USE TECH

Упрощаем и укрощаем UI для Android с помощью аннотаций

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)

USETECH

Обсудим:

- Аннотации, процессинг
- KAPT vs KSP, KSP как современный подход
- Анатомия процессора KSP
- Адаптируем адаптер. Биндинги
- Из View в Compose. Насомposим экран

Аннотации

Тэги + метаданные с доп.информацией о помечаемых элементах

```
@Target(AnnotationTarget.CLASS)
@Retention(AnnotationRetention.SOURCE)
annotation class Adapter(val holders: Array<KClass<*>>,
val bindType: KClass<*>)

@Target(AnnotationTarget.CLASS)
@Retention(AnnotationRetention.SOURCE)
annotation class BindVH(val bindType: KClass<*>)

@Retention(AnnotationRetention.SOURCE)
@Target(AnnotationTarget.FUNCTION)
annotation class BindSetup

@Target(AnnotationTarget.VALUE_PARAMETER)
@Retention(AnnotationRetention.SOURCE)
annotation class BindLayout

@Retention(AnnotationRetention.SOURCE)
@Target(AnnotationTarget.FUNCTION)
annotation class BindView
```

USE TECH

Для чего процессинг аннотаций

- Анализ кода (проверка на ошибки и т.п)
- Генерация кода для общих целей
- Упрощение абстракций
- Сокращение кода для написания разработчиком

Для чего процессинг аннотаций

- Анализ кода (проверка на ошибки и т.п)
- Генерация кода для общих целей
- Упрощение абстракций

Сокращение кода для написания разработчиком

USETECH

Когда не подойдет процессинг аннотаций

- Изменение существующего кода

Kotlin Compiler Plugin

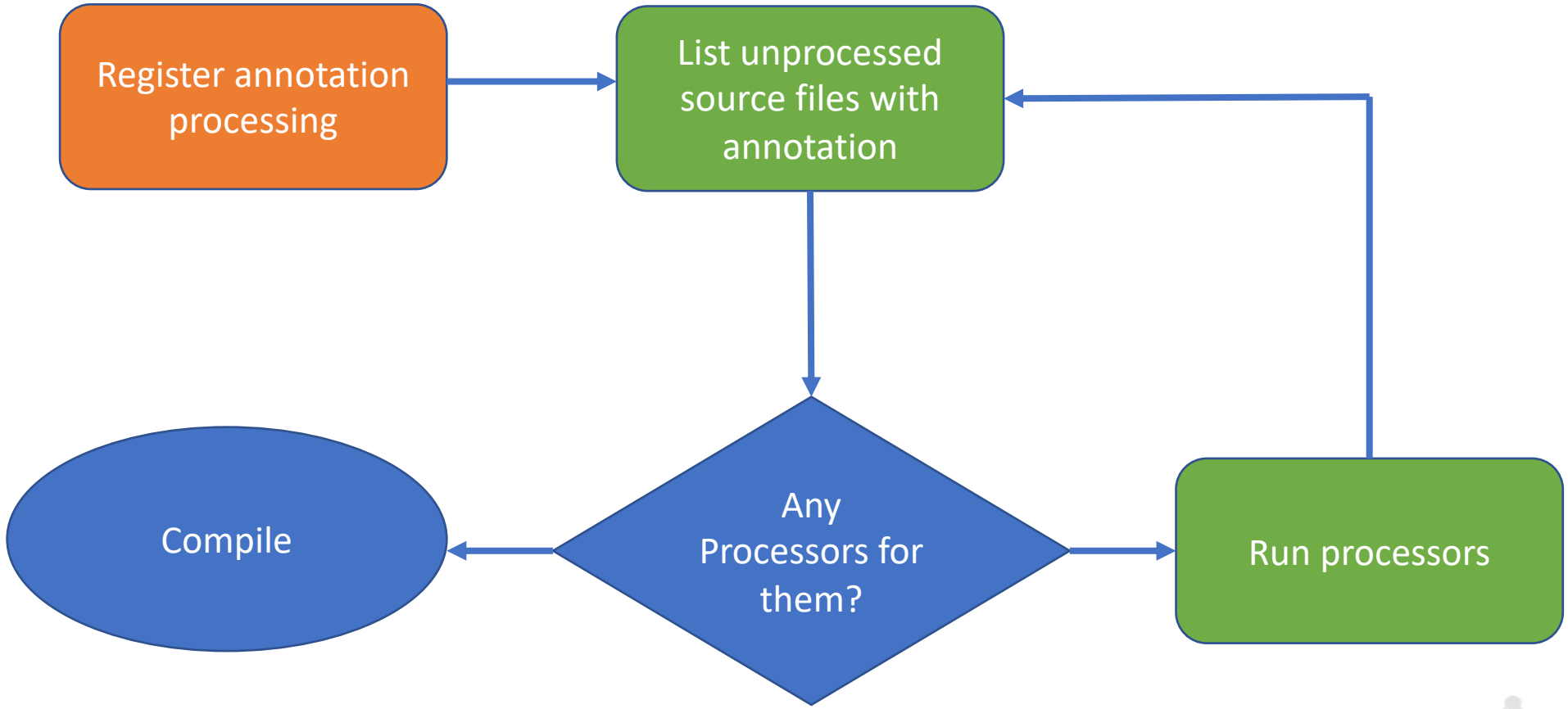
Когда не подойдет процессинг аннотаций

- Изменение существующего кода

Kotlin Compiler Plugin

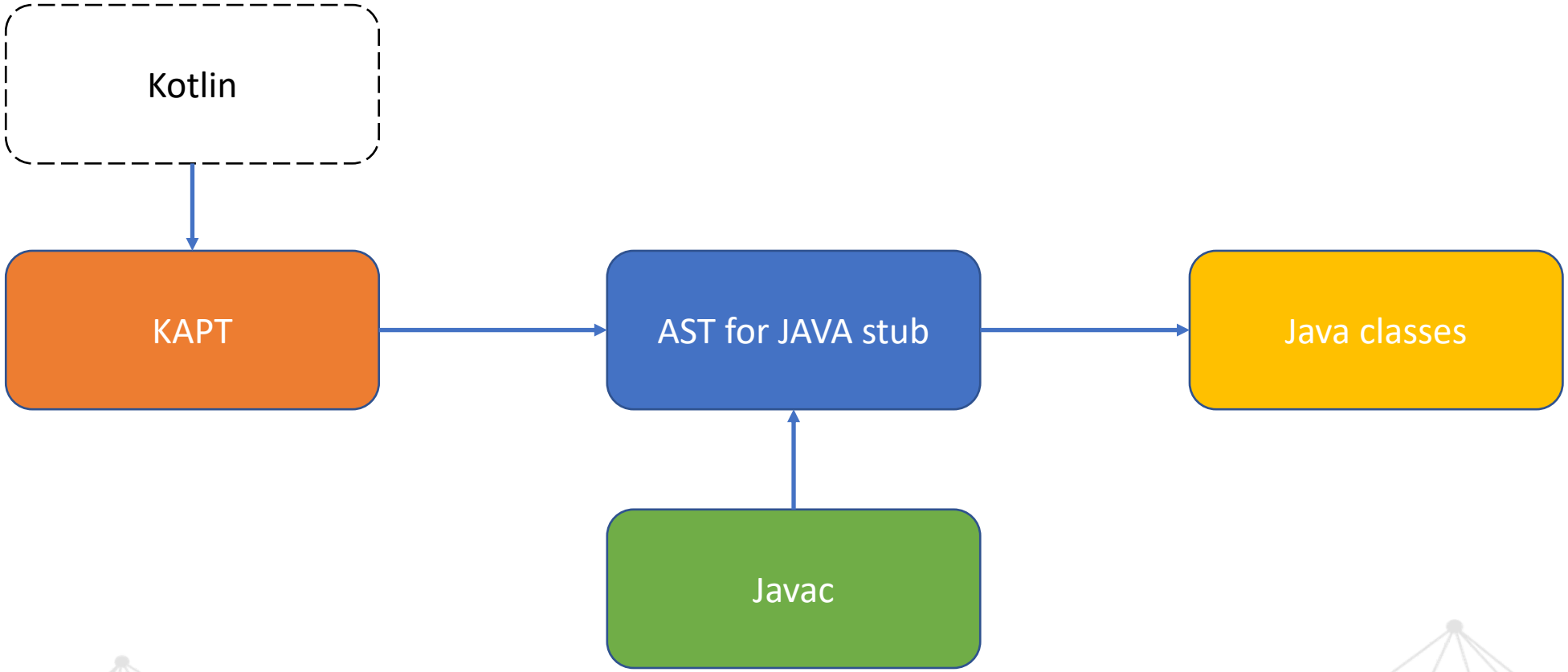
KCP +KSP

Процессинг аннотаций. Основы

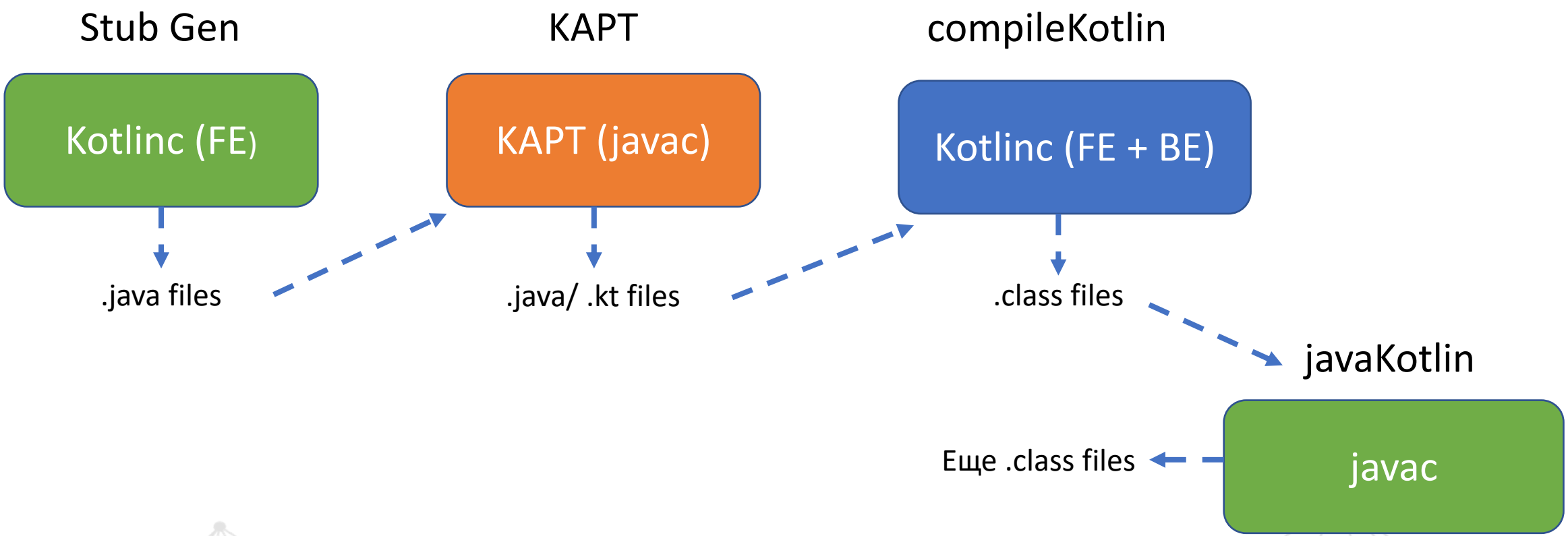


KAPT (Kotlin annotation processing tool) 2/3

JSR 269



KAPT. Компиляция



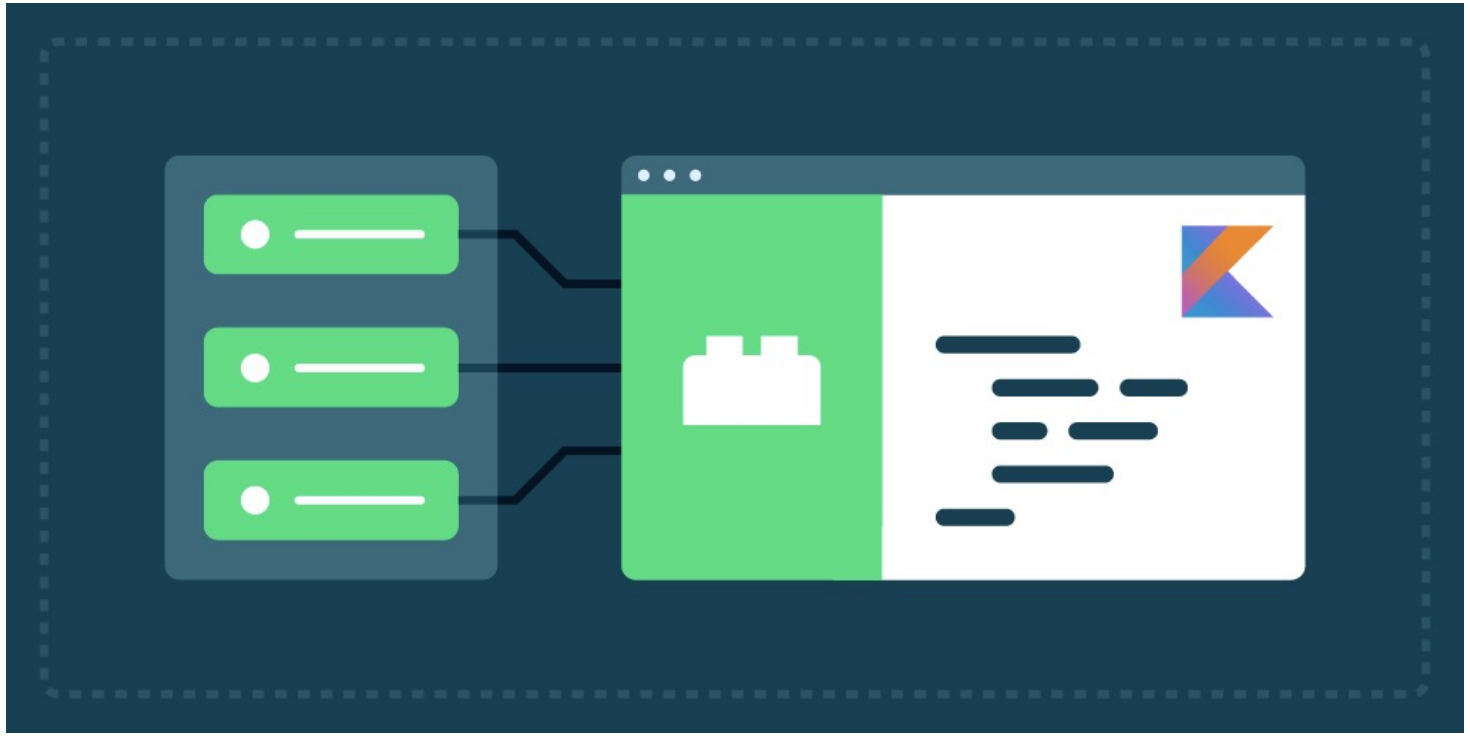
Известные решения на КАРТ (примеры)

- Dagger/ Hilt
- Retrofit
- ButterKnife
- Room
- Юнит-тесты разные …

Kotlin Symbol Processing

Stable since 1.5.31
1.8.10 – 1.0.9

https://github.com/google/ksp

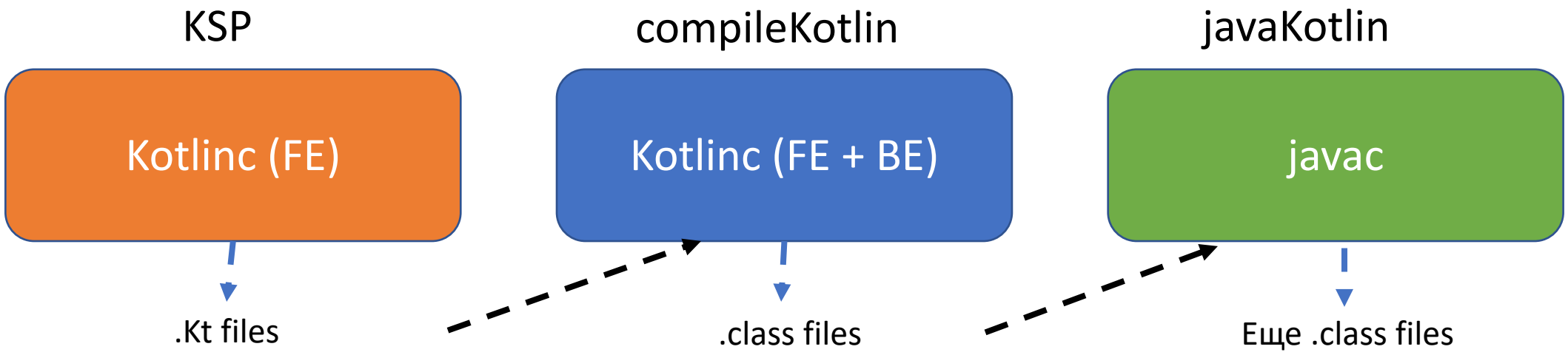


Kotlin Symbol Processing. Назначение

- Разработка легковесных плагинов компиляции
- Разработка процессоров аннотации

```
@Target(AnnotationTarget.CLASS)
@Retention(AnnotationRetention.SOURCE)
annotation class Adapter(val holders: Array<KClass<*>>,
val bindType: KClass<*>)

@Target(AnnotationTarget.CLASS)
@Retention(AnnotationRetention.SOURCE)
annotation class BindVH(val bindType: KClass<*>)

@Retention(AnnotationRetention.SOURCE)
@Target(AnnotationTarget.FUNCTION)
annotation class BindSetup

@Target(AnnotationTarget.VALUE_PARAMETER)
@Retention(AnnotationRetention.SOURCE)
annotation class BindLayout

@Retention(AnnotationRetention.SOURCE)
@Target(AnnotationTarget.FUNCTION)
annotation class BindView
```

KSP. Компиляция



KAPT vs KSP

- KAPT генерирует дополнительно Java код
- Java заглушки.
- Больше времени
- Больше кода

USE TECH

Преимущества KSP

Чище

Прямой доступ к AST

Быстрее

Нет заглушек Java

Совместимость

Java code

Выигрыш: до 25% времени компиляции

https://kotlinlang.org/docs/ksp-why-ksp.html#comparison-to-kapt

USETECH

Google I/O 2023. Рекомендованная технология

#Google I/O

KSP

kapt

Kotlin Annotation Processing Tool

Generates Java stubs from Kotlin code, allowing annotations processors written for Java to work with Kotlin

Kotlin code → Java stubs → Java Annotation Processors

Java is a registered trademark of Oracle and/or its affiliates.

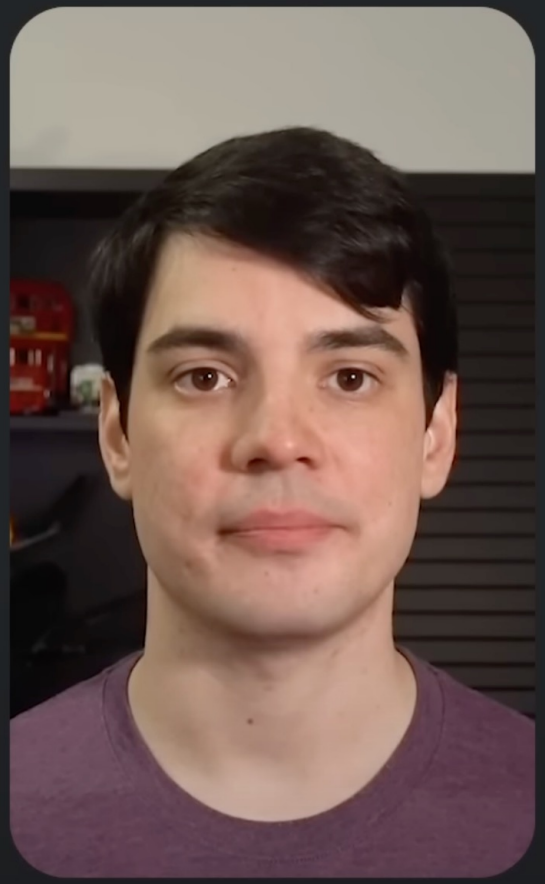
KSP

goo.gle/ksp

Kotlin Symbol Processing

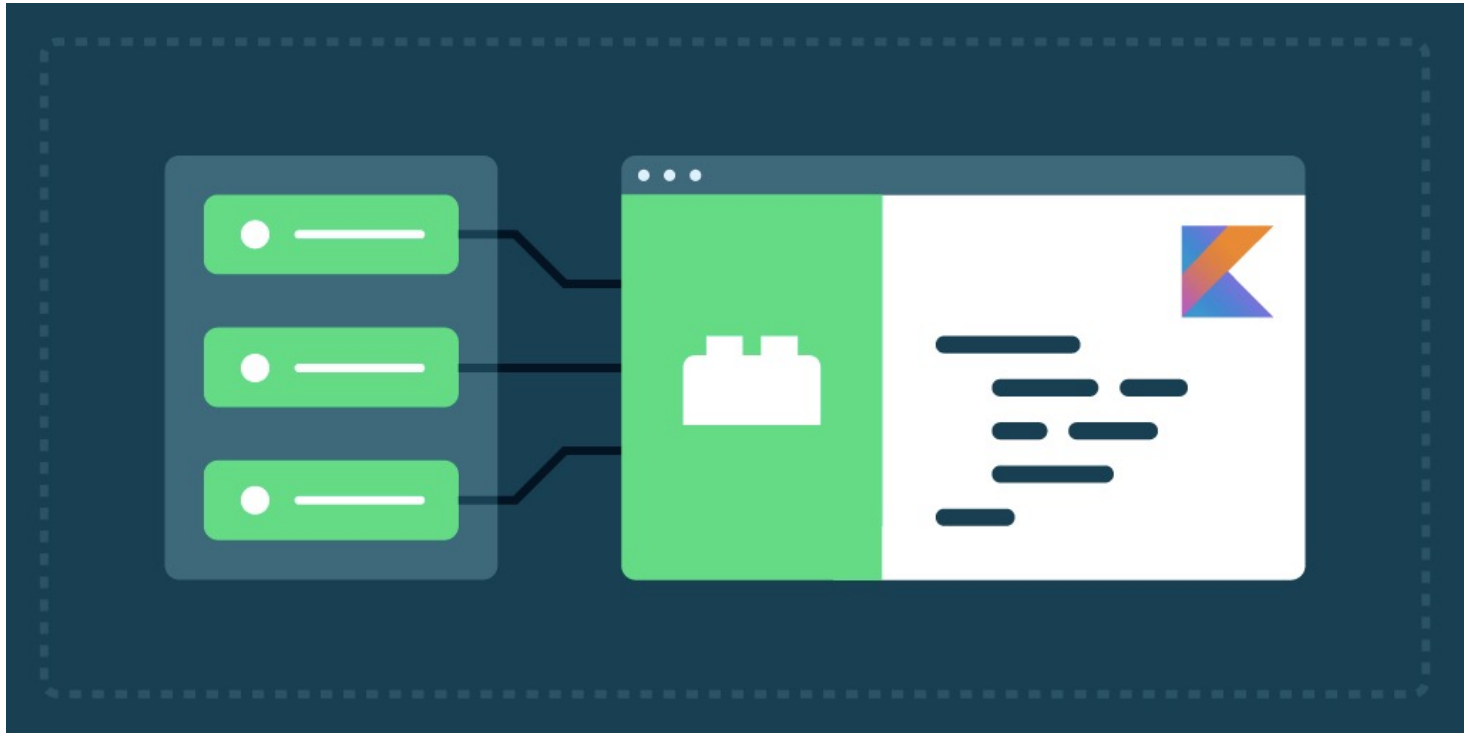
A Kotlin-first alternative to kapt, directly analyzing Kotlin code

Kotlin code → KSP Processors



Migrate from kapt to KSP

https://developer.android.com/studio/build/migrate-to-ksp



Migrate from kapt to KSP

DataBinding все еще на KAPT

> ★ **Note:** While not a traditionally-included library dependency, Data Binding also uses an annotation processor to provide its functionality, and KSP support for Data Binding is not planned. You can mitigate the impact of kapt on your build by isolating the usages of Data Binding to separate modules.

USETECH

Migrate from KAPT to KSP

- Room
- Glide
- Mochi
- Koin

USE TECH

Migrate from KAPT to KSP

- Room
- Glide
- Mochi
- Koin
  +Dagger

Androidx Annotations

Annotation

API Reference
androidx.annotation

Expose metadata that helps tools and other developers understand your app's code.

This table lists all the artifacts in the `androidx.annotation` group.

Artifact	Stable Release	Release Candidate	Beta Release	Alpha Release
annotation	1.6.0	-	-	1.7.0-alpha02
annotation-experimental	1.4.0-dev01	-	-	-

Androidx Annotations

- @StringRes, @DimenRes, @DrawableRes…
- @UiThread, @WorkerThread, @MainThread…
- @IntRange, @FloatRange, @Size…

Попробуем сами

Что оптимизировать можно

Однотипные задачи:

- Различные адаптеры для списков
- Генерация View с параметрами
- Навигация
- Бизнес-логика
- Dependency Injection…

Что оптимизировать будем (сегодня)

Однотипные задачи:

- Различные адаптеры для списков
- Подключение и настройка View
- Генерация View с параметрами (View <-> Composable)

Что оптимизировать не будем (сегодня)

Однотипные задачи:

- Бизнес-логика
- Dependency Injection

DI + KSP

БЫЛО, ПОЭТОМУ НЕ СЕГОДНЯ

Что оптимизировать не будем (сегодня)

Однотипные задачи:

- Бизнес-логика
- Dependency Injection

USECASE + KSP

БЫЛО, ПОЭТОМУ НЕ СЕГОДНЯ

Подготовим базовый процессор KSP

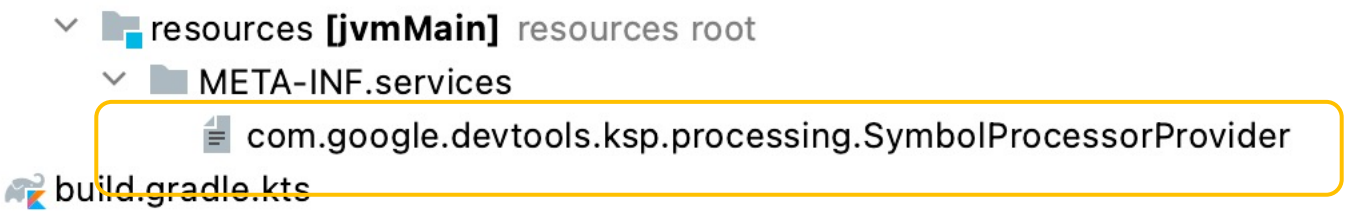
Kotlin symbol processing. Анатомия

Annotation Processor ->
Symbol processor

```
/**
 * Интерфейс провайдера
 */
fun interface SymbolProcessorProvider {

    fun create(environment: SymbolProcessorEnvironment): SymbolProcessor
}


//Интерфейс процессора
interface SymbolProcessor {

    /**
    * Смотрим сюда
    */
    fun process(resolver: Resolver): List<KSAnnotated>

    fun finish() {}


    fun onError() {}
}
```

Kotlin symbol processing. Анатомия

Подключаем провайдер в ресурсы




```
// Декларируем все ProcessorProvider
com.azharkova.processor.network.NetworkProcessorProvider
com.azharkova.processor.core.FactoryProcessorProvider
com.azharkova.processor.core.UsecaseProcessorProvider
```

Kotlin symbol processing. Как работает

Resolver = доступ ко всем данным

Обработка напрямую

Или Visitor

```
class AdapterProcessor constructor(val env: SymbolProcessorEnvironment)
: SymbolProcessor {

    val visitor = CustomVisitor()

    override fun process(resolver: Resolver): List<KSAnnotated> {

        val adapters = getAdapters(resolver)

        /**....*/
        generateImplClass(data, codeGenerator)
        return emptyList()
    }

    private fun getAdapters(resolver: Resolver): Sequence<KSClassDeclaration> {
        return resolver.getSymbolsWithAnnotation((Adapter::class.java).name)
            .filterIsInstance<KSClassDeclaration>().distinct()
    }
}
```

USETECH

Kotlin symbol processing. Как работает

Через KSClassDeclaration/KSFunctionDeclaration/KSFile получаем доступ к типам и их параметрам.

```
class CustomVisitor : KSVisitorVoid() {
    override fun visitClassDeclaration(classDeclaration: KSClassDeclaration, data: Unit) {
        classDeclaration.getDeclaredFunctions().map { it.accept(this, Unit) }
    }

    override fun visitFunctionDeclaration(function: KSFunctionDeclaration, data: Unit) {
        functions.add(function)
    }

    override fun visitFile(file: KSFile, data: Unit) {
        file.declarations.map { it.accept(this, Unit) }
    }
}
```

USE TECH

Kotlin poet

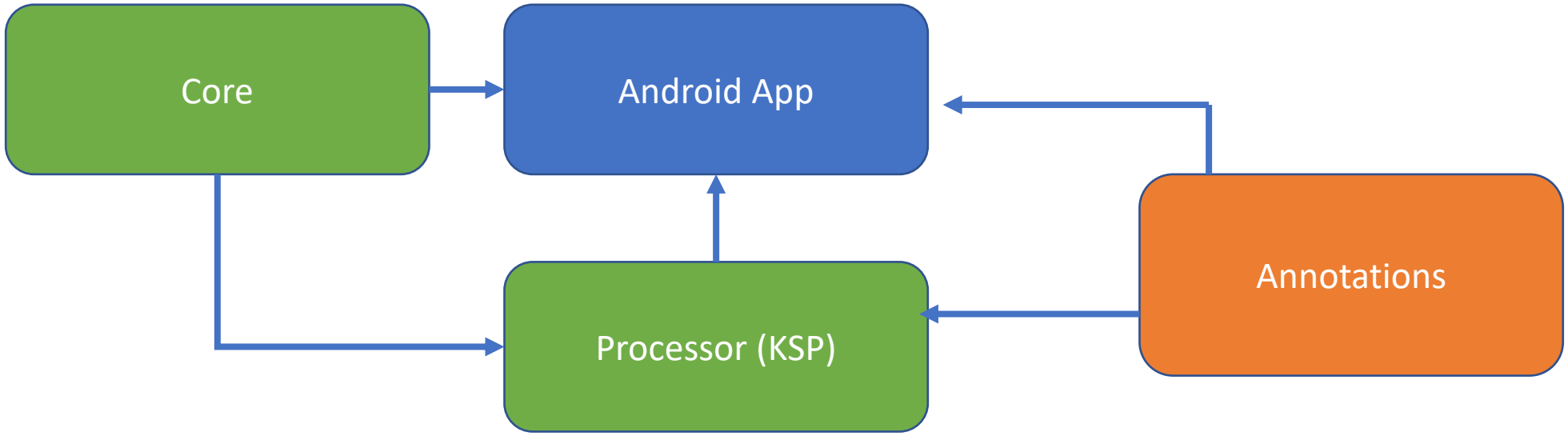
Используем, чтобы сгенерировать .kt исходники

https://square.github.io/kotlinpoet/

```
val greeterClass = ClassName("", "Greeter")
val file = FileSpec.builder("", "HelloWorld")
    .addType(TypeSpec.classBuilder("Greeter")
        .primaryConstructor(FunSpec.constructorBuilder()
            .addParameter("name", String::class)
            .build())
        .addProperty(PropertySpec.builder("name", String::class)
            .initializer("name")
            .build())
        .addFunction(FunSpec.builder("greet")
            .addStatement("println(%P)", "Hello, \$name")
            .build())
        .build())
    .addFunction(FunSpec.builder("main")
        .addParameter("args", String::class, VARARG)
        .addStatement("%T(args[0]).greet()", greeterClass)
        .build())
    .build()

file.writeTo(System.out)
```

```
class Greeter(val name: String) {
  fun greet() {
    println("""Hello, $name""")
  }
}

fun main(vararg args: String) {
  Greeter(args[0]).greet()
}
```

Подключаем к проекту



Подключаем к проекту

Используем symbol-processing-api

```
plugins {
    id("com.google.devtools.ksp")
    id("java-library")
    id("org.jetbrains.kotlin.jvm")
}



dependencies {
    implementation(kotlin("stdlib"))
    implementation(project(":annotations"))
    implementation(project(":core"))
    implementation("com.google.devtools.ksp:symbol-processing-api:1.8.0-1.0.8")

}
```

USE TECH

Подключаем к проекту

KotlinPoet

```
dependencies {
    /***/
    implementation("com.squareup:kotlinpoet-ksp:1.12.0")
    implementation("com.squareup:kotlinpoet:1.12.0")
}
```

Подключаем к проекту

```
plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
    id 'com.google.devtools.ksp'
}

//sourceSets
kotlin {
    sourceSets {
        main.kotlin.srcDirs += 'build/generated/ksp/'
    }
}

dependencies {
    implementation project(":kspgenprocessor")
    implementation project(':annotations')
    ksp project(":kspgenprocessor")
}
```

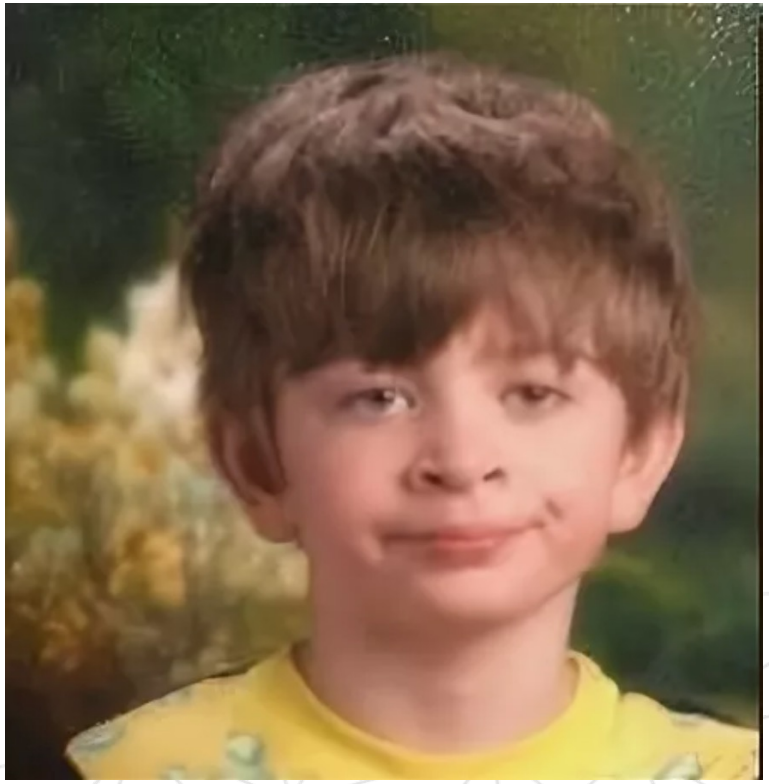
Стандартный план-капкан по работе с KSP

1. Ищем KSDeclaration с определенными аннотациями
2. Маппим модели, зависимости и параметры
3. Генерируем классы и методы
4. Profit!

USETECH

Kotlin Symbol Processing. Ограничения

- Нельзя изменить существующие данные и файлы
- Только генерация новых конструкций в новых файлах
- Типы указываем явно – указание ссылки на родительский/абстрактный тип/интерфейс дает доступ к родителю
- Kotlin/Java примитивы, KClass<*>, Array, String



USETECH

Кейс 1. Оптимизируем списки и адаптеры

Упрощаем адаптер

- Adapter + ViewHolder = много избыточного кода

```
class TestAdapter : RecyclerView.Adapter<TestViewHolder>() {
    val items: MutableList<Int> = mutableListOf()

    fun setupItems(items: List<Int>) {
        this.items.addAll(items)
    }

    override fun onCreateViewHolder(parent: ViewGroup,
    viewType: Int): TestViewHolder {

        return TestViewHolder(ItemTestLayoutBinding
        .inflate( LayoutInflater.from(
            parent.context
        ), parent, false))
    }

    override fun onBindViewHolder(holder: TestViewHolder,
            position: Int) {
        holder.tag = position
        holder.bindItem(items.get(position))
    }

    override fun getItemCount(): Int = items.count()

}
```

USE TECH

Упрощаем адаптер

Сгенерируем весь избыточный код сами

```
@BindVH(Int::class)
class TestViewHolder(@BindLayout val itemViewBinding: ItemTestLayoutBinding) :
RecyclerView.ViewHolder(itemViewBinding.root) {

    @BindSetup
    fun setupData(data: Int) {
    itemViewBinding.text.text = data.toString()
    }
}

@Adapter(holders = [TestViewHolder::class], Int::class)
abstract class TestAdapter
```

Упрощаем адаптер. Аннотации для адаптера

@Adapter:
- holders
- bindType

@BindVH

```
@Target(AnnotationTarget.CLASS)
@Retention(AnnotationRetention.SOURCE)
annotation class Adapter(val holders: Array<KClass<*>>,
val bindType: KClass<*>)

@Target(AnnotationTarget.CLASS)
@Retention(AnnotationRetention.SOURCE)
annotation class BindVH(val bindType: KClass<*>)
```

Упрощаем адаптер. Аннотации для параметра

Параметры:

- ViewHolders (своя аннотация)
- BindType

```
@Target(AnnotationTarget.CLASS)
@Retention(AnnotationRetention.SOURCE)
annotation class Adapter(val holders: Array<KClass<*>>,
                         val bindType: KClass<*>)


@Adapter(holders = [TestViewHolder::class], Int::class)
abstract class TestAdapter
```

Упрощаем адаптер. ViewHoldersData

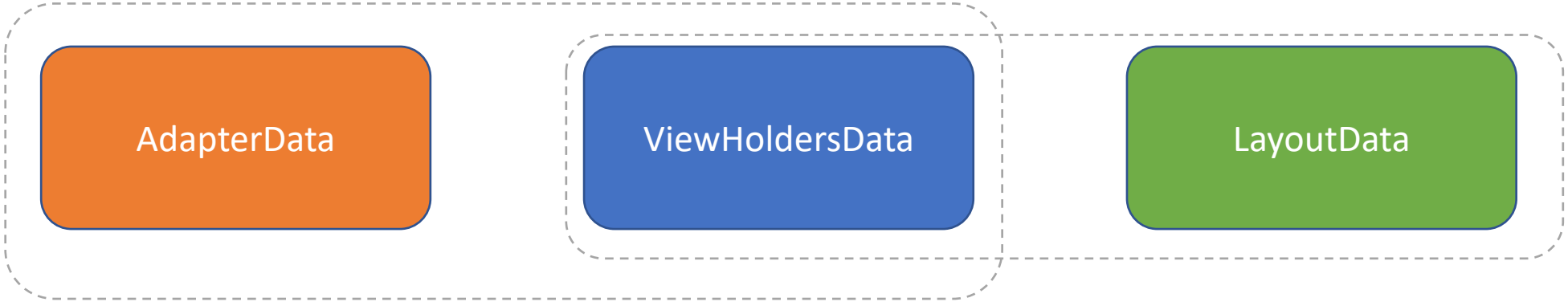
- BindType (аннотация)
- Layout (конструктор)

```
@BindVH(Int::class)
class TestViewHolder(@BindLayout val itemViewBinding: ItemTestLayoutBinding) :
RecyclerView.ViewHolder(itemViewBinding.root) {

    @BindSetup
    fun setupData(data: Int) {
    itemViewBinding.text.text = data.toString()
    }
}
```

План-капкан для адаптеров и списков

1. Ищем все классы с @Adapters
2. Получаем параметры из аннотации: список типов ViewHolders и тип данных
3. Ищем все @BindVH ViewHolders
4. Маппим для каждого инфо о layout и параметрах
5. Собираем всю инфо для адаптера
6. Генерируем

Подготовим модели



Упрощаем адаптер. Поиск адаптеров

Ищем всех носителей
@Adapter

```
class AdapterProcessor constructor(val env: SymbolProcessorEnvironment)
: SymbolProcessor {

    protected val logger: KSPLogger = env.logger
    protected val codeGenerator = env.codeGenerator

    override fun process(resolver: Resolver): List<KSAnnotated> {

        val adapters = getAdapters(resolver)
        val data = adapters.mapNotNull {
            it.toAdapterData(resolver)
        }.toList()

        generateImplClass(data, codeGenerator)
        return emptyList()
    }
/**....*/
}
```

Упрощаем адаптер. Поиск адаптеров

Ищем всех носителей
@Adapter

```
fun getAdapters(resolver: Resolver): Sequence<KSClassDeclaration> {
    return resolver
        .getSymbolsWithAnnotation((Adapter::class.java).name)
        .filterIsInstance<KSClassDeclaration>()
        .distinct()
}
```

Упрощаем адаптер. Получение параметров

Параметры:
- ViewHolders (своя аннотация)
- BindType

```
@Target(AnnotationTarget.CLASS)
@Retention(AnnotationRetention.SOURCE)
annotation class Adapter(val holders: Array<KClass<*>>,
                         val bindType: KClass<*>)


@Adapter(holders = [TestViewHolder::class], Int::class)
abstract class TestAdapter
```

Упрощаем адаптер. Получение параметров

Параметры:
- ViewHolders (своя аннотация)
- BindType

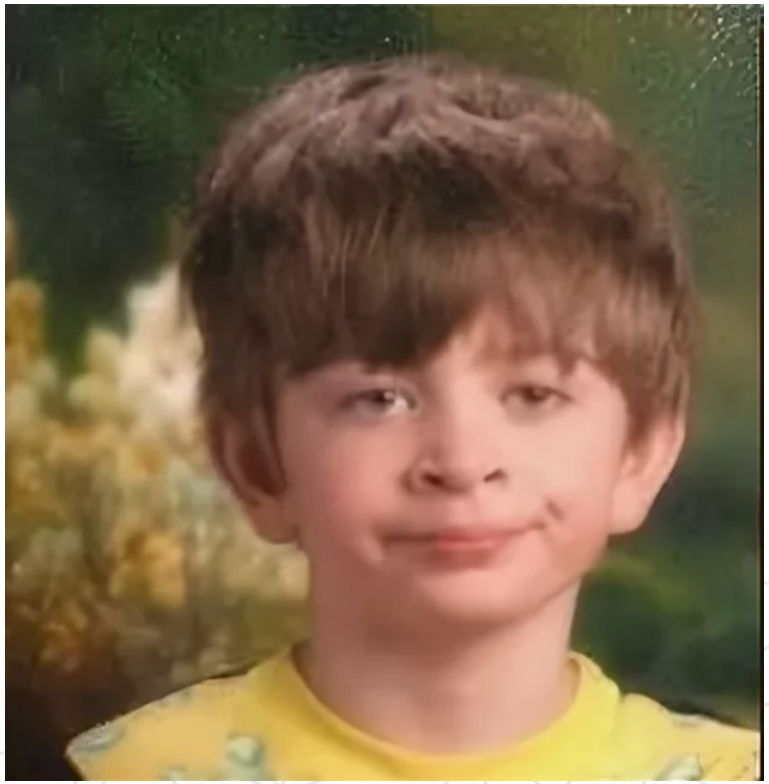
```
@Target(AnnotationTarget.CLASS)
@Retention(AnnotationRetention.SOURCE)
annotation class Adapter(val holders: Array<KClass<*>>,
                         val bindType: KClass<*>)


@Adapter(holders = [TestViewHolder::class], Int::class)
abstract class TestAdapter
```

Абстрактный класс??

Kotlin symbol processing. Ограничения

Новый код:

- Использовать функции-расширения
- Использовать DI-фабрики для сопоставления базового типа и нового



USETECH

Упрощаем адаптер. Получение параметров

Считываем параметры из аннотации

```
val bindType = annotation?.let {
    getParamValue(annotation, "bindType")
}
val viewholdersParams = getParamValueList(annotation, "holders")
val viewholders = getViewHolders(
    resolver,
    viewholdersParams.orEmpty()
        .map { it.declaration.qualifiedName?.asString().orEmpty() })
```

Упрощаем адаптер. Получение параметров

Считываем параметры из аннотации

```
val bindType = annotation?.let {
    getParamValue(annotation, "bindType")
}
val viewholdersParams = getParamValueList(annotation, "holders")
val viewholders = getViewHolders(
    resolver,
    viewholdersParams.orEmpty()
        .map { it.declaration.qualifiedName?.asString().orEmpty() })
```

USETECH

Упрощаем адаптер. Получение VH

ViewHolders

```
fun getViewHolders(resolver: Resolver, filterNames: List<String>):
Sequence<KSClassDeclaration> {
    return resolver.getSymbolsWithAnnotation((BindVH::class.java).name)
        .filterIsInstance<KSClassDeclaration>().distinct().filter {
        filterNames.contains(it.qualifiedName?.asString().orEmpty())
        }
    }
```

Упрощаем адаптер. ViewHoldersData

@BindVH

Параметры:
- BindType

Вспомогательные:
- BindLayout
- BindSetup

```
@Target(AnnotationTarget.CLASS)
@Retention(AnnotationRetention.SOURCE)
annotation class BindVH(val bindType: KClass<*>)

@Retention(AnnotationRetention.SOURCE)
@Target(AnnotationTarget.FUNCTION)
annotation class BindSetup

@Target(AnnotationTarget.VALUE_PARAMETER)
@Retention(AnnotationRetention.SOURCE)
annotation class BindLayout
```

Упрощаем адаптер. ViewHoldersData

@BindSetup надо писать самим

```
@BindVH(Int::class)
class TestViewHolder(@BindLayout val itemViewBinding: ItemTestLayoutBinding) :
RecyclerView.ViewHolder(itemViewBinding.root) {

    @BindSetup
    fun setupData(data: Int) {
    itemViewBinding.text.text = data.toString()
    }
}
```

Упрощаем адаптер. ViewHoldersData

@BindSetup надо писать самим

```
@BindVH(Int::class)
class TestViewHolder(@BindLayout val itemViewBinding: ItemTestLayoutBinding) :
RecyclerView.ViewHolder(itemViewBinding.root) {

    @BindSetup
    fun setupData(data: Int) {
    itemViewBinding.text.text = data.toString()
    }
}
```

Упрощаем адаптер. ViewHoldersData

Маппим в ViewHoldersData

```
fun KSClassDeclaration.toViewHolderData(resolver: Resolver): ViewHolderData {
        val annotation = getAnnotation(this, "BindVH", BindVH::class.java.name)
        val bindType = annotation?.let {
            getParamValueType(annotation, "bindType")
        }
       /**
        Код какой-то
       */

        val function = //Get BindSetup

        val layoutParameter = //Get BindLayout

        val layoutData = layoutParameter?.let {
            LayoutData(it.type.toString(), BINDING_PACKAGE)
        }
        return ViewHolderData(layoutData,
        name,
        packageName,
        bindType = returnType,
        setupFunc = FUNCTION_NAME)
    }
```

Упрощаем адаптер. ViewHoldersData

Маппим в ViewHoldersData

```
fun KSClassDeclaration.toViewHolderData(resolver: Resolver): ViewHolderData {
        val annotation = getAnnotation(this, "BindVH", BindVH::class.java.name)
        val bindType = annotation?.let {
            getParamValueType(annotation, "bindType")
        }
       /**
        Код какой-то
       */

        val function = //Get BindSetup

        val layoutParameter = //Get BindLayout

        val layoutData = layoutParameter?.let {
            LayoutData(it.type.toString(), BINDING_PACKAGE)
        }
        return ViewHolderData(layoutData,
        name,
        packageName,
        bindType = returnType,
        setupFunc = FUNCTION_NAME)
    }
```

Упрощаем адаптер. ViewHoldersData

Маппим в ViewHoldersData

```
fun KSClassDeclaration.toViewHolderData(resolver: Resolver): ViewHolderData {
        val annotation = getAnnotation(this, "BindVH", BindVH::class.java.name)
        val bindType = annotation?.let {
            getParamValueType(annotation, "bindType")
        }
       /**
        Код какой-то
       */

        val function = //Get BindSetup

        val layoutParameter = //Get BindLayout

        val layoutData = layoutParameter?.let {
            LayoutData(it.type.toString(), BINDING_PACKAGE)
        }
        return ViewHolderData(layoutData,
        name,
        packageName,
        bindType = returnType,
        setupFunc = FUNCTION_NAME)
    }
```

Упрощаем адаптер. ViewHoldersData

BindSetup

```
val function = this.getAllFunctions().firstOrNull{ function ->
        .joinToString (" "))
        function.annotations
        .any { it.shortName.getShortName() == BindSetup::class.java.simpleName.orEmpty()}
    }
```

Упрощаем адаптер. ViewHoldersData

LayoutData

```
val layoutParameter = this.getConstructors()?.map{
        it.parameters.firstOrNull {
            it.annotations.any{
                it.shortName.getShortName() == BindLayout::class.simpleName.orEmpty()
            }
        }
    }?.firstOrNull()
    val layoutData = layoutParameter?.let {
        LayoutData(it.type.toString(), BINDING_PACKAGE)
    }
```

Упрощаем адаптер. Получение параметров

Маппим в AdaptersData

```
fun KSClassDeclaration.toAdapterData(resolver: Resolver):AdapterData? {
    val adapter = this
    val annotation = getAnnotation(adapter, "Adapter", Adapter::class.java.name)

    return  annotation?.let {
        val bindType = annotation?.let {
            getParamValueType(annotation, "bindType")
        }

        val viewholdersParams = getParamValueList(annotation, "holders")
        val viewholders = getViewHolders(/**GET VIEW HOLDERS*/)


        AdapterData(this.simpleName.getShortName(),
        this.packageName.asString(),

        viewholders = viewholders.map {
            it.toViewHolderData(resolver)
        }.toList(),
        itemsType = ReturnsTypeData(/**...BIND TYPE*/))
    }

    return null
}
```

Упрощаем адаптер. Получение параметров

Маппим в AdaptersData

```
fun KSClassDeclaration.toAdapterData(resolver: Resolver):AdapterData? {
    val adapter = this
    val annotation = getAnnotation(adapter, "Adapter", Adapter::class.java.name)

    return  annotation?.let {
        val bindType = annotation?.let {
            getParamValueType(annotation, "bindType")
        }

        val viewholdersParams = getParamValueList(annotation, "holders")
        val viewholders = getViewHolders(/**GET VIEW HOLDERS*/)

        AdapterData(this.simpleName.getShortName(),
        this.packageName.asString(),

        viewholders = viewholders.map {
            it.toViewHolderData(resolver)
        }.toList(),
        itemsType = ReturnsTypeData(/**...BIND TYPE*/))
    }

    return null
}
```

USE TECH

Упрощаем адаптер. Собираем вместе

Маппинг моделей –> генерация

```
class AdapterProcessor constructor(val env: SymbolProcessorEnvironment)
: SymbolProcessor {

    protected val logger: KSPLogger = env.logger
    protected val codeGenerator = env.codeGenerator

    override fun process(resolver: Resolver): List<KSAnnotated> {

        val adapters = getAdapters(resolver)
        val data = adapters.mapNotNull {
            it.toAdapterData(resolver)
        }.toList()

        generateImplClass(data, codeGenerator)
        return emptyList()
    }
/**....*/
}
```

Упрощаем адаптер. Генерация Kotlin Poet

Маппинг моделей –> генерация

```
fun generateImplClass(adapters:  List<AdapterData>, codeGenerator: CodeGenerator) {
    adapters.forEach { classData ->
        val fileSource = classData.generateClassSource()

        val packageName = classData.packageName
        val className = classData.name
        val fileName = "_${className}Impl"

        codeGenerator.createNewFile(Dependencies.ALL_FILES, packageName,
        fileName , "kt").use { output ->
            OutputStreamWriter(output).use { writer ->
                writer.write(fileSource)
            }
        }
    }
}
```

Упрощаем адаптер. Исходное

```
class TestAdapter : RecyclerView.Adapter<TestViewHolder>() {
    val items: MutableList<Int> = mutableListOf()

    fun setupItems(items: List<Int>) {
        this.items.addAll(items)
    }

    override fun onCreateViewHolder(parent: ViewGroup,
    viewType: Int): TestViewHolder {

        return TestViewHolder(ItemTestLayoutBinding
        .inflate( LayoutInflater.from(
            parent.context
        ), parent, false))
    }

    override fun onBindViewHolder(holder: TestViewHolder,
            position: Int) {
        holder.tag = position
        holder.bindItem(items.get(position))
    }

    override fun getItemCount(): Int = items.count()

}
```

Упрощаем адаптеры. Исходные

Property:

- items

Functions:

- setupItems
- onCreateViewHolder
- onBindViewHolder
- getItemsCount

Упрощаем адаптер. Генерация Kotlin Poet

Маппинг моделей –> генерация

```
fun AdapterData.generateClassSource(): String {
    val classData = this
    /**...*/
    val itemsProperty = PropertySpec.builder("items", MutableList:
    :class.asTypeName().parameterizedBy(TypeVariableName(inputType!!)))
        .mutable(true)
        .initializer("mutableListOf()")
        .build()
    val setupFunc = FunSpec.builder("setupItems")/**...*/.build()

    val createFunc = FunSpec.builder("onCreateViewHolder")/**...*/.build()

    val itemsCount =  FunSpec.builder("getItemCount"). /**...*/.build()

    val bindFunc = FunSpec.builder("onBindViewHolder")/**...*/.build()
```

Упрощаем адаптер. Генерация Kotlin Poet

Маппинг моделей –> генерация

```
    val implClassSpec = TypeSpec.classBuilder("${classData.name}Impl")
        .superclass(ClassName("","androidx.recyclerview.widget.RecyclerView.Adapter")
        .parameterizedBy(TypeVariableName(type!!)))
        .addProperty(itemsProperty)
        .addFunction(createFunc)
        .addFunction(bindFunc)
        .addFunction(itemsCount)
        .addFunction(setupFunc)
        .build()

    return FileSpec.builder(classData.packageName, implClassName)
        .addFileComment("Generated automatically")
        .addType(implClassSpec)
        .addImports(classData.imports)
        .build()
}
```

Упрощаем адаптер. Расширение ViewHolder

Добавить код в класс ViewHolder только через расширение

```
/**
*public fun TestViewHolder.bindItem(item: kotlin.Int): Unit {
        setupData(item)
*}
*/
val bindExt = FunSpec.builder("bindItem")
    .receiver(TypeVariableName("${viewHolder.name}"))
    .addStatement("${viewHolder..setupFunc}(item)")
    .addParameter("item", TypeVariableName("${viewHolder.bindType.name}"))
    .build()


//Add to file
return FileSpec.builder(classData.packageName, implClassName)
    .addFileComment("Generated automatically")
    .addType(implClassSpec)
    .addFunction(bindExt)
    .addImports(classData.imports + listOf("android.view.LayoutInflater"))
    .build().toString().replace(WILDCARDIMPORT, "*")
```

Упростили адаптер. Результат

_TestAdapterImpl.kt

Generated source files should not be edited. The changes will be lost when sources are regenerated.

```kotlin
// Generated automatically
package com.azharkova.kspgenandroid

import ...

public fun TestViewHolder.bindItem(item: kotlin.Int): Unit {
    setupData(item)
}

```

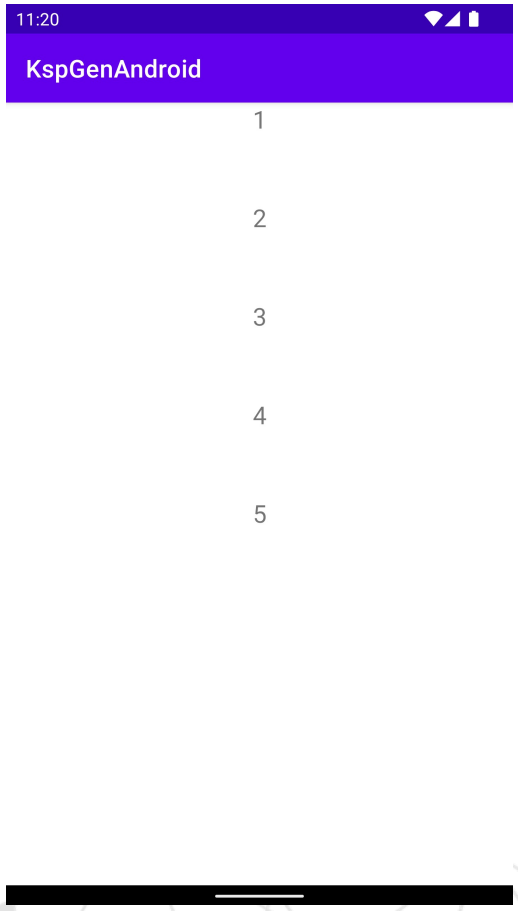
Упростили адаптер. Результат

```
15
16  public class TestAdapterImpl : RecyclerView.Adapter<TestViewHolder>() {
17      public var items: List<kotlin.Int> = mutableListOf()
18
19      public override fun onCreateViewHolder(parent: android.view.ViewGroup, viewType: Int):
20              TestViewHolder =
21          TestViewHolder(
22              ItemTestLayoutBinding.inflate(
23                  LayoutInflater.from(parent.context),
24                  parent, attachToParent: false
25              )
26          )
```

Проверяем результат

Минимум кода,
максимум результата

```
val adapter = TestAdapterImpl().apply {
    binding.list.layoutManager = LinearLayoutManager(this@MainActivity)
    binding.list.adapter = this
}
adapter.setupItems(listOf(1,2,3,4,5))
```



Выводы

- Процессор KSP – основа всего
- Пишешь шаблон и процессор один раз, используешь повторно
- Код генерируется сам

Выводы

- Процессор KSP – основа всего
- Пишешь шаблон и процессор один раз, используешь повторно
- Код генерируется сам

Можно генерировать View

Кейс 2. Замахнемся на Composable

@Composable + KCP. Достаточно ли аннотации?

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

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

USE TECH

@Composable + KSP. Достаточно аннотации

View < - > Composable

```
@Target(AnnotationTarget.CLASS, AnnotationTarget.FUNCTION)
@Retention(AnnotationRetention.SOURCE)
annotation class ToView

@Target(AnnotationTarget.CLASS)
@Retention(AnnotationRetention.SOURCE)
annotation class ToComposable
```

View < - > Composable

1. AndroidView

2. AndroidViewBinding + inflate

USE TECH

@ToComposable

set-function -> Modifier

```
@ToComposable
class TestText @JvmOverloads constructor(
    context: Context,
    attrs: AttributeSet? = null,
    defStyle: Int = 0
) : AppCompatTextView(context, attrs, defStyle) {

    //To Modifier
    fun setText(text: String) {
        //....
    }
}
```

USE TECH

@ToComposable

Factory

Update

```
@Composable
public fun TestText(
  click: (() -> Unit),
  text: String,
  modifier: Modifier = Modifier,
  viewConfig: TestText.() -> Unit,
): Unit {
  AndroidView(
    factory = { context -> TestText(context) },
    modifier = modifier,
    update = { view ->
      view.apply {
        setText(text)
        setClick().apply {
          click.invoke()
        }
        viewConfig()
      }
    }
  )
}
```

@ToComposable

Factory

Update

```
@Composable
public fun TestText(
  click: (() -> Unit),
  text: String,
  modifier: Modifier = Modifier,
  viewConfig: TestText.() -> Unit,
): Unit {
  AndroidView(
    factory = { context -> TestText(context) },
    modifier = modifier,
    update = { view ->
      view.apply {
        setText(text)
        setClick().apply {
          click.invoke()
        }
        viewConfig()
      }
    }
  )
}
```

План-капкан для ToComposable + AndroidView

1. Ищем все @ToComposable View
2. Примитивные свойства – > конструктор
3. Свойства – объекты –> mutable
4. Генерируем

План-капкан для ToComposable + AndroidView

1. Ищем все @ToComposable View
2. Примитивные свойства – > конструктор
3. Свойства – объекты –> mutable
4. Генерируем

Проще некуда

ComposableProcessor

ViewVisitor – процессинг + генерация

```
class ComposeProcessor constructor(private val env: SymbolProcessorEnvironment)
    : SymbolProcessor {

    override fun process(resolver: Resolver): List<KSAnnotated> {
        val viewVisitor = ViewVisitor(resolver,codeGenerator, logger)

        val views = toComposables(resolver)

        views.forEach {
            it.accept(viewVisitor,Unit)
        }
        it.accept(functionVisitor, Unit)
       }
        return emptyList()
    }

}
```

ComposableProcessor

ToComposable

```
fun toComposables(resolver: Resolver):Sequence<KSClassDeclaration> {
    return resolver.getSymbolsWithAnnotation((ToComposable::class.java).name)
        .filterIsInstance<KSClassDeclaration>().distinct()
  }
```

@ToComposable. Функция

Аннотация
+ параметры
+ модификаторы

```
val funSpecBuilder = FunSpec.builder(functionName)
        .addAnnotation(composableClassName)
        .addParameters(paramSpecBuilders)
        .addParameter(
            ParameterSpec.builder("modifier", modifierClassName)
                .defaultValue("%T", modifierClassName)
                .build()
        )
        .addParameter(
            ParameterSpec.builder("viewConfig", viewConfigLambdaTypeName)
                .build()
        )
```

@ToComposable. AndroidView

Заворачиваем в AndroidView и шаблон

```
.addCode(
        """
        |%T(
        |  factory = { context -> $className(context) },
        |  modifier = modifier,
        |  update = { view ->
        |    view.${addApplyBlockIfNecessary()}
        |  }
        |)
        """.trimMargin(),
        androidViewClassName,
    )
```

@ToComposable. Маппинг параметров

```
fun addApplyBlockIfNecessary(): String {
    return if (paramSpecBuilders.isNotEmpty()) {
        """
        |apply {
        |      ${
            paramFunctionMap.keys.joinToString("\n      ") { function ->
                "${function.simpleName.asString()}(" +
                        "${
                            paramFunctionMap[function].orEmpty().joinToString { param ->
                                param.name?.asString() ?: ""
                            }
                        })"
            }
        }
        | ${modifiersList.joinToString("\n")}
        |      viewConfig()
        |    }
        """.trimMargin()
    } else {
        "apply {\n ${modifiersList.joinToString("\n")} \nviewConfig() }"
    }
}
```

@ToComposable. Маппинг параметров

ParamFunctionsMap (Function, Parameters):

- Есть параметры
- Не override
- ReturnType = Unit
- Public

Добавим события

Composable. ClickModifier

@ClickModifier

```
@ToComposable
class TestText @JvmOverloads constructor(
    context: Context,
    attrs: AttributeSet? = null,
    defStyle: Int = 0
) : AppCompatTextView(context, attrs, defStyle) {


    @PropertyModifier
    var testProperty: String = ""

    @ClickModifier
    fun setClick() {
        this.setOnClickListener {  }
    }
}
```

Composable. ClickModifier

@ClickModifier -> в модификаторы

```
getAnnotation(declaration,"ClickModifier",ClickModifier::class.java.name)?.let {

        modifiersList += "${declaration.simpleName.asString()}()"
        +".apply{\n${declaration.simpleName.asString().replace("set","").lowercase()}.invoke()}"

        ParameterSpec.builder(
            name = declaration.simpleName.asString().replace("set","").lowercase(),
            type = TypeVariableName(declaration.isFunction())
        ).build()
}
```

Composable. Modifier

@ClickModifier

kotlin/.../TestText.kt

Generated source files should not be edited. The changes will be lo

```kotlin
package com.azharkova.kspgenandroid

import ...

@Composable
public fun TestText(
  click: (() -> Unit),
  text: String,
  modifier: Modifier = Modifier,
  viewConfig: TestText.() -> Unit,
): Unit {
  AndroidView(
    factory = { context -> TestText(context) },
    modifier = modifier,
    update = { view ->
      view.apply { this: TestText
        setText(text)
        setClick().apply { this: Unit
          click.invoke()
        }
        viewConfig()
      }
    }
  )
}
```

View < - > Composable

@ToComposable

@ToView + Click

ViewHolder -> ToComposable?

AndroidView не подойдет

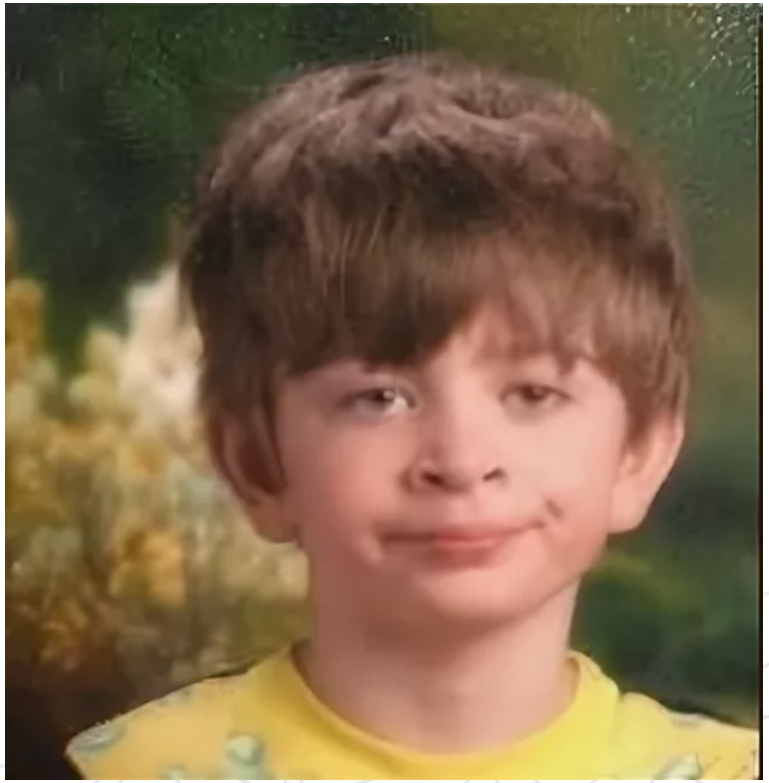
'Better Drugs, Better Health': A Human Genome Pioneer Talks About the Future of Genetics

2023-04-11T20:06:00Z

ViewHolder -> @ToComposable

1 Сконвертировать все элементы в ToComposable?

2 Заменить контролы на Composable?



USETECH

Подключим ViewHolder через AndroidViewBinding

ViewHolder -> Composable

LayoutBinding -> Composable

'Better Drugs, Better Health': A Human Genome Pioneer Talks About the Future of Genetics

2023-04-11T20:06:00Z

ViewHolder -> Composable. AndroidViewBinding

LayoutBinding -> Composable

```
@Composable
public fun NewsItemViewComposable(model: NewsItemModel): Unit {
  AndroidViewBinding(ItemLayoutBinding::inflate) {
    //SETUP DATA

  }
}
```

ViewHolder -> Composable. DataBinding

DataBinding

```
@Composable
public fun NewsItemViewComposable(model: NewsItemModel): Unit {
  AndroidViewBinding(ItemLayoutBinding::inflate) {
    //SETUP DATA

  }
}
```

ViewHolder -> Composable. Map аннотация

@MapModel

@MapView(model)

```
//Map data
@MapModel
data class NewsItemModel(
    @MapText(R.id.title_news) val title: String?, val description: String?,
    @MapImage(R.id.image_news) val urlToImage: String?,
    @MapText(R.id.date_news) val publishedAt: String?,
    val content: String?
)

//View + Binding
@MapView(model = NewsItemModel::class)
class NewsItemView(@BindLayout val itemViewBinding: ItemLayoutBinding)
```

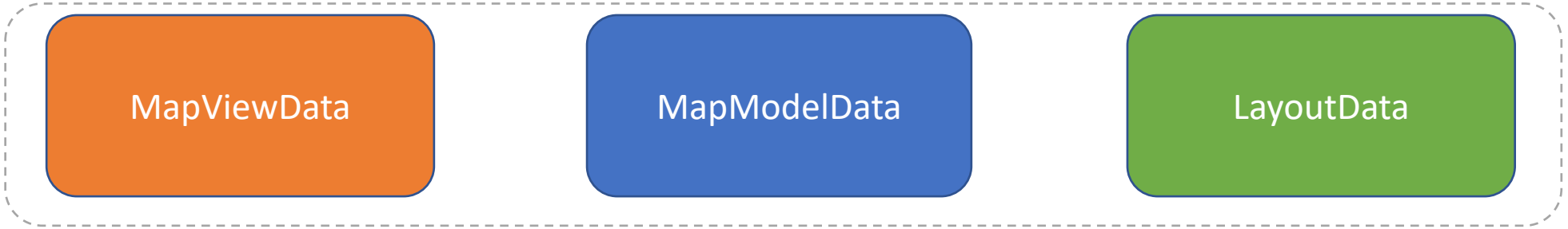
ViewHolder -> Composable. Map аннотация

@MapModel

@MapView(model)

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

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

@Target(AnnotationTarget.CLASS)
@Retention(AnnotationRetention.SOURCE)
annotation class MapView(val model: KClass<*>)

@Target(AnnotationTarget.CLASS)
@Retention(AnnotationRetention.SOURCE)
annotation class MapModel
```

ViewHolder -> Composable. Подготовим модели



MapViewData

Маппим модель

```
fun KSClassDeclaration.toViewData(resolver: Resolver): MapViewData? {
    val declaration = this

    /**
    Get name + package
    */

    val annotation = getAnnotation(this, "MapView", MapView::class.java.name)
    return annotation?.let {

        val model = getParamValueType(it, "model")
        val layoutData = getBindLayoutData()

        val modelData = getModelsData(model)

        MapViewData(
            name,
            packageName,
            model = model,
            layout = layoutData,
            imports,
            modelData = modelData
        )
    } ?: null
}
```

ViewHolder -> Composable. MapModelData

Маппим текстовые поля
и изображения

```
fun KSClassDeclaration.toModelData(): MapModelData? {
        val declaration = this

        val properties = declaration.getAllProperties().toList()

        val textFields: List<FieldData> = getTextProperties()


        val imageFields: List<FieldData> = getImageProperties()

        return MapModelData(textFields + imageFields)
    }
```

ViewHolder -> Composable. MapModelData

Маппим текстовые поля
и изображения

```
fun KSClassDeclaration.toModelData(): MapModelData? {
        val declaration = this

        val properties = declaration.getAllProperties().toList()

        val textFields: List<FieldData> = getTextProperties()


        val imageFields: List<FieldData> = getImageProperties()

        return MapModelData(textFields + imageFields)
    }
```

ViewHolder -> Composable. MapModelData

Маппим текстовые поля
и изображения

List<FieldData>

```
val properties = declaration.getAllProperties().toList()
val textFields = properties.mapNotNull { property ->
    getAnnotation(property, "MapText", MapText::class.java.name)?.let {
        val id = (getParamValue(it, "id") as? Int)

        FieldData(
            id = id ?: 0,
            fieldName = "",
            valueName = property.simpleName.asString(),
            type = Field.TEXT
        )

    } ?: null
}
```

ViewHolder -> Composable. MapModelData

Маппим текстовые поля
и изображения

List<FieldData>

```
val properties = declaration.getAllProperties().toList()
val textFields = properties.mapNotNull { property ->
    getAnnotation(property, "MapText", MapText::class.java.name)?.let {
        val id = (getParamValue(it, "id") as? Int)

        FieldData(
            id = id ?: 0,
            fieldName = "",
            valueName = property.simpleName.asString(),
            type = Field.TEXT
        )

    } ?: null
}
```

ViewHolder -> Composable. MapModelData

Маппим текстовые поля
и изображения

List<FieldData>

```
val properties = declaration.getAllProperties().toList()
val textFields = properties.mapNotNull { property ->
    getAnnotation(property, "MapText", MapText::class.java.name)?.let {
        val id = (getParamValue(it, "id") as? Int)

        FieldData(
            id = id ?: 0,
            fieldName = "",
            valueName = property.simpleName.asString(),
            type = Field.TEXT
        )

    } ?: null
}
```

MapViewData

Маппим layout

```
fun KSClassDeclaration.toViewData(resolver: Resolver): MapViewData? {
        val declaration = this

        /**
        Get name + package
        */

        val annotation = getAnnotation(this, "MapView", MapView::class.java.name)
        return annotation?.let {

            val model = getParamValueType(it, "model")
            val layoutData = getBindLayoutData()

            val modelData = getModelsData(model)

            MapViewData(
                name,
                packageName,
                model = model,
                layout = layoutData,
                imports,
                modelData = modelData
            )
        } ?: null
    }
```

MapViewData

Готово

```
fun KSClassDeclaration.toViewData(resolver: Resolver): MapViewData? {
        val declaration = this

        /**
        Get name + package
        */

        val annotation = getAnnotation(this, "MapView", MapView::class.java.name)
        return annotation?.let {

            val model = getParamValueType(it, "model")
            val layoutData = getBindLayoutData()

            val modelData = getModelsData(model)

            MapViewData(
                name,
                packageName,
                model = model,
                layout = layoutData,
                imports,
                modelData = modelData
            )
        } ?: null
    }
```

ViewHolder -> Composable. MapViewData. Генерируем

findViewById – id

```
val funSpecBuilder = FunSpec.builder(implClassName)
    .addAnnotation(composableClassName)
    .addParameter(ParameterSpec.builder("model",
    TypeVariableName(model.resolveTypeName())).build())
    .addStatement("AndroidViewBinding(${classData.layout?.name}::inflate) {")
    .apply {
        classData.modelData?.fields?.forEach {
            addStatement("(this.root.rootView.findViewById(${it.id})${(it.statement())}")
        }
    }
    .addStatement("}")
    .build()
```

ViewHolder -> Composable. MapViewData. Генерируем

findViewById – id

```
val funSpecBuilder = FunSpec.builder(implClassName)
    .addAnnotation(composableClassName)
    .addParameter(ParameterSpec.builder("model",
    TypeVariableName(model.resolveTypeName())).build())
    .addStatement("AndroidViewBinding(${classData.layout?.name}::inflate) {")
    .apply {
        classData.modelData?.fields?.forEach {
            addStatement("(this.root.rootView.findViewById(${it.id})${(it.statement())}")
        }
    }
    .addStatement("}")
    .build()
```

BindView + KCP

Можно было бы дополнить…

```
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)
```

MapViewData. Проверяем

_NewsItemViewImpl.kt

Generated source files should not be edited. The changes will be lost when sources are regenerated.

Code

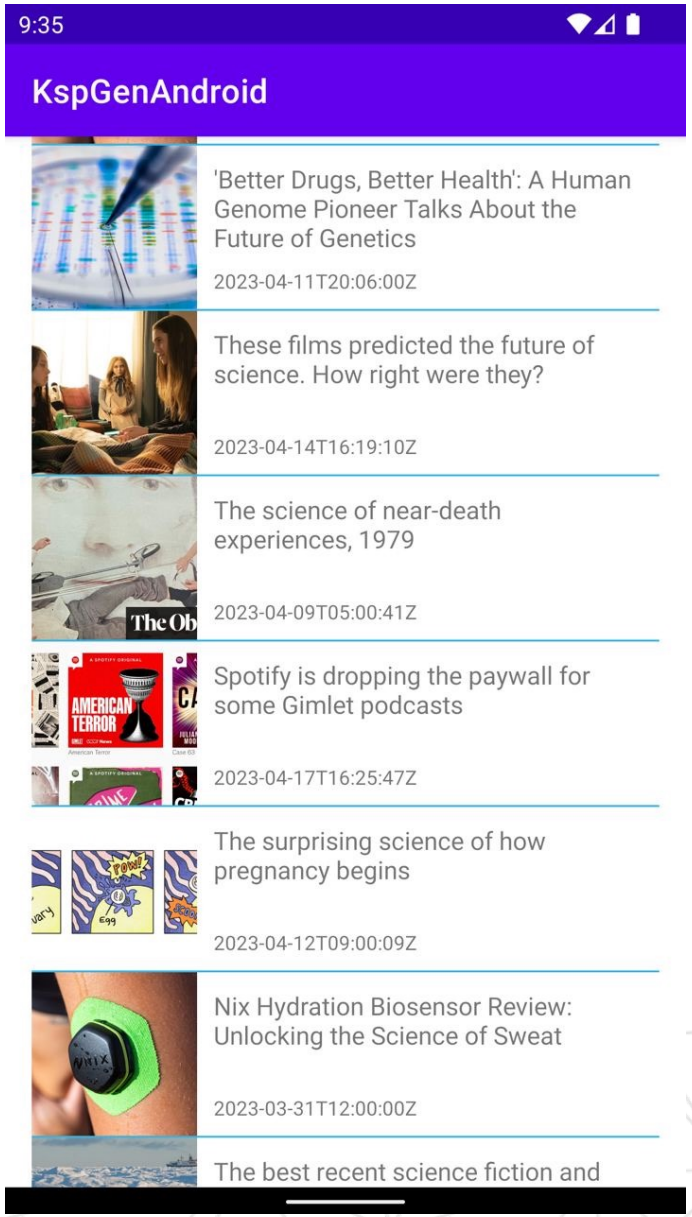
```kotlin
// Generated automatically
package com.azharkova.kspgenandroid.data

import ...

@Composable
public fun NewsItemViewComposable(model: NewsItemModel): Unit {
    AndroidViewBinding(ItemLayoutBinding::inflate) { this: ItemLayoutBinding
        (this.root.rootView.findViewById( id: 2131231190) as? TextView)?.text = model.title
        (this.root.rootView.findViewById( id: 2131230870) as? TextView)?.text = model.publishedAt
        (this.root.rootView.findViewById( id: 2131230949) as? ImageView)?.load(model.urlToImage)
    }
}
```

Размялись. Соберем из этого список

Кейс 3. Насоmроsим целый экран

KSP. Composable экран с логикой

Готовый сэмпл
https://github.com/anioutkazharkova/kgen-android



Composable экран с логикой. Аннотация и подключение

@ListScreen:
- VM (ViewModel)
- Item (элемент списка)

```
@Target(AnnotationTarget.CLASS)
@Retention(AnnotationRetention.SOURCE)
annotation class ListScreen(val vm: KClass<*>, val item: KClass<*>)

@ListScreen(NewsListVM::class, NewsItemView::class)
class NewsListScreenView
```

План-капкан для Composable экрана

1. Ищем все @ListScreen класс
2. Из аннотации берем ссылку (KClass) на VM и элемент списка
3. Для элемента генерируем Composable (по кейс 2)
4. Маппим все данные для экрана
5. Генерируем

Composable экран с логикой. Биндинг ViewModel

Используем готовую VM
@VmResult – data property

```
//Результат запроса в VM
@Retention(AnnotationRetention.SOURCE)
@Target(AnnotationTarget.PROPERTY)
annotation class VmResult


interface IVM {
    fun loadData()
}

class NewsListVM: ViewModel(),IVM {
    @VmResult
    var data = MutableStateFlow<List<NewsItemModel>?>(null)
    /**
     Код разный
    */
}
```

Composable экран с логикой. Модель для списка

Model – VM
View – item
Property - Data

```
data class ListScreenData(
    var name: String = "",
    var packageName: String = "",
    var model: KSType? = null,
    var view: KSType? = null,
    var propertyName: String = "",
    var returnType: KSType? = null,
    var imports: List<String> = emptyList(),
)
```

Composable экран с логикой. Итоговый процессор

MapViewData

ListScreenData

```
class ViewProcessor constructor(private val env: SymbolProcessorEnvironment) :
    SymbolProcessor {

    override fun process(resolver: Resolver): List<KSAnnotated> {
        var views = getViews(resolver)
        generateImplClass( views.mapNotNull {
            it.toViewData(resolver)
        }.toList(),codeGenerator)
        var lists = getLists(resolver)

        val listData = lists.mapNotNull {
            it.toListData()
        }.toList()

       generateList(listData, codeGenerator,logger)
        return emptyList()
    }
    }
```

Composable экран с логикой. Итоговый процессор

MapViewData

ListScreenData

```
class ViewProcessor constructor(private val env: SymbolProcessorEnvironment) :
    SymbolProcessor {

    override fun process(resolver: Resolver): List<KSAnnotated> {
        var views = getViews(resolver)
        generateImplClass( views.mapNotNull {
            it.toViewData(resolver)
        }.toList(),codeGenerator)
        var lists = getLists(resolver)

        val listData = lists.mapNotNull {
            it.toListData()
        }.toList()

       generateList(listData, codeGenerator,logger)
        return emptyList()
    }
    }
```

Composable экран с логикой. ListScreenData. Маппинг

```
fun KSClassDeclaration.toListData():ListScreenData? {
        val declaration = this
        val name = declaration.simpleName.asString()
        val packageName = declaration.packageName.asString()

      return getAnnotation(this, "ListScreen", ListScreen::class.java.name)?.let {
            annotation ->

            val model = getParamValueType(annotation, "vm")
            val view =  getParamValueType(annotation, "item")

            var propertyName: String = ""
            var propertyType: KSType? = null

            model?.declaration?.let {
                val result = (it as? KSClassDeclaration)?.findVMProperty()

            propertyName = result?.simpleName?.asString().orEmpty()
            propertyType = result?.type?.resolve()
            }

            ListScreenData(name,packageName,model,view,propertyName,propertyType,imports)
        } ?: null
    }
```

Composable экран с логикой.ListScreenData. Генерация

Шаблонируем по максимуму

```
val implClassName = "${classData.name}Composable"

val funSpecBuilder = FunSpec.builder(implClassName)
    .addAnnotation(composableClassName)
   .addParameter(ParameterSpec.builder("viewModel",
       TypeVariableName(model.resolveTypeName())).build())
    .addStatement("LaunchedEffect(Unit) {\n" +
            "        viewModel.loadData()\n" +
            "    }")
    .addStatement("val data by viewModel.${propertyName}.collectAsState()")
    .addStatement("LazyColumn(\n" +
            "        contentPadding =  PaddingValues(horizontal = 16.dp, vertical = 8.dp)\n) {\n"
            "        items(data.orEmpty()) {\n" +
            "            ${classData.view?.resolveTypeName()}Composable(it)\n" +
            "        }\n" +
            "    }")
    .build()
```

Composable экран с логикой. Проверяем результат

Минимум кода,
максимум результата

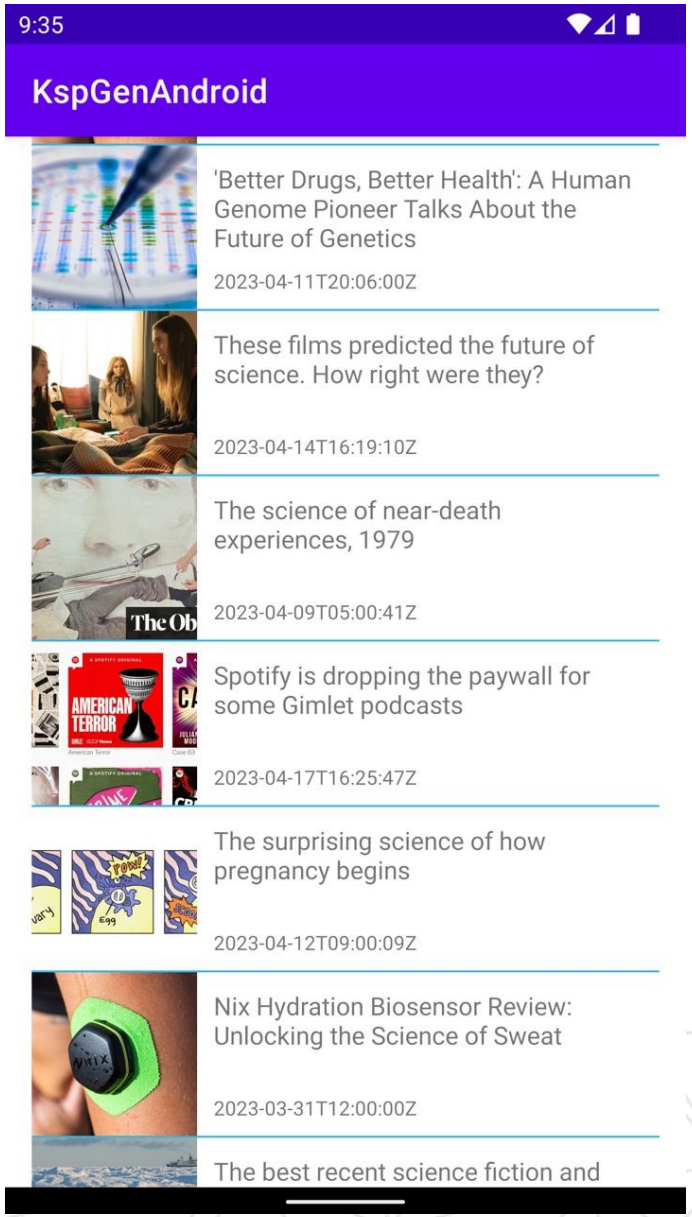
Generated source files should not be edited. The changes will be lost when sources are regenerated.

```
1   // Generated automatically
2   package com.azharkova.kspgenandroid
3
4   import ...
20
21  @Composable
22  public fun NewsListScreenViewComposable(viewModel: NewsListVM): Unit {
23      LaunchedEffect(Unit) { this: CoroutineScope
24          viewModel.loadData()
25      }
26      val data by viewModel.data.collectAsState()
27      LazyColumn(
28          contentPadding = PaddingValues(horizontal = 16.dp, vertical = 8.dp)
29      ) { this: LazyListScope
30          items(data.orEmpty()) { this: LazyItemScope  it: NewsItemModel
31              NewsItemViewComposable(it)
32          }
33      }
34  }
```

USE TECH

Composable экран с логикой. Проверяем результат

Минимум кода,
максимум результата



Выводы

- Самое главное – правильный процессор
- Пиши раз – генерируй постоянно
- Для генерации нового кода, а не изменения старого
- Ограничения архитектурные (расширения или DI)
- Совмещайте с КСР

Sources

- https://github.com/google/ksp#kotlin-symbol-processing-api
- https://github.com/anioutkazharkova/ksp-kmm-cases
- https://developer.android.com/build/migrate-to-ksp
- https://github.com/anioutkazharkova/kgen-android

Спасибо за внимание!

@anioutkajarkova

azharkova
prettygeeknotes