Kotlin Goodies for Testing

Pasha Finkelshteyn, BellSoft

layout: two-cols whoami

- developer
- Average enjoyer
- Developer at BellSoft
- Data engineer
- Speaker



layout: center Who are you?

- 1. You write code/tests in Java or Kotlin
- 2. You wanna migrate to Kotlin, *or*
- 3. You wanna make your tests in Kotlin better, *or*
- 4. You wanna introduce Kotlin in your projects!

layout: center How are we doing it?

- 1. Using DSLs (domain specific languages)
- 2. Structuring our code better
- 3. Using smart libraries

layout: image-right	image: /bdd.png		

BDD

- 1. Given
- 2. When
- 3. Then

```
Feature: An example

Scenario: The example

Given I have a username "pasha"

When I type username in field "x"

And Press OK button

Then I get an error "Enter password"
```

What's the problem here?

```
Feature: An example

Scenario: The example

Given I have a username "pasha"

When I type username in field "x"

And Press OK button

Then I get an error "Enter password"
```

This line might intersect with **any** test

```
Feature: An example

Scenario: The example

Given I have a username "pasha"

When I type username in field "x"

And Press OK button

Then I get an error "Enter password"
```

Actually **any** line can

```
Feature: An example

Scenario: The example

Given I have a username "pasha"

When I type username in field "x"

And Press OK button

Then I get an error "Enter password"
```

Actually **any** line can

Steps match by **name** O_o:

- No obscure text files
- No string matching
- everything in one place

- No obscure text files
- No string matching
- everything in one place

- No obscure text files
- No string matching
- everything in one place

- No obscure text files
- No string matching
- everything in one place

- No obscure text files
- No string matching
- everything in one place

- No obscure text files
- No string matching
- everything in one place

- No obscure text files
- No string matching
- everything in one place

```
class MyTests : DescribeSpec({
    describe("score") {
        it("start as zero") { /* test here */ }
       describe("with a strike") {
           it("adds ten") { /* test here */ }
            it("carries strike to the next frame") { /* test here */ }
        describe("for the opposite team") {
            it("Should negate one score") { /* test here */ }
})
```

```
describe("score") {
```

```
it("start as zero") { /* test here */ }
```

```
describe("with a strike") {
    it("adds ten") { /* test here */ }
    it("carries strike to the next frame") { /* test here */ }
```

```
describe("for the opposite team") {
    it("Should negate one score") { /* test here */ }
```

```
class AnnotationSpecExample : AnnotationSpec() {
    @BeforeEach
    fun beforeTest() {
        println("Before each test")
    <u>a</u>Test
    fun test1() {
        1 shouldBe 1
    aTest
    fun test2() {
        3 shouldBe 3
```

```
class AnnotationSpecExample : AnnotationSpec() {
```

```
@BeforeEach
```

```
@BeforeEach
fun beforeTest() {
    println("Before each test")
```

```
aTest
fun test1() {
    1 shouldBe 1
```

```
class AnnotationSpecExample : AnnotationSpec() {
    @BeforeEach
    fun beforeTest() {
        println("Before each test")
    <u>a</u>Test
    fun test1() {
        1 shouldBe 1
    aTest
    fun test2() {
        3 shouldBe 3
```

Kotest has its own assertions

kotest-assertions-core

- kotest-assertions-core
- kotest-assertions-json

- kotest-assertions-core
- kotest-assertions-json
- kotlintest-assertions-jsoup

- kotest-assertions-core
- kotest-assertions-json
- kotlintest-assertions-jsoup
- And namy others

Core Assertions

```
1  1 shouldBe 1
2  1 shouldNotBe 2
3  "Hello" shouldBe "Hello"
4  listOf(1,2,3) shouldContain 2
5  mapOf("a" to 1, "b" to 2) shouldContainKey "a"
6  null shouldBe null
7  "abc" shouldStartWith "a"
8  "xyz" shouldEndWith "z"
9  5 shouldBeGreaterThan 3
10  2 shouldBeLessThan 5
```

```
1  1 shouldBe 1
2  1 shouldNotBe 2
3  "Hello" shouldBe "Hello"
4  listOf(1,2,3) shouldContain 2
5  mapOf("a" to 1, "b" to 2) shouldContainKey "a"
6  null shouldBe null
7  "abc" shouldStartWith "a"
8  "xyz" shouldEndWith "z"
9  5 shouldBeGreaterThan 3
10  2 shouldBeLessThan 5
```

```
1  1 shouldBe 1
2  1 shouldNotBe 2
3  "Hello" shouldBe "Hello"
4  listOf(1,2,3) shouldContain 2
5  mapOf("a" to 1, "b" to 2) shouldContainKey "a"
6  null shouldBe null
7  "abc" shouldStartWith "a"
8  "xyz" shouldEndWith "z"
9  5 shouldBeGreaterThan 3
10  2 shouldBeLessThan 5
```

```
1  1 shouldBe 1
2  1 shouldNotBe 2
3  "Hello" shouldBe "Hello"
4  listOf(1,2,3) shouldContain 2
5  mapOf("a" to 1, "b" to 2) shouldContainKey "a"
6  null shouldBe null
7  "abc" shouldStartWith "a"
8  "xyz" shouldEndWith "z"
9  5 shouldBeGreaterThan 3
10  2 shouldBeLessThan 5
```

```
1  1 shouldBe 1
2  1 shouldNotBe 2
3  "Hello" shouldBe "Hello"
4  listOf(1,2,3) shouldContain 2
5  mapOf("a" to 1, "b" to 2) shouldContainKey "a"
6  null shouldBe null
7  "abc" shouldStartWith "a"
8  "xyz" shouldEndWith "z"
9  5 shouldBeGreaterThan 3
10  2 shouldBeLessThan 5
```

JSON Assertions

```
"{}".shouldBeValidJson()
"{}".shouldBeJson0bject()
a.shouldEqualJson(b, compareJson0ptions { array0rder = Array0rder.Strict })
```

Or even...

JSON Assertions

```
"{}".shouldBeValidJson()
"{}".shouldBeJsonObject()
a.shouldEqualJson(b, compareJsonOptions { arrayOrder = ArrayOrder.Strict })
```

Or even...

```
val addressSchema = jsonSchema {
```

```
obj { // object is reserved, obj was chosen over jsonObject for brevity but could be char
```

```
withProperty("street", required = true) { string() }
```

```
withProperty("zipCode", required = true) {
 integer {
    beEven() and beInRange(10000...99999) // supports constructing a matcher that will be
```

```
integer {
  beEven() and beInRange(10000...99999) // supports constructing a matcher that will be
```

```
beEven() and beInRange(10000...99999) // supports constructing a matcher that will be
```

```
additional Properties = false // triggers failure if other properties are defined in act
```

```
val personSchema = jsonSchema {
```

```
withProperty("name", required = true) { string() }
```

```
withProperty("address") { addressSchema() } // Schemas can re-use other schemas
```

```
"{}" shouldMatchSchema personSchema
```

```
""" { "name": "Emil", "age": 34 } """
```

Jsoup

```
1 element.shouldHaveChildWithTag(tag)
2 element.shouldHaveText(text)
3 element.shouldHaveAttribute(name)
```

element.shouldHaveSrc(src)

50 shades of kotest

https://kotest.io

Fun spec	String Spec
Should spec	Describe spec
Behaviour spec	Word spec
Free spec	Feature spec
Expect spec	Annotation spec

layout: center How do I run the same tests with different inputs?

Data-driven testing a.k.a Parametrized tests

```
static Stream<Arguments> stringIntAndListProvider() {
    return Stream.of(
        arguments("apple", 1, Arrays.asList("a", "b")),
        arguments("lemon", 2, Arrays.asList("x", "y"))
    );
}

@ParameterizedTest
@MethodSource("stringIntAndListProvider")
```

```
4          arguments("lemon", 2, Arrays.asList("x", "y"))
5          );
6     }
7          @ParameterizedTest
8          @MethodSource("stringIntAndListProvider")
9          void testWithMultiArgMethodSource(String str, int num, List<String> list) {/
10          // snip
11     }
```

```
4          arguments("lemon", 2, Arrays.asList("x", "y"))
5         );
6    }
7          aParameterizedTest
8          aMethodSource("stringIntAndListProvider")
9          void testWithMultiArgMethodSource(String str, int num, List<String> list) {/
10          // snip
11    }
```

```
static Stream<Arguments> stringIntAndListProvider() {
    return Stream.of(
        arguments("apple", 1, Arrays.asList("a", "b")),
        arguments("lemon", 2, Arrays.asList("x", "y"))
    );
}

@ParameterizedTest
@MethodSource("stringIntAndListProvider")
```

```
static Stream<Arguments> stringIntAndListProvider() {
   return Stream.of(
        arguments("apple", 1, Arrays.asList("a", "b")),
        arguments("lemon", 2, Arrays.asList("x", "y"))
   );
}

AParameterizedTest

@MethodSource("stringIntAndListProvider")
```

```
static Stream<Arguments> stringIntAndListProvider() {
    return Stream.of(
        arguments("apple", 1, Arrays.asList("a", "b")),
        arguments("lemon", 2, Arrays.asList("x", "y"))
    );
}

@ParameterizedTest
@MethodSource("stringIntAndListProvider")
```

```
static Stream<Arguments> stringIntAndListProvider() {
    return Stream.of(
        arguments("apple", 1, Arrays.asList("a", "b")),
        arguments("lemon", 2, Arrays.asList("x", "y"))
    );
}

arguments("stringIntAndListProvider")
```

```
static Stream<Arguments> stringIntAndListProvider() {
    return Stream.of(
        arguments("apple", 1, Arrays.asList("a", "b")),
        arguments("lemon", 2, Arrays.asList("x", "y"))
    );
}

@ParameterizedTest
@MethodSource("stringIntAndListProvider")
```

```
static Stream<Arguments> stringIntAndListProvider() {
    return Stream.of(
        arguments("apple", 1, Arrays.asList("a", "b")),
        arguments("lemon", 2, Arrays.asList("x", "y"))
    );
}

@ParameterizedTest
@MethodSource("stringIntAndListProvider")
```

stringIntAndListProvider , really?

```
static Stream<Arguments> stringIntAndListProvider() {
    return Stream.of(
        arguments("apple", 1, Arrays.asList("a", "b")),
        arguments("lemon", 2, Arrays.asList("x", "y"))
    );
}

@ParameterizedTest
@MethodSource("stringIntAndListProvider")
```

stringIntAndListProvider , really?

Becomes cluttered

```
static Stream<Arguments> stringIntAndListProvider() {
    return Stream.of(
        arguments("apple", 1, Arrays.asList("a", "b")),
        arguments("lemon", 2, Arrays.asList("x", "y"))
    );
}

@ParameterizedTest
@MethodSource("stringIntAndListProvider")
```

stringIntAndListProvider , really?

- Becomes cluttered
- Not typed (enough)

```
static Stream<Arguments> stringIntAndListProvider() {
    return Stream.of(
        arguments("apple", 1, Arrays.asList("a", "b")),
        arguments("lemon", 2, Arrays.asList("x", "y"))
    );
}

@ParameterizedTest
@MethodSource("stringIntAndListProvider")
```

stringIntAndListProvider , really?

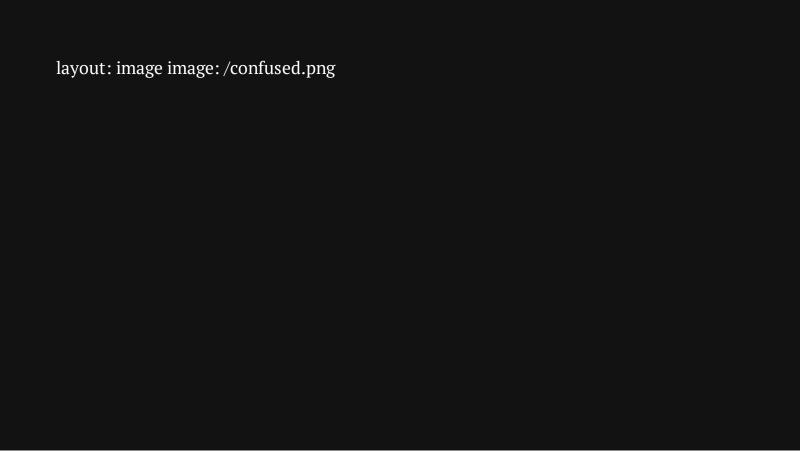
- Becomes cluttered
- Not typed (enough)
- Bound by a string!

Parametrized tests? Kotlin (with kotest)

```
context("Pythag triples tests") {
    withData(
        PythagTriple(3, 4, 5),
        PythagTriple(6, 8, 10),
        PythagTriple(8, 15, 17),
        PythagTriple(7, 24, 25)
    ) { (a, b, c) ->
        isPythagTriple(a, b, c) shouldBe true
    }
}
```

Parametrized tests? Kotlin (with kotest)

```
context("Pythag triples tests") {
    withData(
        PythagTriple(3, 4, 5),
        PythagTriple(6, 8, 10),
        PythagTriple(8, 15, 17),
        PythagTriple(7, 24, 25)
    ) { (a, b, c) ->
        isPythagTriple(a, b, c) shouldBe true
    }
}
```



What if

I need to test things I don't know I need to

test?

```
withData(
    "love",
    "secret",
    "god", // Who will remember the reference?
) { input ->
    MD5.compute(input) shouldBe libMd5(input)
}
```

```
withData(
    "love",
    "secret",
    "god", // Who will remember the reference?
    ) { input ->
        MD5.compute(input) shouldBe libMd5(input)
}
```

```
withData(
   "love",
   "secret",
   "god", // Who will remember the reference?
   ) { input ->
        MD5.compute(input) shouldBe libMd5(input)
}
```

```
withData(
    "love",
    "secret",
    "god", // Who will remember the reference?
) { input ->
    MD5.compute(input) shouldBe libMd5(input)
}
```

```
withData(
    "love",
    "secret",
    "god", // Who will remember the reference?
) { input ->
    MD5.compute(input) shouldBe libMd5(input)
}
```

Launching...

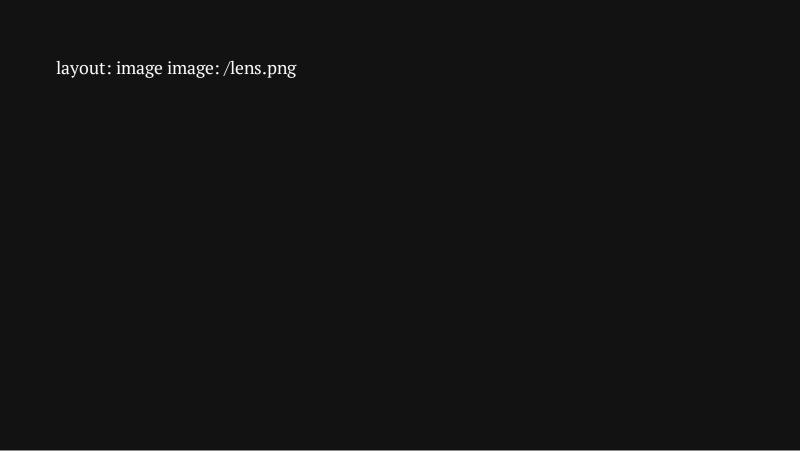
clicks: 5

Never write your own hashing (unless...)

```
withData(
    "love",
    "secret",
    "god", // Who will remember the reference?
) { input ->
    MD5.compute(input) shouldBe libMd5(input)
}
```

Launching...

- 1. **✓** love
- 2. ✓ secret
- 3. **✓** god



Time to investigate!

Property tests!

To test things we can not even imagine

layout: statement Demo

```
checkAll(
Arb.string(
codepoints = Arb.of(

(Char.MIN_VALUE.code..Char.MAX_VALUE.code).map(::Codepoint)
),

range = 0..52

))

input ->

MD5.compute(input) shouldBe libMd5(input)

}
```

```
checkAll(
Arb.string(
codepoints = Arb.of(

(Char.MIN_VALUE.code..Char.MAX_VALUE.code).map(::Codepoint)
),
range = 0..52

))
(input ->
MD5.compute(input) shouldBe libMd5(input)

}
```

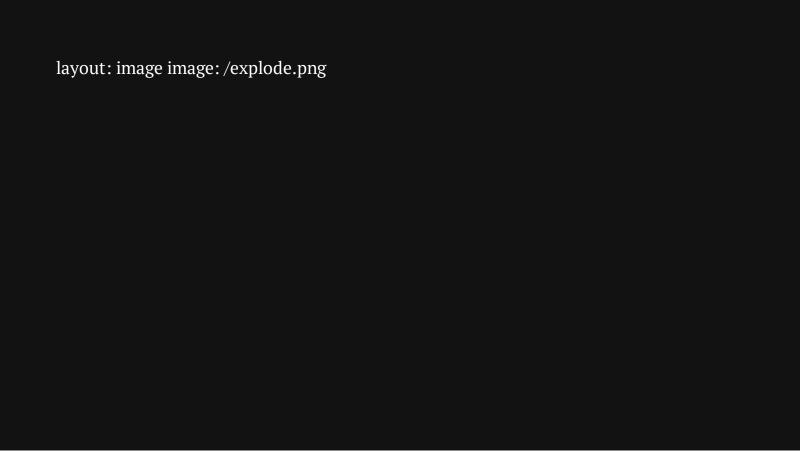
```
checkAll(
Arb.string(
codepoints = Arb.of(

(Char.MIN_VALUE.code..Char.MAX_VALUE.code).map(::Codepoint)
),
range = 0..52

))

input ->
MD5.compute(input) shouldBe libMd5(input)

}
```





Wow, it was a complex sample!

```
Attempting to shrink arg "οΛοσοσοσοσοσοσοσοδα" fail

Shrink #1: "οΛοσοσοσοσοσοσοσοδα" fail

Shrink #3: "οΛοσοδα" fail

Shrink #4: "οΛω" fail

Shrink #5: "οΛ" fail

Shrink #6: "Δ" fail

Shrink #7: <empty string> pass

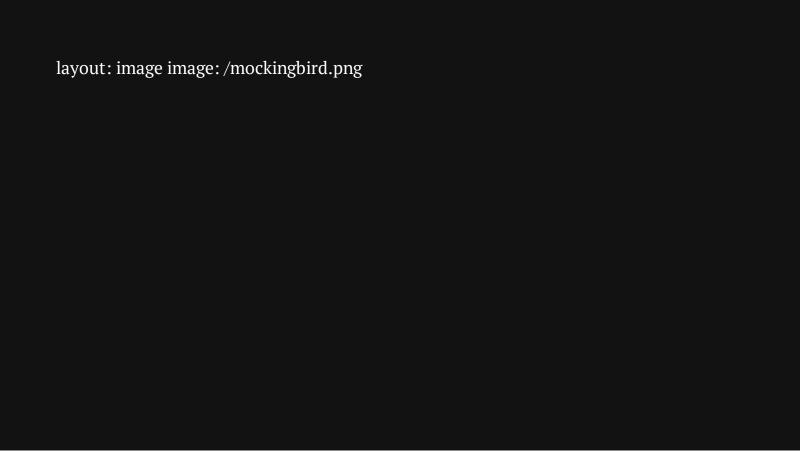
Shrink result (after 7 shrinks) => "Δ"
```

Repeat? Yes!

- reproduce the test exactly like in the failed instance
- debug it
- fix it
- PROFIT!!

Repeat? Yes!

- reproduce the test exactly like in the failed instance
- debug it
- fix it
- PROFIT!!





Disclaimer

I believe one should mock, not fake. Because

- 1. Granular control
- 2. Can check that something was NOT called

```
class UserService(val repo: UserRepository) {
    fun getAllUsers() = repo.allUsers()
}

class UserRepository {
    fun allUsers(): List<User> { TODO("demo") }
}

class User(val name: String, val age: Int)
```

```
class UserService(val repo: UserRepository) {
   fun getAllUsers() = repo.allUsers()
}

class UserRepository {
   fun allUsers(): List<User> { TODO("demo") }
}

class User(val name: String, val age: Int)
```

```
class UserService(val repo: UserRepository) {
   fun getAllUsers() = repo.allUsers()
}

class UserRepository {
   fun allUsers(): List<User> { TODO("demo") }
}

class User(val name: String, val age: Int)
```

```
class UserService(val repo: UserRepository) {
   fun getAllUsers() = repo.allUsers()
}

class UserRepository {
   fun allUsers(): List<User> { TODO("demo") }
}

class User(val name: String, val age: Int)
```

```
class UserService(val repo: UserRepository) {
    fun getAllUsers() = repo.allUsers()
}

class UserRepository {
    fun allUsers(): List<User> { TODO("demo") }
}

class User(val name: String, val age: Int)
```

```
class UserService(val repo: UserRepository) {
    fun getAllUsers() = repo.allUsers()
}

class UserRepository {
    fun allUsers(): List<User> { TODO("demo") }
}

class User(val name: String, val age: Int)
```

```
class UserService(val repo: UserRepository) {
   fun getAllUsers() = repo.allUsers()
}

class UserRepository {
   fun allUsers(): List<User> { TODO("demo") }
}

class User(val name: String, val age: Int)
```

```
class UserService(val repo: UserRepository) {
   fun getAllUsers() = repo.allUsers()
}

class UserRepository {
   fun allUsers(): List<User> { TODO("demo") }
}

class User(val name: String, val age: Int)
```

```
UserRepository mockRepository = Mockito.mock(UserRepository.class);
List<User> users = Arrays.asList(new User("Alice", 25), new User("Bob", 30), new User("Charl
// Define the expected behavior of the mock repository
Mockito.when(mockRepository.allUsers()).thenReturn(users);
UserService userService = new UserService(mockRepository);
// Call the method under test
List<User> result = userService.getAllUsers();
Assertions.assertEquals(users, result);
```

```
UserRepository mockRepository = Mockito.mock(UserRepository.class);
```

```
List<User> users = Arrays.asList(new User("Alice", 25), new User("Bob", 30), new User("Charl
```

```
// Define the expected behavior of the mock repository
Mockito.when(mockRepository.allUsers()).thenReturn(users);
```

```
UserService userService = new UserService(mockRepository);
```

```
// Call the method under test
List<User> result = userService.getAllUsers();
```

```
Assertions.assertEquals(users, result);
```

```
val users = listOf(User("Alice", 25), User("Bob", 30), User("Charlie", 35))
val repo = mockk<UserRepository> {
    every { allUsers() } returns users
}
val service = UserService(repo)
service.getAllUsers() shouldBe users
```

```
val users = listOf(User("Alice", 25), User("Bob", 30), User("Charlie", 35))
val repo = mockk<UserRepository> {
    every { allUsers() } returns users
}
val service = UserService(repo)
service.getAllUsers() shouldBe users
```

```
val users = listOf(User("Alice", 25), User("Bob", 30), User("Charlie", 35))
val repo = mockk<UserRepository> {
    every { allUsers() } returns users
}
val service = UserService(repo)
service.getAllUsers() shouldBe users
```

```
val users = listOf(User("Alice", 25), User("Bob", 30), User("Charlie", 35))
val repo = mockk<UserRepository> {
    every { allUsers() } returns users
}
val service = UserService(repo)
service.getAllUsers() shouldBe users
```

```
val users = listOf(User("Alice", 25), User("Bob", 30), User("Charlie", 35))
val repo = mockk<UserRepository> {
    every { allUsers() } returns users
}
val service = UserService(repo)
service.getAllUsers() shouldBe users
```

```
val users = listOf(User("Alice", 25), User("Bob", 30), User("Charlie", 35))
val repo = mockk<UserRepository> {
    every { allUsers() } returns users
}
val service = UserService(repo)
service.getAllUsers() shouldBe users
```

```
val users = listOf(User("Alice", 25), User("Bob", 30), User("Charlie", 35))
val repo = mockk<UserRepository> {
    every { allUsers() } returns users
}
val service = UserService(repo)
service.getAllUsers() shouldBe users
```

Nice DSL

Code describing UserRepository is inside {}

What if I want to mock users too?

```
val repo = mockk<UserRepository> {
    every { allUsers() } returns listOf(
       mockk {
            every { name } returns "Pasha"
       mockk {
            every { name } returns "Alexandra"
        },
val service = UserService(repo)
service.getAllUsers()[1].name shouldBe "Alexandra"
```

```
val repo = mockk<UserRepository> {
```

```
every { allUsers() } returns listOf(
```

```
mockk {
    every { name } returns "Pasha"
},
```

```
mockk {
    every { name } returns "Alexandra"
```

```
val service = UserService(repo)
service.getAllUsers()[1].name shouldBe "Alexandra"
```

```
val service = UserService(repo)
service.getAllUsers()[1].name shouldBe "Alexandra"
```

Users are described inside User mocks!

https://docs.atriumlib.org/

https://docs.atriumlib.org/

```
expect(service.getAllUsers()) toContain o inAny order but only the entries(

its { name } toEqual "Pash"

its { age } toEqual 39

},

its { name } toEqual "Mark"

its { name } toEqual 16

}

)
```

https://docs.atriumlib.org/

```
expect(service.getAllUsers()) toContain o inAny order but only the entries(

its { name } toEqual "Pash"

its { age } toEqual 39

},

its { name } toEqual "Mark"

its { age } toEqual 16

}

)
```

I've made a mistake to show the error message

https://docs.atriumlib.org/

```
expect(service.getAllUsers()) toContain o inAny order but only the entries(

its { name } toEqual "Pash"

its { age } toEqual 39

},

its { name } toEqual "Mark"

its { age } toEqual 16

}

)
```

I've made a mistake to show the error message

https://docs.atriumlib.org/

```
expect(service.getAllUsers()) toContain o inAny order but only the entries(

its { name } toEqual "Pash"

its { age } toEqual 39

},

its { name } toEqual "Mark"

its { age } toEqual 16

}
```

I've made a mistake to show the error message

```
(java.util.Arrays.ArrayList <315506631>)
I expected subject: [User(#3), User(#4)]
♦ to contain only, in any order:
 x an element which needs:
     » ▶ its.definedIn(MockTest.kt:42):
         ■ to equal: "Pash"
                                 <2026086646>
     » ▶ its.definedIn(MockTest.kt:43):
         ■ to equal: 36 (kotlin.Int <116289638>)

✓ an element which needs:
     » ▶ its.definedIn(MockTest.kt:46):
         ■ to equal: "Mark" <574921197>
     » ▶ its.definedIn(MockTest.kt:47):
         ■ to equal: 13 (kotlin.Int <453296736>)
  following elements were mismatched:
     • User(#3) (User <1041341319>)
```

```
I expected subject: [User(#3), User(#4)]
                                               (java.util.Arrays.ArrayList <315506631>)
```

```
x an element which needs:
```

```
following elements were mismatched:
  • User(#3) (User <1041341319>)
```

```
» ▶ its.definedIn(MockTest.kt:42):
   ■ to equal: "Pash" <2026086646>
```

```
(java.util.Arrays.ArrayList <315506631>)
I expected subject: [User(#3), User(#4)]
♦ to contain only, in any order:
 x an element which needs:
     » ▶ its.definedIn(MockTest.kt:42):
         ■ to equal: "Pash"
                                 <2026086646>
     » ▶ its.definedIn(MockTest.kt:43):
         ■ to equal: 36 (kotlin.Int <116289638>)

✓ an element which needs:
     » ▶ its.definedIn(MockTest.kt:46):
         ■ to equal: "Mark" <574921197>
     » ▶ its.definedIn(MockTest.kt:47):
         ■ to equal: 13 (kotlin.Int <453296736>)
  following elements were mismatched:
     • User(#3) (User <1041341319>)
```

```
I expected subject: [User(#3), User(#4)]
                                             (java.util.Arrays.ArrayList <315506631>)
♦ to contain only, in any order:
 x an element which needs:
     » ▶ its.definedIn(MockTest.kt:42):
         ■ to equal: "Pash"
                                 <2026086646>
     » ▶ its.definedIn(MockTest.kt:43):
         ■ to equal: 36 (kotlin.Int <116289638>)

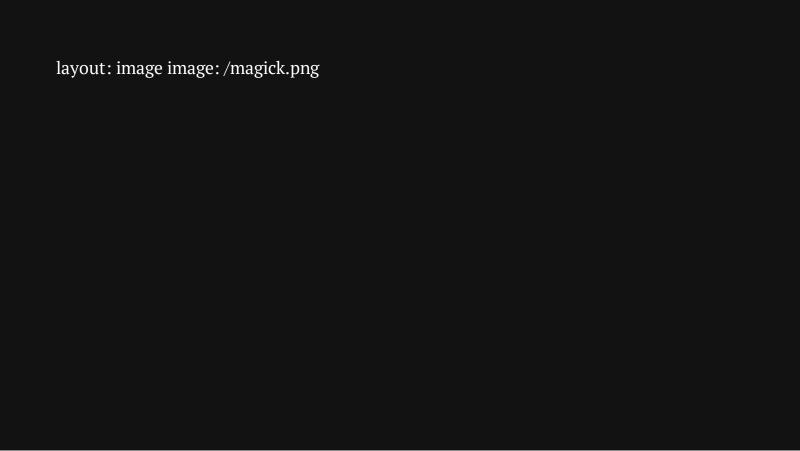
✓ an element which needs:
     » ▶ its.definedIn(MockTest.kt:46):
         ■ to equal: "Mark" <574921197>
     » ▶ its.definedIn(MockTest.kt:47):
         ■ to equal: 13 (kotlin.Int <453296736>)
  following elements were mismatched:
     • User(#3) (User <1041341319>)
```

- Readable as a text
- Shows the errors

Dessert

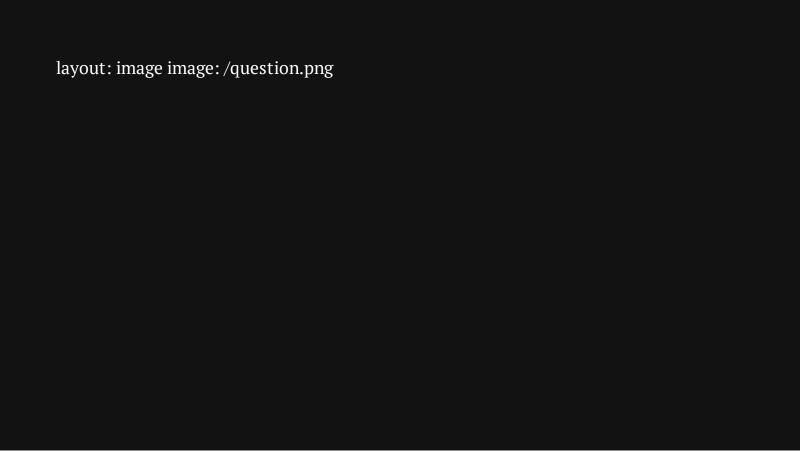
Now, when we saw different nice DSL-using capabilities...

```
ideaTest<App>("Create Kafka connection") {
    showBDTPanel {
        createBaseConnection("Kafka") {
            fillCredentialsForKafka(connectionName)
            checkSuccessfulMessage()
            ok()
        }
        checkMonitoringConnection(monitoringToolWindow)
    }
}
```



How does it work? Magick!

Dive deeper



What did we learn?

What did we learn?

- 1. Tests should be readable
- 2. Testing is not "just unit tests" sometimes
- 3. Kotest gives good structure to the tests.
- 4. Kotest has built-in property tests
- 5. Atrium gives us assertions DSL

layout: cover	· background: /1	monk.png		
		, 3		

Thank you! Time for questions 😂

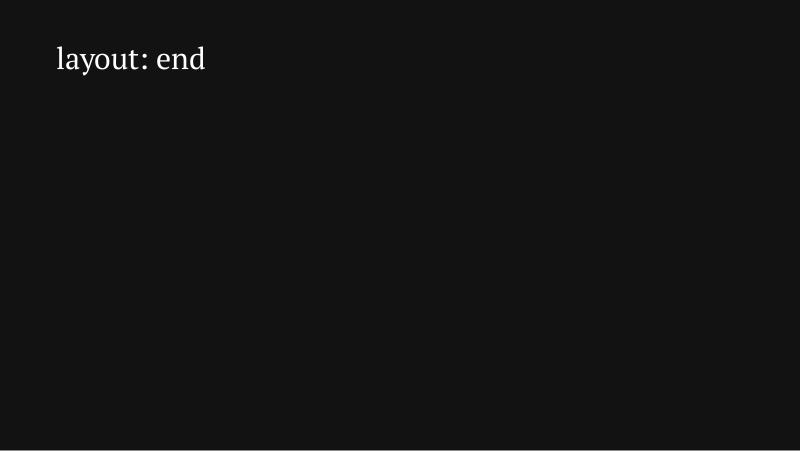
y asm0di0

me@asm0dey.site

a @asm0dey.site

√ o in **√** asm0dey

thttps://linktr.ee/asm0dey



```
1 ideaTest<App>("Create Kafka connection") {
```

```
1 showBDTPanel {
```

```
1 createBaseConnection("Kafka") {
```

```
fun createBaseConnection(typeOfConnection: String, block: ConnectionDialog.() -> Unit) {
   step("Open connection dialog") {
      if (app.isKafka()) {
        frame.find<ComponentFixture>(byXpath("//div[contains(@tooltiptext.key, 'toolwing clickTypeOfConnection(typeOfConnection, block)
        pause(5000)
```

```
1 createBaseConnection("Kafka") {
```

layout: center And this is the abyss!

But you can do the same!

Go back to summary

