Kotlin Goodies for Testing

Pasha Finkelshteyn, BellSoft

layout: two-cols

`whoami`

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

layout: image image: /glitched-slide.webp

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

clicks: 3

Cucumber

```
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?

clicks: 3

Cucumber

```
1  Feature: An example
2    Scenario: The example
3      Given I have a username "pasha"
4      When I type username in field "x"
5      And Press OK button
6      Then I get an error "Enter password"
```

This line might intersect with **any** test

clicks: 3

Cucumber

```
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

clicks: 3

Cucumber

```
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:

How can we do it in Kotlin?

Kotest: https://kotest.io

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

```
3           `when`("I sit on it") {
4               then("I should be able to fly") {
5                   // test code
6               }
7           }
8           `when`("I throw it away") {
9               then("it should come back") {
10                  // test code
11              }
12          }
13      }
```

How can we do it in Kotlin?

Kotest: https://kotest.io

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

```
class MyTests : BehaviorSpec({
    given("a broomstick") {
        `when`("I sit on it") {
            then("I should be able to fly") {
                // test code
            }
        }
        `when`("I throw it away") {
            then("it should come back") {
                // test code
            }
```

How can we do it in Kotlin?

Kotest: https://kotest.io

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

```
class MyTests : BehaviorSpec({
    given("a broomstick") {
        `when`("I sit on it") {
            then("I should be able to fly") {
                // test code
            }
        }
        `when`("I throw it away") {
            then("it should come back") {
                // test code
            }
```

How can we do it in Kotlin?

Kotest: https://kotest.io

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

```
3           `when`("I sit on it") {
4               then("I should be able to fly") {
5                   // test code
6               }
7           }
8           `when`("I throw it away") {
9               then("it should come back") {
10                  // test code
11              }
12          }
13      }
```

How can we do it in Kotlin?

Kotest: https://kotest.io

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

```
4                 then("I should be able to fly") {
5                     // test code
6                 }
7             }
8             `when`("I throw it away") {
9                 then("it should come back") {
10                    // test code
11                }
12            }
13        }
14    })
```

How can we do it in Kotlin?

Kotest: https://kotest.io

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

```
4                 then("I should be able to fly") {
5                     // test code
6                 }
7             }
8             `when`("I throw it away") {
9                 then("it should come back") {
10                    // test code
11                }
12            }
13        }
14    })
```

How can we do it in Kotlin?

Kotest: https://kotest.io

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

```
3           `when`("I sit on it") {
4               then("I should be able to fly") {
5                   // test code
6               }
7           }
8           `when`("I throw it away") {
9               then("it should come back") {
10                  // test code
11              }
12          }
13      }
```

Another flavor of Kotest

Kotest: https://kotest.io

```
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 */ }
        }
    }
})
```

Another flavor of Kotest

Kotest: https://kotest.io

```
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 */ }
        }
    }
})
```

Another flavor of Kotest

Kotest: https://kotest.io

```
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 */ }
        }
    }
})
```

Another flavor of Kotest

Kotest: https://kotest.io

```
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 */ }
        }
    }
})
```

Another flavor of Kotest

Kotest: https://kotest.io

```
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 */ }
        }
    }
})
```

Migrating from JUnit?

```
1   class AnnotationSpecExample : AnnotationSpec() {
2       @BeforeEach
3       fun beforeTest() {
4           println("Before each test")
5       }
6       @Test
7       fun test1() {
8           1 shouldBe 1
9       }
10      @Test
11      fun test2() {
12          3 shouldBe 3
13      }
14  }
```

Migrating from JUnit?

```
1   class AnnotationSpecExample : AnnotationSpec() {
2       @BeforeEach
3       fun beforeTest() {
4           println("Before each test")
5       }
6       @Test
7       fun test1() {
8           1 shouldBe 1
9       }
10      @Test
11      fun test2() {
12          3 shouldBe 3
13      }
14  }
```

Migrating from JUnit?

```
1   class AnnotationSpecExample : AnnotationSpec() {
2       @BeforeEach
3       fun beforeTest() {
4           println("Before each test")
5       }
6       @Test
7       fun test1() {
8           1 shouldBe 1
9       }
10      @Test
11      fun test2() {
12          3 shouldBe 3
13      }
14  }
```

Migrating from JUnit?

```
class AnnotationSpecExample : AnnotationSpec() {
    @BeforeEach
    fun beforeTest() {
        println("Before each test")
    }
    @Test
    fun test1() {
        1 shouldBe 1
    }
    @Test
    fun test2() {
        3 shouldBe 3
    }
}
```

Migrating from JUnit?

```
1   class AnnotationSpecExample : AnnotationSpec() {
2       @BeforeEach
3       fun beforeTest() {
4           println("Before each test")
5       }
6       @Test
7       fun test1() {
8           1 shouldBe 1
9       }
10      @Test
11      fun test2() {
12          3 shouldBe 3
13      }
14  }
```

Migrating from JUnit?

```
1   class AnnotationSpecExample : AnnotationSpec() {
2       @BeforeEach
3       fun beforeTest() {
4           println("Before each test")
5       }
6       @Test
7       fun test1() {
8           1 shouldBe 1
9       }
10      @Test
11      fun test2() {
12          3 shouldBe 3
13      }
14  }
```

explanation `shouldBe` better

Kotest has its own assertions

explanation `shouldBe` better

Kotest has its own assertions

- kotest-assertions-core

explanation `shouldBe` better

Kotest has its own assertions

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

explanation `shouldBe` better

Kotest has its own assertions

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

explanation `shouldBe` better

Kotest has its own assertions

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

Core Assertions

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

Core Assertions

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

Core Assertions

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

Core Assertions

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

Core Assertions

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

JSON Assertions

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

Or even…

JSON Assertions

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

Or even…

Schema Validation

```
1   val addressSchema = jsonSchema {
2     obj {   // object is reserved, obj was chosen over jsonObject for brevity but could be cha
3       withProperty("street", required = true) { string() }
4       withProperty("zipCode", required = true) {
5         integer {
6           beEven() and beInRange(10000..99999)   // supports constructing a matcher that will
7         }
8       }
9       additionalProperties = false   // triggers failure if other properties are defined in ad
10    }
11  }
12
13  val personSchema = jsonSchema {
14    obj {
15      withProperty("name", required = true) { string() }
16      withProperty("address") { addressSchema() } // Schemas can re-use other schemas 🎉
```

Schema Validation

```
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 b
      }
    }
    additionalProperties = false   // triggers failure if other properties are defined in act
  }
}

val personSchema = jsonSchema {
  obj {
    withProperty("name", required = true) { string() }
    withProperty("address") { addressSchema() } // Schemas can re-use other schemas 🎉
```

Schema Validation

```
1   val addressSchema = jsonSchema {
2     obj {   // object is reserved, obj was chosen over jsonObject for brevity but could be cha
3       withProperty("street", required = true) { string() }
4       withProperty("zipCode", required = true) {
5         integer {
6           beEven() and beInRange(10000..99999)   // supports constructing a matcher that will
7         }
8       }
9       additionalProperties = false   // triggers failure if other properties are defined in ad
10    }
11  }
12
13  val personSchema = jsonSchema {
14    obj {
15      withProperty("name", required = true) { string() }
16      withProperty("address") { addressSchema() } // Schemas can re-use other schemas 🎉
```

Schema Validation

```
1   val addressSchema = jsonSchema {
2     obj {   // object is reserved, obj was chosen over jsonObject for brevity but could be char
3       withProperty("street", required = true) { string() }
4       withProperty("zipCode", required = true) {
5         integer {
6           beEven() and beInRange(10000..99999)   // supports constructing a matcher that will b
7         }
8       }
9       additionalProperties = false   // triggers failure if other properties are defined in act
10    }
11  }
12
13  val personSchema = jsonSchema {
14    obj {
15      withProperty("name", required = true) { string() }
16      withProperty("address") { addressSchema() } // Schemas can re-use other schemas 🎉
```

Schema Validation

```
1   val addressSchema = jsonSchema {
2     obj {   // object is reserved, obj was chosen over jsonObject for brevity but could be char
3       withProperty("street", required = true) { string() }
4       withProperty("zipCode", required = true) {
5         integer {
6           beEven() and beInRange(10000..99999)   // supports constructing a matcher that will b
7         }
8       }
9       additionalProperties = false   // triggers failure if other properties are defined in act
10    }
11  }
12
13  val personSchema = jsonSchema {
14    obj {
15      withProperty("name", required = true) { string() }
16      withProperty("address") { addressSchema() } // Schemas can re-use other schemas 🎉
```

Schema Validation

```
1   val addressSchema = jsonSchema {
2     obj {   // object is reserved, obj was chosen over jsonObject for brevity but could be char
3       withProperty("street", required = true) { string() }
4       withProperty("zipCode", required = true) {
5         integer {
6           beEven() and beInRange(10000..99999)   // supports constructing a matcher that will b
7         }
8       }
9       additionalProperties = false   // triggers failure if other properties are defined in act
10    }
11  }
12
13  val personSchema = jsonSchema {
14    obj {
15      withProperty("name", required = true) { string() }
16      withProperty("address") { addressSchema() } // Schemas can re-use other schemas 🎉
```

Schema Validation

```
1   val addressSchema = jsonSchema {
2     obj {   // object is reserved, obj was chosen over jsonObject for brevity but could be char
3       withProperty("street", required = true) { string() }
4       withProperty("zipCode", required = true) {
5         integer {
6           beEven() and beInRange(10000..99999)   // supports constructing a matcher that will b
7         }
8       }
9       additionalProperties = false   // triggers failure if other properties are defined in act
10    }
11  }
12
13  val personSchema = jsonSchema {
14    obj {
15      withProperty("name", required = true) { string() }
16      withProperty("address") { addressSchema() } // Schemas can re-use other schemas 🎉
17    }
```

Schema Validation

```
5          integer {
6            beEven() and beInRange(10000..99999)   // supports constructing a matcher that will
7          }
8        }
9        additionalProperties = false   // triggers failure if other properties are defined in ad
10     }
11   }
12
13   val personSchema = jsonSchema {
14     obj {
15       withProperty("name", required = true) { string() }
16       withProperty("address") { addressSchema() } // Schemas can re-use other schemas 🎉
17     }
18   }
19
20   "{}" shouldMatchSchema personSchema
21
```

Schema Validation

```
7          }
8        }
9        additionalProperties = false   // triggers failure if other properties are defined in ad
10     }
11   }
12
13   val personSchema = jsonSchema {
14     obj {
15       withProperty("name", required = true) { string() }
16       withProperty("address") { addressSchema() } // Schemas can re-use other schemas 🎉
17     }
18   }
19
20   "{}" shouldMatchSchema personSchema
21
22   // fails with:
23   // $.name => Expected string, but was undefined
```

Schema Validation

```
8         }
9         additionalProperties = false   // triggers failure if other properties are defined in ac
10      }
11    }
12
13    val personSchema = jsonSchema {
14      obj {
15        withProperty("name", required = true) { string() }
16        withProperty("address") { addressSchema() } // Schemas can re-use other schemas 🎉
17      }
18    }
19
20    "{}" shouldMatchSchema personSchema
21
22    // fails with:
23    // $.name => Expected string, but was undefined
24
```

Schema Validation

```
11  }
12
13  val personSchema = jsonSchema {
14    obj {
15      withProperty("name", required = true) { string() }
16      withProperty("address") { addressSchema() } // Schemas can re-use other schemas 🎉
17    }
18  }
19
20  "{}" shouldMatchSchema personSchema
21
22  // fails with:
23  // $.name => Expected string, but was undefined
24
25  """ { "name": "Emil", "age": 34 } """
26  // Passes, since address isn't required and `additionalProperties` are allowed
```

Schema Validation

```
11  }
12
13  val personSchema = jsonSchema {
14    obj {
15      withProperty("name", required = true) { string() }
16      withProperty("address") { addressSchema() } // Schemas can re-use other schemas 🎉
17    }
18  }
19
20  "{}" shouldMatchSchema personSchema
21
22  // fails with:
23  // $.name => Expected string, but was undefined
24
25  """ { "name": "Emil", "age": 34 } """
26  // Passes, since address isn't required and `additionalProperties` are allowed
```

Jsoup

```
element.shouldHaveChildWithTag(tag)
element.shouldHaveText(text)
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

Parametrized tests? Java

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

Parametrized tests? Java

```
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   }
```

Parametrized tests? Java

```
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   }
```

Parametrized tests? Java

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

Parametrized tests? Java

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

Parametrized tests? Java

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

Parametrized tests? Java

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

Parametrized tests? Java

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

Parametrized tests? Java

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

Parametrized tests? Java

```java
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?

Parametrized tests? Java

```java
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

Parametrized tests? Java

```java
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)

Parametrized tests? Java

```java
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)

```kotlin
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)

```kotlin
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)

```kotlin
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)

```kotlin
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)

```kotlin
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)

```
1   context("Pythag triples tests") {
2     withData(
3       PythagTriple(3, 4, 5),
4       PythagTriple(6, 8, 10),
5       PythagTriple(8, 15, 17),
6       PythagTriple(7, 24, 25)
7     ) { (a, b, c) ->
8       isPythagTriple(a, b, c) shouldBe true
9     }
10  }
```

layout: image image: /confused.png

What if

I need to test things I don’t know I need to test?

clicks: 5

Never write your own hashing (unless...)

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

clicks: 5

Never write your own hashing (unless...)

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

clicks: 5

Never write your own hashing (unless...)

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

clicks: 5

Never write your own hashing (unless...)

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

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...

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

layout: image image: /lens.png

Time to investigate!

Property tests!

To test things we can not even *imagine*

layout: statement

Demo

Another example

```
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)
}
```

Another example

```
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)
}
```

Another example

```
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)
}
```

Another example

```
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)
}
```

Another example

```
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)
}
```

Another example

```
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)
}
```

Another example

```
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)
}
```

Another example

```
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)
}
```

layout: image image: /explode.png

Error!

```
Property test failed for inputs

0) "▫Ʌ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫Ẩ▫▫▫▫▫Ở▫▫▫▫▫▫▫▫▫▫ŋ▨▨▨▨"

Caused by io.kotest.assertions.AssertionFailedError: expected:<"90b23a970ed1b799850a24140ae8
```

Wow, it was a complex sample!

But there is more…

Error!

```
Property test failed for inputs

0) "▫Ƛ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫Ấ▫▫▫▫▫Ở▫▫▫▫▫▫▫▫▫▫ŋ▨▨▨▨"

Caused by io.kotest.assertions.AssertionFailedError: expected:<"90b23a970ed1b799850a24140ae8
```

Wow, it was a complex sample!

But there is more…

Error!

```
Property test failed for inputs

0) "▫Л▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫Ấ▫▫▫▫▫Ở▫▫▫▫▫▫▫▫▫▫ŋ▨▨▨▨"

Caused by io.kotest.assertions.AssertionFailedError: expected:<"90b23a970ed1b799850a24140ae8
```

Wow, it was a complex sample!

But there is more…

Error!

```
Property test failed for inputs

0) "▫Ƛ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫Ấ▫▫▫▫▫Ở▫▫▫▫▫▫▫▫▫▫ŋ▨▨▨▨"

Caused by io.kotest.assertions.AssertionFailedError: expected:<"90b23a970ed1b799850a24140ae8
```

Wow, it was a complex sample!

But there is more…

Error!

```
Attempting to shrink arg "▫Ʌ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫Ấ▫▫▫▫▫Ở▫▫▫▫▫▫▫▫▫▫ŋ⊠⊠⊠⊠"
Shrink #1: "▫Ʌ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫⊠⊠⊠" fail
Shrink #2: "▫Ʌ▫▫▫▫▫▫▫▫▫⊠*" fail
Shrink #3: "▫Ʌ▫▫▫▫⊠" fail
Shrink #4: "▫Ʌ⊠" fail
Shrink #5: "▫Ʌ" fail
Shrink #6: "⊠" fail
Shrink #7: <empty string> pass
Shrink result (after 7 shrinks) => "⊠"
```

And there is more!

Error!

```
Attempting to shrink arg "▫ℳ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫Ấ▫▫▫▫▫Ở▫▫▫▫▫▫▫▫▫ŋ⊠⊠⊠⊠"
Shrink #1: "▫ℳ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫⊠⊠⊠" fail
Shrink #2: "▫ℳ▫▫▫▫▫▫▫▫▫⊠∗" fail
Shrink #3: "▫ℳ▫▫▫⊠" fail
Shrink #4: "▫ℳ⊠" fail
Shrink #5: "▫ℳ" fail
Shrink #6: "⊠" fail
Shrink #7: <empty string> pass
Shrink result (after 7 shrinks) => "⊠"
```

And there is more!

Error!

```
Attempting to shrink arg "▫Ʌ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫Ấ▫▫▫▫▫Ở▫▫▫▫▫▫▫▫▫▫ŋ⊠⊠⊠⊠"
Shrink #1: "▫Ʌ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫⊠⊠⊠" fail
Shrink #2: "▫Ʌ▫▫▫▫▫▫▫▫⊠*" fail
Shrink #3: "▫Ʌ▫▫▫⊠" fail
Shrink #4: "▫Ʌ⊠" fail
Shrink #5: "▫Ʌ" fail
Shrink #6: "⊠" fail
Shrink #7: <empty string> pass
Shrink result (after 7 shrinks) => "⊠"
```

And there is more!

Error!

```
Attempting to shrink arg "▫Ʌ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫Ẩ▫▫▫▫▫Ở▫▫▫▫▫▫▫▫▫▫ŋ⊠⊠⊠⊠"
Shrink #1: "▫Ʌ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫⊠⊠⊠" fail
Shrink #2: "▫Ʌ▫▫▫▫▫▫▫▫▫⊠*" fail
Shrink #3: "▫Ʌ▫▫▫▫⊠" fail
Shrink #4: "▫Ʌ⊠" fail
Shrink #5: "▫Ʌ" fail
Shrink #6: "⊠" fail
Shrink #7: <empty string> pass
Shrink result (after 7 shrinks) => "⊠"
```

And there is more!

Error!

```
Attempting to shrink arg "▫Ʌ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫Ẩ▫▫▫▫▫Ở▫▫▫▫▫▫▫▫▫▫ŋ⊠⊠⊠⊠"
Shrink #1: "▫Ʌ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫⊠⊠⊠" fail
Shrink #2: "▫Ʌ▫▫▫▫▫▫▫▫▫⊠*" fail
Shrink #3: "▫Ʌ▫▫▫▫⊠" fail
Shrink #4: "▫Ʌ⊠" fail
Shrink #5: "▫Ʌ" fail
Shrink #6: "⊠" fail
Shrink #7: <empty string> pass
Shrink result (after 7 shrinks) => "⊠"
```

And there is more!

Error!

```
Attempting to shrink arg "□Ꮑ□□□□□□□□□□□□□□□□□□□□□□□□Ẩ□□□□□Ở□□□□□□□□□ŋ̈⊠⊠⊠⊠"
Shrink #1: "□Ꮑ□□□□□□□□□□□□□□□□□□□⊠⊠⊠" fail
Shrink #2: "□Ꮑ□□□□□□□□⊠✷" fail
Shrink #3: "□Ꮑ□□□⊠" fail
Shrink #4: "□Ꮑ⊠" fail
Shrink #5: "□Ꮑ" fail
Shrink #6: "⊠" fail
Shrink #7: <empty string> pass
Shrink result (after 7 shrinks) => "⊠"
```

And there is more!

Error!

```
Attempting to shrink arg "▫ℳ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫Ẩ▫▫▫▫▫Ở▫▫▫▫▫▫▫▫▫▫ḕ⊠⊠⊠⊠"
Shrink #1: "▫ℳ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫⊠⊠⊠" fail
Shrink #2: "▫ℳ▫▫▫▫▫▫▫▫▫⊠∗" fail
Shrink #3: "▫ℳ▫▫▫▫⊠" fail
Shrink #4: "▫ℳ⊠" fail
Shrink #5: "▫ℳ" fail
Shrink #6: "⊠" fail
Shrink #7: <empty string> pass
Shrink result (after 7 shrinks) => "⊠"
```

And there is more!

Error!

```
Attempting to shrink arg "▫Ʌ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫Ẩ▫▫▫▫▫Ở▫▫▫▫▫▫▫▫▫▫ŋ⊠⊠⊠⊠"
Shrink #1: "▫Ʌ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫⊠⊠⊠" fail
Shrink #2: "▫Ʌ▫▫▫▫▫▫▫▫▫⊠*" fail
Shrink #3: "▫Ʌ▫▫▫⊠" fail
Shrink #4: "▫Ʌ⊠" fail
Shrink #5: "▫Ʌ" fail
Shrink #6: "⊠" fail
Shrink #7: <empty string> pass
Shrink result (after 7 shrinks) => "⊠"
```

And there is more!

Error!

```
Attempting to shrink arg "▫Ʌ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫Ẩ▫▫▫▫▫Ở▫▫▫▫▫▫▫▫▫▫ŋ⊠⊠⊠⊠"
Shrink #1: "▫Ʌ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫⊠⊠⊠" fail
Shrink #2: "▫Ʌ▫▫▫▫▫▫▫▫▫⊠*" fail
Shrink #3: "▫Ʌ▫▫▫⊠" fail
Shrink #4: "▫Ʌ⊠" fail
Shrink #5: "▫Ʌ" fail
Shrink #6: "⊠" fail
Shrink #7: <empty string> pass
Shrink result (after 7 shrinks) => "⊠"
```

And there is more!

Error!

```
Attempting to shrink arg "▫ℳ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫Ẩ▫▫▫▫▫Ở▫▫▫▫▫▫▫▫▫▫ḕ⊠⊠⊠⊠"
Shrink #1: "▫ℳ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫⊠⊠⊠" fail
Shrink #2: "▫ℳ▫▫▫▫▫▫▫▫▫⊠∗" fail
Shrink #3: "▫ℳ▫▫▫▫⊠" fail
Shrink #4: "▫ℳ⊠" fail
Shrink #5: "▫ℳ" fail
Shrink #6: "⊠" fail
Shrink #7: <empty string> pass
Shrink result (after 7 shrinks) => "⊠"
```

And there is more!

Error!

```
Property failed after 1 attempts

  Arg 0: "⊠" (shrunk from ▫Λ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫Ấ▫▫▫▫▫Ở▫▫▫▫▫▫▫▫▫▫ḿ▫▫▫▫)

Repeat this test by using seed 4919313423746630853
```

Error!

```
Property failed after 1 attempts

  Arg 0: "⊠" (shrunk from ▫Ƙ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫Ẩ▫▫▫▫▫Ở▫▫▫▫▫▫▫▫▫▫ḿ▫▫▫▫)

Repeat this test by using seed 4919313423746630853
```

Error!

```
Property failed after 1 attempts

  Arg 0: "⊠" (shrunk from ▫Λ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫Ấ▫▫▫▫▫Ở▫▫▫▫▫▫▫▫▫ỊỊ▫▫▫▫)

Repeat this test by using seed 4919313423746630853
```

Error!

```
Property failed after 1 attempts

  Arg 0: "⊠" (shrunk from ▫Ƙ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫Ẩ▫▫▫▫▫Ở▫▫▫▫▫▫▫▫▫▫ḿ▫▫▫▫)

Repeat this test by using seed 4919313423746630853
```

Repeat? Yes!

```
checkAll(
    PropTestConfig(seed = 4919313423746630853),
    Arb.string(
        codepoints = Arb.of(
            ('a'.code..Char.MAX_HIGH_SURROGATE.code)
                .map(::Codepoint)
        ),
    )
) { input ->
    MD5.compute(input) shouldBe libMd5(input)
}
```

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

Repeat? Yes!

```
checkAll(
    PropTestConfig(seed = 4919313423746630853),
    Arb.string(
        codepoints = Arb.of(
            ('a'.code..Char.MAX_HIGH_SURROGATE.code)
                .map(::Codepoint)
        ),
    )
) { input ->
    MD5.compute(input) shouldBe libMd5(input)
}
```

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

layout: image image: /mockingbird.png

Time to mock!

Disclaimer

I believe one should mock, not fake. Because

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

Time to mock

Setup

```
1  class UserService(val repo: UserRepository) {
2      fun getAllUsers() = repo.allUsers()
3  }
4
5  class UserRepository {
6      fun allUsers(): List<User> { TODO("demo") }
7  }
8
9  class User(val name: String, val age: Int)
```

Time to mock

Setup

```
1  class UserService(val repo: UserRepository) {
2      fun getAllUsers() = repo.allUsers()
3  }
4
5  class UserRepository {
6      fun allUsers(): List<User> { TODO("demo") }
7  }
8
9  class User(val name: String, val age: Int)
```

Time to mock

Setup

```
1  class UserService(val repo: UserRepository) {
2      fun getAllUsers() = repo.allUsers()
3  }
4
5  class UserRepository {
6      fun allUsers(): List<User> { TODO("demo") }
7  }
8
9  class User(val name: String, val age: Int)
```

Time to mock

Setup

```
1  class UserService(val repo: UserRepository) {
2      fun getAllUsers() = repo.allUsers()
3  }
4
5  class UserRepository {
6      fun allUsers(): List<User> { TODO("demo") }
7  }
8
9  class User(val name: String, val age: Int)
```

Time to mock

Setup

```
1  class UserService(val repo: UserRepository) {
2      fun getAllUsers() = repo.allUsers()
3  }
4
5  class UserRepository {
6      fun allUsers(): List<User> { TODO("demo") }
7  }
8
9  class User(val name: String, val age: Int)
```

Time to mock

Setup

```
1  class UserService(val repo: UserRepository) {
2      fun getAllUsers() = repo.allUsers()
3  }
4
5  class UserRepository {
6      fun allUsers(): List<User> { TODO("demo") }
7  }
8
9  class User(val name: String, val age: Int)
```

Time to mock

Setup

```
1  class UserService(val repo: UserRepository) {
2      fun getAllUsers() = repo.allUsers()
3  }
4
5  class UserRepository {
6      fun allUsers(): List<User> { TODO("demo") }
7  }
8
9  class User(val name: String, val age: Int)
```

Time to mock

Setup

```
1  class UserService(val repo: UserRepository) {
2      fun getAllUsers() = repo.allUsers()
3  }
4
5  class UserRepository {
6      fun allUsers(): List<User> { TODO("demo") }
7  }
8
9  class User(val name: String, val age: Int)
```

How the test looks in Java

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

// Create some sample user data
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);

// Create an instance of the UserService with the mock repository
UserService userService = new UserService(mockRepository);

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

// Verify the result
Assertions.assertEquals(users, result);
```

How the test looks in Java

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

// Create some sample user data
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);

// Create an instance of the UserService with the mock repository
UserService userService = new UserService(mockRepository);

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

// Verify the result
Assertions.assertEquals(users, result);
```

How the test looks in Java

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

// Create some sample user data
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);

// Create an instance of the UserService with the mock repository
UserService userService = new UserService(mockRepository);

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

// Verify the result
Assertions.assertEquals(users, result);
```

How the test looks in Java

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

// Create some sample user data
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);

// Create an instance of the UserService with the mock repository
UserService userService = new UserService(mockRepository);

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

// Verify the result
Assertions.assertEquals(users, result);
```

How the test looks in Java

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

// Create some sample user data
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);

// Create an instance of the UserService with the mock repository
UserService userService = new UserService(mockRepository);

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

// Verify the result
Assertions.assertEquals(users, result);
```

How the test looks in Java

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

// Create some sample user data
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);

// Create an instance of the UserService with the mock repository
UserService userService = new UserService(mockRepository);

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

// Verify the result
Assertions.assertEquals(users, result);
```

How the test looks in Java

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

// Create some sample user data
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);

// Create an instance of the UserService with the mock repository
UserService userService = new UserService(mockRepository);

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

// Verify the result
Assertions.assertEquals(users, result);
```

Time to mock! (MockK edition)

```
1  val users = listOf(User("Alice", 25), User("Bob", 30), User("Charlie", 35))
2  val repo = mockk<UserRepository> {
3      every { allUsers() } returns users
4  }
5  val service = UserService(repo)
6  service.getAllUsers() shouldBe users
```

Time to mock! (MockK edition)

```
1   val users = listOf(User("Alice", 25), User("Bob", 30), User("Charlie", 35))
2   val repo = mockk<UserRepository> {
3       every { allUsers() } returns users
4   }
5   val service = UserService(repo)
6   service.getAllUsers() shouldBe users
```

Time to mock! (MockK edition)

```
1  val users = listOf(User("Alice", 25), User("Bob", 30), User("Charlie", 35))
2  val repo = mockk<UserRepository> {
3      every { allUsers() } returns users
4  }
5  val service = UserService(repo)
6  service.getAllUsers() shouldBe users
```

Time to mock! (MockK edition)

```
1  val users = listOf(User("Alice", 25), User("Bob", 30), User("Charlie", 35))
2  val repo = mockk<UserRepository> {
3      every { allUsers() } returns users
4  }
5  val service = UserService(repo)
6  service.getAllUsers() shouldBe users
```

Time to mock! (MockK edition)

```
1  val users = listOf(User("Alice", 25), User("Bob", 30), User("Charlie", 35))
2  val repo = mockk<UserRepository> {
3      every { allUsers() } returns users
4  }
5  val service = UserService(repo)
6  service.getAllUsers() shouldBe users
```

Time to mock! (MockK edition)

```
1   val users = listOf(User("Alice", 25), User("Bob", 30), User("Charlie", 35))
2   val repo = mockk<UserRepository> {
3       every { allUsers() } returns users
4   }
5   val service = UserService(repo)
6   service.getAllUsers() shouldBe users
```

Time to mock! (MockK edition)

```
1   val users = listOf(User("Alice", 25), User("Bob", 30), User("Charlie", 35))
2   val repo = mockk<UserRepository> {
3       every { allUsers() } returns users
4   }
5   val service = UserService(repo)
6   service.getAllUsers() shouldBe users
```

Nice DSL

Code describing `UserRepository` is inside `{}`

What if I want to mock users too?

Time to mock users! (mockk edition)

```
1   val repo = mockk<UserRepository> {
2       every { allUsers() } returns listOf(
3           mockk {
4               every { name } returns "Pasha"
5           },
6           mockk {
7               every { name } returns "Alexandra"
8           },
9       )
10  }
11  val service = UserService(repo)
12  service.getAllUsers()[1].name shouldBe "Alexandra"
```

Time to mock users! (mockk edition)

```
1   val repo = mockk<UserRepository> {
2       every { allUsers() } returns listOf(
3           mockk {
4               every { name } returns "Pasha"
5           },
6           mockk {
7               every { name } returns "Alexandra"
8           },
9       )
10  }
11  val service = UserService(repo)
12  service.getAllUsers()[1].name shouldBe "Alexandra"
```

Time to mock users! (mockk edition)

```
1   val repo = mockk<UserRepository> {
2       every { allUsers() } returns listOf(
3           mockk {
4               every { name } returns "Pasha"
5           },
6           mockk {
7               every { name } returns "Alexandra"
8           },
9       )
10  }
11  val service = UserService(repo)
12  service.getAllUsers()[1].name shouldBe "Alexandra"
```

Time to mock users! (mockk edition)

```
1   val repo = mockk<UserRepository> {
2       every { allUsers() } returns listOf(
3           mockk {
4               every { name } returns "Pasha"
5           },
6           mockk {
7               every { name } returns "Alexandra"
8           },
9       )
10  }
11  val service = UserService(repo)
12  service.getAllUsers()[1].name shouldBe "Alexandra"
```

Time to mock users! (mockk edition)

```
1   val repo = mockk<UserRepository> {
2       every { allUsers() } returns listOf(
3           mockk {
4               every { name } returns "Pasha"
5           },
6           mockk {
7               every { name } returns "Alexandra"
8           },
9       )
10  }
11  val service = UserService(repo)
12  service.getAllUsers()[1].name shouldBe "Alexandra"
```

Time to mock users! (mockk edition)

```
1   val repo = mockk<UserRepository> {
2       every { allUsers() } returns listOf(
3           mockk {
4               every { name } returns "Pasha"
5           },
6           mockk {
7               every { name } returns "Alexandra"
8           },
9       )
10  }
11  val service = UserService(repo)
12  service.getAllUsers()[1].name shouldBe "Alexandra"
```

Time to mock users! (mockk edition)

```
1   val repo = mockk<UserRepository> {
2       every { allUsers() } returns listOf(
3           mockk {
4               every { name } returns "Pasha"
5           },
6           mockk {
7               every { name } returns "Alexandra"
8           },
9       )
10  }
11  val service = UserService(repo)
12  service.getAllUsers()[1].name shouldBe "Alexandra"
```

Users are described inside `User` mocks!

Atrium expectations!

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
    }
)
```

Atrium expectations!

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
    }
)
```

Atrium expectations!

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

Atrium expectations!

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

Atrium expectations!

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

Error report

```
I expected subject: [User(#3), User(#4)]        (java.util.Arrays.ArrayList <315506631>)
◆ to contain only, in any order:
  ✘ 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>)
```

Error report

```
I expected subject: [User(#3), User(#4)]        (java.util.Arrays.ArrayList <315506631>)
◆ to contain only, in any order:
  ✘ 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>)
```

Error report

```
I expected subject: [User(#3), User(#4)]        (java.util.Arrays.ArrayList <315506631>)
◆ to contain only, in any order:
  ✘ 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>)
```

Error report

```
I expected subject: [User(#3), User(#4)]        (java.util.Arrays.ArrayList <315506631>)
◆ to contain only, in any order:
  ✘ 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>)
```

Error report

```
I expected subject: [User(#3), User(#4)]        (java.util.Arrays.ArrayList <315506631>)
◆ to contain only, in any order:
  ✘ 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>)
```

Error report

```
I expected subject: [User(#3), User(#4)]        (java.util.Arrays.ArrayList <315506631>)
◆ to contain only, in any order:
  ✘ 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>)
```

Error report

```
I expected subject: [User(#3), User(#4)]        (java.util.Arrays.ArrayList <315506631>)
◆ to contain only, in any order:
  ✘ 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)
    }
}
```

layout: image image: /magick.png

How does it work? Magick!

Dive deeper

layout: image image: /question.png

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: /monk.png

Thank you! Time for questions 😄

asm0di0

me@asm0dey.site

@asm0dey.site

asm0dey

https://linktr.ee/asm0dey

layout: end

Deeper dive into JetBrains tests

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

really means:

```
fun <T : App> Group.ideaTest(title: String,
                             enabled: Boolean = true,
                             isMuted: Boolean = false,
                             testBody: T.(ctx: IdeaNodeContext<T>) -> Unit) {
    IdeaNodeContext<T>(scenarioContext).apply {
        test(title, enabled, isMuted) {
            app.testBody(this)
        }
    }
}
```

Deeper dive into JetBrains tests

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

really means:

```
fun <T : App> Group.ideaTest(title: String,
                             enabled: Boolean = true,
                             isMuted: Boolean = false,
                             testBody: T.(ctx: IdeaNodeContext<T>) -> Unit) {
    IdeaNodeContext<T>(scenarioContext).apply {
        test(title, enabled, isMuted) {
            app.testBody(this)
        }
    }
}
```

Deeper dive into JetBrains tests

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

really means:

```
1   fun <T : App> Group.ideaTest(title: String,
2                                enabled: Boolean = true,
3                                isMuted: Boolean = false,
4                                testBody: T.(ctx: IdeaNodeContext<T>) -> Unit) {
5       IdeaNodeContext<T>(scenarioContext).apply {
6           test(title, enabled, isMuted) {
7               app.testBody(this)
8           }
9       }
10  }
```

Deeper dive into JetBrains tests

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

really means:

```
fun <T : App> Group.ideaTest(title: String,
                             enabled: Boolean = true,
                             isMuted: Boolean = false,
                             testBody: T.(ctx: IdeaNodeContext<T>) -> Unit) {
    IdeaNodeContext<T>(scenarioContext).apply {
        test(title, enabled, isMuted) {
            app.testBody(this)
        }
    }
}
```

Deeper dive into JetBrains tests

```
1        showBDTPanel {
```

really means:

```
1   fun App.showBDTPanel(block: BDTPanel.() -> Unit) {
2       val app = this
3       ideaFrame {
4           val frame = this
5           step("Open Big Data Tools Panel") {
6               openBDTPanel(app)
7               if (isKafka()) {
8                   try {
9                       container(byXpath("//div[@accessiblename='Kafka Tool Window' and @class=
10                          BDTPanel(app, frame, this).block()
11                      }
```

Deeper dive into JetBrains tests

```
1        showBDTPanel {
```

really means:

```
1   fun App.showBDTPanel(block: BDTPanel.() -> Unit) {
2       val app = this
3       ideaFrame {
4           val frame = this
5           step("Open Big Data Tools Panel") {
6               openBDTPanel(app)
7               if (isKafka()) {
8                   try {
9                       container(byXpath("//div[@accessiblename='Kafka Tool Window' and @class=
10                          BDTPanel(app, frame, this).block()
11                      }
```

Deeper dive into JetBrains tests

```
1        showBDTPanel {
```

really means:

```
1   fun App.showBDTPanel(block: BDTPanel.() -> Unit) {
2       val app = this
3       ideaFrame {
4           val frame = this
5           step("Open Big Data Tools Panel") {
6               openBDTPanel(app)
7               if (isKafka()) {
8                   try {
9                       container(byXpath("//div[@accessiblename='Kafka Tool Window' and @class=
10                          BDTPanel(app, frame, this).block()
11                      }
```

Deeper dive into JetBrains tests

```
1        showBDTPanel {
```

really means:

```
1   fun App.showBDTPanel(block: BDTPanel.() -> Unit) {
2       val app = this
3       ideaFrame {
4           val frame = this
5           step("Open Big Data Tools Panel") {
6               openBDTPanel(app)
7               if (isKafka()) {
8                   try {
9                       container(byXpath("//div[@accessiblename='Kafka Tool Window' and @class=
10                          BDTPanel(app, frame, this).block()
11                      }
```

Deeper dive into JetBrains tests

```
1        showBDTPanel {
```

really means:

```
1   fun App.showBDTPanel(block: BDTPanel.() -> Unit) {
2       val app = this
3       ideaFrame {
4           val frame = this
5           step("Open Big Data Tools Panel") {
6               openBDTPanel(app)
7               if (isKafka()) {
8                   try {
9                       container(byXpath("//div[@accessiblename='Kafka Tool Window' and @class=
10                          BDTPanel(app, frame, this).block()
11                      }
```

Deeper dive into JetBrains tests

```
1        showBDTPanel {
```

really means:

```
1   fun App.showBDTPanel(block: BDTPanel.() -> Unit) {
2       val app = this
3       ideaFrame {
4           val frame = this
5           step("Open Big Data Tools Panel") {
6               openBDTPanel(app)
7               if (isKafka()) {
8                   try {
9                       container(byXpath("//div[@accessiblename='Kafka Tool Window' and @class=
10                          BDTPanel(app, frame, this).block()
11                      }
```

Deeper dive into JetBrains tests

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

really means:

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

Deeper dive into JetBrains tests

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

really means:

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

Deeper dive into JetBrains tests

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

really means:

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

Deeper dive into JetBrains tests

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

really means:

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

Deeper dive into JetBrains tests

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

really means:

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

layout: center

And this is the abyss!

But you can do the same!

Go back to summary

layout: end