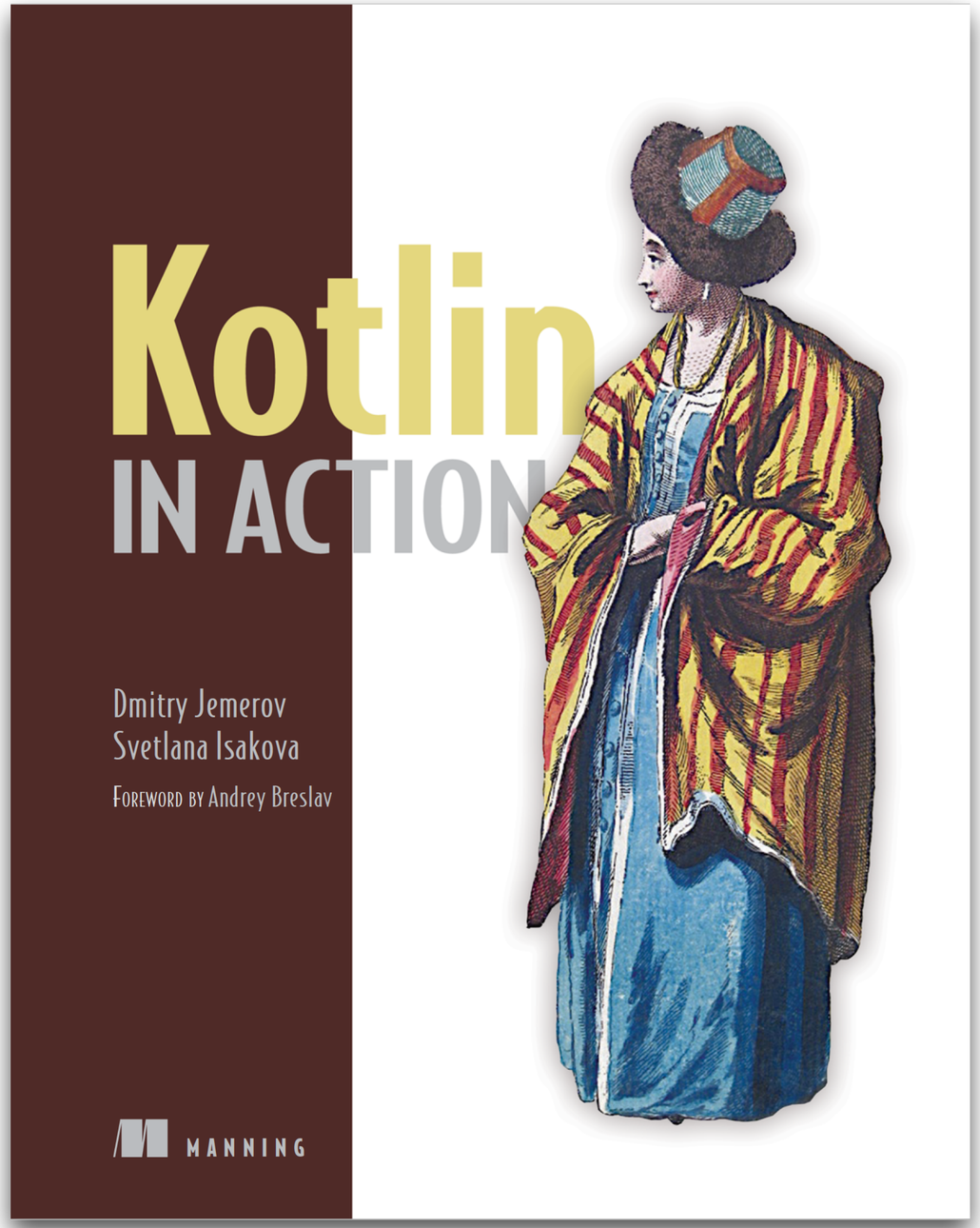
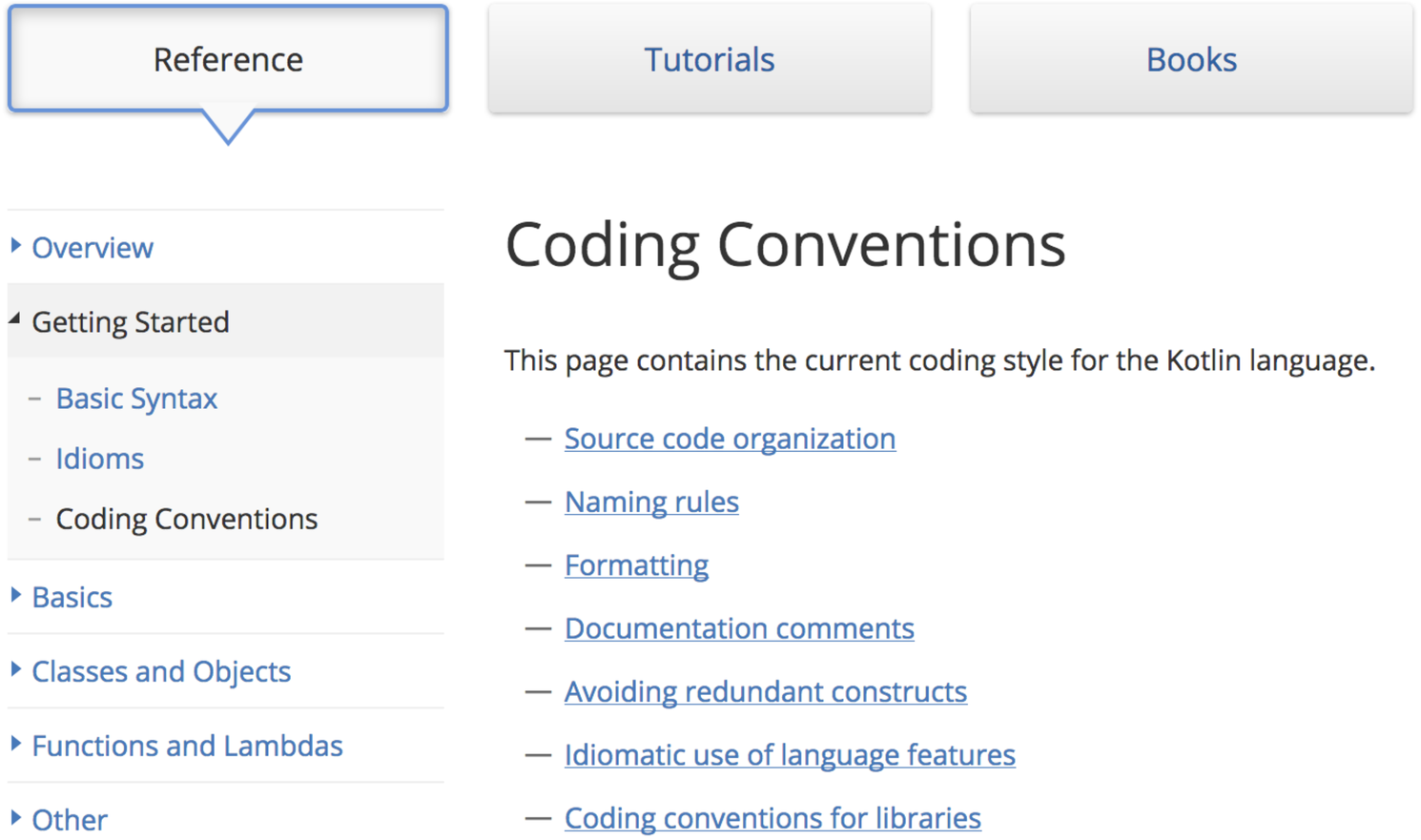
Идиоматичный Kotlin: От форматирования до DSL

Дмитрий Жемеров <yole@jetbrains.com>


Kotlin
IN ACTION
Dmitry Jemerov
Svetlana Isakova
Foreword by Andrey Breslav
MANNING


Kotlin
в действии
Дмитрий Жемеров
Светлана Исакова
ДМК
MANNING

http://kotlinlang.org/docs/reference/coding-conventions.html

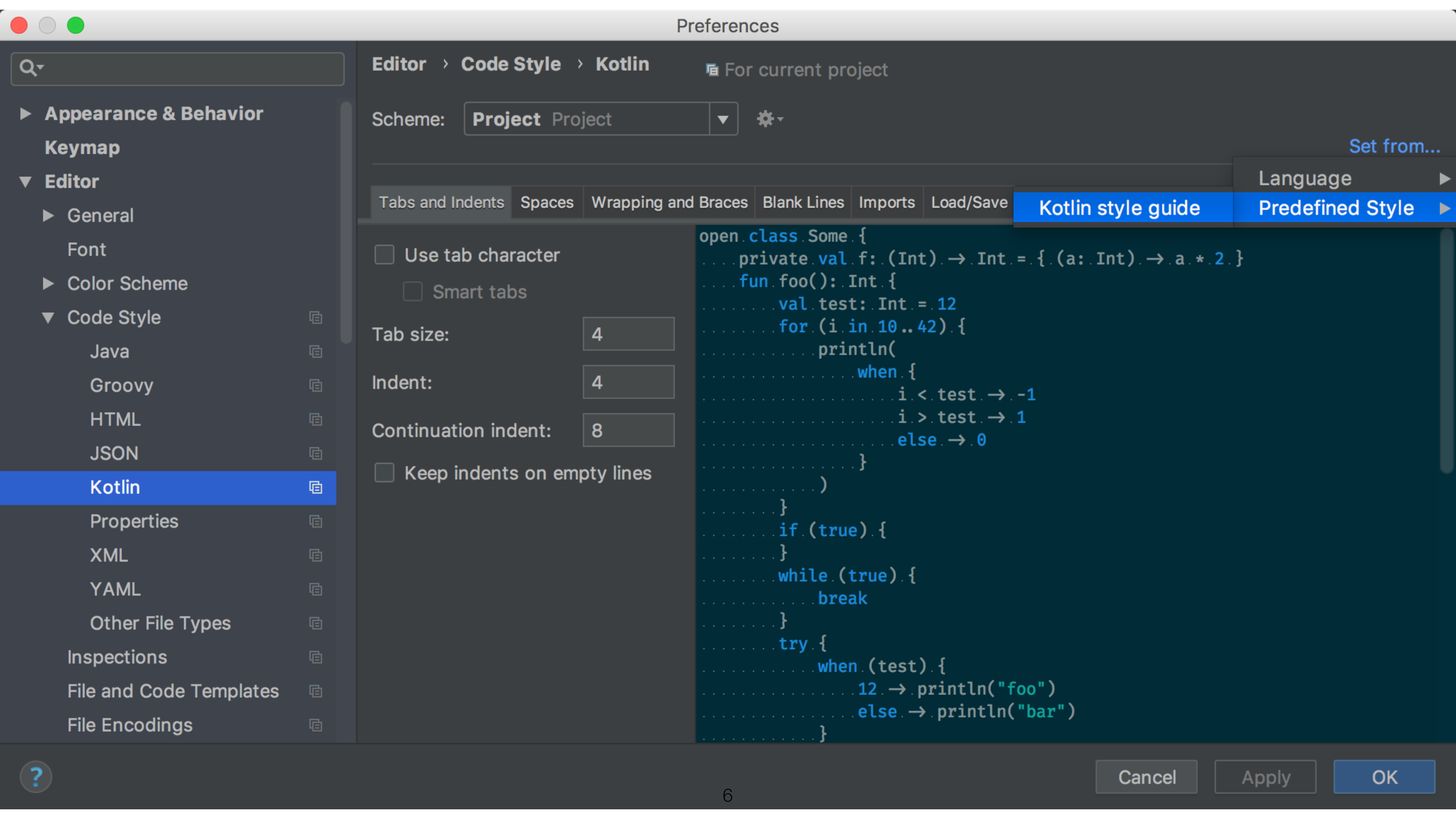


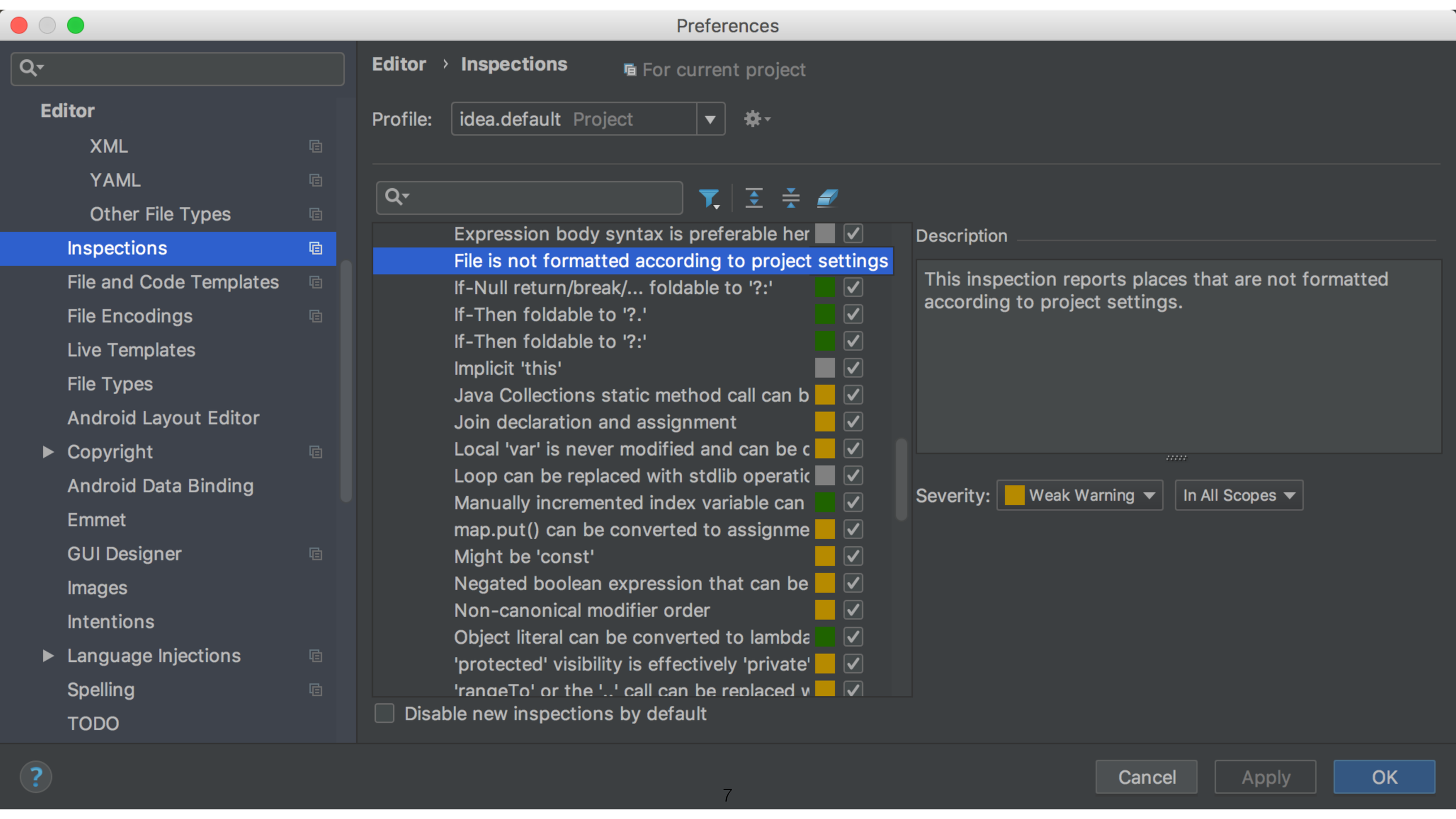
Idiomatic:

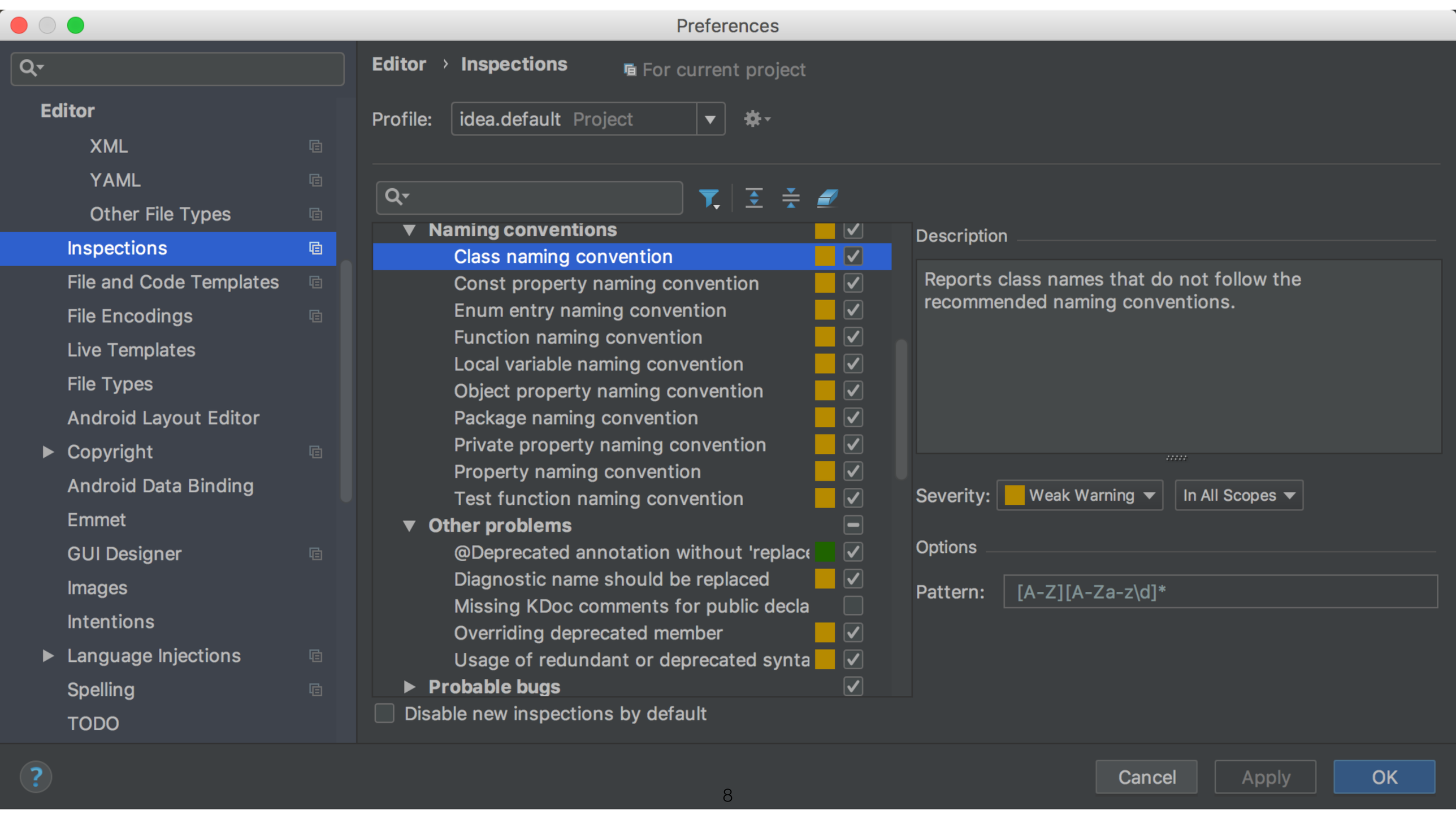
using, containing, or denoting expressions that are natural to a native speaker.

- **Соответствующий общепринятому стилю**
- **Эффективно использующий возможности языка**

Основы


Preferences
Editor › Code Style › Kotlin
For current project
Scheme: Project Project
Set from...
Language
Predefined Style
Kotlin style guide
Tabs and Indents
Spaces
Wrapping and Braces
Blank Lines
Imports
Load/Save
Use tab character
Smart tabs
Tab size: 4
Indent: 4
Continuation indent: 8
Keep indents on empty lines
Appearance & Behavior
Keymap
Editor
General
Font
Color Scheme
Code Style
Java
Groovy
HTML
JSON
Kotlin
Properties
XML
YAML
Other File Types
Inspections
File and Code Templates
File Encodings
open class Some {
private val f: (Int) → Int = { (a: Int) → a * 2 }
fun foo(): Int {
val test: Int = 12
for (i in 10..42) {
println(
when {
i < test → -1
i > test → 1
else → 0
}
)
}
if (true) {
}
while (true) {
break
}
try {
when (test) {
12 → println("foo")
else → println("bar")
}
Cancel
Apply
OK


Preferences
Editor
XML
YAML
Other File Types
Inspections
File and Code Templates
File Encodings
Live Templates
File Types
Android Layout Editor
Copyright
Android Data Binding
Emmet
GUI Designer
Images
Intentions
Language Injections
Spelling
TODO
Editor › Inspections
For current project
Profile: idea.default Project
Expression body syntax is preferable her
File is not formatted according to project settings
If-Null return/break/... foldable to '?:'
If-Then foldable to '?.'
If-Then foldable to '?:'
Implicit 'this'
Java Collections static method call can b
Join declaration and assignment
Local 'var' is never modified and can be c
Loop can be replaced with stdlib operatic
Manually incremented index variable can
map.put() can be converted to assignme
Might be 'const'
Negated boolean expression that can be
Non-canonical modifier order
Object literal can be converted to lambda
'protected' visibility is effectively 'private'
'rangeTo' or the '..' call can be replaced w
Disable new inspections by default
Description
This inspection reports places that are not formatted according to project settings.
Severity: Weak Warning
In All Scopes
Cancel
Apply
OK


Preferences
Editor
XML
YAML
Other File Types
Inspections
File and Code Templates
File Encodings
Live Templates
File Types
Android Layout Editor
Copyright
Android Data Binding
Emmet
GUI Designer
Images
Intentions
Language Injections
Spelling
TODO
Editor › Inspections
For current project
Profile: idea.default Project
Naming conventions
Class naming convention
Const property naming convention
Enum entry naming convention
Function naming convention
Local variable naming convention
Object property naming convention
Package naming convention
Private property naming convention
Property naming convention
Test function naming convention
Other problems
@Deprecated annotation without 'replace
Diagnostic name should be replaced
Missing KDoc comments for public decla
Overriding deprecated member
Usage of redundant or deprecated synta
Probable bugs
Disable new inspections by default
Description
Reports class names that do not follow the recommended naming conventions.
Severity: Weak Warning
In All Scopes
Options
Pattern: [A-Z][A-Za-z\d]*
Cancel
Apply
OK

Выражения

‘when’ как выражение

```
fun parseNum(number: String): Int? {
    when (number) {
        "one" -> return 1
        "two" -> return 2
        else -> return null
    }
}
```

```
fun parseNum(number: String) =
    when (number) {
        "one" -> 1
        "two" -> 2
        else -> null
    }
```

'when' и sealed классы

```
abstract class Result

class Success : Result()
class Failure(val message: String)
    : Result()

fun handleResult(result: Result) =
  when (result) {
    is Success -> "OK!"
    is Failure ->
      "Failed: ${result.message}"
    else ->
      throw IllegalArgumentException()
  }
```

```
sealed class Result

class Success : Result()
class Failure(val message: String)
    : Result()

fun handleResult(result: Result) =
  when (result) {
    is Success -> "OK!"
    is Failure ->
      "Failed: ${result.message}"
  }
```

Элвис

```
fun foo(name: String?) {
    val s = if (name != null)
        name
    else
        "?"
}
```

```
fun foo(name: String?) {
    val s = name ?: "?"
}
```

Элвис перед return

```
class Person(
    val name: String?,
    val age: Int?
)
```

```
fun processPerson(p: Person) {
  val age = p.age
  if (age == null) return
  …
}
```

```
fun processPerson(p: Person) {
  val age: Int = p.age ?: return
  …
}
```

Элвис перед throw

```
class Person(
    val name: String?,
    val age: Int?
)
```

```
fun processPerson(p: Person) {
  val name = p.name
  if (name == null)
    throw IllegalArgumentException(
        "Name required”)
  …
}
```

```
fun processPerson(p: Person) {
  val name: String = p.name ?:
    throw IllegalArgumentException(
      "Name required”)
  …
}
```

Проверка параметра на null через ?.let

```
class Person(
    val name: String,
    val address: String?
)
```

```
fun printAddress(person: Person) {
  if (person.address != null) {
    println(person.address)
 }
}
```

```
fun printAddress(person: Person) {
    person.address?.let {
        println(it)
    }
}
```

```
public inline fun <T, R> T.let(block: (T) -> R): R {
    return block(this)
}
```

Инициализация объектов через apply

```
class Label {
    var text: String = ""
    var tooltip: String = ""
}
```

```
fun createLabel(): Label {
    val label = Label()
    label.text = "Click here"
    label.tooltip = "Help"
    return label
}
```

```
fun createLabel() =
    Label().apply {
        text = "Click here"
        tooltip = "Help"
    }
```

Строки с тремя кавычками

```
val s = "Greetings!\n" +
        "Welcome to our system!\n" +
        "Enter your request:"
```

```
val s = """Greetings!
           Welcome to our system!
           Enter your request:"""
    .trimIndent()
```

Строки с тремя кавычками

```
val s = "Greetings!\n" +
        "  Welcome to our system!\n" +
        "Enter your request:"
```

```
val s =
    """Greetings!
      |  Welcome to our system!
      |Enter your request:"""
    .trimMargin()
```

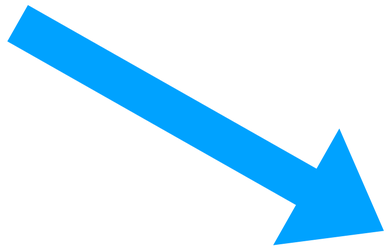
Диапазон вместо двух сравнений

```
fun isLatinUppercase(c: Char) =
    c >= 'A' && c <= 'Z'
```

```
fun isLatinUppercase(c: Char) =
    c in 'A'..'Z'
```

Диапазон вместо двух сравнений

```
fun isLatinUppercase(c: Char) =
    c in 'A'..'Z'
```



```
boolean var10000;
if ('A' <= c) {
    if ('Z' >= c) {
        var10000 = true;
        return var10000;
    }
}

var10000 = false;
return var10000;
```

until в циклах

```
fun main(args: Array<String>) {
  for (i in 0..args.size-1) {
    println("$i: ${args[i]}")
  }
}
```

```
fun main(args: Array<String>) {
  for (i in 0 until args.size) {
    println("$i: ${args[i]}")
  }
}
```

withIndex в циклах

```
fun main(args: Array<String>) {
  for (i in 0 until args.size) {
    println("$i: ${args[i]}")
  }
}
```

```
fun main(args: Array<String>) {
  for ((i, arg) in args.withIndex()) {
    println("$i: $arg")
  }
}
```

Классы и функции

Используйте значения параметров по умолчанию вместо перегрузок

```
class Phonebook {
  fun print() {
    print(",")
  }

  fun print(separator: String) {
  }
}

fun main(args: Array<String>) {
  Phonebook().print("|")
}
```

```
class Phonebook {
  fun print(
      separator: String = ","
  ) {
  }
}

fun main(args: Array<String>) {
  Phonebook().print(
      separator = "|"
  )
}
```

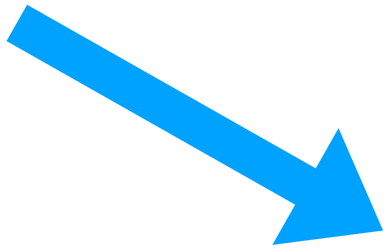
Используйте lateinit для свойств, которые нельзя инициализировать в конструкторе

```
class MyTest {
  class State(val data: String)

  var state: State? = null

  @Before fun setup() {
    state = State("abc")
  }

  @Test fun foo() {
    Assert.assertEquals(
        "abc", state!!.data)
  }
}
```

```
class MyTest {
  class State(val data: String)

  lateinit var state: State

  @Before fun setup() {
    state = State("abc")
  }

  @Test fun foo() {
    Assert.assertEquals(
        "abc", state.data)
  }
}
```

Используйте lateinit для свойств, которые нельзя инициализировать в конструкторе

```
class MyTest {
  @Test fun foo() {
    Assert.assertEquals(
        "abc", state.data)
  }
}
```



```
public final void foo() {
   MyTest.State var10001 = this.state;
   if (this.state == null) {
      Intrinsics.throwUninitializedPropertyAccessException("state");
   }

   Assert.assertEquals("abc", var10001.getData());
}
```

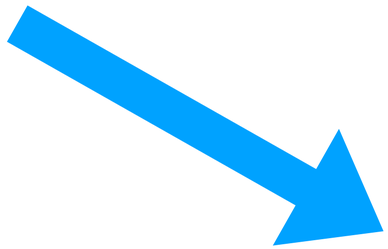
Возвращайте несколько значений через data class

```
fun namedNum(): Pair<Int, String> =
    1 to "Player One"

fun main(args: Array<String>) {
  val pair = namedNum()
  val number = pair.first
  val name = pair.second
}
```

```
data class GameResult(
    val rank: Int,
    val name: String
)

fun namedNum() =
    GameResult(1, "Player One")

fun main(args: Array<String>) {
  val (number, name) =
      namedNum()
}
```

Возвращайте несколько значений через data class

```
 val (number, name) =
     namedNum()
}
```



```
GameResult var3 = namedNum();
int var1 = var3.component1();
String name = var3.component2();
```

Используйте мультидекларации в циклах

```
fun printMap(m: Map<String, String>) {
  for (e in m.entries) {
    println("${e.key} -> ${e.value}")
  }
}
```

```
fun printMap(m: Map<String, String>) {
  for ((key, value) in m) {
    println("$key -> $value")
  }
}
```

Используйте мультидекларации для списков

```
data class NameExt(
    val name: String,
    val ext: String?
)
```

```
fun splitNameExt(fn: String): NameExt {
  if ('.' in fn) {
    val parts = fn.split('.', limit = 2)
    return NameExt(parts[0], parts[1])
  }
  return NameExt(fn, null)
}
```

```
fun splitNameExt(fn: String): NameExt {
   if ('.' in fn) {
      val (name, ext) =
          fn.split('.', limit = 2)
      return NameExt(name, ext)
   }
   return NameExt(fn, null)
}
```

Используйте мультидекларации для списков

```
fun splitNameExt(fn: String): NameExt {
    if ('.' in fn) {
        val (name, ext) =
            fn.split('.', limit = 2)
        return NameExt(name, ext)
    }
    return NameExt(fn, null)
}
```

```
/**
 * Returns 1st *element* from the collection.
 */
@kotlin.internal.InlineOnly
public inline operator fun <T> List<T>.component1(): T {
    return get(0)
}
```

Стандартная библиотека

require() для проверки аргументов

```
class Person(
    val name: String?,
    val age: Int
)
```

```
fun processPerson(p: Person) {
  if (p.age < 18) {
    throw IllegalArgumentException(
        "Adult required"
      )
  }
}
```

```
fun processPerson(p: Person) {
  require(p.age >= 18) {
    "Adult required"
  }
}
```

‘filterIsInstance’ фильтрует список по типу объекта

```
fun findStrings(objs: List<Any>) =
    objs.filter { it is String }
```

```
fun findStrings(objs: List<Any>) =
  objs.filterIsInstance<String>()
```

возвращает **List<Any>**

возвращает **List<String>**

‘mapNotNull’ применяет функцию и выбирает не-null возвращаемые значения

```
data class Result(
    val data: Any?,
    val error: String?
)
```

```
fun listErrors(
    results: List<Result>
) =
    results
        .map { it.error }
        .filterNotNull()
```

```
fun listErrors(
   results: List<Result>
) =
   results.mapNotNull {
      it.error
   }
```

‘compareBy’ сравнивает по нескольким ключам

```
class Person(
    val name: String,
    val age: Int
)
```

```
fun sortPersons(persons: List<Person>) =
    persons.sortedWith(
        Comparator<Person> { p1, p2 ->
            val rc =
                p1.name.compareTo(p2.name)

            if (rc != 0)
                rc
            else
                p1.age - p2.age
        })
```

```
fun sortPersons(persons: List<Person>) =
    persons.sortedWith(
        compareBy(Person::name,
                  Person::age))
```

‘groupBy’ группирует элементы в коллекции

```
class Request(
    val url: String,
    val remoteIP: String,
    val timestamp: Long
)
```

```
fun analyzeLog(log: List<Request>) {
  val map = mutableMapOf<String,
     MutableList<Request>>()

  for (request in log) {
    map.getOrPut(request.url)
           { mutableListOf() }
        .add(request)
  }
}
```

```
fun analyzeLog(log: List<Request>) {
  val map = log.groupBy(Request::url)
}
```

Методы String для разбора строк

```
data class PathParts(
    val dir: String,
    val name: String
)
```

```
val pattern = Regex("(.+)/([^/]*)")

fun splitPath(path: String): PathParts {
  val match = pattern.matchEntire(path)
        ?: return PathParts("", path)

  return PathParts(match.groupValues[1],
      match.groupValues[2])
}
```

```
fun splitPath(path: String) =
  PathParts(
    path.substringBeforeLast('/', ""),
    path.substringAfterLast('/')
  )
```

Доменно-ориентированные языки (DSL)

Что отличает DSL

- Отсутствие синтаксического шума
- Ограничение набора языковых возможностей
- Декларативный подход (обычно)

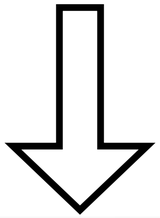
Внутренние DSL

- Способ организации кода на языке программирования общего назначения
- Не требует написания парсера, поддержки в IDE и т.д.
- I know it’s a DSL when I see it?

Язык => грамматика

Грамматика => структура

```
project.dependencies.add("compile", "junit:junit:4.12")
project.dependencies.add("compile", "com.google.inject:guice:4.1.0")
```



```
dependencies {
    compile("junit:junit:4.12")
    compile("com.google.inject:guice:4.1.0")
}
```

Лямбды с получателем

Лямбды с получателем: использование

```
val s = buildString { sb ->
    sb.append("Hello, ")
    sb.append("World!")
}
println(s)
```

```
val s = buildString {
    this.append("Hello, ")
    this.append("World!")
}
println(s)
```

Лямбды с получателем: использование

```
val s = buildString { sb ->
    sb.append("Hello, ")
    sb.append("World!")
}
println(s)
```

```
val s = buildString {
    append("Hello, ")
    append("World!")
}
println(s)
```

Лямбды с получателем: использование

```
val s = buildString { sb ->
    sb.append("Hello, ")
    sb.append("World!")
}
println(s)
```

```
val s = buildString {
    append("Hello, ")
    append("World!")
}
println(s)
```

this: StringBuilder

Лямбды с получателем: объявление

```
fun buildString(
    action: (StringBuilder) -> Unit
): String {
    val sb = StringBuilder()
    action(sb)
    return sb.toString()
}
```

```
fun buildString(
    action: StringBuilder.() -> Unit
): String {
    val sb = StringBuilder()
    sb.action()
    return sb.toString()
}
```

Изменение контекста = структура

```
val sb = StringBuilder()
sb.append("Hello, ")
sb.append("World!")
return sb.toString()
```

```
return buildString {
    append("Hello, ")
    append("World!")
}
```

Gradle DSL: изменение контекста

```
dependencies {
    compile("junit:junit:4.12")
    compile("com.google.inject:guice:4.1.0")
}

configure<JavaPluginConvention> {
    sourceCompatibility = JavaVersion.VERSION_1_8
}
tasks.withType<KotlinCompile> {
    kotlinOptions.jvmTarget = "1.8"
}
```

this: DependencyHandlerScope

this: JavaPluginConvention

this: KotlinCompile

DSL для HTML

```
fun createTable() =
    table {  this: TABLE
        tr {  this: TR
            td {  this: TD
                // content goes here
            }
        }
    }
```

DSL для HTML: циклы

```
fun createTable(items: List<Item>) =
    table {
        for (item in items) {
            tr {
                td {
                    // content goes here
                }
            }
        }
    }
```

DSL для HTML: абстракции

```
fun createTable(items: List<Item>) =
    table {
        for (item in items) {
            renderItem(item)
        }
    }

fun TABLE.renderItem(item: Item) =
    tr {
        td {
            // content goes here
        }
    }
```

DSL для HTML: реализация

```
open class Tag(val name: String) {
    private val children = mutableListOf<Tag>()

    protected fun <T : Tag> doInit(child: T, init: T.() -> Unit) {
        child.init()
        children.add(child)
    }
}

fun table(init: TABLE.() -> Unit) = TABLE().apply(init)

class TABLE : Tag("table") {
    fun tr(init: TR.() -> Unit) = doInit(TR(), init)
}

class TR : Tag("tr") {
    fun td(init: TD.() -> Unit) = doInit(TD(), init)
}
```

DSL для HTML

https://github.com/kotlin/kotlinx.html

DSL для тестов

```
val name = getLanguageName()
Assert.assertTrue(
    name.startsWith("Kot")
)
```

```
val name = getLanguageName()
name should startWith("Kot")
```

```
infix fun <T> T.should(verifier: (T) -> Unit) = verifier(this)

fun startWith(prefix: String) = { value: String ->
  if (!value.startsWith(prefix))
    throw AssertionError("String $value does not start with $prefix")
}
```

DSL для тестов

```
val name = getLanguageName()
name should startWith("Kot")
```

```
val name = getLanguageName()
name should start with "Kot"
```

```
infix fun String.should(x: start) = StartWrapper(this)

object start

class StartWrapper(val value: String) {
    infix fun with(prefix: String) { ... }
}
```

Главное

- Не пишите лишнего
- Оставьте форматирование IDE
- Обращайте внимание на замечания от IDE
- DSL - это красиво, освойте их

Вопросы?

yole@jetbrains.com
@intelliyole