sSpec Quick Reference

Core API

Arbitrary Block
```
object should satisfy: [:arg| …]
object should not satisfy: [:arg| …]
```

Equality
```
object should equal: <value>
object should not equal: <value>
```

Floating Point Comparison
```
object should be within: <delta> of: <val>
object should not be within: <delta> of: <val>
```

Identity
```
object should be: <value>
object should not be: <value>
```

Arbitrary Predicate
```
object should predicate
object should be predicate
object should not predicate
object should not be predicate
```

Direct Instance
```
object should be an instance of: <class>
object should not be an instance of: <class>
```

Ancestor Class
```
object should be a kind of: <class>
object should not be a kind of: <class>
```

Type
```
object should respond to: <symbol>
object should not respond to: <symbol>
```

Raising
```
[] should raise: <exception>
[] should not raise: <exception>
[] should raise
[] should not raise
```

Containment
```
object should include: <object>
object should not include: <object>
```

Index Containment
```
object should include key: <object>
object should not include key: <object>
```

Exact Size
```
object should have: <n> in: #things
```

Lower Bound
```
object should have at least: <n> in: #things
```

Upper Bound
```
object should have at most: <n> in: #things
```

Mock API

Creating a mock
```
self mock: <name>
self mock: <name> withOptions: <options>

self mock: 'm' withOptions: #(nullObject)
```

Expecting Messages
```
mock shouldReceive: <selector>
```

Remaining mock related messages are sent to the result of this send.

Arbitrary Message Handling
```
andDo: <block>
```

Expecting Arguments
```
omitted to accept any arguments
with: #noArgs
with: arg1
with: arg1 with: arg2
with: arg1 with: arg2 with: arg3
with: arg1 with: arg2 with: arg3 with: arg4
withAll: aSequencableCollection
```

Argument Constraints (used in the above)

`#anything, #numeric, #boolean, #string`
`DuckTypeArgConstraint with: #size`

accepts anything that responds to all of the message(s), forms with up to 4 `with:` clauses, and a `withAll:`

Receive Counts
```
never
anyNumberOfTimes
once
twice
exactly: n times
atLeastOnce
atLeastTwice
atLeast: n times
atMost: n times
```

Return Values
```
andReturn: valueOrBlock
```

Consecutive return values
```
andReturnConsecutively: valueArray
```

Raising
```
andRaise: anException
```

Usage

Contexts
Subclass `SSpec.SpecContext`, adding a setup method to initialize any instance variables that the contained specs will use. Specifications are unary methods in the *specs* protocol.

Suites: Hierarchical Collections of Specs
Suites contain a collection of specs… each one an instance of a `SpecContext` subclass, each embodying a single spec method and it's, well… context.

`SomeSpecContext suite` returns a suite containing all specifications defined in `SomeSpecContext`.

Custom suites can be made:

```
suite := SpecSuite named: 'suite name'.
suite
   add: SomeSpecContext suite;
   add: SomeOtherSpecContext suite.
```

The resulting suite can be given to a runner (see blow) to be executed.

From the Workspace

`TextSpecRunner terse`
: creates a runner that outputs to Transcript, outputing '.' for passing specs, and 'X' for failing specs.

`TextSpecRunner verbose`
: creates a runner that outputs to Transcript, outputing context and specification names

Both of the above forms output a summary and information about each failure.

Also, there are versions of the about that take a stream argument which specifies where output should go.

`run: aContextClass`
: once you have a TextSpecRunner, you can have it run a context

`run: aSpecSuite`
: you can also provide a suite for it to run

Below is a sample of the output formats of both terse & verbose run modes. The text runner is typically used when a results of a complete spec run is required for reporting purposes of some kind.

Refactoring Browser Integration

sSpec is integrated with the standard Refactoring Browser in the same was as sUnit. When a bundle/package, class, protocol, or method is selected that encompasses one or more specifications, the sSpec control panel will show at the bottom of the browser code pane.

Output Formats

Terse output:

```
X.
2 specs, 1 failures
0.056 seconds
1) ExpectationNotMetError in 'sample context with failing spec should fail'
should be even
SSpec.ShouldHelper(SSpec.ShouldBase)>>failWithMessage:
```

Verbose output:

```
sample context with failing spec
   should fail (FAILED 1)
   should pass

2 specs, 1 failures
0.158 seconds
1) ExpectationNotMetError in 'sample context with failing spec should fail'
should be even
SSpec.ShouldHelper(SSpec.ShouldBase)>>failWithMessage:
```