizing and running tests
- Graphical and textual test runners

source: JUnit 3.8.1. FAQ

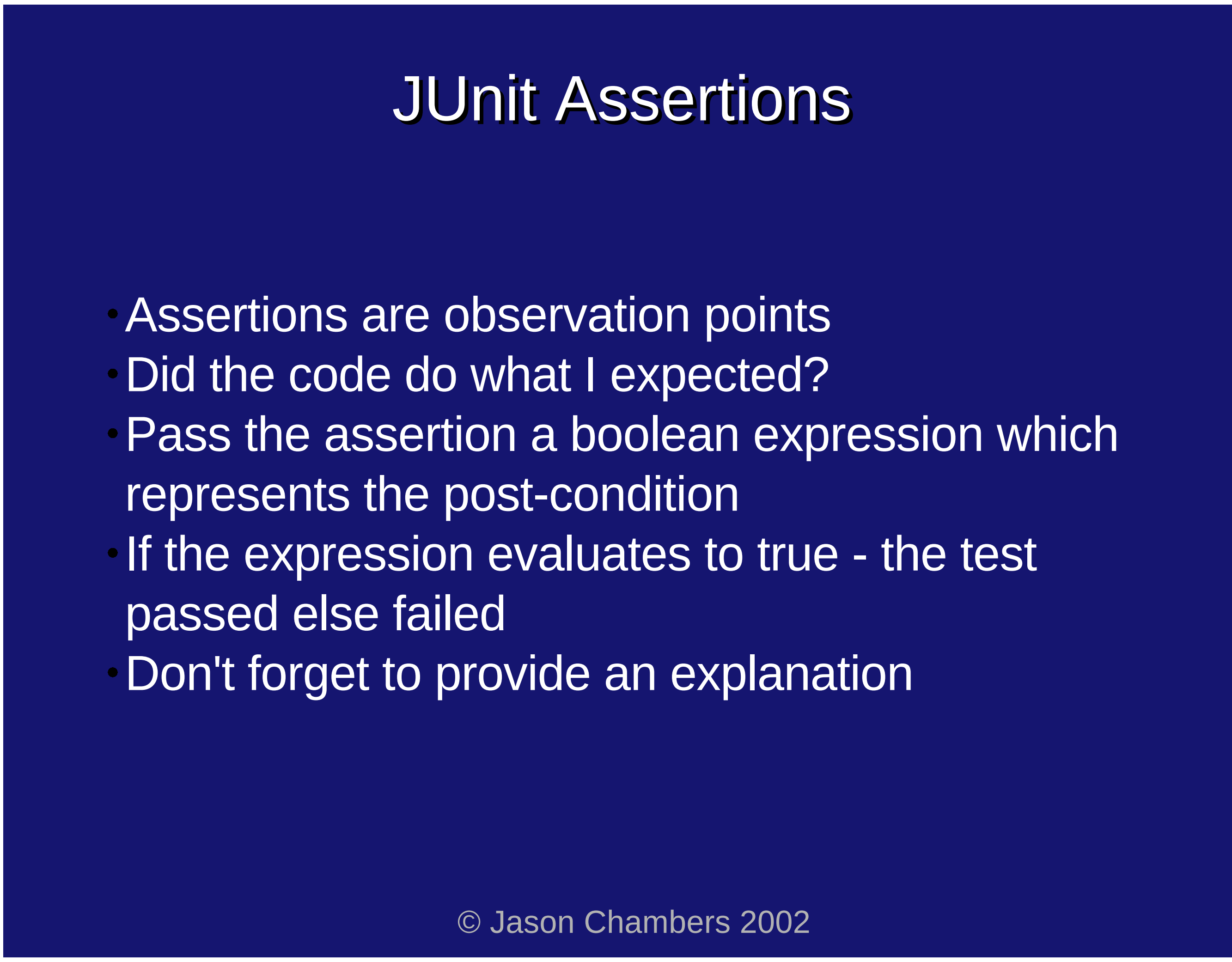
Writing a test case

```java
package mypackage;
import junit.framework.*;
public class SomeClassTest extends TestCase {

  /** A test */
  public void testDoSomething() {
    SomeClass o = new SomeClass();
    Assert.assertTrue("Expect doSomething() ==1",
                      o.doSomething() == 1);
  }

  /** Another test */
  public void testDoSomethingElse() {...}

  /** Compose the tests into a suite - called by a
      TestRunner */
  public static Test suite() {
    /* All public void testXXX() methods will be
       called by the runner (Reflection at work) */
    return new TestSuite(SomeClassTest.class);
  }
}
```



JUnit Assertions

- Assertions are observation points
- Did the code do what I expected?
- Pass the assertion a boolean expression which represents the post-condition
- If the expression evaluates to true - the test passed else failed
- Don't forget to provide an explanation

Fixtures

- Optional
- Provides opportunity to centralize test initialization and shutdown code for the suite
- Override setUp and tearDown
- As JUnit calls your test methods it will call setUp before and tearDown after each call
- Call sequence is setUp(), testXXX(), tearDown(), setUp(), testXXY(), tearDown() etc.

Composing TestCases

```java
public class AllTests {
    public static Test suite() {

        TestSuite suite = new TestSuite();

        // Add all the TestCases for the package here
        // .. can get tedious
        suite.addTest(SomeClassTest.suite());

        return suite;
    }
}
```

How about searching for *Test instead?

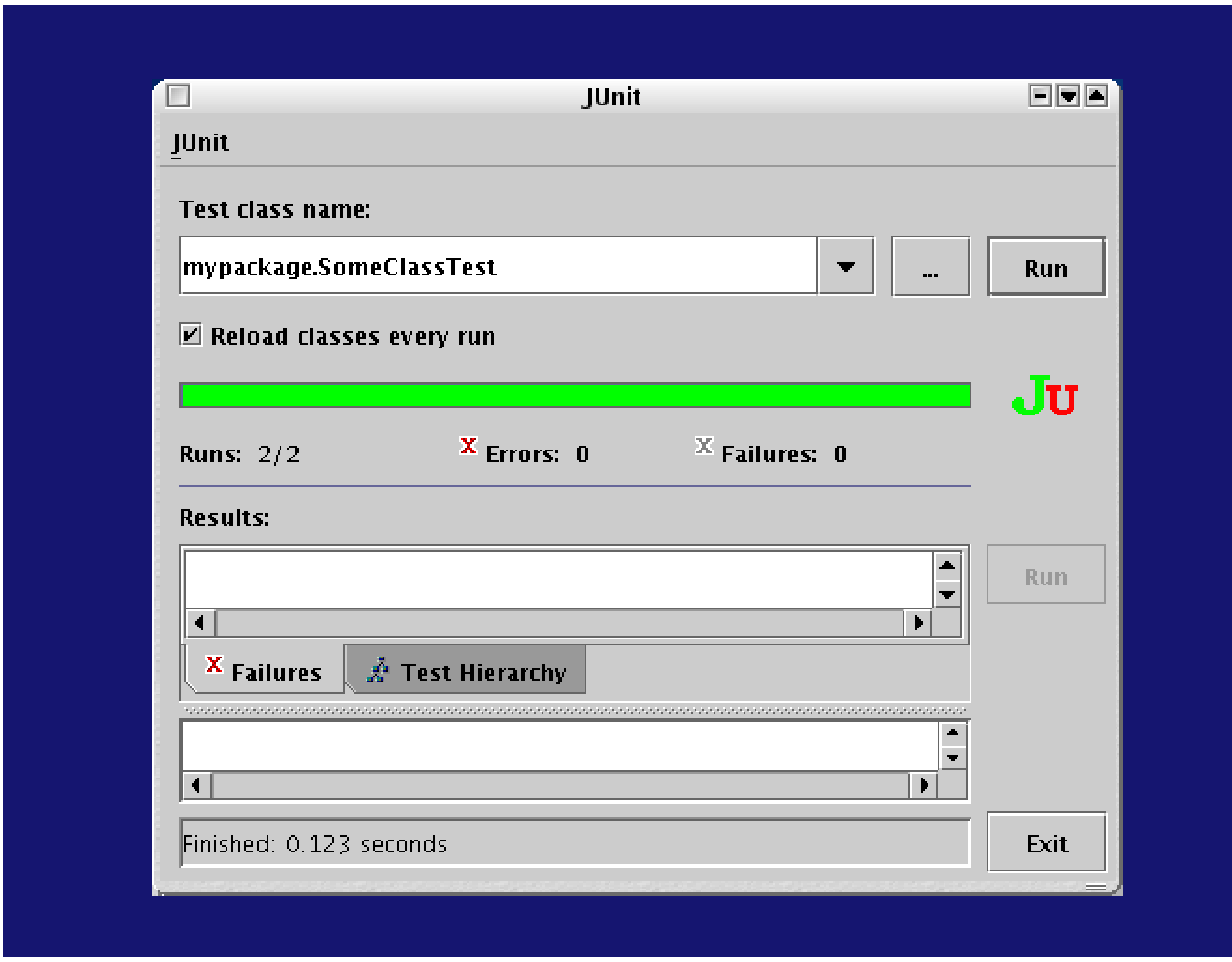
Running using the text testrunner

```
# java junit.textui.TestRunner mypackage.SomeClassTest
..
Time: 0.005

OK (2 tests)
#
```

Running using the Swing TestRunner

```
# java junit.swingui.TestRunner mypackage.SomeClassTest
```


JUnit
JUnit
Test class name:
mypackage.SomeClassTest
...
Run
Reload classes every run
Ju
Runs: 2/2
Errors: 0
Failures: 0
Results:
Run
Failures
Test Hierarchy
Finished: 0.123 seconds
Exit

Auto launching a TestRunner

- Simply add a main() method to the TestCase as follows using the TestRunner of your choice

```
package mypackage;
import junit.framework.*;
public class SomeClassTest extends TestCase {

  ...

  public static void main(String args[]) {
    junit.textui.TestRunner.run(suite());
  }
}
```

Running via Ant

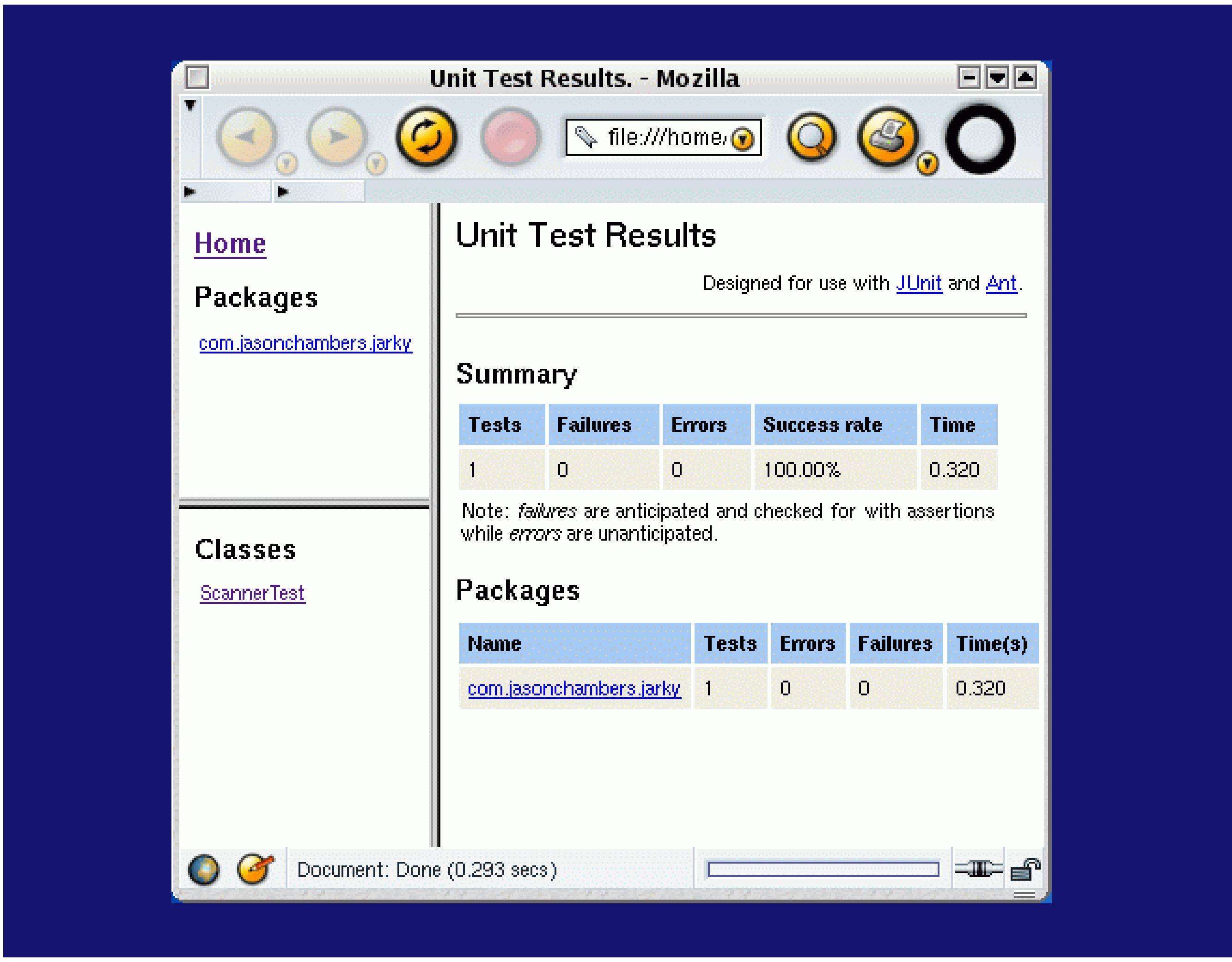
- Ant and JUnit are a great combination
- Simply create targets for building and running your JUnit TestCases
- Ant is bundled with <junit> task to make running your TestCases a breeze
- There is also a <junitreport> task for producing reports

Ant: Building your TestCases

```
<target name="compiletests" depends="jar"
        description="Compiles all tests.">
   <javac srcdir="${test.dir}"
           destdir="${build.testcases}">
      <classpath>
         <pathelement location="${build.lib}/jarky.jar"/>
         <pathelement location="${junit.jar}"/>
      </classpath>
   </javac>
</target>
```

Ant: Running your TestCases

```
<target name="runtests" depends="compiletests">
   <junit printsummary="no" fork="no" haltonfailure="no">
      <classpath refid="runtest.classpath"/>
      <formatter type="xml"/>
      <test name="com.jasonchambers.jarky.ScannerTest"
            todir="${reports.dir}"/>
   </junit>
   <junitreport todir="${reports.dir}">
      <fileset dir="${reports.dir}">
         <include name="TEST-*.xml"/>
      </fileset>
      <report format="frames" todir="${reports.dir}/html"/>
   </junitreport>
</target>
```



Unit Test Results. - Mozilla

file:///home/

Home

Packages

com.jasonchambers.jarky

Classes

ScannerTest

Unit Test Results

Designed for use with JUnit and Ant.

Summary

Tests	Failures	Errors	Success rate	Time
1	0	0	100.00%	0.320

Note: *failures* are anticipated and checked for with assertions while *errors* are unanticipated.

Packages

Name	Tests	Errors	Failures	Time(s)
com.jasonchambers.jarky	1	0	0	0.320

Document: Done (0.293 secs)

Demo time

- textui & swingui TestRunners on SimpleClass
- ScannerTest - <junit> & <junitreport>

Best Practices

- Test early. Test often. Test automatically. (Pragmatic Programmer)
- Write the TestCase first.. before you even write the class!
- TestCases end with Test e.g. TestCase for `Scanner` class is `ScannerTest`
- TestCase methods must be `public void testXXX()` if you want JUnit to find them through reflection
- Put TestCases in same Java package - provides opportunity to exercise package friendly methods
- Put TestCases in separate directory

Best Practices (cont'd)

- Use OO principles for developing TestCases
- Consider weaving test execution into build process (easy when using Ant)
- Tests should be short, focussed and plentiful
- Don't waste your time testing simple getters/setters
- Avoid temporaral coupling - do not assume testXXX will run before testYYY
- Javadoc your TestCases = unit testing specification
- Grow your tests

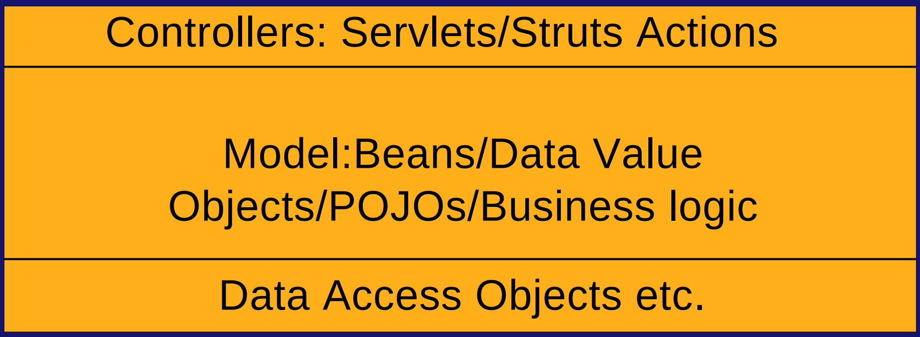
Don't quote me!

- "*If the code is changed, assume it is broken until proven otherwise*"

John Carnell, AJSS 2002

Challenges

- Testing model should be straightforward - tier edges are more challenging.
- Tip - control/reduce tier-spillage & design for testability
- How do you unit test a HttpServlet? How do you test a DAO?


Controllers: Servlets/Struts Actions
Model:Beans/Data Value Objects/POJOs/Business logic
Data Access Objects etc.

What about old code?

- It is difficult to write tests for code that has already been written
- Focus on writing tests for new code
- Maybe re-factor already existing code - make it more "testable"

Enhancing JUnit

- Dbunit - sets up the database in a known state before executing your tests
- Cactus - for unit testing server-side Java code such as servlets
- JUnitPerf - measure the performance and scalability of existing JUnit tests
- JUB - JUnit test case Builder
- ....

Quiz

- Q. If you have 100% test coverage and all tests pass, is the program considered proven to be correct?
- A. No. Testing merely raises the level of confidence in change. If you are interested in mathematically proving correctness, look at formal methods.

Resources

- www.junit.org
- "Pragmatic Progammer: From Journeyman to Master", Andrew Hunt & David Thomas
- Read the "JUnit - A Cooks Tour" if you are interested in the design of JUnit.
- dbunit.sourceforge.net
- jakarta.apache.org/cactus
- www.clarkware.com/software/JUnitPerf.html
- www.javaworld.com/javaworld/jw-12-2000/jw-1221-junit.html
- www-106.ibm.com/developerworks/library/j-ant/?dwzone=java
- jub.sourceforge.net

My weapons of choice

- SuSE 8.0
- KDE 3.0.0
- bash 2.0.5
- JDK 1.4
- Tomcat 4.0.4
- JUnit 3.8.1
- Struts 1.1
- Ant 1.5.1
- XEmacs 21.1/JDE 2.3.0
- Mozilla 1.1
- KPresenter 1.1 (KOffice)

It's a wrap!

- Questions?
- Thanks for listening
- jason_chambers@yahoo.com