Como implementar verificações de análise estática usando Lint para deixar seu código mais seguro

Guilherme Krzisch

“Verificações de análise estática usando Lint”

- O que é?
- Porque eu iria querer implementar uma nova verificação?

Exemplo

- Detectar que esquecemos de mostrar o Toast

```
fun showToast(context: Context, message: String) {
    Toast.makeText(context, message, Toast.LENGTH_LONG)
}
```

- Detectar que um método está declarando mais do que 5 parâmetros

```
fun handleAuthorizationFlow(cardNumber: String,
                            cardName: String,
                            expirationDate: String,
                            birthdayDate: String,
                            cvv: String,
                            hasRedeem: Boolean)
```

Análise Estática

- O que é?
  - **Tempo de compilação** vs Tempo de execução
  - Prevenir erros ou impor um estilo de código
- Diferentes ferramentas
  - Java vs Kotlin? Permitem adicionar novas regras?
  - Android lint, checkstyle, FindBugs, ktlint, detekt (mais exemplos [1] [2])

(Android) Lint

- Como adicionar uma nova regra?
  - Detector: responsável por analisar o código
  - Issue: encapsula um erro Lint
  - Implementation: configuração do Detector
  - Registry: lista de todos os erros Lint existentes
- Conjunto de regras existentes: BuiltInIssueRegistry

Exemplo

- Detectar que esquecemos de mostrar o Toast

```
fun showToast(context: Context, message: String) {
    Toast.makeText(context, message, Toast.LENGTH_LONG)
}
```

Toast created but not shown: did you forget to call show() ?

Call show() ⌥⇧↩ More actions... ⌥↩

- Detectar que um método está declarando mais do que 5 parâmetros

Detectar que um método está declarando mais do que 5 parâmetros

```
val ISSUE: Issue = Issue.create(
    id = "TooManyParametersDetector",
    briefDescription = "Method should not declare more than 5 parameters",
    explanation = """
            You should limit the number of parameters you method receive,
            in order to make your method more legible and easier to use correctly.
        """.trimIndent(),
    category = Category.CORRECTNESS,
    priority = 5,
    severity = Severity.WARNING,
    androidSpecific = true,
    implementation = IMPLEMENTATION
)
```

Detectar que um método está declarando mais do que 5 parâmetros

```
class TooManyParametersDetector : Detector(), SourceCodeScanner
```

```
private val IMPLEMENTATION = Implementation(
    TooManyParametersDetector::class.java,
    Scope.JAVA_FILE_SCOPE
)
```

Detectar que um método está declarando mais do que 5 parâmetros

```
class IssueRegistry : IssueRegistry() {

    override val api: Int = CURRENT_API

    override val issues: List<Issue>
        get() = listOf(TooManyParametersDetector.ISSUE)

}
```

Detectar que um método está declarando mais do que 5 parâmetros

```
class TooManyParametersDetectorTest : AndroidSdkLintDetectorTest() {

    override fun getDetector(): Detector = TooManyParametersDetector()

    override fun getIssues(): MutableList<Issue> = mutableListOf(TooManyParametersDetector.ISSUE)
```

Detectar que um método está declarando mais do que 5 parâmetros

```
@Test
fun `java method with 1 parameter`() {
    val javaFile = java(
        source: """
        package com.brokoli.lint;

        class MyClass {

            public void methodWith1Parameter(boolean first) {

            }

        }
    """
    ).indented()

    val lintResult = lint()
        .files(javaFile)
        .run()

    lintResult.expectClean()
}
```

```kotlin
@Test
fun `kotlin method with 6 parameters`() {
    val kotlinFile = kotlin(
        source: """
        package com.brokoli.lint;

        class MyClass {

            fun methodWith6Parameters(first: Boolean, second: String, third: Int, fourth: Long, fifth: Char, sixth: Boolean) {

            }

        }
    """
    ).indented()

    val lintResult = lint()
        .files(kotlinFile)
        .run()

    lintResult
        .expectWarningCount( expectedCount: 1)
        .expect(
            """
            src/com/brokoli/lint/MyClass.kt:5: Warning: Method should not declare more than 5 parameters [TooManyParametersDetector]
                fun methodWith6Parameters(first: Boolean, second: String, third: Int, fourth: Long, fifth: Char, sixth: Boolean) {
                ^
            0 errors, 1 warnings
            """.trimIndent()
        )
}
```

```
UFile (package = com.brokoli.lint)
    UClass (name = MyClass)
        UAnnotationMethod (name = methodWith6Parameters)
            UParameter (name = first)
                UAnnotation (fqName = org.jetbrains.annotations.NotNull)
            UParameter (name = second)
                UAnnotation (fqName = org.jetbrains.annotations.NotNull)
            UParameter (name = third)
                UAnnotation (fqName = org.jetbrains.annotations.NotNull)
            UParameter (name = fourth)
                UAnnotation (fqName = org.jetbrains.annotations.NotNull)
            UParameter (name = fifth)
                UAnnotation (fqName = org.jetbrains.annotations.NotNull)
            UParameter (name = sixth)
                UAnnotation (fqName = org.jetbrains.annotations.NotNull)
            UBlockExpression
        UAnnotationMethod (name = MyClass)
```

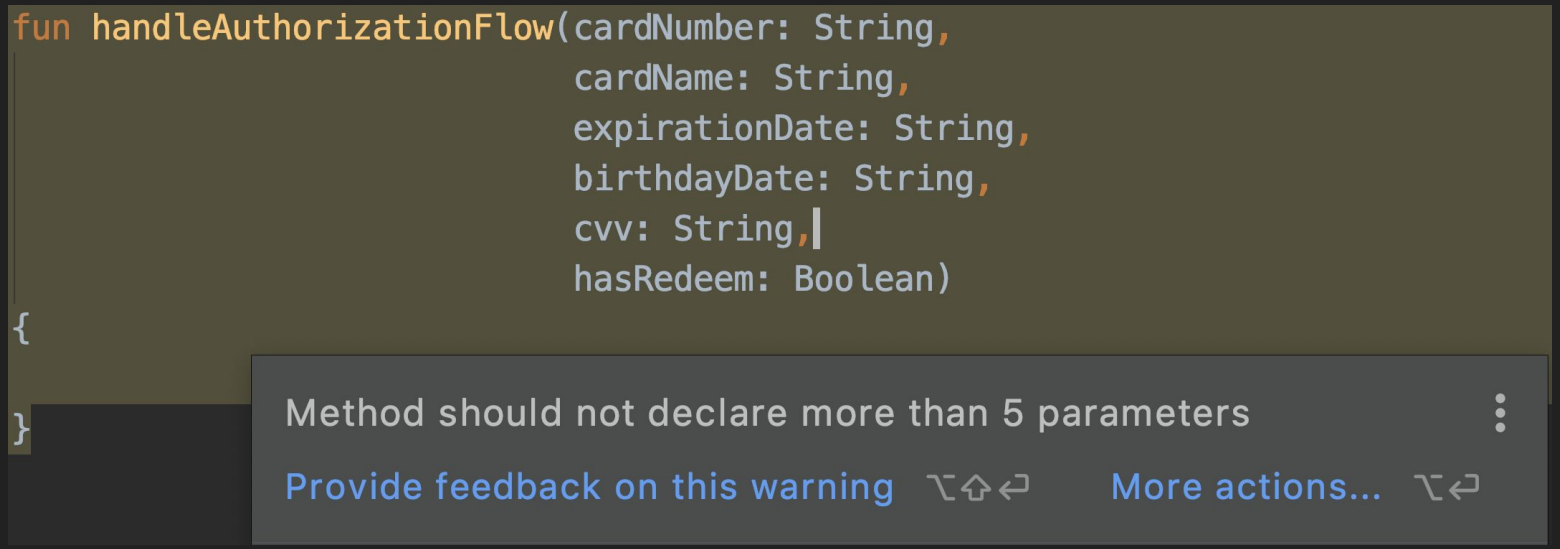
Detectar que um método está declarando mais do que 5 parâmetros

```
class TooManyParametersDetector : Detector(), SourceCodeScanner {

    override fun getApplicableUastTypes(): List<Class<out UElement>>? {
        return listOf(UMethod::class.java)
    }

    override fun createUastHandler(context: JavaContext): UElementHandler? {
        return MethodHandler(context)
    }

    inner class MethodHandler(private val context: JavaContext) : UElementHandler() {

        override fun visitMethod(node: UMethod) {
            if(node.parameters.size > MAX_NUMBER_OF_METHOD_PARAMETERS) {
                reportUsage(context, context.getLocation(node))
            }
        }

    }
```

Detectar que um método está declarando mais do que 5 parâmetros

```
private fun reportUsage(context: JavaContext, location: Location) {
    context.report(
        issue = ISSUE,
        location = location,
        message = "Method should not declare more than 5 parameters"
    )
}
```

Detectar que um método está declarando mais do que 5 parâmetros



Exceções Java vs Kotlin

- Java - Exceções checadas vs não-checadas
- Kotlin - Exceções não-checadas (i.e. não use exceções para controlar o fluxo de execução do programa)
- Java + Kotlin?
  - Paradigmas diferentes
  - Regra para detectar que código em Kotlin não está tratando exceções checadas em Java

Exceções

```
val javaFile = java(
    source: """
    package com.brokoli.lint;

    import java.io.IOException;

    class MyJavaClass {

        public void javaMethod() throws IOException {
            throw new IOException();
        }

    }
"""
).indented()
```

Exceções

```
UFile (package = com.brokoli.lint)
    UImportStatement (isOnDemand = false)
        UClass (name = MyJavaClass)
            UMethod (name = javaMethod)
                UBlockExpression
                    UThrowExpression
                        UCallExpression (kind = UastCallKind(name='constructor_call'), argCount = 0))
                            USimpleNameReferenceExpression (identifier = IOException)
```

Exceções

```
val kotlinFile = kotlin(
    source: """
    package com.brokoli.lint

    class MyKotlinClass {

        fun kotlinMethod() {
            MyJavaClass().javaMethod()
        }

    }
"""
).indented()
```

```
val kotlinFile = kotlin(
    source: """
    package com.brokoli.lint

    import java.io.IOException

    class MyKotlinClass {

        fun kotlinMethod() {
            try {
                MyJavaClass().javaMethod()
            } catch (exception: IOException) {

            }
        }

    }
"""
).indented()
```

```
UFile (package = com.brokoli.lint)
    UClass (name = MyKotlinClass)
        UAnnotationMethod (name = kotlinMethod)
            UBlockExpression
                UQualifiedReferenceExpression
                    UCallExpression (kind = UastCallKind(name='constructor_call'), argCount = 0))
                        UIdentifier (Identifier (MyJavaClass))
                        USimpleNameReferenceExpression (identifier = <init>, resolvesTo = MyJavaClass)
                    UCallExpression (kind = UastCallKind(name='method_call'), argCount = 0))
                        UIdentifier (Identifier (javaMethod))
                        USimpleNameReferenceExpression (identifier = javaMethod, resolvesTo = null)
        UAnnotationMethod (name = MyKotlinClass)
```

```
UAnnotationMethod (name = kotlinMethod)
    UBlockExpression
        UTryExpression
            UBlockExpression
                UQualifiedReferenceExpression
                    UCallExpression (kind = UastCallKind(name='constructor_call'), argCount = 0))
                        UIdentifier (Identifier (MyJavaClass))
                        USimpleNameReferenceExpression (identifier = <init>, resolvesTo = MyJavaClass)
                    UCallExpression (kind = UastCallKind(name='method_call'), argCount = 0))
                        UIdentifier (Identifier (javaMethod))
                        USimpleNameReferenceExpression (identifier = javaMethod, resolvesTo = null)
            UCatchClause (exception)
                UBlockExpression
```

Exceções

```
private val IMPLEMENTATION = Implementation(
    KotlinShouldHandleJavaExceptionsDetector::class.java,
    Scope.JAVA_FILE_SCOPE
)

val ISSUE: Issue = Issue
    .create(
        id = "KotlinShouldHandleJavaExceptionsDetector",
        briefDescription = "Kotlin code is calling Java method which throws checked exceptions",
        explanation = """
            Kotlin does not support checked Exceptions,
            so we should not rely on Exceptions to propagate errors in our application.
            More details in: https://kotlinlang.org/docs/reference/exceptions.html
        """.trimIndent(),
        category = Category.CORRECTNESS,
        priority = 9,
        severity = Severity.ERROR,
        androidSpecific = true,
        implementation = IMPLEMENTATION
    )
```

Exceções

```
class KotlinShouldHandleJavaExceptionsDetector : Detector(), SourceCodeScanner {

    override fun getApplicableUastTypes(): List<Class<out UElement>>? {
        return listOf(UCallExpression::class.java)
    }

    override fun createUastHandler(context: JavaContext): UElementHandler? {
        return KotlinThrowsHandler(context)
    }

    inner class KotlinThrowsHandler(private val context: JavaContext) : UElementHandler() {

        override fun visitCallExpression(node: UCallExpression) {
            node.accept(AnotherVisitor(context))
        }

    }
```

Exceções

```
private inner class AnotherVisitor(private val context: JavaContext) : AbstractUastVisitor() {

    override fun visitCallExpression(node: UCallExpression): Boolean {
        val throwTypes = node.resolve()?.throwsTypes.orEmpty()
        if(throwTypes.isNotEmpty()) {
            if(!hasTryCatch(node.uastParent, throwTypes.toList())) {
                reportUsage(context, node)
            }
        }
        return super.visitCallExpression(node)
    }
```

Exceções

```
private fun hasTryCatch(node: UElement?, throwTypes: List<JvmReferenceType>): Boolean {
    if(node == null) {
        return false
    }
    if(node is KotlinUCatchClause) {
        return false
    }
    if(node is KotlinUTryExpression) {
        if(catchExceptions(throwTypes, node.catchClauses)) {
            return true
        }
    }
    return hasTryCatch(node.uastParent, throwTypes)
}
```

Exceções

```
private fun catchExceptions(throwTypes: List<JvmReferenceType>,
                            catchClauses: List<KotlinUCatchClause>): Boolean {
    val catchTypes = catchClauses.flatMap { catchClause -> catchClause.types }
    return throwTypes.all { throwType ->
        catchTypes.any { catchType ->
            catchException((throwType as PsiClassReferenceType).deepComponentType, catchType)
        }
    }
}
```

Exceções

```
private fun catchException(throwType: PsiType, compareType: PsiType): Boolean {
    if(throwType.canonicalText == compareType.canonicalText) {
        return true
    }
    for(throwSuperType in throwType.superTypes) {
        if(catchException(throwSuperType, compareType)) {
            return true
        }
    }
    return false
}
```

Exceções

```
fun kotlinMethod() {
    JavaClass().methodThrowsCheckedException()
}

fun
```

Kotlin code is calling Java method which throws checked exceptions. Be aware to handle it accordingly!

Provide feedback on this warning ⌥⇧⏎ More actions... ⌥⏎

Obrigado!

- Repositório https://github.com/guilhermekrz/CustomAndroidLint
- Perguntas?