Unit Tests

Overview

- Unit tests are meant to be very small and efficient.
- Each method (aka each test) should only validate one logical piece of code.
- Method names should be long and descriptive
- Methods should have an Assert.* call in it

Example:

```java
public void getAllUsers_shouldNotReturnNull() throws Exception {
    Assert.assertNotNull(Context.getUserService().getAllUsers());
}
```

Note, however, that it is preferable to statically import the members and use Hamcrest matchers such as the below:

```java
public void getAllUsers_shouldNotReturnNull() throws Exception {
    // assertThat is imported from Assert; notNullValue is imported from Matchers
    assertThat(Context.getUserService().getAllUsers(), notNullValue());
}
```

When in a test class that extends BaseContextSensitiveTest, some test data are available to you automatically. See the standardTestDataset.xml and initialInMemoryTestDataSet.xml files for content. See "Setting Up the Expected Database" below for how to add your own test data.

```java
public class MyTests extends BaseContextSensitiveTest {
    public void getPatientByName_shouldReturnValidPatients() {
        List patients = Context.getPatientService().getPatientsByName("bob");
        Assert.assertEquals(34, patients.get(0).getPatientId());
    }
}
```

Be sure to read our Unit Testing Conventions as well.

Getting Started

To test a class that is not associated with the Context unit testing is only a three step process:

1) Identify the method you want to test:

```java
package org.openmr;

/**
 * This does blah blah blah...
 */
public class MyObject {

...

/**
 * This method does something blah blah blah
 * @param arg this does something
 * @return the Object
 */
public Object someComplicatedCall(String arg) throws Exception {
    ...does something interesting here...
}
```
2) Create a test case for it:

```java
package org.openmrs;

import static org.hamcrest.CoreMatchers.notNullValue;
import static org.junit.Assert.assertThat;

/**
 * Tests for MyObject
 */
public class MyObjectTest {

/**
 * Get the X output of ComplicatedCall when given a non null argument
 */
@Test
public void someComplicatedCall_shouldDoXGivenSomeNonNull() throws Exception {
    MyObject obj = new MyObject();
    String output = obj.someComplicatedCall("someArgValue");
    assertThat(output, notNullValue());
}
}
```

3) Run your test class in eclipse:

1. Right click on the class name in the "Navigator" or "Package Explorer" view
2. Select "Run As" -> JUnit Test

**Testing the OpenMRS API**

(and other classes that use the OpenMRS Context object) To test a class that requires the Context object and its method calls, simply extend org.openmrs.
test.BaseContextSensitiveTest. The BaseContextSensitiveTest class will run through the Spring setup to put the *Services into the Context object. This
startup takes a few seconds, so when you can, create simple tests (see above) and do not call methods on Context.

1) Identify the interface for the service you’re testing:

```java
package org.openmrs.api;

/**
 * This service does blah blah blah...
 */
public interface MyService {
    ...

/**
 * This method does something blah blah blah
 * @param fname
 * @param lname
 * @return patients
 */
// NB the interesting details are in the implementation
public List<Patient> findPatient(String fname, String lname) throws Exception;
```

2) Create a test for this method:
package org.openmrs.api;

import static org.hamcrest.CoreMatchers.hasSize;
import static org.junit.Assert.assertThat;

/**
 * Tests on the methods found in MyService
 */
public class MyServiceTest extends BaseContextSensitiveTest {

    /**
     * Setup run before all of our tests
     */
    @Before
    public void setup() {
        initializeInMemoryDatabase();
        executeDataSet("org/openmrs/test/api/myservice/include/patients.xml");
    }

    ...

    /**
     * Check that output looks right for the short patients
     */
    @Test
    public void findPatients_shouldGetPatientsGivenValidFirstNameAndLastName() throws Exception {
        List<Patient> patients = Context.getPatientService().getPatients("John Doe", null, null, false);
        assertThat(patients, hasSize(5));
    }

    ...
}

3) Run your test class. In eclipse:

1. Right click on the class name in the "Navigator" or "Package Explorer" view
2. Select "Run As" > JUnit Test

Setting up the expected database

By default, the Context is set up to run against an in-memory database called h2. This database is automatically built from the Hibernate mapping files. BaseContextSensitiveTest classes will automatically load the initialInMemoryTestDataSet.xml and the standardTestDataset.xml. You may use any objects defined there for your tests. To authenticate your test to the Context object, call authenticated() . This will pop up a box asking the current user for an admin/password.

Creating an xml dataset for your unit test

You can add your own test data xml files. By convention, test data sets are placed in src/test/resources in the project structure. It's easiest to generate the xml off of your current openmrs mysql database. Use the org.openmrs.test.CreateInitialDataSet file to generate it. You can write data sets by hand as well. Look at unit test data sets in the source code repository for examples.

To load your data set, call

executeDataSet("YourTestDataSyncCreateTest.xml"); // You may wish to add this executeDataSet call to your method annotated with @Before so that it runs before each test

Testing against your own mysql database

This is not recommended, and not valid for committed unit tests. Create a xml dataset and use the in-memory database for everyone to use. To use your mysql database defined by your runtime properties, override the `BaseContextSensitiveTest.useInMemoryDatabase()` method and have it return false.

Batch JUnit Testing
Run "mvn test" at a command line
The generated reports can be viewed in /target

Committing to the Database

In order to have your Context sensitive tests actually commit to the database instead of rolling back at the end, all you need to do is add the @Rollback (false) annotation in your test method. You can also call getConnection().commit() or even use @NotTransactional and rely on methods which you test to manage transactions. Please remember that if you commit to the database, you need to clean up after yourself by calling for example deleteAllData() at some point. Otherwise data you committed will remain in the database and affect execution of subsequent tests, which is a bad practice.

There are really only two use-cases for doing this:

1. you're not using an in-memory hsql database and you want to look at the output
2. you need to check something that happens after the transaction has closed.

Case #1 should only be a temporary test and should not be committed. Case #2 must have a cleanup test if it is to be run with other tests. This cleanup test should delete all the data in the db so that future tests' integrity is not compromised.

```java
@Test
@Rollback(false)
public void firstTestThatDoesntRollbackThings() {
    iLeaveDataInTheDatabase();
}

@Test
@Rollback(false)
public void secondTestToCleanUpAfterTheFirst() {
    try {
        doSomeMoreTestsHereOnTheNonRolledBackData();
    }
    finally {
        deleteAllData();
    }
}
```

See UserServiceTest.java #shouldCreateUserWhoIsPatientAlready() for an example.

JUnit Environment Variables

JUnit Environment variables can override system environment variables. For example, if you want a different OPENMRS_RUNTIME_PROPERTIES_FILE for JUnit.

1. Right Click on a JUnit test. For example, CreateInitialDataSet.java.
2. Select Run As --> Open Run Dialog
3. Select the Environment tab and add a New environment variable.

Resources

• See Module Unit Testing for using JUnit in modules.
• See Load Testing
• See GUI Unit Testing

FAQ

• How can I authenticate to the Context without putting my username and password in the unit test code?
  Use the authenticate() method found in BaseContextSensitiveTest
• That authenticate method is awesome...but I want to authenticate without user intervention. How can I do that easily?
  Define two properties in your runtime properties file: junit.username and junit.password.
• Why am I getting the error org.openmrs.api.APIException: Service not found: interface org.openmrs.api ..... in my unit test?
  You're trying to use a basic unit test to do Context/API testing. Extend the org.openmrs.test.BaseContextSensitiveTest class.
• Why am I getting the error: Already value [] for key [] bound to thread [main]?
  Your web unit test is probably extending BaseContextSensitiveTest instead of BaseWebContextSensitiveTest
• Why am I getting the error: org.openmrs.api.APIException: contextDAO is null . . . in my unit test?
  Spring was not properly set up for OpenMRS. Try running a maven install on your local copy of OpenMRS (In Eclipse click run as, then select maven install).