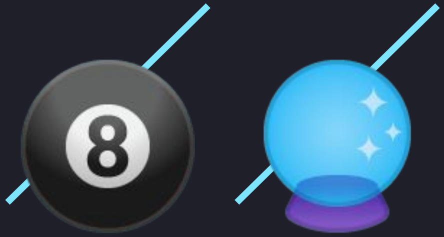
Dagger-Hilt-Koin

A comparison

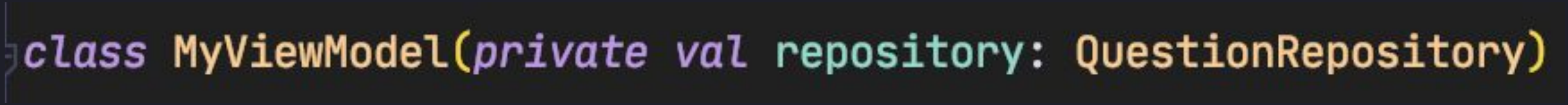
@maiatoday

@maiatoday



What is Dependency Injection

Pattern

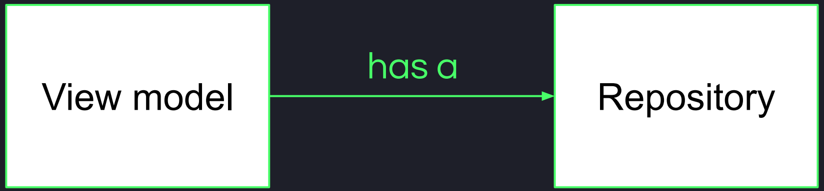


```
class MyViewModel(private val repository: QuestionRepository)
```

Give all the dependent objects

No local construction

Inversion of control



View model

has a

Repository

Why do we need it?

Loose coupling - Object has no knowledge of construction of dependencies

Testing - pass in mocks

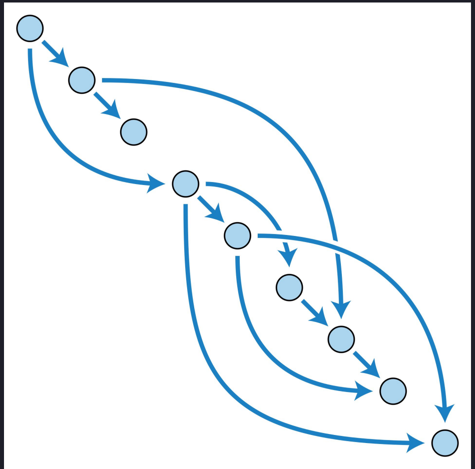
Reduce cognitive load - (once it is set up)

Reduce boilerplate

Dagger

directed **a**cyclic **g**raph (DAG)

is a graph that is directed and without cycles connecting the other edges



Or

No circular dependencies

Dagger

Android dagger

ThermoSiphon

Which one to use?

Heater

CoffeeMaker

@Component

CoffeeLogger !?

@SubComponent

ElectricHeater

@Binds

But I just want to code my feature!

Where do I put things?

@Module

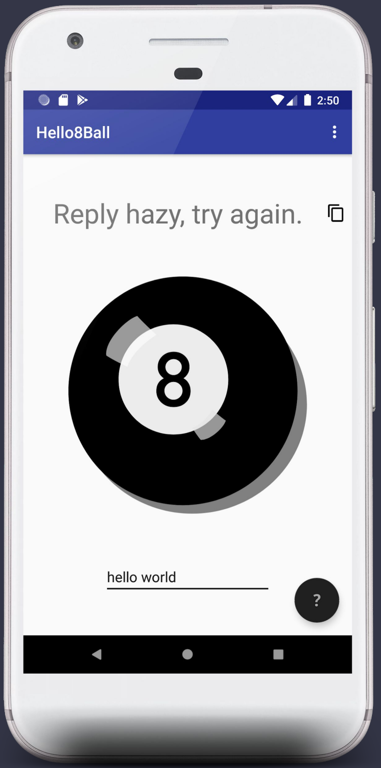
@Inject

Manual*

Koin**

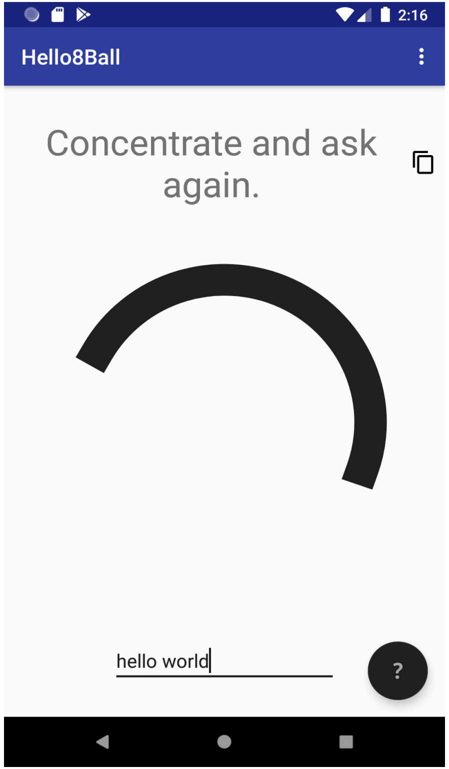
Hilt (Dagger)

Hello 8 Ball



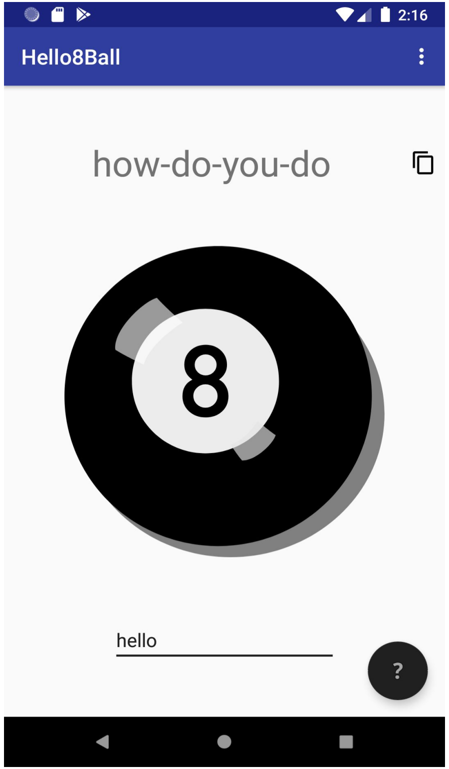
Reply hazy, try again.

hello world



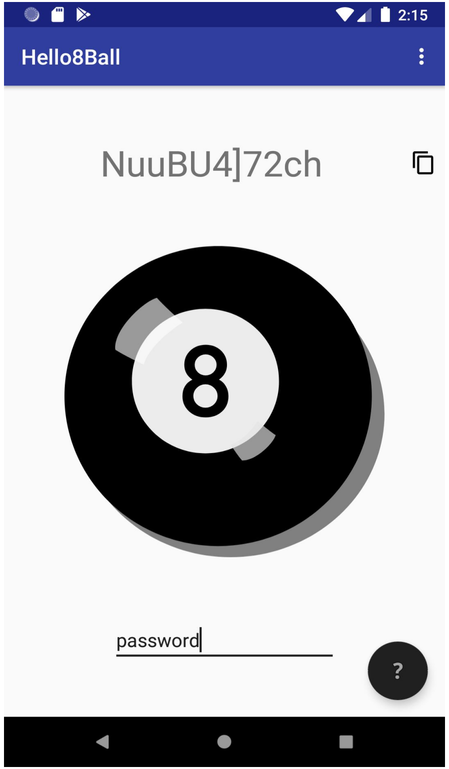
Concentrate and ask again.

hello world



how-do-you-do

hello



2:15

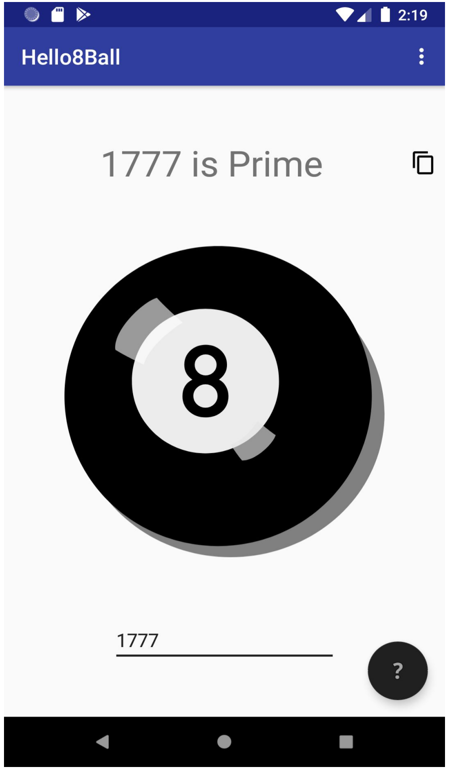
Hello8Ball

NuuBU4]72ch

8

password

?



2:19

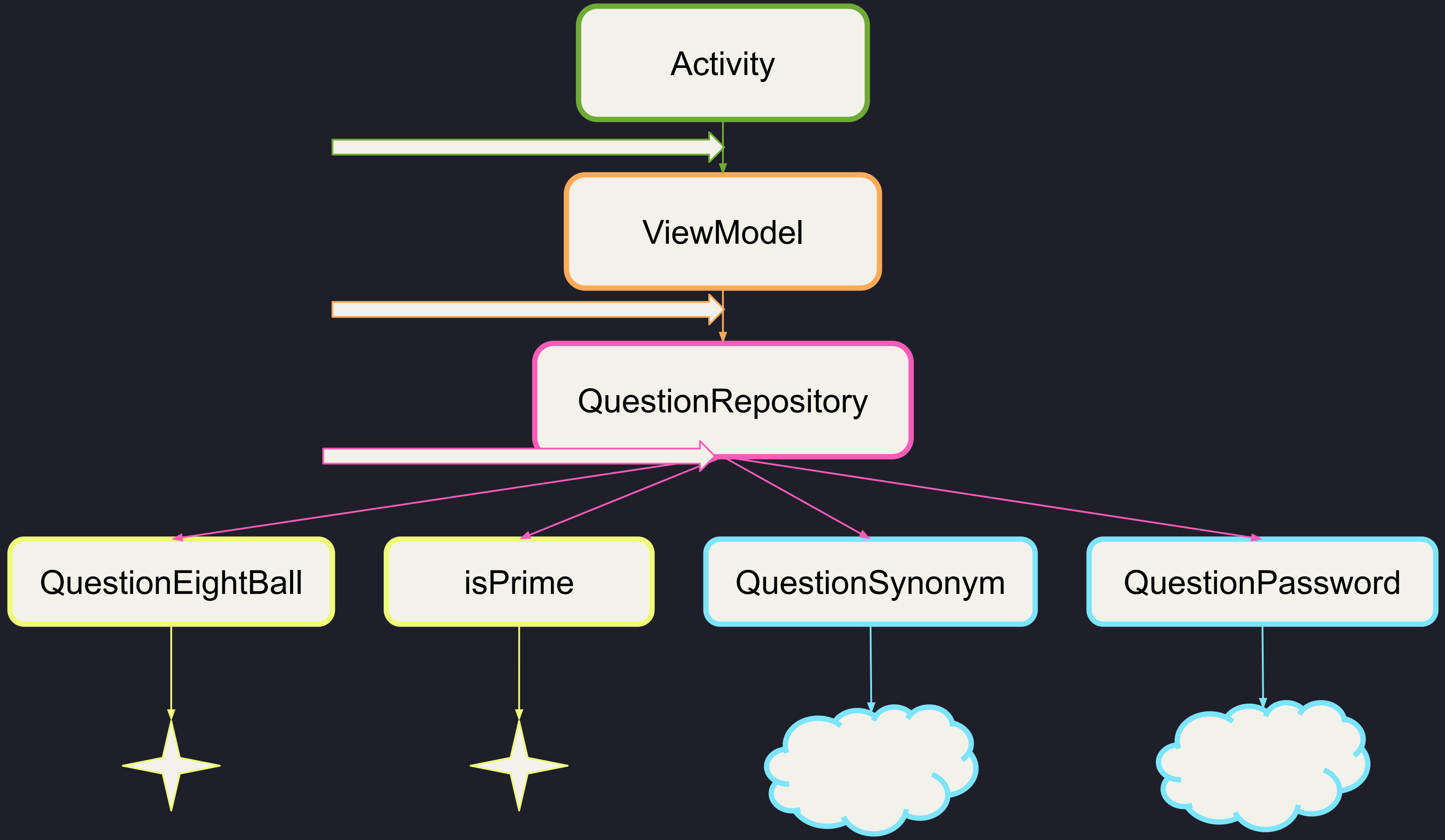
Hello8Ball

1777 is Prime

8

1777

?


Activity
ViewModel
QuestionRepository
QuestionEightBall
isPrime
QuestionSynonym
QuestionPassword

How to do it yourself

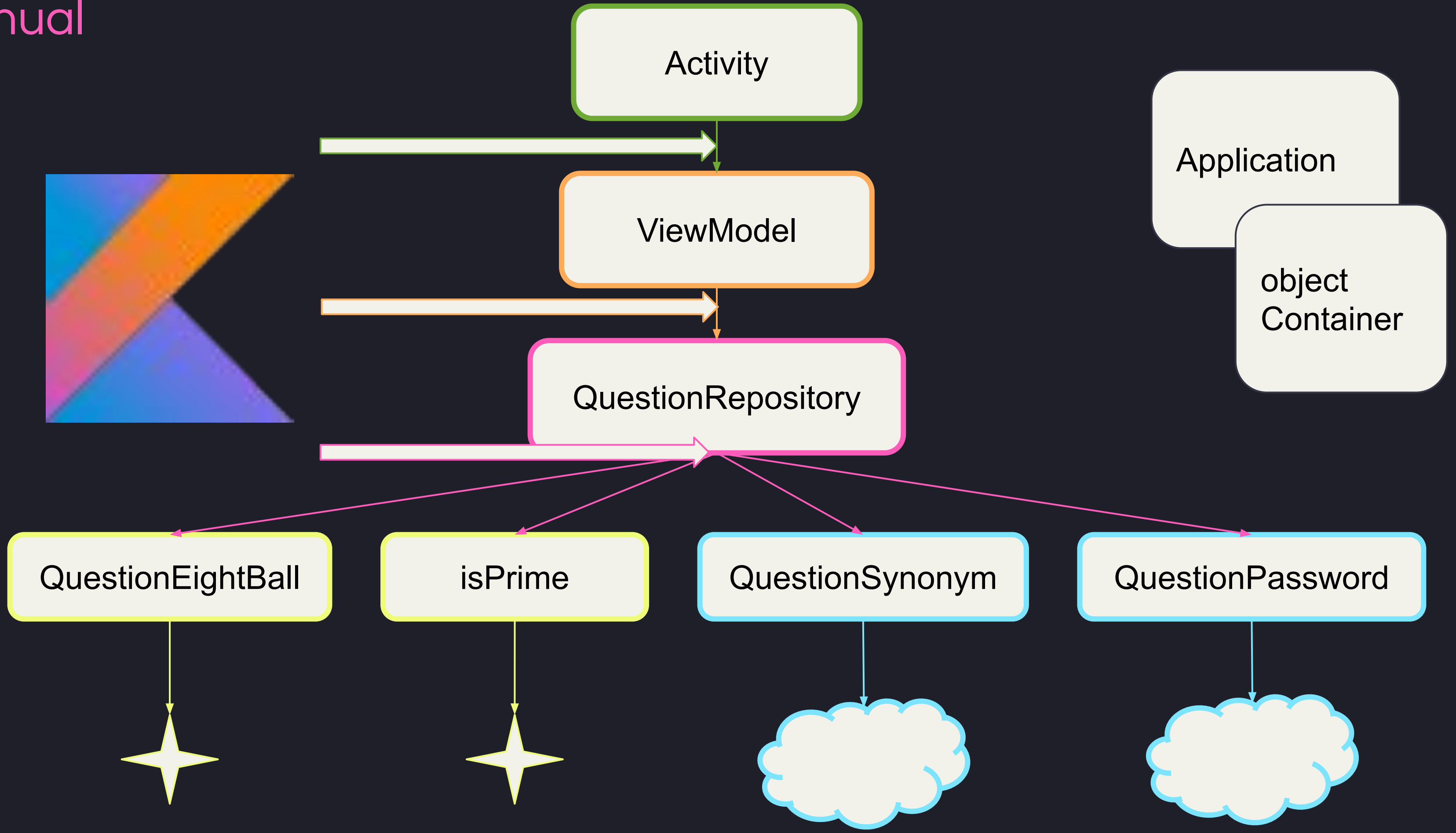
Constructor injection:

- Constructor parameters
- Use Kotlin Default parameters

Activity- field injection:

- Container Object / Container in application

Manual



Activity

ViewModel

QuestionRepository

QuestionEightBall

isPrime

QuestionSynonym

QuestionPassword

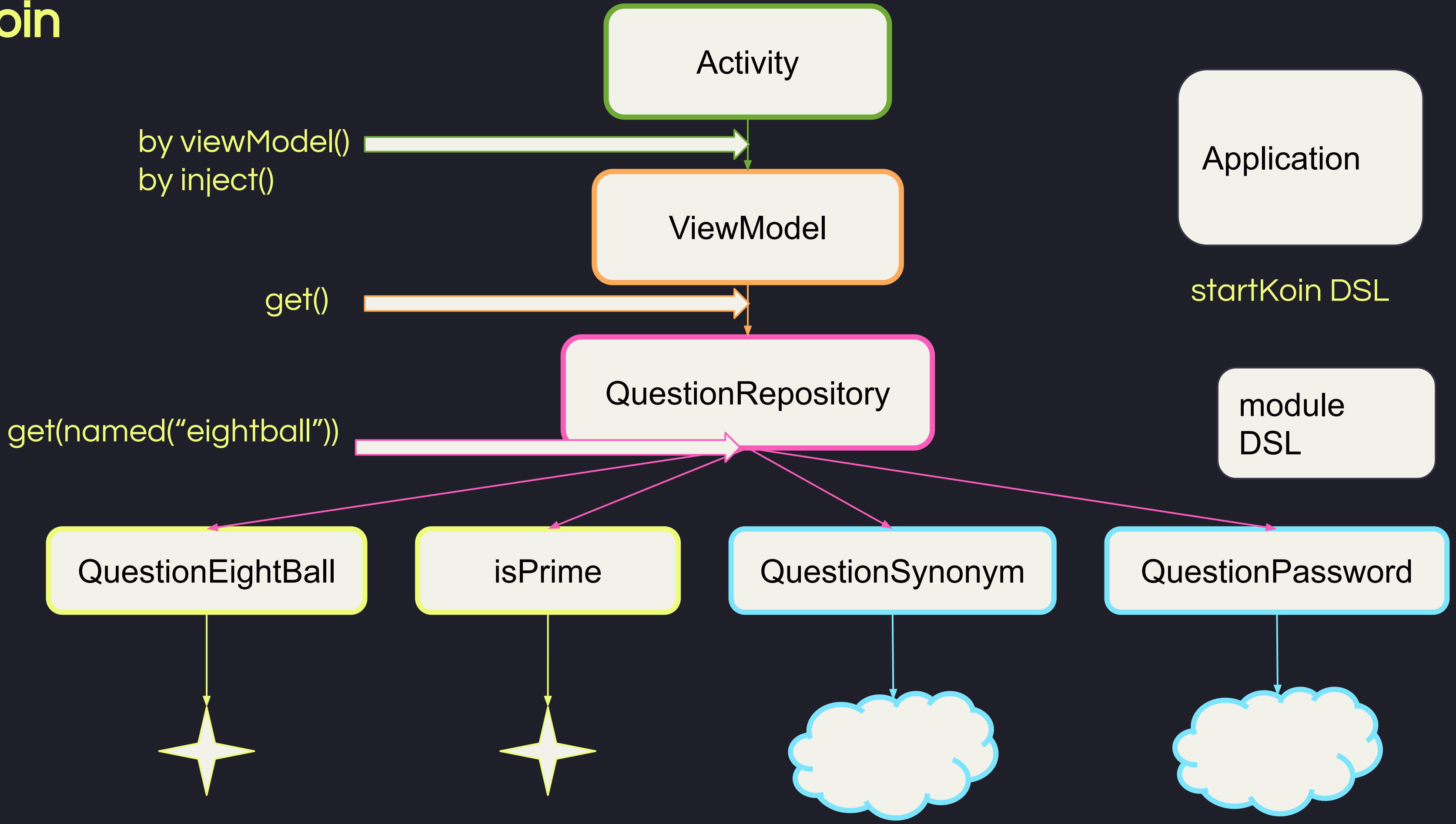
Application

object
Container

Koin - Main features

- Not dependency injection
- Service locator pattern in a library
- No annotation or code generation
- All Kotlin
- DSL
- Knows about view models and context
- Start koin with test modules for integration tests

Koin

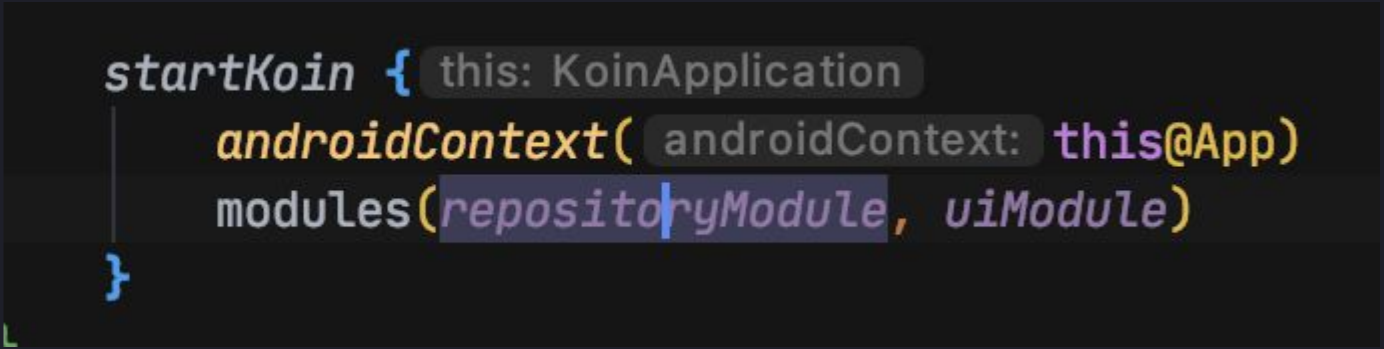


Application

startKoin DSL

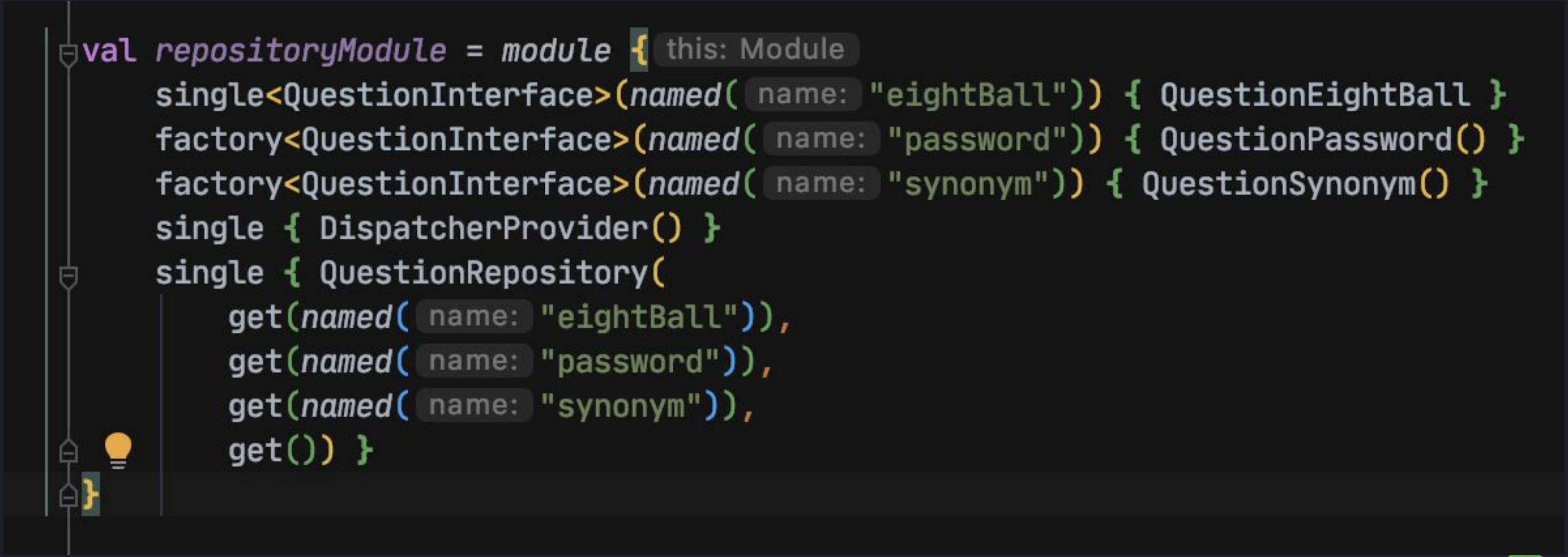
module DSL

Koin - Application



```
startKoin { this: KoinApplication
    androidContext( androidContext: this@App)
    modules(repositoryModule, uiModule)
}
```

Koin - Module

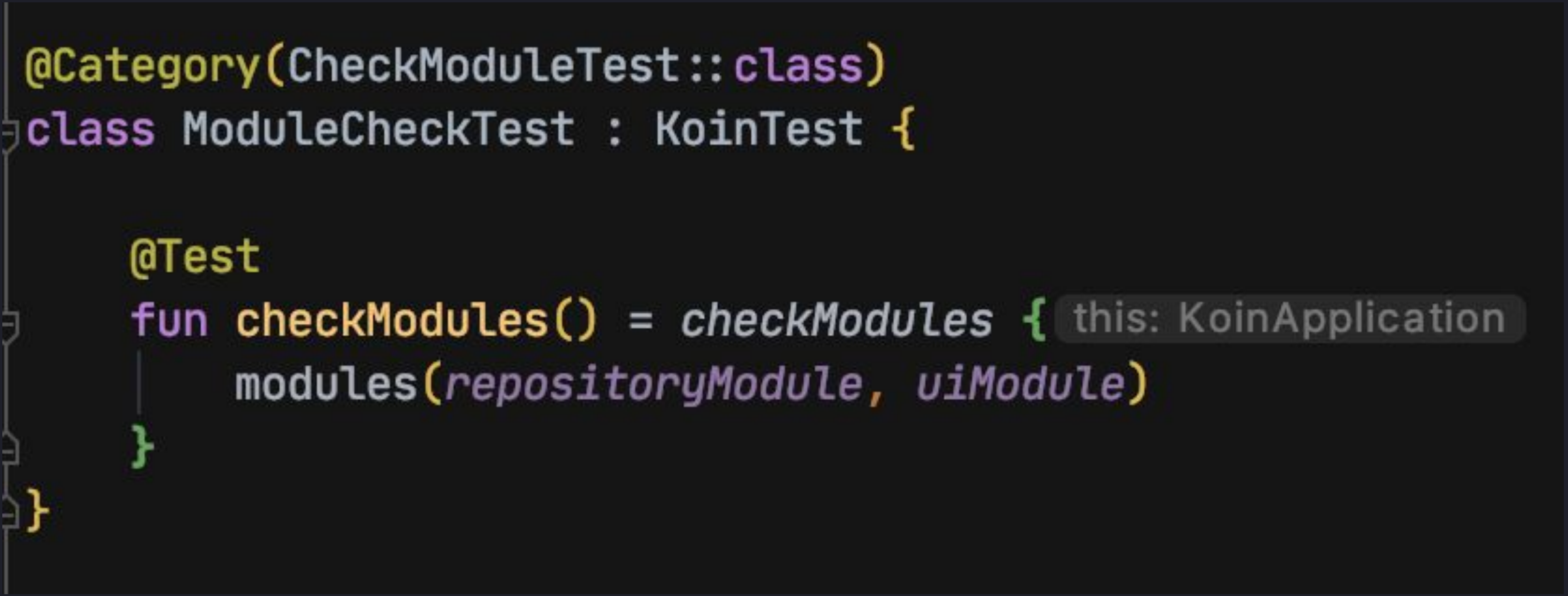


```
val repositoryModule = module { this: Module
    single<QuestionInterface>(named( name: "eightBall")) { QuestionEightBall }
    factory<QuestionInterface>(named( name: "password")) { QuestionPassword() }
    factory<QuestionInterface>(named( name: "synonym")) { QuestionSynonym() }
    single { DispatcherProvider() }
    single { QuestionRepository(
        get(named( name: "eightBall")),
        get(named( name: "password")),
        get(named( name: "synonym")),
        get()) }
}
```

✓

😀

But this is not DI !? - what about runtime crashes



```
@Category(CheckModuleTest::class)
class ModuleCheckTest : KoinTest {

    @Test
    fun checkModules() = checkModules { this: KoinApplication
        modules(repositoryModule, uiModule)
    }
}
```

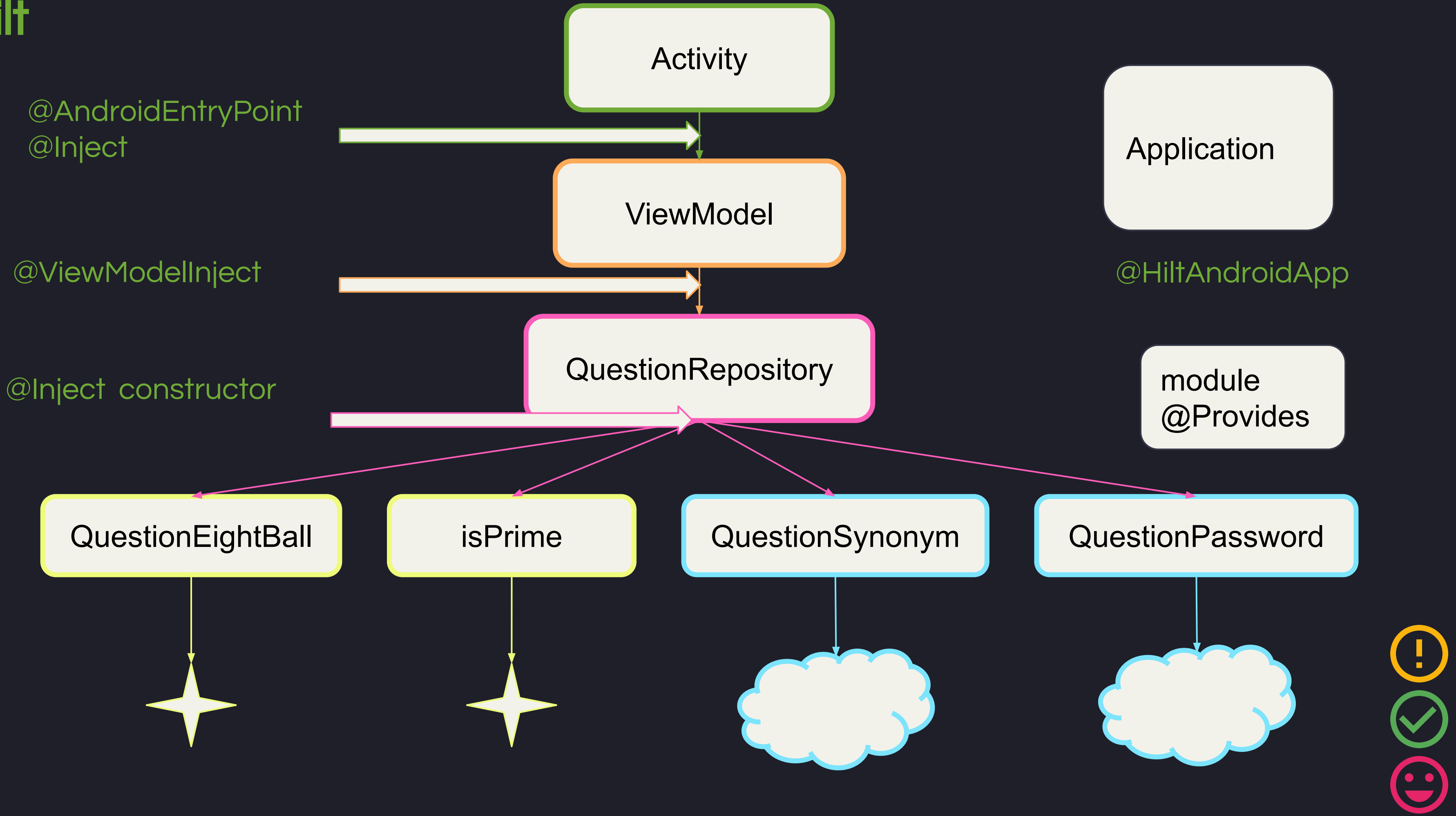
Koin - Recipe

1. Add library
2. Setup: **Application** + manifest
3. Setup: Make **modules**
4. Use: Constructor parameters
5. Use: Parameter with method get()
6. Use: Unit test to check
7. Use: Add to integration tests and unit tests if necessary

Hilt - main features

- Built on Dagger
- Official consistent
- Annotate Injection points
- Annotate providers
- Basic library - AndroidX library for view models
- Knows about Activity, Fragment, View, Service, Context, BroadcastReceiver
- Support for integration tests
- IDE support and Cheat Sheet

Hilt



@AndroidEntryPoint
@Inject

Activity

ViewModel

@ViewModelInject

QuestionRepository

@Inject constructor

QuestionEightBall

isPrime

QuestionSynonym

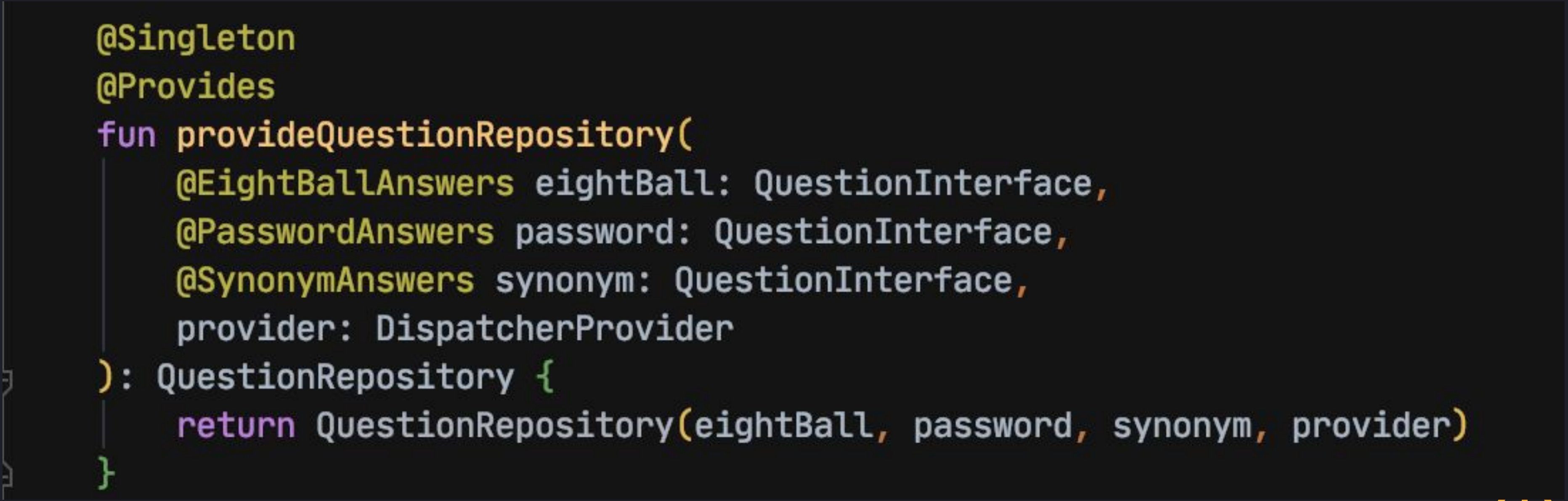
QuestionPassword

Application

@HiltAndroidApp

module
@Provides

Hilt - Module



```
@Singleton
@Provides
fun provideQuestionRepository(
    @EightBallAnswers eightBall: QuestionInterface,
    @PasswordAnswers password: QuestionInterface,
    @SynonymAnswers synonym: QuestionInterface,
    provider: DispatcherProvider
): QuestionRepository {
    return QuestionRepository(eightBall, password, synonym, provider)
}
```

Hilt - Recipe

1. Add library
2. Setup: **Application** + manifest @HiltAndroidApp
3. Setup: Make **module** @Module @InstallIn @Provides
4. Use: Constructor @Inject
5. Use: Parameter with method Inject()
6. Use: Add to integration tests and unit tests if necessary

Contrast and Compare

Manual - 0B

Pros

Simple

Quick to get started

No extra libraries

No code generation

Cons

Boilerplate for factories

Build and manage containers and created objects

Weird bugs ... sometimes (memory, lifecycles)

Need object or application

Service locator anti-pattern

Koin - 600B

Pros

Easy to get started - easy to understand

Simple DSL

Multiple modules

No code generation

No reflection

Cons

Service locator anti-pattern

Run time errors

Hilt - 41B

Pros

Android Official

Built on Dagger but easier to use

IDE support

Android aware

Build type aware

Co-exist with Dagger - easy migration

Cheat sheet

Cons

Alpha

Annotation processors and code generation could make builds slower

Which one?

Really small simple app - Roll your own

Med to large app -

- Before Hilt: **Koin** or maybe Dagger
- After Hilt: **Hilt**

But I already have Dagger ...

Leave it alone or migrate to Hilt - you can have both

Questions?

https://github.com/maiatoday/Hello8Ball

Manual on branch *master*

Koin on branch *koin*

Hilt on branch *hilt*

@maiatoday

References

Manual

https://developer.android.com/training/dependency-injection/manual

https://blog.kotlin-academy.com/dependency-injection-the-pattern-without-the-framework-33cfa9d5f312

Koin

https://insert-koin.io/

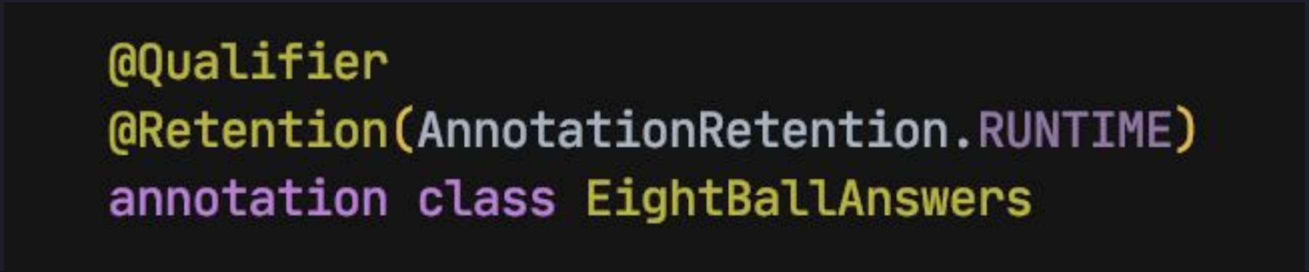
Hilt

https://developer.android.com/training/dependency-injection/hilt-android

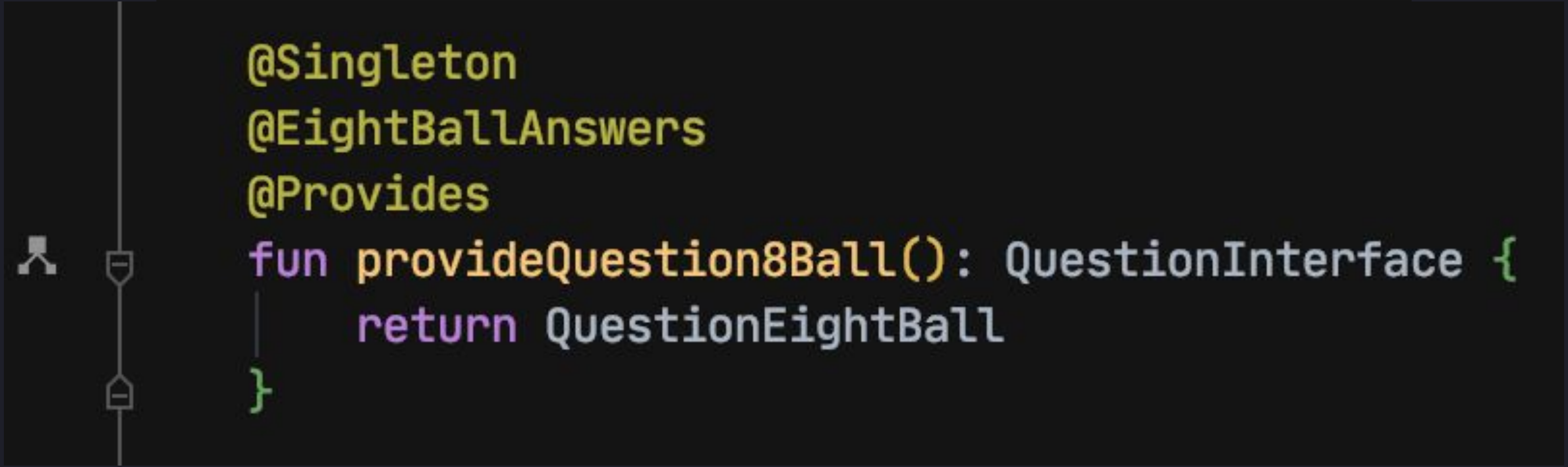
https://proandroiddev.com/viewmodel-from-dagger-to-hilt-223056dd9b

https://developer.android.com/images/training/dependency-injection/hilt-annotations.pdf

Hilt - Module

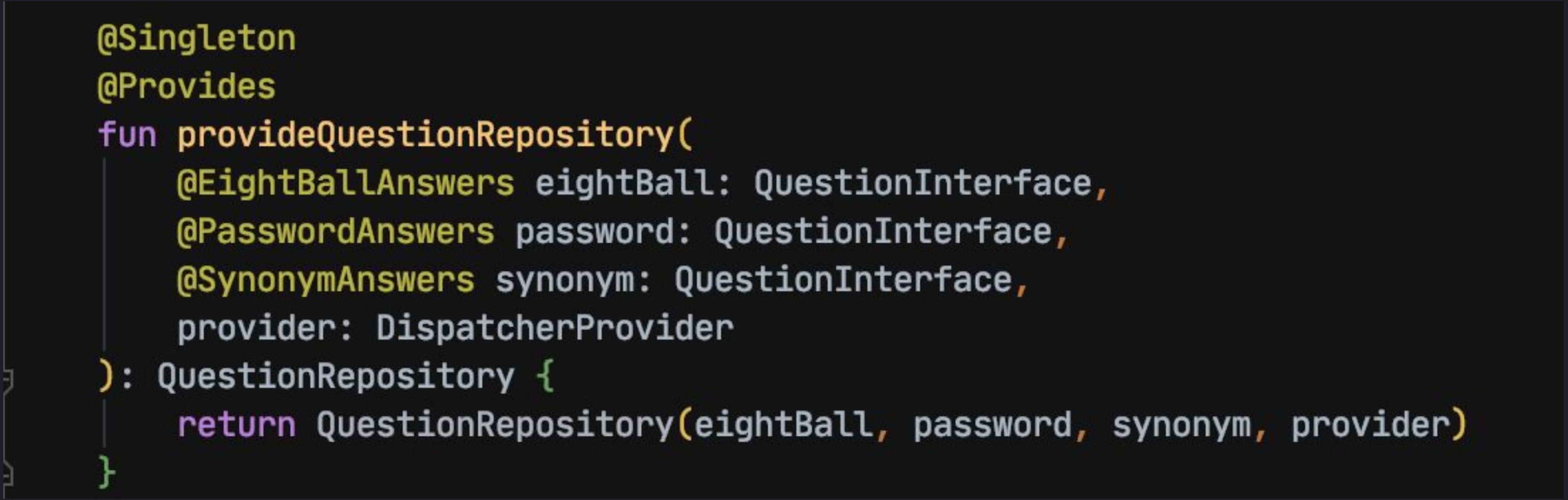


```
@Qualifier
@Retention(AnnotationRetention.RUNTIME)
annotation class EightBallAnswers
```



```
@Singleton
@EightBallAnswers
@Provides
fun provideQuestion8Ball(): QuestionInterface {
    return QuestionEightBall
}
```

Hilt - Module



```kotlin
@Singleton
@Provides
fun provideQuestionRepository(
    @EightBallAnswers eightBall: QuestionInterface,
    @PasswordAnswers password: QuestionInterface,
    @SynonymAnswers synonym: QuestionInterface,
    provider: DispatcherProvider
): QuestionRepository {
    return QuestionRepository(eightBall, password, synonym, provider)
}
```