Яндекс

Как мы делаем Яндекс.Карты: DI

Денис Загаевский, Android developer

https://github.com/zagayevskiy/android-multimodule-di-example

О чём будем говорить?

О чём будем говорить?

〉 **Модуляризация**

О чём будем говорить?

〉 Модуляризация

〉 **Требования к межмодульному DI**

О чём будем говорить?

- Модуляризация
- Требования к межмодульному DI
- **Получение зависимостей**

Where is my deps, Billy?

I need my deps

О чём будем говорить?

- Модуляризация
- Требования к межмодульному DI
- Получение зависимостей
- **Предоставление зависимостей**

Dagger…

О чём будем говорить?

- Модуляризация
- Требования к межмодульному DI
- Получение зависимостей
- Предоставление зависимостей
- **Использование зависимостей**

Dagger…

Dagger Everywhere

О чём будем говорить?

- Модуляризация
- Требования к межмодульному DI
- Получение зависимостей
- Предоставление зависимостей
- Использование зависимостей
- **Минусы / ограничения / исключения**

Модуляризация

Зачем?

Зачем?

〉 **Ускорение разработки**

Зачем?

〉 Ускорение разработки

〉 **Поставка фичей в другие приложения**

Зачем?

〉 Ускорение разработки

〉 Поставка фичей в другие приложения

〉 **Уменьшение связанности**

Зачем?

- Ускорение разработки
- Поставка фичей в другие приложения
- Уменьшение связанности

Ребята

Давайте жить лучше

Типы модулей

- App modules
- Feature modules
- Core modules

Типы модулей

〉App modules


:yandexmaps


:search:sample

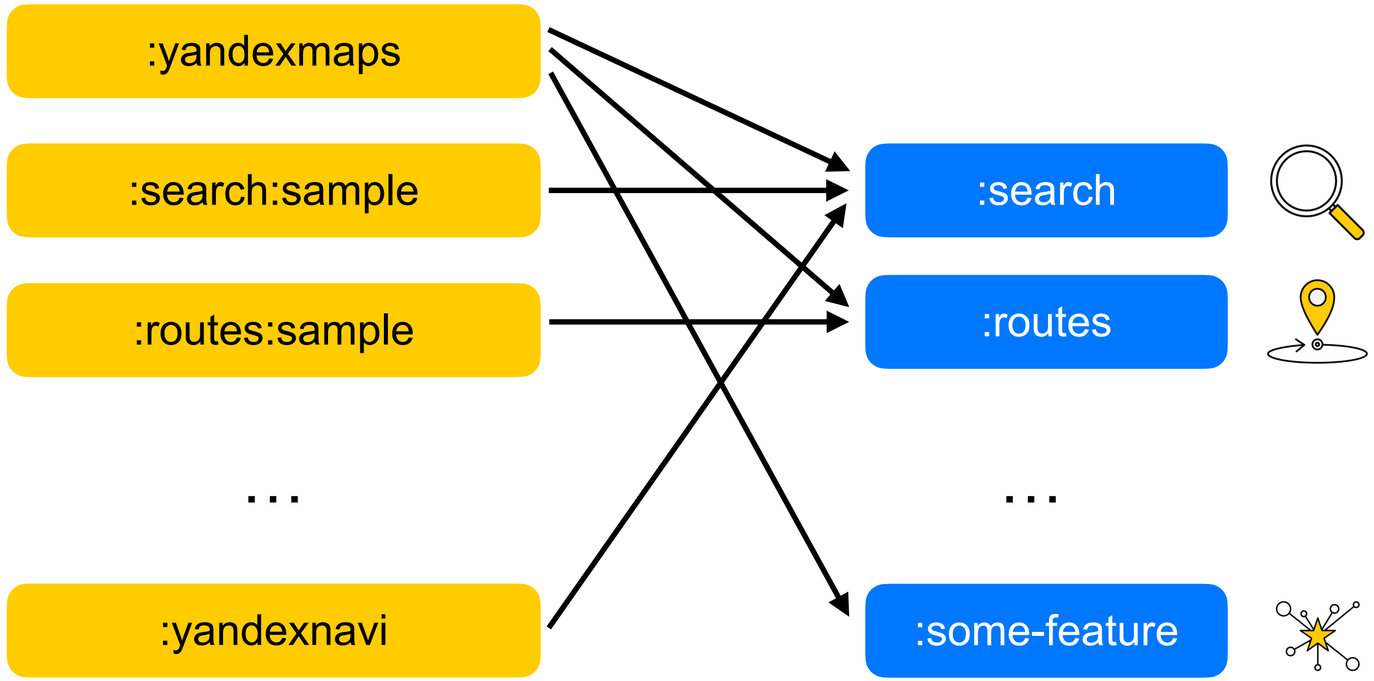

:routes:sample

...


:yandexnavi

Типы модулей

〉 App modules 〉 **Feature modules**



Типичный Feature module

:search

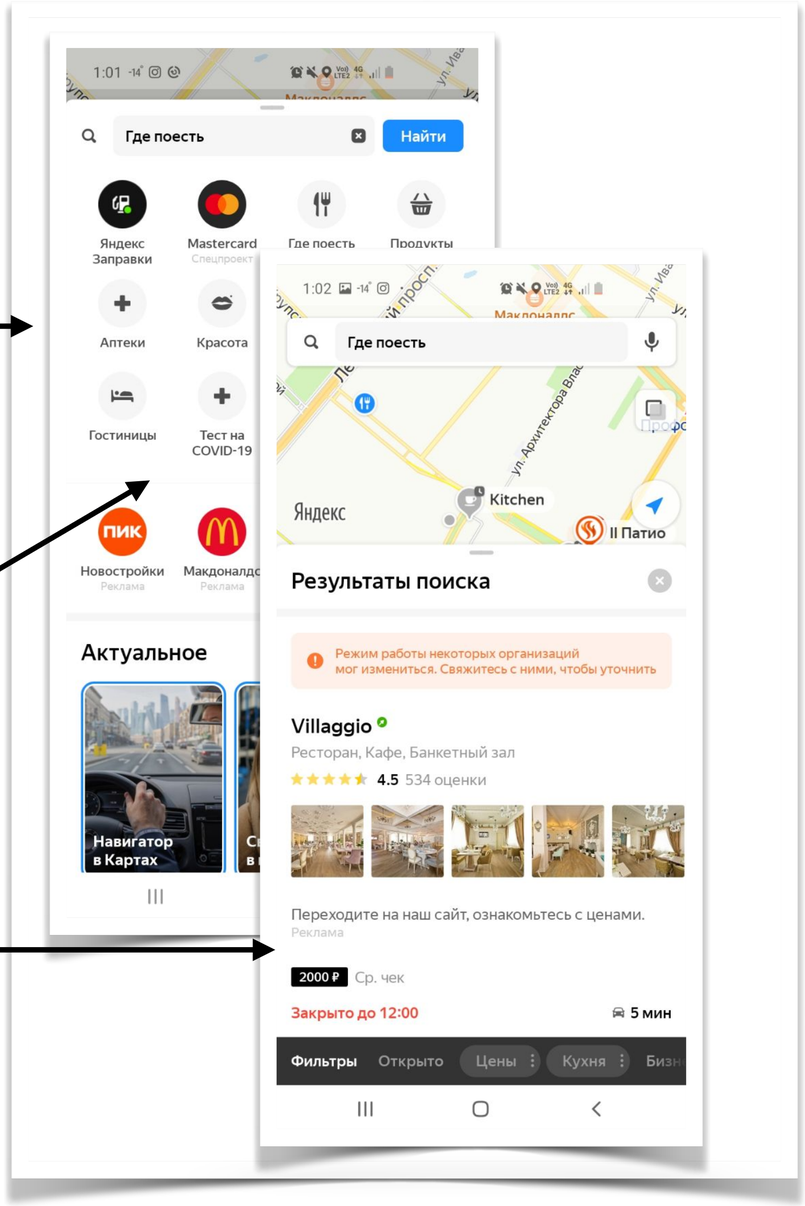
.search.api

SearchFragment

.search.internal

SuggestFragment

SearchResultsFragment



Типичный Feature module

:search

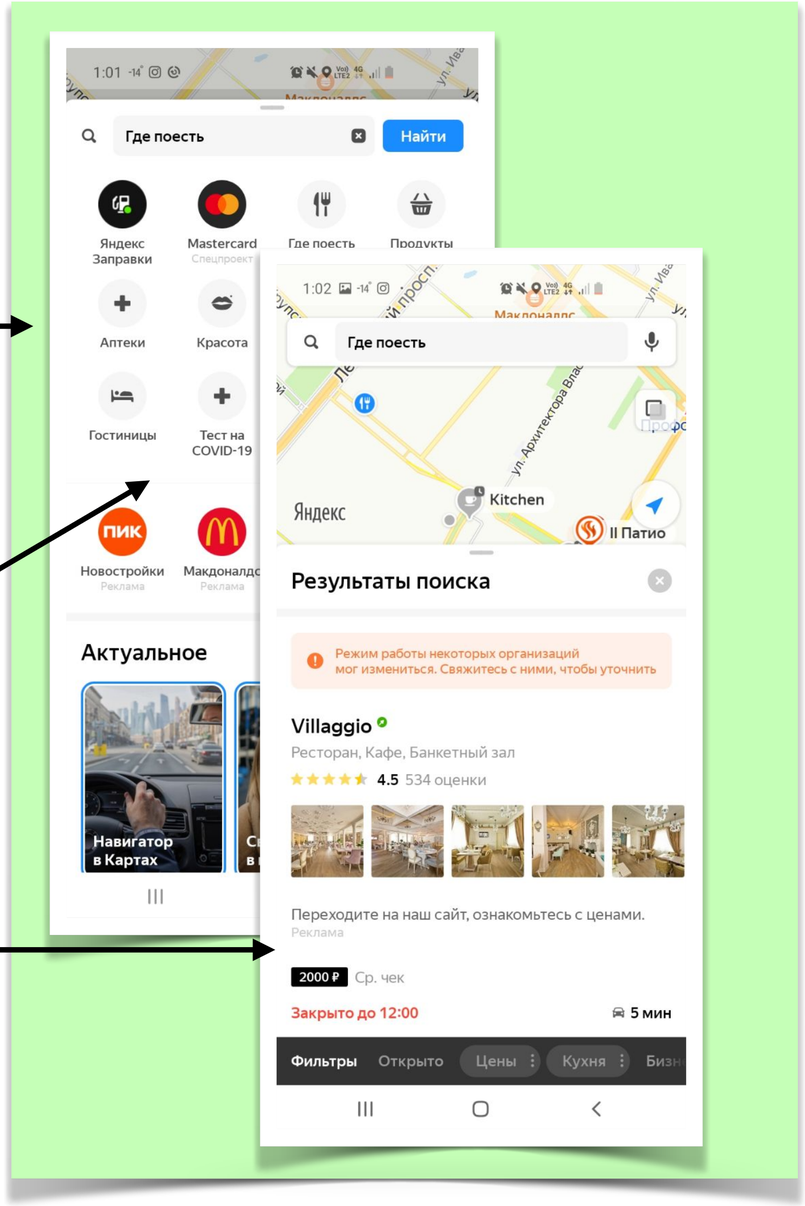
.search.api

SearchFragment

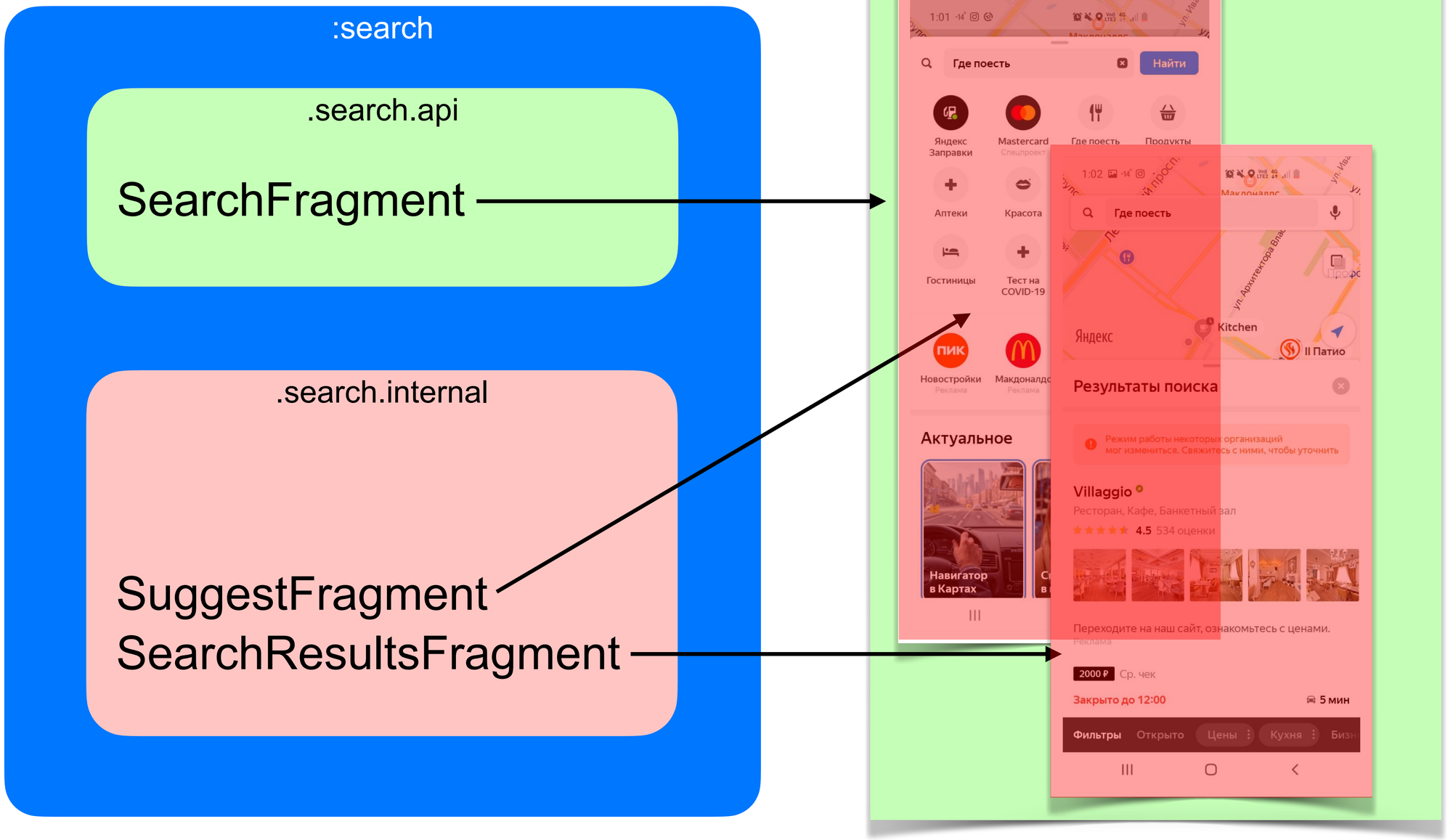
.search.internal

SuggestFragment

SearchResultsFragment

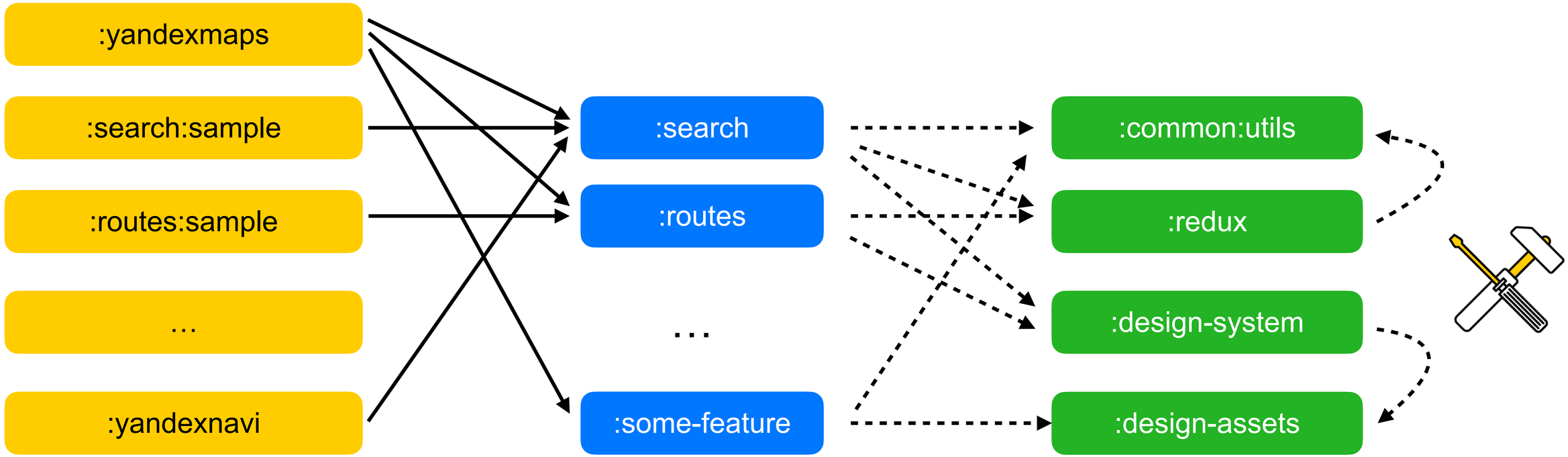


Типичный Feature module



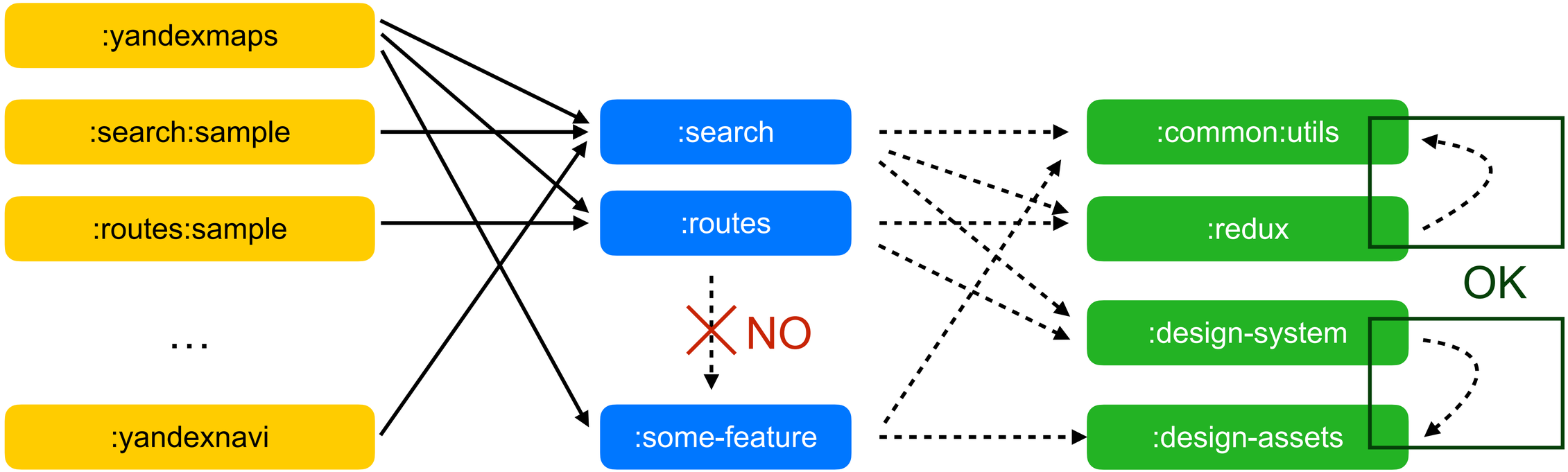
Типы модулей

〉 App modules 〉 Feature modules 〉 **Core modules**



Типы модулей

〉 App modules 〉 Feature modules 〉 **Core modules**



А что с Dependency Injection?

- App modules
- Feature modules
- Core modules

А что с Dependency Injection?

- **App modules - неинтересно**
- Feature modules
- Core modules

А что с Dependency Injection?

- App modules - неинтересно
- Feature modules
- **Core modules - не нужно**

А что с Dependency Injection?

- App modules - неинтересно
- **Feature modules - как построить DI между модулями?**
- Core modules - не нужно

Требования к межмодульному DI

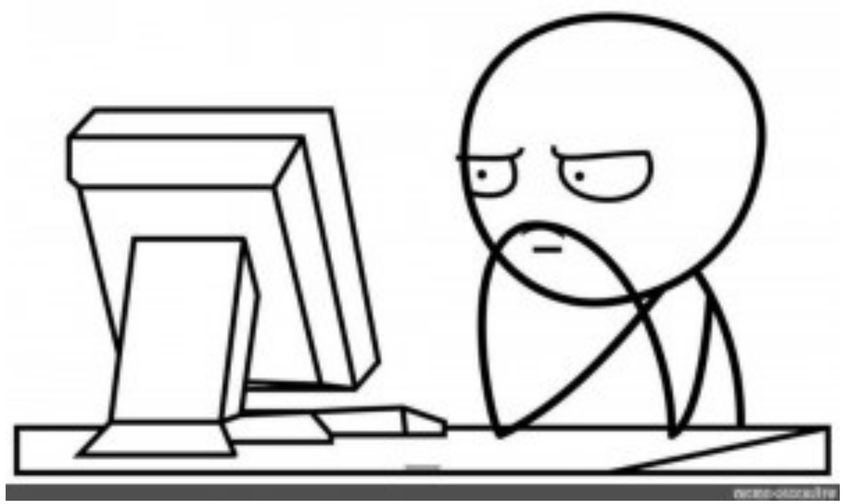
Требования к межмодульному DI

〉 Уметь единообразно работать с зависимостями

Требования к межмодульному DI

〉**Уметь единообразно работать с зависимостями**

1. Не задумываться о деталях при создании нового модуля



Когда создаешь новый модуль

Требования к межмодульному DI

〉**Уметь единообразно работать с зависимостями**

1. Не задумываться о деталях при создании новой фичи
2. Не разбираться, откуда что взялось в незнакомой фиче

Когда правишь баг и попал в неизведанное

Требования к межмодульному DI

〉 Уметь единообразно работать с зависимостями

〉 **Не зависеть от конкретного DI-фрейворка**

Требования к межмодульному DI

〉 Уметь единообразно работать с зависимостями

〉 **Не зависеть от конкретного DI-фрейворка**

Dagger…

Dagger Everywhere

Требования к межмодульному DI

〉Уметь единообразно работать с зависимостями

〉**Не зависеть от конкретного DI-фрейворка**

1. API модуля не содержит Dagger-аннотаций

Требования к межмодульному DI

〉Уметь единообразно работать с зависимостями

〉**Не зависеть от конкретного DI-фрейворка**

1. API модуля не содержит Dagger-аннотаций
2. Внутри каждого модуля - свой DI-граф

Требования к межмодульному DI

〉 Уметь единообразно работать с зависимостями

〉 Не зависеть от конкретного DI-фрейворка:

〉 **Зависимости модуля должны быть легко определимы**

Требования к межмодульному DI

〉 Уметь единообразно работать с зависимостями

〉 Не зависеть от конкретного DI-фрейворка:

〉 **Зависимости модуля должны быть легко определимы**

Когда правишь баг и попал в неизведанное. Опять.

Требования к межмодульному DI

〉 Уметь единообразно работать с зависимостями

〉 Не зависеть от конкретного DI-фрейворка:

〉 Зависимости модуля должны быть легко определимы

Вспомним план

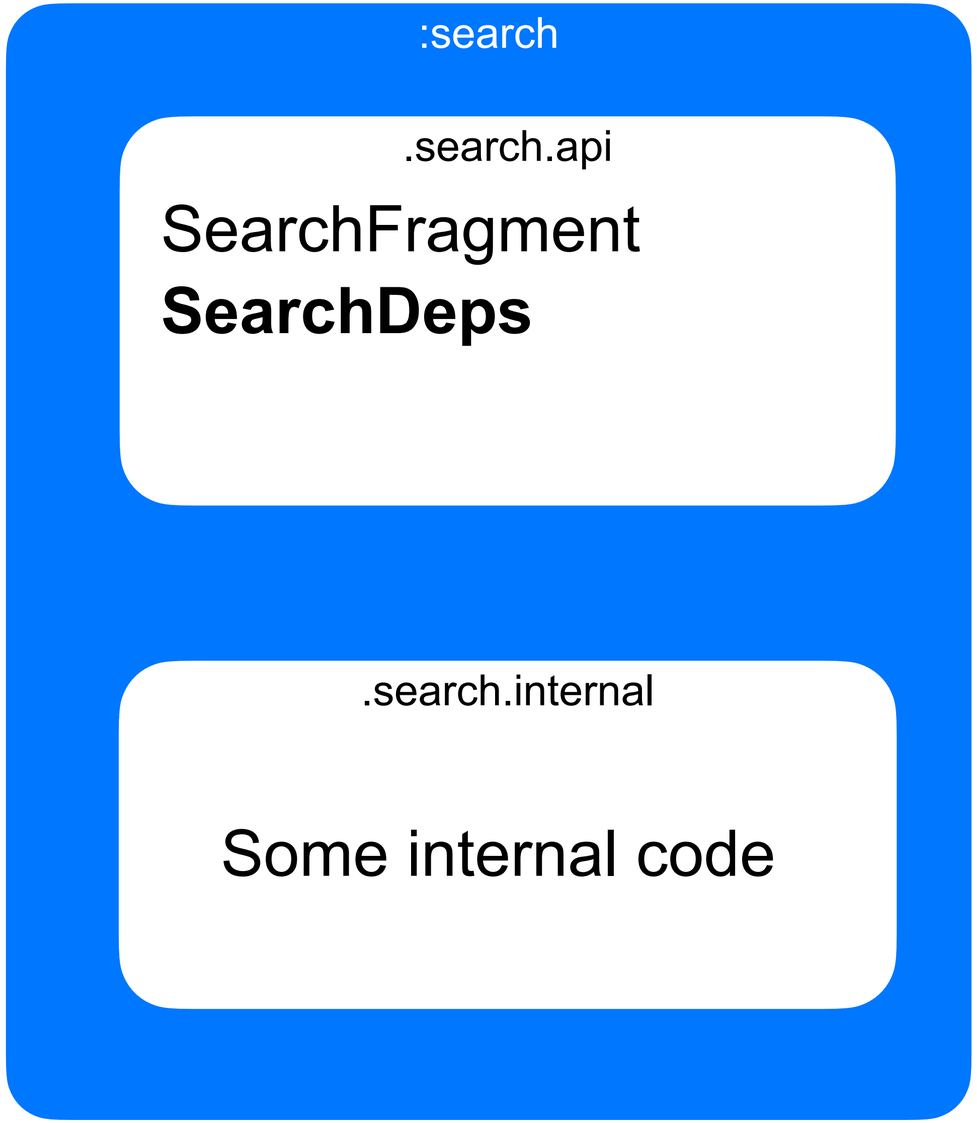
- Модуляризация
- Требования к межмодульному DI
- **Получение зависимостей**
- Предоставление зависимостей
- Использование зависимостей
- Минусы / ограничения / исключения

Получение зависимостей

Feature module API

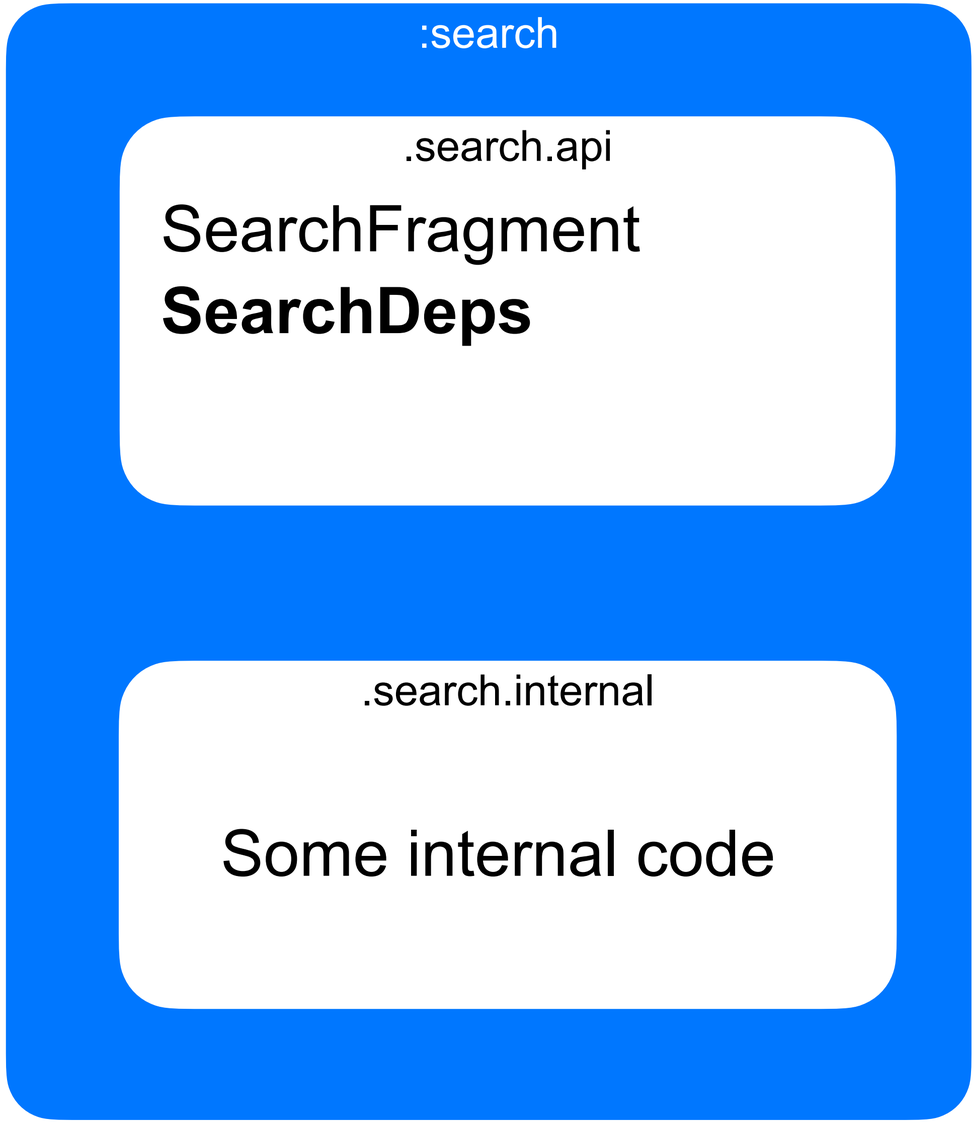

```
class SearchFragment: Fragment() {
    //…
}
```

Feature module API



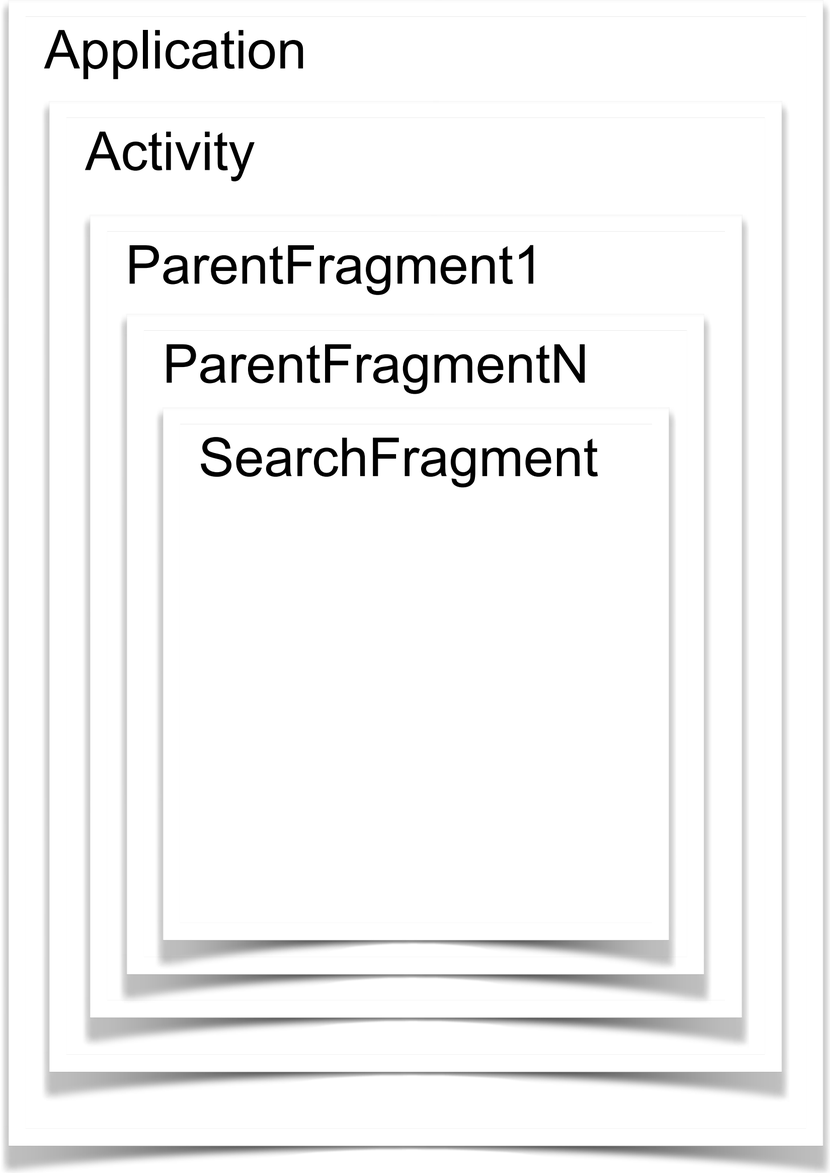

```
class SearchFragment: Fragment() {
    //…
}
```

Feature module API

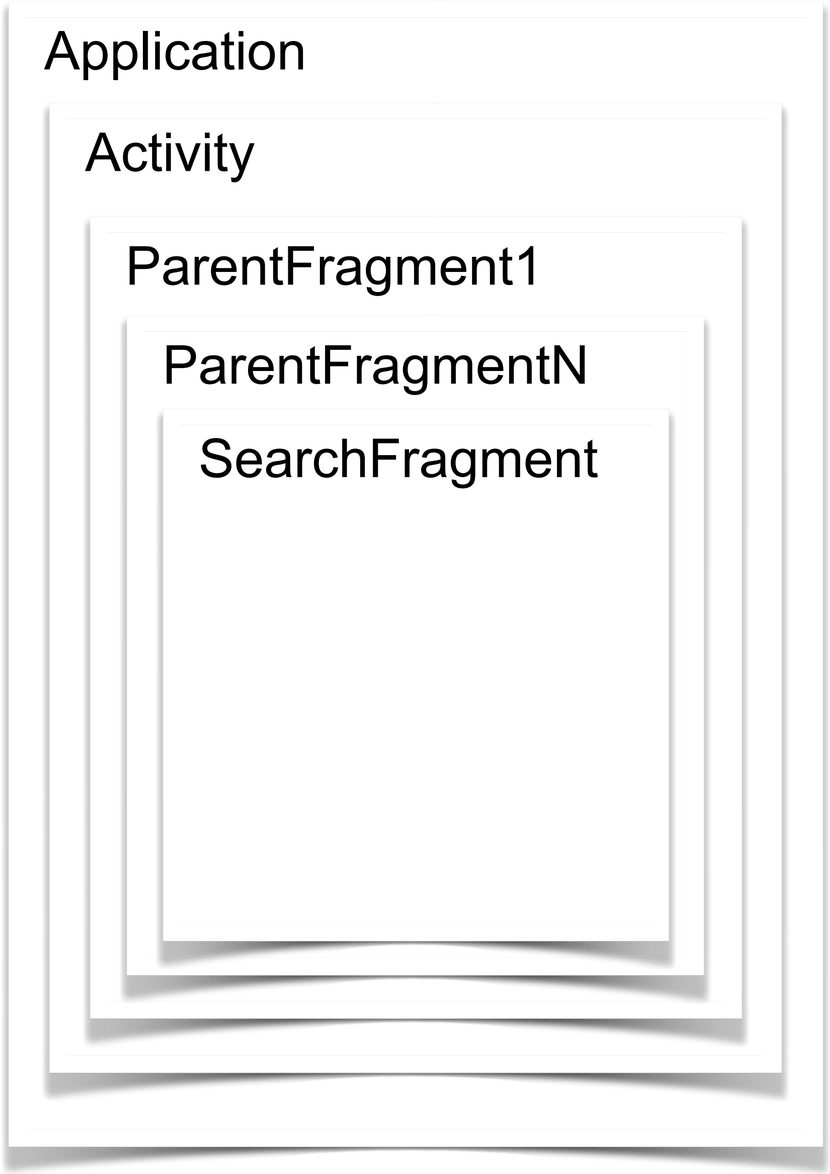



```
class SearchFragment: Fragment() {
    //…
}

interface SearchDeps {
    val logger: Logger
    val searchManager: SearchManager
    // Something else…
}
```

Как во фрагменте получить зависимости?

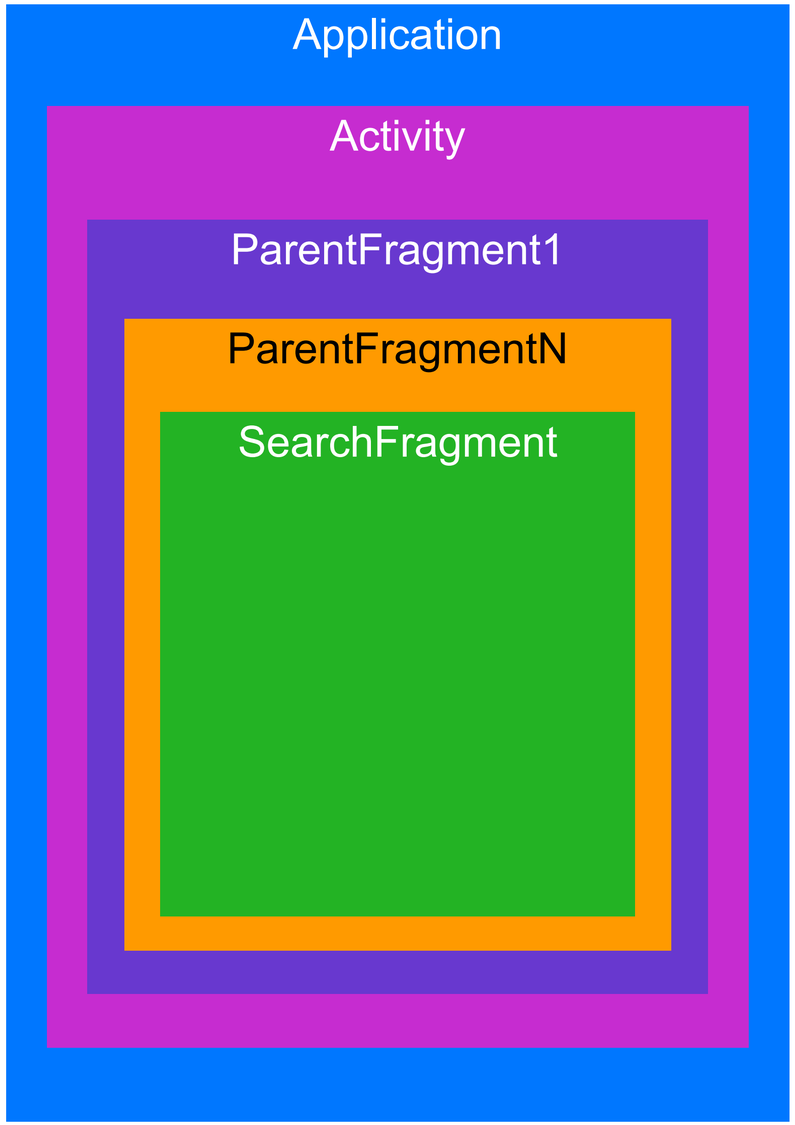


Как во фрагменте получить зависимости?



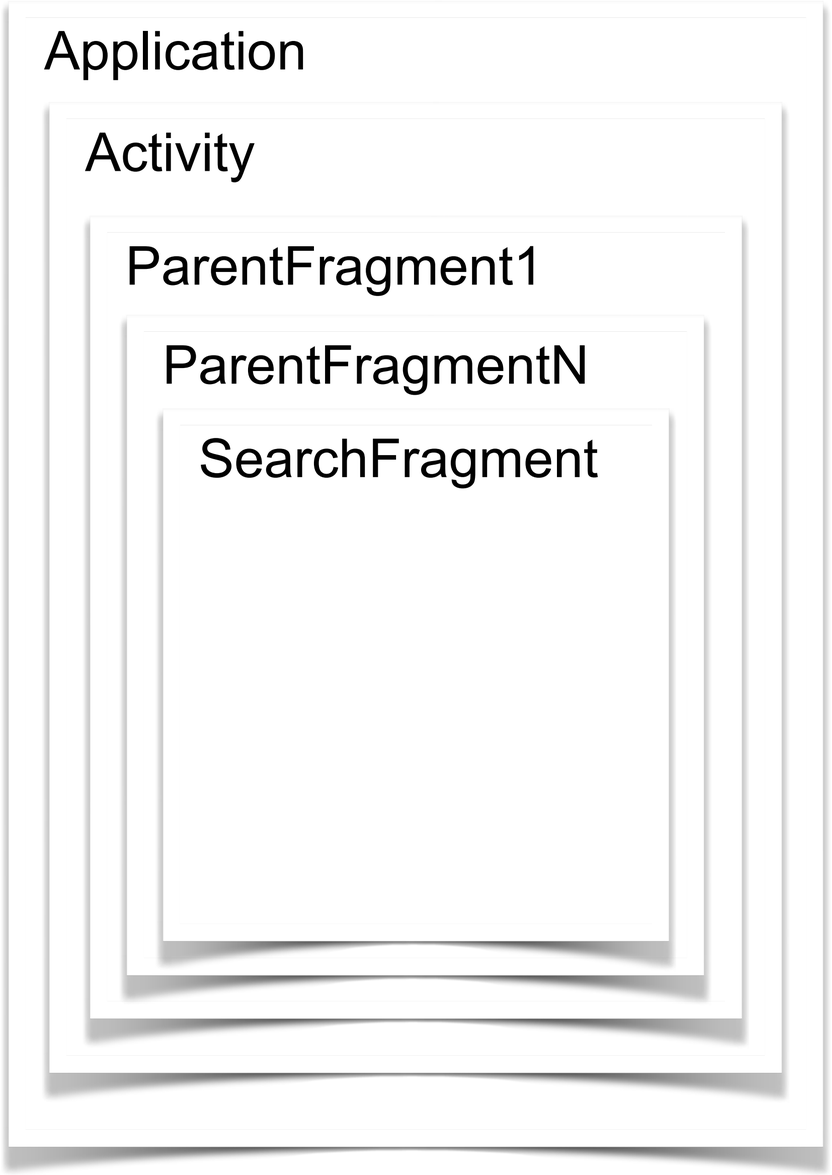

```
fun Fragment.findDependencies(???):???{
    return parents.find { ??? }
}
```

Как во фрагменте получить зависимости?

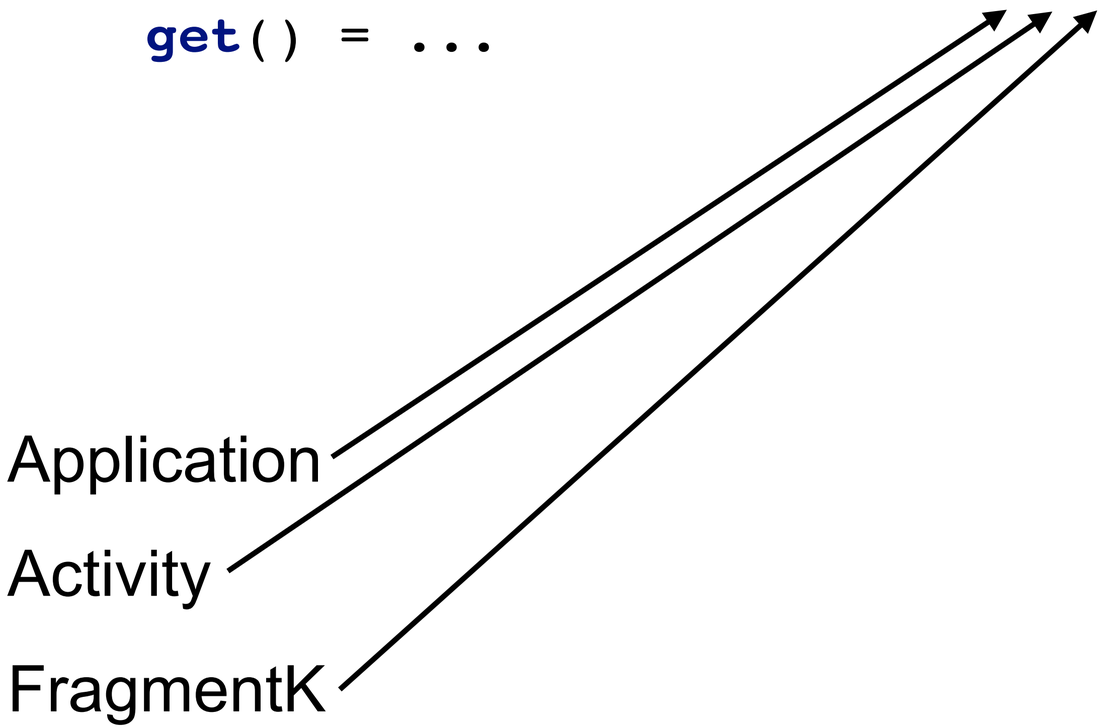



```
fun Fragment.findDependencies(???):???{
    return parents.find { ??? }
}

//returns [ParentN..Parent1, Activity, App]
val Fragment.parents: Iterable<???>
    get() = ...
```

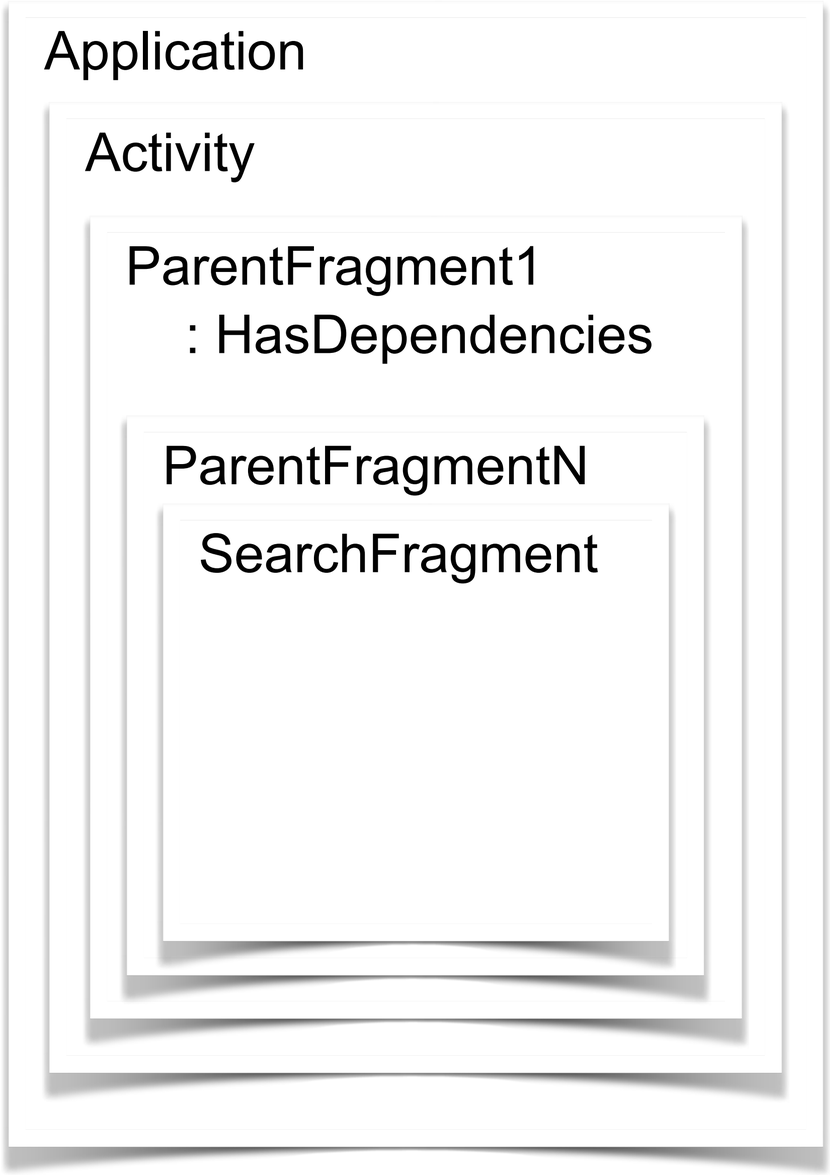
Как во фрагменте получить зависимости?




```
fun Fragment.findDependencies(???):???{
    return parents.find { ??? }
}

//returns [ParentN..Parent1, Activity, App]
val Fragment.parents: Iterable<???>
    get() = ...
```



Как во фрагменте получить зависимости?




```
fun Fragment.findDependencies(???):???{
    return parents.find { ??? }
}
```

```
interface HasDependencies
```

```
//returns [ParentN..Parent1, Activity, App]
val Fragment.parents: Iterable<HasDependencies>
    get() = ...
```

Application: HasDependencies

Activity: HasDependencies

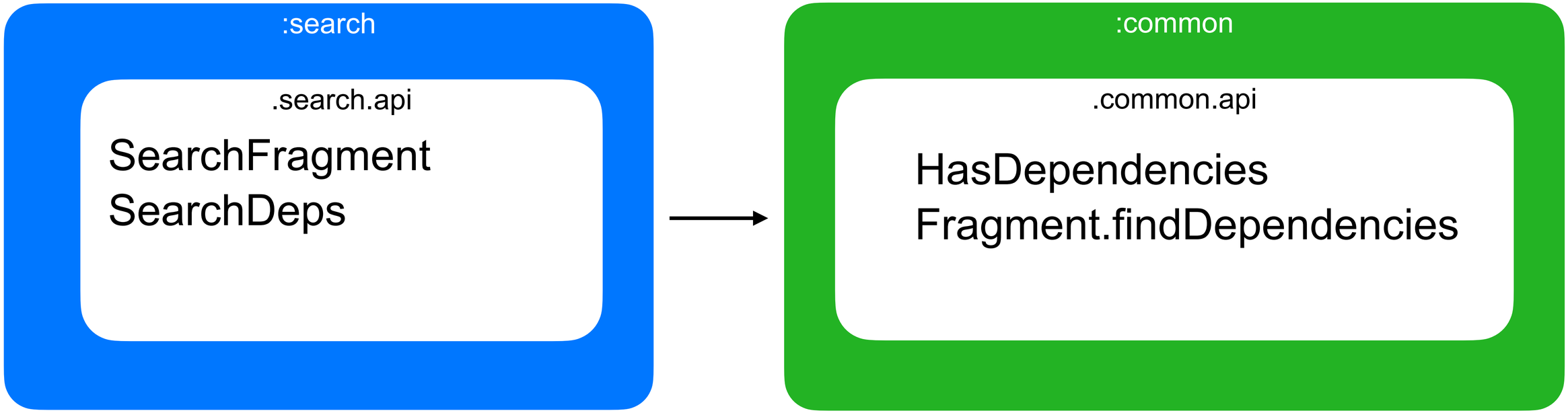
FragmentK: HasDependencies

А что возвращает функция поиска?

```
interface HasDependencies

fun Fragment.findDependencies(???):???{
    return parents.find { ??? }
}
```

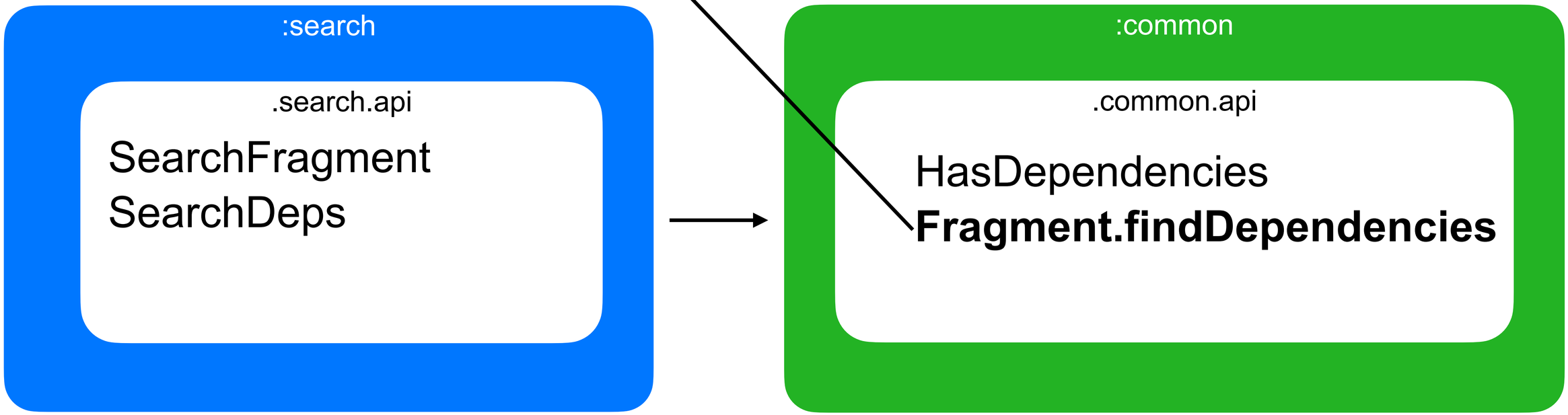
Что возвращает функция поиска?

```
interface HasDependencies

fun Fragment.findDependencies(???):???{
    return parents.find { ??? }
}
```



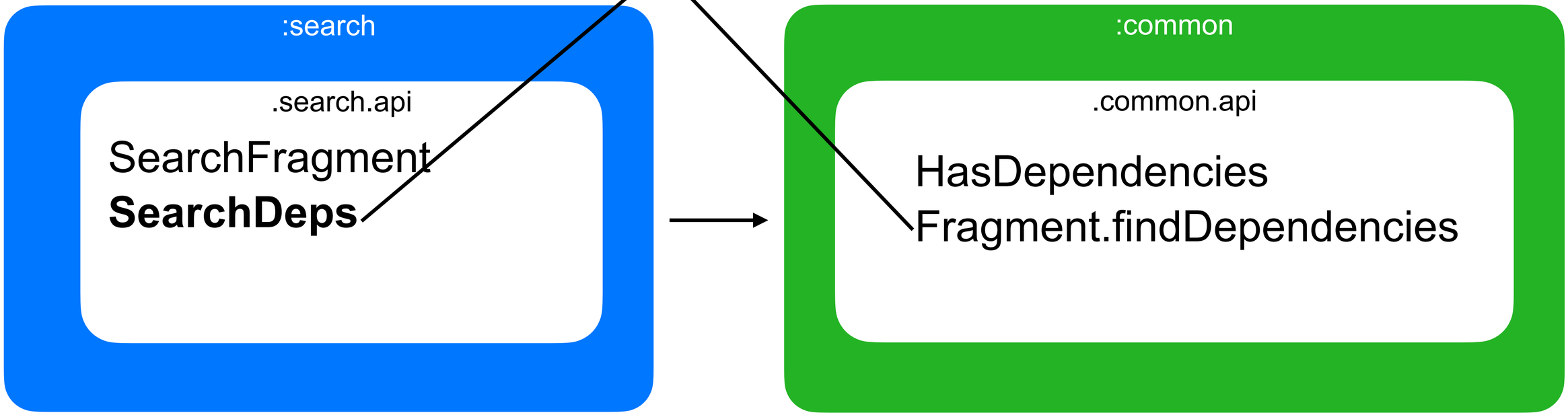
Что возвращает функция поиска?

```
interface HasDependencies

fun Fragment.findDependencies(???):???{
    return parents.find { ??? }
}
```



Что возвращает функция поиска?

```
interface HasDependencies

fun Fragment.findDependencies(???):???{
    return parents.find { ??? }
}
```



Что возвращает функция поиска?

```
fun Fragment.findDependencies(???):???{
    return parents.find { ??? }
}
```

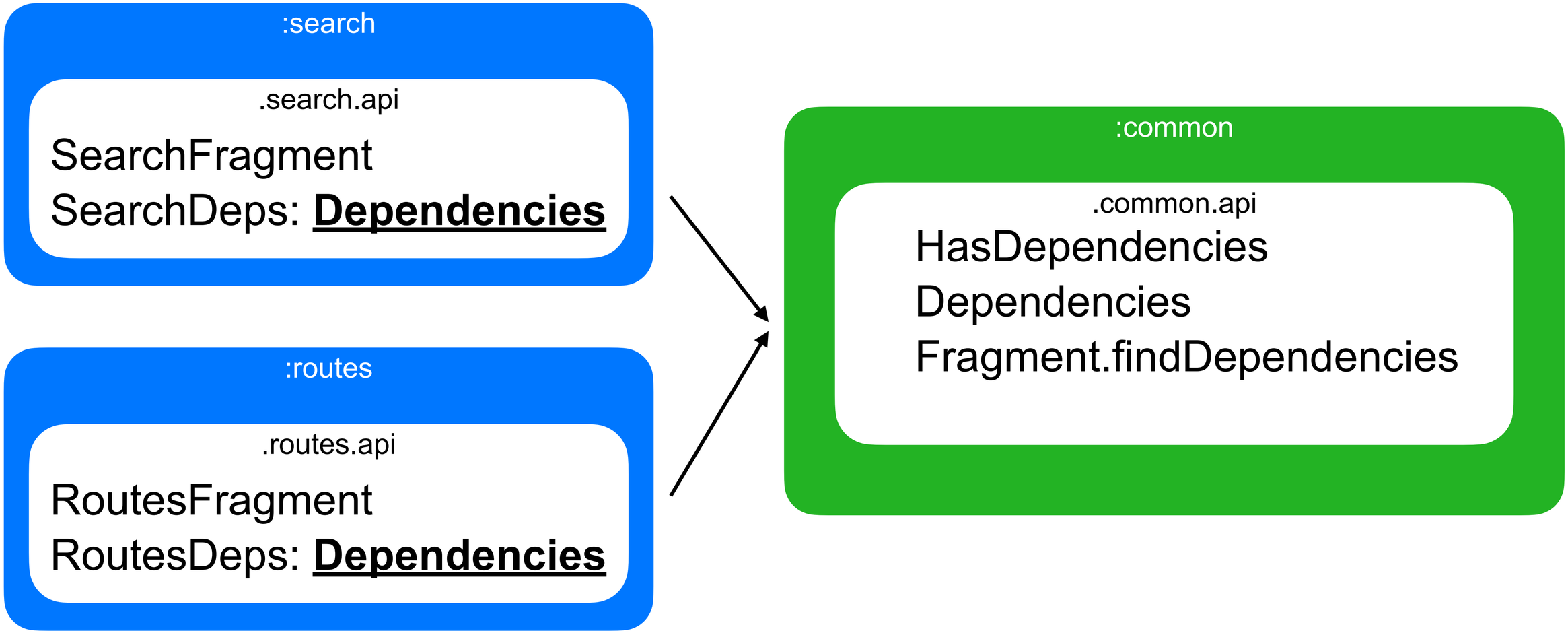
```
interface SearchDeps {
    val searchManager: SearchManager
    // Something else…
}
```

Что возвращает функция поиска?

```
interface Dependencies

fun <D: Dependencies> Fragment.findDependencies(???): D {
    return parents.find { ??? }
}
```

```
interface SearchDeps: Dependencies {
    val searchManager: SearchManager
    // Something else…
}
```

Что возвращает функция поиска?



А что получает функция поиска на вход?

```
interface HasDependencies

interface Dependencies

fun <D: Dependencies> Fragment.findDependencies(???): D {
    return parents.find { ??? }
}
```

```
interface SearchDeps: Dependencies {
    val searchManager: SearchManager
    // Something else…
}
```

А что получает функция поиска на вход?

```
interface HasDependencies

interface Dependencies

fun <D: Dependencies> Fragment.findDependencies(
     c: Class<D>
): D {
    return parents.find { ??? }
}
```

```
interface SearchDeps: Dependencies {
    val searchManager: SearchManager
    // Something else…
}
```

Что хранится внутри HasDependencies?

```
interface HasDependencies {
    ???
}

interface Dependencies

fun <D: Dependencies> Fragment.findDependencies(
     c: Class<D>
): D {
    return parents.find { ??? }
}
```

Что хранится внутри HasDependencies?

```
interface HasDependencies {
    ???
}

interface Dependencies

fun <D: Dependencies> Fragment.findDependencies(
     c: Class<D>
): D {
    return parents.find { ??? }
}
```

Что хранится внутри HasDependencies?

```
typealias DepsMap = Map<Class<out Dependencies>, Dependencies>

interface HasDependencies {
    val depsDepsMap: DepsMap
}

interface Dependencies

fun <D: Dependencies> Fragment.findDependencies(
     c: Class<D>
): D {
    return parents.find { ??? }
}
```

Реализация поиска зависимостей

```
typealias DepsMap = Map<Class<out Dependencies>, Dependencies>

interface HasDependencies {
    val depsMap: DepsMap
}

fun <D: Dependencies> Fragment.findDependencies(
     c: Class<D>
): D {
    return parents
        .mapNotNull { it.depsMap[c] }
        .firstOrNull() as D?
        ?: throw IllegalStateException("Has no $c in $parents")
}

findDependencies(SearchDeps::class.java)
```

Реализация поиска зависимостей

```
typealias DepsMap = Map<Class<out Dependencies>, Dependencies>

interface HasDependencies {
    val depsMap: DepsMap
}

fun <D: Dependencies> Fragment.findDependencies(
     c: Class<D>
): D {
    return parents
        .mapNotNull { it.depsMap[c] }
        .firstOrNull() as D?
        ?: throw IllegalStateException("Has no $c in $parents")
}

findDependencies(SearchDeps::class.java)
```

Реализация поиска зависимостей

```
typealias DepsMap = Map<Class<out Dependencies>, Dependencies>

interface HasDependencies {
    val depsMap: DepsMap
}

fun <D: Dependencies> Fragment.findDependencies(
     c: Class<D>
): D {
    return parents //[ParentN..Parent1, Activity, App]
        .mapNotNull { it.depsMap[c] }
        .firstOrNull() as D?
        ?: throw IllegalStateException("Has no $c in $parents")
}

findDependencies(SearchDeps::class.java)
```

Реализация поиска зависимостей

```
typealias DepsMap = Map<Class<out Dependencies>, Dependencies>

interface HasDependencies {
    val depsMap: DepsMap
}

fun <D: Dependencies> Fragment.findDependencies(
    c: Class<D>
): D {
    return parents
        .mapNotNull { it.depsMap[c] }
        .firstOrNull() as D?
        ?: throw IllegalStateException("Has no $c in $parents")
}

findDependencies(SearchDeps::class.java)
```

Реализация поиска зависимостей

```
typealias DepsMap = Map<Class<out Dependencies>, Dependencies>

interface HasDependencies {
    val depsMap: DepsMap
}

fun <D: Dependencies> Fragment.findDependencies(
     c: Class<D>
): D {
    return parents
        .mapNotNull { it.depsMap[c] }
        .firstOrNull() as D?
        ?: throw IllegalStateException("Has no $c in $parents")
}

findDependencies(SearchDeps::class.java)
```

Реализация поиска зависимостей

```
typealias DepsMap = Map<Class<out Dependencies>, Dependencies>

interface HasDependencies {
    val depsMap: DepsMap
}

fun <D: Dependencies> Fragment.findDependencies(
     c: Class<D>
): D {
    return parents
        .mapNotNull { it.depsMap[c] }
        .firstOrNull() as D?
        ?: throw IllegalStateException("Has no $c in $parents")
}

findDependencies(SearchDeps::class.java)
```

Реализация поиска зависимостей

```
typealias DepsMap = Map<Class<out Dependencies>, Dependencies>

interface HasDependencies {
    val depsMap: DepsMap
}

fun <D: Dependencies> Fragment.findDependencies(
     c: Class<D>
): D {
    return parents
        .mapNotNull { it.depsMap[c] }
        .firstOrNull() as D?
        ?: throw IllegalStateException("Has no $c in $parents")
}

findDependencies(SearchDeps::class.java)
```

Сахарку?

```
interface HasDependencies {
    val depsMap: DepsMap
}

inline fun <reified D: Dependencies> Fragment.findDependencies(): D {
   return parents
     .mapNotNull { it.depsMap[D::class.java] }
     .firstOrNull() as D?
     ?: throw IllegalStateException("No ${D::class.java} in $parents")
}
```

```
findDependencies<SearchDeps>()
```

Сахарку?

```
interface HasDependencies {
    val depsMap: DepsMap
}

inline fun <reified D: Dependencies> Fragment.findDependencies(): D {
   return parents
     .mapNotNull { it.depsMap[D::class.java] }
     .firstOrNull() as D?
     ?: throw IllegalStateException("No ${D::class.java} in $parents")
}
```

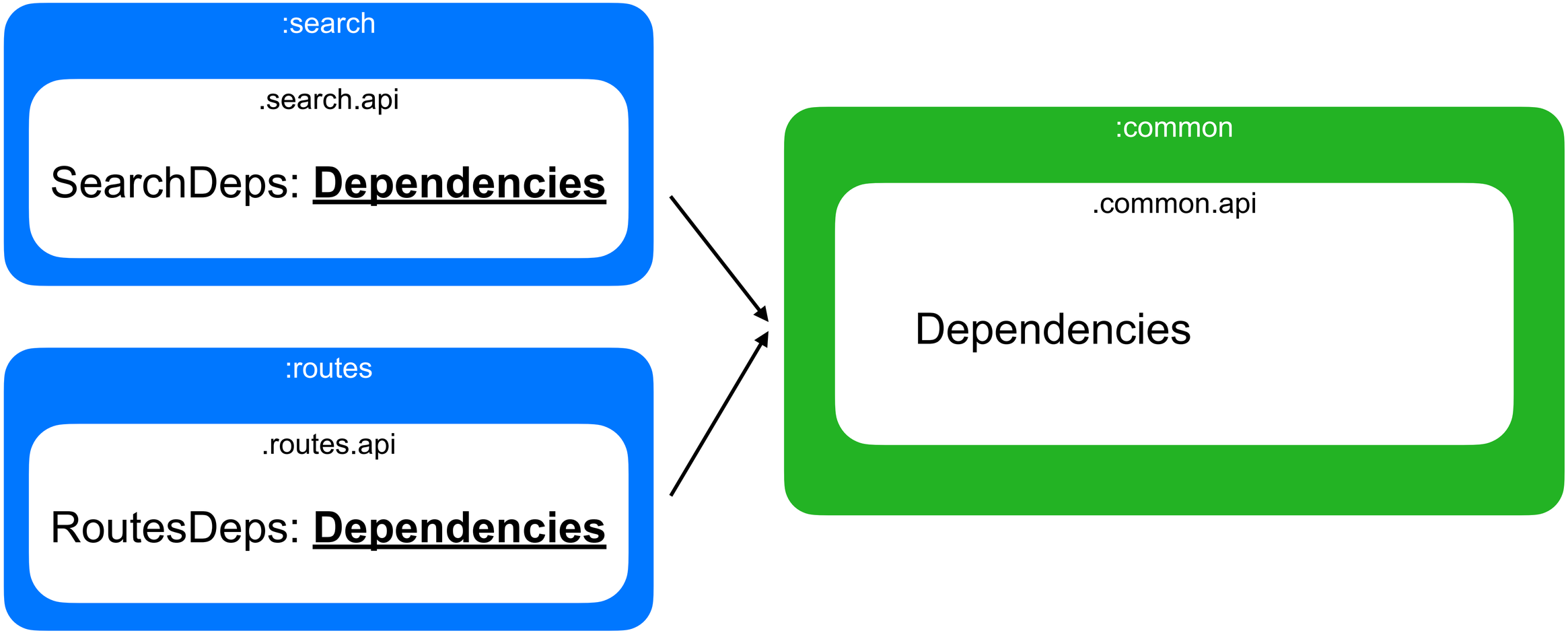
```
findDependencies<SearchDeps>()
```

Настало время подвести итоги

```
interface Dependencies
```

Настало время подвести итоги

```
interface Dependencies
```



Настало время подвести итоги

```
interface Dependencies

typealias DepsMap = Map<Class<out Dependencies>, Dependencies>
```

Настало время подвести итоги

```
interface Dependencies

typealias DepsMap = Map<Class<out Dependencies>, Dependencies>

interface HasDependencies {
    val depsMap: DepsMap
}
```

Application: HasDependencies

Activity: HasDependencies

FragmentK: HasDependencies

Настало время подвести итоги

```
interface Dependencies

typealias DepsMap = Map<Class<out Dependencies>, Dependencies>

interface HasDependencies {
    val depsMap: DepsMap
}

inline fun <reified D: Dependencies> Fragment.findDependencies(): D
```

Настало время подвести итоги

```
interface Dependencies

typealias DepsMap = Map<Class<out Dependencies>, Dependencies>

interface HasDependencies {
    val depsMap: DepsMap
}

inline fun <reified D: Dependencies> Fragment.findDependencies(): D
```

*findDependencies<Feature*Deps>()

Итого

- Feature fragments получают зависимости единообразно
- Интерфейсы написаны нами
- Из Dependencies модуля видно, от чего он зависит

Вспомним план

- Модуляризация
- Требования к межмодульному DI
- Получение зависимостей
- **Предоставление зависимостей**
- Использование зависимостей
- Минусы / ограничения / исключения

Предоставление зависимостей

Предоставление зависимостей

```
typealias DepsMap = Map<Class<out Dependencies>, Dependencies>

interface HasDependencies {
    val depsMap: DepsMap
}


interface SearchDeps: Dependencies {
    val logger: Logger
}
```

Простой случай

```
: search
interface SearchDeps: Dependencies {
    val logger: Logger
}


:search:sample
class SampleActivity: Activity(), HasDependencies {
    override val depsMap: DepsMap = mapOf(
            SearchDeps::class.java to object: SearchDeps {
                override val logger = LoggerImpl()
            }
    )
}
```

Объявлены достаточно простые зависимости

```
: search
interface SearchDeps: Dependencies {
    val logger: Logger
}
```

```
:search:sample
class SampleActivity: Activity(), HasDependencies {
    override val depsMap: DepsMap = mapOf(
            SearchDeps::class.java to object: SearchDeps {
                override val logger = LoggerImpl()
            }
    )
}
```

Реализуем интерфейс HasDependencies

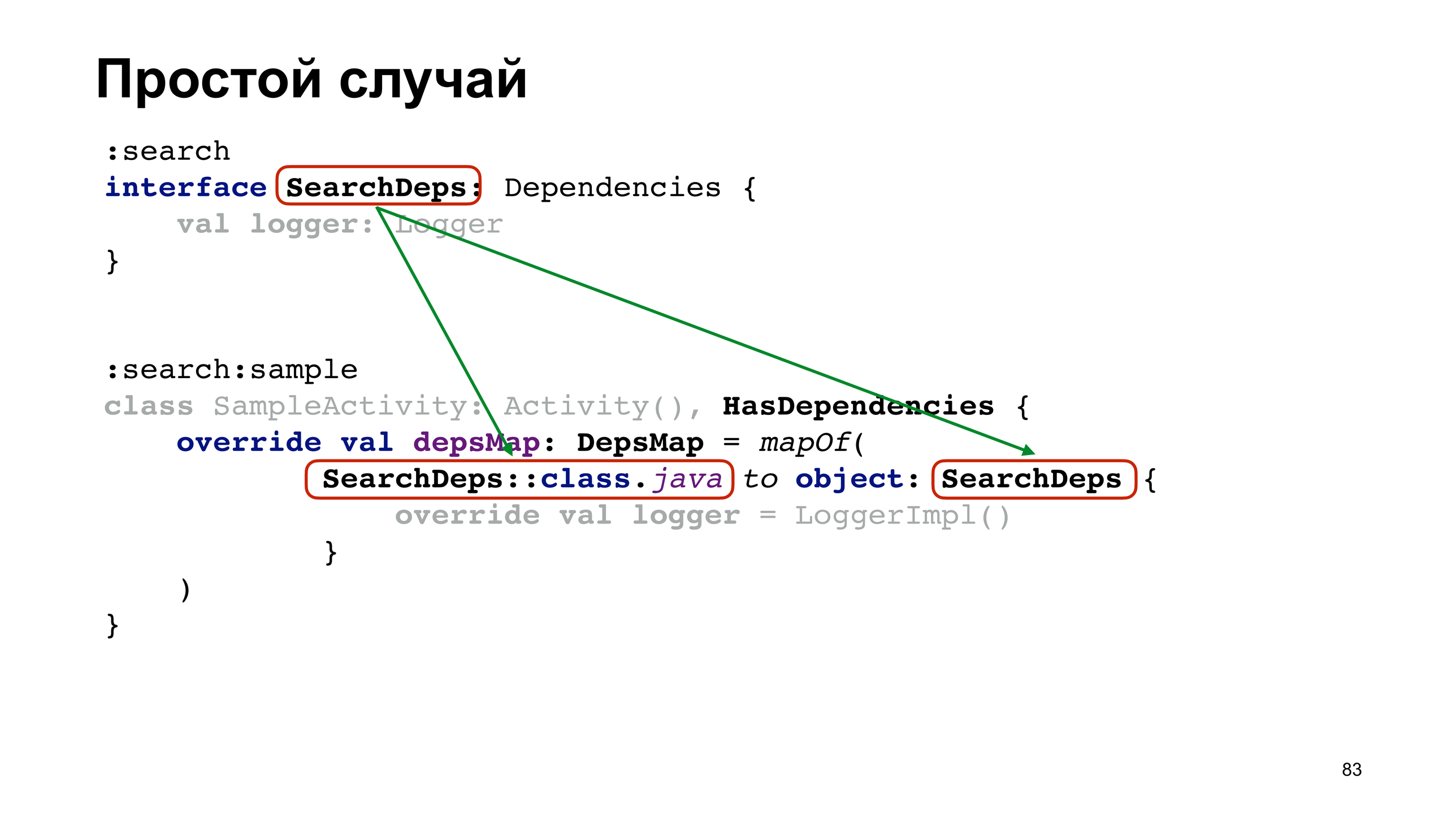
```
: search
interface SearchDeps: Dependencies {
    val logger: Logger
}


:search:sample
class SampleActivity: Activity(), HasDependencies {
    override val depsMap: DepsMap = mapOf(
            SearchDeps::class.java to object: SearchDeps {
                override val logger = LoggerImpl()
            }
    )
}
```

Реализуем интерфейс HasDependencies

```
:search
interface SearchDeps: Dependencies {
    val logger: Logger
}


:search:sample
class SampleActivity: Activity(), HasDependencies {
    override val depsMap: DepsMap = mapOf(
            SearchDeps::class.java to object: SearchDeps {
                override val logger = LoggerImpl()
            }
    )
}
```

Простой случай

```
:search
interface SearchDeps: Dependencies {
    val logger: Logger
}


:search:sample
class SampleActivity: Activity(), HasDependencies {
    override val depsMap: DepsMap = mapOf(
            SearchDeps::class.java to object: SearchDeps {
                override val logger = LoggerImpl()
            }
    )
}
```



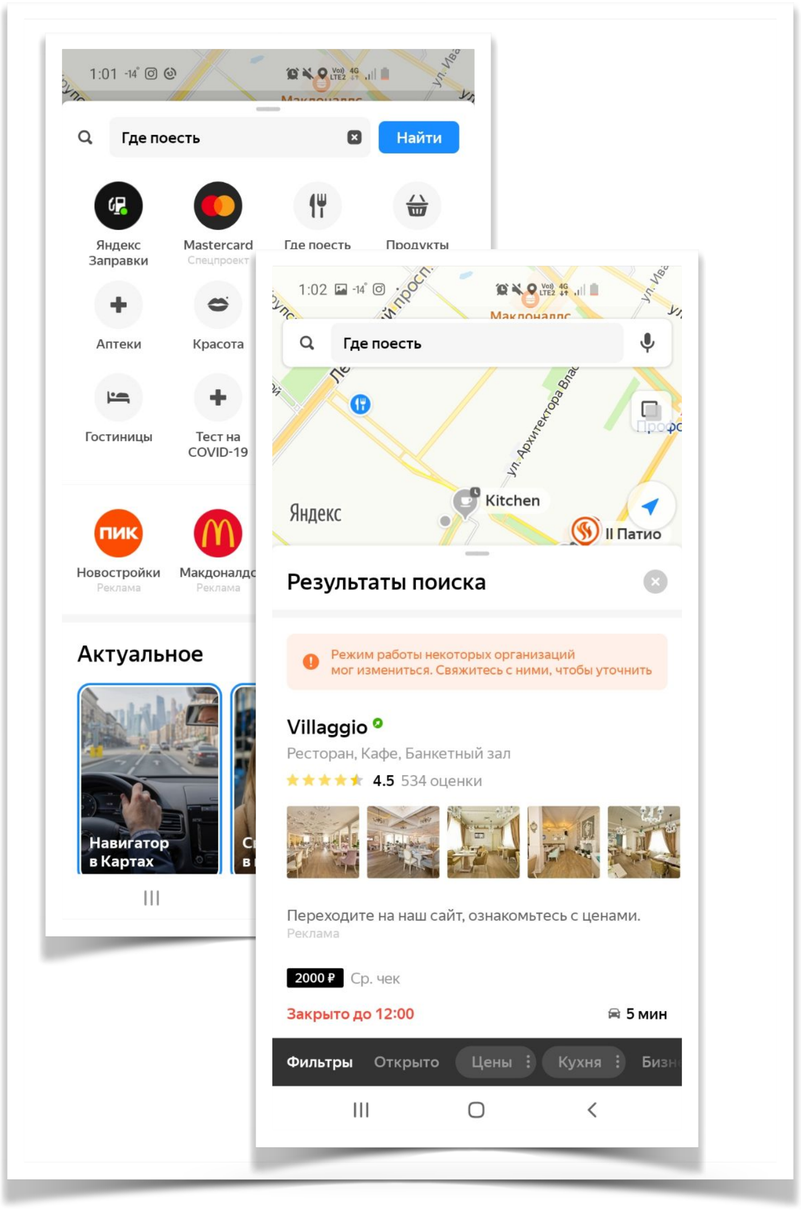
Реализуем непосредственно зависимость

```
:search
interface SearchDeps: Dependencies {
    val logger: Logger
}


:search:sample
class SampleActivity: Activity(), HasDependencies {
    override val depsMap: DepsMap = mapOf(
            SearchDeps::class.java to object: SearchDeps {
                override val logger = LoggerImpl()
            }
    )
}
```

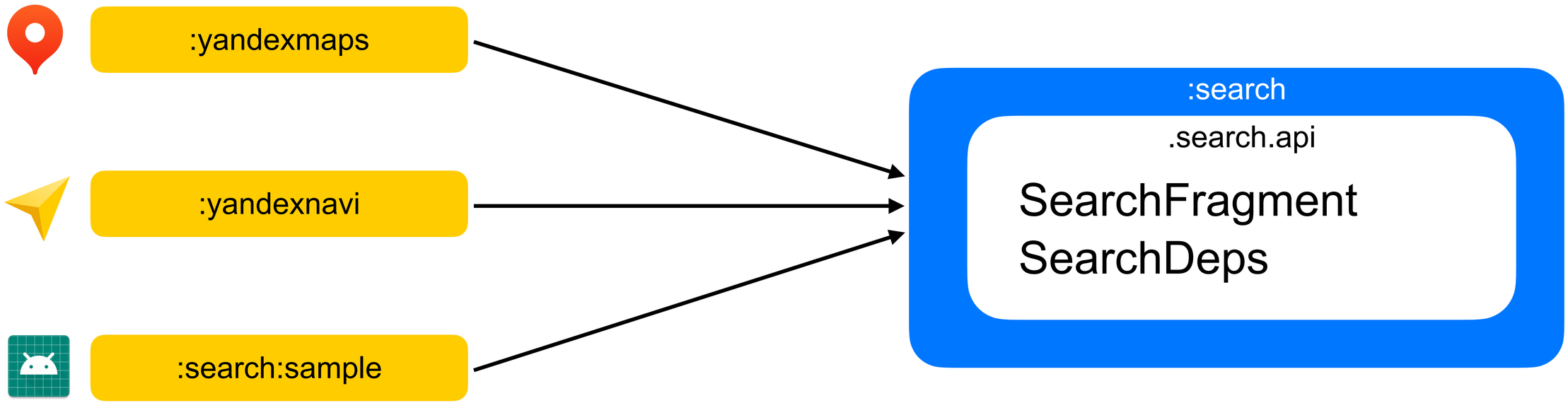
```
:search:sample
class SampleActivity: Activity(), HasDependencies {
    override val depsMap: DepsMap = mapOf(
            SearchDeps::class.java to object: SearchDeps {
                override val logger = LoggerImpl()
            }
    )

    override fun onCreate(…) {
      commit(SearchFragment())
    }
}
```

Сложный случай

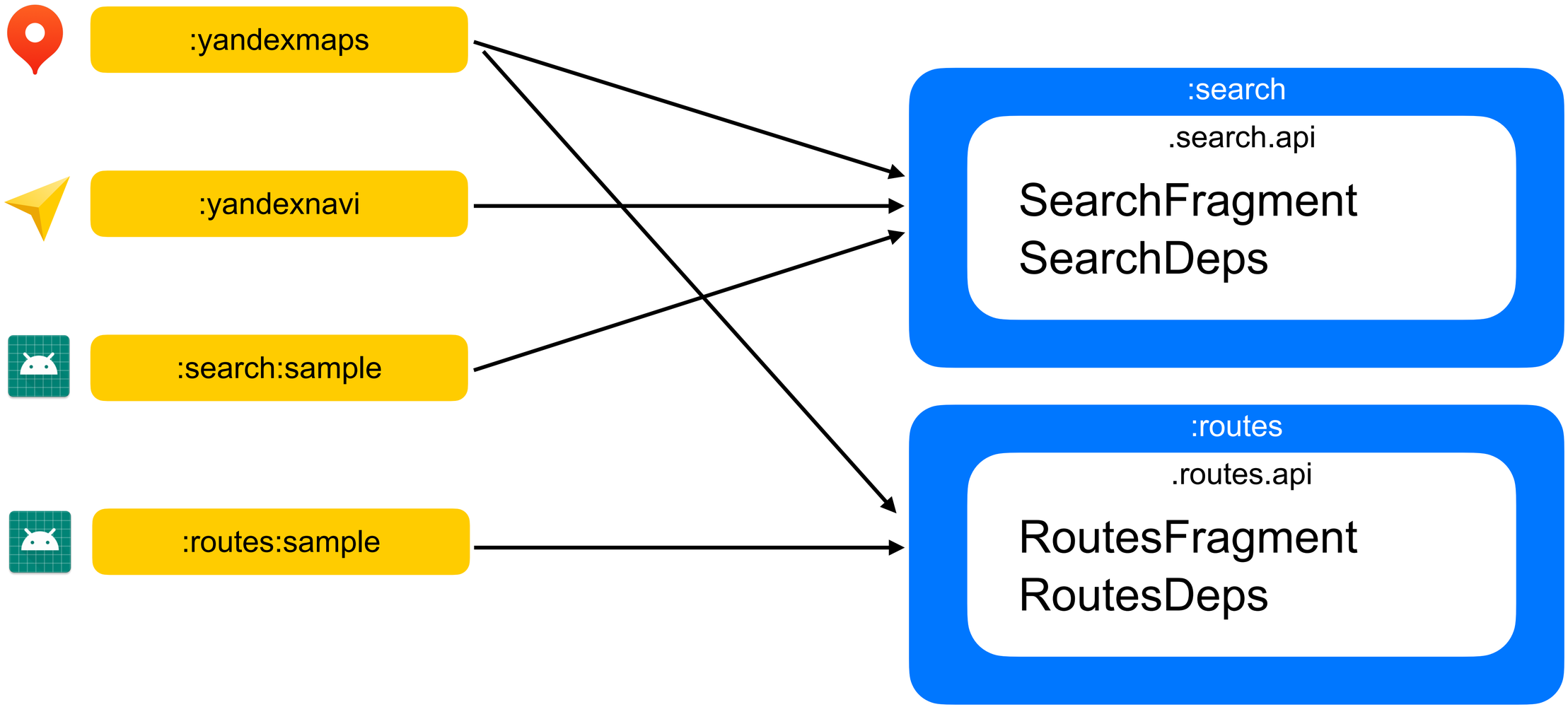
```
interface SearchDependencies : Dependencies {
    val searchHistoryService: SearchHistoryService
    val search: Search
    val map: Map
    val camera: Camera
    val searchLocationService: SearchLocationService
    val searchBannersConfigService: SearchBannersConfigService
    val application: Application
    val refWatcher: RefWatcherWrapper?
    val externalSearchPreferences: ExternalSearchPreferences
    val categoriesProvider: CategoriesProvider
    val searchLayer: SearchLayer
    val searchRecognizer: SearchRecognizer
    val searchExternalNavigator: SearchExternalNavigator
    val routeSerpSearchClickListener: RouteSerpSearchClickListener
    val searchStateMutator: SearchStateMutator
    val searchExitStrategy: SearchExitStrategy
    val searchResultCardProvider: SearchResultCardProvider<*>
    val mtThreadCardControllerProvider: MtThreadCardControllerProvider<*>
    val mtStopCardControllerProvider: MtStopCardControllerProvider<*>
    val offerProvider: MastercardOfferProvider
    val moshi: Moshi
    val viewPool: PrefetchRecycledViewPool
    val prefetcherManager: SnippetPrefetcherManager
    val fluidContainerShoreSupplier: FluidContainerShoreSupplier
    val purse: Purse
    val facebookLogger: SearchFacebookLogger?
    val snippetFactory: SnippetFactory
    val experimentsProvider: SearchExperimentsProvider
    val searchCallbacks: SearchControllerCallbacks
    val showcaseLookupService: ShowcaseLookupService
    val transportOverlayTemporaryDisabler: SearchTransportOverlayTemporaryDisabler
    val keyboardManager: KeyboardManager
    val contextProvider: ActivityContextProvider
    val modulePlacement: ModulePlacement
    val cameraMovementController: SearchCameraController
    val searchOptionsFactory: SearchOptionsFactory
    val searchLineExternalInteractor: SearchLineExternalInteractor
}
```



Сложный случай



Сложный случай



Сложный случай

```
:yandexmaps
class MainActivity: Activity(), HasDependencies {
    override val depsMap: DepsMap = mapOf(
            SearchDeps::class.java to object: SearchDeps { },
            RoutesDeps::class.java to object: RoutesDeps { },
            …
    )

}
```

Dagger 2: Multibindings

Dagger 2: Multibindings

dagger.Module A

Binds “One” to 1
Binds “Three” to 3

dagger.Module B

Binds “Two” to 2
Binds “Four” to 4

Dagger 2: Multibindings

dagger.Module A

Binds “One” to 1
Binds “Three” to 3

dagger.Component A
modules=[A::class]

dagger.Module B

Binds “Two” to 2
Binds “Four” to 4

dagger.Component AB
modules=[A::class, B::class]

Dagger 2: Multibindings

dagger.Module A

Binds “One” to 1
Binds “Three” to 3

dagger.Component A
modules=[A::class]

mapOf(
“One” to 1,
“Three” to 3)

dagger.Module B

Binds “Two” to 2
Binds “Four” to 4

dagger.Component AB
modules=[A::class, B::class]

mapOf(
“One” to 1,
“Three” to 3,
“Two” to 2,
“Four” to 4)

Dagger 2: Multibindings

dagger.Module SearchDepsBinding

Binds SearchDeps::**class**
to
Impl1

dagger.Module RoutesDepsBinding

Binds RoutesDeps::**class**
to
Impl2

dagger.Component
MainActivityComponet modules=[
SearchDepsBinding::class,
RoutesDepsBinding::class]

```
mapOf(
  SearchDeps::class to impl1
  RoutesDeps::class to impl2
)
```

Dagger 2: Multibindings

dagger.Module SearchDepsBinding

Binds SearchDeps::**class**
to
impl1

dagger.Module RoutesDepsBinding

Binds RoutesDeps::**class**
to
impl2

dagger.Component
MainActivityComponet modules=[
SearchDepsBinding::class,
RoutesDepsBinding::class]

```
depsMap: DepsMap = mapOf(
  SearchDeps::class to impl1
  RoutesDeps::class to impl2
)
```

```
typealias DepsMap = Map<Class<out Dependencies>, Dependencies>
```

Dagger 2: Multibindings

```
:yandexmaps
@dagger.Module
interface SearchDepsBindingsModule {
    @dagger.Binds
    @dagger.IntoMap
    @DependenciesKey(SearchDeps::class)
    fun bindSearchDeps(impl: ???): Dependencies
}
```

Dagger 2: Multibindings

```
:yandexmaps
@dagger.Module
interface SearchDepsBindingsModule {
    @dagger.Binds
    @dagger.IntoMap
    @DependenciesKey(SearchDeps::class)
    fun bindSearchDeps(impl: ???): Dependencies
}
```

Dagger 2: Multibindings

```
:yandexmaps
@dagger.Module
interface SearchDepsBindingsModule {
    @dagger.Binds
    @dagger.IntoMap
    @DependenciesKey(SearchDeps::class)
    fun bindSearchDeps(impl: ???): Dependencies
}
```

Dagger 2: Multibindings

```
:common
@dagger.MapKey
annotation class DependenciesKey(val value: KClass<out Dependencies>)

:yandexmaps
@dagger.Module
interface SearchDepsBindingsModule {
    @dagger.Binds
    @dagger.IntoMap
    @DependenciesKey(SearchDeps::class)
    fun bindSearchDeps(impl: ???): Dependencies
}
```

Dagger 2: Multibindings

```
:common
@dagger.MapKey
annotation class DependenciesKey(val value: KClass<out Dependencies>)

:yandexmaps
@dagger.Module
interface SearchDepsBindingsModule {
    @dagger.Binds
    @dagger.IntoMap
    @DependenciesKey(SearchDeps::class)
    fun bindSearchDeps(impl: ???): Dependencies
}
```

KEY

Dagger 2: Multibindings

```
:common
@dagger.MapKey
annotation class DependenciesKey(val value: KClass<out Dependencies>)

:yandexmaps
@dagger.Module
interface SearchDepsBindingsModule {
    @dagger.Binds
    @dagger.IntoMap
    @DependenciesKey(SearchDeps::class)
    fun bindSearchDeps(impl: ???): Dependencies
}
```

VALUE

Dagger 2: Multibindings

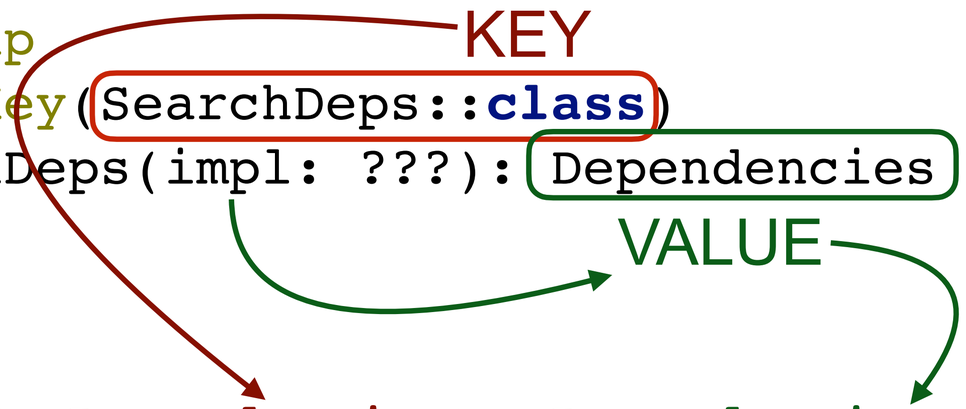
```
:common
@dagger.MapKey
annotation class DependenciesKey(val value: KClass<out Dependencies>)

:yandexmaps
@dagger.Module
interface SearchDepsBindingsModule {
    @dagger.Binds
    @dagger.IntoMap
    @DependenciesKey(SearchDeps::class)
    fun bindSearchDeps(impl: ???): Dependencies
}
```

KEY

VALUE

Dagger 2: Multibindings

```
:common
@dagger.MapKey
annotation class DependenciesKey(val value: KClass<out Dependencies>)

:yandexmaps
@dagger.Module
interface SearchDepsBindingsModule {
    @dagger.Binds
    @dagger.IntoMap
    @DependenciesKey(SearchDeps::class)
    fun bindSearchDeps(impl: ???): Dependencies
}

GENERATED JAVA
Map<Class<? extends Dependencies>, Dependencies>
```



Dagger 2: Multibindings

:common

```
typealias DepsMap = Map<Class<out Dependencies>, Dependencies>

interface HasDependencies {
    val depsMap: DepsMap
}
```

:yandexmaps

```
GENERATED JAVA
Map<Class<? extends Dependencies>, Dependencies>>
```

Dagger 2: Multibindings

```
:yandexmaps
@dagger.Module
interface SearchDepsBindingsModule {
    @dagger.Binds
    @dagger.IntoMap
    @DependenciesKey(Searchs::class)
    fun bindSearchDeps(impl: ???): Dependencies
}
```

Dagger 2: Multibindings

```
:yandexmaps
@dagger.Module
interface SearchDepsBindingsModule {
    @dagger.Binds
    @dagger.IntoMap
    @DependenciesKey(SearchDeps::class)
    fun bindSearchDeps(impl: ???): Dependencies
}

@dagger.Component
interface MainActivityComponent {
  val map: Map
  val camera: Camera
  val moshi: Moshi
  val viewPool: PrefetchRecycledViewPool
  …
}
```

Dagger 2: Multibindings

```
:yandexmaps
@dagger.Module
interface SearchDepsBindingsModule {
    @dagger.Binds
    @dagger.IntoMap
    @DependenciesKey(SearchDeps::class)
    fun bindSearchDeps(impl: ???): Dependencies
}

@dagger.Component
interface MainActivityComponent: SearchDeps {
  override val map: Map
  override val camera: Camera
  override val moshi: Moshi
  override val viewPool: PrefetchRecycledViewPool
  …
}
```

Dagger 2: Multibindings

```
:yandexmaps
@dagger.Module
interface SearchDepsBindingsModule {
    @dagger.Binds
    @dagger.IntoMap
    @DependenciesKey(SearchDeps::class)
    fun bindSearchDeps(impl: ???): Dependencies
}

@dagger.Component
interface MainActivityComponent: SearchDeps {
    fun inject(activity: MainActivity)
}
```

Dagger 2: Multibindings

```
:yandexmaps
@dagger.Module
interface SearchDepsBindingsModule {
    @dagger.Binds
    @dagger.IntoMap
    @DependenciesKey(SearchDeps::class)
    fun bindSearchDeps(impl: MainActivityComponent): Dependencies
}

@dagger.Component(modules = [SearchDepsBindingsModule::class])
interface MainActivityComponent: SearchDeps {
    fun inject(activity: MainActivity)
}
```

Dagger 2: Multibindings

```
:yandexmaps
@dagger.Module
interface SearchDepsBindingsModule {
    @dagger.Binds
    @dagger.IntoMap
    @DependenciesKey(SearchDeps::class)
    fun bindSearchDeps(impl: MainActivityComponent): Dependencies
}

@dagger.Component(modules = [SearchDepsBindingsModule::class])
interface MainActivityComponent: SearchDeps {
    fun inject(activity: MainActivity)
}
```

Реализация зависимостей

```
@dagger.Component(modules = [SearchDepsBindingsModule::class])
interface MainActivityComponent : SearchDeps {
    fun inject(activity: MainActivity)
}

class MainActivity : Activity(), HasDependencies {
    @Inject
    override lateinit var depsMap: DepsMap

    override fun onCreate(…) {
        DaggerMainActivityComponent.factory().create().inject(this)
    }
}
```

Реализация зависимостей

```
@dagger.Component(modules = [SearchDepsBindingsModule::class])
interface MainActivityComponent : SearchDeps {
    fun inject(activity: MainActivity)
}

class MainActivity : Activity(), HasDependencies {
    @Inject
    override lateinit var depsMap: DepsMap

    override fun onCreate(…) {
        DaggerMapsActivityComponent.factory().create().inject(this)
    }
}
```

Реализация зависимостей

```
@dagger.Component(modules = [SearchDepsBindingsModule::class])
interface MainActivityComponent : SearchDeps {
    fun inject(activity: MainActivity)
}

class MainActivity : Activity(), HasDependencies {
    @Inject
    override lateinit var depsMap: DepsMap

    override fun onCreate(…) {
        DaggerMainActivityComponent.factory().create().inject(this)
    }
}
```

Реализация зависимостей

```
@dagger.Component(modules = [SearchDepsBindingsModule::class])
interface MainActivityComponent : SearchDeps {
    fun inject(activity: MainActivity)
}

class MainActivity : Activity(), HasDependencies {
    @Inject
    override lateinit var depsMap: DepsMap

    override fun onCreate(…) {
        DaggerMainActivityComponent.factory().create().inject(this)
    }
}
```

Настало время подвести итоги

〉Узлы иерархии реализуют HasDependencies

Application: HasDependencies

Activity: HasDependencies

FragmentK: HasDependencies

Настало время подвести итоги

〉Узлы иерархии реализуют HasDependencies

Application: HasDependencies

Activity: HasDependencies

FragmentK: HasDependencies

Настало время подвести итоги

- Узлы иерархии реализуют HasDependencies
- **@Inject override var**
- Component extends Dependencies
- @Binds @IntoMap Class<Dependency> -> Component

Настало время подвести итоги

〉 Узлы иерархии реализуют HasDependencies

〉 @Inject override var

〉 **Component реализует Dependencies**

Настало время подвести итоги

- Application/Activity/Fragment extends HasDependencies
- @Inject override
- Component реализует Dependencies
- **@Binds @IntoMap Class<Dependency> → Component**

Вспомним план

- Модуляризация
- Требования к межмодульному DI
- Получение зависимостей
- Предоставление зависимостей
- **Использование зависимостей**
- Минусы / ограничения / исключения

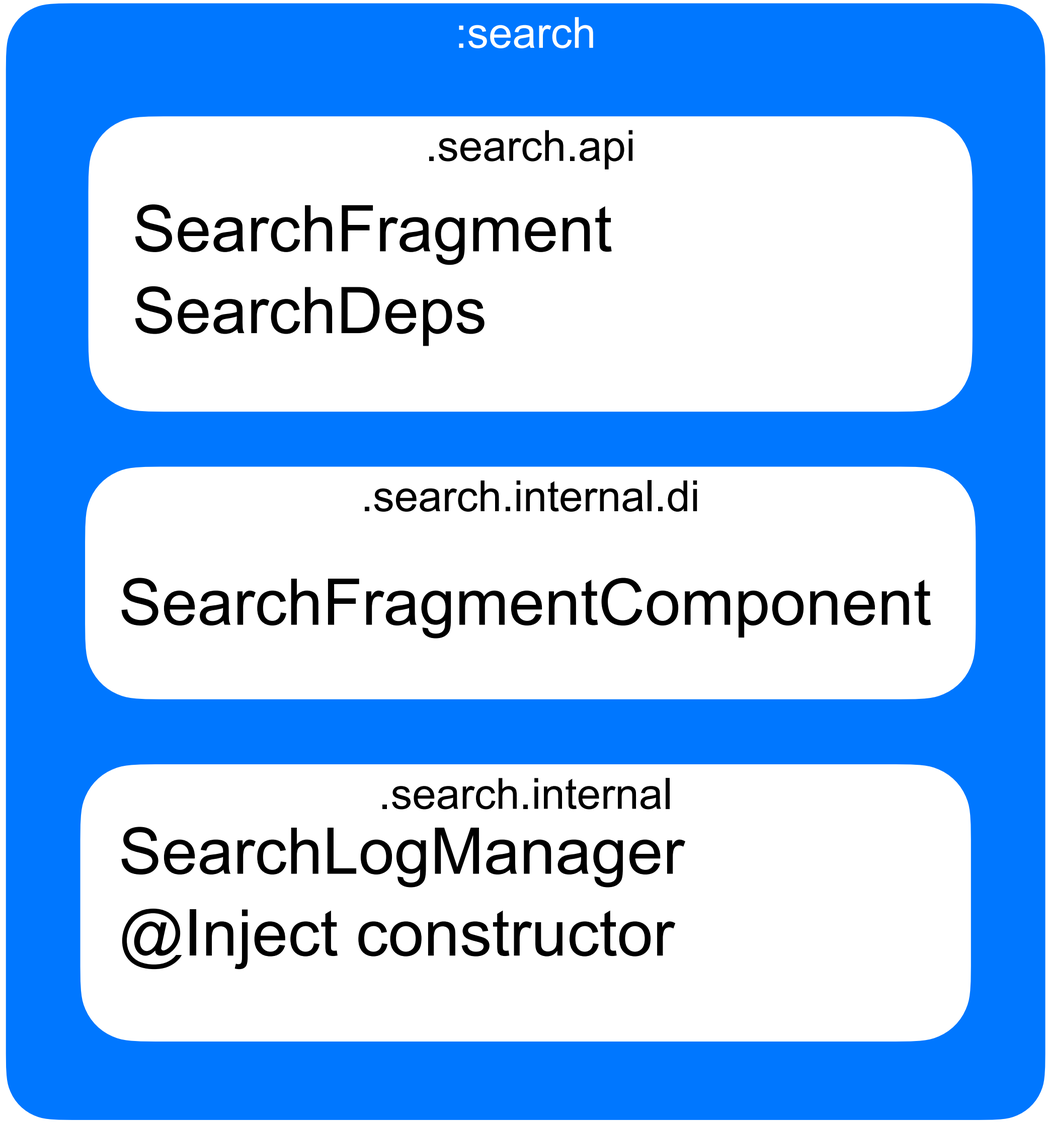
Использование зависимостей

Как во фрагменте использовать зависимости?

```
findDependencies<FeatureDeps>()
```


ВСЁ!

Внутренний граф



Внутренний граф

```
@Component
internal interface SearchFragmentComponent {
    @Component.Factory
    interface Factory {
        fun create(@BindsInstance deps: SearchDeps)
            : SearchFragmentComponent
    }
    fun inject(fragment: SearchFragment)
}

class SearchFragment : Fragment(){
    @Inject internal lateinit var logManager: SearchLogManager
    override fun onCreate(…) {
        DaggerSearchFragmentComponent.factory()
                .create(findDependencies())
                .inject(this)
        logManager.writeLog()
    }
}
```

Внутренний граф

```
@Component
internal interface SearchFragmentComponent {
    @Component.Factory
    interface Factory {
        fun create(@BindsInstance deps: SearchDeps)
            : SearchFragmentComponent
    }
    fun inject(fragment: SearchFragment)
}

class SearchFragment : Fragment(){
    @Inject internal lateinit var logManager: SearchLogManager
    override fun onCreate(…) {
        DaggerSearchFragmentComponent.factory()
                .create(findDependencies())
                .inject(this)
        logManager.writeLog()
    }
}
```

Внутренний граф

```
@Component
internal interface SearchFragmentComponent {
    @Component.Factory
    interface Factory {
        fun create(@BindsInstance deps: SearchDeps)
            : SearchFragmentComponent
    }
    fun inject(fragment: SearchFragment)
}

class SearchFragment : Fragment(){
    @Inject internal lateinit var logManager: SearchLogManager
    override fun onCreate(…) {
        DaggerSearchFragmentComponent.factory()
                .create(findDependencies())
                .inject(this)
        logManager.writeLog()
    }
}
```

Внутренний граф

```
@Component
internal interface SearchFragmentComponent {
    @Component.Factory
    interface Factory {
        fun create(@BindsInstance deps: SearchDeps)
            : SearchFragmentComponent
    }
    fun inject(fragment: SearchFragment)
}

class SearchFragment : Fragment(){
    @Inject internal lateinit var logManager: SearchLogManager
    override fun onCreate(…) {
        DaggerSearchFragmentComponent.factory()
                .create(findDependencies())
                .inject(this)
        logManager.writeLog()
    }
}
```

Использование зависимостей

```
internal class SearchLogManager @Inject constructor(
    private val deps: SearchDeps
) {
    fun writeLog() {
        deps.logger.log("...")
    }
}
```

Использование зависимостей

```
internal class SearchLogManager @Inject constructor(
    private val deps: SearchDeps,
    private val internalLogger: InternalLogger,
) {
    fun writeLog() {
        deps.logger.log(“…")
        internalLogger.log(“...")
    }
}
```

Внутренний граф

```
@Component(dependencies = [SearchDeps::class])
internal interface SearchFragmentComponent {
    @Component.Factory
    interface Factory {
        fun create(deps: SearchDeps): SearchFragmentComponent
    }

    fun inject(fragment: SearchFragment)
}
```

Внутренний граф

```
@Component(dependencies = [SearchDeps::class])
internal interface SearchFragmentComponent {
    @Component.Factory
    interface Factory {
        fun create(deps: SearchDeps): SearchFragmentComponent
    }

    fun inject(fragment: SearchFragment)
}


interface SearchDeps: Dependencies {
    val logger: Logger
}
```

Внутренний граф

```
@Component(dependencies = [SearchDeps::class])
internal interface SearchFragmentComponent {
    @Component.Factory
    interface Factory {
        fun create(deps: SearchDeps): SearchFragmentComponent
    }

    fun inject(fragment: SearchFragment)
}


interface SearchDeps: Dependencies {
    val logger: Logger
}
```

Использование зависимостей

```
internal class SearchManager @Inject constructor(
    private val logger: Logger,
    private val internalLogger: InternalLogger,
) {
    fun writeLog() {
        logger.log(“…")
        internalLogger.log(“...")
    }
}
```

Настало время подвести итоги

- Отдельный внутренний граф
- Component dependencies

Вспомним план

- Модуляризация
- Требования к межмодульному DI
- Получение зависимостей
- Предоставление зависимостей
- Использование зависимостей
- **Минусы / ограничения / исключения**

Минусы
Ограничения
Исключения

Связывание во время выполнения

Связывание во время выполнения

```
inline fun <reified D: Dependencies> Fragment.findDependencies(): D {
    return parents
        .mapNotNull { it.depsMap[D::class.java] }
        .firstOrNull() as D?
        ?: throw IllegalStateException("No ${D::class.java} in $parents")
}
```

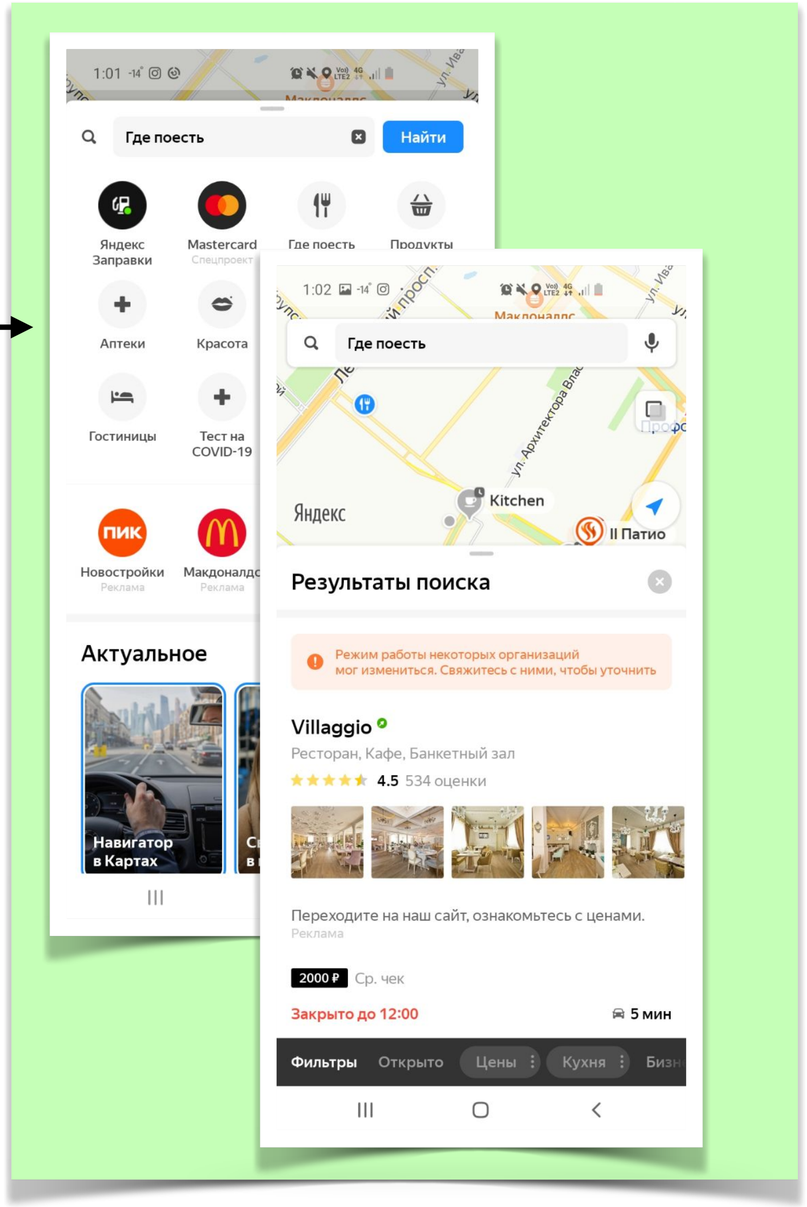
```
findDependencies<SearchDeps>()
```

:search

.search.api

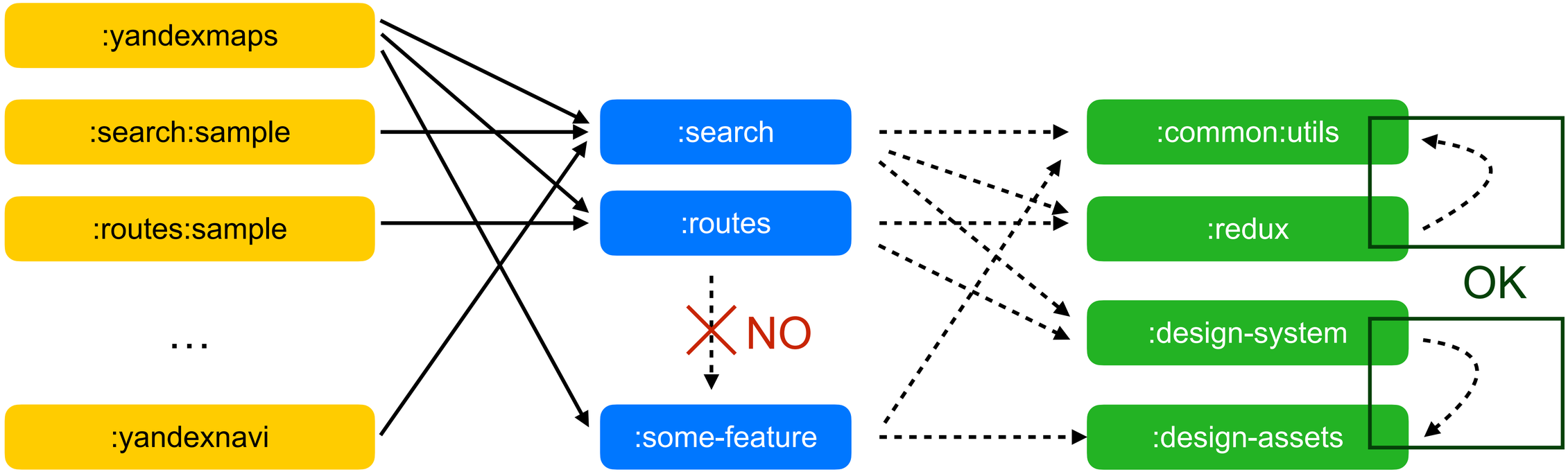
SearchFragment

SearchDeps



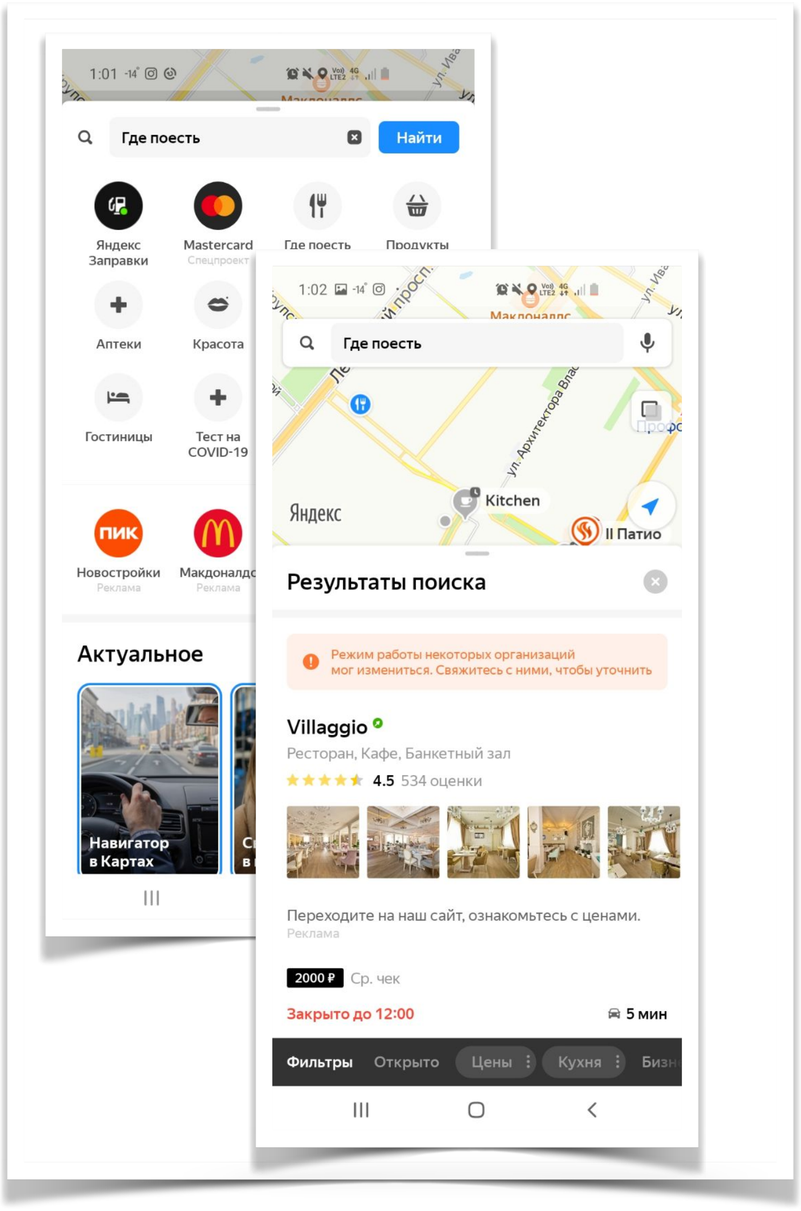
Ограничения на зависимость фичей друг от друга

〉App modules 〉Feature modules 〉Core modules



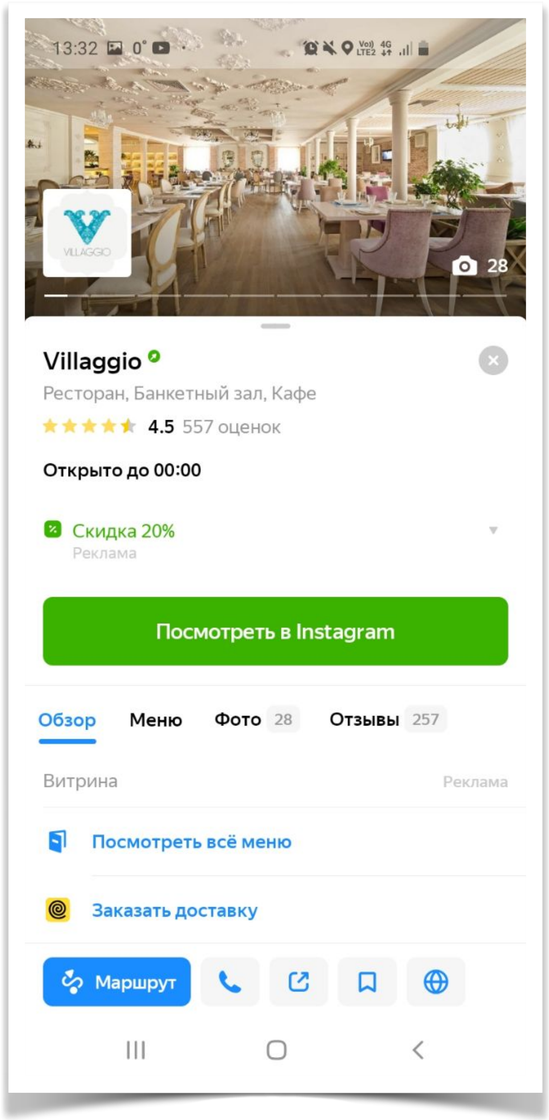
Зависимости большой фичи

```
interface SearchDependencies : Dependencies {
    val searchHistoryService: SearchHistoryService
    val search: Search
    val map: Map
    val camera: Camera
    val searchLocationService: SearchLocationService
    val searchBannersConfigService: SearchBannersConfigService
    val application: Application
    val refWatcher: RefWatcherWrapper?
    val externalSearchPreferences: ExternalSearchPreferences
    val categoriesProvider: CategoriesProvider
    val searchLayer: SearchLayer
    val searchRecognizer: SearchRecognizer
    val searchExternalNavigator: SearchExternalNavigator
    val routeSerpSearchClickListener: RouteSerpSearchClickListener
    val searchStateMutator: SearchStateMutator
    val searchExitStrategy: SearchExitStrategy
    val searchResultCardProvider: SearchResultCardProvider<*>
    val mtThreadCardControllerProvider: MtThreadCardControllerProvider<*>
    val mtStopCardControllerProvider: MtStopCardControllerProvider<*>
    val offerProvider: MastercardOfferProvider
    val moshi: Moshi
    val viewPool: PrefetchRecycledViewPool
    val prefetcherManager: SnippetPrefetcherManager
    val fluidContainerShoreSupplier: FluidContainerShoreSupplier
    val purse: Purse
    val facebookLogger: SearchFacebookLogger?
    val snippetFactory: SnippetFactory
    val experimentsProvider: SearchExperimentsProvider
    val searchCallbacks: SearchControllerCallbacks
    val showcaseLookupService: ShowcaseLookupService
    val transportOverlayTemporaryDisabler: SearchTransportOverlayTemporaryDisabler
    val keyboardManager: KeyboardManager
    val contextProvider: ActivityContextProvider
    val modulePlacement: ModulePlacement
    val cameraMovementController: SearchCameraController
    val searchOptionsFactory: SearchOptionsFactory
    val searchLineExternalInteractor: SearchLineExternalInteractor
}
```

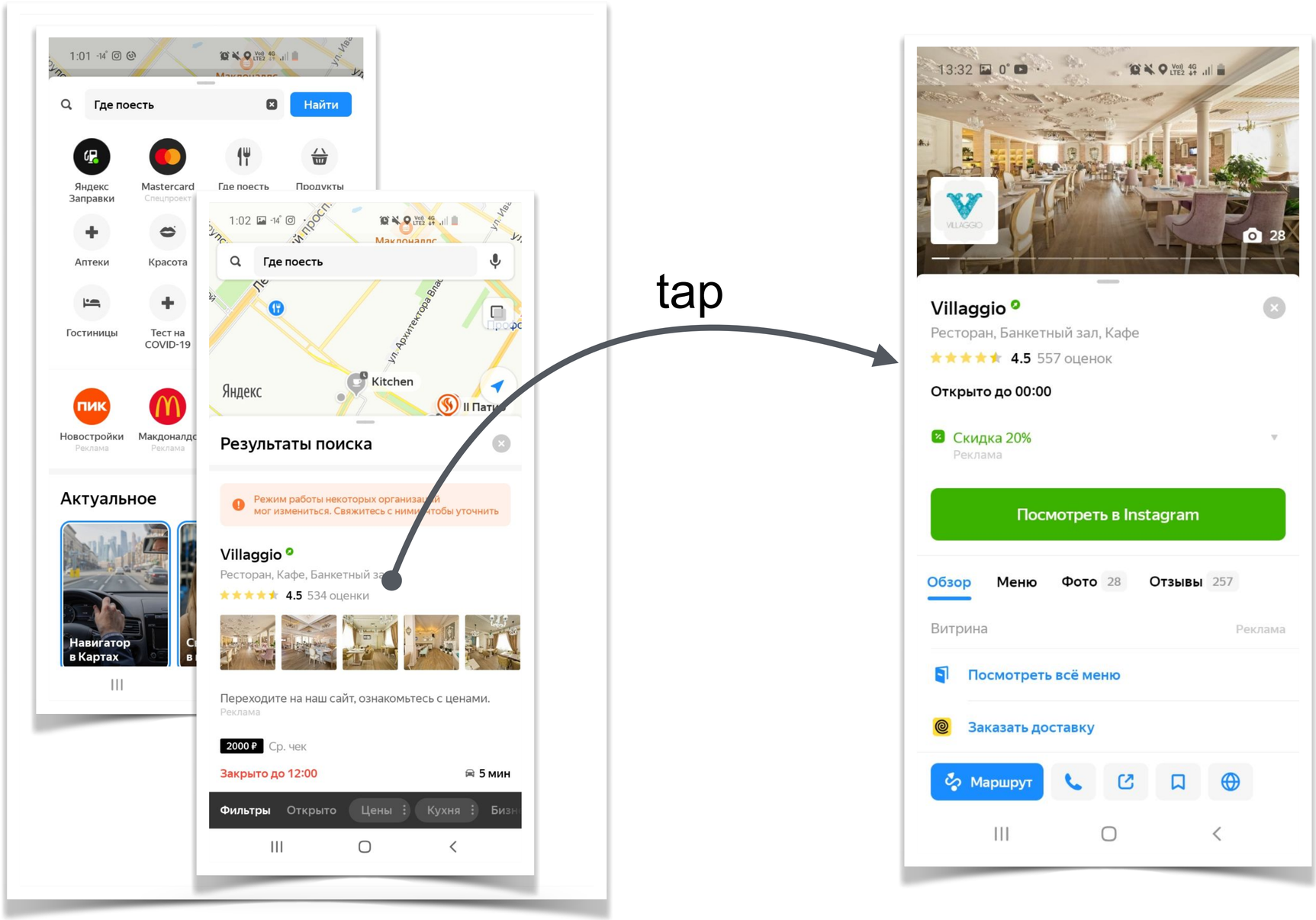


Зависимости другой большой фичи

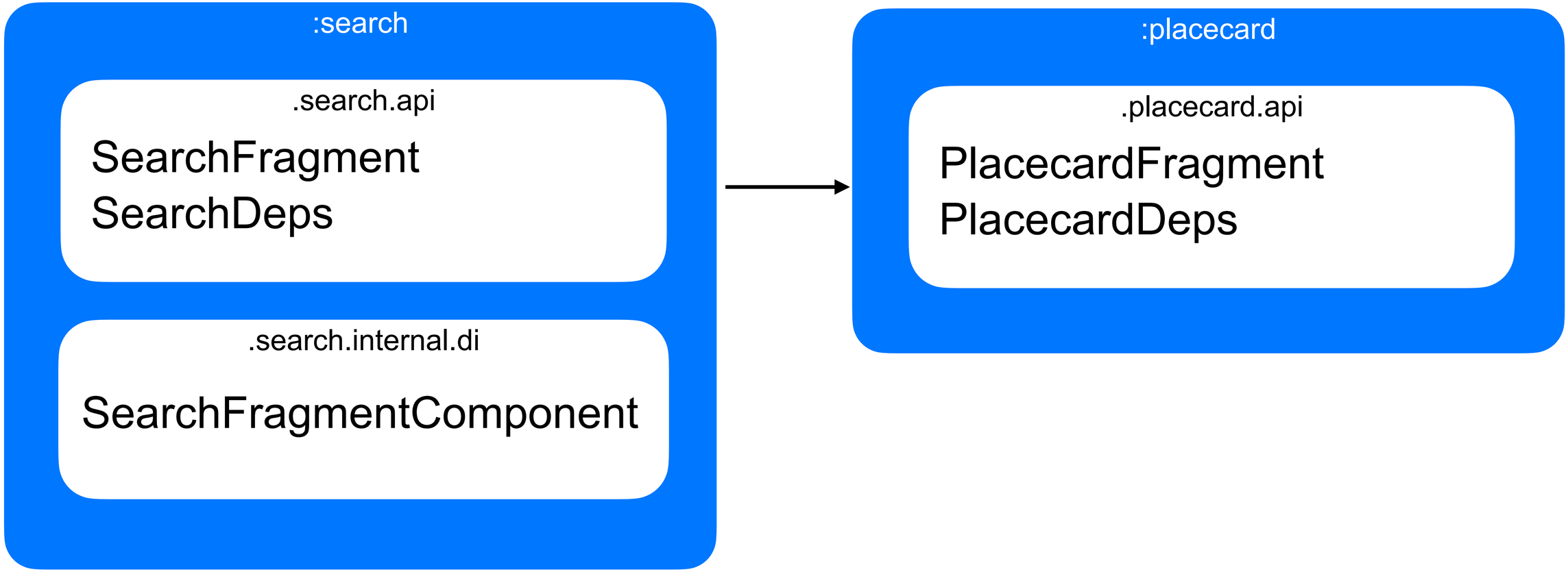
```
interface PlacecardDependencies: Dependencies {
    val externalNavigator: GeoObjectPlacecardExternalNavigator
    val ratingBlockEpicsNavigator: PlacecardRatingBlockNavigator
    val compositingStrategy: PlacecardListCompositingStrategy
    val reviewsService: ReviewsService
    val reviewReactionsService: ReviewReactionsService
    val myReviewsService: MyReviewsService
    val entrancesCommander: EntrancesCommander
    val reviewsAuthService: ReviewsAuthService
    val entrancesCameraOperator: EntrancesCameraOperator
    val purse: Purse
    val photosAuthService: PhotosAuthService
    val photoUploadManager: PhotoUploadManager
    val distanceInfoFormatter: DistanceInfoFormatter
    val cardGeoObjectRegistry: CardGeoObjectRegistry
    val pinVisibilityChecker: PinVisibilityChecker
    val mapCameraLock: MapCameraLock
    val preferencesFactory: PreferencesFactoryProvider
    val geoObjectCallbacks: GeoObjectPlacecardControllerCallbacks
    val experimentManager: PlacecardExperimentManager
    val debugPreferencesProvider: PlacecardDebugPreferencesProvider
    val indoorLevelUpdater: PlacecardIndoorLevelUpdater
    val contextProvider: ActivityContextProvider
    val suggestCategoriesProvider: ToponymSuggestCategoriesProvider
    val searchOptionsFactory: SearchOptionsFactory
    val eventFetcher: EventFetcher
    val placecardLoggingParametersProvider: PlacecardLoggingParametersProvider
    val modulePlacement: ModulePlacement
    val tabsProvider: PlacecardExternalTabsProvider
    val taxiInfo: TaxiInfoService
    val taxiApplicationManager: TaxiApplicationManager
    val taxiNavigationManager: TaxiNavigationManager
    val storiesService: StoriesService
    val storiesStorage: StoriesStorage
    val shareMessageProvider: ShareMessageProvider
    val feedbackQueriesFactory: FeedbackWebQueriesFactory
    val camera: Camera
    val carsharingManager: CarsharingManager
    val carsharingApplicationManager: CarsharingApplicationManager
    val placecardDebugSettings: PlacecardDebugSettings
    val placecardComposingSettings: PlacecardComposingSettings
}
```



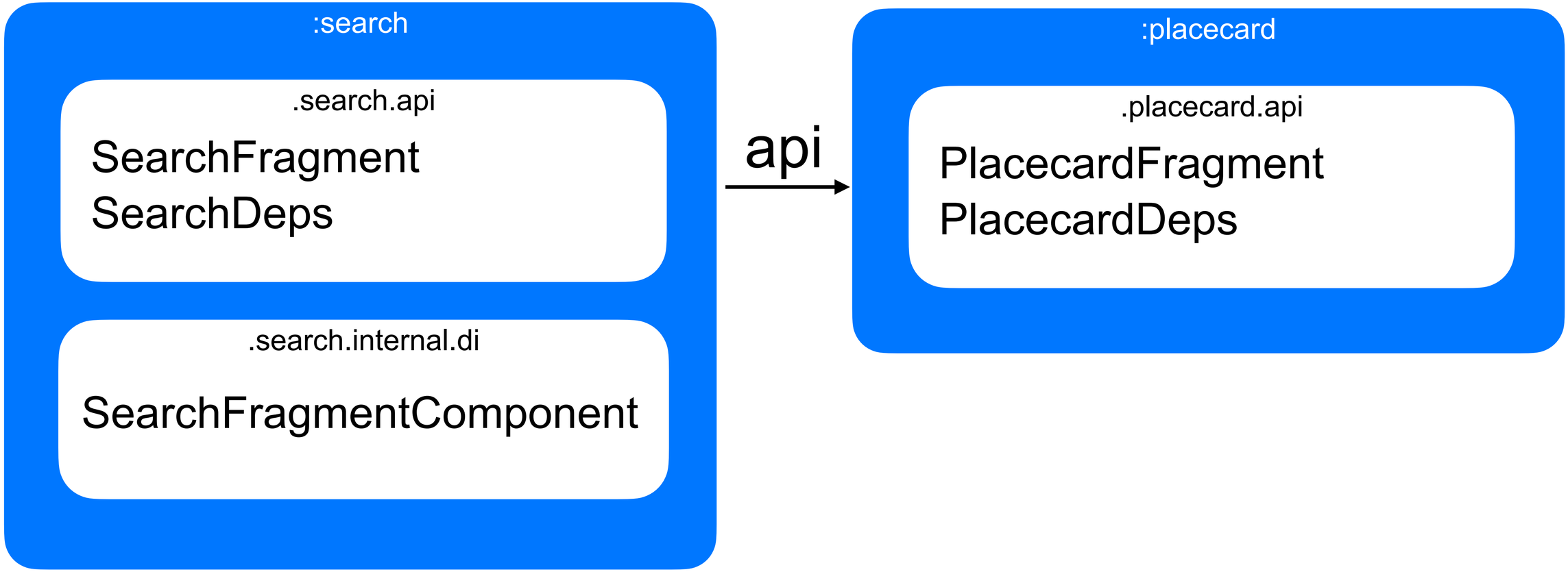
Зависимость одной фичи от другой



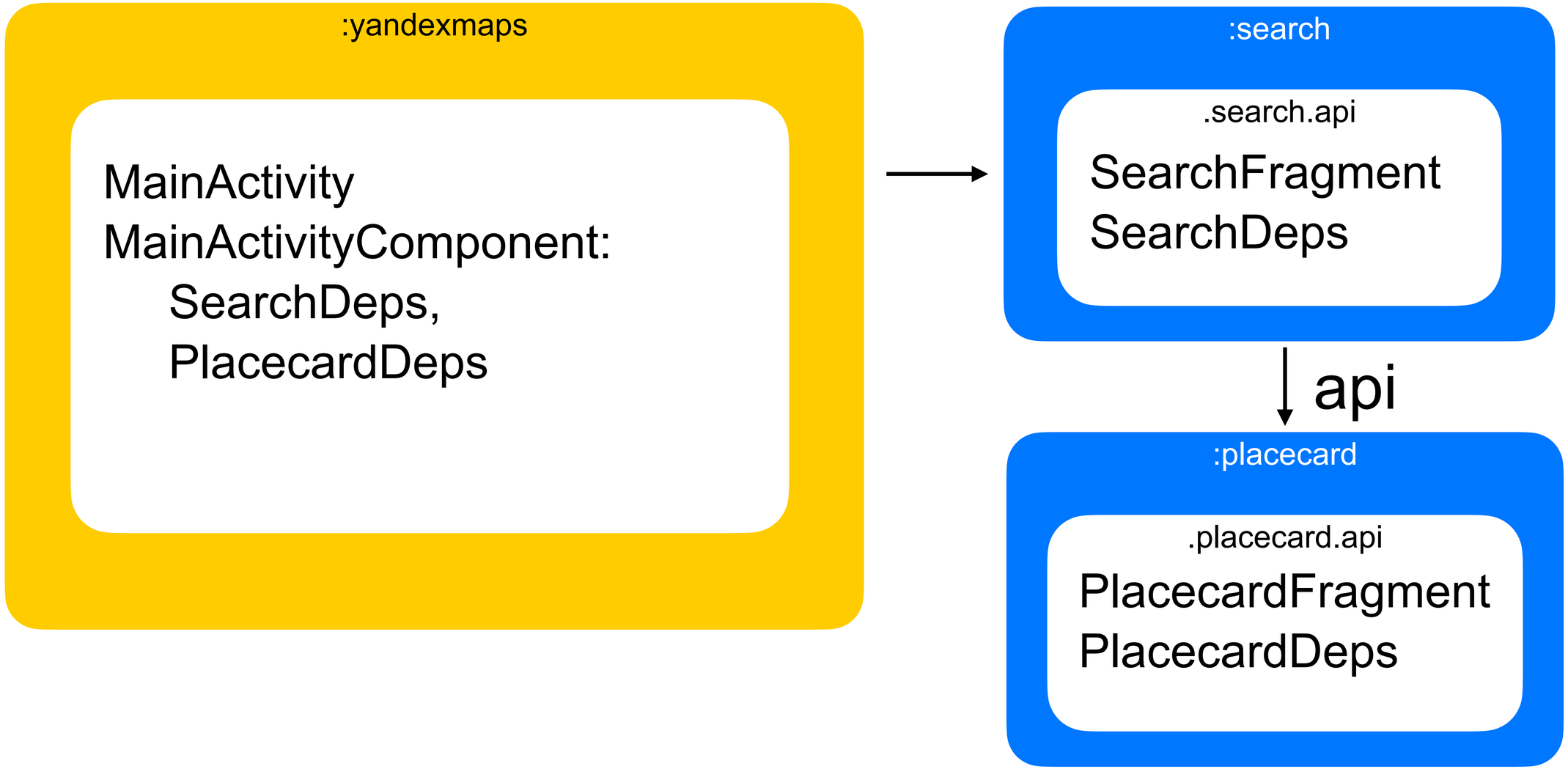
Зависимость одной фичи от другой



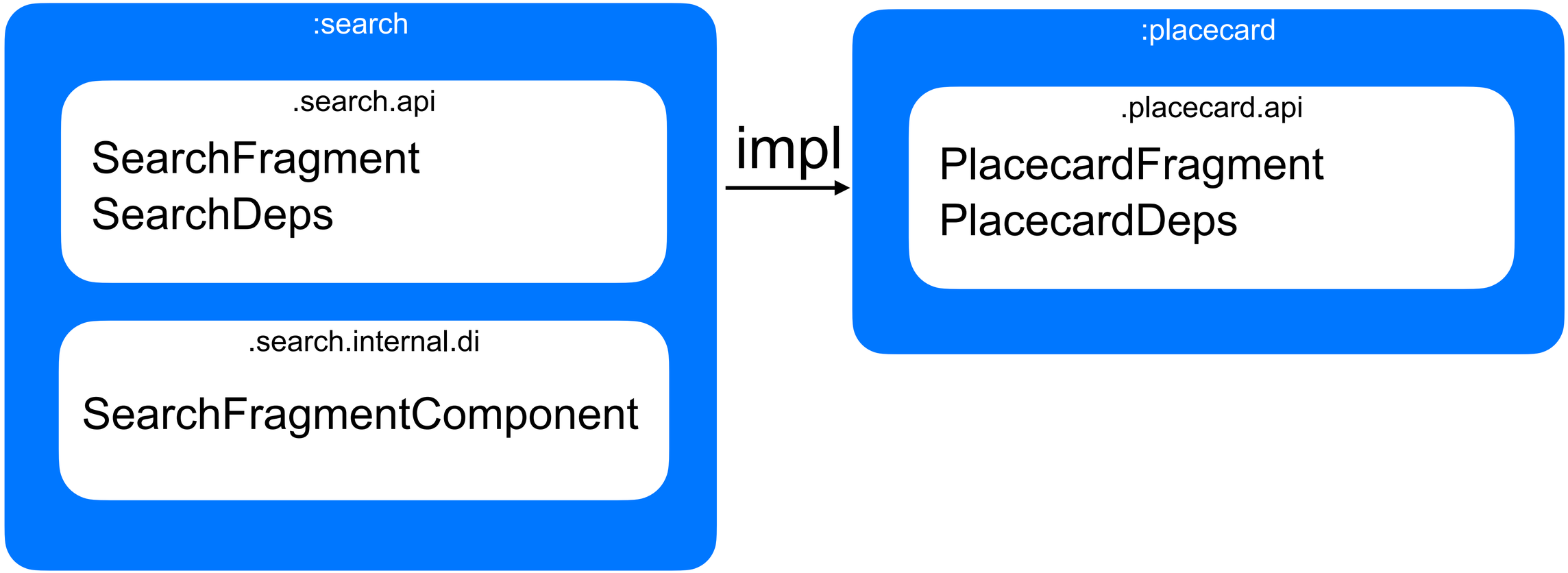
Зависимость одной фичи от другой по api



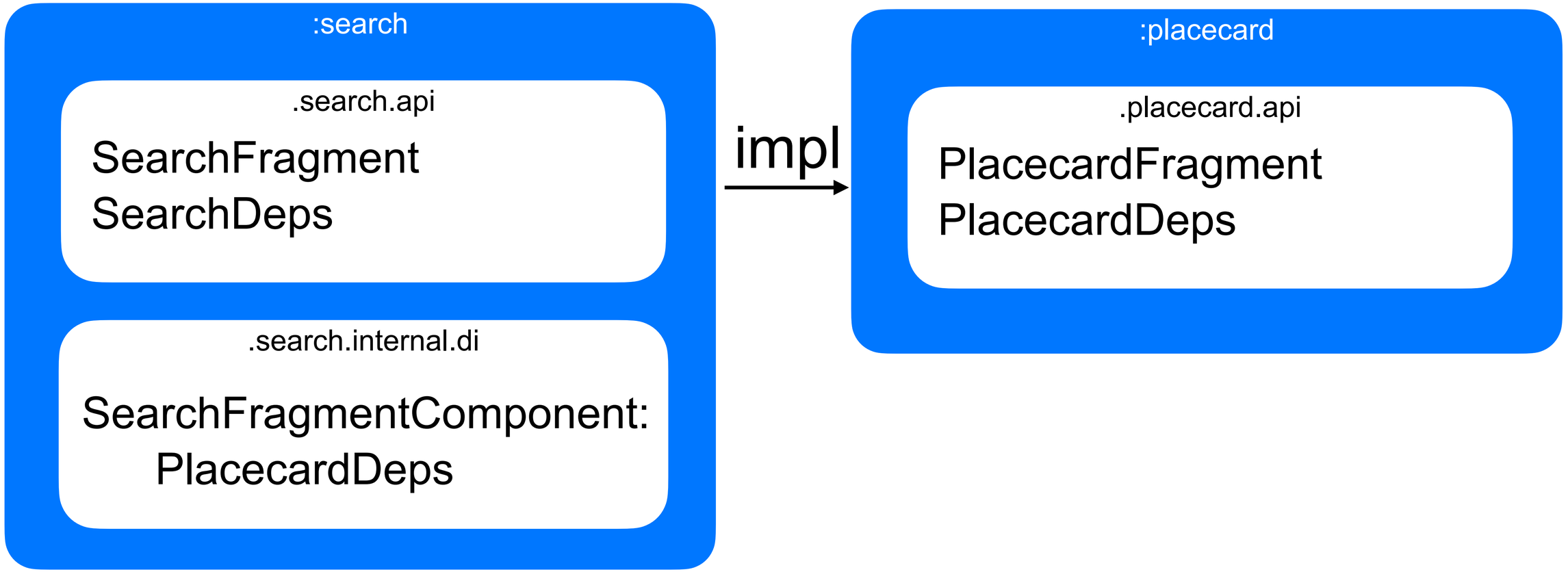
Зависимость одной фичи от другой по api



Зависимость одной фичи от другой по implementation



Зависимость одной фичи от другой по implementation

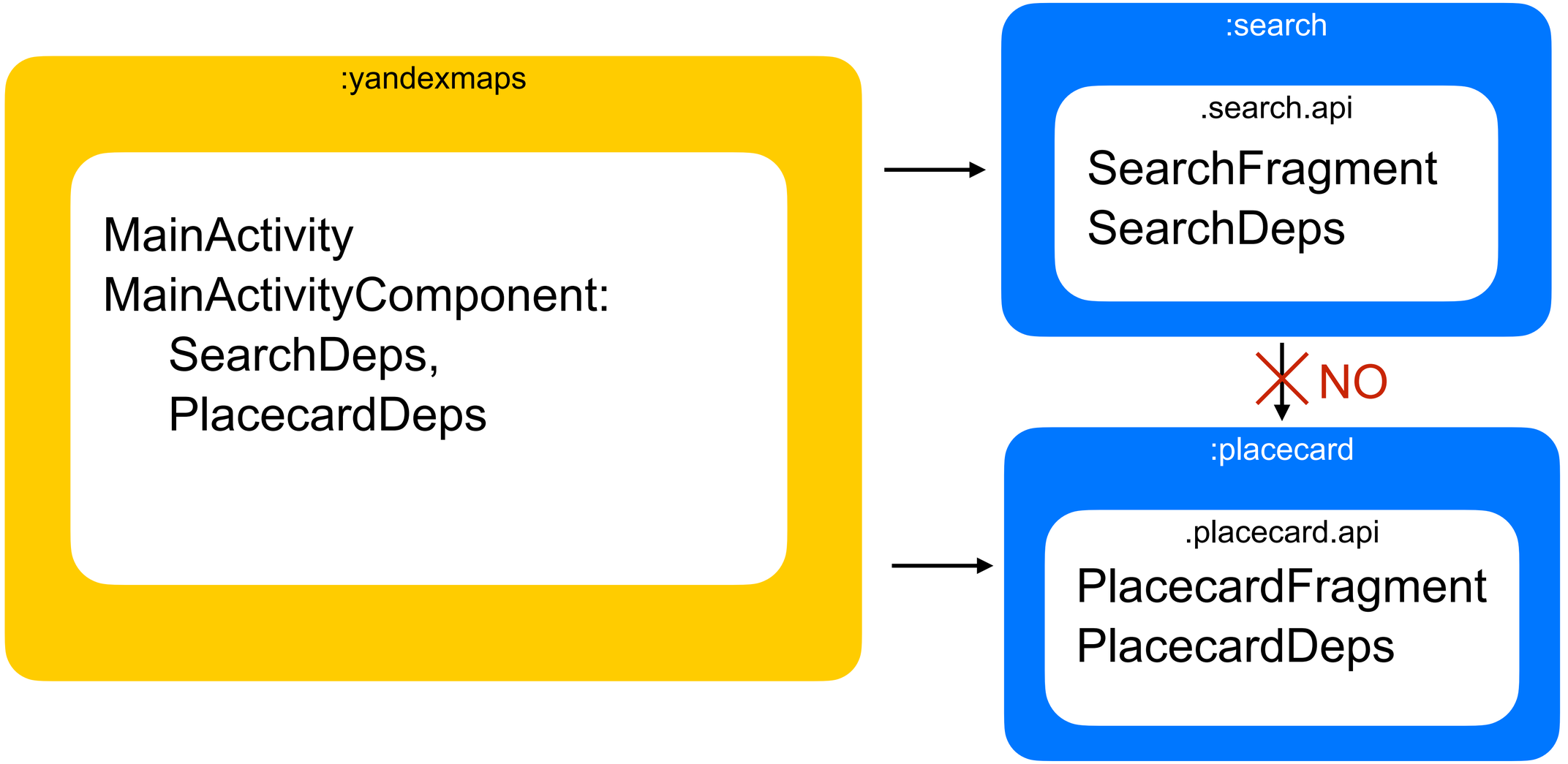


Зависимость одной фичи от другой по implementation

```
interface SearchDependencies : Dependencies {
    val searchHistoryService: SearchHistoryService
    val search: Search
    val map: Map
    val camera: Camera
    val searchLocationService: SearchLocationService
    val searchBannersConfigService: SearchBannersConfigService
    val application: Application
    val refWatcher: RefWatcherWrapper?
    val externalSearchPreferences: ExternalSearchPreferences
    val categoriesProvider: CategoriesProvider
    val searchLayer: SearchLayer
    val searchRecognizer: SearchRecognizer
    val searchExternalNavigator: SearchExternalNavigator
    val routeSerpSearchClickListener: RouteSerpSearchClickListener
    val searchStateMutator: SearchStateMutator
    val searchExitStrategy: SearchExitStrategy
    val searchResultCardProvider: SearchResultCardProvider<*>
    val mtThreadCardControllerProvider: MtThreadCardControllerProvider<*>
    val mtStopCardControllerProvider: MtStopCardControllerProvider<*>
    val offerProvider: MastercardOfferProvider
    val moshi: Moshi
    val viewPool: PrefetchRecycledViewPool
    val prefetcherManager: SnippetPrefetcherManager
    val fluidContainerShoreSupplier: FluidContainerShoreSupplier
    val purse: Purse
    val facebookLogger: SearchFacebookLogger?
    val snippetFactory: SnippetFactory
    val experimentsProvider: SearchExperimentsProvider
    val searchCallbacks: SearchControllerCallbacks
    val showcaseLookupService: ShowcaseLookupService
    val transportOverlayTemporaryDisabler: SearchTransportOverlayTemporaryDisabler
    val keyboardManager: KeyboardManager
    val contextProvider: ActivityContextProvider
    val modulePlacement: ModulePlacement
    val cameraMovementController: SearchCameraController
    val searchOptionsFactory: SearchOptionsFactory
    val searchLineExternalInteractor: SearchLineExternalInteractor
}
```

```
interface PlacecardDependencies: Dependencies {
    val externalNavigator: GeoObjectPlacecardExternalNavigator
    val ratingBlockEpicsNavigator: PlacecardRatingBlockNavigator
    val compositingStrategy: PlacecardListCompositingStrategy
    val reviewsService: ReviewsService
    val reviewReactionsService: ReviewReactionsService
    val myReviewsService: MyReviewsService
    val entrancesCommander: EntrancesCommander
    val reviewsAuthService: ReviewsAuthService
    val entrancesCameraOperator: EntrancesCameraOperator
    val purse: Purse
    val photosAuthService: PhotosAuthService
    val photoUploadManager: PhotoUploadManager
    val distanceInfoFormatter: DistanceInfoFormatter
    val cardGeoObjectRegistry: CardGeoObjectRegistry
    val pinVisibilityChecker: PinVisibilityChecker
    val mapCameraLock: MapCameraLock
    val preferencesFactory: PreferencesFactoryProvider
    val geoObjectCallbacks: GeoObjectPlacecardControllerCallbacks
    val experimentManager: PlacecardExperimentManager
    val debugPreferencesProvider: PlacecardDebugPreferencesProvider
    val indoorLevelUpdater: PlacecardIndoorLevelUpdater
    val contextProvider: ActivityContextProvider
    val suggestCategoriesProvider: ToponymSuggestCategoriesProvider
    val searchOptionsFactory: SearchOptionsFactory
    val eventFetcher: EventFetcher
    val placecardLoggingParametersProvider: PlacecardLoggingParametersProvider
    val modulePlacement: ModulePlacement
    val tabsProvider: PlacecardExternalTabsProvider
    val taxiInfo: TaxiInfoService
    val taxiApplicationManager: TaxiApplicationManager
    val taxiNavigationManager: TaxiNavigationManager
    val storiesService: StoriesService
    val storiesStorage: StoriesStorage
    val shareMessageProvider: ShareMessageProvider
    val feedbackQueriesFactory: FeedbackWebQueriesFactory
    val camera: Camera
    val carsharingManager: CarsharingManager
    val carsharingApplicationManager: CarsharingApplicationManager
    val placecardDebugSettings: PlacecardDebugSettings
    val placecardComposingSettings: PlacecardComposingSettings
}
```

Разрываем зависимость между фичами



Разрываем зависимость между фичами

:search:api

```
interface PlacecardProvider {
    fun placecard(data: GeoObject): Fragment
}
```

Разрываем зависимость между фичами

:search:api

```
interface PlacecardProvider {
    fun placecard(data: GeoObject): Fragment
}
interface SearchDeps {
    …
    val placecardProvider: PlacecardProvider
}
```

Разрываем зависимость между фичами

:yandexmaps

```
class PlacecardProviderImpl: PlacecardProvider {
    override fun placecard(data: GeoObject)
            = PlacecardFragment.newInstance(data)
}
```

Разрываем зависимость между фичами

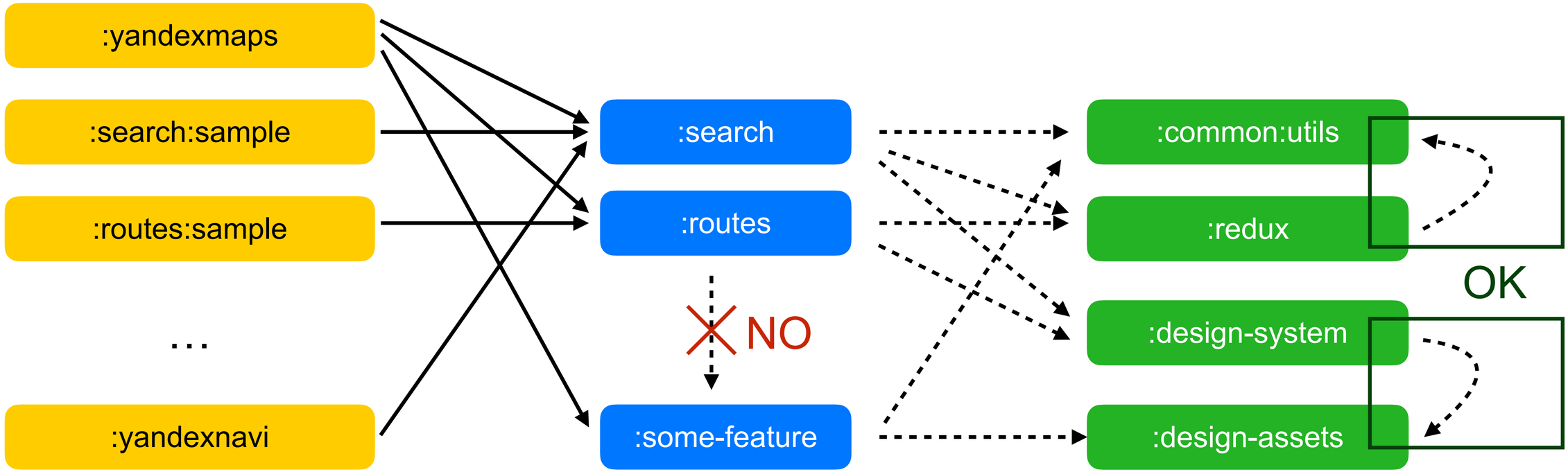
:yandexmaps

```
class PlacecardProviderImpl: PlacecardProvider {
    override fun placecard(data: GeoObject)
            = PlacecardFragment.newInstance(data)
}
```

```
@dagger.Component(modules = […])
interface MainActivityComponent
   : SearchDeps, PlacecardDeps
{
    fun inject(activity: MainActivity)
}
```

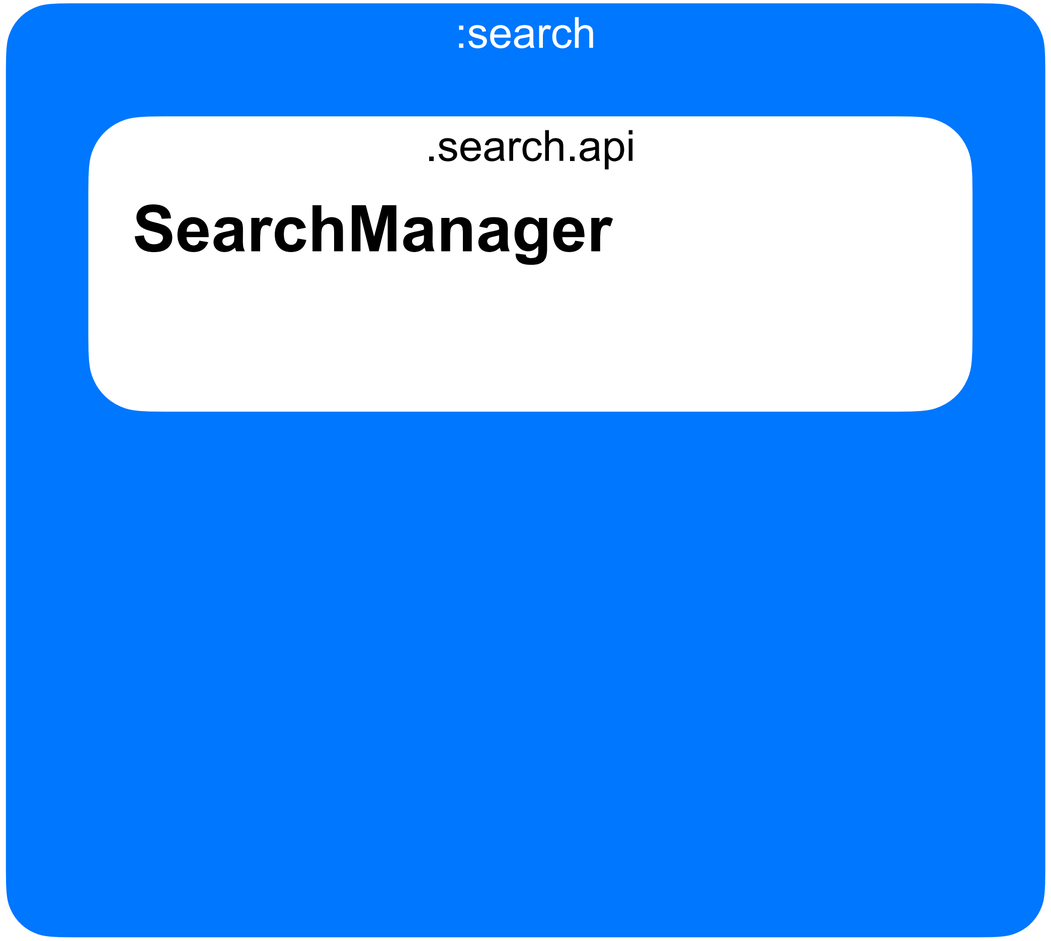
Ограничения на зависимость фичей друг от друга

〉App modules 〉Feature modules 〉Core modules

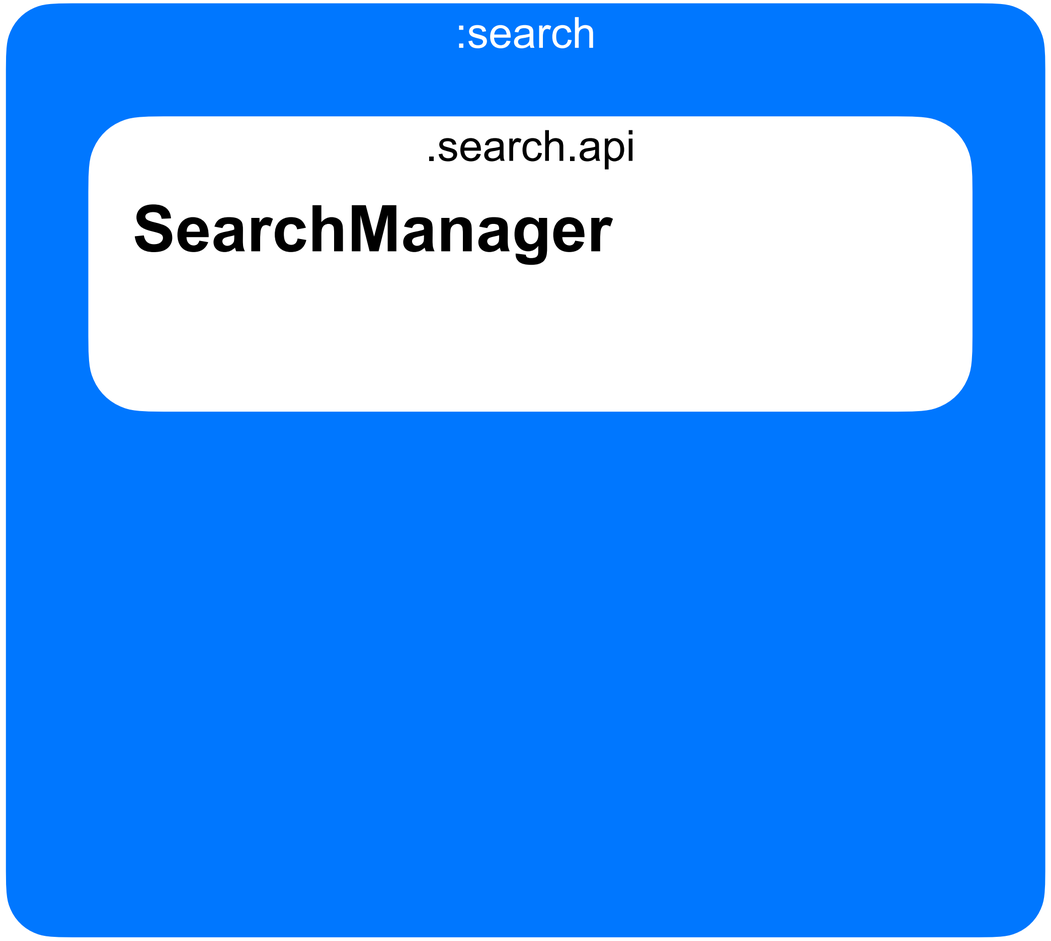


Не UI фича

Не UI фича

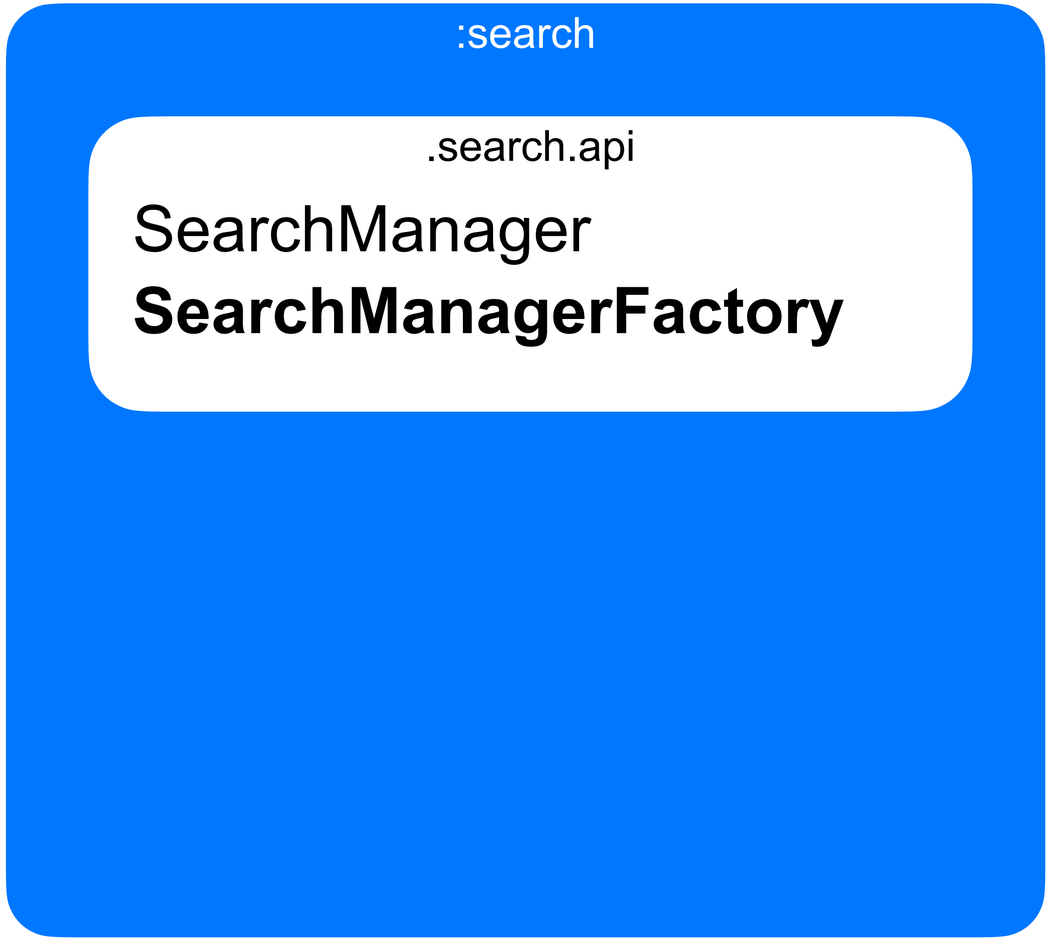


Не UI фича



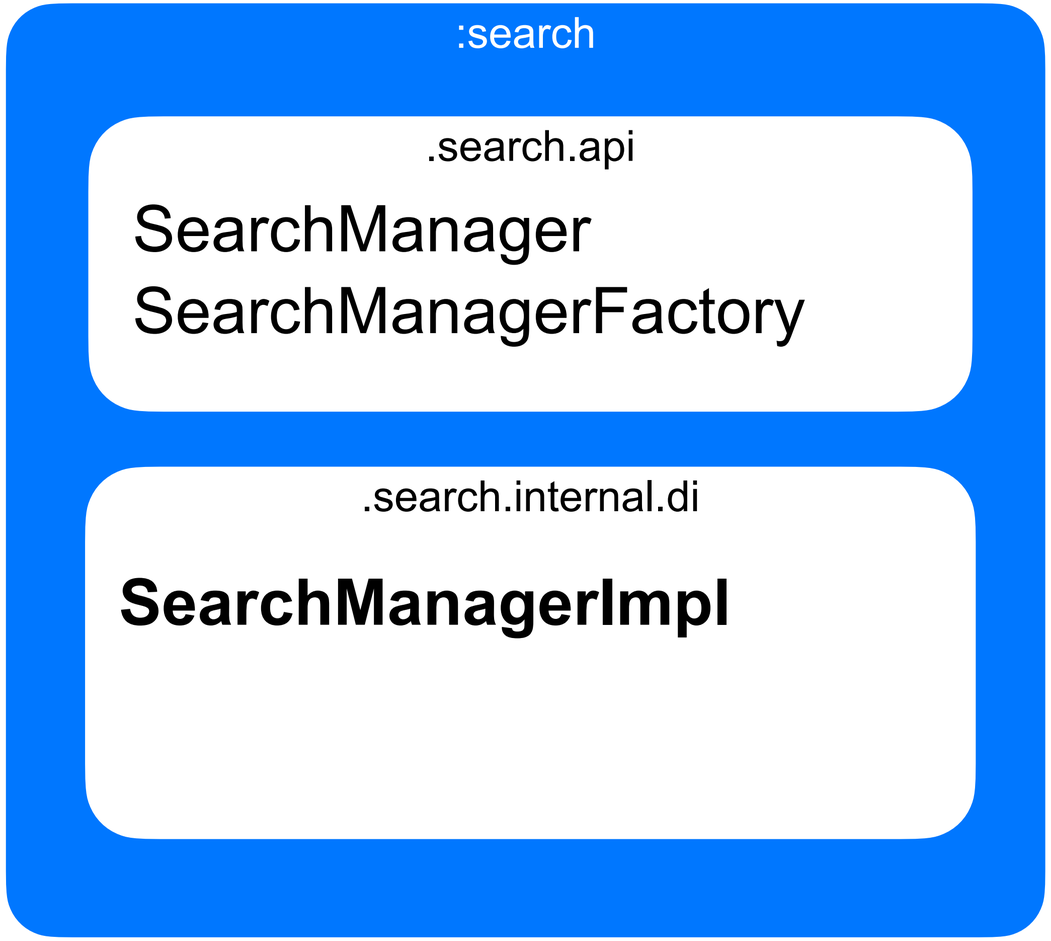

```
interface SearchManager {
    fun search(text: String)
}
```

Не UI фича



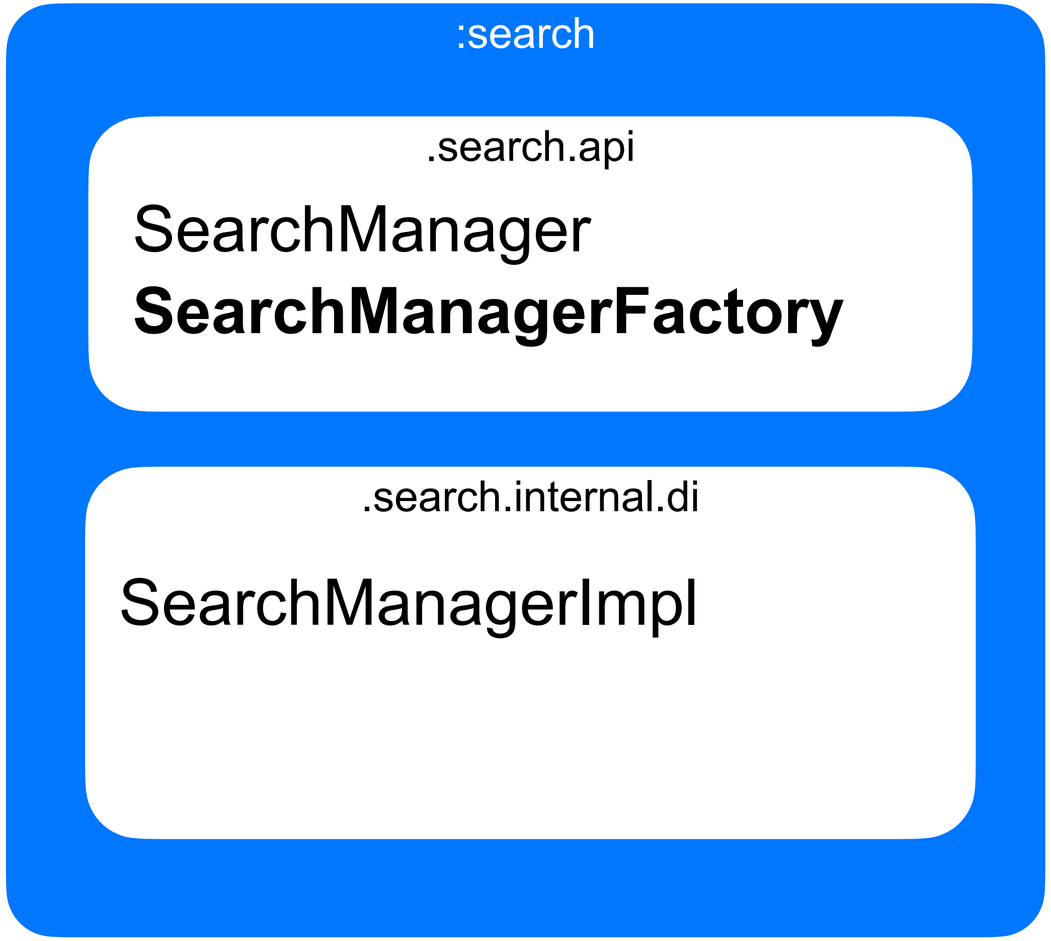

```
interface SearchManager {
    fun search(text: String)
}

object SearchManagerFactory {
    fun create(
        …
    ): SearchManager = …
}
```

Не UI фича



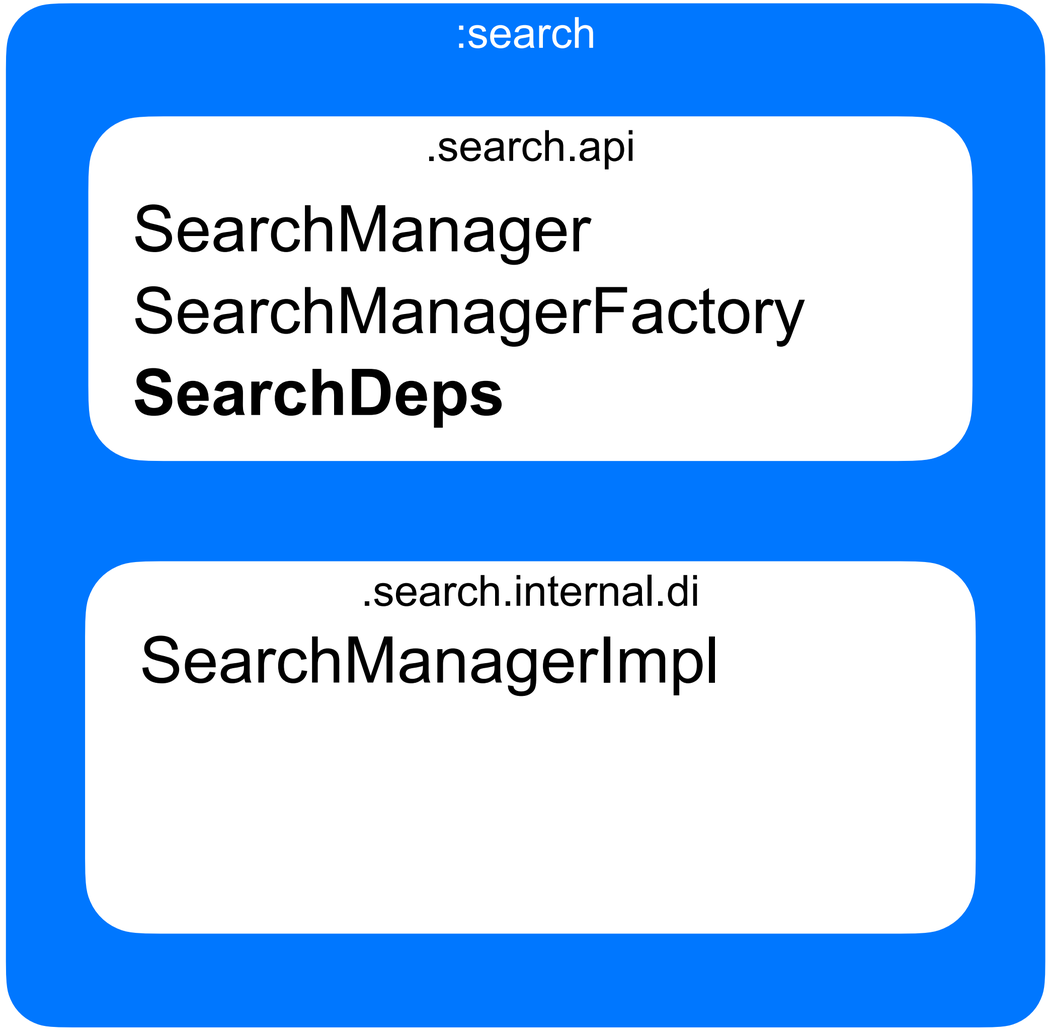

```
interface SearchManager {
    fun search(text: String)
}

internal class SearchManagerImpl(
        val logger: Logger,
        val format: Formatter,
        …
): SearchManager {
    override fun search(…) {
        …
    }

}
```

Не UI фича



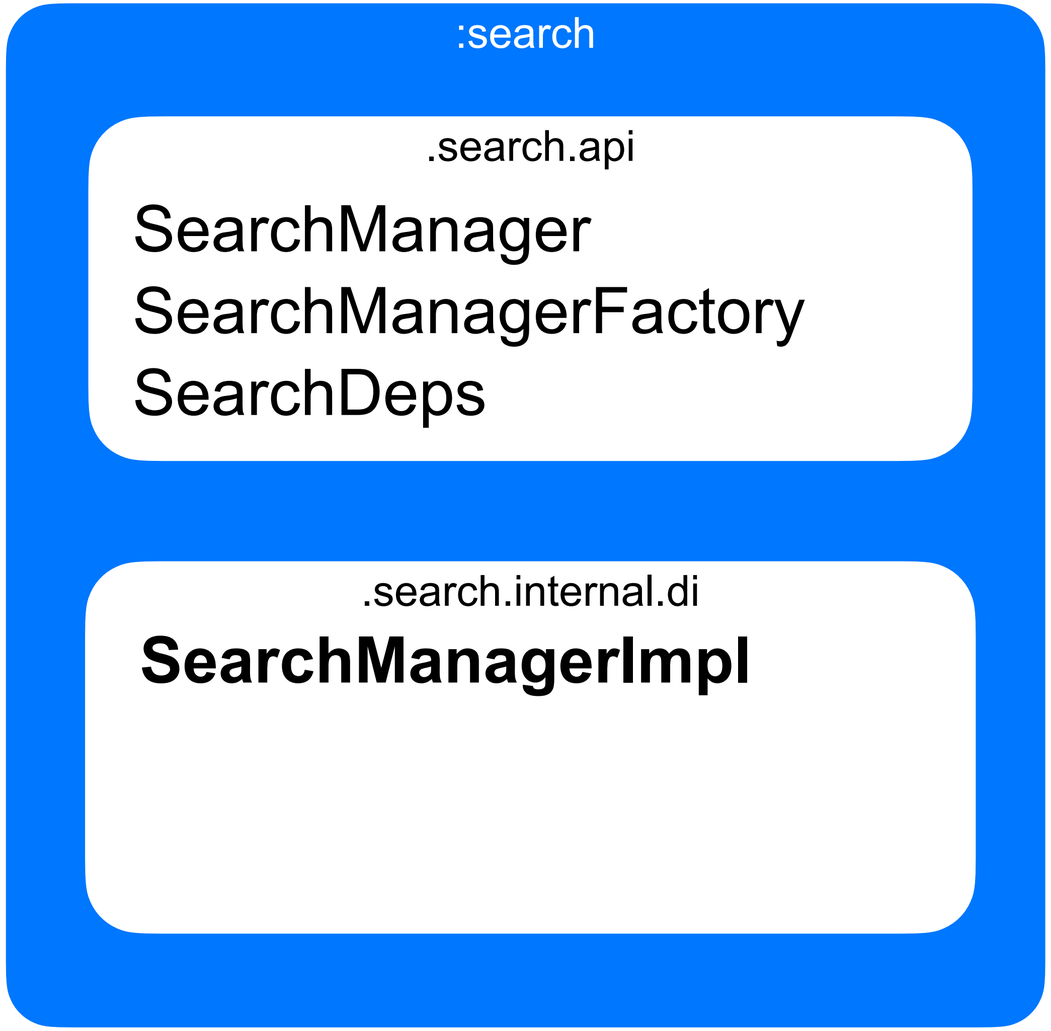

```
interface SearchManager {
    fun search(text: String)
}

object SearchManagerFactory {
    fun create(
            logger: Logger,
            format: Formatter,
            …
    ): SearchManager {
        return SearchManagerImpl(
            logger, format, …
        )
    }
}
```

Не UI фича



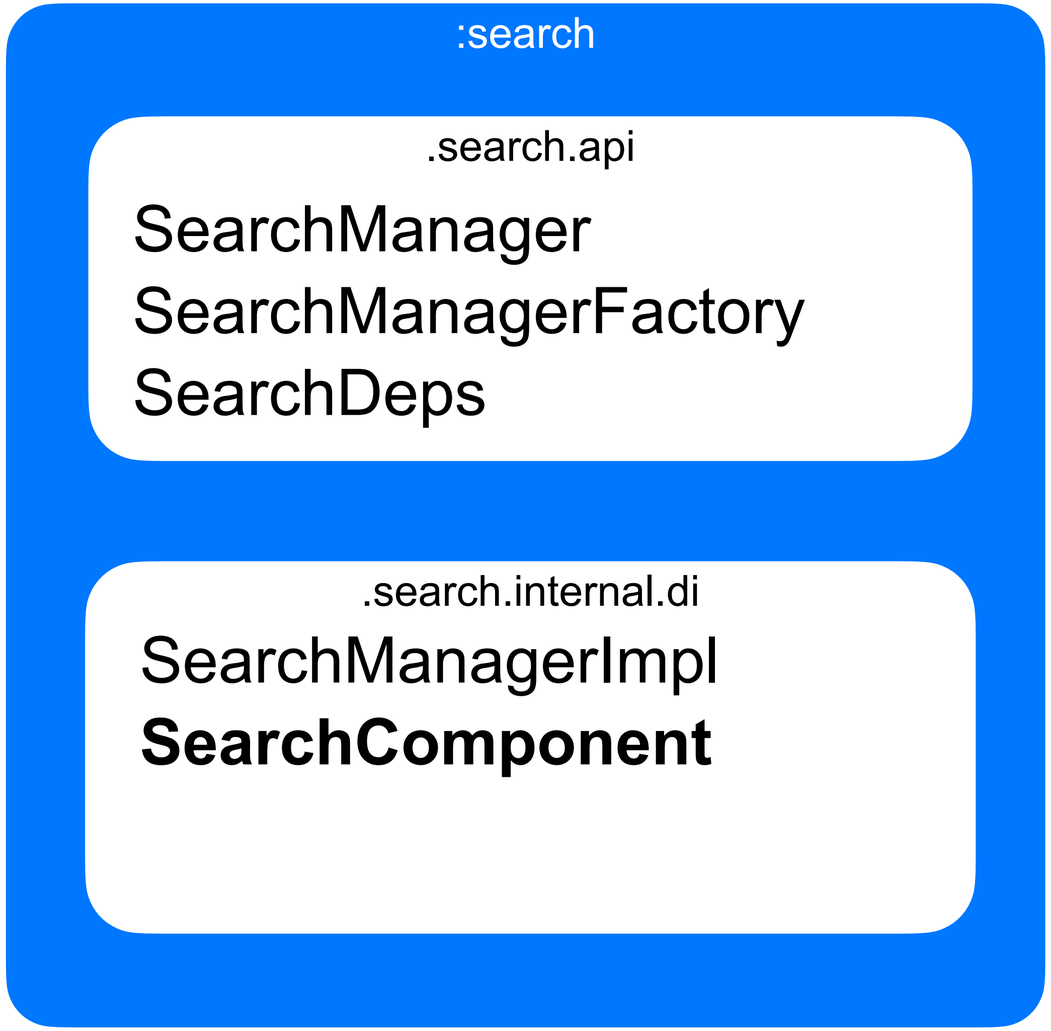

```
interface SearchDeps {
    val logger: Logger
    val format: Formatter
    …
}
```

Не UI фича



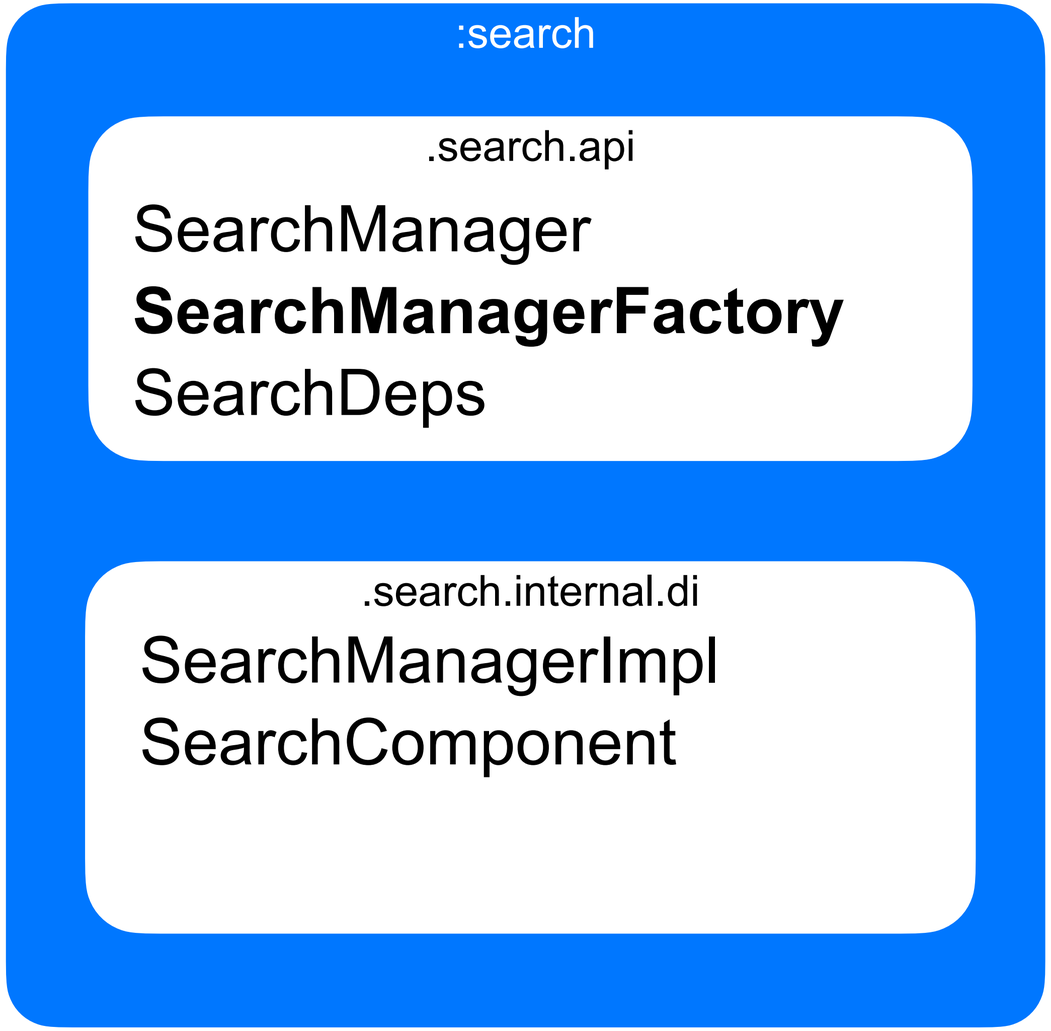

```
internal class SearchManagerImpl
    @Inject constructor(
        val logger: Logger,
        val format: Formatter,
        …
): SearchManager {
    override fun search(…) {
        //...
    }

}
```

Не UI фича




```
@Component(dependencies = [
    SearchDeps::class
])
internal interface SearchComponent {

    //standard code

    val impl: SearchManagerImpl
}
```

Не UI фича




```
object SearchManagerFactory {
    fun create(deps: SearchDeps)
      : SearchManager {
        return DaggerSearchComponent
            .factory()
            .create(deps)
           .impl
    }
}
```

Настало время подвести итоги

Настало время подвести итоги

- Единообразный интерфейс зависимостей

```
interface SearchDeps {
    val logger: Logger
    val format: Formatter
    …
}
```

Настало время подвести итоги

- Единообразный интерфейс зависимостей
- **Фабрика принимает единственный аргумент**

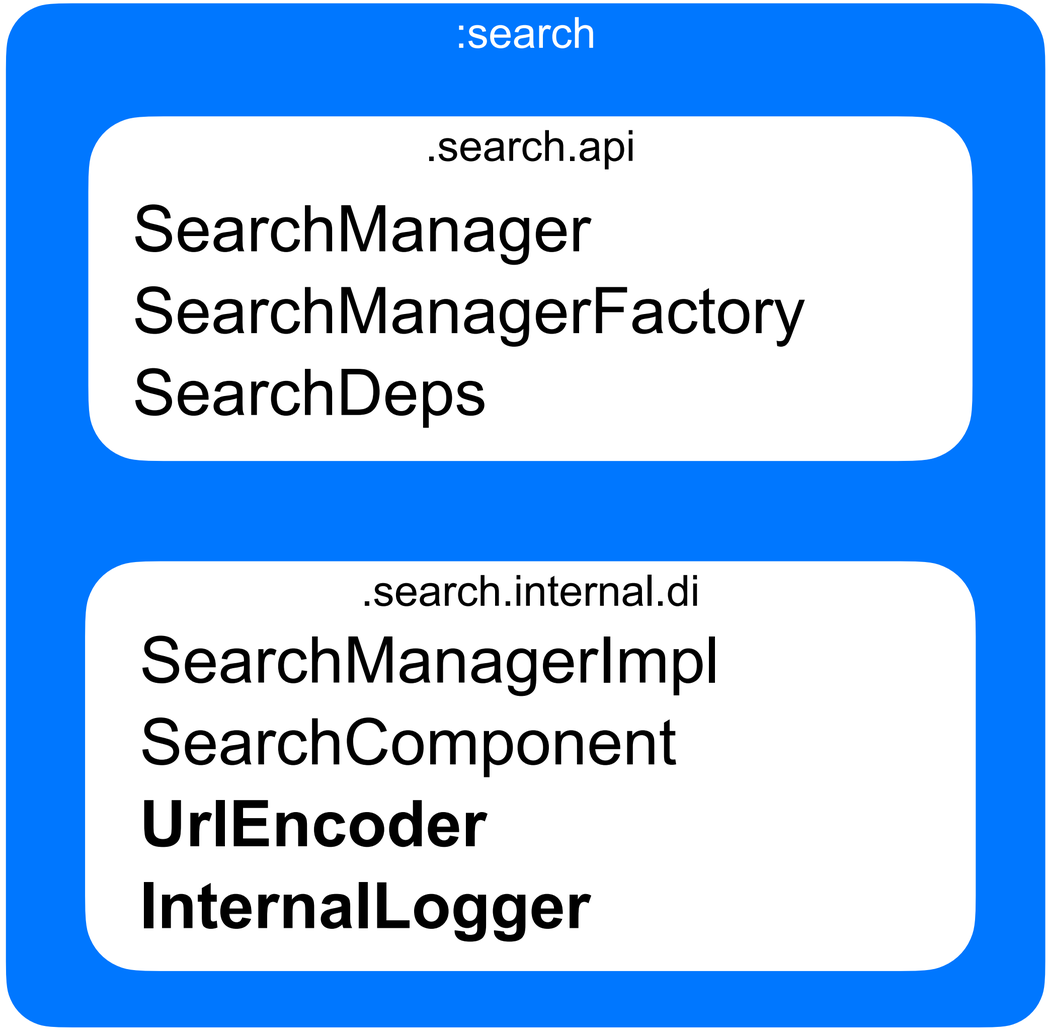
```
object SearchManagerFactory {
    fun create(deps: SearchDeps): SearchManager {
        return DaggerSearchComponent
            .factory()
            .create(deps)
            .impl
    }
}
```

Настало время подвести итоги

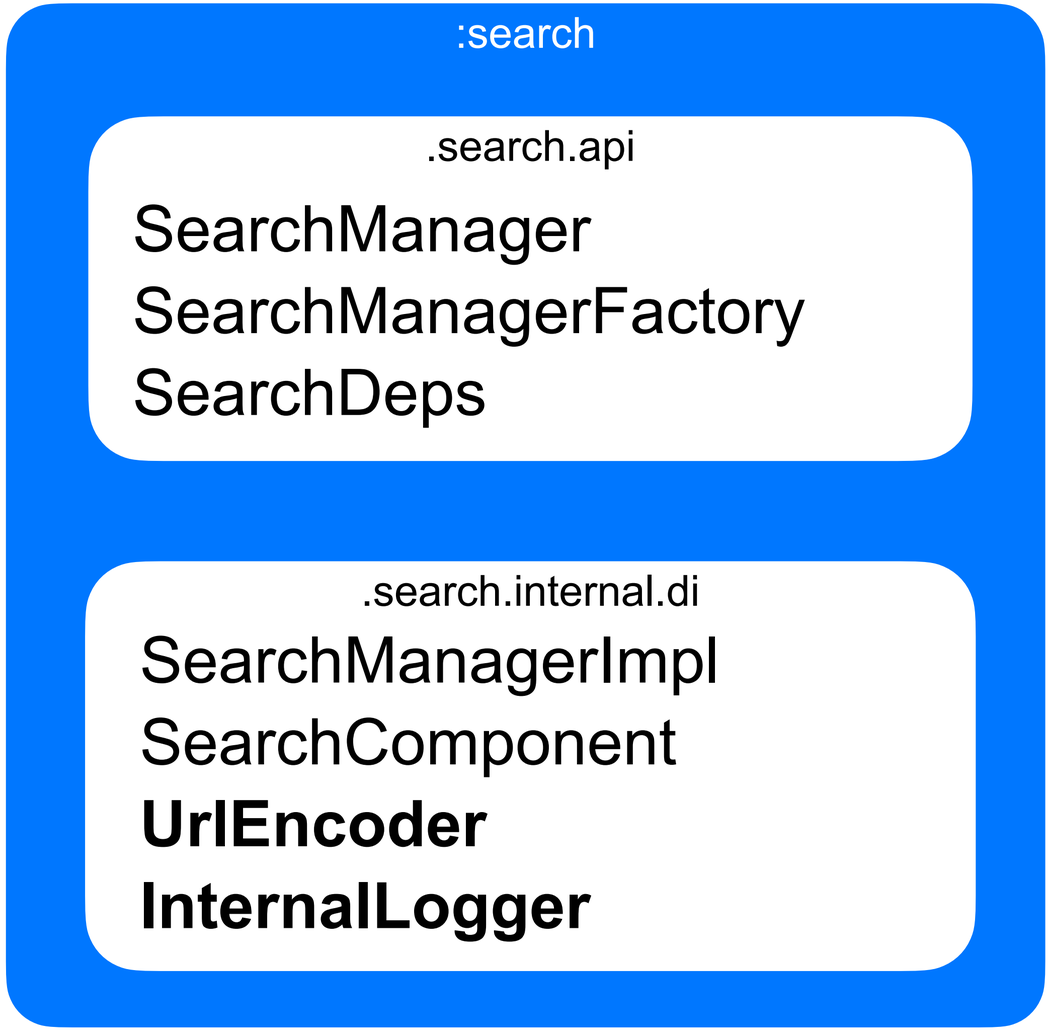
- Единообразный интерфейс зависимостей
- Фабрика принимает единственный аргумент
- Фабрика реализована с помощью Dagger’a

```
object SearchManagerFactory {
    fun create(deps: SearchDeps): SearchManager {
        return DaggerSearchComponent
            .factory()
            .create(deps)
            .impl
    }
}
```

Фабрика реализована с помощью Dagger’a

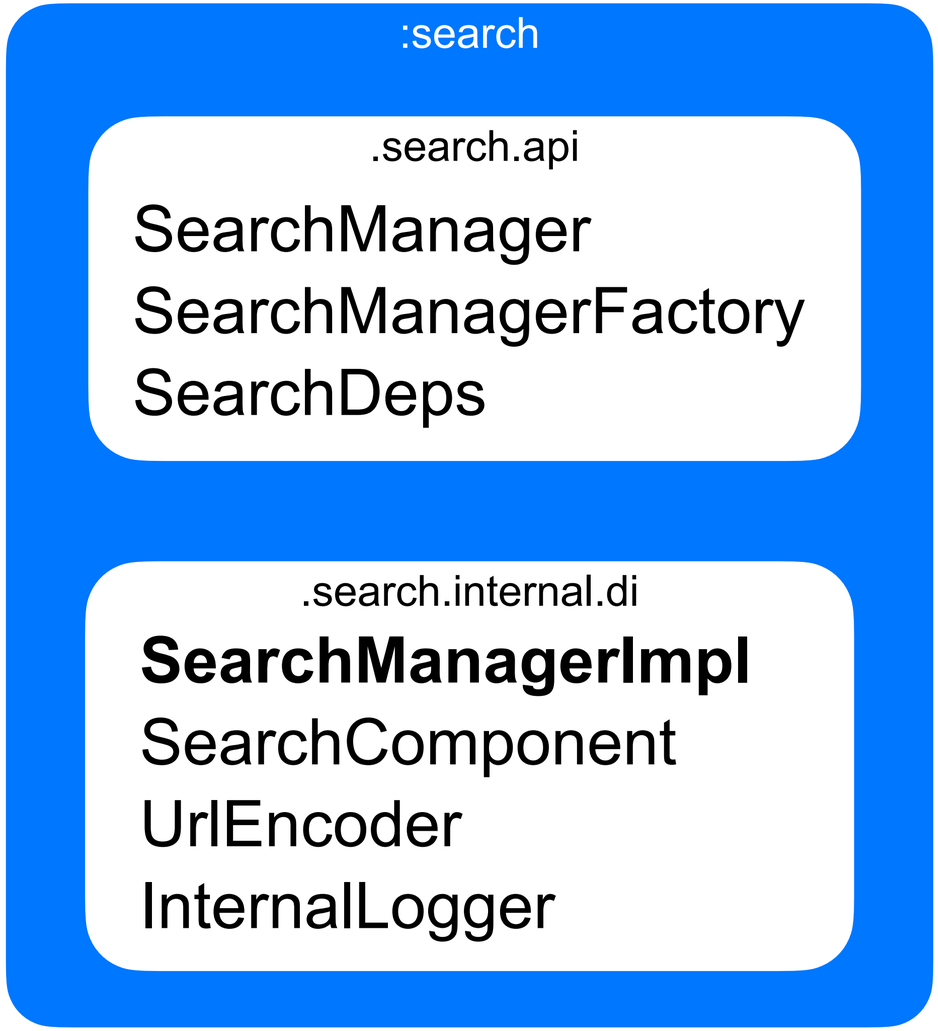


Фабрика реализована с помощью Dagger’a




```
internal class InternalLogger
  @Inject constructor(
        val logger: Logger,
        val format: Formatter
  )

internal class UrlEncoder
  @Inject constructor(
        val format: Formatter
  )
```

Фабрика реализована с помощью Dagger’a




```
internal class InternalLogger
  @Inject constructor(
      val logger: Logger,
      val format: Formatter)

internal class InternalUrlEncoder
  @Inject constructor(
      val format: Formatter)

internal class SearchManagerImpl
  @Inject constructor(
      val logger: InternalLogger,
      val format: Formatter,
      val encoder: InternalUrlEncoder)
```

Настало время подвести итоги

- Единообразный интерфейс зависимостей
- Фабрика принимает единственный аргумент
- Фабрика реализована с помощью Dagger’a
- **Можно использовать сколь угодно сложный граф**

Финальные итоги

Финальные итоги

〉 Подход к модуляризации

Финальные итоги

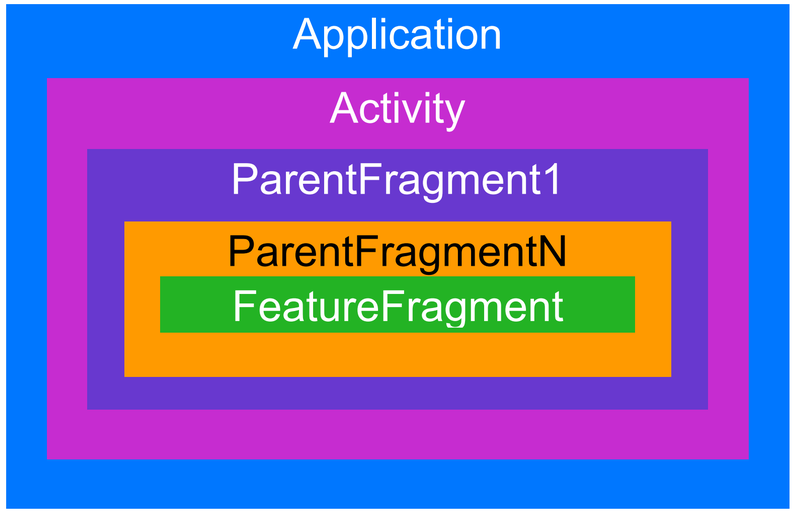
〉 Подход к модуляризации

〉 **Требования к межмодульному DI**

1. Единообразие
2. Независимость от фреймворка
3. Понятный API модуля

Финальные итоги

- Подход к модуляризации
- Требования к межмодульному DI
- **Поиск зависимостей в цепочке родителей**



Финальные итоги

- Подход к модуляризации
- Требования к межмодульному DI
- Поиск зависимостей в цепочке родителей
- **Два способа предоставления зависимостей**

Dagger

Or not?

Финальные итоги

- Подход к модуляризации
- Требования к межмодульному DI
- Поиск зависимостей в цепочке родителей
- Два способа предоставления зависимостей
- **Даггер на службе модулей**

Финальные итоги

- Подход к модуляризации
- Требования к межмодульному DI
- Поиск зависимостей в цепочке родителей
- Два способа предоставления зависимостей
- Даггер на службе модулей
- **Минусы / ограничения / исключения**


ВСЁ!

?

https://github.com/zagayevskiy/android-multimodule-di-example