Integrate Kotlin Coroutines and JUnit 5

Ruslan Ibragimov

Agenda

- JUnit & Coroutines: **Problems**
- **JUnit 5**: Platform, Jupiter, etc
- JUnit & Coroutines: **Solutions**
- **Testing Coroutines**

5

Coroutines meets Testing

```
@Test
fun `test get by email`() {
    val userApi = UserApi(HttpClient())

    val user = userApi.getByEmail("Andrey.Breslav@JetBrains.com")
    assertEquals("Andrey Breslav", user.name)
}
```

Coroutines meets Testing

```
        fun getByEmail(email: String): User
   ↓
suspend fun getByEmail(email: String): User
```

Coroutines meets Testing

Kotlin: Suspend function 'getByEmail' should be called only from a coroutine or another suspend function

```
@Test
fun `test get by email`() {
    val userApi = UserApi(HttpClient())

    val user = userApi.getByEmail("Andrey.Breslav@JetBrains.com")
    assertEquals("Andrey Breslav", user.name)
}
```

Coroutines meets Testing

No test were found

```
@Test
suspend fun `test get by email`() {
    val userApi = UserApi(HttpClient())

    val user = userApi.getByEmail("Andrey.Breslav@JetBrains.com")
    assertEquals("Andrey Breslav", user.name)
}
```

Coroutines meets Testing

Tests passed: 1

```
@Test
fun `test get by email`() = runBlocking {
    val userApi = UserApi(HttpClient())

    val user = userApi.getByEmail("Andrey.Breslav@JetBrains.com")
    assertEquals("Andrey Breslav", user.name)
}
```

Coroutines meets Testing

```
@Test
fun `test get by email not found`() {
    val userApi = UserApi(HttpClient())

    assertThrows<UserNotFoundException> {
        userApi.getByEmail("ruslan@ibragimov.by")
    }
}
```

✓

Coroutines meets Testing

```
@Test
fun `test get by email not found`() = runBlocking {
    val userApi = UserApi(HttpClient())

    assertThrows<UserNotFoundException> {
        userApi.getByEmail("ruslan@ibragimov.by")
    }
}
```

Coroutines meets Testing

JUnit test should return **Unit**

```
@Test
fun `test get by email not found`(): UserNotFoundException = runBlocking {
    val userApi = UserApi(HttpClient())

    assertThrows<UserNotFoundException> {
        userApi.getByEmail("ruslan@ibragimov.by")
    }
}
```

Kotlin: Suspend function 'getByEmail' should be called only from a coroutine or another suspend function

JUNIT 5

JUnit 5

Intellij Idea 2016.2

Eclipse 4.7.1 (October 2017)

Gradle 4.6 (July 2016 / April 2018)

Maven Surfire 2.22.0 (June 2018)

NetBeans 10 (December 27, 2018)

JUnit 5

```kotlin
@Test
suspend fun `test get by email`()
```

Implicit Argument

```kotlin
@Test
suspend fun `test get by email`(continuation: Continuation<*>)
```

JUnit 5

```kotlin
class ContinuationParameterResolver : ParameterResolver {
    override fun supportsParameter(
        parameterContext: ParameterContext,
        extensionContext: ExtensionContext
    ): Boolean {
        return parameterContext.parameter.type == Continuation::class.java
    }

    override fun resolveParameter(
        parameterContext: ParameterContext,
        extensionContext: ExtensionContext
    ): Continuation<Any?> {
        return object : Continuation<Any?> {
            override fun resumeWith(result: Result<Any?>) {
                // fail or success current test
            }

            override val context: CoroutineContext
                get() = EmptyCoroutineContext
        }
    }
}
```

JUnit 5

```
class ContinuationParameterResolver : ParameterResolver {
    override fun supportsParameter(
        parameterContext: ParameterContext,
        extensionContext: ExtensionContext
    ): Boolean {
        return parameterContext.parameter.type == Continuation::class.java
    }

    override fun resolveParameter(
        parameterContext: ParameterContext,
        extensionContext: ExtensionContext
    ): Continuation<Any?> {
        return object : Continuation<Any?> {
            override fun resumeWith(result: Result<Any?>) {
                // fail or success current test
            }

            override val context: CoroutineContext
                get() = EmptyCoroutineContext
        }
    }
}
```

JUnit 5

```
class ContinuationParameterResolver : ParameterResolver {
    override fun supportsParameter(
        parameterContext: ParameterContext,
        extensionContext: ExtensionContext
    ): Boolean {
        return parameterContext.parameter.type == Continuation::class.java
    }

    override fun resolveParameter(
        parameterContext: ParameterContext,
        extensionContext: ExtensionContext
    ): Continuation<Any?> {
        return object : Continuation<Any?> {
            override fun resumeWith(result: Result<Any?>) {
                // fail or success current test
            }

            override val context: CoroutineContext
                get() = EmptyCoroutineContext
        }
    }
}
```

JUnit 5

No test were found

```
@ExtendWith(ContinuationParameterResolver::class)
class UserApiTest {
    @Test
    suspend fun `test get by email`() {
        // ..
    }
}
```

JUnit 5

```
@Test
suspend fun `test get by email`()
```

Return Type

```
@Test
suspend fun `test get by email`(continuation: Continuation<*>): Any
```

JUnit 5

```
suspend fun `test get by email`(): Any {
    // ...
    if (userApi(email) == Intrinsics.COROUTINE_SUSPENDED) {
        return Intrinsics.COROUTINE_SUSPENDED
    }
    // ...
}
```

JUnit 5: Extension

Lifecycle Callbacks:

- BeforeAllCallback
  - BeforeEachCallback
    - BeforeTestExecutionCallback
    - AfterTestExecutionCallback
  - AfterEachCallback
- AfterAllCallback

JUnit 5: Extension

TestExecutionExceptionHandler

ExecutionCondition

TestInstanceFactory

TestInstancePostProcessor

ParameterResolver

TestTemplateInvocationContextProvider

JUnit 5: Extension

TestExecutionExceptionHandler

ExecutionCondition

TestInstanceFactory

TestInstancePostProcessor

ParameterResolver

TestTemplateInvocationContextProvider

JUnit 5: Dynamic tests

```
@TestFactory
fun `dynamic api test example`(): List<DynamicTest> {
    val userApi = UserApi(HttpClient())

    return listOf(
        dynamicTest("test get by email") {
            val user = userApi.getByEmail("Andrey.Breslav@JetBrains.com")
            assertEquals("Andrey Breslav", user.name)
        },
        dynamicTest("test get by email not found") {
            assertThrows<UserNotFoundException> {
                userApi.getByEmail("ruslan@ibragimov.by")
            }
        }
    )
}
```

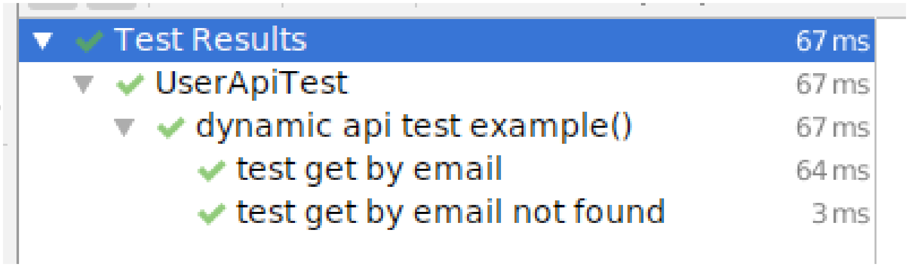
JUnit 5: Dynamic tests

```
"foo bar" { /* .(╯°□°)╯︵ ┻━┻ */ }
```

```
operator fun String.invoke(body: suspend () → Unit): DynamicTest {
    return dynamicTest(this) {
        runBlocking {
            body()
        }
    }
}
```

JUnit 5: Dynamic tests

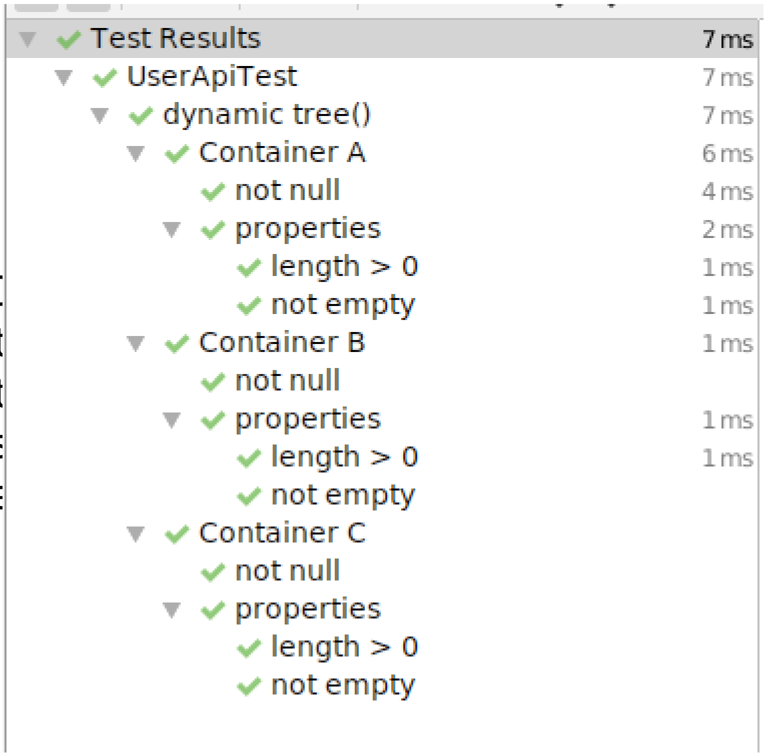
```
@TestFactory
fun `dynamic api test example`(): List<D
    val userApi = UserApi(HttpClient())

    return listOf(
        "test get by email" {
            val user = userApi.getByEmail("Andrey.Breslav@JetBrains.com")
            assertEquals("Andrey Breslav", user.name)
        },
        "test get by email not found" {
            assertThrows<UserNotFoundException> {
                userApi.getByEmail("ruslan@ibragimov.by")
            }
        }
    )
}
```



```
▼ ✓ Test Results                              67 ms
  ▼ ✓ UserApiTest                             67 ms
    ▼ ✓ dynamic api test example()            67 ms
        ✓ test get by email                   64 ms
        ✓ test get by email not found          3 ms
```

JUnit 5: Dynamic tests

```
@TestFactory
fun `dynamic tree`(): List<DynamicContainer> {
    return listOf("A", "B", "C").map {
        dynamicContainer("Container $it", listO
            dynamicTest("not null") { assertNot
            dynamicContainer("properties", list
                dynamicTest("length > 0") { ass
                dynamicTest("not empty") { asse
            ))
        ))
    }
}
```



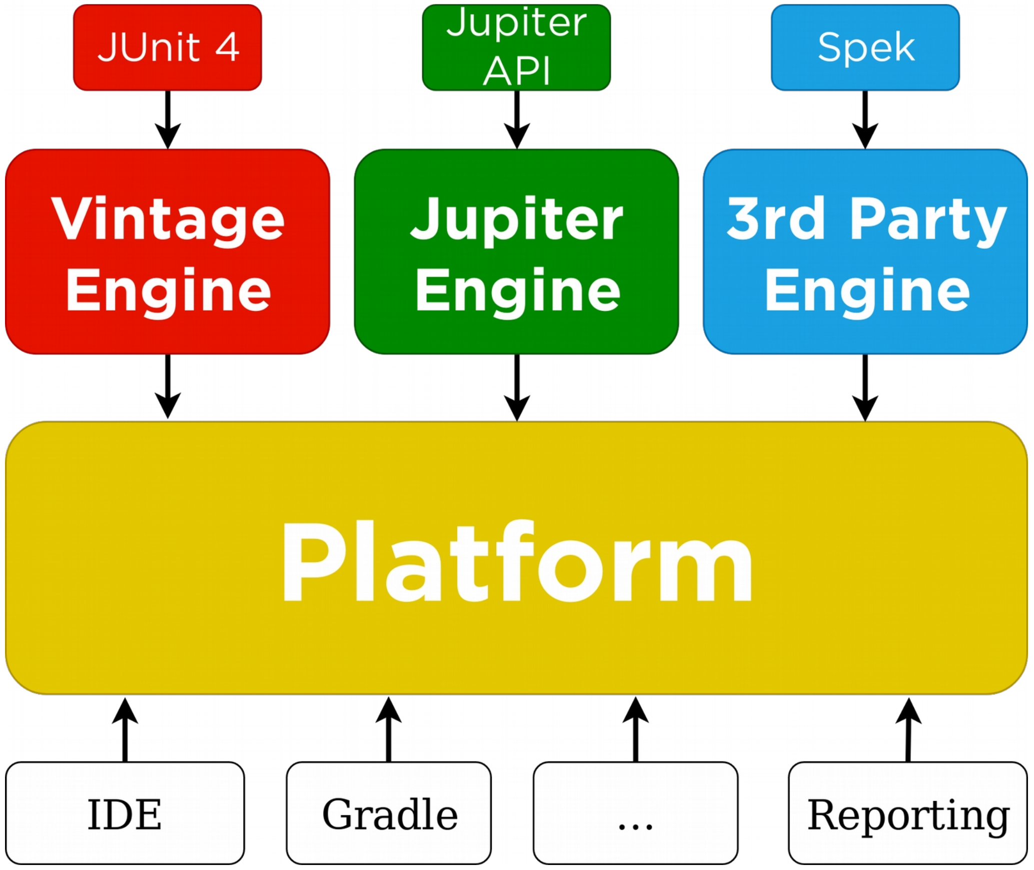
```
Test Results                    7 ms
  UserApiTest                   7 ms
    dynamic tree()              7 ms
      Container A               6 ms
        not null                4 ms
        properties              2 ms
          length > 0            1 ms
          not empty             1 ms
      Container B               1 ms
        not null
        properties              1 ms
          length > 0            1 ms
          not empty
      Container C
        not null
        properties
          length > 0
          not empty
```

JUnit 5

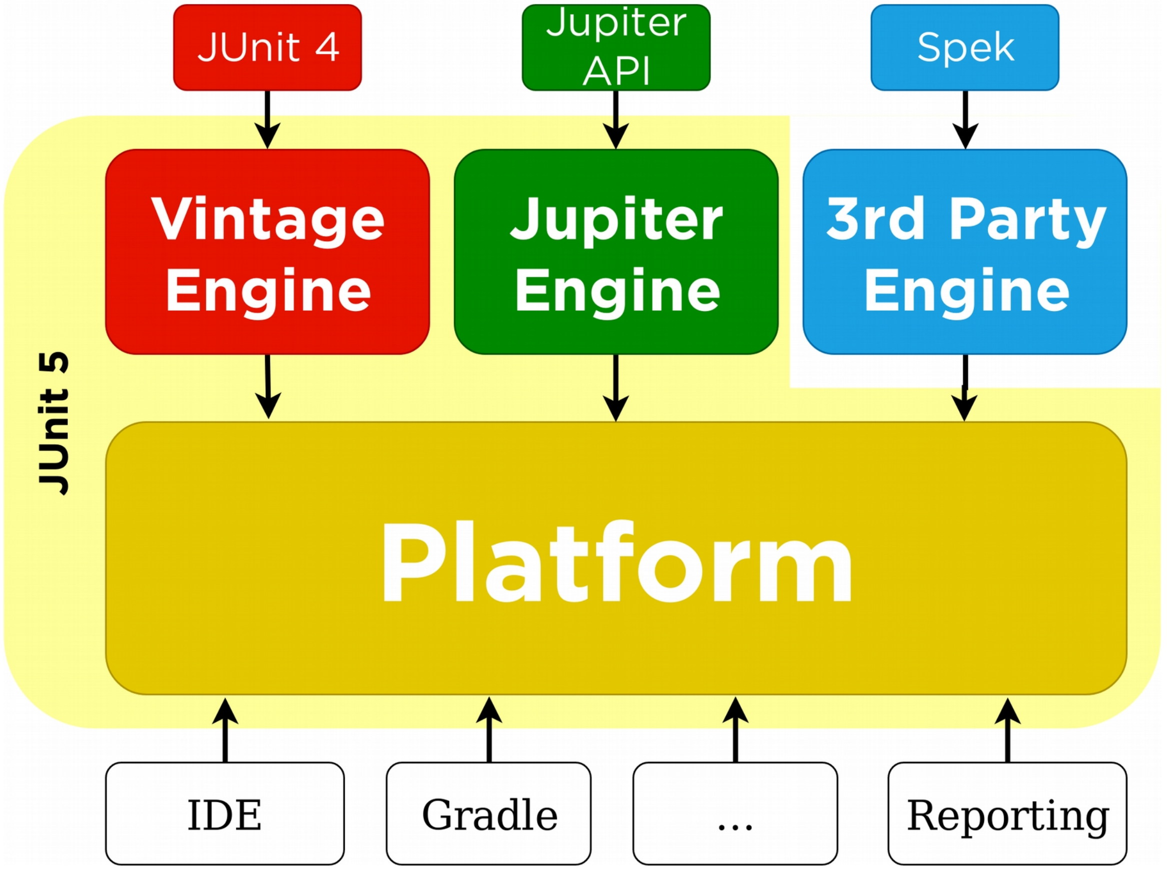
JUnit 5:

- Platform
  - API for **Launchers** and **TestEngines**
- Vintage
  - JUnit 3 & JUnit 4 **TestEngine**
- Jupiter
  - New model for writing tests

Architecture



Architecture



3rd Party Test Engines

Spek

KotlinTest

dynatest

Cucumber

Drools Scenario

jqwik

Mainrunner

Specsy

3rd Party Test Engines

Spek

KotlinTest

dynatest

Cucumber

Drools Scenario

jqwik

Mainrunner

Specsy

dynatest

```
class CalculatorTest : DynaTest({

    test("calculator instantiation test") {
        Calculator()
    }

    group("tests the plusOne() function") {
        test("one plusOne") {
            expect(2) { Calculator().plusOne(1) }
        }
    }
})
```

dynatest

```
class CalculatorTest : DynaTest({

    test("calculator instantiation test") {
        Calculator()
        suspendCall()
    }

    group("tests the plusOne() function") {
        test("one plusOne") {
            expect(2) { Calculator().plusOne(1) }
        }
    }
})
```

Spek

```
object CalculatorSpec : Spek({
    describe("A calculator") {
        it("calculator instantiation test") {
            Calculator()
        }

        describe("addition") {
            it("one plusOne") {
                assertEquals(2, Calculator().plusOne(1) )
            }
        }
    }
})
```

Spek

```kotlin
object CalculatorSpec : Spek({
    describe("A calculator") {
        it("calculator instantiation test") {
            Calculator()
            suspendCall()
        }

        describe("addition") {
            it("one plusOne") {
                assertEquals(2, Calculator().plusOne(1) )
            }
        }
    }
})
```

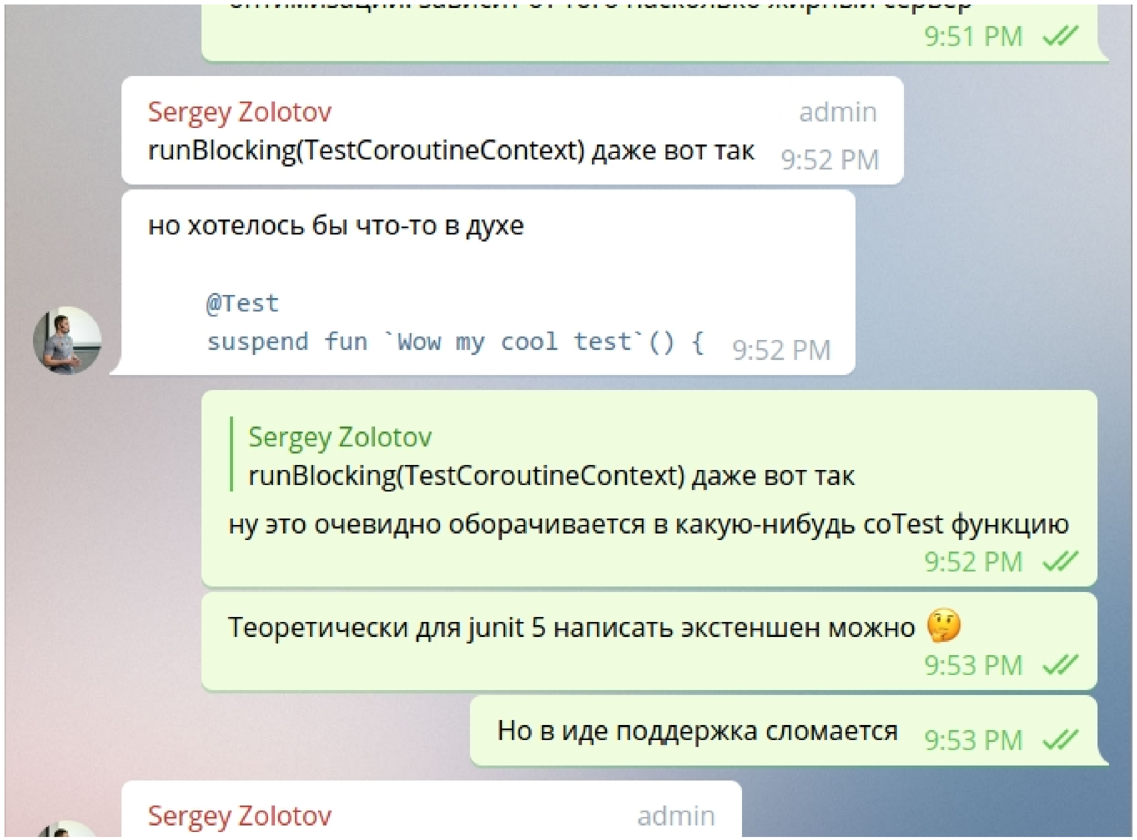
KotlinTest

```
class MyTests : StringSpec({
    "calculator should be instantiable" {
        Calculator()
    }
    "one plus one should be two" {
        Calculator().plusOne(1) should be(2)
    }
})
```

KotlinTest

```
class MyTests : StringSpec({
    "calculator should be inst
        Calculator()
        suspendCall()
    }
    "one plus one should be tw
        Calculator().plusOne(1
    }
})
```

Writing Test Engine



9:51 PM

Sergey Zolotov admin
runBlocking(TestCoroutineContext) даже вот так 9:52 PM

но хотелось бы что-то в духе

```
@Test
suspend fun `Wow my cool test`() {
```

9:52 PM

> Sergey Zolotov
> runBlocking(TestCoroutineContext) даже вот так

ну это очевидно оборачивается в какую-нибудь coTest функцию
9:52 PM

Теоретически для junit 5 написать экстеншен можно 🤔
9:53 PM

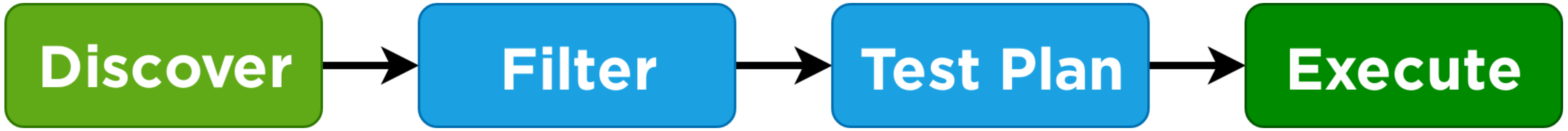
Но в иде поддержка сломается 9:53 PM

Sergey Zolotov admin

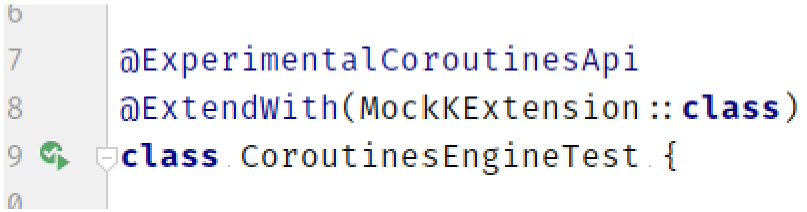
Writing Test Engine

```
class KotlinLvivEngine : TestEngine {
    override fun getId() = "kotlin-lviv"

    override fun discover(
        discoveryRequest: EngineDiscoveryRequest,
        uniqueId: UniqueId
    ): TestDescriptor = EngineDescriptor(
        UniqueId.forEngine("kotlin-lviv"),
        "Kotlin Lviv"
    )

    override fun execute(request: ExecutionRequest) {
    }
}
```

Writing Test Engine



Writing Test Engine: Discover



```
@ExperimentalCoroutinesApi
@ExtendWith(MockKExtension::class)
class CoroutinesEngineTest {
```

ClassSelector
MethodSelector
ClasspathRootSelector
FileSelector
ModuleSelector
ClasspathResourceSelector
UniqueIdSelector
UriSelector
DirectorySelector

Writing Test Engine: Discover

```kotlin
override fun discover(
    discoveryRequest: EngineDiscoveryRequest,
    uniqueId: UniqueId
): TestDescriptor {
    val root = EngineDescriptor(LVIV_ENGINE_UID, LVIV_ENGINE_NAME)

    discoveryRequest.getSelectorsByType(MethodSelector::class.java)
        .forEach { selector ->
            selector.javaMethod.kotlinFunction?.let {
                if (it.isSuspend) {
                    root.addChild(MethodTestDescriptor(it, selector.javaClass.kotlin))
                }
            }
        }

    return root
}
```

Writing Test Engine: Discover

```
override fun discover(
    discoveryRequest: EngineDiscoveryRequest,
    uniqueId: UniqueId
): TestDescriptor {
    val root = EngineDescriptor(LVIV_ENGINE_UID, LVIV_ENGINE_NAME)

    discoveryRequest.getSelectorsByType(MethodSelector::class.java)
        .forEach { selector ->
            selector.javaMethod.kotlinFunction?.let {
                if (it.isSuspend) {
                    root.addChild(MethodTestDescriptor(it, selector.javaClass.kotlin))
                }
            }
        }

    return root
}
```

Writing Test Engine: Discover

```kotlin
override fun discover(
    discoveryRequest: EngineDiscoveryRequest,
    uniqueId: UniqueId
): TestDescriptor {
    val root = EngineDescriptor(LVIV_ENGINE_UID, LVIV_ENGINE_NAME)

    discoveryRequest.getSelectorsByType(MethodSelector::class.java)
        .forEach { selector ->
            selector.javaMethod.kotlinFunction?.let {
                if (it.isSuspend) {
                    root.addChild(MethodTestDescriptor(it, selector.javaClass.kotlin))
                }
            }
        }

    return root
}
```

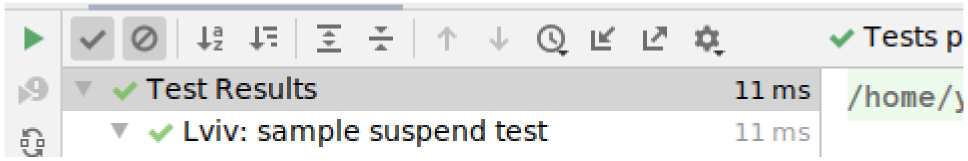
Writing Test Engine: Discover

```kotlin
override fun discover(
    discoveryRequest: EngineDiscoveryRequest,
    uniqueId: UniqueId
): TestDescriptor {
    val root = EngineDescriptor(LVIV_ENGINE_UID, LVIV_ENGINE_NAME)

    discoveryRequest.getSelectorsByType(MethodSelector::class.java)
        .forEach { selector →
            selector.javaMethod.kotlinFunction?.let {
                if (it.isSuspend) {
                    root.addChild(MethodTestDescriptor(it, selector.javaClass.kotlin))
                }
            }
        }

    return root
}
```

Writing Test Engine: Discover

```
class MethodTestDescriptor(
    val function: KFunction<*>,
    val enclosureClass: KClass<*>
) : AbstractTestDescriptor(
    LVIV_ENGINE_UID.append("method", function.name),
    "Lviv: ${function.name}"
) {
    override fun getType(): TestDescriptor.Type = TestDescriptor.Type.TEST
}
```

Writing Test Engine: Discover


Tests p

Test Results 11 ms

Lviv: sample suspend test 11 ms

/home/

```
class MethodTestDescriptor(
    val function: KFunction<*>,
    val enclosureClass: KClass<*>
) : AbstractTestDescriptor(
    LVIV_ENGINE_UID.append("method", function.name),
    "Lviv: ${function.name}"
) {
    override fun getType(): TestDescriptor.Type = TestDescriptor.Type.TEST
}
```

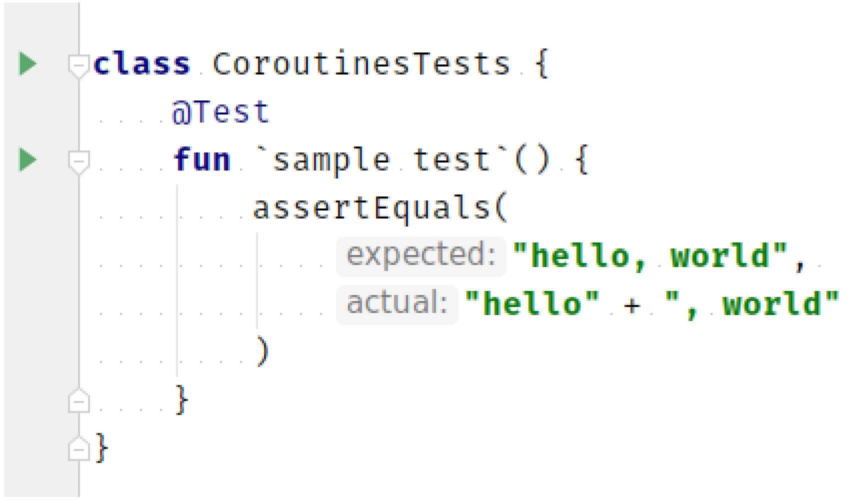
Writing Test Engine: Discover

```
class MethodTestDescriptor(
    val function: KFunction<*>,
    val enclosureClass: KClass<*>
) : AbstractTestDescriptor(
    LVIV_ENGINE_UID.append("method", function.name),
    "Lviv: ${function.name}"
) {
    override fun getType(): TestDescriptor.Type = TestDescriptor.Type.TEST
}
```

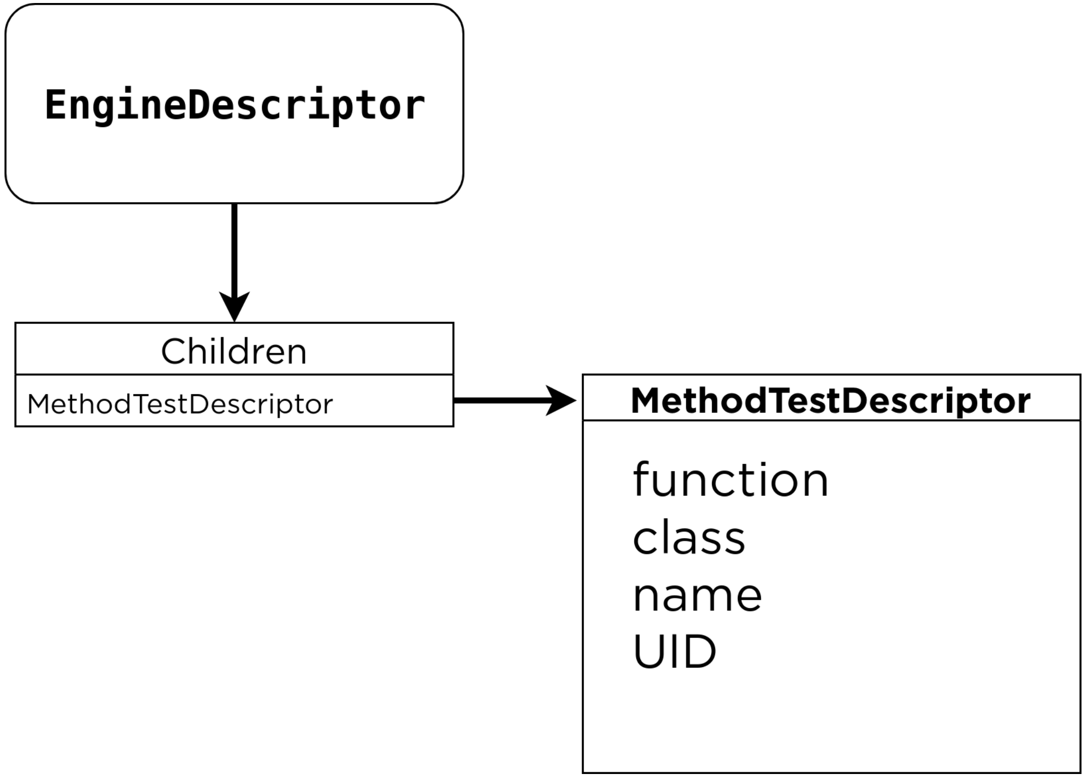
Writing Test Engine: Discover

```
class MethodTestDescriptor(
    val function: KFunction<*>,
    val enclosureClass: KClass<*>
) : AbstractTestDescriptor(
    LVIV_ENGINE_UID.append("method", function.name),
    "Lviv: ${function.name}"
) {
    override fun getType(): TestDescriptor.Type = TestDescriptor.Type.TEST
}
```

Writing Test Engine: Discover



```kotlin
class CoroutinesTests {
    @Test
    fun `sample test`() {
        assertEquals(
            expected: "hello, world",
            actual: "hello" + ", world"
        )
    }
}
```



Writing Test Engine: Execute

```kotlin
override fun execute(request: ExecutionRequest)
```

ExecutionRequest
TestDescriptor ExecutionListener

Writing Test Engine: Execute

```kotlin
override fun execute(request: ExecutionRequest) {
    val engine = request.rootTestDescriptor
    val listener = request.engineExecutionListener
    listener.executionStarted(engine)
    engine.children.forEach { child ->
        if (child is MethodTestDescriptor) {
            listener.executionStarted(child)
            try {
                runBlocking {
                    child.function.callSuspend(child.enclosureClass.createInstance())
                }
                listener.executionFinished(child, TestExecutionResult.successful())
            } catch (e: Throwable) {
                listener.executionFinished(child, TestExecutionResult.failed(e))
            }
        }
    }
    listener.executionFinished(engine, TestExecutionResult.successful())
}
```

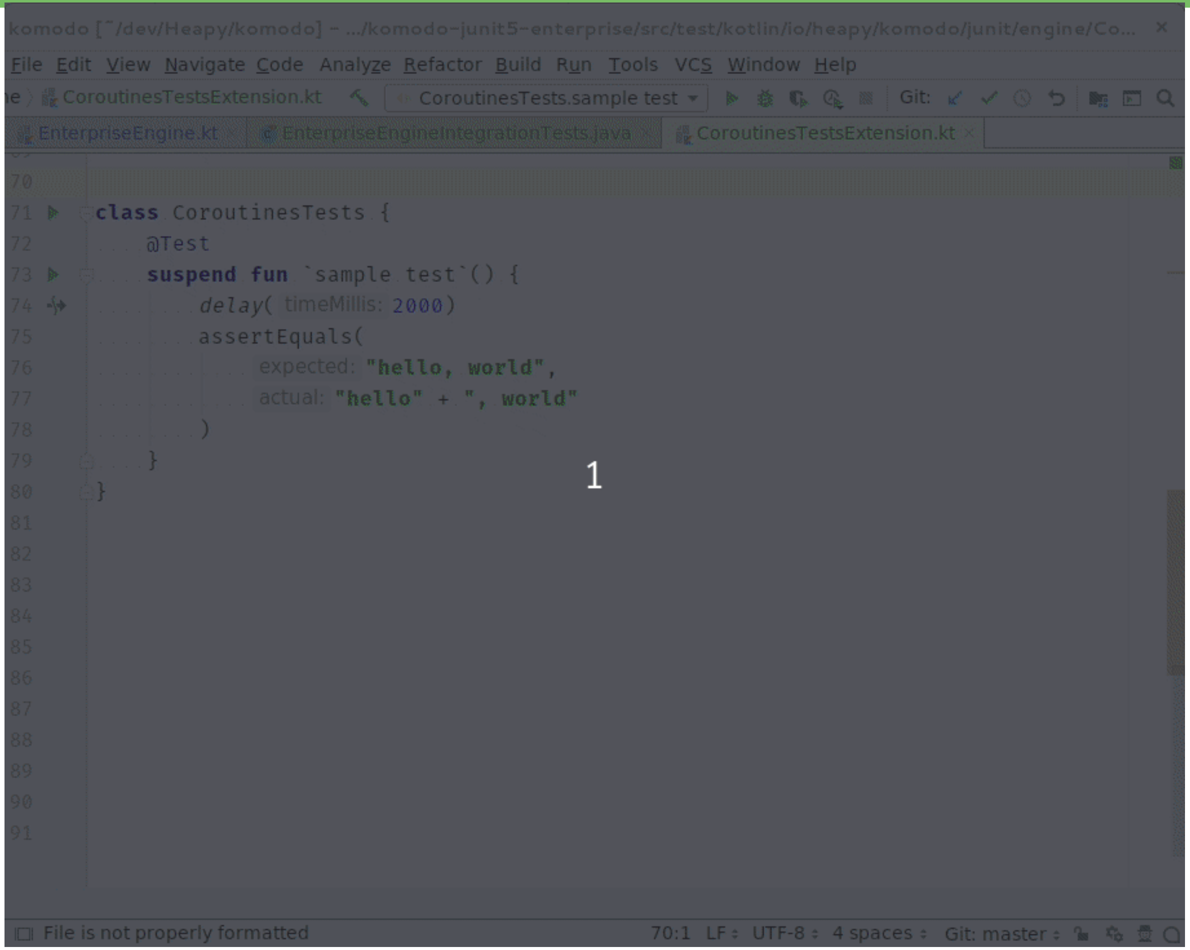
Writing Test Engine: Execute

```kotlin
override fun execute(request: ExecutionRequest) {
    val engine = request.rootTestDescriptor
    val listener = request.engineExecutionListener
    listener.executionStarted(engine)
    engine.children.forEach { child ->
        if (child is MethodTestDescriptor) {
            listener.executionStarted(child)
            try {
                runBlocking {
                    child.function.callSuspend(child.enclosureClass.createInstance())
                }
                listener.executionFinished(child, TestExecutionResult.successful())
            } catch (e: Throwable) {
                listener.executionFinished(child, TestExecutionResult.failed(e))
            }
        }
    }
    listener.executionFinished(engine, TestExecutionResult.successful())
}
```

Writing Test Engine: Execute

```kotlin
override fun execute(request: ExecutionRequest) {
    val engine = request.rootTestDescriptor
    val listener = request.engineExecutionListener
    listener.executionStarted(engine)
    engine.children.forEach { child →
        if (child is MethodTestDescriptor) {
            listener.executionStarted(child)
            try {
                runBlocking {
                    child.function.callSuspend(child.enclosureClass.createInstance())
                }
                listener.executionFinished(child, TestExecutionResult.successful())
            } catch (e: Throwable) {
                listener.executionFinished(child, TestExecutionResult.failed(e))
            }
        }
    }
    listener.executionFinished(engine, TestExecutionResult.successful())
}
```

Writing Test Engine: Execute


komodo [~/dev/Heapy/komodo] - .../komodo-junit5-enterprise/src/test/kotlin/io/heapy/komodo/junit/engine/Co...
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
CoroutinesTestsExtension.kt
CoroutinesTests.sample test
Git:
EnterpriseEngine.kt
EnterpriseEngineIntegrationTests.java
CoroutinesTestsExtension.kt

```kotlin
class CoroutinesTests {
    @Test
    suspend fun `sample test`() {
        delay( timeMillis: 2000)
        assertEquals(
            expected: "hello, world",
            actual: "hello" + ", world"
        )
    }
}
```

1

File is not properly formatted   70:1 LF UTF-8 4 spaces Git: master

But who monitors the monitor?

Should I cover tests with tests?

SO I HEARD YOU LIKE MONITORING
SO WE CONFIGURED A MONITOR TO MONITOR YOUR MONITOR
quickmeme.com

Writing Tests for Test Engine

```
testImplementation("org.junit.platform:junit-platform-testkit")
```

Writing Tests for Test Engine

```kotlin
@Test
fun `🆗 execute lviv kotlin engine`() {
    val discoveryRequest = request().selectors(DiscoverySelectors.selectMethod(
        LvivEngineTest::class.java,
        LvivEngineTest::`suspend test`.javaMethod
    )).build()
    val executionResults = EngineTestKit.execute(KotlinLvivEngine(), discoveryRequest)

    executionResults.all().assertStatistics { it.started(2).finished(2).succeeded(2) }
    executionResults.tests().assertStatistics { it.started(1).finished(1).failed(0) }

    val testDescriptor = executionResults.tests().succeeded().list().first().testDescriptor

    assertAll(
        { assertEquals("Lviv: suspend test", testDescriptor.displayName) },
        { assertEquals("Lviv: suspend test", testDescriptor.legacyReportingName) },
        { assertTrue(testDescriptor is MethodTestDescriptor) }
    )
}
```

Writing Tests for Test Engine

```
@Test
fun `🆗 execute lviv kotlin engine`() {
    val discoveryRequest = request().selectors(DiscoverySelectors.selectMethod(
        LvivEngineTest::class.java,
        LvivEngineTest::`suspend test`.javaMethod
    )).build()
    val executionResults = EngineTestKit.execute(KotlinLvivEngine(), discoveryRequest)

    executionResults.all().assertStatistics { it.started(2).finished(2).succeeded(2) }
    executionResults.tests().assertStatistics { it.started(1).finished(1).failed(0) }

    val testDescriptor = executionResults.tests().succeeded().list().first().testDescriptor

    assertAll(
        { assertEquals("Lviv: suspend test", testDescriptor.displayName) },
        { assertEquals("Lviv: suspend test", testDescriptor.legacyReportingName) },
        { assertTrue(testDescriptor is MethodTestDescriptor) }
    )
}
```

Writing Tests for Test Engine

```
@Test
fun `🆗 execute lviv kotlin engine`() {
    val discoveryRequest = request().selectors(DiscoverySelectors.selectMethod(
        LvivEngineTest::class.java,
        LvivEngineTest::`suspend test`.javaMethod
    )).build()
    val executionResults = EngineTestKit.execute(KotlinLvivEngine(), discoveryRequest)

    executionResults.all().assertStatistics { it.started(2).finished(2).succeeded(2) }
    executionResults.tests().assertStatistics { it.started(1).finished(1).failed(0) }

    val testDescriptor = executionResults.tests().succeeded().list().first().testDescriptor

    assertAll(
        { assertEquals("Lviv: suspend test", testDescriptor.displayName) },
        { assertEquals("Lviv: suspend test", testDescriptor.legacyReportingName) },
        { assertTrue(testDescriptor is MethodTestDescriptor) }
    )
}
```

JUnit 5: Meta annotations

```
@[Tag("slow") Test]
suspend fun `test get by email`() = runBlocking {
    val userApi = UserApi(HttpClient())

    val user = userApi.getByEmail("Andrey.Breslav@JetBrains.com")
    assertEquals("Andrey Breslav", user.name)
}
```

JUnit 5: Meta annotations

```
@Target(AnnotationTarget.CLASS, AnnotationTarget.FUNCTION)
@Retention(AnnotationRetention.RUNTIME)
@Tag("slow")
@Test
annotation class SlowTest
```

JUnit 5: Meta annotations

```
@SlowTest
suspend fun `test get by email`() = runBlocking {
    val userApi = UserApi(HttpClient())

    val user = userApi.getByEmail("Andrey.Breslav@JetBrains.com")
    assertEquals("Andrey Breslav", user.name)
}

// build.gradle.kts
tasks.test {
    useJUnitPlatform {
        excludeTags("slow")
    }
}
```

Let’s ~~Rock!~~ Mockk!

java.lang.IllegalArgumentException:
Callable expects 3 arguments, but 2 were provided.

```
@ExtendWith(MockKExtension::class)
class CoroutinesEngineTest {
    @Test
    suspend fun `co sample test`(@MockK userApi: UserApi) {
        coEvery { userApi.getByEmail("foo") } returns "bar"
        assertEquals(userApi.getByEmail("foo"), "bar")
    }
}
```

JUnit Jupiter

DI for constructors and methods

TestInstanceFactory

Parameterized test classes

@RegisterExtension

@Nested test classes

@RepeatedTest, @ParameterizedTest, @TestFactory

@TestInstance lifecycle management

...

Solution

"can I copy your homework?"

"yeah just change it up a bit so it doesn't look obvious you copied"

"ok"

Enterprise Engine

```kotlin
internal abstract class IsTestableMethod(
    private val annotationType: Class<out Annotation>,
    private val mustReturnVoid: Boolean
) : Predicate<Method> {

    override fun test(candidate: Method): Boolean {
        // Please do not collapse the following into a single statement.
        if (isStatic(candidate)) return false
        if (isPrivate(candidate)) return false
        if (isAbstract(candidate)) return false
        if (!isSuspend(candidate)) return false
        return isAnnotated(candidate, this.annotationType)
    }

    internal fun isSuspend(candidate: Method): Boolean {
        return candidate.kotlinFunction?.isSuspend ?: false
    }
}
```

Enterprise Engine

```
internal abstract class IsTestableMethod(
    private val annotationType: Class<out Annotation>,
    private val mustReturnVoid: Boolean
) : Predicate<Method> {

    override fun test(candidate: Method): Boolean {
        // Please do not collapse the following into a single statement.
        if (isStatic(candidate)) return false
        if (isPrivate(candidate)) return false
        if (isAbstract(candidate)) return false
        if (!isSuspend(candidate)) return false
        return isAnnotated(candidate, this.annotationType)
    }

    internal fun isSuspend(candidate: Method): Boolean {
        return candidate.kotlinFunction?.isSuspend ?: false
    }
}
```

Enterprise Engine

```kotlin
@Test
suspend fun `test get by email`(continuation: Continuation<*>)
```

```java
private Object resolveParameter(
        ParameterContext parameterContext,
        Executable executable,
        ExtensionContext extensionContext,
        ExtensionRegistry extensionRegistry
) {

    try {
        if (parameterContext.getParameter().getType().equals(Continuation.class)) {
            return null;
        }
        // ...
    }
}
```

Enterprise Engine

```kotlin
fun invokeMethod(method: Method, target: Any?, vararg args: Any): Any? {
    try {
        return runBlocking {
            makeAccessible(method)
                .kotlinFunction
                ?.callSuspend(target, *args.dropLast(1).toTypedArray())
        }
        // ...
    }
}
```

Let’s ~~Rock!~~ Mockk!

Test passed: 1

```
@ExtendWith(MockKExtension::class)
class CoroutinesEngineTest {
    @Test
    suspend fun `co sample test`(@MockK userApi: UserApi) {
        coEvery { userApi.getByEmail("foo") } returns "bar"
        assertEquals(userApi.getByEmail("foo"), "bar")
    }
}
```

Enterprise Engine

kotlin-coroutines-test

```kotlin
class AndroidTest {
    private val mainThreadSurrogate = newSingleThreadContext("UI thread")

    @BeforeEach
    fun setUp() {
        Dispatchers.setMain(mainThreadSurrogate)
    }

    @AfterEach
    fun tearDown() {
        Dispatchers.resetMain()
        mainThreadSurrogate.close()
    }

    @Test
    fun testSomeUI(): Unit = runBlocking {
        launch(Dispatchers.Main) {
            // Will be launched in the mainThreadSurrogate dispatcher
            // ...
        }

        Unit
    }
}
```

kotlin-coroutines-test

```
class MainDispatcherExtension : BeforeEachCallback, AfterEachCallback {
    private val mainThreadSurrogate = newSingleThreadContext("UI thread")

    override fun beforeEach(context: ExtensionContext) {
        Dispatchers.setMain(mainThreadSurrogate)
    }

    override fun afterEach(context: ExtensionContext?) {
        Dispatchers.resetMain()
        mainThreadSurrogate.close()
    }
}
```

kotlin-coroutines-test

```kotlin
@ExtendWith(MainDispatcherExtension::class)
class AndroidTest {
    @Test
    fun testSomeUI(): Unit = runBlocking {
        launch(Dispatchers.Main) {
            // Will be launched in the mainThreadSurrogate dispatcher
            //  ...
        }

        Unit
    }
}
```

kotlin-coroutines-test

Kotlin: Unresolved reference

```
@ExtendWith(MainDispatcherExtension::class)
class AndroidTest {
    @Test
    suspend fun testSomeUI() {
        launch(Dispatchers.Main) {
            // Will be launched in the mainThreadSurrogate dispatcher
            //  ...
        }
    }
}
```

kotlin-coroutines-test

```
@ExtendWith(MainDispatcherExtension::class)
class AndroidTest {
    @Test
    suspend fun testSomeUI() = coroutineScope {
        launch(Dispatchers.Main) {
            // Will be launched in the mainThreadSurrogate dispatcher
            //  ...
        }
    }
}
```

kotlin-coroutines-test

```
class AndroidTest {
    @Test
    suspend fun testSomeUI(scope: CoroutineScope) {
        scope.launch(Dispatchers.Main) {
            // Will be launched in the mainThreadSurrogate dispatcher
            //  ...
        }
    }
}
```

kotlin-coroutines-test

```
class AndroidTest {
    @Test
    suspend fun CoroutineScope.testSomeUI() {
        launch(Dispatchers.Main) {
            // Will be launched in the mainThreadSurrogate dispatcher
            // ...
        }
    }
}
```

kotlin-coroutines-test

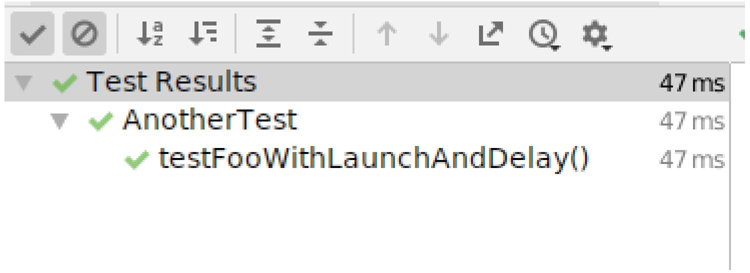
```kotlin
class AndroidTest {
    suspend fun testSomeUI(scope: CoroutineScope) {}
    // Equal on ByteCode level
    suspend fun CoroutineScope.testSomeUI() {}
}
```

kotlin-coroutines-test

```
@Test
fun testFooWithLaunchAndDelay() = runBlockingTest {
    foo()
    advanceTimeBy(1_000)
}

fun CoroutineScope.foo() {
    launch {
        println(1)
        delay(1_000)
        println(2)
    }
}
```

kotlin-coroutines-test

```kotlin
@Test
fun testFooWithLaunchAndDelay() = runBlockingTest {
    foo()
    advanceTimeBy(1_000)
}

fun CoroutineScope.foo() {
    launch {
        println(1)
        delay(1_000)
        println(2)
    }
}
```



```
▼ ✓ Test Results                      47 ms
  ▼ ✓ AnotherTest                     47 ms
      ✓ testFooWithLaunchAndDelay()   47 ms
```

kotlin-coroutines-test

```
@Test
fun TestCoroutineScope.testFooWithLaunchAndDelay() {
    foo()
    advanceTimeBy(1_000)
}

fun CoroutineScope.foo() {
    launch {
        println(1)
        delay(1_000)
        println(2)
    }
}
```

Enterprise Engine: Scopes

```
@Test
suspend fun `test get by email`(continuation: Continuation<*>)

private Object resolveParameter(
        ParameterContext parameterContext,
        Executable executable,
        ExtensionContext extensionContext,
        ExtensionRegistry extensionRegistry
) {

    try {
        if (parameterContext.getParameter().getType().equals(Continuation.class)) {
            return null;
        }
        // ...
    }
}
```

Enterprise Engine: Scopes

```kotlin
@Test
suspend fun `test get by email`(continuation: Continuation<*>)
```

↓

```kotlin
@Test
suspend fun `test get by email`(
    scope: CoroutineScope /* TestCoroutineScope */,
    continuation: Continuation<*>
)
```

Enterprise Engine: Scopes

```
if (parameterContext.getParameter().getType().equals(Continuation.class)) {
    return null;
}
```

⬇

```
if (parameterContext.getParameter().getType().equals(Continuation.class)) {
    return null;
}

if (parameterContext.getParameter().getType().equals(TestCoroutineScope.class)) {
    return TEST_COROUTINE_SCOPE;
}

if (parameterContext.getParameter().getType().equals(CoroutineScope.class)) {
    return COROUTINE_SCOPE;
}
```

Enterprise Engine: Scopes

```kotlin
fun invokeMethod(method: Method, target: Any?, vararg args: Any): Any? {
    try {
        return runBlocking {
            makeAccessible(method)
                .kotlinFunction
                ?.callSuspend(target, *args.dropLast(1).toTypedArray())
        }
        // ...
    }
}
```

Enterprise Engine: Scopes

```
val params = args.asList().dropLast(1)
if (params.contains(ExecutableInvoker.TEST_COROUTINE_SCOPE)) {
    return runBlockingTest {
        val callArgs = params.map {
            if (it == ExecutableInvoker.TEST_COROUTINE_SCOPE) this else it
        }.toTypedArray()

        makeAccessible(method).kotlinFunction?.callSuspend(target, *callArgs)
    }
} else if (params.contains(COROUTINE_SCOPE)) {
    return runBlocking {
        val callArgs = params.map {
            if (it == ExecutableInvoker.COROUTINE_SCOPE) this else it
        }.toTypedArray()

        makeAccessible(method).kotlinFunction?.callSuspend(target, *callArgs)
    }
} else {
    return runBlocking {
        makeAccessible(method).kotlinFunction?.callSuspend(target, *params.toTypedArray())
    }
}
```

Enterprise Engine: Scopes

```kotlin
val params = args.asList().dropLast(1)
if (params.contains(ExecutableInvoker.TEST_COROUTINE_SCOPE)) {
    return runBlockingTest {
        val callArgs = params.map {
            if (it == ExecutableInvoker.TEST_COROUTINE_SCOPE) this else it
        }.toTypedArray()

        makeAccessible(method).kotlinFunction?.callSuspend(target, *callArgs)
    }
} else if (params.contains(COROUTINE_SCOPE)) {
    return runBlocking {
        val callArgs = params.map {
            if (it == ExecutableInvoker.COROUTINE_SCOPE) this else it
        }.toTypedArray()

        makeAccessible(method).kotlinFunction?.callSuspend(target, *callArgs)
    }
} else {
    return runBlocking {
        makeAccessible(method).kotlinFunction?.callSuspend(target, *params.toTypedArray())
    }
}
```

Extensions

```
@Test
suspend fun `test get by email not found`() {
    val userApi = UserApi(HttpClient())

    assertThrows<UserNotFoundException> {
        userApi.getByEmail("ruslan@ibragimov.by")
    }
}
```

Kotlin: Suspend function 'getByEmail' should be called only from a coroutine or another suspend function

Extensions

- assertThrows

  -
    ```
    inline fun <reified T : Throwable> assertThrows(
        noinline executable: suspend () -> Unit
    ): T = Assertions.assertThrows(T::class.java, Executable {
        runBlocking {
            executable()
        }
    })
    ```
- assertAll

Performance

```
@Test
fun test1..1000() {
    assertEquals(1, 1)
}
```

175 ms

```
@Test
suspend fun TestCoroutineScope.test1..1000() {
    assertEquals(1, 1)
}
```

747 ms

```
@Test
fun test1..1000() = runBlockingTest {
    assertEquals(1, 1)
}
```

733 ms

Takeaway

JUnit 5 and Jupiter 👌

Writing own **TestEngine** is easy

But implement **Jupiter API** is not

Extensions FTW

Feedback Wanted!

`https://bit.ly/30ZOZDE`

Kotlin
Belarus
User Group
USE THE
KOTLIN