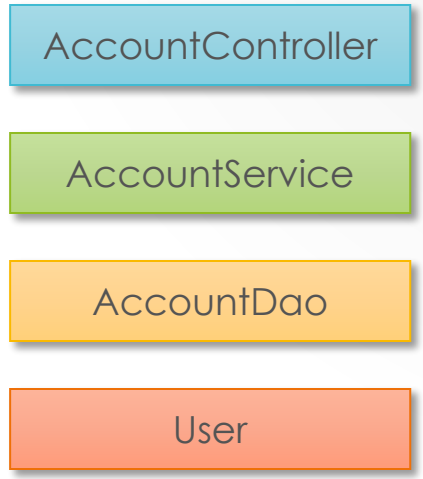
UNIT TESTING

"Imperfect tests, run frequently, are much better than perfect tests that are never written at all“
By: tjjenk2

JUNIT FRAMEWORKS

UNIT TESTING CONCEPTS

- Isolate your tests
- Focus on a specific concern
  - Expected behavior
  - Excepted state change
- Avoid brittle unit tests
  - e.g. counting properties (properties change often)
- Keep testing performance high
- Mock external functionality
  - **Lenient**: Irrelevant method calls that aren't needed for the test on mock objects are allowed and are answered with a default response (e.g. false, 0, null)
  - **Strict**: an exception would be thrown for every unexpected method call



PROBLEMS WITH JUNIT

Concept	JUnit	Hamcrest	Mockito	PowerMock	JsonAssert	spring-test
Limited Assertions	X					
Powerful Assertions		X				
Mocking Stubbing Spys			X			
Mock private, static, final, constructor, etc.				X		
JSON Asserts					X	
Test private field or method						X

THE SPOCK FRAMEWORK

WHY SPOCK?

Concept	Framework Types	
Unit Testing	JUnit TestNG	Spock Framework
Mocking / Stubbing / Spies	Mockito JMockit EasyMock jMock PowerMock	
Behavior Driven Design	Cucumber JBehave	

- It incorporates the best concepts from many frameworks
- It is for Java and Groovy applications, but leverages Groovy syntax
- It has built-in mocking and stubbing
- It has succinct syntax for data driven testing which can greatly reduce code
- Other frameworks, like PowerMock and Hamcrest, can still be used with Spock

SPOCK TEST CLASS

```
import spock.lang.*

class YourSpockSpec extends Specification {
```

Fields

- Static Fields – should only be used for constants; otherwise use @shared fields
- Instance Fields – are not shared between feature methods
- @Shared Fields – are shared between feature methods

Fixture Methods

Life Cycle Methods (all optional)

- setupSpec() and cleanupSpec() – invoked before/after the first/last feature method
- setup() and cleanup() – invoked before/after each feature method

Feature Methods

Test Methods

An Explicit Phase Block is Required

- Setup Phase (optional): setup: / given:
- Stimulus Phase: expect: / when:
- Response Phase: expect: / then:
- Cleanup Phase (optional): cleanup:
- data-driven (optional, always last): where:

Helper Methods

NO Phase Blocks

```
}
```

SIMPLE SPOCK TEST CLASS

```
import spock.lang.*

class MyFirstSpec extends Specification {
  def "example spock feature" () {
    given:
        def mapGroovy = [:]
        Map<String,String> mapJava =  new HashMap<>()

    when:
        mapJava.put(null, "elem")
    then:
        notThrown(NullPointerException)
        mapJava instanceof HashMap

    and: "I am syntactic sugar"

    when:
        mapGroovy << [ null : "elem" ]
    then:
        notThrown(NullPointerException)
        mapGroovy  instanceof LinkedHashMap
  }
}
```

CONDITION

- Describes an expected state
- Similar to assert
- Written as a plain boolean expression
- Can produce a non-boolean value; evaluated as a Groovy truth
  - Non-empty Collections and arrays are true
  - Non-zero numbers are true
  - Non-null object references are coerced to true

```
Math.max(1, 2) == 7// condition (groovy power assert)

Condition not satisfied:

Math.max(1, 2) == 7
    |          |
    2          false

at SomeSpec.someFeature (Script1.groovy:#)
```

INTERACTION

- Describes how objects communicate with each other by way of method calls (a way to test behavior)
- Accomplished with mocking
- Internally, when declared in then: block are moved to before the preceding when: block
- Invocation order is enforced between, but not within then: blocks
- They are always scoped to a feature method; cannot declare in:
  - a static method, setupSpec method, or cleanupSpec method

```
An interaction has 4 parts:

1 * subscriber.receive("hello")
|   |           |       |
|   |           |       argument constraint
|   |           method constraint
|   target constraint
cardinality
```

INTERACTION CONTINUED

Cardinality

```
1 * subscriber.receive("hello")      // exactly one call
0 * subscriber.receive("hello")      // zero calls
(1..3) * subscriber.receive("hello") // between one and three calls (inclusive)
(1.._) * subscriber.receive("hello") // at least one call
(_..3) * subscriber.receive("hello") // at most three calls
_ * subscriber.receive("hello")      // any number of calls, including zero
                                     // (rarely needed; see 'Strict Mocking')
```

Target Constraint

```
1 * subscriber.receive("hello") // a call to 'subscriber'
1 * _.receive("hello")          // a call to any mock object
```

Method Constraint

```
1 * subscriber.receive("hello") // a method named 'receive'
1 * subscriber./r.*e/("hello")  // a method whose name matches the given regular expression
                                // (here: method name starts with 'r' and ends in 'e')
```

Argument Constraint

```
1 * subscriber.receive("hello")     // an argument that is equal to the String "hello"
1 * subscriber.receive(!"hello")    // an argument that is unequal to the String "hello"
1 * subscriber.receive()            // the empty argument list (would never match in our example)
1 * subscriber.receive(_)           // any single argument (including null)
1 * subscriber.receive(*_)          // any argument list (including the empty argument list)
1 * subscriber.receive(!null)       // any non-null argument
1 * subscriber.receive(_ as String) // any non-null argument that is-a String
1 * subscriber.receive({ it.size() > 3 }) // an argument that satisfies the given predicate
                                          // (here: message length is greater than 3)
```

FEATURE METHOD PHASE BLOCKS

SETUP: BLOCK

- Performs feature initialization
- It must be first
- given: is an alias (sometimes used for BDD)
- It is optional; implicit declaration occurs when omitted

```
setup:
   def dateFormatStr = "yyyyMMdd HHmmss"
   def stack = new Stack()
```

CLEANUP: BLOCK

- Used to free any resources that were used by the feature method
- It must be last (unless there is a where:)
- It is invoked even if an exception occurs in a previous block
- Must be coded defensively
- It is optional; implicit declaration occurs when omitted

```
cleanup:
    file?.delete() // assuming you created the file in the method
```

EXPECT: BLOCK

- Describe stimulus and response in a single expression
- All expressions are implicitly treated as conditions
  - Only then: and expect: can do this – to do this elsewhere, you must use **assert**
- Can only contain conditions and variable definitions
- Use to describe purely functional methods (versus when/then)

```
expect:
   def expected = 2          // variable definition
   Math.max(1, 2) == expected // condition (similar to Junit assert)
```

WHEN: THEN: BLOCK

- Together, they describe a stimulus and expected response
- when: blocks can contain arbitrary code
- then: blocks are restricted to:
  - Conditions, exception conditions, interactions, and variable definitions
- A feature method can contain multiple pairs of when-then blocks

```
when:
    String invalidValue = null                          // variable definition
    DatasourceType.determineValue(invalidValue)  // stimulus
then:
    thrown(IllegalArgumentException.class)          // response
```

WHERE: BLOCK

- Used to write data-driven feature methods
- Must be the last block, but can be repeated
- Two ways to implement
  - Data pipes (variable << provider)
    - connects a data variable to a data provider
    - Any object that groovy can iterate over can be used as a provider
    - After all iterations have completed, the close() method is implicitly called on provider
  - Data table
    - Must have at least two columns (can use _ if only need one column)
- Each iteration gets its own instance and setup() and cleanup() are called before each
- To share objects for each iteration, they must be static or @Shared
- Use @Unroll to treat each iteration as a separate, trackable test

WHERE: BLOCK EXAMPLE

```
import spock.lang.*

class MyFirstSpec extends Specification {
  @Unroll
  def "method names can have variables (a,b,c) = (#a, #b, #c)" () {
    expect:
        Math.max(a, b) == c

    where: "one value can come from a data table"
        a | _
        5 | _
        1 | _

     and: "other values can come from a data pipe"
        b << [1, 9]
        c << [5, 9]
  }
}
```

```
MyFirstSpec (two tests completed successfully)
- method names can have variables (a,b,c) = (5, 1, 5)
- method names can have variables (a,b,c) = (1, 9, 9)
```

MOCKS, STUBS, AND SPIES