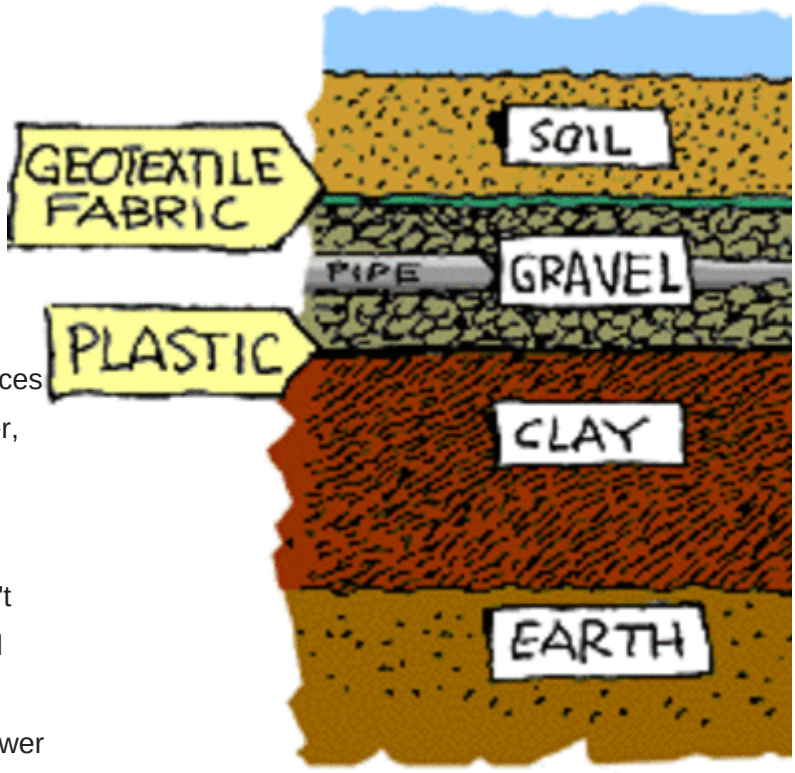
How To Sort Your Unit Tests By Layer In TestNG

By Roger Keays, 11 October 2012

Each of my **unit tests** extends from a different **base class** depending on what layer of my application they are testing. i.e.




```
BaseTest
 `- BaseDBTest
     `- BaseOpsTest
         `- BaseUITest
```

These superclasses are very convenient places to put **shared test code** like starting a server, opening the database, doing a HTTP get or even just checking that assert is enabled.

When testing the whole application, it doesn't make much sense to me to run tests from all layers in **random order**. There are clear dependencies between the layers and if a lower layer fails I want to know about it **before** it blows up into the upper layers (if you believe its better to mock unit test dependencies then you might as well stop reading now).

Since I use **maven's surefire plugin** to run my tests there is no **testng.xml** and that's the way I like it. The problem is setting up TestNG dependencies in the tests to get them to run in the right order.

The brute force approach is to add **groups** and **dependsOnGroups**attributes to every leaf test class. Unfortunately you can't put them on the superclasses because they inherit each other and you end up with circular dependencies.

```
@Test(groups={"DB"})
public class WidgetDBTest extends BaseDBTest {}

@Test(groups={"UI"}, dependsOnGroups={"DB"})
public class WidgetUITest extends BaseUITest {}
```

This works fine except when you want to run WidgetUITest alone. TestNG will fail because the test depends on a non-existing group called "DB". AlwaysRun doesn't help us here because the

dependency doesn't fail - it just can't be found, and IgnoreMissingDependencies isn't implemented for groups.

The best workaround I found for this problem is to put an empty test in the base class which ensures that a test group for each layer is always created:

```
public class BaseTest {

    @Test(groups={"Java", "DB", "Ops", "UI"})
    public void create_test_groups() {}
}
```

You can stop there if you're happy with that solution. Personally I don't see why I should annotate all my subclasses when they all have the same superclass, so I took advantage of TestNG's IAnnotationTransformer listener like this:

```
public class TestGrouper implements IAnnotationTransformer {

@Override
public void transform(ITestAnnotation test, Class testClass,
Constructor c, Method method) {

// get the class this test is on
Class layer;
if (method != null) {
layer = method.getDeclaringClass();
} else if (c != null) {
layer = c.getDeclaringClass();
} else if (testClass != null) {
layer = testClass;
} else {
throw new IllegalArgumentException(
"Couldn't find declaring class for test");
}

// setup layer groups based on the test class
String addGroup = null;
String addDepends = null;
if (BaseUITest.class.isAssignableFrom(layer)) {
addGroup = "UI";
addDepends = "Ops";
```

```
} else if (BaseOpsTest.class.isAssignableFrom(layer)) {
addGroup = "Ops";
addDepends = "DB";
} else if (BaseDBTest.class.isAssignableFrom(layer)) {
addGroup = "DB";
addDepends = "Java";
} else if (BaseTest.class.isAssignableFrom(layer)) {
addGroup = "Java";
}

// append to existing groups and dependencies
if (addGroup != null) {
int length = test.getGroups().length;
String [] groups = new String[length + 1];
System.arraycopy(test.getGroups(), 0, groups, 0, length);
groups[length] = addGroup;
test.setGroups(groups);
}
if (addDepends != null) {
int length = test.getDependsOnGroups().length;
String [] depends = new String[length + 1];
System.arraycopy(test.getDependsOnGroups(), 0, depends, 0, length);
depends[length] = addDepends;
test.setDependsOnGroups(depends);

// TestNG has no way to ignore group dependencies, so this method
// relies on the superclass to setup test groups with
// @Test(groups={"Java", "DB", "..."})
}
}
}
```

Since I'm launching from maven this listener goes in the`pom.xml` plugin configuration. Otherwise you pass it to TestNG with `-listener` on the command line.

```
<!-- unit test configuration -->
 <plugin>
   <artifactId>maven-surefire-plugin</artifactId>
   <version>2.12</version>
   <configuration>
     <properties>
       <property>
         <name>listener</name>
```

```
            <value>com.example.TestGrouper</value>
          </property>
        </properties>
      </configuration>
    </plugin>
```

I would have liked to implement this with test **priorities**, but TestNG buckets test priorities AFTER it calculates dependencies so it ends up running UI tests with no dependencies before DB tests with dependencies even though the DB tests have higher priority.

On the other hand, the grouping method is useful for reporting and testing individual layer groups.

About Roger Keays

Roger Keays is an artist, an engineer, and a student of life. He has no fixed addressand has leftfootprints on 40-something different countries around the world. Roger is addicted to surfing. His other interests are music, psychology, languages, the proper use of semicolons, and finding good food.