Lecture 4

Unit Testing

SOEN 6441, Summer 2018

René Witte
Department of Computer Science
and Software Engineering
Concordia University

Outline

Software Testing

René Witte

Testing

Stone age testing: debug output (`println...`)

- Testing done manually
- Tests run by developer only, get removed/lost later
- No automated (regression) tests
- No metrics about test coverage
- No “safety net” for refactoring

Test automation

Extreme programming (XP) popularized the notion of test-driven development

- required tool support

DevOps and related practices emphasizes code quality through

- Continuous integration
- Test become essential part of the code base
- Writing (unit) tests now closely related with development

René Witte

JUnit History

Junit is a (the most?) popular unit testing framework for Java.

- Kent Beck developed the first xUnit automated test tool for Smalltalk in mid-90's
- Kent Beck and Erich Gamma (one of the Design Patterns GoF) developed JUnit on a flight from Zurich to Washington, D.C.
- Martin Fowler: "Never in the field of software development was so much owed by so many to so few lines of code."
- First release: 2000(?)
- JUnit 4.0: 2006
- JUnit 5.0: September 2017

JUnit Test Example

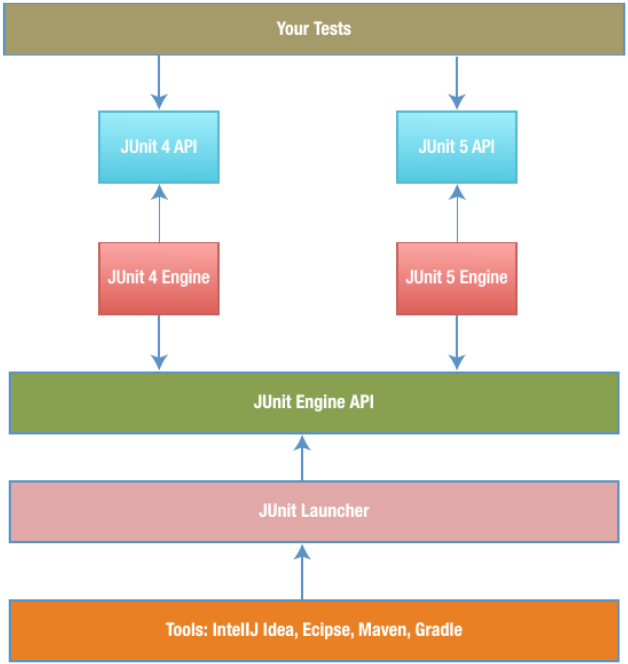
```java
import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.Test;

class FirstJUnit5Tests {
    @Test
    void myFirstTest() {
        assertEquals(2, 1 + 1);
    }
}
```

JUnit Terminology

Definitions

- A test runner is software that runs tests and reports results.
  Many implementations: standalone GUI, command line, integrated into IDE (e.g., Eclipse)
- A test suite is a collection of test cases.
- A test case tests the response of a single method to a particular set of inputs.
- A unit test is a test of the smallest element of code you can sensibly test, usually a single class.
- A test fixture is the environment in which a test is run. A new fixture is set up before each test case is executed, and torn down afterwards.
  E.g., creating a database with some entries required for the test.

JUnit Architecture



JUnit Annotations

JUnit supports a number of code annotations

- `@Test` Denotes that a method is a test method
  (JUnit 4 provides additional attributes)
- `@BeforeEach` runs a method before each test (to setup test fixture)
  (`@Before` in JUnit 4)
- `@AfterEach` runs a method after each test (to tear down test fixture)
  (`@After` in JUnit 4)
- `@BeforeAll` runs a method once, before all tests (`@BeforeClass` in JUnit 4)
- `@AfterAll` runs a method once after all tests to tear down test fixture
  (`@AfterClass` in JUnit 4)

Test Coding

- Tests are not executed in any defined order
- Hence, JUnit tests must be independent of each other
  (e.g., you cannot use one test method to create an object that you check in a second test method)
- Test one property per method (can have multiple assertions, e.g., for different fields of an object)

A Standard Test Class (I)


```java
import static org.junit.jupiter.api.Assertions.fail;

import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
```

A Standard Test Class (II)

```
class StandardTests {
    @BeforeAll
    static void initAll() { ... }

    @BeforeEach
    void init() { ... }

    @Test
    void succeedingTest() { ... }

    @Test
    void failingTest() {
        fail("a failing test");
    }

    @Test
    @Disabled("for demonstration purposes")
    void skippedTest() {
        // not executed
    }

    @AfterEach
    void tearDown() { ... }

    @AfterAll
    static void tearDownAll() { ... }
}
```

Assertions

Java Assert

```
AssertStatement:
    assert booleanExpression ;
    assert booleanExpression : errorMessage ;
```

Working with Java `assert`

Compare

```
assert names.remove(null);
```

vs

```
boolean nullsRemoved = names.remove(null);
assert nullsRemoved;
```

Coding Guidelines

- Assertions in Java are turned off by default (need `java -ea`)
- Your code must run correctly without assertions
- Production code generally must not terminate

JUnit Assertions

```
public class AssertionsTest {

  @Test
  public void test() {
    String obj1 = "junit";
    String obj2 = "junit";
    String obj3 = "test";
    String obj4 = "test";
    String obj5 = null;
    int var1 = 1;
    int var2 = 2;
    int[] arithmetic1 = { 1, 2, 3 };
    int[] arithmetic2 = { 1, 2, 3 };

    assertEquals(obj1, obj2);
    assertSame(obj3, obj4);
    assertNotSame(obj2, obj4);
    assertNotNull(obj1);
    assertNull(obj5);
    assertTrue(var1  var2);
    assertArrayEquals(arithmetic1, arithmetic2);
  }
}
```

JUnit Assertions

Old-Style Assertions

Separate assertion statements, e.g.

```
assertEquals(expected, actual);
```

or

```
assertEquals(expected, actual, string);
```

Note: the correct (expected) result comes *before* the computed result

New Style: Hamcrest Matchers (since JUnit 4.4)

Only using `assertThat` plus different *matchers*

```
assertThat(actual, is(equalTo(expected)));
```

or

```
assertThat(string, actual, is(equalTo(expected)));
```

Advantages: better error messages, type safety, improved readability, extendable matchers

Use the Hamcrest matchers for all new tests you write!

https://objectpartners.com/2013/09/18/the-benefits-of-using-assertthat-over-other-assert-methods-in-unit-tests/

Hamcrest Core Matchers

static import `org.hamcrest.CoreMatchers`

- allOf()
- any()
- anyOf()
- anything()
- describedAs()
- equalTo()
- instanceOf()
- is()
- not()
- notNullValue()
- nullValue()
- sameInstance()

Examples

```
assertThat(actual, is(expected));
assertThat(actual, is(not(expected)));
```

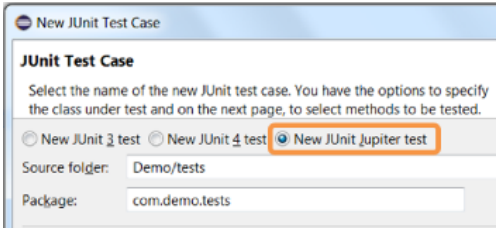
Running JUnit

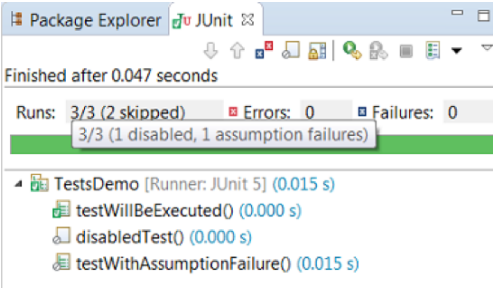
Eclipse

Eclipse has support for

- creating and
- running JUnit tests

(refer to the documentation for your Eclipse release (e.g., Neon or Oxygen)





(refer to the documentation for your version of Eclipse)

Jenkins (https://jenkins.io/)

JUnit Integration in Jenkins

Jenkins

search ? log in

Jenkins › TextMining-LODeXporter › #72 › Test Results › Info.semanticsoftware.lodexporter › LODeXporterTest ENABLE AUTO REFRESH

- Back to Project
- Status
- Changes
- Console Output
- View Build Information
- History
- Git Build Data
- Checkstyle Warnings
- FindBugs Warnings
- PMD Warnings
- Open Tasks
- Static Analysis Warnings
- See Fingerprints
- Test Result
- Coverage Report
- Previous Build
- Next Build

Test Result : LODeXporterTest

0 failures (±0)

6 tests (±0)
Took 41 sec.

All Tests

Test name	Duration	Status
testGetCustomURI	12 sec	Passed
testGetMappingFile	0.4 sec	Passed
testInitConflict	50 ms	Passed
testInitFail	4 ms	Passed
testReInit	0.77 sec	Passed
testRunPR	2.5 sec	Passed

Test Coverage

Calculating Test Coverage

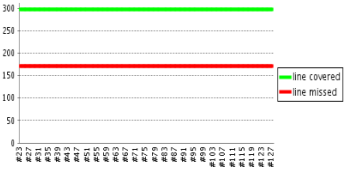
Need to know the coverage of the unit tests

JaCoCo

JaCoCo (Java Code Coverage Library) is an open source tool for computing test coverage through byte code analysis when running unit tests.

Jacoco Report Example

Download `jacoco.exec` binary coverage file



Overall Coverage Summary

name	instruction	branch	complexity	line	method	class
all	55%	47%	50%	63%	79%	86%
classes	M: 1057 C: 1314	M: 97 C: 85	M: 88 C: 88	M: 171 C: 297	M: 17 C: 63	M: 1 C: 6

Coverage Breakdown by Package

name	instruction	branch	complexity	line	method	class
Info.semanticsoftware.lodexporter	M: 708 C: 719 50%	M: 65 C: 53 45%	M: 61 C: 57 48%	M: 110 C: 157 59%	M: 13 C: 41 76%	M: 1 C: 5 83%
Info.semanticsoftware.lodexporter.tdb	M: 349 C: 595 63%	M: 32 C: 32 50%	M: 27 C: 31 53%	M: 61 C: 140 70%	M: 4 C: 22 85%	M: 0 C: 1 100%

Jacoco Report: Method Details

name	instruction	branch	complexity	line	method
TDBTripleStoreImpl()	M: 0 C: 6 100%	M: 0 C: 0 100%	M: 0 C: 1 100%	M: 0 C: 2 100%	M: 0 C: 1 100%
addPropertyToSubject(Map, Resource, String)	M: 32 C: 31 49%	M: 3 C: 3 50%	M: 3 C: 1 25%	M: 4 C: 5 56%	M: 0 C: 1 100%
beginTransaction(TripleStoreInterface.TransactionType)	M: 0 C: 13 100%	M: 0 C: 2 100%	M: 0 C: 2 100%	M: 0 C: 4 100%	M: 0 C: 1 100%
connect()	M: 0 C: 15 100%	M: 0 C: 0 100%	M: 0 C: 1 100%	M: 0 C: 3 100%	M: 0 C: 1 100%
connect(String)	M: 16 C: 0 0%	M: 0 C: 0 100%	M: 1 C: 0 0%	M: 3 C: 0 0%	M: 1 C: 0 0%
disconnect()	M: 0 C: 18 100%	M: 0 C: 0 100%	M: 0 C: 1 100%	M: 0 C: 4 100%	M: 0 C: 1 100%
endTransaction()	M: 0 C: 7 100%	M: 0 C: 0 100%	M: 0 C: 1 100%	M: 0 C: 3 100%	M: 0 C: 1 100%
exportTriplesToFile(String)	M: 55 C: 26 32%	M: 6 C: 2 25%	M: 4 C: 1 20%	M: 2 C: 6 75%	M: 0 C: 1 100%
getPropertyMappings(String)	M: 0 C: 15 100%	M: 0 C: 0 100%	M: 0 C: 1 100%	M: 0 C: 5 100%	M: 0 C: 1 100%
getRelationMappings(String)	M: 0 C: 16 100%	M: 0 C: 0 100%	M: 0 C: 1 100%	M: 0 C: 5 100%	M: 0 C: 1 100%
getSubjectMappings(String)	M: 0 C: 8	M: 0 C: 0	M: 0 C: 1	M: 0 C: 2	M: 0 C: 1

Jacoco Report: Source Code Details

```
485:
486:         /* (non-Javadoc)
487:          * @see info.semanticsoftware.lodexporter.TripleStoreInterface#exportTriplesToFile(java.lang.String)
488:          */
489:         @Override
490:         public final void exportTriplesToFile(final String fileName) {
491:                 model = dataset.getDefaultModel();
492:                 try (FileOutputStream os = new FileOutputStream(fileName)) {
493:                         RDFDataMgr.write(os, model, RDFFormat.NQUADS_UTF8) ;
494:•                } catch (IOException e) {
495:                         LOGGER.error("Error writing triples to file: " + fileName, e);
496:                         throw new GateRuntimeException("Error writing triples to file: " + fileName, e);
497:                 }
498:         }
499: }
```

Unit Testing: Summary

Conclusions

- Start writing tests at the same time as you implement your classes
- May influence class design to improve testability
- Added initial development effort, but:
- experience shows significant return on investment due to improved code quality
- Increases individual and team confidence in code base
- Prerequisite for modern release workflows

Outline

Reading Material

Required

- JUnit User Guide, http://junit.org/junit5/docs/current/user-guide/

References

- JUnit, http://junit.org/
- JaCoCo, http://www.jacoco.org/jacoco/
- Jenkins, https://jenkins.io/

Supplemental

- Shekhar Gulati, Rahul Sharma: *Java Unit Testing with JUnit 5*, Apress, 2017. https://encore.concordia.ca/iii/encore/record/C__Rb3512472 *(Ebook can be downloaded through Concordia Library)*
- Michael Olan. 2003. Unit testing: test early, test often. *Journal of Computing Sciences in Colleges* 19, 2 (December 2003), 319–328. https://dl.acm.org/citation.cfm?id=948785.948830 *(access through Concordia Library)*