Spock 101

MaPhi Werner

codecentric

GR8Conf EUROPE 2018

Write Tests

because you will regret it if you don’t

What is Spock?

A **test framework**

and **domain-specific language** (DSL)

for the **JVM**

supporting **data-driven testing**

and strong built-in **mocking** capabilities

Yours Truly

Name	MaPhi Werner
Company	codecentric AG
City	Stuttgart
Country	Germany
Email	maphi@codecentric.de
Twitter	maphiwe

Gradle Setup

```
apply plugin: "groovy"
apply plugin: "application"

repositories {
    mavenCentral()
}

dependencies {
    compile ‘org.codehaus.groovy:groovy-all:2.4.15'
    testCompile 'org.spockframework:spock-core:1.1-groovy-2.4'
}
```

The Simplest Test

```
import spock.lang.Specification

class SimpleStart extends Specification {

    void "One plus Two is Three"() {
        expect:
        1+2 == 3
    }

}
```

Given, When, Then

```
void “The += operator concatenates two Strings”() {

    given: “the first half of the best-known String in IT“
    String myString = "Hello"

    when: “the second half is concatenated to it"
    myString += " World"

    then: "the result is the sum of both parts"
    myString == "Hello World"
    myString.length() == 11
}
```

Setup, Cleanup

```
void "Inserting increases the DB size by one"() {
    setup: "Initialising the database"
    Database db = new Database()

    when:
    db.insert("Hello World")

    then:
    db.size() == 1

    cleanup:
    db.close()
}
```

A Failed Test ...

```
void "the two lists are equal"() {
    given: "two lists that are not really equal"
    List firstList = ["a", "b", "c"]
    List secondList = ["a", "b", "d"]

    expect: "a failure, but on purpose"
    firstList == secondList
}
```

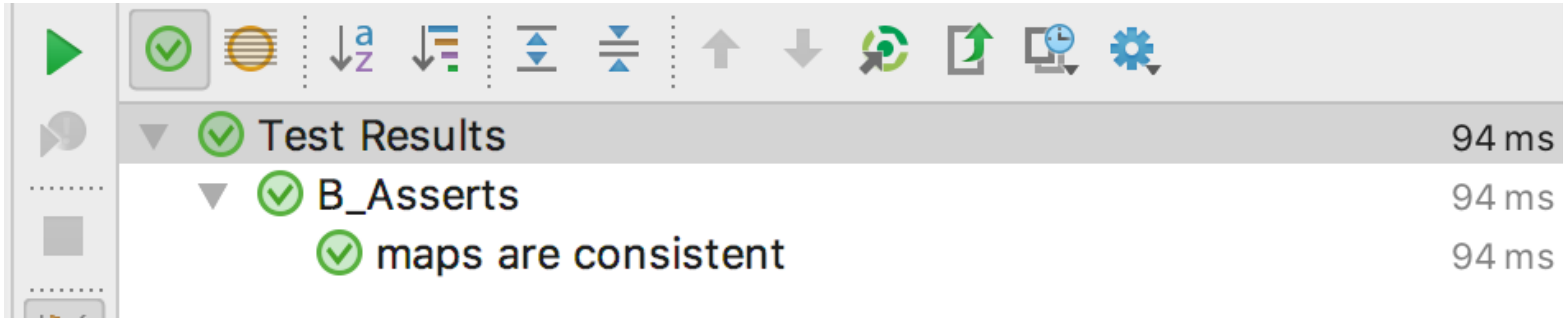
… Welcome Power Asserts

```
Condition not satisfied:

firstList == secondList
|          |  |
[a, b, c]  |  [a, b, d]
           false
```

codecentric

Red or Green?

```
void “maps are consistent"() {
    given:
    Map myMap = [a:1, b:2]

    expect:
    for (i in myMap.values()) {
        myMap.i == 1
    }
    myMap.with {
        a == 2
        b == 2
    }
}
```



```
Test Results                    94 ms
  B_Asserts                     94 ms
    maps are consistent         94 ms
```

Beware Closures!

```
void "checks in closures don't auto-assert"() {
    given:
    Map myMap = [a:1, b:2]

    expect:
    for (i in myMap.values()) { // for has no return value
        assert i == 1           // assert must be explicit
    }
    myMap.with {
        assert a == 2
        assert b == 2           // Closure return value is equal to last expression
    }
}
```

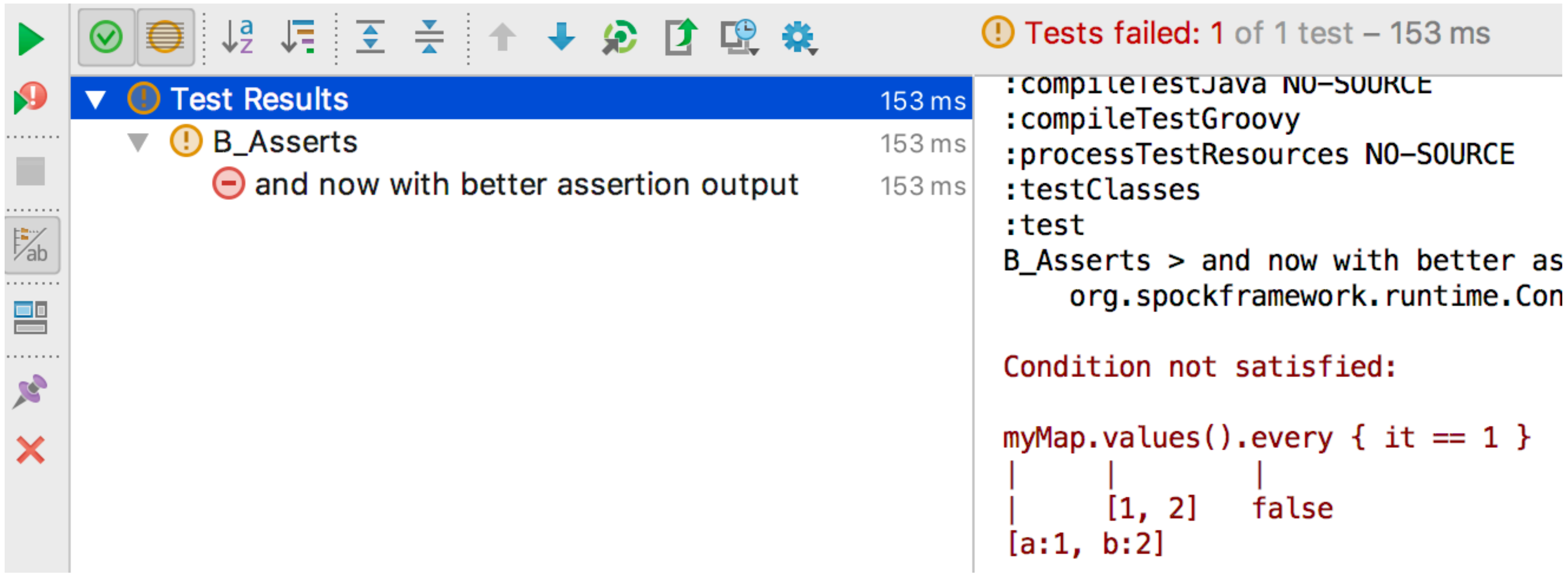
Spock with() instead of Groovy with

```
void “Spock with() instead of Groovy with"() {
    given:
    Map myMap = [a:1, b:2]

    expect:
    for (i in myMap.keySet()) { // for has no return value
        assert myMap.i == 1     // assert must be explicit
    }
    with(myMap) {
        a == 2
        b == 2
    }
}
```

codecentric

Map/List operations instead of *for* loop

```
void "Map operations and Spock with() are even better"() {
    given:
    Map myMap = [a:1, b:2]

    expect:
    myMap.values().every { it == 1 }

    with(myMap) {
        a == 2
        b == 2
    }
}
```



Tests failed: 1 of 1 test – 153 ms

```
Test Results                                   153 ms
  B_Asserts                                    153 ms
    and now with better assertion output       153 ms
```

```
:compileTestJava NO-SOURCE
:compileTestGroovy
:processTestResources NO-SOURCE
:testClasses
:test
B_Asserts > and now with better as
    org.spockframework.runtime.Con

Condition not satisfied:

myMap.values().every { it == 1 }
|     |        |
|     [1, 2]   false
[a:1, b:2]
```

codecentric

Exception Handling

```
private static void thisBreaks() {
    throw new IllegalArgumentException(“The important message")
}

void "an exception is thrown"() {
    when:
    thisBreaks()

    then:
    def myException = thrown(IllegalArgumentException)

    and: “there is something more”
    myException.message == "The important message”
}
```

Fields

```
class TwoCoupledTests extends
Specification {

    int counter = 1

    void "counter is two"() {
        when: "the counter is increased"
        ++counter

        then: "it should be two"
        counter == 2
    }

    void "counter is three"() {
        when: "the counter is increased"
        ++counter

        then: "it should be three"
        counter == 3
    }
}
```

Fields

```
class TwoCoupledTests extends
Specification {
```

Tests failed: 1, passed: 1 of 2 tests

```
Test Results                 145 ms
  C_SpecificationSetup       145 ms
    counter is two            74 ms
    counter is three          71 ms
```

```
Condition not satisfied:

counter == 3
|       |
2       false
```

```
        then: "it should be two"
        counter == 2
    }
```

```
                                          then: "it should be three"
                                          counter == 3
                                      }
                                  }
```

Fields

```
class TwoCoupledTests extends
Specification {


    @Shared int counter = 1

    void "counter is two"() {
        when: "the counter is increased”
        ++counter

        then: "it should be two"
        counter == 2
    }

    void "counter is three"() {
        when: "the counter is increased"
        ++counter

        then: "it should be three"
        counter == 3
    }
}
```

Protecting against Test Leakage

```
void "counter is two with guard"() {
    expect:
    counter == 1

    when: "the counter is increased"
    ++counter

    then: "it should be two"
    counter == 2
}
```

old

```
void "increment increases value by one"() {
    given:
    int i = 0

    when:
    ++i

    then:
    i == old(i) + 1
}
```

Fixture Methods

```
void setupSpec() {} // Executed once before all tests

void setup() {}  // Executed before every test

void cleanup() {} // Executed after every test

void cleanupSpec() {}  // Executed once after all tests
```

Inheritance for Specifications

```
abstract class ParentSpecification extends Specification {
    void setupSpec() { println "Parent Setup Spec" }
    void setup () { println "Parent Setup" }
    void cleanup() { println "Parent Cleanup" }
    void cleanupSpec() { println "Parent Cleanup Spec" }
}

class ChildSpecification extends ParentSpecification {
    void setupSpec() { println "Child Setup Spec" }
    void setup () { println "Child Setup" }
    void cleanup() { println "Child Cleanup" }
    void cleanupSpec() { println "Child Cleanup Spec" }
}
```

```
Parent Setup Spec
    Child Setup Spec
       Parent Setup
          Child Setup


          Child Cleanup
       Parent Cleanup
   Child Cleanup Spec
Parent Cleanup Spec
```

AutoCleanup

```
class SomeResource {
    void close() {}
}

@AutoCleanup // Calls close() after each test
SomeResource myResource = new SomeResource()
```

codecentric

AutoCleanup with Parameter

```
class OtherResource {
    void cleanup() {}
}

@AutoCleanup("cleanup") // Calls cleanup() after each test
@Shared
OtherResource otherResource = new OtherResource()
```

Where

```
void "number is even"(int dataElements) {
    expect:
    dataElements % 2 == 0

    where:
    dataElements << [2, 4, 6, 8]
}
```

Where: Multiple Data Pipes

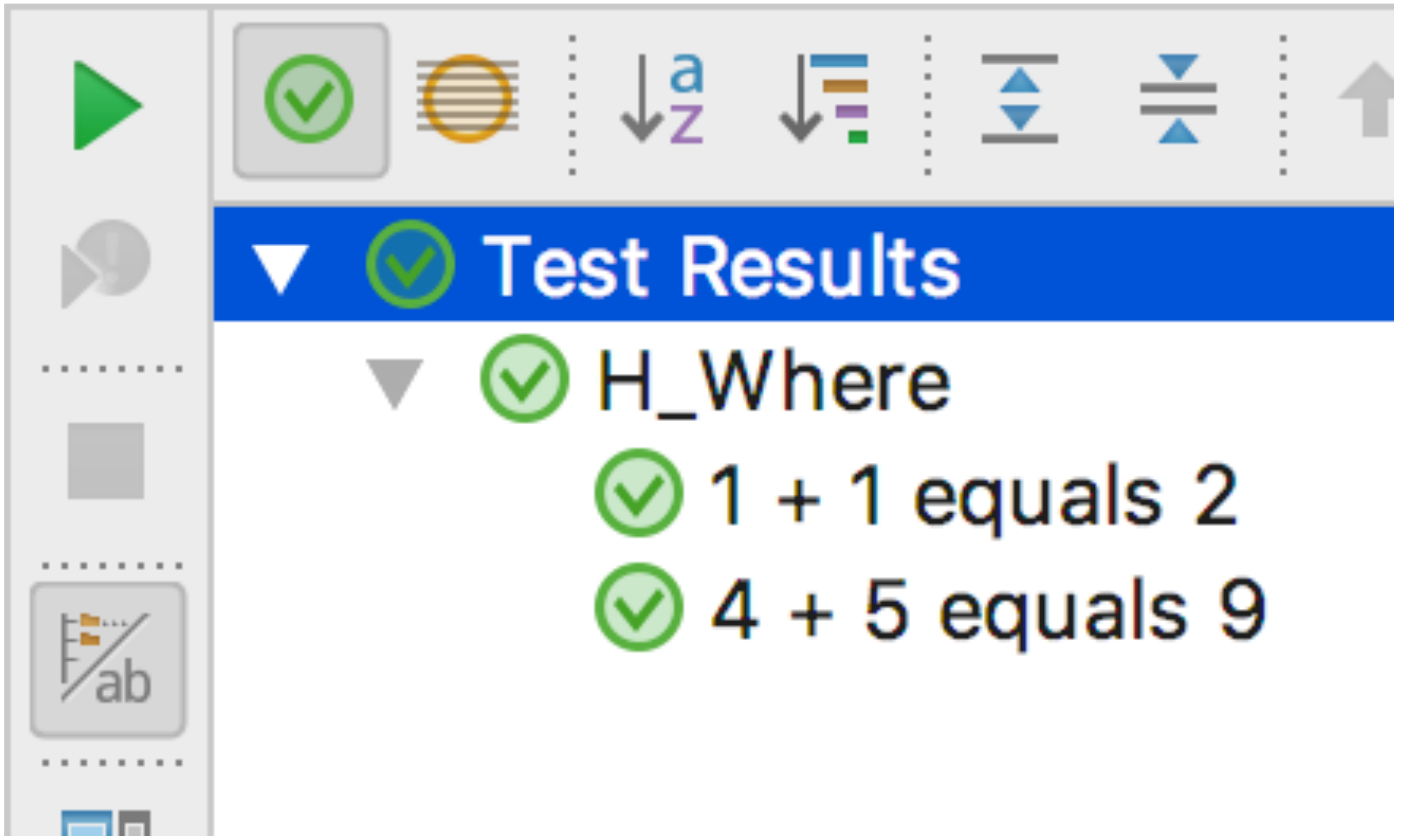
```
void "String is of correct length"(String myString, int myLength) {
    expect:
    myString.length() == myLength

    where:
    myString << ["Hello", "World"]
    myLength << stringLength()
}

private static int[] stringLength() {
    return [5, 5]
}
```

Where: Data Table

```
void "Addition works as expected"(int a, int b, int c) {
    expect:
    a + b == c

    where:
    a | b | c
    1 | 1 | 2
    4 | 5 | 9
}
```

Where: @Unroll

```
@Unroll
void "#a + #b equals #c"(int a, int b, Number c) {
    expect:
    a + b == c

    where:
    a | b
    1 | 1
    4 | 5

    c = a.plus(b)
}
```



```
Test Results
  H_Where
    1 + 1 equals 2
    4 + 5 equals 9
```

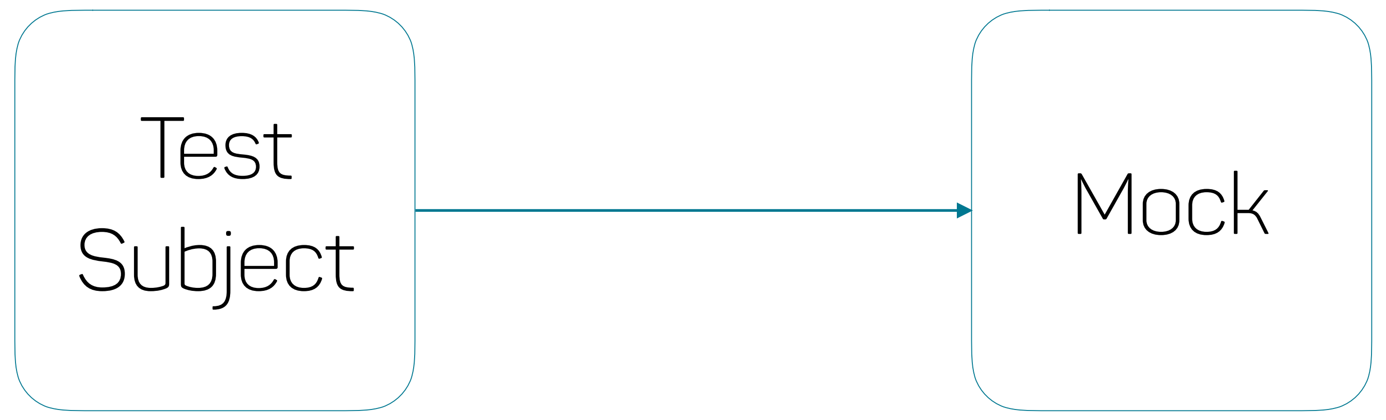
Interaction Based Testing

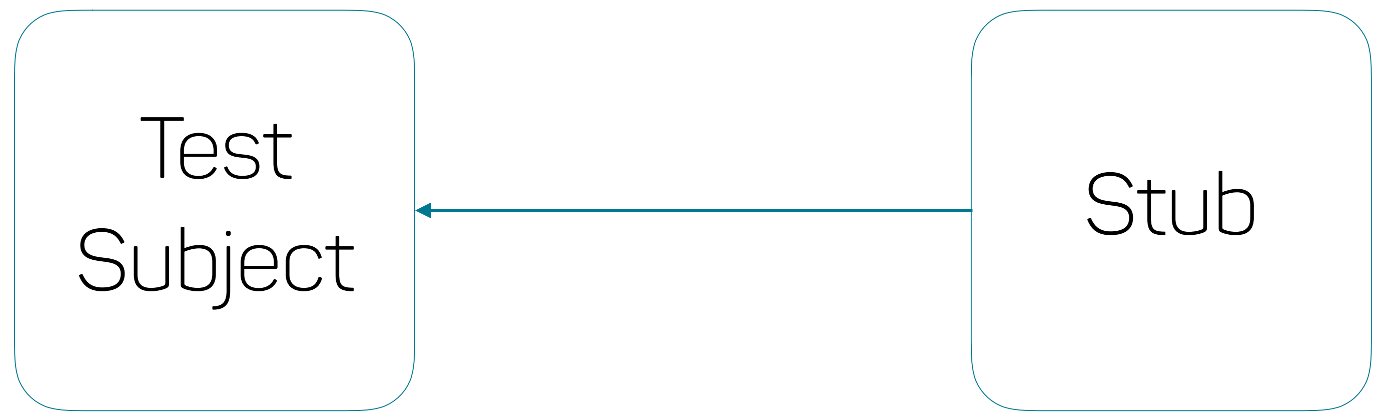
Mock

- Test stand-in for collaborator
- Tracking what calls are made **to** the collaborator

Stub

- Test stand-in for collaborator
- Providing pre-determined responses **from** the collaborator





Mocking

```
interface Data { int retrieve(String key) }
int doubleData(Data myData, String key) { return 2 * myData.retrieve(key) }


def "retrieve is called once"() {
    given:
    def myData = Mock(Data)

    when:
    doubleData(myData, "keyString")

    then:
    1 * myData.retrieve(_)
    0 * _
}
```

Syntax	Cardinality
`0 *`	Not at all
`1 *`	Exactly once
`(1..3) *`	One, two, or three times
`(1.._) *`	At least once
`(_..4) *`	No more than four times
`_ *`	Any number of times

codecentric

Mocking a Class

```
class DataClass implements Data {
    int retrieve (String key) { return 0 }
}

def "mocking the DataClass"() {
    given:
    DataClass dataClass = Mock()

    when:
    doubleData(dataClass, "keyString")

    then:
    1 * dataClass.retrieve("keyString")
}
```

```
Cannot create mock for class MocksAndStubs$DataClass. Mocking of non-interface types requires a code generation library. Please put byte-buddy-1.6.4 or cglib-nodep-3.2 or higher on the class path.
```

```
dependencies {
    testCompile 'cglib:cglib-nodep:3.2.6'
}
```

Stubbing

```
interface Data { int retrieve(String key) }
int doubleData(Data myData, String key) { return 2 * myData.retrieve(key) }

def "checkData is working as expected"() {
    given:
    Data myData = Mock()
    myData.retrieve("keyString") >> 3
    myData.retrieve("otherString") >> 5

    expect:
    6 == doubleData(myData, "keyString")
    10 == doubleData(myData, "otherString")
}
```

Syntax	Constraint
`(“hello”)`	Equal to “hello”
`(!”hello”)`	Any argument unequal to “hello”
`()`	Empty argument
`(_)`	Any single argument
`(* _)`	Any argument list
`(_ as String)`	Any String argument
`( { it.size > 3 } )`	Any argument matching the predicate

codecentric

More Ways to Stub

```
5 * myData.retrieve(_) >>> [1, 2] \
                       >> { String key -> key.length()} \
                       >> { args -> args.size() } \
                       >> { throw new InternalError() }
```

Built-In Extensions

Annotation	Effect
`@Ignore`	Skip test
`@IgnoreRest`	Skip all other tests
`@IgnoreIf(<Predicate>)`	Skip if predicate is true
`@Requires(<Predicate>)`	Run if predicate is true
`@PendingFeature`	Mark test as skipped, report error if succeeds
`@Stepwise`	Force execution order
`@Timeout(<duration>)`	Fail test after duration
`@ConfineMetaClassChanges(<ClassList>)`	Reset meta classes in cleanup
`@RestoreSystemProperties`	Reset system properties in cleanup

Doc Annotation	Effect
`@Title`	Natural language name of specification
`@Narrative`	Natural language description of specification
`@See`	Link to external content
`@Issue`	Link to issue tracking system
`@Subject`	Subject of test

Questions?

maphi@codecentric.de

https://twitter.com/maphiwe

https://www.linkedin.com/in/maphiwerner/