Unit-Test style

Your Rating: Results: 101 rates

With Magnolia 4.5 we switched to JUnit 4. As it is fully backwards compatible, this does not enforce you to write your tests in the new v4-style (e.g. use @Test, @Before etc.).

Junit 4 style

- all new test will be written in JUnit 4 style
- existing tests have been converted SCRUM-317

Behavior Driven Development (BDD) Style

- in order to increase readability, we use the bdd-like style: *// GIVEN*, *// WHEN*, *// THEN*
- test-method names should describe what they're doing (-> proper sentence)
- as in all our java code we use camelCaseStyle names for test and their methods

pure Mockito

```
// GIVEN
<all code required to set up the situation goes here or in dedicated setUp (@Before) methods>

// WHEN
< execute whatever method you actually want to test>

// THEN
<verify your expectations>
```

More on testing Magnolia with Mockito can be found here.

Granularity

If you're testing multiple scenarios (having multiple // WHEN - // THEN section is an indicator for that), put each of them into a dedicated test method:

- if an early scenario fails, the others will still be executed: you'll always exactly know what is broken and what not
- as you have explicit test methods, you can name them explicitly: helps a lot to understand - not only when one of them is failing
    - e.g. if you wanted to test a String-To-Integer-Parser
        - canParsePositiveInteger
        - canParseNegativeInteger
        - throwsParseExceptionOnInvalidChars

Matchers

An example will be more explicit than a verbose explanation:

```
        assertThat(filterNode, hasContent("clientCallbacks", "mgnl:content"));

        // you can even apply multiple conditions in one go:
        assertThat(filterNode, allOf(
                not(hasContent("clientCallback")),
                hasContent("clientCallbacks", "mgnl:content")
        ));
```

... and a sample output when the test fails:

```
java.lang.AssertionError:
Expected: a child node named 'clientCallbacks' of type 'mgnl:content'
     but: config:/server/filters/dummySecurityFilter[mgnl:content] has a child node named 'clientCallbacks' but
it is not of type 'mgnl:content'
```

Much more expressive that a series of calls to the content/jcr apis and regular assert methods right ?
This is done via Hamcrest matchers.

Make sure you have the following imports

```
import static info.magnolia.test.hamcrest.ContentMatchers.*;
import static info.magnolia.test.hamcrest.UtilMatchers.*;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.CoreMatchers.*;
import static org.junit.Assert.*;
```

`org.junit.Assert` also has an `assertThat()` method, which works just as well, but Hamcrest's describe the problems **much** better, which is why the hamcrest imports need to be above junit's (which should be fine alphabetically).
See https://github.com/KentBeck/junit/issues/36 for an update on this.

Have a look at the existing matchers in Hamcrest, as well as ours in the `info.magnolia.test.hamcrest` package
... then use Google for more examples.

Tests expecting Exceptions

Various techniques are possible (see Greg's examples at git.magnolia-cms.com:user/gjoseph/kt/matchers.git (ExceptionMatchingTest), but we favor this most readable and controllable one:

```
/**
 * @see info.magnolia.test.hamcrest.ExecutionMatcherTest for more examples !
 */
@Test
public void executionMatchers() {
    assertThat(new Execution() {
        public void evaluate() throws Exception {
            divide(4, 0);
        }
    }, throwsAnException(instanceOf(ArithmeticException.class).withMessage(containsString("zero"))));
}

/**
 * Use Java8's lambdas for readability/conciseness.
 */
@Test
public void executionMatchersWithJava8() {
    assertThat(() -> divide(4, 0), throwsAnException(instanceOf(ArithmeticException.class).withMessage
(containsString("zero"))));

}
```

Clean up global state

It's important that your test dont leave things behind and risk affecting other tests.

- Always do Components.setProvider(null) if you set a ComponentProvider
- Always call MgnlContext.setInstance(null) if you set a Context
- Always reset any properties you set

Always add this in you test (it gets executed after every test method):

```
    @After
    public void tearDown() throws Exception {
        ComponentsTestUtil.clear();
        SystemProperty.clear();
        MgnlContext.setInstance(null);
        Components.setProvider(null);
    }
```

Temporarily disable Test

For the rare case that you have to temporarily disable tests, always use the @Ignore annotation - don't comment out whole test methods because

- this will preserve the imports
- will be considered when refactoring
- it's easily possible to search for ignored tests and reanimate them (not obvious to find commented out tests)