alfalab

Spock и Android

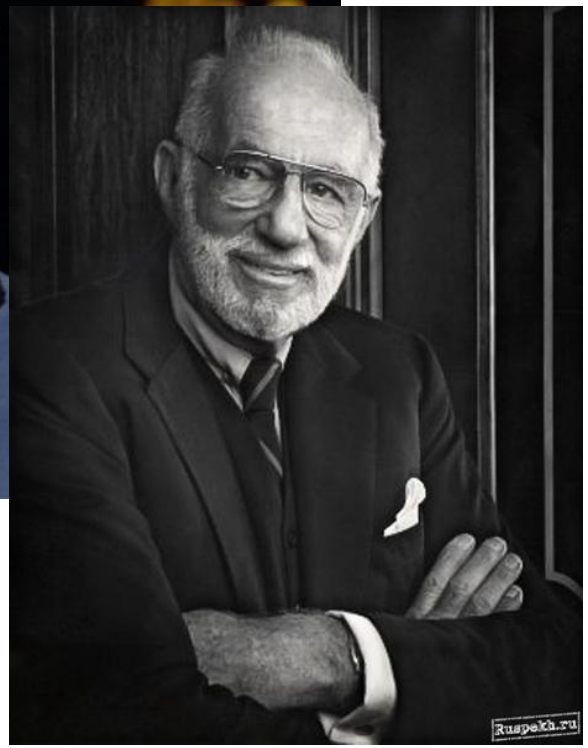
Обо мне

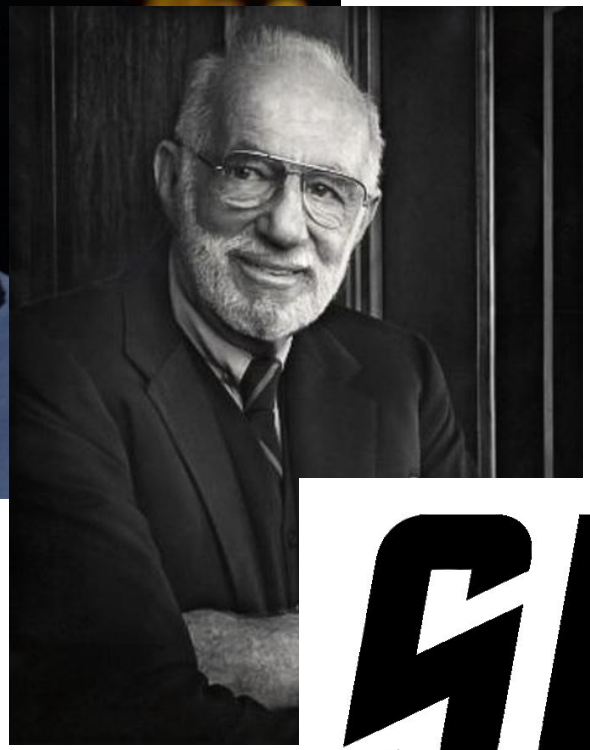
Александр Ежков

Работаю в Альфа-Банке
Пишу тесты на spock

Чего не будет

- Что такое тестирование
- Зачем нужно писать тесты
- Как писать тесты вообще, и для android в частности
- Сравнения со Spek, JUnit, TestNG и другими фреймворками
- Бенчмарков компиляции и времени прогона тестов


Ruspekh.ru



SPOCK

Spock это:

- фреймворк для тестирования
- подходит для java, groovy и kotlin приложений
- тесты пишутся на groovy
- имеет собственный junit runner
- совместим с большинством IDE, build tools и CI систем
- умеет все, что и твой любимый тестовый фреймворк, только немного лучше

Начнем с примера

```
class Account {

   var balance: BigDecimal = BigDecimal.ZERO

   fun pay(amount: BigDecimal) {
      balance = balance.subtract(amount)
   }
}
```

Первое знакомство

```
class AccountSpec extends Specification {

  def "pay from account"() {
    given:
      def account = new Account()
      account.balance = 100
    when:
      account.pay(50)
    then:
      account.balance == 50
  }
}
```

```
class Account {

  var balance: BigDecimal = BigDecimal.ZERO

  fun pay(amount: BigDecimal) {
    balance = balance.subtract(amount)
  }
}
```

Первое знакомство

```
class AccountSpec extends Specification {

  def "pay from account"() {
    given:
      def account = new Account()
      account.balance = 100
    when:
      account.pay(50)
    then:
      account.balance == 50
  }
}
```

```
class Account {

  var balance: BigDecimal = BigDecimal.ZERO

  fun pay(amount: BigDecimal) {
    balance = balance.subtract(amount)
  }
}
```

Первое знакомство

```
class AccountSpec extends Specification {

  def "pay from account"() {
    given:
      def account = new Account()
      account.balance = 100
    when:
      account.pay(50)
    then:
      account.balance == 50
  }
}
```

```
class Account {

  var balance: BigDecimal = BigDecimal.ZERO

  fun pay(amount: BigDecimal) {
    balance = balance.subtract(amount)
  }
}
```

Первое знакомство

```
class AccountSpec extends Specification {

  def "pay from account"() {
    given:
      def account = new Account()
      account.balance = 100
    when:
      account.pay(50)
    then:
      account.balance == 50
  }
}
```

```
class Account {

  var balance: BigDecimal = BigDecimal.ZERO

  fun pay(amount: BigDecimal) {
    balance = balance.subtract(amount)
  }
}
```

Первое знакомство

```
class AccountSpec extends Specification {

  def "pay from account"() {
    given:
      def account = new Account()
      account.balance = 100
    when:
      account.pay(50)
    then:
      account.balance == 50
  }
}
```

```
class Account {

  var balance: BigDecimal = BigDecimal.ZERO

  fun pay(amount: BigDecimal) {
    balance = balance.subtract(amount)
  }
}
```

Первое знакомство

```
class AccountSpec extends Specification {

  def "pay from account"() {
    given:
      def account = new Account()
      account.balance = 100
    when:
      account.pay(50)
    then:
      account.balance == 50
  }
}
```

```
class Account {

  var balance: BigDecimal = BigDecimal.ZERO

  fun pay(amount: BigDecimal) {
    balance = balance.subtract(amount)
  }
}
```

Первое знакомство

```
class AccountSpec extends Specification {

  def "pay from account"() {
    given:
      def account = new Account()
      account.balance = 100
    when:
      account.pay(50)
    then:
      account.balance == 50
  }
}
```

```
class Account {

  var balance: BigDecimal = BigDecimal.ZERO

  fun pay(amount: BigDecimal) {
    balance = balance.subtract(amount)
  }
}
```

Первое знакомство

```
class AccountSpec extends Specification {

  def "pay from account"() {
    given:
      def account = new Account()
      account.balance = 100
    when:
      account.pay(50)
    then:
      account.balance == 50
  }
}
```

```
class Account {

  var balance: BigDecimal = BigDecimal.ZERO

  fun pay(amount: BigDecimal) {
    balance = balance.subtract(amount)
  }
}
```

Проверка результата

```
class AccountSpec extends Specification {

  def "pay from account"() {
    given:
      def account = new Account()
      account.balance = 100
    when:
      account.pay(50)
    then:
      account.balance == 50
  }
}
```

```
class Account {

  var balance: BigDecimal = BigDecimal.ZERO

  fun pay(amount: BigDecimal) {
    balance = balance.subtract(amount)
  }
}
```

Комментарии к блокам

```
class AccountSpec extends Specification {

  def "pay from account"() {
      given: "Account balance is 100"
         def account = new Account()
         account.balance = 100
      when: "Pay 50"
         account.pay(50)
      then: "Now balance is 50"
         account.balance == 50
  }
}
```

Комментарии к блокам

```
class AccountSpec extends Specification {

  def "pay from account"() {
      given: "Account balance is 100"
        def account = new Account()
        account.balance = 100
      when: "Pay 50"
        account.pay(50)
      then: "Now balance is 50"
        account.balance == 50
  }
}
```

Комментарии к блокам

```
class AccountSpec extends Specification {

  def "pay from account"() {
    given: "Create new account"
      def account = new Account()
    and: "Refill account to 100"
      account.balance = 100
    when: "Pay 50"
      account.pay(50)
    then: "Now balance is 50"
      account.balance == 50
  }
}
```

Два теста в классе

```
class AccountSpec extends Specification {
  Account account

  def setup() {
    account = new Account()
  }

  def "pay from account"() {
    given:
      account.balance = 100
    when:
      account.pay(50)
    then:
      account.balance == 50
  }

  def "pay from empty balance account"() {
    given:
      account.balance = 0
    when:
      account.pay(50)
    then:
      thrown(IllegalStateException)
  }
}
```

Два теста в классе

```
class AccountSpec extends Specification {
  Account account

  def setup() {
    account = new Account()
  }

  def "pay from account"() {
    given:
      account.balance = 100
    when:
      account.pay(50)
    then:
      account.balance == 50
  }

  def "pay from empty balance account "() {
    given:
      account.balance = 0
    when:
      account.pay(50)
    then:
      thrown(IllegalStateException)
  }
}
```

Два теста в классе

```
class AccountSpec extends Specification {
  Account account

  def setup() {
    account = new Account()
  }

  def "pay from account"() {
    given:
      account.balance = 100
    when:
      account.pay(50)
    then:
      account.balance == 50
  }

  def "pay from empty balance account"() {
    given:
      account.balance = 0
    when:
      account.pay(50)
    then:
      thrown(IllegalStateException)
  }
}
```

Аналоги из junit

def setup(){}	@Before	Перед каждым тестом
def cleanup(){}	@After	После каждого теста
def setupSpec(){}	@BeforeAll	Один раз перед первым тестом в классе
def cleanupSpec(){}	@AfterAll	Один раз после последнего теста в классе

Еще блоки

expect:	заменяет when и then одним блоком
setup:	тоже самое, что и given
cleanup:	вызывается в конце теста
and:	вспомогательный блок
where:	используется для написания Data Driver Testing

Проверка исключений

```
class Account {

    var balance: BigDecimal = BigDecimal.ZERO

    fun pay(amount: BigDecimal) {
        if (balance < amount) {
            throw IllegalStateException("Balance less than amount")
        }
        balance = balance.subtract(amount)
    }
}
```

Проверка исключений

```
def "pay from empty balance account"() {
  given:
    def account = new Account()
    account.balance = 0
  when:
    account.pay(50)
  then:
    thrown(IllegalStateException)
}
```

Проверка исключений

```
def "pay from empty balance account"() {
  given:
    def account = new Account()
    account.balance = 0
  when:
    account.pay(50)
  then:
    thrown(IllegalStateException)
}
```

Проверка исключений

```
def "pay from empty balance account"() {
  given:
    def account = new Account()
    account.balance = 0
  when:
    account.pay(50)
  then:
    IllegalStateException exception = thrown()
    exception.message == "Balance less than amount"
}
```

Вернемся к нашему первому тесту

```
class AccountSpec extends Specification {

  def "pay from account"() {
    given:
      def account = new Account()
      account.balance = 100
    when:
      account.pay(50)
    then:
      account.balance == 40
  }
}
```

Добавим в тест ошибку

```
class AccountSpec extends Specification {

  def "pay from account"() {
    given:
      def account = new Account()
      account.balance = 100
    when:
      account.pay(50)
    then:
      account.balance == 40
  }
}
```

Понятное отображение ошибок

```
Condition not satisfied:

account.balance == 40
|       |       |
|       50      false
ru.alfabank.spock.Account@65d6b83b

Expected :40

Actual   :50

 <Click to see difference>


    at ru.alfabank.spock.AccountSpec.pay from account(AccountSpec.groovy:19)


Process finished with exit code 255
```

Data driven testing

- Проверяют код с одинаковым поведением
- Но разными наборами данных

Проверим работает ли математика

```
def "maximum of two numbers"() {
  expect:
    Math.max(1, 3) == 3
}
```

Проверим работает ли математика

```
def "maximum of two numbers"() {
  expect:
     Math.max(1, 3) == 3
     Math.max(7, 4) == 7
}
```

Проверим работает ли математика

```
def "maximum of two numbers"() {
  expect:
     Math.max(1, 3) == 3
     Math.max(7, 4) == 7
     Math.max(0, 0) == 0
}
```

Какие проблемы тут есть

- Код и данные перемешаны
- Нет возможности генерировать данные или получать их из внешних источников
- Код приходится дублировать или выносить в отдельный метод
- Тест упадет на первой неработающей проверке

Было бы удобно не дублировать код

```
def "maximum of two numbers"() {
  expect:
    Math.max(a, b) == c
    ...
}
```

Код отдельно, данные отдельно

```
def "maximum of two numbers"() {
  expect:
    Math.max(a, b) == c
  where:
    a | b | c
    1 | 3 | 3
    7 | 4 | 7
    0 | 0 | 0
}
```

Код отдельно, данные отдельно

```
def "maximum of two numbers"() {
  expect:
    Math.max(a, b) == c
  where:
    a | b || c
    1 | 3 || 3
    7 | 4 || 7
    0 | 0 || 0
}
```

Сделаем DDT для нашего класса

```
def "test pay"() {
  given:
    account.balance = balance
  when:
    account.pay(amount)
  then:
    account.balance == newBalance
  where:
    balance | amount | newBalance
    100     | 50     | 50
    100     | 40     | 60
```

Альтернативный вариант

```
def "test pay"() {
  given:
    account.balance = balance
  when:
    account.pay(amount)
  then:
    account.balance == newBalance
  where:
    balance << [100, 100]
    amount << [50, 40]
    newBalance << [50, 60]
}
```

В отчете будет такая запись

Test Summary

1	1	0	0.141s
tests	failures	ignored	duration

0%
successful

Failed tests | Packages | Classes

AccountSpec. test pay

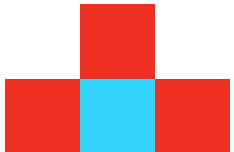
Что именно пошло не так

alfalab

test pay

```
Condition not satisfied:

account.balance == newBalance
|       |       |  |
|       50      |  10
|               false
ru.alfabank.spock.Account@11327d37

        at ru.alfabank.spock.AccountSpec.test pay(AccountSpec.groovy:36)
```

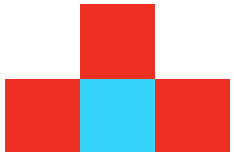
```
Condition not satisfied:

account.balance == newBalance
|       |       |  |
|       60      |  10
|               false
ru.alfabank.spock.Account@5c262272

        at ru.alfabank.spock.AccountSpec.test pay(AccountSpec.groovy:36)
```



A

Добавим аннотацию @Unroll

```
@Unroll
def "test pay"() {
  given:
    account.balance = balance
  when:
    account.pay(amount)
  then:
    account.balance == newBalance
  where:
    balance | amount | newBalance
    100     | 50     | 10
    100     | 40     | 10
}
```



Отчет о тесте

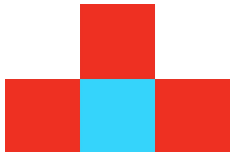
Test Summary

2	2	0	0.138s
tests	failures	ignored	duration

0%
successful

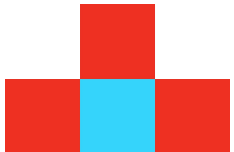
Failed tests | Packages | Classes

AccountSpec. test pay[0]
AccountSpec. test pay[1]



Щепотка магии

```
@Unroll
def "test pay #amount when origin balance #balance"() {
  given:
    account.balance = balance
  when:
    account.pay(amount)
  then:
    account.balance == newBalance
  where:
    balance | amount | newBalance
    100     | 50     | 10
    100     | 40     | 10
}
```



Отчет стал немного приятнее

alfalab

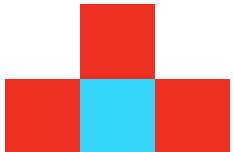
Test Summary

2	2	0	0.170s
tests	failures	ignored	duration

0%
successful

Failed tests | Packages | Classes

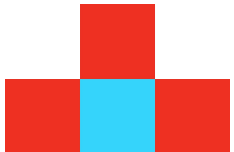
AccountSpec. test pay 40 when origin balance 100
AccountSpec. test pay 50 when origin balance 100



A

Какие аннотации есть еще

@Ignore	Игнорирует тест/класс
@IgnoreRest	Игнорирует все остальные тесты
@Shared	Аналог static поля
@Timeout	Тест должен выполниться за указанное время
@FailsWith	Тест должен завершится с указанной ошибкой



Тестирование взаимодействия

- Spock позволяет создавать моки без дополнительных зависимостей вроде Mockito
- Мокировать можно интерфейсы и не final классы

Рассмотрим пример

```
class Publisher {

  val subscribers = mutableListOf<Subscriber>()

  fun send(message: String) {
     subscribers.forEach { it.receive(message) }
  }
}

interface Subscriber {
  fun receive(message: String)
}
```

Напишем тест

```
class PublisherSpec extends Specification {

  def "publish test"() {
    given:
      def publisher = new Publisher()
      def subscriber = Mock(Subscriber)
      publisher.subscribers << subscriber
    when:
      publisher.send("Some message")
    then:
      1 * subscriber.receive(_)
  }
}
```

Создание мока

```
class PublisherSpec extends Specification {

  def "publish test"() {
    given:
      def publisher = new Publisher()
      def subscriber = Mock(Subscriber)
      publisher.subscribers << subscriber
    when:
      publisher.send("Some message")
    then:
      1 * subscriber.receive(_)
  }
}
```

Или так

```
class PublisherSpec extends Specification {

  def "publish test"() {
    given:
      def publisher = new Publisher()
      Subscriber subscriber = Mock()
      publisher.subscribers << subscriber
    when:
      publisher.send("Some message")
    then:
      1 * subscriber.receive(_)
  }
}
```

Проверка взаимодействия

```
class PublisherSpec extends Specification {

  def "publish test"() {
    given:
      def publisher = new Publisher()
      def subscriber = Mock(Subscriber)
      publisher.subscribers << subscriber
    when:
      publisher.send("Some message")
    then:
      1 * subscriber.receive(_)
  }
}
```

Количество вызовов

```
class PublisherSpec extends Specification {

  def "publish test"() {
    given:
      def publisher = new Publisher()
      def subscriber = Mock(Subscriber)
      publisher.subscribers << subscriber
    when:
      publisher.send("Some message")
    then:
      1 * subscriber.receive(_)
  }
}
```

Объект взаимодействия

```
class PublisherSpec extends Specification {

  def "publish test"() {
    given:
      def publisher = new Publisher()
      def subscriber = Mock(Subscriber)
      publisher.subscribers << subscriber
    when:
      publisher.send("Some message")
    then:
      1 * subscriber.receive(_)
  }
}
```

Метод взаимодействия

```
class PublisherSpec extends Specification {

  def "publish test"() {
    given:
      def publisher = new Publisher()
      def subscriber = Mock(Subscriber)
      publisher.subscribers << subscriber
    when:
      publisher.send("Some message")
    then:
      1 * subscriber.receive(_)
  }
}
```

Аргумент взаимодействия

```
class PublisherSpec extends Specification {

  def "publish test"() {
    given:
      def publisher = new Publisher()
      def subscriber = Mock(Subscriber)
      publisher.subscribers << subscriber
    when:
      publisher.send("Some message")
    then:
      1 * subscriber.receive(_)
  }
}
```

Напишем тест

```
class PublisherSpec extends Specification {

  def "publish test"() {
    given:
      def publisher = new Publisher()
      def subscriber = Mock(Subscriber)
      publisher.subscribers << subscriber
    when:
      publisher.send("Some message")
    then:
      1 * subscriber.receive("Some message")
  }
}
```

Тоже самое в Mockito

```
verify(subscriber, times(1)).receive(anyString())
```

Возможные шаблоны взаимодействия

1 * subscriber.receive(_)	Один раз
0 * subscriber.receive(_)	Ни одного раза
(1..3) * subscriber.receive(_)	От одного до трех включительно
(_..3) * subscriber.receive(_)	Не больше трех
(1.._) * subscriber.receive(_)	По крайней мере один
_ * subscriber.receive(_)	Любое количество

Шаблоны объекта

1 * subscriber.receive(_)	Должен быть вызван метод объекта subscriber
1 * _.receive(_)	Должен быть вызван метод любого объекта

Шаблоны метода

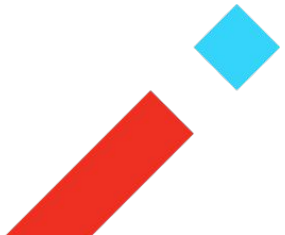
`1 * subscriber.receive(_)`	Должен быть вызван метод с именем receive
`1 * subscriber./r.*e/(_)`	Должен быть вызван метод удовлетворяющий заданному регулярному выражению

Шаблоны аргумента

`1 * subscriber.receive("hello")`	Аргумент будет равен “hello”
`1 * subscriber.receive(!"hello")`	Аргумент будет не равен “hello”
`1 * subscriber.receive()`	Без аргументов
`1 * subscriber.receive(_)`	С любым одним аргументом
`1 * subscriber.receive(*_)`	С любым количеством любых аргументов
`1 * subscriber.receive(!null)`	Аргумент не null
`1 * subscriber.receive(_ as String)`	С указанным типом
`1 * subscriber.receive({ it.size() > 3 })`	Или с заданным условием

Можно комбинировать

```
(1..3) * process./r.*e/("ls", "-a", _, !null, { ["abcdefghiklmnopqrstuwx1"].contains(it) })
```



Усложним пример

```
class Publisher2 {

  val subscribers = mutableListOf<Subscriber2>()

  fun send(message: String) {
     subscribers.forEach {
        if (it.isActive) {
           it.receive(message)
        }
     }
  }
}

interface Subscriber2 {
  val isActive: Boolean
  fun receive(message: String)
}
```

Усложним пример

```
class Publisher2 {

  val subscribers = mutableListOf<Subscriber2>()

  fun send(message: String) {
     subscribers.forEach {
        if (it.isActive) {
           it.receive(message)
        }
     }
  }
}

interface Subscriber2 {
  val isActive: Boolean
  fun receive(message: String)
}
```

Усложним пример

```
class Publisher2 {

  val subscribers = mutableListOf<Subscriber2>()

  fun send(message: String) {
    subscribers.forEach {
      if (it.isActive) {
        it.receive(message)
      }
    }
  }
}

interface Subscriber2 {
  val isActive: Boolean
  fun receive(message: String)
}
```

Напишем тест

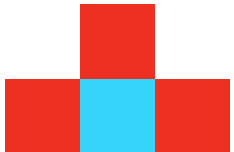
```
class Publisher2Spec extends Specification {

  def "send test"() {
    given:
      def publisher = new Publisher2()

      def subscriber = Mock(Subscriber2)
      subscriber.active >> true

      def subscriber2 = Mock(Subscriber2)
      subscriber2.active >> false

      publisher.subscribers << subscriber << subscriber2
    when:
      publisher.send("Hello, Spock")
    then:
      1 * subscriber.receive(_)
      0 * subscriber2.receive(_)
  }
}
```

Напишем тест

```
class Publisher2Spec extends Specification {

  def "send test"() {
    given:
      def publisher = new Publisher2()

      def subscriber = Mock(Subscriber2)
      subscriber.active >> true

      def subscriber2 = Mock(Subscriber2)
      subscriber2.active >> false

      publisher.subscribers << subscriber << subscriber2
    when:
      publisher.send("Hello, Spock")
    then:
      1 * subscriber.receive(_)
      0 * subscriber2.receive(_)
  }
}
```

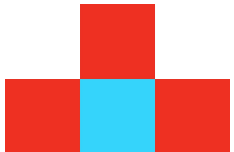
Еще немного поменяем код

```
class Publisher3 {

  val subscribers = mutableListOf<Subscriber3>()

  fun send(message: String): Boolean {
    return subscribers.map { it.receive(message) }.reduce { acc, b -> acc && b }
  }
}

interface Subscriber3 {
  fun receive(message: String): Boolean
}
```



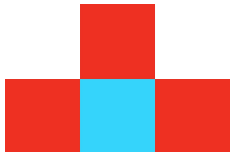
Еще немного поменяем код

```
class Publisher3 {

  val subscribers = mutableListOf<Subscriber3>()

  fun send(message: String): Boolean {
    return subscribers.map { it.receive(message) }.reduce { acc, b -> acc && b }
  }
}

interface Subscriber3 {
  fun receive(message: String): Boolean
}
```



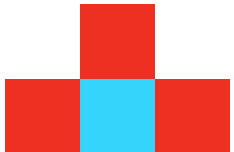
Проверим

```
class Publisher3Spec extends Specification {

  def "send test"() {
    given:
      def publisher = new Publisher3()
      def subscriber = Mock(Subscriber3)
      def subscriber2 = Mock(Subscriber3)
      publisher.subscribers << subscriber << subscriber2
    when:
      def actual = publisher.send("Hello")
    then:
      1 * subscriber.receive(_) >> true
      1 * subscriber2.receive(_) >> true
      actual == true
  }
}
```



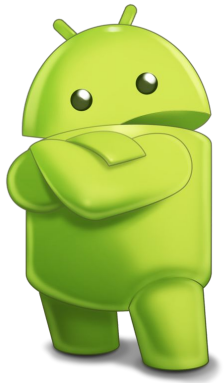
Проверим

```
class Publisher3Spec extends Specification {

  def "send test"() {
    given:
      def publisher = new Publisher3()
      def subscriber = Mock(Subscriber3)
      def subscriber2 = Mock(Subscriber3)
      publisher.subscribers << subscriber << subscriber2
    when:
      def actual = publisher.send("Hello")
    then:
      1 * subscriber.receive(_) >> true
      1 * subscriber2.receive(_) >> true
      actual == true
  }
}
```



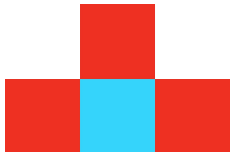
А что же про Android

1. Как там Robolectric
2. Как подключить spock в проект
3. Подводные камни



Добавим Robolectric

```
@RunWith(RobolectricTestRunner)
class SimpleViewTest extends Specification {

  def "set text"() {
    given:
      TextView textView = new TextView(RuntimeEnvironment.application)
    when:
      textView.text = "test"
    then:
      textView.text == "test"
  }
}
```

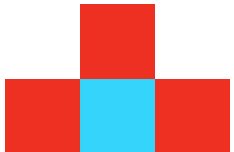


И что получим

```
java.lang.Exception: No runnable methods
    at org.junit.runners.BlockJUnit4ClassRunner.validateInstanceMethods(BlockJUnit4ClassRunner.java:191)
    at org.junit.runners.BlockJUnit4ClassRunner.collectInitializationErrors(BlockJUnit4ClassRunner.java:128)
    at org.junit.runners.ParentRunner.validate(ParentRunner.java:416)
    at org.junit.runners.ParentRunner.<init>(ParentRunner.java:84)
    at org.junit.runners.BlockJUnit4ClassRunner.<init>(BlockJUnit4ClassRunner.java:65)
    at org.robolectric.internal.SandboxTestRunner.<init>(SandboxTestRunner.java:43)
    at org.robolectric.RobolectricTestRunner.<init>(RobolectricTestRunner.java:77) <4 internal calls>
    at org.junit.internal.builders.AnnotatedBuilder.buildRunner(AnnotatedBuilder.java:104)
    at org.junit.internal.builders.AnnotatedBuilder.runnerForClass(AnnotatedBuilder.java:86) <1 internal calls>
    at org.junit.internal.builders.AllDefaultPossibilitiesBuilder.runnerForClass(AllDefaultPossibilitiesBuilder.java:26) <1 internal calls>
    at org.junit.internal.requests.ClassRequest.getRunner(ClassRequest.java:33) <9 internal calls>
```

Spock использует свой Runner

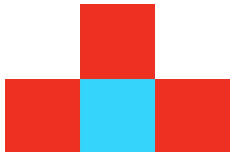
```
@RunWith(Sputnik.class)
public abstract class Specification extends MockingApi {
    ...
}
```



Есть решение

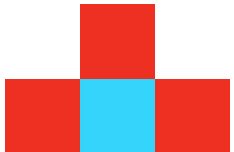
https://github.com/hkhc/electricspock

Заменяет базовый класс на ElectricSpecification с собственным Runner`ом.



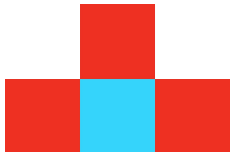
Тест теперь выглядит так

```
class SimpleViewTest extends ElectricSpecification {

  def "set text"() {
    given:
      TextView textView = new TextView(RuntimeEnvironment.application)
    when:
      textView.text = "test"
    then:
      textView.text == "test"
  }
}
```



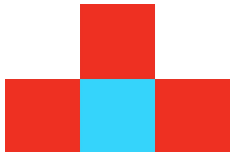
Что это нам стоит

- Версия robolectric зависит от версии electricspock
- electricspock может быть заброшен как robospock
  - electricspock поддерживает один человек
  - robolectric, иногда, кардинально меняет апи



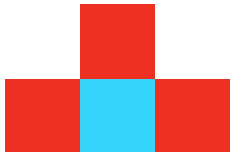
Как подключить Spock:

```
buildscript {
   dependencies {
      classpath 'org.codehaus.groovy:groovy-android-gradle-plugin:2.0.0'
   }
}

apply plugin: 'groovyx.android'

dependencies {
   testImplementation 'org.codehaus.groovy:groovy-all:2.4.13'
   testImplementation 'org.spockframework:spock-core:1.1-groovy-2.4'
   testImplementation 'cglib:cglib-nodep:3.2.5'
}
```



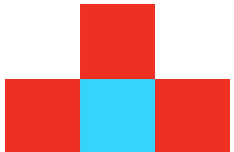
Groovy Android Plugin

```
buildscript {
  dependencies {
    classpath 'org.codehaus.groovy:groovy-android-gradle-plugin:2.0.0'
  }
}

apply plugin: 'groovyx.android'

dependencies {
  testImplementation 'org.codehaus.groovy:groovy-all:2.4.13'
  testImplementation 'org.spockframework:spock-core:1.1-groovy-2.4'
  testImplementation 'cglib:cglib-nodep:3.2.5'
}
```



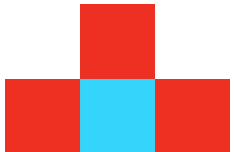
И собственно сам Spock

```
buildscript {
   dependencies {
      classpath 'org.codehaus.groovy:groovy-android-gradle-plugin:2.0.0'
   }
}

apply plugin: 'groovyx.android'

dependencies {
   testImplementation 'org.codehaus.groovy:groovy-all:2.4.13'
   testImplementation 'org.spockframework:spock-core:1.1-groovy-2.4'
   testImplementation 'cglib:cglib-nodep:3.2.5'
}
```



Подводные камни

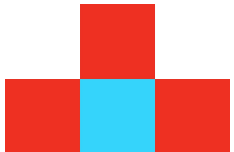
- За разработку плагина отвечает groovy community
- Отставание от релизов android и kotlin плагинов
- С чем мы столкнулись:
  - Не было поддержки kotlin
  - Невозможно было перейти на версию 3+ и 26 билд тулс
  - Пришлось пересобирать groovy



Артефакты с кодом в Android Studio

Вы не узнаете, пока не запустите тесты:

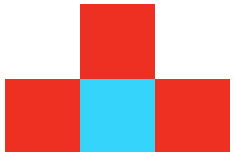
- Об отсутствии импорта класса или метода
- О изменении декларации метода
- О переименовании или удалении класса или метода



Выводы:

alfalab

1. Структурированные тесты
2. Простые проверки
3. Подробные сообщения об ошибках
4. Data Driven Testing
5. Свой механизм мокирования
6. Лаконичная проверка взаимодейсвия
7. Совместимость с android
8. Зависимость от groovy plugin и библиотек совместимости



Ссылки:

http://spockframework.org/ - документация

https://habrahabr.ru/post/137561/

https://www.youtube.com/watch?v=L_Rc3kjZrfQ - подробный доклад Евгения Борисова

Вопросы?!

alfalab