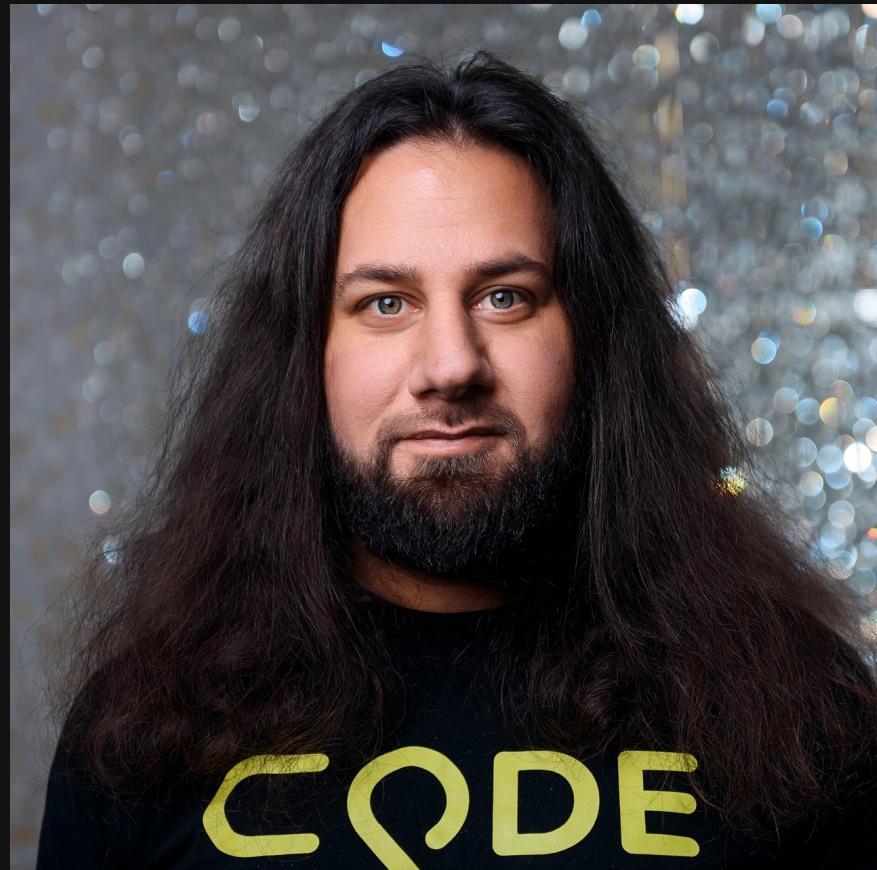


Kotlin Goodies for Testing

Pasha Finkelshteyn, 

`whoami`

-    developer
- Average  enjoyer
- Developer  at 
- Data engineer
- Speaker  and streamer 
(<https://twitch.tv/jetbrains> or /)



All programming activities have one common trait

They require testing!

Who are you?

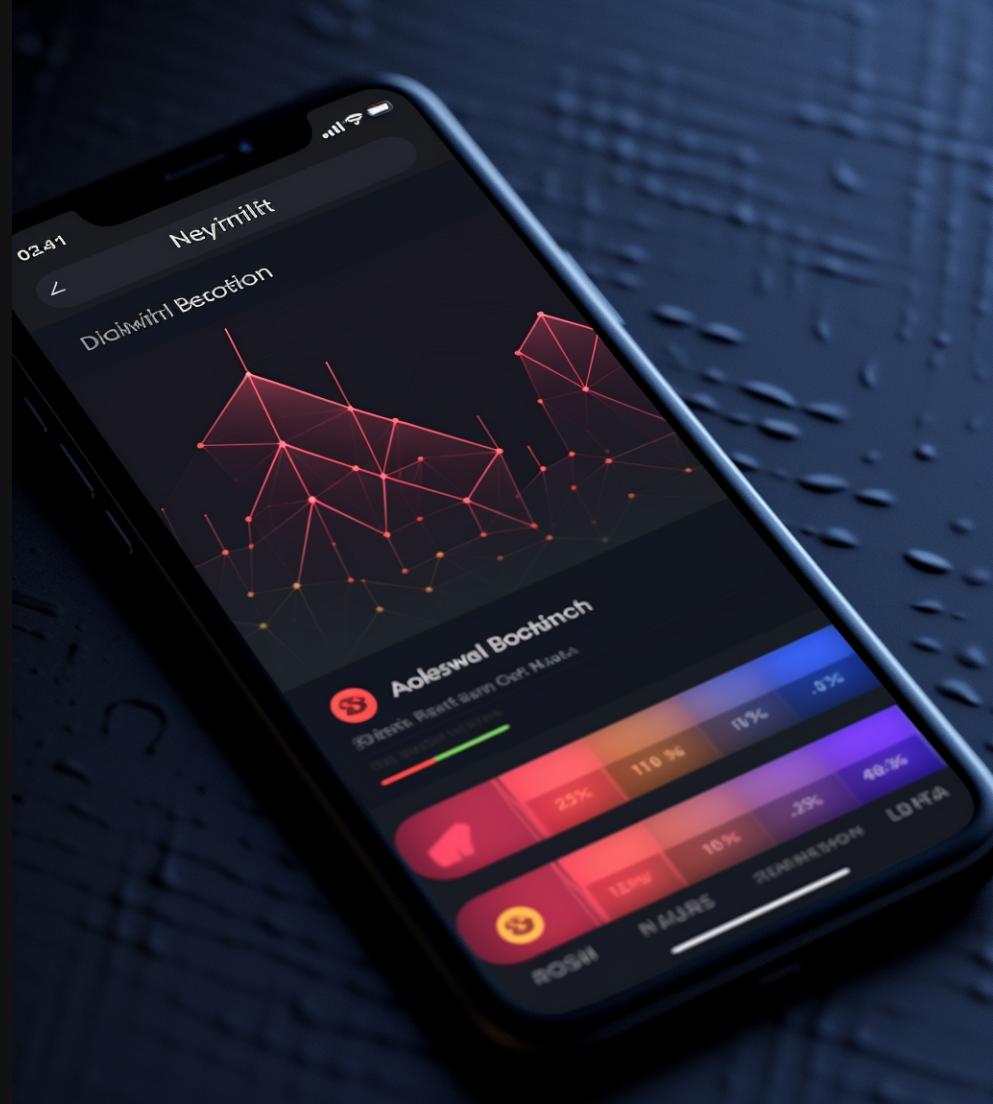
1. You write code/tests in Java or Kotlin
2. You wanna migrate to Kotlin, *or*
3. You wanna make your tests in Kotlin better, *or*
4. You wanna introduce Kotlin in your projects!

How are we doing it?

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

BDD

1. Given
2. When
3. Then



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"
```

What's the problem here?

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

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"
```

Actually **any** line can

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"
```

Actually **any** line can

Steps match by **name** O_o:

```
1 @And("Press OK button")
2 void press_ok_button(){/*TODO*/}
```

How can we do it in Kotlin?

Kotest: <https://kotest.io>

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

```
1  class MyTests : BehaviorSpec{  
2      given("a broomstick") {  
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     }  
14 }
```

Behavior Spec

Popular with people who like to write tests in the *BDD* style,
`BehaviorSpec` allows you to use `given`, `when`, `then`.

```
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

```
1  class MyTests : BehaviorSpec{  
2      given("a broomstick") {  
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     }  
14 }
```

Behavior Spec

Popular with people who like to write tests in the *BDD* style,
`BehaviorSpec` allows you to use `given`, `when`, `then`.

```
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

```
1  class MyTests : BehaviorSpec({  
2      given("a broomstick") {  
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     }  
14 })
```

Behavior Spec

Popular with people who like to write tests in the *BDD* style,
`BehaviorSpec` allows you to use `given`, `when`, `then`.

```
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

```
1  class MyTests : BehaviorSpec({  
2      given("a broomstick") {  
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     }  
14 })
```

Behavior Spec

Popular with people who like to write tests in the *BDD* style,
`BehaviorSpec` allows you to use `given`, `when`, `then`.

```
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

```
1  class MyTests : BehaviorSpec({  
2      given("a broomstick") {  
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     }  
14 })
```

Behavior Spec

Popular with people who like to write tests in the *BDD* style,
`BehaviorSpec` allows you to use `given`, `when`, `then`.

```
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

```
1  class MyTests : BehaviorSpec({  
2      given("a broomstick") {  
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     }  
14 })
```

Behavior Spec

Popular with people who like to write tests in the *BDD* style,
`BehaviorSpec` allows you to use `given`, `when`, `then`.

```
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

```
1  class MyTests : BehaviorSpec{  
2      given("a broomstick") {  
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     }  
14 }
```

Behavior Spec

Popular with people who like to write tests in the *BDD* style,
`BehaviorSpec` allows you to use `given`, `when`, `then`.

```
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  
            }  
        }  
    }  
}
```

Another flavor of Kotest

Kotest: <https://kotest.io>

```
1  class MyTests : DescribeSpec(){  
2      describe("score") {  
3          it("start as zero") { /* test here */ }  
4          describe("with a strike") {  
5              it("adds ten") { /* test here */ }  
6              it("carries strike to the next frame") { /* test here */ }  
7          }  
8          describe("for the opposite team") {  
9              it("Should negate one score") { /* test here */ }  
10         }  
11     }  
12 })
```

Another flavor of Kotest

Kotest: <https://kotest.io>

```
1  class MyTests : DescribeSpec({  
2      describe("score") {  
3          it("start as zero") { /* test here */ }  
4          describe("with a strike") {  
5              it("adds ten") { /* test here */ }  
6              it("carries strike to the next frame") { /* test here */ }  
7          }  
8          describe("for the opposite team") {  
9              it("Should negate one score") { /* test here */ }  
10         }  
11     }  
12 })
```

Another flavor of Kotest

Kotest: <https://kotest.io>

```
1  class MyTests : DescribeSpec{  
2      describe("score") {  
3          it("start as zero") { /* test here */ }  
4          describe("with a strike") {  
5              it("adds ten") { /* test here */ }  
6              it("carries strike to the next frame") { /* test here */ }  
7          }  
8          describe("for the opposite team") {  
9              it("Should negate one score") { /* test here */ }  
10         }  
11     }  
12 })
```

Another flavor of Kotest

Kotest: <https://kotest.io>

```
1  class MyTests : DescribeSpec{  
2      describe("score") {  
3          it("start as zero") { /* test here */ }  
4          describe("with a strike") {  
5              it("adds ten") { /* test here */ }  
6              it("carries strike to the next frame") { /* test here */ }  
7          }  
8          describe("for the opposite team") {  
9              it("Should negate one score") { /* test here */ }  
10         }  
11     }  
12 })
```

Another flavor of Kotest

Kotest: <https://kotest.io>

```
1  class MyTests : DescribeSpec({  
2      describe("score") {  
3          it("start as zero") { /* test here */ }  
4          describe("with a strike") {  
5              it("adds ten") { /* test here */ }  
6              it("carries strike to the next frame") { /* test here */ }  
7          }  
8          describe("for the opposite team") {  
9              it("Should negate one score") { /* test here */ }  
10         }  
11     }  
12 })
```

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?

```
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 }
```

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

How do I run the same tests with different inputs?
Data-driven testing a.k.a Parametrized tests

Parametrized tests? Java

```
1  static Stream<Arguments> stringIntAndListProvider() {
2      return Stream.of(
3          arguments("apple", 1, Arrays.asList("a", "b")),
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

```
1 static Stream<Arguments> stringIntAndListProvider() {
2     return Stream.of(
3         arguments("apple", 1, Arrays.asList("a", "b")),
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

```
1 static Stream<Arguments> stringIntAndListProvider() {
2     return Stream.of(
3         arguments("apple", 1, Arrays.asList("a", "b")),
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

```
1 static Stream<Arguments> stringIntAndListProvider() {
2     return Stream.of(
3         arguments("apple", 1, Arrays.asList("a", "b")),
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

```
1 static Stream<Arguments> stringIntAndListProvider() {
2     return Stream.of(
3         arguments("apple", 1, Arrays.asList("a", "b")),
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

```
1 static Stream<Arguments> stringIntAndListProvider() {
2     return Stream.of(
3         arguments("apple", 1, Arrays.asList("a", "b")),
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

```
1 static Stream<Arguments> stringIntAndListProvider() {
2     return Stream.of(
3         arguments("apple", 1, Arrays.asList("a", "b")),
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

```
1 static Stream<Arguments> stringIntAndListProvider() {
2     return Stream.of(
3         arguments("apple", 1, Arrays.asList("a", "b")),
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

```
1 static Stream<Arguments> stringIntAndListProvider() {
2     return Stream.of(
3         arguments("apple", 1, Arrays.asList("a", "b")),
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

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

`stringIntAndListProvider` , really?

Parametrized tests? Java

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

`stringIntAndListProvider` , really?

- Becomes cluttered

Parametrized tests? Java

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

`stringIntAndListProvider` , really?

- Becomes cluttered
- Not typed (enough)

Parametrized tests? Java

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

`stringIntAndListProvider` , really?

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

Parametrized tests? Kotlin (with kotest)

```
1 fun isPythagTriple(a: Int, b: Int, c: Int): Boolean = a * a + b * b == c * c
2 // snip...
3 withData( // ddt
4     Triple(3, 4, 5),
5     Triple(6, 8, 10),
6     Triple(8, 15, 17),
7     Triple(7, 24, 25)
8 ) { (a, b, c) -> // destructuring
9     isPythagTriple(a, b, c) shouldBe true //test
10 }
```

Parametrized tests? Kotlin (with kotest)

```
1  fun isPythagTriple(a: Int, b: Int, c: Int): Boolean = a * a + b * b == c * c
2  // snip...
3  withData( // ddt
4      Triple(3, 4, 5),
5      Triple(6, 8, 10),
6      Triple(8, 15, 17),
7      Triple(7, 24, 25)
8  ) { (a, b, c) -> // destructuring
9      isPythagTriple(a, b, c) shouldBe true //test
10 }
```

Parametrized tests? Kotlin (with kotest)

```
1 fun isPythagTriple(a: Int, b: Int, c: Int): Boolean = a * a + b * b == c * c
2 // snip...
3 withData( // ddt
4     Triple(3, 4, 5),
5     Triple(6, 8, 10),
6     Triple(8, 15, 17),
7     Triple(7, 24, 25)
8 ) { (a, b, c) -> // destructuring
9     isPythagTriple(a, b, c) shouldBe true //test
10 }
```

Parametrized tests? Kotlin (with kotest)

```
1 fun isPythagTriple(a: Int, b: Int, c: Int): Boolean = a * a + b * b == c * c
2 // snip...
3 withData( // ddt
4     Triple(3, 4, 5),
5     Triple(6, 8, 10),
6     Triple(8, 15, 17),
7     Triple(7, 24, 25)
8 ) { (a, b, c) -> // destructuring
9     isPythagTriple(a, b, c) shouldBe true //test
10 }
```

Parametrized tests? Kotlin (with kotest)

```
1 fun isPythagTriple(a: Int, b: Int, c: Int): Boolean = a * a + b * b == c * c
2 // snip...
3 withData( // ddt
4     Triple(3, 4, 5),
5     Triple(6, 8, 10),
6     Triple(8, 15, 17),
7     Triple(7, 24, 25)
8 ) { (a, b, c) -> // destructuring
9     isPythagTriple(a, b, c) shouldBe true //test
10 }
```

Parametrized tests? Kotlin (with kotest)

```
1  fun isPythagTriple(a: Int, b: Int, c: Int): Boolean = a * a + b * b == c * c
2  // snip...
3  withData( // ddt
4      Triple(3, 4, 5),
5      Triple(6, 8, 10),
6      Triple(8, 15, 17),
7      Triple(7, 24, 25)
8  ) { (a, b, c) -> // destructuring
9      isPythagTriple(a, b, c) shouldBe true //test
10 }
```

What if

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



Never write your own hashing (unless...)

```
1  withData(  
2      "love",  
3      "secret",  
4      "god", // Who will remember the reference?  
5  ) { input ->  
6      MD5.compute(input) shouldBe libMd5(input)  
7 }
```

Never write your own hashing (unless...)

```
1  withData(  
2      "love",  
3      "secret",  
4      "god", // Who will remember the reference?  
5  ) { input ->  
6      MD5.compute(input) shouldBe libMd5(input)  
7  }
```

Never write your own hashing (unless...)

```
1  withData(  
2      "love",  
3      "secret",  
4      "god", // Who will remember the reference?  
5  ) { input ->  
6      MD5.compute(input) shouldBe libMd5(input)  
7  }
```

Never write your own hashing (unless...)

```
1  withData(  
2      "love",  
3      "secret",  
4      "god", // Who will remember the reference?  
5  ) { input ->  
6      MD5.compute(input) shouldBe libMd5(input)  
7  }
```

Never write your own hashing (unless...)

```
1  withData(  
2      "love",  
3      "secret",  
4      "god", // Who will remember the reference?  
5  ) { input ->  
6      MD5.compute(input) shouldBe libMd5(input)  
7 }
```

Launching...

Never write your own hashing (unless...)

```
1  withData(  
2      "love",  
3      "secret",  
4      "god", // Who will remember the reference?  
5  ) { input ->  
6      MD5.compute(input) shouldBe libMd5(input)  
7 }
```

Launching...

1. ✓ love
2. ✓ secret
3. ✓ god

Time to investigate!



Property tests!

To test things we can not even *imagine*

```
1  checkAll(
2      Arb.string(
3          codepoints = Arb.of(
4              (Char.MIN_VALUE.code .. Char.MAX_VALUE.code).map(::Codepoint)
5          ),
6          range = 0 .. 52
7      )
8  ) { input ->
9      MD5.compute(input) shouldBe libMd5(input)
10 }
```

Property tests!

To test things we can not even *imagine*

```
1  checkAll(
2      Arb.string(
3          codepoints = Arb.of(
4              (Char.MIN_VALUE.code..Char.MAX_VALUE.code).map(::Codepoint)
5          ),
6          range = 0..52
7      )
8  ) { input ->
9      MD5.compute(input) shouldBe libMd5(input)
10 }
```

Property tests!

To test things we can not even *imagine*

```
1  checkAll(
2      Arb.string(
3          codepoints = Arb.of(
4              (Char.MIN_VALUE.code..Char.MAX_VALUE.code).map(::Codepoint)
5          ),
6          range = 0..52
7      )
8  ) { input ->
9      MD5.compute(input) shouldBe libMd5(input)
10 }
```

Property tests!

To test things we can not even *imagine*

```
1  checkAll(
2      Arb.string(
3          codepoints = Arb.of(
4              (Char.MIN_VALUE.code..Char.MAX_VALUE.code).map(::Codepoint)
5          ),
6          range = 0..52
7      )
8  ) { input ->
9      MD5.compute(input) shouldBe libMd5(input)
10 }
```

Property tests!

To test things we can not even *imagine*

```
1  checkAll(
2      Arb.string(
3          codepoints = Arb.of(
4              (Char.MIN_VALUE.code .. Char.MAX_VALUE.code).map(::Codepoint)
5          ),
6          range = 0 .. 52
7      )
8  ) { input ->
9      MD5.compute(input) shouldBe libMd5(input)
10 }
```

Property tests!

To test things we can not even *imagine*

```
1  checkAll(
2      Arb.string(
3          codepoints = Arb.of(
4              (Char.MIN_VALUE.code..Char.MAX_VALUE.code).map(::Codepoint)
5          ),
6          range = 0..52
7      )
8  ) { input ->
9      MD5.compute(input) shouldBe libMd5(input)
10 }
```

Property tests!

To test things we can not even *imagine*

```
1  checkAll(
2      Arb.string(
3          codepoints = Arb.of(
4              (Char.MIN_VALUE.code..Char.MAX_VALUE.code).map(::Codepoint)
5          ),
6          range = 0..52
7      )
8  ) { input ->
9      MD5.compute(input) shouldBe libMd5(input)
10 }
```

Property tests!

To test things we can not even *imagine*

```
1  checkAll(
2      Arb.string(
3          codepoints = Arb.of(
4              (Char.MIN_VALUE.code..Char.MAX_VALUE.code).map(::Codepoint)
5          ),
6          range = 0..52
7      )
8  ) { input ->
9      MD5.compute(input) shouldBe libMd5(input)
10 }
```



Error!

Wow, it was a complex sample!

But there is more...

Error!

Wow, it was a complex sample!

But there is more...

Error!

Wow, it was a complex sample!

But there is more...

Error!

Wow, it was a complex sample!

But there is more...

Error!

And there is more!

Repeat? Yes!

```
1  checkAll(
2      PropTestConfig(seed = 4919313423746630853),
3      Arb.string(
4          codepoints = Arb.of(
5              'a'.code..Char.MAX_HIGH_SURROGATE.code)
6              .map(::Codepoint)
7          ),
8      )
9  ) { input ->
10     MD5.compute(input) shouldBe libMd5(input)
11 }
```

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

Repeat? Yes!

```
1  checkAll(
2      PropTestConfig(seed = 4919313423746630853),
3      Arb.string(
4          codepoints = Arb.of(
5              ('a'.code..Char.MAX_HIGH_SURROGATE.code)
6                  .map(::Codepoint)
7          ),
8      )
9  ) { input ->
10     MD5.compute(input) shouldBe libMd5(input)
11 }
```

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



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

```
1 UserRepository mockRepository = Mockito.mock(UserRepository.class);
2
3 // Create some sample user data
4 List<User> users = Arrays.asList(new User("Alice", 25), new User("Bob", 30), new User("Charlie", 35));
5
6 // Define the expected behavior of the mock repository
7 Mockito.when(mockRepository.allUsers()).thenReturn(users);
8
9 // Create an instance of the UserService with the mock repository
10 UserService userService = new UserService(mockRepository);
11
12 // Call the method under test
13 List<User> result = userService.getAllUsers();
14
15 // Verify the result
16 Assertions.assertEquals(users, result);
```

How the test looks in Java

```
1 UserRepository mockRepository = Mockito.mock(UserRepository.class);
2
3 // Create some sample user data
4 List<User> users = Arrays.asList(new User("Alice", 25), new User("Bob", 30), new User("Charlie", 35));
5
6 // Define the expected behavior of the mock repository
7 Mockito.when(mockRepository.allUsers()).thenReturn(users);
8
9 // Create an instance of the UserService with the mock repository
10 UserService userService = new UserService(mockRepository);
11
12 // Call the method under test
13 List<User> result = userService.getAllUsers();
14
15 // Verify the result
16 Assertions.assertEquals(users, result);
```

How the test looks in Java

```
1 UserRepository mockRepository = Mockito.mock(UserRepository.class);
2
3 // Create some sample user data
4 List<User> users = Arrays.asList(new User("Alice", 25), new User("Bob", 30), new User("Charlie", 35));
5
6 // Define the expected behavior of the mock repository
7 Mockito.when(mockRepository.allUsers()).thenReturn(users);
8
9 // Create an instance of the UserService with the mock repository
10 UserService userService = new UserService(mockRepository);
11
12 // Call the method under test
13 List<User> result = userService.getAllUsers();
14
15 // Verify the result
16 Assertions.assertEquals(users, result);
```

How the test looks in Java

```
1 UserRepository mockRepository = Mockito.mock(UserRepository.class);
2
3 // Create some sample user data
4 List<User> users = Arrays.asList(new User("Alice", 25), new User("Bob", 30), new User("Charlie", 35));
5
6 // Define the expected behavior of the mock repository
7 Mockito.when(mockRepository.allUsers()).thenReturn(users);
8
9 // Create an instance of the UserService with the mock repository
10 UserService userService = new UserService(mockRepository);
11
12 // Call the method under test
13 List<User> result = userService.getAllUsers();
14
15 // Verify the result
16 Assertions.assertEquals(users, result);
```

How the test looks in Java

```
1 UserRepository mockRepository = Mockito.mock(UserRepository.class);
2
3 // Create some sample user data
4 List<User> users = Arrays.asList(new User("Alice", 25), new User("Bob", 30), new User("Charlie", 35));
5
6 // Define the expected behavior of the mock repository
7 Mockito.when(mockRepository.allUsers()).thenReturn(users);
8
9 // Create an instance of the UserService with the mock repository
10 UserService userService = new UserService(mockRepository);
11
12 // Call the method under test
13 List<User> result = userService.getAllUsers();
14
15 // Verify the result
16 Assertions.assertEquals(users, result);
```

How the test looks in Java

```
1 UserRepository mockRepository = Mockito.mock(UserRepository.class);
2
3 // Create some sample user data
4 List<User> users = Arrays.asList(new User("Alice", 25), new User("Bob", 30), new User("Charlie", 35));
5
6 // Define the expected behavior of the mock repository
7 Mockito.when(mockRepository.allUsers()).thenReturn(users);
8
9 // Create an instance of the UserService with the mock repository
10 UserService userService = new UserService(mockRepository);
11
12 // Call the method under test
13 List<User> result = userService.getAllUsers();
14
15 // Verify the result
16 Assertions.assertEquals(users, result);
```

How the test looks in Java

```
1 UserRepository mockRepository = Mockito.mock(UserRepository.class);
2
3 // Create some sample user data
4 List<User> users = Arrays.asList(new User("Alice", 25), new User("Bob", 30), new User("Charlie", 35));
5
6 // Define the expected behavior of the mock repository
7 Mockito.when(mockRepository.allUsers()).thenReturn(users);
8
9 // Create an instance of the UserService with the mock repository
10 UserService userService = new UserService(mockRepository);
11
12 // Call the method under test
13 List<User> result = userService.getAllUsers();
14
15 // Verify the result
16 Assertions.assertEquals(users, result);
```

Time to mock! (MockK edition)

```
1 val users = listOf(new User("Alice", 25), new User("Bob", 30), new 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(new User("Alice", 25), new User("Bob", 30), new 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(new User("Alice", 25), new User("Bob", 30), new 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(new User("Alice", 25), new User("Bob", 30), new 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(new User("Alice", 25), new User("Bob", 30), new 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(new User("Alice", 25), new User("Bob", 30), new 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(new User("Alice", 25), new User("Bob", 30), new 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"
```

Thank Alexandra for helping me with this presentation!

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"
```

Thank Alexandra for helping me with this presentation!

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"
```

Thank Alexandra for helping me with this presentation!

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"
```

Thank Alexandra for helping me with this presentation!

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"
```

Thank Alexandra for helping me with this presentation!

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"
```

Thank Alexandra for helping me with this presentation!

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"
```

Thank Alexandra for helping me with this presentation!

Users are described inside `User` mocks!

Atrium expectations!

<https://docs.atriumlib.org/>

```
1  expect(service.getAllUsers()) toContain o inAny order but only the entries(
2      {
3          its { name } toEqual "Pash"
4          its { age } toEqual 36
5      },
6      {
7          its { name } toEqual "Mark"
8          its { age } toEqual 13
9      }
10 )
```

Atrium expectations!

<https://docs.atriumlib.org/>

```
1  expect(service.getAllUsers()) toContain o inAny order but only the entries(
2      {
3          its { name } toEqual "Pash"
4          its { age } toEqual 36
5      },
6      {
7          its { name } toEqual "Mark"
8          its { age } toEqual 13
9      }
10 )
```

Atrium expectations!

<https://docs.atriumlib.org/>

```
1  expect(service.getAllUsers()) toContain o inAny order but only the entries(
2    {
3      its { name } toEqual "Pash"
4      its { age } toEqual 36
5    },
6    {
7      its { name } toEqual "Mark"
8      its { age } toEqual 13
9    }
10 )
```

I've made a mistake to show the error message

Atrium expectations!

<https://docs.atriumlib.org/>

```
1  expect(service.getAllUsers()) toContain o inAny order but only the entries(
2    {
3      its { name } toEqual "Pash"
4      its { age } toEqual 36
5    },
6    {
7      its { name } toEqual "Mark"
8      its { age } toEqual 13
9    }
10 )
```

I've made a mistake to show the error message

Atrium expectations!

<https://docs.atriumlib.org/>

```
1  expect(service.getAllUsers()) toContain o inAny order but only the entries(
2    {
3      its { name } toEqual "Pash"
4      its { age } toEqual 36
5    },
6    {
7      its { name } toEqual "Mark"
8      its { age } toEqual 13
9    }
10 )
```

I've made a mistake to show the error message

Error report

```
1 I expected subject: [User(#3), User(#4)]          (java.util.Arrays.ArrayList <315506631>)
2 ◆ to contain only, in any order:
3   ✘ an element which needs:
4     » ▶ its.isDefinedIn(MockTest.kt:42):
5       ■ to equal: "Pash"           <2026086646>
6     » ▶ its.isDefinedIn(MockTest.kt:43):
7       ■ to equal: 36            (kotlin.Int <116289638>)
8   ✓ an element which needs:
9     » ▶ its.isDefinedIn(MockTest.kt:46):
10      ■ to equal: "Mark"         <574921197>
11     » ▶ its.isDefinedIn(MockTest.kt:47):
12      ■ to equal: 13            (kotlin.Int <453296736>)
13 !!! following elements were mismatched:
14   ◦ User(#3)             (User <1041341319>)
```

Error report

```
1 I expected subject: [User(#3), User(#4)]          (java.util.Arrays.ArrayList <315506631>)
2 ◆ to contain only, in any order:
3   ✘ an element which needs:
4     » ▶ its.isDefinedIn(MockTest.kt:42):
5       ■ to equal: "Pash"           <2026086646>
6     » ▶ its.isDefinedIn(MockTest.kt:43):
7       ■ to equal: 36            (kotlin.Int <116289638>)
8   ✓ an element which needs:
9     » ▶ its.isDefinedIn(MockTest.kt:46):
10      ■ to equal: "Mark"        <574921197>
11     » ▶ its.isDefinedIn(MockTest.kt:47):
12      ■ to equal: 13            (kotlin.Int <453296736>)
13 !!! following elements were mismatched:
14   ◦ User(#3)             (User <1041341319>)
```

Error report

```
1 I expected subject: [User(#3), User(#4)]          (java.util.Arrays.ArrayList <315506631>)
2 ◆ to contain only, in any order:
3   ✘ an element which needs:
4     » ▶ its.isDefinedIn(MockTest.kt:42):
5       ■ to equal: "Pash"           <2026086646>
6     » ▶ its.isDefinedIn(MockTest.kt:43):
7       ■ to equal: 36            (kotlin.Int <116289638>)
8   ✓ an element which needs:
9     » ▶ its.isDefinedIn(MockTest.kt:46):
10      ■ to equal: "Mark"         <574921197>
11     » ▶ its.isDefinedIn(MockTest.kt:47):
12      ■ to equal: 13            (kotlin.Int <453296736>)
13 !!! following elements were mismatched:
14   ◦ User(#3)             (User <1041341319>)
```

Error report

```
1 I expected subject: [User(#3), User(#4)]          (java.util.Arrays.ArrayList <315506631>)
2 ◆ to contain only, in any order:
3   ✘ an element which needs:
4     » ▶ its.isDefinedIn(MockTest.kt:42):
5       ■ to equal: "Pash"           <2026086646>
6     » ▶ its.isDefinedIn(MockTest.kt:43):
7       ■ to equal: 36            (kotlin.Int <116289638>)
8   ✓ an element which needs:
9     » ▶ its.isDefinedIn(MockTest.kt:46):
10      ■ to equal: "Mark"         <574921197>
11     » ▶ its.isDefinedIn(MockTest.kt:47):
12      ■ to equal: 13            (kotlin.Int <453296736>)
13 !!! following elements were mismatched:
14   ◦ User(#3)             (User <1041341319>)
```

Error report

```
1 I expected subject: [User(#3), User(#4)]          (java.util.Arrays.ArrayList <315506631>)
2 ◆ to contain only, in any order:
3   ✘ an element which needs:
4     » ▶ its.isDefinedIn(MockTest.kt:42):
5       ■ to equal: "Pash"              <2026086646>
6     » ▶ its.isDefinedIn(MockTest.kt:43):
7       ■ to equal: 36                (kotlin.Int <116289638>)
8   ✓ an element which needs:
9     » ▶ its.isDefinedIn(MockTest.kt:46):
10      ■ to equal: "Mark"           <574921197>
11     » ▶ its.isDefinedIn(MockTest.kt:47):
12      ■ to equal: 13                (kotlin.Int <453296736>)
13 !!! following elements were mismatched:
14   ◦ User(#3)                  (User <1041341319>)
```

Error report

```
1 I expected subject: [User(#3), User(#4)]          (java.util.Arrays.ArrayList <315506631>)
2 ◆ to contain only, in any order:
3   ✘ an element which needs:
4     » ▶ its.isDefinedIn(MockTest.kt:42):
5       ■ to equal: "Pash"           <2026086646>
6     » ▶ its.isDefinedIn(MockTest.kt:43):
7       ■ to equal: 36            (kotlin.Int <116289638>)
8   ✓ an element which needs:
9     » ▶ its.isDefinedIn(MockTest.kt:46):
10      ■ to equal: "Mark"         <574921197>
11     » ▶ its.isDefinedIn(MockTest.kt:47):
12      ■ to equal: 13            (kotlin.Int <453296736>)
13 !!! following elements were mismatched:
14   ◦ User(#3)             (User <1041341319>)
```

Error report

```
1 I expected subject: [User(#3), User(#4)]          (java.util.Arrays.ArrayList <315506631>)
2 ◆ to contain only, in any order:
3   ✘ an element which needs:
4     » ▶ its.isDefinedIn(MockTest.kt:42):
5       ■ to equal: "Pash"           <2026086646>
6     » ▶ its.isDefinedIn(MockTest.kt:43):
7       ■ to equal: 36            (kotlin.Int <116289638>)
8   ✓ an element which needs:
9     » ▶ its.isDefinedIn(MockTest.kt:46):
10      ■ to equal: "Mark"         <574921197>
11     » ▶ its.isDefinedIn(MockTest.kt:47):
12      ■ to equal: 13            (kotlin.Int <453296736>)
13   !! following elements were mismatched:
14     ° User(#3)             (User <1041341319>)
```

- Readable as a text
- Shows the errors

Dessert

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

```
1 ideaTest<App>("Create Kafka connection") {
2     showBDTPanel {
3         createBaseConnection("Kafka") {
4             fillCredentialsForKafka(connectionName)
5             checkSuccessfulMessage()
6             ok()
7         }
8         checkMonitoringConnection(monitoringToolWindow)
9     }
10 }
```

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

Thank you! Time for questions



🐦 asm0di0

👤 @asm0dey@fosstodon.org

✉️ me@asm0dey.site

.twitch https://jb.gg/twitch

🐱 🐾 💬 asm0dey

🌐 <https://linktr.ee/asm0dey>





END

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

```
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

```
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

```
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

```
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='InternalDecoratorImpl ']"))
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(xpath("//div[@accessibleName='Kafka Tool Window' and @class='InternalDecoratorImpl ']"))
10                     BDTPanel(app, frame, this).block()
11                 }
12             }
13         }
14     }
15 }
```

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(xpath("//div[@accessibleName='Kafka Tool Window' and @class='InternalDecoratorImpl ']"))
10                     BDTPanel(app, frame, this).block()
11                 }
12             }
13         }
14     }
15 }
```

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(xpath("//div[@accessibleName='Kafka Tool Window' and @class='InternalDecoratorImpl ']"))
10                     BDTPanel(app, frame, this).block()
11                 }
12             }
13         }
14     }
15 }
```

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(xpath("//div[@accessibleName='Kafka Tool Window' and @class='InternalDecoratorImpl ']"))
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='InternalDecoratorImpl ']"
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, 'toolwindow.tab.add')]")).click()
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, 'toolwindow.tab.add')]")).click()
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, 'toolwindow.tab.add')]")).click()
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, 'toolwindow.tab.add')]")).click()
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, 'toolwindow.tab.add')]")).click()
5              clickTypeOfConnection(typeOfConnection, block)
6              pause(5000)
```

And this is the abyss!

But you can do the same!

Go back to summary

END