Koin

Comparación con Dagger-Hilt y Koin Annotations

¿Que es Koin?

- Framework de Inyección de Dependencias (DI)
- Desarrollado en Kotlin, para Kotlin
- Lógica similar a un ServiceLocator

Dependencias

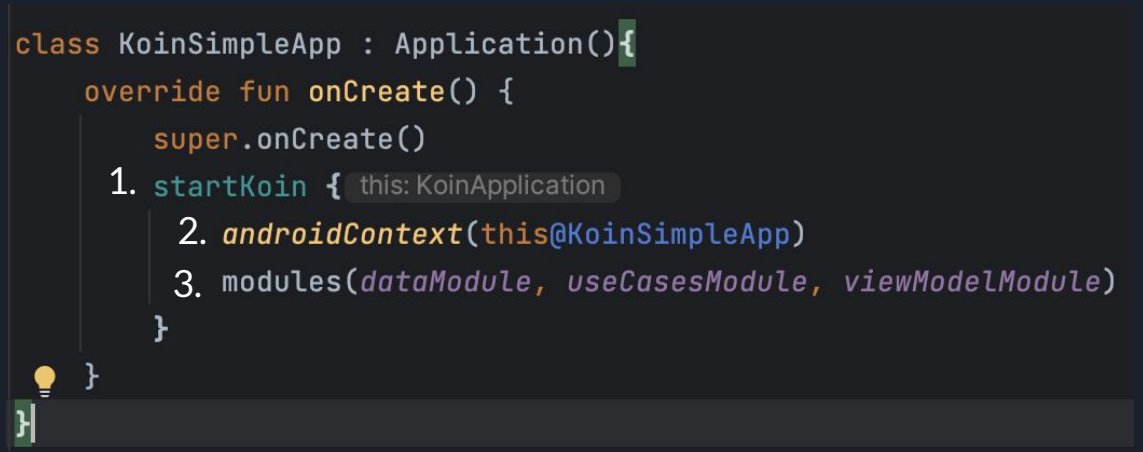
En las últimas versiones, Koin ha sacado dos BoM (una para las clases comunes y otra para Koin annotations) para unificar versiones válidas entre sí

```
implementation 'io.insert-koin:koin-bom:3.5.1' // BoM que incluye las versiones de las distintas dependencias de Koin

implementation 'io.insert-koin:koin-android' // Dependencia con la funcionalidad principal

implementation 'io.insert-koin:koin-test-junit4' // Dependencia para tests con JUnit4
```

Inicializando Koin

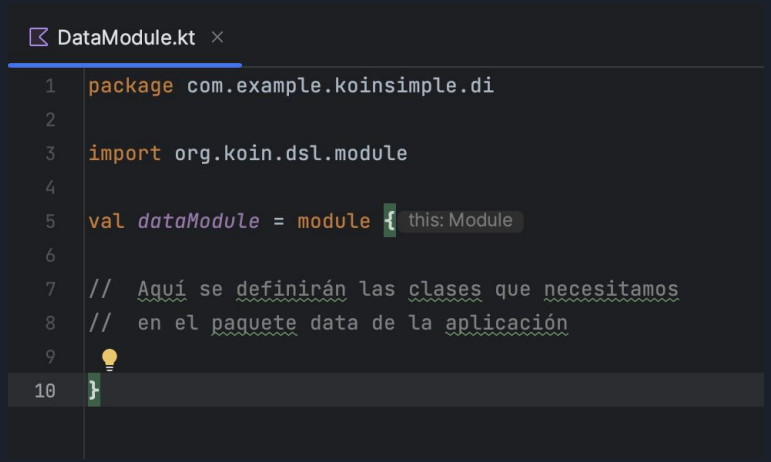
Para inicializar Koin, en el onCreate de la clase que extienda de Application() se debe poner lo siguiente:

1. Método para inicializar Koin
2. Si alguna clase necesita inyectar el Context, debemos indicarlo de esta forma
3. El listado de módulos que contienen las definiciones de las clases de la app



Modules

Los módulos son una variable donde se reunirán las definiciones de las clases que Koin podrá proveer a lo largo de la ejecución de la aplicación. Estas clases pueden ser tanto propias como de terceros y no es obligatorio que estén todas las definiciones dentro del mismo módulo. Se declaran así:



Single, Factory y ViewModel

En Koin, las definiciones de clases pueden ser de tres tipos:

Single: Equivalente a Singleton. Cada vez que se pida una clase creada como Single, se devolverá la misma instancia de la misma.

Factory: Cada vez que se pida una clase creada como Factory, se creará y devolverá una instancia nueva de la misma.

ViewModel: Es una instancia que facilita la creación de los View Models (la clase debe extender de androidx.lifecycle.ViewModel).

```
val viewModelModule = module { this: Module
    viewModel { TopRatedMoviesViewModel(get()) }
}
```

```
val useCasesModule = module { this: Module
    factory { GetActorsByMovie(get()) }
}
```

```
val dataModule = module { this: Module
    single { this: Scope it: ParametersHolder
        AuthInterceptor(BuildConfig.BASE_URL)
    }
}
```

Como inyectar el View Model en las vistas

Para inyectar los viewModels en Fragments y/o Activities, se llama a la función by viewModel() de `org.koin.androidx.viewmodel.ext.android.viewModel` . No hace falta hacer más nada en estas clases.

```
import org.koin.androidx.viewmodel.ext.android.viewModel


Tomás Macri
class TopRatedMoviesFragment : Fragment() {

    private var _binding: FragmentTopRatedMoviesBinding? = null
    Tomás Macri
    private val binding get() = _binding!!

    private lateinit var movieAdapter: MovieAdapter

    private val viewModel: TopRatedMoviesViewModel by viewModel()
```

Definición de clases: get()

En cualquier proyecto vamos a contar con clases que tengamos que definir que a su vez necesite de otras clases para crear su instancia. Si estas últimas las hemos definido en algún módulo, podremos inyectarlas con un get(). Por ejemplo:

```
class TopRatedMoviesViewModel constructor(
    private val getTopRatedMoviesUC: GetTopRatedMovies,
) : ViewModel() {
```

```
val useCasesModule = module { this: Module
    factory { GetTopRatedMovies(get()) }
}
```

```
val viewModelModule = module { this: Module
    viewModel { TopRatedMoviesViewModel(get()) }
}
```

Definición de clases: ****Of(::****)

Para facilitar el mantenimiento y la legibilidad de los módulos, las tres anotaciones mencionadas antes se pueden definir de esta forma siempre y cuando la clase a crear no contenga parámetros en su constructor o que Koin tenga la definición de cada uno de ellos)

```
val useCasesModule = module { this: Module
    factory { GetTopRatedMovies(get()) }
}
```

```
val useCasesModule = module { this: Module
    factoryOf(::GetTopRatedMovies)
}
```

```
val viewModelModule = module { this: Module
    viewModel { DetalleMovieViewModel(get(), get(), get()) }
}
```

```
val viewModelModule = module { this: Module
    viewModelOf(::DetalleMovieViewModel)
}
```

```
val dataModule = module { this: Module
    single {  MovieRepositoryImpl(get()) }
}
```

```
val dataModule = module { this: Module
    singleOf(::MovieRepositoryImpl)
}
```

Otras cosas...

Algunas cosas que suelen utilizarse en proyectos y que se pueden hacer con Koin son:

- Binding de una clase con su interfaz
- Ponerle un name a lo que se inyecta
- Recuperar una definición por su tipo
- Recuperar una definición su nombre
- Inyectar el Context de Android (debe hacerse en la Application también)

```
singleOf (::MovieRepositoryImpl) bind MovieRepository::class

single(named( name: "api_key")) { BuildConfig.API_KEY }

single(named( name: "base_url")){ BuildConfig.BASE_URL }

single { MoshiConverterFactory.create() }

single { this: Scope  it: ParametersHolder
    Retrofit.Builder() Retrofit.Builder
        .baseUrl(get<String>(named( name: "base_url"))) Retrofit.Builder
        .addConverterFactory(get<MoshiConverterFactory>())
        .client(get())
        .build()
}

single { get<Retrofit>().create(MovieApi::class.java) }
```

```
single { StringProvider(androidContext()) }
```

Comparación con Dagger-Hilt

Koin	Dagger-Hilt
Hecha en Kotlin (más liviana)	Hecha en Java
La performance de la app se ve afectada	La performance de la app no se ve afectada
Soporta proyectos de KMM	No soporta apps de KMM
La excepciones se producen en tiempo de ejecución	La excepciones se producen en tiempo de compilación
Tiene un menor tiempo de compilación	Tiene un mayor tiempo de compilación

Validación de las dependencias antes de compilar con un Test

Para poder comprobar que estamos generando las dependencias que la app inyectará en los constructores antes de compilar la app, se puede hacer lo siguiente (aquellas que no, como Retrofit u OkHttpClient, no he encontrado una manera de validar que se inyecten)

1. Crear un único módulo que incluya todos nuestros módulos de Koin

```
val applicationModule = module { this: Module
    includes(useCasesModule, viewModelModule, dataModule)
}
```

2. Crear el siguiente test → Verificará que todas las clases se pueden instanciar con las clases de sus constructores

```
class KoinUnitTest {

    @Test
    fun applicationModule_isCorrect() {
        applicationModule.verify(extraTypes = listOf(Context::class))
    }
}
```

* El método aún es experimental

Koin Annotations

Koin annotations se ha creado para mantener el mismo funcionamiento de Koin, pero con anotaciones como Dagger-Hilt. Es un proyecto bastante más nuevo (la primera versión estable es de Junio del 2022) y, por lo tanto, aún no recomendaría su uso en proyectos de clientes reales. Se puede utilizar dentro del proyecto junto con el Koin clásico

Koin Annotations - Gradle

```
//Annotations (Module:app)
plugins {
    id "com.google.devtools.ksp" version "1.9.10-1.0.13" //https://github.com/google/ksp/releases
}
android {
    //Necesario para que los módulos generados con Koin Annotations (@Module) se guarden en nuestra variante (si no se guardan en la carpeta de código generado y se pierden en cada compilación)
    applicationVariants.configureEach { variant ->
        variant.sourceSets.java.each {
            it.srcDirs += "build/generated/ksp/$variant.name/kotlin"
        }
    }
}
dependencies {
  implementation 'io.insert-koin:koin-bom:3.5.1' //BoM de Koin igual que la anterior
  implementation "io.insert-koin:koin-android" //Misma dependencia que en el proyecto anterior
  implementation "io.insert-koin:koin-annotations-bom:1.3.0" // BoM aparte para Koin Annotations
  implementation 'io.insert-koin:koin-annotations' // Dependencia de Koin annotations
  ksp "io.insert-koin:koin-ksp-compiler:1.3.0" //Tarea que generará los módulos en tiempo de compilación
```

Inicializando Koin Annotations

Utilizar el import `org.koin.ksp.generated.*` que se genera en tiempo de compilación para declarar el/los módulo/s

```
package com.example.koinannotations

import android.app.Application
import org.koin.android.ext.koin.androidContext
import org.koin.core.annotation.ComponentScan
import org.koin.core.annotation.Module
import org.koin.core.context.startKoin
import org.koin.ksp.generated.module


class KoinAnnotationsApp : Application() {
    override fun onCreate() {
        super.onCreate()
        startKoin { this: KoinApplication
            androidContext(this@KoinAnnotationsApp)
            modules(AppModule().module)
        }
    }
}

@Module
@ComponentScan //Se puede especificar la ruta del paquete que abarcará el módulo o, por defecto, se coge la ruta de la clase
class AppModule
```

Single, Factory y ViewModel

Estos tipos de instancias se declaran así:

```
@Single
fun provideMovieService(retrofit: Retrofit): MovieApi {
    return retrofit.create(MovieApi::class.java)
}
```

```
@KoinViewModel
class TopRatedMoviesViewModel (
    private val getTopRatedMoviesUC: GetTopRatedMovies,
) : ViewModel() {
```

```
@Factory
class GetTopRatedMovies(private val repository: MovieRepository) {
    fun invoke() = repository.getTopRatedMovies()
}
```

Otras cosas...

- Ponerle un name a lo que se inyecta
- Recuperar una definición su nombre
- El Contexto se inyecta de la misma forma
- Las clases externas se reciben en el constructor
- Binding de una clase con su interfaz

```
@Single(binds = [MovieRepository::class])
class MovieRepositoryImpl(
    private val movieApi: MovieApi,
) : BaseApiResponse(), MovieRepository {
```

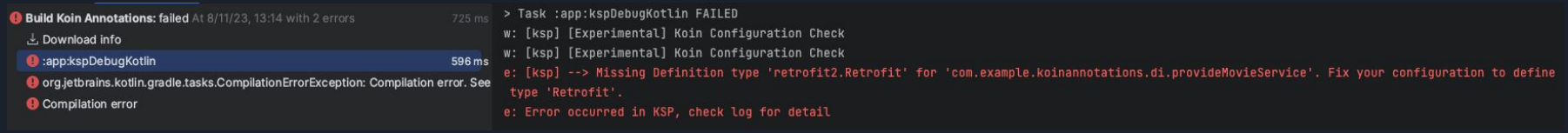
```
@Single
@Named("base_url")
fun provideBaseUrl(): String {
    return BuildConfig.BASE_URL
}

@Single
@Named("api_key")
fun provideApiKey(): String {
    return BuildConfig.API_KEY
}

@Single
fun provideRetrofit(
    okHttpClient: OkHttpClient,
    @Named("base_url") baseUrl: String,
    moshiConverterFactory: MoshiConverterFactory
): Retrofit {
    return Retrofit.Builder()
        .baseUrl(baseUrl)
        .addConverterFactory(moshiConverterFactory)
        .client(okHttpClient)
        .build()
}
```

Validación de las dependencias en tiempo de compilación (a partir de la 1.3.0)

Una de las últimas cosas que se han añadido a Koin Annotations es la validación de las dependencias en tiempo de compilación, lo que permitiría que este framework lance excepciones antes de compilar la app al igual que Dagger-Hilt.

⚠️ Esto aún tiene errores y no es algo lo suficientemente estable a día de hoy ⚠️

Como ejemplo, si se borrara el método provideRetrofit(...) y se compilara se produciría el siguiente error:



Conclusiones personales

- Es bastante simple
- Faltan mejoras, pero se ve que es una librería que está en pleno desarrollo
- Puede ir creciendo junto con la demanda de proyectos en KMM
- A medio o largo plazo, Koin Annotations podría comenzar a quitarle protagonismo a Dagger-Hilt

K&A

Enlaces útiles

- Repositorio con ambos proyectos
- Documentación oficial de Koin para Android
- Documentación oficial de Koin Annotations
- Configuración de Koin en un proyecto
- Releases de KSP
- Koin BoM
- Koin Annotations BoM