Koin 3.2 Cheat Sheet For Android

koin by kotzilla

www.kotzilla.io

@kotzilla_io

Koin is a Kotlin library for writing dependency injection in a concise and pragmatic way. Get documentation at insert-koin.io

Gradle Setup

Check the Koin setup page for latest version : setup.insert-koin.io

```
implementation "io.insert-koin:koin-android:$koin_version"
```

Modules

To start define components in Koin, just declare them in a module:

```
val myModule = module { }
```

Declaring your Components

Use the Koin DSL to describe your components inside modules, to inject them by constructor. Use a simple function to build your instance with function DSL:

```
class MyClassA()
class MyClassB(val classA : MyClassA)

val myModule = module {
    single { MyClassA() }
    single { MyClassB(get()) }
}
```

Or directly use a class constructor, with constructor DSL:

```
class MyClassA()
class MyClassB(val classA : MyClassA)

val myModule = module {
    singleOf(::MyClassA)
    singleOf(::MyClassB)
}
```

Koin DSL Keywords

In a Koin module, you can use the following DSL keywords:

Function DSL	Constructor DSL	Description
includes	-	Includes other Koin modules
single	singleOf	Singleton instance definition (unique instance)
factory	factoryOf	Factory instance definition (instance recreated each time)
scope	-	Define a Scope for given type or qualifier
scoped	scopedOf	Define a scoped instance (live for the duration of the scope)
viewModel	viewModelOf	Android ViewModel instance
fragment	fragmentOf	Android Fragment instance
worker	workerOf	Android Worker for work Manager

In Function DSL resolve your dependency (with get()), nullable dependency (with getOrNull()) and load a property from Koin (with getProperty()). Constructor DSL is meant to be used when you only have get().

Naming & Bindings

With your definitions, you can also some extra options, like qualifier (naming) or any extra type binding. Following expressions are equivalent:

```
single(named("myRepo")) { Repo(get()) } bind IRepo:class

singleOf(::Repo) {
    named("myRepo")
    bind<IRepo>()
}
```

Start Koin

Start and configure your Koin instance with the following Start DSL :

Start DSL	Description
startKoin	Start a Koin instance
androidLogger	Use the Android logger for Koin
androidContext	Use the Android Context to be resolved in Koin
modules	Load a list of Koin modules

```
class MyApplication : Application(){
        override fun onCreate() {
            super.onCreate()

             startKoin {
               androidLogger()
               androidContext(this@MyApplication)
               modules(appModule)
             }
        }
}
```

Retrieving dependencies with Koin API

To retrieve some definitions in your Android classes. Just use the by inject() or the by viewModel() functions to inject instances:

```
class MyActivity : AppCompatActivity() {

    val presenter by inject<Presenter>()

    val myViewModel by viewModel<MyViewModel>()
}
```

You may use KoinComponent interface to help Koin extensions if needed.

Koin Annotations Cheat Sheet For Android

koin by kotzilla

www.kotzilla.io

@kotzilla_io

Koin is a Kotlin library for writing dependency injection in a concise and pragmatic way. Get documentation at insert-koin.io

KSP & Gradle Setup

Check the Koin setup page for latest version : setup.insert-koin.io
Be sure to configure your project to use KSP generated sources.

Declaring your Components

Use Koin Annotations on your classes to declare it in Koin. Annotated classes will be injected into their first constructor

```
@Single
class MyClassA()

@Single
class MyClassB(val classA : MyClassA)
```

Any interface is automatically detected as a bound type :

```
@Single
class MyClassB(val classA : MyClassA) : MyInterface
```

But you can also specify a type binding by hand :

```
@Single(binds = [MyOtherInterface::class])
```

Injected dependency can be nullable :

```
@Single
class MyClassB(val classA : MyClassA?)
```

Injected parameter can declared just by tagging your parameter with @InjectedParam

```
@Single
class MyClass(@InjectedParam val id : String)
```

Properties can declared just by tagging your parameter with @Property

```
@Single
class MyClass(@Property("key") val key : String)
```

The @Named annotation allow to add a qualifier on a definition. Just use on the property to inject the dependency with the qualifier :

```
@Single
@Named("class_a")
class MyClassB(val classA : MyClassA)

@Single
class MyClassB(@Named("class_a") val classA : MyClassA)
```

Modules

To gather annotated Koin definitions in modules, just annotate a class with @Module and @ComponentScan to scan for annotated components in the given package :

```
@Module
@ComponentScan("com.my.package")
class MyModule
```

Instead of scanning components, you can also declare components from the module class :

```
@Module
class MyModule {

   @Single
   fun myClassA() = MyClassA()

   @Single
   fun myClassB(classA : MyClassA) = MyClassB(classA)
}
```

You can specify included modules directly on the @Module annotation with includes attribute :

```
@Module(includes = [Module1::class])
class MyModule
```

Koin Annotations

The following annotations can be used in classes:

Annotation	Description
@Module	Declare class as a Koin module. Can also use includes parameter, to specify included modules
@ComponentScan	Scan Koin definitions for given package
@Single	Singleton instance definition (unique instance)
@Factory	Factory instance definition (instance recreated each time)
@Scope	Define a Scope for given type or qualifier
@Scoped	Define a scoped instance (live for the duration of the scope) for given type scope
@KoinViewModel	Android ViewModel instance

You can inject the Android context easily: just use a Context parameter in your constructor.

Start Koin

Start and configure your Koin instance with the regular Start DSL. To use an annotated modul class, just instantiate it and use .module generated extension. Don't forget to use the right import for generated sources :

```
// Use Koin Generation
import org.koin.ksp.generated.*

startKoin {
    androidLogger()
    androidContext(this@MyApplication)
    modules(MyModule().module)
}
```

Retrieving dependencies with Koin API

To retrieve some definitions in your Android classes. Just use the by inject() or the by viewModel() functions to insert instances