Google Developer Experts

Refactoring and Test Fakes

Crafting Resilient Code with Confidence

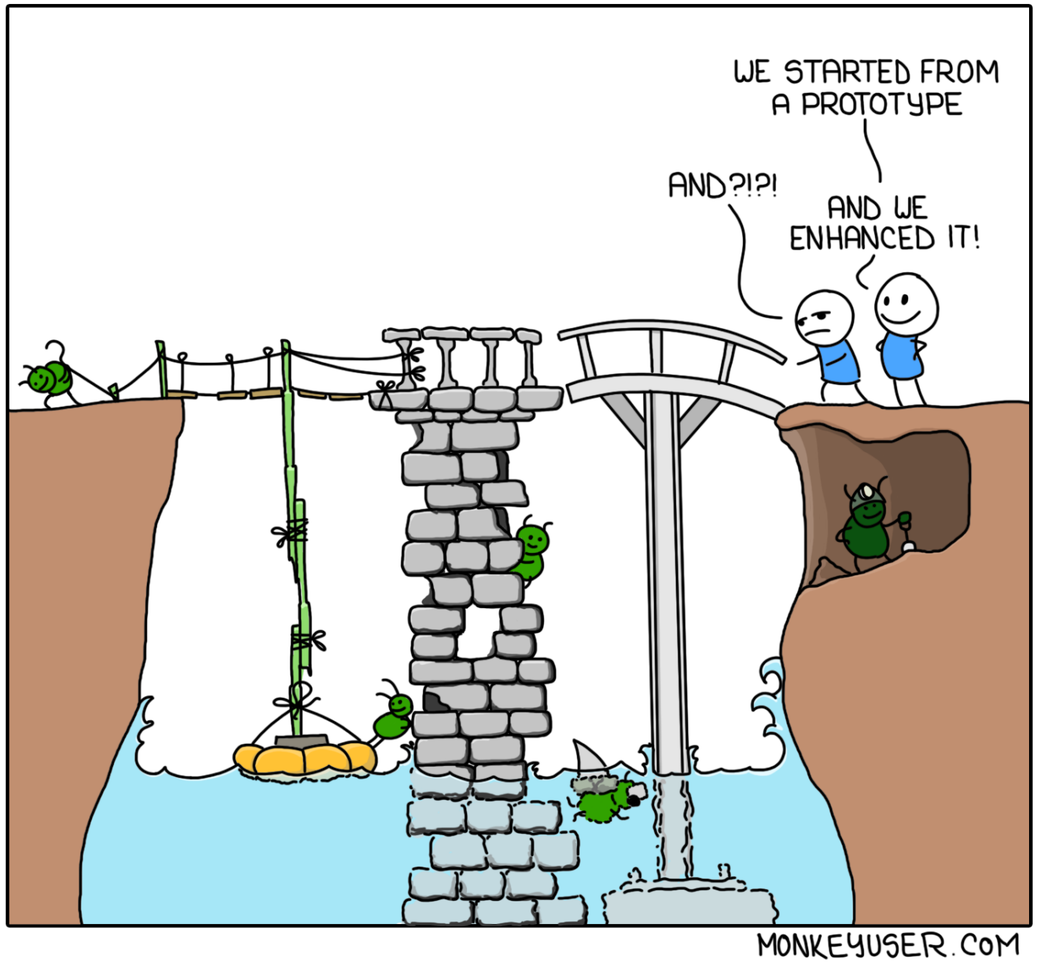
mDevCamp Prague - Apr 24' 🇨🇿

Ash Davies - SumUp Services GmbH
Android / Kotlin GDE - Berlin
ashdavies.dev

sumup®

Photo by Nubelson Fernandes on Unsplash

PRODUCTION READY

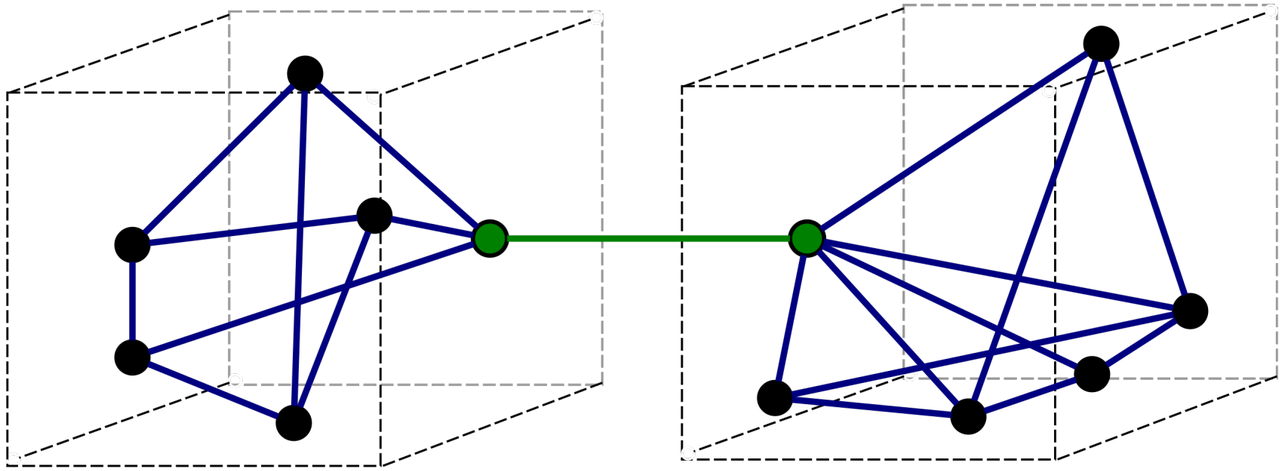


Legacy (adj.)

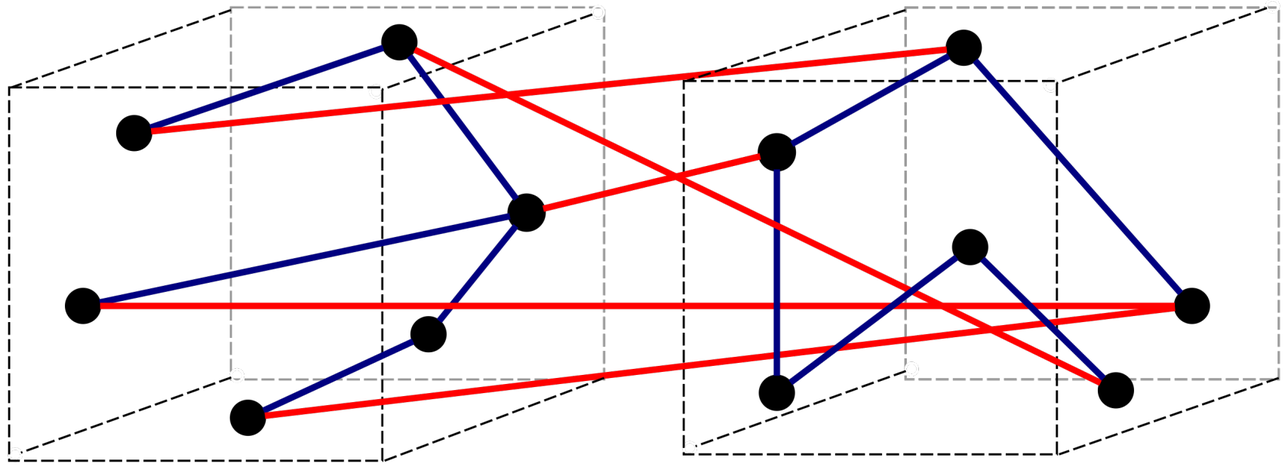
Denoting or relating to software or hardware that has been superseded but is difficult to replace because of its wide use.

Uncertainty

Coupling 🔌

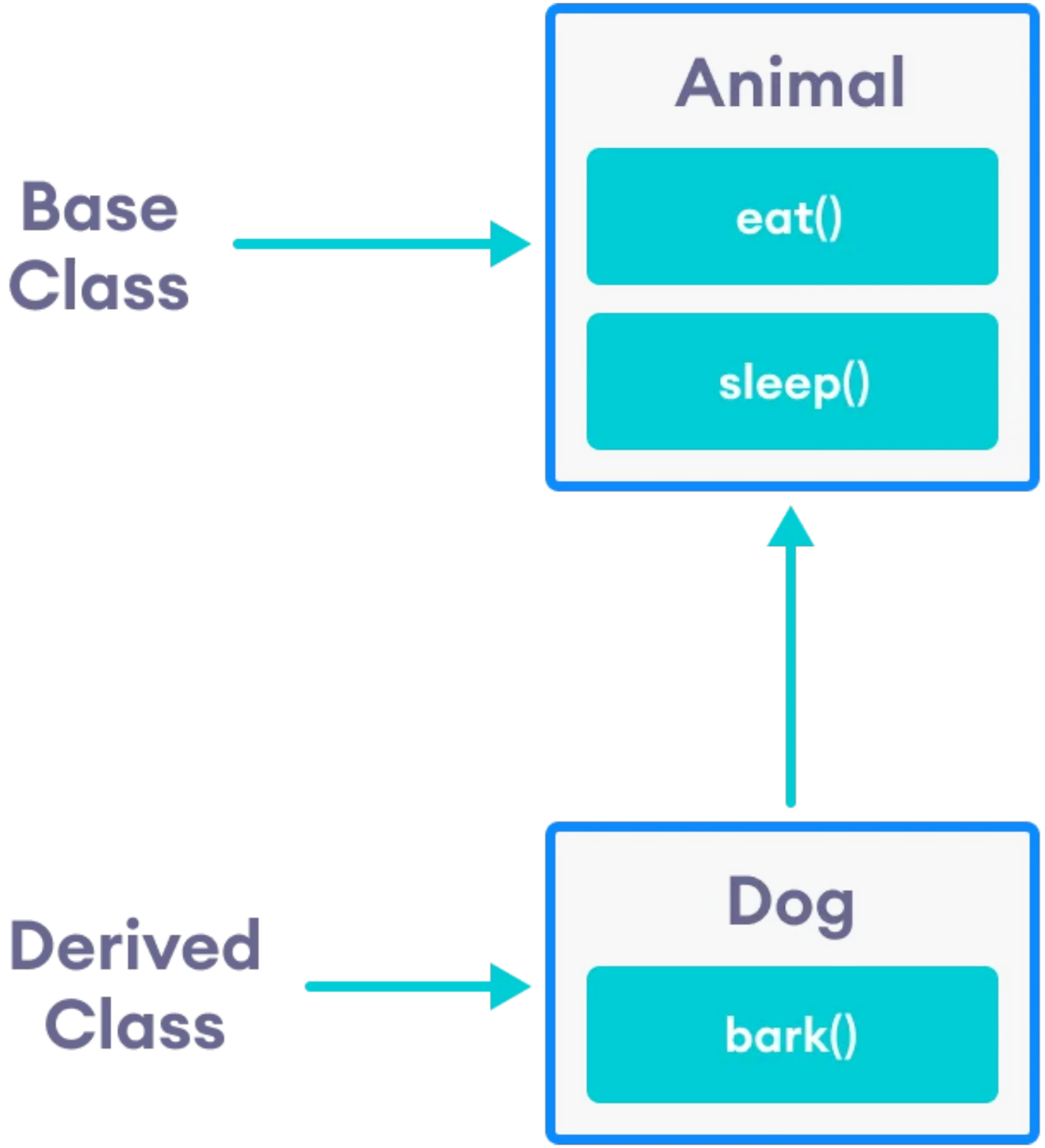


a) Good (loose coupling, high cohesion)



b) Bad (high coupling, low cohesion)

Inheritance



Mutability

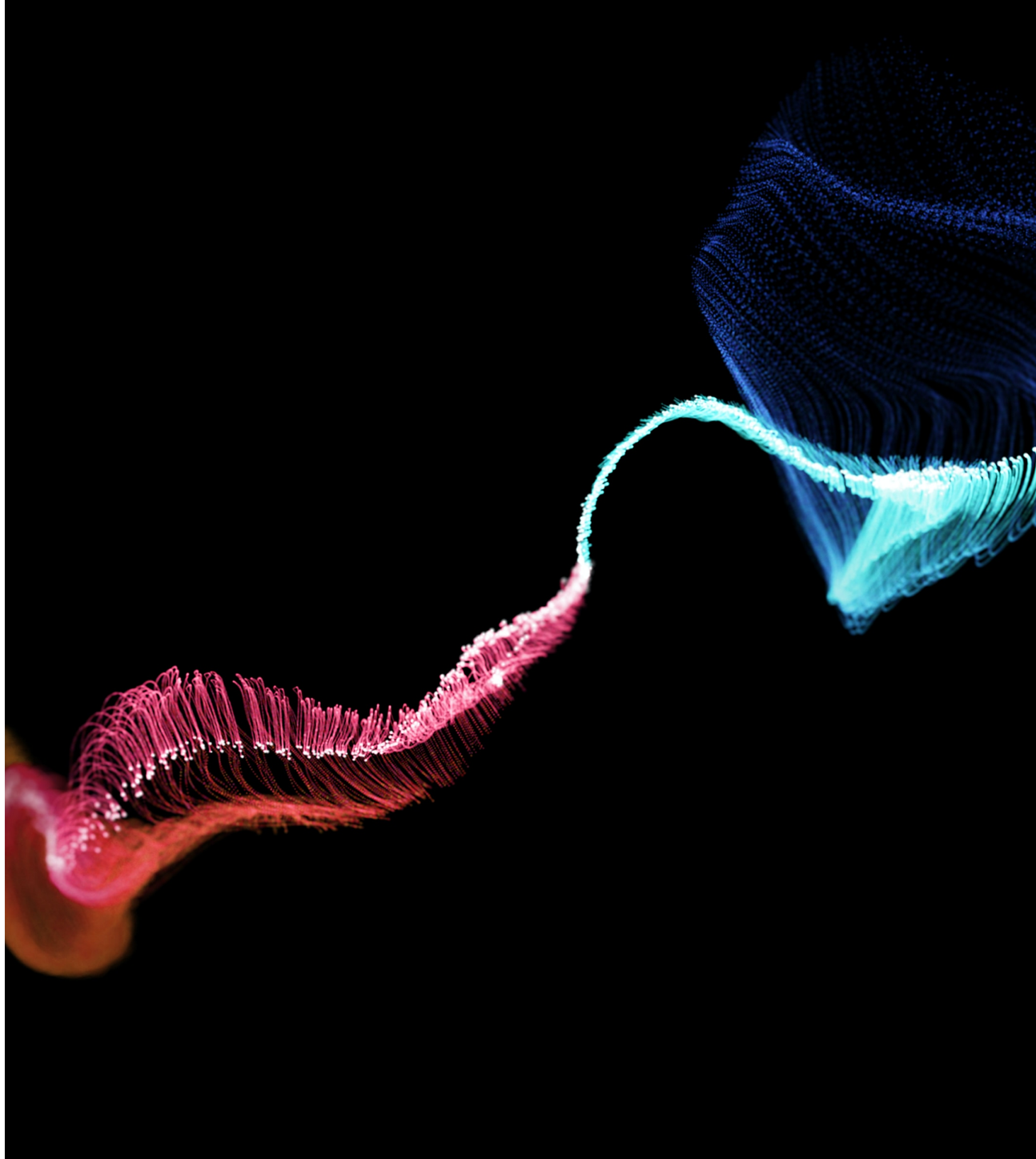


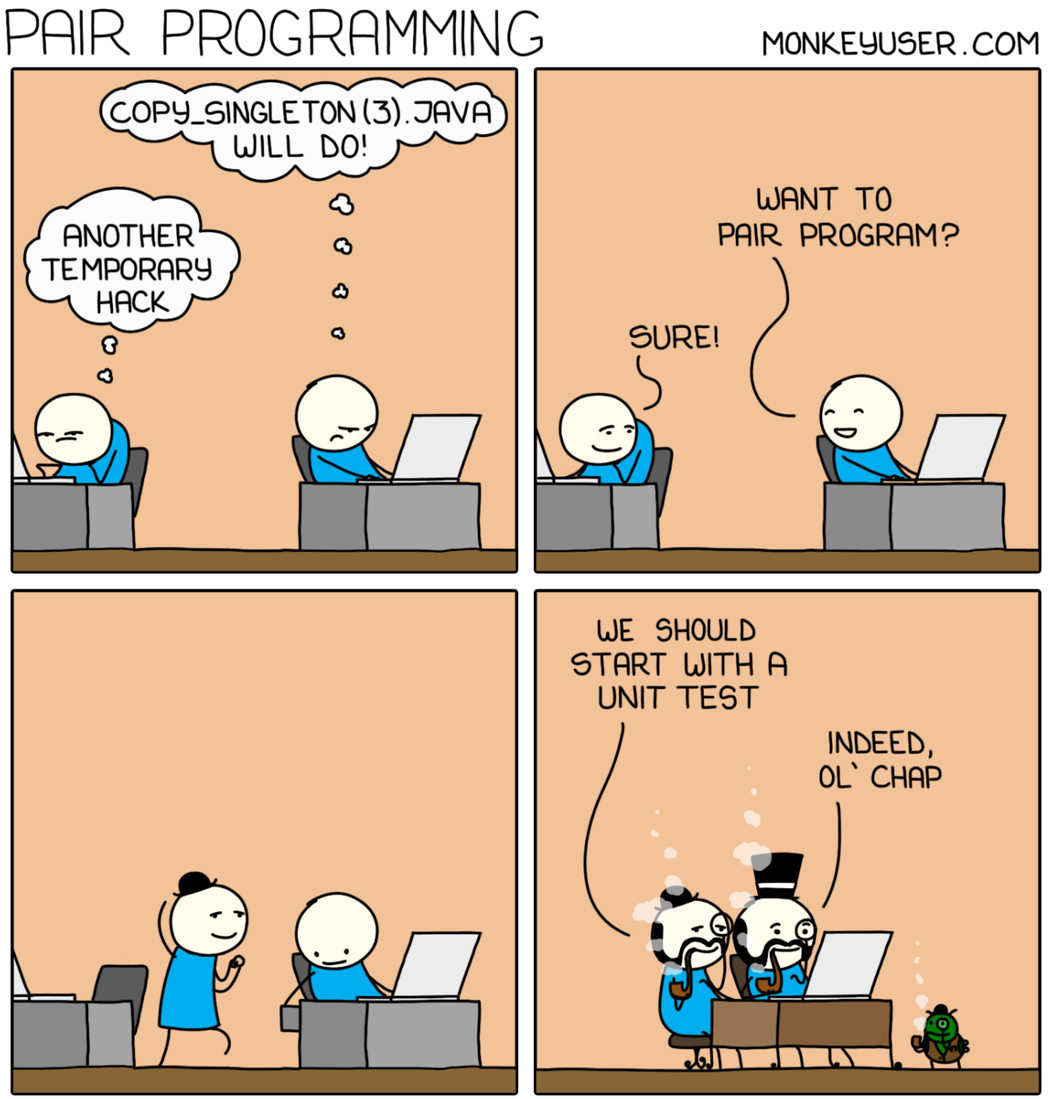
Photo by Richard Horvath on Unsplash

Unpredictability

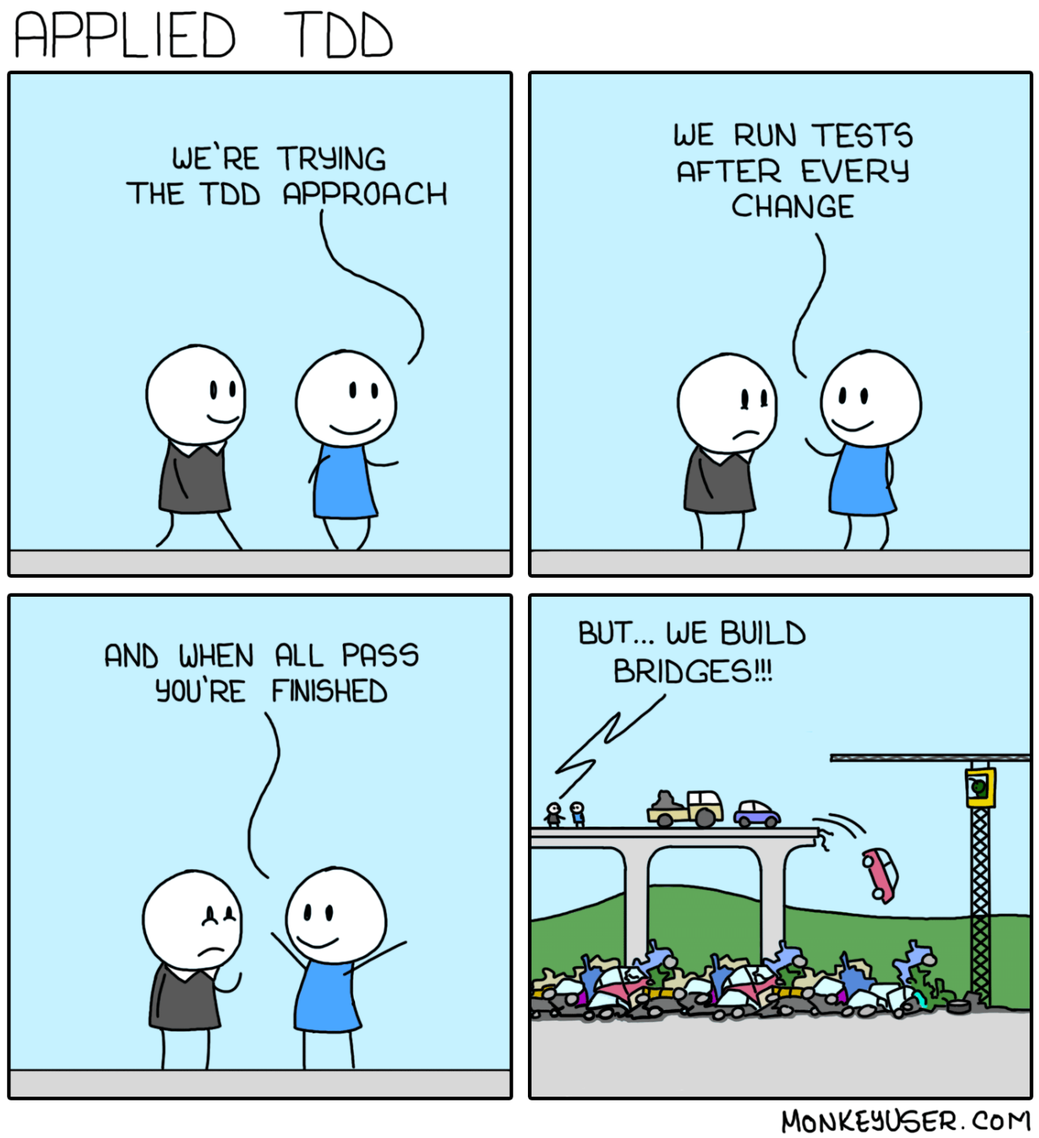
?

Extreme Programming 🤘

Pair Programming



Test Driven Development



Offensive Programming < 🖕 >


Kotlin

Kotlin: Mutability

Kotlin: Mutability

```
fun sumAbsolute(list: MutableList<Int>): Int {
  for (i in list.indices) list[i] = abs(list[i])
  return list.sum()
}
```

Kotlin: Mutability

```
private val GROUNDHOG_DAY = TODO("java.util.Date()")
fun startOfSpring(): java.util.Date = GROUNDHOG_DAY

val partyDate = startOfSpring()
partyDate.month = partyDate.month + 1

// Date is mutable 🤦
```

Kotlin: Mutability

Shared Mutable State ⚠️

Kotlin: Immutability

Unidirectional Data Flow 🏆

Kotlin: Immutability

Collections

```
fun List<T>.toMutableList(): MutableList<T>

fun Map<K, V>.toMutableMap(): MutableMap<K, V>

fun Set<T>.toMutableSet(): MutableSet<T>
```

Kotlin: Immutability

`org.jetbrains.kotlinx:kotlinx-collections-immutable`

```
fun Iterable<T>.toPersistentList(): PersistentList<T>

fun Iterable<T>.toPersistentSet(): PersistentSet<T>
```

Kotlin: Immutability

IntelliJ IDEA

```
val immutableValue = true
var mutableValue = false
```

Kotlin: Immutability

Final Concretions 

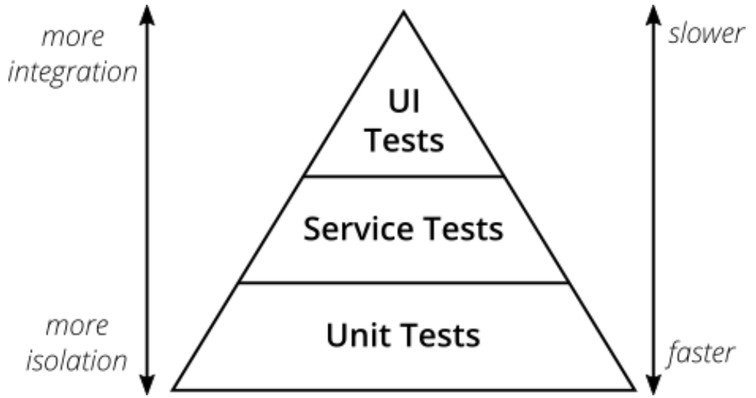
By default, Kotlin classes are final – they can't be inherited

Testing

Photo by DiEGO MüLLER on Unsplash

Photos by Daniel Romero, Taylor Vick on Unsplash

Testing


more
integration
more
isolation
UI
Tests
Service Tests
Unit Tests
slower
faster

Anti-Patterns and Code Smell

Refactoring

Seams

Photo by Jackson David on Unsplash

Refactoring: Seams

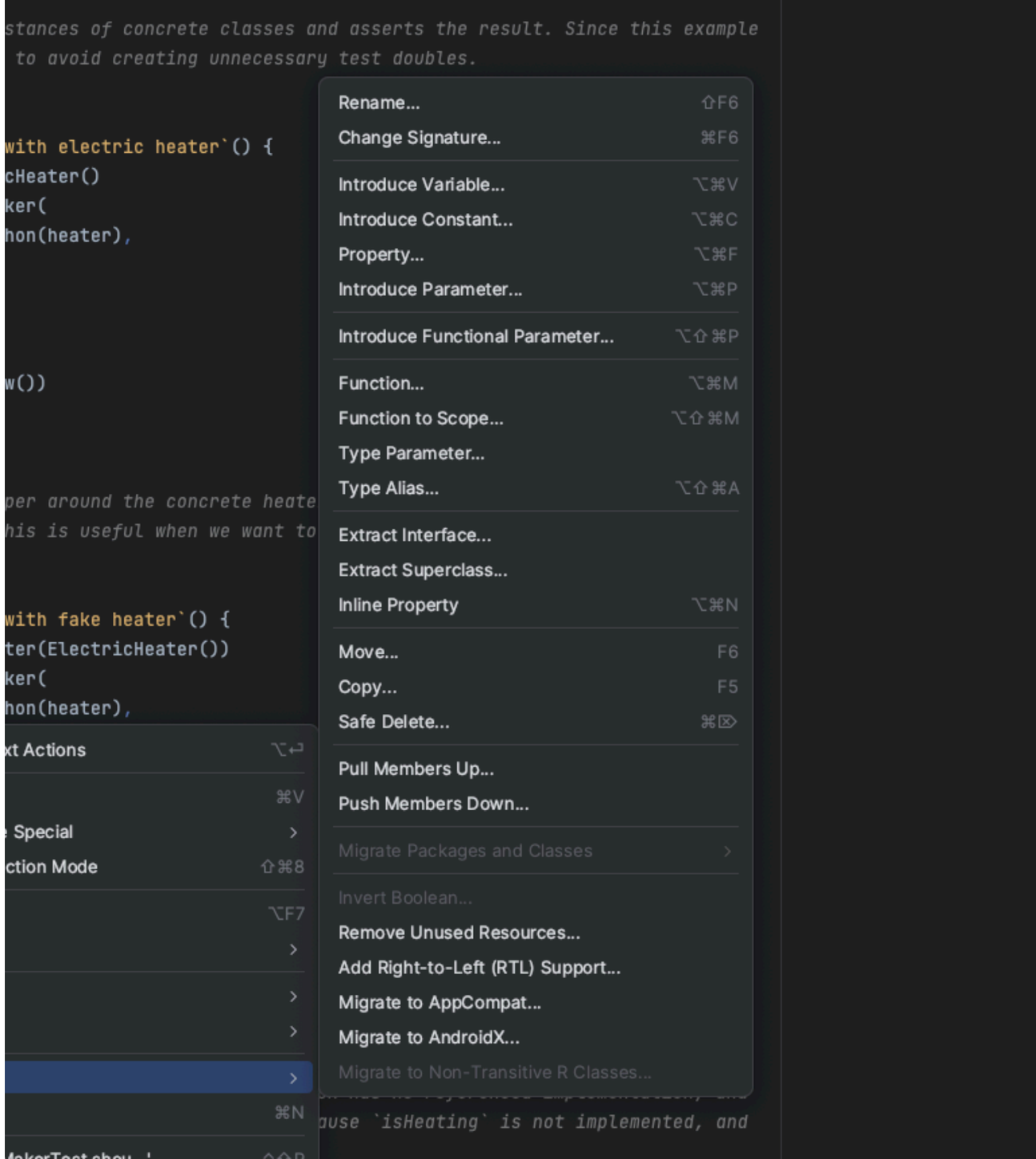
```
internal class FitFilter {
  private val parser: Parser = Parser.newInstance()
}
```

Refactoring: Seams

```
===================================================================
diff --git a/FitFilter.kt b/FitFilter.kt

- internal class FitFilter {
-   private val parser: Parser = Parser.newInstance()
- }
-
+ internal fun interface FitFilter {
+   fun filter(input: String): String
+ }
+
+ internal fun FitFilter(parser: Parser) = FitFilter { input ->
+   parser.parse(input)
+ }
```

Refactoring: Seams



Refactoring

Dependency Injection

Refactoring

Dependency Injection

```
===================================================================
diff --git a/CoffeeMaker.kt b/CoffeeMaker.kt

- internal class CoffeeMaker {
-   private val heater: Heater = ElectricHeater()
-   private val pump: Pump = Thermosiphon(heater)
- }

+ internal class CoffeeMaker(
+   private val heater: Heater,
+   private val pump: Pump,
+ )
```

Refactoring

Dependency Injection

```
==================================================================
diff --git a/CoffeeMaker.kt b/CoffeeMaker.kt

+ internal class CoffeeMaker(
+   private val heater: Heater,
-   private val thermosiphon: Thermosiphon,
+   private val pump: Pump,
+ )
+
+ internal interface Pump {
+   fun pump()
+ }
+
- internal class Thermosiphon {
+ internal class Thermosiphon : Pump {
```

Testing: Dependencies

Dependency Injection

```
internal class CoffeeMaker(
  private val heater: Heater,
  private val pump: Pump,
)
```

Testing: Dependencies

Dependency Injection

```
val heater = NuclearFusionHeater() // Expensive!

val maker = CoffeeMaker(
  pump = Thermosiphon(heater),
  heater = heater,
)

assertTrue(maker.brew())
```

Testing: Dependencies

Dependency Injection

```
val heater = DiskCachedHeater() // Stateful!

val maker = CoffeeMaker(
  pump = Thermosiphon(heater),
  heater = heater,
)

assertTrue(maker.brew())
```

Testing: Dependencies

Dependency Injection

```
val heater = UnbalancedHeater() // Error prone!

val maker = CoffeeMaker(
  pump = Thermosiphon(heater),
  heater = heater,
)

assertTrue(maker.brew())
```

Testing: Dependencies

Test Doubles

Test Doubles

Mocks

Test Doubles: Mocks

```
val heater = mock<Heater>()
val pump = mock<Pump>()

val maker = CoffeeMaker(
  heater = heater,
  pump = pump,
)

assertTrue(maker.brew())
```

Test Doubles: Mocks

```
val heater = mock<Heater>()
val pump = mock<Pump>()

val maker = CoffeeMaker(
  heater = heater,
  pump = pump,
)

assertTrue(maker.brew()) // ⚠ Fails!
```

Test Doubles: Mocks

```
val heater = mock<Heater> {
  on { isHeating } doAnswer { true }
}

val pump = mock<Pump> {
  on { pump() } doAnswer { true }
}

val maker = CoffeeMaker(
  heater = heater,
  pump = pump,
)

assertTrue(maker.brew())
```

Test Doubles: Mocks

Footguns 🦶🔫

Test Doubles: Mocks

Accidental Invocation

```
spy(emptyList<String>()) {
  on { get(0) } doAnswer { "foo" } // throws IndexOutOfBoundsException
}
```

Test Doubles: Mocks

API Sensitivity

```
internal interface Heater {
  val isHeating: Boolean
}

val heater = mock<Heater> {
  on { isHeating } doAnswer { true }
}
```

Test Doubles: Mocks

API Sensitivity

```
  internal interface Heater {
+   fun <T : Any> heat(body: () -> T): T
    val isHeating: Boolean
  }

  val heater = mock<Heater> {
    on { isHeating } doAnswer { true }
  }
```

Test Doubles: Mocks

Data Classes

Test Doubles: Mocks

~~Data Classes~~

Just Don't.

Test Doubles: Mocks

Default Answers

```
==========================================================================
diff --git a/PumpTest.kt b/PumpTest.kt

- val heater: Heater = mock()
+ val heater: Heater = mock(defaultAnswer = RETURNS_SMART_NULLS)
+
+ val isHeating: Boolean = heater.isHeating // false
```

Test Doubles: Mocks

Performance

Expensive real implementations replaced by expensive mocks.

- **Runtime code generation**
- **Bytecode manipulation**
- **Reflection 😱**

Test Doubles: Mocks

Dynamic Mutability

```
internal class CoffeeMakerTest {

  private lateinit var heater: Heater

  @Before
  fun setUp() {
    heater = mock {
      on { isHeating } doAnswer { true }
    }
  }

  @Test
  fun `should brew coffee`() {
    // heater already has state!
  }
}
```

Test Doubles: Mocks

Dynamic Mutability

Framework generated mocks introduce a shared, mutable, dynamic, runtime declaration.

Unpredictability

Costs

Unpredictability: Costs

Learning Curve

Unpredictability: Costs
Peer Review

Unpredictability: Costs

Risk of Bugs

Unpredictability: Costs

Slowed Feature Delivery 💰💰💰

Unpredictability

Victims

- **Junior developers**
- **New team members**
- **Future you**

Test Doubles: Mocks

Don't Mock Classes You Don't Own

testing.googleblog.com/2020/07/testing-on-toilet-dont-mock-types-you.html

You Don't Own Your Code!

Your code belongs to your team.

Be considerate.

Test Doubles: Mocks

Interaction Verification 👎

Test Doubles

Mocks

Test Doubles: Stubs

Test Doubles: Stubs

Simple

```
internal interface Pump {
  fun pump(): Boolean
}

internal object StubPump : Pump {
  override fun pump(): Boolean = true
}
```

Test Doubles: Stubs

Idiomatic

```
internal fun interface Pump {
  fun pump(): Boolean
}

val stub = Pump { true }
```

Testing: Stubs

API Sensitive

```
+ private const val DEFAULT_AMOUNT = 250 // ml
+
- internal fun interface Pump {
+ internal interface Pump {
-    fun pump(): Boolean
+    fun pump(amount: Int = DEFAULT_AMOUNT): Boolean
+ }
```

Testing: Stubs

API Sensitive

```
private const val DEFAULT_AMOUNT = 250 // ml

internal interface Pump {
  fun pump(amount: Int = DEFAULT_AMOUNT): Boolean
}

val stub = Pump { true } // Compilation failure...
```

Testing: Fakes 🏆

Testing: Fakes

```
public class FakePump(private val onPump: (Boolean) -> Boolean) : Pump {

  public val pumped = mutableListOf<Pair<Boolean, Boolean>>()

  override fun pump(full: Boolean): Boolean = onPump(full).also {
    pumped += full to it
  }
}
```

Testing: Fakes

Additional Behaviour

```
class DelegatingHeater(private val delegate: Heater) : Heater by delegate {

  private val _drinks = mutableListOf<Any>()
  val drinks: List<Any> by ::_drinks

  override fun <T : Any> heat(body: () -> T): T {
    return delegate.heat(body).also { _drinks += it }
  }
}
```

Testing: Fakes

Authoring

Testing: Fakes

Qualifications

Those who wrote the code are the most uniquely qualified to write the tests.

Testing: In Memory

```
internal fun interface CoffeeStore {
  fun has(type: CoffeeType): Boolean
}

internal enum class CoffeeType {
  CAPPUCCINO,
  ESPRESSO,
  LATTE,
}
```

Testing: In Memory

```
internal class InMemoryCoffeeStore : CoffeeStore {

  private val _stock = mutableMapOf<CoffeeType, Int>()
  val stock: Map<CoffeeType, Int> by ::_stock

  override fun has(type: CoffeeType): Boolean {
    return (_stock[type] ?: 0) > 0
  }

  fun add(type: CoffeeType, amount: Int = 1) {
    _stock[type] = (_stock[type] ?: 0) + amount
  }
}
```

Be Better

Interface Segregation

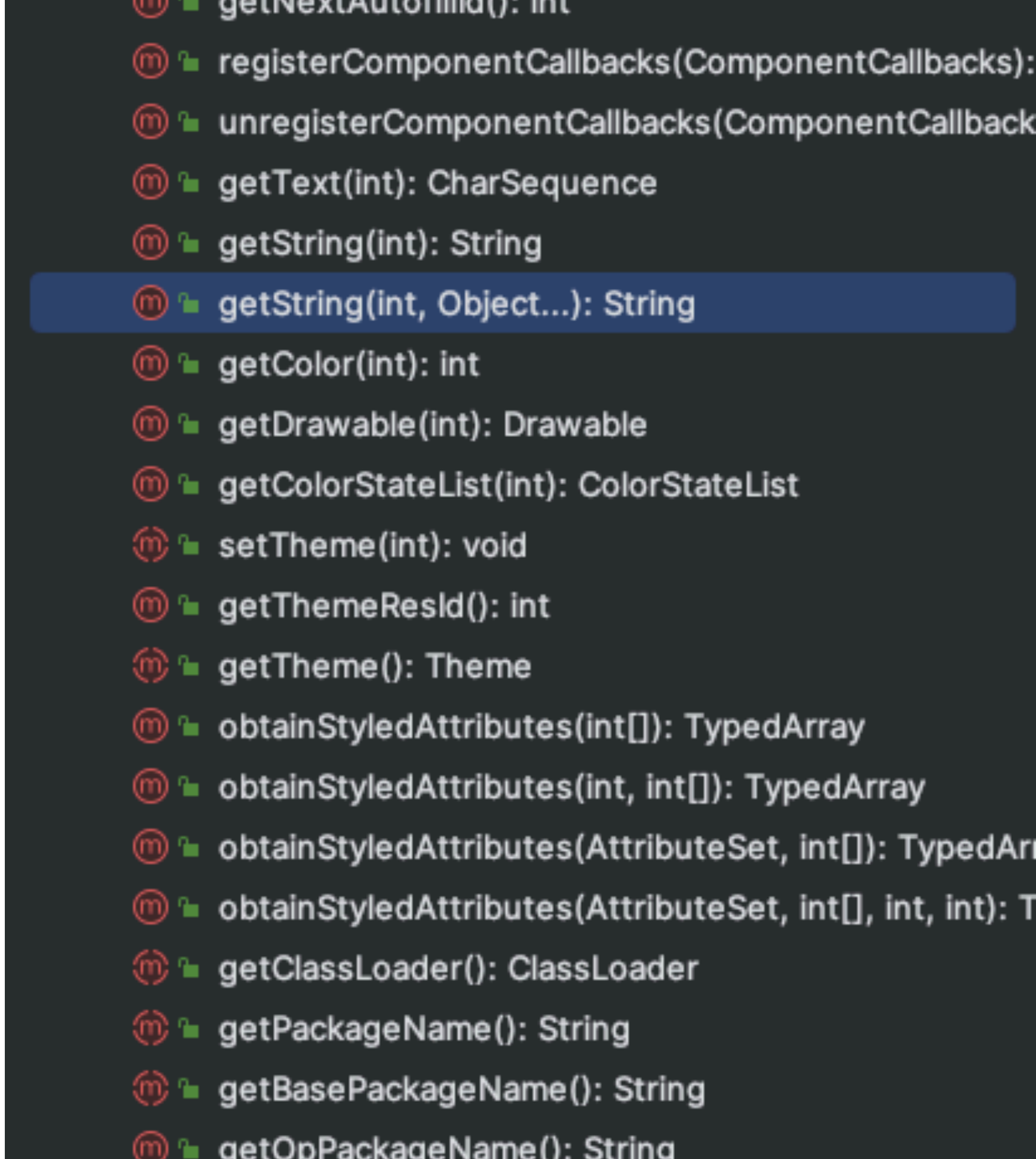
No code should be forced to depend on methods it does not use.

Image: dribbble.com/shots/3251806-Interface-Segregation-Principle

Testing: Android

Legacy Inheritance

android.content.Context



Recap

Extreme Programming

Recap

Shared Mutable State

Recap

Test Doubles

Thanks!

Further Reading

- Martin Flower: Mocks Aren't Stubs
  martinfowler.com/articles/mocksArentStubs.html
- Martin Fowler: Practical Test Pyramid
  martinfowler.com/articles/practical-test-pyramid.html
- Images: Monkey User
  monkeyuser.com
- Michael Feathers: Working Effectively with Legacy Code
  ISBN: 978-0-13117-705-5
- Sam Edwards: Wrapping Mockito Mocks for Reusability
  handstandsam.com/2020/06/08/wrapping-mockito-mocks-for-reusability
- Steve Freeman, Nat Pryce: Growing Object-Oriented Software, Guided by Tests
  ISBN: 978-0-32150-362-6
- Testing on the Toilet: Don't mock Types You Don't Own
  testing.googleblog.com/2020/07/testing-on-toilet-dont-mock-types-you.html
- Testing on the Toilet: Know Your Test Doubles
  testing.googleblog.com/2013/07/testing-on-toilet-know-your-test-doubles.html
- Marcello Galhardo: No Mocks Allowed
  marcellogalhardo.dev/posts/no-mocks-allowed