Schlüsselwörter

as	assert	break	case
catch	class	const	continue
def	default	do	else
enum	extends	false	finally
for	goto	if	implements
import	in	instanceof	interface
new	null	package	return
super	switch	this	throw
throws	trait	true	try
var	while		

- Im Prinzip dieselben Schlüsselwörter wie in Java
- Dürfen nicht für Variablen und Methodennamen benutzt werden
- - const, goto, strictfp und threadsafe sind Platzhalter und haben bisher keine Verwendung in Groovy

Kontextuelle Schlüsselwörter

- Werden nur in bestimmten Fällen benutzt
- Dürfen für Variablennamen benutzt werden

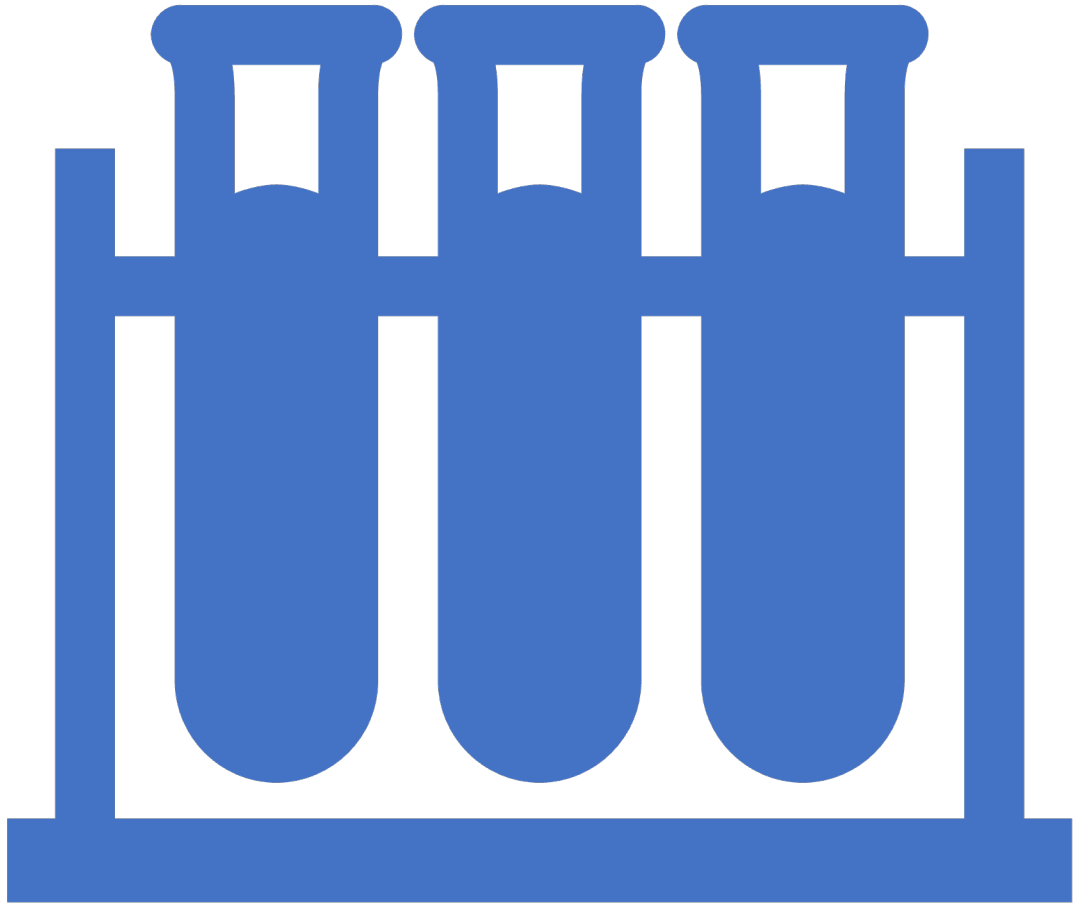
as in permitsrecord

sealed trait var yields

Schlüsselwörter doch als Variablennamen benutzen:

```
1   class A {
2       def "this"() {
3           print "Diese Methode trägt den Namen \"this\"."
4       }
5
6       def "while"(){
7           this.this()
8       }
9   }
10  //Beispielaufruf
11  A a = new A()
12  a.while()
```

Testen



Testen in Groovy

- Support für Junit 5 (und älter)
- Liefert eigenen Satz von Testmethoden, um die testgetriebene Programmierung zu erleichtern

-> Power Assertions

-> Spock

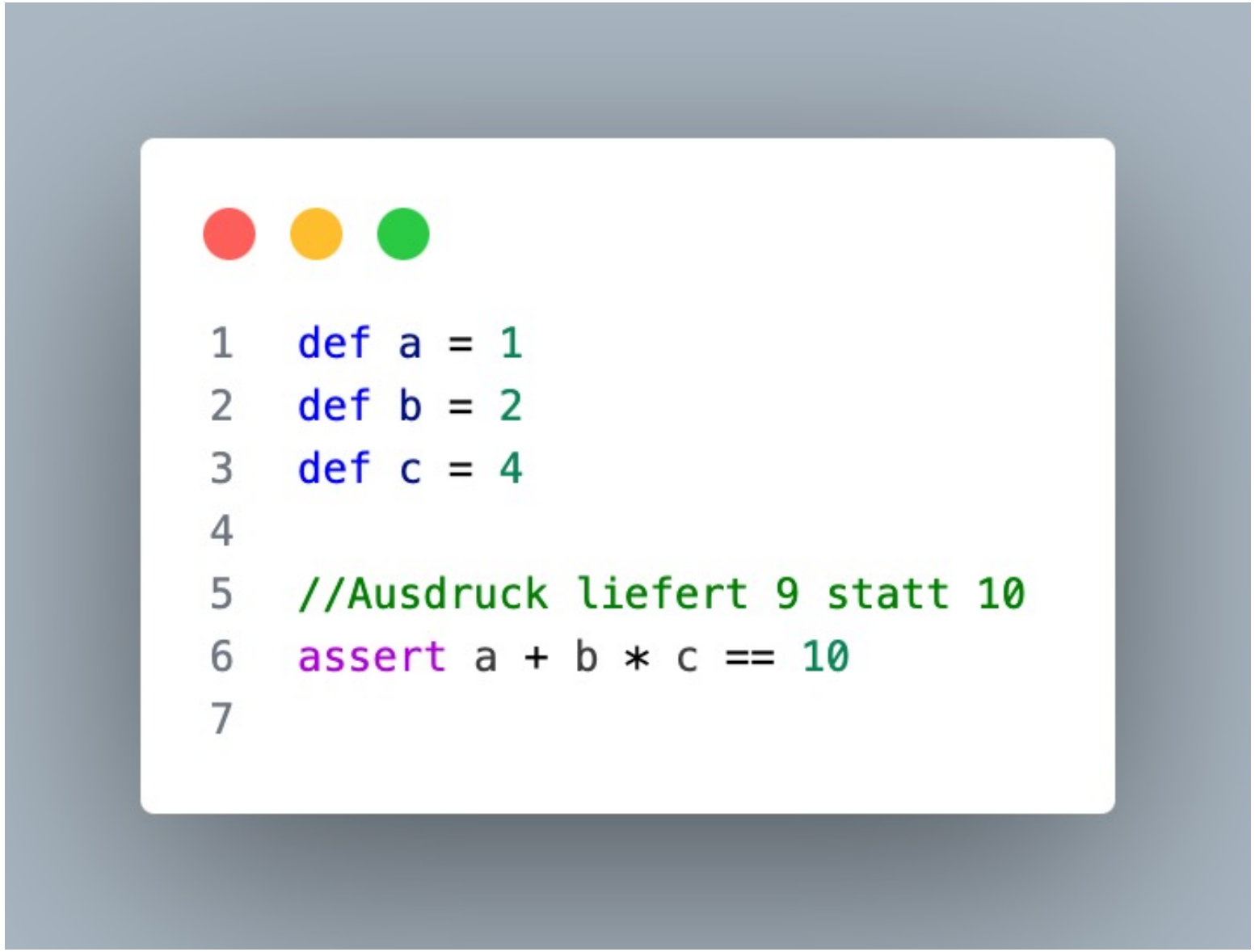
Power Assertions

- Sind im Gegensatz zu den Java Assertions automatisch aktiviert
- Erleichtern die Fehlersuche und das Debugging
- Wenn eine Assertion fehlschlägt:

➔Zeigt die Power Assertion den Ausdruck mitsamt Werten in einer übersichtlichen, mehrzeiligen Darstellung an.

➔Entwickler sieht genau, welcher Teil des Ausdrucks die Probleme verursacht

Beispiel



```
1   def a = 1
2   def b = 2
3   def c = 4
4
5   //Ausdruck liefert 9 statt 10
6   assert a + b * c == 10
7
```

Outcome:

```
Assertion failed:

assert a + b * c == 10
       | | | | | |
       1 9 2 8 4 false

    at test.run(test.groovy:5)

[Done] exited with code=1 in 0.538 seconds
```

Achtung

Es kann während der Auswertung von Power Assertion zu inkonsistenten Fehlermeldungen kommen

-> Das liegt daran, dass bei den Power Assertions nur Referenzen auf die Werte gespeichert werden

-> Werden die Werte also durch eine Methode verändert, wird der vorherige Stand in der Fehlernachricht nicht mehr angezeigt

```
1   def getLastAndRemove(list) {
2       return list.remove(list.size() - 1)
3   }
4   def list = [1, 2, 3]
5   assert getLastAndRemove(list) == 4
```

Beispiel

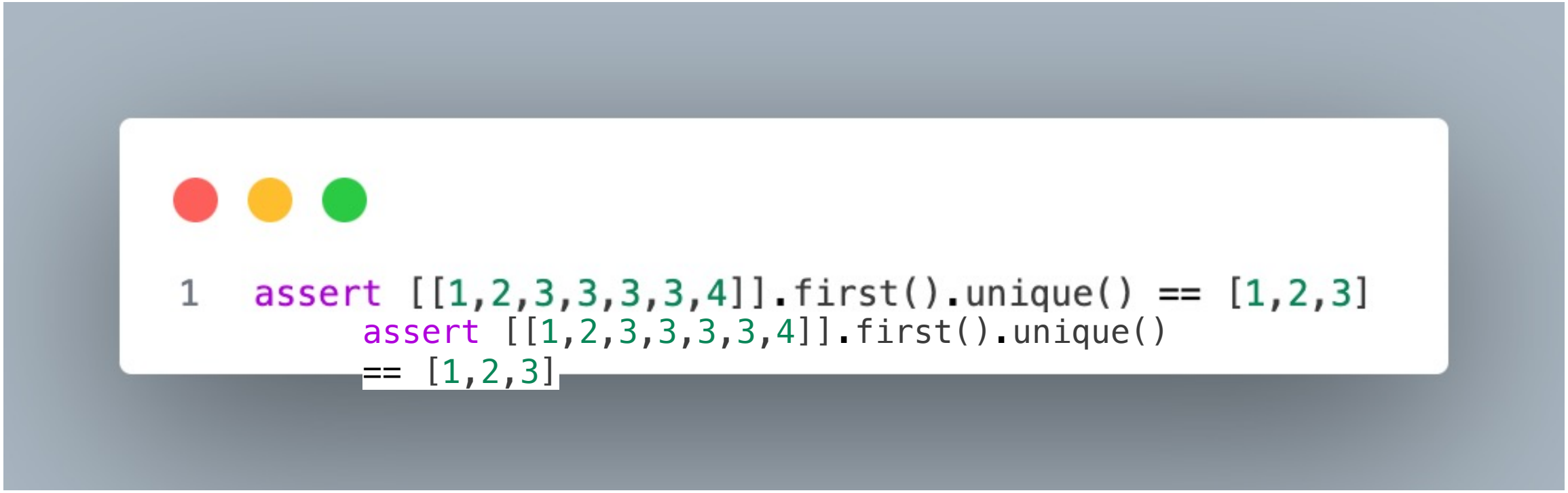
Outcome

- Es wird nicht der Stand von list angezeigt, wie er vor der Auswertung der Power Assertion war, sondern wie er danach war.

```
Assertion failed:

assert getLastAndRemove(list) == 4
       |                |      |
       3                |      false
                        [1, 2]

    at test.run(test.groovy:7)
```

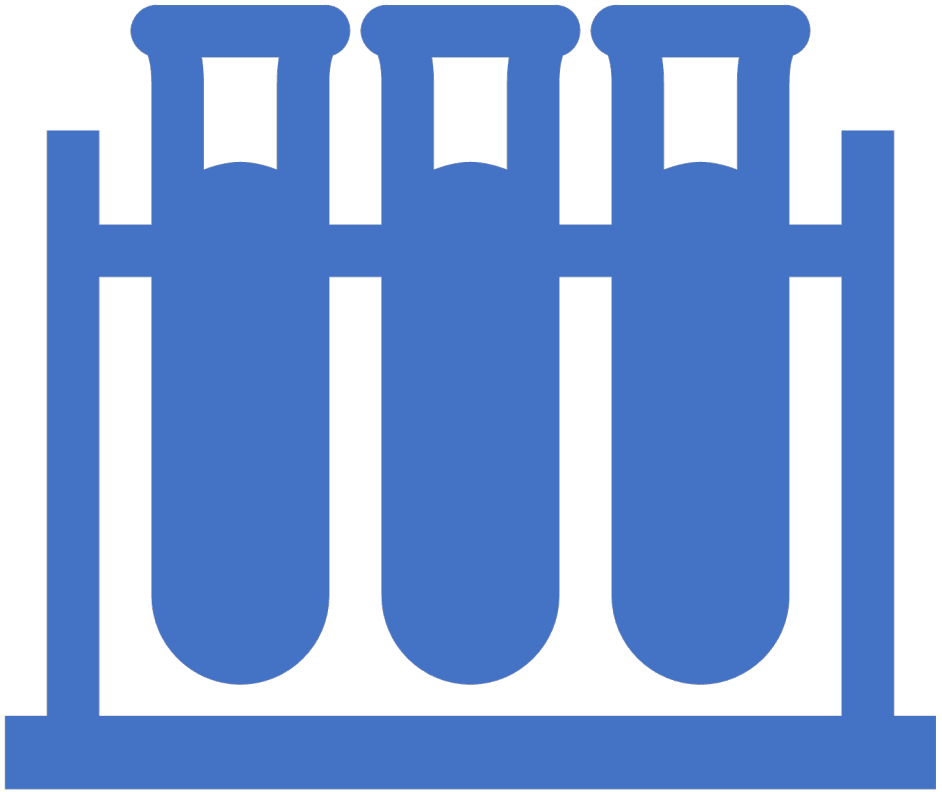


Komplexeres Beispiel

```
Assertion failed:

assert [[1,2,3,3,3,3,4]].first().unique() == [1,2,3]
       |                |       |        |
       |                |       |        false
       |                |       [1, 2, 3, 4]
       |                [1, 2, 3, 4]
       |
       at test.run(test.groovy:1)
```

Outcome:

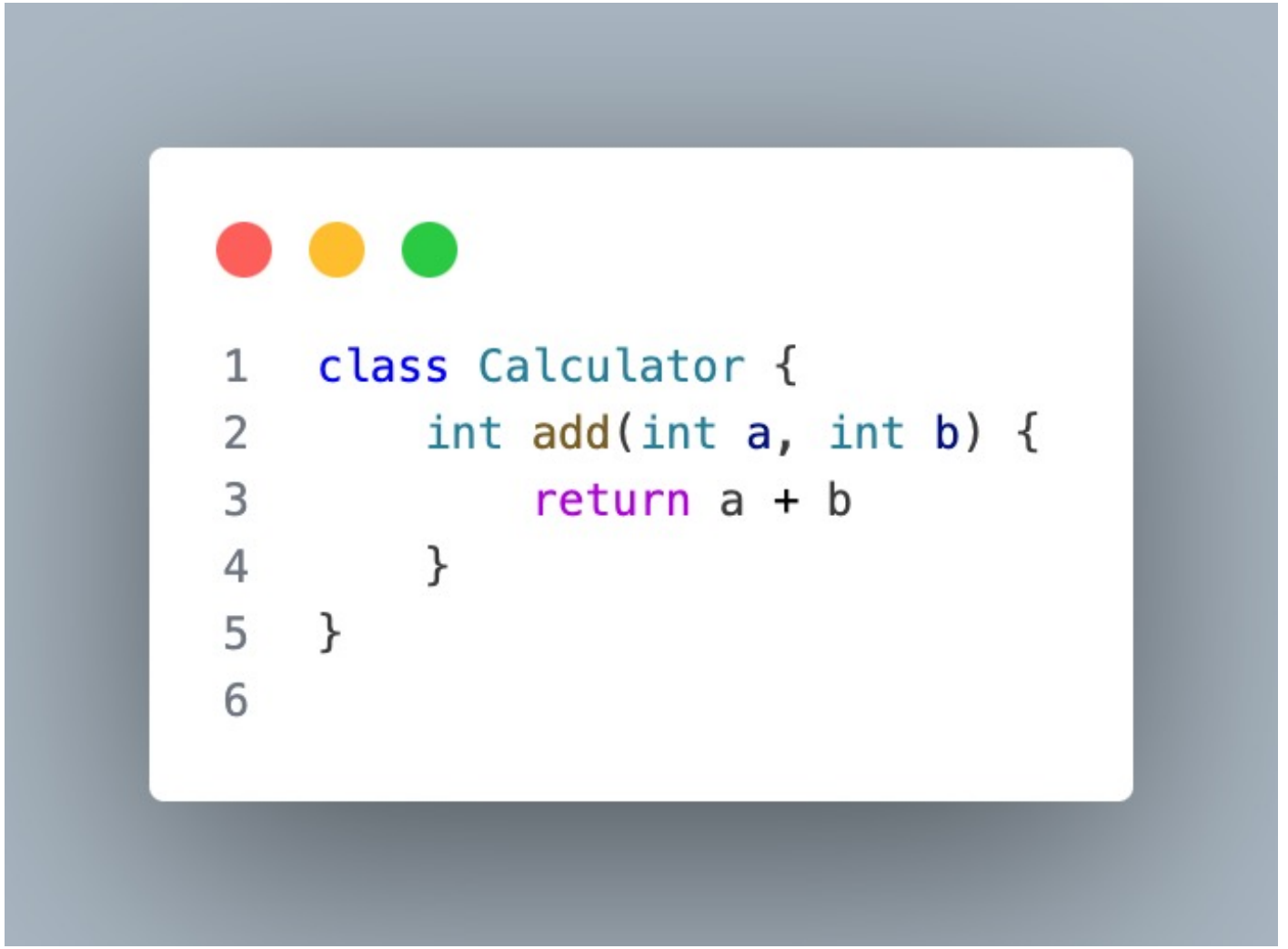


Spock

- Spezifikationsgetriebene Syntax
- Lesbare und verständlichere Tests
- Sowohl in Java als auch in Groovy verfügbar, wurde jedoch in Groovy geschrieben
- Benutzung durch den Import von spock.lang.Specification

➔Testklassen müssen von dieser Klasse erben

Beispiel: Calculator



```
1   class Calculator {
2       int add(int a, int b) {
3           return a + b
4       }
5   }
6
```

Beispiel:

1) Name der Methode ist ein String, der die Erwartungen an den Test beschreibt
2) Das Schlüsselwort „setup“ beschreibt die Voraussetzungen / Ausgangssituation für den Test (vgl.: @BeforeEach in Junit)
3) „when“: Die zu testende Methode
4) “then“: Das erwartete Verhalten

```groovy
import spock.lang.Specification

class CalculatorSpec extends Specification {
    //1
    def "addition of two numbers"() {
        //2
        setup: "a calculator"
            def calculator = new Calculator()

        //3
        when: "the numbers 2 and 3 are added"
            def result = calculator.add(2, 3)

        //4
        then: "the result is 5"
            result == 5
    }
}
```