Object Partners Inc.

Grails 1.3.x Testing Cheat Sheet

Command Line

All tests	grails test-app
Failed test	grails test-app -rerun
Specific test class	grails test-app PresentationController
Specific test case	grails test-app PresentationController.testFoo
Only unit test	grails test-app unit:
Only integration test	grails test-app integration:
Only Controller test	grails test-app *Controller
Only test in a package	grails test-app com.foo.*
Show output	grails test-app -echoOut -echoErr
Create Integration Test	grails create-integration-test *Name*
Create Unit Test	grails create-unit-test *Name*

Unit Testing

Rules of Thumb:

- 3 A's of a Test = Arrange, Act, Assert
- No I/O (DB, File, Socket, etc)
- Test Behavior NOT the Platform
- Use Test Double for all collaborators

Test Doubles

Stubs: Simplest possible stand-in implementation for the real thing. (Map of closures)

Spies: Used to capture and observation point the class under test. (Map of closures or mockFor)

Mocks: Strictest of the 3; used when you want control over input, output, method call count, and method call order. (mockFor with verifyMock() call)

What gets mocked out: Controllers

Controller Methods	Helper Methods	
redirect	getRedirectArgs	getResponse
forward	getForwardArgs	getSession
chain	getChainArgs	getParams
render	getRenderArgs	getFlash
withFormat	getTemplate	getChainModel
withForm	getModelAndView	getActionName
log	setModelAndView	getControllerName
	getRequest	getServletContext

What gets mocked out: Domain

findAll	load	validate	delete
findAllWhere	getAll	getErrors	discard
findAllBy*	ident	hasErrors	refresh
findBy*	exists	setErrors	attach
get	count	clearErrors	addTo*
read	list	save	removeFrom*
create			

= don't do anything in testing

Object **Partners** Inc.

Grails 1.3.x Testing Cheat Sheet

Basic JUnit Assertions

- assertTrue(true)
- assertFalse(false)
- assertEquals(expect, actual)
- assertNull(val)
- assertNotNull(val)
- assertArrayEquals(char[], char[])
- assertNotSame(obj1, obj2)
- assertSame(obj1, obj2)
- assertThat(actual, matcher)
- fail()

mockForConstraintsTest(*Class*)

```
void testEventNameShouldNotValidateIfNull() {
    def event = new Event()

    assertFalse event.validate()
    assertTrue event.hasErrors()

    assert event.errors['name'] == 'nullable'
}
```

Domain Expectations Example

```
void testEventNameConstraints() {
    Expectations.applyTo Event

    def event = new Event()

    event.expectNameIsNotNullable()
    event.expectNameIsNotBlank()
    event.expectNameHasMaxSize(40)
    event.expectNameHasAValidator()
    event.expectNameIsValid()
}
```

Controller Unit Test Example

```
void testAddSucceedsWhenSpeakerIsFound() {
    controller.params.title = "Test Presentation"
    def user = new User(id:1)
    controller.springSecurityService = [currentUser: user]
    controller.accessCodeService =
        [createFrom: {title, event -> "abcd1234"}]
    mockDomain Presentation

    controller.add()

    controller.with{
        assert redirectArgs.size() == 2
        assert redirectArgs.controller == "speaker"
        assert redirectArgs.action == "index"
        assert flash.message ==
            "'Test Presentation' added with access code:
abcd1234"
    }
}
```

mockConfig()

```
mockConfig('''
    braintree {
        merchantKey = "abcd1234"
    }
''')
```

OR

```
mockConfig(
    new File('grails-app/conf/Config.groovy').text
)
```

Object **Partners** Inc.

Grails 1.3.x Testing Cheat Sheet

Spock Test Cases:

Testing Plugin	Spock Plugin
GrailsUnitTestCase	UnitSpec
ControllerUnitTestCase	ControllerSpec
TagLibUnitTestCase	TagLibSpec
GroovyTestCase	IntegrationSpec
GroovyPagesTestCase	GroovyPagesSpec

Spock Basic Parameterized

```
class ParameterizedSpec extends UnitSpec{

    @Unroll("#a + #b = #c")
    def "summation of 2 numbers"() {
        expect: a + b == c

        where:
            a |b  |c
            1 |1  |2
            2 |1  |3
            3 |2  |6
    }
}
```

Error output:

```
Condition not satisfied:
a + b == c
| | | |  |
3 5 2 |  6
      false
```

Spock Parameterized Controller Tests

```
class PresentationControllerSpec extends ControllerSpec{

    @Shared def user = new User(id:1)

    @Unroll("add presentation for user: #testUser with title: #title")
    def "call to add a new presentation for logged in user"() {
        given:"collaborators are mocked"
            mockDomain Presentation
            mockDomain Speaker, [new Speaker(user:user)]
            controller.springSecurityService = [currentUser: testUser ]
            controller.accessCodeService =
                  [createFrom:{title, event -> "1234abcd"}]
        and:"prarams are set"
            controller.params.event = "Event Name"
            controller.params.date = new Date()
            controller.params.title = title
        when:"call the action"
            controller.add()

        then:"should be redirected to '/speaker/index'"
            controller.redirectArgs.controller == 'speaker'
            controller.redirectArgs.action == 'index'

        and: "should have correct flash message"
            controller.flash.message == expectedFlashMessage

        where:
            title | testUser |expectedFlashMessage
            'test'| user     |"'test' added with access code: 1234abcd"
            null  | user     |"Could not add null to your list at this time"
            'test'| null     |"Could not find a Speaker to add test to your
list at this time"
    }
}
```

Object **Partners** Inc.

Grails 1.3.x Testing Cheat Sheet

Url Mapping Testing

```
void testDefaultControllerActionIdMapping() {
    assertUrlMapping(
      "/event/show/3210",
      controller: 'event',
      action: 'show') { id = 3210 }

    assertUrlMapping(
      "/presentation/show/125",
      controller: 'presentation',
      action: 'show') { id = 125 }
}

void testErrorPages() {
    assertUrlMapping(500, view: "error")
    assertUrlMapping(404, view: "error404")
}
```

Tag Lib Testing

```
class LoopbackTagLibTests extends TagLibUnitTestCase {
    protected void setUp() {
        super.setUp()
    }

    protected void tearDown() {
        super.tearDown()
    }

    void testFixedWidthIpDefaultsTag() {
        tagLib.fixedWidthIp(null, null)
        assertEquals ".............", tagLib.out.toString()
    }

    void testFixedWidthIpParameters() {
        tagLib.fixedWidthIp(
            ip: '127.0.0.1',
            width: '20',
            pad: '=', null
        )

        assertEquals "127.0.0.1==========", tagLib.out.toString()

        shouldFail {
            tagLib.fixedWidthIp(
                  ip: '127.0.0.1',
                  width: 'not-a-number',
                  pad: '=', null
            )
        }
    }
}
```

GSP Test:

```
class LoopbackTagLibGSPTests extends GroovyPagesTestCase {
    void testFixedWidthIp() {
        def template = '<g:fixedWidthIp ip="127.0.0.1" width="21"
pad="-"/>'

        assertOutputEquals('127.0.0.1----------', template)
    }

    void testComment() {
        def template = '<g:comment comment="${comment}"/>'

        Comment comment = new Comment(
                id: 1,
                dateCreated: Date.parse("MM/dd/yyyy","6/21/2011"),
                text: 'comment_text',
                clientIPAddress: '127.0.0.1'
        )
        def expected = """<div class="comment" id="comment_">
    <h2>comment_text</h2>
    <div class="commentInfo">
        <span class="comment_date">Jun 21, 2011 @ 12:00 AM </span>
        <span class="comment_client">127.0.0.1</span>
    </div>
</div>
"""
        assertOutputEquals(expected, template, [comment: comment])
    }
}
```


Zan Thrash

@zanthrash

zan.thrash@objectpartners.com

Colin Harrington

@ColinHarrington

colin.harrington@objectpartners.com

Integration Service Test

```
class CommentServiceTests extends GrailsUnitTestCase {

    def commentService

    void testCommentsForIp() {
        Presentation presentation = Presentation.build()
        presentation.addToComments(
                Comment.build(
                        clientIPAddress: "192.168.0.1",
                        text: "wow awesome"
                )
        )

        presentation.addToComments(
                Comment.build(
                        clientIPAddress: "192.168.0.1",
                        text: "Superb description"
                )
        )
        presentation.addToComments(
                Comment.build(
                        clientIPAddress: "127.0.0.1",
                        text: "This needs work"
                )

        )
        presentation.save(failOnError: true)


        assertEquals(
            2,
            commentService.commentsForIp(
                "192.168.0.1",
                presentation
            )?.size())

        assertEquals(
            1,
            commentService.commentsForIp(
                "127.0.0.1",
                presentation
            )?.size())

    }
}
```