Declaring with a Kotlin function

Use Koin Annotations on your top-level function to declare it in Koin:

```
@Single
fun buildMyClassB(classA : MyClassA) = MyClassB(classA)
```

Modules

Annotate a class with @Module and @ComponentScan, to scan for annotated components in the given package:

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

You can also declare components directly in a module class:

```
@Module
class MyModule {

    @Single
    fun myClassA() = MyClassA()

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

Specify included modules directly on the `@Module` annotation with the includes attribute:

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

Koin Annotations

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
@KoinWorker	Android Worker WorkManager

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 module class, just instantiate it and use the .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)
}
```

Insert-koin.io

Koin 3.5 & Koin Annotations 1.2

Cheat Sheet

Now Available
Koin 3.5 Long-Term Support: offering professional support, updates, bug fixes, and security patches for 18+ months.

koin Kotlin

kotzilla

Koin is developed by Kotzilla and open-source contributors

kotzilla.io

Koin 3.5

Koin is the Kotlin integration framework to help you build any kind of Kotlin application, from Android mobile to backend Ktor server applications, including **Kotlin Multiplatform and Compose**.

Gradle Setup

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

Declaring your Components with Modules

Use the Koin DSL to define your components within modules:

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

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

Or directly with constructors:

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

Start Koin

Start and configure your Koin application with the following 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)
        }
    }
}
```

Koin DSL Keywords

In a Koin module, you can use the following 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

Lazy Modules & Background Loading

You may define modules with `lazyModule`, to load them in the background later:

```
val myLazyModule = lazyModule {
        // definitions

}
```

In your Koin application use the `lazyModules` function to load your lazy modules in the background:

```
startKoin {
                // background loading
    lazyModules(appModule)
}

KoinPlatform.getKoin().runOnKoinStarted { koin ->
        // run code after start is completed
}
```

Retrieving dependencies with Koin API

To retrieve definitions in your Android classes with by inject() or the by viewModel():

```
class MyActivity : AppCompatActivity() {

    val presenter by inject<Presenter>()
    val myViewModel by viewModel<MyViewModel>()
}
```

Koin Annotations 1.2

The objective of the **Koin Annotations** project is to facilitate the declaration of Koin definitions in a fast and intuitive manner and generate all the underlying Koin DSL for you.

KSP & Gradle Setup

Visit the Koin setup page for the latest version: setup.insert-koin.io
Ensure you configure your project to utilize KSP generated sources.

Declaring your Components

Use Koin annotations on your classes to declare them in Koin. Annotated classes will be injected via 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
```

Injected dependency can be nullable:

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

Lazy type and List types are detected

```
// Detect Lazy
@Single
class MyClassB(val classA : Lazy<MyClassA>)

// Detect List
@Single
class MyClassB(val allInterfaces : List<MyInterface>)
```

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

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