Оно само: используем плагины компиляции

Анна Жаркова
Lead Mobile developer

Обо мне

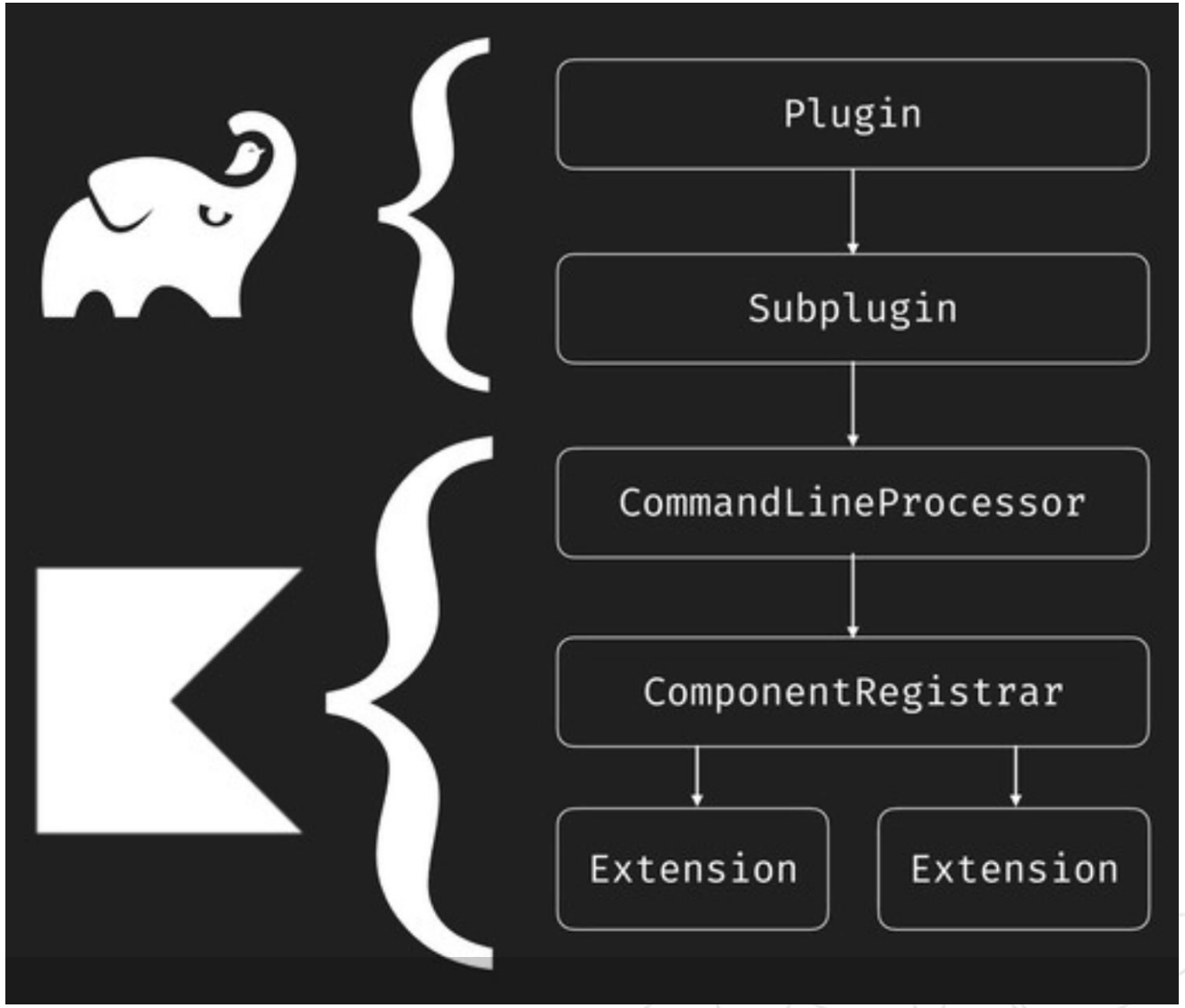
- В мобильной разработке с 2013
- Ведущий мобильный разработчик в Usetech
- Нативная разработка под iOS и Android (Swift/Objective-C, Kotlin/Java) кросс-платформа (Xamarin, Kotlin multiplatform)
- Ментор, управляю командой направления
- Спикер на конференциях AppsConf, Mobius, TechTrain, DroidCon (2022)
- Преподаватель в Otus (iOS Pro и базовый)
- Автор статей по мобильной разработке (SwiftUI, iOS, KMM)

Обсудим:

- Kotlin compiler plugins. Принцип работы.
- Анатомия и настройка плагина
- Кейс 1. Генерация логики (Usecase)
- Кейс 2. Работаем с UI. Делаем свой ViewBinding
- Из View в Compose (спойлер: лучше KSP)

KCP – Kotlin Plugin Compiler

Мощный инструмент для изменения текущего кода во время компиляции



Что могут плагины компилятора

- Доступ к абстрактному синтаксическому дереву
- Изменяют существующий код
- Мультиплатформенное решение

Для чего плагины компилятора

- Модифицировать internal/private код
- Мета-программирование (анализ код с помощью аннотаций)
- Кодогенерация

Оптимизация бойлерплейта

Для чего плагины компилятора

- Модифицировать internal/private код
- Мета-программирование (анализ код с помощью аннотаций)
- Кодогенерация

Оптимизация бойлерплейта

Когда не подойдут плагины компилятора

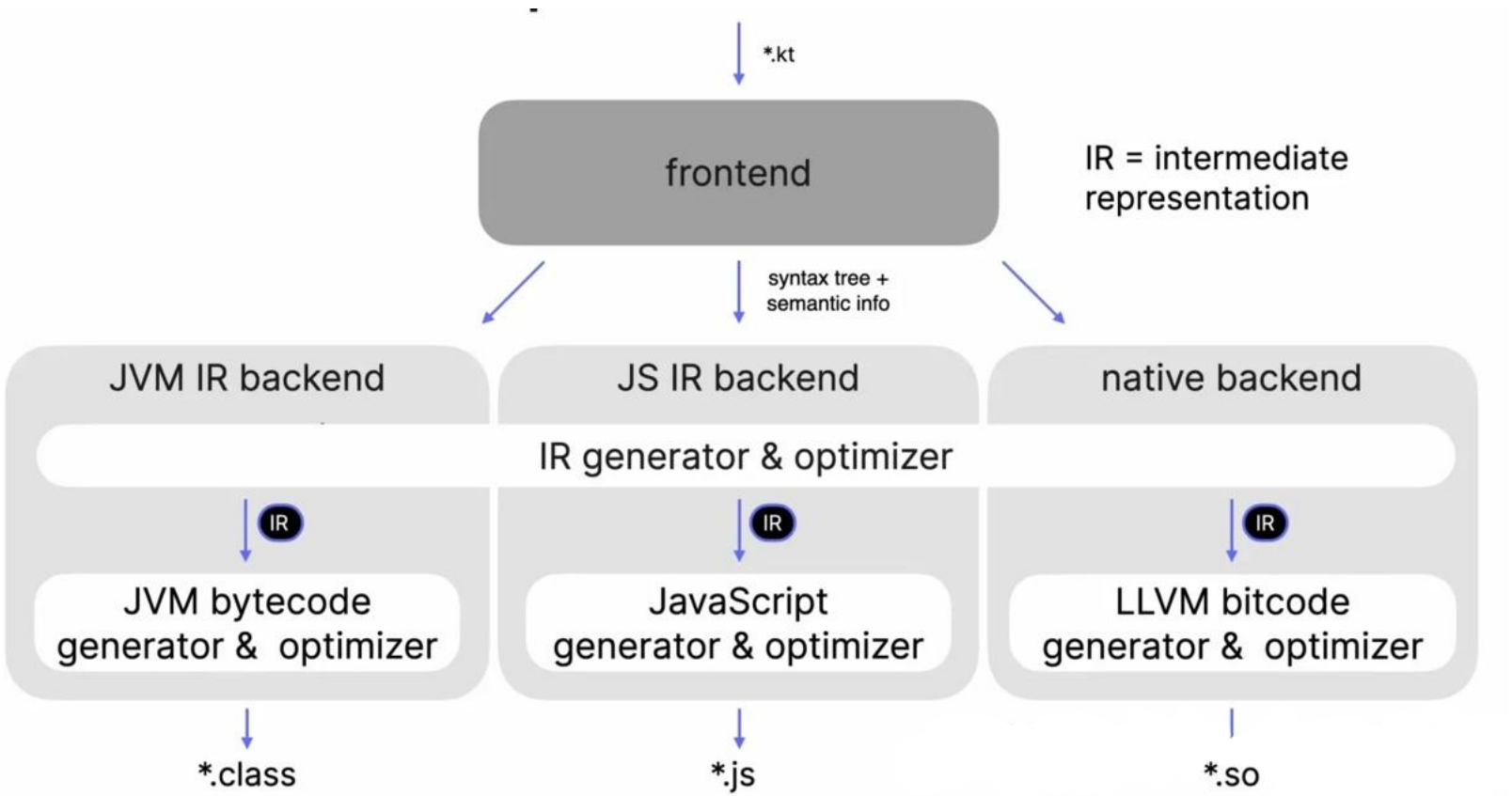
Когда их можно заменить процессингом аннотаций:

- Анализ и генерация только нового кода
- Совмещение с другими плагинами компиляции (KSP идет до своих плагинов компилятора)

Или слишком много работы

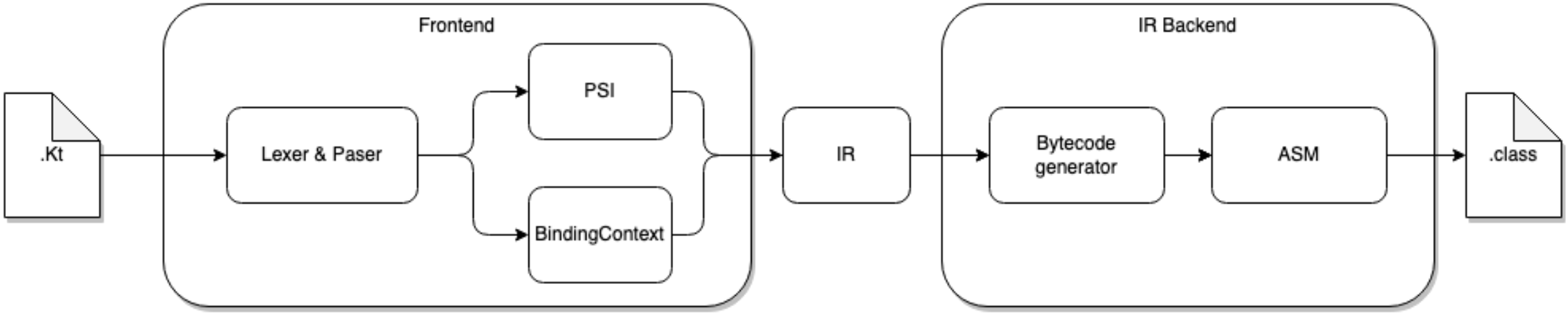
Попробуем свои силы, но сначала посмотрим, что внутри

Компиляция IR



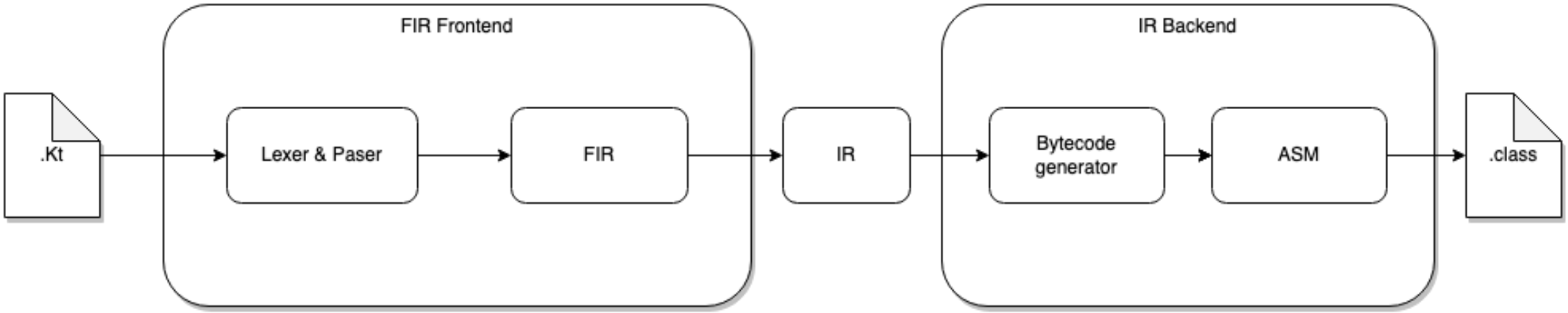
Компиляция.K1

IR backend по умолчанию с Kotlin 1.5.0

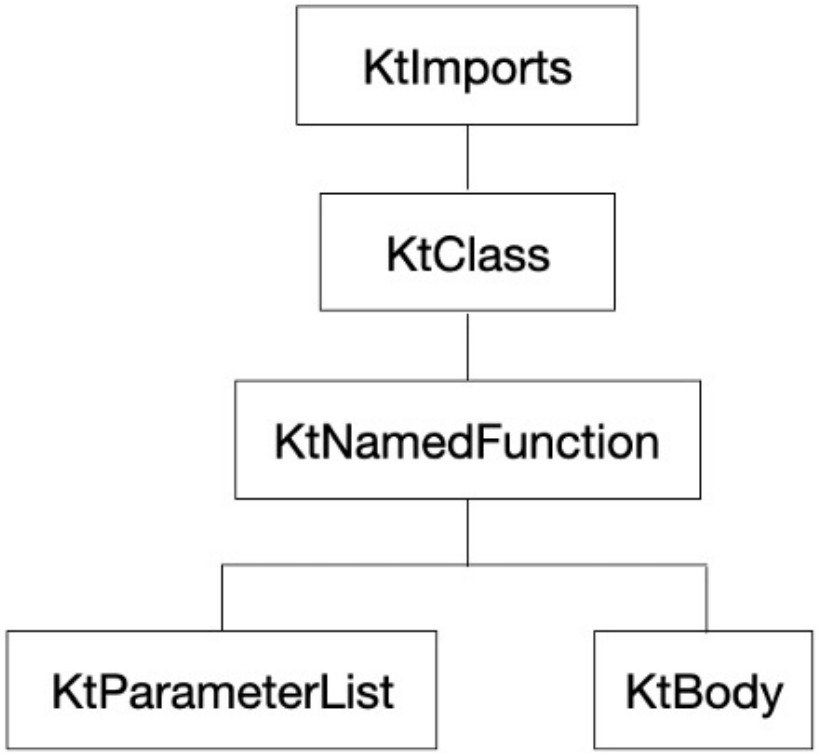


Компиляция FIR&IR.K2

FIR совмещает PSI и Binding Condext



Programming Structure Interface (PSI)

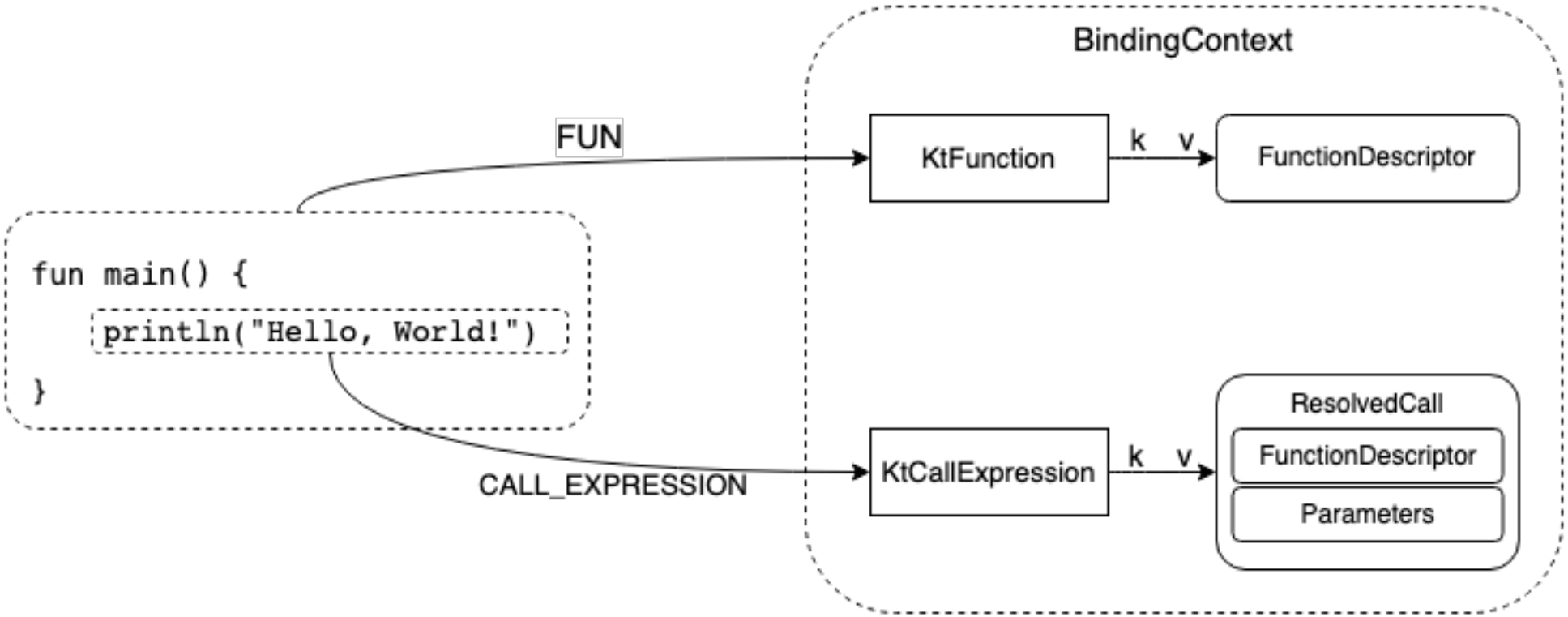


IR, пример кода

```
//Код
suspend fun test() {
    println(returnsInt())
}

// Kotlin IR
FUN name:test visibility:public modality:FINAL <> () returnType:kotlin.Unit [suspend]
  BLOCK_BODY
    CALL 'public final fun println (message: kotlin.Int): kotlin.Unit [inline]
         declared in kotlin.io.ConsoleKt' type=kotlin.Unit origin=null
      message: CALL 'public final fun returnsInt (): kotlin.Int [suspend]
                    declared in <root>.TestKt' type=kotlin.Int origin=null
```

IR обработка AST

Элементы PSI - > дескрипторы



Kotlin Compiler plugin. Способы обработки данных

- ClassLoweringPass/ FileLoweringPass
- IrVisitor
- IrElementTransformer

ClassLoweringPass

Доступ к данным через класс

```
interface ClassLoweringPass : FileLoweringPass {
    fun lower(irClass: IrClass)
    override fun lower(irFile: IrFile) = runOnFilePostfix(irFile)
}

fun ClassLoweringPass.runOnFilePostfix(irFile: IrFile) {
    irFile.acceptVoid(ClassLoweringVisitor(this))
}
```

ClassLoweringPass

ClassLoweringVisitor – рекурсивный обход элементов

```
private class ClassLoweringVisitor(
    private val loweringPass: ClassLoweringPass
) : IrElementVisitorVoid {
    override fun visitElement(element: IrElement) {
        element.acceptChildrenVoid(this)
    }

    override fun visitClass(declaration: IrClass) {
        declaration.acceptChildrenVoid(this)
        loweringPass.lower(declaration)
    }
}
```

IrElementTransformer

Прямой доступ к дескрипторам через метод

```
//базовый интерфейс
interface IrElementTransformer<in D> : IrElementVisitor<IrElement, D>

//Примеры вызовов
fun visitClass(declaration: IrClass, data: D)
fun visitElement(element: IrElement, data: D)
fun visitField(declaration: IrField, data: D)
fun visitProperty(declaration: IrProperty, data: D)
fun visitCall(expression: IrCall, data: D)
```

Что выбрать

Обработка элемента зависит от атрибутов класса

ClassLoweringPass

Обработка элемента зависит от атрибутов элемента

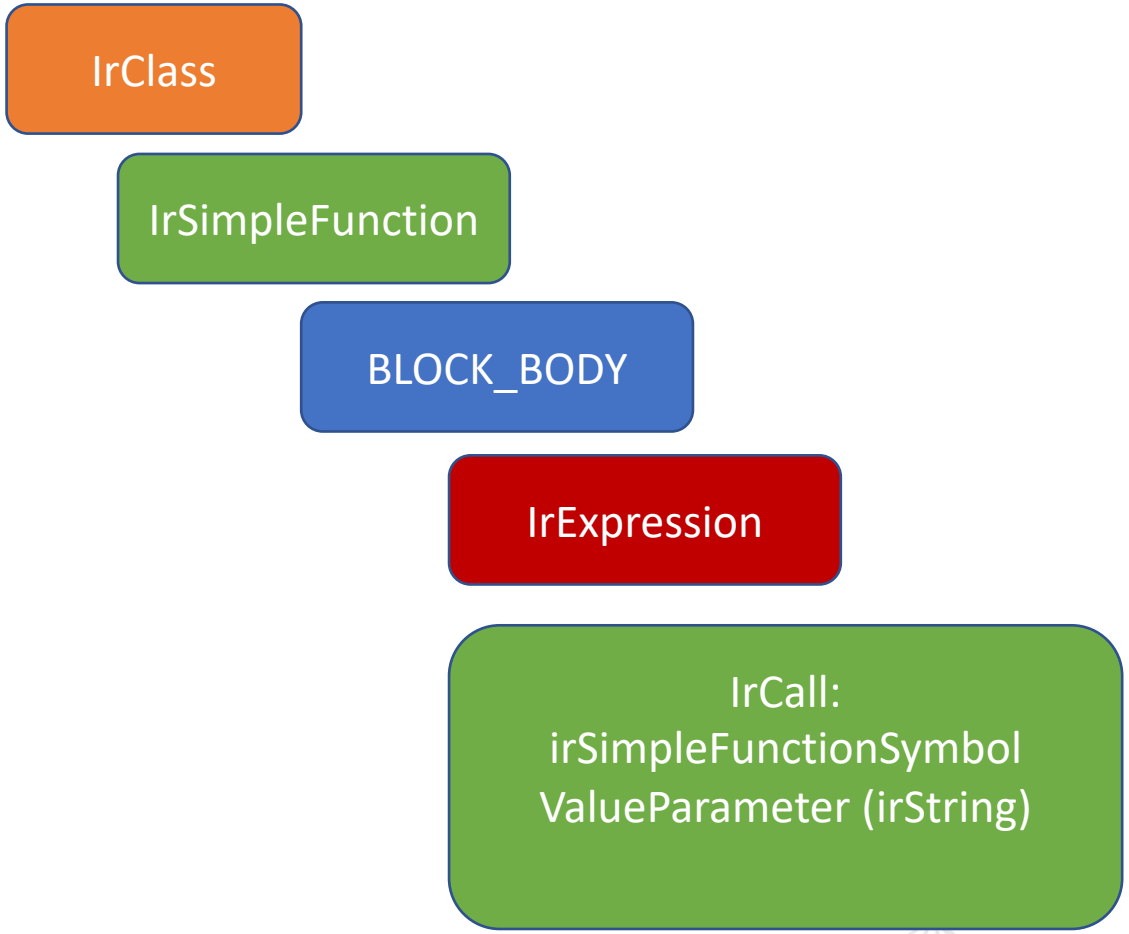
IrElementTransformer

Пример 0

```
//До
class Test

//После
class Test {
    fun test() {
        print("Hello")
    }
}
```

Пример 0

```
//До
class Test

//После
class Test {
    fun test() {
        print("Hello")
    }
}
```



Пример 0

```
override fun lower(irClass: IrClass) {
    val testFunction = irClass.addFunction{
        name = Name.identifier("test")
        returnType = irBuiltIns.unitType
    }.apply {
        this.parent = irClass
        this.dispatchReceiverParameter = irClass.thisReceiver?.copyTo(this)
        this.body = DeclarationIrBuilder(pluginContext,this.symbol).irBlockBody {
            +irReturn(irCall(funPrintln!!.symbol).also{
                it.putValueArgument(0,irString("Hello"))
            })
        }
    }
}
```

От теории к настоящей практике

Практические кейсы

Мета-программирование для оптимизация однотипных задач:

- Хелперы (сериализаторы, parcelize)
- Инжекция (DI)
- Бизнес-логика
- UI

Практические кейсы

- Хелперы (сериализаторы, parcelize)
- Инжекция (DI)
- Автогенерация логики (usecase)
- UI (ViewBinding, Compose)

DI на Compiler plugins

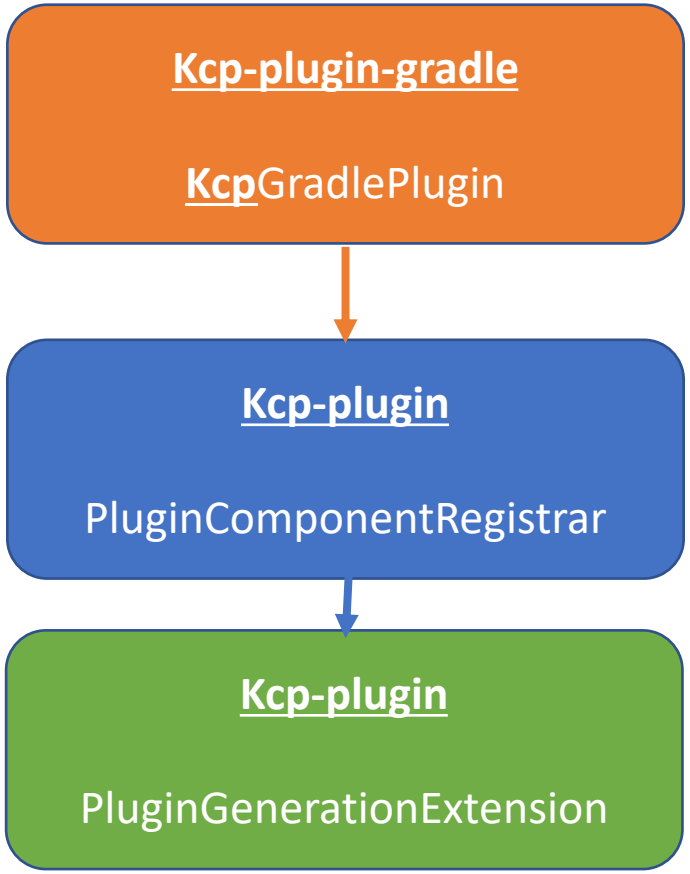
Практические кейсы

- Автогенерация (usecase)
- UI (ViewBinding, Compose)

От простого к сложному

Начнем с настройки и подключения

Архитектура плагина



Создаем KotlinCompilerPluginSupportPlugin

```
class KcpGradlePlugin : KotlinCompilerPluginSupportPlugin {
    override fun isApplicable(kotlinCompilation: KotlinCompilation<*>): Boolean = true

    override fun getCompilerPluginId(): String = PLUGIN_ID

    override fun getPluginArtifact(): SubpluginArtifact = SubpluginArtifact(
        groupId = GROUP_ID,
        artifactId = ARTEFACT_ID,
        version = VERSION
    )
    override fun applyToCompilation(kotlinCompilation: KotlinCompilation<*>):
    Provider<List<SubpluginOption>> {
        return kotlinCompilation.target.project.provider {
            val options:List<SubpluginOption> = mutableListOf()
            options
        }
    }
}
```

Настраиваем плагин. Kcp-plugin-gradle

Kcp_plugin_gradle build.gradle.kts

```
plugins {
    id("java-gradle-plugin")
    kotlin("jvm")
    id("maven-publish")
}
group = "com.azharkova.kcp.plugin"
version = VERSION
dependencies {
    implementation(kotlin("gradle-plugin-api"))
}
gradlePlugin {
    plugins {
        create("kcp_plugin") {
            id = "kcp_plugin"
            implementationClass = "com.azharkova.kcp.plugin.KcpGradlePlugin"
        }
    }
}
```

Настраиваем плагин. Kcp-plugin

Kcp_plugin build.gradle.kts

```
dependencies {
    compileOnly("org.jetbrains.kotlin:kotlin-compiler-embeddable:1.8.10")
    compileOnly("org.jetbrains.kotlin:kotlin-compiler:1.8.10")
    kapt("com.google.auto.service:auto-service:1.0-rc7")
    compileOnly("com.google.auto.service:auto-service-annotations:1.0-rc7")
}

group = "com.azharkova.kcp.plugin"
version = VERSION
```

Настраиваем плагин. Сам проект

build.gradle.kts

```
buildscript {
    dependencies {
        classpath("com.android.tools.build:gradle:7.4.2")
        classpath("com.azharkova.kcp.plugin:kcp_plugin:0.1.2")
    }
}


plugins {
    id("org.jetbrains.kotlin.jvm") version "1.8.0" apply false
    id("kcp_plugin") version "0.1.2" apply true
    id("org.javamodularity.moduleplugin") version "1.8.12" apply false
}
```

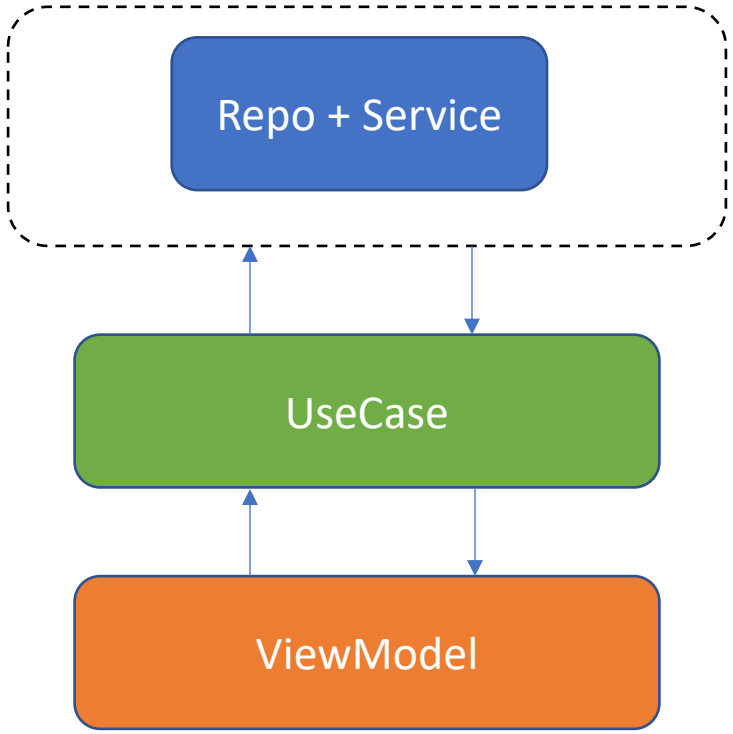
CompilerPluginRegistrar

Подключаем расширения @AutoService(CompilerPluginRegistrar)

```
@OptIn(ExperimentalCompilerApi::class)
@AutoService(CompilerPluginRegistrar::class)
class PluginComponentRegistrar() : CompilerPluginRegistrar() {

    override val supportsK2: Boolean
        get() = false

    override fun ExtensionStorage.registerExtensions(configuration: CompilerConfiguration) {
        val messageCollector = configuration.get(CLIConfigurationKeys.MESSAGE_COLLECTOR_KEY, MessageCollector.NONE)

        SyntheticResolveExtension.registerExtension(SyntethicExtension())
        IrGenerationExtension.registerExtension(AndroidGenerationExtenstion(messageCollector = messageCollector))
        IrGenerationExtension.registerExtension(PluginGenerationExtension(messageCollector = messageCollector))
    }
}
```

Пример 1. Оптимизируем логику. Usecase

Usecase. Обертка запросов к Api



Usecase generation

```
//До генерации
@GenUseCase(repo = NewsApi::class, request = "loadNews")
class NewsLoadCase


//После генерации
class NewsLoadCase {
    companion object {
        fun usecase() = $usecase
    }

    object $usecase : GenericUseCase<Unit, NewsList> {
        //Тут можно вставить вызов DI
        val repo by lazy {
            NewsApiImpl()
        }
        override suspend fun execute(param: Unit?): NewsList {
            return repo.loadNews()
        }
    }
}
```

Usecase. Обертка запросов к Api

Repo – интерфейс API

Request – ключ к методу API

```
@Target(AnnotationTarget.CLASS)
@Retention(AnnotationRetention.SOURCE)
annotation class GenUseCase(val repo: KClass<*>, val request: String)

//Использование
@GenUseCase(repo = NewsApi::class, request = "loadNews")
class NewsLoadCase
```

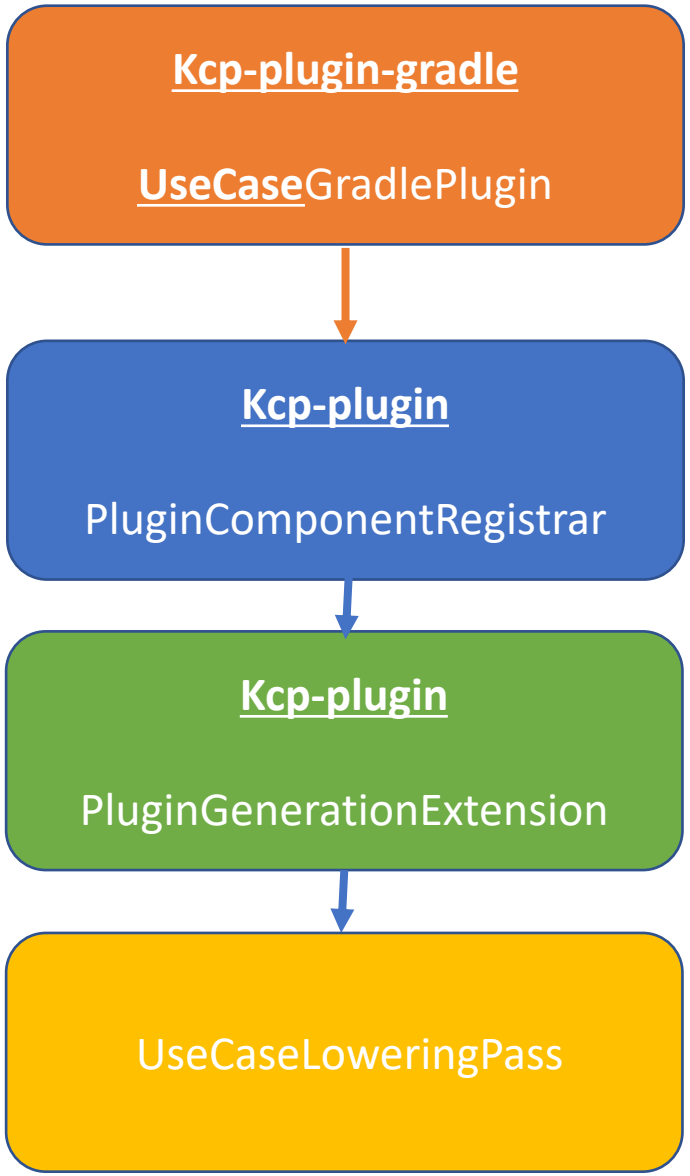
Usecase. Обертка запросов к Api. Базовый

Назначение: класс-обертка для вызова запросов Api

```
abstract public class CoroutineUseCase<in T, out R>(
    val dispatcher: CoroutineDispatcher = ioDispatcher
) {

    /**
    * Метод для реализации
    */
    abstract suspend fun execute(param: T): R

     suspend operator fun invoke(param: T): Result<R> = withContext(dispatcher) {
         runCatching { execute(param) }
     }
}
```

Usecase generation

@GenUsecase



Регистрация расширения

IrGenerationExtension

```
class PluginGenerationExtension (private val messageCollector: MessageCollector)
    : IrGenerationExtension {

    override fun generate(moduleFragment: IrModuleFragment, pluginContext: IrPluginContext) {
        UseCaseLoweringPass(pluginContext, messageCollector).lower(moduleFragment)
    }
}
```

Kotlin Compiler plugin. Ограничения

Тип данных/путь к методу должен быть известен плагину.

- Тип в том же модуле,
- Явный путь хардкод/из параметра.
- Через дескриптор

```
Caused by: java.lang.NullPointerException
    at com.azharkova.kmm.plugin.domain.SyntethicExtensionKt
    .createGetterDescriptor(SyntethicExtension.kt:85)
    at com.azharkova.kmm.plugin.domain.SyntethicExtension
    .generateSyntheticMethods(SyntethicExtension.kt:53)
    at org.jetbrains.kotlin.resolve.extensions.SyntheticResolveExtension$Companion$getInstance$1
    .generateSyntheticMethods(SyntheticResolveExtension.kt:96)
```

Абстракция не работает

- Тип должен быть указан явно и точно. Указание через родительский тип замещает реализацией по умолчанию

```
Compilation failed: Internal compiler error:
no implementation found for FUN name:execute visibility:public
modality:ABSTRACT <> ($this:com.azharkova.core.GenericUseCase<T of com.azharkova.core.GenericUseCase,
R of com.azharkova.core.GenericUseCase>, param:T of com.azharkova.core.GenericUseCase?)
returnType:R of com.azharkova.core.GenericUseCase [suspend]when building itable for CLASS
INTERFACE name:GenericUseCase modality:ABSTRACT visibility:public superTypes:[kotlin.Any]
implementation in CLASS OBJECT name:$usecase modality:FINAL visibility:public superTypes:
[com.azharkova.core.GenericUseCase<kotlin.Any, com.azharkova.kmmkspcases.data.NewsList>]
```

Usecase generation

Базовый интерфейс + вызов execute в расширении

Полная реализация в базовом родителе обращается к методам родителя без переопределения

```
interface GenericUseCase<T : Any,R: Any> {
    suspend fun execute(param: T? = null): R
}

//Выносим в расширение
suspend fun<T:Any,R:Any> GenericUseCase<T,R>.request(param: T? = null): Result<R>
= withContext(ioDispatcher) {
    runCatching {
        execute(param)
    }
}
```

Входная точка

- Существующий метод без body или свойство nullable

Тип должен уже существовать

```
class NewsLoadCase {
    companion object {
        fun usecase(): NewsLoadCaseImpl? = null
    }

}
```

Входная точка

- Сгенерированная точка

$object – внутренний синглтон

```
class NewsLoadCase {
    companion object {
        fun usecase(): GenericUseCase<Unit,NewsList> = $object
    }
}
```

Синтетические расширения. Companion

- Создаем companion для классов с условием
- fun usecase()

```
class SyntethicExtension(): SyntheticResolveExtension {

//Companion
    override fun getSyntheticCompanionObjectNameIfNeeded(thisDescriptor: ClassDescriptor)
    : Name? =
        if (thisDescriptor.isUsecase)   Names.DEFAULT_COMPANION else null

//Entry point
    override fun getSyntheticFunctionNames(thisDescriptor: ClassDescriptor): List<Name> =
        if (thisDescriptor.isCompanionObject && thisDescriptor.isUsecaseCompanion) {
            listOf(Names.USECASE_METHOD)
        } else {
            emptyList()
        }
```

Синтетические расширения. Заглушки

Генерируем getter для результата метода-заглушки

```
override fun generateSyntheticMethods(
    thisDescriptor: ClassDescriptor,name: Name,
    bindingContext: BindingContext,
    fromSupertypes: List<SimpleFunctionDescriptor>,
    result: MutableCollection<SimpleFunctionDescriptor>
) {
    if (name != Names.USECASE_METHOD) return
    val classDescriptor = getForCompanion(thisDescriptor) ?: return

    //Генерируем геттер и добавляем
    result.add(createGetterDescriptor(thisDescriptor, classDescriptor))
}
```

Синтетические расширения. Заглушки

Генерируем getter для результата метода-заглушки

```
fun createGetterDescriptor(
    companionClass: ClassDescriptor,
    clazz: ClassDescriptor,
    params: List<KotlinType>
): SimpleFunctionDescriptor {
    return SimpleFunctionDescriptorImpl.create(
        companionClass, Annotations.EMPTY,
        Names.USECASE_METHOD,CallableMemberDescriptor.Kind.SYNTHESIZED,
        companionClass.source).apply {

    val returnType = KotlinTypeFactory.simpleNotNullType(/**...*/)

    this.initialize(
        null, companionClass.thisAsReceiverParameter,
        emptyList(),emptyList(),returnType,
        Modality.FINAL,DescriptorVisibilities.PUBLIC )
    }
}
```

Синтетические расширения. Тип результата

- Результирующий тип на основе параметров

```
fun createGetterDescriptor(
    companionClass: ClassDescriptor,
    clazz: ClassDescriptor,
    params: List<KotlinType>
): SimpleFunctionDescriptor {
    return SimpleFunctionDescriptorImpl.create(
        companionClass, Annotations.EMPTY,
        Names.USECASE_METHOD,CallableMemberDescriptor.Kind.SYNTHESIZED,
        companionClass.source).apply {

    val returnType = KotlinTypeFactory.simpleNotNullType(/**...*/)

    this.initialize(
        null, companionClass.thisAsReceiverParameter,
        emptyList(),emptyList(),returnType,
        Modality.FINAL,DescriptorVisibilities.PUBLIC )
    }
}
```

Синтетические расширения. Тип результата

- Результирующий тип на основе параметров
- Generic

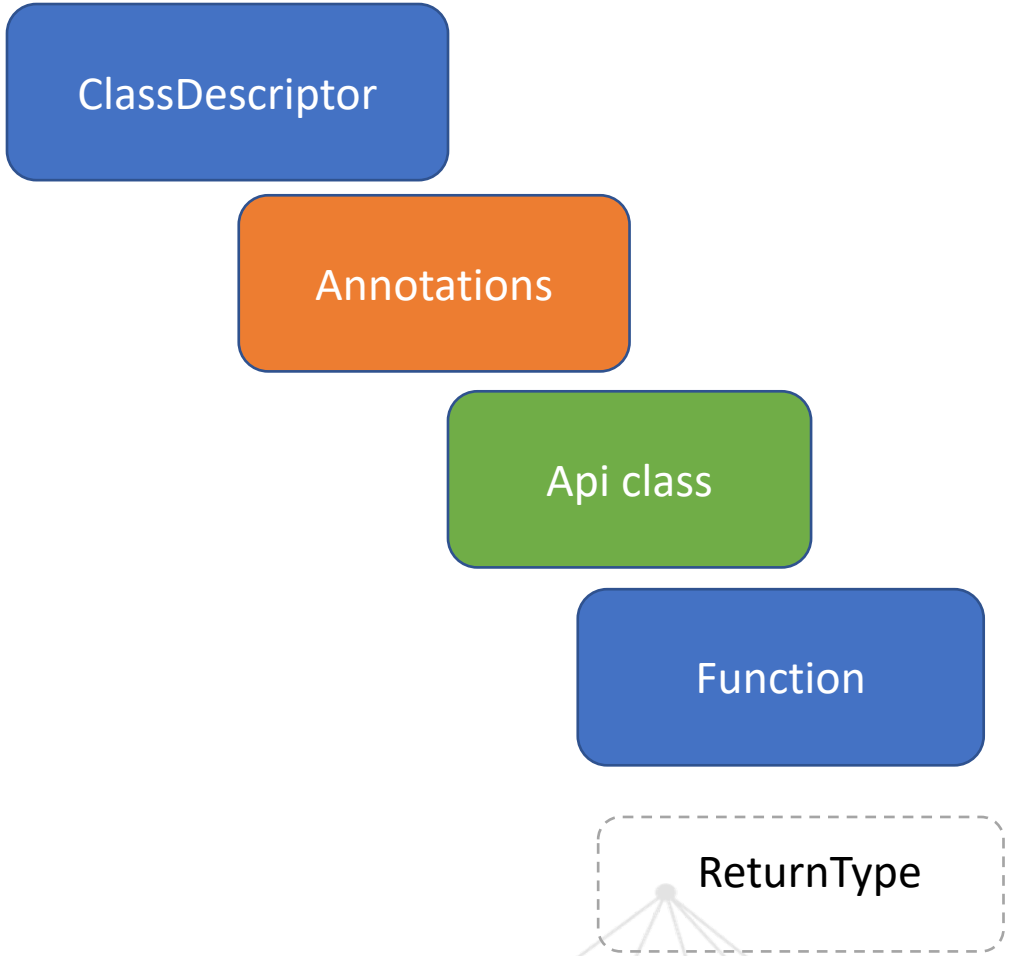
Параметры скрыты???

```
val returnType = KotlinTypeFactory.simpleNotNullType(
        TypeAttributes.Empty,
        usecaseClass,
        emptyList()
        params.map {
            TypeProjectionImpl(it)
        }
    )
```

Generic. Проблема результирующего типа (наш кейс)

Тип зависит от типа функции –>
поиск через label

Bad Performance



Проблема портирования на мультиплатформу

Интероп objective-C/Kotlin Generic
https://kotlinlang.org/docs/native-objc-interop.html#generics

```
Compilation failed: Internal compiler error: no implementation found
for FUN name:execute visibility:public modality:ABSTRACT <>
($this:com.azharkova.core.GenericUseCase<T of com.azharkova.core.GenericUseCase,
R of com.azharkova.core.GenericUseCase>, param:T of com.azharkova.core.GenericUseCase?)
returnType:R of com.azharkova.core.GenericUseCase [suspend]
when building itable for CLASS INTERFACE name:GenericUseCase modality:ABSTRACT
visibility:public superTypes:[kotlin.Any]
implementation in CLASS OBJECT name:$usecase modality:FINAL visibility:public
superTypes:[com.azharkova.core.GenericUseCase<kotlin.Any, com.azharkova.kmmkspcases.data.NewsList>]
at
CLASS OBJECT name:$usecase modality:FINAL visibility:public
superTypes:[com.azharkova.core.GenericUseCase<kotlin.Any, com.azharkova.kmmkspcases.data.NewsList>]

 * Source files:
 * Compiler version info: Konan: 1.8.0 / Kotlin: 1.8.0
 * Output kind: FRAMEWORK
```

Usecase generation. Дубль 2

```
@GenUseCase(repo = NewsApi::class, request = "loadNews")
class NewsLoadCase

//После генерации
class NewsLoadCase {
    companion object {
        fun usecase() = $usecase
    }

    object $usecase : SuspendUseCase {
        val repo by lazy {
            NewsApiImpl()
        }

        override suspend fun execute(param: Any?): Any {
            return repo.loadNews()
        }

    }
}
```

Usecase generation. Дубль 2

Базовый интерфейс + вызов execute в расширении

Без generics

```
public interface  SuspendUseCase {

    suspend fun execute(param: Any?): Any
}

public suspend fun SuspendUseCase.request(param: Any? = null): Result<*>
= withContext(Dispatchers.Default) {
    runCatching {
        execute(param)
    }
}
```

GenUseCase.Class lowering

```
class UseCaseLoweringPass(private val pluginContext: IrPluginContext,
private val messageCollector: MessageCollector) :
    ClassLoweringPass {
    /**
    Декларации
    */

    //Преобразования
    override fun lower(irClass: IrClass) {
        if (!irClass.toIrBasedDescriptor().isUseCase()) {
            return
        }
        /*** Наша работа
        *
        */
    }
}
```

GenUseCase.Class lowering

```
class UseCaseLoweringPass(private val pluginContext: IrPluginContext,
private val messageCollector: MessageCollector) :
    ClassLoweringPass {
    /**
    Декларации
    */
    override fun lower(irClass: IrClass) {
        if (!irClass.toIrBasedDescriptor().isUseCase()) {
            return
        }
        //Получаем ссылку на параметры из аннотации Repo и ParamIn
        irClass.retrieveParams()
        //Функция запроса из API
        requestFunction =
            apiImplType?.getClass()?.functions?.firstOrNull { it.name.asString() == request }
        //Входной параметр функции из API
        paramIn = requestFunction?.valueParameters?.firstOrNull()
        //Добавляем внутренний класс object:SuspendUseCase
        val implClass = irClass.addUseCaseImpl()
        //Добавляем lazy свойство для API
        implClass.addRepoLazyFunc()
        //Добавляем
        implClass.addExecutionFunction()
        //Соединяем companion основного класса и объект внутреннего класса
        generateFactory(irClass.companionObject() as IrClass, implClass)
        messageCollector.report(CompilerMessageSeverity.WARNING, irClass.dump())
    }
/**
Магия
*/
```

GenUseCase.Class lowering

1. Ссылки на параметры из аннотации Repo, метод
2. Функция запроса из API
3. Типы параметров
4. Внутренний класс object:SuspendUseCase
5. Добавляем lazy свойство для API
6. Override execute
7. Соединяем с Companion

GenUseCase. Параметры из аннотации

Вытаскиваем параметры из аннотации

```
private fun IrClass.retrieveParams() {
        this.toIrBasedDescriptor().annotations
            .firstOrNull { it.fqName == usecaseName }?.let {
                it.allValueArguments.forEach { (name, value) ->
                //Это ссылка на тип Api интерфейса
                    if (name.asString() == "repo") {
                        apiType =
                            pluginContext.referenceClass((value.value as NormalClass).classId)
                                ?.defaultType
                    }
                    //Грубый парсинг из NormalClass
                    if (name.asString() == "request") {
                        request = value.toString().replace("\"", "")
                    }
                }
            }
```

GenUseCase. Параметры из аннотации

Для запросов к Api нам потребуется ссылка на класс.

```
        //А это ссылка на тип класса Api с реализацией методов
        apiImplType = pluginContext.referenceClass(
            FqName(
                apiType?.classFqName?.asString().orEmpty() + "Impl"
            )
        )!!.defaultType
    }
```

GenUseCase. Целевая функция

Функция вызова и тип входного параметра

```
//Функция запроса из API
requestFunction =
    apiImplType?.getClass()?.functions?
    .firstOrNull { it.name.asString() == request }
//Входной параметр функции из API
paramIn = requestFunction?.valueParameters?.firstOrNull()
```

GenUseCase. Внутренний класс

Object $usecase :SuspendUseCase

addChild – родитель исходный класс

```
val useCaseCls = irFactory.buildClass{
    name = Names.USECASE_IMPL
    modality = Modality.FINAL
    visibility = DescriptorVisibilities.PUBLIC
    kind = ClassKind.OBJECT
}
//SuspendUseCase
useCaseCls.superTypes = listOf(suspendUseCaseType)

this.addChild(useCaseCls)
```

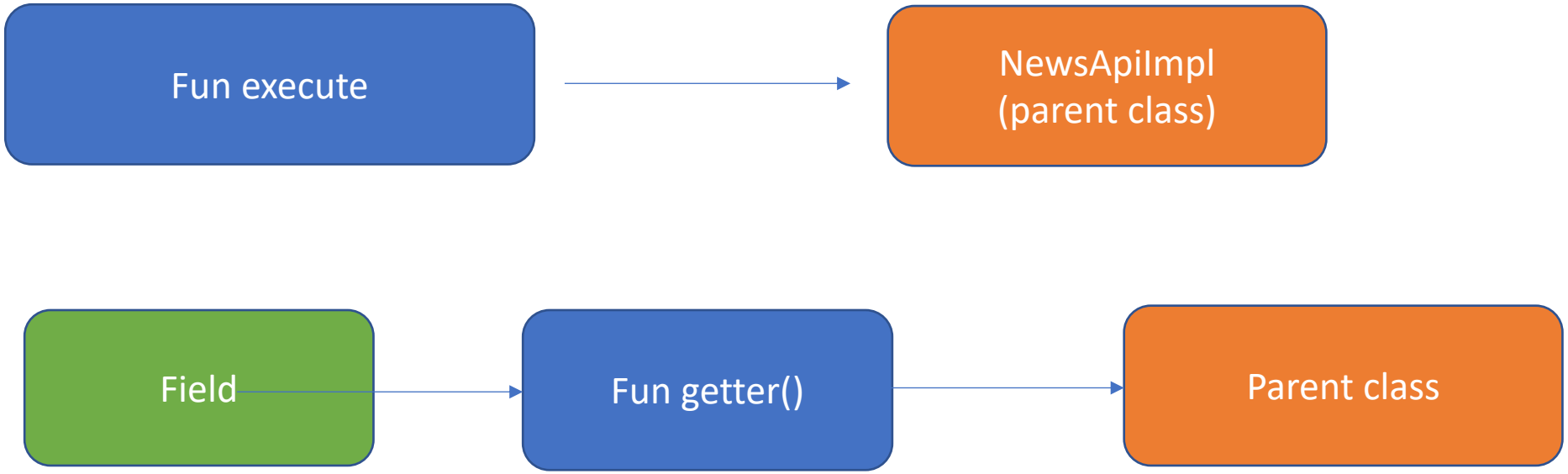
GenUseCase. Внутренний класс

Сам себе ресивер

```
val receiver = buildValueParameter(useCaseCls) {
    name = Name.special("<this>")
    type = IrSimpleTypeImpl(
        classifier = useCaseCls.symbol,
        hasQuestionMark = false, emptyList(), emptyList()
    )
    origin = IrDeclarationOrigin.INSTANCE_RECEIVER
}
useCaseCls.thisReceiver = receiver
receiver.parent = useCaseCls

val constructor = useCaseCls.addConstructor {/**магия*/}
```

Dispatch receiver

Держатель вызываемых методов или функции поля



GenUseCase. Обращение к сервису

Repo вызываем, как lazy.
Для верного обращения функция возвращает явный тип

```
private fun IrClass.addRepoLazyFunc() {
        val implClass = this
        repoFunction = implClass.addFunction {
            name = Names.REPO
            visibility = DescriptorVisibilities.PRIVATE
            returnType = apiImplType!!
        }.apply {
            val function = this
            dispatchReceiverParameter = implClass.thisReceiver?.copyTo(this)
            this.body = createLazyBody(function)
        }
    }
```

GenUseCase. Lazy field

```
private fun makeLazyField(function: IrFunction, module: IrClass): IrField {
    val lazyType = lazyFunction.returnType.getClass()!!.typeWith(function.returnType)
    val field = module.addField {
        //..provide parameters
    }
    field.initializer = with(DeclarationIrBuilder(pluginContext, field.symbol)) {
        val factoryFunction = field.factory.buildFun {
            name = Name.special("<internal_injection_initializer>")
            returnType = function.returnType
            visibility = DescriptorVisibilities.LOCAL
            origin = IrDeclarationOrigin.LOCAL_FUNCTION_FOR_LAMBDA
        }.apply {
            parent = field
            body = DeclarationIrBuilder(pluginContext, symbol).irBlockBody {/***/}
        }

        val functionExpression = IrFunctionExpressionImpl(...)
        irExprBody(
            irCall(lazyFunction.symbol, lazyType).also {
                it.putTypeArgument(0, function.returnType)
                it.putValueArgument(0, functionExpression)
            }
        )
    }
    return field
}
```

GenUseCase. Lazy field

```
val lazyType = lazyFunction.returnType.getClass()!!.typeWith(function.returnType)
    val field = module.addField {
      type = lazyType
      name = Name.identifier("__di__${function.name.asString()}")
      visibility = DescriptorVisibilities.PRIVATE
      startOffset = function.startOffset
      endOffset = function.endOffset
    }
```

GenUseCase. Lazy field

```
field.initializer = with(DeclarationIrBuilder(pluginContext, field.symbol)) {
    val factoryFunction = field.factory.buildFun {
        name = Name.special("<internal_injection_initializer>")
        returnType = function.returnType
        visibility = DescriptorVisibilities.LOCAL
        origin = IrDeclarationOrigin.LOCAL_FUNCTION_FOR_LAMBDA
    }.apply {
        parent = field
        body = DeclarationIrBuilder(pluginContext, symbol).irBlockBody {
            //..return new body
        }
    }

    val functionExpression = IrFunctionExpressionImpl(
        startOffset,
        endOffset,
        pluginContext.irBuiltIns.functionN(0).typeWith(function.returnType),
        factoryFunction,
        IrStatementOrigin.LAMBDA
    )
```

GenUseCase. Обращение к сервису

Переопределяется базовая функция SuspendUseCase

```
private fun IrClass.addExecutionFunction() {
  val implClass = this
  requestFunction?.let { function ->
     implClass.addFunction {
          name = Names.EXECUTE
        /**...*/
      }.apply {
          overriddenSymbols = listOf(pluginContext.executeFunction().symbol!!)
          addValueParameter {
              name = Names.PARAM
              type = pluginContext.referenceClass(Names.ANY)
              ?.createType(true, emptyList())
          }
          dispatchReceiverParameter = implClass.thisReceiver?.copyTo(this)
          this.addExecuteBody()
      }
  }}
```

Execute body. Dispatch receiver

Ресивер = класс-хозяин вызываемого метода или свойства

```
newFunction.body =
DeclarationIrBuilder(pluginContext, newFunction.symbol).irBlockBody {
    +irReturn(irCall(requestFunction!!.symbol, requestFunction!!.returnType).apply {
        paramIn?.let { paramIn ->
            this.putValueArgument(0, irGet(paramIn))
        }
        dispatchReceiver =
            irCall(repoFunction!!.symbol, repoFunction!!.returnType).apply {
                paramIn?.let {
                    this.putValueArgument(0, irGet(paramIn!!))
                }
                dispatchReceiver =
                    irGet(newFunction.dispatchReceiverParameter!!)
            }
    })
}
```

Подключаем и проверяем

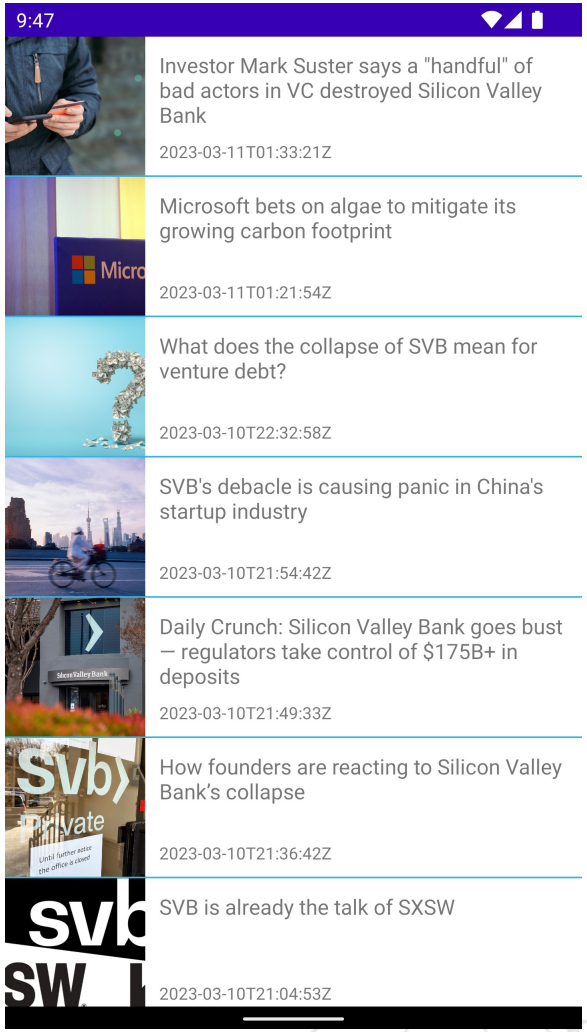
Добавляем в архитектуру
напрямую или через DI

https://github.com/anioutkazharkova/ksp-kmm-cases
KSP + KCP

```
class NewsLisViewModel () : ViewModel() {
    var newsList: MutableStateFlow<NewsList?> = MutableStateFlow(null)

    fun loadNews() {
        scope.launch {
            NewsLoadCase.usecase().request().onSuccess { data ->
                (data as? NewsList)?.let {
                    newsList.tryEmit(it)
                }
            }
        }
    }
}
```

Готовый сэмпл

https://github.com/anioutkazharkova/ksp-kmm-cases

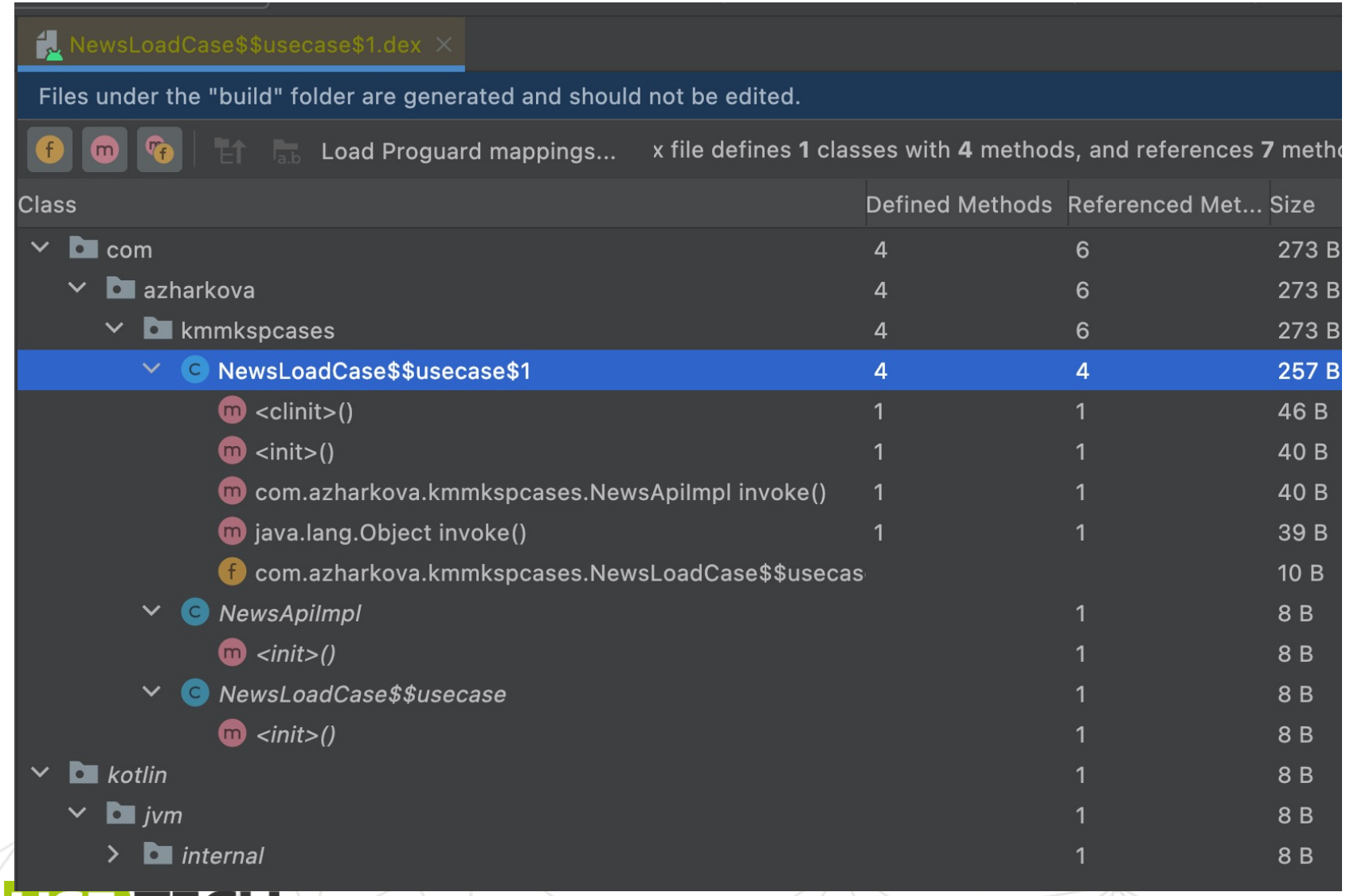


Проверяем результат в dex

Смотрим dex

Dex будет доступен после компиляции

@GenUseCase. Dex



@GenUseCase. Байткод

В байткоде можно посмотреть все сгенерированные методы, свойства и вызовы

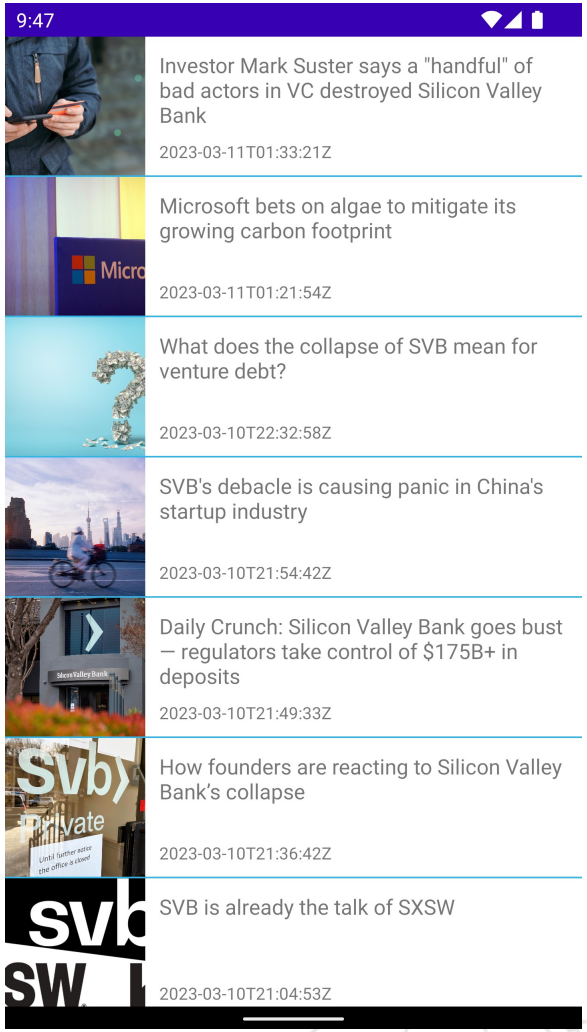
DEX Byte Code for NewsLoadCase$$usecase$1

```
.class final Lcom/azharkova/kmmkspcases/NewsLoadCase$$usecase$1;
.super Lkotlin/jvm/internal/Lambda;
.source "NewsLoadCase.kt"

# interfaces
.implements Lkotlin/jvm/functions/Function0;


# annotations
.annotation system Ldalvik/annotation/EnclosingMethod;
    value = Lcom/azharkova/kmmkspcases/NewsLoadCase$$usecase;-><init>()V
.end annotation

.annotation system Ldalvik/annotation/InnerClass;
    accessFlags = 0x18
    name = null
.end annotation

.annotation system Ldalvik/annotation/Signature;
    value = {
        "Lkotlin/jvm/internal/Lambda;",
        "Lkotlin/jvm/functions/Function0<",
        "Lcom/azharkova/kmmkspcases/NewsApiImpl;",
        ">;"
    }
```

Готовый сэмпл

Повторяем ссылку

https://github.com/anioutkazharkova/ksp-kmm-cases



Q & A или идем дальше?

Пример 2. UI.
Поиграем в Jake Wharton. Bind View

BindView

Автоподключение View по id

```
class CustomText(context: Context, set: AttributeSet? = null)
        : ConstraintLayout(context,set) {

    @BindView(R.id.text)
    var text: TextView? = null

}

@Target(AnnotationTarget.PROPERTY)
@Retention(AnnotationRetention.SOURCE)
annotation class BindView(val id: Int)
```

BindView

Автоподключение View по id

```
class CustomText(context: Context, set: AttributeSet? = null)
        : ConstraintLayout(context,set) {

    @BindView(R.id.text)
    var text: TextView? = this.findViewById(R.id.text)

}
```

Трансформируем свойство. IrElementTransformer

Берем только свойства с @BindView

```
class AndroidTransformer(val pluginContext: IrPluginContext,
    private val messageCollector: MessageCollector) :
    IrElementTransformerVoidWithContext() {

        override fun visitPropertyNew(declaration: IrProperty): IrStatement {
            if (declaration.isBindable != null) {
                /**
                * Магия по трансформации
                */
            }

            return super.visitPropertyNew(declaration)
        }
    }
```

Особенности работы с UI Android

- Не все опции доступны напрямую (internal/private)
- Все придумано до нас

Используем готовые исходники расширения для андроид

Что берем за наводку

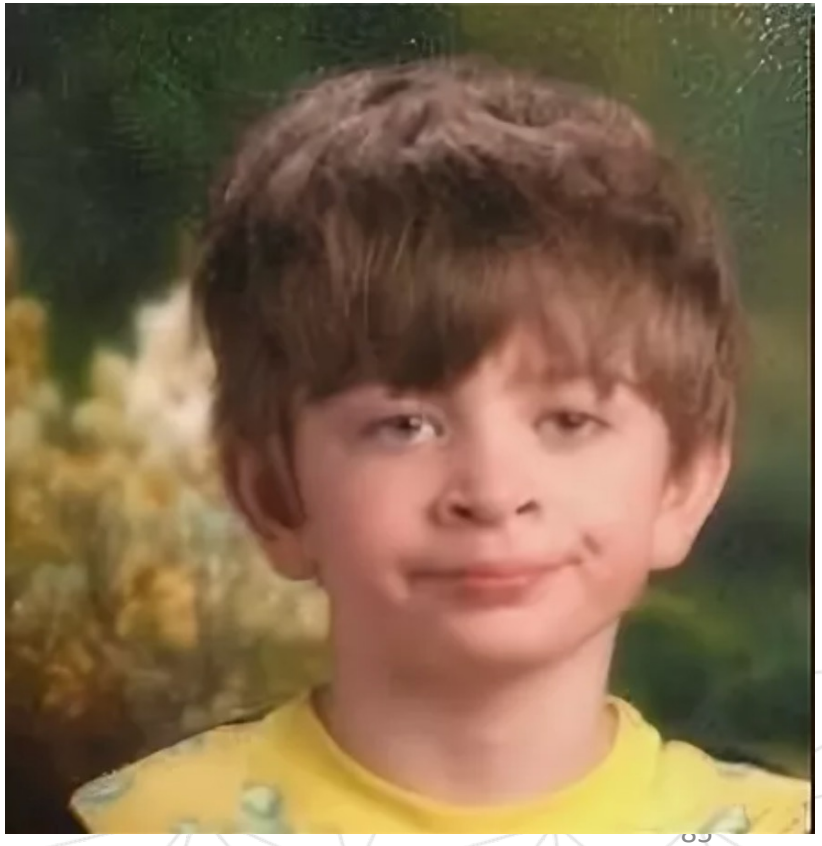
- ButterKnife ?

- AndroidIrExtensions

Что берем за наводку

ButterKnife не подойдет

- AnnotationProcessor
- Compiler скрыт



Что берем за наводку

- ~~ButterKnife ? – AnnotationProcessor, Compiler скрыт~~ 👎

- AndroidIrExtensions – Pure compiler solution 👍

AndroidIrExtensions.FindViewById

```
private fun IrBuilderWithScope.irFindViewById(
    receiver: IrExpression, id: IrExpression, container: AndroidContainerType
): IrExpression {
    val getView = // this[.getView()?|.getContainerView()?].findViewById(R$id.<name>)
    val findViewByIdParent = if (getView == null) container.fqName else FqName(AndroidConst.VIEW_FQNAME)
    val findViewById = createMethod(findViewByIdParent.child("findViewById"), nullableViewType) {
        addValueParameter("id", pluginContext.irBuiltIns.intType)
    }
    val findViewCall = irCall(findViewById.symbol).apply { putValueArgument(0, id) }
    return if (getView == null) {
        findViewCall.apply { dispatchReceiver = receiver }
    } else {
        irSafeLet(findViewCall.type, irCall(getView.symbol).apply { dispatchReceiver = receiver }) { parent ->
            findViewCall.apply { dispatchReceiver = irGet(parent) }
        }
    }
}
```

AndroidIrExtensions. Вспомогательные методы

Cache методы для класса

Оптимизируем затраты и скорость

```
class CustomView {
 private  _$_findViewCache: HashMap

 fun $findCachedViewById(int var1):View {
    if (this._$_findViewCache == null) {
       this._$_findViewCache = HashMap()
    }
    var2 = this._$_findViewCache.get(var1)
    if (var2 == null) {
       var2 = this.findViewById(var1)
       this._$_findViewCache.put(var1, var2)
    }
    return var2
 }
 fun _$_clearFindViewByIdCache() {/**...*/}
}
```

AndroidIrExtensions. Вспомогательные методы

Cache методы для класса.
Заимствуем из исходников AndroidIrExtensions

```
override fun visitClassNew(declaration: IrClass): IrStatement {
        val cacheField = declaration.getCacheField()
        declaration.declarations += cacheField // _$_findViewCache
        declaration.declarations += declaration.getClearCacheFun() // _$_clearFindViewByIdCache
        declaration.declarations += declaration.getCachedFindViewByIdFun() //_$_findCachedViewById
        declaration.createCache() //cache create
        return super.visitClassNew(declaration)
    }
```

Добавляем вызов

Вызываем cache-вариант

```
override fun visitPropertyNew(declaration: IrProperty): IrStatement {
        declaration.isBindable?.let { annotation ->
            var valueData = retrieveParamId(annotation)
            val function =  declaration.parentClassOrNull?.findViewByIdCached(pluginContext)
            declaration.getter?.apply {
                origin = IrDeclarationOrigin.DEFINED
                body = IrBlockBuilder(/**...*/).irBlockBody {
                    val self = addDispatchReceiver { type = declaration.parentAsClass.defaultType }
                    +irReturn(irCall(function!!.symbol, declaration.getter!!.returnType).apply {
                        this.putValueArgument(0,
                            irInt(valueData, pluginContext.irBuiltIns.intType))
                        dispatchReceiver = irGet(self)
                    })
                }
            }
        }
        return super.visitPropertyNew(declaration)
    }
```

Проверяем

Выводим dump

```
PROPERTY name:text visibility:public modality:FINAL [var]
  annotations:
    BindView(id = '2131231182')

FUN name:<get-text> visibility:public modality:FINAL <> ($this:CustomText) returnType:android.widget.TextView?
    correspondingProperty: PROPERTY name:text visibility:public modality:FINAL [var]
    $this: VALUE_PARAMETER name:$this type:CustomText
    BLOCK_BODY
      RETURN type=kotlin.Nothing from='public final fun <get-text> (): android.widget.TextView? declared in CustomText'
        CALL 'public open fun findViewByIdCached (id: kotlin.Int): android.view.View? declared in CustomText'
        type=android.widget.TextView? origin=null
          $this: GET_VAR '$this: CustomText declared in CustomText.<get-text>' type=CustomText origin=null
          id: CONST Int type=kotlin.Int value=2131231182
```

Проверяем

Запускаем

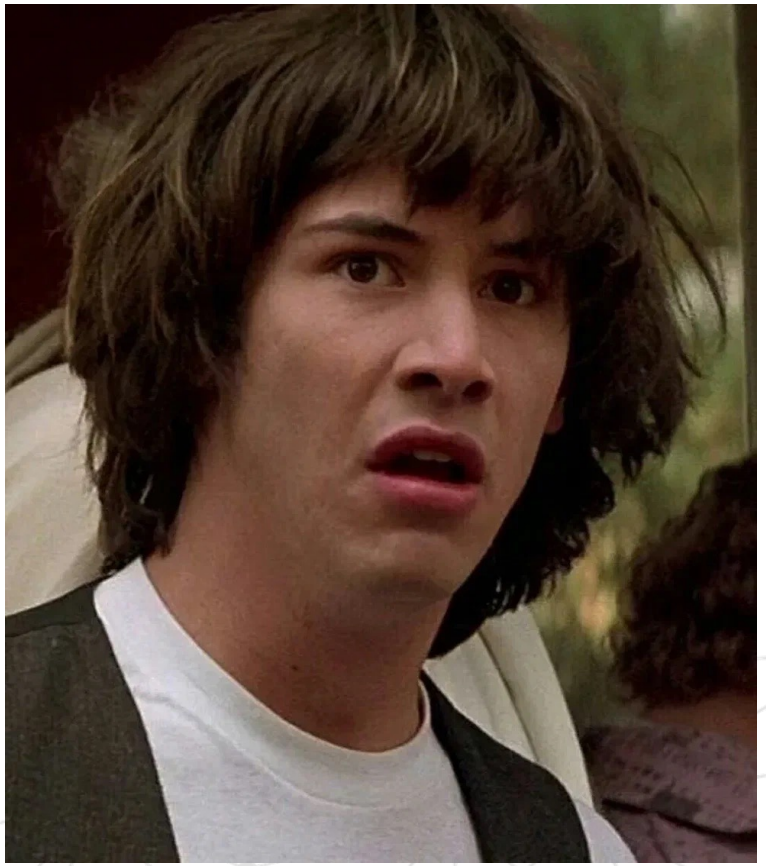

```
class MainActivity : AppCompatActivity() {

    @BindView(R.id.text_test)
    var text: TextView? = null

    @BindView(R.id.custom_test)
    var customText: CustomText? = null
```

Q & A или идем дальше?

*Замахнемся на генерацию с Compose

@Composable. Достаточно ли аннотации?

```
//From
@ToComposable
class CustomText(context: Context):ConstraintLayout(context)

//To
@Composable
fun viewComposable() {
    AndroidView(factory = {CustomText(it)})
}
```



USETECH

@Composable. Достаточно ли аннотации?

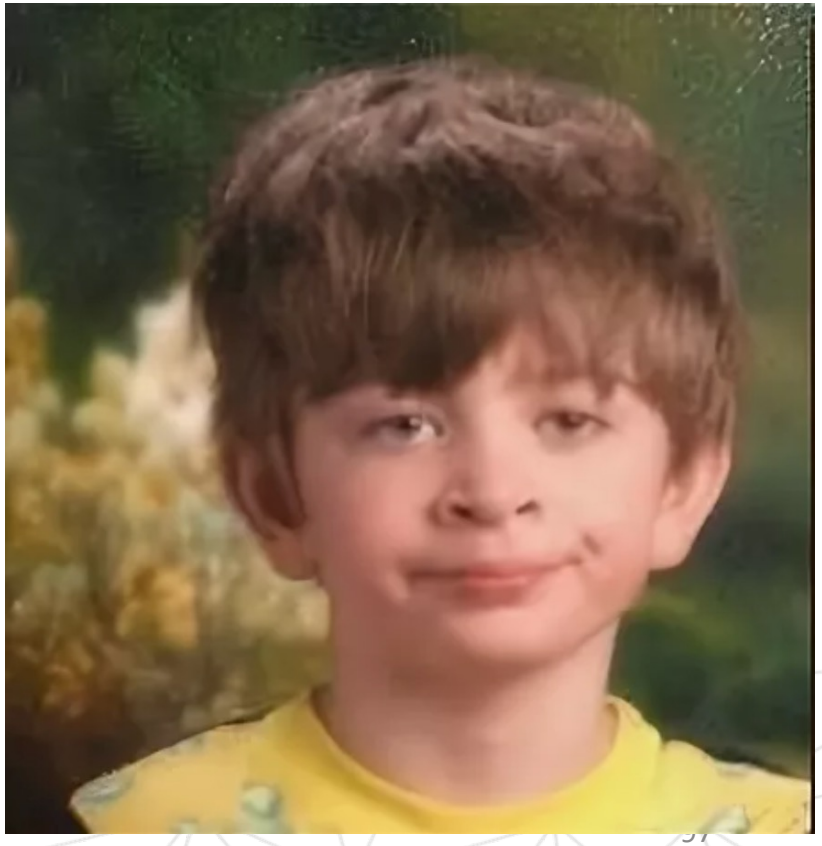
Сдампим в рамках теста

```
FUN name:viewComposable visibility:public modality:FINAL <> ($this:CustomText,
$composer:androidx.compose.runtime.Composer?,
$changed:kotlin.Int) returnType:kotlin.Unit
 annotations:
   Composable
   ComposableTarget(applier = 'androidx.compose.ui.UiComposable')
 $this: VALUE_PARAMETER name:<this> type:com.azharkova.androidkcp.CustomText
 VALUE_PARAMETER name:$composer index:0 type:androidx.compose.runtime.Composer? [assignable]
 VALUE_PARAMETER name:$changed index:1 type:kotlin.Int
 BLOCK_BODY
```

@Composable. Достаточно ли аннотации?

Недостаточно

Скрыта информация о BLOCK_BODY



@Composable. Под капотом

Свой компилятор (Compose Compiler) – свои правила

Много скрытых нюансов

```
// До компиляции
@Composable
fun CustomText(str: String) {
    Text(str)
}
//После компиляции
@Composable
   fun CustomText(str: String, $composer: Composer<*>, $changed: Int) {
      $composer.startRestartGroup(...)
      // ...
      Text(str)
      $composer.endRestartGroup()?.updateScope { next ->
         CustomText(str, next, $changed or 0b1)
} }
```

@Composable. Вывод

- Либо тянуть весь Compiler Compose в свой код
- Либо стучаться в Compose из другого решения (KSP)

@Composable. На почитать

Где смотреть код:

- ClassStabilityTransformer
- ComposerParamTransformer: добавляем к Composable $composer
- ComposableFunctionBodyTransformer: добавляем к узлам Composable startXXXGroup/endXXXGroup

@Composable. На посмотреть

Positional memoization. Как работает одна из главных концепций Jetpack Compose

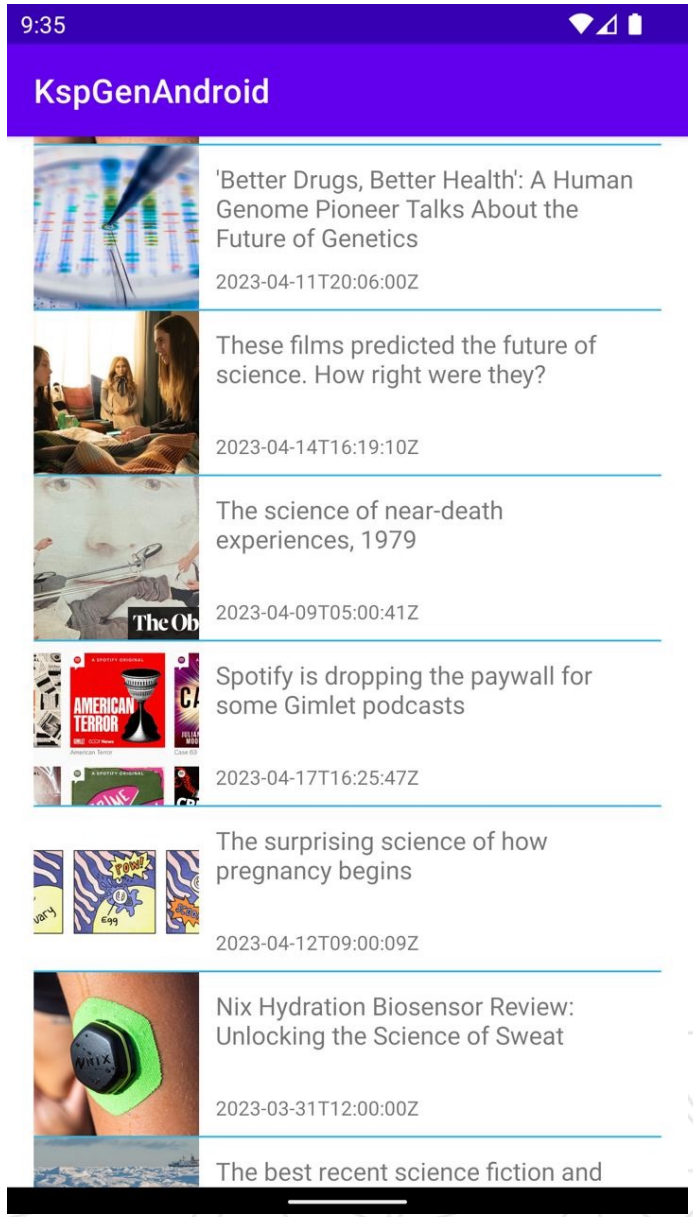
https://www.droidcon.com/2021/11/10/a-hitchhikers-guide-to-compose-compiler-composers-compiler-plugins-and-snapshots/

https://www.droidcon.com/2022/08/02/explained-compose-compiler-and-runtime/

View to Composable. KSP

Смотрите тут
https://github.com/anioutkazharkova/kgen-android

И на Mobius 19 мая



View to Composable. KSP

И на Mobius 19 мая

ДОКЛАД UI/UX в мобильной разработке 19.05 / 18:45 – 19:30 (UTC+7)

Упрощаем и укрощаем UI для Android с помощью аннотаций

Спикеры

Анна Жаркова
Usetech

Приглашенные эксперты

Сергей Боиштян
Авито

USETECH

Выводы

- Плагины легковесны и гибки в работе
- Могут дополнять другие инструменты (KSP)
- Уместны в использовании без пересечения с скрытым API
- Для каждой задачи используйте подходящий инструмент

Выводы

- Недостаточно документации для работы
- Приходится учиться на чужом коде
- API меняется и зависит от версии Kotlin

Sources

- https://github.com/google/ksp#kotlin-symbol-processing-api
- https://github.com/anioutkazharkova/ksp-kmm-cases
- https://github.com/bnorm/kotlin-ir-plugin-template
- https://blog.bnorm.dev/writing-your-second-compiler-plugin-part-5
- https://github.com/anioutkazharkova/android-kcp
- https://syntaxbug.com/ddbe2adf60/

Спасибо за внимание!

@anioutkajarkova

azharkova
prettygeeknotes