RuStore

Рыбаков Игорь, Android разработчик

Ковешников Евгений, Android разработчик

Как сделать библиотеку, чтобы ей пользовались

Конфликт зависимостей

Самая крутая библиотека

+

Проект с большим количеством уже существующих зависимостей

=

Разбор ошибок

Пример

```
FATAL EXCEPTION: main
Process: ru.vk.testshade_9879899, PID: 6592
java.lang.NoSuchMethodError: No virtual method getData()Ljava/lang/String; in class Lcom/example/library_c/LibC; or its super classes (declaration of 'com.example.library_c.LibC' appears in /data/app/~~Ct1l1AdorltAqhKPz7x7eA==/ru.vk
                                                                                                                                                                .testshade_9879899-v5UxKIliYaakyNOS__GCmA==/base.apk)
  at com.example.library_a.LibA.invokeLibC(LibA.kt:10)
  at ru.vk.testshade.MainActivity.onCreate(MainActivity.kt:15)
  at android.app.Activity.performCreate(Activity.java:8342)
  at android.app.Activity.performCreate(Activity.java:8321)
  at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1417)
  at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:3626)
  at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:3782)
  at android.app.servertransaction.LaunchActivityItem.execute(LaunchActivityItem.java:101)
  at android.app.servertransaction.TransactionExecutor.executeCallbacks(TransactionExecutor.java:138)
  at android.app.servertransaction.TransactionExecutor.execute(TransactionExecutor.java:95)
  at android.app.ActivityThread$H.handleMessage(ActivityThread.java:2307)
  at android.os.Handler.dispatchMessage(Handler.java:106)
  at android.os.Looper.loopOnce(Looper.java:201)
  at android.os.Looper.loop(Looper.java:288)
  at android.app.ActivityThread.main(ActivityThread.java:7924) <1 internal line>
  at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:548)
  at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:936)
```

Почему так происходит

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val resultA = LibA().invoke()
        val resultB = LibB().invoke()
    }
}
```

```
public class LibA {

    fun invoke(): String =
        LibC().getData()
}
```

```
public class LibB {

    fun invoke(): String =
        LibC().getData(version = 2)
}
```

```
public class LibC {

    public fun getData(): String =
        "version: 1"
}
```

```
public class LibC {

    public fun getData(version: Int): String =
        "version: $version"
}
```

Пример

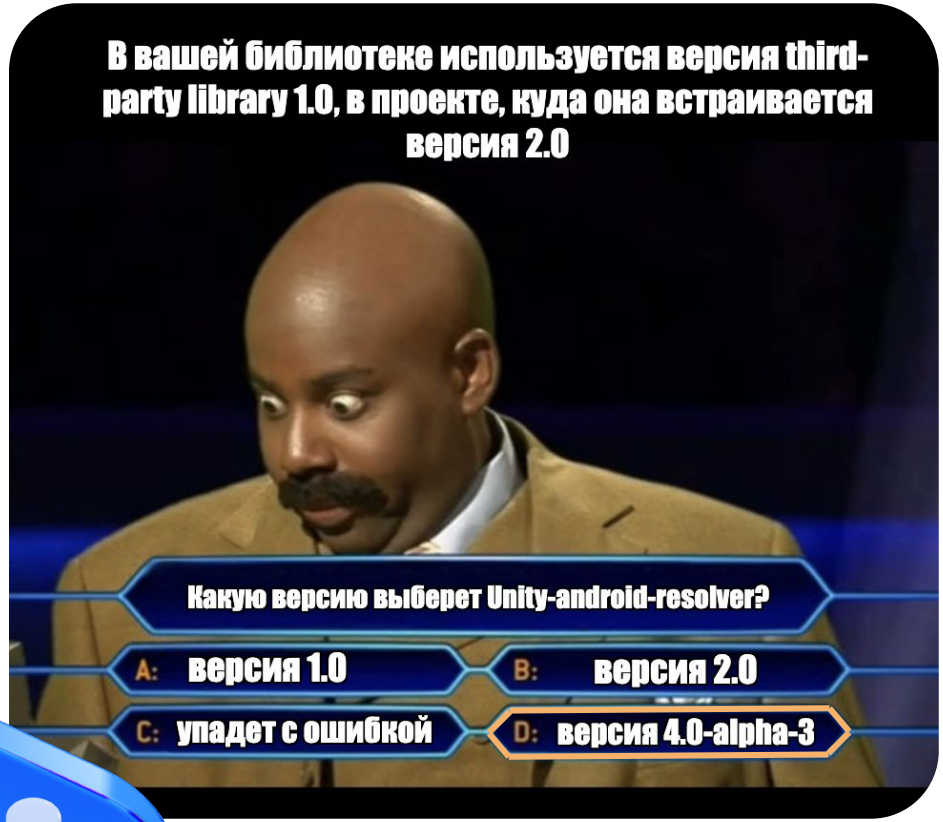
```
+--- ru.vk.testshade:library-a:{strictly 1.0} -> 1.0 (c)
+--- ru.vk.testshade:library-b:{strictly 1.0} -> 1.0 (c)
```

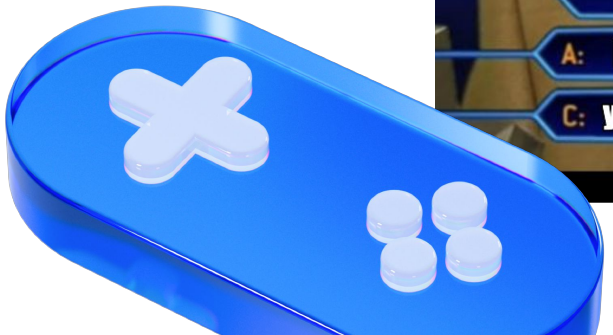
```
+--- ru.vk.testshade:library-b:1.0
|    +--- ru.vk.testshade:library-c:2.0 (*)
```

```
+--- ru.vk.testshade:library-a:1.0
|    +--- ru.vk.testshade:library-c:1.0 -> 2.0
```

Вопрос
в студию!

Вопрос в студию!





Почему возникают конфликты

- Библиотеки используется куча саб зависимостей (Retrofit, DI)
- Прекращается поддержка библиотеки после релиза и зависимости библиотеки устаревают
- Библиотека спроектирована под конкретный стек проекта и не проверялась с другими библиотеками

Сделать библиотеку, чтобы ей пользовались

Даже самая лучшая библиотека не будет использоваться, если для интеграции необходима переработка зависимостей проекта

Сложности интеграции отпугивают от внедрения*

* Никто не хочет утех с gradle на протяжении недели :)

Стратегии уменьшения внешних зависимостей

- Использование SourceSet
- Shadowing зависимостей
- Отказ от использования зависимостей
- Ограничение на использование зависимостей

Использование SourceSet

Использование SourceSet

- Увеличение времени сборки
- Самостоятельная поддержка актуальных версий
- Конфликт имен при компиляции
- Не у всех библиотек открыт исходный код

Отказ от использования зависимостей

Отказ от использования зависимостей

- Увеличение времени разработки
- Потенциально больше багов в проекте
- Высокая квалификация разработчиков
- Отказ от привычного стека

Ограничение на использование зависимостей

Ограничение на использование зависимостей

- Никто не читает документацию
- Не все соблюдают семантическое версионирование
- Поддержка документации в актуальном состоянии

Shadowing зависимостей

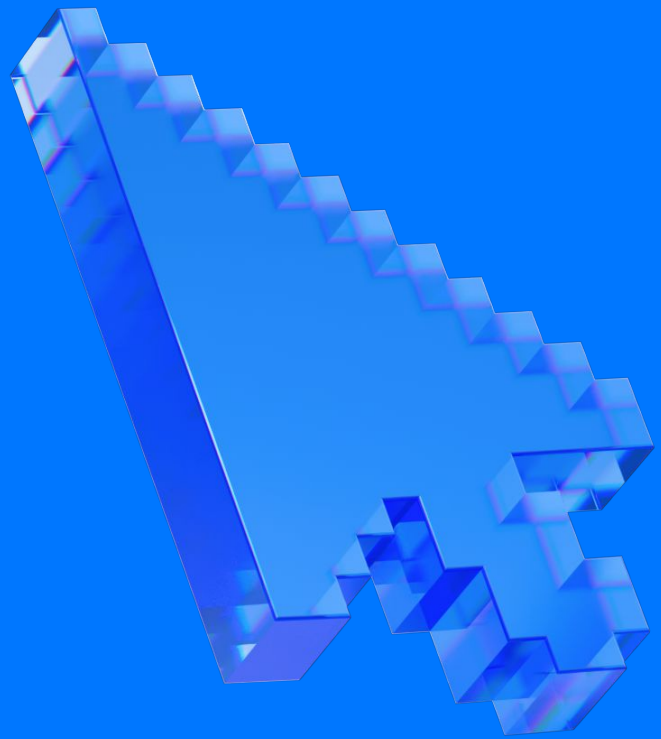
Shadowing зависимостей

- Увеличение веса конечной библиотеки
- Дополнительный gradle-plugin с необходимостью поддерживать AGP и версию Gradle
- Увеличение скорости сборки
- Существующие решения не поддерживаются разработчиками

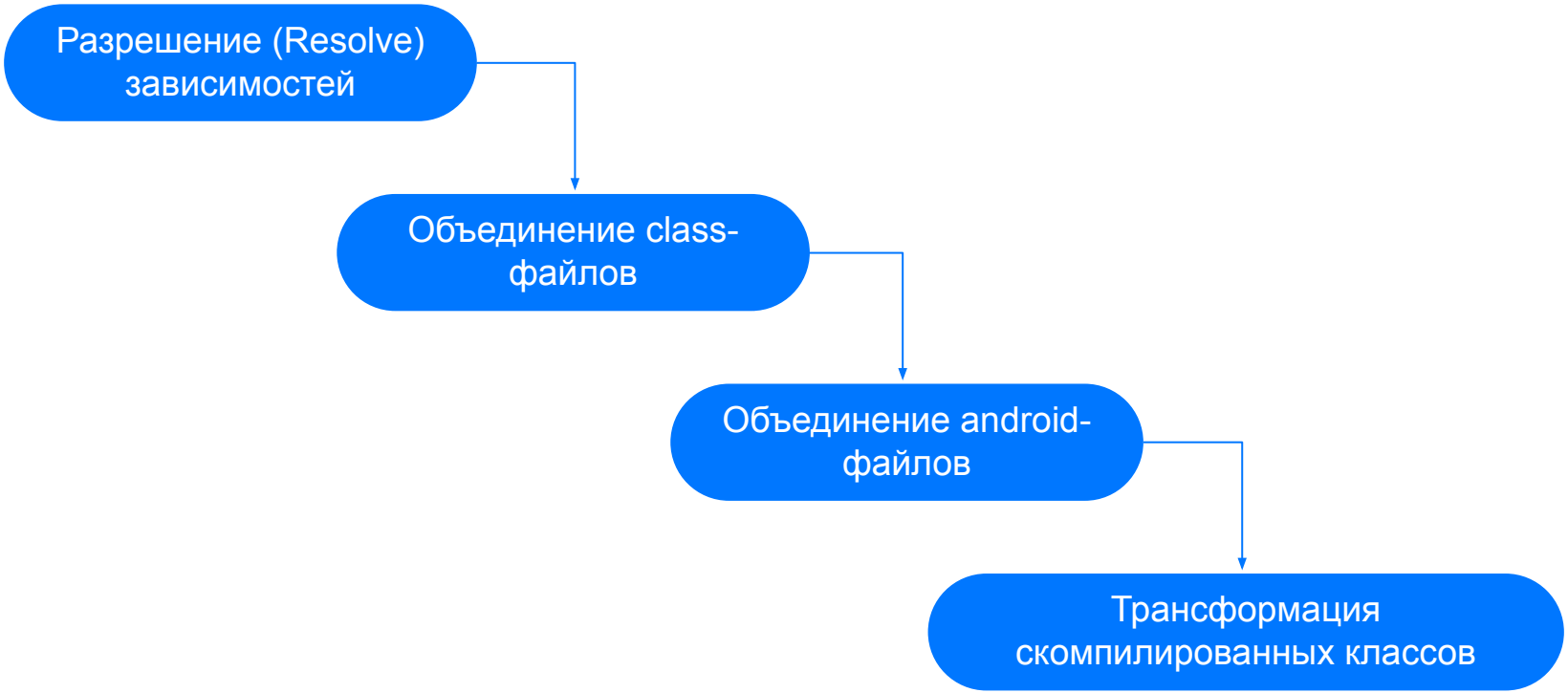
Что используем мы

Source set + Shadowing + Отказ от зависимостей

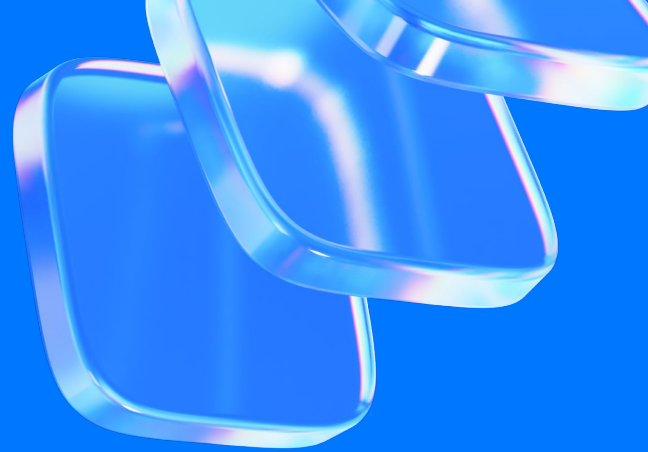
Как сделать плагин для Shadowing`а зависимостей?



Алгоритм работы плагина



1. Разрешение (Resolve) зависимостей
2. Объединение class-файлов
3. Объединение android-файлов:
    a. manifest-файлов
    b. resources, assets, jni файлов
    c. proguard и consumer-proguard файлов
4. Трансформация скомпилированных классов



Этап конфигурации

```
private val shadedConfigurations = mutableSetOf<Configuration>()
private val configurationVariants = mutableSetOf<LibraryVariant>() //заполняем на этапе конфигурации

buildTypes.names.forEach { buildTypeName ->
    val configName = buildTypeName + SHADED_CONFIG_NAME
    val configuration = project.configurations.create(configName)

    configuration.isTransitive = false
    project.gradle.addListener(ShadedConfigurationDependenciesInitializer(project, configuration))
    shadedConfigurations.add(configuration)
}

class ShadedConfigurationDependenciesInitializer(
    private val project: Project,
    private val configuration: Configuration,
) : DependencyResolutionListener {

    ...

    override fun beforeResolve(resolvableDependencies: ResolvableDependencies) {
        configuration.dependencies.onEach { dependency ->
                project.dependencies.add(compileOnlyConfigName, dependency)
            // все зависимости ставим как зависимости compileOnly конфигурации
        }
        project.gradle.removeListener(this)
    }
}
```

Этап получения зависимостей

```
target.afterEvaluate {
    configurationVariants.forEach { variant ->
        val shadedResolvedArtifacts = mutableSetOf<ResolvedArtifact>()

        shadedConfigurations
            .forEach { configuration ->
                val resolvedArtifact = mutableListOf<ResolvedArtifact>()
                configuration.resolvedConfiguration.resolvedArtifacts.forEach { artifact ->
                    if (ARTIFACT_TYPE_AAR != artifact.type && ARTIFACT_TYPE_JAR != artifact.type) {

                        throw ProjectConfigurationException(...)
                    }

                    resolvedArtifact.add(artifact)
                }
                shadedResolvedArtifacts.addAll(resolvedArtifact)
            }

        processConfigurationVariant(// Дальнейшая работа по встраиванию зависимостей
            ...
            shadedResolvedArtifacts = shadedResolvedArtifacts.toList(),
            ...
        )
}
```

1. Разрешение (Resolve) зависимостей
2. Объединение class-файлов
3. Объединение android-файлов:
    a. manifest-файлов
    b. resources, assets, jni файлов
    c. proguard и consumer-proguard файлов
4. Трансформация скомпилированных классов

Распаковка aar-зависимостей зависимостей

```
val unpackingFolder = File("<root_path>/intermediates/unpack_dir/<artifact_name>)

val listCopyTasks = resolvedArtifacts.map { resolvedArtifact ->
    unpackingFolder.mkdirs()

    project.tasks.register("unpackTaskName", Copy::class.java) { task ->
        task.from(project.zipTree(artifactFile.absolutePath))
        task.into(unpackingFolder)

        task.doFirst {
            unpackingFolder.delete()
        }
    }
}
```

Извлечение .class из распакованной AAR

```
class YourPlugin @Inject constructor(
    private val fileOperations: FileOperations,
) : Plugin<Project> {
    ...

    val outputDir = "${project.buildDir}/intermediates/unpack_dir/merge_classes/${variantName}"
    resolvedArtifacts
        .filter { unpackingFolder.exists() }
        .map { File(unpackingFolder, "classes.jar") }
        .forEach { jarFile ->
            fileOperations.copy { copySpec ->
                copySpec.from(fileOperations.zipTree(jarFile))
                copySpec.into(outputDir)
            }
        }
}
```

Извлечение .class из распакованной AAR из libs

```
class YourPlugin @Inject constructor(
    private val fileOperations: FileOperations,
) : Plugin<Project> {
    ...
    val outputDir = "${project.buildDir}/intermediates/unpack_dir/merge_classes/${variantName}"
    resolvedArtifacts
        .filter { unpackingFolder.exists() }
        .flatMap { File(unpackingFolder, "libs").listFiles() }
        .filter { jar -> jar.isFile && jar.name.endsWith(".jar") }
        .forEach { jarFile ->
            fileOperations.copy { copySpec ->
                copySpec.from(fileOperations.zipTree(jarFile))
                copySpec.into(outputDir)
                copySpec.exclude("META-INF")
            }
        }
}
```

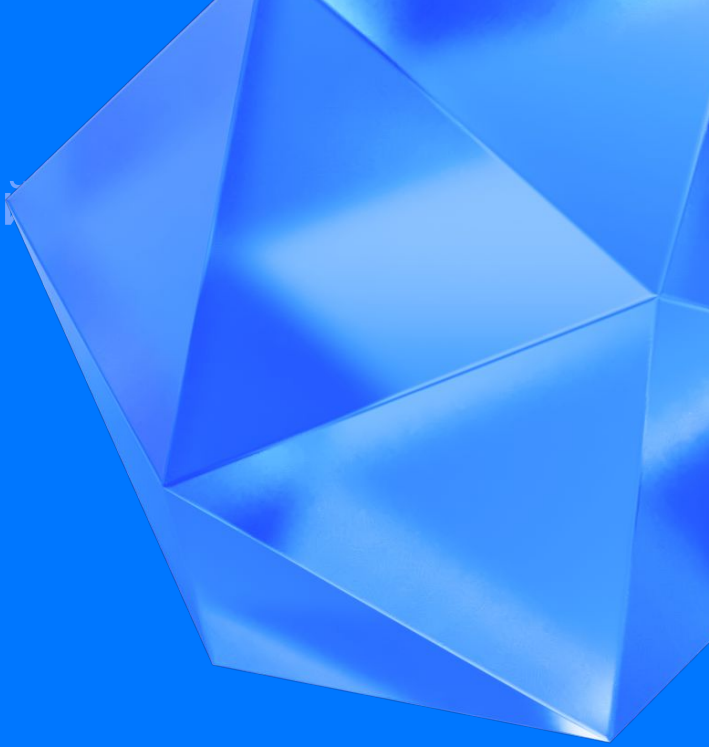
Извлечение .class из .jar зависимости

```
class YourPlugin @Inject constructor(
    private val fileOperations: FileOperations,
) : Plugin<Project> {
    ...

    val outputDir = "${project.buildDir}/intermediates/unpack_dir/merge_classes/${variantName}"
    resolvedArtifacts
        .filter { resolvedArtifacts.type == "jar" }
        .forEach { jarFile ->
            fileOperations.copy { copySpec ->
                copySpec.from(fileOperations.zipTree(jarFile))
                copySpec.into(outputDir)
                copySpec.exclude("META-INF")
            }
        }
    }
```

Объединение всех .class-файлов

```
val outputDir = "${project.buildDir}/intermediates/unpack_dir/merge_classes/${variantName}"
val javacDir = File("<buildRootDirectoryPath>/intermediates/javac/<buildVariantName>/classes","shadedPrefix")

fileOperations.copy { copySpec ->
    copySpec.from(outputDir)
    copySpec.into(javacDir)
    copySpec.exclude("META-INF/")
}

fileOperations.copy { copySpec ->
    copySpec.from("${outputDir.absolutePath}/META-INF")
    copySpec.into("<buildRootDirectoryPath>/tmp/kotlin-classes/<buildVariantName>/META-INF")
    copySpec.include(KOTLIN_MODULE)
}
```

1. Разрешение (Resolve) зависимостей
2. Объединение class-файлов
3. Объединение android-файлов:
    a. manifest-файлов
    b. resources, assets, jni файлов
    c. proguard и consumer-proguard файлов
4. Трансформация скомпилированных классов



Объединение Manifest файлов

```
abstract class MergeManifestTask : DefaultTask() {

    private var primaryManifestFile: File? = null

    private var dependenciesManifestFiles: List<File>? = null

    @TaskAction
    fun execute() {
        val mergerInvoker = ManifestMerger2.newMerger(
            primaryManifestFile,
            LoggerWrapper(logger),
            ManifestMerger2.MergeType.LIBRARY,
        )
        ...
    }
}
```

Объединение Manifest файлов

```
val manifestProviders = dependenciesManifestFiles
    ?.filter { it.exists() }
    ?.map { file ->
        object : ManifestProvider {
            override fun getManifest(): File = file.absoluteFile
            override fun getName(): String = file.name
        }
    }

mergerInvoker.addManifestProviders(manifestProviders)

val mergingReport = mergerInvoker.merge()
```

Объединение Manifest файлов

```
outputs.files.firstOrNull()?.let { outputFile ->
    BufferedWriter(
        OutputStreamWriter(
            FileOutputStream(outputFile), StandardCharsets.UTF_8.name(),
        ),
    ).apply {
        append(mergingReport.getMergedDocument(MergingReport.MergedManifestKind.MERGED))
        flush()
        close()
    }
}
```

1. Разрешение (Resolve) зависимостей
2. Объединение class-файлов
3. Объединение android-файлов:
    a. manifest-файлов
    b. resources, assets, jni файлов
    c. proguard и consumer-proguard файлов
4. Трансформация скомпилированных классов

Объединение resources, assets, jni файлов

```
val taskProvider = project.tasks.named("generate${variantName}<AndroidAdditionalFiles>")
* AndroidAdditionalFiles - где может принимать значение Resources, Assets, Jni
val unpackingFolder = File("<root_path>/intermediates/unpack_dir/<artifact_name>)

taskProvider.configure { task ->
        project.extensions.configure<LibraryExtension> {
        sourceSets.forEach { sourceSet ->
            if (sourceSet.name == variantName) {
                resolvedArtifacts.forEach { ->
                    sourceSet.res.srcDir(File(unpackingFolder, “res”)) // или
                    sourceSet.assets.srcDir(File(unpackingFolder, “assets”)) // или
                    sourceSet.jniLibs.srcDir(File(unpackingFolder, “jni”)) // или
                }
            }
        }
    }
}
```

1. Разрешение (Resolve) зависимостей
2. Объединение class-файлов
3. Объединение android-файлов:
    a. manifest-файлов
    b. resources, assets, jni файлов
    c. proguard и consumer-proguard файлов
4. Трансформация скомпилированных классов

Объединение proguard и consumer-proguard файлов

```
val taskProvider = project.tasks.named("merge${variantName}<ProguardFiles>")
* ProguardFiles - где может принимать значение ConsumerProguardFiles, GeneratedProguardFiles
val unpackingFolder = File("<root_path>/intermediates/unpack_dir/<artifact_name>)

mergeConsumerProguardFilesTask.configure { task ->
    task.doLast {
        val proguardFiles = resolvedArtifacts.map { File(unpackingFolder, "proguard.txt")}
        val outputFile = task.outputs.files.firstOrNull() ?: error("Result proguard file is null")
        if (!outputFile.exists()) {
            outputFile.createNewFile()
        }
        proguardFiles.forEach { proguardFile ->
            outputFile.appendText(proguardFile.readText())
        }
    }
}
```

1. Разрешение (Resolve) зависимостей
2. Объединение class-файлов
3. Объединение android-файлов:
    a. manifest-файлов
    b. resources, assets, jni файлов
    c. proguard и consumer-proguard файлов
4. Трансформация бинарных файлов

Трансформация бинарных файлов

В classes-файлов встраиваемых библиотек используется ссылка на старый R-файл, что приводит к крашам в runtime.

Для реализации shading-а зависимостей необходимо изменять пути до classes-файлов встраиваемых библиотек во всех бинарных файлах конечной библиотеки.

Возможности ASM

- модификация и генерация байт-кода
- доступ ко всей структуре класса
- доступ к аннотациям
- доступ к модулям

и т.д.

Создание Transformer-a

```
internal class PackageNameRepository : Serializable {

    private var packageNames: List<String> = listOf()

     fun get(): List<String> = packageNames

     fun set(names: List<String>) { packageNames = names }
}

internal interface ShadedTransformerParams : InstrumentationParameters {

    @get:Input
    val packageNameRepository: Property<PackageNameRepository>
}


internal abstract class ShadedRClassesTransformer : AsmClassVisitorFactory<ShadedTransformerParams> {
      ....
     val resolvedArtifactsPackageNames = parameters.get().packageNameRepository.get().get()
}
```

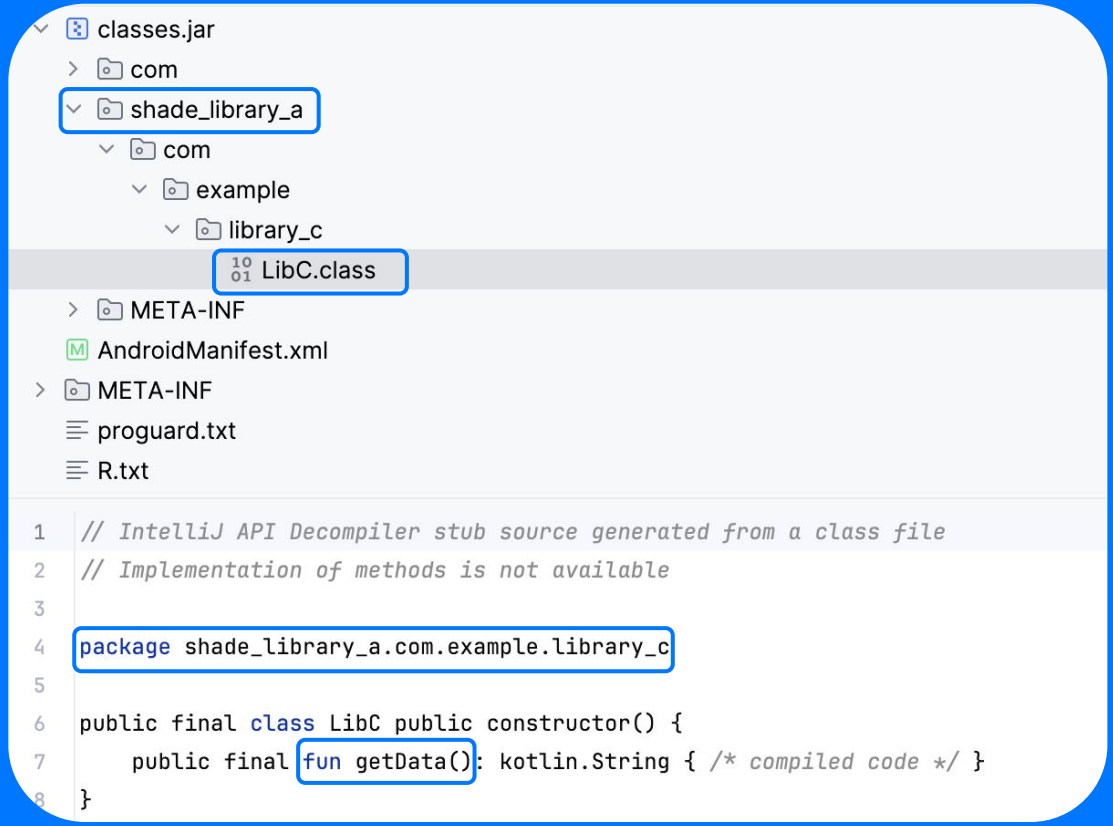
Регистрация трансформации

```
project.extensions.configure<LibraryAndroidComponentsExtension> {
  onVariants { libraryVariant ->
    libraryVariant.instrumentation.transformClassesWith(
      ShadedRClassesTransformer::class.java
      InstrumentationScope.PROJECT,
    ) { params ->
          params.packageNameRepository.set(packageNameRepository)
    }
  }
}
```

Реализация AnnotationVisitor

```
class ShadedAnnotationVisitor(
    visitor: AnnotationVisitor?,
    private val packageName: String,
    private val shadedPackagePrefix: String,
    private val resolvedArtifactsPackageNames: List<String>,
) : AnnotationVisitor(Opcodes.ASM9, visitor) {

    override fun visit(name: String?, value: Any?) {
        val shadedName = name.shadedValue(
            shadedPackagePrefix = shadedPackagePrefix,
            firstLevelLibraryPackageName = packageName,
            resolvedArtifactsPackageNames = resolvedArtifactsPackageNames,
        )

        val shadedValue = if (value is String?) {
            value.shadedValue(
                shadedPackagePrefix = shadedPackagePrefix,
                firstLevelLibraryPackageName = packageName,
                resolvedArtifactsPackageNames = resolvedArtifactsPackageNames,
            )
        } else {
            value
        }
        super.visit(shadedName, shadedValue)
    }

    override fun visitEnum(name: String?, descriptor: String?, value: String?)  {...}

    override fun visitAnnotation(name: String?, descriptor: String?): AnnotationVisitor {...}
}
```

Результаты встраивания



Результаты встраивания

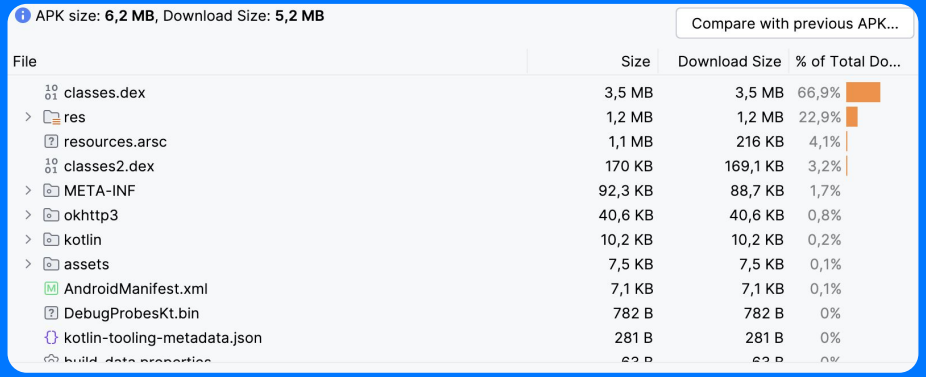
```
...
#25 = Utf8               shade_library_a/com/example/library_c/R$string
#26 = Class              #25            // shade_library_a/com/example/library_c/R$string
...
#51 = Fieldref           #47.#50        // com/example/library_a/R$string.lib_c_res:I

  public final java.lang.String getData();
    descriptor: ()Ljava/lang/String;
    flags: (0x0011) ACC_PUBLIC, ACC_FINAL
    Code:
      stack=3, locals=3, args_size=3
         0: aload_1
         1: ldc           #36                 //
         3: invokestatic  #42                 // Method
kotlin/jvm/internal/Intrinsics.checkNotNullParameter:(Ljava/lang/Object;Ljava/lang/String;)V
         6: new           #44                 // class java/lang/StringBuilder
         9: dup
        10: invokespecial #45                 // Method java/lang/StringBuilder."<init>":()V
        13: aload_1
        14: getstatic     #51                 // Field com/example/library_a/R$string.lib_c_res:I
        17: invokevirtual #57                 // Method android/content/Context.getString:(I)Ljava/lang/String;
        20: invokevirtual #61                 // Method java/lang/StringBuilder.append:(Ljava/lang/String;)Ljava/lang/StringBuilder;
        23: ldc           #63                 // String :
        25: invokevirtual #61                 // Method java/lang/StringBuilder.append:(Ljava/lang/String;)Ljava/lang/StringBuilder;
        28: iload_2
        29: invokevirtual #66                 // Method java/lang/StringBuilder.append:(I)Ljava/lang/StringBuilder;
        32: invokevirtual #70                 // Method java/lang/StringBuilder.toString:()Ljava/lang/String;
        35: areturn
```

Результаты shaded зависимости в библиотеку

File	Size	Download Size	% of Total Do...
classes.jar	63,2 KB	63,2 KB	99,3%
ru	60,5 KB	60,4 KB	95%
AndroidManifest.xml	309 B	309 B	0,5%
META-INF	122 B	122 B	0,2%
R.txt	2 B	2 B	0%

File	Size	Download Size	% of Total Do...
classes.jar	642 KB	642 KB	99,9%
shade_sdk_public_metrics	562,9 KB	562 KB	87,5%
ru	58 KB	57,9 KB	9%
META-INF	186 B	186 B	0%
AndroidManifest.xml	309 B	309 B	0%
META-INF	122 B	122 B	0%
proguard.txt	2 B	2 B	0%
R.txt	2 B	2 B	0%

Результаты встраивания библиотеки в проект



APK size: **6,2 MB**, Download Size: **5,2 MB**

Compare with previous APK...

File	Size	Download Size	% of Total Do...
classes.dex	3,5 MB	3,5 MB	66,9%
res	1,2 MB	1,2 MB	22,9%
resources.arsc	1,1 MB	216 KB	4,1%
classes2.dex	170 KB	169,1 KB	3,2%
META-INF	92,3 KB	88,7 KB	1,7%
okhttp3	40,6 KB	40,6 KB	0,8%
kotlin	10,2 KB	10,2 KB	0,2%
assets	7,5 KB	7,5 KB	0,1%
AndroidManifest.xml	7,1 KB	7,1 KB	0,1%
DebugProbesKt.bin	782 B	782 B	0%
kotlin-tooling-metadata.json	281 B	281 B	0%

APK size: **6,2 MB**, Download Size: **5,2 MB**

Compare with previous APK...

File	Size	Download Size	% of Total Do...
classes.dex	3,5 MB	3,5 MB	66,9%
res	1,2 MB	1,2 MB	22,9%
resources.arsc	1,1 MB	216 KB	4,1%
classes2.dex	170 KB	169,1 KB	3,2%
META-INF	92,3 KB	88,7 KB	1,7%
okhttp3	40,6 KB	40,6 KB	0,8%
kotlin	10,2 KB	10,2 KB	0,2%
assets	7,5 KB	7,5 KB	0,1%
AndroidManifest.xml	7,1 KB	7,1 KB	0,1%
DebugProbesKt.bin	782 B	782 B	0%
kotlin-tooling-metadata.json	281 B	281 B	0%

Выводы

- Думайте о том, как пользователям проще внедрить вашу библиотеку
- Выбирайте те стратегии уменьшения зависимостей, которые удобны вам
- Если для вашего проекта необходимы индивидуальные инструменты,
то не бойтесь их создавать

Готовы ответить на ваши вопросы!

TG-канал для разработчиков

TG-чат с разработчиками