Spock testing framework

http://docs.spockframework.org

https://github.com/lsparlin/spock-sandbox

https://github.com/kensipe/spock-demos-nfjs

Spock

Why Spock?

Spock

Why Spock?

- Expressive and concise

Spock

Why Spock?

- Expressive and concise
- Very easy to read

Spock

Why Spock?

- Expressive and concise
- Very easy to read
- Runs with jUnit test runner

Spock

Why Spock?

- Expressive and concise
- Very easy to read
- Runs with jUnit test runner
- Mocking and Stubbing is built-in

Spock

Why Spock?

- Expressive and concise
- Very easy to read
- Runs with jUnit test runner
- Mocking and Stubbing is built-in
- Enables more intense testing

Spock

Why Spock?

- Expressive and concise
- Very easy to read
- Runs with jUnit test runner
- Mocking and Stubbing is built-in
- Enables more intense testing
- Very informative failure messages

Spock

Why Spock?

- Expressive and concise
- Very easy to read
- Runs with jUnit test runner
- Mocking and Stubbing is built-in
- Enables more intense testing
- Very informative failure messages

Spock vs jUnit

Spock vs jUnit

jUnit

```
public class GuavaJoinerTest {
    private Joiner joiner;

    @Before
    public void setUp() {
        joiner = Joiner.on(", ");
    }

    @Test
    public void testBasicStringJoins() {
        String[] input = new String[] { "one", "two", "three" };
        assertEquals("one, two, three", joiner.join(input));

        input = new String[] { "", "four" };
        assertEquals(", four", joiner.join(input));

        Example example1 = new Example();
        example1.setName("First");
        Example example2 = new Example();
        example2.setName("Second");
        Example[] exampleInput = new Example[] { example1, example2 };
        assertEquals("Example [name=First], Example [name=Second]",
                     joiner.join(exampleInput));

        input = new String[0];
        assertEquals("", joiner.join(input));
    }
```

Spock vs jUnit

Spock

```
class GuavaJoinerSpec extends Specification {
    def joiner = Joiner.on(", ")

    def "join() joins object toStrings on seperator"() {
        expect:
        joinResult == joiner.join(input)

        where:
        input                                   | joinResult
        []                                      | ""
        ["one", "two", "three"]                 | "one, two, three"
        ["", "four"]                            | ", four"
        [ new Example(name: "First"),
            new Example(name: "Second")]        | "Example [name=First], Example [name=Second]"
    }
```

Groovy Crash Course

Groovy Crash Course

Differences from Java

Groovy Crash Course

Differences from Java

- Semicolons optional

Groovy Crash Course

Differences from Java

- Semicolons optional
- return keyword optional

Groovy Crash Course

Differences from Java

- Semicolons optional
- return keyword optional
- methods and classes are public by default

Groovy Crash Course

Differences from Java

- Semicolons optional
- return keyword optional
- methods and classes are public by default
- closures

Groovy Crash Course

Differences from Java

- Semicolons optional
- return keyword optional
- methods and classes are public by default
- closures
- Dynamic *or* static typing

Groovy Crash Course

Differences from Java

- Semicolons optional
- return keyword optional
- methods and classes are public by default
- closures
- Dynamic *or* static typing
- Ability to overload operators

Groovy Crash Course

Differences from Java

- Semicolons optional
- return keyword optional
- methods and classes are public by default
- closures
- Dynamic *or* static typing
- Ability to overload operators

Spock Basics

Spock Basics

Basic Specification

```
import spock.lang.Specification

class FirstSpec extends Specification {

    // fields
    // fixture methods (setup, cleanup)
    // feature methods (tests)
    // helper methods

}
```

Spock Basics

Feature Methods

```
def simpleFeature() {
    // blocks
}
```

Spock Basics

Feature Methods

```
def "simple feature description"() {
    // blocks
}
```

Spock Basics

Feature Methods - Blocks

setup:

when:

then:

where:

Spock Basics

Feature Methods - Blocks

setup: = **given:**

when:
then: = **expect:**

where:

Spock Basics

Feature Methods - Blocks

```
def "simple feature"() {
    setup:
    // prepare objects for testing

    when:
    // provide stimulus  to code under test

    then:
    // express expected response to stimulus

    cleanup:
    // any cleanup steps (optional)
}
```

Spock Basics

Stubs

```
class FirstSpec extends Specification {
    ExampleDao exampleDao = Stub()

    def setup() {
        exampleDao.findOne(_) >> null
    }
```

Spock Basics

Mocks

```
class FirstSpec extends Specification {
    ExampleDao exampleDao = Mock()

    def simpleFeature() {
        setup: // setup object under test

        when: // apply stimulus

        then:
        1 * exampleDao.findOne(1) >> null
    }
```

Spock Basics

Benefits of Mocking

Spock Basics

Benefits of Mocking

- Mock behavior of dependency classes

Spock Basics

Benefits of Mocking

- Mock behavior of dependency classes
- Isolate code under test

Spock Basics

Benefits of Mocking

- Mock behavior of dependency classes
- Isolate code under test
- Fast

Spock Basics

Benefits of Mocking

- Mock behavior of dependency classes
- Isolate code under test
- Fast

Spock Demonstration

…

Getting Spock

ANT

- Search Artifactory
  - groovy-all-2.1.2.jar
  - spock-core-0.7-groovy-2.0.jar
- Include in WEB-INF/lib

Artifactory: http://apt.oreillyauto.com:8080/artifactory

Getting Spock

Gradle

```
...
apply plugin: 'groovy'
...
dependencies {
    groovy "org.codehaus.groovy:groovy-all:2.1.2"
    groovy "org.spockframework:spock-core:0.7-groovy-2.0"
}
...
war {
    classpath = classpath - project.configurations.groovy
}
```

Summary

Summary

- Starting knowledge of Spock

Summary

- Starting knowledge of Spock
- Groovy Basics

Summary

- Starting knowledge of Spock
- Groovy Basics
- Desire to use Spock to improve Unit Testing!

Summary

- Starting knowledge of Spock
- Groovy Basics
- Desire to use Spock to improve Unit Testing!

Questions

…