SPOCK FRAMEWORK

TESTOWANIE JAVY

Spock - a po co to komu?

Autorzy Spock'a przedstawiają go jako język, który swoim pięknym i wyrazistym kodem ma wyróżniać się z tłumu obecnych technologii.

Spock jest pochodną JUnita, Rspec'a, Mockito oraz kilku innych

Jest kompatybilny z wieloma środowiskami

Możliwości

- Testowanie Javy
- Testowanie Groovy

- Testy jednostkowe
- Mockowanie
- Fakowanie

Jak zacząć korzystać?

Wymagane:

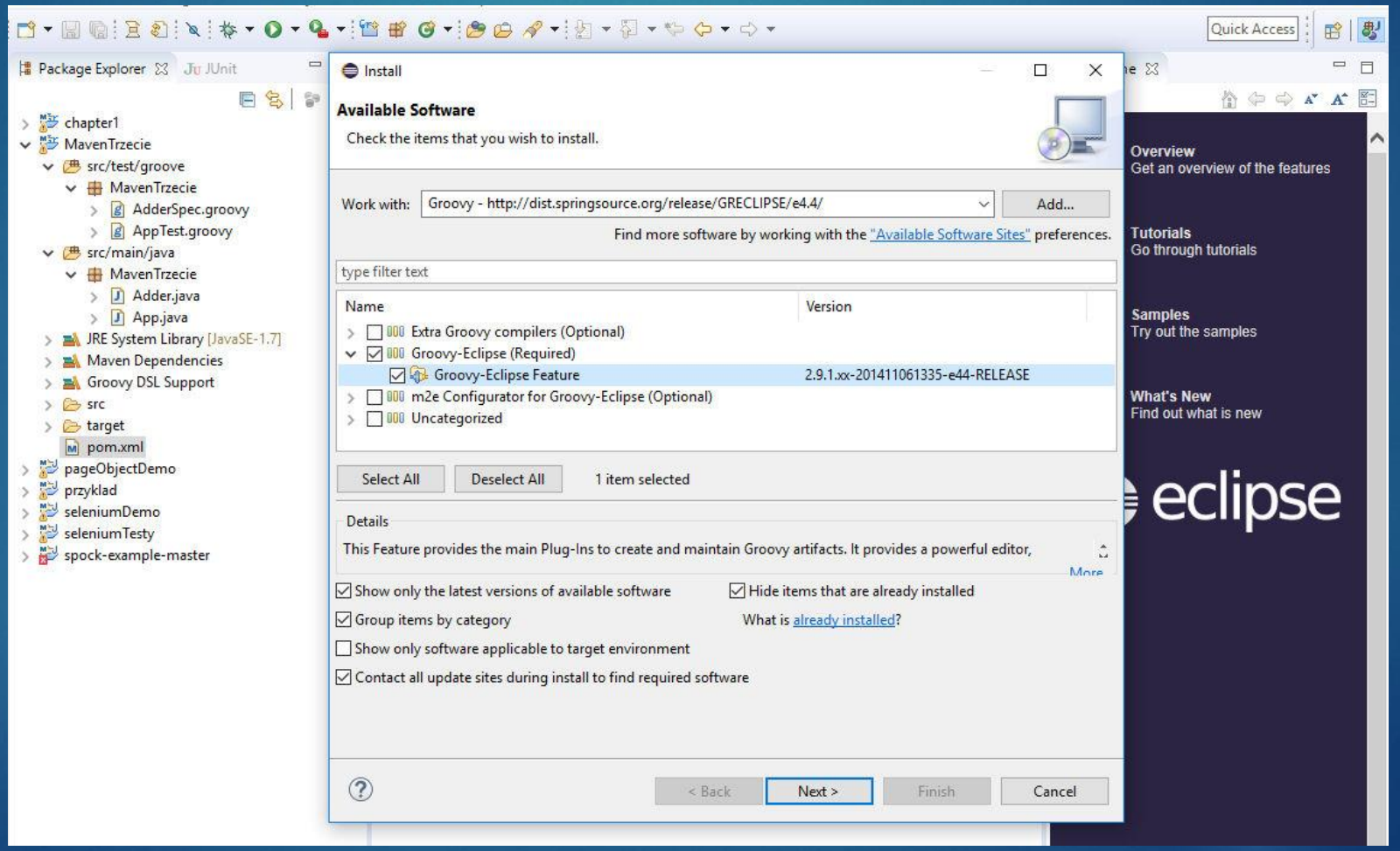
- Java Development Kit 6 +
- Groovy 2 +

Środowisko:

- Maven 2+ / Gradlew / Ant 1.7 +
- Eclipse / IDEA

ECLIPSE

Wymagana jest wtyczka Groovy Eclipse



Groovy – kolejny język?

- Pochodna JAVY
- Prosty w nauce
- Integracja z JAVĄ
- Nowoczesny
- Współpracuje ze Spocki'em :-)

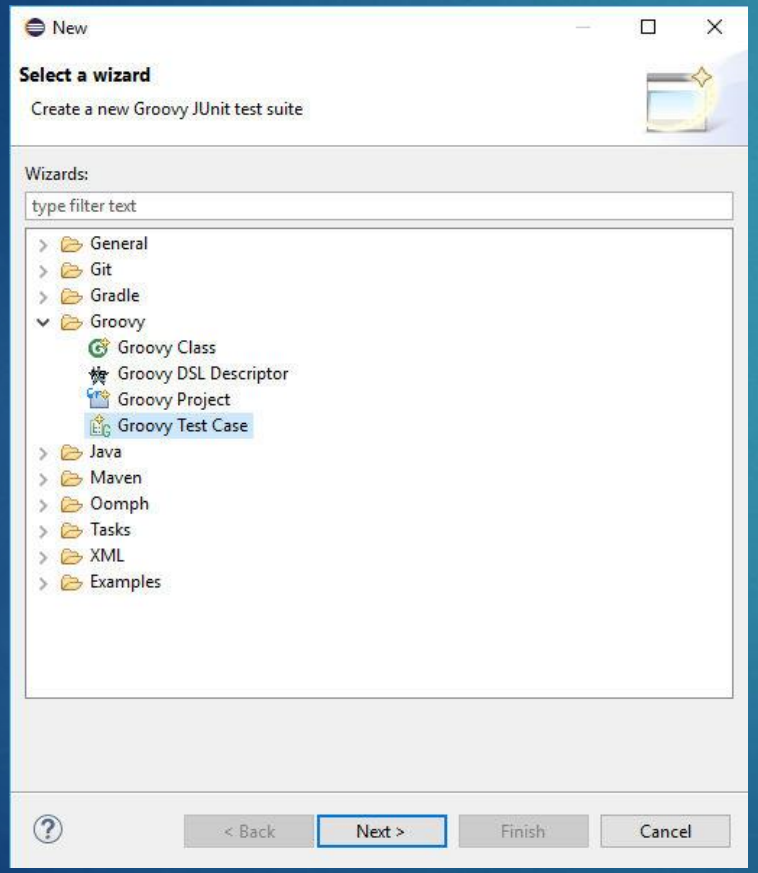
POM.XML

Import spock'a do projektu

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.spockframework</groupId>
    <artifactId>spock-core</artifactId>
    <version>1.1-groovy-2.4</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Testy

- Tworzymy nowy test Groovy



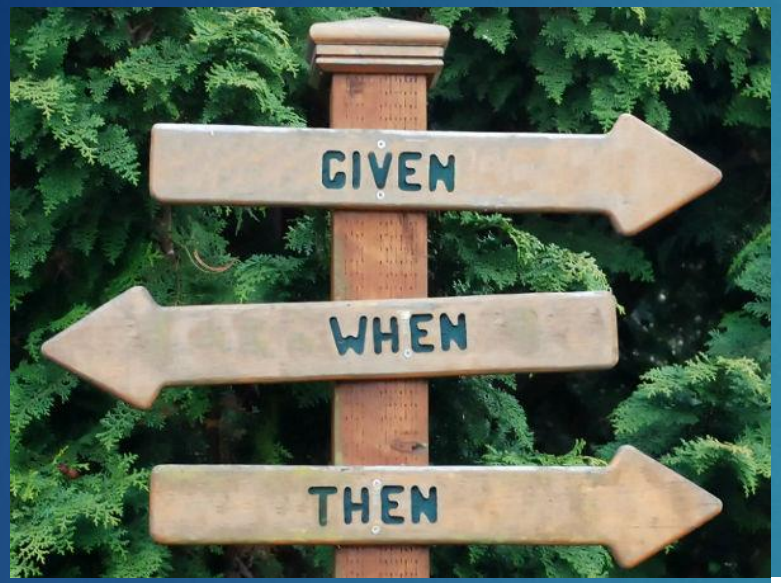
- Importujemy bibliotekę spock.lang.* oraz podpinamy klasę pod spock.lang.Specification

przyklad.groovy

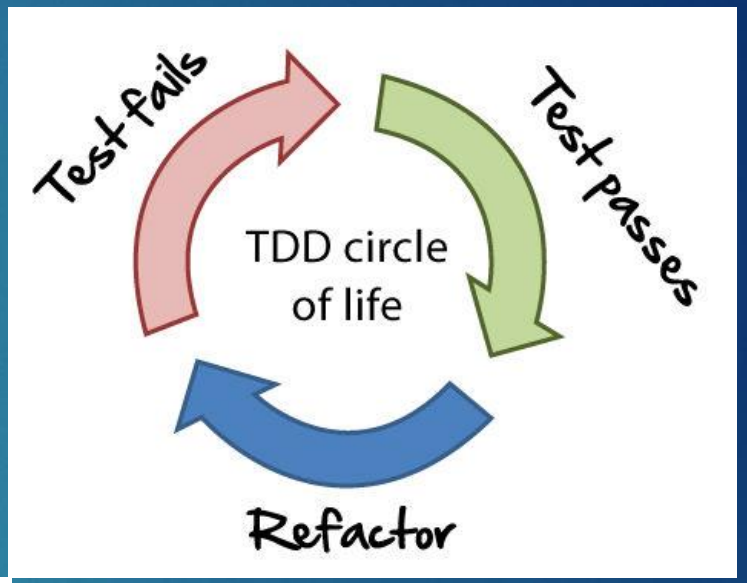
```groovy
package MavenTrzecie;

import static org.junit.Assert.*
import org.junit.Test
import spock.lang.*

class przyklad extends spock.lang.Specification{

    @Test
    public void test() {
        fail("Not yet implemented");
    }

}
```

Jak możemy testować? - dowoli

- BDD



- TDD



- MOCKS, FAKES

Kalkulator - przykład

- Testujemy klasę Calculator.java zawierającą funkcje dla kalkulatora

Calculator.java

Calculator.java

```java
package MavenTrzecie;

public class Calculator {

    public int add(int a, int b) {
        return a+b;
    }

    public int sub(int a, int b) {
        return a-b;
    }

    public int mul(int a, int b) {
        return a*b;
    }

    public int div(int a, int b) {
        return a/b;
    }

    public int pow(int a, int b) {
        int c = a;
        for(int i=1;i<b;i++)
            c = c * a;
        return c;
    }
}
```

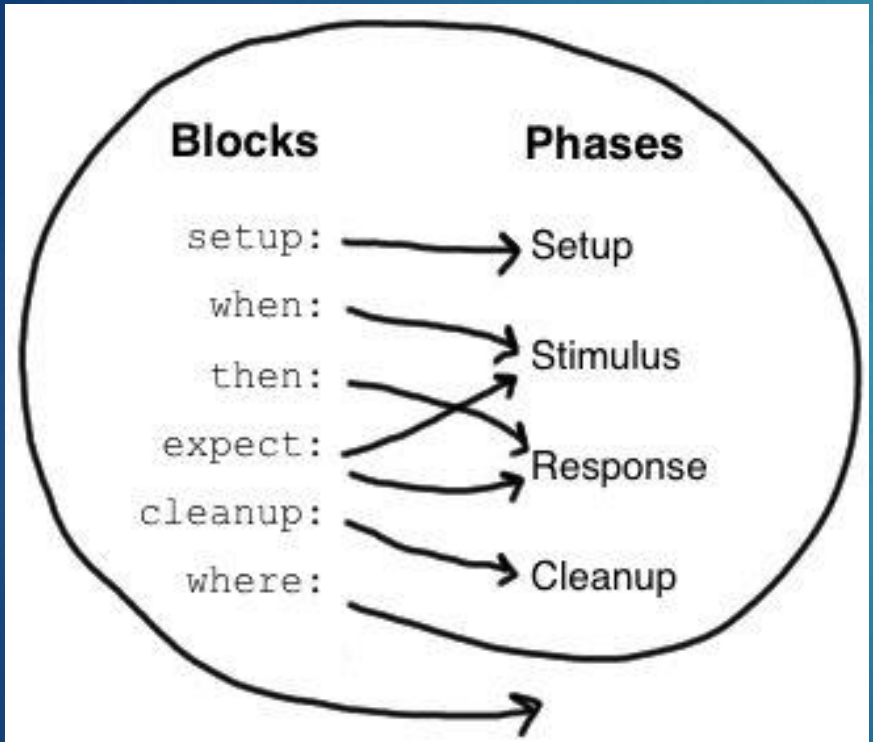
CalculatorTest.groovy

CalculatorTest.groovy

```groovy
class CalculatorTest extends spock.lang.Specification{
    def "Adding two numbers to return the sum"() {
        when: "a new Adder class is created"
        def adder = new Calculator();

        then: "1 plus 1 is 2"
        adder.add(1, 1) == 2
    }

    def "Order of numbers does not matter"() {
        when: "a new Adder class is created"
        def adder = new Calculator();

        then: "2 plus 3 is 5"
        adder.add(2, 3) == 5

        and: "3 plus 2 is also 5"
        adder.add(3, 2) == 5
    }

    def "Subtracting two numbers to return the sub"() {
        when: "a new Adder class is created"
        def adder = new Calculator();

        then: "6 minus -3 is 9"
        adder.sub(6, -3) == 9
    }
```

Schemat blokowy

- Spock wspiera schematy blokowe



Bloczki dzielimy na 6 części

- setup: wstępne ustawienia (odpowiednik @Before)
- when, then: akcja i skutek (obsługa wyjątków)
- expect: Wyłącznie dla wyników (uboższy then)
- cleanup: czyściciel (odpowiednik @After)
- where: TDD, blok dla naszych danych

BDD - przykład

- Przykład z użyciem Data Table

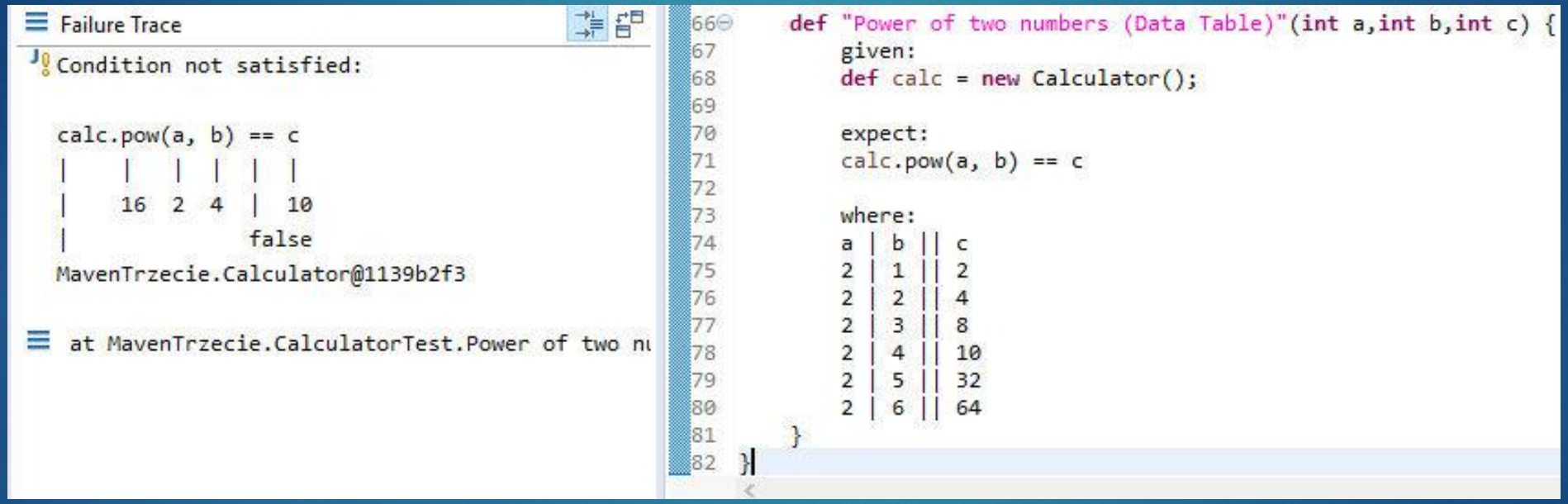
```
def "Power of two numbers (Data Table)"(int a,int b,int c) {
    setup:
    def calc = new Calculator();

    expect:
    calc.pow(a, b) == c

    where:
    a | b | c
    2 | 1 | 2
    2 | 2 | 4
    2 | 3 | 8
    2 | 4 | 16
    2 | 5 | 32
    2 | 6 | 64
}
```

Wsparcie dla pojedynczej | bądź | |

- setup: tworzymy Kalkulator
- expect: oczekujemy, że pow z a i b wyniesie c
- where: podajemy dane do testów

Raport o błędach

- Schemat prezentujący problem



MOCK'owanie

- Wymagane użycie biblioteki CGLIB

```groovy
class Mock extends spock.lang.Specification{

    UserService userService
    UserController userController

    def setup() {
        userService = Mock()
        userController = new UserController(userService: userService)
    }

    def 'create new User with arguments'(){
        given:
            String name = "Jan"
            String last = "Kowalski"

        when:
            userController.createUser(name,last)

        then:
            1 * userService.createUser(name, last) >> { String e, String n -> new User(name: e, last: n)}
    }

    def 'mock user with arguments'(){
        given:
            String name = "Jan"
            String lastname = "Kowalski"
            User user = new User()

        when:
            userController.createUser(name,lastname)

        then:
            1 * userService.createUser(name, lastname) >> user
    }
}
```

Wywoływanie metod

- 1 * - dokładnie jedno wywołanie
- 0 * - zero wywołań
- (1..5) * – od 1 do 5
- (_..4) * – max 4 wywołania
- _ * – dowolna ilość również 0

Dziękuje

Wykorzystane źródła

- http://spockframework.org
- http://spockframework.org/spock/docs/1.1
- http://presentationtier.com/doing-tdd-bdd-style
- http://www.groovy-lang.org