TDD with Spock

Craig Atkinson

Agenda

1. Spock Basics
2. Introduction to Test Driven Development
3. Coding Workshop

Spock

Behavior-style test framework in Groovy with support for easy data-driven testing

Behavior-Style Testing

Test cases separated into three main sections

- given (setup)
- when (execute method under test)
- then (verify results)

Data-driven tests

Run **same test body** with **multiple sets** of test **inputs** and expected **outputs**

Writing a Spock test

Test Class Name

Test class name ends in **`Spec`** or **`Specification`** and class extends **`spock.lang.Specification`**

```
class BankAccountSpec extends Specification {

}
```

Test Case Name

Test case method names can be descriptive sentences

```
def "after depositing 10 dollars then balance should be 10 dollars"() {

}
```

Test case body

```
class BankAccountSpec extends Specification {
  def "after depositing 10 dollars then balance should be 10 dollars"() {
    given:
    BankAccount bankAccount = new BankAccount()

    when:
    bankAccount.deposit(10)

    then:
    assert bankAccount.balance == 10
  }
}
```

Setup Method

Run code before each test method

```
def setup() {
  // Setup code goes here
}
```

Cleanup Method

Run code after each test method

```
def cleanup() {
  // Cleanup code goes here
}
```

Data-Driven Testing

Where block

```
where:
input1 | input2 || output
4      | 6      || 5
12     | 18     || 15
20     | 14     || 17
```

@Unroll

Include inputs and outputs from `where:` block in tests results

```groovy
@Unroll
def 'depositing #amount should increase balance to #expectedBalance'() {
  given:
  BankAccount bankAccount = new BankAccount()

  when:
  bankAccount.deposit(amount)

  then:
  assert bankAccount.balance == expectedBalance

  where:
  amount || expectedBalance
  10     || 10
  20     || 20
}
```

Groovy Power Assert

```
assert result == expectedValue
```

```
def 'x plus y equals z'() {
    when:
    int x = 4
    int y = 5
    int z = 10

    then:
    assert x + y == z
}
```

```
Condition not satisfied:

x + y == z
| | | |  |
4 9 5 |  10
      false
```

Test Driven Development (TDD)

Use tests to help guide development

TDD in a Nutshell

1. Write tests
2. Run tests, verify failure (**red**)
3. Write only enough code to make tests pass
4. Run tests, verify they pass (**green**)
5. Cleanup code (**refactor**)

TDD Benefits

- Tests tell us when we're done coding a feature
- Avoid writing unnecessary or untested code
- Easily write testable code
- Remove any need for worrying about code coverage

Safety Net

Extensive test suite that serves as a **safety net** for code changes

- Last-minute requirement changes
- Performance improvements
- Code cleanup
- Upgrade libraries and frameworks

Executable Documentation

Thoroughly document expected behavior in tests

Workshop

`BankAccount` class

- balance
- deposit
- withdraw

Fetch Project

```
git clone https://github.com/craigatk/tdd-spock.git

cd tdd-spock
```

Project Structure

```
src/main/groovy/bank

src/test/groovy/bank
```

Gradle for Running Tests

```
gradlew test --info
```

Create First Test

`BankAccountSpec` with one test that creates a new **`BankAccount`** and verifies **`bankAccount.balance`** is 0

- Hint: 'balance' should be BigDecimal type

```
class BankAccountSpec extends Specification {
  def "bank account starting balance should be 0"() {
    given:
    BankAccount bankAccount = new BankAccount()

    when:
    BigDecimal startingBalance = bankAccount.balance

    then:
    assert startingBalance == 0
  }
}
```

Run test, verify failure

Hint: Should be a test compilation failure because BankAccount class does not exist yet

```
gradlew test --info
```

Make Test Pass

Write minimal code to make test pass

```
class BankAccount {
  BigDecimal balance = 0
}
```

Run test, verify it passes

Write Test for "deposit" Method

Takes one parameter, a BigDecimal 'amount' and should increase the balance

```
def 'deposit should increase balance'() {
  given:
  BankAccount bankAccount = new BankAccount()

  when:
  bankAccount.deposit(10)

  then:
  assert bankAccount.balance == 10
}
```

Red, Green

- Run test, verify failure
- Write the minimal code to make the test pass

```
void deposit(BigDecimal amount) {
  balance = 10
}
```

Additional Test Case

Using a **`where:`** block, expand our test method to two cases, one that deposits 10 dollars and one that deposits 20 dollars

```
@Unroll
def 'depositing #amount should increase balance to #expectedBalance'() {
  given:
  BankAccount bankAccount = new BankAccount()

  when:
  bankAccount.deposit(amount)

  then:
  assert bankAccount.balance == expectedBalance

  where:
  amount || expectedBalance
  10     || 10
  20     || 20
}
```

Red, Green

- Run tests, verify failure
- Write full deposit method

```
void deposit(BigDecimal amount) {
  balance += amount
}
```

Withdraw method

Write a test case for a **`withdraw(BigDecimal amount)`** method that reduces the balance by the given amount

Hint: In the test setup, create a new BankAccount() and deposit money into it.

```
def "withdraw should reduce balance"() {
  given:
  BankAccount bankAccount = new BankAccount()

  bankAccount.deposit(20)

  when:
  bankAccount.withdraw(15)

  then:
  assert bankAccount.balance == 5
}
```

Red, Green

- Run test, verify failure
- Write the minimal code to make the test pass

```
void withdraw(BigDecimal amount) {
  balance = 5
}
```

Additional test cases

Similar to **`deposit()`** method, use **`where:`** block to add two additional test cases that withdraw different amounts

```
@Unroll
def "withdrawing #amount should reduce balance to #expectedBalance"() {
  given:
  BankAccount bankAccount = new BankAccount()

  bankAccount.deposit(20)

  when:
  bankAccount.withdraw(amount)

  then:
  assert bankAccount.balance == expectedBalance

  where:
  amount || expectedBalance
  5      || 15
  10     || 10
  15     || 5
}
```

```
void withdraw(BigDecimal amount) {
  balance -= amount
}
```

Run tests, verify pass

We TDD'ed a BankAccount!

Recap

- Groovy testing with Spock
- Benefits of TDD
- Test-drive Groovy bank account

Q&A

Thanks for attending!

Contact info

craig.atkinson@objectpartners.com

@craigatk1