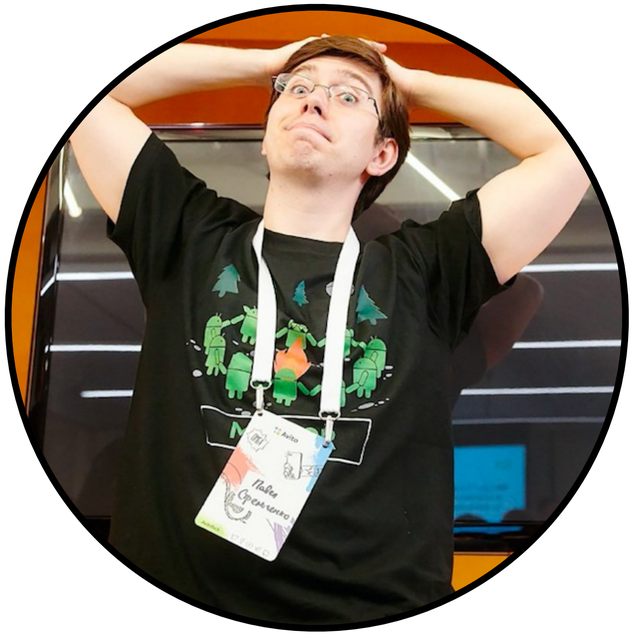
Фантастические плагины и где они обитают

Павел Стрельченко

Павел Стрельченко



Android-разработчик в HeadHunter

Ментор и лектор в Android Academy MSK

Увлекаюсь разработкой плагинов

О чем сегодня будем говорить?

1. Зачем плагин? Почему плагин?
2. Основы разработки плагинов
3. Внутренности IDEA: компоненты, PSI
4. UI, DI, генерация и модификация кода
5. Что делать дальше?

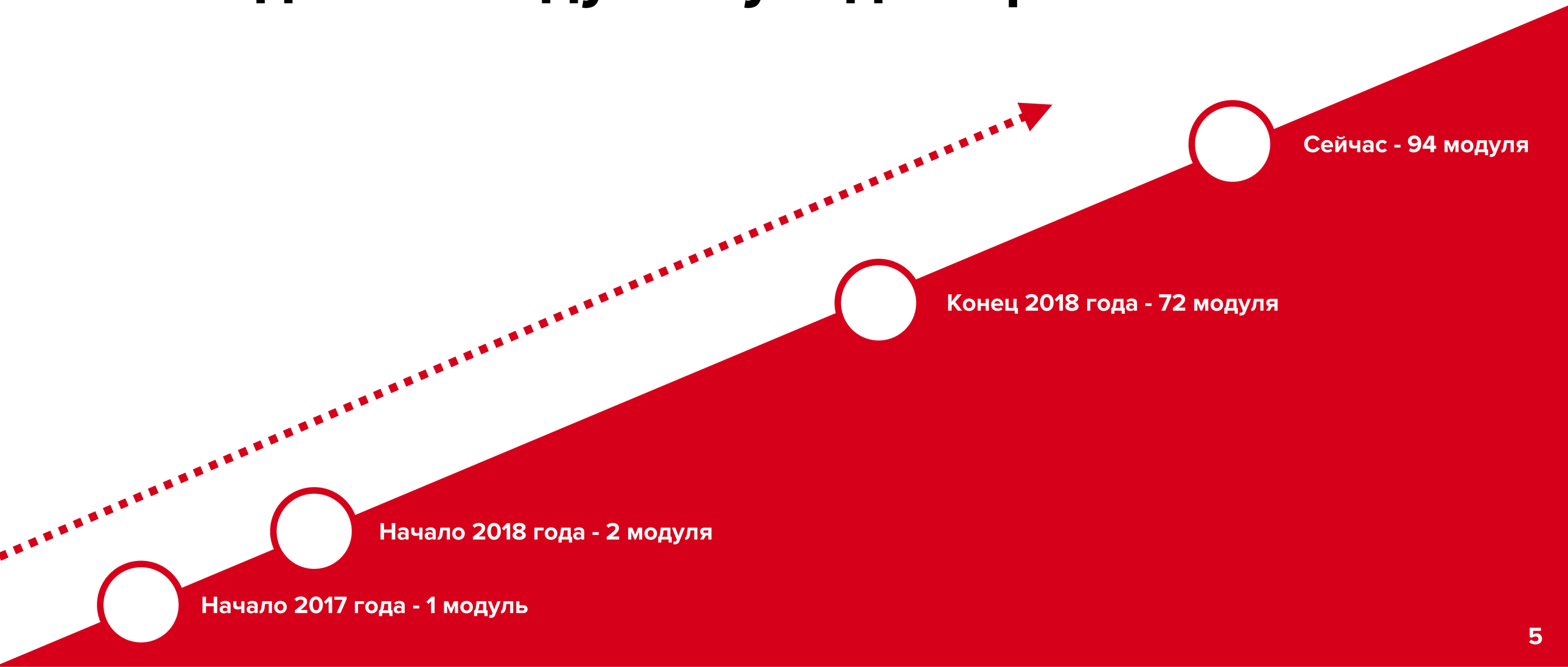
Ссылка на слайды

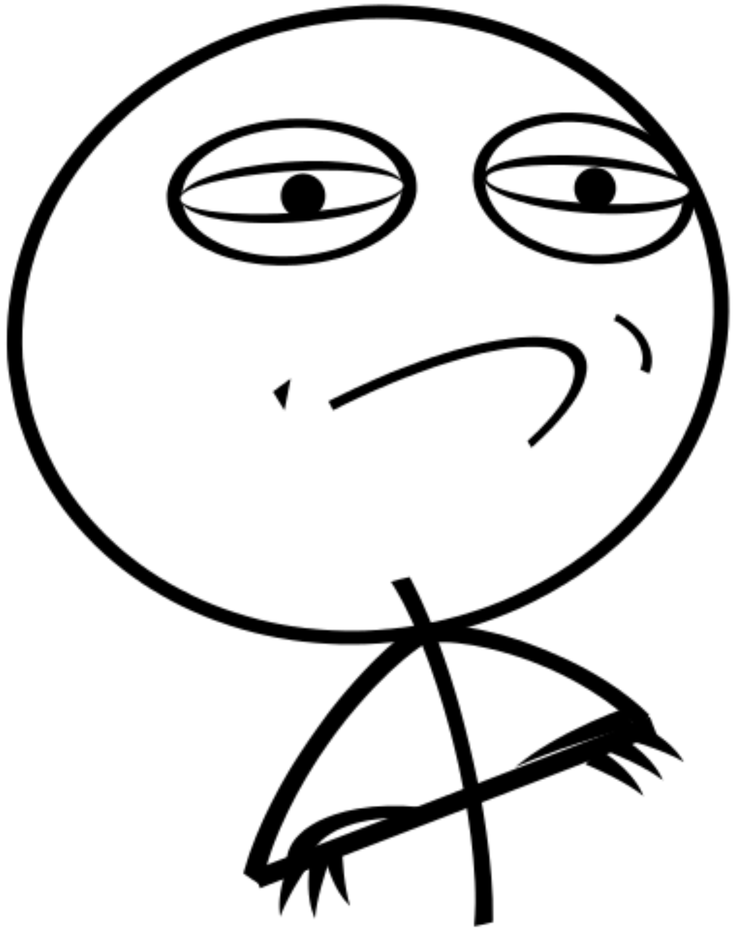
Зачем плагин?

Почему плагин?

Число модулей непрерывно растет

На создание модулей уходит время

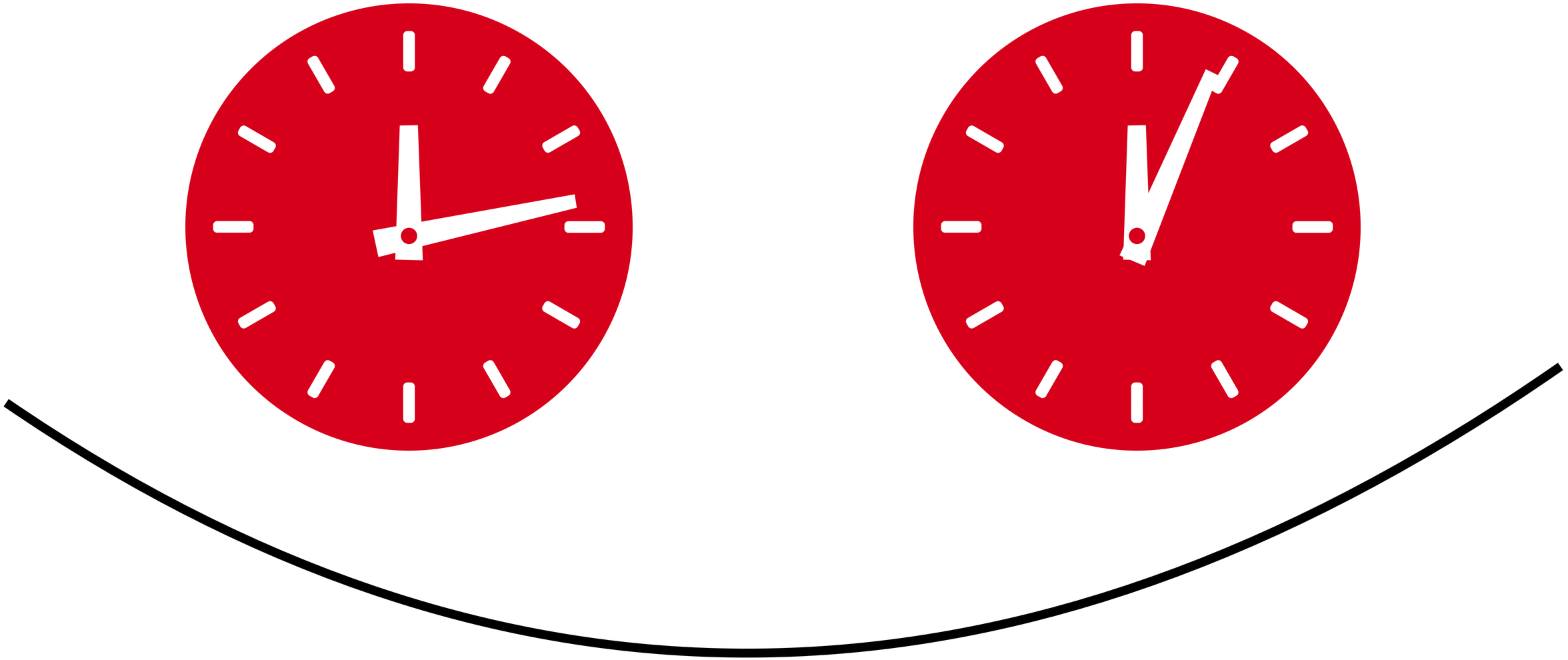




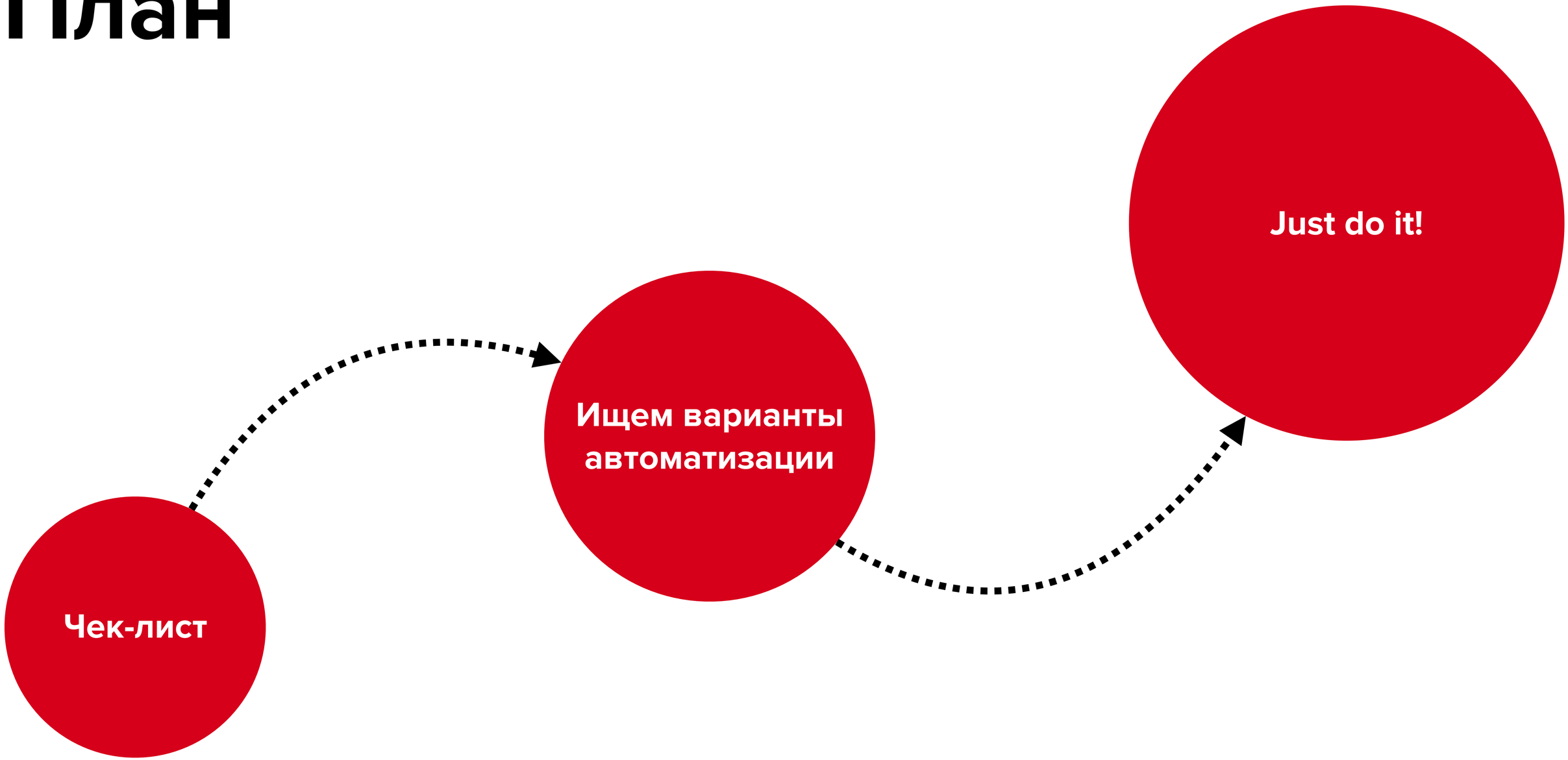
«Слабо автоматизировать?»

До плагина — 1 модуль ≒ 12 мин

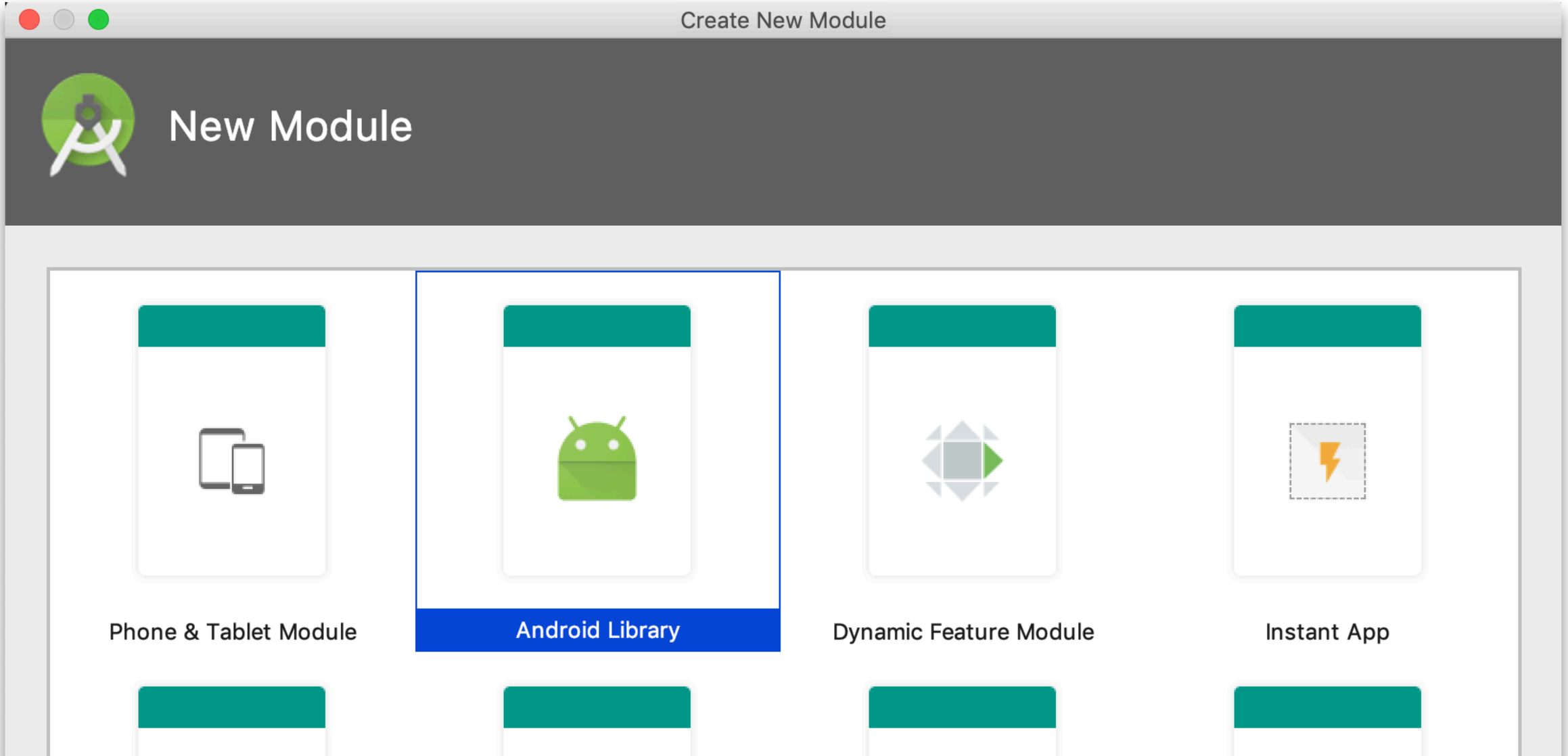
С плагином — в 2 раза быстрее



План



Создаем library-модуль



Прописываем пути к модулю

```
include ':analytics'
project(':analytics').projectDir =
    new File(settingsDir, ‘core/framework-metrics/analytics')

include ':feature-worknear'
project(':feature-worknear').projectDir =
    new File(settingsDir, ‘feature/feature-worknear’)
```

Меняем константы build.gradle

```
android {
    compileSdkVersion 27

    defaultConfig {
        minSdkVersion 15
        targetSdkVersion 27

    …

}
```

Меняем константы build.gradle

```
android {
    compileSdkVersion rootProject.ext.targetSdkVersion

    defaultConfig {
        minSdkVersion rootProject.ext.minSdkVersion
        targetSdkVersion rootProject.ext.targetSdkVersion

    ...

}
```

Подключаем плагины Kotlin-а

```
apply plugin: 'com.android.library'
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-kapt'
apply plugin: 'kotlin-android-extensions'


android {

...
```

Подключаем плагины Kotlin-a

```
apply plugin: 'com.android.library'
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-kapt'
apply plugin: 'kotlin-android-extensions'


android {

...
```

Подключаем плагины Kotlin-а

```
apply plugin: 'com.android.library'
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-kapt'
apply plugin: 'kotlin-android-extensions'


android {

...
```

Подключаем либы и модули

```
dependencies {
    def libraries = rootProject.ext.deps

    // Common modules
    compileOnly project(':logger')
    compileOnly project(':analytics')
    compileOnly project(':core-utils')
    compileOnly project(':common')

    // Kotlin
    compileOnly libraries.kotlin

    // DI
    compileOnly libraries.toothpick
    kapt libraries.toothpickCompiler

...
```

Подключаем либы и модули

```
dependencies {
    def libraries = rootProject.ext.deps

    // Common modules
    compileOnly project(':logger')
    compileOnly project(':analytics')
    compileOnly project(':core-utils')
    compileOnly project(':common')

    // Kotlin
    compileOnly libraries.kotlin

    // DI
    compileOnly libraries.toothpick
    kapt libraries.toothpickCompiler

...
```

Подключаем либы и модули

```
dependencies {
    def libraries = rootProject.ext.deps

    // Common modules
    compileOnly project(':logger')
    compileOnly project(':analytics')
    compileOnly project(':core-utils')
    compileOnly project(':common')

    // Kotlin
    compileOnly libraries.kotlin

    // DI
    compileOnly libraries.toothpick
    kapt libraries.toothpickCompiler

...
```

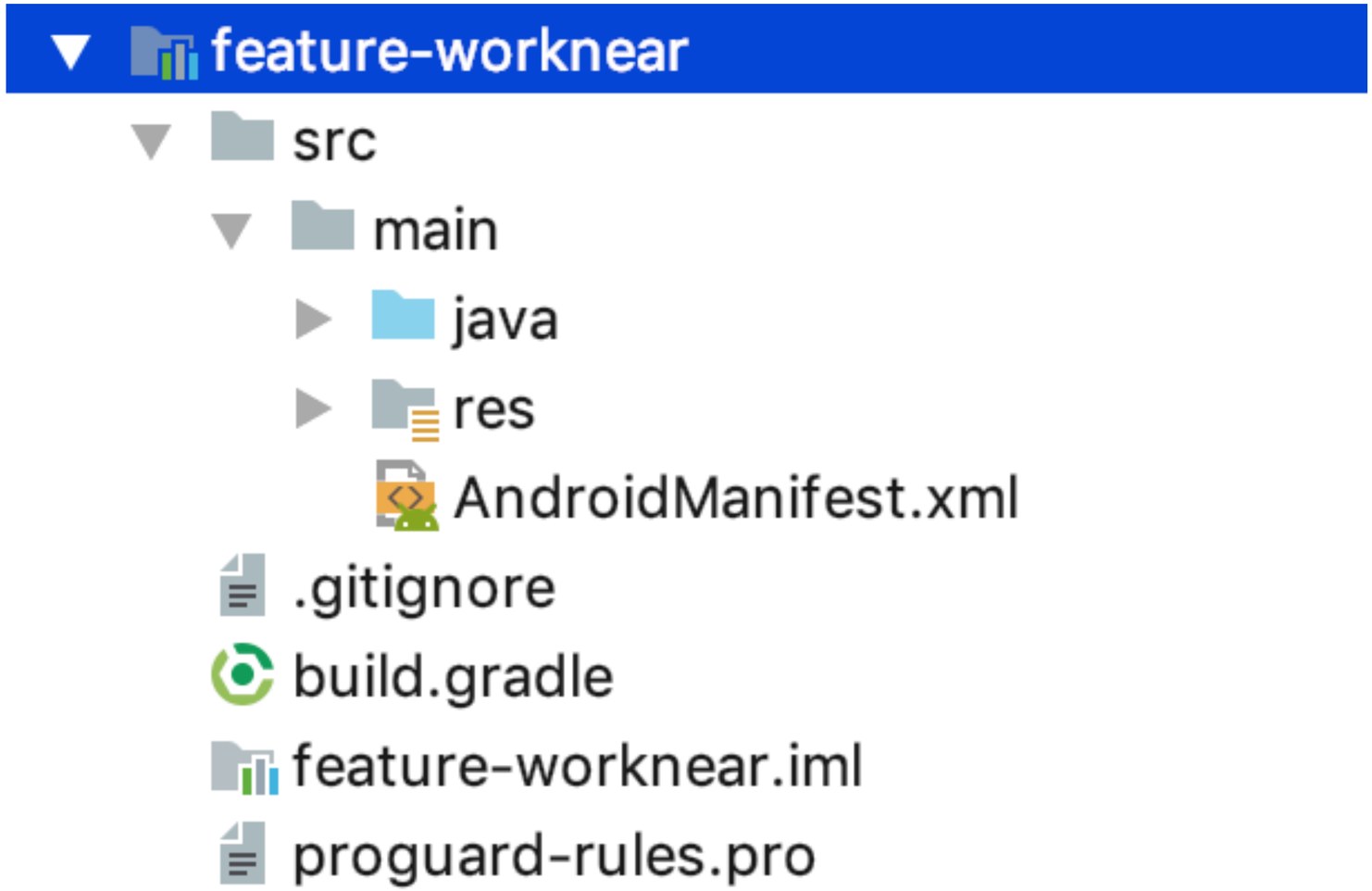
Настраиваем kapt для Toothpick

```
// Feature module build.gradle

defaultConfig {
    ...

    javaCompileOptions {
        annotationProcessorOptions {
            arguments = [
                toothpick_registry_package_name: "ru.hh.feature_worknear"
            ]
        }
    }

    ...
```

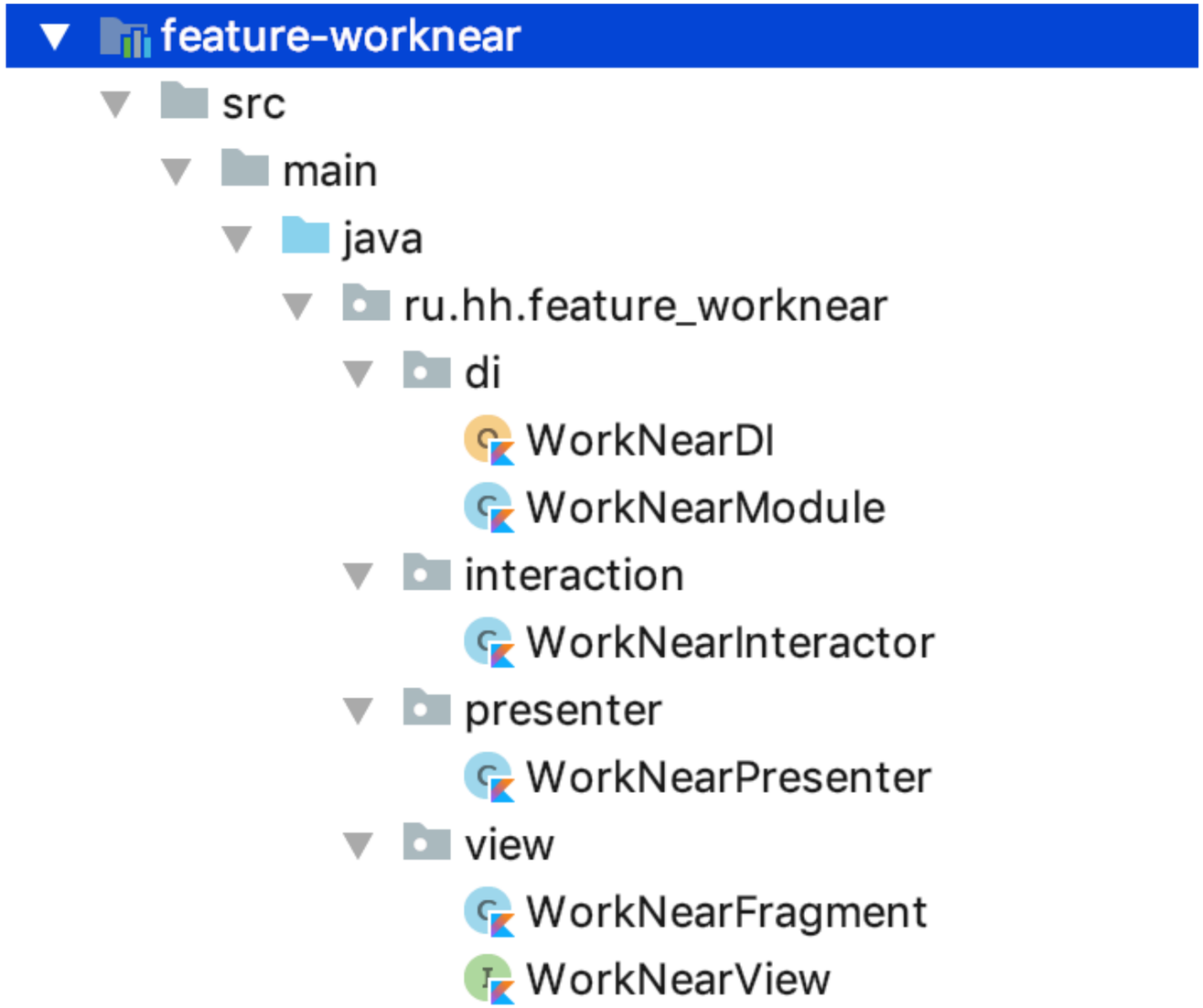
Настраиваем kapt-а для Moxy

```
// Feature module build.gradle

android {
    ...

    kapt {
        arguments {
            arg("moxyReflectorPackage", "ru.hh.feature_worknear")
        }
    }

    ...
```

Генерим новые файлы



Генерим новые файлы



Донастраиваем Toothpick в app

```
// App module build.gradle

defaultConfig {
    ...

    javaCompileOptions {
        annotationProcessorOptions {
            arguments = [
                toothpick_registry_package_name: "ru.hh.android",
                toothpick_registry_children_package_names: [
                    "ru.hh.analytics",
                    "ru.hh.feature_worknear",
                    ...
                ].join(",")
            ]
        }
    }
    ...
```

Донастраиваем Toothpick в app

```
// App module build.gradle

defaultConfig {
    ...

    javaCompileOptions {
        annotationProcessorOptions {
            arguments = [
                toothpick_registry_package_name: "ru.hh.android",
                toothpick_registry_children_package_names: [
                    "ru.hh.analytics",
                    "ru.hh.feature_worknear",
                    ...
                ].join(",")
            ]
        }
    }
    ...
```

Донастраиваем Moxy в app

```
// App module file

@RegisterMoxyReflectorPackages(
        "ru.hh.feature_force_update",
        "ru.hh.feature_profile",
        "ru.hh.feature_worknear"
        ...
) class MoxyReflectorStub
```

Донастраиваем Moxy в app

```
// App module file

@RegisterMoxyReflectorPackages(
        "ru.hh.feature_force_update",
        "ru.hh.feature_profile",
        "ru.hh.feature_worknear"
        ...
) class MoxyReflectorStub
```

Подключаем модуль

```
dependencies {
    def libraries = rootProject.ext.deps

    implementation project(':logger')
    implementation project(':dependency-handler')
    implementation project(':common')
    implementation project(':analytics')

    implementation project(':feature_worknear')
    ...
```

Подключаем модуль

```
dependencies {
    def libraries = rootProject.ext.deps

    implementation project(':logger')
    implementation project(':dependency-handler')
    implementation project(':common')
    implementation project(':analytics')

    implementation project(':feature_worknear')
    ...
```

Чек-лист

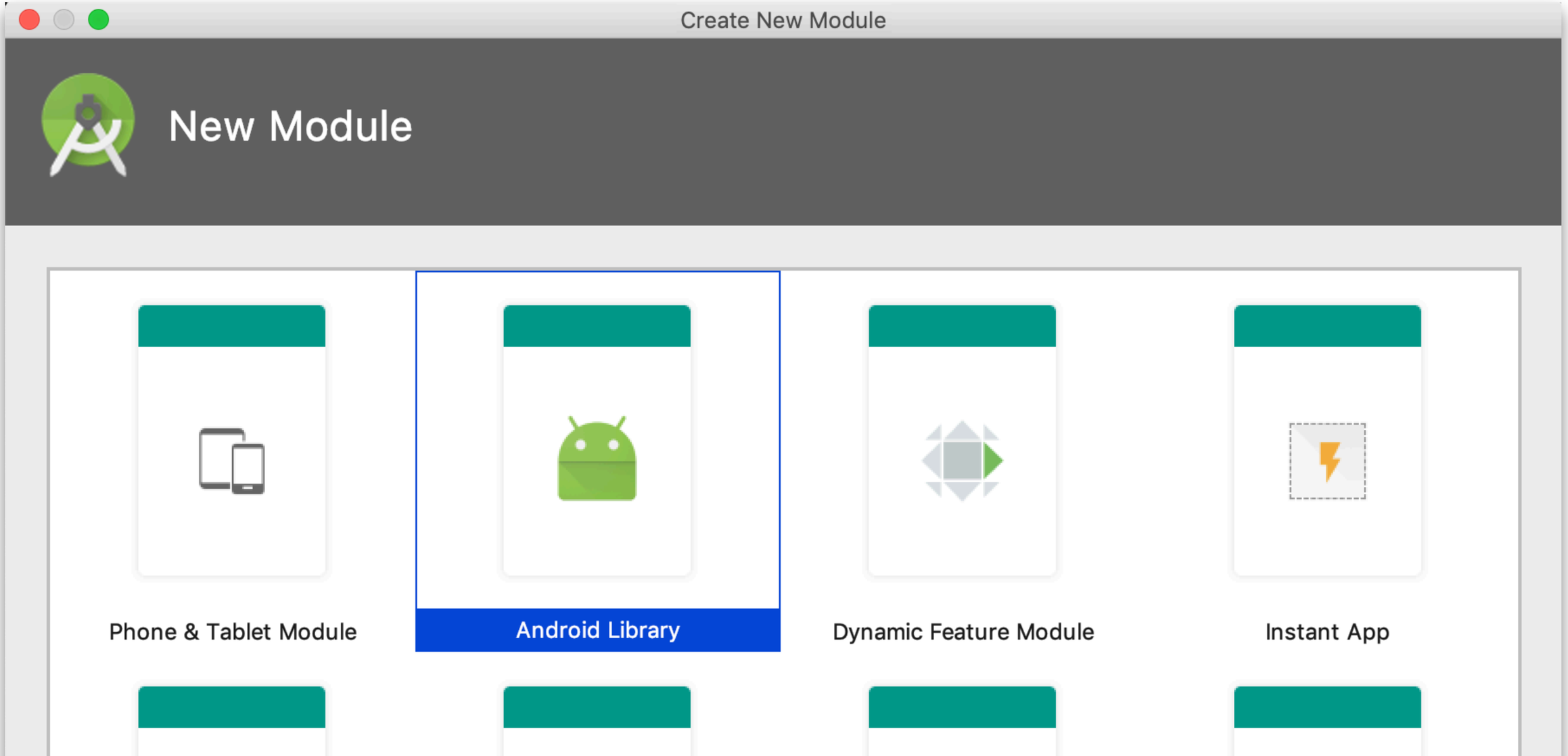
1. Создать модуль
2. Модифицировать set radle
3. Заменить adle модуля
4. лагины Kotlin-a
5. ить общие библиотеки и модули
6. Настроить kapt для Toothpick
7. Настроить kapt для Moxy
8. Сгенерировать базовые файлы модуля
9. Подключить созданный модуль в app

Так жить нельзя

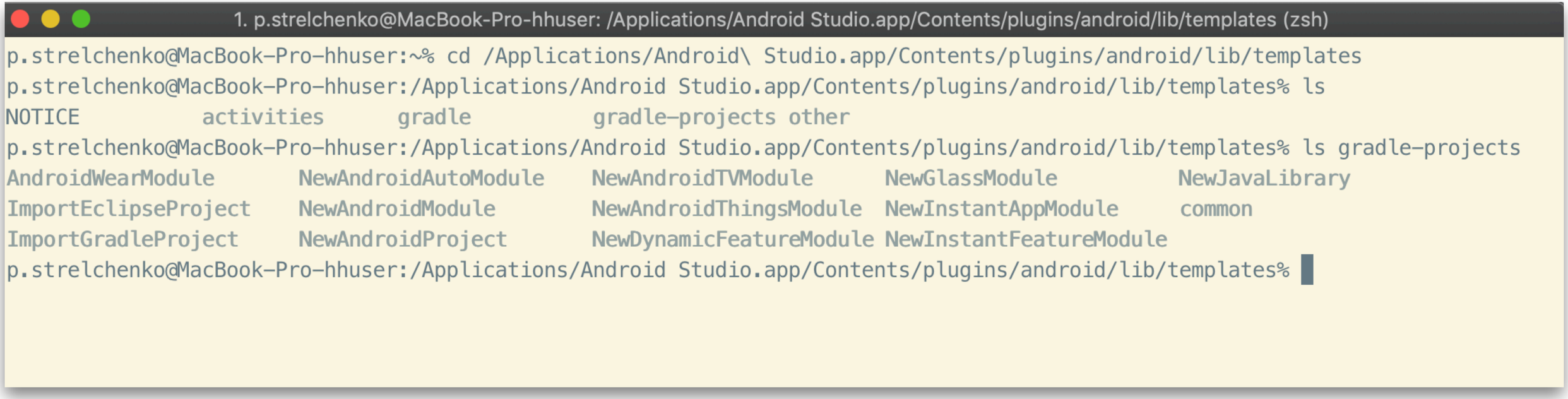
Варианты реализации

1. Ctrl + C — Ctrl + V

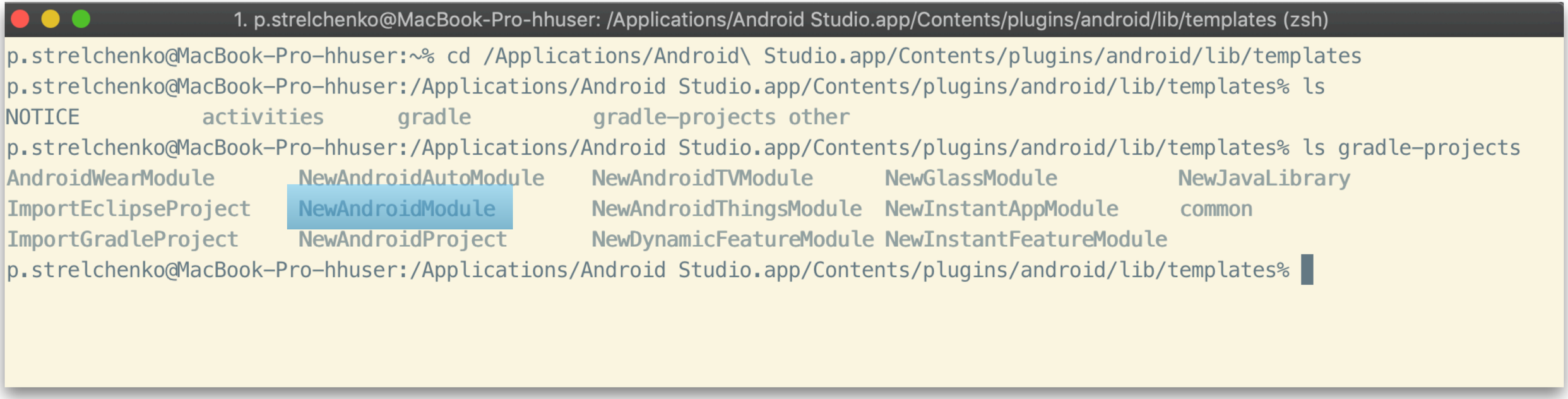
Ищем реализацию меню



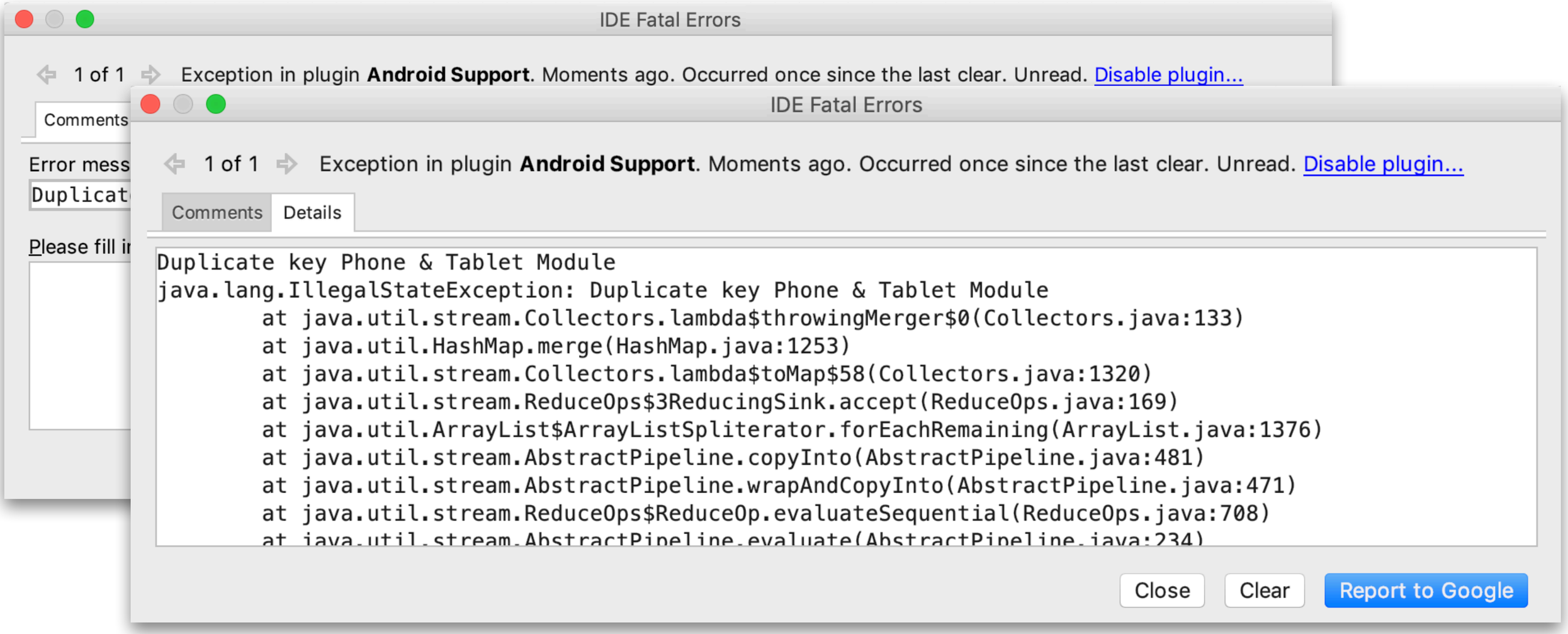
Ищем реализацию меню



Ищем реализацию меню



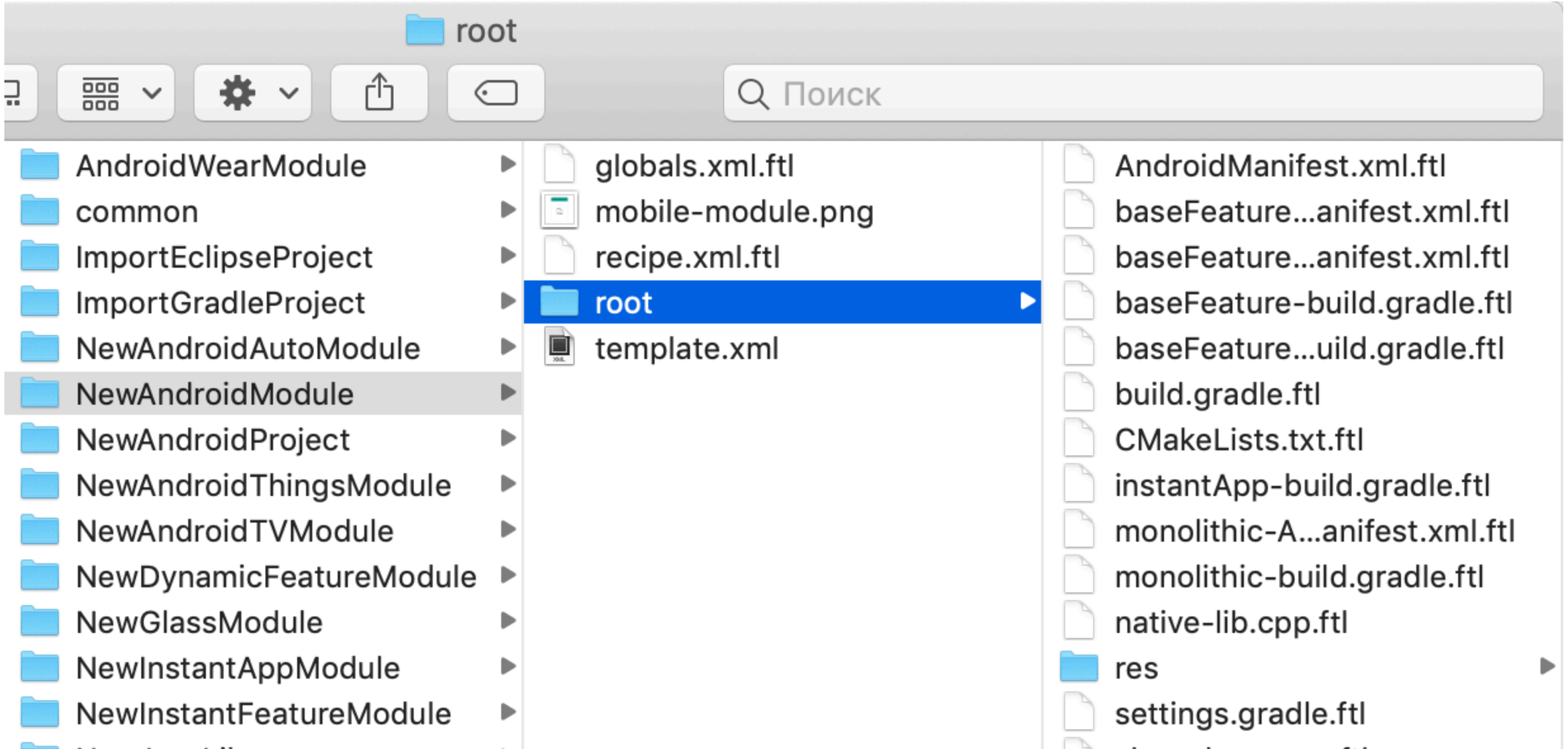
Ctrl + C — Ctrl + V ?..


IDE Fatal Errors
1 of 1
Exception in plugin Android Support. Moments ago. Occurred once since the last clear. Unread. Disable plugin...
Comments
Error mess
Duplicat
Please fill ir
IDE Fatal Errors
1 of 1
Exception in plugin Android Support. Moments ago. Occurred once since the last clear. Unread. Disable plugin...
Comments
Details
Duplicate key Phone & Tablet Module
java.lang.IllegalStateException: Duplicate key Phone & Tablet Module
at java.util.stream.Collectors.lambda$throwingMerger$0(Collectors.java:133)
at java.util.HashMap.merge(HashMap.java:1253)
at java.util.stream.Collectors.lambda$toMap$58(Collectors.java:1320)
at java.util.stream.ReduceOps$3ReducingSink.accept(ReduceOps.java:169)
at java.util.ArrayList$ArrayListSpliterator.forEachRemaining(ArrayList.java:1376)
at java.util.stream.AbstractPipeline.copyInto(AbstractPipeline.java:481)
at java.util.stream.AbstractPipeline.wrapAndCopyInto(AbstractPipeline.java:471)
at java.util.stream.ReduceOps$ReduceOp.evaluateSequential(ReduceOps.java:708)
Close
Clear
Report to Google

Варианты реализации

1. ~~Ctrl + C — Ctrl + V~~
2. **FreeMarker templates**

А что внутри шаблонов?



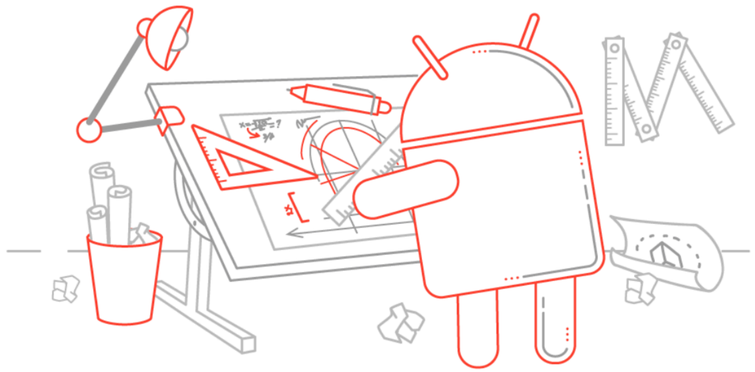
FreeMarker

Fi5t 12 января 2016 в 13:43

Тотальная шаблонизация

Блог компании Redmadrobot, Разработка мобильных приложений, Разработка под Android

Tutorial

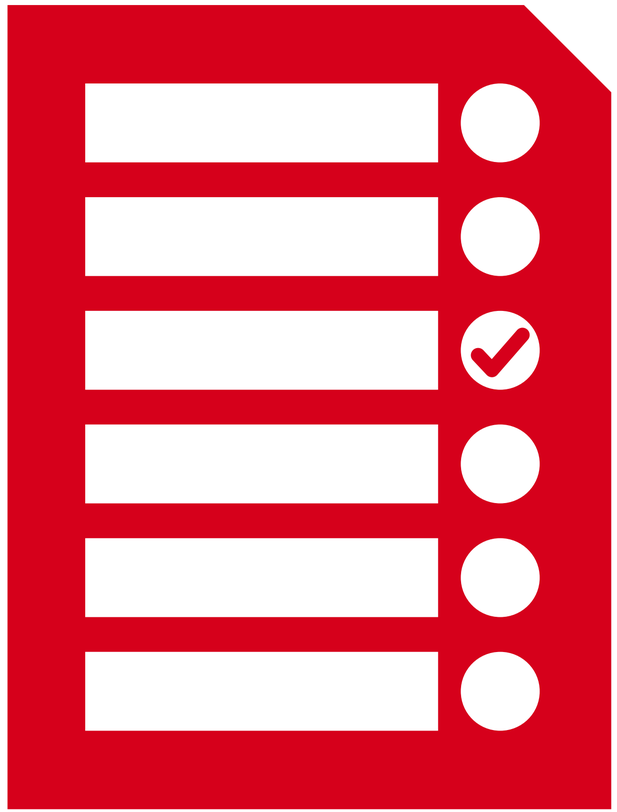


Когда ~~собаке~~ программисту нечего делать, он начинает все автоматизировать. Мне по роду своей деятельности приходится писать много кода и, конечно, хочется какие-то повторяющие вещи обобщить в виде библиотек, скриптов или шаблонов для Android Studio. О них и поговорим.

https://clck.ru/FDx9h

https://clck.ru/FDx48

Чек-лист



1. Создать модуль
2. Модифицировать settings.gradle
3. Заменить константы в build.gradle модуля
4. Подключить gradle-плагины Kotlin-a
5. Подключить общие библиотеки и модули
6. Настроить kapt для Toothpick
7. Настроить kapt для Moxy
8. Сгенерировать базовые файлы модуля
9. Подключить созданный модуль в app

Чек-лист

Генерация нового кода

Изменение существующего кода

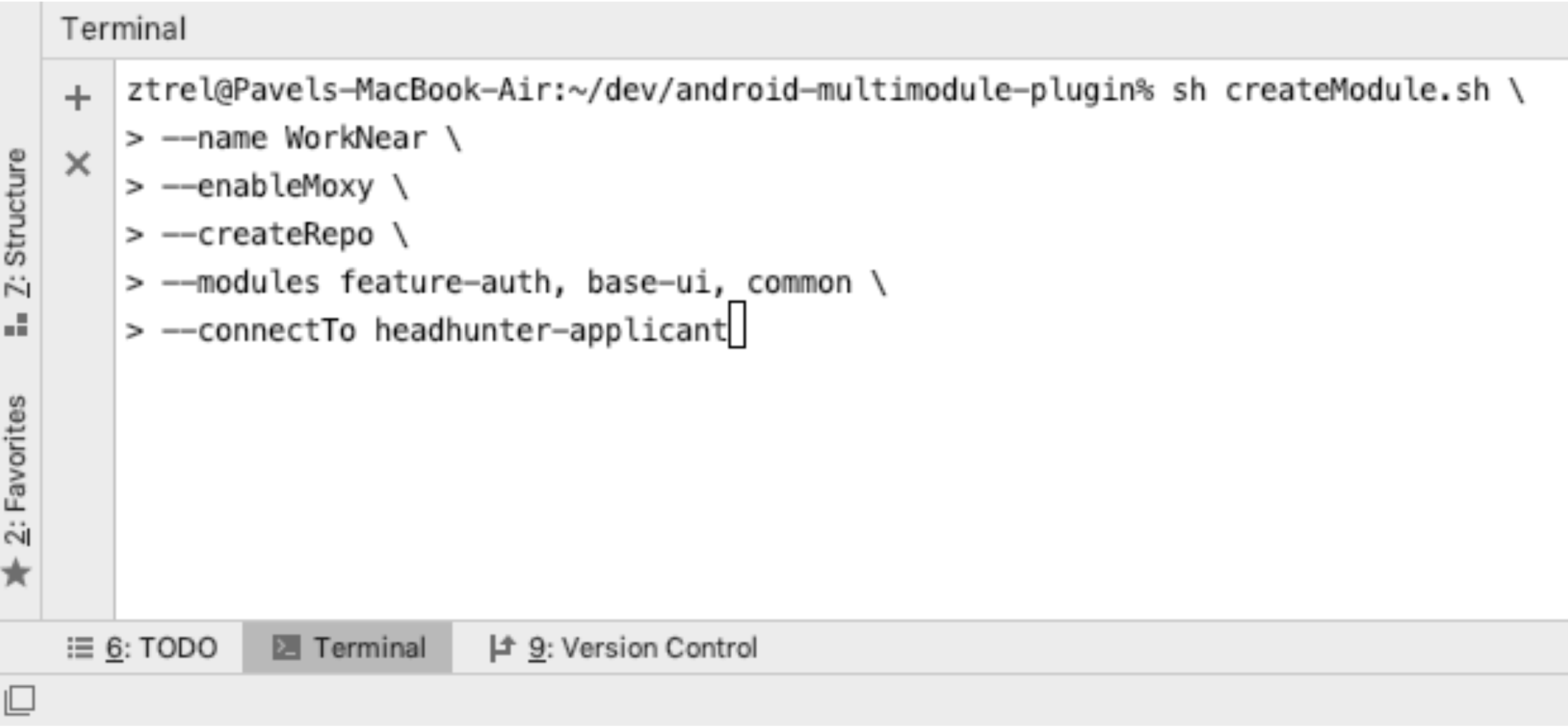
Settings.gradle merge

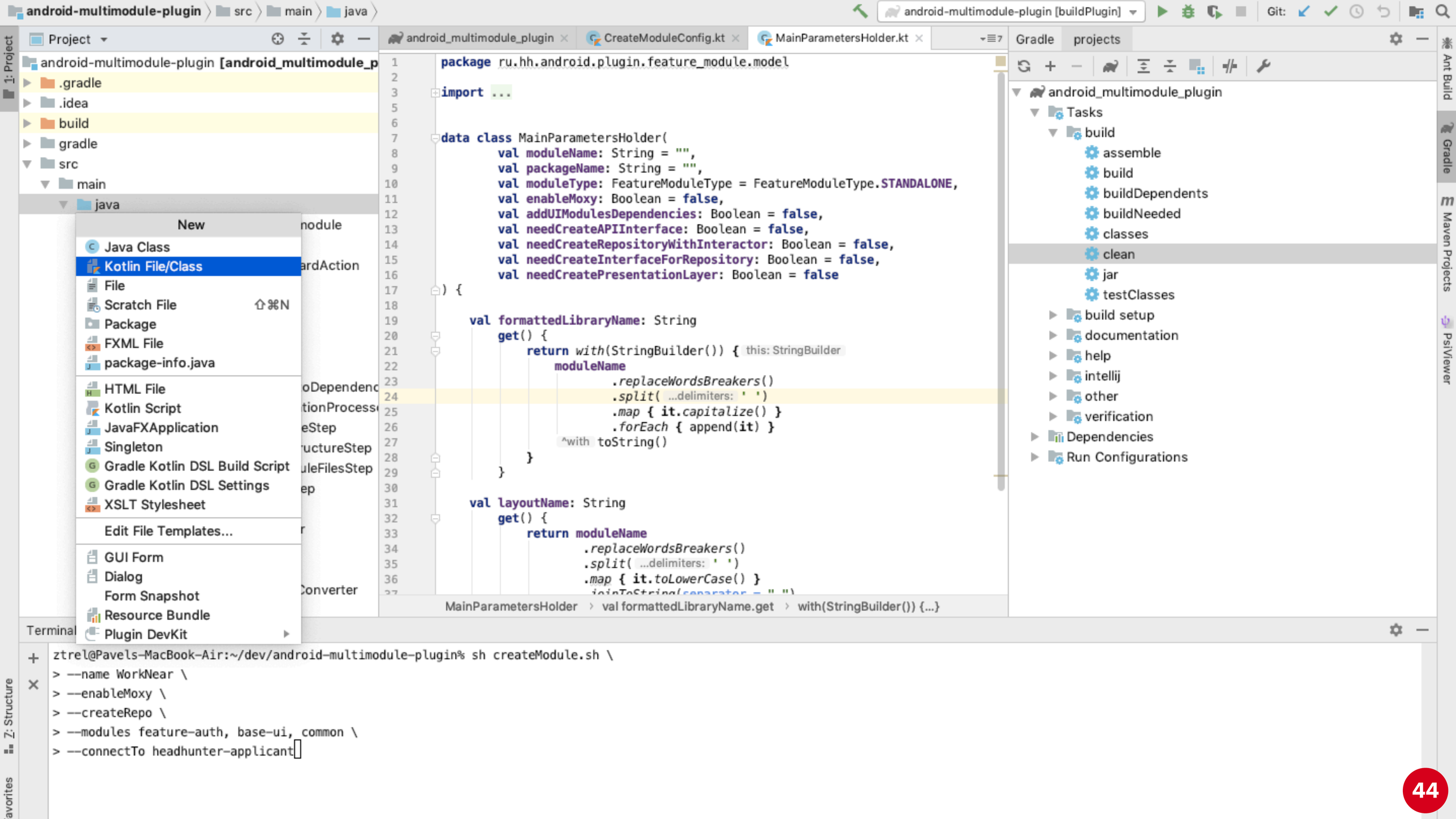
```
for (String line : Splitter.on('\n').omitEmptyStrings().trimResults().split(source)) {
  if (!line.startsWith("include")) {
    throw new RuntimeException("When merging settings.gradle files, only include directives can be merged.");
  }
  line = line.substring("include".length()).trim();
```

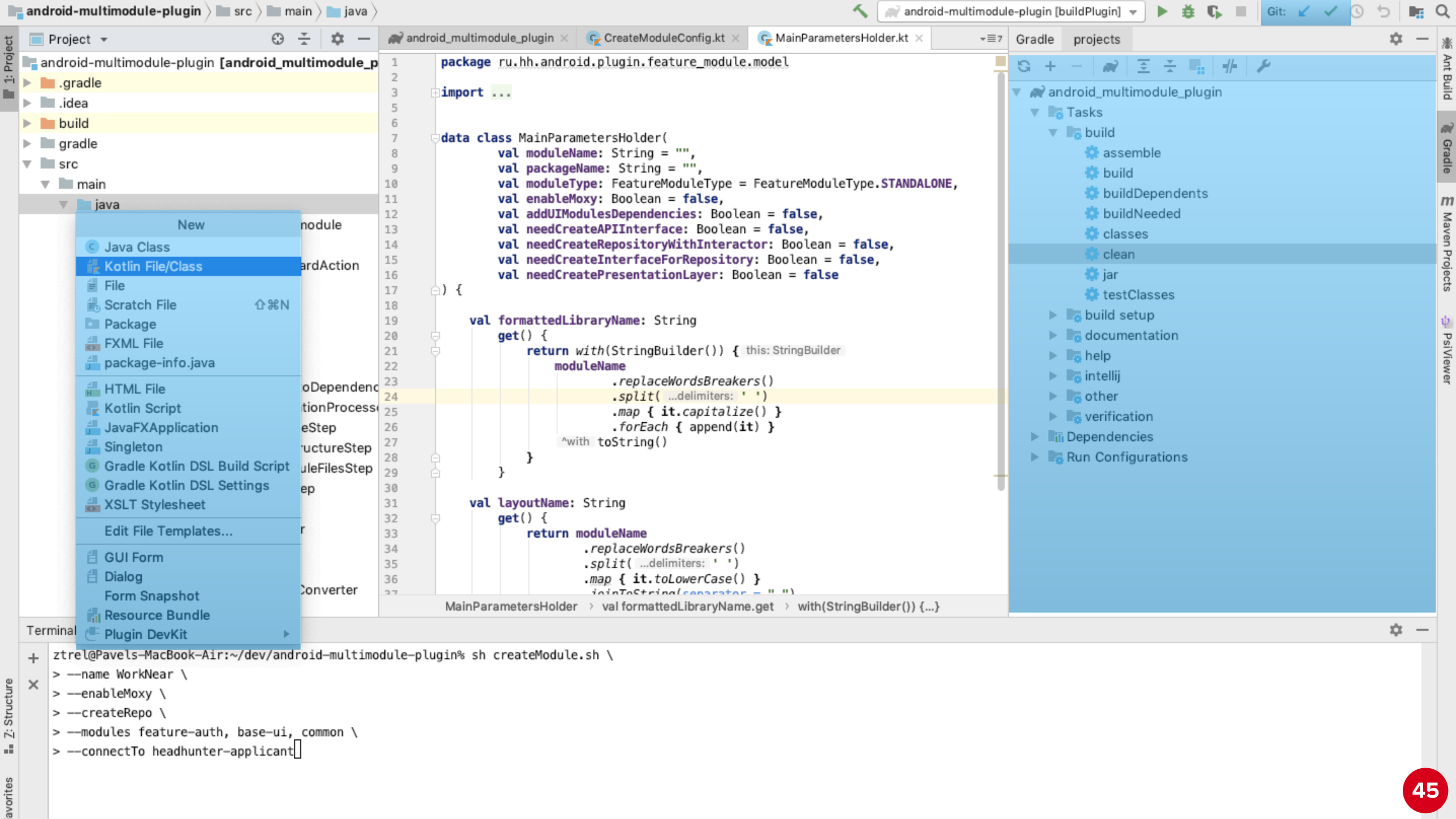
FreeMarker banned

Варианты реализации

1. ~~Ctrl + C — Ctrl + V~~
2. ~~FreeMarker templates~~
3. **Консольная утилита**


Terminal
ztrel@Pavels-MacBook-Air:~/dev/android-multimodule-plugin% sh createModule.sh \
> --name WorkNear \
> --enableMoxy \
> --createRepo \
> --modules feature-auth, base-ui, common \
> --connectTo headhunter-applicant
7: Structure
2: Favorites
6: TODO
Terminal
9: Version Control


android-multimodule-plugin
src
main
java
android-multimodule-plugin [buildPlugin]
Git:
Project
android-multimodule-plugin [android_multimodule_p
.gradle
.idea
build
gradle
src
main
java
New
Java Class
Kotlin File/Class
File
Scratch File ⇧⌘N
Package
FXML File
package-info.java
HTML File
Kotlin Script
JavaFXApplication
Singleton
Gradle Kotlin DSL Build Script
Gradle Kotlin DSL Settings
XSLT Stylesheet
Edit File Templates...
GUI Form
Dialog
Form Snapshot
Resource Bundle
Plugin DevKit
android_multimodule_plugin
CreateModuleConfig.kt
MainParametersHolder.kt
package ru.hh.android.plugin.feature_module.model
import ...
data class MainParametersHolder(
val moduleName: String = "",
val packageName: String = "",
val moduleType: FeatureModuleType = FeatureModuleType.STANDALONE,
val enableMoxy: Boolean = false,
val addUIModulesDependencies: Boolean = false,
val needCreateAPIInterface: Boolean = false,
val needCreateRepositoryWithInteractor: Boolean = false,
val needCreateInterfaceForRepository: Boolean = false,
val needCreatePresentationLayer: Boolean = false
) {
val formattedLibraryName: String
get() {
return with(StringBuilder()) { this: StringBuilder
moduleName
.replaceWordsBreakers()
.split(...delimiters: ' ')
.map { it.capitalize() }
.forEach { append(it) }
^with toString()
}
}
val layoutName: String
get() {
return moduleName
.replaceWordsBreakers()
.split(...delimiters: ' ')
.map { it.toLowerCase() }
MainParametersHolder › val formattedLibraryName.get › with(StringBuilder()) {...}
Gradle
projects
android_multimodule_plugin
Tasks
build
assemble
build
buildDependents
buildNeeded
classes
clean
jar
testClasses
build setup
documentation
help
intellij
other
verification
Dependencies
Run Configurations
Terminal
ztrel@Pavels-MacBook-Air:~/dev/android-multimodule-plugin% sh createModule.sh \
> --name WorkNear \
> --enableMoxy \
> --createRepo \
> --modules feature-auth, base-ui, common \
> --connectTo headhunter-applicant
1: Project
Z: Structure
Favorites
Ant Build
Gradle
Maven Projects
PsiViewer
44


android-multimodule-plugin
src
main
java
android-multimodule-plugin [buildPlugin]
Git:
Project
android-multimodule-plugin [android_multimodule_p
.gradle
.idea
build
gradle
src
main
java
New
Java Class
Kotlin File/Class
File
Scratch File
Package
FXML File
package-info.java
HTML File
Kotlin Script
JavaFXApplication
Singleton
Gradle Kotlin DSL Build Script
Gradle Kotlin DSL Settings
XSLT Stylesheet
Edit File Templates...
GUI Form
Dialog
Form Snapshot
Resource Bundle
Plugin DevKit
android_multimodule_plugin
CreateModuleConfig.kt
MainParametersHolder.kt
package ru.hh.android.plugin.feature_module.model
import ...
data class MainParametersHolder(
val moduleName: String = "",
val packageName: String = "",
val moduleType: FeatureModuleType = FeatureModuleType.STANDALONE,
val enableMoxy: Boolean = false,
val addUIModulesDependencies: Boolean = false,
val needCreateAPIInterface: Boolean = false,
val needCreateRepositoryWithInteractor: Boolean = false,
val needCreateInterfaceForRepository: Boolean = false,
val needCreatePresentationLayer: Boolean = false
) {
val formattedLibraryName: String
get() {
return with(StringBuilder()) { this: StringBuilder
moduleName
.replaceWordsBreakers()
.split(...delimiters: ' ')
.map { it.capitalize() }
.forEach { append(it) }
^with toString()
}
}
val layoutName: String
get() {
return moduleName
.replaceWordsBreakers()
.split(...delimiters: ' ')
.map { it.toLowerCase() }
MainParametersHolder › val formattedLibraryName.get › with(StringBuilder()) {...}
Gradle
projects
android_multimodule_plugin
Tasks
build
assemble
build
buildDependents
buildNeeded
classes
clean
jar
testClasses
build setup
documentation
help
intellij
other
verification
Dependencies
Run Configurations
Terminal
ztrel@Pavels-MacBook-Air:~/dev/android-multimodule-plugin% sh createModule.sh \
> --name WorkNear \
> --enableMoxy \
> --createRepo \
> --modules feature-auth, base-ui, common \
> --connectTo headhunter-applicant
1: Project
Z: Structure
Favorites
Ant Build
Gradle
Maven Projects
PsiViewer
45

Product

All products

Tags

- [ ] Apps, Notification and Interaction Applications
- [ ] Administration Tools
- [ ] Android
- [ ] Agile
- [ ] Annotations
- [ ] Auto-property
- [ ] AngularJS
- [ ] Build
- [ ] Build Runners
- [ ] Build and Debug

FILTER

4266 plugins found

Search for ...

- [ ] Has source code
- [ ] Show only featured plugins

Sort by: relevance

BashSupport

By Joachim Ansorg

languages featured by jetbrains

Bash language support for the IntelliJ platform.

Mar 29, 2019 9 811 022 4.5

Scala

By Evgueny Vigdorchik, Alexander Podkhalyuzin, Dmitry Naydanov, Mikhail Mutcianko, Pavel Fatin

languages featured by jetbrains

Adds support for the Scala language. The following features are available for free with IntelliJ IDEA Community Edition:.

Apr 06, 2019 11 717 781 4.5

JET BRAINS

IDEs .NET Team Tools Dev Guide Sign In

Product

All products

Tags

- [ ] Apps, Notification and Interaction Applications
- [ ] Administration Tools
- [ ] Android
- [ ] Agile
- [ ] Annotations
- [ ] Auto-property
- [ ] AngularJS
- [ ] Build
- [ ] Build Runners
- [ ] Build and Debug

FILTER

4266 plugins found

Search for ...

- [ ] Has source code
- [ ] Show only featured plugins

Sort by: relevance

BashSupport

By Joachim Ansorg

languages featured by jetbrains

Bash language support for the IntelliJ platform.

Mar 29, 2019 9 811 022 4.5

Scala

By Evgueny Vigdorchik, Alexander Podkhalyuzin, Dmitry Naydanov, Mikhail Mutcianko, Pavel Fatin

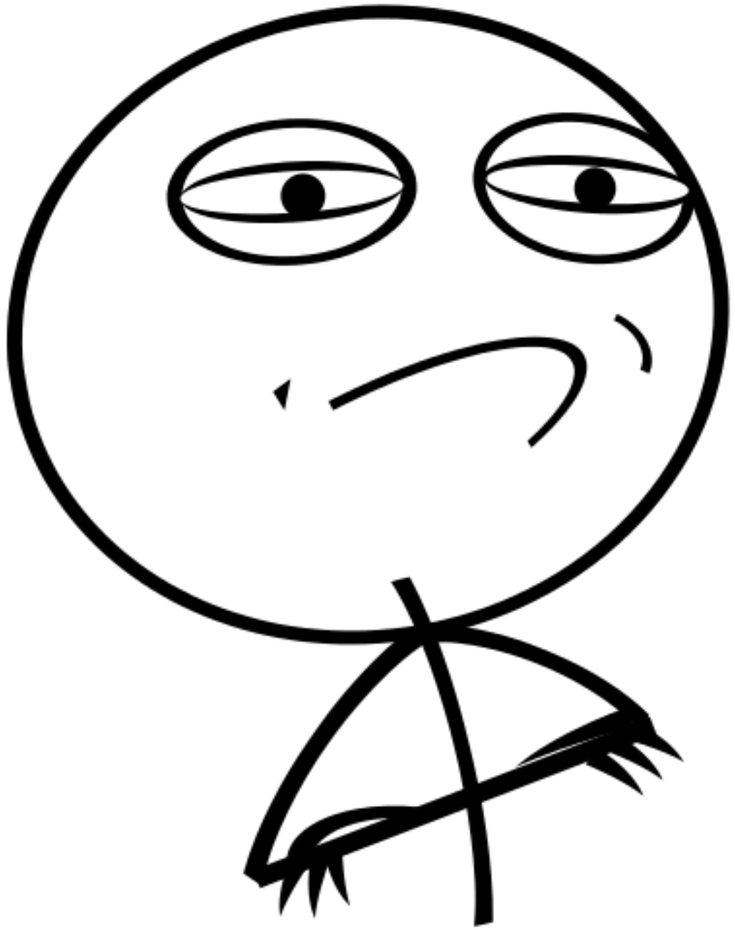
languages featured by jetbrains

Adds support for the Scala language. The following features are available for free with IntelliJ IDEA Community Edition:.

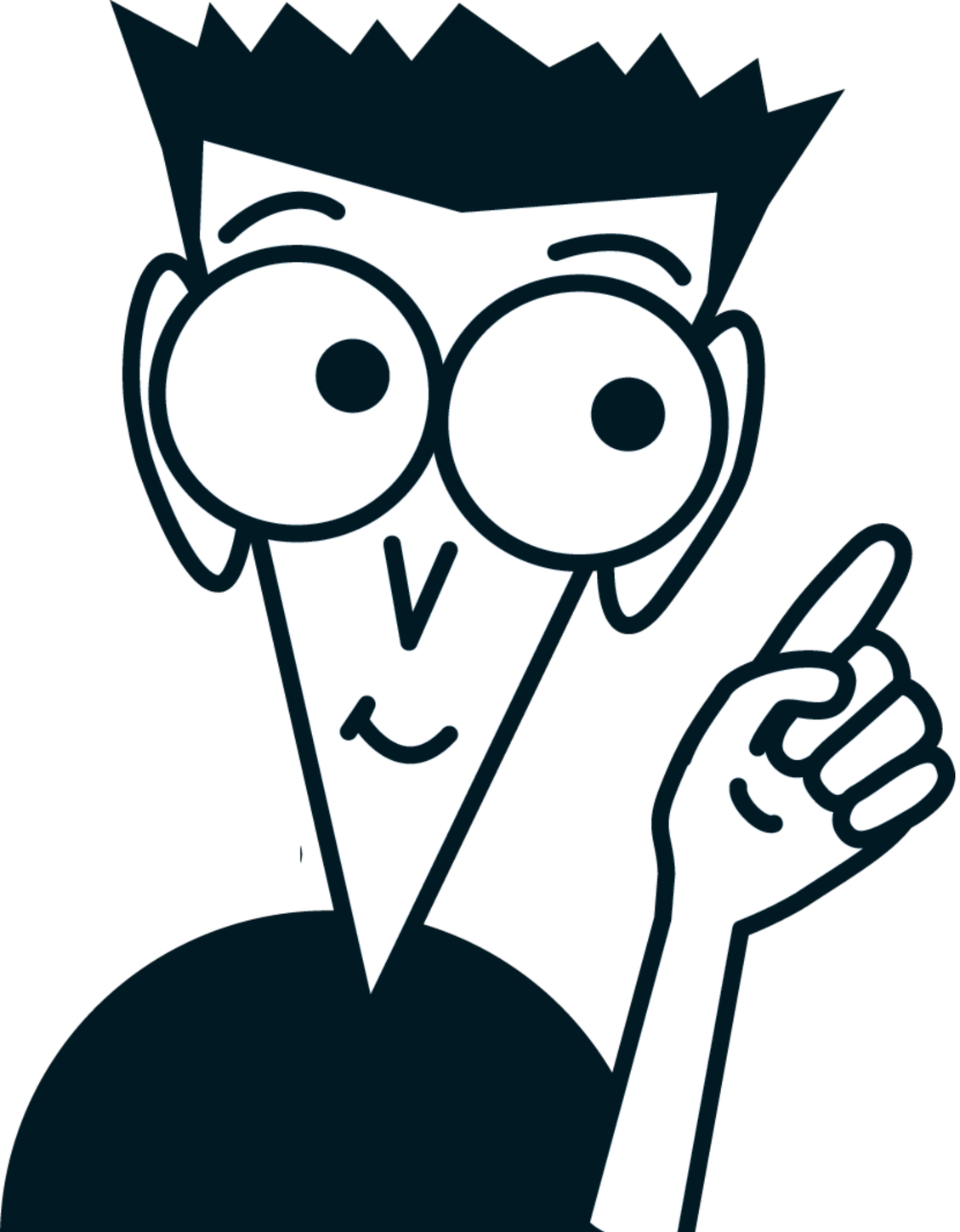
Apr 06, 2019 11 717 781 4.5

Варианты реализации

1. ~~Ctrl + C — Ctrl + V~~
2. ~~FreeMarker templates~~
3. ~~Консольная утилита~~
4. **Intellij IDEA плагин**



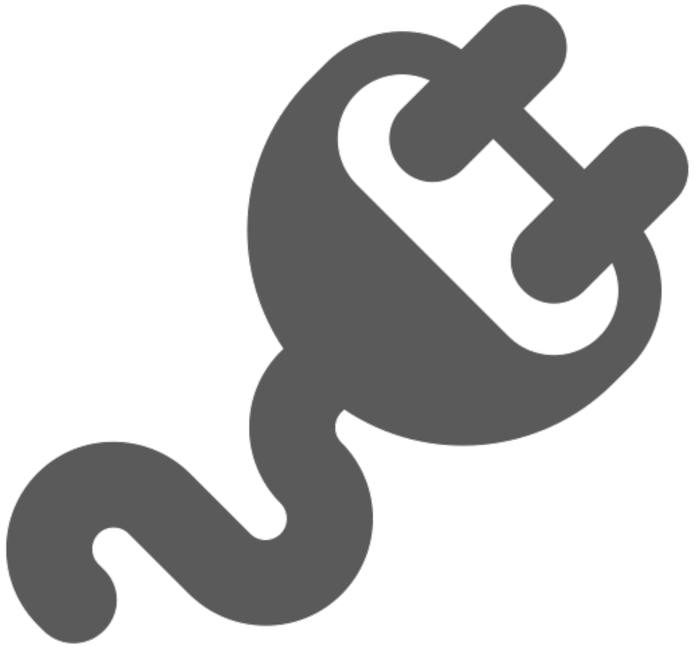
Сделаем плагин



Основы разработки плагинов

Вам потребуются

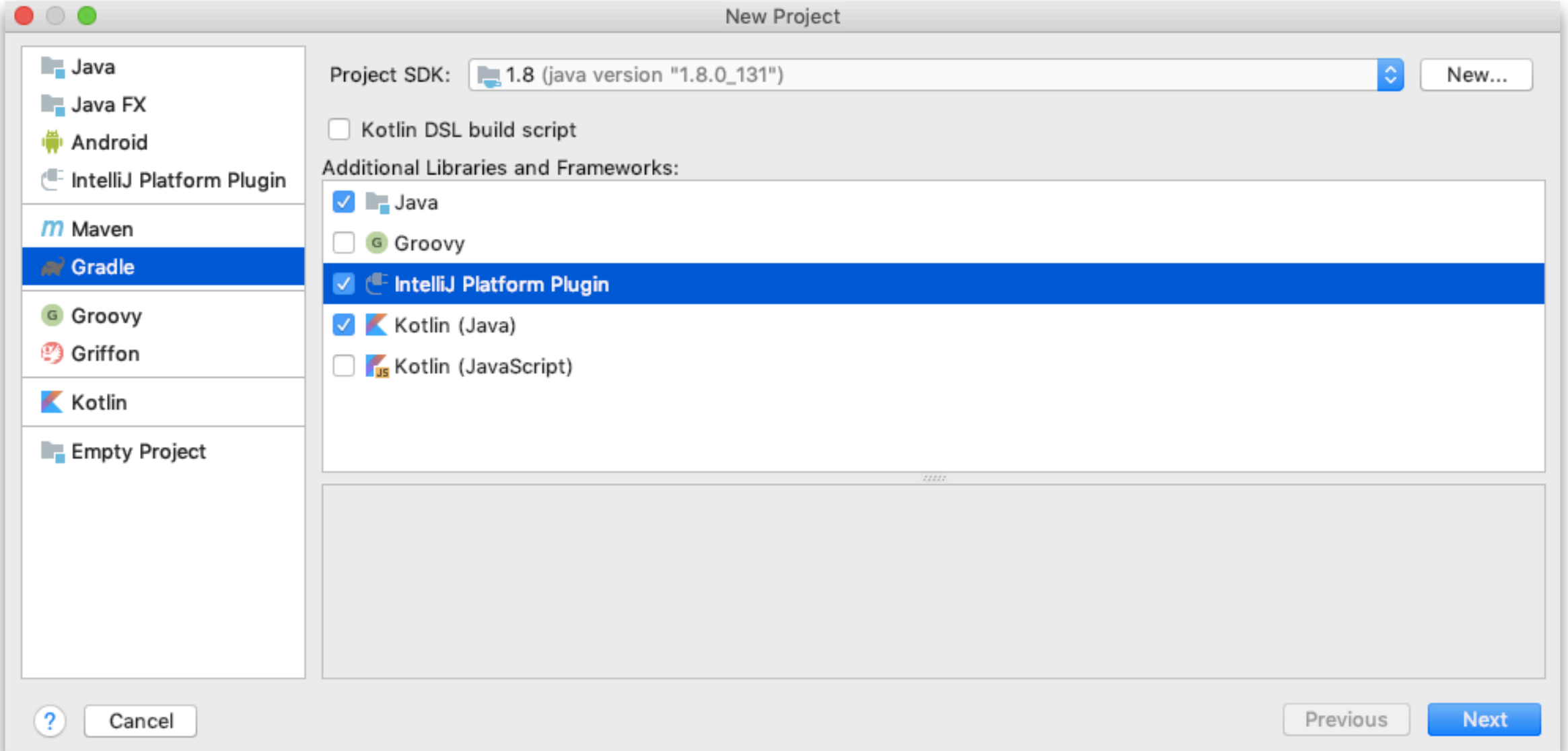
Intellij IDEA CE



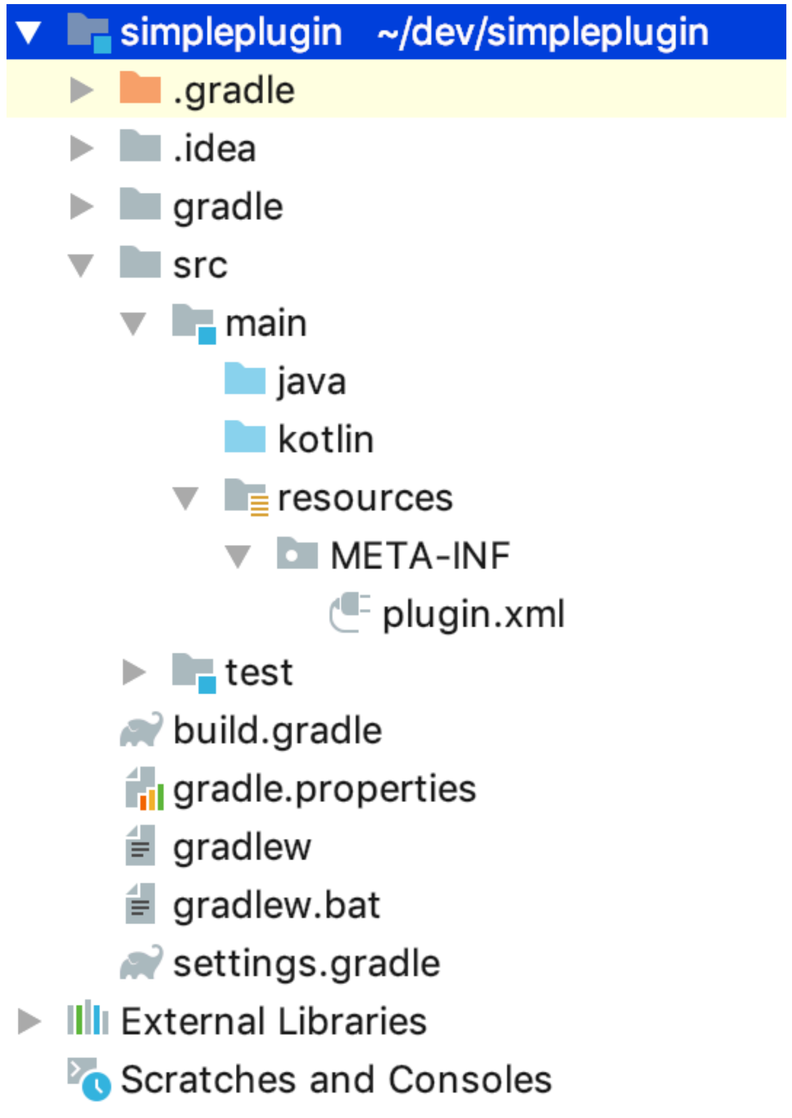
Plugin DevKit

Any JVM language

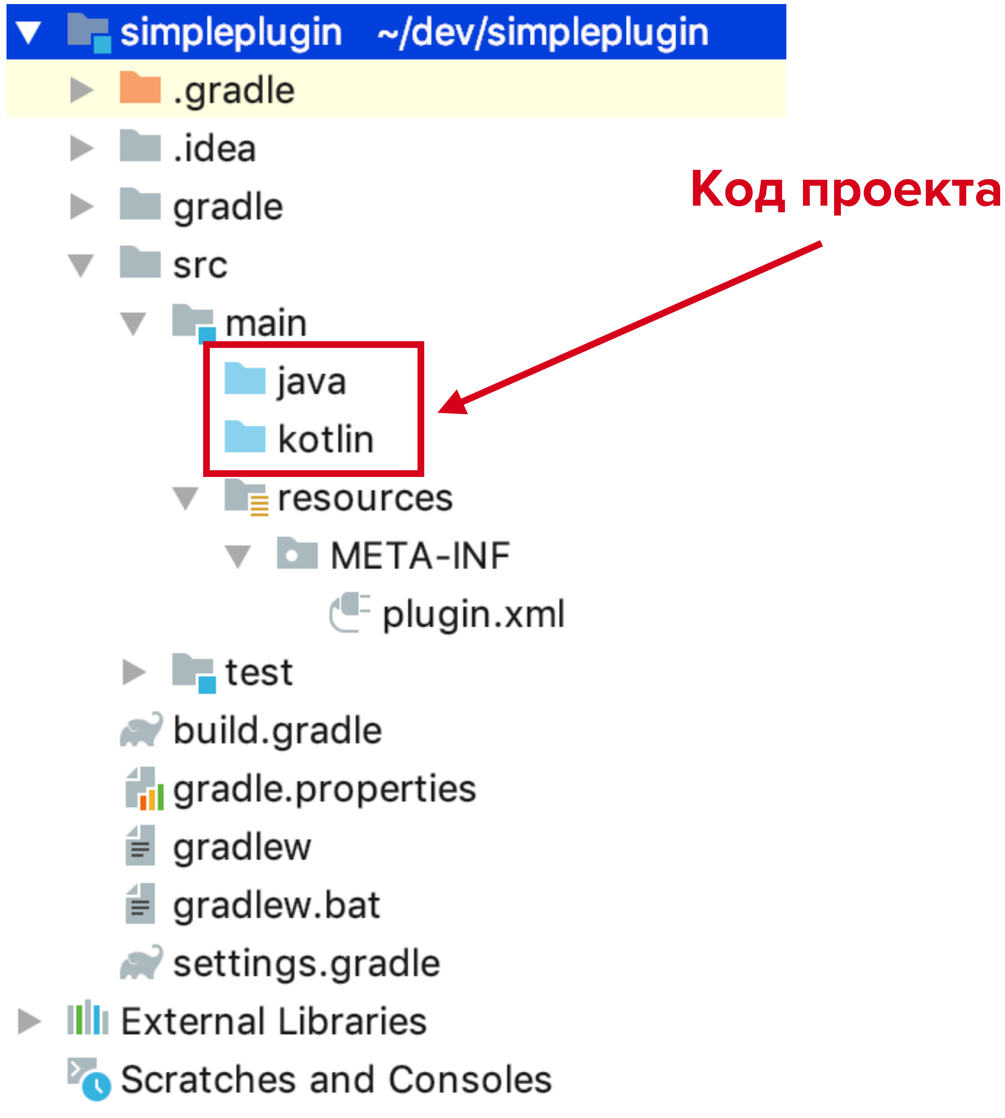
Создаем проект плагина



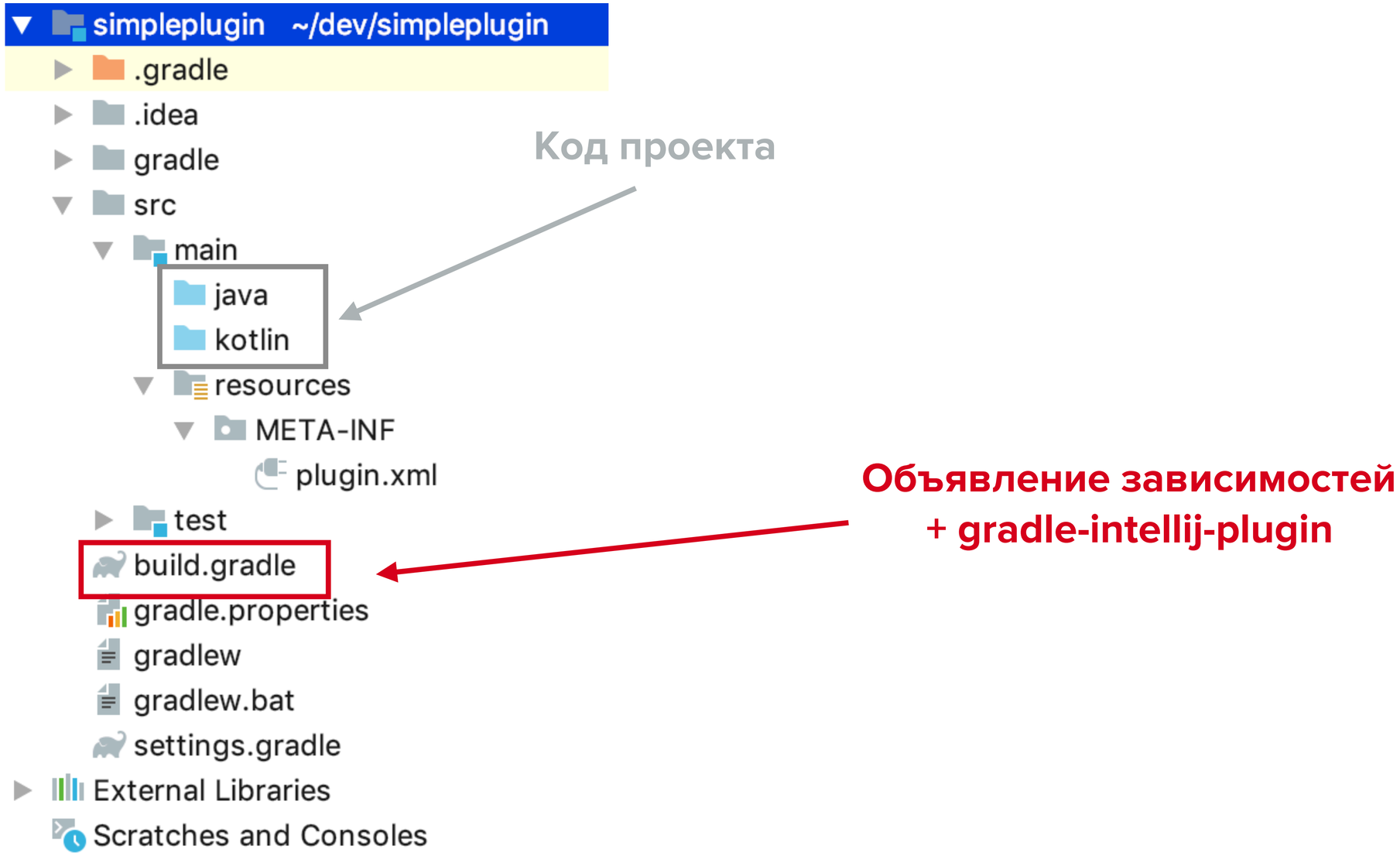
Структура плагина



Структура плагина



Структура плагина



gradle-intellij-plugin

1. Позволяет использовать Gradle

gradle-intellij-plugin

1. Позволяет использовать Gradle
2. **Добавляет полезные gradle-таски**

gradle-intellij-plugin

1. Позволяет использовать Gradle
2. **Добавляет полезные gradle-таски**
   - 2.1 **runIde**

gradle-intellij-plugin

1. Позволяет использовать Gradle
2. **Добавляет полезные gradle-таски**
   - 2.1 **runIde**
   - 2.2 **buildPlugin**

gradle-intellij-plugin

1. Позволяет использовать Gradle
2. **Добавляет полезные gradle-таски**
   - 2.1 **runIde**
   - 2.2 **buildPlugin**
   - 2.3 **verifyPlugin**

gradle-intellij-plugin

1. Позволяет использовать Gradle
2. Добавляет полезные gradle-таски
    - 2.1 runIde
    - 2.2 buildPlugin
    - 2.3 verifyPlugin
3. **Становится проще добавлять зависимости от других плагинов**

gradle-intellij-plugin

1. Позволяет использовать Gradle
2. Добавляет полезные gradle-таски
   - 2.1 runIde
   - 2.2 buildPlugin
   - 2.3 verifyPlugin
3. **Становится проще добавлять зависимости от других плагинов**

Структура плагина

plugin.xml

```
<idea-plugin>
    <id>com.experiment.simpleplugin</id>
    <name>Hello, world</name>
    <vendor
        email="myemail@yourcompany.com"
        url="http://www.mycompany.com">
            My company
    </vendor>

    <description><![CDATA[
    My first ever plugin - try to open Hello world dialog<br>
    ]]></description>

    ...
<idea-plugin>
```

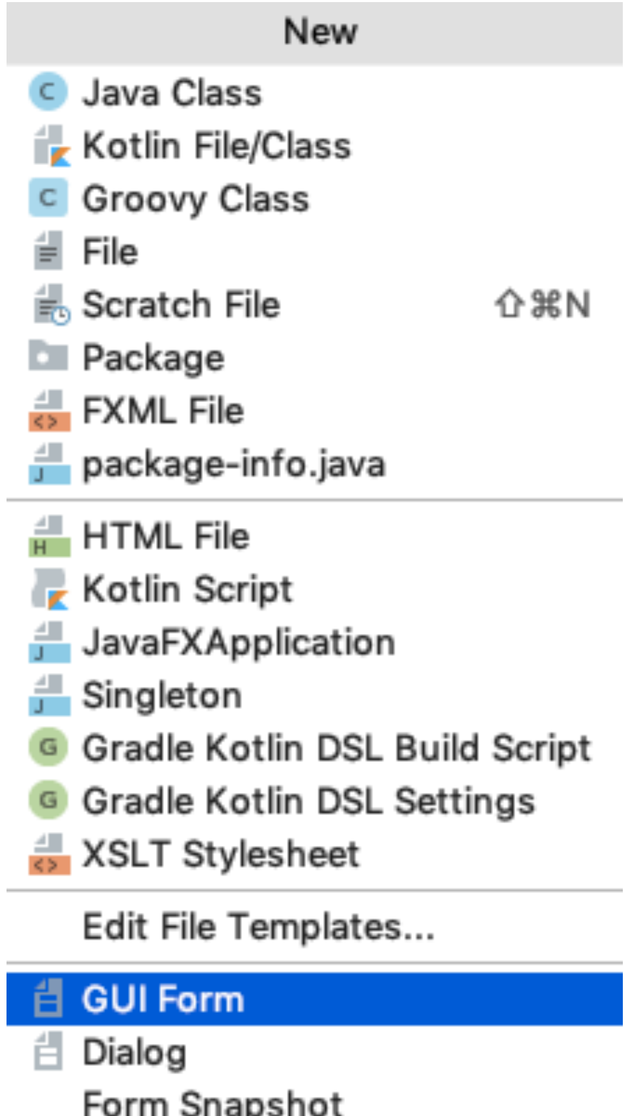
plugin.xml

```
<idea-plugin>
    ...

    <depends>com.intellij.modules.lang</depends>
    <depends>org.jetbrains.kotlin</depends>
    <depends>org.intellij.groovy</depends>

    <idea-version since-build="163"/>

    ...
<idea-plugin>
```

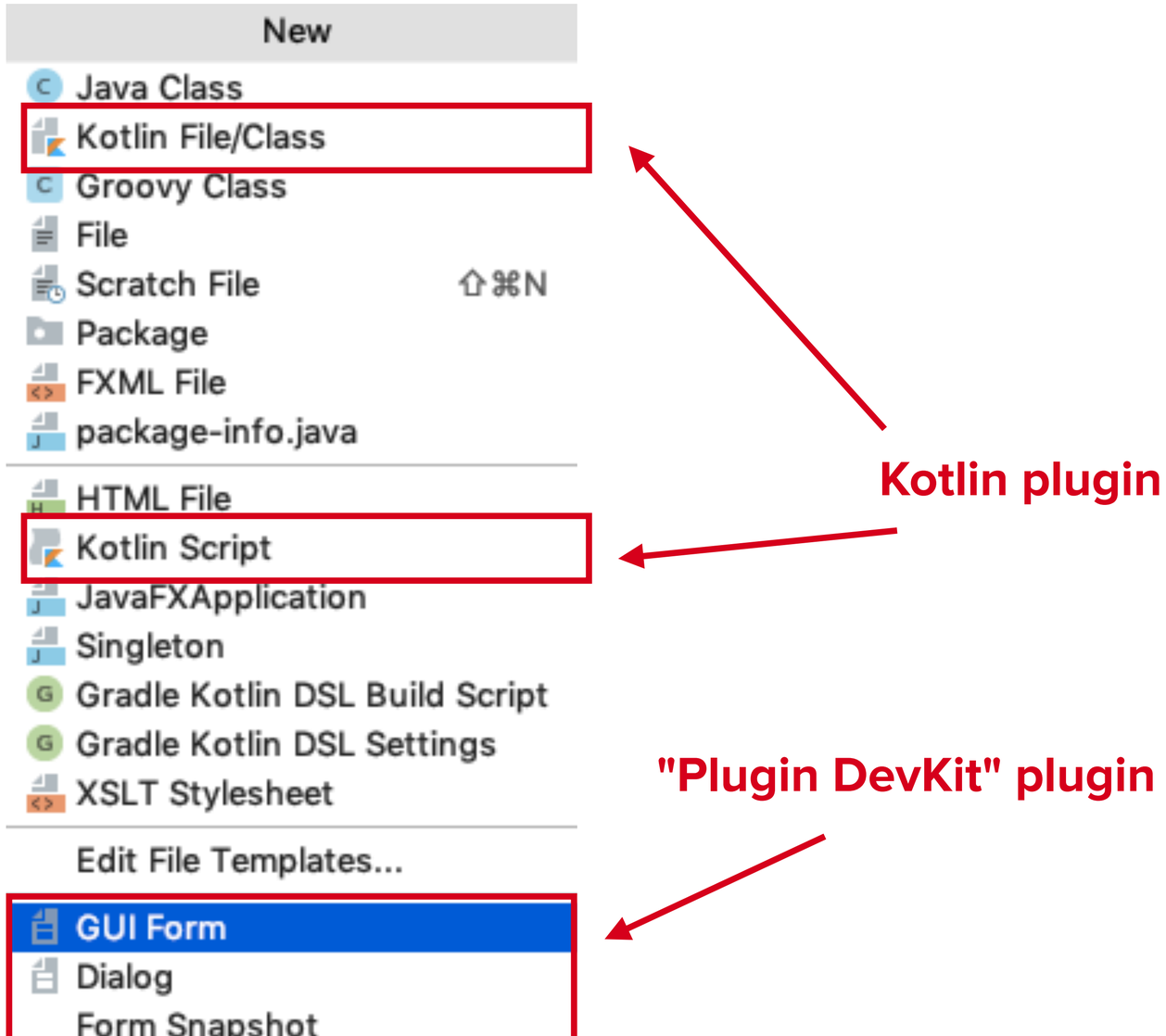
plugin.xml

```
<idea-plugin>
    ...

    <actions>
        <group description="My actions" id="MyActionGroup" text="My actions">
            <separator/>

            <action id="com.experiment.actions.OpenHelloWorldAction"
                    class="com.experiment.actions.OpenHelloWorldAction"
                    text="Show Hello world" description="Open dialog">
                <add-to-group group-id="NewGroup" anchor="last"/>
            </action>
        </group>
    </actions>

    ...
<idea-plugin>
```

Actions


New
Java Class
Kotlin File/Class
Groovy Class
File
Scratch File ⇧⌘N
Package
FXML File
package-info.java
HTML File
Kotlin Script
JavaFXApplication
Singleton
Gradle Kotlin DSL Build Script
Gradle Kotlin DSL Settings
XSLT Stylesheet
Edit File Templates...
GUI Form
Dialog
Form Snapshot

Actions



Kotlin plugin

"Plugin DevKit" plugin

Как создаются Action-s

```
class OpenHelloWorldAction : AnAction() {

    override fun actionPerformed(actionEvent: AnActionEvent) {
        val project = actionEvent.project

        Messages.showMessageDialog(
            project,
            message: "Hello world!",
            title: "Greeting",
            Messages.getInformationIcon()
        )
    }

}
```

Как создаются Action-s

```
class OpenHelloWorldAction : AnAction() {

    override fun actionPerformed(actionEvent: AnActionEvent) {
        val project = actionEvent.project

        Messages.showMessageDialog(
            project,
            message: "Hello world!",
            title: "Greeting",
            Messages.getInformationIcon()
        )
    }

}
```

Как создаются Action-s

```
class OpenHelloWorldAction : AnAction() {

    override fun actionPerformed(actionEvent: AnActionEvent) {
        val project = actionEvent.project

        Messages.showMessageDialog(
            project,
            message: "Hello world!",
            title: "Greeting",
            Messages.getInformationIcon()
        )
    }

}
```

Как создаются Action-s

```
class OpenHelloWorldAction : AnAction() {

    override fun actionPerformed(actionEvent: AnActionEvent) {
        val project = actionEvent.project

        Messages.showMessageDialog(
            project,
            message: "Hello world!",
            title: "Greeting",
            Messages.getInformationIcon()
        )
    }

}
```

Как создаются Action-s

```
class OpenHelloWorldAction : AnAction() {

    override fun actionPerformed(actionEvent: AnActionEvent) {
        val project = actionEvent.project

        Messages.showMessageDialog(
            project,
            message: "Hello world!",
            title: "Greeting",
            Messages.getInformationIcon()
        )
    }

}
```

Как создаются Action-s

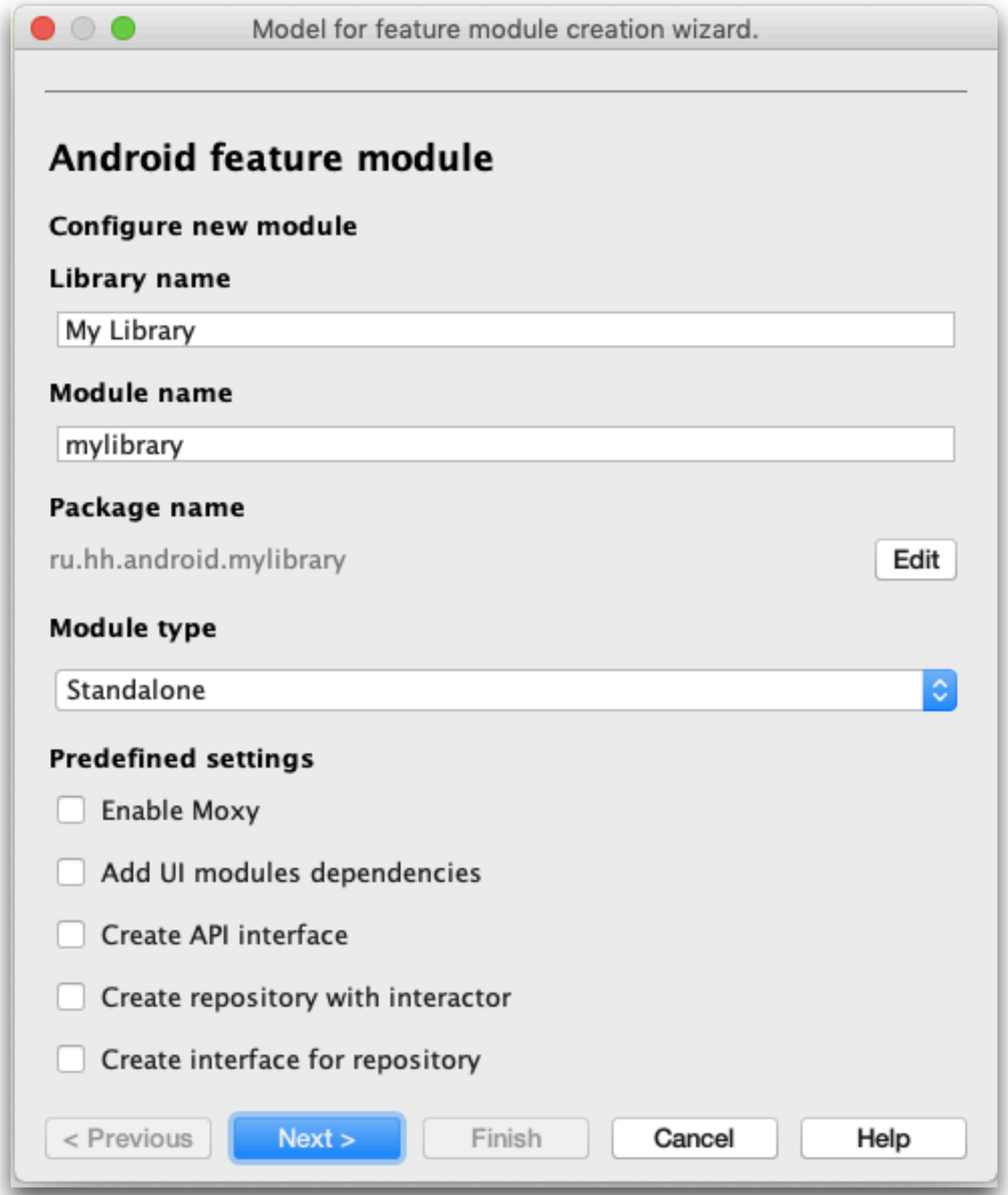
```
class OpenHelloWorldAction : AnAction() {

    ...

    override fun update(e: AnActionEvent) {
        super.update(e)
        // TODO – Here we can update our action (for example, disable it)
    }

    override fun beforeActionPerformedUpdate(e: AnActionEvent) {
        super.beforeActionPerformedUpdate(e)
        // TODO – This method calls right before 'actionPerformed'
    }

}
```

Как создаются Action-s

```
class OpenHelloWorldAction : AnAction() {

    ...

    override fun update(e: AnActionEvent) {
        super.update(e)
        // TODO - Here we can update our action (for example, disable it)
    }

    override fun beforeActionPerformedUpdate(e: AnActionEvent) {
        super.beforeActionPerformedUpdate(e)
        // TODO - This method calls right before 'actionPerformed'
    }

}
```

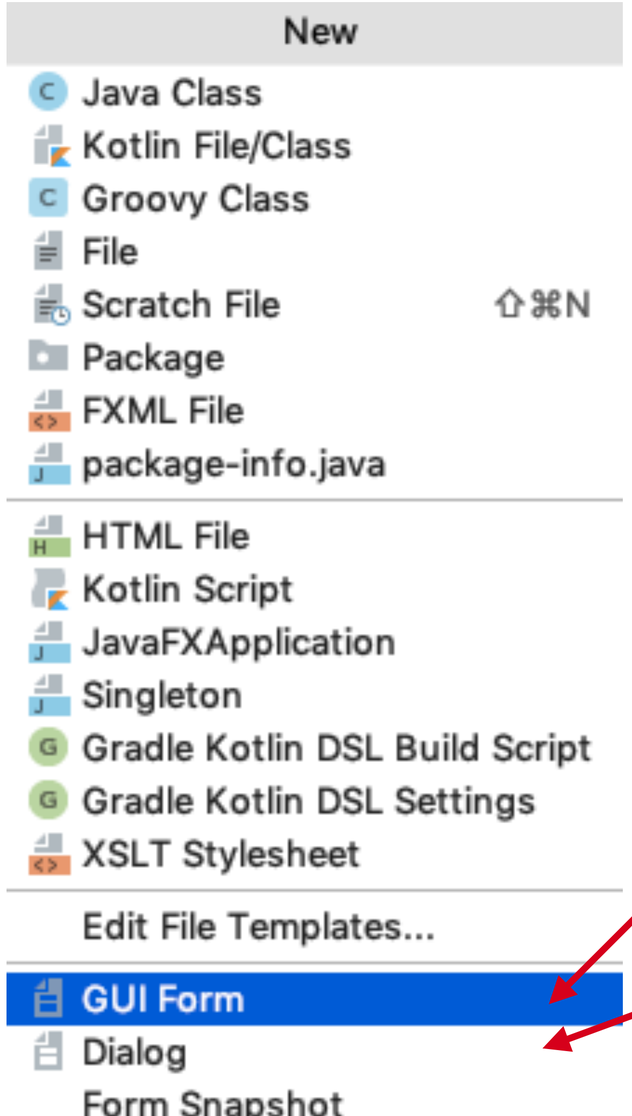
Как создаются Action-s

```
class OpenHelloWorldAction : AnAction() {

    ...

    override fun update(e: AnActionEvent) {
        super.update(e)
        // TODO - Here we can update our action (for example, disable it)
    }

    override fun beforeActionPerformedUpdate(e: AnActionEvent) {
        super.beforeActionPerformedUpdate(e)
        // TODO - This method calls right before 'actionPerformed'
    }

}
```



Если нужен свой дизайн диалогов - придется потрудиться

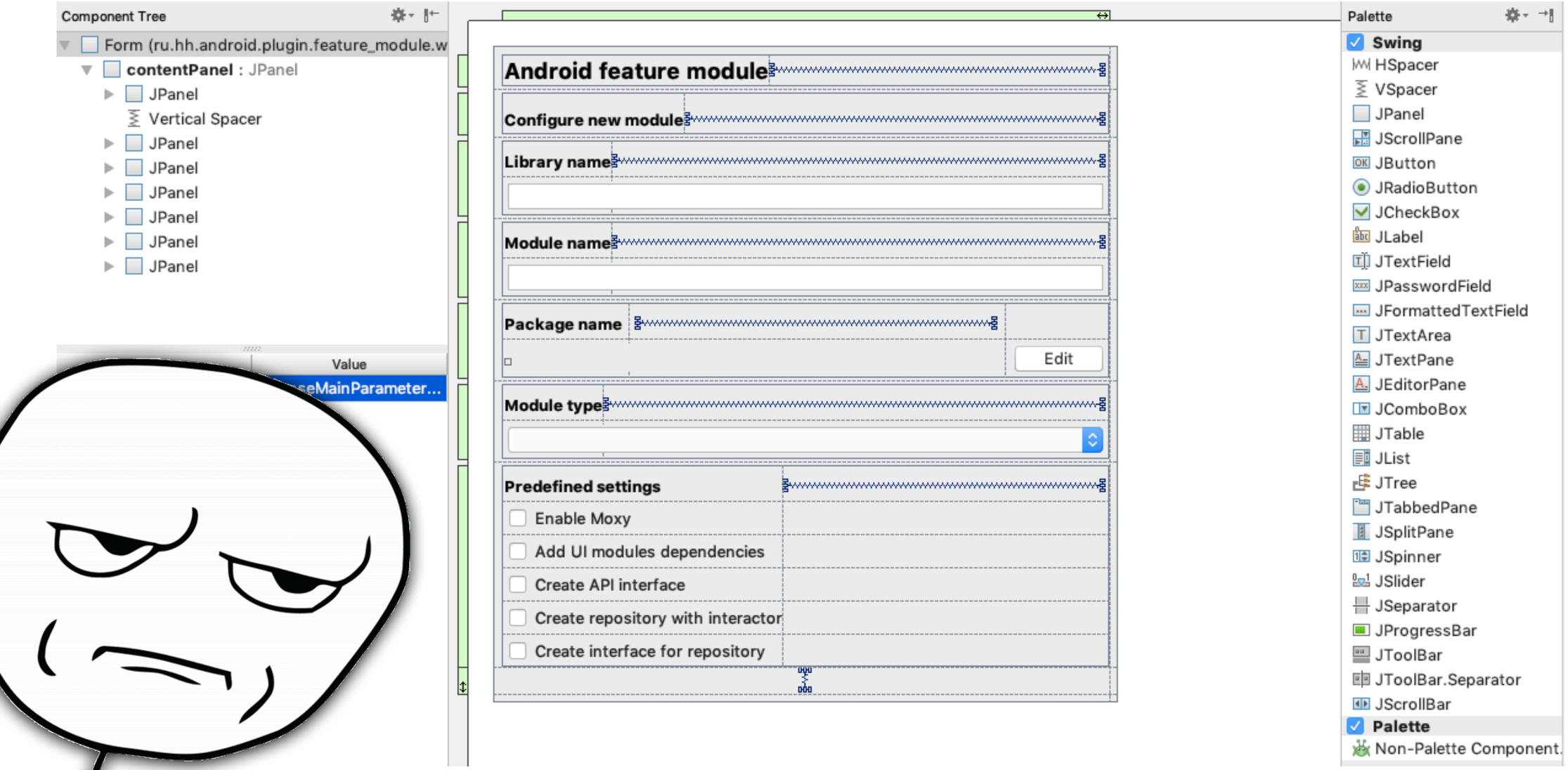
Дизайнер форм



Пустая форма

Диалог с двумя кнопками

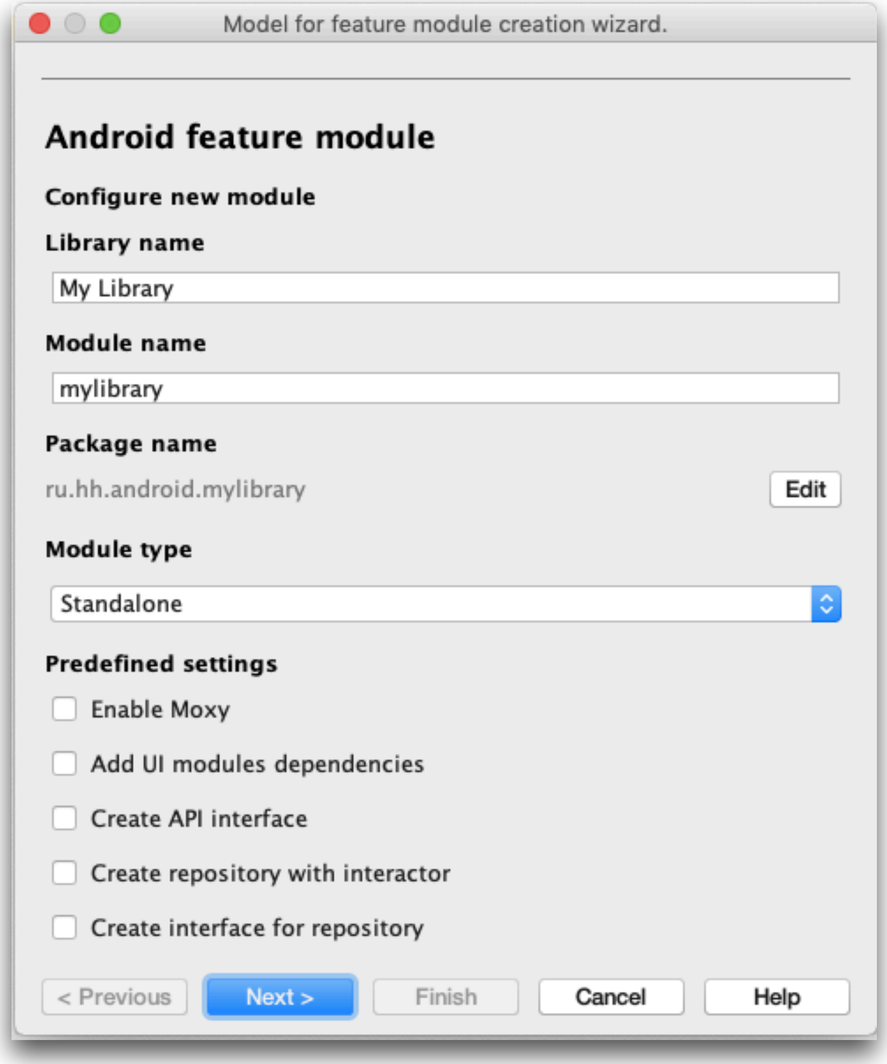
Дизайнер форм



Дизайнер форм

```
<?xml version="1.0" encoding="UTF-8"?>
<form xmlns="http://www.intellij.com/uidesigner/form/" version="1" bind-to-class="
ru.hh.android.plugin.feature_module.wizard.step.choose_main_parameters.ChooseMainParametersWizardStep">
  <grid id="27dc6" binding="contentPanel" layout-manager="GridLayoutManager" row-count="8" column-count="1" same-size-horizontally="false"
  same-size-vertically="false" hgap="-1" vgap="-1">
    <margin top="5" left="5" bottom="5" right="5"/>
    <constraints>
      <xy x="20" y="20" width="508" height="534"/>
    </constraints>
    <properties/>
    <border type="none"/>
    <children>
      <grid id="a329e" layout-manager="GridLayoutManager" row-count="1" column-count="2" same-size-horizontally="false" same-size-vertically="
      false" hgap="-1" vgap="-1">
        <margin top="0" left="0"Ащк bottom="0" right="0"/>
        <constraints>
          <grid row="0" column="0" row-span="1" col-span="1" vsize-policy="3" hsize-policy="3" anchor="0" fill="3" indent="0"
          use-parent-layout="false"/>
        </constraints>
        <properties/>
        <border type="none"/>
        <children>
          <component id="24ecb" class="javax.swing.JLabel">
            <constraints>
              <grid row="0" column="0" row-span="1" col-span="1" vsize-policy="0" hsize-policy="0" anchor="8" fill="0" indent="0"
              use-parent-layout="false"/>
            </constraints>
            <properties>
              <font size="18" style="1"/>
              <text value="Android feature module"/>
            </properties>
          </component>
          <hspacer id="e9e5c">
            <constraints>
              <grid row="0" column="1" row-span="1" col-span="1" vsize-policy="1" hsize-policy="6" anchor="0" fill="1" indent="0"
```

UI form = .form + .java (.kt)




```
class FormBoundClass : DialogWrapper() {

    private lateinit var contentPanel: JPanel
    private lateinit var libraryNameTextField: JTextField
    private lateinit var moduleNameTextField: JTextField


    override fun createCenterPanel(): JComponent? {
        return contentPanel
    }


    private fun createUIComponents() {
        libraryNameTextField = JTextField()
        libraryNameTextField.onTextChanged {
            System.out.println("Text changed!")
        }
    }

}
```

Резюмируем основы

1. Intellij IDEA CE + Plugin Dev Kit + Java

Резюмируем основы

1. Intellij IDEA CE + Plugin Dev Kit + Java
2. **gradle-intellij-plugin сильно упрощает жизнь**

Резюмируем основы

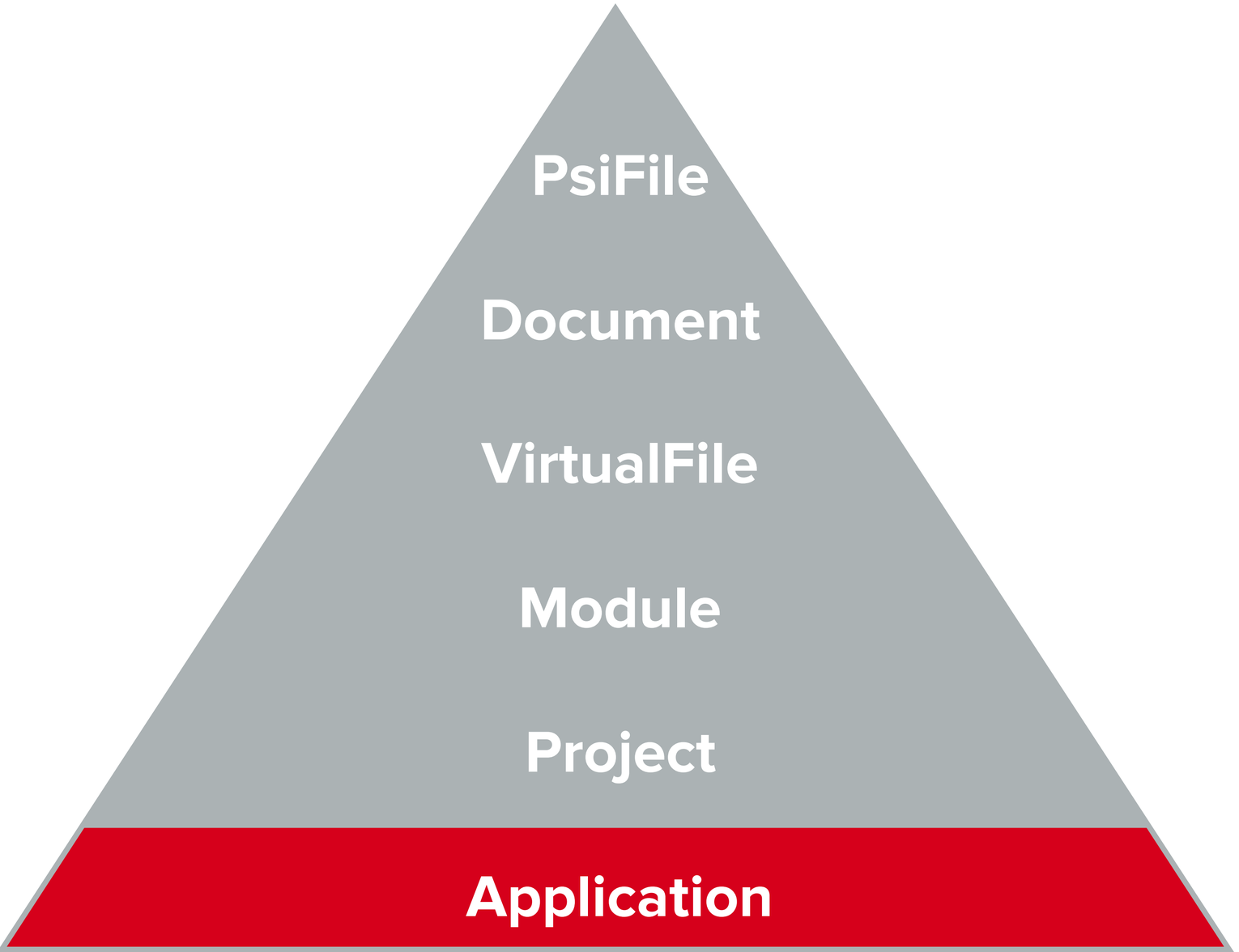
1. Intellij IDEA CE + Plugin Dev Kit + Java
2. gradle-intellij-plugin сильно упрощает жизнь
3. **Не пишите свой UI, если это не необходимо**

Резюмируем основы

1. Intellij IDEA CE + Plugin Dev Kit + Java
2. gradle-intellij-plugin сильно упрощает жизнь
3. Не пишите свой UI, если это не необходимо
4. **В плагине может быть сколько угодно Action-ов**

Внутренности IDEA: компоненты, PSI

Application



Project

PsiFile

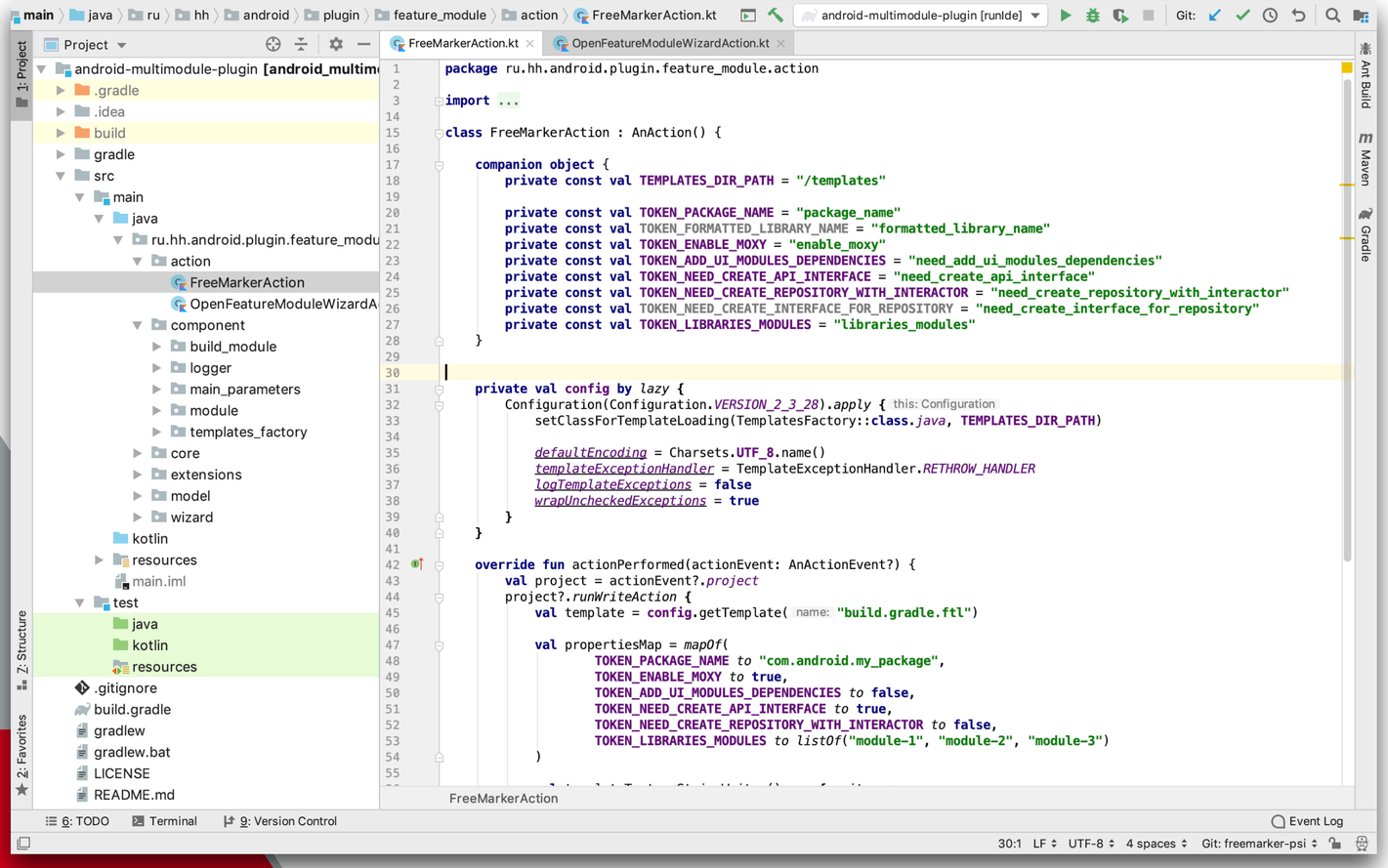
Document

VirtualFile

Module

Project

Application



Module

PsiFile

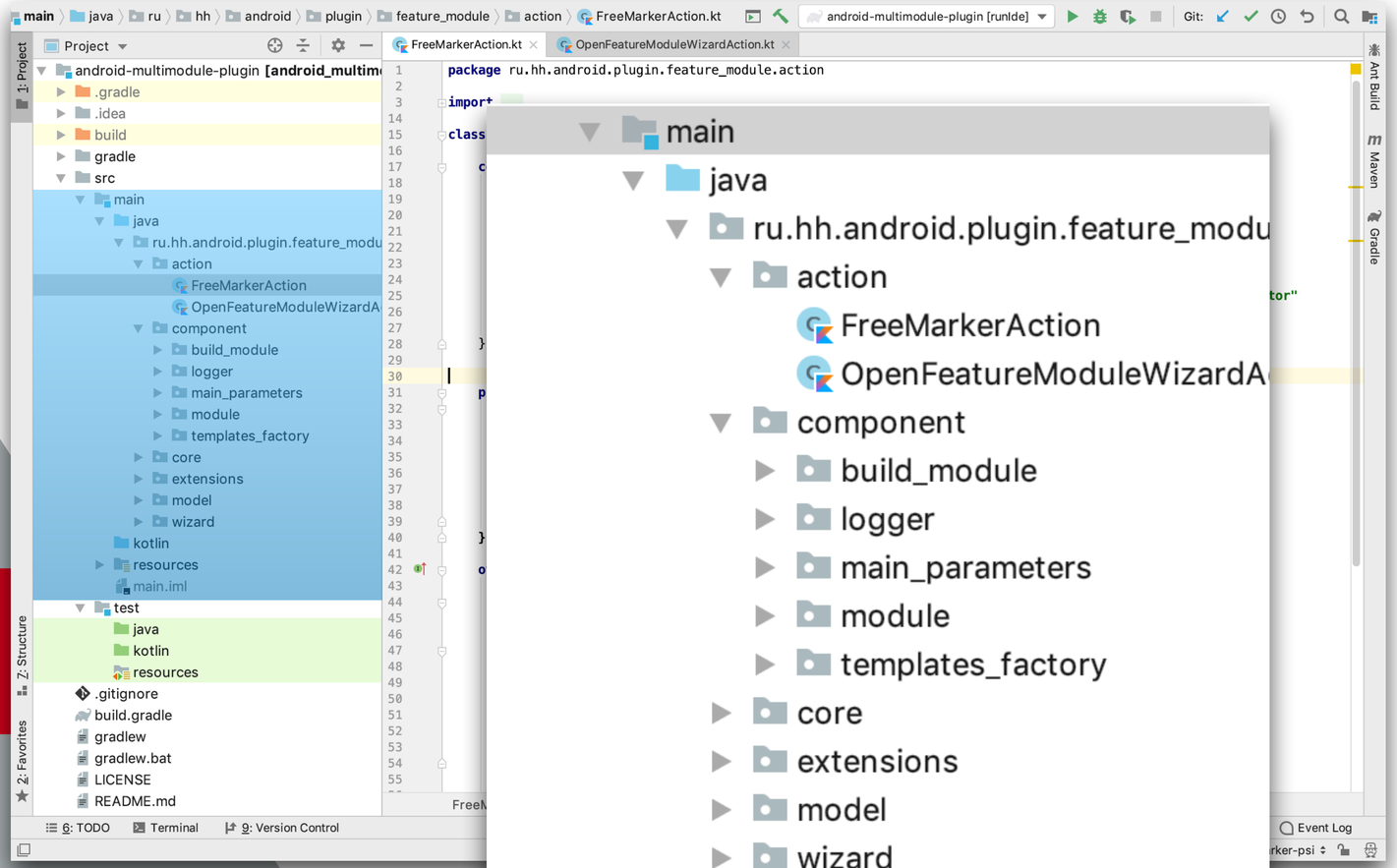
Document

VirtualFile

Module

Project

Application



VirtualFile

PsiFile

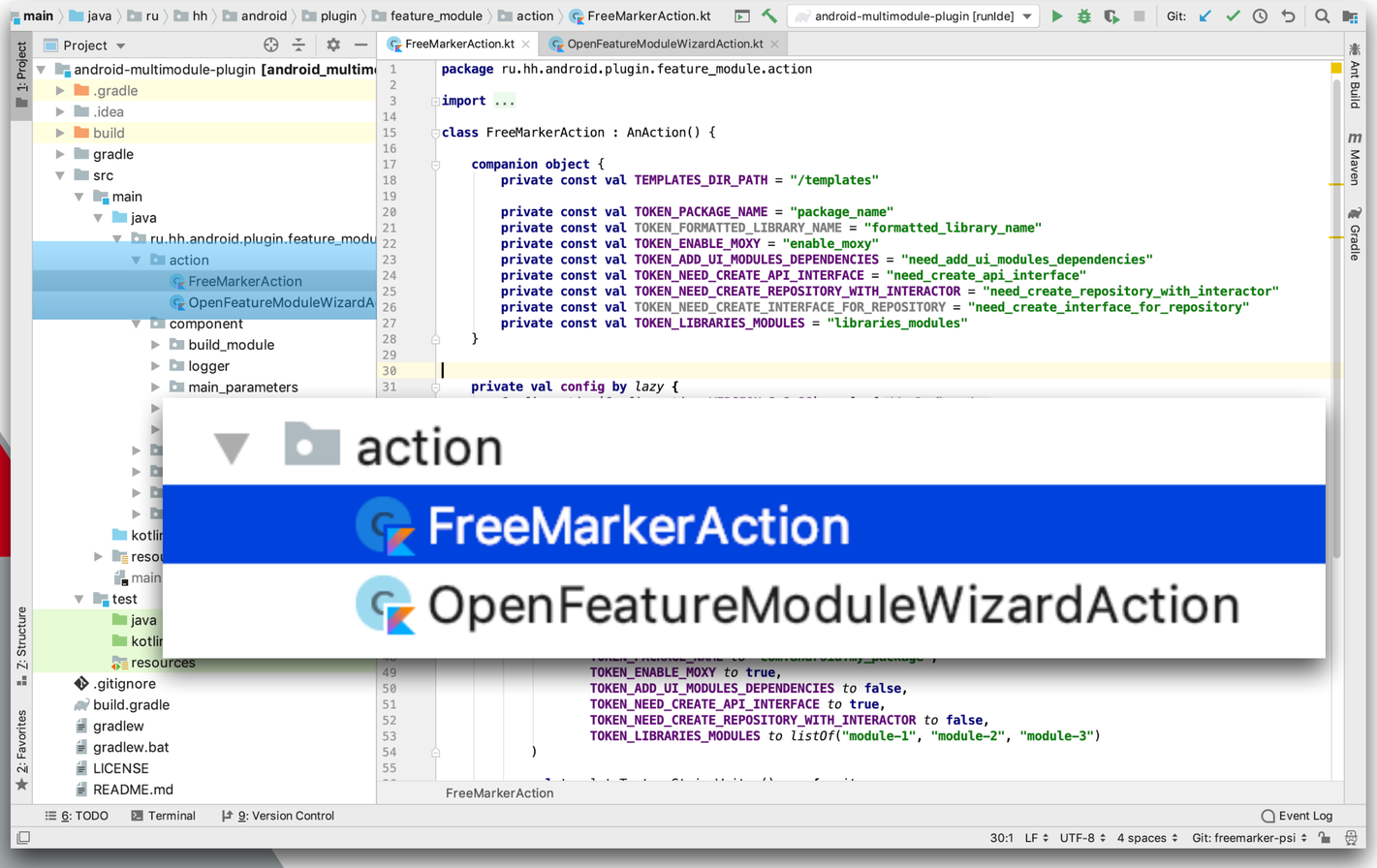
Document

VirtualFile

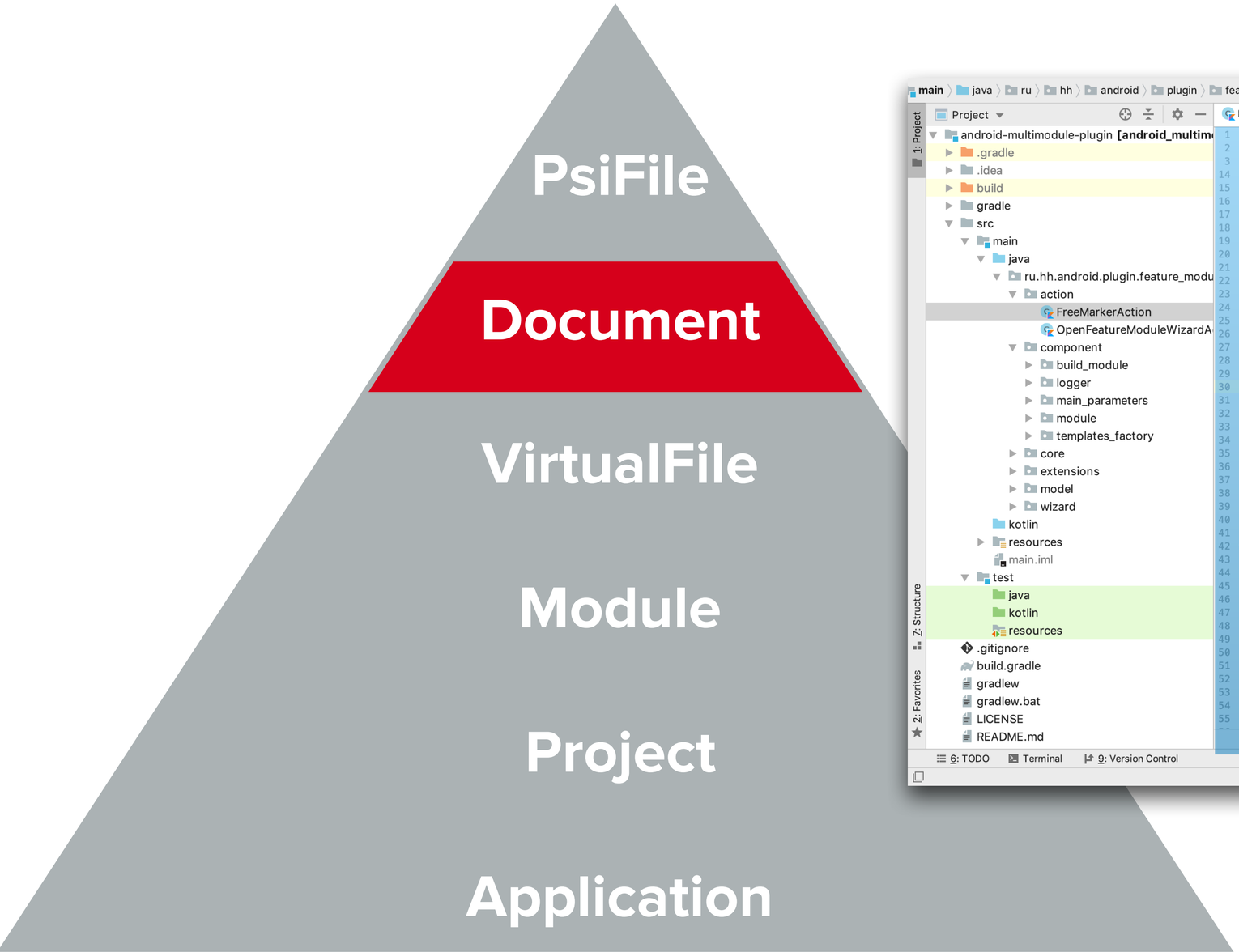
Module

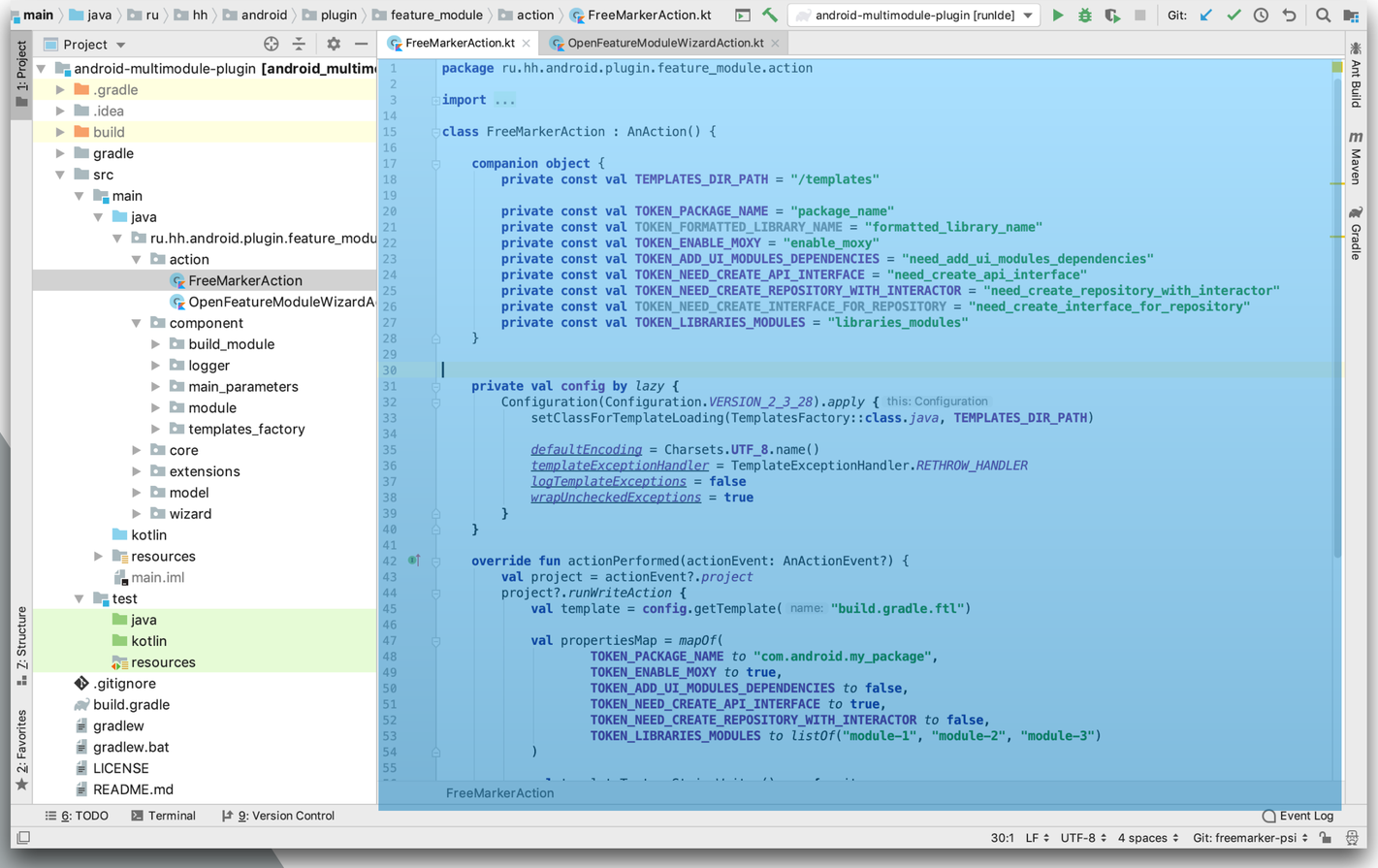
Project

Application



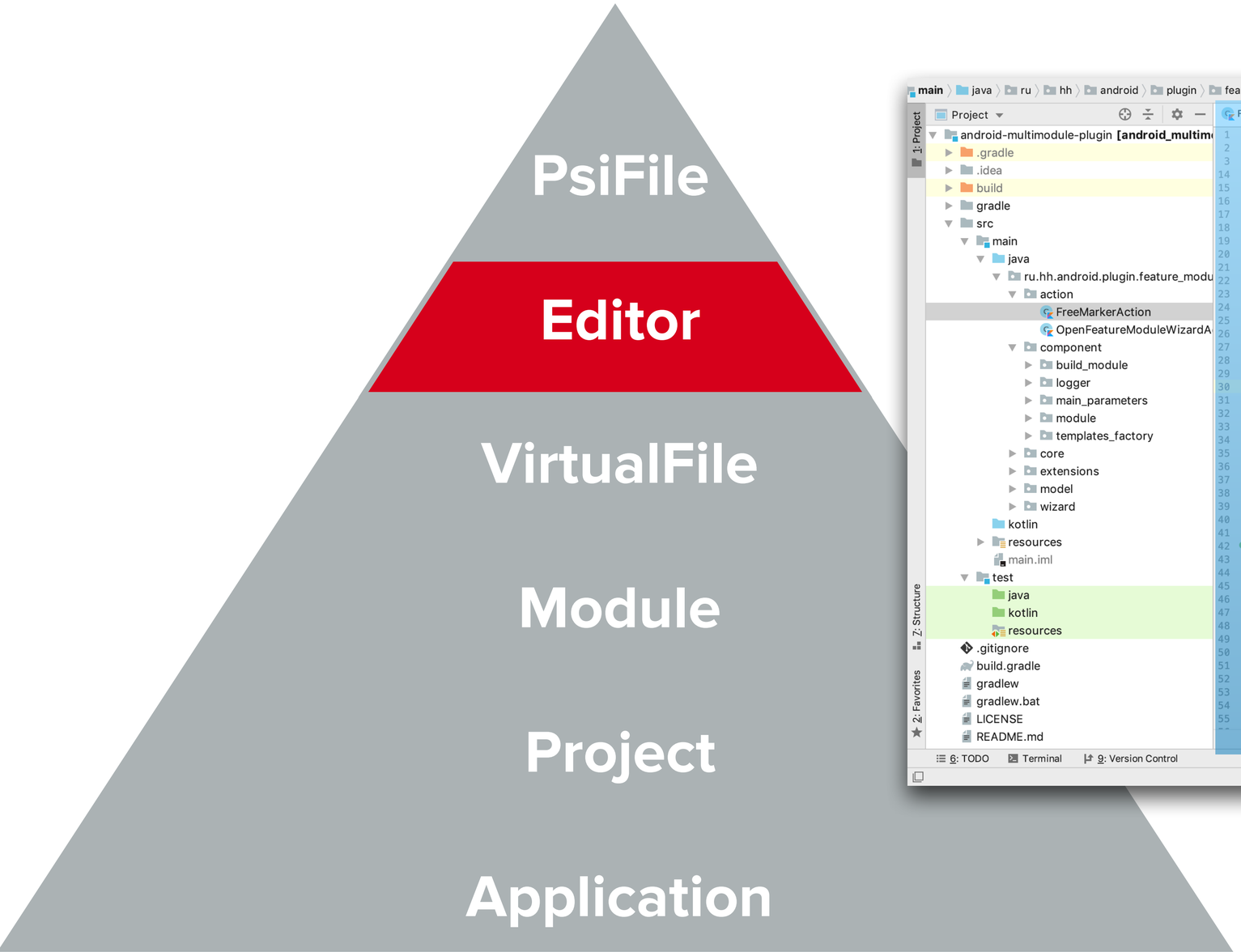
Document

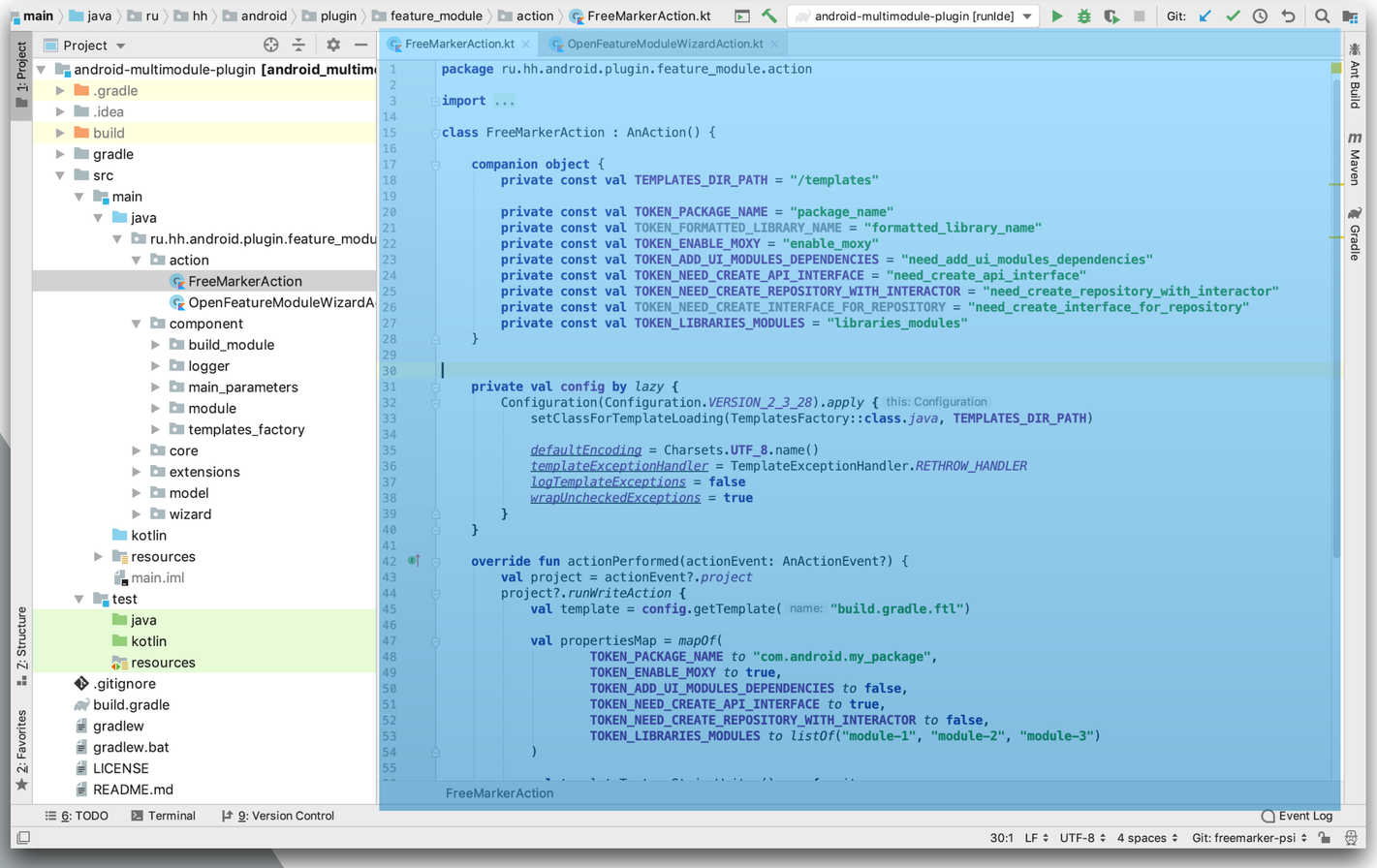




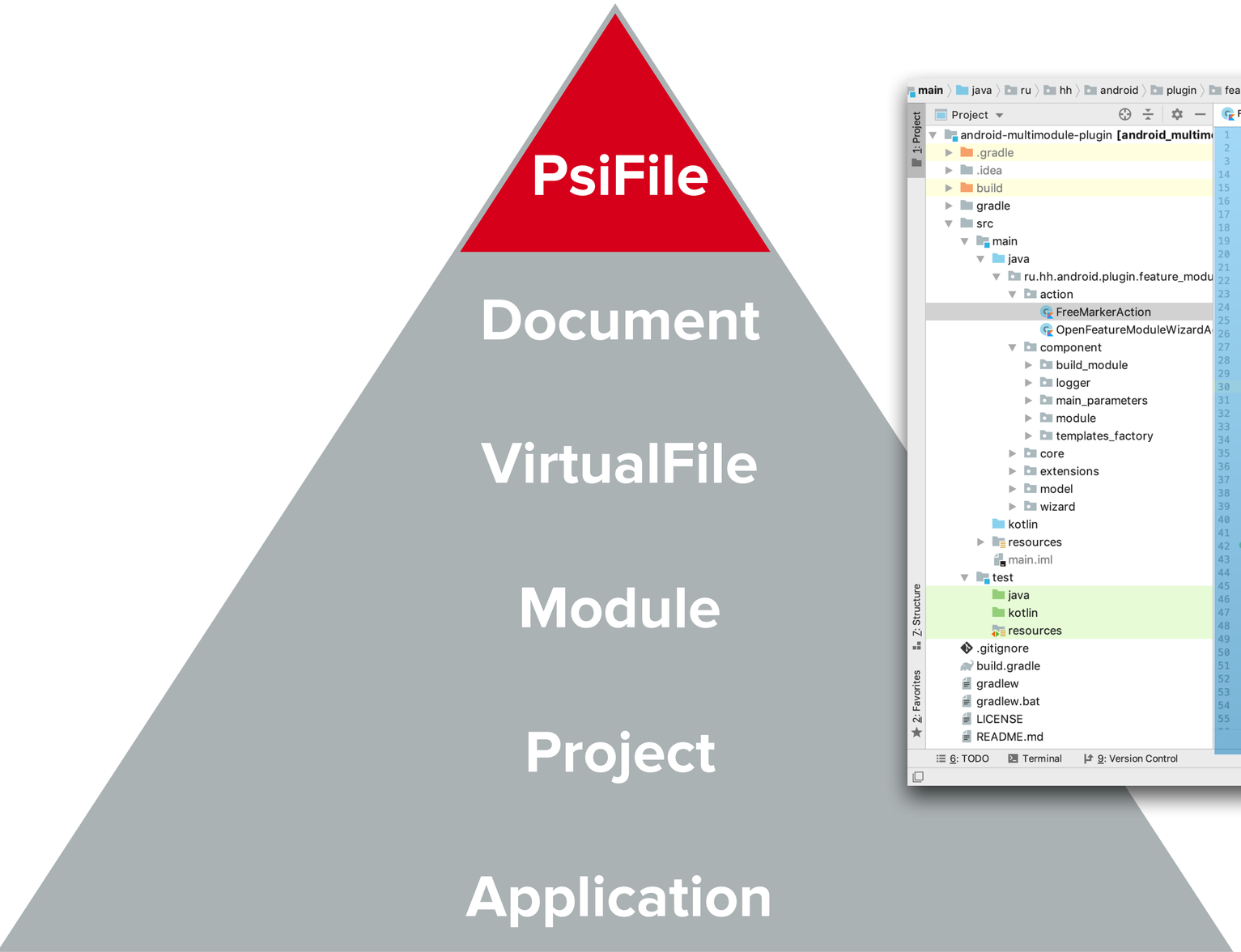

```
package ru.hh.android.plugin.feature_module.action

import ...

class FreeMarkerAction : AnAction() {

    companion object {
        private const val TEMPLATES_DIR_PATH = "/templates"

        private const val TOKEN_PACKAGE_NAME = "package_name"
        private const val TOKEN_FORMATTED_LIBRARY_NAME = "formatted_library_name"
        private const val TOKEN_ENABLE_MOXY = "enable_moxy"
        private const val TOKEN_ADD_UI_MODULES_DEPENDENCIES = "need_add_ui_modules_dependencies"
        private const val TOKEN_NEED_CREATE_API_INTERFACE = "need_create_api_interface"
        private const val TOKEN_NEED_CREATE_REPOSITORY_WITH_INTERACTOR = "need_create_repository_with_interactor"
        private const val TOKEN_NEED_CREATE_INTERFACE_FOR_REPOSITORY = "need_create_interface_for_repository"
        private const val TOKEN_LIBRARIES_MODULES = "libraries_modules"
    }

    private val config by lazy {
        Configuration(Configuration.VERSION_2_3_28).apply { this: Configuration
            setClassForTemplateLoading(TemplatesFactory::class.java, TEMPLATES_DIR_PATH)

            defaultEncoding = Charsets.UTF_8.name()
            templateExceptionHandler = TemplateExceptionHandler.RETHROW_HANDLER
            logTemplateExceptions = false
            wrapUncheckedExceptions = true
        }
    }

    override fun actionPerformed(actionEvent: AnActionEvent?) {
        val project = actionEvent?.project
        project?.runWriteAction {
            val template = config.getTemplate( name: "build.gradle.ftl")

            val propertiesMap = mapOf(
                    TOKEN_PACKAGE_NAME to "com.android.my_package",
                    TOKEN_ENABLE_MOXY to true,
                    TOKEN_ADD_UI_MODULES_DEPENDENCIES to false,
                    TOKEN_NEED_CREATE_API_INTERFACE to true,
                    TOKEN_NEED_CREATE_REPOSITORY_WITH_INTERACTOR to false,
                    TOKEN_LIBRARIES_MODULES to listOf("module-1", "module-2", "module-3")
            )
```

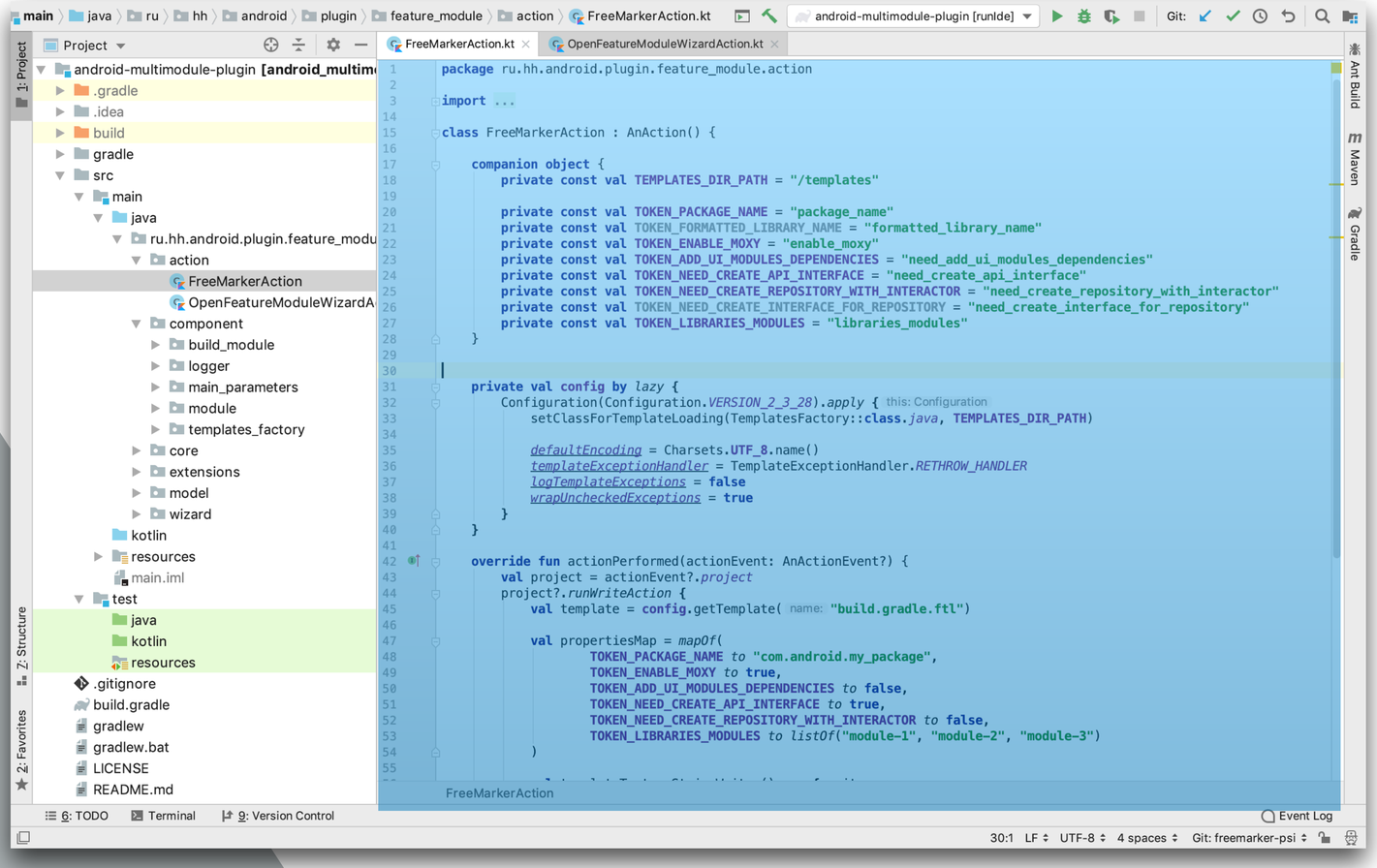
Editor


PsiFile
Editor
VirtualFile
Module
Project
Application


main
java
ru
hh
android
plugin
feature_module
action
FreeMarkerAction.kt
android-multimodule-plugin [runIde]
Git:
Project
FreeMarkerAction.kt
OpenFeatureModuleWizardAction.kt
android-multimodule-plugin [android_multim
.gradle
.idea
build
gradle
src
main
java
ru.hh.android.plugin.feature_modu
action
FreeMarkerAction
OpenFeatureModuleWizardA
component
build_module
logger
main_parameters
module
templates_factory
core
extensions
model
wizard
kotlin
resources
main.iml
test
java
kotlin
resources
.gitignore
build.gradle
gradlew
gradlew.bat
LICENSE
README.md
package ru.hh.android.plugin.feature_module.action
import ...
class FreeMarkerAction : AnAction() {
companion object {
private const val TEMPLATES_DIR_PATH = "/templates"
private const val TOKEN_PACKAGE_NAME = "package_name"
private const val TOKEN_FORMATTED_LIBRARY_NAME = "formatted_library_name"
private const val TOKEN_ENABLE_MOXY = "enable_moxy"
private const val TOKEN_ADD_UI_MODULES_DEPENDENCIES = "need_add_ui_modules_dependencies"
private const val TOKEN_NEED_CREATE_API_INTERFACE = "need_create_api_interface"
private const val TOKEN_NEED_CREATE_REPOSITORY_WITH_INTERACTOR = "need_create_repository_with_interactor"
private const val TOKEN_NEED_CREATE_INTERFACE_FOR_REPOSITORY = "need_create_interface_for_repository"
private const val TOKEN_LIBRARIES_MODULES = "libraries_modules"
}
private val config by lazy {
Configuration(Configuration.VERSION_2_3_28).apply { this: Configuration
setClassForTemplateLoading(TemplatesFactory::class.java, TEMPLATES_DIR_PATH)
defaultEncoding = Charsets.UTF_8.name()
templateExceptionHandler = TemplateExceptionHandler.RETHROW_HANDLER
logTemplateExceptions = false
wrapUncheckedExceptions = true
}
}
override fun actionPerformed(actionEvent: AnActionEvent?) {
val project = actionEvent?.project
project?.runWriteAction {
val template = config.getTemplate(name: "build.gradle.ftl")
val propertiesMap = mapOf(
TOKEN_PACKAGE_NAME to "com.android.my_package",
TOKEN_ENABLE_MOXY to true,
TOKEN_ADD_UI_MODULES_DEPENDENCIES to false,
TOKEN_NEED_CREATE_API_INTERFACE to true,
TOKEN_NEED_CREATE_REPOSITORY_WITH_INTERACTOR to false,
TOKEN_LIBRARIES_MODULES to listOf("module-1", "module-2", "module-3")
)
FreeMarkerAction
6: TODO
Terminal
9: Version Control
Event Log
30:1
LF
UTF-8
4 spaces
Git: freemarker-psi
1: Project
2: Favorites
7: Structure
Ant Build
Maven
Gradle

PsiFile


PsiFile
Document
VirtualFile
Module
Project
Application


main java ru hh android plugin feature_module action FreeMarkerAction.kt
android-multimodule-plugin [runIde]
Git:
Project
android-multimodule-plugin [android_multim
.gradle
.idea
build
gradle
src
main
java
ru.hh.android.plugin.feature_modu
action
FreeMarkerAction
OpenFeatureModuleWizardA
component
build_module
logger
main_parameters
module
templates_factory
core
extensions
model
wizard
kotlin
resources
main.iml
test
java
kotlin
resources
.gitignore
build.gradle
gradlew
gradlew.bat
LICENSE
README.md
FreeMarkerAction.kt
OpenFeatureModuleWizardAction.kt
package ru.hh.android.plugin.feature_module.action
import ...
class FreeMarkerAction : AnAction() {
companion object {
private const val TEMPLATES_DIR_PATH = "/templates"
private const val TOKEN_PACKAGE_NAME = "package_name"
private const val TOKEN_FORMATTED_LIBRARY_NAME = "formatted_library_name"
private const val TOKEN_ENABLE_MOXY = "enable_moxy"
private const val TOKEN_ADD_UI_MODULES_DEPENDENCIES = "need_add_ui_modules_dependencies"
private const val TOKEN_NEED_CREATE_API_INTERFACE = "need_create_api_interface"
private const val TOKEN_NEED_CREATE_REPOSITORY_WITH_INTERACTOR = "need_create_repository_with_interactor"
private const val TOKEN_NEED_CREATE_INTERFACE_FOR_REPOSITORY = "need_create_interface_for_repository"
private const val TOKEN_LIBRARIES_MODULES = "libraries_modules"
}
private val config by lazy {
Configuration(Configuration.VERSION_2_3_28).apply { this: Configuration
setClassForTemplateLoading(TemplatesFactory::class.java, TEMPLATES_DIR_PATH)
defaultEncoding = Charsets.UTF_8.name()
templateExceptionHandler = TemplateExceptionHandler.RETHROW_HANDLER
logTemplateExceptions = false
wrapUncheckedExceptions = true
}
}
override fun actionPerformed(actionEvent: AnActionEvent?) {
val project = actionEvent?.project
project?.runWriteAction {
val template = config.getTemplate(name: "build.gradle.ftl")
val propertiesMap = mapOf(
TOKEN_PACKAGE_NAME to "com.android.my_package",
TOKEN_ENABLE_MOXY to true,
TOKEN_ADD_UI_MODULES_DEPENDENCIES to false,
TOKEN_NEED_CREATE_API_INTERFACE to true,
TOKEN_NEED_CREATE_REPOSITORY_WITH_INTERACTOR to false,
TOKEN_LIBRARIES_MODULES to listOf("module-1", "module-2", "module-3")
)
FreeMarkerAction
6: TODO
Terminal
9: Version Control
Event Log
30:1 LF UTF-8 4 spaces Git: freemarker-psi
1: Project
7: Structure
2: Favorites
Ant Build
Maven
Gradle

PSI = Program Structure Interface

Важно! Если не учесть структуру PSI, плагин будет работать не так, как вы хотите

PSI

```
package com.experiment;

import javax.inject.Inject;

class SomeClass {

    @Inject
    String injectedString;

    public void someMethod() {
        System.out.println(injectedString);
    }
}
```

PSI

```
package com.experiment;

import javax.inject.Inject;

class SomeClass {

    @Inject
    String injectedString;

    public void someMethod() {
        System.out.println(injectedString);
    }
}
```

PSI

Package statement

Imports

```
package com.experiment;

import javax.inject.Inject;

class SomeClass {

    @Inject
    String injectedString;

    public void someMethod() {
        System.out.println(injectedString);
    }
}
```

PSI

Package statement

Imports

Classes

```
package com.experiment;

import javax.inject.Inject;

class SomeClass {

    @Inject
    String injectedString;

    public void someMethod() {
        System.out.println(injectedString);
    }
}
```

PSI

```
package com.experiment;

import javax.inject.Inject;

class SomeClass {

    @Inject
    String injectedString;

    public void someMethod() {
        System.out.println(injectedString);
    }
}
```

Package statement

Imports

Classes

Fields

PSI

```
package com.experiment;

import javax.inject.Inject;

class SomeClass {

    @Inject
    String injectedString;

    public void someMethod() {
        System.out.println(injectedString);
    }
}
```

Package statement

Imports

Classes

Fields

Methods

PSI

```
package com.experiment;

import javax.inject.Inject;

class SomeClass {

    @Inject
    String injectedString;

    public void someMethod() {
        System.out.println(injectedString);
    }
}
```

Package statement

Imports

Classes

Fields

Methods

Annotations

PSI

```
package com.experiment;

import javax.inject.Inject;

class SomeClass {

    @Inject
    String injectedString;

    public void someMethod() {
        System.out.println(injectedString);
    }
}
```

Package statement

Imports

Classes

Fields

Methods

Annotations

Keywords

PSI

```
package com.experiment;

import javax.inject.Inject;

class SomeClass {

    @Inject
    String injectedString;

    public void someMethod() {
        System.out.println(injectedString);
    }
}
```

Package statement

Imports

Classes

Fields

Methods

Annotations

Keywords

Types

PSI

```
package com.experiment;

import javax.inject.Inject;

class SomeClass {

    @Inject
    String injectedString;

    public void someMethod() {
        System.out.println(injectedString);
    }
}
```

Package statement

Imports

Classes

Fields

Methods

Annotations

Keywords

Types

Modifiers

PSI

```
package com.experiment;

import javax.inject.Inject;

class SomeClass {

    @Inject
    String injectedString;

    public void someMethod() {
        System.out.println(injectedString);
    }
}
```

Package statement

Imports

Classes

Fields

Methods

Annotations

Keywords

Types

Modifiers

Identifiers

PSI

```java
package com.experiment;

import javax.inject.Inject;

class SomeClass {

    @Inject
    String injectedString;

    public void someMethod() {
        System.out.println(injectedString);
    }
}
```

Package statement

Imports

Classes

Fields

Methods

Annotations

Keywords

Types

Modifiers

Identifiers

References

PSI

```
package com.experiment;

import javax.inject.Inject;

class SomeClass {

    @Inject
    String injectedString;

    public void someMethod() {
        System.out.println(injectedString);
    }
}
```

Package statement

Imports

Classes

Fields

Methods

Annotations

Keywords

Types

Modifiers

Identifiers

References

Expressions

PSI

```
package com.experiment;

import javax.inject.Inject;

class SomeClass {

    @Inject
    String injectedString;

    public void someMethod() {
        System.out.println(injectedString);
    }
}
```

Package statement

Imports

Classes

Fields

Methods

Annotations

Keywords

Types

Modifiers

Identifiers

References

Expressions

Tokens

PsiElement

```
package com.experiment;

import javax.inject.Inject;

class SomeClass {

    @Inject
    String injectedString;

    public void someMethod() {
        System.out.println(injectedString);
    }
}
```

PsiPackageStatement

PsiImportList

PsiClass

PsiField

PsiMethod

PsiAnnotation

PsiKeyword

PsiTypeElement

PsiModifierList

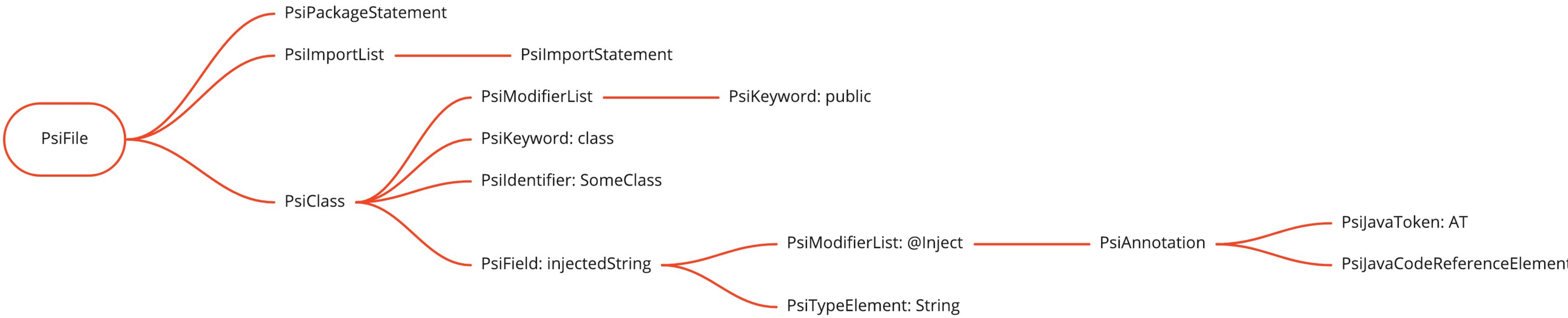
PsiIdentifier

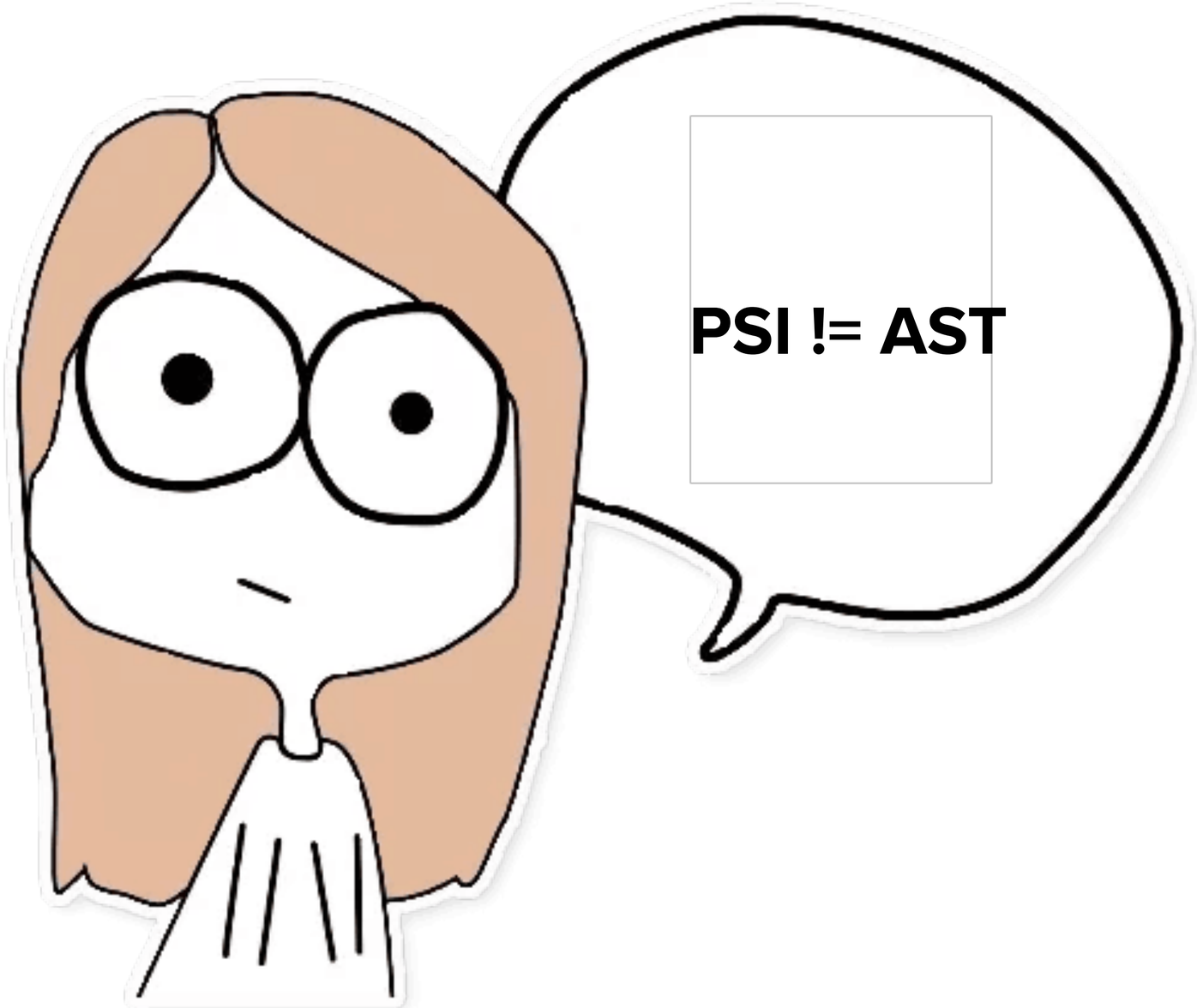
PsiReferenceList

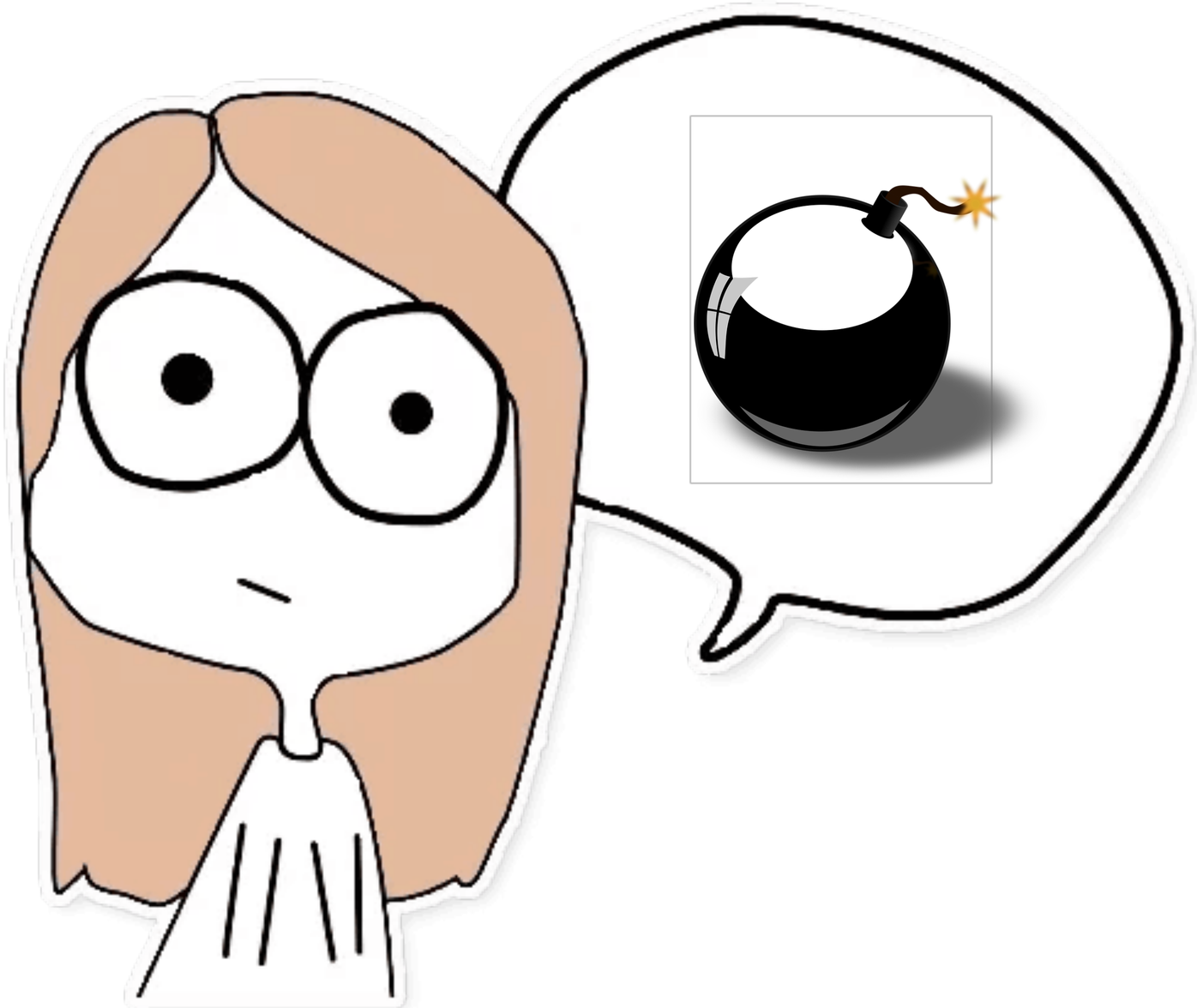
PsiExpression

PsiJavaToken

Иерархия PsiFile




PSI != AST



Важное из внутренностей

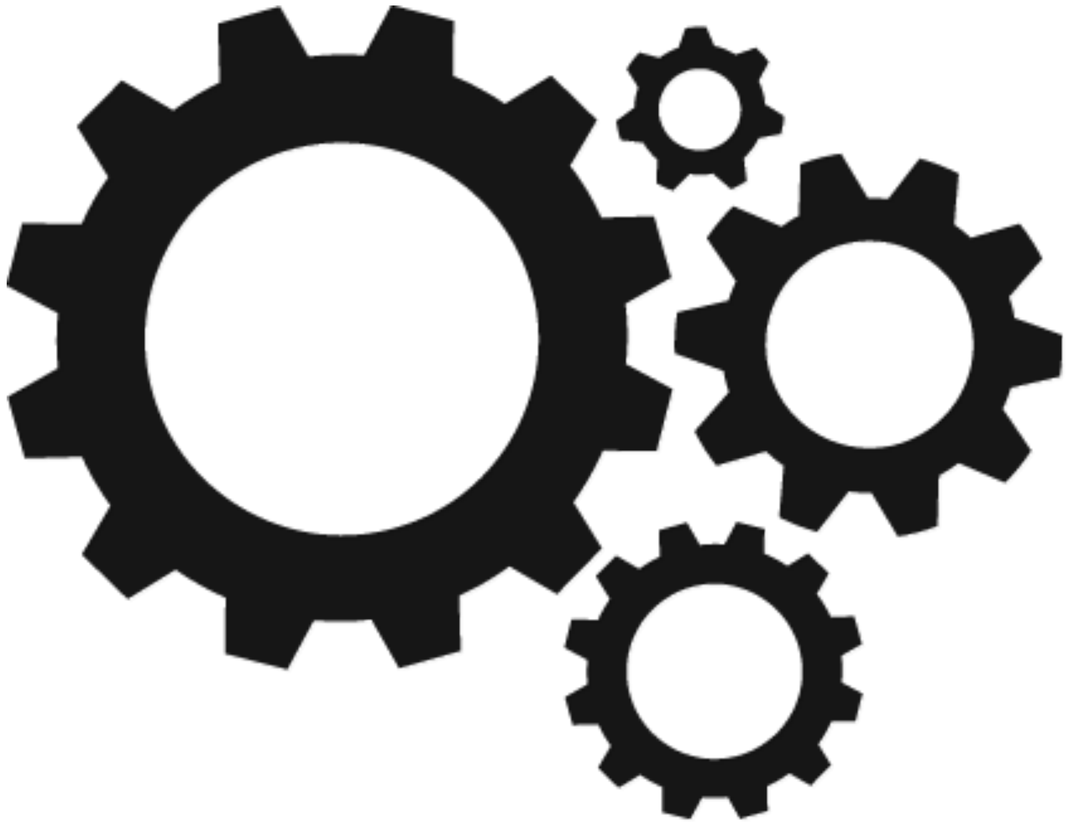
1. PSI нужен для представления программы в IDEA

Важное из внутренностей

1. PSI нужен для представления программы в IDEA
2. **PSI-структурой пронизана вся IDEA**

Важное из внутренностей

1. PSI нужен для представления программы в IDEA
2. PSI-структурой пронизана вся IDEA
3. **Для каждого языка программирования - свой PSI**



UI, DI, генерация и модификация кода

Model for feature module creation w

Android feature module

Configure new module

Library name

My Library

Module name

mylibrary

Package name

ru.hh.android.mylibrary

Module type

Standalone

Predefined settings

- [ ] Enable Moxy
- [ ] Add UI modules dependencies
- [ ] Create API interface
- [ ] Create repository with interactor
- [ ] Create interface for repository

< Previous | Next > | Finish | Ca

Model for feature module creation wizard.

- [x] feature_module
- [x] core_module

Enable All

Please select a module to see its description

< Previous | Next > | Finish | Cancel

Model for feature module creation wizard.

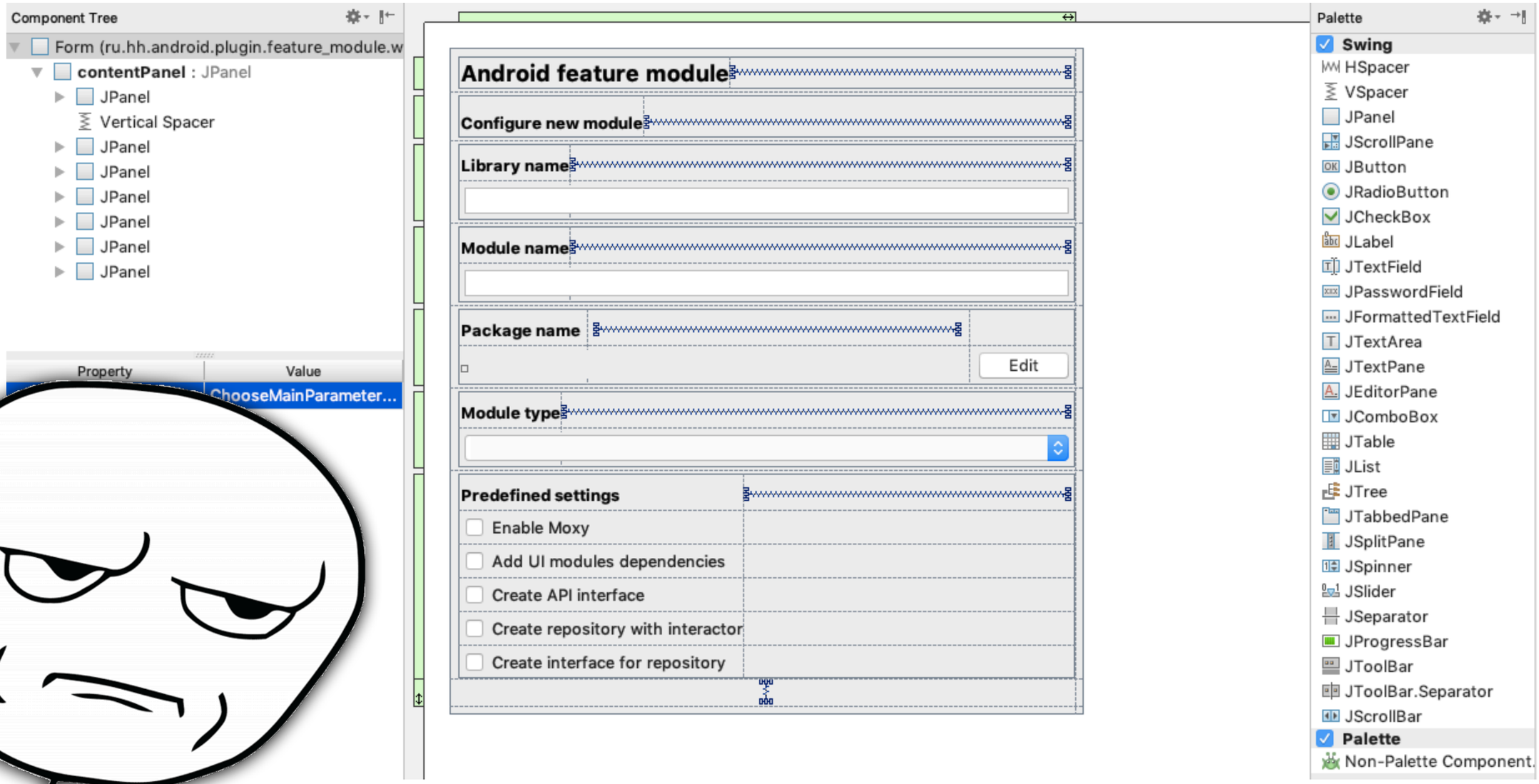
- [x] application

Enable All | Disable All

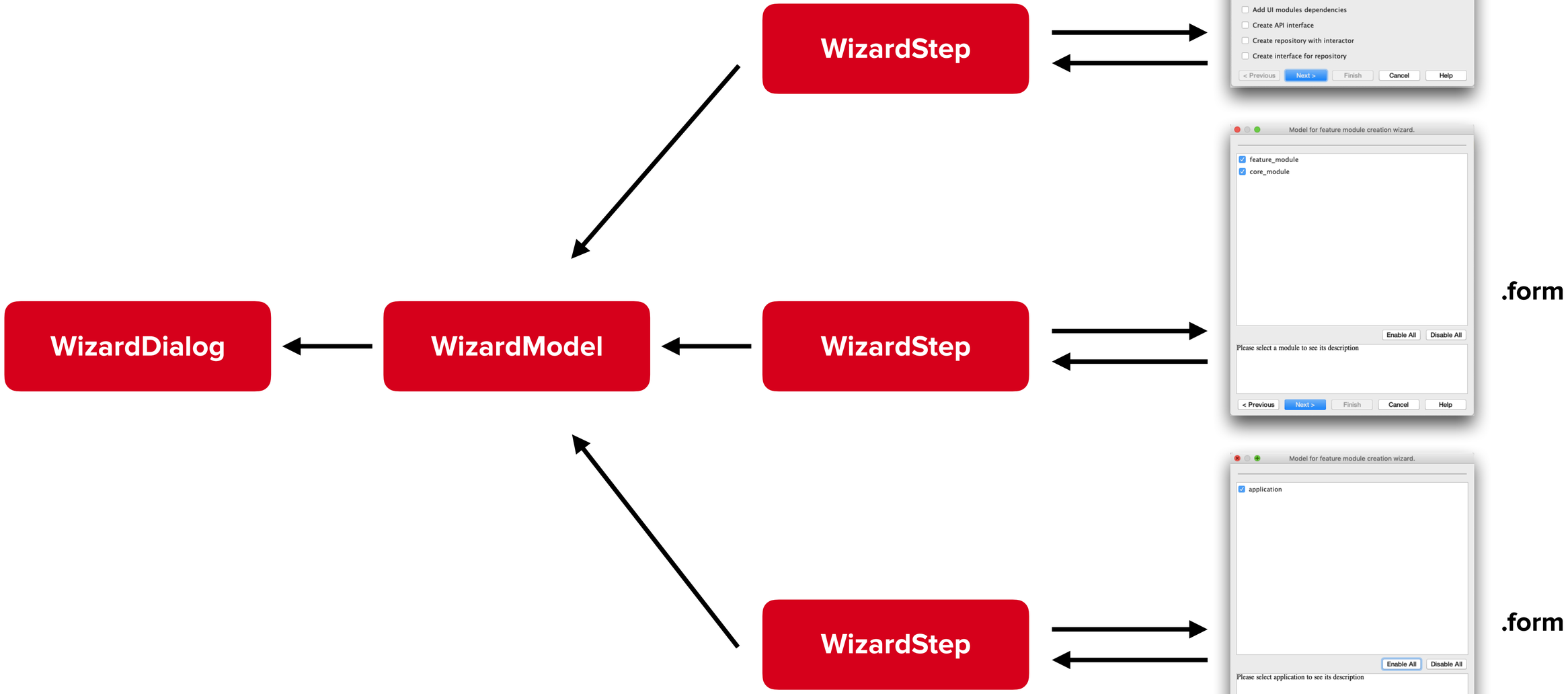
Please select application to see its description

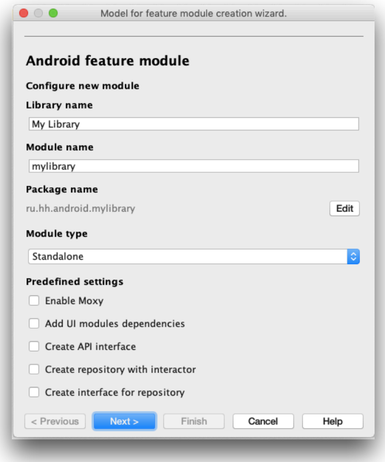
< Previous | Next > | Finish | Cancel | Help

Делать многостраничный UI — БОЛЬНО



WizardDialog





WizardDialog

```
class MyWizardDialog(
    model: MyWizardModel
): WizardDialog<MyWizardModel>(canBeParent: true, tryApplicationModal: true, model)
```

WizardDialog

```
class MyWizardDialog(
    model: MyWizardModel,
    private val onFinishButtonClickedListener: (MyWizardModel) -> Unit
): WizardDialog<MyWizardModel>(canBeParent: true, tryApplicationModal: true, model) {

    override fun onWizardGoalAchieved() {
        super.onWizardGoalAchieved()
        onFinishButtonClickedListener.invoke(myModel)
    }

}
```

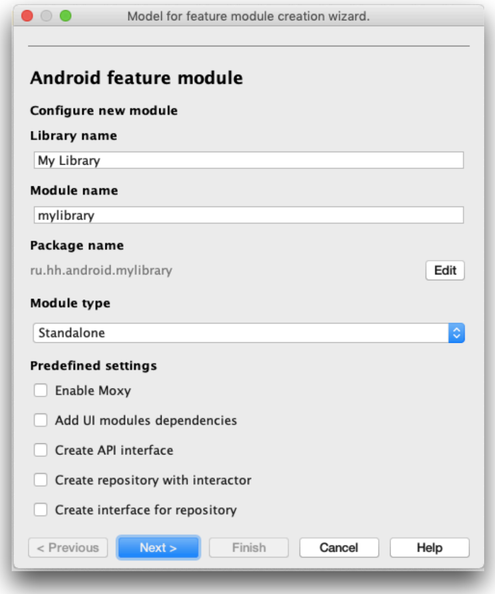
WizardModel

```
class MyWizardModel: WizardModel(title: "Title for my wizard") {

    init {
        this.add(MyWizardStep1())
        this.add(MyWizardStep2())
        this.add(MyWizardStep3())
    }

}
```

WizardStep

```
class MyWizardStep1: WizardStep<MyWizardModel>() {

    private lateinit var contentPanel: JPanel

    override fun prepare(state: WizardNavigationState?): JComponent {
        return contentPanel
    }

}
```

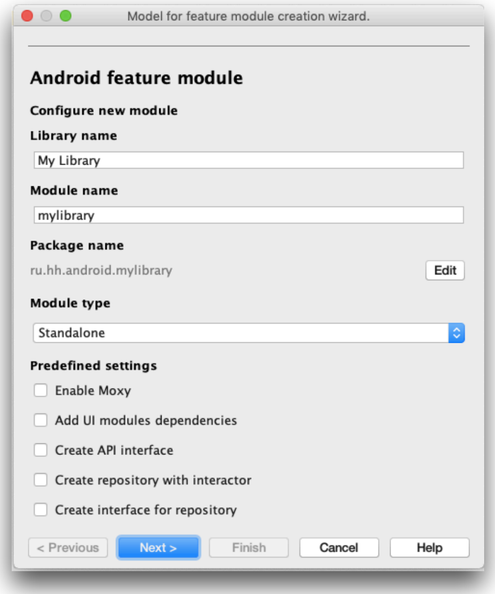
+ .form ==>



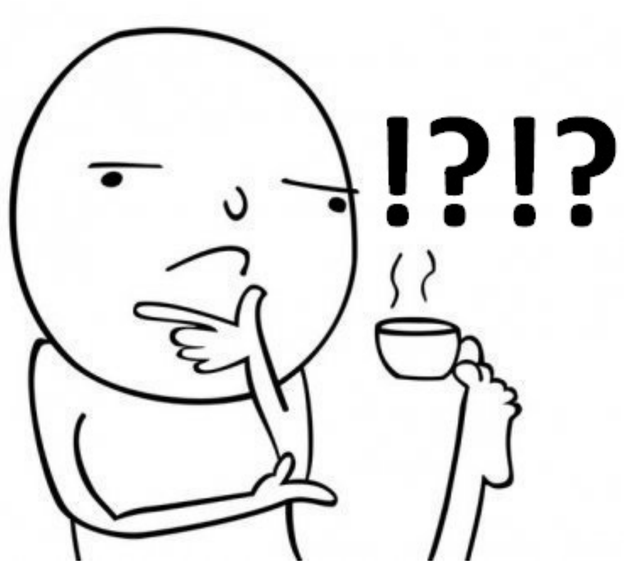
WizardStep - ожидание

```
class MyWizardStep1: WizardStep<MyWizardModel>() {

    private lateinit var contentPanel: JPanel

    override fun prepare(state: WizardNavigationState?): JComponent {
        return contentPanel
    }

}
```

+ .form ==>



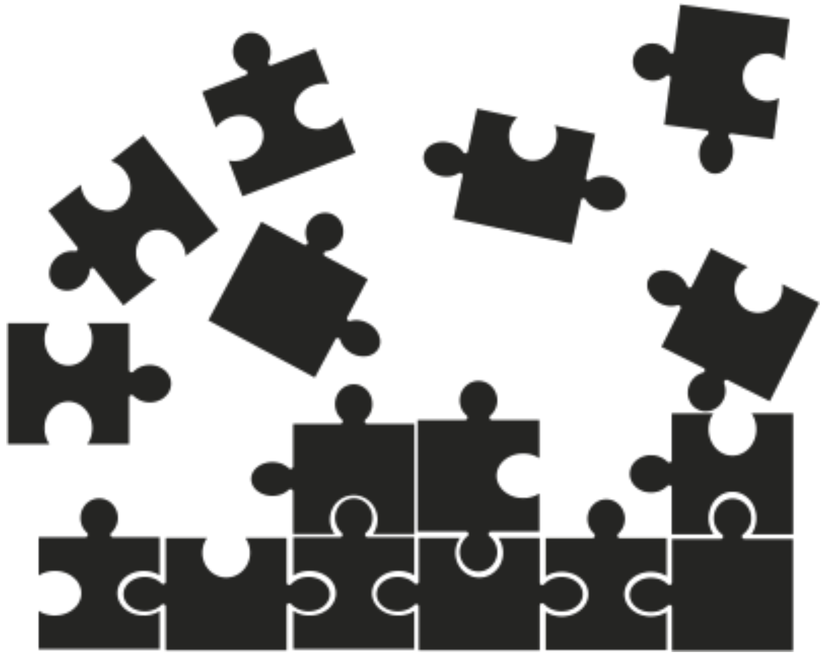
WizardStep - реальность




```
class ChooseMainParametersWizardStep: WizardStep<PluginWizardModel>() {

    private lateinit var contentPanel: JPanel
    private lateinit var libraryNameTextField: JTextField
    private lateinit var moduleNameTextField: JTextField
    private lateinit var packageNameTextField: JTextField
    private lateinit var editPackageNameButton: JButton
    private lateinit var packageNameLabel: JLabel
    private lateinit var moduleTypeComboBox: JComboBox<*>
    private lateinit var enableMoxyCheckBox: JCheckBox
    private lateinit var addUIModulesDependenciesCheckBox: JCheckBox
    private lateinit var createAPIInterfaceCheckBox: JCheckBox
    private lateinit var createRepositoryWithInteractorCheckBox: JCheckBox
    private lateinit var createInterfaceForRepositoryCheckBox: JCheckBox


    override fun prepare(state: WizardNavigationState?): JComponent? = contentPanel


    override fun onCancel() { ... }

    override fun onNext(model: PluginWizardModel): WizardStep<*> { ... }

    override fun onPrevious(model: PluginWizardModel): WizardStep<*> { ... }

}
```

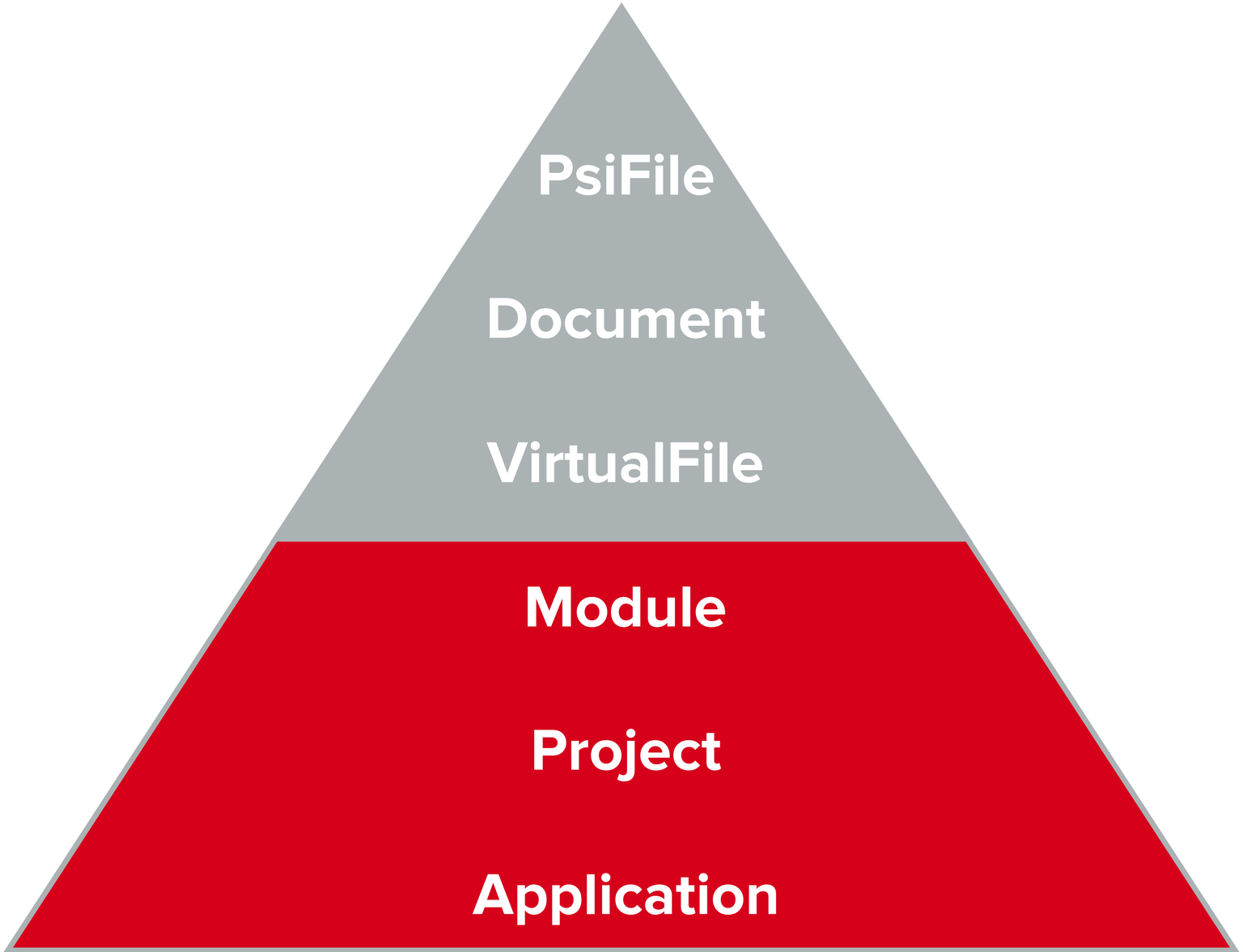
Вывод

Нужен многостраничный UI - используй WizardDialog

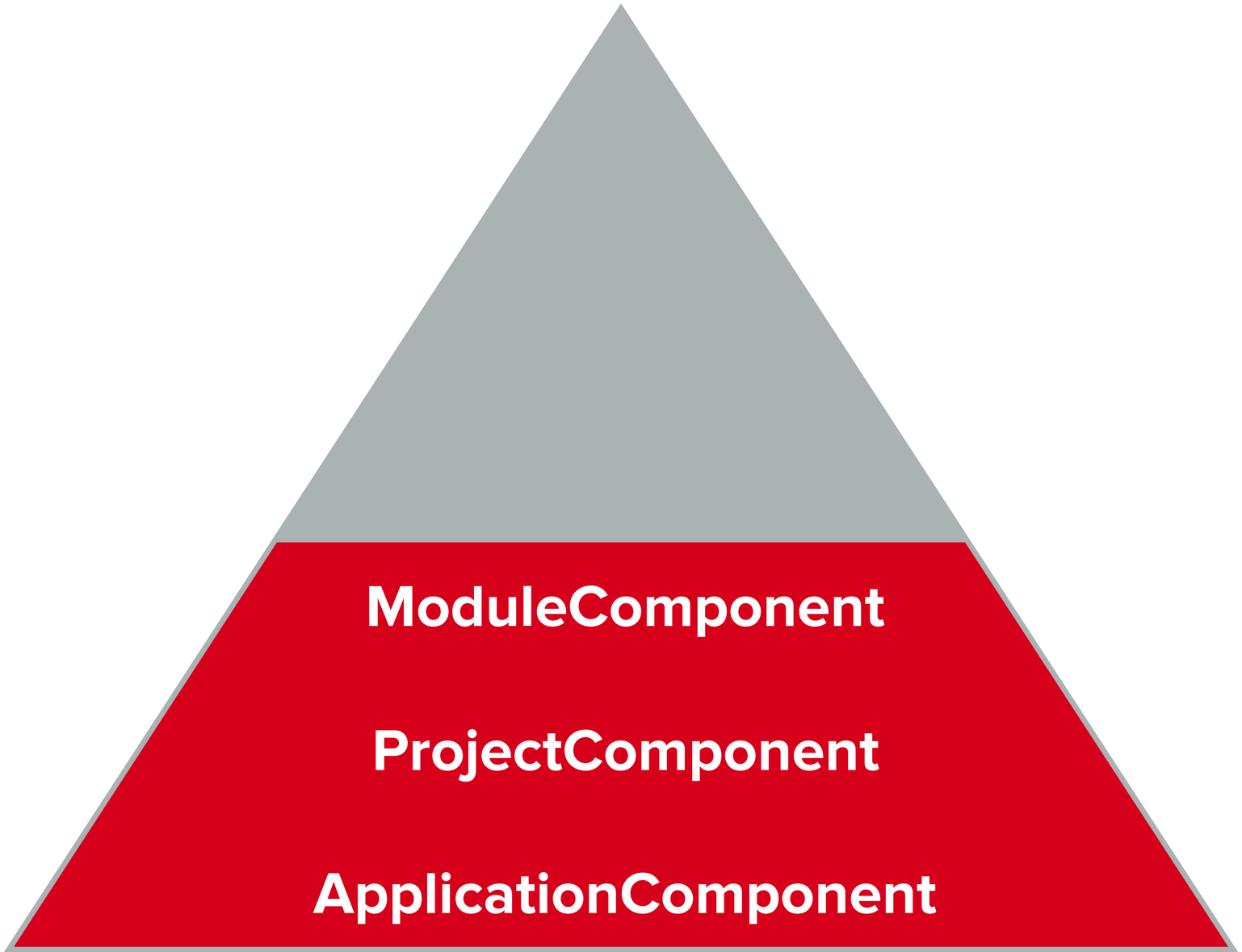


DI в плагинах

Intellij IDEA DI framework



Intellij IDEA DI framework



Intellij IDEA DI framework

1. Создать класс, реализующий один из Component-интерфейсов

Components

```
class MyAppComponent: ApplicationComponent

class MyProjectComponent: ProjectComponent

class MyModuleComponent: ModuleComponent
```

Components

```
class MyAppComponent(val application: Application): ApplicationComponent

class MyProjectComponent(val project: Project): ProjectComponent

class MyModuleComponent(val module: Module): ModuleComponent
```

Components

```
class MyAppComponent(
    val application: Application,
    val anotherApplicationComponent: AnotherAppComponent
): ApplicationComponent

class MyProjectComponent(
    val project: Project,
    val anotherProjectComponent: AnotherProjectComponent,
    val myAppComponent: MyAppComponent
): ProjectComponent

class MyModuleComponent(
    val module: Module,
    val anotherModuleComponent: AnotherModuleComponent,
    val myProjectComponent: MyProjectComponent,
    val myAppComponent: MyAppComponent
): ModuleComponent
```

Intellij IDEA DI framework

1. Создать класс, реализующий один из Component-интерфейсов
2. **Зарегистрировать компонент в plugin.xml**

plugin.xml

```
<idea-plugin>

    <project-components>
        <component>
            <interface-class>
                com.experiment.MyProjectComponent
            </interface-class>

            <implementation-class>
                com.experiments.MyProjectComponentImpl
            </implementation-class>
        </component>
    </project-components>

</idea-plugin>
```

plugin.xml

```
<idea-plugin>

    <project-components>
        <component>
            <interface-class>
                com.experiment.MyProjectComponent
            </interface-class>

            <implementation-class>
                com.experiments.MyProjectComponentImpl
            </implementation-class>
        </component>
    </project-components>

</idea-plugin>
```

plugin.xml

```
<idea-plugin>

    <project-components>
        <component>
            <interface-class>
                com.experiment.MyProjectComponent
            </interface-class>

            <implementation-class>
                com.experiments.MyProjectComponentImpl
            </implementation-class>
        </component>
    </project-components>

</idea-plugin>
```

plugin.xml

```
<idea-plugin>

    <project-components>
        <component>
            <interface-class>
                com.experiment.MyProjectComponent
            </interface-class>

            <implementation-class>
                com.experiments.MyProjectComponentImpl
            </implementation-class>
        </component>
    </project-components>

</idea-plugin>
```

Intellij IDEA DI framework

1. Создать класс, реализующий один из Component-интерфейсов
2. Зарегистрировать компонент в plugin.xml
3. **Получить компонент из соответствующего объекта**

Components

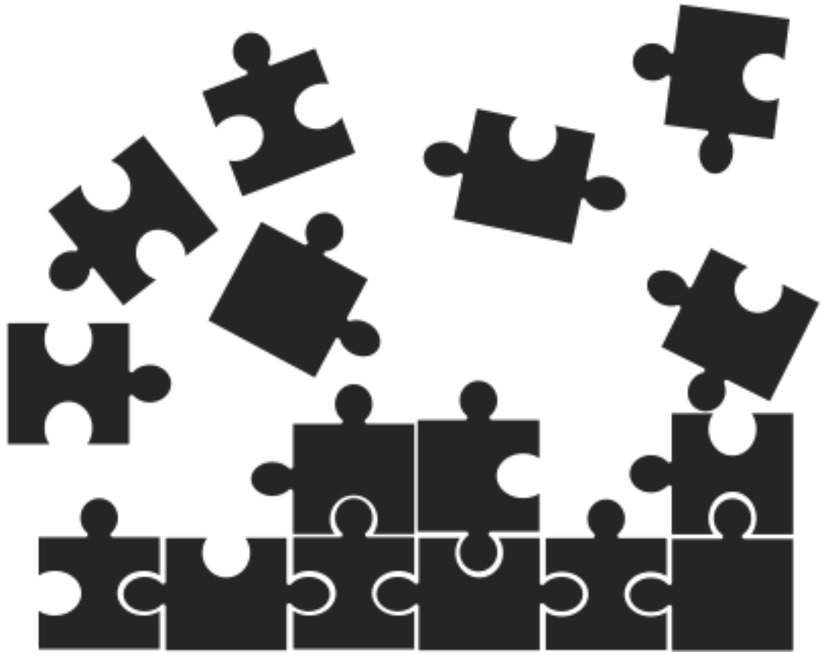
```
val appComponent = application.getComponent(MyAppComponent::class.java)

val myProjectComponent = project.getComponent(MyProjectComponent::class.java)

val myModuleComponent = module.getComponent(MyModuleComponent::class.java)
```

Выводы

1. В Intellij IDEA есть встроенный DI

Выводы

1. В Intellij IDEA есть встроенный DI
2. **С помощью DI можно достучаться до уже написанных компонентов**



Генерация кода

Пример шаблона

```
apply plugin: 'com.android.library'

<if (isKotlinProject) {
    apply plugin: 'kotlin-android'
    apply plugin: 'kotlin-kapt'

    <if (isModuleWithUI) {
        apply plugin: 'kotlin-android-extensions'
    }>
}>

...
android {
    ...
    <if (isMoxyEnabled) {
        kapt {
            arguments {
                arg("moxyReflectorPackage", '<include var="packageName">')
            }
        }
    }>
    ...
}

...
dependencies {
    compileOnly project(':common')
    compileOnly project(':core-utils')

    <for (moduleName in enabledModules) {
        compileOnly project('<include var="moduleName">')
    }>
    ...
}
```

Пример шаблона

```
apply plugin: 'com.android.library'

<if (isKotlinProject) {
    apply plugin: 'kotlin-android'
    apply plugin: 'kotlin-kapt'

    <if (isModuleWithUI) {
        apply plugin: 'kotlin-android-extensions'
    }>
}>

...
android {
    ...
    <if (isMoxyEnabled) {
        kapt {
            arguments {
                arg("moxyReflectorPackage", '<include var="packageName">')
            }
        }
    }>
    ...
}

...
dependencies {
    compileOnly project(':common')
```

Пример шаблона

```
apply plugin: 'com.android.library'

<if (isKotlinProject) {
    apply plugin: 'kotlin-android'
    apply plugin: 'kotlin-kapt'

    <if (isModuleWithUI) {
        apply plugin: 'kotlin-android-extensions'
    }>
}>

...
android {
    ...
   <if (isMoxyEnabled) {
          kapt {
              arguments {
                  arg("moxyReflectorPackage", '<include var="packageName">')
              }
          }
      }>
    ...
}

...
dependencies {
    compileOnly project(':common')
    compileOnly project(':core-utils')
```

Пример шаблона

```
...

dependencies {
    compileOnly project(':common')
    compileOnly project(':core-utils')

    <for (moduleName in enabledModules) {
        compileOnly project('<include var="moduleName">')
    }>
    ...
}
```

Требования к генератору

1. Возможность использования переменных
2. Блоки условий
3. Наличие циклов

Варианты

1. Написать свой генератор кода

rib-intellij-plugin

uber / RIBs

Watch 171 | Star 4,275 | Fork 398

Code | Issues 31 | Pull requests 6 | Projects 1 | Wiki | Insights

Branch: master | Create new file | Upload files | Find file | History

RIBs / android / tooling / rib-intellij-plugin / src / main / resources / templates / kotlin /

rendecano and AttwellBrian Updated templates and codes based on review comments ... Latest commit f214dfd on 23 Jan 2018

..		
RibBuilder.kt.template	Updated templates and codes based on review comments	a year ago
RibInteractorWithEmptyPresenter.kt.t...	Updated templates and codes based on review comments	a year ago
RibInteractorWithEmptyPresenterTest...	Updated templates and codes based on review comments	a year ago
RibInteractorWithPresenter.kt.template	Updated templates and codes based on review comments	a year ago
RibInteractorWithPresenterTest.kt.te...	Updated templates and codes based on review comments	a year ago
RibRouter.kt.template	Updated templates and codes based on review comments	a year ago
RibRouterTest.kt.template	Updated templates and codes based on review comments	a year ago

rib-intellij-plugin

```
import javax.inject.Inject;

/**
 * Coordinates Business Logic for {@link ${rib_name}Scope}.
 *
 * TODO describe the logic of this scope.
 */
@RibInteractor
public class ${rib_name}Interactor extends Interactor<EmptyPresenter, ${rib_name}Router> {

${partial: RibInteractorDidBecomeActive.java.partial}
${partial: RibInteractorWillResignActive.java.partial}
}
```

Варианты

1. ~~Написать свой генератор кода~~
2. **FileTemplateManager**

Варианты

1. ~~Написать свой генератор кода~~
2. **FileTemplateManager**
    - 2.1 **В качестве движка использует Velocity**

Варианты

1. ~~Написать свой генератор кода~~
2. **FileTemplateManager**
   - 2.1 **В качестве движка использует Velocity**
   - 2.2 **Не умеет генерировать файлы, отличающиеся от .java и .xml**

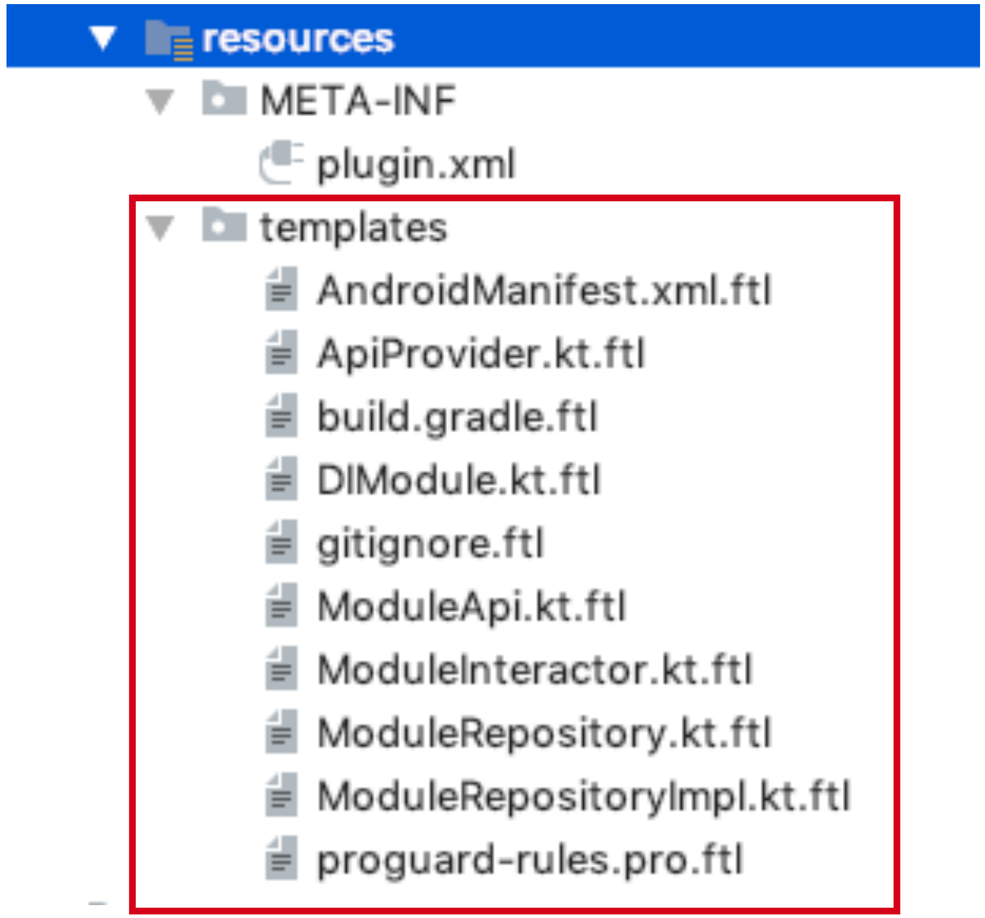
Варианты

1. ~~Написать свой генератор кода~~
2. ~~FileTemplateManager~~
   - 2.1 ~~В качестве движка использует Velocity~~
   - 2.2 ~~Не умеет генерировать файлы, отличающиеся от .java и .xml~~
3. **FreeMarker**

Варианты

1. ~~Написать свой генератор кода~~
2. ~~FileTemplateManager~~
   - 2.1 ~~В качестве движка использует Velocity~~
   - 2.2 ~~Не умеет генерировать файлы, отличающиеся от .java и .xml~~
3. **FreeMarker**

Добавляем шаблоны файлов



Добавляем зависимость от либо

```
dependencies {
    compile 'org.jetbrains.kotlin:kotlin-stdlib-jdk8'
    compile 'org.freemarker:freemarker:2.3.28'
    compile 'commons-io:commons-io:2.4'
    compile 'com.arello-mobile:moxy:1.5.5'

    testCompile group: 'junit', name: 'junit', version: '4.12'
}
```

Добавляем зависимость от либо

```
dependencies {
    compile 'org.jetbrains.kotlin:kotlin-stdlib-jdk8'
    compile 'org.freemarker:freemarker:2.3.28'
    compile 'commons-io:commons-io:2.4'
    compile 'com.arello-mobile:moxy:1.5.5'

    testCompile group: 'junit', name: 'junit', version: '4.12'
}
```

Конфигурируем FreeMarker

```
class TemplatesFactory(val project: Project) : ProjectComponent {

  private val freeMarkerConfig by lazy {
      Configuration(Configuration.VERSION_2_3_28).apply {
          setClassForTemplateLoading(
              TemplatesFactory::class.java,
              "/templates"
          )

          defaultEncoding = Charsets.UTF_8.name()
          templateExceptionHandler = TemplateExceptionHandler.RETHROW_HANDLER
          logTemplateExceptions = false
          wrapUncheckedExceptions = true
      }
  }
```

...

Создаем файлы из шаблонов

```
class TemplatesFactory(val project: Project) : ProjectComponent {
...

  fun generate(
    pathToFile: String,
    templateFileName: String,
    data: Map<String, Any>
  ) {
      val template = freeMarkerConfig.getTemplate(templateFileName)

      FileWriter(pathToFile, false).use { writer ->
          template.process(data, writer)
      }
  }

}
```

Создаем файлы из шаблонов

```
class TemplatesFactory(val project: Project) : ProjectComponent {
...

  fun generate(
    pathToFile: String,
    templateFileName: String,
    data: Map<String, Any>
  ) {
      val template = freeMarkerConfig.getTemplate(templateFileName)

      FileWriter(pathToFile, false).use { writer ->
          template.process(data, writer)
      }
  }

}
```

Создаем файлы из шаблонов

```
class TemplatesFactory(val project: Project) : ProjectComponent {
...

  fun generate(
    pathToFile: String,
    templateFileName: String,
    data: Map<String, Any>
  ) {
      val template = freemarkerConfig.getTemplate(templateFileName)

      FileWriter(pathToFile, false).use { writer ->
          template.process(data, writer)
      }
  }

}
```

Don't do this

PSI = Program Structure Interface

Создаем структуру PsiDirectory

```
fun generatePsiDirectories(project: Project, config: CreateModuleConfig) {
    val projectPsiDir = project.baseDir.toPsiDirectory(project) ?: return

    val rootDir = when (config.featureModuleType) {
        STANDALONE -> projectPsiDir
        FEATURE_MODULE -> projectPsiDir.findSubdirectory("feature")
        CORE_MODULE -> projectPsiDir.findSubdirectory("core")
    } ?: return

    val featureModulePsiDir = rootDir.createSubdirectory(config.moduleName)
    ...
}
```

Создаем структуру PsiDirectory

```
fun generatePsiDirectories(project: Project, config: CreateModuleConfig) {
    val projectPsiDir = project.baseDir.toPsiDirectory(project) ?: return

    val rootDir = when (config.featureModuleType) {
        STANDALONE -> projectPsiDir
        FEATURE_MODULE -> projectPsiDir.findSubdirectory("feature")
        CORE_MODULE -> projectPsiDir.findSubdirectory("core")
    } ?: return

    val featureModulePsiDir = rootDir.createSubdirectory(config.moduleName)
    ...
}
```

Создаем структуру PsiDirectory

```
fun generatePsiDirectories(project: Project, config: CreateModuleConfig) {
    val projectPsiDir = project.baseDir.toPsiDirectory(project) ?: return

    val rootDir = when (config.featureModuleType) {
        STANDALONE -> projectPsiDir
        FEATURE_MODULE -> projectPsiDir.findSubdirectory("feature")
        CORE_MODULE -> projectPsiDir.findSubdirectory("core")
    } ?: return

    val featureModulePsiDir = rootDir.createSubdirectory(config.moduleName)
    ...
}
```

PsiDirectory | PsiFileFactory | FileType | Генерация

Создаем структуру PsiDirectory

```
fun generatePsiDirectories(project: Project, config: CreateModuleConfig) {
    val projectPsiDir = project.baseDir.toPsiDirectory(project) ?: return

    val rootDir = when (config.featureModuleType) {
        STANDALONE -> projectPsiDir
        FEATURE_MODULE -> projectPsiDir.findSubdirectory("feature")
        CORE_MODULE -> projectPsiDir.findSubdirectory("core")
    } ?: return

    val featureModulePsiDir = rootDir.createSubdirectory(config.moduleName)
    ...
}
```

Создаем структуру PsiDirectory

```
fun generatePsiDirectories(project: Project, config: CreateModuleConfig) {
    val featureModulePsiDir = rootDir.createSubdirectory(config.moduleName)

    ...

    val dirsMap = mutableMapOf<String, PsiDirectory?>().apply {
        this["module_folder"] = featureModulePsiDir
        this["src"] = featureModulePsiDir.createSubdirectory("src")
        this["main"] = this["src"]?.createSubdirectory("main")

        createPackageNameFolders(config)

        ...
    }
}
```

Создаем структуру PsiDirectory

```
fun generatePsiDirectories(project: Project, config: CreateModuleConfig) {
    val featureModulePsiDir = rootDir.createSubdirectory(config.moduleName)

    ...

    val dirsMap = mutableMapOf<String, PsiDirectory?>().apply {
        this["module_folder"] = featureModulePsiDir
        this["src"] = featureModulePsiDir.createSubdirectory("src")
        this["main"] = this["src"]?.createSubdirectory("main")

        createPackageNameFolders(config)

        ...
    }
}
```

PsiDirectory | PsiFileFactory | FileType | Генерация

Создаем структуру PsiDirectory

```
fun generatePsiDirectories(project: Project, config: CreateModuleConfig) {
    val featureModulePsiDir = rootDir.createSubdirectory(config.moduleName)

    ...

    val dirsMap = mutableMapOf<String, PsiDirectory?>().apply {
        this["module_folder"] = featureModulePsiDir
        this["src"] = featureModulePsiDir.createSubdirectory("src")
        this["main"] = this["src"]?.createSubdirectory("main")

        createPackageNameFolders(config)

        ...
    }
}
```

PsiDirectory | PsiFileFactory | FileType | Генерация

Как будем создавать файлы?

```
val psiFileFactory = PsiFileFactory.getInstance(project)

val psiFile = psiFileFactory.createFileFromText(
    outputFileName,
    outputFileType,
    outputFileText
)
```

Как будем создавать файлы?

```
val psiFileFactory = PsiFileFactory.getInstance(project)

val psiFile = psiFileFactory.createFileFromText(
    outputFileName,
    outputFileType,
    outputFileText
)
```

PsiDirectory | PsiFileFactory | FileType | Генерация

Как будем создавать файлы?

```
val psiFileFactory = PsiFileFactory.getInstance(project)

val psiFile = psiFileFactory.createFileFromText(
    outputFileName,
    outputFileType,
    outputFileText
)
```

PsiDirectory | PsiFileFactory | FileType | Генерация

Как будем создавать файлы?

```
val psiFileFactory = PsiFileFactory.getInstance(project)

val psiFile = psiFileFactory.createFileFromText(
    outputFileName,
    outputFileType,
    outputFileText
)
```

Как будем создавать файлы?

```
val psiFileFactory = PsiFileFactory.getInstance(project)

val psiFile = psiFileFactory.createFileFromText(
    outputFileName,
    outputFileType,
    outputFileText
)
```

PsiDirectory | **PsiFileFactory** | FileType | Генерация

Как будем создавать файлы?

```
val psiFileFactory = PsiFileFactory.getInstance(project)

val psiFile = psiFileFactory.createFileFromText(
    outputFileName,
    outputFileType,
    outputFileText
)
```

PsiDirectory | PsiFileFactory | FileType | Генерация

FileType

1. JavaFileType
2. XmlFileType

FileType

1. JavaFileType
2. XmlFileType
3. **GroovyFileType ?**

FileType

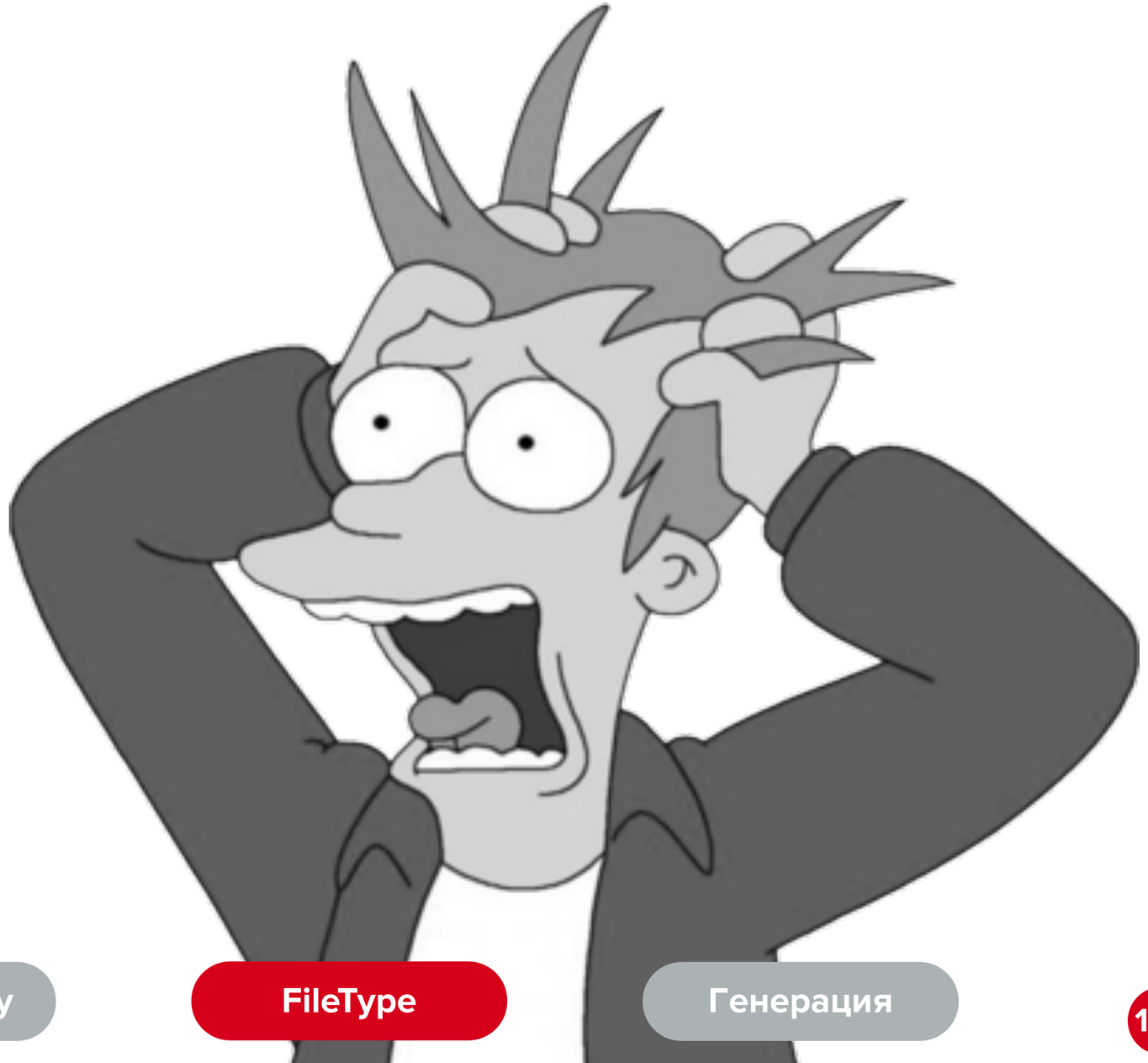
1. JavaFileType
2. XmlFileType
3. **GroovyFileType ?**
4. **KotlinFileType ??**

FileType

1. JavaFileType
2. XmlFileType
3. **GroovyFileType ?**
4. **KotlinFileType ??**
5. **ProguardFileType ???**

FileType

1. JavaFileType
2. XmlFileType
3. **GroovyFileType ?**
4. **KotlinFileType ??**
5. **ProguardFileType ???**
6. **GitIgnoreFileType ????**



Plugin dependencies

3 GroovyFileType → Groovy-плагин

4 KotlinFileType → Kotlin-плагин

5 ProguardFileType → Android-плагин

Plugin’s build.gradle

```
intellij {
    version '2018.2.4'
    updateSinceUntilBuild false
    pluginName = 'AndroidFeatureModule'
    plugins = ['android', 'Groovy', 'kotlin']

    intellij.alternativeIdePath =
        '/Applications/Android Studio.app'
}
```

Plugin’s build.gradle

```
intellij {
    version '2018.2.4'
    updateSinceUntilBuild false
    pluginName = 'AndroidFeatureModule'
    plugins = ['android', 'Groovy', 'kotlin']

    intellij.alternativeIdePath =
        '/Applications/Android Studio.app'
}
```

Plugin’s build.gradle

```
intellij {
    version '2018.2.4'
    updateSinceUntilBuild false
    pluginName = 'AndroidFeatureModule'
    plugins = ['android', 'Groovy', 'kotlin']

    intellij.alternativeIdePath =
        '/Applications/Android Studio.app'
}
```

gradle-intellij-plugin Github

PsiDirectory | PsiFileFactory | FileType | Генерация

plugin.xml

```
<idea-plugin>

    ...

    <depends>org.jetbrains.android</depends>
    <depends>org.jetbrains.kotlin</depends>
    <depends>org.intellij.groovy</depends>

</idea-plugin>
```

plugin.xml

```
<idea-plugin>

    ...

    <depends>org.jetbrains.android</depends>
    <depends>org.jetbrains.kotlin</depends>
    <depends>org.intellij.groovy</depends>

</idea-plugin>
```

GitIgnoreLanguage

```
class IgnoreLanguage private constructor()
    : Language(ID: "ru.hh.plugins.Ignore", mimeTypes: "ignore", null),
      InjectableLanguage {

    companion object {
        val INSTANCE = IgnoreLanguage()
    }

    override fun getDisplayName(): String {
        return "Ignore() ($id)"
    }

}
```

GitIgnoreLanguage

```
class IgnoreLanguage private constructor()
    : Language(ID: "ru.hh.plugins.Ignore", mimeTypes: "ignore", null),
      InjectableLanguage {

    companion object {
        val INSTANCE = IgnoreLanguage()
    }

    override fun getDisplayName(): String {
        return "Ignore() ($id)"
    }

}
```

GitIgnoreFileType

```
class IgnoreFileType(language: Language) : LanguageFileType(language) {

    companion object {
        val INSTANCE = IgnoreFileType(IgnoreLanguage.INSTANCE)
    }

    override fun getName(): String = "gitignore file"
    override fun getDescription() = "gitignore files"
    override fun getDefaultExtension(): String = "gitignore"
    override fun getIcon(): Icon? = null

}
```

TemplateData

```
fun generateTemplateData(
    dirsMap: Map<String, PsiDirectory?>,
    config: CreateModuleConfig
): List<TemplateData> {
    return mutableListOf<TemplateData>().apply {
        this += TemplateData(
            templateFileName = "AndroidManifest.xml.ftl",
            outputFileName = "AndroidManifest.xml",
            outputFileType = XmlFileType.INSTANCE,
            outputPsiDirectory = dirsMap["main"]
        )
        ...
    }
}
```

PsiDirectory | PsiFileFactory | FileType | **Генерация**

TemplateData

```
fun generateTemplateData(
    dirsMap: Map<String, PsiDirectory?>,
    config: CreateModuleConfig
): List<TemplateData> {
    return mutableListOf<TemplateData>().apply {
        this += TemplateData(
            templateFileName = "AndroidManifest.xml.ftl",
            outputFileName = "AndroidManifest.xml",
            outputFileType = XmlFileType.INSTANCE,
            outputPsiDirectory = dirsMap["main"]
        )
        ...
    }
}
```

PsiDirectory | PsiFileFactory | FileType | **Генерация**

TemplateData

```
fun generateTemplateData(
    dirsMap: Map<String, PsiDirectory?>,
    config: CreateModuleConfig
): List<TemplateData> {
    return mutableListOf<TemplateData>().apply {
        this += TemplateData(
            templateFileName = "AndroidManifest.xml.ftl",
            outputFileName = "AndroidManifest.xml",
            outputFileType = XmlFileType.INSTANCE,
            outputPsiDirectory = dirsMap["main"]
        )
        ...
    }
}
```

PsiDirectory | PsiFileFactory | FileType | **Генерация**

TemplateData

```
fun generateTemplateData(
    dirsMap: Map<String, PsiDirectory?>,
    config: CreateModuleConfig
): List<TemplateData> {
    return mutableListOf<TemplateData>().apply {
        this += TemplateData(
            templateFileName = "AndroidManifest.xml.ftl",
            outputFileName = "AndroidManifest.xml",
            outputFileType = XmlFileType.INSTANCE,
            outputPsiDirectory = dirsMap["main"]
        )
        ...
    }
}
```

PsiDirectory | PsiFileFactory | FileType | **Генерация**

TemplateData

```
fun generateTemplateData(
    dirsMap: Map<String, PsiDirectory?>,
    config: CreateModuleConfig
): List<TemplateData> {
    return mutableListOf<TemplateData>().apply {
        this += TemplateData(
            templateFileName = "AndroidManifest.xml.ftl",
            outputFileName = "AndroidManifest.xml",
            outputFileType = XmlFileType.INSTANCE,
            outputPsiDirectory = dirsMap["main"]
        )
        ...
    }
}
```

PsiDirectory | PsiFileFactory | FileType | **Генерация**

TemplateData

```
fun generateTemplateData(
    dirsMap: Map<String, PsiDirectory?>,
    config: CreateModuleConfig
): List<TemplateData> {
    return mutableListOf<TemplateData>().apply {
        this += TemplateData(
            templateFileName = "AndroidManifest.xml.ftl",
            outputFileName = "AndroidManifest.xml",
            outputFileType = XmlFileType.INSTANCE,
            outputPsiDirectory = dirsMap["main"]
        )
        ...
    }
}
```

PsiDirectory | PsiFileFactory | FileType | **Генерация**

FreeMarker

```
class TemplatesFactory(val project: Project) : ProjectComponent {

  fun generate(data: TemplateData, properties: Map<String, Any>) {
    val templateFile = freeMarkerConfig.getTemplate(data.templateFileName)

    val templateText = StringWriter().use { writer ->
      templateFile.process(properties, writer)

      writer.buffer.toString()
    }

    val psiFile = PsiFileFactory.getInstance(project).createFileFromText(
        data.outputFileName, data.outputFileType, text
    )
    data.outputPsiDirectory?.add(psiFile)
  }
  ...
}
```

PsiDirectory

PsiFileFactory

FileType

Генерация

FreeMarker

```
class TemplatesFactory(val project: Project) : ProjectComponent {

  fun generate(data: TemplateData, properties: Map<String, Any>) {
    val templateFile = freeMarkerConfig.getTemplate(data.templateFileName)

    val templateText = StringWriter().use { writer ->
      templateFile.process(properties, writer)

      writer.buffer.toString()
    }

    val psiFile = PsiFileFactory.getInstance(project).createFileFromText(
        data.outputFileName, data.outputFileType, text
    )
    data.outputPsiDirectory?.add(psiFile)
  }
  ...
}
```

PsiDirectory

TemplateData

FileType

Генерация

FreeMarker

```
class TemplatesFactory(val project: Project) : ProjectComponent {

  fun generate(data: TemplateData, properties: Map<String, Any>) {
    val templateFile = freeMarkerConfig.getTemplate(data.templateFileName)

    val templateText = StringWriter().use { writer ->
      templateFile.process(properties, writer)

      writer.buffer.toString()
    }

    val psiFile = PsiFileFactory.getInstance(project).createFileFromText(
        data.outputFileName, data.outputFileType, text
    )
    data.outputPsiDirectory?.add(psiFile)
  }
  ...
}
```

PsiDirectory

TemplateData

FileType

Генерация

FreeMarker

```
class TemplatesFactory(val project: Project) : ProjectComponent {

  fun generate(data: TemplateData, properties: Map<String, Any>) {
    val templateFile = freeMarkerConfig.getTemplate(data.templateFileName)

    val templateText = StringWriter().use { writer ->
      templateFile.process(properties, writer)

      writer.buffer.toString()
    }

    val psiFile = PsiFileFactory.getInstance(project).createFileFromText(
        data.outputFileName, data.outputFileType, text
    )
    data.outputPsiDirectory?.add(psiFile)
  }
  ...
}
```

PsiDirectory

TemplateData

FileType

Генерация

FreeMarker

```
class TemplatesFactory(val project: Project) : ProjectComponent {

  fun generate(data: TemplateData, properties: Map<String, Any>) {
    val templateFile = freeMarkerConfig.getTemplate(data.templateFileName)

    val templateText = StringWriter().use { writer ->
      templateFile.process(properties, writer)

      writer.buffer.toString()
    }

    val psiFile = PsiFileFactory.getInstance(project).createFileFromText(
        data.outputFileName, data.outputFileType, text
    )
    data.outputPsiDirectory?.add(psiFile)
  }
  ...
}
```

PsiDirectory

TemplateData

FileType

Генерация

Резюме

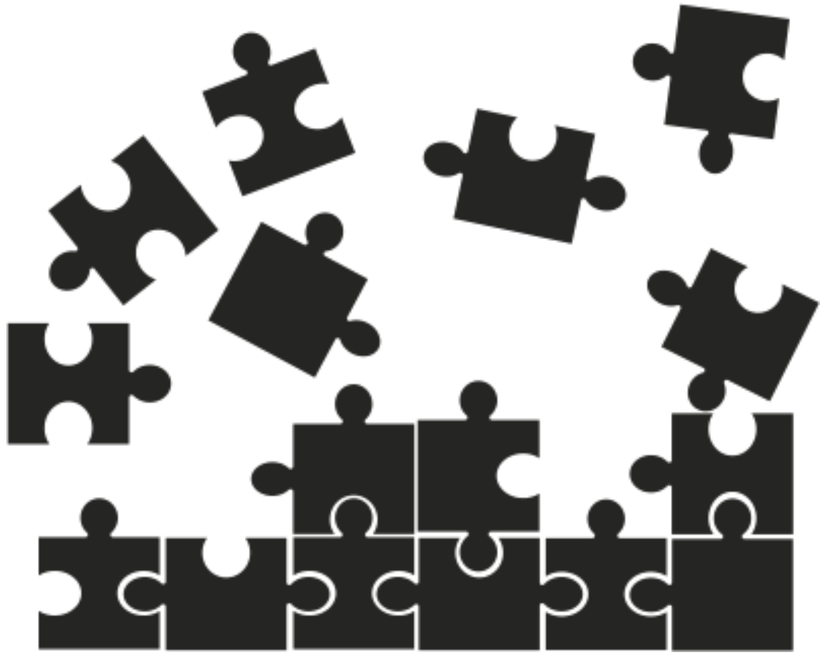
1. Текст файлов можно генерировать FreeMarker-ом

Резюме

1. Текст файлов можно генерировать FreeMarker-ом
2. **Для генерации файлов учитывайте PSI-структуру**

Резюме

1. Текст файлов можно генерировать FreeMarker-ом
2. Для генерации файлов учитывайте PSI-структуру
3. **Придется где-то найти FileType-ы**



Модификация кода

Модификация кода

1. Модифицировать settings.gradle

Модификация кода

1. **Модифицировать settings.gradle**

Найти файл

Добавить код

settings.gradle

```
val projectBaseDirPath = project.basePath ?: return
val settingsPathFile = projectBaseDirPath + "/settings.gradle"

val settingsFile = File(settingsPathFile)

settingsFile.appendText("include ':$moduleName'")
settingsFile.appendText(
 "project(':$moduleName').projectDir = new File(settingsDir, '$folderPath')"
)
```

Найти файл

Добавить код

settings.gradle

```
val projectBaseDirPath = project.basePath ?: return
val settingsPathFile = projectBaseDirPath + "/settings.gradle"

val settingsFile = File(settingsPathFile)

settingsFile.appendText("include ':$moduleName'")
settingsFile.appendText(
 "project(':$moduleName').projectDir = new File(settingsDir, '$folderPath')"
)
```

Найти файл

Добавить код

settings.gradle

```
val projectBaseDirPath = project.basePath ?: return
val settingsPathFile = projectBaseDirPath + "/settings.gradle"

val settingsFile = File(settingsPathFile)

settingsFile.appendText("include ':$moduleName'")
settingsFile.appendText(
 "project(':$moduleName').projectDir = new File(settingsDir, '$folderPath')"
)
```

Найти файл

Добавить код

settings.gradle

```
val projectBaseDirPath = project.basePath ?: return
val settingsPathFile = projectBaseDirPath + "/settings.gradle"

val settingsFile = File(settingsPathFile)

settingsFile.appendText("include ':$moduleName'")
settingsFile.appendText(
 "project(':$moduleName').projectDir = new File(settingsDir, '$folderPath')"
)
```

Найти файл

Добавить код

settings.gradle

```
val projectBaseDirPath = project.basePath ?: return
val settingsPathFile = projectBaseDirPath + "/settings.gradle"

val settingsFile = File(settingsPathFile)

settingsFile.appendText("include ':$moduleName'")
settingsFile.appendText(
 "project(':$moduleName').projectDir = new File(settingsDir, '$folderPath')"
)
```

Найти файл

Добавить код

Модификация кода

1. Модифицировать settings.gradle
2. **Настроить kapt для Toothpick**

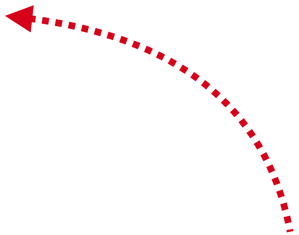
Модификация кода

1. Модифицировать settings.gradle
2. **Настроить kapt для Toothpick**

Поиск PsiFile | Поиск PsiElement | Создание строки | Вставка

build.gradle

```
android {
    defaultConfig {
        javaCompileOptions {
            annotationProcessorOptions {
                arguments = [
                    toothpick_registry_package_name: 'ru.android.hh',
                    toothpick_registry_children_package_names: [
                        "ru.hh.network_source",
                        "ru.hh.remote_config_source.applicant",
                        ...
                    ].join(',')
                ]
            }
        }
    }
    ...
}
```



Нужно добавить новый пакет сюда

Ищем Psi-версию build.gradle

```
val appModule = ModuleManager.getInstance(project)
                        .modules.toList()
                        .first { it.name == "headhunter-applicant" }

val buildGradlePsiFile = FilenameIndex.getFilesByName(
                            appModule.project,
                            "build.gradle",
                            appModule.moduleContentScope
                        ).first()

...
```

Поиск PsiFile | Поиск PsiElement | Создание строки | Вставка

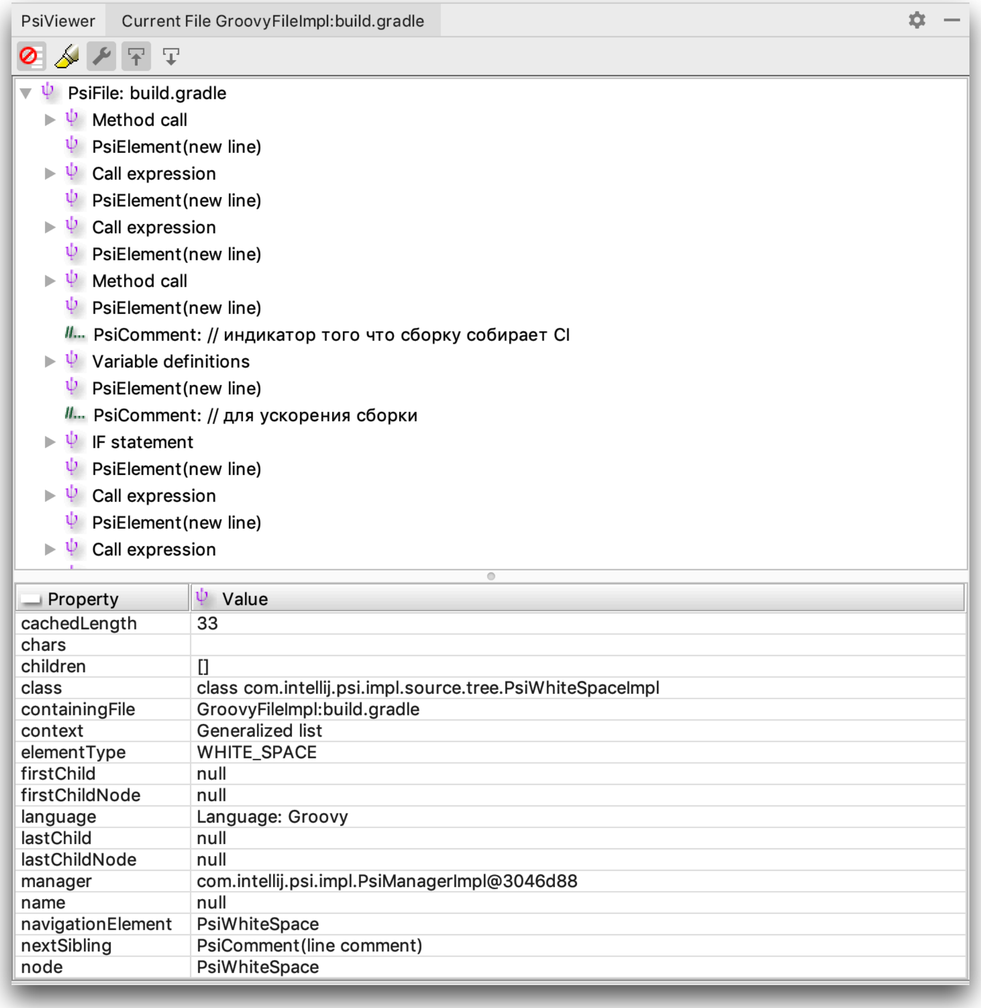
Ищем Psi-версию build.gradle

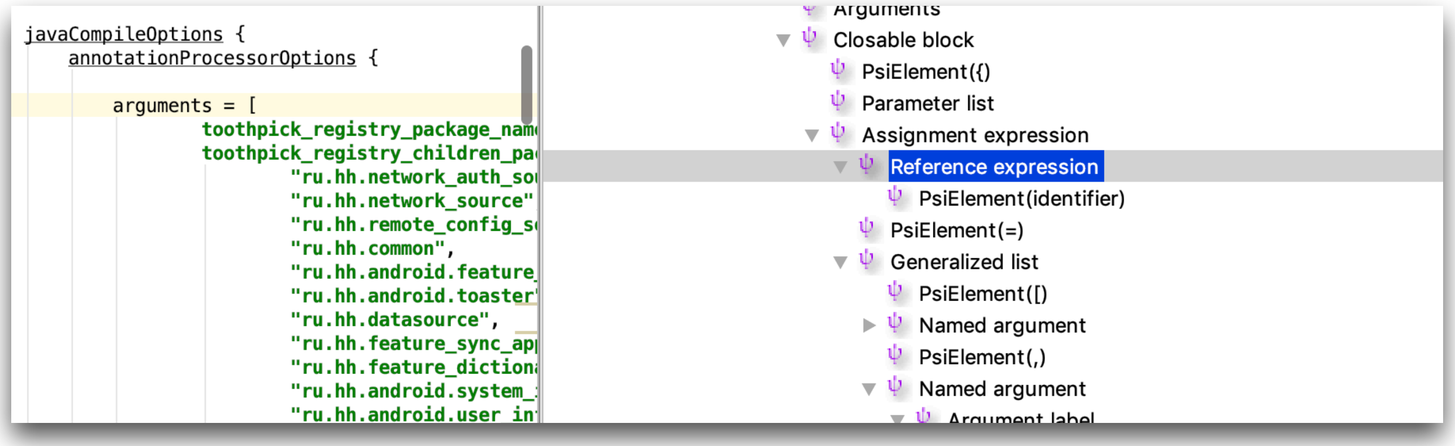
```
val appModule = ModuleManager.getInstance(project)
                        .modules.toList()
                        .first { it.name == "headhunter-applicant" }

val buildGradlePsiFile = FilenameIndex.getFilesByName(
                                appModule.project,
                                "build.gradle",
                                appModule.moduleContentScope
                            ).first()

...
```

Ищем Psi-версию build.gradle

```
val appModule = ModuleManager.getInstance(project)
                        .modules.toList()
                        .first { it.name == "headhunter-applicant" }

val buildGradlePsiFile = FilenameIndex.getFilesByName(
                                appModule.project,
                                "build.gradle",
                                appModule.moduleContentScope
                            ).first()

...
```

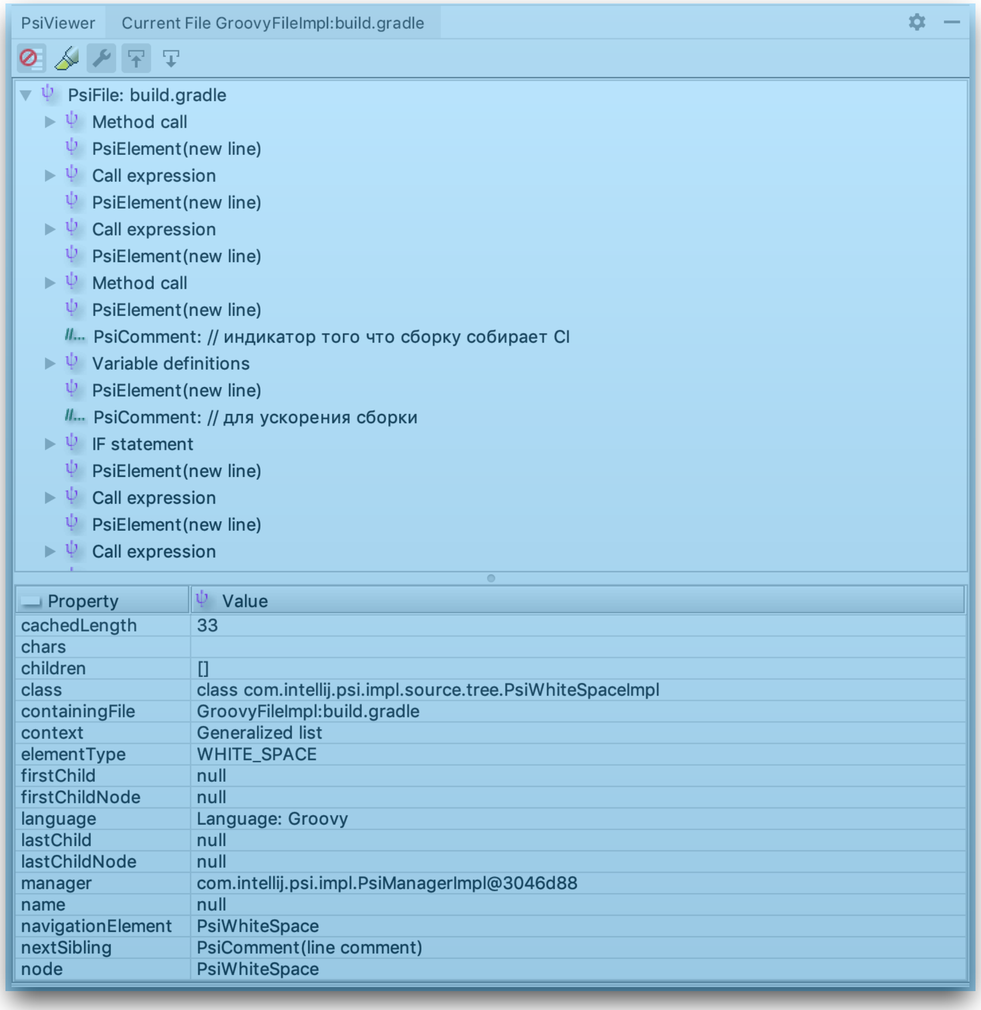
PsiViewer plugin

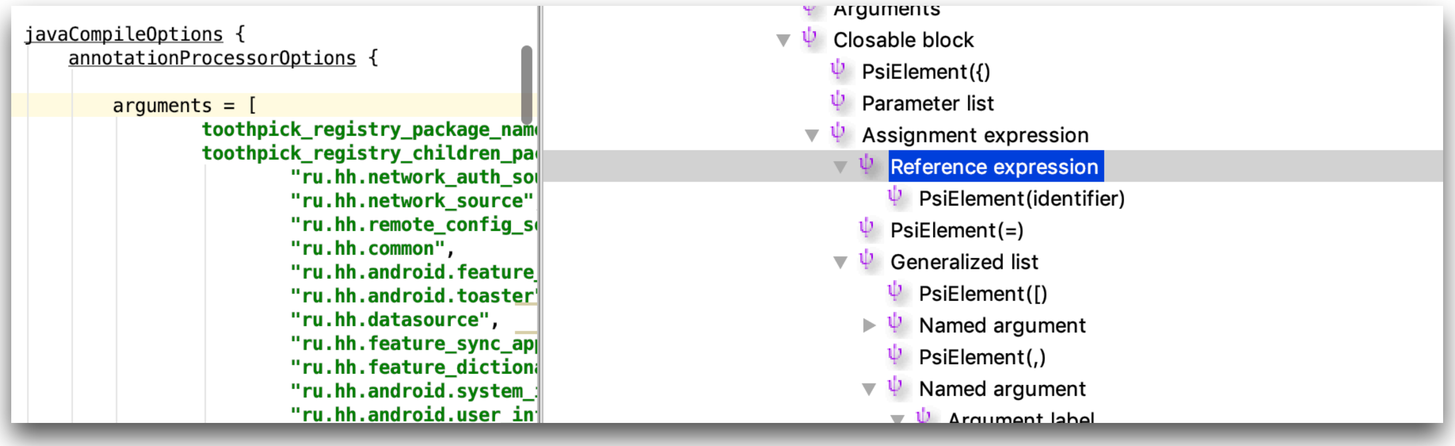




Поиск PsiFile | **Поиск PsiElement** | Создание строки | Вставка

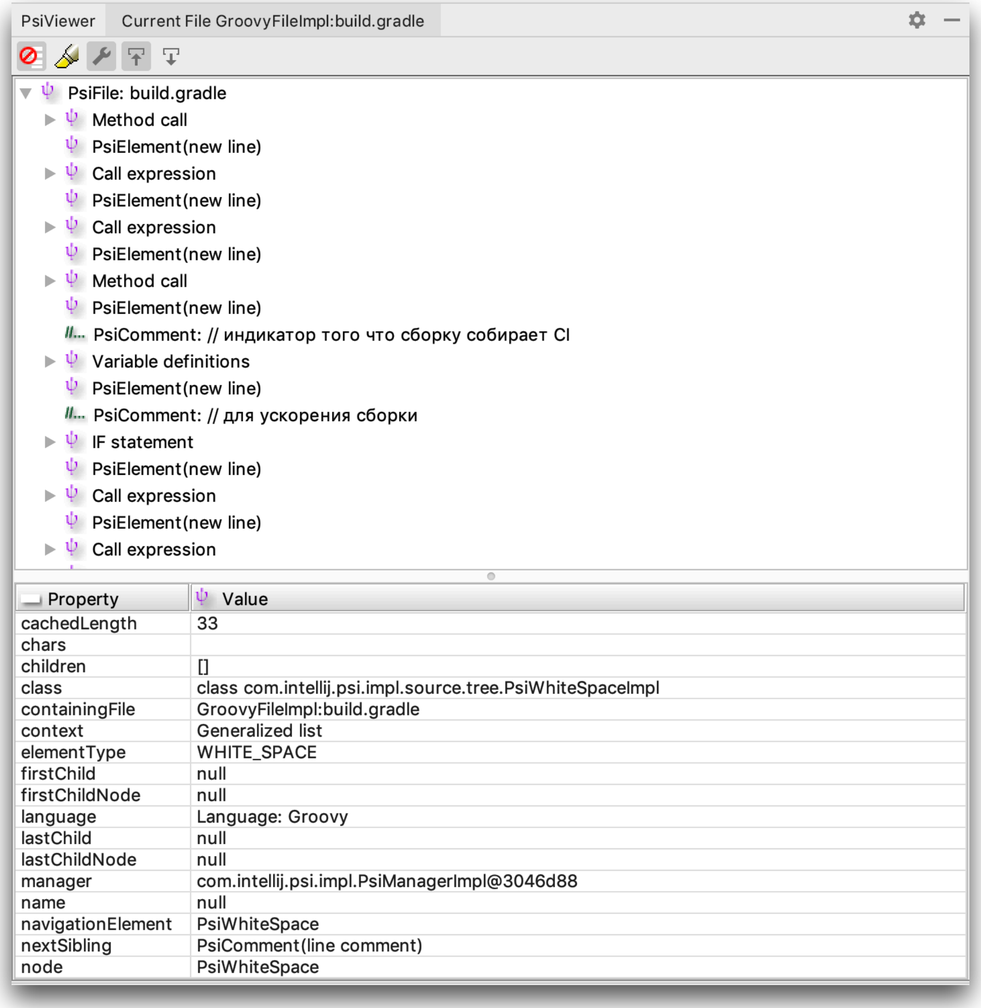
PsiViewer plugin

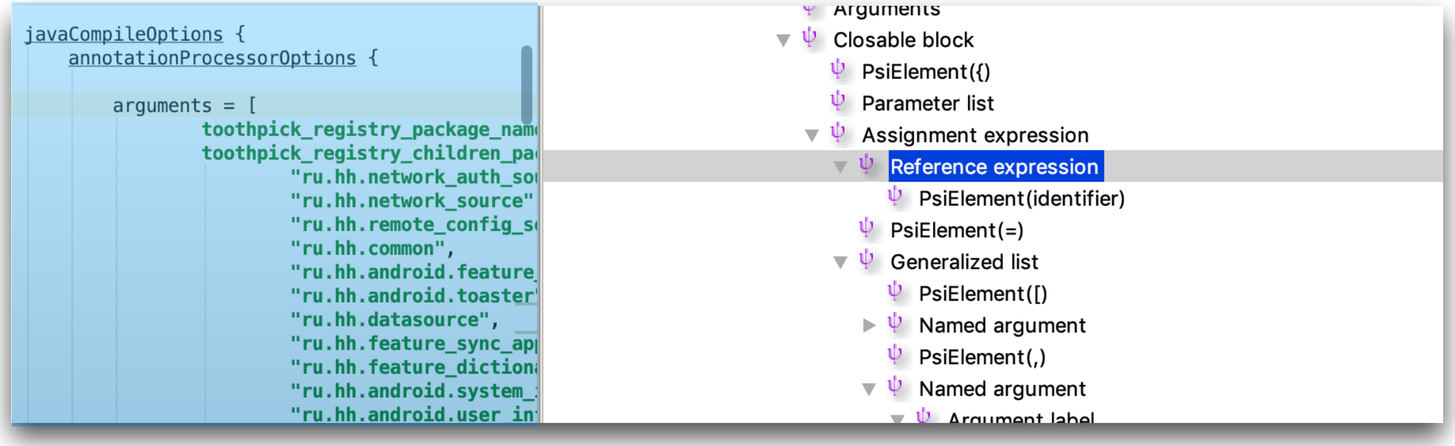




Поиск PsiFile | **Поиск PsiElement** | Создание строки | Вставка

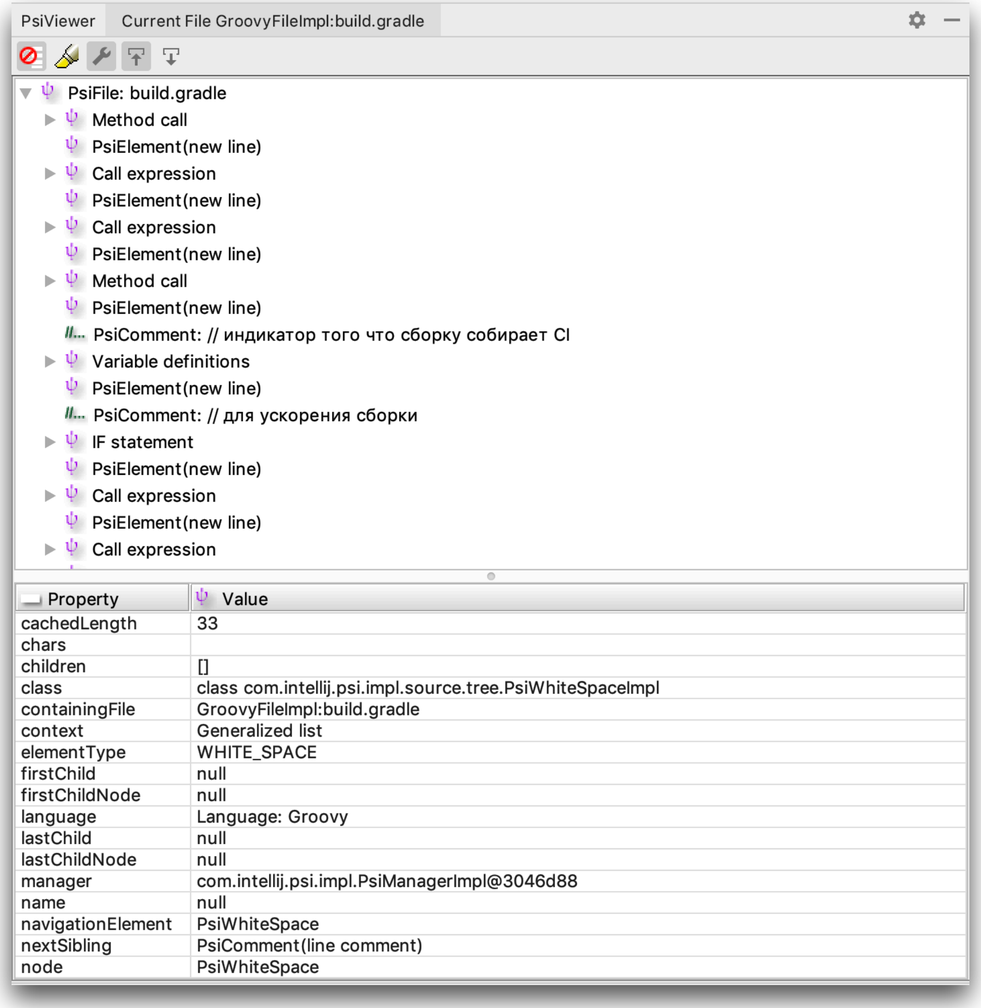
PsiViewer plugin

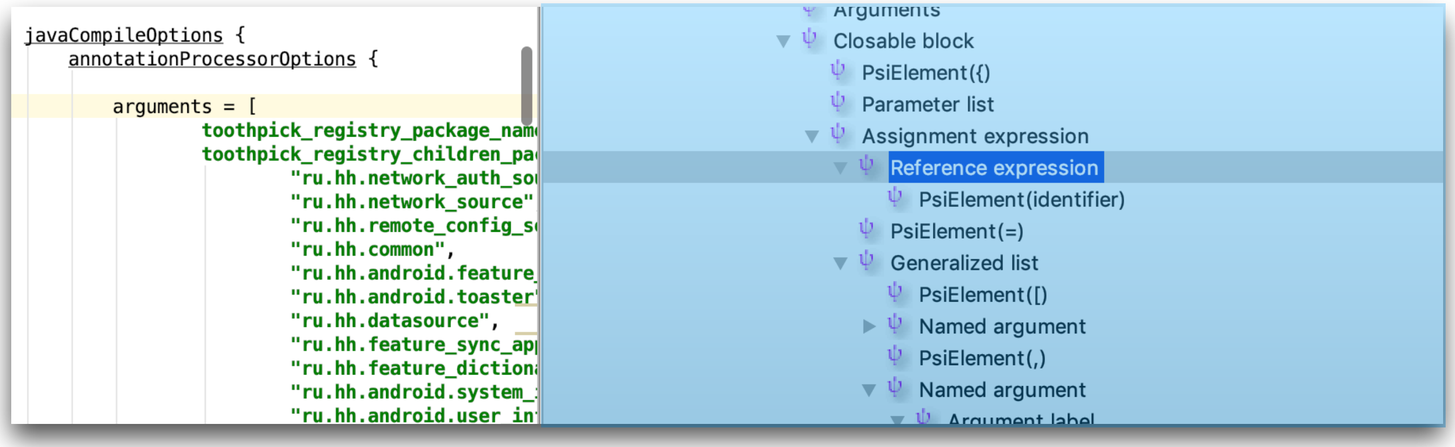




Поиск PsiFile | **Поиск PsiElement** | Создание строки | Вставка

PsiViewer plugin





Поиск PsiFile | **Поиск PsiElement** | Создание строки | Вставка

Ищем Psi-версию build.gradle

```
val appModule = ModuleManager.getInstance(project)
                        .modules.toList()
                        .first { it.name == "headhunter-applicant" }

val buildGradlePsiFile = FilenameIndex.getFilesByName(
                                appModule.project,
                                "build.gradle",
                                appModule.moduleContentScope
                            ).first()

...
```

Ищем нужный PsiElement

```
...

val targetPsiElement = buildGradlePsiFile.originalFile
    .collectDescendantsOfType<GrAssignmentExpression>()
    .firstOrNull { it.text.startsWith("arguments") }
    ?.lastChild
    ?.firstChildWithStartText("toothpick_registry_children_package_names")
    ?.collectDescendantsOfType<GrListOrMap>()
    ?.firstOrNull()
    ?: return

...
```

Создаем PsiElement — строку

```
JavaPsiFacade.getElementsFactory(project)
```

```
GroovyPsiElementFactory.getInstance(project)
```

```
KtPsiFactory(project)
```

Создаем PsiElement — строку

```
JavaPsiFacade.getElementsFactory(project)
```

```
GroovyPsiElementFactory.getInstance(project)
```

```
KtPsiFactory(project)
```

Ищем нужный PsiElement

```
...

val targetPsiElement = buildGradlePsiFile.originalFile
    .collectDescendantsOfType<GrAssignmentExpression>()
    .firstOrNull { it.text.startsWith("arguments") }
    ?.lastChild
    ?.firstChildWithStartText("toothpick_registry_children_package_names")
    ?.collectDescendantsOfType<GrListOrMap>()
    ?.firstOrNull()
    ?: return

...
```

Создаем PsiElement — строку

```
...

val factory = GroovyPsiElementFactory.getInstance(buildGradlePsiFile.project)

val packageName = config.mainParams.packageName
val newArgumentItem = factory.createStringLiteralForReference(packageName)

targetPsiElement.add(newArgumentItem)
```

Создаем PsiElement — строку

```
...

val factory = GroovyPsiElementFactory.getInstance(buildGradlePsiFile.project)

val packageName = config.mainParams.packageName
val newArgumentItem = factory.createStringLiteralForReference(packageName)

targetPsiElement.add(newArgumentItem)
```

Создаем PsiElement — строку

```
...

val factory = GroovyPsiElementFactory.getInstance(buildGradlePsiFile.project)

val packageName = config.mainParams.packageName
val newArgumentItem = factory.createStringLiteralForReference(packageName)

targetPsiElement.add(newArgumentItem)
```

Добавляем строку в PsiElement

```
...

val factory = GroovyPsiElementFactory.getInstance(buildGradlePsiFile.project)

val packageName = config.mainParams.packageName
val newArgumentItem = factory.createStringLiteralForReference(packageName)

targetPsiElement.add(newArgumentItem)
```

Модификация кода

1. Модифицировать settings.gradle
2. Настроить kapt для Toothpick
3. **Настроить kapt для Moxy**

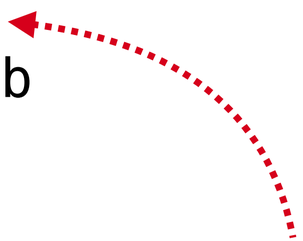
Модификация кода

1. Модифицировать settings.gradle
2. Настроить kapt для Toothpick
3. **Настроить kapt для Moxy**

Поиск PsiClass | Получить PsiElement | Добавить пакет | Пересоздание

MoxyReflectorStub

```
@RegisterMoxyReflectorPackages(
        "ru.hh.feature_chat.ui.moxy",
        "ru.hh.feature_webclient",
        "ru.hh.android.auth.applicant.ui.web_auth",
        ...
) class MoxyReflectorStub
```



Нужно добавить новый пакет сюда

Ищем MoxyReflectorStub

```
val appModule = ModuleManager.getInstance(project)
                    .modules.toList()
                    .first { it.name == "headhunter-applicant" }

val psiManager = PsiManager.getInstance(appModule.project)

val annotatedPsiClass = ClassUtil.findPsiClass(
    psiManager,
    "com.arellomobile.mvp.RegisterMoxyReflectorPackages"
)?.let { annotationPsiClass ->
    AnnotatedMembersSearch.search(
        annotationPsiClass,
        appModule.moduleContentScope
    ).findAll()
}?.firstOrNull() ?: return
```

Поиск PsiClass | Получить PsiElement | Добавить пакет | Пересоздание

Ищем MoxyReflectorStub

```
val appModule = ModuleManager.getInstance(project)
                        .modules.toList()
                        .first { it.name == "headhunter-applicant" }

val psiManager = PsiManager.getInstance(appModule.project)

val annotatedPsiClass = ClassUtil.findPsiClass(
    psiManager,
    "com.arellomobile.mvp.RegisterMoxyReflectorPackages"
)?.let { annotationPsiClass ->
    AnnotatedMembersSearch.search(
        annotationPsiClass,
        appModule.moduleContentScope
    ).findAll()
}?.firstOrNull() ?: return
```

Поиск PsiClass | Получить PsiElement | Добавить пакет | Пересоздание

Ищем MoxyReflectorStub

```
val appModule = ModuleManager.getInstance(project)
                    .modules.toList()
                    .first { it.name == "headhunter-applicant" }

val psiManager = PsiManager.getInstance(appModule.project)

val annotatedPsiClass = ClassUtil.findPsiClass(
    psiManager,
    "com.arellomobile.mvp.RegisterMoxyReflectorPackages"
)?.let { annotationPsiClass ->
    AnnotatedMembersSearch.search(
        annotationPsiClass,
        appModule.moduleContentScope
    ).findAll()
}?.firstOrNull() ?: return
```

Поиск PsiClass | Получить PsiElement | Добавить пакет | Пересоздание

Ищем MoxyReflectorStub

```
val appModule = ModuleManager.getInstance(project)
                    .modules.toList()
                    .first { it.name == "headhunter-applicant" }

val psiManager = PsiManager.getInstance(appModule.project)

val annotatedPsiClass = ClassUtil.findPsiClass(
    psiManager,
    "com.arellomobile.mvp.RegisterMoxyReflectorPackages"
)?.let { annotationPsiClass ->
    AnnotatedMembersSearch.search(
        annotationPsiClass,
        appModule.moduleContentScope
    ).findAll()
}?.firstOrNull() ?: return
```

Поиск PsiClass | Получить PsiElement | Добавить пакет | Пересоздание

Ищем MoxyReflectorStub

```
val appModule = ModuleManager.getInstance(project)
                    .modules.toList()
                    .first { it.name == "headhunter-applicant" }

val psiManager = PsiManager.getInstance(appModule.project)

val annotatedPsiClass = ClassUtil.findPsiClass(
    psiManager,
    "com.arellomobile.mvp.RegisterMoxyReflectorPackages"
)?.let { annotationPsiClass ->
    AnnotatedMembersSearch.search(
        annotationPsiClass,
        appModule.moduleContentScope
    ).findAll()
}?.firstOrNull() ?: return
```

Поиск PsiClass | Получить PsiElement | Добавить пакет | Пересоздание

Получаем содержимое аннотации

```
val annotatedPsiClass = ...

val annotationPsiElement = (annotatedPsiClass
    .annotations
    .first() as KtLightAnnotationForSourceEntry
).kotlinOrigin

val packagesPsiElements = annotationPsiElement
    .collectDescendantsOfType<KtValueArgumentList>()
    .first()
    .collectDescendantsOfType<KtValueArgument>()
```

Получаем содержимое аннотации

```
val annotatedPsiClass = ...

val annotationPsiElement = (annotatedPsiClass
    .annotations
    .first() as KtLightAnnotationForSourceEntry
).kotlinOrigin

val packagesPsiElements = annotationPsiElement
    .collectDescendantsOfType<KtValueArgumentList>()
    .first()
    .collectDescendantsOfType<KtValueArgument>()
```

Получаем содержимое аннотации

```
val annotatedPsiClass = ...

val annotationPsiElement = (annotatedPsiClass
    .annotations
    .first() as KtLightAnnotationForSourceEntry
).kotlinOrigin

val packagesPsiElements = annotationPsiElement
    .collectDescendantsOfType<KtValueArgumentList>()
    .first()
    .collectDescendantsOfType<KtValueArgument>()
```

Получаем содержимое аннотации

```
val packagesPsiElements = annotationPsiElement
    .collectDescendantsOfType<KtValueArgumentList>()
    .first()
    .collectDescendantsOfType<KtValueArgument>()

val updatedPackagesList = packagesPsiElements
                            .mapTo(mutableListOf()) { it.text }
                            .apply { this += "\"${config.packageName}\"" }
```

Получаем содержимое аннотации

```
val packagesPsiElements = annotationPsiElement
    .collectDescendantsOfType<KtValueArgumentList>()
    .first()
    .collectDescendantsOfType<KtValueArgument>()

val updatedPackagesList = packagesPsiElements
                            .mapTo(mutableListOf()) { it.text }
                            .apply { this += "\"${config.packageName}\"" }
```

Пересоздаем аннотацию

```
val updatedPackagesList = packagesPsiElements
                              .mapTo(mutableListOf()) { it.text }
                              .apply { this += "\"${config.packageName}\"" }

val newAnnotationValue = updatedPackagesList.joinToString(separator = ",\n")

val kotlinPsiFactory = KtPsiFactory(project)
val newAnnotationPsiElement = kotlinPsiFactory.createAnnotationEntry(
    "@RegisterMoxyReflectorPackages(\n$newAnnotationValue\n)"
)

val replaced = annotationPsiElement.replace(newAnnotationPsiElement)

CodeStyleManager.getInstance(module.project).reformat(replaced)
```

Пересоздаем аннотацию

```
val updatedPackagesList = packagesPsiElements
                              .mapTo(mutableListOf()) { it.text }
                              .apply { this += "\"${config.packageName}\"" }

val newAnnotationValue = updatedPackagesList.joinToString(separator = ",\n")

val kotlinPsiFactory = KtPsiFactory(project)
val newAnnotationPsiElement = kotlinPsiFactory.createAnnotationEntry(
    "@RegisterMoxyReflectorPackages(\n$newAnnotationValue\n)"
)

val replaced = annotationPsiElement.replace(newAnnotationPsiElement)

CodeStyleManager.getInstance(module.project).reformat(replaced)
```

Поиск PsiClass | Получить PsiElement | Добавить пакет | Пересоздание

Пересоздаем аннотацию

```
val updatedPackagesList = packagesPsiElements
                            .mapTo(mutableListOf()) { it.text }
                            .apply { this += "\"${config.packageName}\"" }

val newAnnotationValue = updatedPackagesList.joinToString(separator = ",\n")

val kotlinPsiFactory = KtPsiFactory(project)
val newAnnotationPsiElement = kotlinPsiFactory.createAnnotationEntry(
    "@RegisterMoxyReflectorPackages(\n$newAnnotationValue\n)"
)

val replaced = annotationPsiElement.replace(newAnnotationPsiElement)

CodeStyleManager.getInstance(module.project).reformat(replaced)
```

Поиск PsiClass | Получить PsiElement | Добавить пакет | Пересоздание

Пересоздаем аннотацию

```
val updatedPackagesList = packagesPsiElements
                            .mapTo(mutableListOf()) { it.text }
                            .apply { this += "\"${config.packageName}\"" }

val newAnnotationValue = updatedPackagesList.joinToString(separator = ",\n")

val kotlinPsiFactory = KtPsiFactory(project)
val newAnnotationPsiElement = kotlinPsiFactory.createAnnotationEntry(
    "@RegisterMoxyReflectorPackages(\n$newAnnotationValue\n)"
)

val replaced = annotationPsiElement.replace(newAnnotationPsiElement)

CodeStyleManager.getInstance(module.project).reformat(replaced)
```

Поиск PsiClass | Получить PsiElement | Добавить пакет | Пересоздание

Правим code style :-)

```
val updatedPackagesList = packagesPsiElements
                            .mapTo(mutableListOf()) { it.text }
                            .apply { this += "\"${config.packageName}\"" }

val newAnnotationValue = updatedPackagesList.joinToString(separator = ",\n")

val kotlinPsiFactory = KtPsiFactory(project)
val newAnnotationPsiElement = kotlinPsiFactory.createAnnotationEntry(
    "@RegisterMoxyReflectorPackages(\n$newAnnotationValue\n)"
)

val replaced = annotationPsiElement.replace(newAnnotationPsiElement)

CodeStyleManager.getInstance(module.project).reformat(replaced)
```

Поиск PsiClass | Получить PsiElement | Добавить пакет | Пересоздание

Резюме

1. Модификация кода возможна

Резюме

1. Модификация кода возможна
2. **Делайте ее через PSI**

Резюме

1. Модификация кода возможна
2. Делайте ее через PSI
3. **Помните, что PSI привязан к конкретному языку**

Что делать дальше?

Плагины позволяют автоматизировать создание модулей

1. Разрабатывать плагины - несложно (теперь)

Плагины позволяют автоматизировать создание модулей

1. Разрабатывать плагины - несложно (теперь)
2. Мы смогли ускорить создание модуля в 2 раза

Плагины позволяют автоматизировать создание модулей

1. Разрабатывать плагины - несложно (теперь)
2. Мы смогли ускорить создание модуля в 2 раза
3. Смотрите чужие плагины

Плагины позволяют автоматизировать создание модулей

1. Разрабатывать плагины - несложно (теперь)
2. Мы смогли ускорить создание модуля в 2 раза
3. Смотрите чужие плагины
4. Нужен утилитный класс - набирайте ...Util, ...Manager

Плагины позволяют автоматизировать создание модулей

1. Разрабатывать плагины - несложно (теперь)
2. Мы смогли ускорить создание модуля в 2 раза
3. Смотрите чужие плагины
4. Нужен утилитный класс - набирайте ...Util, ...Manager
5. Отлаживайте на маленьких проектах

Форкните наш проект на Гитхабе

Спросите меня о чем-нибудь :-)

1. Зачем плагин? Почему плагин?
2. Основы разработки плагинов
3. Внутренности IDEA: компоненты, PSI
4. UI, DI, генерация и модификация кода
5. Что делать дальше?

Ссылка на слайды