



Spock Framework

Czym jest Spock?

- Framework do testów jednostkowych
- Pierwsza stabilna wersja w 2015
- Alternatywa dla JUnit + Mockito – posiada wbudowane narzędzia do mockowania
- Testy pisane w Groovy'm
- Korzysta z JUnitowego runnera

<http://spockframework.org/>

<http://groovy-lang.org/>



Apache Groovy

- Java bez średników?☺
- Kompatybilny z Javą
- Kompilowany do bytecode, działa na JVM
- Język skryptowy
- Utrzymywany przez Apache
- Używany m.in. w Gradle, Grails
- Działa ze Springiem

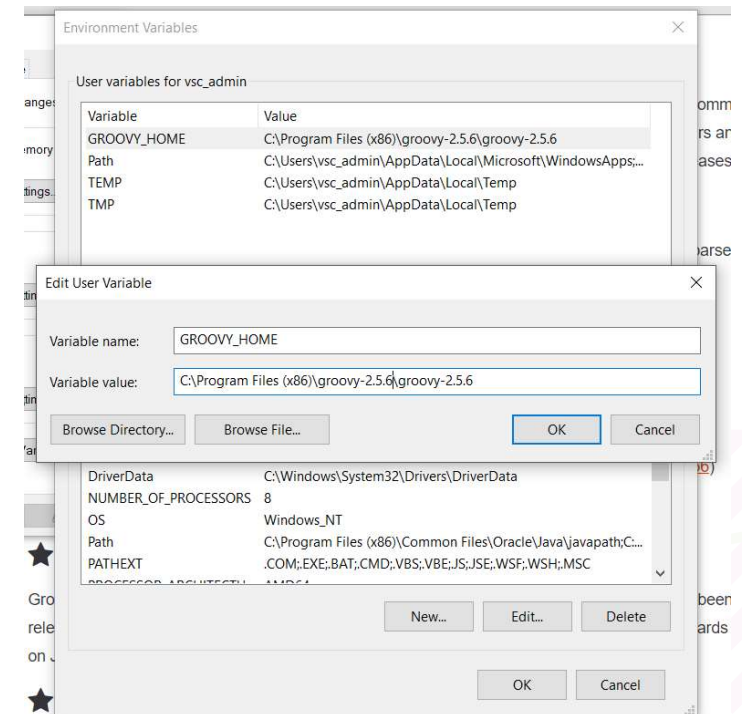
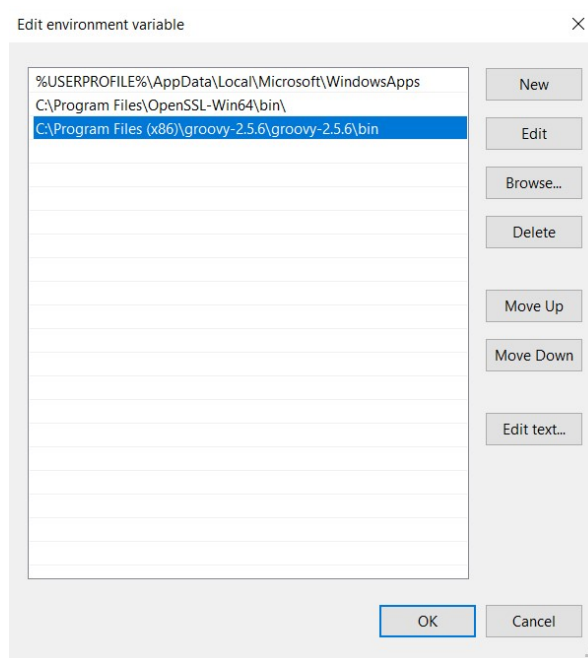
```
deathStar.causeDamage(damage);  
throw;  
deathStar.getHealthPoints();
```

Semicolon is unnecessary more... (Ctrl+F1)



Konfiguracja

- Pobierz Groovy SDK 2.5 z <https://groovy.apache.org/download.html>
- Dodaj zmienną GROOVY_HOME
- Edytuj Path



Konfiguracja cd.

```
dependencies {  
    compile group: 'org.codehaus.groovy', name: 'groovy-all', version: '2.5.0', ext: 'pom'  
    testCompile group: 'org.spockframework', name: 'spock-core', version: '1.3-groovy-2.5'  
}
```



 **Spock Framework Enhancements** v0.11

Framework integration

 Mar 07, 2016  66.2K 

This plugin provides support for the Spock specification framework.

- Adds syntax highlighting to Spock labels
- Provides inspections for label ordering
- A Spock spec template and a collection of live templates for feature methods contributed by @fpape
- Code generation assistance in specs for setup, cleanup, and adding a test method contributed by @fpape

Change Notes

Fixes for 16 EAP compatibility.

Vendor: Matt Cholick

Size: 93.2 K

Po co używać Spocka?

- ✓ Live templates
- ✓ Czytelny output
- ✓ Testy parametryzowane – tabela zamiast @Parameters
- ✓ Składnia jest bardziej elastyczna (np. tworzenie kolekcji)
- ✓ Mniej kodu
- ✓ Lepsza czytelność
- ✓ Wbudowane narzędzie do mockowania

```
//Groovy way
def groovyList = ['item1', 'item2']
groovyList << 'item3'

//Java way
List<String> javaList = new ArrayList<>()
javaList.add("item1")
javaList.add("item2")
javaList.add("item3")
```

Czytelny output

```
Condition not satisfied:
```

```
c == calculator.add(a, b)
```

```
| | | | | |
```

```
16| | | 15 3 12
```

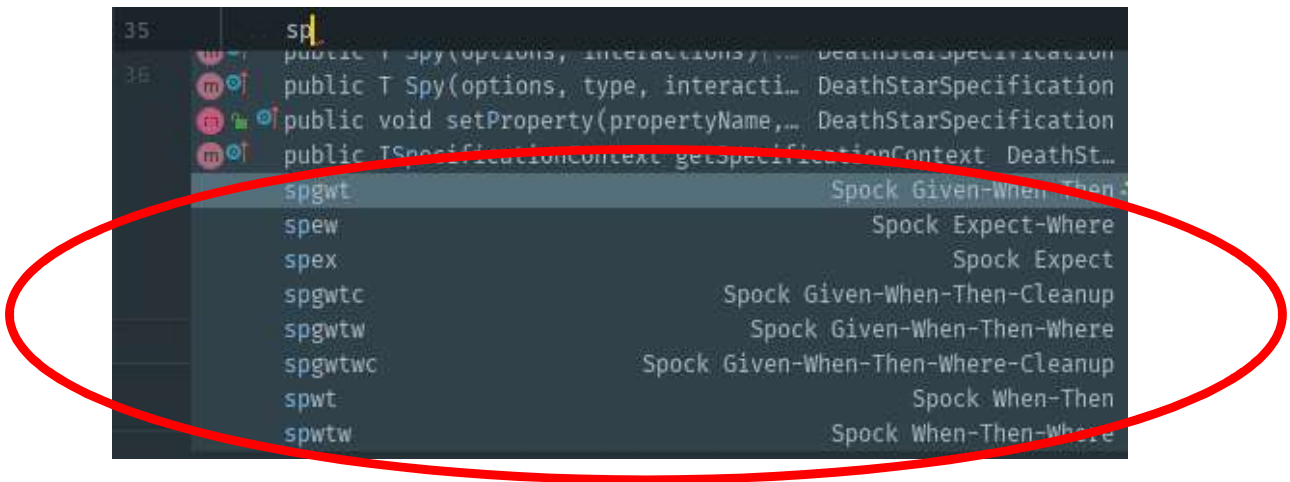
```
| <calculator.Calculator@41d477ed>
```

```
false
```

```
<Click to see difference>
```

```
--- at calculator.CalculatorSpec.should add #a to #b and give #c(CalculatorSpec.groovy:25)
```

Live templates



```
35 sp
36 public T Spy(options, interactions, ... DeathStarSpecification
public void setProperty(propertyName, ... DeathStarSpecification
public ISpecificationContext getSpecificationContext() DeathStar...
spgwt Spock Given-When-Then
spew Spock Expect-Where
spex Spock Expect
spgwtc Spock Given-When-Then-Cleanup
spgwtw Spock Given-When-Then-Where
spgwtwc Spock Given-When-Then-Where-Cleanup
spwt Spock When-Then
spwtw Spock When-Then-Where
```


Live template: given-when-then

spgwt + TAB

```
def T() {  
  given:  
  
  when:  
    // TODO implement when  
  then:  
    // TODO implement then  
}
```

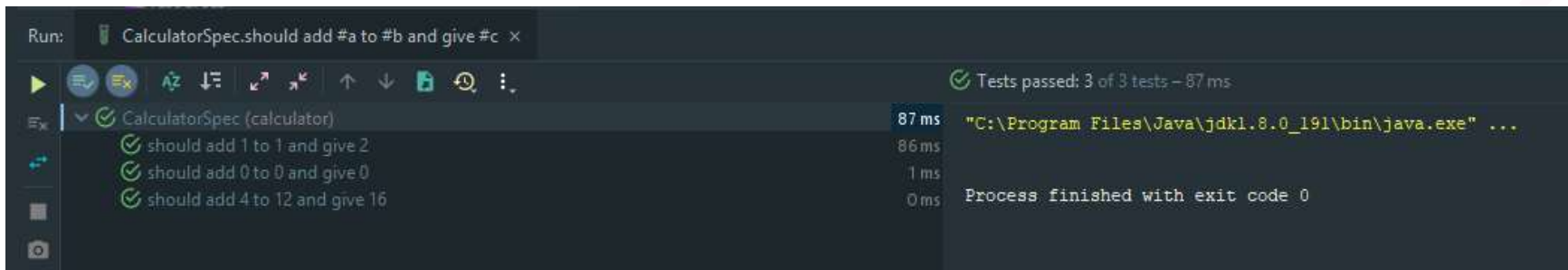
Testy parametryzowane

- Blok expect zawiera warunek
- Blok where zawiera dane
- Test jest uruchamiany dla każdego zestawu danych

```
def "should add numbers correctly"() {  
  expect:  
    c == calculator.add(a, b)  
  where:  
    a | b || c  
    1 | 1 || 2  
    0 | 0 || 0  
    4 | 12 || 16  
}
```

Testy parametryzowane cd.

```
@Unroll
def "should add #a to #b and give #c"() {
    expect:
    c == calculator.add(a, b)
    where:
    a | b || c
    1 | 1 || 2
    0 | 0 || 0
    4 | 12 || 16
}
```



The screenshot shows an IDE's test runner window. The title bar reads "Run: CalculatorSpec.should add #a to #b and give #c x". The test results are as follows:

Test Name	Duration	Status
CalculatorSpec (calculator)	87 ms	Passed
should add 1 to 1 and give 2	86 ms	Passed
should add 0 to 0 and give 0	1 ms	Passed
should add 4 to 12 and give 16	0 ms	Passed

Summary: Tests passed: 3 of 3 tests - 87 ms

Output: "C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" ...

Process finished with exit code 0

Składnia

```
def "should explode when too much damage is caused"() {  
  given:  
    deathStar = new DeathStar( name: "Exploding death star", healthPoints: 500)  
  when:  
    deathStar.causeDamage( damage: 600)  
  then:  
    thrown(ExplosionException)  
}
```

```
def 'list should be filled with products'() {  
  given:  
    Product apple = new Product( name: 'apple', price: 1.5)  
    Product carrot = new Product( name: 'carrot', price: 1.2)  
    def list = [apple, carrot]  
  when:  
    list << new Product( name: 'potato', price: 0.2)  
  then:  
    list.size == 3  
}
```

Mock vs Stub vs Spy

Mock obsługuje niespodziewane wywołania metod i zwraca domyślną wartość typu (np. *false* gdy metoda zwraca boolean)

	Oferuje prawdziwą implementację?	Zmiana zachowania metod (operator >>)	Sprawdzanie wywołań (badanie interakcji)
Stub	Nie	Tak	Nie
Mock	Nie	Tak	Tak
Spy	Tak	Tak	Tak

Stub w Spocku

1. Przy tworzeniu podajemy interfejs
2. Dwa warianty tworzenia stuba

```
reader = Stub(TextFileReader) //Groovy style  
  
TextFileReader reader2 = Stub() //Java style
```

3. Wywołanie metody + operator „>>” + zwracany obiekt

```
reader.readFile(_ as String) >> "Very serious text file content"
```

Mock w Spocku

- Dwa warianty tworzenia Mocka – również podajemy interfejs

```
reader = Mock(TextFileReader) //Groovy style  
  
TextFileReader reader2 = Mock() //Java style
```

- Mock

```
3 * reader.connect(_)
```

- Mock + Stub w jednej linii

```
1 * reader.readFile(_ as String) >> "Even more serious text file content"
```

Argument constraints

Odpowiednik matcherów

```
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(endsWith("lo"))  // any non-null argument that is-a String
1 * subscriber.receive({ it.size() > 3 && it.contains('a') })
// an argument that satisfies the given predicate, meaning that
// code argument constraints need to return true or false
// depending on whether they match or not
// (here: message length is greater than 3 and contains the character a)
```

Źródło:

http://spockframework.org/spock/docs/1.3/interaction_based_testing.html#_argument_constraints

Spy w Spocku

- Spy zachowuje się jak obiekt, ale można go modyfikować tak jak Mock
- Dwa warianty tworzenia Spy'a – podajemy klasę (nie interfejs)

```
def reader = Spy(TextFileReaderImpl) //Groovy style  
  
TextFileReaderImpl reader2 = Spy() //Java style
```

- Interakcja

```
1 * reader.readFile(_ as String)
```

Spy w Spocku

- Spy zachowuje się jak obiekt, ale można go modyfikować tak jak Mock
- Dwa warianty tworzenia Spy'a – podajemy klasę (nie interfejs)

```
def reader = Spy(TextFileReaderImpl) //Groovy style  
  
TextFileReaderImpl reader2 = Spy() //Java style
```

- Interakcja

```
1 * reader.readFile(_ as String)
```

Linki

- <https://github.com/michalgorecki/Spock-Exercise-1> - repozytorium do pobrania ćwiczenia
- <http://spockframework.org/>
- <https://groovy.apache.org/download.html> - do pobrania Groovy SDK
- <https://www.baeldung.com/groovy-spock>

<https://www.facebook.com/events/2375803885996062/>



Miejsce() Łódź

Data() 13 kwietnia 2019

- > Czy sam mógłbyś wymyślić GIT?
- > Types in Frontend World
- > Spock vs JUnit 5
- > Serverless design w praktyce
- > Augmented Reality

_experience digital transition!



TRANSITION
TECHNOLOGIES



TRANSITION
TECHNOLOGIES

Pytania