

if(kakao)2021

테스트 코드 한줄을 작성하기 까지의 고난

테스트하는 방법은 알려드리지 않습니다

조성준 Ronda.Ha
카카오

Test를 해야하는 이유

우리 서비스의 Architecture

테스트를 하기 위한 준비

테스트 측정

결론

Test를 해야하는 이유

우리 서비스의 Architecture

테스트를 하기 위한 준비

테스트 측정

결론

우리가 알고 있는 이유들

- 개발 과정에서 문제를 미리 발견할 수 있다.
- 리팩토링을 안심하고 할 수 있다.
- 빠른시간내에 코드의 동작 방식과 결과를 확인할 수 있다.
- 좋은 테스트 코드를 연습하다보면 자연스럽게 좋은 코드가 만들어진다.
- 의도한 대로 동작되는것을 자신감(?)있게 말할 수 있다.
- ...

또 다른 이유

Clean Agile - 로버트C. 마틴

애자일의 기술 실천 방법은 모든 애자일 활동 중 가장 핵심적인 요소다.

기술 실천 방법 없이 애자일을 도입하려는 시도는 실패할 수밖에 없다.

기술 실천 방법

TDD

리팩터링

단순한
설계

짝 프로그래밍

또 다른 이유

Clean Agile - 로버트C. 마틴

애자일의 기술 실천 방법은 모든 애자일 활동 중 가장 핵심적인 요소다.

기술 실천 방법 없이 애자일을 도입하려는 시도는 실패할 수밖에 없다.

기술 실천 방법

TDD

리팩터링

단순한
설계

짝 프로그래밍

또 다른 이유

Clean Agile - 로버트C. 마틴

애자일의 기술 실천 방법은 모든 애자일 활동 중 가장 핵심적인 요소다.

기술 실천 방법 없이 애자일을 도입하려는 시도는 실패할 수밖에 없다.

기술 실천 방법

TDD

리팩터링

단순한
설계

짜크 프로
그래밍

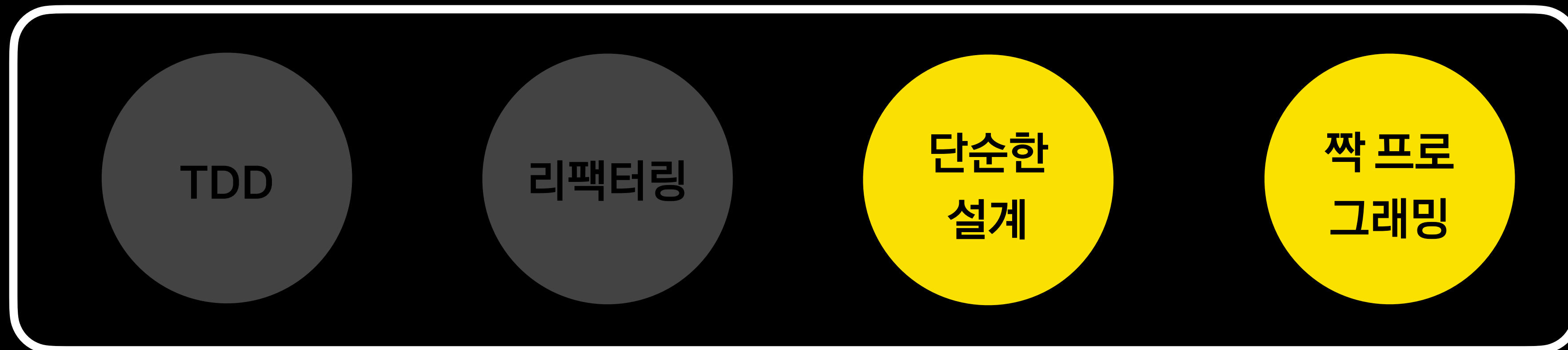
또 다른 이유

Clean Agile - 로버트C. 마틴

애자일의 기술 실천 방법은 모든 애자일 활동 중 가장 핵심적인 요소다.

기술 실천 방법 없이 애자일을 도입하려는 시도는 실패할 수밖에 없다.

기술 실천 방법



Test를 해야하는 이유

우리 서비스의 Architecture

테스트를 하기 위한 준비

테스트 측정

결론

적용되어 있는 것들

- Clean Architecture
- MVVM
- Multi Module
- 그리고 Coroutines, Hilt

과거와 현재의 모듈

과거

app(모듈)

- presentation
- domain
- data

현재

app

presentation

domain

data

과거와 현재의 모듈

과거

app(모듈)

- presentation
- domain
- data

현재

app

presentation

domain

data

어떤 모듈이?

– App

DI

Application Class

– Presentation

View

ViewModel

– Domain

Entity

Repository Interface

UseCase

– Data

Repository

DB

Service

어떤 모듈이?

– App

DI

Application Class

– Presentation

View

ViewModel

– Domain

Entity

Repository Interface

UseCase

– Data

Repository

DB

Service

어떤 모듈이?

– App

DI

Application Class

– Presentation

View

ViewModel

– Domain

Entity

Repository Interface

UseCase

– Data

Repository

DB

Service

어떤 모듈이?

– App

DI

Application Class

– Presentation

View

ViewModel

– Domain

Entity

Repository Interface

UseCase

– Data

Repository

DB

Service

테스트가 없다

좋은 Architecture를 잘 구성해
두었지만 테스트가 없다

Test를 해야하는 이유

우리 서비스의 Architecture

테스트를 하기 위한 준비

테스트 측정

결론

테스트를 하기 위한 준비

- ViewModel Test
- Coroutines Test
- JUnit5에서 변경점
- 빨간 막대에서 초록 막대를 보기까지 걸린 시간

테스트를 하기 위한 준비

- ViewModel Test
- Coroutines Test
- JUnit5에서 변경점
- 빨간 막대에서 초록 막대를 보기까지 걸린 시간

테스트를 하기 위한 준비

- ViewModel Test

- LiveData 변경에 대한 테스트

- 비동기 작업에 대한 테스트

- Coroutines Test

- JUnit5에서 변경점

- 빨간 막대에서 초록 막대를 보기까지 걸린 시간

```

@HiltViewModel
class SomethingViewModel @Inject constructor():
    ViewModel() {

        private val _somethingEvent = MutableLiveData<String>()
        val somethingEvent: LiveData<String> get() =
            _somethingEvent

        fun somethingMethod() {
            _somethingEvent.value = "something"
        }
    }

```

```

class SomethingViewModelTest {

    @Test
    fun somethingTestCase() {

        val somethingViewModel = SomethingViewModel()
        somethingViewModel.somethingMethod()

        val expected = "something"
        assertEquals(expected,
            somethingViewModel.somethingEvent.value)
    }
}

```

Method `getMainLooper` in `android.os.Looper` not mocked. See <http://g.co/android-dev/2014/04/03/testing>
 java.lang.RuntimeException: Method `getMainLooper` in `android.os.Looper` not
 at `android.os.Looper.getMainLooper(Looper.java)`

테스트는 다른 thread에서 실행되기 때문에 실패한다

```
@MainThread
```

```
protected void setValue(T value) {  
    assertMainThread("setValue");  
    mVersion++;  
    mData = value;  
    dispatchingValue(null);  
}
```

```
static void assertMainThread(String methodName) {  
    if (!ArchTaskExecutor.getInstance().isMainThread()) {  
        throw new IllegalStateException("Cannot invoke " + methodName + " on a background thread");  
    }  
}
```

```
public class InstantTaskExecutorRule extends TestWatcher {
    @Override
    protected void starting(Description description) {
        super.starting(description);
        ArchTaskExecutor.getInstance().setDelegate(new TaskExecutor() {
            @Override
            public void executeOnDiskIO(Runnable runnable) {
                runnable.run();
            }

            @Override
            public void postToMainThread(Runnable runnable) {
                runnable.run();
            }

            @Override
            public boolean isMainThread() {
                return true;
            }
        });
    }

    @Override
    protected void finished(Description description) {
        super.finished(description);
        ArchTaskExecutor.getInstance().setDelegate(null);
    }
}
```



```
class SomethingViewModelTest {
```

```
    @get:Rule
```

```
    var instantExecutorRule = InstantTaskExecutorRule()
```

```
    @Test
```

```
    fun somethingTestCase() {
```

```
        val somethingViewModel = SomethingViewModel()
```

```
        somethingViewModel.somethingMethod()
```

```
        val expected = "something"
```

```
        assertEquals(expected, somethingViewModel.somethingEvent.value)
```

```
    }
```

```
}
```

테스트를 하기 위한 준비

- ViewModel Test
- Coroutines Test
- JUnit5에서 변경점
- 빨간 막대에서 초록 막대를 보기까지 걸린 시간

테스트를 하기 위한 준비

- ViewModel Test
- Coroutines Test
 - viewModelScope.launch 에서 테스트
 - 다른 Dispatcher가 있을 때 테스트
- JUnit5에서 변경점
- 빨간 막대에서 초록 막대를 보기까지 걸린 시간

```

@HiltViewModel
class SomethingViewModel @Inject constructor():
ViewModel() {

    private val _somethingEvent =
MutableLiveData<String>()
    val somethingEvent: LiveData<String> get() =
_somethingEvent

    fun somethingMethod() {
        viewModelScope.launch {
            _somethingEvent.value = "something"
        }
    }
}

```

```

class SomethingViewModelTest {

    @Test
    fun somethingTestCase() {

        val somethingViewModel = SomethingViewModel()
        somethingViewModel.somethingMethod()

        val expected = "something"
        assertEquals(expected,somethingViewModel.somethingEvent.value)
    }
}

```

```

Exception in thread "main @coroutine#1" java.lang.IllegalStateException Create breakpoint : Module with the Main dispatcher had failed to initialize. For tests Dispatchers.setMain from kotlinx-coroutines-test module can be used
at kotlinx.coroutines.internal.MissingMainCoroutineDispatcher.missing(MainDispatchers.kt:96)
at kotlinx.coroutines.internal.MissingMainCoroutineDispatcher.isDispatchNeeded(MainDispatchers.kt:71)
at kotlinx.coroutines.DispatchedKt.resumeCancellable(Dispatched.kt:420)
at kotlinx.coroutines.intrinsics.CancellableKt.startCoroutineCancellable(Cancellable.kt:26)
at kotlinx.coroutines.CoroutineStart.invoke(CoroutineStart.kt:109)
at kotlinx.coroutines.AbstractCoroutine.start(AbstractCoroutine.kt:154)
at kotlinx.coroutines.BuildersKt__Builders_commonKt.launch(Builders.common.kt:54)
at kotlinx.coroutines.BuildersKt.launch(Unknown Source)

```

Test thread

Test Body

Kotlinx-coroutines-test 에서 제공하는 TestCoroutineDispatcher를 사용해보자

```
private val testDispatcher = TestCoroutineDispatcher()
```

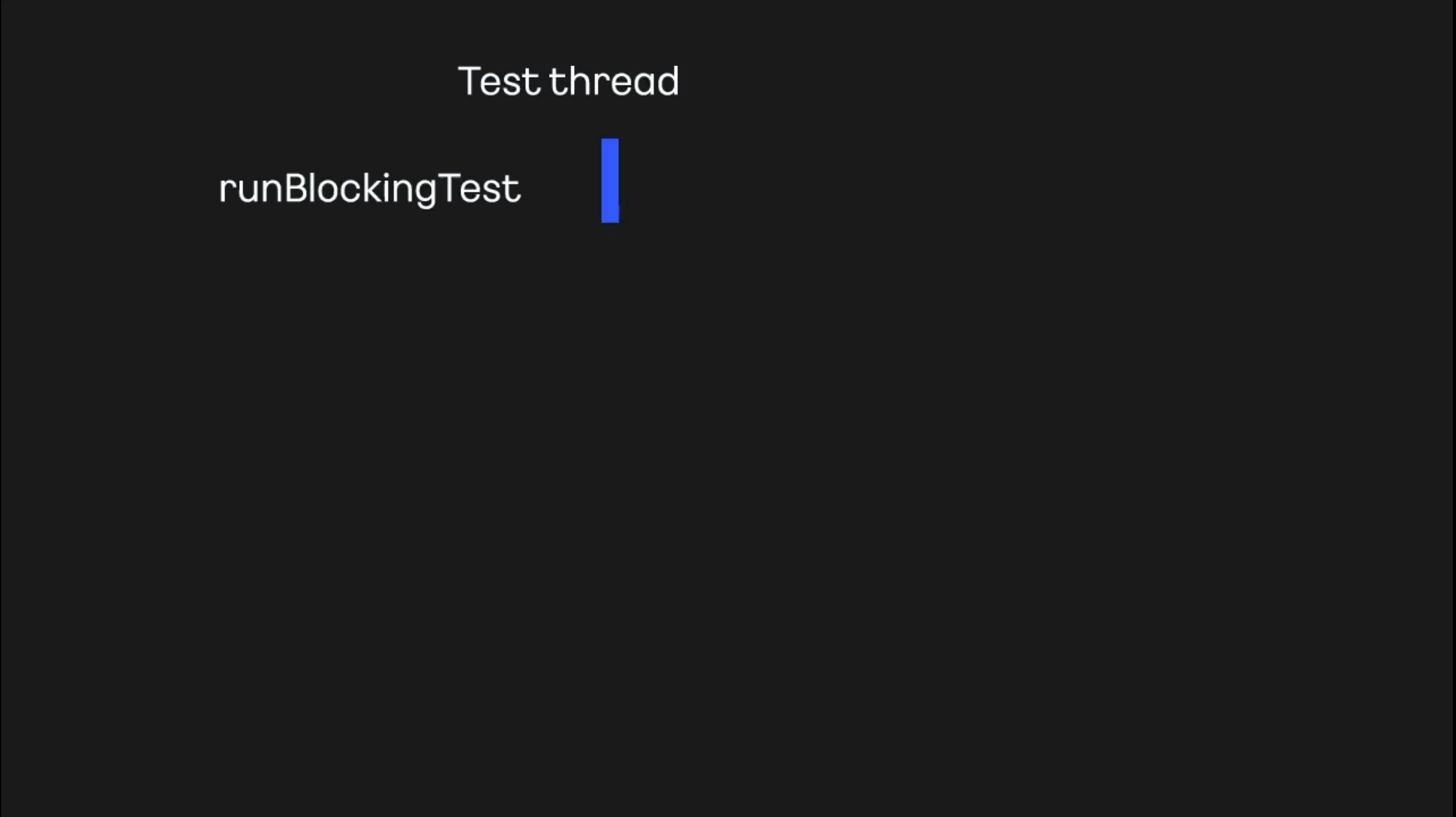
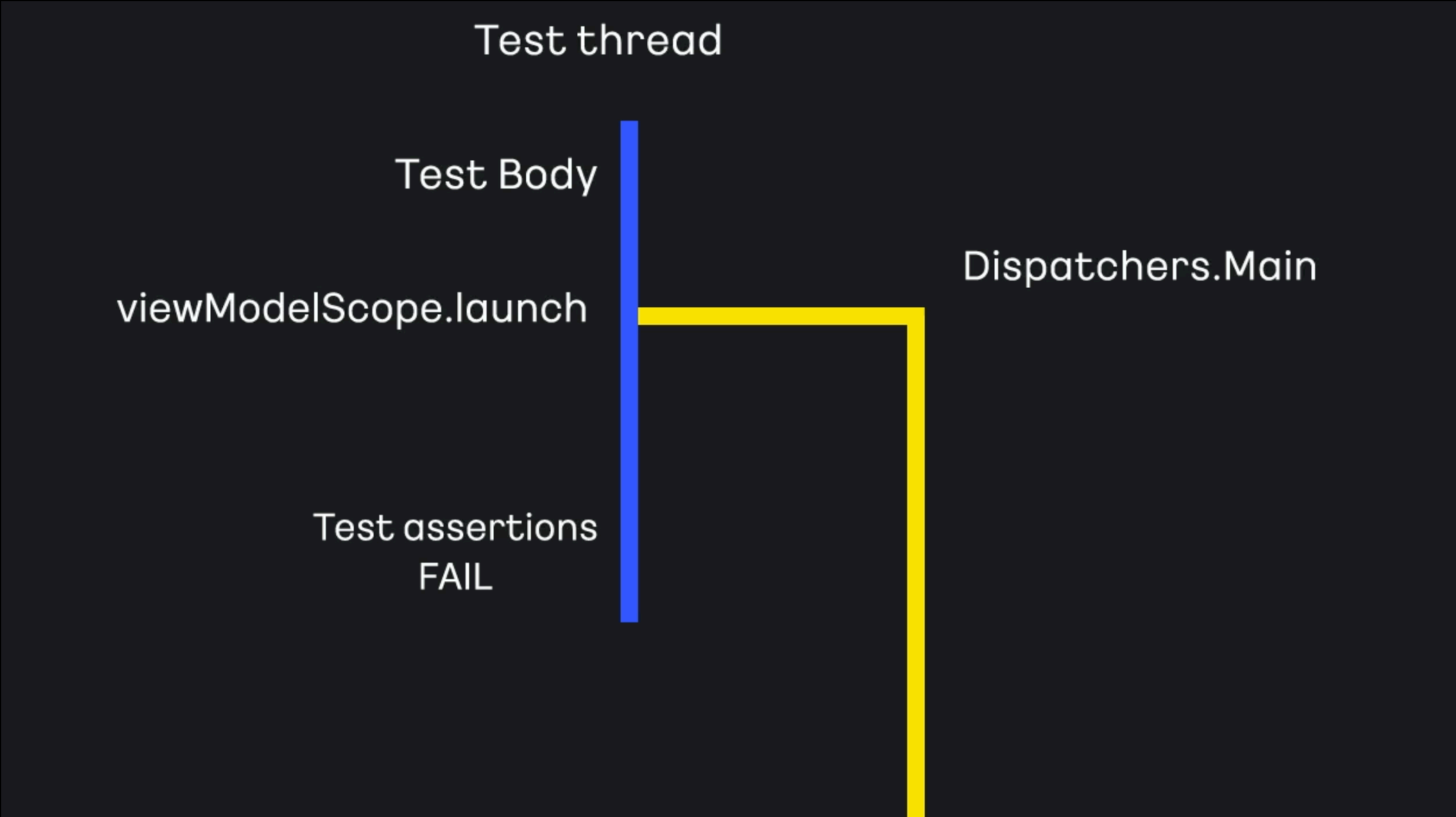
```
@Before
```

```
fun setUp() {  
    Dispatchers.setMain(testDispatcher)  
}
```

```
@After
```

```
fun tearDown() {  
    Dispatchers.resetMain()  
    testDispatcher.cleanupTestCoroutines()  
}
```

수정 후 흐름도 비교



```
@ExperimentalCoroutinesApi
class MainCoroutineRule(
    val testDispatcher: TestCoroutineDispatcher = TestCoroutineDispatcher()
): TestWatcher() {

    override fun starting(description: Description?) {
        super.starting(description)
        Dispatchers.setMain(testDispatcher)
    }

    override fun finished(description: Description?) {
        super.finished(description)
        Dispatchers.resetMain()
        testDispatcher.cleanupTestCoroutines()
    }

    fun runBlockingTest(block: suspend TestCoroutineScope.() -> Unit) {
        return testDispatcher.runBlockingTest(block)
    }
}
```



```
class SomethingViewModelTest {
```

```
    @get:Rule
```

```
    var instantExecutorRule = InstantTaskExecutorRule()
```

```
    @get:Rule
```

```
    var mainCoroutineRule = MainCoroutineRule()
```

```
    @Test
```

```
    fun somethingTestCase() {
```

```
        val somethingViewModel = SomethingViewModel()
```

```
        somethingViewModel.somethingMethod()
```

```
        val expected = "something"
```

```
        assertEquals(expected, somethingViewModel.somethingEvent.value)
```

```
    }
```

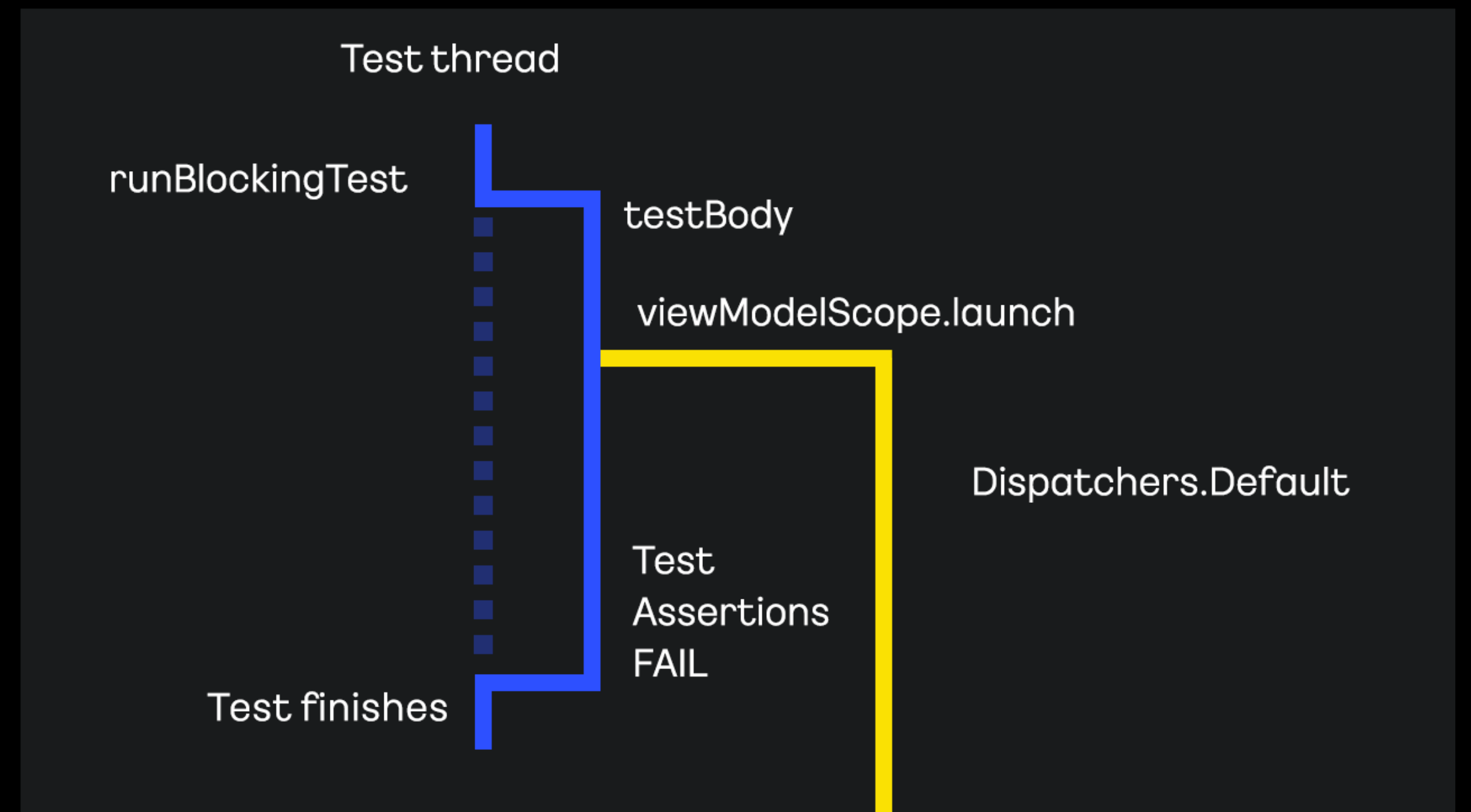
```
}
```

새로운 Dispatchers가 있을 때

```
@HiltViewModel
class SomethingViewModel @Inject constructor():
    ViewModel() {

        private val _somethingEvent =
            MutableLiveData<String>()
        val somethingEvent: LiveData<String>
            get() = _somethingEvent

        fun somethingMethod() {
            viewModelScope.launch(Dispatchers.Default) {
                _somethingEvent.value = "something"
            }
        }
    }
}
```

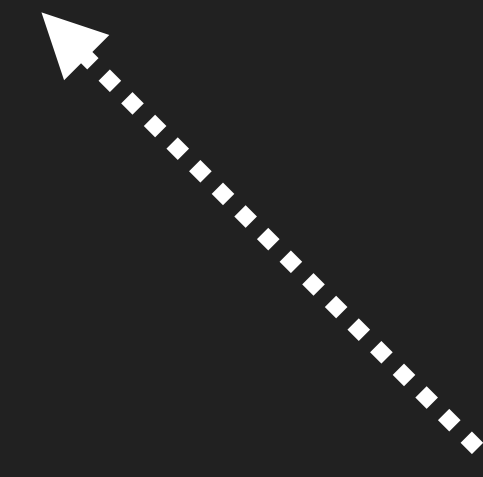
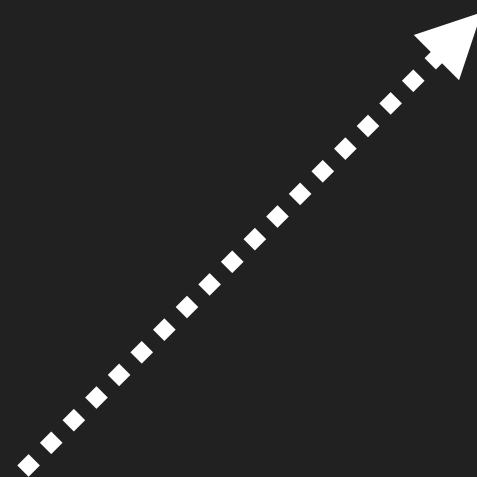


```
@HiltViewModel
class SomethingViewModel @Inject constructor(
    private val dispatcher: DispatcherProvider
): ViewModel() {

    private val _somethingEvent = MutableLiveData<String>()
    val somethingEvent: LiveData<String> get() = _somethingEvent

    fun somethingMethod() {
        viewModelScope.launch(dispatcher.default) {
            _somethingEvent.value = "something"
        }
    }
}
```

```
interface DispatcherProvider {  
    val default: CoroutineDispatcher  
    val io: CoroutineDispatcher  
    val main: CoroutineDispatcher  
}
```



```
class TestDispatcherProvider(private val testDispatcher:  
TestCoroutineDispatcher): DispatcherProvider {  
    override val default: CoroutineDispatcher  
        get() = testDispatcher  
    override val io: CoroutineDispatcher  
        get() = testDispatcher  
    override val main: CoroutineDispatcher  
        get() = testDispatcher  
}
```

```
class DefaultDispatcherProvider : DispatcherProvider {  
    override val default: CoroutineDispatcher  
        get() = Dispatchers.Default  
    override val io: CoroutineDispatcher  
        get() = Dispatchers.IO  
    override val main: CoroutineDispatcher  
        get() = Dispatchers.Main  
}
```

```
@ExperimentalCoroutinesApi
```

```
class SomethingViewModelTest {
```

```
    @get:Rule
```

```
    var instantExecutorRule = InstantTaskExecutorRule()
```

```
    @get:Rule
```

```
    var mainCoroutineRule = MainCoroutineRule()
```

```
    @Test
```

```
    fun somethingTestCase() {
```

```
        val testDispatcherProvider = TestDispatcherProvider(mainCoroutineRule.testDispatcher)
```

```
        val somethingViewModel = SomethingViewModel(testDispatcherProvider)
```

```
        somethingViewModel.somethingMethod()
```

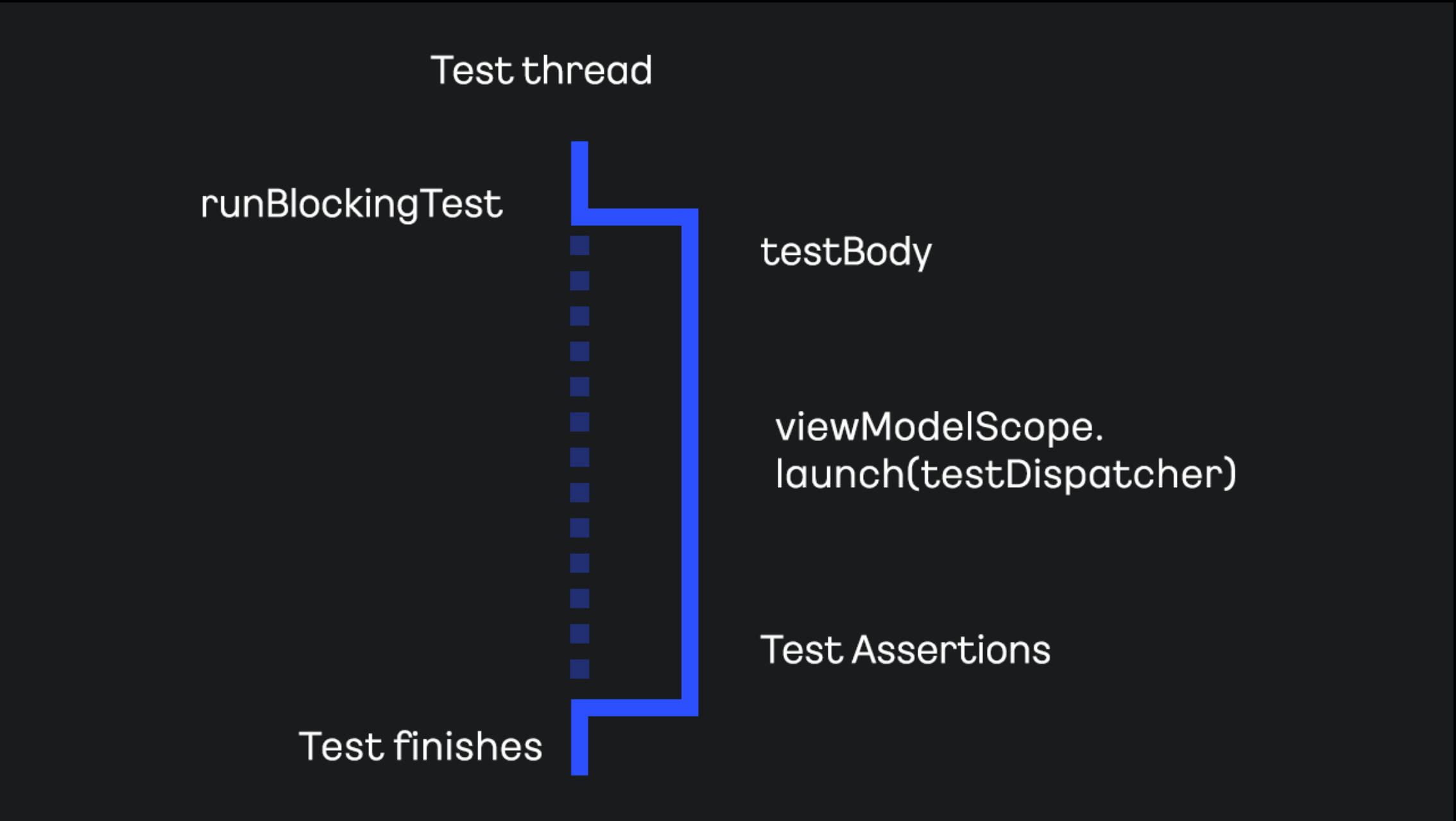
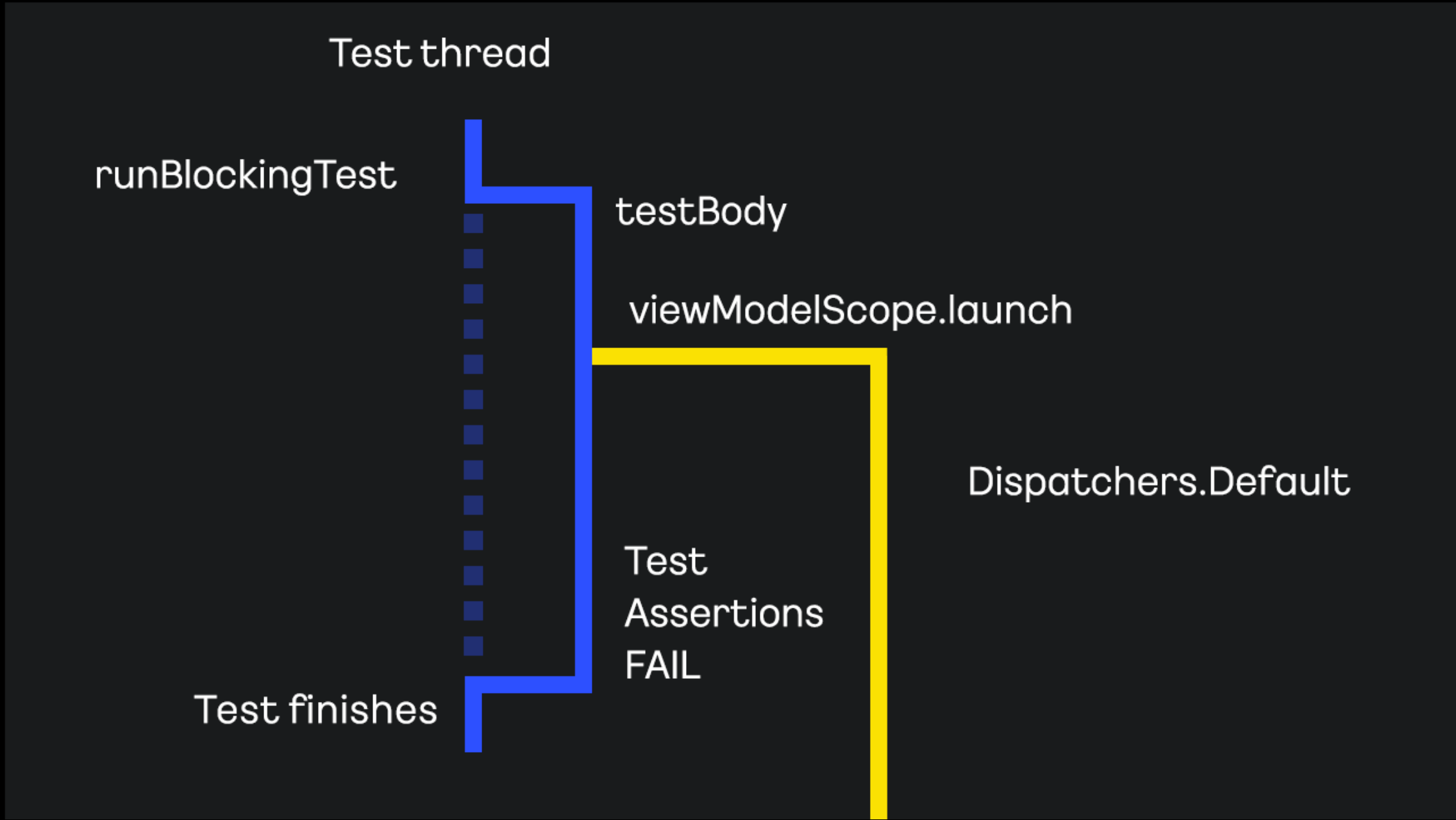
```
        val expected = "something"
```

```
        assertEquals(expected, somethingViewModel.somethingEvent.value)
```

```
    }
```

```
}
```

수정 후 흐름도 비교



```
fun somethingMethod() {  
  
    _progressEvent.value = true  
  
    viewModelScope.launch {  
        _somethingEvent.value = "something"  
        _progressEvent.value = false  
    }  
}
```

@Test

```
fun somethingTestCase() {
```

```
    val somethingViewModel = SomethingViewModel()  
    somethingViewModel.somethingMethod()
```

```
    assertTrue(somethingViewModel.progressEvent.value?: false)
```

```
    assertFalse(somethingViewModel.progressEvent.value?: true)
```

```
}
```



```
fun somethingMethod() {
```

```
    _progressEvent.value = true
```

Check!!

```
    viewModelScope.launch {
```

```
        _somethingEvent.value = "something"
```

```
        _progressEvent.value = false
```

Check!!

```
    }
```

```
}
```



```

@Test
fun somethingTestCase() = mainCoroutineRule.runBlockingTest {

    val somethingViewModel = SomethingViewModel()

    pauseDispatcher()                testDispatcher 정지

    somethingViewModel.somethingMethod()

    assertTrue(somethingViewModel.progressEvent.value?: false)
                                                                    Check!!

    resumeDispatcher()              testDispatcher 시작

    assertFalse(somethingViewModel.progressEvent.value?: true)
                                                                    Check!!
}

```

```

fun somethingMethod() {

    _progressEvent.value = true

    정작 viewModelScope.launch {
        _somethingEvent.value = "something"

        _progressEvent.value = false

    }

}

```

테스트를 하기 위한 준비

- ViewModel Test
- Coroutines Test
- JUnit5에서 변경점
- 빨간 막대에서 초록 막대를 보기까지 걸린 시간

```

public class InstantTaskExecutorRule extends TestWatcher {
    @Override
    protected void starting(Description description) {
        super.starting(description);
        ArchTaskExecutor.getInstance().setDelegate(new TaskExecutor() {
            @Override
            public void executeOnDiskIO(Runnable runnable) {
                runnable.run();
            }

            @Override
            public void postToMainThread(Runnable runnable) {
                runnable.run();
            }

            @Override
            public boolean isMainThread() {
                return true;
            }
        });
    }

    @Override
    protected void finished(Description description) {
        super.finished(description);
        ArchTaskExecutor.getInstance().setDelegate(null);
    }
}

```

```

class InstantTaskExecutorExtension: BeforeEachCallback, AfterEachCallback {
    override fun beforeEach(context: ExtensionContext?) {
        ArchTaskExecutor.getInstance().setDelegate(object : TaskExecutor(){
            override fun executeOnDiskIO(runnable: Runnable) {
                runnable.run()
            }

            override fun postToMainThread(runnable: Runnable) {
                runnable.run()
            }

            override fun isMainThread(): Boolean {
                return true
            }
        })
    }

    override fun afterEach(context: ExtensionContext?) {
        ArchTaskExecutor.getInstance().setDelegate(null)
    }
}

```

```
@ExperimentalCoroutinesApi
@ExtendWith(MockitoExtension::class, InstantTaskExecutorExtension::class)
class SomethingViewModelTest {

    //    @get:Rule
    //    var instantExecutorRule = InstantTaskExecutorRule()

    @get:Rule
    var mainCoroutineRule = MainCoroutineRule()

    @Test
    fun somethingTestCase() {

        val somethingViewModel = SomethingViewModel()
        somethingViewModel.somethingMethod()

        val expected = "something"
        assertEquals(expected, somethingViewModel.somethingEvent.value)
    }
}
```

```

@ExperimentalCoroutinesApi
class MainCoroutineRule(
    val testDispatcher: TestCoroutineDispatcher = TestCoroutineDispatcher()
): TestWatcher() {

    override fun starting(description: Description?) {
        super.starting(description)
        Dispatchers.setMain(testDispatcher)
    }

    override fun finished(description: Description?) {
        super.finished(description)
        Dispatchers.resetMain()
        testDispatcher.cleanupTestCoroutines()
    }

    fun runBlockingTest(block: suspend TestCoroutineScope.() -> Unit) {
        return testDispatcher.runBlockingTest(block)
    }
}

@ExperimentalCoroutinesApi
class MainCoroutineExtension(
    val testDispatcher: TestCoroutineDispatcher = TestCoroutineDispatcher()
): BeforeEachCallback, AfterEachCallback {

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

    override fun afterEach(context: ExtensionContext?) {
        Dispatchers.resetMain()
        testDispatcher.cleanupTestCoroutines()
    }

    fun runBlockingTest(block: suspend TestCoroutineScope.() -> Unit) {
        return testDispatcher.runBlockingTest(block)
    }
}

```

```
@ExperimentalCoroutinesApi
@ExtendWith(MockitoExtension::class, InstantTaskExecutorExtension::class)
class SomethingViewModelTest {
```

```
    // @get:Rule
    // var instantExecutorRule = InstantTaskExecutorRule()
```

```
    // @get:Rule
    // var mainCoroutineRule = MainCoroutineRule()
```

```
    companion object {
        @JvmField
        @RegisterExtension
        val mainCoroutineExtension = MainCoroutineExtension()
    }
```

```
    @Test
```

```
    fun somethingTestCase() = mainCoroutineExtension.runBlockingTest {
```

```
        val somethingViewModel = SomethingViewModel(mainCoroutineExtension.testDispatcher)
```

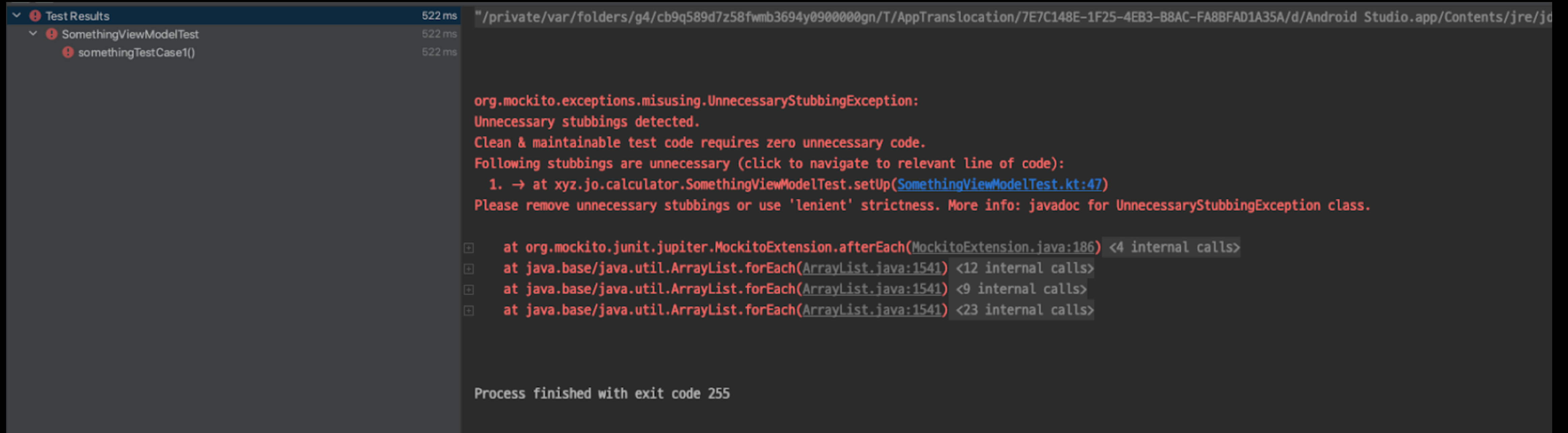
```
        somethingViewModel.somethingMethod()
```

```
        val expected = "something"
        assertEquals(expected, somethingViewModel.somethingEvent.value)
```

```
    }
```

```
}
```

UnnecessaryStubbingException



The screenshot shows the Test Results pane in Android Studio. The test 'somethingTestCase1()' in 'SomethingViewModelTest' failed with an 'UnnecessaryStubbingException' after 522 ms. The exception message states that unnecessary stubbings were detected and provides a list of them. The first item is a stub for 'xyz.jo.calculator.SomethingViewModelTest.setUp()' at line 47 of 'SomethingViewModelTest.kt'. The stack trace shows the exception was thrown from MockitoExtension, with internal calls from ArrayList.forEach(). The process finished with exit code 255.

```
Test Results 522 ms
  SomethingViewModelTest 522 ms
    somethingTestCase1() 522 ms

org.mockito.exceptions.misusing.UnnecessaryStubbingException:
Unnecessary stubbings detected.
Clean & maintainable test code requires zero unnecessary code.
Following stubbings are unnecessary (click to navigate to relevant line of code):
  1. → at xyz.jo.calculator.SomethingViewModelTest.setUp(SomethingViewModelTest.kt:47)
Please remove unnecessary stubbings or use 'lenient' strictness. More info: javadoc for UnnecessaryStubbingException class.

at org.mockito.junit.jupiter.MockitoExtension.afterEach(MockitoExtension.java:186) <4 internal calls>
at java.base/java.util.ArrayList.forEach(ArrayList.java:1541) <12 internal calls>
at java.base/java.util.ArrayList.forEach(ArrayList.java:1541) <9 internal calls>
at java.base/java.util.ArrayList.forEach(ArrayList.java:1541) <23 internal calls>

Process finished with exit code 255
```

```
@Test
```

```
fun somethingTestCase() = mainCoroutineExtension.runBlockingTest {
```

```
    whenever(somethingRepository.getString()).thenReturn("something")
```

```
    val somethingViewModel = SomethingViewModel()  
    somethingViewModel.somethingMethod()
```

```
    val expected = "something"  
    assertEquals(expected, somethingViewModel.somethingEvent.value)
```

```
}
```

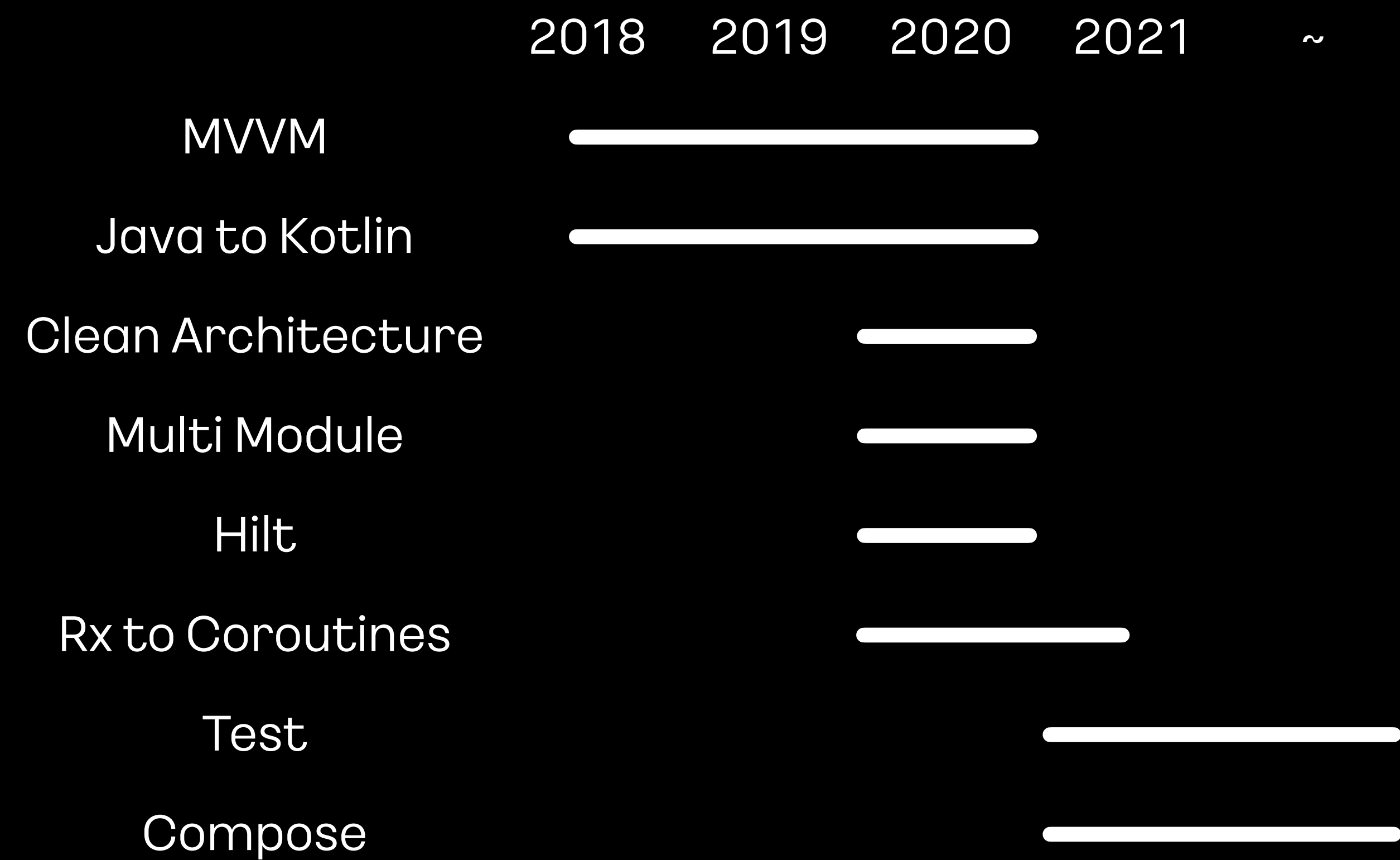

회피하는 방법

1. `lenient().when'(somethingRepository.getSomething()).thenReturn("")`
2. `@RunWith(MockitoJUnitRunner.Silent::class)`
3. `@MockitoSettings(strictness = Strictness.LENIENT)`

테스트를 하기 위한 준비

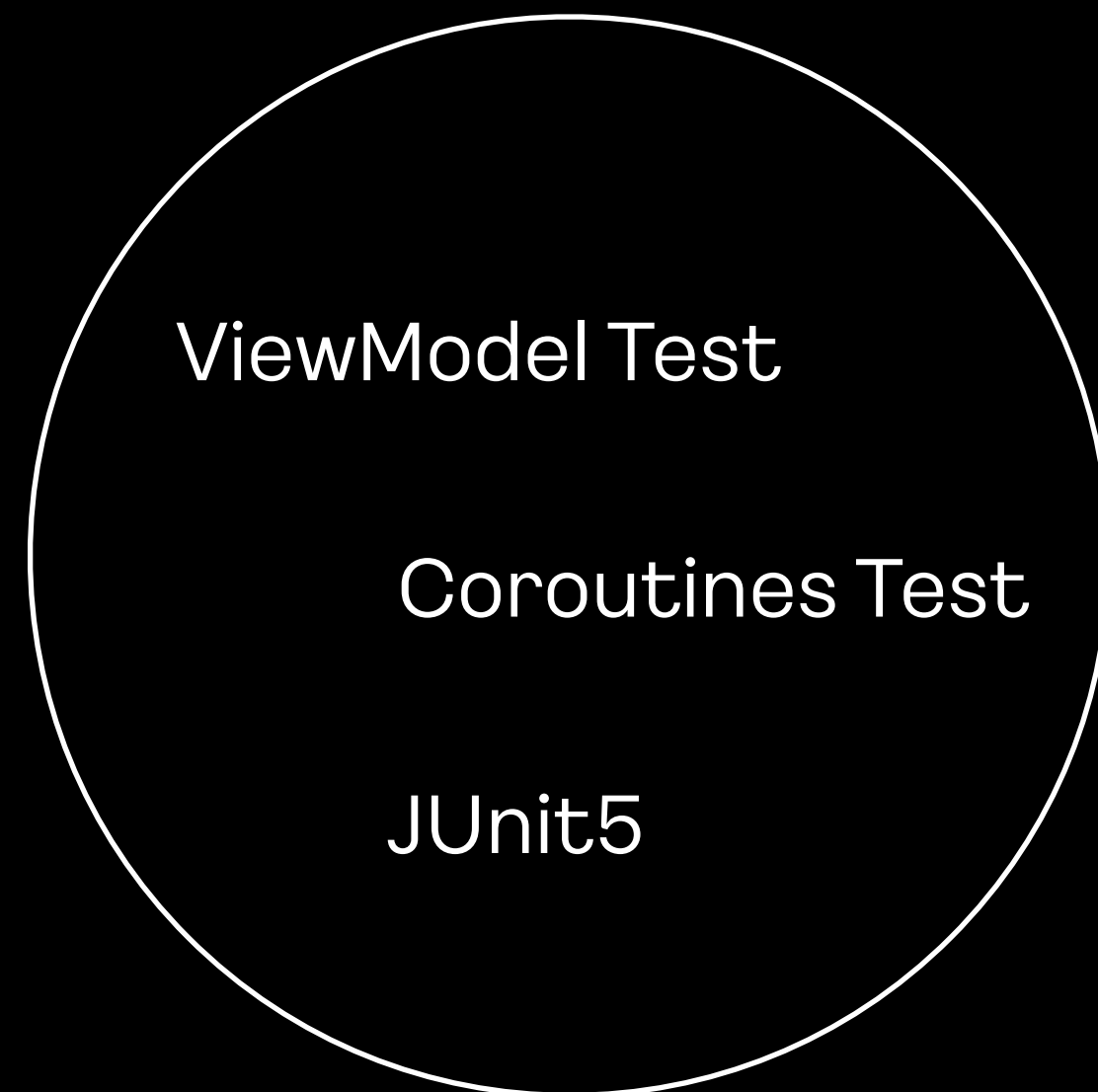
- ViewModel Test
- Coroutines Test
- JUnit5에서 변경점
- 빨간 막대에서 초록 막대를 보기까지 걸린 시간

천천히 Legacy 제거



파악, 학습, 적용 까지 시간

3주



1주



Test를 해야하는 이유

우리 서비스의 Architecture

테스트를 하기 위한 준비

테스트 측정

결론

무엇을 어떻게 측정하는가

1. 왜 ViewModel만 테스트 하고 있을까?


- UI 테스트가 어렵다
- 각 Layer 사이에 Mapper가 아직 완벽하지 않아서 Data Layer의 테스트를 미룸
- 다 하기엔 너무 많다...

2. Jacoco를 이용한 Test Coverage

- 결과물에 ViewModel Class 만 나오게 수정

언제 측정 하는가

Artifacts


Name 

Brunch_4.0.9_4090_coverage.zip

brunchapp-live-release.aab

brunchapp-live-release.apk

20210803_092534

 AndroidManifest

날짜	버전	Missed Instruction	Instruction	coverage (%)
2021.07.06	4.0.7_4071	37,580	38,098	1%
2021.07.29	4.0.8	36,114	37,916	4%
2021.08.04	4.1.0	35,620	38,189	6%
2021.08.25	4.1.1	35,613	39,116	8%

MoBil에서 Release 빌드 시 자동으로 테스트 후 측정, 압축 파일 생성

Test를 해야하는 이유

우리 서비스의 Architecture

테스트를 하기 위한 준비

테스트 측정

결론

결론

1. 할 수 있었던 이유? 방법? 배경?

- 급하지 않게 레거시 제거
- 샘플작성, 필요하면 라이트닝 토크
- 몸 프로그래밍
- 본인이 하고 싶은것

2. 현재, 미래

- 현재는 8% 이지만, 목표 Coverage 없이 오늘 보다 나은 내일이 목표
- Domain, Data layer 까지 테스트 하는게 목표

감사합니다.