JUnit5 to the Rescue!

JUnit5: Reuse the Test Class Instance

```
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
class RepositoryTest {
    private val mongo = startMongoContainer().apply {
        configure()
    }
    private val repo = Repository(mongo.host, mongo.port)

    @Test
    fun test1() { }

}
```

Concise

Idiomatic

JUnit5: Reuse the Test Class Instance

```
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
class RepositoryTest {
    private val mongo: GenericContainer
    private val repo: Repository
    init {
        mongo = startMongoContainer().apply {
            configure()
        }
        repo = Repository(mongo.host, mongo.port)
    }
}
```

JUnit5: Change the Lifecycle Default

src/test/resources/junit-platform.properties:

```
junit.jupiter.testinstance.lifecycle.default = per_class
```

~~@TestInstance(TestInstance.Lifecycle.PER_CLASS)~~

Classes Are Final by Default

Solutions

- Interfaces
- `open` explicitly
- Mockito: Enable incubating feature to mock final classes
- **MockK**

MockK


```
verifySequence {
    clientMock.getUser(2)
    repoMock.saveUser(user)
}
```

```
java.lang.AssertionError: Verification failed: calls are
not exactly matching verification sequence

Matchers:
UserClient(#5).getUser(eq(2))
UserRepo(#4).saveUser(eq(User(id=1, name=Ben, age=29))))

Calls:
1) UserClient(#5).getUser(1)
2) UserRepo(#4).saveUser(User(id=1, name=Ben, age=29))
```

Data Classes for Parameterized Tests

```
data class TestData(
    val input: String?,
    val expected: Token?
)
```

Conclusion

Best Practices for Testing in Kotlin

JUnit5 ♥ Kotlin

@TestInstance(PER_CLASS)

Naming, Grouping

Backticks

@Nested

Libraries

Choose your own gear

Mock Handling

Don't recreate; reset!

MockK

Data Classes FTW

Equals Assertions

Creation Helper

@ParameterizedTest

Spreadshirt

For two years