SOEN 387 Web-based Enterprise Application Design

Stuart Thiel

Concordia University
Department of Computer & Software Engineering

Fall, 2015

Test Driven Development

- Test Driven Development (TDD), write tests before code
- Write few tests
- Test for a few new features
- Watch tests fail
- Write the code
- Run the tests
- Fix and repeat

Why TDD

- You must know what you want first
- You are testing a solution before you have it
- Builds useful test suite
- Constant progress
- Writing tests **later** often does not happen

Structure of Test Approach

- Set Up
- Run Tests
- Validate Results
- Clean Up

Is TDD Perfect?

- Not see forest for the trees
- Sometimes forget big picture and take many tiny irrelevant steps in weird directions
- Can get in habit of writing tests that depend on state of previous tests (bad)
- Easy to forget to include integration in these tests

Testing in Java

- Eclipse knows how to deal with test suites
- Good support
- Write the tests, run as JUnit
- You will do that this Friday

Some Testing Code

```
public class TestPerson {

        public final static String BASEURL =
          "http://localhost:8080/HWTDD/";
        static XPath xPath =
          XPathFactory.newInstance().newXPath();

        @Test
        public void addPerson() throws SAXException,
          IOException, XpathException {
                System.out.println("AddPerson");
                //Make request to get list of people,
                //confirm that it is empty

                //Make sure bob's not there
                deletePerson("bob", "marley");
                Document doc = listPeople();
                XMLAssert.assertXpathNotExists(
                  "/people/person[@firstname=\"bob\" and "
                  +"@lastname=\"marley\"]", doc);
```

Setting Up Tests

- Tests generally run top to bottom
- You can configure JUnit to run pre-ordered sets of tests
- It is very flexible
- This test sets itself up inside
- It makes sure the person that we add is not there
- If they were, we could not tell if we had added them properly

Setup and Teardown

- Can also use BeforeClass/AfterClass
- Before/After
- Makes writing test setup/teardown easier
- teardown is just “cleaning up”

Naming Conventions

- There are many old ones
- starting tests with the word “test”
- Annotations make most of these historical
- Some languages may still require it
- Allows tests to be found automagically

Setup and Teardown

- Run your test
- Then validate
- I like to test "Use Cases" and their scenarios

Test and Validation Example

```
        addPerson("bob", "marley");
        doc = listPeople();
        XMLAssert.assertXpathExists(
          "/people/person[@firstname='bob' and "
          +"@lastname=\"marley\"]", doc);

        deletePerson("bob", "marley");
    }
```

Nuts and Bolts

- XMLAssert vs. Assert?
- Who wrote addPerson and deletePerson?
- XPATH?

Outline I

Outputting XML

- You can find a taglib to output XML
- But you need to generate tags to use it!
- Still, might have some convenience

Some XML Generation Code

```
<%@ page trimDirectiveWhitespaces="true" %>
<%@ page language="java" contentType="text/xml;
  charset=UTF-8" pageEncoding="UTF-8"%>
<?xml version="1.0" encoding="UTF-8"?>
<checkers>
<status>success</status>
 <challenge id="${challenge.id }"
   version="${challenge.version }"
   status="${challenge.status.id }" >
  <challenger refid="${challenge.challenger.id }"/>
  <challengee refid="${challenge.challengee.id }"/>
 </challenge>
</checkers>
```

Some XML Generation Code With Looping

```
<%@ page trimDirectiveWhitespaces="true" %>
<%@ page language="java" contentType="text/xml;
  charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"
  prefix="c" %>
<?xml version="1.0" encoding="UTF-8"?>
<checkers>
<status>success</status>
 <challenges>
 <c:forEach var="challenge" items="${challenges }">
 <challenge id="${challenge.id }"
   version="${challenge.version }"
   status="${challenge.status.id }" >
  <challenger refid="${challenge.challenger.id }"/>
  <challengee refid="${challenge.challengee.id }"/>
 </challenge>
</c:forEach>
 </challenges>
</checkers>
```

Outline I

Outputting XML

- You can find your own
- I found json-taglib
- It looks like it does everything I want
- You won't need it for this course

json-taglib JSP

```
<json:object>
  <json:property name="string1" value="this is a string"/>
  <json:property name="string2" value="   and another string   "/>
  <json:property name="untrimmedString"
                 trim="false"
                 value="  and an untrimmed string   " />
  <json:property name="usingTheBody">
    This data is in the tag body.
    1+1 is ${1+1}
  </json:property>

  <json:property name="bool1" value="${true}"/>
  <json:property name="bool2" value="${false}"/>

  <json:property name="numeric1" value="${1+2}"/>
  <json:property name="numeric2" value="${-500}"/>
  <json:property name="numeric3" value="${123.456}"/>
</json:object>
```

json-taglib Generated JSON

```
{
  "string1": "this is a string",
  "string2": "and another string",
  "untrimmedString": "  and an untrimmed string   ",
  "usingTheBody": "This data is in the tag body.\r\n    1+1 is 2",
  "bool1": true,
  "bool2": false,
  "numeric1": 3,
  "numeric2": -500,
  "numeric3": 123.456
}
```