How to build a custom ruleset

Paul Dingemans

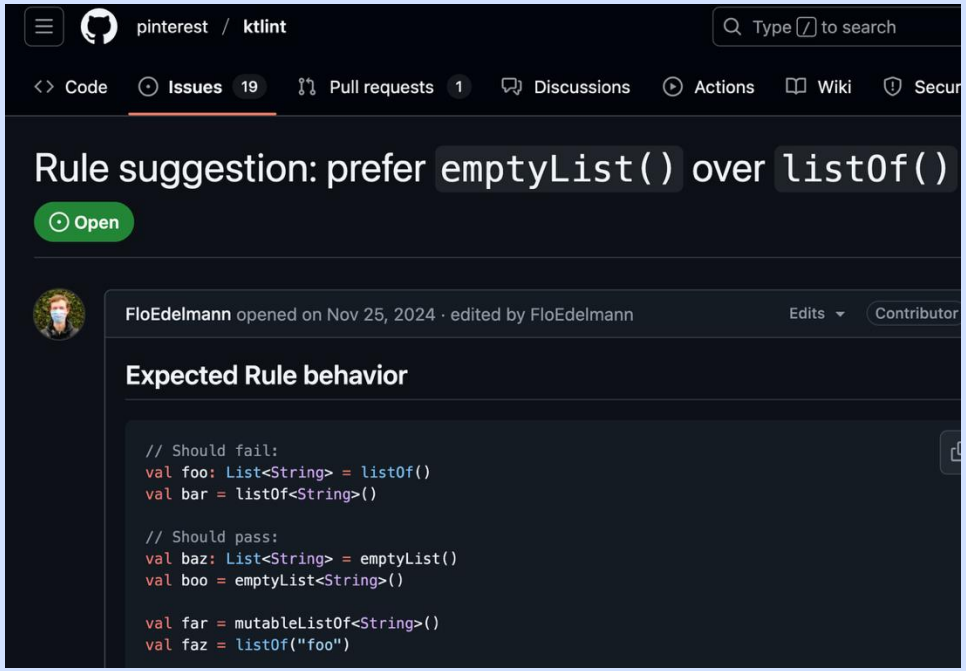
KtLint Maintainer
since 2022

Software engineer
@Bol since 2018

bol.

DEMO RULE

- KtLint has no outstanding request for (pure) formatting
- But it can be used for transforming Kotlin code in other ways as well



⚠ Other tools like Modern's Open Rewrite, or Error Prone Refaster might be more suitable for this

DEMO RULE

Goal: Change collection constructor calls without arguments

```
val emptyList1 = listOf<String>()
val emptyList2: List<String> = listOf()

val emptySet1 = setOf<String>()
val emptySet2: Set<String> = setOf()

val emptyMap1 = mapOf<String, String>()
val emptyMap2: Map<String, String> = mapOf()
```

DEMO RULE

Goal: Change collection constructor calls without arguments to factory methods

```
val emptyList1 = emptyList<String>()
val emptyList2: List<String> = emptyList()

val emptySet1 = emptySet<String>()
val emptySet2: Set<String> = emptySet()

val emptyMap1 = emptyMap<String, String>()
val emptyMap2: Map<String, String> = emptyMap()
```

1 Setup new Ktlint Ruleset project

ktlint

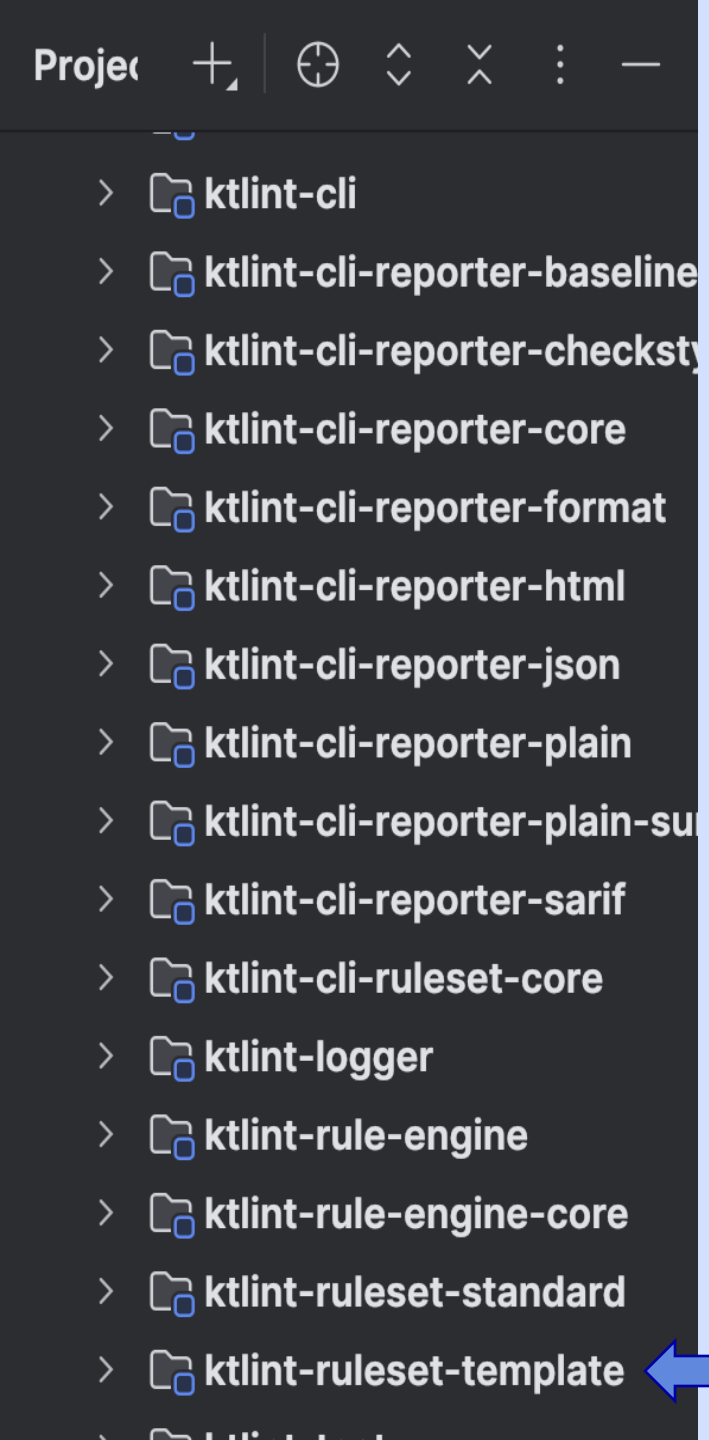
Setup new Ktlint Ruleset project

1. Create new project directory “ktlint-ruleset-demo-kdd”

Setup new Ktlint Ruleset project

1. Create new project directory “ktlint-ruleset-demo-kdd”
2. Add “.gitignore”

```
/.idea/
/.gradle/
/build/
```

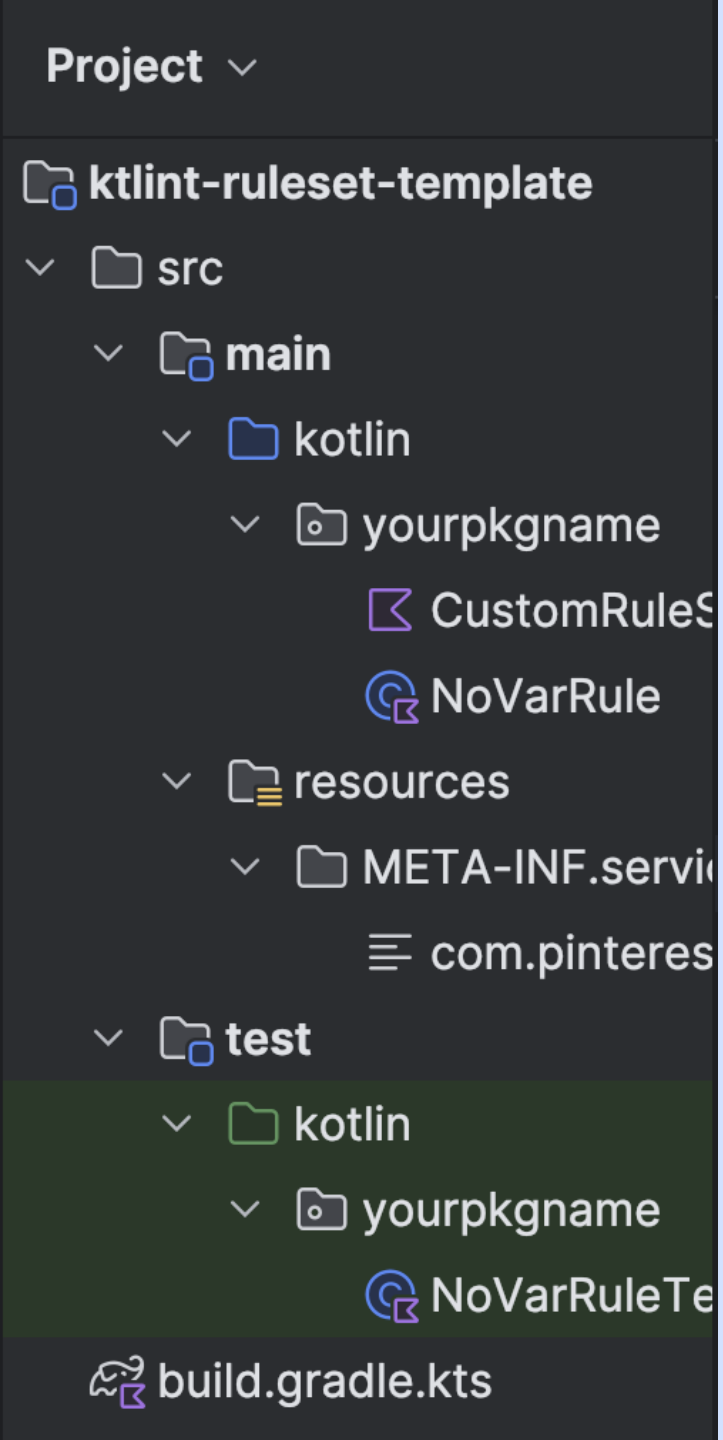


Setup new Ktlint Ruleset project

1. Create new project directory “ktlint-ruleset-demo-kdd”
2. Add “.gitignore”

```
/.idea/
/.gradle/
/build/
```

3. Copy all files from https://github.com/pinterest/ktlint/tree/master/ktlint-ruleset-template



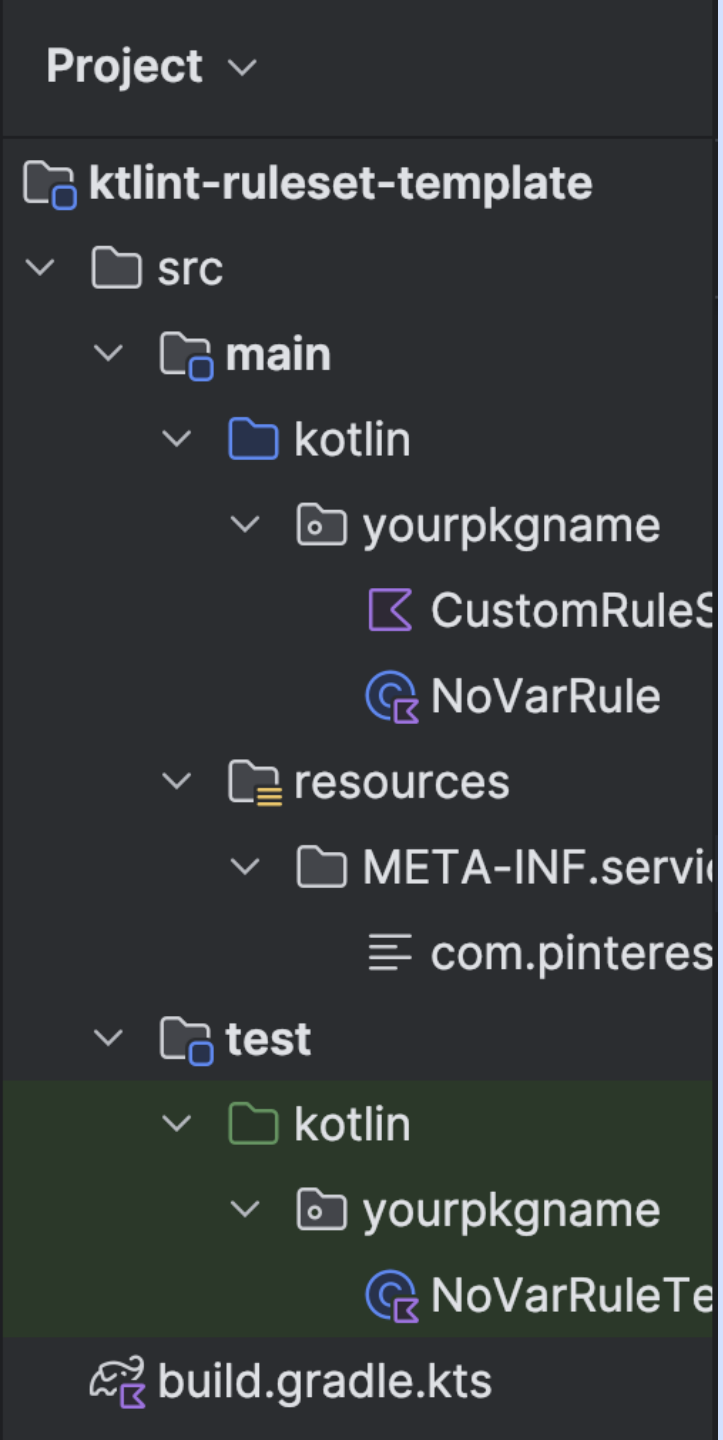
Setup new Ktlint Ruleset project

1. Create new project directory “ktlint-ruleset-demo-kdd”
2. Add “.gitignore”

```
/.idea/
/.gradle/
/build/
```

3. Copy all files from https://github.com/pinterest/ktlint/tree/master/ktlint-ruleset-template
4. Decomment block below in “gradle.build.kts”

```
repositories {
    mavenCentral()
}
```



Setup new Ktlint Ruleset project

1. Create new project directory “ktlint-ruleset-demo-kdd”
2. Add “.gitignore”

```
/.idea/
/.gradle/
/build/
```

3. Copy all files from https://github.com/pinterest/ktlint/tree/master/ktlint-ruleset-template
4. Decomment block below in “gradle.build.kts”

```
repositories {
    mavenCentral()
}
```

5. Run gradle test

2 Abstract Syntax Tree

Basic knowledge

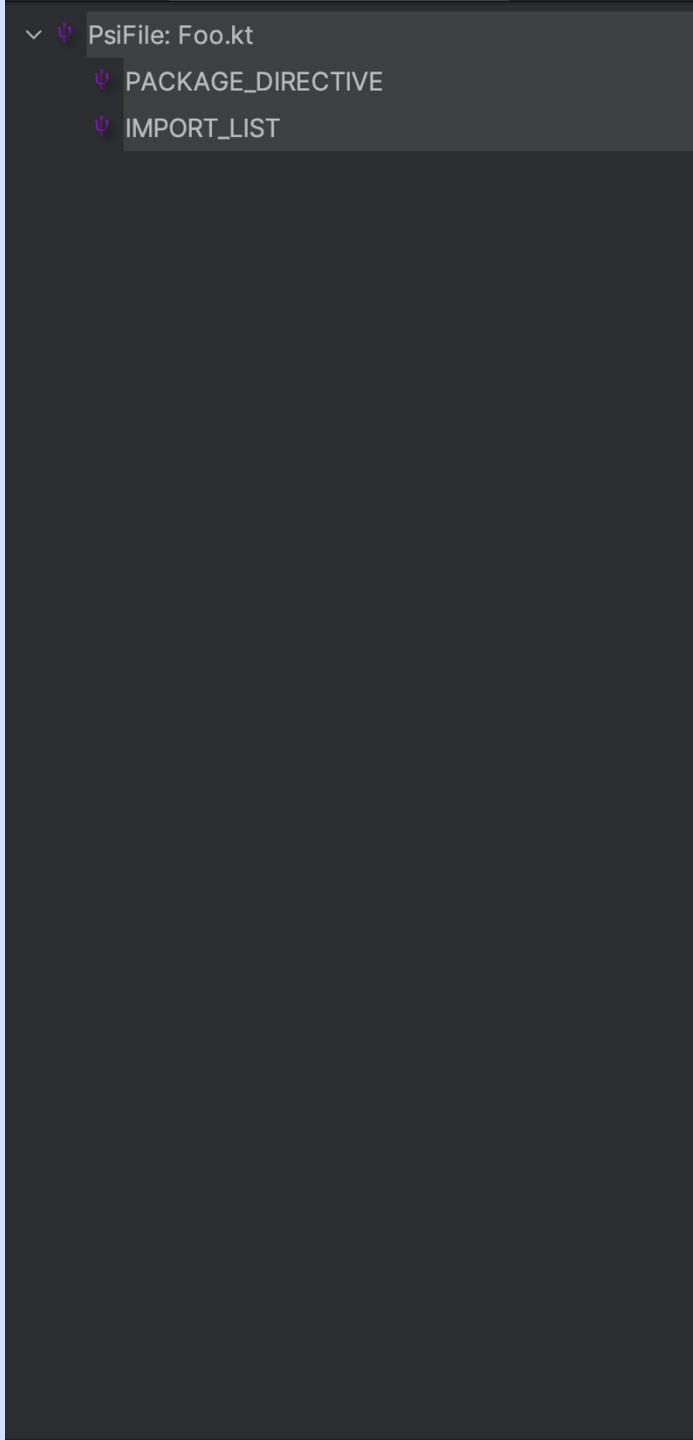
Abstract Syntax Tree (AST)

- Each valid Kotlin file can be parsed into a hierarchical tree of AST Nodes
- Produced via the embedded Kotlin compiler

```
data class Foo(
    val a: String,
    val b: String,
)
```

Abstract Syntax Tree (AST)

- Each valid Kotlin file can be parsed into a hierarchical tree of AST Nodes
- Produced via the embedded Kotlin compiler

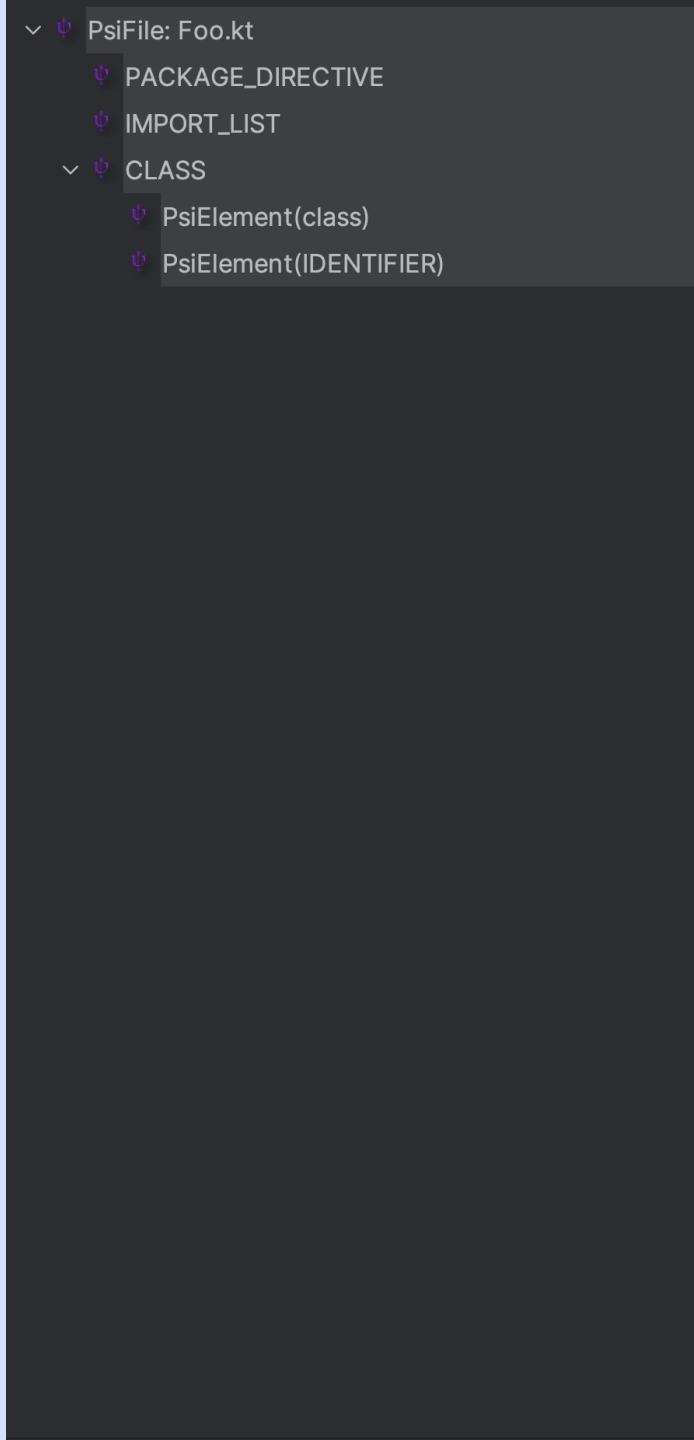


Abstract Syntax Tree (AST)

- Each valid Kotlin file can be parsed into

  a hierarchical tree of AST Nodes
- Produced via the embedded Kotlin compiler
- PSI Elements are AST Nodes with additional context information

```
class Foo
```

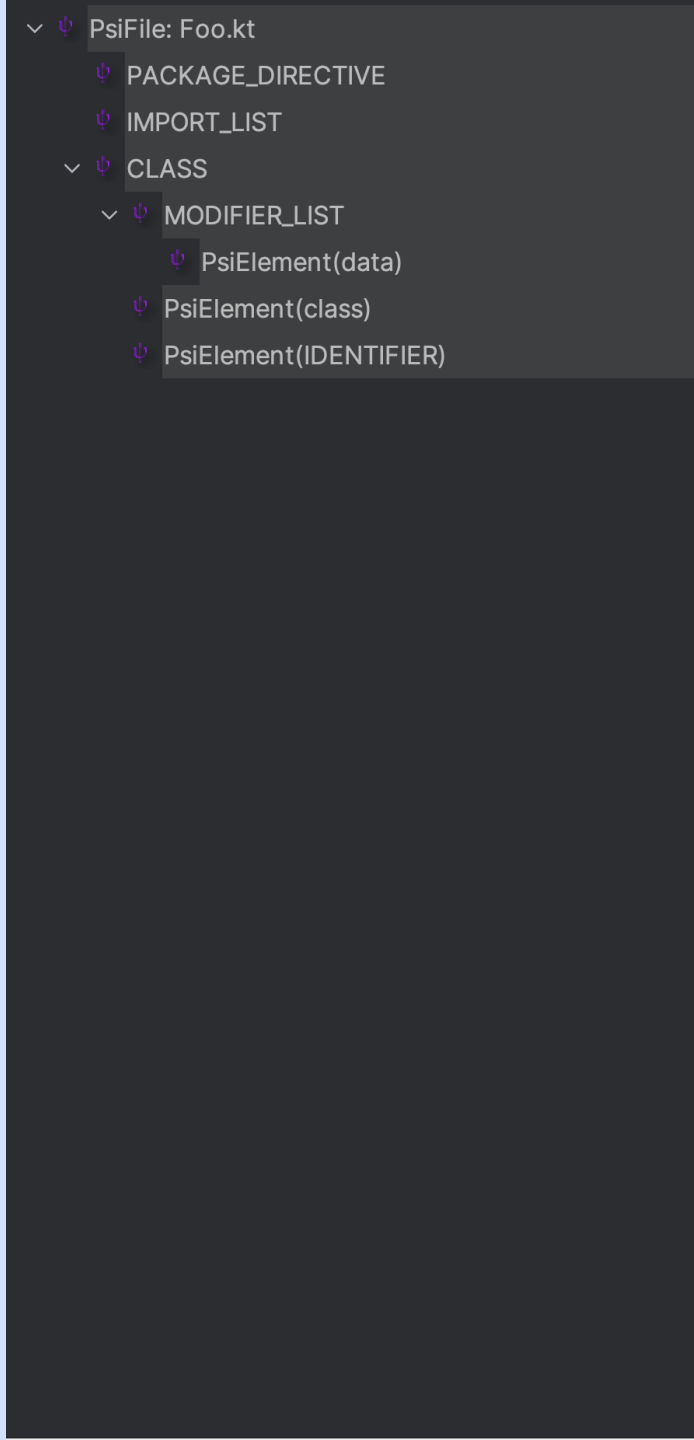


Abstract Syntax Tree (AST)

- Each valid Kotlin file can be parsed into

  a hierarchical tree of AST Nodes
- Produced via the embedded Kotlin compiler
- PSI Elements are AST Nodes with additional context information

```
data class Foo
```

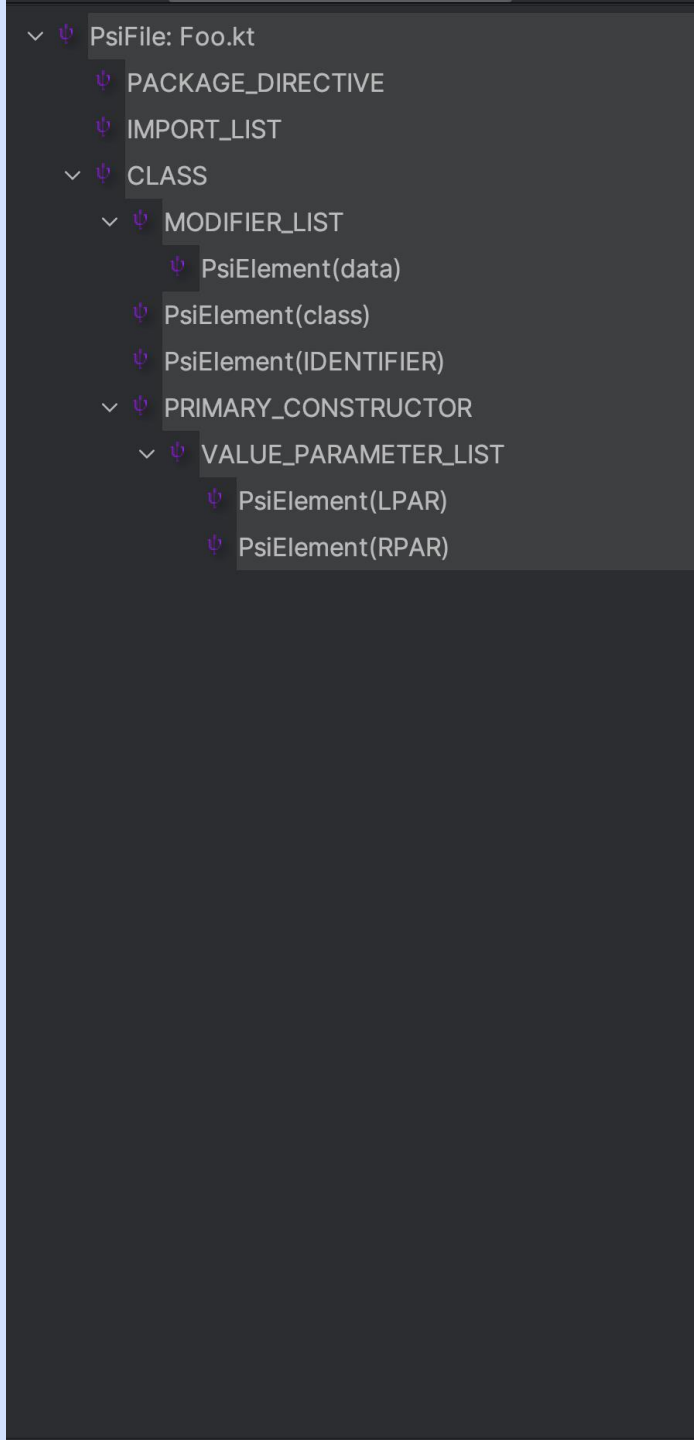


Abstract Syntax Tree (AST)

- Each valid Kotlin file can be parsed into

  a hierarchical tree of AST Nodes
- Produced via the embedded Kotlin compiler
- PSI Elements are AST Nodes with additional context information

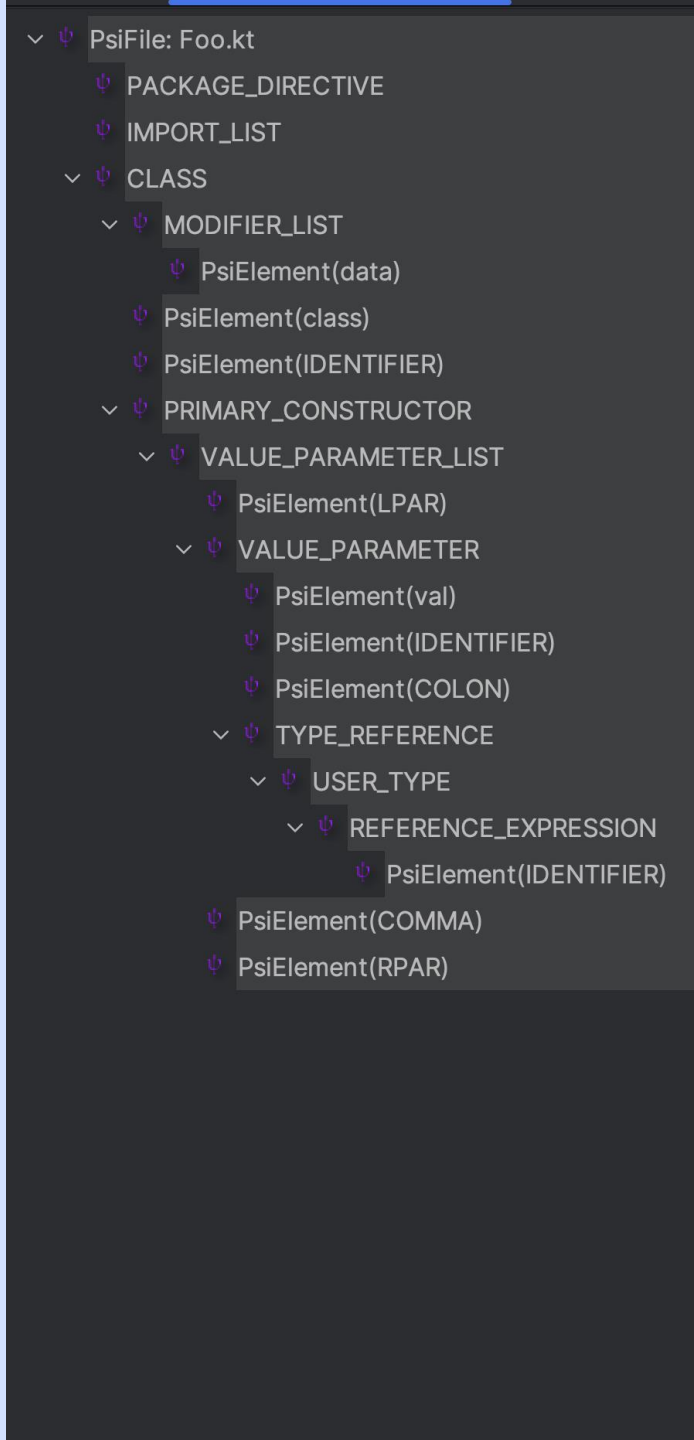
```
data class Foo(


)
```



Abstract Syntax Tree (AST)

- Each valid Kotlin file can be parsed into a hierarchical tree of AST Nodes
- Produced via the embedded Kotlin compiler
- PSI Elements are AST Nodes with additional context information

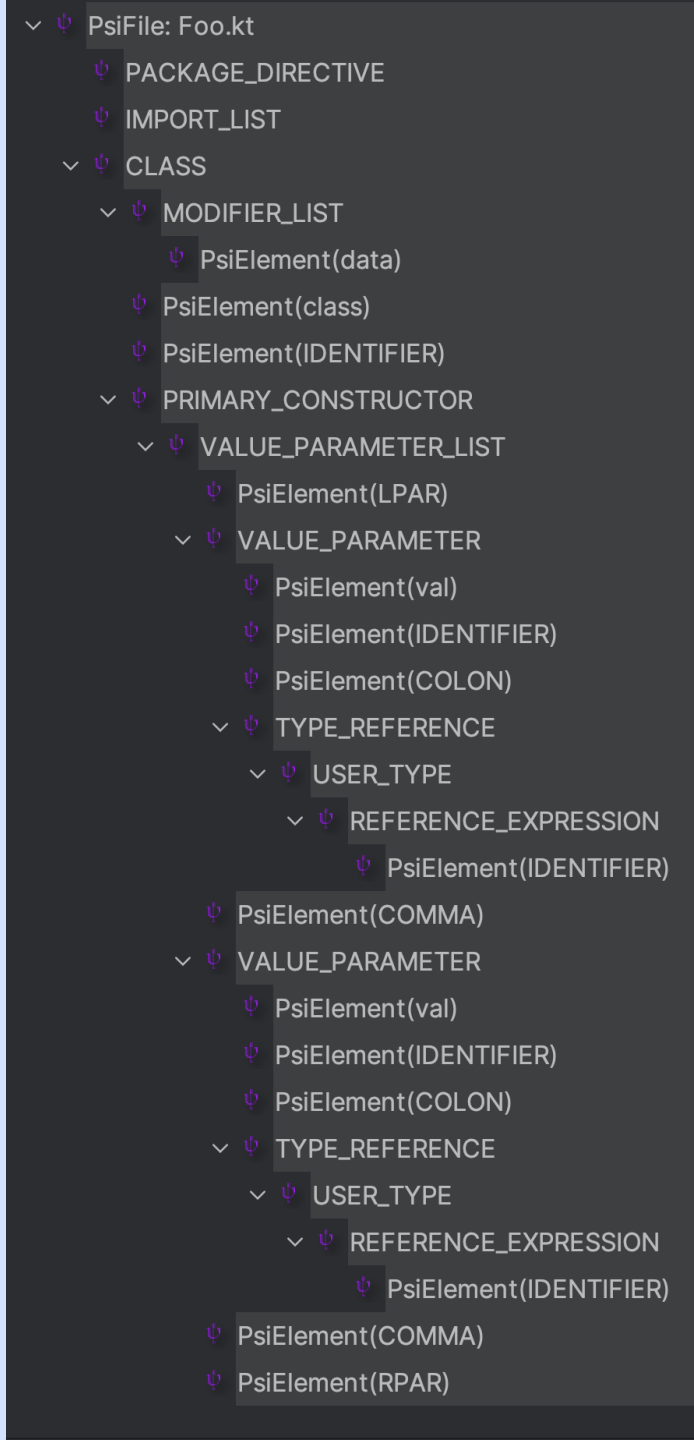
```
data class Foo(
    val a: String,

)
```



Abstract Syntax Tree (AST)

- Each valid Kotlin file can be parsed into a hierarchical tree of AST Nodes
- Produced via the embedded Kotlin compiler
- PSI Elements are AST Nodes with additional context information

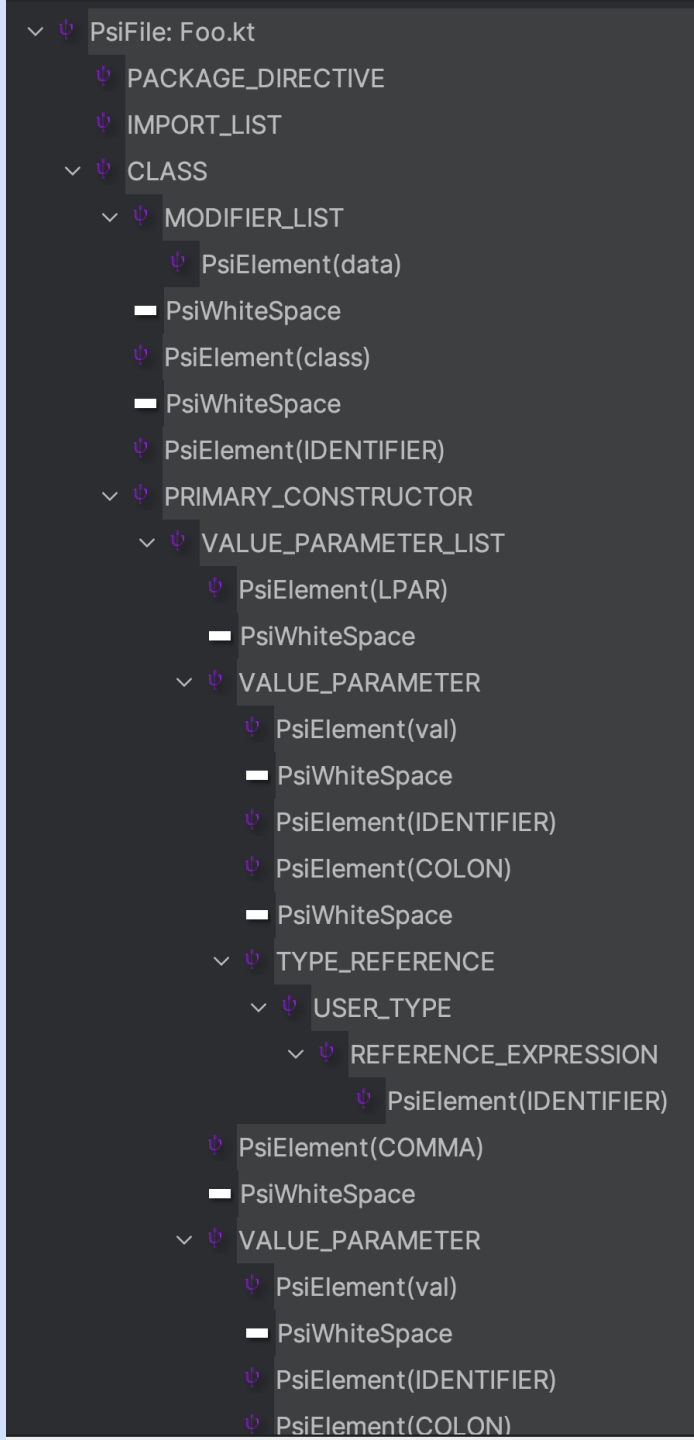
```
data class Foo(
    val a: String,
    val b: String,
)
```



Abstract Syntax Tree (AST)

- Each valid Kotlin file can be parsed into a hierarchical tree of AST Nodes
- Produced via the embedded Kotlin compiler
- PSI Elements are AST Nodes with additional context information
- Whitespaces are also AST nodes

```
data class Foo(
    val a: String,
    val b: String,
)
```



3 Implementing the rule

Base implementation of test class

```
class EmptyCollectionInitializationRuleTest {
}
```

```
class EmptyCollectionInitializationRuleTest {
    @Test
    fun `No violations found when variable is initialized with emptyList()`() {
        val code =
            """
            val x = emptyList<String>()
            """.trimIndent()
        // assertions
    }
}
```

```
class EmptyCollectionInitializationRuleTest {
    @Test
    fun `No violations found when variable is initialized with emptyList()`() {
        val code =
            """
            val x = emptyList<String>()
            """.trimIndent()
        assertThatRule { EmptyCollectionInitializationRule() }(code)
            // assertions
    }
}
```

```
class EmptyCollectionInitializationRuleTest {
    private val emptyCollectionInitializationRuleAssertThat = assertThatRule { EmptyCollectionInitializationRule() }

    @Test
    fun `No violations found when variable is initialized with emptyList()`() {
        val code =
            """
            val x = emptyList<String>()
            """.trimIndent()
        emptyCollectionInitializationRuleAssertThat(code).hasNoLintViolations()
    }
}
```

❌ Compilation error

Base implementation of class

```
public class EmptyCollectionInitializationRule {
}
```

```
class EmptyCollectionInitializationRule :
    Rule(
        ruleId = RuleId("$CUSTOM_RULE_SET_ID:empty-collection-initialization-rule"),
        about =
            About(
                maintainer = "Your name",
                repositoryUrl = "https://github.com/your/project/",
                issueTrackerUrl = "https://github.com/your/project/issues",
            ),
    )
```

☑ Test succeeds

About is used for unhandled exceptions

```
Rule 'custom-rule-set-id:empty-collection-initialization-rule' throws exception in file '<stdin>' at position (1:1)
   Rule maintainer: Your name
   Issue tracker  : https://github.com/your/project/issues
   Repository     : https://github.com/your/project/
com.pinterest.ktlint.rule.engine.api.KtLintRuleException: Rule 'custom-rule-set-id:empty-collection-initialization-rule'
   Rule maintainer: Your name
   Issue tracker  : https://github.com/your/project/issues
   Repository     : https://github.com/your/project/
    at com.pinterest.ktlint.rule.engine.internal.RuleExecutionContext.executeRule(RuleExecutionContext.kt:75)
    at com.pinterest.ktlint.rule.engine.internal.CodeFormatter.executeRule(CodeFormatter.kt:138)
    at com.pinterest.ktlint.rule.engine.internal.CodeFormatter.format(CodeFormatter.kt:109)
    at com.pinterest.ktlint.rule.engine.internal.CodeFormatter.format(CodeFormatter.kt:58)
    at com.pinterest.ktlint.rule.engine.internal.CodeFormatter.format(CodeFormatter.kt:28)
    at com.pinterest.ktlint.rule.engine.api.KtLintRuleEngine.lint(KtLintRuleEngine.kt:91)
    at com.pinterest.ktlint.test.KtLintAssertThatAssertable.lint(KtLintAssertThat.kt:723)
    at com.pinterest.ktlint.test.KtLintAssertThatAssertable.hasNoLintViolations(KtLintAssertThat.kt:441)
 <5 folded frames>
Caused by: kotlin.NotImplementedError: An operation is not implemented.
    at yourpkgname.EmptyCollectionInitializationRule.beforeVisitChildNodes(EmptyCollectionInitializationRule.kt:23)
```

Rule class

```
public open class Rule(  159 Inheritors
    public open val ruleId: RuleId,
    public open val about: About,
    public open val visitorModifiers: Set<VisitorModifier> = emptySet(),  2 Implementations
    public open val usesEditorConfigProperties: Set<EditorConfigProperty<*>> = emptySet(),  2 Implementations
) {
    public open fun beforeFirstNode(editorConfig: EditorConfig) {}  45 Overrides

    @Deprecated(message = "Marked for removal in Ktlint 2.0. Please implement interface RuleAutocorrectApproveHandler.")
    public open fun beforeVisitChildNodes(
        node: ASTNode,
        autoCorrect: Boolean,
        emit: (offset: Int, errorMessage: String, canBeAutoCorrected: Boolean) -> Unit,
    ) {
    }

    @Deprecated(message = "Marked for removal in Ktlint 2.0. Please implement interface RuleAutocorrectApproveHandler.")
    public open fun afterVisitChildNodes(
        node: ASTNode,
        autoCorrect: Boolean,
        emit: (offset: Int, errorMessage: String, canBeAutoCorrected: Boolean) -> Unit,
    ) {
    }

    public open fun afterLastNode() {}  6 Usages  2 Overrides
}
```

- Visitor pattern
- The deprecated signatures will be removed in Ktlint 2.0. See RuleAutocorrectApproveHandler for new signatures

RuleAutocorrectApproveHandler

- Contains the replacements for the deprecated Rule signatures

```
public interface RuleAutocorrectApproveHandler {  148 Implementations
    public fun beforeVisitChildNodes(  112 Overrides
        node: ASTNode,
        emit: (offset: Int, errorMessage: String, canBeAutoCorrected: Boolean) -> AutocorrectDecision,
    ) {
    }

    public fun afterVisitChildNodes(  2 Usages  4 Overrides
        node: ASTNode,
        emit: (offset: Int, errorMessage: String, canBeAutoCorrected: Boolean) -> AutocorrectDecision,
    ) {
    }
}
```

- New signatures defer the autocorrect decision per violation to the API Consumer
  - Implemented for all Standard Ktlint rules
  - Strongly encouraged (but optional) for custom rulesets in Ktlint 1.x
  - Mandatory in Ktlint 2.0

```
class EmptyCollectionInitializationRule :
    Rule(
        ruleId = RuleId("$CUSTOM_RULE_SET_ID:empty-collection-initialization-rule"),
        about =
            About(
                maintainer = "Your name",
                repositoryUrl = "https://github.com/your/project/",
                issueTrackerUrl = "https://github.com/your/project/issues",
            ),
    ),
    RuleAutocorrectApproveHandler
```

Test succeeds

```
open class CustomRule internal constructor(
    id: String,
) : Rule(
        ruleId = RuleId("$CUSTOM_RULE_SET_ID:$id"),
        about =
            About(
                maintainer = "Your name",
                repositoryUrl = "https://github.com/your/project/",
                issueTrackerUrl = "https://github.com/your/project/issues",
            ),
    ),
    RuleAutocorrectApproveHandler

class EmptyCollectionInitializationRule : CustomRule("empty-collection-initialization-rule")
```

Assert violations for “listOf” in test class

```kotlin
class EmptyCollectionInitializationRuleTest {
    private val emptyCollectionInitializationRuleAssertThat = assertThatRule { EmptyCollectionInitializationRule() }

    @Test
    fun `Given a collection initialized with listOf() then replace with emptyList()`() {
        val code =
            """
            val x = listOf<String>()
            val y: List<String> = listOf()
            val z = listOf("zzz")
            """.trimIndent()
        emptyCollectionInitializationRuleAssertThat(code)
            // assertions



    }
}
```

```kotlin
class EmptyCollectionInitializationRuleTest {
    private val emptyCollectionInitializationRuleAssertThat = assertThatRule { EmptyCollectionInitializationRule() }

    @Test
    fun `Given a collection initialized with listOf() then replace with emptyList()`() {
        val code =
            """
            val x = listOf<String>()
            val y: List<String> = listOf()
            val z = listOf("zzz")
            """.trimIndent()
        emptyCollectionInitializationRuleAssertThat(code)
            .hasLintViolations(
                LintViolation(1, 9, "Use 'emptyList' instead of 'listOf' to create an empty collection"),
                LintViolation(2, 23, "Use 'emptyList' instead of 'listOf' to create an empty collection"),
            )
    }
}
```

❌ Test failure

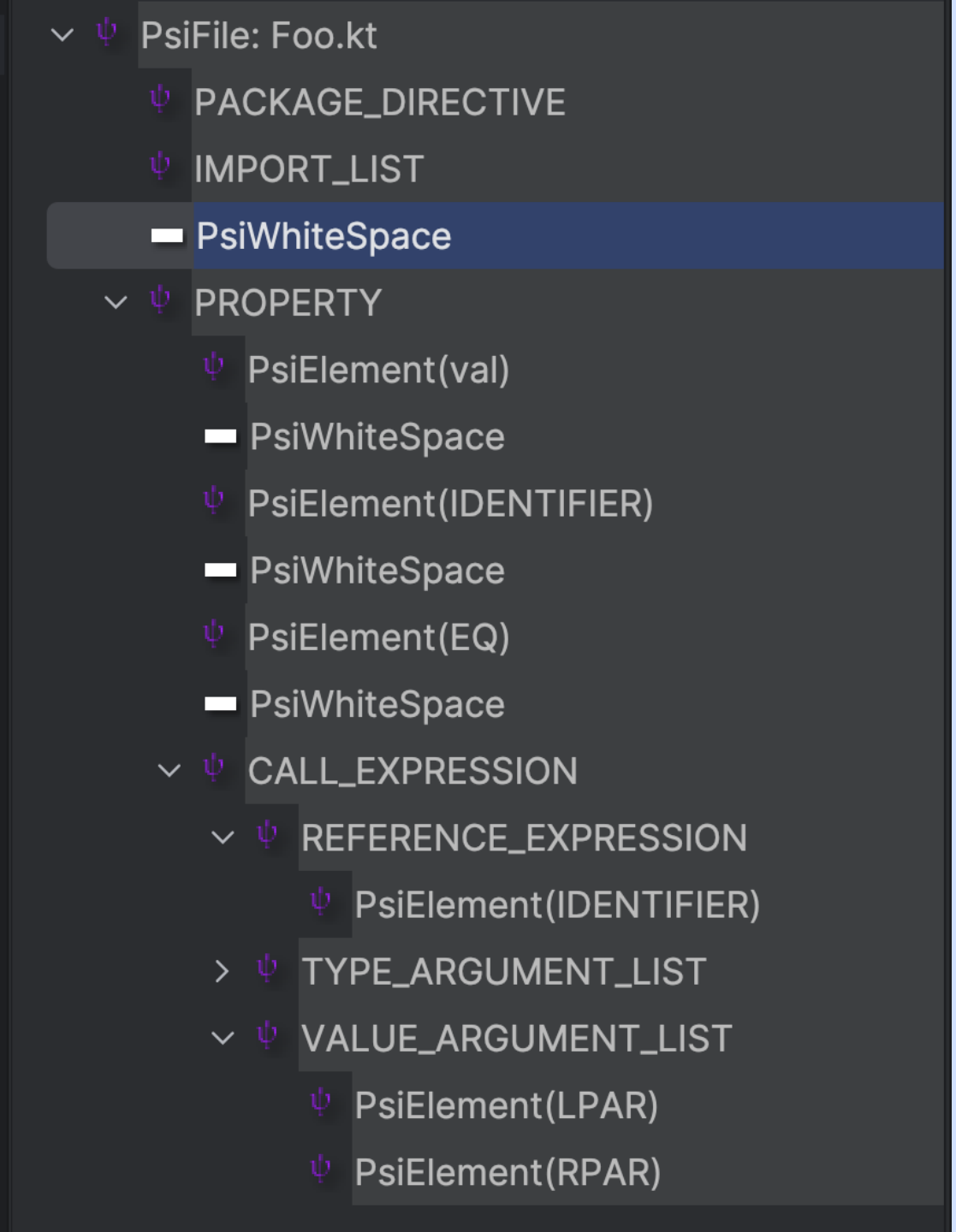
Emit the violations for ”listOf()”

```
class EmptyCollectionInitializationRule : CustomRule("empty-collection-initialization-rule") {
    override fun beforeVisitChildNodes(
        node: ASTNode,
        emit: (offset: Int, errorMessage: String, canBeAutoCorrected: Boolean) -> AutocorrectDecision,
    ) {
    }
}
```

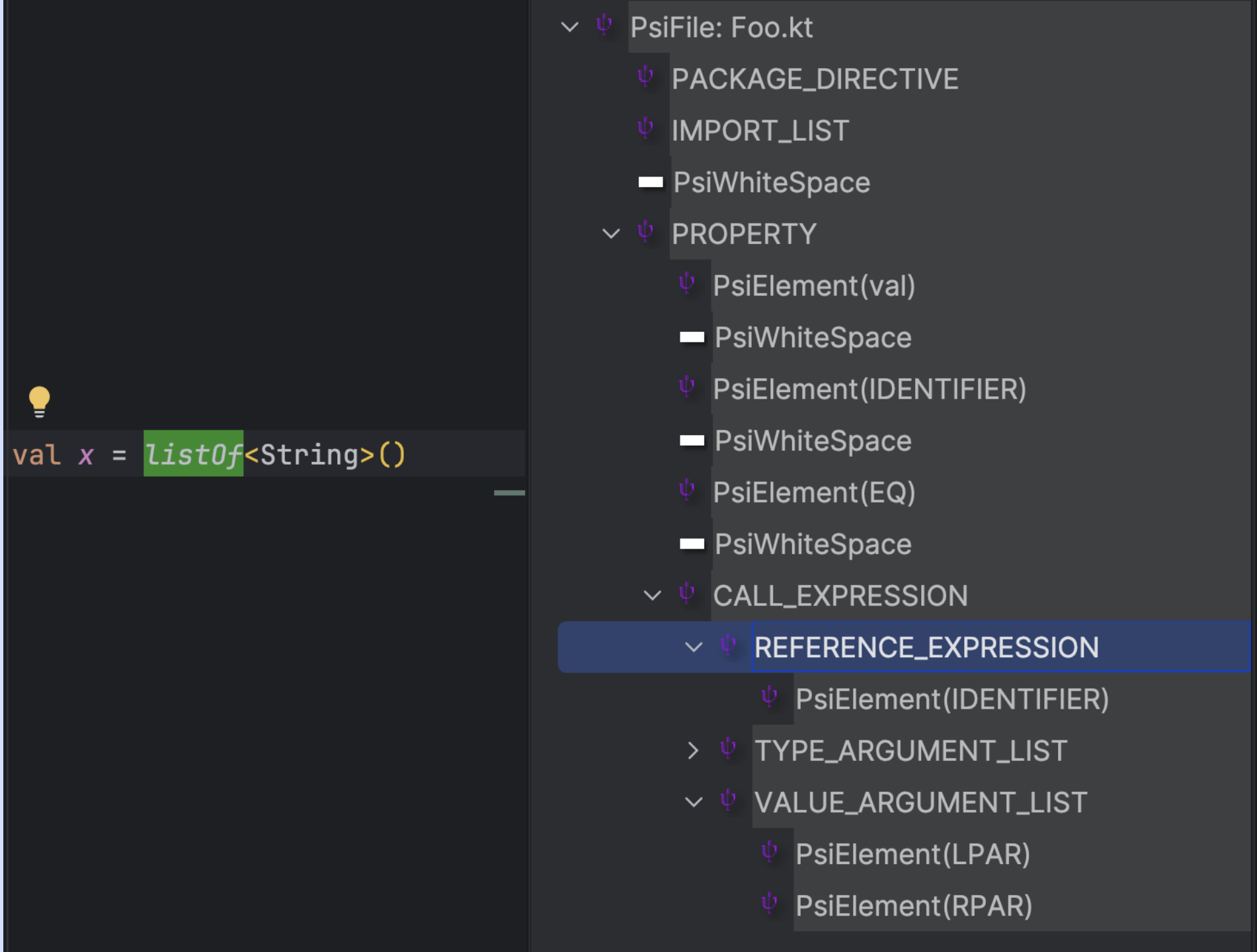
```
class EmptyCollectionInitializationRule : CustomRule("empty-collection-initialization-rule") {
    override fun beforeVisitChildNodes(
        node: ASTNode,
        emit: (offset: Int, errorMessage: String, canBeAutoCorrected: Boolean) -> AutocorrectDecision,
    ) {
        node
            // do something
    }
}
```

Inspect AST

```
val x = listOf<String>()
```

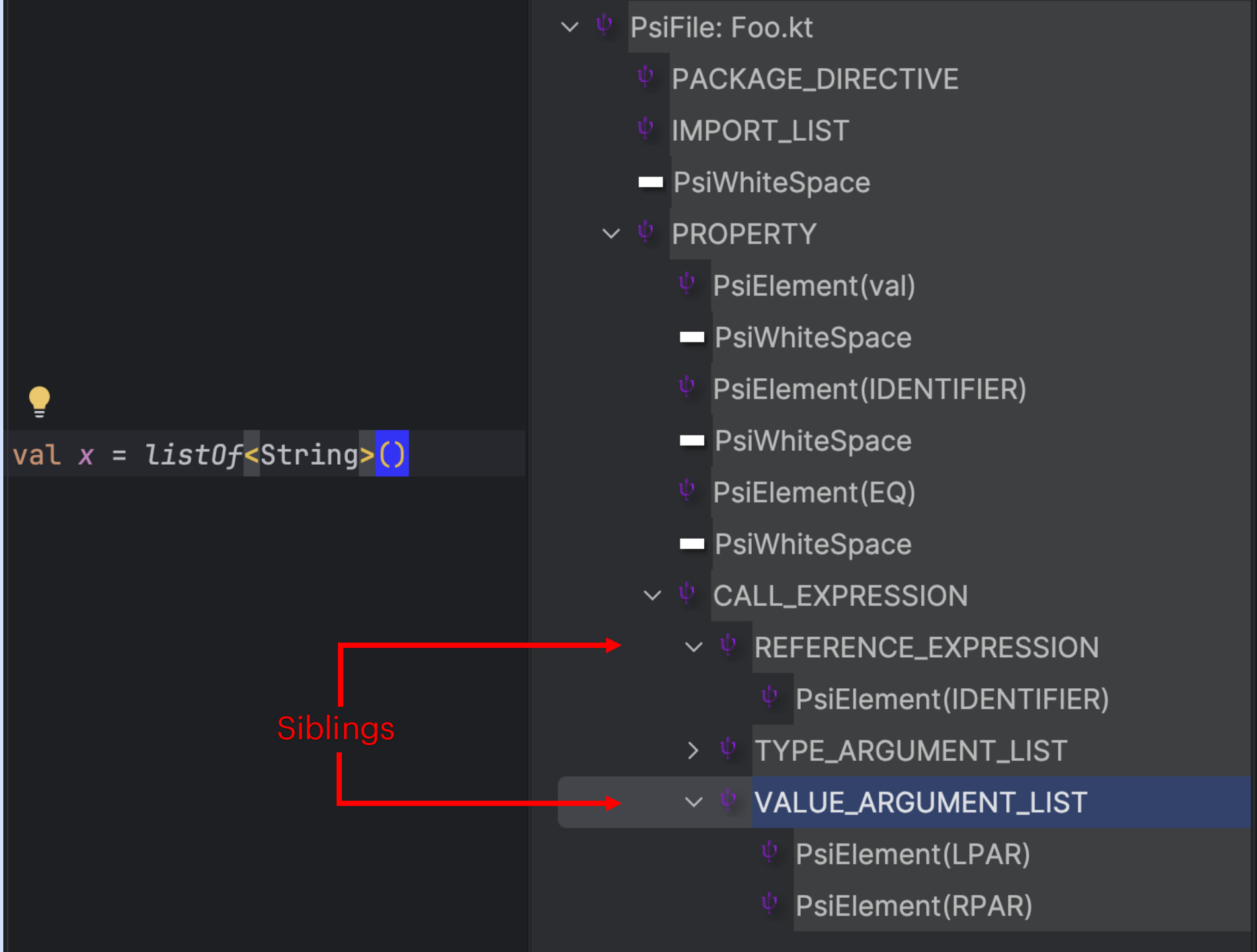


Inspect AST


val x = listOf<String>()
PsiFile: Foo.kt
PACKAGE_DIRECTIVE
IMPORT_LIST
PsiWhiteSpace
PROPERTY
PsiElement(val)
PsiWhiteSpace
PsiElement(IDENTIFIER)
PsiWhiteSpace
PsiElement(EQ)
PsiWhiteSpace
CALL_EXPRESSION
REFERENCE_EXPRESSION
PsiElement(IDENTIFIER)
TYPE_ARGUMENT_LIST
VALUE_ARGUMENT_LIST
PsiElement(LPAR)
PsiElement(RPAR)

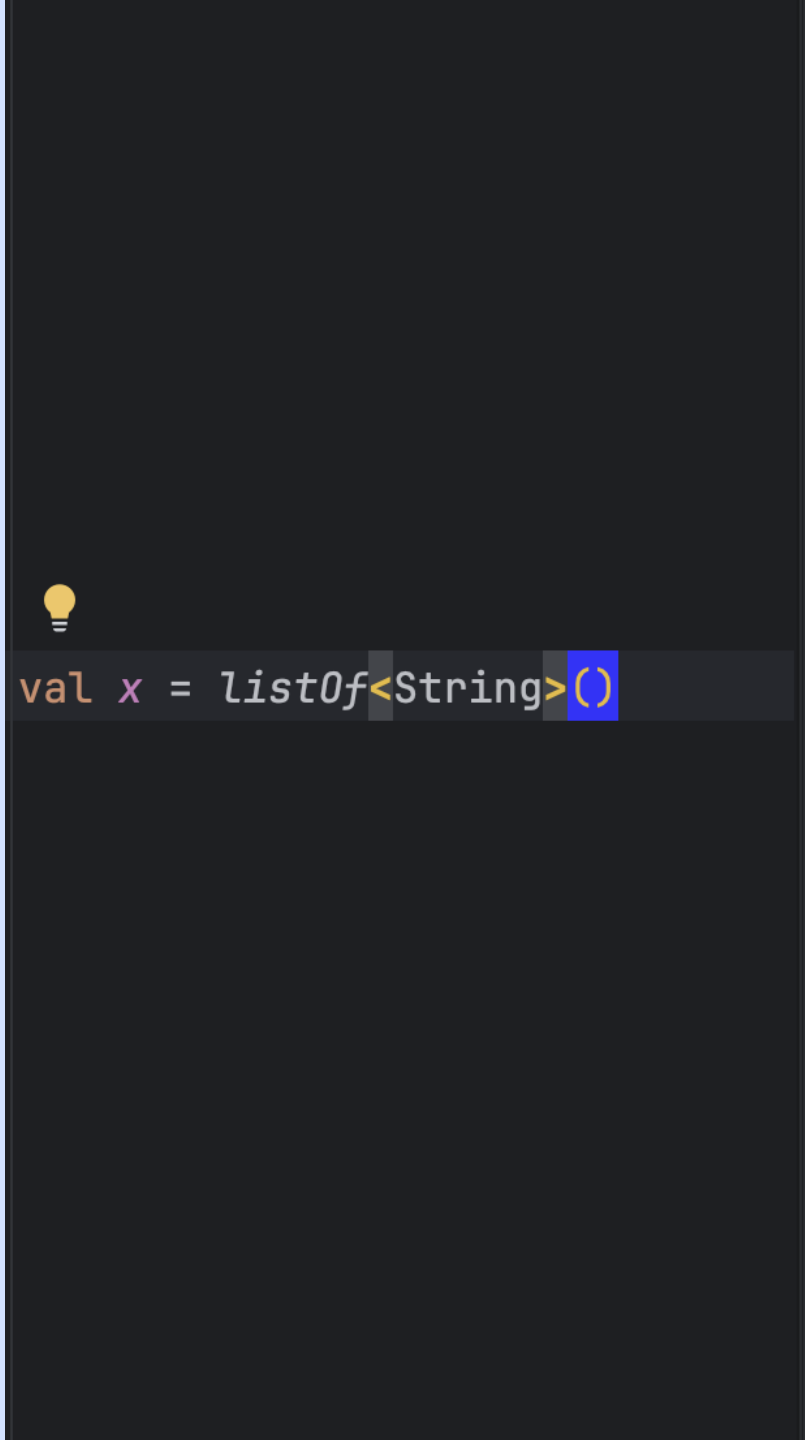
```kotlin
class EmptyCollectionInitializationRule : CustomRule("empty-collection-initialization-rule") {
    override fun beforeVisitChildNodes(
        node: ASTNode,
        emit: (offset: Int, errorMessage: String, canBeAutoCorrected: Boolean) -> AutocorrectDecision,
    ) {
        node
            .takeIf { it.text == "listOf" }
    }
}
```

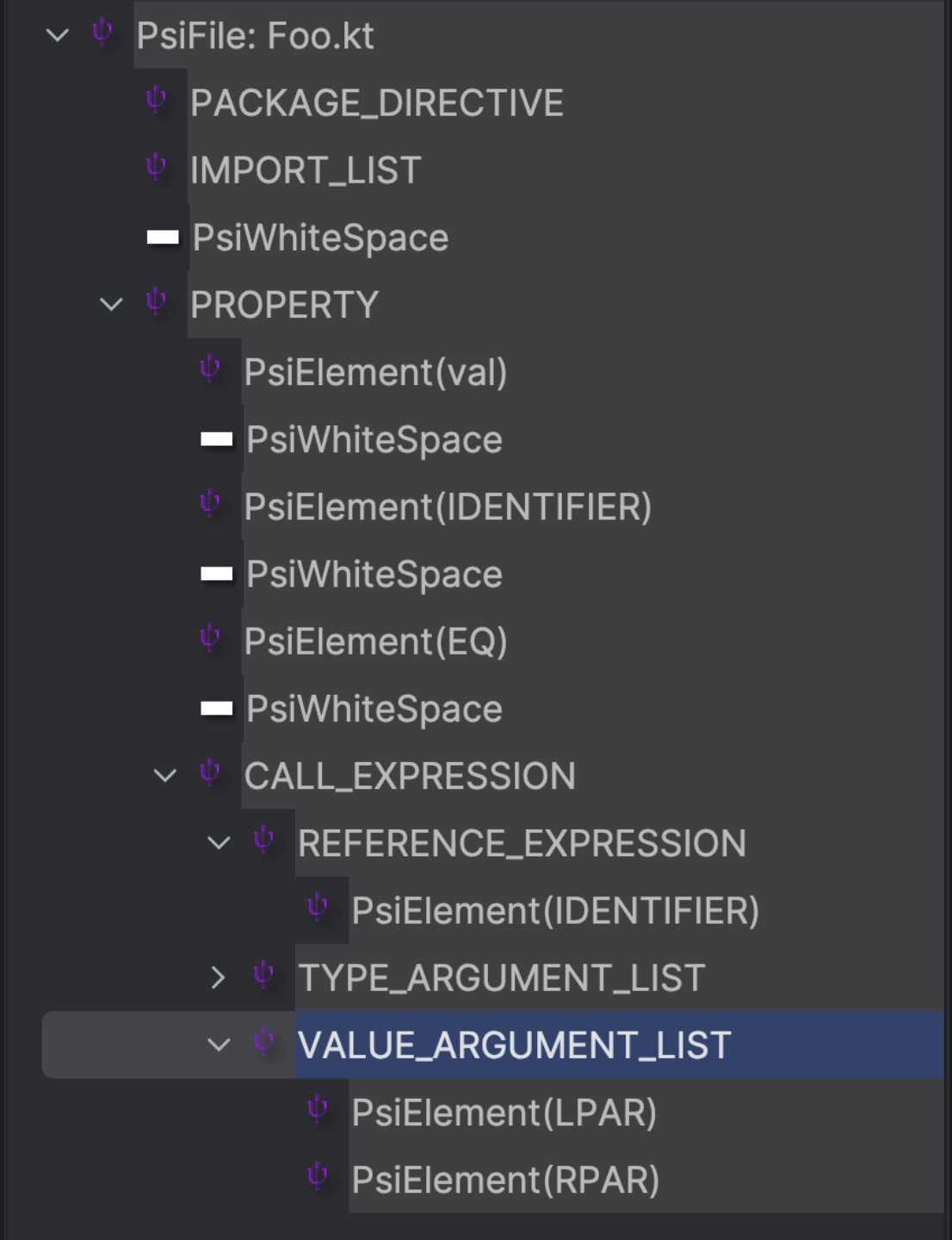
Inspect AST


val x = listOf<String>()
PsiFile: Foo.kt
PACKAGE_DIRECTIVE
IMPORT_LIST
PsiWhiteSpace
PROPERTY
PsiElement(val)
PsiWhiteSpace
PsiElement(IDENTIFIER)
PsiWhiteSpace
PsiElement(EQ)
PsiWhiteSpace
CALL_EXPRESSION
REFERENCE_EXPRESSION
PsiElement(IDENTIFIER)
TYPE_ARGUMENT_LIST
VALUE_ARGUMENT_LIST
PsiElement(LPAR)
PsiElement(RPAR)
Siblings

```
class EmptyCollectionInitializationRule : CustomRule("empty-collection-initialization-rule") {
    override fun beforeVisitChildNodes(
        node: ASTNode,
        emit: (offset: Int, errorMessage: String, canBeAutoCorrected: Boolean) -> AutocorrectDecision,
    ) {
        node
            .takeIf { it.text == "listOf" }
            ?.nextSibling { it.elementType == VALUE_ARGUMENT_LIST }
    }
}
```

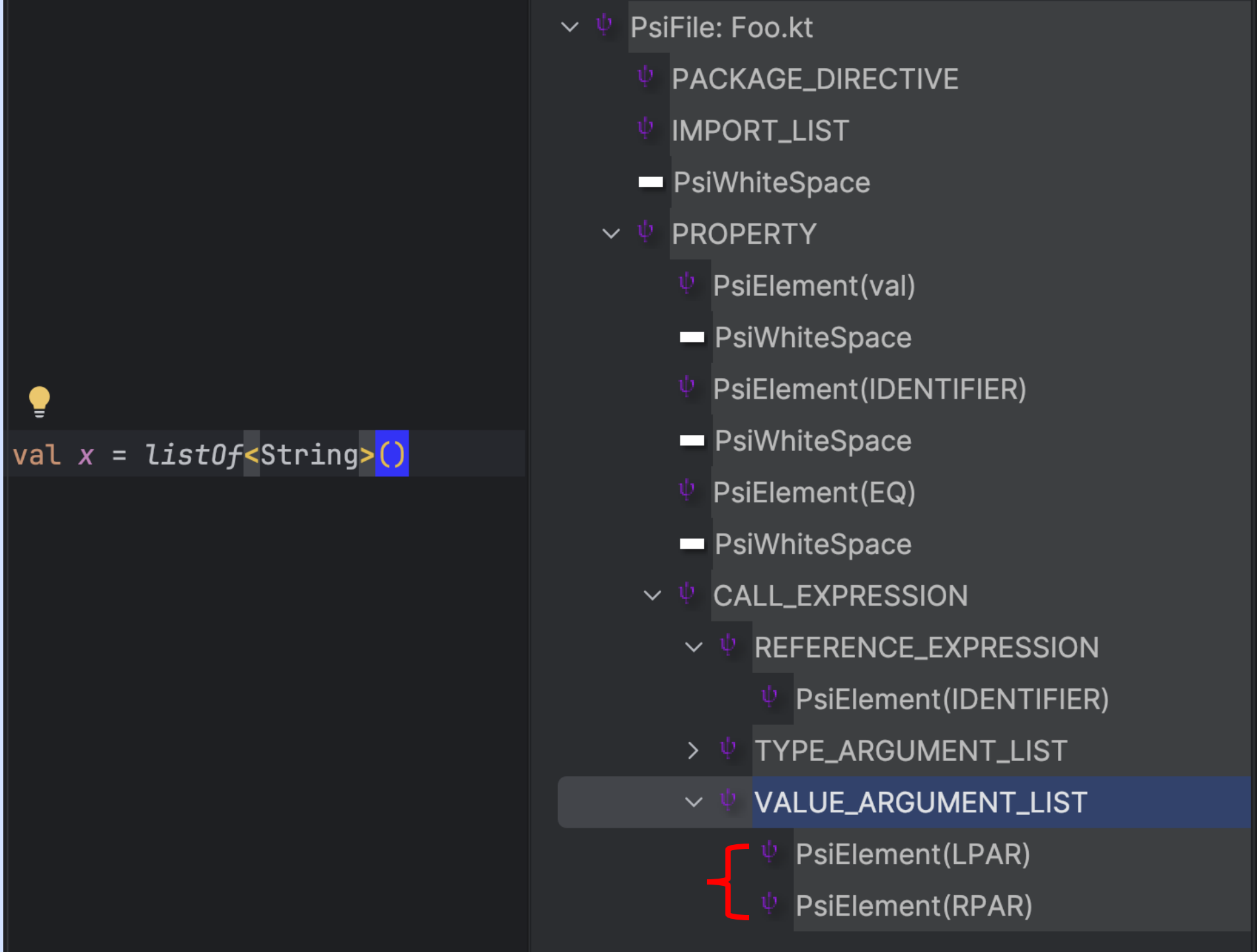
Inspect AST


val x = listOf<String>()


PsiFile: Foo.kt
PACKAGE_DIRECTIVE
IMPORT_LIST
PsiWhiteSpace
PROPERTY
PsiElement(val)
PsiWhiteSpace
PsiElement(IDENTIFIER)
PsiWhiteSpace
PsiElement(EQ)
PsiWhiteSpace
CALL_EXPRESSION
REFERENCE_EXPRESSION
PsiElement(IDENTIFIER)
TYPE_ARGUMENT_LIST
VALUE_ARGUMENT_LIST
PsiElement(LPAR)
PsiElement(RPAR)

```kotlin
class EmptyCollectionInitializationRule : CustomRule("empty-collection-initialization-rule") {
    override fun beforeVisitChildNodes(
        node: ASTNode,
        emit: (offset: Int, errorMessage: String, canBeAutoCorrected: Boolean) -> AutocorrectDecision,
    ) {
        node
            .takeIf { it.text == "listOf" }
            ?.nextSibling { it.elementType == VALUE_ARGUMENT_LIST }
            ?.children()
    }
}
```

Inspect AST

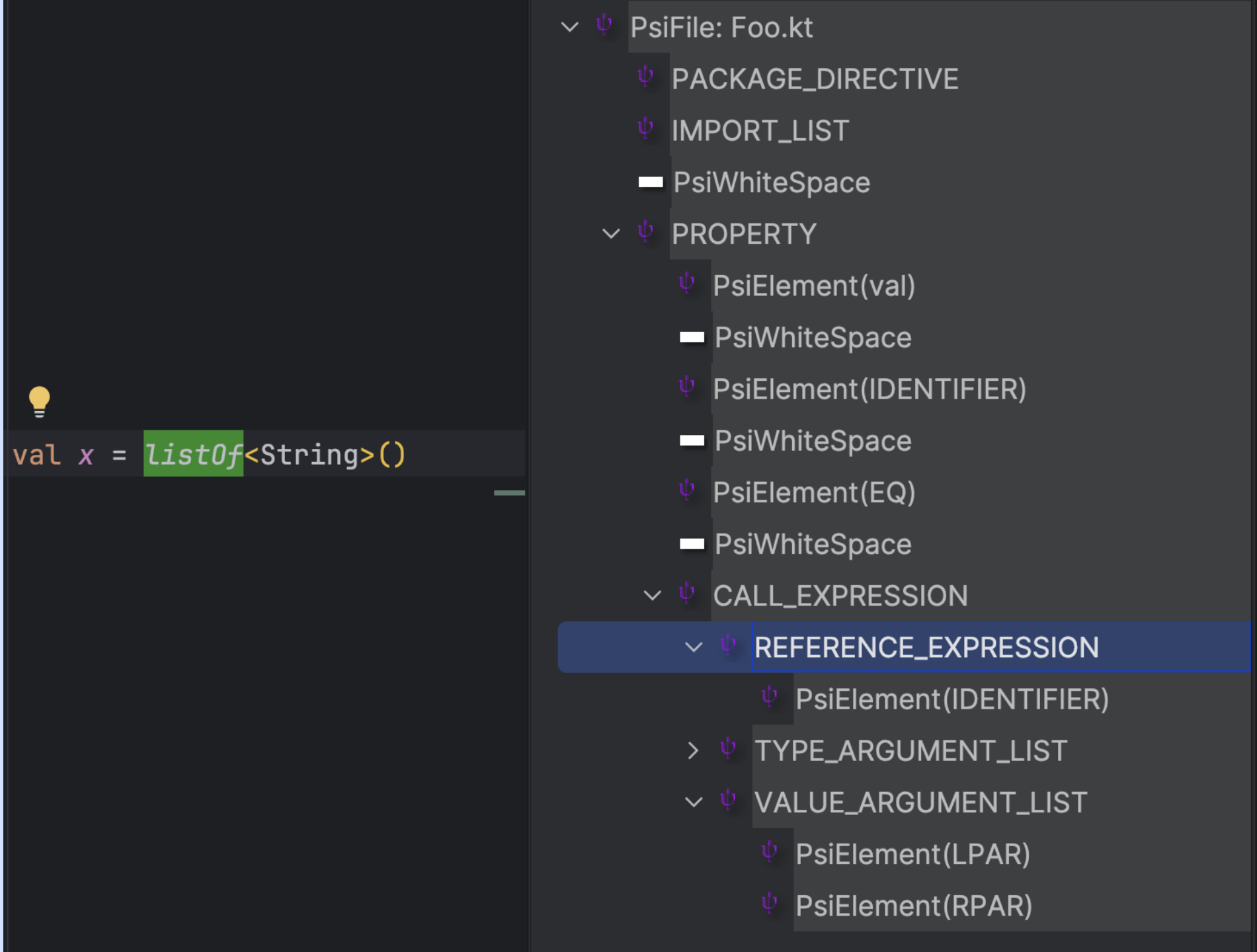

val x = listOf<String>()
PsiFile: Foo.kt
PACKAGE_DIRECTIVE
IMPORT_LIST
PsiWhiteSpace
PROPERTY
PsiElement(val)
PsiWhiteSpace
PsiElement(IDENTIFIER)
PsiWhiteSpace
PsiElement(EQ)
PsiWhiteSpace
CALL_EXPRESSION
REFERENCE_EXPRESSION
PsiElement(IDENTIFIER)
TYPE_ARGUMENT_LIST
VALUE_ARGUMENT_LIST
PsiElement(LPAR)
PsiElement(RPAR)

```
class EmptyCollectionInitializationRule : CustomRule("empty-collection-initialization-rule") {
    override fun beforeVisitChildNodes(
        node: ASTNode,
        emit: (offset: Int, errorMessage: String, canBeAutoCorrected: Boolean) -> AutocorrectDecision,
    ) {
        node
            .takeIf { it.text == "listOf" }
            ?.nextSibling { it.elementType == VALUE_ARGUMENT_LIST }
            ?.children()
            ?.filterNot { it.elementType == LPAR || it.elementType == RPAR }
            ?.toList()
            ?.ifEmpty {

            }
    }
}
```

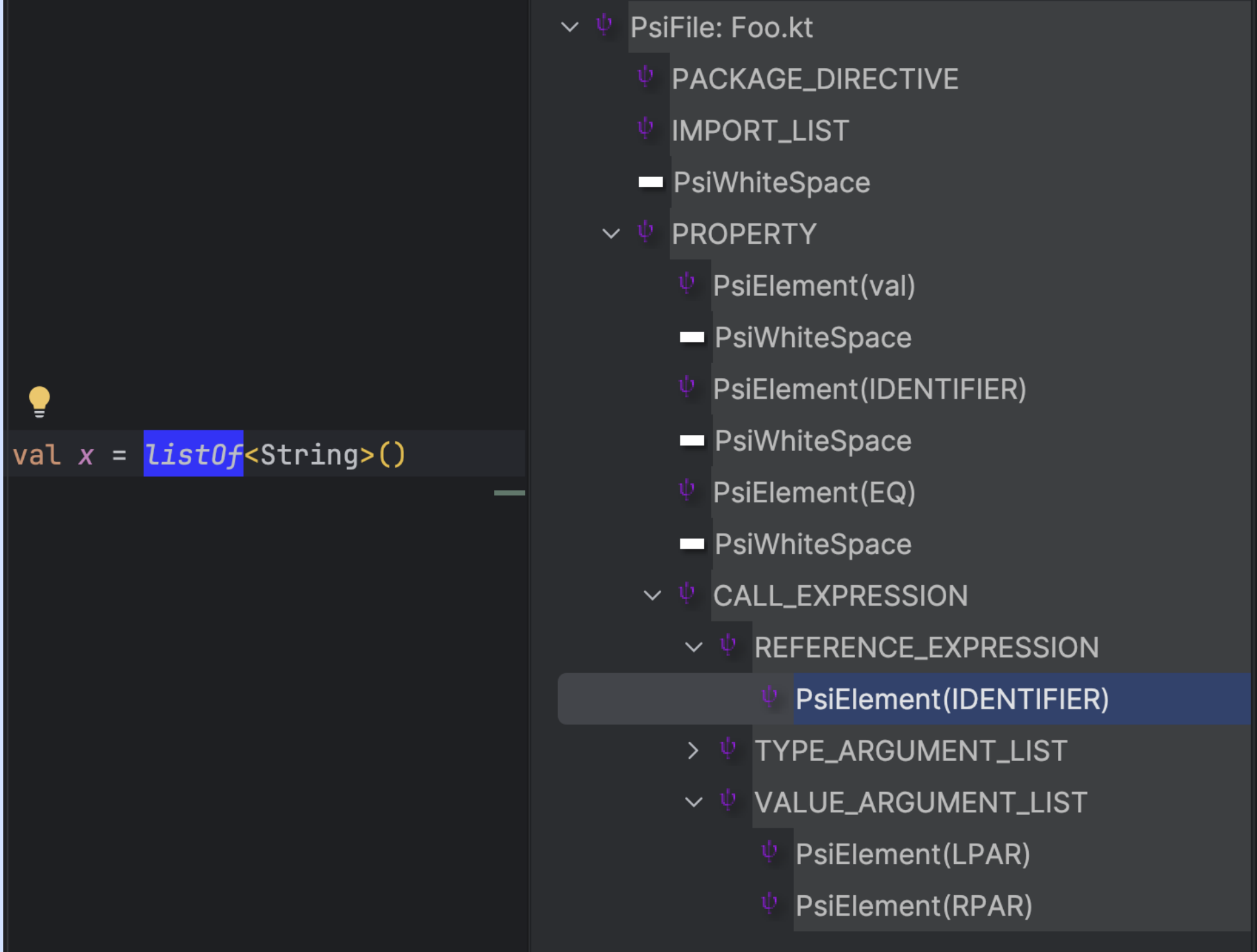
```
class EmptyCollectionInitializationRule : CustomRule("empty-collection-initialization-rule") {
    override fun beforeVisitChildNodes(
        node: ASTNode,
        emit: (offset: Int, errorMessage: String, canBeAutoCorrected: Boolean) -> AutocorrectDecision,
    ) {
        node
            .takeIf { it.text == "listOf" }
            ?.nextSibling { it.elementType == VALUE_ARGUMENT_LIST }
            ?.children()
            ?.filterNot { it.elementType == LPAR || it.elementType == RPAR }
            ?.toList()
            ?.ifEmpty {
                emit(
                    node.startOffset,
                    "Use 'emptyList' instead of 'listOf' to create an empty collection",
                    true,
                )
            }
    }
}
```

Comparing the value of an entire node is risky

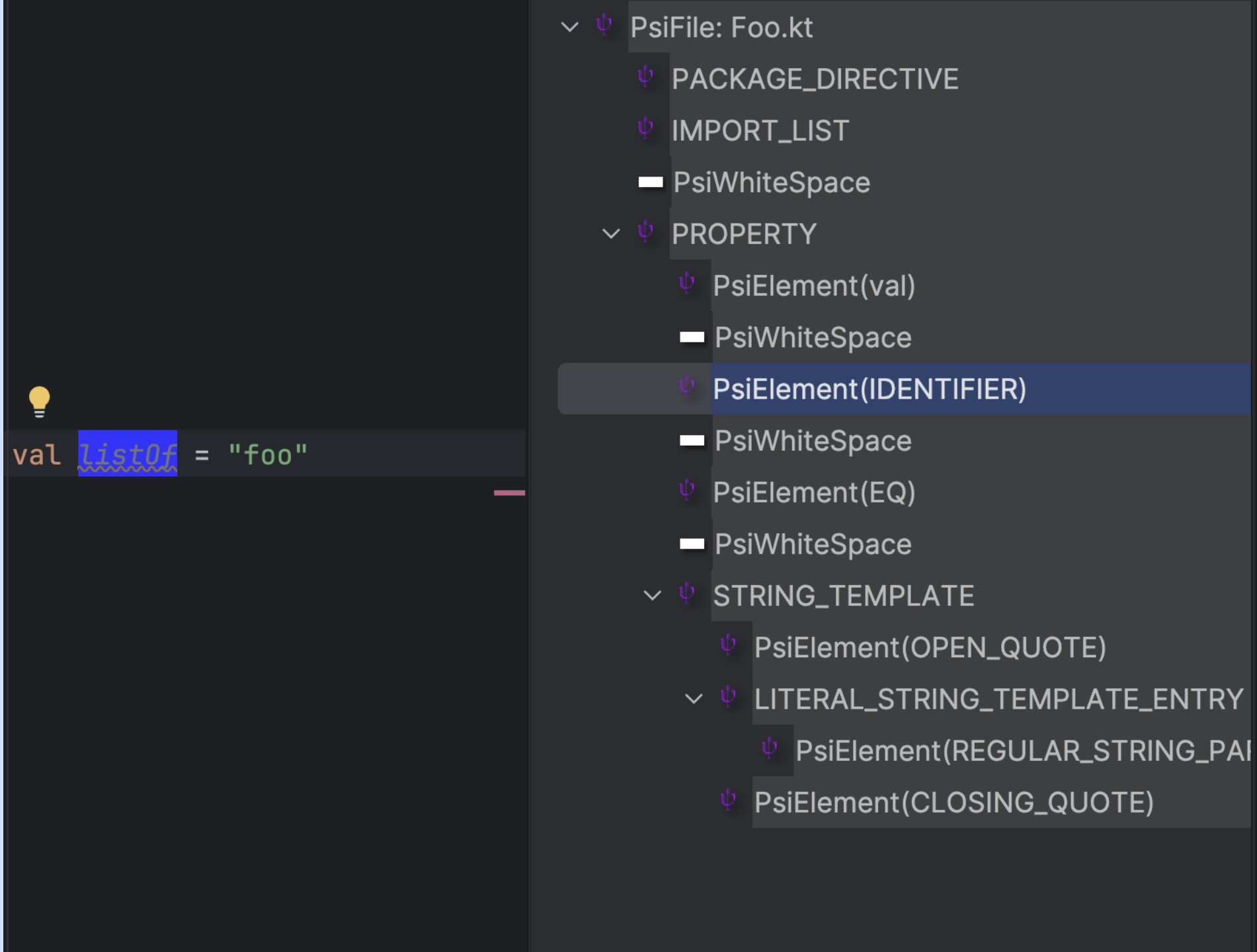
Inspect AST


val x = listOf<String>()
PsiFile: Foo.kt
PACKAGE_DIRECTIVE
IMPORT_LIST
PsiWhiteSpace
PROPERTY
PsiElement(val)
PsiWhiteSpace
PsiElement(IDENTIFIER)
PsiWhiteSpace
PsiElement(EQ)
PsiWhiteSpace
CALL_EXPRESSION
REFERENCE_EXPRESSION
PsiElement(IDENTIFIER)
TYPE_ARGUMENT_LIST
VALUE_ARGUMENT_LIST
PsiElement(LPAR)
PsiElement(RPAR)

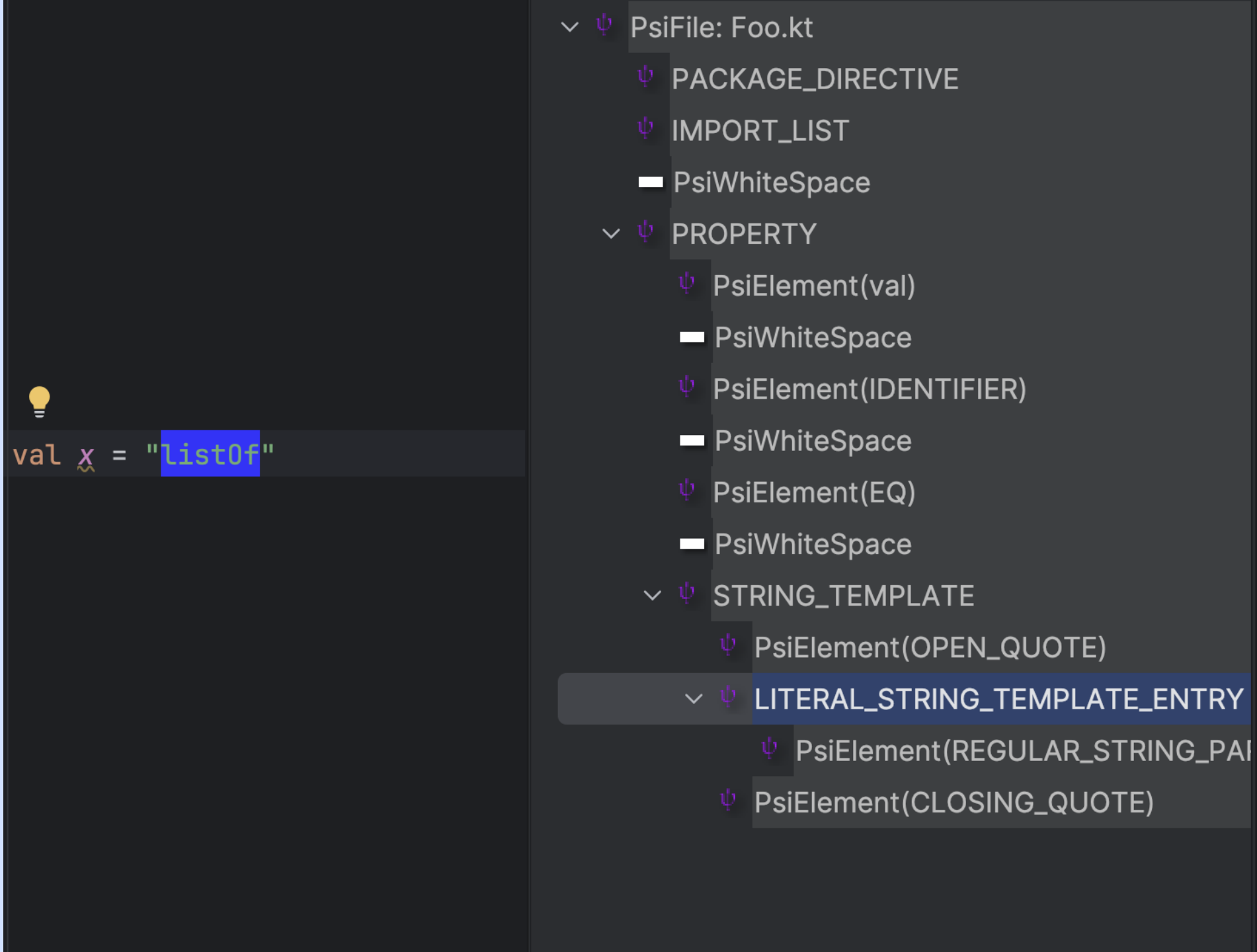
Inspect AST


val x = listOf<String>()
PsiFile: Foo.kt
PACKAGE_DIRECTIVE
IMPORT_LIST
PsiWhiteSpace
PROPERTY
PsiElement(val)
PsiWhiteSpace
PsiElement(IDENTIFIER)
PsiWhiteSpace
PsiElement(EQ)
PsiWhiteSpace
CALL_EXPRESSION
REFERENCE_EXPRESSION
PsiElement(IDENTIFIER)
TYPE_ARGUMENT_LIST
VALUE_ARGUMENT_LIST
PsiElement(LPAR)
PsiElement(RPAR)

Inspect AST


val listOf = "foo"
PsiFile: Foo.kt
PACKAGE_DIRECTIVE
IMPORT_LIST
PsiWhiteSpace
PROPERTY
PsiElement(val)
PsiWhiteSpace
PsiElement(IDENTIFIER)
PsiWhiteSpace
PsiElement(EQ)
PsiWhiteSpace
STRING_TEMPLATE
PsiElement(OPEN_QUOTE)
LITERAL_STRING_TEMPLATE_ENTRY
PsiElement(REGULAR_STRING_PAI
PsiElement(CLOSING_QUOTE)

Inspect AST


val x = "listOf"
PsiFile: Foo.kt
PACKAGE_DIRECTIVE
IMPORT_LIST
PsiWhiteSpace
PROPERTY
PsiElement(val)
PsiWhiteSpace
PsiElement(IDENTIFIER)
PsiWhiteSpace
PsiElement(EQ)
PsiWhiteSpace
STRING_TEMPLATE
PsiElement(OPEN_QUOTE)
LITERAL_STRING_TEMPLATE_ENTRY
PsiElement(REGULAR_STRING_PAI
PsiElement(CLOSING_QUOTE)

```kotlin
@Test
fun `Do not detect other usages of literal listOf`() {
    val code =
        """
        val y = "listOf"
        val listOf = "z"
        """.trimIndent()
    emptyCollectionInitializationRuleAssertThat(code).hasNoLintViolations()
}
```

☑ Test succeeds

Why does it succeed?

```kotlin
@Test
fun `Do not detect other usages of literal listOf`() {
    val code =
        """
        val y = "listOf"
        val listOf = "z"
        """.trimIndent()
    emptyCollectionInitializationRuleAssertThat(code).hasNoLintViolations()
}
```

Thanks to the safe operator, and lookup of next sibling of type VALUE_ARGUMENT_LIST

```kotlin
class EmptyCollectionInitializationRule : CustomRule("empty-collection-initialization-rule") {
    override fun beforeVisitChildNodes(...) {
        node
            .takeIf { it.text == "listOf" }
            ?.nextSibling { it.elementType == VALUE_ARGUMENT_LIST }
            ...
    }
}
```

Assert autocorrect of violations “listOf()”

```kotlin
@Test
fun `Given a collection initialized with listOf() then report violations`() {
    val code =
        """
        val x = listOf<String>()
        val y: List<String> = listOf()
        val z = listOf("zzz")
        """.trimIndent()
    emptyCollectionInitializationRuleAssertThat(code)
        .hasLintViolations(
            LintViolation(1, 9, "Use 'emptyList' instead of 'listOf' to create an empty collection"),
            LintViolation(2, 23, "Use 'emptyList' instead of 'listOf' to create an empty collection"),
        )
}
```

```kotlin
@Test
fun `Given a collection initialized with listOf() then replace with emptyList()`() {
    val code =
        """
        val x = listOf<String>()
        val y: List<String> = listOf()
        val z = listOf("zzz")
        """.trimIndent()
    emptyCollectionInitializationRuleAssertThat(code)
        .hasLintViolations(
            LintViolation(1, 9, "Use 'emptyList' instead of 'listOf' to create an empty collection"),
            LintViolation(2, 23, "Use 'emptyList' instead of 'listOf' to create an empty collection"),
        )
}
```

```kotlin
@Test
fun `Given a collection initialized with listOf() then replace with emptyList()`() {
    val code =
        """
        val x = listOf<String>()
        val y: List<String> = listOf()
        val z = listOf("zzz")
        """.trimIndent()
    val formattedCode =
        """
        val x = emptyList<String>()
        val y: List<String> = emptyList()
        val z = listOf("zzz")
        """.trimIndent()
    emptyCollectionInitializationRuleAssertThat(code)
        .hasLintViolations(
            LintViolation(1, 9, "Use 'emptyList' instead of 'listOf' to create an empty collection"),
            LintViolation(2, 23, "Use 'emptyList' instead of 'listOf' to create an empty collection"),
        )
}
```

```kotlin
@Test
fun `Given a collection initialized with listOf() then replace with emptyList()`() {
    val code =
        """
        val x = listOf<String>()
        val y: List<String> = listOf()
        val z = listOf("zzz")
        """.trimIndent()
    val formattedCode =
        """
        val x = emptyList<String>()
        val y: List<String> = emptyList()
        val z = listOf("zzz")
        """.trimIndent()
    emptyCollectionInitializationRuleAssertThat(code)
        .hasLintViolations(
            LintViolation(1, 9, "Use 'emptyList' instead of 'listOf' to create an empty collection"),
            LintViolation(2, 23, "Use 'emptyList' instead of 'listOf' to create an empty collection"),
        ).isFormattedAs(formattedCode)
}
```

❌ Test failure

Autocorrect violations “listOf()”

```
class EmptyCollectionInitializationRule : CustomRule("empty-collection-initialization-rule") {
    override fun beforeVisitChildNodes(
        node: ASTNode,
        emit: (offset: Int, errorMessage: String, canBeAutoCorrected: Boolean) -> AutocorrectDecision,
    ) {
        node
            .takeIf { it.text == "listOf" }
            ?.nextSibling { it.elementType == VALUE_ARGUMENT_LIST }
            ?.children()
            ?.filterNot { it.elementType == LPAR || it.elementType == RPAR }
            ?.toList()
            ?.ifEmpty {
                emit(
                    node.startOffset,
                    "Use 'emptyList' instead of 'listOf' to create an empty collection",
                    true,
                )
            }
    }
}
```

```
class EmptyCollectionInitializationRule : CustomRule("empty-collection-initialization-rule") {
    override fun beforeVisitChildNodes(
        node: ASTNode,
        emit: (offset: Int, errorMessage: String, canBeAutoCorrected: Boolean) -> AutocorrectDecision,
    ) {
        node
            .takeIf { it.text == "listOf" }
            ?.nextSibling { it.elementType == VALUE_ARGUMENT_LIST }
            ?.children()
            ?.filterNot { it.elementType == LPAR || it.elementType == RPAR }
            ?.toList()
            ?.ifEmpty {
                emit(
                    node.startOffset,
                    "Use 'emptyList' instead of 'listOf' to create an empty collection",
                    true,
                ).ifAutocorrectAllowed {

                }
            }
    }
}
```

```
class EmptyCollectionInitializationRule : CustomRule("empty-collection-initialization-rule") {
    override fun beforeVisitChildNodes(
        node: ASTNode,
        emit: (offset: Int, errorMessage: String, canBeAutoCorrected: Boolean) -> AutocorrectDecision,
    ) {
        node
            .takeIf { it.text == "listOf" }
            ?.nextSibling { it.elementType == VALUE_ARGUMENT_LIST }
            ?.children()
            ?.filterNot { it.elementType == LPAR || it.elementType == RPAR }
            ?.toList()
            ?.ifEmpty {
                emit(
                    node.startOffset,
                    "Use 'emptyList' instead of 'listOf' to create an empty collection",
                    true,
                ).ifAutocorrectAllowed {
                    node
                        .findChildByType(IDENTIFIER)
                }
            }
    }
}
```

```
class EmptyCollectionInitializationRule : CustomRule("empty-collection-initialization-rule") {
    override fun beforeVisitChildNodes(
        node: ASTNode,
        emit: (offset: Int, errorMessage: String, canBeAutoCorrected: Boolean) -> AutocorrectDecision,
    ) {
        node
            .takeIf { it.text == "listOf" }
            ?.nextSibling { it.elementType == VALUE_ARGUMENT_LIST }
            ?.children()
            ?.filterNot { it.elementType == LPAR || it.elementType == RPAR }
            ?.toList()
            ?.ifEmpty {
                emit(
                    node.startOffset,
                    "Use 'emptyList' instead of 'listOf' to create an empty collection",
                    true,
                ).ifAutocorrectAllowed {
                    node
                        .findChildByType(IDENTIFIER)
                        .safeAs<LeafElement>()
                }
            }
    }
}
```

```
class EmptyCollectionInitializationRule : CustomRule("empty-collection-initialization-rule") {
    override fun beforeVisitChildNodes(
        node: ASTNode,
        emit: (offset: Int, errorMessage: String, canBeAutoCorrected: Boolean) -> AutocorrectDecision,
    ) {
        node
            .takeIf { it.text == "listOf" }
            ?.nextSibling { it.elementType == VALUE_ARGUMENT_LIST }
            ?.children()
            ?.filterNot { it.elementType == LPAR || it.elementType == RPAR }
            ?.toList()
            ?.ifEmpty {
                emit(
                    node.startOffset,
                    "Use 'emptyList' instead of 'listOf' to create an empty collection",
                    true,
                ).ifAutocorrectAllowed {
                    node
                        .findChildByType(IDENTIFIER)
                        .safeAs<LeafElement>()
                        ?.rawReplaceWithText("emptyList")
                }
            }
    }
}
```

☑ Test succeeds

Assert violations for sets and maps

```
@Test
fun `Given a collection initialized with setOf() then replace with emptySet()`() {
    val code =
        """
        val x = setOf<String>()
        val y: Set<String> = setOf()
        val z = setOf("zzz")
        """.trimIndent()
    val formattedCode =
        """
        val x = emptySet<String>()
        val y: Set<String> = emptySet()
        val z = setOf("zzz")
        """.trimIndent()
    emptyCollectionInitializationRuleAssertThat(code)
        .hasLintViolations(
            LintViolation(1, 9, "Use 'emptySet' instead of 'setOf' to create an empty collection"),
            LintViolation(2, 22, "Use 'emptySet' instead of 'setOf' to create an empty collection"),
        ).isFormattedAs(formattedCode)
}
```

❌ Test failure

```kotlin
@Test
fun `Given a collection initialized with mapOf() then replace with emptyMap()`() {
    val code =
        """
        val x = mapOf<String, String>()
        val y: Map<String, String> = mapOf()
        val z = mapOf("zzz", "ZZZ")
        """.trimIndent()
    val formattedCode =
        """
        val x = emptyMap<String, String>()
        val y: Map<String, String> = emptyMap()
        val z = mapOf("zzz", "ZZZ")
        """.trimIndent()
    emptyCollectionInitializationRuleAssertThat(code)
        .hasLintViolations(
            LintViolation(1, 9, "Use 'emptyMap' instead of 'mapOf' to create an empty collection"),
            LintViolation(2, 30, "Use 'emptyMap' instead of 'mapOf' to create an empty collection"),
        ).isFormattedAs(formattedCode)
}
```

❌ Test failure

Autocorrect violations for sets and maps

```
class EmptyCollectionInitializationRule : CustomRule("empty-collection-initialization-rule") {
    override fun beforeVisitChildNodes(
        node: ASTNode,
        emit: (offset: Int, errorMessage: String, canBeAutoCorrected: Boolean) -> AutocorrectDecision,
    ) {
        node
            .takeIf { it.text == "listOf" }
            ?.nextSibling { it.elementType == VALUE_ARGUMENT_LIST }
            ?.children()
            ?.filterNot { it.elementType == LPAR || it.elementType == RPAR }
            ?.toList()
            ?.ifEmpty {
                emit(
                    node.startOffset,
                    "Use 'emptyList' instead of 'listOf' to create an empty collection",
                    true,
                ).ifAutocorrectAllowed {
                    node
                        .findChildByType(IDENTIFIER)
                        .safeAs<LeafElement>()
                        ?.rawReplaceWithText("emptyList")
                }
            }
    }
}
```

```kotlin
class EmptyCollectionInitializationRule : CustomRule("empty-collection-initialization-rule") {
    override fun beforeVisitChildNodes(
        node: ASTNode,
        emit: (Int, String, Boolean) -> AutocorrectDecision,
    ) {
        ...
    }

    private companion object {
        val COLLECTIONS_REFERENCES =
            mapOf(
                "listOf" to "emptyList",
                "setOf" to "emptySet",
                "mapOf" to "emptyMap",
            )
    }
}
```

```kotlin
class EmptyCollectionInitializationRule : CustomRule("empty-collection-initialization-rule") {
  override fun beforeVisitChildNodes(
    node: ASTNode,
    emit: (offset: Int, errorMessage: String, canBeAutoCorrected: Boolean) -> AutocorrectDecision,
  ) {
    node
      .takeIf { it.text in COLLECTIONS_REFERENCES.keys }
      ?.nextSibling { it.elementType == VALUE_ARGUMENT_LIST }
      ?.children()
      ?.filterNot { it.elementType == LPAR || it.elementType == RPAR }
      ?.toList()
      ?.ifEmpty {
        emit(
          node.startOffset,
          "Use 'emptyList' instead of 'listOf' to create an empty collection",
          true,
        ).ifAutocorrectAllowed {
          node
            .findChildByType(IDENTIFIER)
            .safeAs<LeafElement>()
            ?.rawReplaceWithText("emptyList")
        }
      }
  }
  ...
```

```
class EmptyCollectionInitializationRule : CustomRule("empty-collection-initialization-rule") {
    override fun beforeVisitChildNodes(
        node: ASTNode,
        emit: (offset: Int, errorMessage: String, canBeAutoCorrected: Boolean) -> AutocorrectDecision,
    ) {
        node
            .takeIf { it.text in COLLECTIONS_REFERENCES.keys }
            ?.nextSibling { it.elementType == VALUE_ARGUMENT_LIST }
            ?.children()
            ?.filterNot { it.elementType == LPAR || it.elementType == RPAR }
            ?.toList()
            ?.ifEmpty {
                val emptyCollectionReferenceId = COLLECTIONS_REFERENCES[node.text]!!
                emit(
                    node.startOffset,
                    "Use 'emptyList' instead of 'listOf' to create an empty collection",
                    true,
                ).ifAutocorrectAllowed {
                    node
                        .findChildByType(IDENTIFIER)
                        .safeAs<LeafElement>()
                        ?.rawReplaceWithText("emptyList")
                }
            }
    }
```

```
class EmptyCollectionInitializationRule : CustomRule("empty-collection-initialization-rule") {
    override fun beforeVisitChildNodes(
        node: ASTNode,
        emit: (Int, String, Boolean) -> AutocorrectDecision,
    ) {
        node
            .takeIf { it.text in COLLECTIONS_REFERENCES.keys }
            ?.nextSibling { it.elementType == VALUE_ARGUMENT_LIST }
            ?.children()
            ?.filterNot { it.elementType == LPAR || it.elementType == RPAR }
            ?.toList()
            ?.ifEmpty {
                val emptyCollectionReferenceId = COLLECTIONS_REFERENCES[node.text]!!
                emit(
                    node.startOffset,
                    "Use '$emptyCollectionReferenceId' instead of '${node.text}' to create an empty collection",
                    true,
                ).ifAutocorrectAllowed {
                    node
                        .findChildByType(IDENTIFIER)
                        .safeAs<LeafElement>()
                        ?.rawReplaceWithText(emptyCollectionReferenceId)
                }
            }
    }
```

3 Bootstrapping via RuleSetProvider

Run ruleset with lint/format with Ktlint-CLI

```
val x = emptyList<String>()
val y: List<String> = emptyList()
val z = listOf("zzz")
```

Run ruleset with lint/format with Ktlint-CLI

```
val x = emptyList<String>()
val y: List<String> = emptyList()
val z = listOf("zzz")
```

- Provide custom ruleset via command line parameter “--ruleset”

```
$ ktlint --ruleset=~/git/ktlint-ruleset-demo-kdd/build/libs/ktlint-ruleset-demo-kdd-1.0-SNAPSHOT.jar
```

Run ruleset with lint/format with Ktlint-CLI

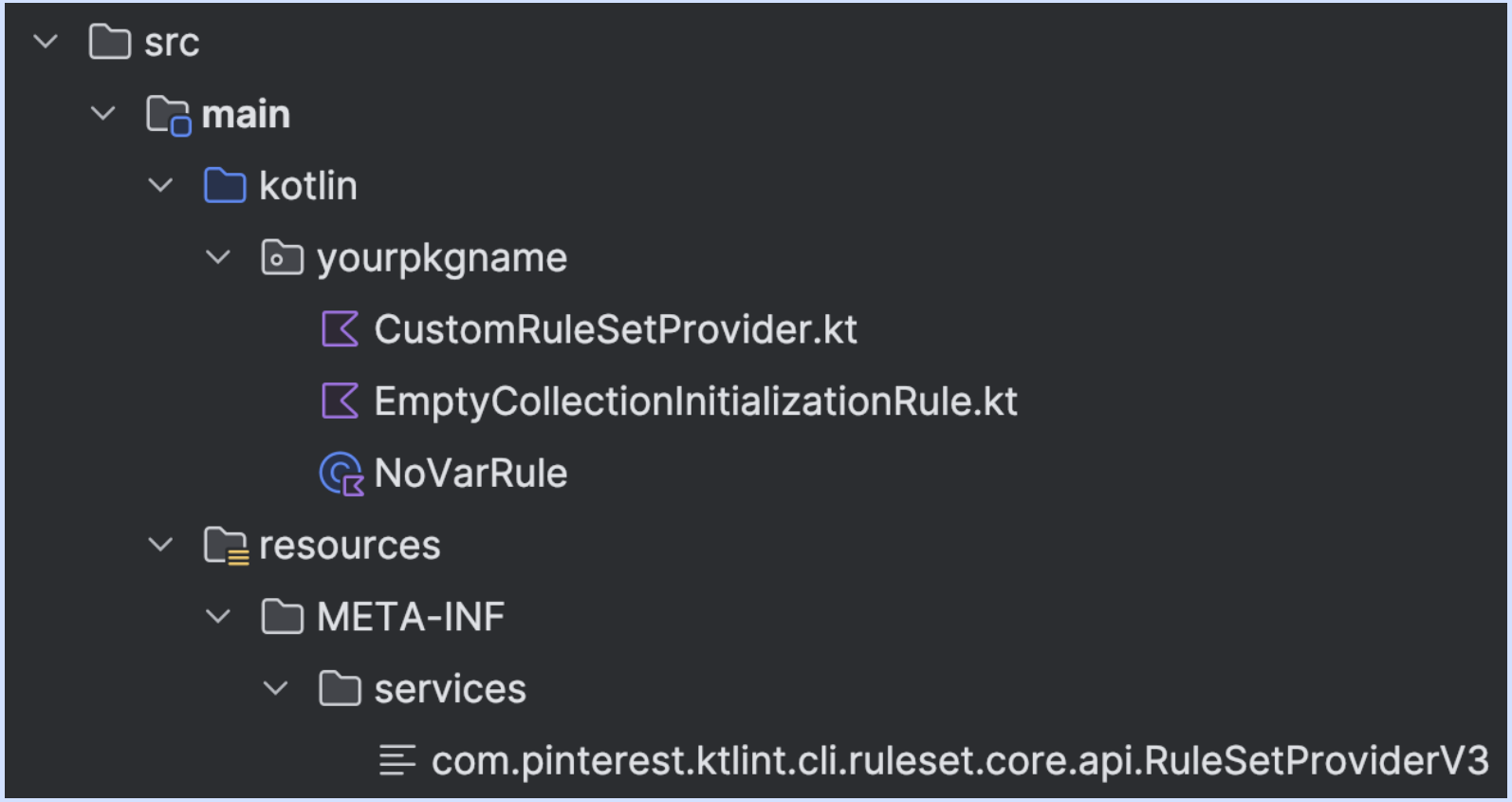
```
val x = emptyList<String>()
val y: List<String> = emptyList()
val z = listOf("zzz")
```

- Provide custom ruleset via command line parameter “--ruleset”

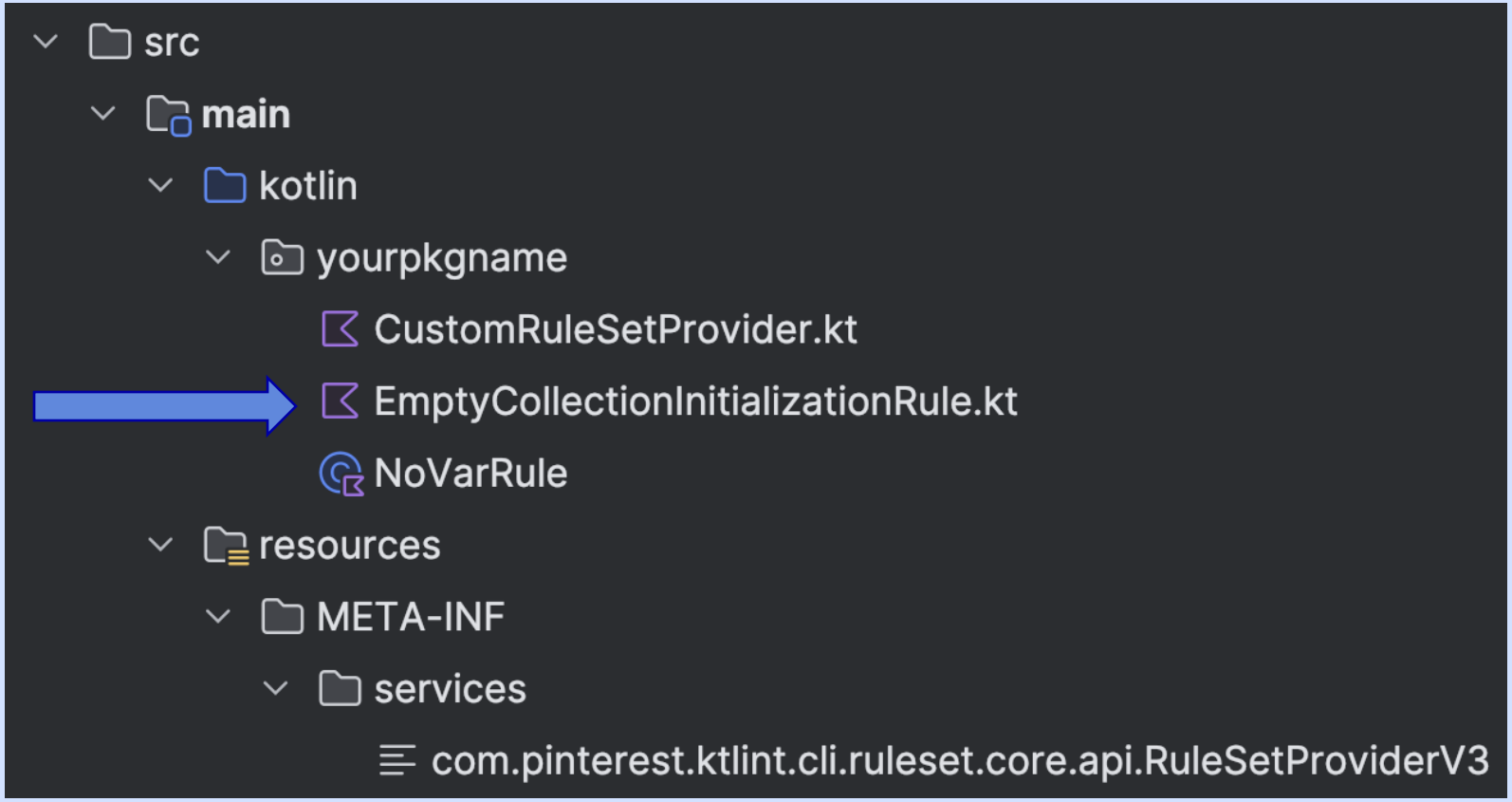
```
$ ktlint --ruleset=~/git/ktlint-ruleset-demo-kdd/build/libs/ktlint-ruleset-demo-kdd-1.0-SNAPSHOT.jar
13:06:21.553 [main] INFO com.pinterest.ktlint.cli.internal.KtlintCommandLine -- Enable default patterns
[**/*.kt, **/*.kts]
```

- No violations found as ruleset is not yet bootstrapped via the RuleSetProvider

Bootstrapping via RuleSetProvider

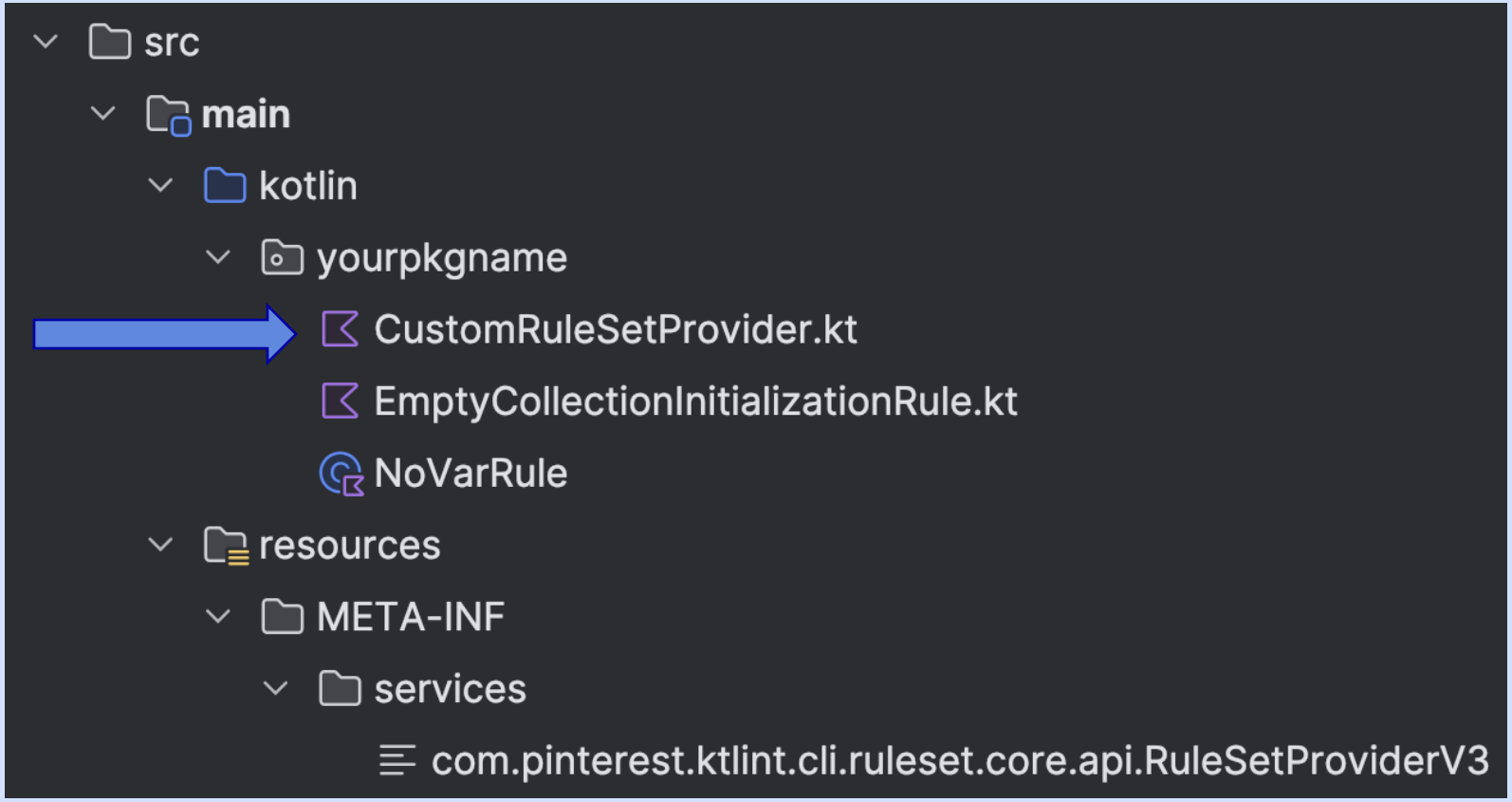


Bootstrapping via RuleSetProvider



1. Rule provides lint and formatting logic

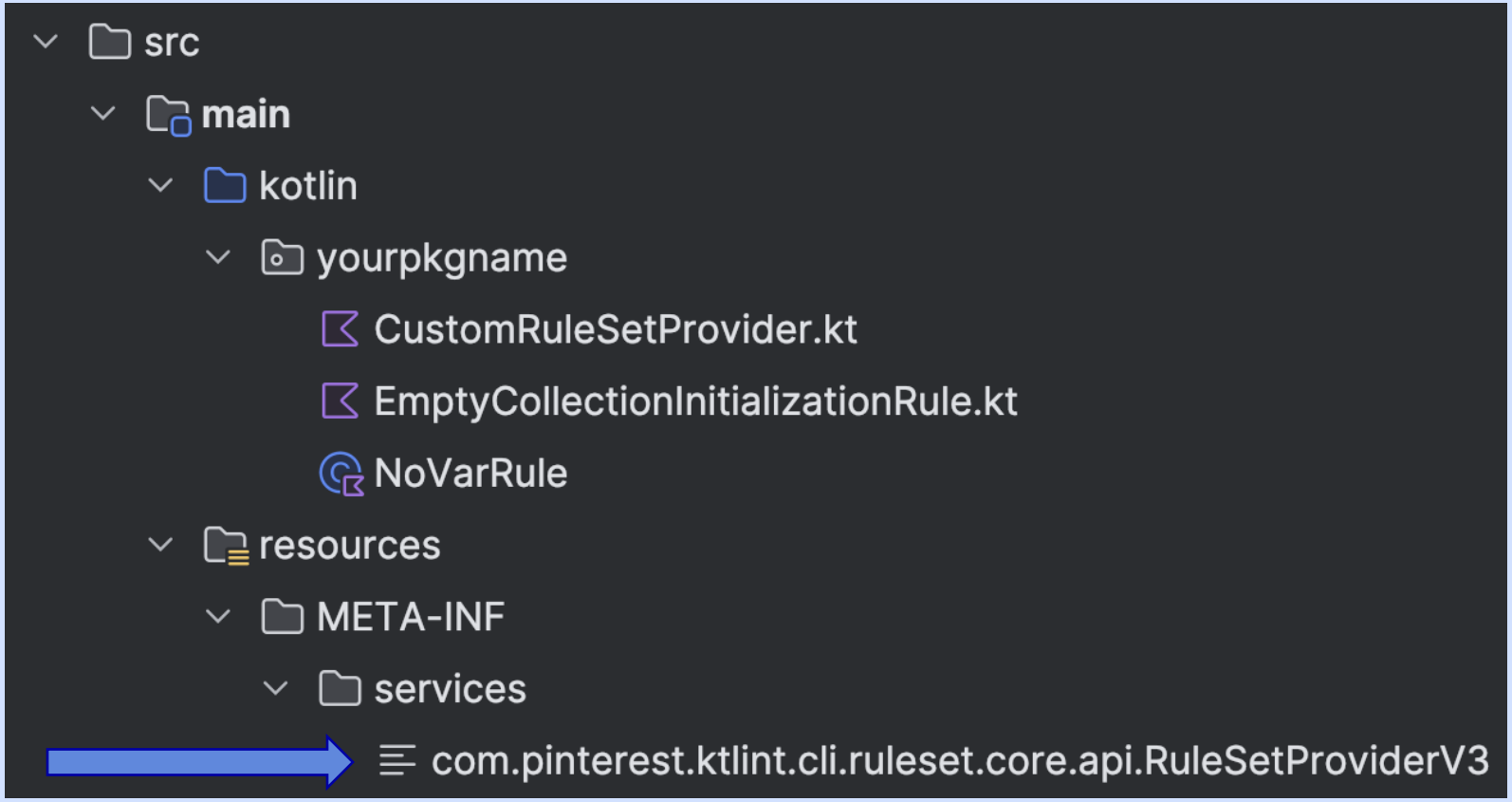
Bootstrapping via RuleSetProvider



1. Rule provides lint and formatting logic

2. RuleSetProvider provides new instances of the rule

Bootstrapping via RuleSetProvider

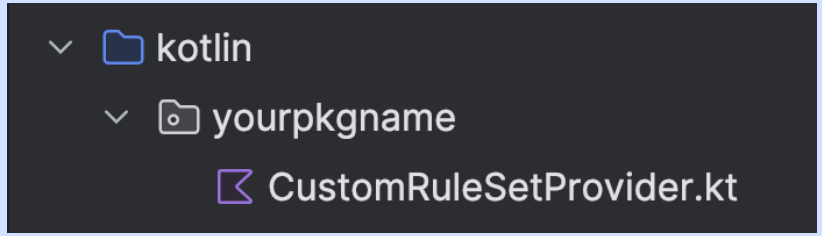


1. Rule provides lint and formatting logic
2. RuleSetProvider provides new instances of the rule
3. ServiceProvider allows Ktlint-CLI to find the RuleSetProvider in the custom ruleset

Service provider

- Naming:
  - Nested directory structure “META-INF/services”
  - File name must be exactly “com.pinterest.ktlint.cli.ruleset.core.api.RuleSetProviderV3”

- Content of the file is the fully qualified name of the RuleSetProvider



RuleSetProvider

```kotlin
package yourpkgname

import com.pinterest.ktlint.cli.ruleset.core.api.RuleSetProviderV3
import com.pinterest.ktlint.rule.engine.core.api.RuleProvider
import com.pinterest.ktlint.rule.engine.core.api.RuleSetId

internal val CUSTOM_RULE_SET_ID = "custom-rule-set-id"

class CustomRuleSetProvider : RuleSetProviderV3(RuleSetId(CUSTOM_RULE_SET_ID)) {
    override fun getRuleProviders(): Set<RuleProvider> =
        setOf(
            RuleProvider { NoVarRule() },
        )
}
```

RuleSetProvider

```
package yourpkgname

import com.pinterest.ktlint.cli.ruleset.core.api.RuleSetProviderV3
import com.pinterest.ktlint.rule.engine.core.api.RuleProvider
import com.pinterest.ktlint.rule.engine.core.api.RuleSetId

internal val CUSTOM_RULE_SET_ID = "custom-rule-set-id"

class CustomRuleSetProvider : RuleSetProviderV3(RuleSetId(CUSTOM_RULE_SET_ID)) {
    override fun getRuleProviders(): Set<RuleProvider> =
        setOf(
            RuleProvider { NoVarRule() },
            RuleProvider { EmptyCollectionInitializationRule() },
        )
}
```

Run ruleset with lint/format with Ktlint-CLI

```
val x = emptyList<String>()
val y: List<String> = emptyList()
val z = listOf("zzz")
```

- Rule is provided via the RuleSetProvider
- Ruleset is correctly provided on command line

```
$ ktlint --ruleset=~/git/ktlint-ruleset-demo-kdd/build/libs/ktlint-ruleset-demo-kdd-1.0-SNAPSHOT.jar –relative
```

Run ruleset with lint/format with Ktlint-CLI

```
val x = emptyList<String>()
val y: List<String> = emptyList()
val z = listOf("zzz")
```

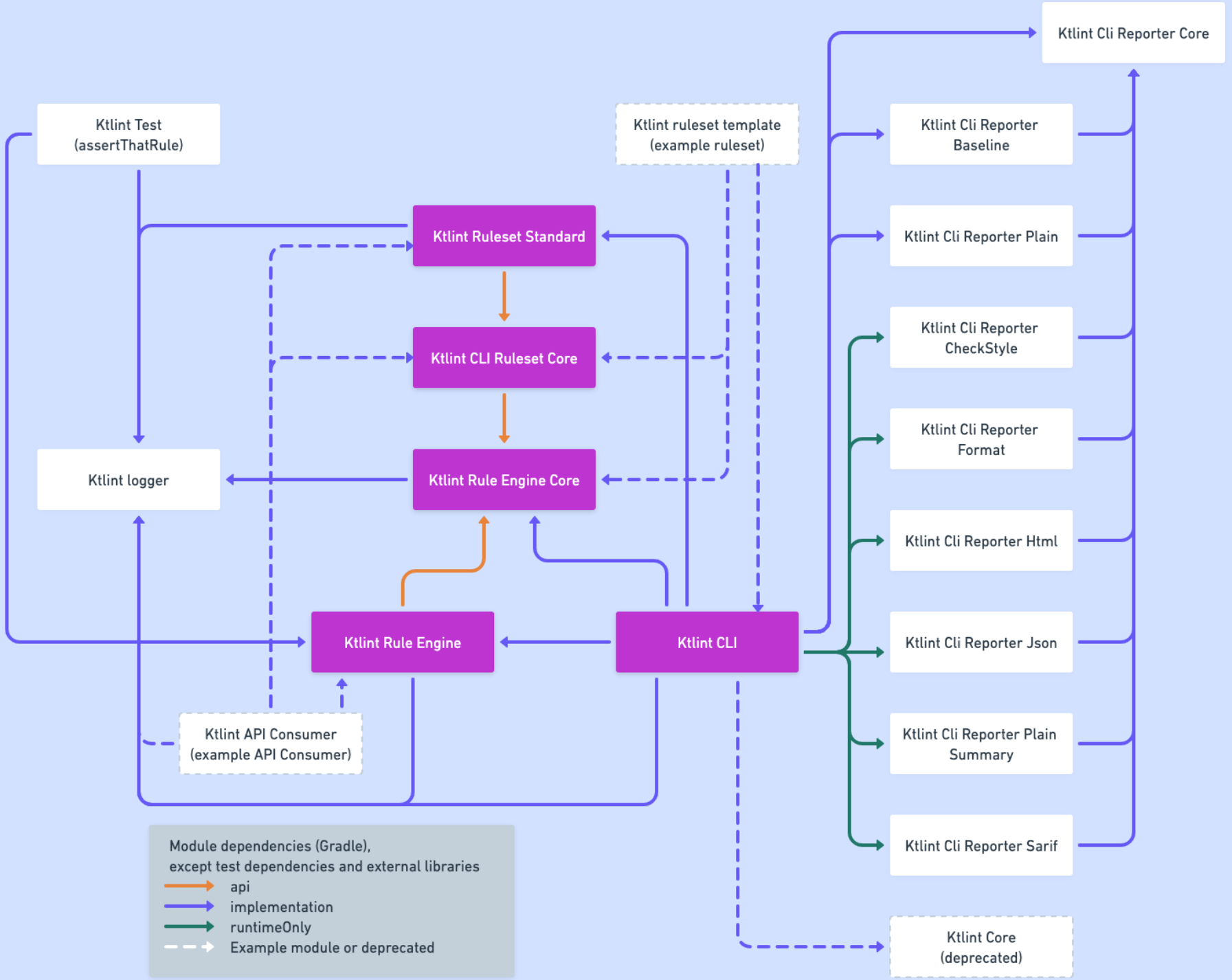
- Rule is provided via the RuleSetProvider
- Ruleset is correctly provided on command line

```
$ ktlint --ruleset=~/git/ktlint-ruleset-demo-kdd/build/libs/ktlint-ruleset-demo-kdd-1.0-SNAPSHOT.jar --relative
13:17:29.057 [main] INFO com.pinterest.ktlint.cli.internal.KtlintCommandLine -- Enable default patterns [**/*.kt, **/*.kts]
src/main/kotlin/Foo.kt:1:9: Use 'emptyList' instead of 'listOf' to create an empty collection (custom-rule-set-id:empty-collection-initialization-rule)
src/main/kotlin/Foo.kt:2:23: Use 'emptyList' instead of 'listOf' to create an empty collection (custom-rule-set-id:empty-collection-initialization-rule)
13:17:29.331 [main] WARN com.pinterest.ktlint.cli.internal.KtlintCommandLine -- Lint has found errors than can be autocorrected using 'ktlint --format'

Summary error count (descending) by rule:
  custom-rule-set-id:empty-collection-initialization-rule: 2
```

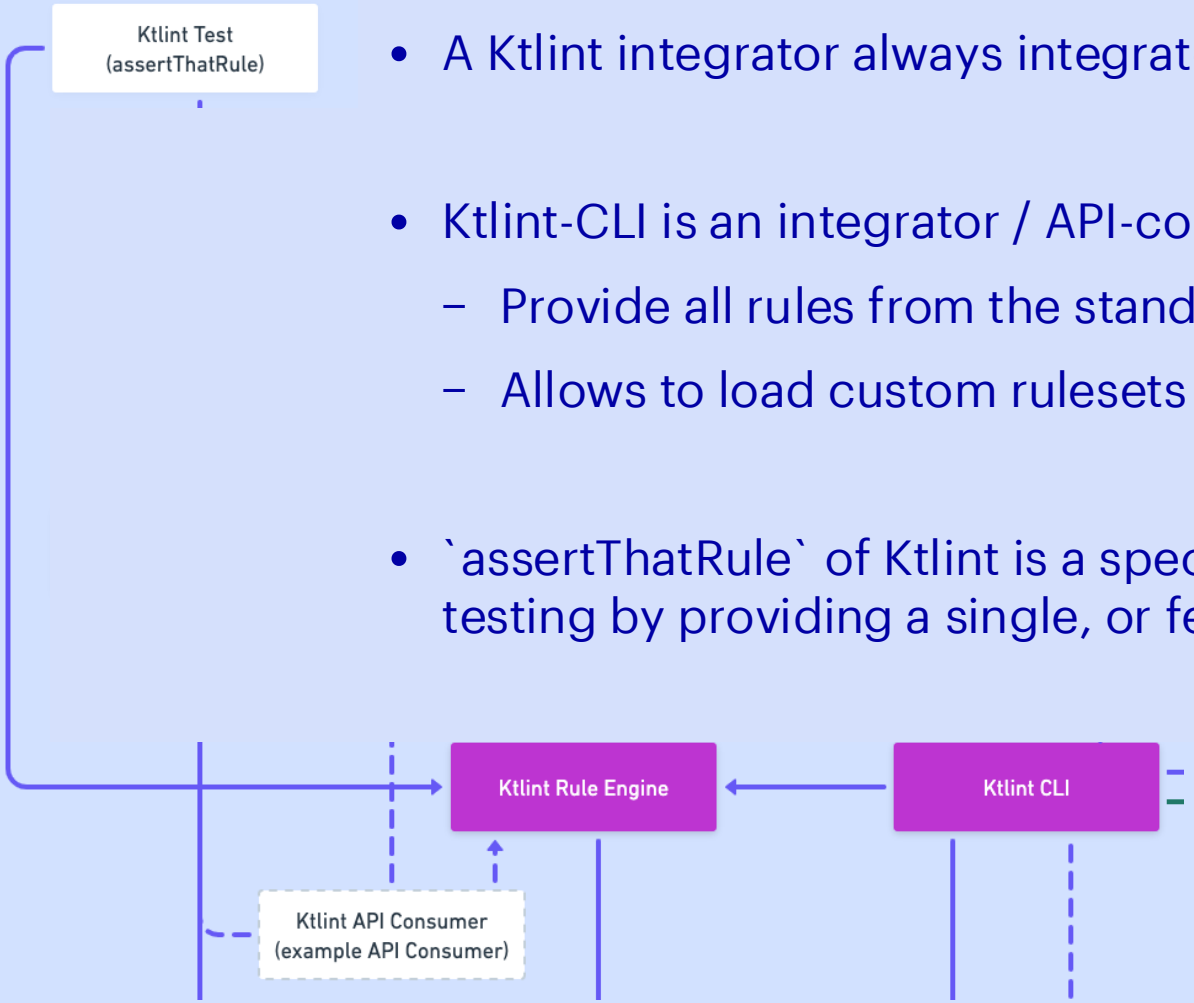
Hey wait a moment ...

How did the unit tests work while the rule was not yet added to RuleSetProvider???

Providing a rule to the KtlintRuleEngine


Ktlint Cli Reporter Core
Ktlint Test
(assertThatRule)
Ktlint ruleset template
(example ruleset)
Ktlint Cli Reporter
Baseline
Ktlint Ruleset Standard
Ktlint Cli Reporter Plain
Ktlint CLI Ruleset Core
Ktlint Cli Reporter
CheckStyle
Ktlint logger
Ktlint Rule Engine Core
Ktlint Cli Reporter
Format
Ktlint Cli Reporter Html
Ktlint Rule Engine
Ktlint CLI
Ktlint Cli Reporter Json
Ktlint API Consumer
(example API Consumer)
Ktlint Cli Reporter Plain
Summary
Ktlint Cli Reporter Sarif
Module dependencies (Gradle),
except test dependencies and external libraries
api
implementation
runtimeOnly
Example module or deprecated
Ktlint Core
(deprecated)

Integrating with Ktlint



- A Ktlint integrator always integrates with the KtlintRuleEngine only
- Ktlint-CLI is an integrator / API-consumer like any other Ktlint integrator
  - Provide all rules from the standard Ktlint ruleset
  - Allows to load custom rulesets
- `assertThatRule` of Ktlint is a special integrator to assist with unit testing by providing a single, or few rules

KtlintRuleEngine API – Providing the rules

- Only provide the rules that you want to run, including
  - Rules from Ktlint Standard Rules
  - Custom rules

```
public class KtLintRuleEngine(
    /**
     * The set of [RuleProvider]s to be invoked by the [KtLintRuleEngine]...
     */
    public val ruleProviders: Set<RuleProvider> = emptySet(),
    ...
) {
```

KtlintRuleEngine – Invoking lint

- Provide the code snippet, or entire file
- Provide callback (optional) to be invoked on each violation found

```
/**
 * Check the [code] for lint errors...
 */
public fun lint(
    code: Code,
    callback: (LintError) -> Unit = { },
) {
    codeFormatter.format(
        code = code,
        autocorrectHandler = NoneAutocorrectHandler,
        callback = { lintError, _ -> callback(lintError) },
        maxFormatRunsPerFile = 1,
    )
}
```

KtlintRuleEngine – Invoking format

- Provide the code snippet, or entire file
- Provide callback which decides whether to autocorrect a violation

```
/**
 * Formats style violations in [code]...
 */
public fun format(
    code: Code,
    rerunAfterAutocorrect: Boolean = true,
    defaultAutocorrect: Boolean = true,
    callback: (LintError) -> AutocorrectDecision,
): String =
    codeFormatter.format(
        code = code,
        autocorrectHandler = LintErrorAutocorrectHandler(defaultAutocorrect, callback),
        maxFormatRunsPerFile = ... ,
    )
```

AssertThatRule

```
@Test
fun `Given a collection initialized with listOf() then replace with emptyList()`() {
    ...
    emptyCollectionInitializationRuleAssertThat(code)
        .hasLintViolations(
            LintViolation(1, 9, "Use 'emptyList' instead of 'listOf' to create an empty collection"),
            LintViolation(2, 23, "Use 'emptyList' instead of 'listOf' to create an empty collection"),
        ).isFormattedAs(formattedCode)
}
```

AssertThatRule

```
public fun hasLintViolations(vararg expectedErrors: LintViolation):
KtLintAssertThatAssertable {
    check(expectedErrors.isNotEmpty())

    val actualLintViolationFields =
        lint()
            .filterCurrentRuleOnly()
            .toLintViolationsFields()
    assertThat(actualLintViolationFields)
        .describedAs("Lint errors which can be automatically corrected")
        .containsExactlyInAnyOrder(*expectedErrors.toLintViolationsFields())
    return this
}

private fun lint(): Set<LintError> {
    val lintErrors = mutableSetOf<LintError>()
    createKtLintRuleEngine().lint(code) { lintError -> lintErrors.add(lintError) }
    return lintErrors
}
```

Thanks

ktlint