nimble

DI Battle: Kotin vs Dagger vs ~~Toothpick~~

Metas

Growth Session #22 - February 15 2019

Intro - What is dependency injection (DI)

Dependency injection (DI) is a technique in which an object is passed as a dependency of another object

Non DI way

```
class Car {

    private val seat: Seat

    init {
        seat = BigSeat()
    }

}


val car = Car()
```

DI way

```
class Car(val seat: Seat)


val bigSeatCar = Car(BigSeat())

val smallSeatCar = Car(SmallSeat())
```

Intro - Why we need DI framework then?

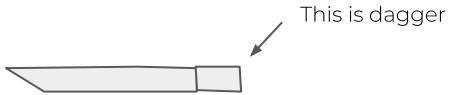
To avoid having to pass a lot of dependencies around, which could be a very time consuming task

```
val apiService = ApiService()
val database = SomeDatabase()

val userRepository = UserRepositoryImpl(apiService, database)
val articleRepository = ArticleRepositoryImpl(apiService)
val rewardRepository = RewardRepositoryImpl(apiService)

val viewModel = MainViewModel(
    userRepository,
    articleRepository,
    rewardRepository
)
```

Intro - Differences

KOIN

- Written in kotlin
- Does not use annotations
- Android support
- Uses DSL style

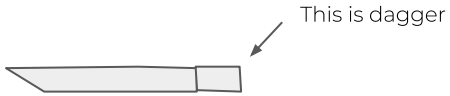


DAGGER

- Written in java
- Official framework from Google
- Use annotations
- Android support

Differences - Create a module

KOIN



DAGGER

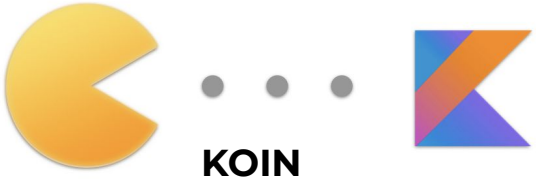
- Put dependencies inside of **module** function
- Use **single { .. }** for binding singleton object and **factory { .. }** to create new instance every time

```
val appModule = module {

    single { RetrofitApiService.service }
    single<CallerApi> { CallerApiRetrofit() }

    factory { BaseRepository() }
    factory { RegisterRepository(androidContext()) }
    factory { MainRepository(androidContext()) }

    viewModel { SplashViewModel() }

}
```

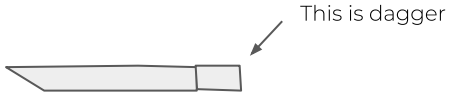
- Create a class then annotate them with **@Module**
- For each dependency annotate them with **@Provides**

```
@Module
class NetworkModule {

    @Provides
    fun provideRetrofitApiService(): RetrofitApiService = RetrofitApiService

    @Provides
    fun provideApiService(): ApiService = RetrofitApiService.service

    @Provides
    fun provideCallerApi(apiService: ApiService): CallerApi = CallerApiRetrofit(apiService)

}
```

Differences - Create app component



KOIN



DAGGER

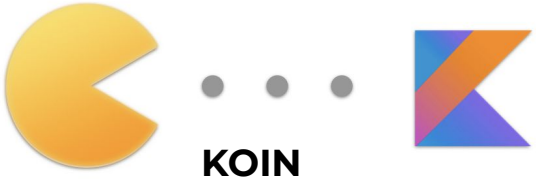
- **startKoin { .. }** is a function
- Put it where the application is created, usually it's a **onCreate(..)**
- It supports some basic android component out of the box like **Context**

```
startKoin {
    androidContext(application)
    androidLogger()
    modules(appModule)
}
```

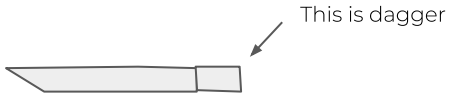
- Create an interface which annotated with **@Component** and define each modules inside
- There are some convention that we need to follow like: it needs to has **build()** function

```
@Component(
    modules = [
        NetworkModule::class,
        RepositoryModule::class
    ]
)
interface AppComponents {

    fun inject(application: OneCardApplication)

    @Component.Builder
    interface Builder {

        @BindsInstance
        fun application(application: Application): Builder

        fun build(): AppComponents

    }

}
```

Differences - Create app component



KOIN



DAGGER

- Just uses **inject()** that koin provided for us

```
class MainActivity: AppCompatActivity() {

    val viewModel: MainViewModel by inject()

}
```

- Use **@Inject** for dependency that we want

```
class MainActivity : AppCompatActivity() {

    @Inject
    val viewModel: MainViewModel

    override fun onCreate() {
        AndroidInjection.inject(this)
        super.onCreate()
    }

}
```

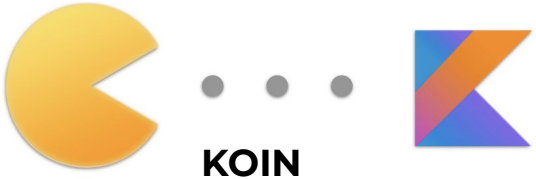
Goals

- Add **Dagger, Toothpick** and **Koin** to **The 1** project.
- See the cost when adding these framework.

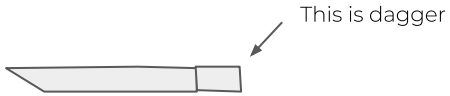
Achievement and Progress

- Add **Dagger** and **Koin** to **The 1** project.
- Try implementing some... testing with these dependency framework.

Conclusion



KOIN



DAGGER

- It's pure kotlin, So It's better when using on koltin project.
- Setting up is lesser than **Dagger.**
- Easier to understand than **Dagger.**
- Better documentation.

- It's written in Java, It's working well with kotlin though.
- Setting up takes time than **Koin.**
- Has a steep learning curve.
- The community is larger, There are so many project using it.
- Confusing documentation.

Thanks!

Contact Nimble

nimblehq.co

hello@nimblehq.co

Bangkok

399 Interchange 21 Sukhumvit Road, Unit #2402-03, Klong Toei, Wattana, Bangkok 10110, Thailand

Singapore

28C Stanley St, Singapore 068737

Hong Kong

20th Floor, Central Tower

28 Queen's Road, Central, Hong Kong

nimble