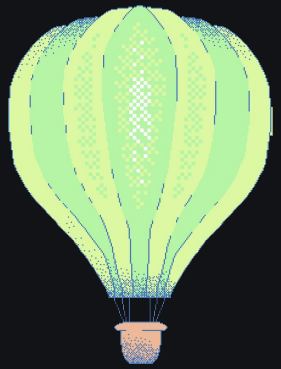


Kodein в Android
Что за зверь и как его готовить

Максим Качинкин
Android Tech Lead
Dodo Engineering

DODO ENGINEERING

Максим Качинкин
Android Tech Lead
Dodo Engineering

Dodo Pizza
Drinkit

Dodo Brands – это про IT




17 стран мира


880+ точек питания


22+ млн клиентов


600+ человек
в Dodo Brands

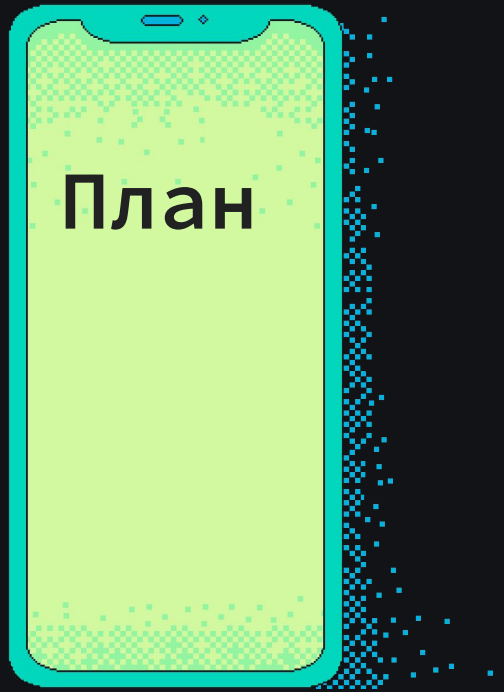
Dodo Brands – это про IT

17 стран мира

880+ точек питания

22+ млн клиентов

600+ человек в Dodo Brands



- Kodein
- Наш опыт с Kodein
- Сравнение с Dagger и Koin

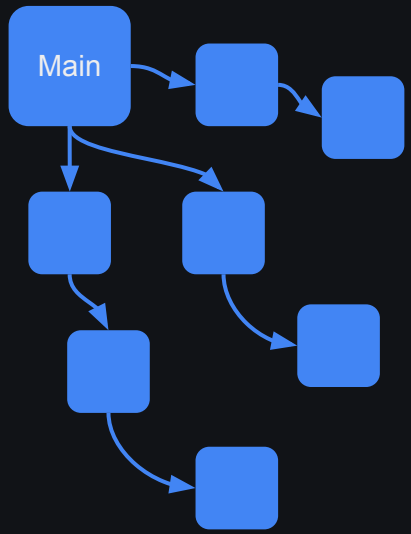
Disclaimer

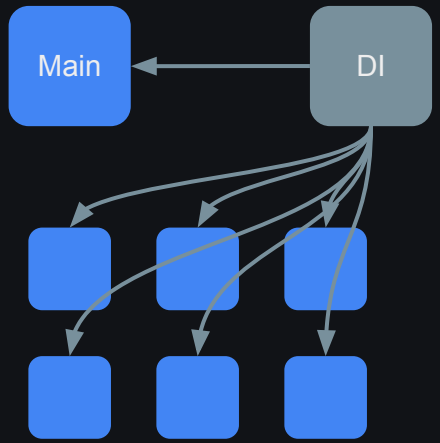
- Это не реклама Kodein. Я рассказываю про свой опыт.

Disclaimer

- Это не реклама Kodein. Я рассказываю про свой опыт.
- Может показаться похожим на Koin. Это нормально!

Dependency injection





- Dagger / **Dagger 2**
- Hilt
- **Koin**
- Kodein
- Toothpick
- Guice
- …

Kodein

- Version 1.x (from apr. 2015): Concept
- Version 2.x (from oct. 2015): Robust API
- ...
- Version 7.x (jun. 2020)

Kodein

- Version 1.x (from apr. 2015): Concept
- Version 2.x (from oct. 2015): Robust API
- ...
- Version 7.x (jun. 2020)
- Version 8.x (soon)

Kodein. Core Principles.

Kodein. Core Principles.

- Declarative DSL

```
fun moduleOfAuth() = DI.Module("authModule") {
    bind<PhoneRemoteDataSource>() with singleton {
        PhoneRemoteDataSourceImpl(instance())
    }

    bind<PhoneDataSource>() with singleton {
        PhoneDataSourceImpl(
            endpoint = instance(),
            countryDataSource = instance()
        )
    }
    ...
}
```

Kodein. Core Principles.

- Declarative DSL
- Reified types

```
inline fun <reified T> instance(): T = container.getInstance(T::class.java)


val smsDataSource: SmsDataSource = instance()
```

Kodein. Core Principles.

- Declarative DSL
- Reified types

```
public inline fun <reified T: Any> DI.Builder.bindSingleton(...
     noinline creator: DirectDI.() -> T
): Unit
```

```
bindSingleton<SmsCodeRetriever> { GoogleSmsCodeRetriever(instance()) }
```

Kodein. Core Principles.

- Declarative DSL
- Reified types
- Infix functions

```
bind<SmsCodeRetriever>().with(singleton { GoogleSmsCodeRetriever(instance()) })

bind<SmsCodeRetriever>() with singleton { GoogleSmsCodeRetriever(instance()) }
```

DI или Service Locator?

DI или Service Locator?

Kodein docs:

Injection & Retrieval

When dependencies are **injected**, the class is *provided* its dependencies at construction.
When dependencies are **retrieved**, the class is *responsible* for getting its own dependencies.

Using dependency **injection** is a bit more cumbersome, but your classes are "pure": they are unaware of the dependency container. Using dependency **retrieval** is easier (and allows more tooling), but it does binds your classes to the *Kodein-DI* API.

Finally, in retrieval, **everything is lazy by default**, while there can be no lazy-loading using injection.

https://kosi-libs.org/kodein/7.19/core/injection-retrieval.html#_injection_retrieval

DI или Service Locator?

```
class CartPaymentDataSourceImpl(di: DIAware) : CartPaymentDataSource, DIAware by di {

    private val paymentDataSource by instance<PaymentDataSource>()
    private val paymentMethodsDataSource by instance<PaymentMethodsDataSource>()
    private val paymentStatusService by instance<PaymentStatusService>()
```

DI или Service Locator?

```
bind<SmsRemoteDataSource>() with singleton {
    SmsRemoteDataSourceImpl(
        endpoint = instance(),
        environment = instance(),
        countryDataSource = instance()
    )
}
```

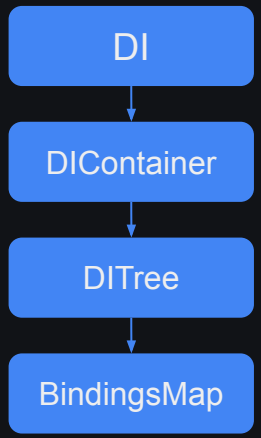
JSR-330

- Есть
- Работает на рефлексии
- Рекомендуется только при миграции с Java проектов

Kodein

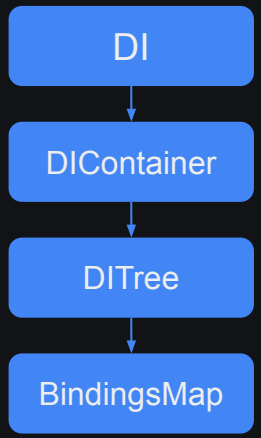
```
bind<String>() with provider { "Hello world" }
```

Kodein




```
bind<String>() with provider { "Hello world" }
```

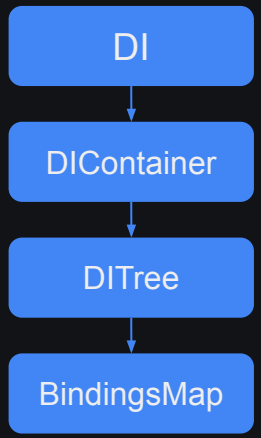
Kodein




```
bind<String>() with provider { "Hello world" }
```

```
Map<DI.Key<*, *, *>, List<DIDefinition<*, *, *>>>
```

Kodein




```
bind<String>() with provider { "Hello world" }
```

```
Map<DI.Key<*, *, *>, List<DIDefinition<*, *, *>>>
```

```
public data class Key<in C : Any, in A, out T: Any>(
        val contextType: TypeToken<in C>,
        val argType: TypeToken<in A>,
        val type: TypeToken<out T>,
        val tag: Any?
)
```

Kodein

```
val di = DI {
    bind<String>() with provider { "Hello world" }
    bind<String>(tag = "name") with provider { "My name" }
}
```

Kodein

```
val di = DI {
    bind<String>() with provider { "Hello world" }
    bind<String>(tag = "name") with provider { "My name" }
    bind<Repository>() with singleton { RepositoryImpl(name = instance()) }
    bind<OrderInfoService>() with factory { orderId ->
        OrderInfoServiceImpl(orderId = orderId, repository = instance())
    }
    bind<OrderInfoService>() with multiton { orderId ->
        OrderInfoServiceImpl(orderId = orderId, repository = instance())
    }
}
```

Многомодульность

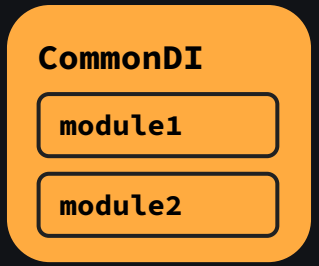
Многомодульность

- Есть DI, есть Modules

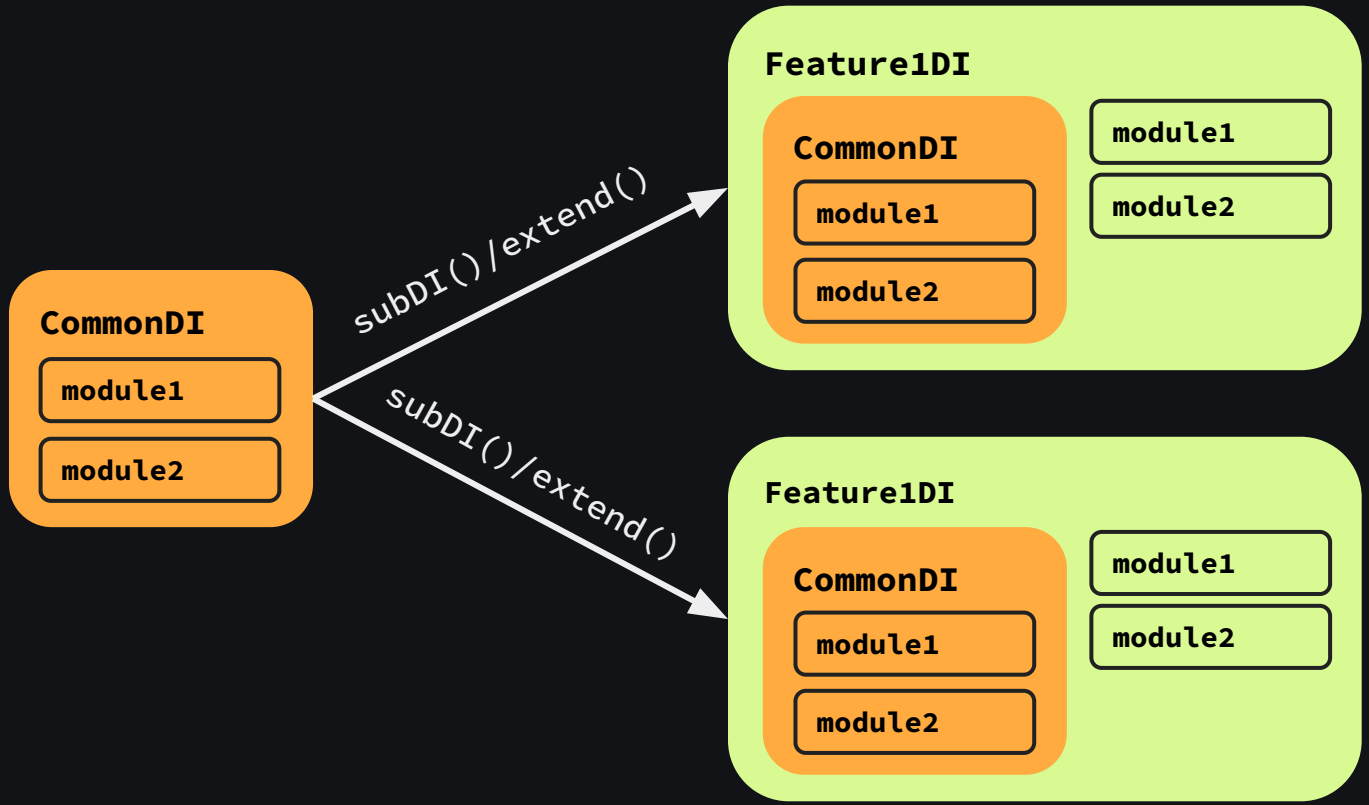
Многомодульность

- Есть DI, есть Modules
- subDI()/extend()/addExtend() - создать subDI

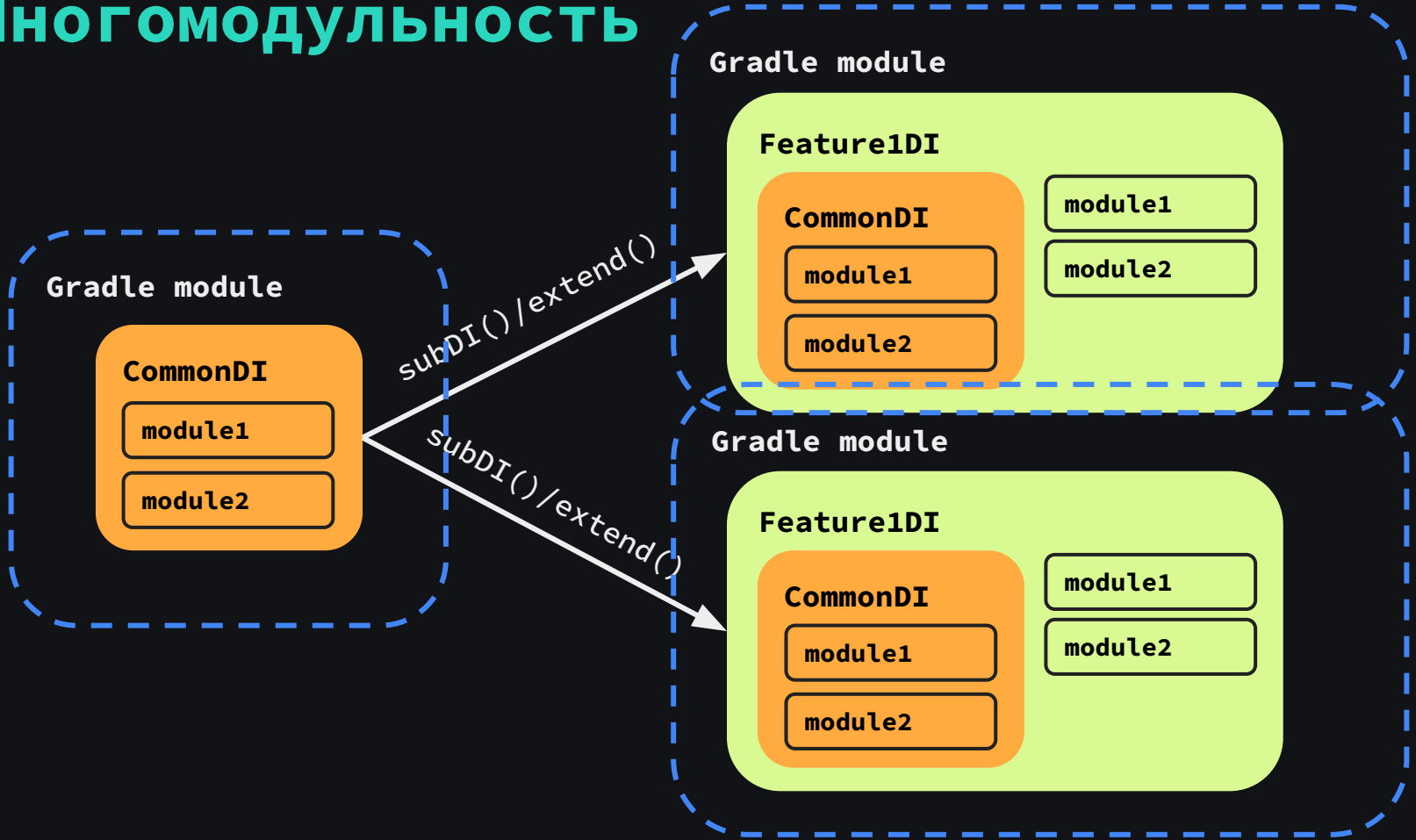
Многомодульность



Многомодульность



Многомодульность



KMM

KMM

- Поддерживает KMM

KMM

- Поддерживает KMM
- androidArm32, androidArm64, iosArm32, iosArm64, iosX64, linuxArm32Hfp, linuxMips32, linuxMipsel32, linuxX64, macosX64, mingwX64

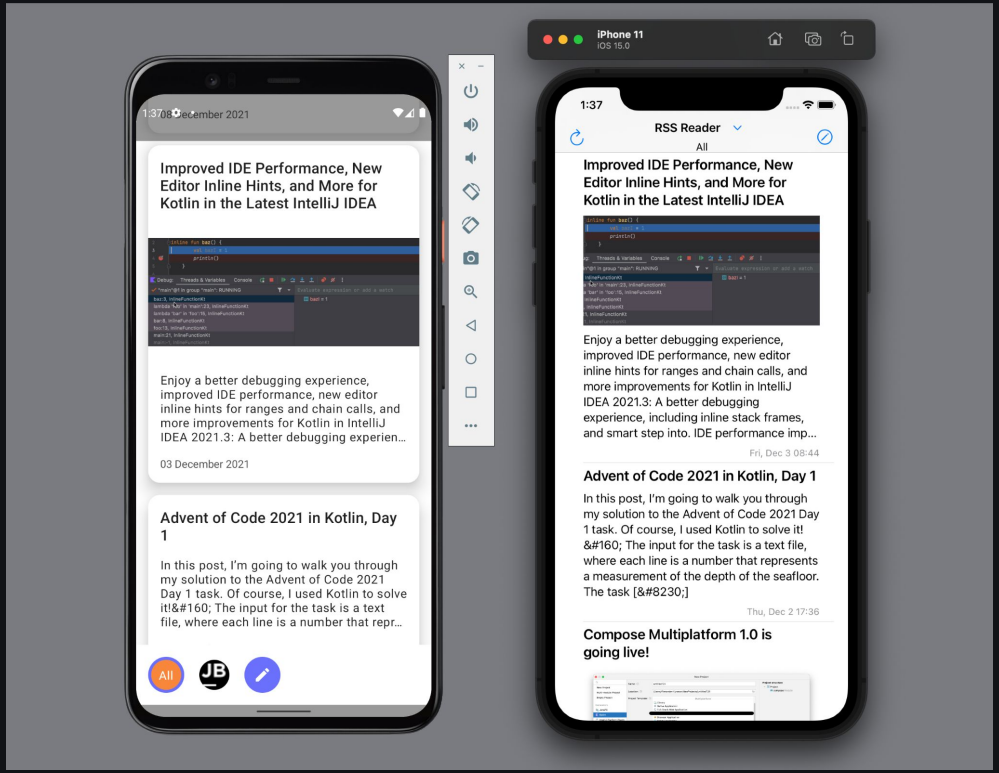
KMM

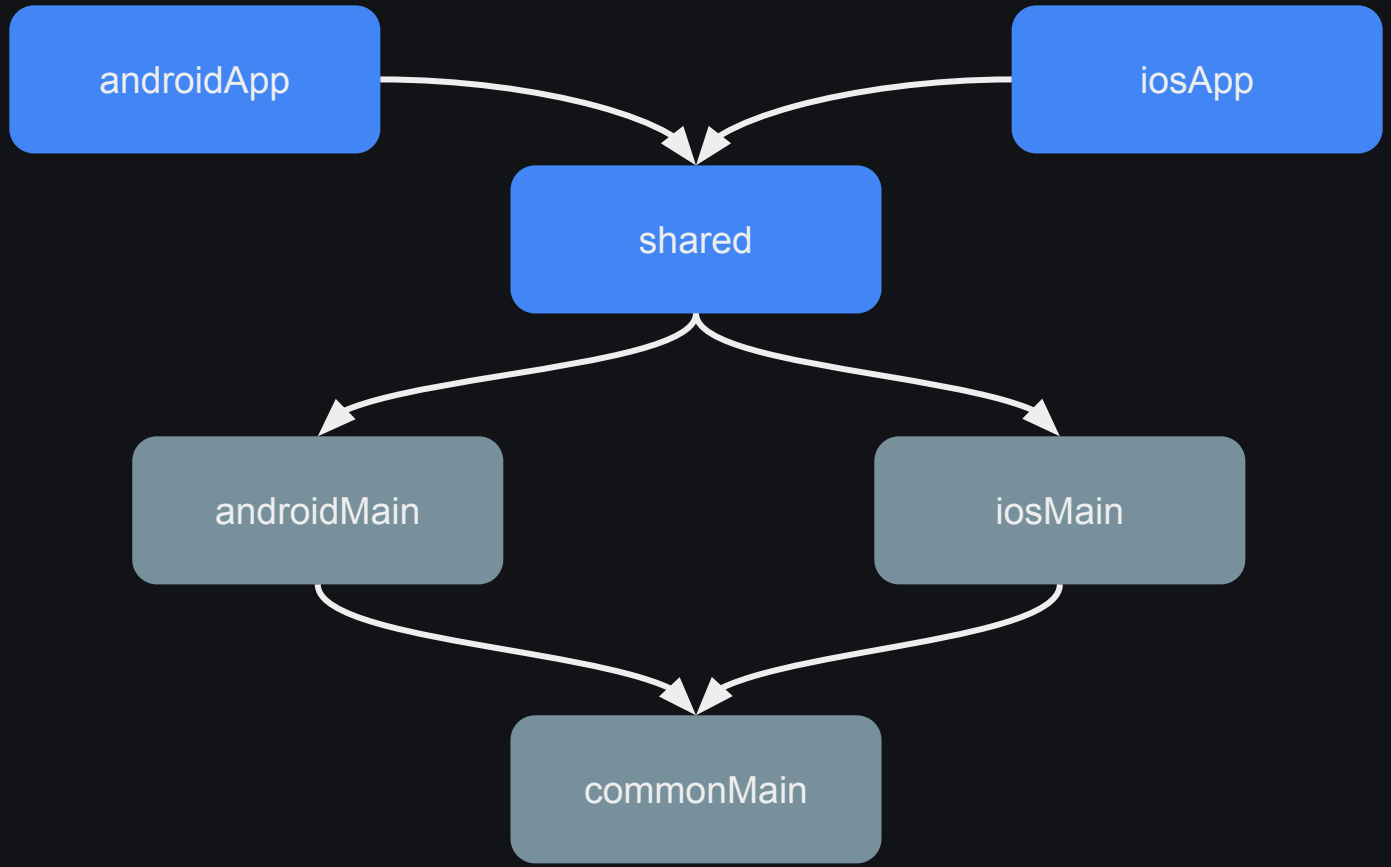
- Поддерживает KMM
- androidArm32, androidArm64, iosArm32, iosArm64, iosX64, linuxArm32Hfp, linuxMips32, linuxMipsel32, linuxX64, macosX64, mingwX64

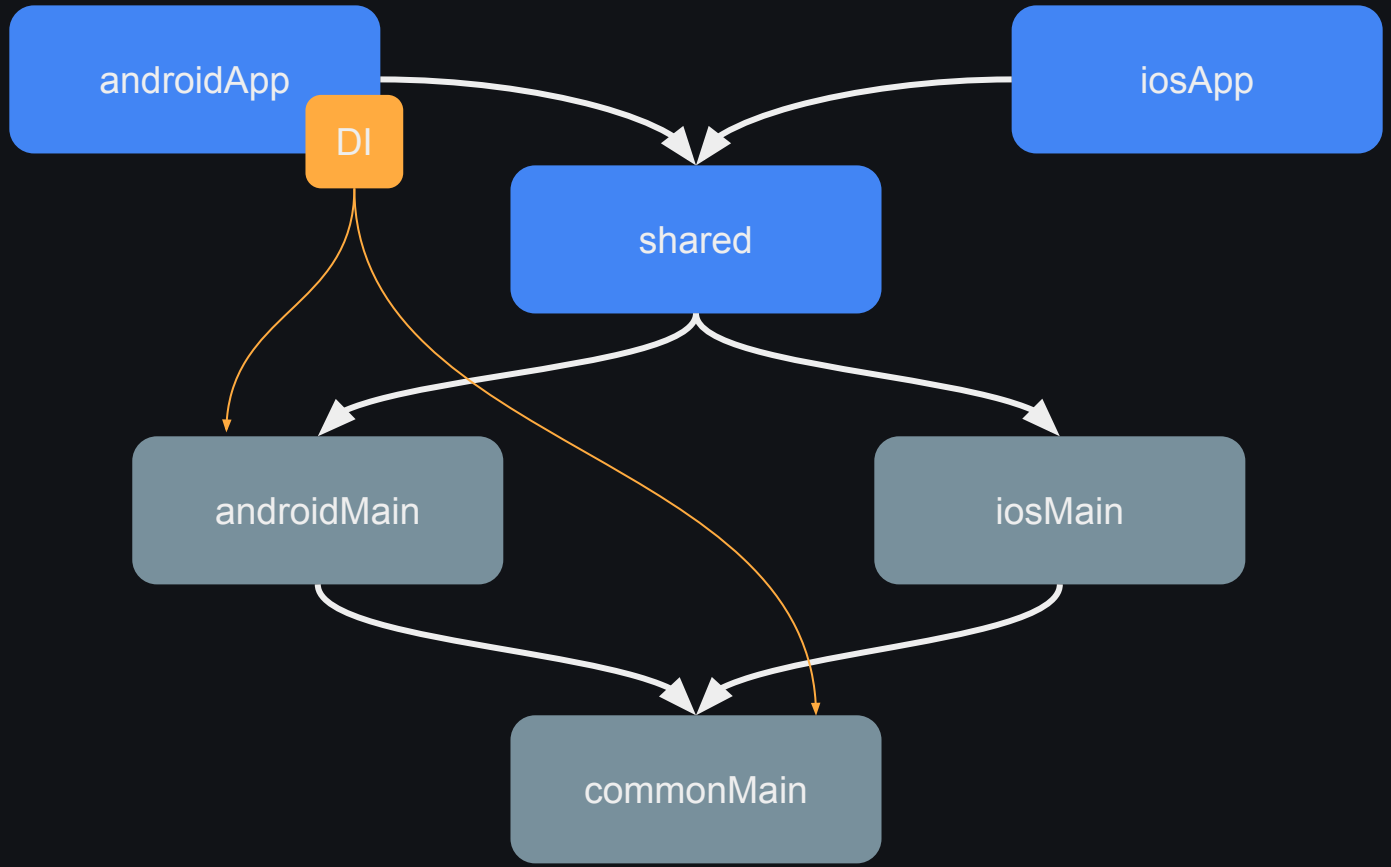
```
sourceSets {
    val commonMain by getting {
        dependencies {
            implementation(libs.kodein)
            ...
        }
    }
}
```

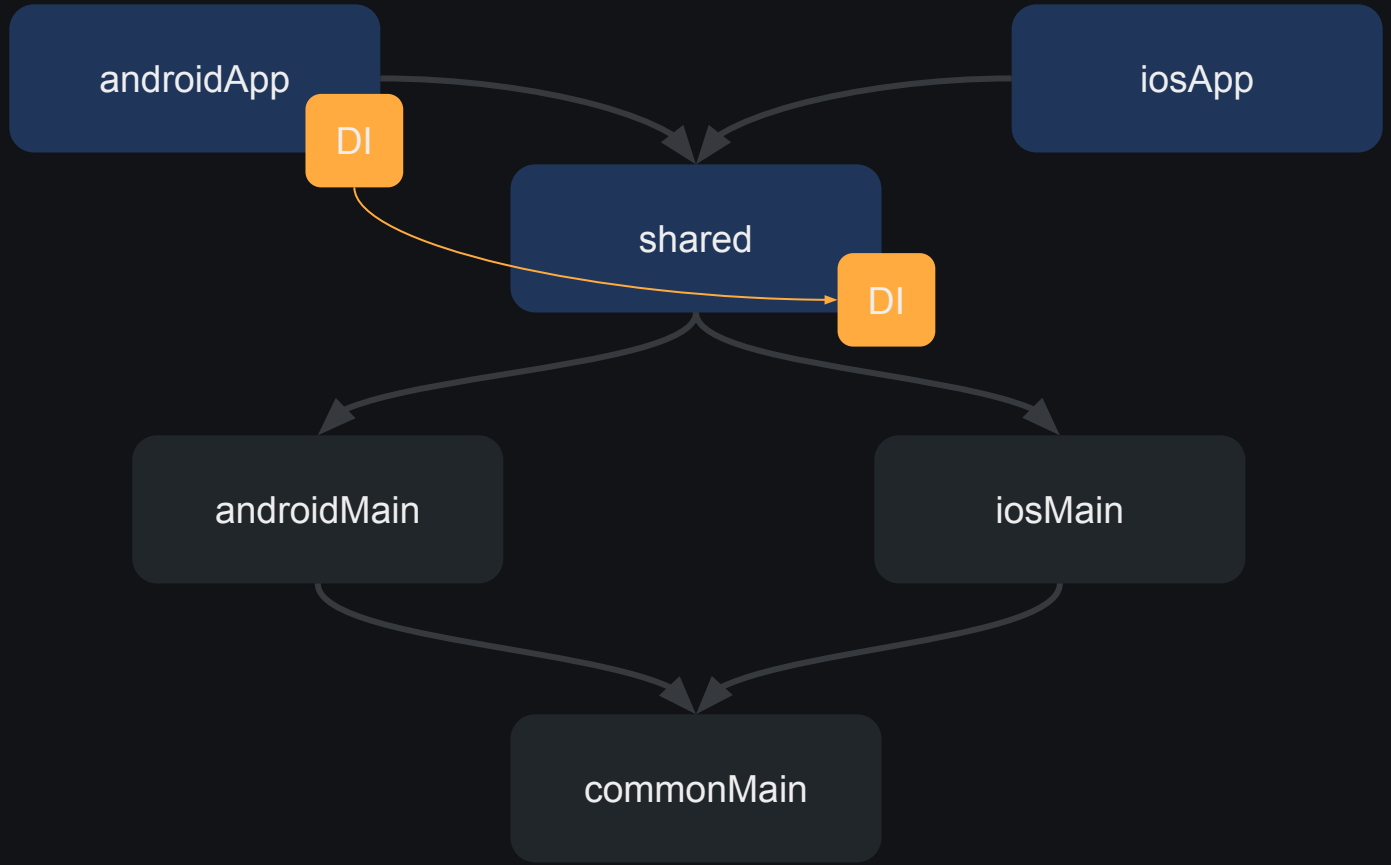
KMM. Пример.

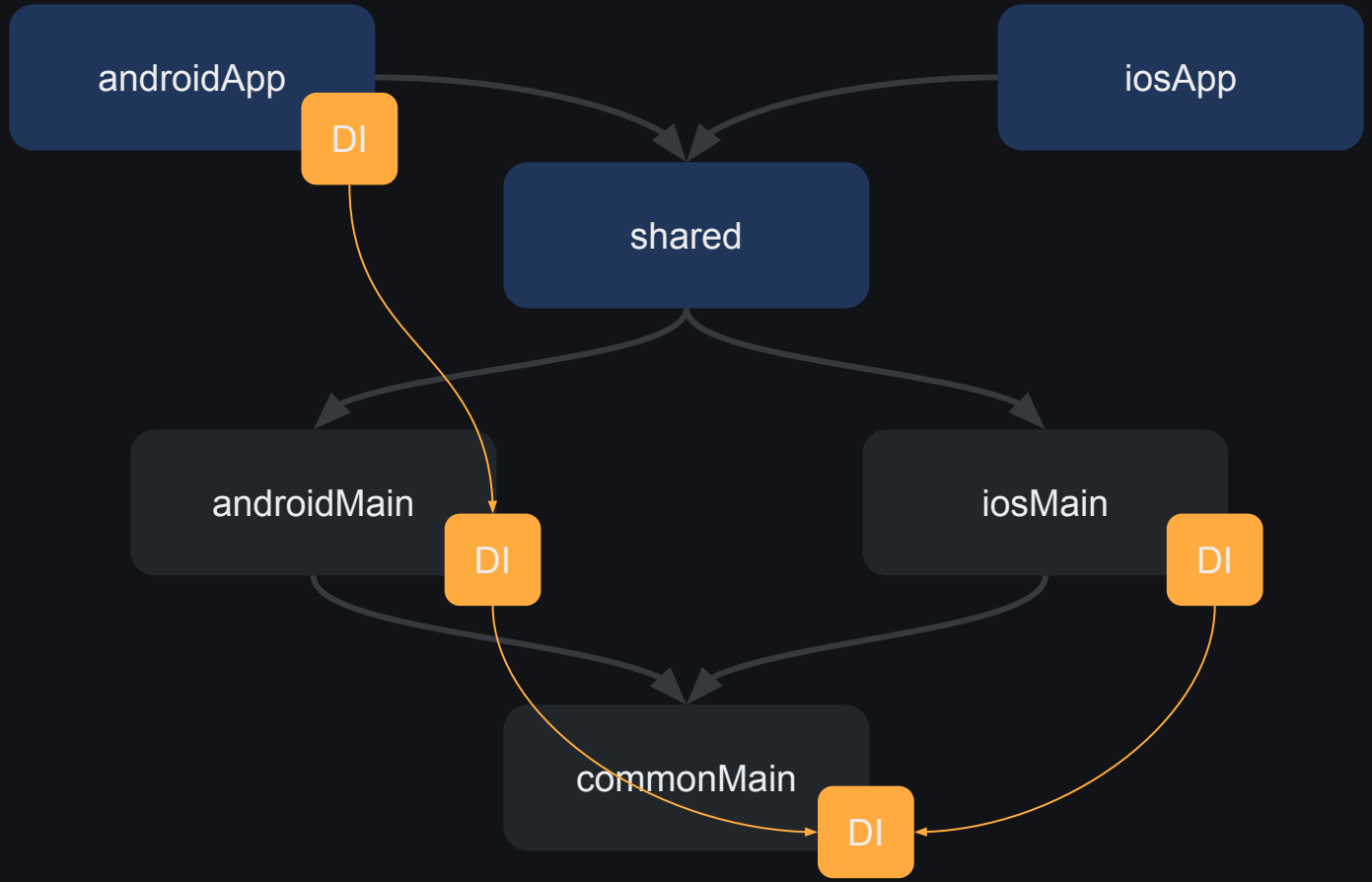
KMM RSS Reader

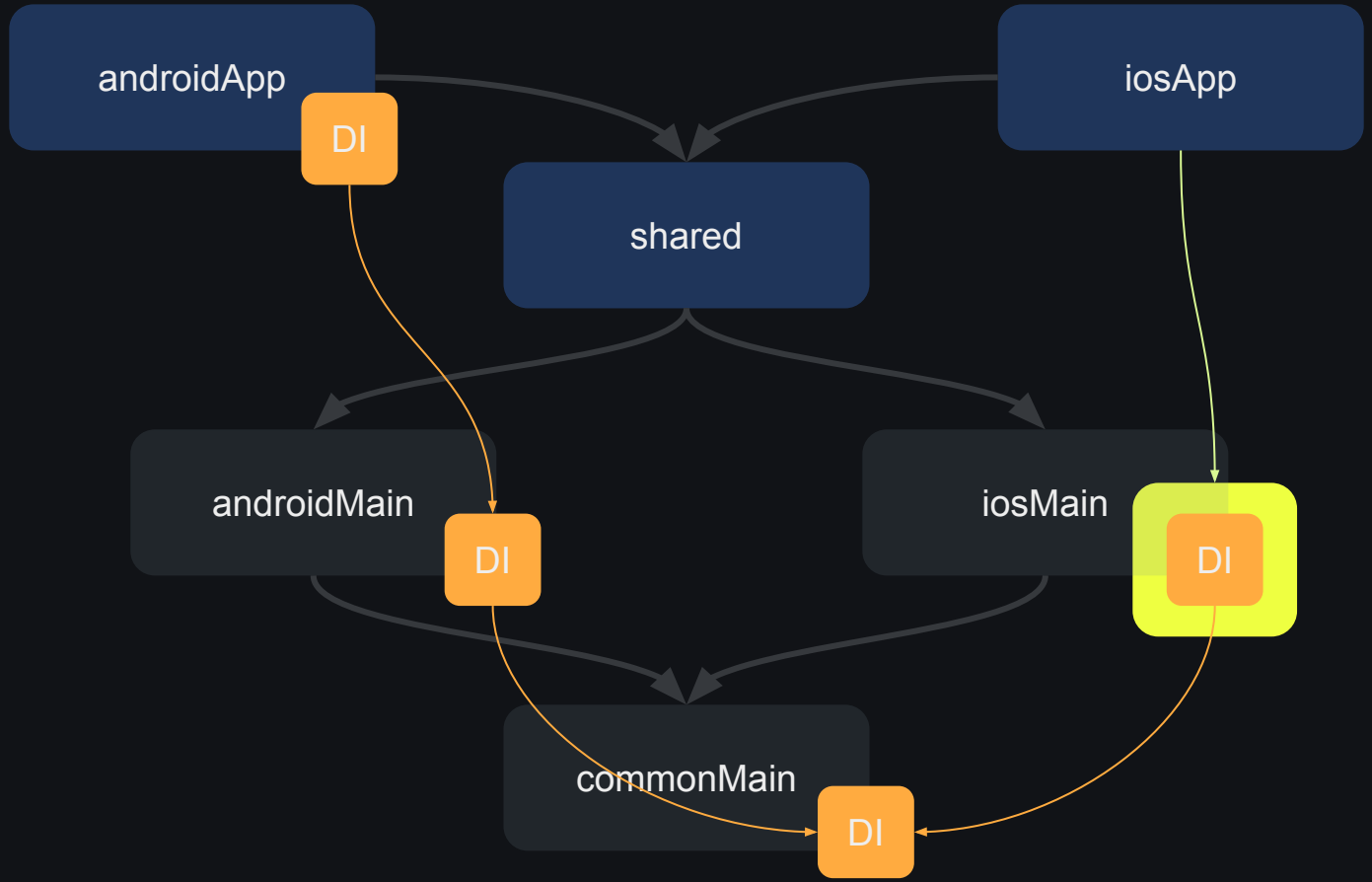



androidApp
iosApp
shared
androidMain
iosMain
commonMain


androidApp
DI
iosApp
shared
androidMain
iosMain
commonMain


androidApp
DI
iosApp
shared
DI
androidMain
iosMain
commonMain


androidApp
DI
iosApp
shared
androidMain
DI
iosMain
DI
commonMain
DI


androidApp
DI
iosApp
shared
androidMain
DI
iosMain
DI
commonMain
DI

```
fun sharedCommonDI() = DI {
    import(sharedCommonModule())
}

fun sharedCommonModule() = DI.Module("sharedCommonModule") {
    bind<Json>() with singleton {
        Json {
            ignoreUnknownKeys = true
            isLenient = true
            encodeDefaults = false
        }
    }

    bind<FeedStorage>() with singleton {
        FeedStorage(
            settings = instance(),
            json = instance(),
        )
    }

    bind<FeedLoader>() with singleton {
        FeedLoader(
            httpClient = instance(),
            parser = instance(),
        )
```

commonMain

commonMain

```
fun sharedCommonDI() = DI {
    import(sharedCommonModule())
}

fun sharedCommonModule() = DI.Module("sharedCommonModule") {
    bind<Json>() with singleton {
        Json {
            ignoreUnknownKeys = true
            isLenient = true
            encodeDefaults = false
        }
    }

    bind<FeedStorage>() with singleton {
        FeedStorage(
            settings = instance(),
            json = instance(),
        )
    }

    bind<FeedLoader>() with singleton {
        FeedLoader(
            httpClient = instance(),
            parser = instance(),
        )
```

```
fun sharedAndroidDI() = DI {
    extend(di = sharedCommonDI(), copy = Copy.All)
    import(sharedAndroidModule())
}

fun sharedAndroidModule() = DI.Module("sharedAndroidModule") {
    bind<Settings>() with singleton {
        SharedPreferencesSettings(
            delegate = instance(arg = RSS_READER_PREF_KEY),
        )
    }

    bind<HttpClient>() with singleton {
        AndroidHttpClient(
            withLog = true,
        )
    }

    bind<FeedParser>() with singleton {
        AndroidFeedParser()
    }
}
```

androidMain

androidMain

```
fun sharedAndroidDI() = DI {
    extend(di = sharedCommonDI(), copy = Copy.All)
    import(sharedAndroidModule())
}

fun sharedAndroidModule() = DI.Module("sharedAndroidModule") {
    bind<Settings>() with singleton {
        SharedPreferencesSettings(
            delegate = instance(arg = RSS_READER_PREF_KEY),
        )
    }

    bind<HttpClient>() with singleton {
        AndroidHttpClient(
            withLog = true,
        )
    }

    bind<FeedParser>() with singleton {
        AndroidFeedParser()
    }
}
```

```
fun sharedIosDI() = DI {
    extend(di = sharedCommonDI(), copy = Copy.All)
    import(sharedIosModule())
}

fun sharedIosModule() = DI.Module("sharedIosModule") {
    bind<Settings>() with singleton {
        NSUserDefaultsSettings(
            delegate = NSUserDefaults.standardUserDefaults(),
        )
    }

    bind<HttpClient>() with singleton {
        IosHttpClient(
            withLog = true,
        )
    }

    bind<FeedParser>() with singleton {
        IosFeedParser()
    }
}
```

iosMain

```kotlin
fun sharedIosDI() = DI {
    extend(di = sharedCommonDI(), copy = Copy.All)
    import(sharedIosModule())
}

fun sharedIosModule() = DI.Module("sharedIosModule") {
    bind<Settings>() with singleton {
        NSUserDefaultsSettings(
            delegate = NSUserDefaults.standardUserDefaults(),
        )
    }

    bind<HttpClient>() with singleton {
        IosHttpClient(
            withLog = true,
        )
    }

    bind<FeedParser>() with singleton {
        IosFeedParser()
    }
}
```

iosMain

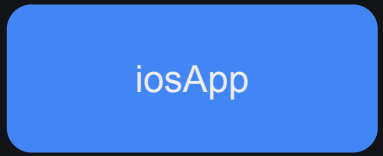
```kotlin
class App : Application(), Configuration.Provider, DIAware {

    override val di: DI = AppDI(app = this)

    ...
}

object AppDI {
    operator fun invoke(app: App) = DI {
        import(androidXModule(app))
        extend(di = sharedAndroidDI(), copy = Copy.All)
    }
}
```

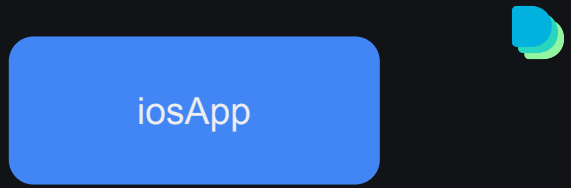
androidApp

```kotlin
object DependenciesFactory {
    fun create(): Dependencies = DependenciesImpl()
}

interface Dependencies {
    fun provideFeedStore(): FeedStore
}

class DependenciesImpl : Dependencies {

    private val di: DI by lazy { sharedIosDI() }

    override fun provideFeedStore(): FeedStore {
        return di.direct.instance()
    }
}
```

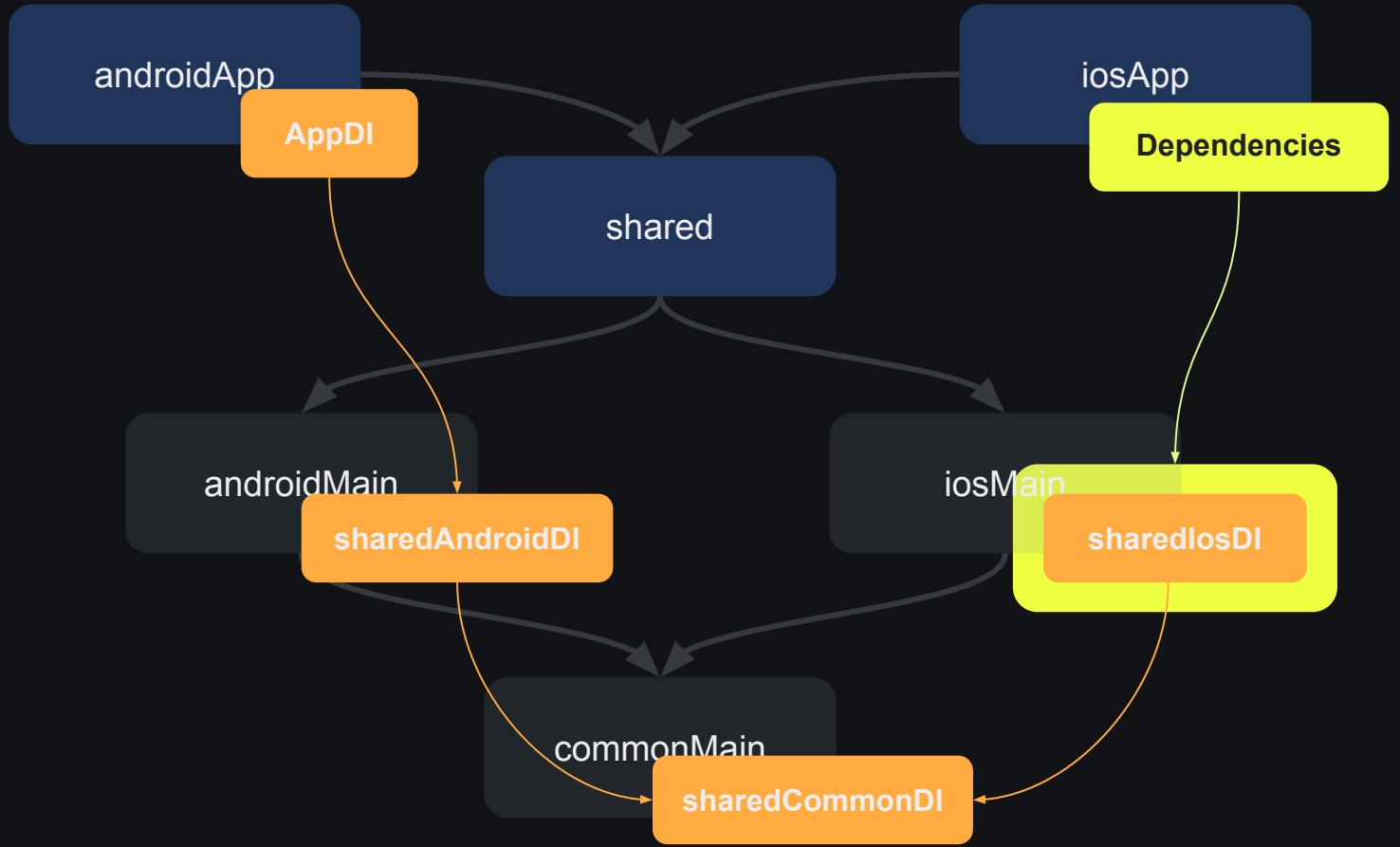
iosMain

```swift
@main
class RSSApp: App {
    let dependencies: Dependencies
    let store: ObservableFeedStore

    required init() {
        deps = DependenciesFactory.shared.create()
        store = ObservableFeedStore(store: dependencies.provideFeedStore())
    }

    var body: some Scene {
        WindowGroup {
            RootView().environmentObject(store)
        }
    }
}
```






```
@main
class RSSApp: App {
    let dependencies: Dependencies
    let store: ObservableFeedStore

    required init() {
        deps = DependenciesFactory.shared.create()
        store = ObservableFeedStore(store: dependencies.provideFeedStore())
    }

    var body: some Scene {
        WindowGroup {
            RootView().environmentObject(store)
        }
    }
}
```


androidApp
AppDI
iosApp
Dependencies
shared
androidMain
sharedAndroidDI
iosMain
sharedIosDI
commonMain
sharedCommonDI

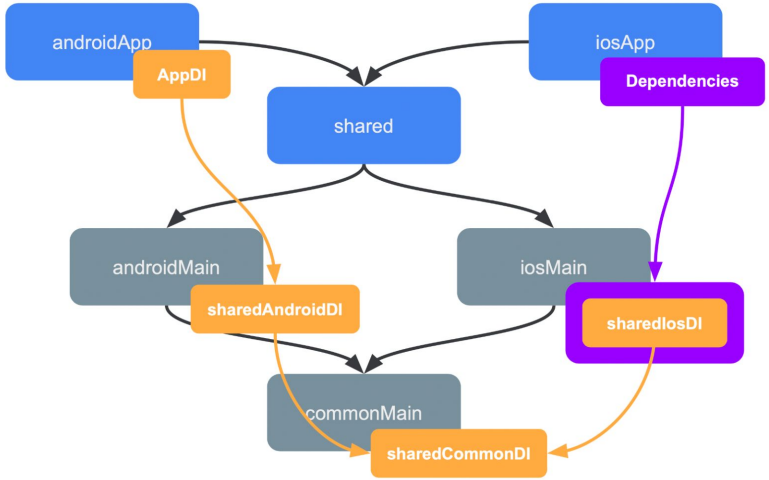
KMM RSS Reader с Kodein

github.com/makzimi/kmm-sample-with-kodein-di

README.MD

KMM RSS Reader with Kodein DI

This is an example of how DI can be organised in a KMM project using DI Kodein. This example shows how Kodein is used not only in the Android application, but also in a shared module in all source sets: commonMain, androidMain and iosMain.



KMM

https://www.youtube.com/watch?v=JtUJc4WYObo

Compose

Compose

```
inline fun <F, reified VM> F.viewModel(
    noinline ownerProducer: () -> ViewModelStoreOwner = { this },
    tag: Any? = null,
): Lazy<VM> where F : Fragment, F : DIAware, VM : ViewModel {
    return createViewModelLazy(
        viewModelClass = VM::class,
        storeProducer = { ownerProducer().viewModelStore },
        factoryProducer = {
            object : ViewModelProvider.Factory {
                override fun <T : ViewModel> create(modelClass: Class<T>): T {
                    val vmProvider = direct.provider<VM>(tag)
                    return vmProvider() as T
                }
            }
        }
    )
}
```

Compose

```
inline fun <F, reified VM> F.viewModel(
    noinline ownerProducer: () -> ViewModelStoreOwner = { this },
    tag: Any? = null,
): Lazy<VM> where F : Fragment, F : DIAware, VM : ViewModel {
    return createViewModelLazy(
        viewModelClass = VM::class,
        storeProducer = { ownerProducer().viewModelStore },
        factoryProducer = {
            object : ViewModelProvider.Factory {
                override fun <T : ViewModel> create(modelClass: Class<T>): T {
                    val vmProvider = direct.provider<VM>(tag)
                    return vmProvider() as T
                }
            }
        }
    )
}
```

Compose

- `implementation 'org.kodein.di:kodein-di-framework-compose:7.18.0'`

Kodein	Compose compiler	Kotlin
7.17.1	Compose 1.3.0-rc2	1.8.0
7.16.0	Compose 1.2.0	1.7.20
7.15.1	Compose 1.2.0	1.7.20
7.15.0	NOT COMPATIBLE	1.7.20
7.15.0-kotlin-1.7.20-RC	NOT COMPATIBLE	1.7.20
7.14.0	1.2.0-alpha01-dev745	1.7.10
7.13.1	1.2.0-alpha01-dev745	1.7.0
7.10.0	1.0.1-rc2	1.6.10

Compose

- implementation 'org.kodein.di:kodein-di-framework-compose:7.18.0'
- Работает через CompositionLocal

Compose

```
val di = DI {
    ...
}

@Composable
fun App() = withDI(di) {
    MyView {
        ContentView()
        BottomView()
    }
}
```

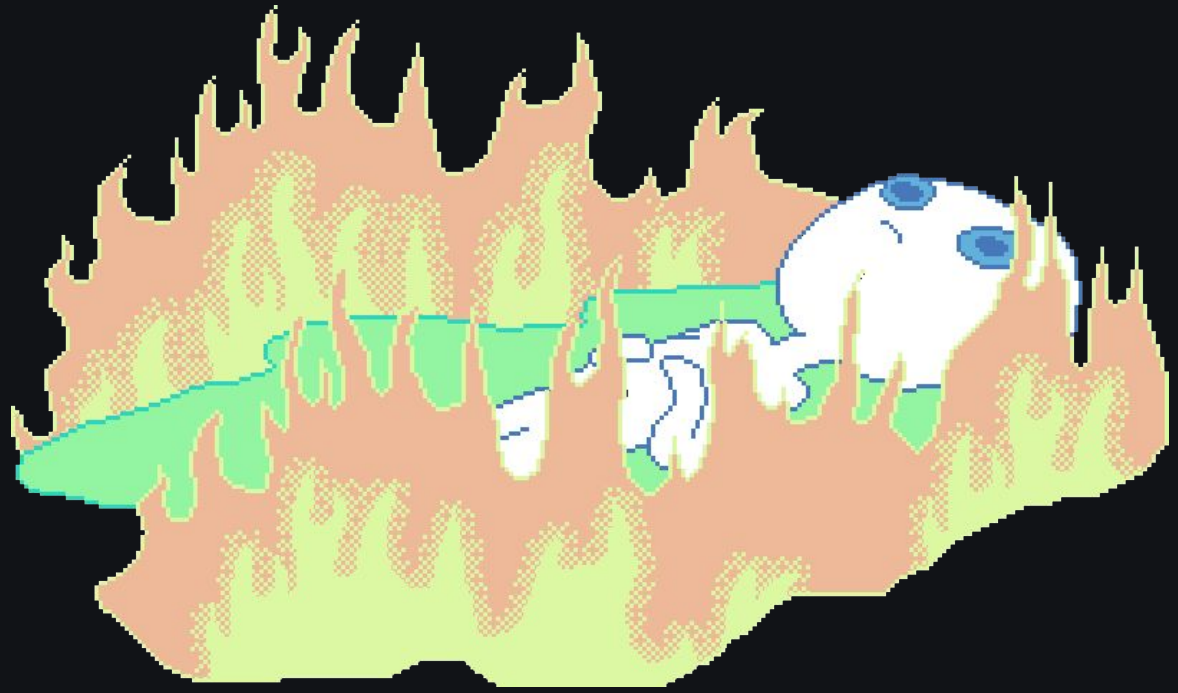
Compose

```
@Composable
fun App() = withDI(aModule, bModule) {
   MyView {
       ContentView()
       BottomView()
   }
}
```

Compose

```
@Composable
fun ContentView() {
    ...
    val di = localDI()
    val service: MyService by di.instance()
    ...
}

@Composable
fun ContentView() {
    ...
    val service: MyService by rememberDI { instance() }
    ...
}
```

Наш опыт с Kodein



Наш опыт с Kodein

- Целостность графа
- Retrieval vs Injection
- ConfigurableDI, Initializers
- Скоупы

Наш опыт с Kodein

- **Целостность графа**
- Retrieval vs Injection
- ConfigurableDI, Initializers
- Скоупы

Целостность графа 

Целостность графа 

- Ошибки в рантайме

Целостность графа 

- Ошибки в рантайме
  - Нет зависимости

```
bind<PhoneEndpoint>() with singleton { instance<Retrofit>().create(PhoneEndpoint::class.java) }


bind<PhoneDataSource>() with singleton(sync = false) {
  PhoneDataSourceImpl(
      endpoint = instance(),
      countryDataSource = instance()
  )
}
```

Целостность графа 

- Ошибки в рантайме
  - Нет зависимости

```
class SuggestUnitsDataSourceImpl(di: DIAware) : SuggestUnitsDataSource, DIAware by di {

    private val menuEndpoint by instance<MenuEndpoint>()
```

Целостность графа 

- Ошибки в рантайме
  - Нет зависимости

```
val di = DI {
    import(moduleA())
    import(moduleB())
}

fun moduleA() = DI.Module("moduleA") {
    bind<Repository>() with singleton { ... }
}

fun moduleB() = DI.Module("moduleB") {
     bind<Interactor>() with singleton { InteractorImpl(repository = instance()) }
}
```

Целостность графа 

- Ошибки в рантайме
  - Нет зависимости
  - Циклические зависимости

Циклические зависимости

```
internal class OkhttpAuthenticator(
     diAware: DIAware
) : Authenticator, DIAware by diAware {

    ...

    private val analyticsTracker: AnalyticsTracker by instance()
```

Циклические зависимости

```
java.lang.RuntimeException: Unable to get provider androidx.startup.InitializationProvider: androidx.startup.Start
     bind<Retrofit>
    ╔╚>bind<OkHttpClient>
    ║  ╚>bind<Authenticator>
    ║    ╚>bind<AnalyticsTracker>
    ║      ╚>bind<Analytics>
    ║        ╚>bind<Collection<out AnalyticsEventInterceptor>>
    ║          ╚>bind<CartDataSource>
    ║            ╚>bind<CartEndpoint>
    ║              ╚>bind<Retrofit>
    ╚═══════════════╝
    at android.app.ActivityThread.installProvider(ActivityThread.java:7244)
    at android.app.ActivityThread.installContentProviders(ActivityThread.java:6780)
```

Целостность графа 

- Нет compile time проверок by design

Целостность графа 

- Нет compile time проверок by design
- Runtime проверки by design

Целостность графа 

- Нет compile time проверок by design
- Runtime проверки by design
- Можно сделать deploy time проверки

Целостность графа 

- Ошибки в рантайме

  -> injection

  -> test

Целостность графа 

- Ошибки в рантайме

  -> injection

  -> test

injection

```
class CartPaymentDataSourceImpl(di: DIAware) : CartPaymentDataSource, DIAware by di {

 private val paymentDataSource by instance<PaymentDataSource>()
 private val paymentMethodsDataSource by instance<PaymentMethodsDataSource>()
 private val paymentStatusService by instance<PaymentStatusService>()
```

injection

```
class CartPaymentDataSourceImpl(di: DIAware) : CartPaymentDataSource, DIAware by di {

 private val paymentDataSource by instance<PaymentDataSource>()
 private val paymentMethodsDataSource by instance<PaymentMethodsDataSource>()
 private val paymentStatusService by instance<PaymentStatusService>()
```

injection

```
class CartPaymentDataSourceImpl(di: DIAware) : CartPaymentDataSource, DIAware by di {

    private val paymentDataSource by instance<PaymentDataSource>()
    private val paymentMethodsDataSource by instance<PaymentMethodsDataSource>()
    private val paymentStatusService by instance<PaymentStatusService>()
```

```
class SmsDataSourceImpl(
 private val dataSourceOfSms: SmsRemoteDataSource,
 private val dataSourceOfPhone: PhoneRemoteDataSource,
 private val sessionProducer: SessionActionProducer,
 private val countryDataSource: CountryDataSource
) : SmsDataSource {
```

```
bind<SmsDataSource>() with singleton {
 SmsDataSourceImpl(
     dataSourceOfSms = instance(),
     dataSourceOfPhone = instance(),
     sessionProducer = instance(),
     countryDataSource = instance()
 )
}
```

Целостность графа 

- Ошибки в рантайме

  -> injection

  -> test

Тесты на DI

- В Kodein нет встроенных инструментов для тестов

Тесты на DI

- В Kodein нет встроенных инструментов для тестов
- Но их обещают добавить

Тесты на DI

- В Kodein нет встроенных инструментов для тестов
- Но их обещают добавить
- UI тесты

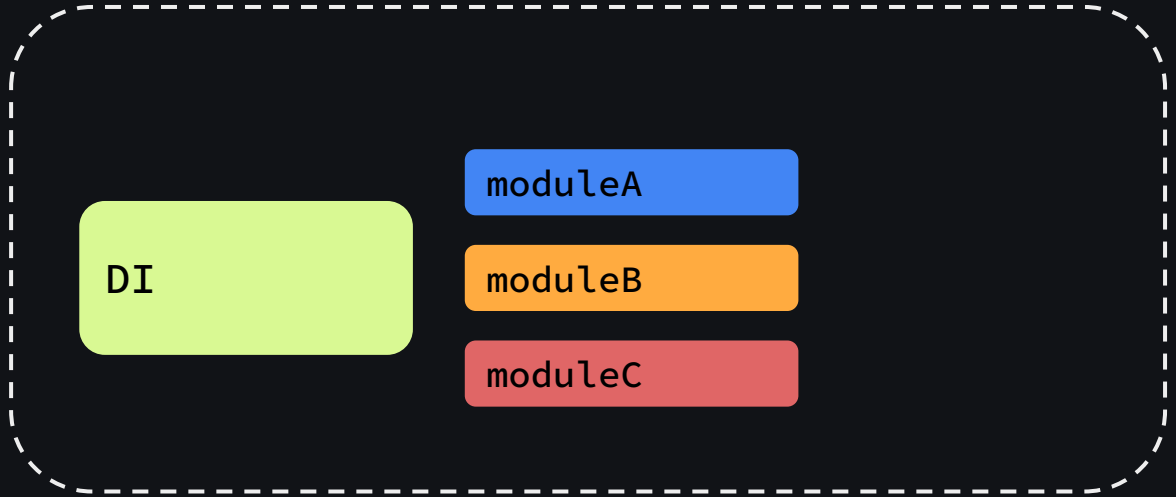
Тесты на DI

- В Kodein нет встроенных инструментов для тестов
- Но их обещают добавить
- UI тесты
- Unit тесты

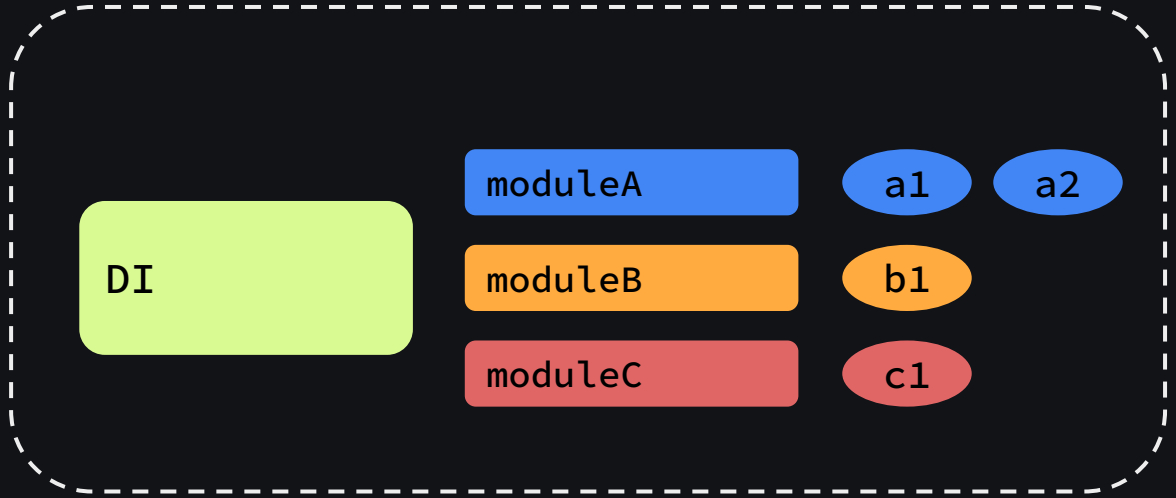
Unit тесты на DI

- Проверяем целостность DI графа (всех DI графов)
- Если у вас retrieval, лучше не надо :)

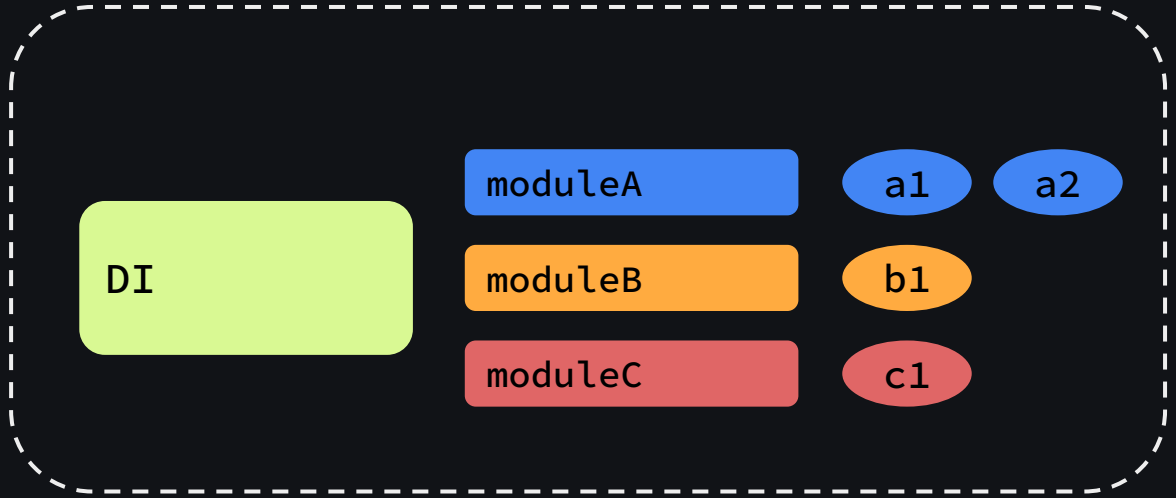
Unit тесты на DI



Unit тесты на DI

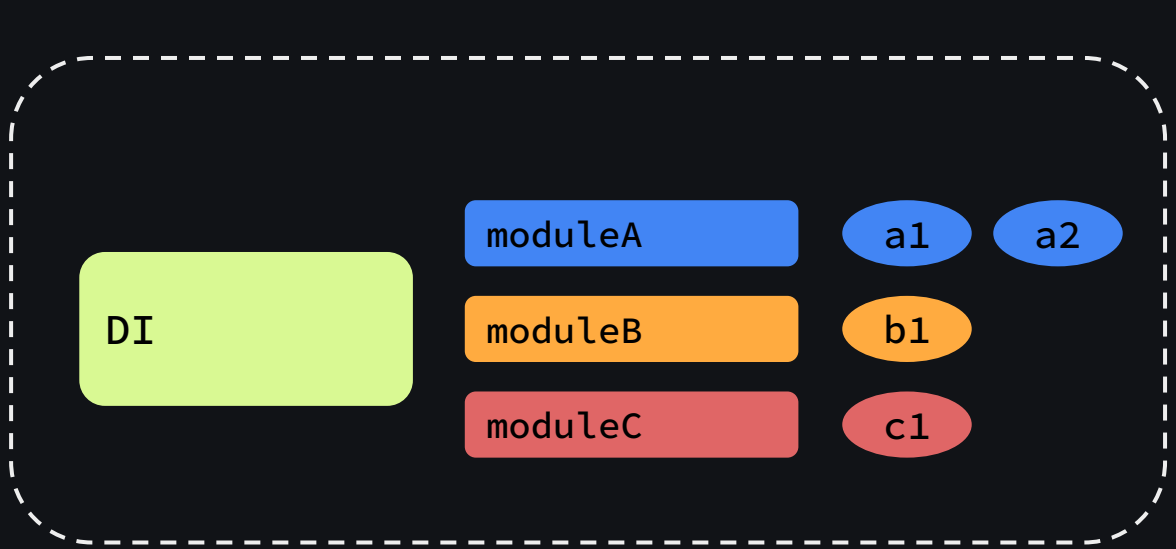


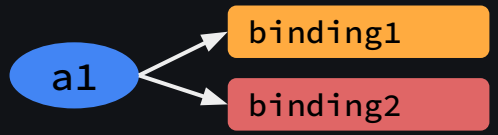
Unit тесты на DI



a1 a2 b1 c1

Unit тесты на DI

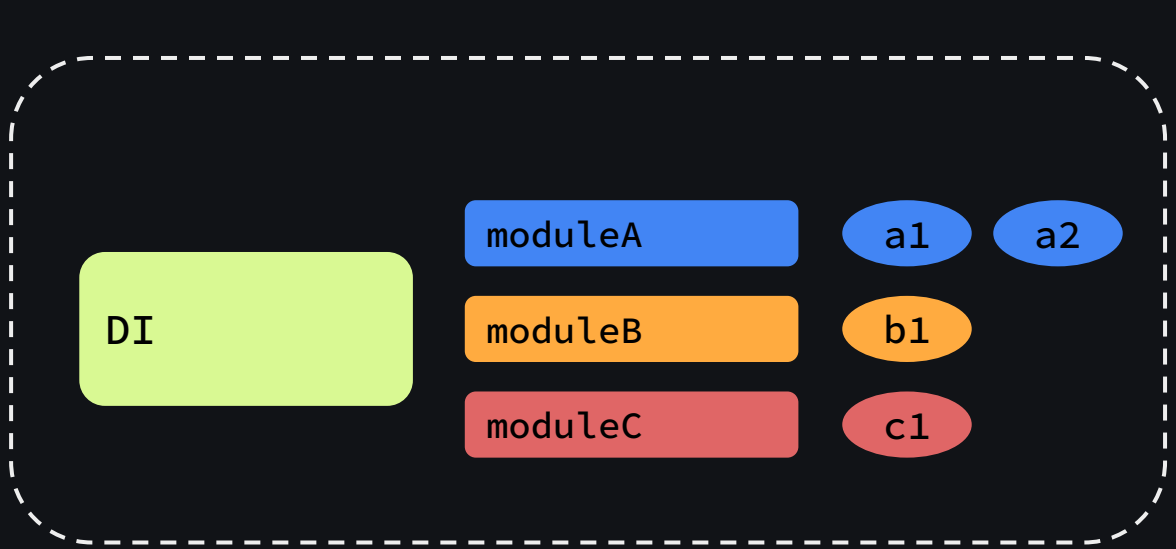


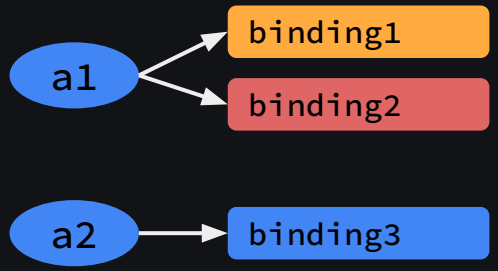


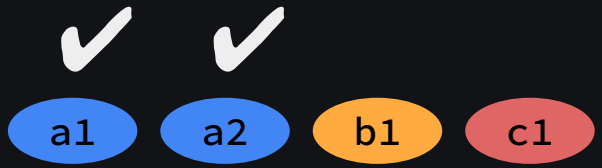
✔

a1 a2 b1 c1

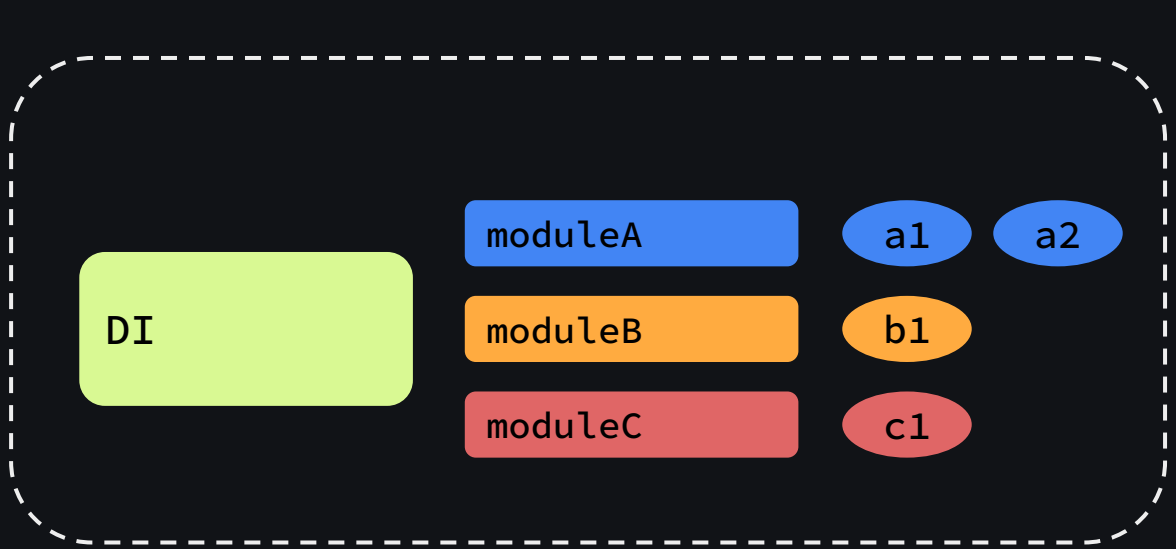
Unit тесты на DI

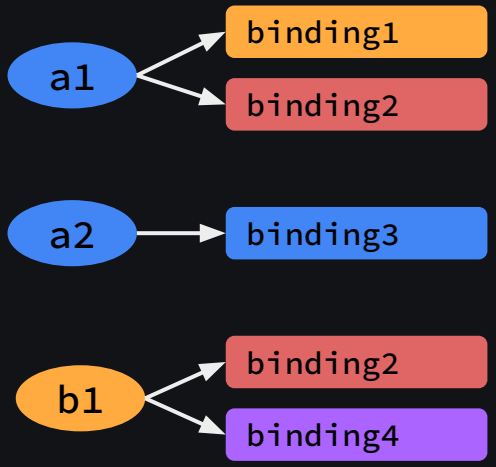


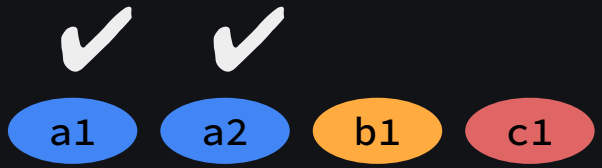




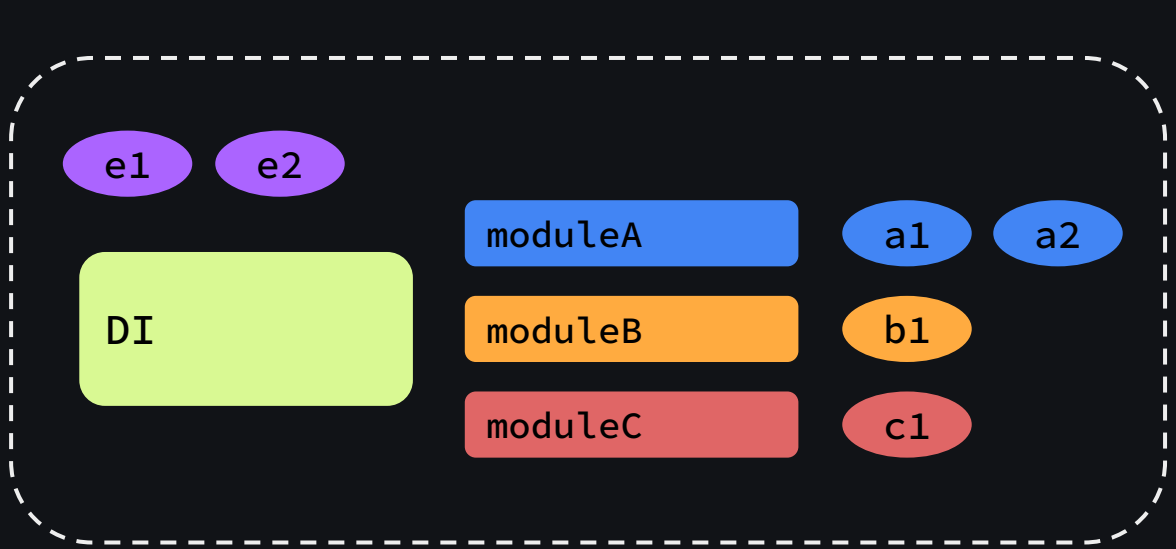
Unit тесты на DI

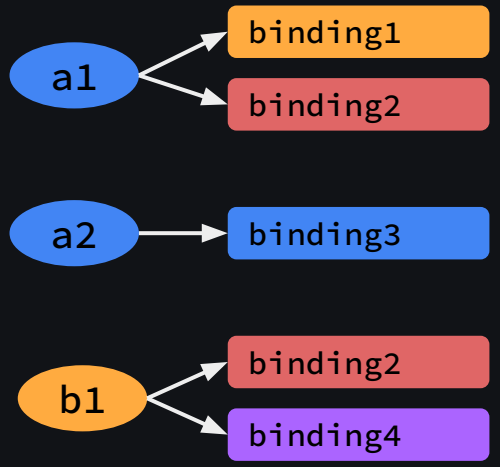


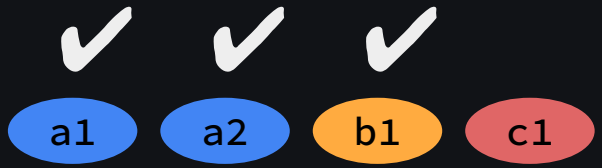




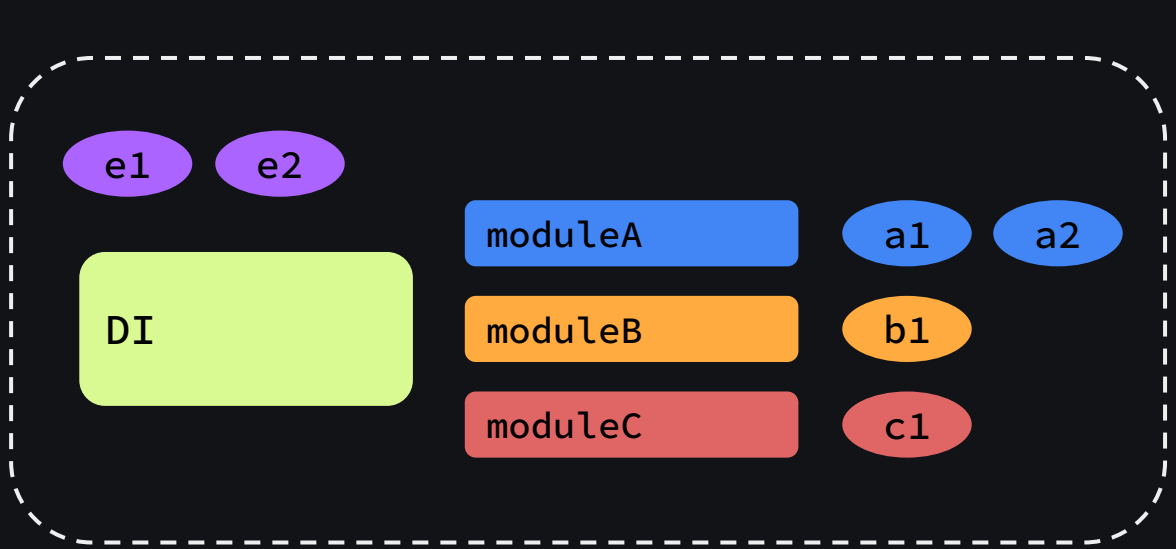
Unit тесты на DI

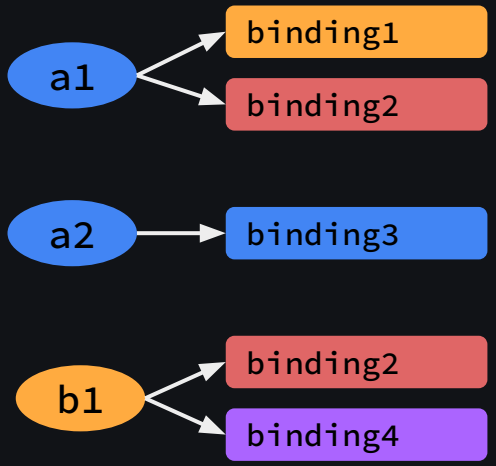






Unit тесты на DI





✔ ✔ ✔ ✔

a1 a2 b1 c1

Unit тесты на DI

```
@Test
fun `Unit Scope DI has integrity`() = withDIContext(context) {
    assertDIHasIntegrity(diSUT)
}
```

Unit тесты на DI

```
fun DITestContext.assertDIHasIntegrity(di: DI) {
    di.container.tree.bindings.forEach {
        assertContainsDependenciesFor(di, it.key.type, it.key.tag, it.key.argType)
    }
}
```

```
inline fun <reified T : Any> DITestContext.assertContainsDependenciesFor(
    di: DI,
    type: TypeToken<out T>,
    tag: Any? = null,
    argType: TypeToken<out Any?> = TypeToken.Unit,
)
```

Unit тесты на DI

```
// 1
val key = Key(
    contextType = TypeToken.Any,
    argType = argType,
    type = type,
    tag = tag,
)
```

Unit тесты на DI

```
bind<SmsDataSource>() with singleton {
    SmsDataSourceImpl(
        dataSourceOfSms = instance(),
        dataSourceOfPhone = instance(),
        sessionProducer = instance(),
        countryDataSource = instance()
    )
}
```

```
/**
* Gets an instance of `T` for the given type and tag.
* ...
*/
public inline fun <reified T : Any> DirectDIAware.instance(tag: Any? = null): T
```

Unit тесты на DI

```
class DirectDIWrapper(
    private val _directDI: DirectDI,
    private val instanceCallback: (TypeToken<*>) -> Unit,
) : DirectDI by _directDI {

    ...

    override fun <T : Any> Instance(
        type: TypeToken<T>,
        tag: Any?,
    ): T {
        instanceCallback(type)
        return _directDI.Instance(type, tag)
    }
```

Unit тесты на DI

```
// 2
val missing = mutableListOf<String>()
val directWrapper = DirectDIWrapper(direct) { clazz: TypeToken<*> ->
    val name = clazz.qualifiedDispString()
    if (!deps.contains(name)) {
        missing.add(name)
    }
}
```

Unit тесты на DI

```
// 3
val noArgBindingDI: NoArgBindingDI<Any> =
        NoArgBindingDIWrap(BindingDIImpl(directWrapper, key, 0))
noArgBindingDI.apply {
    (diBinding as? Singleton<Any, T>)?.creator?.let { it() }
    (diBinding as? Provider<Any, T>)?.creator?.let { it() }
    ...
}
```

Unit тесты на DI

```
// 4
context.removableDependencies().remove(T::class.java.canonicalName)

if (missing.isNotEmpty()) {
    println("Not all dependencies found for: ${T::class.java}. Missing: $missing")
}

return missing.isNotEmpty()
```

Unit тесты на DI

```
bind<Presenter>() with provider {
    PresenterImpl(
        mapper = instance(),
        voFactory = instance(),
        interactor = instance(),
    )
}

bind<Interactor>() with provider {
    InteractorImpl(
        repository = instance(),
    )
}

bind<Repository>() with singleton {
    RepositoryImpl(
        remoteDataSource = instance(),
        persistentDataSource = instance()
    )
}
```

Unit тесты на DI

```
bind<Presenter>() with provider {
    PresenterImpl(
        mapper = instance(),
        voFactory = instance(),
        interactor = instance(),
    )
}

bind<Interactor>() with provider {
    InteractorImpl(
        repository = instance(),
    )
}

bind<Repository>() with singleton {
    RepositoryImpl(
        remoteDataSource = instance(),
        persistentDataSource = instance()
    )
}
```

Unit тесты на DI

```
bind<Presenter>() with provider {
    PresenterImpl(
        mapper = instance(),
        voFactory = instance(),
        interactor = instance(),
    )
}

bind<Interactor>() with provider {
    InteractorImpl(
        repository = instance(),
    )
}

bind<Repository>() with singleton {
    RepositoryImpl(
        remoteDataSource = instance(),
        persistentDataSource = instance()
    )
}
```

Unit тесты на DI

```
bind<Presenter>() with provider {
    PresenterImpl(
        mapper = instance(),
        voFactory = instance(),
        interactor = instance()
    )
}

bind<Interactor>() with provider {
    InteractorImpl(
        repository = instance()
    )
}

bind<Repository>() with singleton {
    RepositoryImpl(
        remoteDataSource = instance(),
        persistentDataSource = instance()
    )
}
```

Unit тесты на DI

```
 @Before
fun setup() {
    diSUT = DI { import(UnitScopeDI(UnitContext(unit = mock(), country = mock()))) }
}

private val context = DITestContext(
    integrityRule = integrityRule,
    externalDependencies = listOf(
        SharedPreferences::class.java,
        Retrofit::class.java,
        Dispatcher::class.java,
        ...
    )
)

@Test
fun `Unit Scope DI has integrity`() = withDIContext(context) {
    assertDIHasIntegrity(diSUT)
}
```

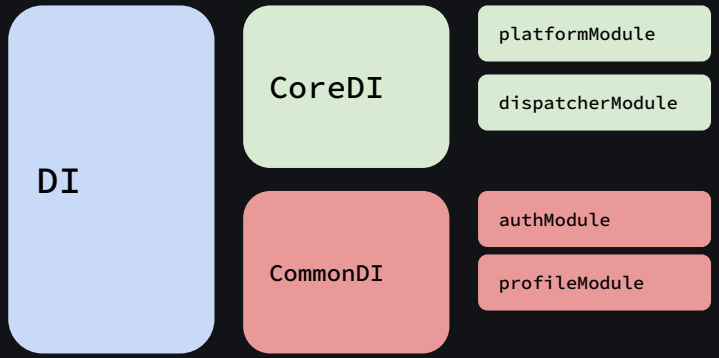
Unit тесты на DI

```
 @Before
fun setup() {
    diSUT = DI { import(UnitScopeDI(UnitContext(unit = mock(), country = mock()))) }
}

private val context = DITestContext(
    integrityRule = integrityRule,
    externalDependencies = listOf(
        SharedPreferences::class.java,
        Retrofit::class.java,
        Dispatcher::class.java,
        ...
    )
)

@Test
fun `Unit Scope DI has integrity`() = withDIContext(context) {
    assertDIHasIntegrity(diSUT)
}
```

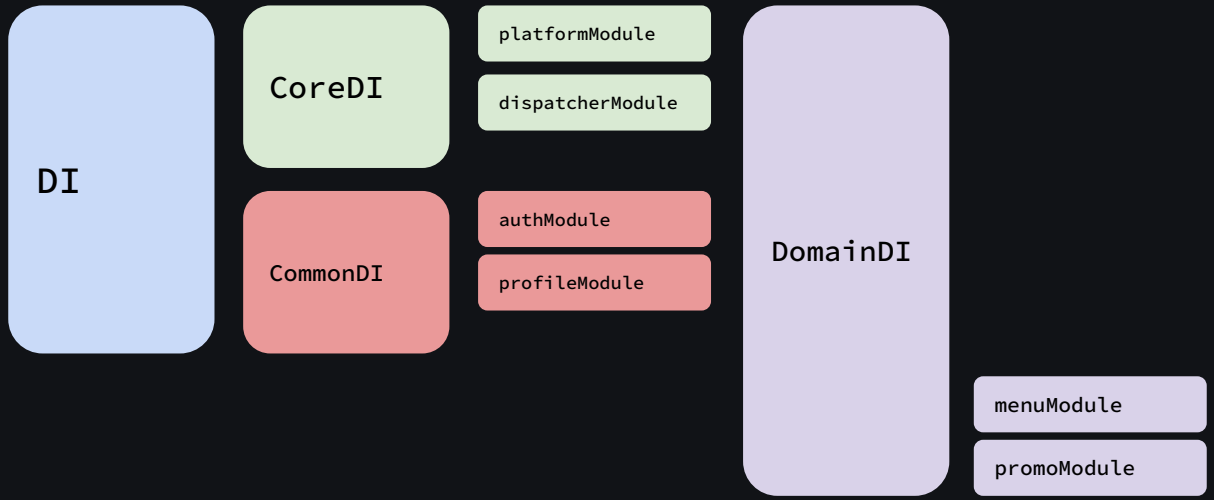
Unit тесты на DI

```
 @Before
fun setup() {
    diSUT = DI { import(UnitScopeDI(UnitContext(unit = mock(), country = mock()))) }
}

private val context = DITestContext(
    integrityRule = integrityRule,
    externalDependencies = listOf(
        SharedPreferences::class.java,
        Retrofit::class.java,
        Dispatcher::class.java,
        ...
    )
)

@Test
fun `Unit Scope DI has integrity`() = withDIContext(context) {
    assertDIHasIntegrity(diSUT)
}
```

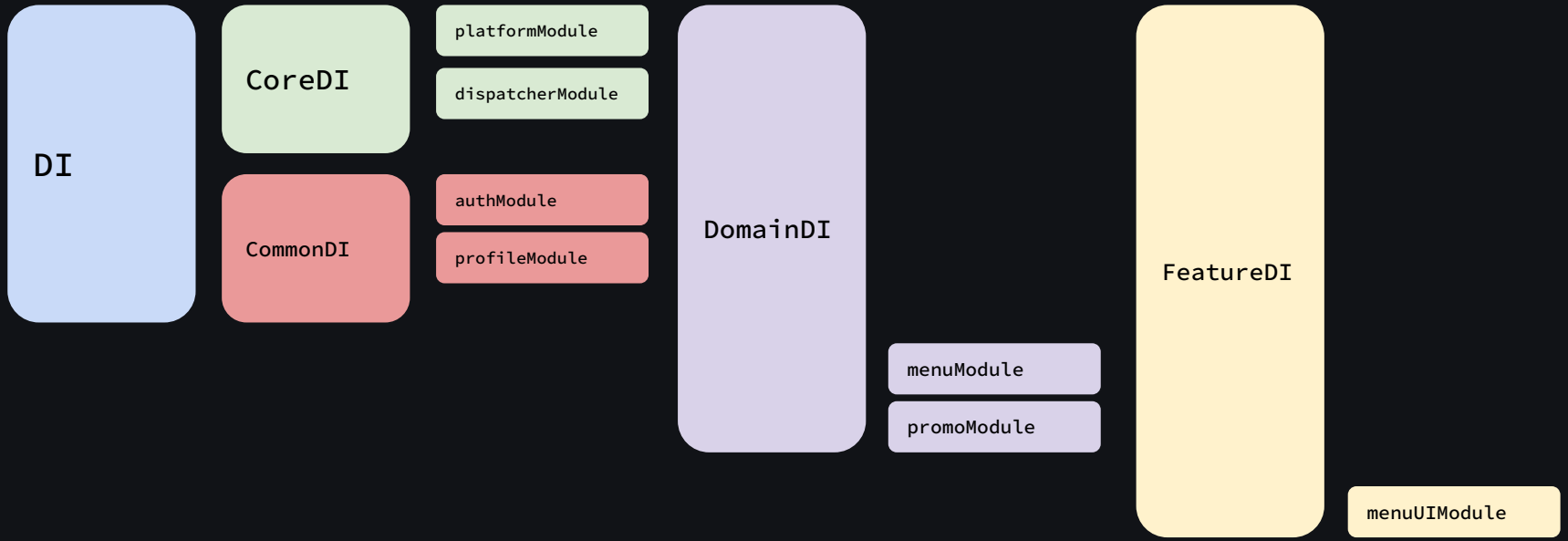
Unit тесты на DI



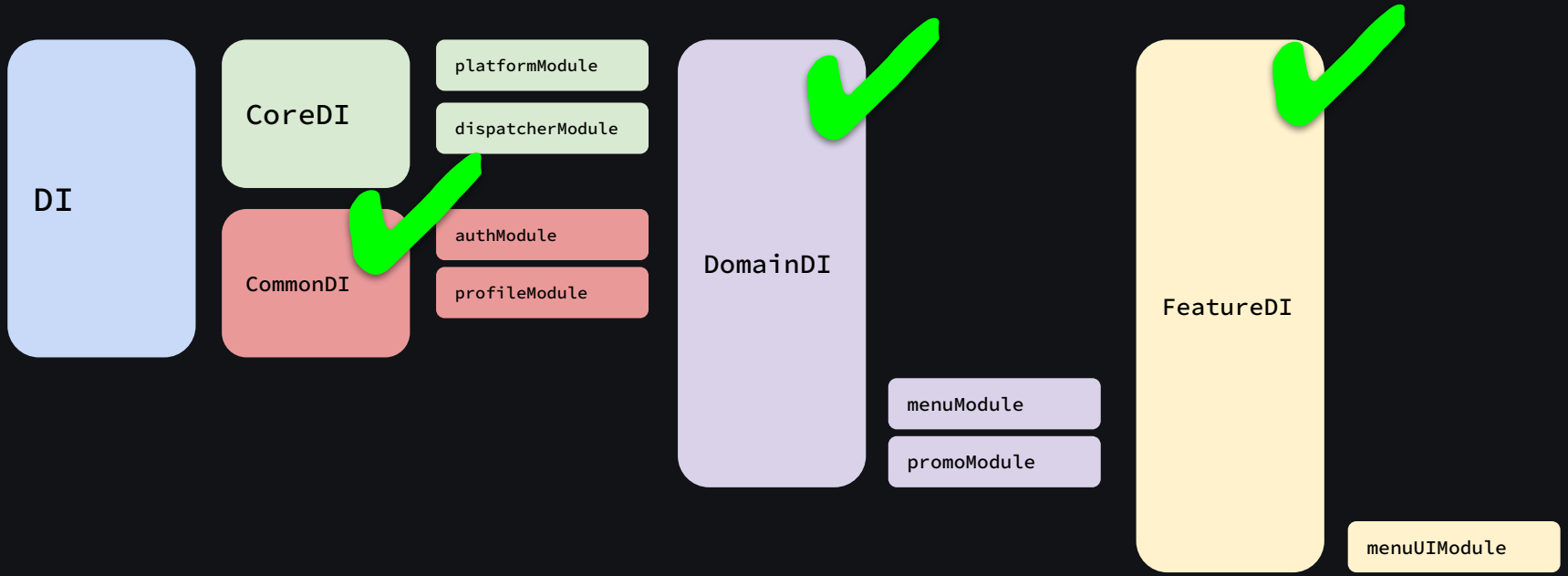
Unit тесты на DI



Unit тесты на DI



Unit тесты на DI



Целостность графа 

- Ошибки в рантайме

  -> injection

  -> test ✔

Наш опыт с Kodein

- **Целостность графа ✔**
- Retrieval vs Injection
- ConfigurableDI, Initializers
- Скоупы

Наш опыт с Kodein

- Целостность графа
- **Retrieval vs Injection**
- ConfigurableDI, Initializers
- Скоупы

Retrieval vs Injection

- Нагрузка на проверку целостности графа
- Проникновение в код

Retrieval vs Injection

- Нагрузка на проверку целостности графа
- Проникновение в код

```
class CartPaymentDataSourceImpl(di: DIAware) : CartPaymentDataSource, DIAware by di {

    private val paymentDataSource by instance<PaymentDataSource>()
    private val paymentMethodsDataSource by instance<PaymentMethodsDataSource>()
    private val paymentStatusService by instance<PaymentStatusService>()
```

Retrieval vs Injection

- Нагрузка на проверку целостности графа
- Проникновение в код

```
class CartPaymentDataSourceImpl(di: DIAware) : CartPaymentDataSource, DIAware by di {

    private val paymentDataSource by instance<PaymentDataSource>()
    private val paymentMethodsDataSource by instance<PaymentMethodsDataSource>()
    private val paymentStatusService by instance<PaymentStatusService>()
```

Retrieval vs Injection

```
override suspend fun pay(
 paymentSender: ScreenName,
 options: PaymentCardSource
) = withContext(ioContext) {
   try {
       paymentStatusService.notifyOrdering()
       val cart = cartDataSource.getCart()
       val paymentType = (paymentMethodsService as PaymentMethodsDataSourceImpl).consumeDefaultSdk()
       val promoSource = (context as DIAware).di.direct.instance<PaymentPromoDataSource>()
       val couponSource = promoSource.consumePromo()
       val couponManually = promoSource.consumePromoManually()
       val coupon = if (options.withPromo) couponSource else null
       val mode = if (couponManually) "Manual" else "Suggested"
       val paymentScreen = paymentSender.value
```

Retrieval vs Injection

```
override suspend fun pay(
 paymentSender: ScreenName,
 options: PaymentCardSource
) = withContext(ioContext) {
   try {
       paymentStatusService.notifyOrdering()
       val cart = cartDataSource.getCart()
       val paymentType = (paymentMethodsService as PaymentMethodsDataSourceImpl).consumeDefaultSdk()
       val promoSource = (context as DIAware).di.direct.instance<PaymentPromoDataSource>()
       val couponSource = promoSource.consumePromo()
       val couponManually = promoSource.consumePromoManually()
       val coupon = if (options.withPromo) couponSource else null
       val mode = if (couponManually) "Manual" else "Suggested"
       val paymentScreen = paymentSender.value
```

Наш опыт с Kodein

- Целостность графа
- **Retrieval vs Injection ✔**
- ConfigurableDI, Initializers
- Скоупы

Наш опыт с Kodein

- Целостность графа
- Retrieval vs Injection
- **ConfigurableDI, Initializers**
- Скоупы

ConfigurableDI

- Конфигурировать постепенно

ConfigurableDI

- Конфигурировать постепенно

```
internal object AppScopeDI {
   operator fun invoke(app: Application): ConfigurableDI = ConfigurableDI(mutable = true).apply {
       addImport(moduleOfScopeManager(app))
       addImport(androidXModule(app))
       addImport(moduleOfCore())
       addImport(moduleOfCart(app))
       addImport(moduleOfAnalytics())
       addImport(moduleOfPayment())
       addImport(moduleOfFirebase())
       ...
```

ConfigurableDI

- Конфигурировать постепенно

```
((context as DIAware).di as ConfigurableDI).addConfig {
    bind<ProductEndpoint>() with singleton {...}
    bind<ProductDataSource>() with singleton {...}
    bind<ProductCustomizationService>() with singleton {...}
    bind<ProductStopsDataSource>() with singleton {...}
    bind<CustomizationStreamService>() with singleton {...}
}
```

ConfigurableDI

- Конфигурировать постепенно
- До первого retrieval всё хорошо

ConfigurableDI

- Конфигурировать постепенно
- До первого retrieval всё хорошо
- Если менять после первого retrieval - сброс всех кешей, удар по производительности

Initializers

Initializers

- androidx.startup

Initializers

- androidx.startup
- работает на Content Provider

Initializers

- androidx.startup
- работает на Content Provider

```
<application>
 <provider
     android:name="androidx.startup.InitializationProvider"
     android:authorities="${applicationId}.androidx-startup"
     android:exported="false"
     tools:node="merge">
   <meta-data
       android:name="ru.drinkit.order.domain.OrderInitializer"
       android:value="androidx.startup" />
 </provider>
</application>
```

Initializers

- androidx.startup
- работает на Content Provider

```
internal class OrderInitializer : Initializer<Unit> {
    override fun create(context: Context) {
        val module = DI.Module("orderDomainModule") {...}
        bind<OrderEndpoint>() with singleton {...}
        bind<OrderDataSource>() with singleton {...}
        ((context as DIAware).di as ConfigurableDI).addImport(module)
    }
```

Наш опыт с Kodein

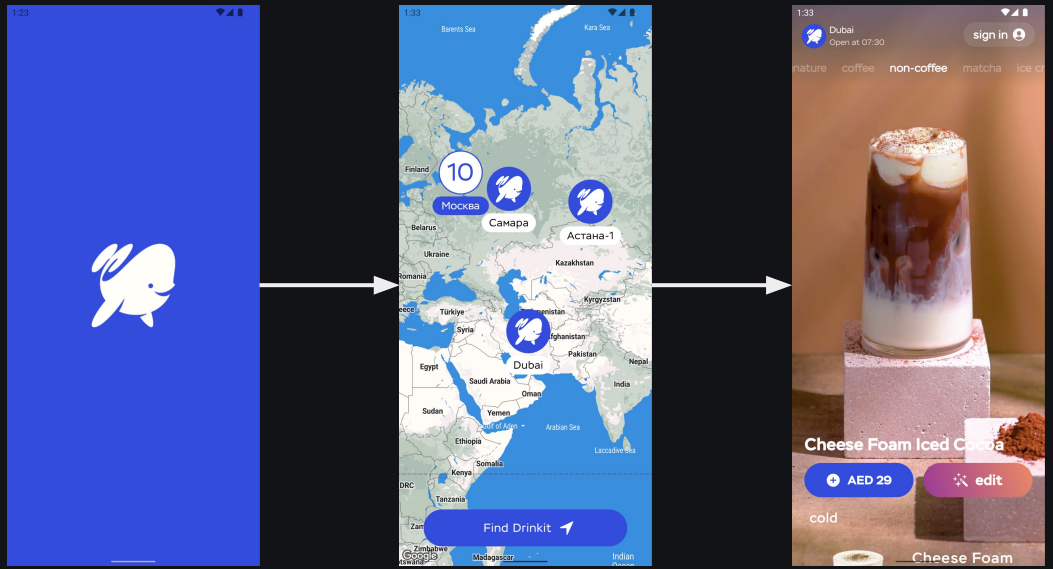
- Целостность графа
- Retrieval vs Injection
- **ConfigurableDI, Initializers ✔**
- Скоупы

Наш опыт с Kodein

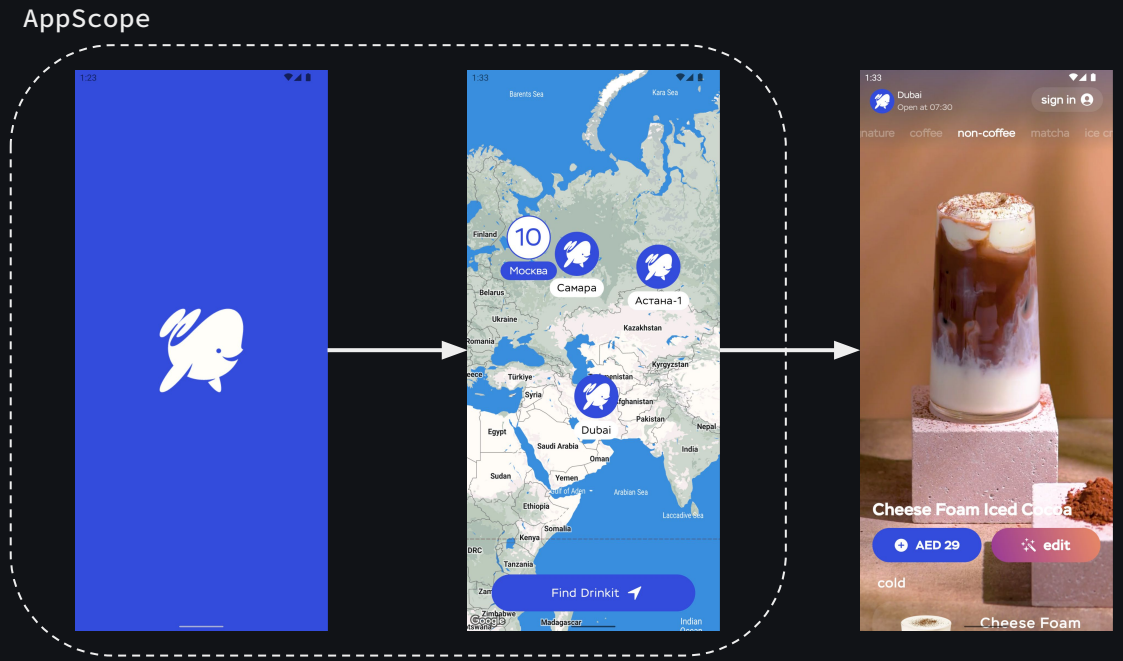
- Целостность графа
- Retrieval vs Injection
- ConfigurableDI, Initializers
- **Скоупы**

Скоупы

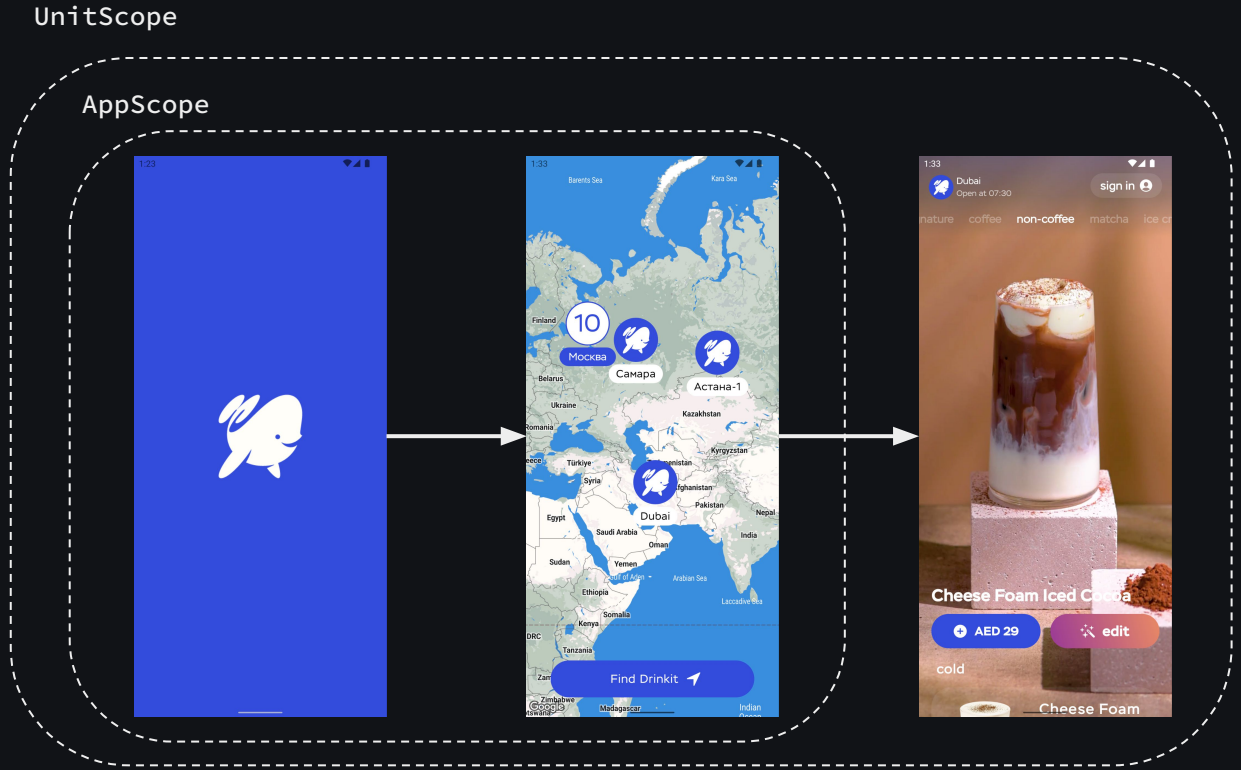
Скоупы


10
Москва
Самара
Астана-1
Dubai
Find Drinkit
Dubai
Open at 07:30
sign in
nature
coffee
non-coffee
matcha
Cheese Foam Iced Cocoa
AED 29
edit
cold
Cheese Foam

Скоупы


AppScope
10
Москва
Самара
Астана-1
Dubai
Find Drinkit
Dubai
Open at 07:30
sign in
nature
coffee
non-coffee
matcha
Cheese Foam Iced Cocoa
AED 29
edit
cold
Cheese Foam

Скоупы


UnitScope
AppScope
10
Москва
Самара
Астана-1
Dubai
Find Drinkit
Dubai
Open at 07:30
sign in
nature
coffee
non-coffee
matcha
Cheese Foam Iced Cocoa
AED 29
edit
cold
Cheese Foam

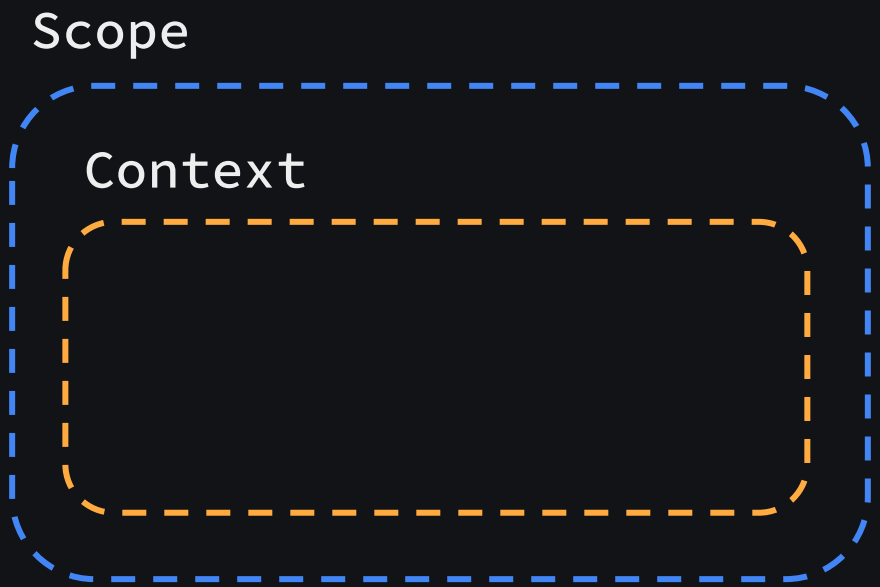
Скоупы

- Скоупы от Kodein
- Скоупы ручками

Скоупы

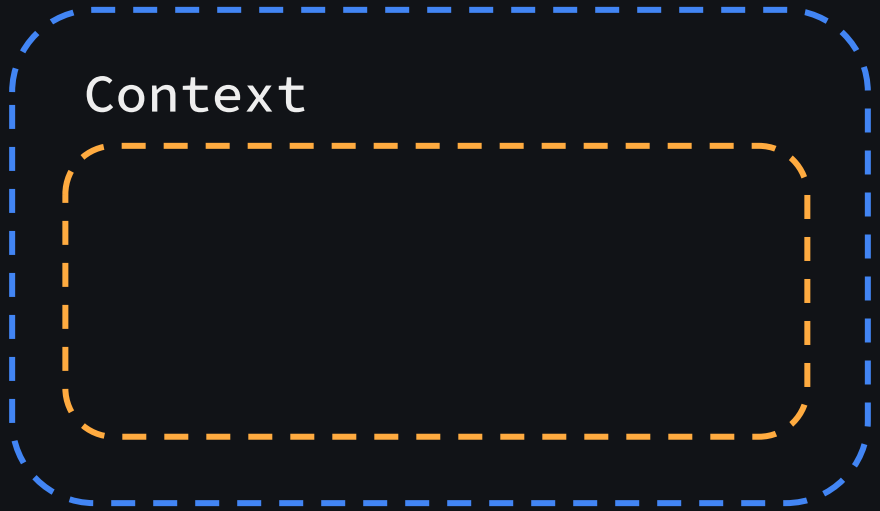
- Скоупы от Kodein
- Скоупы ручками

Скоупы от Kodein



Скоупы от Kodein

```
public interface Scope<in C>
```



Скоупы от Kodein

```
public interface Scope<in C> {

    /**
     * Get a registry for a given context.
     * Should always return the same registry for the same context.
     *
     * @param context The context associated with the returned registry.
     * @return The registry associated with the given context.
     */
    public fun getRegistry(context: C): ScopeRegistry
}
```

Скоупы от Kodein

```
public open class WeakContextScope<in C>(
    public val newRepo: () -> ScopeRegistry = { StandardScopeRegistry() }
) : Scope<C> {

   private val map = WeakHashMap<C, ScopeRegistry>()

   override fun getRegistry(context: C): ScopeRegistry {
       map[context]?.let { return it }
       synchronized(map) {
           return map.getOrPut(context) { newRepo() }
       }
   }

   public companion object Of : WeakContextScope<Any>() {
       @Suppress("UNCHECKED_CAST")
       public fun <T> of(): Scope<T> = this as Scope<T>
   }
}
```

Скоупы от Kodein

```
val customerScope: Scope<CustomerContext> = WeakContextScope.of()
```

Скоупы от Kodein

```
class UpdateCustomerScopeUseCaseImpl(
    private val scopeContextManager: ScopeContextManager,
) : UpdateCustomerScopeUseCase {

    override suspend fun invoke(profile: Profile) {
        when (profile) {
            is Authorized -> scopeContextManager.createNewCustomerScope(
                CustomerContext(isLoggedIn = true, profileName = profile.name)
            )
            NonAuthorized -> scopeContextManager.createNewCustomerScope(
                CustomerContext(isLoggedIn = false, profileName = "")
            )
        }
    }
}
```

Скоупы от Kodein

```
class UpdateCustomerScopeUseCaseImpl(
    private val scopeContextManager: ScopeContextManager,
) : UpdateCustomerScopeUseCase {

    override suspend fun invoke(profile: Profile) {
        when (profile) {
            is Authorized -> scopeContextManager.createNewCustomerScope(
                CustomerContext(isLoggedIn = true, profileName = profile.name)
            )
            NonAuthorized -> scopeContextManager.createNewCustomerScope(
                CustomerContext(isLoggedIn = false, profileName = "")
            )
        }
    }
}
```

Скоупы от Kodein

```
fun createDIRepositoryModule(): DI.Module = DI.Module("DIRepositoryModule") {
   bind<PortfolioRepository>() with scoped(customerScope).singleton {
       PortfolioRepositoryImpl(
           userNameProvider = { context.profileName }
       )
   }
}
```

Скоупы от Kodein

```
fun createDIRepositoryModule(): DI.Module = DI.Module("DIRepositoryModule") {
    bind<PortfolioRepository>() with scoped(customerScope).singleton {
        PortfolioRepositoryImpl(
            userNameProvider = { context.profileName }
        )
    }
}
```

Скоупы от Kodein

```
fun createPortfolioDomainModule(): DI.Module =
    DI.Module("PortfolioDomainModule") {
        bindProvider<GetPortfolioUseCase> {
            GetPortfolioUseCaseImpl(
                portfolioRepository = on(customerContext).instance()




        }
    }
```

Скоупы от Kodein

```
fun createPortfolioDomainModule(): DI.Module =
    DI.Module("PortfolioDomainModule") {
        bindProvider<GetPortfolioUseCase> {
            GetPortfolioUseCaseImpl(
                portfolioRepository = on(customerContext).instance()
                portfolioRepository = on(
                    instance<ScopeContextManager>().run { customerContext }).instance()
                )

        }
    }
```

Скоупы от Kodein

```
fun createPortfolioDomainModule(): DI.Module =
    DI.Module("PortfolioDomainModule") {
        bindProvider<GetPortfolioUseCase> {
            GetPortfolioUseCaseImpl(
                portfolioRepository = on(customerContext).instance()
                portfolioRepository = on(
                    instance<ScopeContextManager>().run { customerContext }).instance()
                )
                portfolioRepository = withContextOf { customerContext }.instance()
        }
    }
```

Скоупы от Kodein 

```
bind<MenuCommunication>() with scoped(unit.scope).singleton { ... }
bind<MenuScrollDataSource>() with scoped(unit.scope).singleton { ... }
bind<MenuRefreshDataSource>() with scoped(unit.scope).singleton { ... }
bind<MenuMapper>() with scoped(unit.scope).singleton(sync = false) { ... }
bind<MenuViewMapper>() with scoped(unit.scope).singleton(sync = false) {...}
bind<SuggestedUnitMapper>() with scoped(unit.scope).singleton(sync = false) {...}
bind<PromoMapper>() with scoped(unit.scope).singleton(sync = false) {...}
```

Скоупы от Kodein 

```
bind<MenuCommunication>() with scoped(unit.scope).singleton { ... }
bind<MenuScrollDataSource>() with scoped(unit.scope).singleton { ... }
bind<MenuRefreshDataSource>() with scoped(unit.scope).singleton { ... }
bind<MenuMapper>() with scoped(unit.scope).singleton(sync = false) { ... }
bind<MenuViewMapper>() with scoped(unit.scope).singleton(sync = false) {...}
bind<SuggestedUnitMapper>() with scoped(unit.scope).singleton(sync = false) {...}
bind<PromoMapper>() with scoped(unit.scope).singleton(sync = false) {...}
```

Скоупы от Kodein

```
bind<ProductPresenter>() with scoped(WeakContextScope.of<Activity>()).provider {
    ProductPresenterImpl(
        productInteractor = instance(),
        menuService = instance(),
        moneyFormatter = instance(),
        dispatcher = instance(),
    )
}
```

Скоупы от Kodein

```
bind<ProductPresenter>() with scoped(WeakContextScope.of<Activity>()).provider {
    ProductPresenterImpl(
        productInteractor = instance(),
        menuService = instance(),
        moneyFormatter = instance(),
        dispatcher = instance(),
    )
}


class ProductFragment : ... {
    ...
    private val viewPresenter by instance<ProductPresenter>()
```

Скоупы от Kodein 

- ActivityRetainedScope
  - Fragment, retainInstance = true

```
/**
* A scope that allows to get an activity-scoped singleton that's independent from the
activity restart.
*/
open class ActivityRetainedScope private constructor(
    private val registryType: RegistryType
) : Scope<Activity> {
   ...
```

Скоупы от Kodein

```
bind<ProductPresenter>() with scoped(ActivityRetainedScope).provider {
    ProductPresenterImpl(
        productInteractor = instance(),
        menuService = instance(),
        moneyFormatter = instance(),
        dispatcher = instance(),
    )
}

class ProductFragment : ... {
    ...
    private val viewPresenter by instance<ProductPresenter>()
```

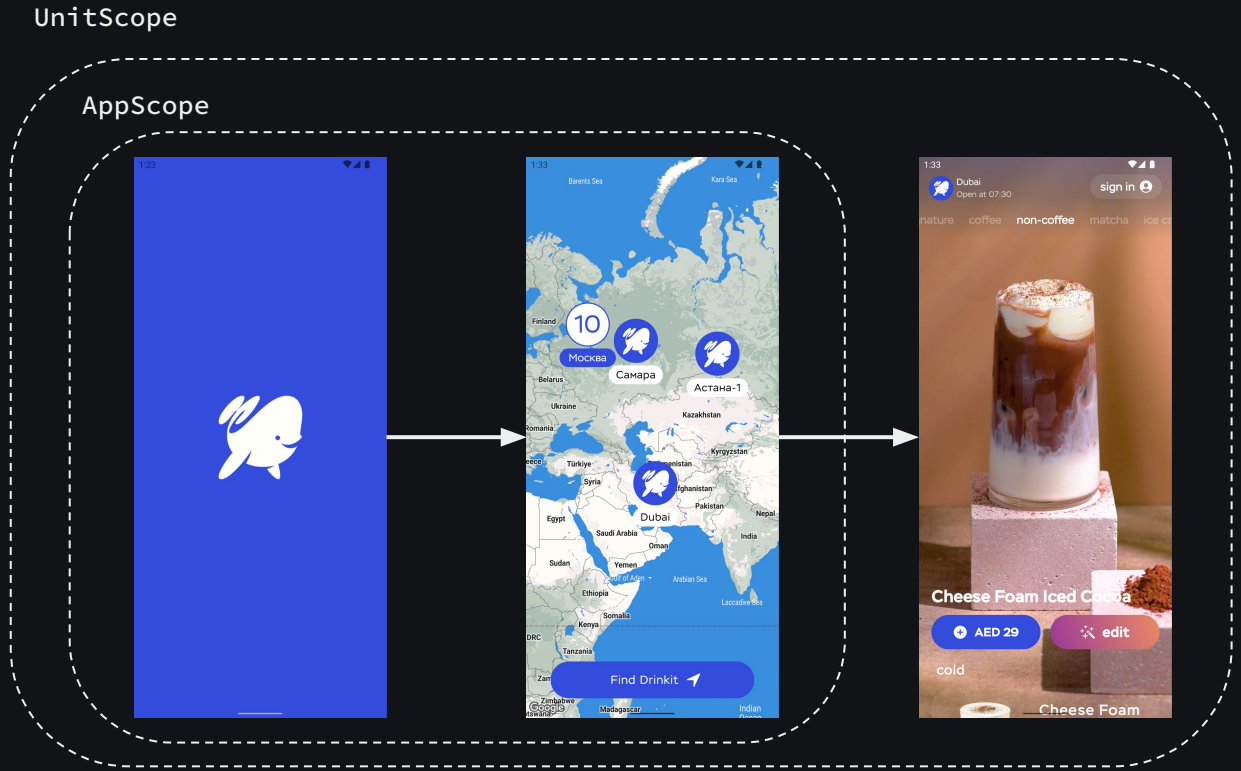
Скоупы

- **Скоупы от Kodein ✔**
- Скоупы ручками

Скоупы

- Скоупы от Kodein
- **Скоупы ручками**
  - **subDI**
  - **контроллировать время жизни самостоятельно**

Скоупы ручками



Скоупы ручками

```
interface DIScope<T : BaseContext> {
 val scopeContext: T
 val di: DI
}


class AppScopeImpl(
    override val scopeContext: AppContext,
) : AppScope {
    override val di = AppScopeDI(scopeContext.application)
}


class UnitScopeImpl(
    private val parentDIAware: DIAware,
    override val scopeContext: UnitContext,
) : UnitScope {
    override val di = subDI(parentDIAware.di) {
        import(UnitScopeDI(scopeContext))
    }
}
```

Скоупы ручками

```
interface DIScope<T : BaseContext> {
 val scopeContext: T
 val di: DI
}


class AppScopeImpl(
    override val scopeContext: AppContext,
) : AppScope {
    override val di = AppScopeDI(scopeContext.application)
}


class UnitScopeImpl(
    private val parentDIAware: DIAware,
    override val scopeContext: UnitContext,
) : UnitScope {
    override val di = subDI(parentDIAware.di) {
        import(UnitScopeDI(scopeContext))
    }
}
```

Скоупы ручками

```
interface DIScope<T : BaseContext> {
 val scopeContext: T
 val di: DI
}


class AppScopeImpl(
    override val scopeContext: AppContext,
) : AppScope {
    override val di = AppScopeDI(scopeContext.application)
}


class UnitScopeImpl(
    private val parentDIAware: DIAware,
    override val scopeContext: UnitContext,
) : UnitScope {
    override val di = subDI(parentDIAware.di) {
        import(UnitScopeDI(scopeContext))
    }
}
```

Скоупы ручками

```
interface DIScope<T : BaseContext> {
 val scopeContext: T
 val di: DI
}


class AppScopeImpl(
    override val scopeContext: AppContext,
) : AppScope {
    override val di = AppScopeDI(scopeContext.application)
}


class UnitScopeImpl(
    private val parentDIAware: DIAware,
    override val scopeContext: UnitContext,
) : UnitScope {
    override val di = subDI(parentDIAware.di) {
        import(UnitScopeDI(scopeContext))
    }
}
```

Скоупы ручками

```
object AppScopeDI {
    operator fun invoke(app: Application) = DI {
        import(moduleOfScopeManager(app))
        import(androidXModule(app))
        import(moduleOfCore())
        import(moduleOfCart(app))
        import(moduleOfAnalytics())
        ...
```

```
object UnitScopeDI {
    operator fun invoke(unitContext: UnitContext): DI.Module = DI.Module(name = "UnitScopeDI") {
        bindInstance { unitContext.unit }
        bindInstance { unitContext.country }
        import(commonMenuModule())
        import(moduleOfMenu())
        import(productModule())
        ...
    }
}
```

Скоупы ручками

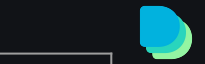
```
object AppScopeDI {
    operator fun invoke(app: Application) = DI {
        import(moduleOfScopeManager(app))
        import(androidXModule(app))
        import(moduleOfCore())
        import(moduleOfCart(app))
        import(moduleOfAnalytics())
        ...
```

```
object UnitScopeDI {
    operator fun invoke(unitContext: UnitContext): DI.Module = DI.Module(name = "UnitScopeDI") {
        bindInstance { unitContext.unit }
        bindInstance { unitContext.country }
        import(commonMenuModule())
        import(moduleOfMenu())
        import(productModule())
        ...
    }
}
```

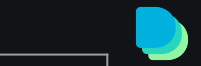
Наш опыт с Kodein

- Целостность графа
- Retrieval vs Injection
- ConfigurableDI, Initializers
- **Скоупы ✔**

Kodein vs Dagger 2 vs Koin



	Dagger 2	Koin	Kodein
Graph safety			
Build performance			
Runtime performance			
Android			
Compose			
KMM			
learning curve			
JSR-330			
Error messages			
Community			

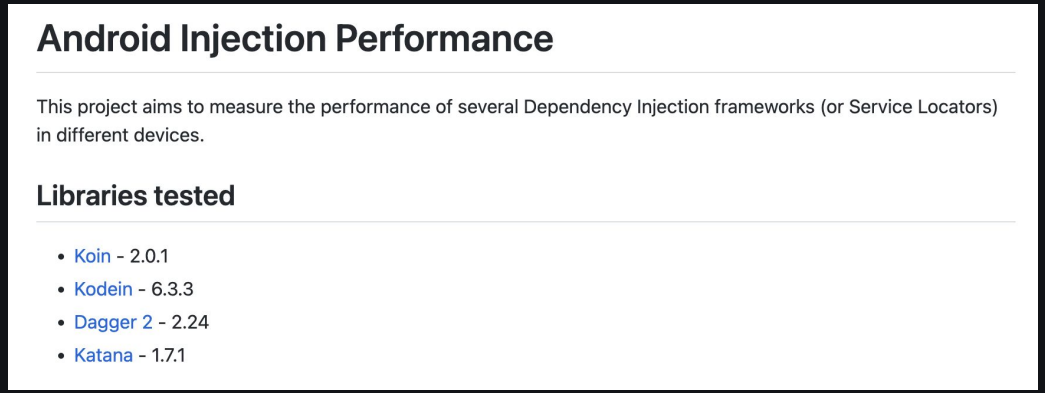


	Dagger 2	Koin	Kodein
Graph safety	Compile time	Deploy time	Run time
Build performance			
Runtime performance			
Android			
Compose			
KMM			
learning curve			
JSR-330			
Error messages			
Community			

	Dagger 2	Koin	Kodein
Graph safety	Compile time	Deploy time	Run time
Build performance	`!= 0`	`0`	`0`
Runtime performance			
Android			
Compose			
KMM			
learning curve			
JSR-330			
Error messages			
Community			

Runtime performance

https://github.com/Sloy/android-dependency-injection-performance



Android Injection Performance

This project aims to measure the performance of several Dependency Injection frameworks (or Service Locators) in different devices.

Libraries tested

- Koin - 2.0.1
- Kodein - 6.3.3
- Dagger 2 - 2.24
- Katana - 1.7.1

Runtime performance

Samsung Galaxy J5

Samsung j5nlte with Android 6.0.1

Library	Setup Kotlin	Setup Java	Inject Kotlin	Inject Java
Koin	51.47 ms	53.65 ms	2.47 ms	2.52 ms
Kodein	73.36 ms	75.21 ms	9.89 ms	9.58 ms
Katana	12.34 ms	12.30 ms	2.00 ms	1.94 ms
Custom	4.85 ms	4.81 ms	0.73 ms	0.84 ms
Dagger	0.02 ms	0.02 ms	0.27 ms	0.23 ms

Samsung Galaxy S8

Samsung dreamlte with Android 8.0.0

Library	Setup Kotlin	Setup Java	Inject Kotlin	Inject Java
Koin	5.68 ms	6.04 ms	0.13 ms	0.21 ms
Kodein	7.13 ms	7.38 ms	0.20 ms	0.21 ms
Katana	0.64 ms	0.68 ms	0.21 ms	0.16 ms
Custom	0.15 ms	0.16 ms	0.11 ms	0.11 ms
Dagger	0.01 ms	0.01 ms	0.10 ms	0.10 ms

Runtime performance

Samsung Galaxy J5

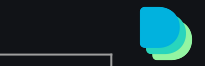
Samsung j5nlte with Android 6.0.1

Library	Setup Kotlin	Setup Java	Inject Kotlin	Inject Java
Koin	51.47 ms	53.65 ms	2.47 ms	2.52 ms
Kodein	73.36 ms	75.21 ms	9.89 ms	9.58 ms
Katana	12.34 ms	12.30 ms	2.00 ms	1.94 ms
Custom	4.85 ms	4.81 ms	0.73 ms	0.84 ms
Dagger	0.02 ms	0.02 ms	0.27 ms	0.23 ms

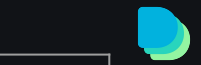
Samsung Galaxy S8

Samsung dreamlte with Android 8.0.0

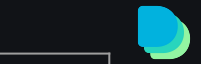
Library	Setup Kotlin	Setup Java	Inject Kotlin	Inject Java
Koin	5.68 ms	6.04 ms	0.13 ms	0.21 ms
Kodein	7.13 ms	7.38 ms	0.20 ms	0.21 ms
Katana	0.64 ms	0.68 ms	0.21 ms	0.16 ms
Custom	0.15 ms	0.16 ms	0.11 ms	0.11 ms
Dagger	0.01 ms	0.01 ms	0.10 ms	0.10 ms



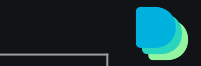
	Dagger 2	Koin	Kodein
Graph safety	Compile time	Deploy time	Run time
Build performance	`!= 0`	`0`	`0`
Runtime performance	`~0`	`0.2-2 ms`	`Koin x 2-3`
Android			
Compose			
KMM			
Learning curve			
JSR-330			
Error messages			
Community			



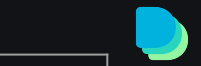
	Dagger 2	Koin	Kodein
Graph safety	Compile time	Deploy time	Run time
Build performance	`!= 0`	`0`	`0`
Runtime performance	`~0`	`0.2-2 ms`	`Koin x 2-3`
Android	✔* (Hilt)	✔	✔
Compose			
KMM			
Learning curve			
JSR-330			
Error messages			
Community			



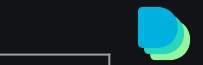
	Dagger 2	Koin	Kodein
Graph safety	Compile time	Deploy time	Run time
Build performance	`!= 0`	`0`	`0`
Runtime performance	`~0`	`0.2-2 ms`	`Koin x 2-3`
Android	✔* (Hilt)	✔	✔
Compose	—	✔	✔
KMM			
Learning curve			
JSR-330			
Error messages			
Community			



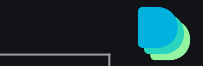
	Dagger 2	Koin	Kodein
Graph safety	Compile time	Deploy time	Run time
Build performance	`!= 0`	`0`	`0`
Runtime performance	`~0`	`0.2-2 ms`	`Koin x 2-3`
Android	✔* (Hilt)	✔	✔
Compose	—	✔	✔
KMM	—	✔	✔
Learning curve			
JSR-330			
Error messages			
Community			



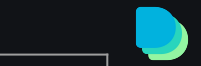
	Dagger 2	Koin	Kodein
Graph safety	Compile time	Deploy time	Run time
Build performance	`!= 0`	`0`	`0`
Runtime performance	`~0`	`0.2-2 ms`	`Koin x 2-3`
Android	✔* `(Hilt)`	✔	✔
Compose	—	✔	✔
KMM	—	✔	✔
Learning curve	Hard	Easy	Easy
JSR-330			
Error messages			
Community			



	Dagger 2	Koin	Kodein
Graph safety	Compile time	Deploy time	Run time
Build performance	`!= 0`	`0`	`0`
Runtime performance	`~0`	`0.2-2 ms`	`Koin x 2-3`
Android	✔* `(Hilt)`	✔	✔
Compose	—	✔	✔
KMM	—	✔	✔
Learning curve	Hard	Easy	Easy
JSR-330	✔	—	✔
Error messages			
Community			



	Dagger 2	Koin	Kodein
Graph safety	Compile time	Deploy time	Run time
Build performance	`!= 0`	`0`	`0`
Runtime performance	`~0`	`0.2-2 ms`	`Koin x 2-3`
Android	✔* `(Hilt)`	✔	✔
Compose	—	✔	✔
KMM	—	✔	✔
Learning curve	Hard	Easy	Easy
JSR-330	✔	—	✔
Error messages	🤯	🧡	🧡
Community			



	Dagger 2	Koin	Kodein
Graph safety	Compile time	Deploy time	Run time
Build performance	`!= 0`	`0`	`0`
Runtime performance	`~0`	`0.2-2 ms`	`Koin x 2-3`
Android	✔* `(Hilt)`	✔	✔
Compose	—	✔	✔
KMM	—	✔	✔
Learning curve	Hard	Easy	Easy
JSR-330	✔	—	✔
Error messages	🤯	🧡	🧡
Community	⭐17k	⭐7.9k	⭐2.9k

Стоил ли переходить на Kodein?

- Новый проект, новый модуль — да, пробуйте
- Большой проект на Dagger — нет
- Не пишете / не планируете тесты — нет
- Планируете KMM — да

Kodein. Выводы.

- Стабильный production ready инструмент
- Богатый API
- Легко дебажить
- Kotlin (KMM и Compose)
- Практически не хуже чем Koin
- Главная проблема - нет deploy safety

Вопросы

Максим Качинкин
Android Tech Lead
Dodo Engineering

Telegram: @maxkachinkin

Мобильное чтиво
@mobilefiction