Dagger-Hilt-Koin

A comparison

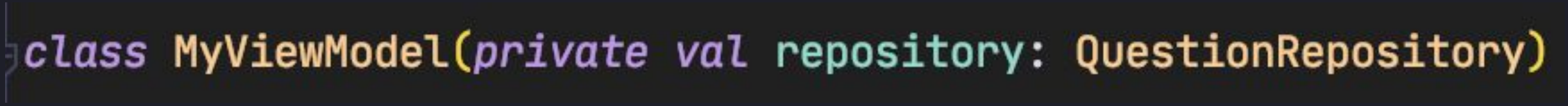
@maiatoday

@maiatoday

~~🎱~~ ~~🔮~~

What is Dependency Injection

Pattern

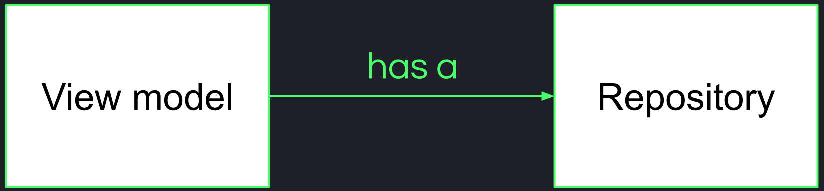


```
class MyViewModel(private val repository: QuestionRepository)
```

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)

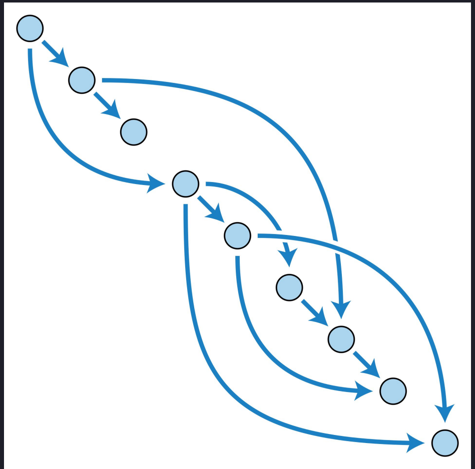
Reduce boilerplate

Separate tests business logic vs construction

Dagger

directed acyclic graph (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

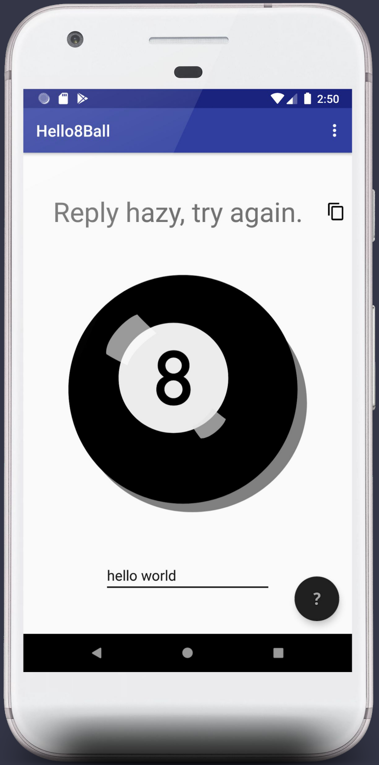
@Inject

Manual*

Koin**

Hilt (Dagger)

Hello 8 Ball

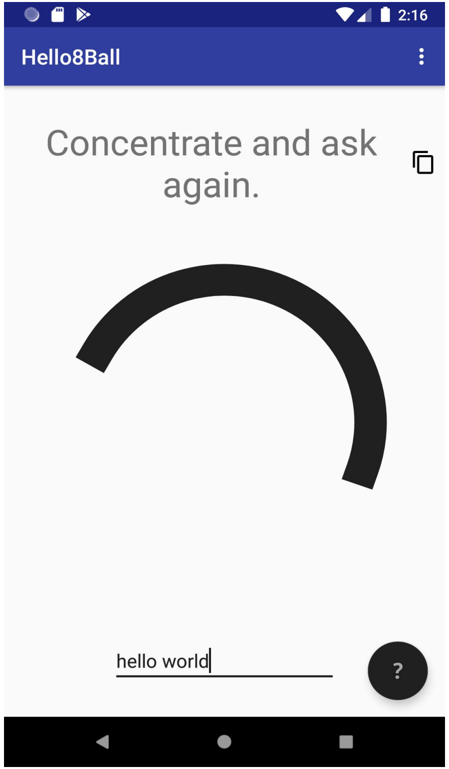


2:50

Hello8Ball

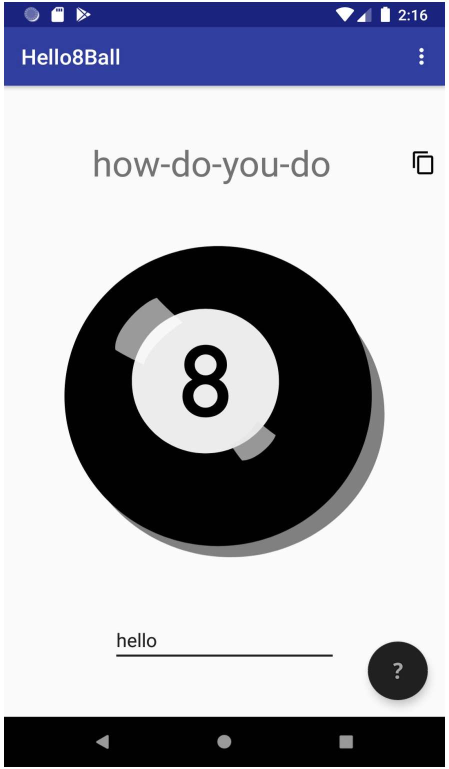
Reply hazy, try again.

hello world



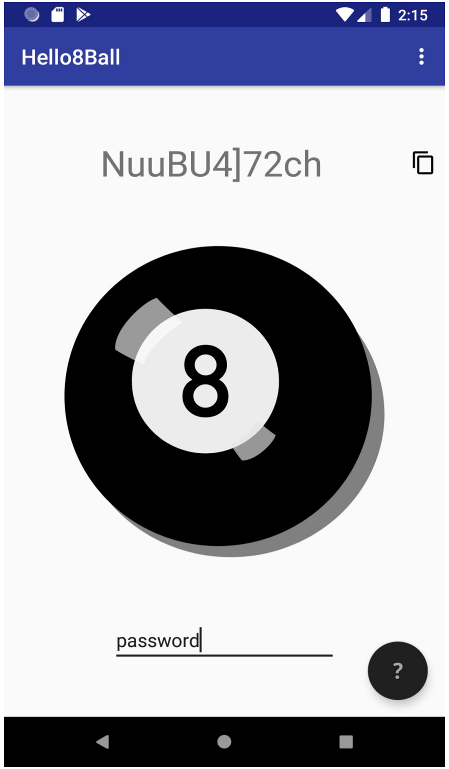
Concentrate and ask again.

hello world



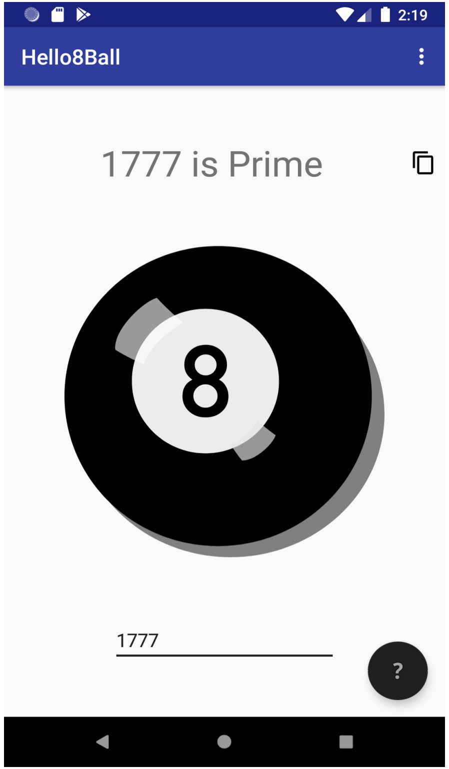
how-do-you-do

hello



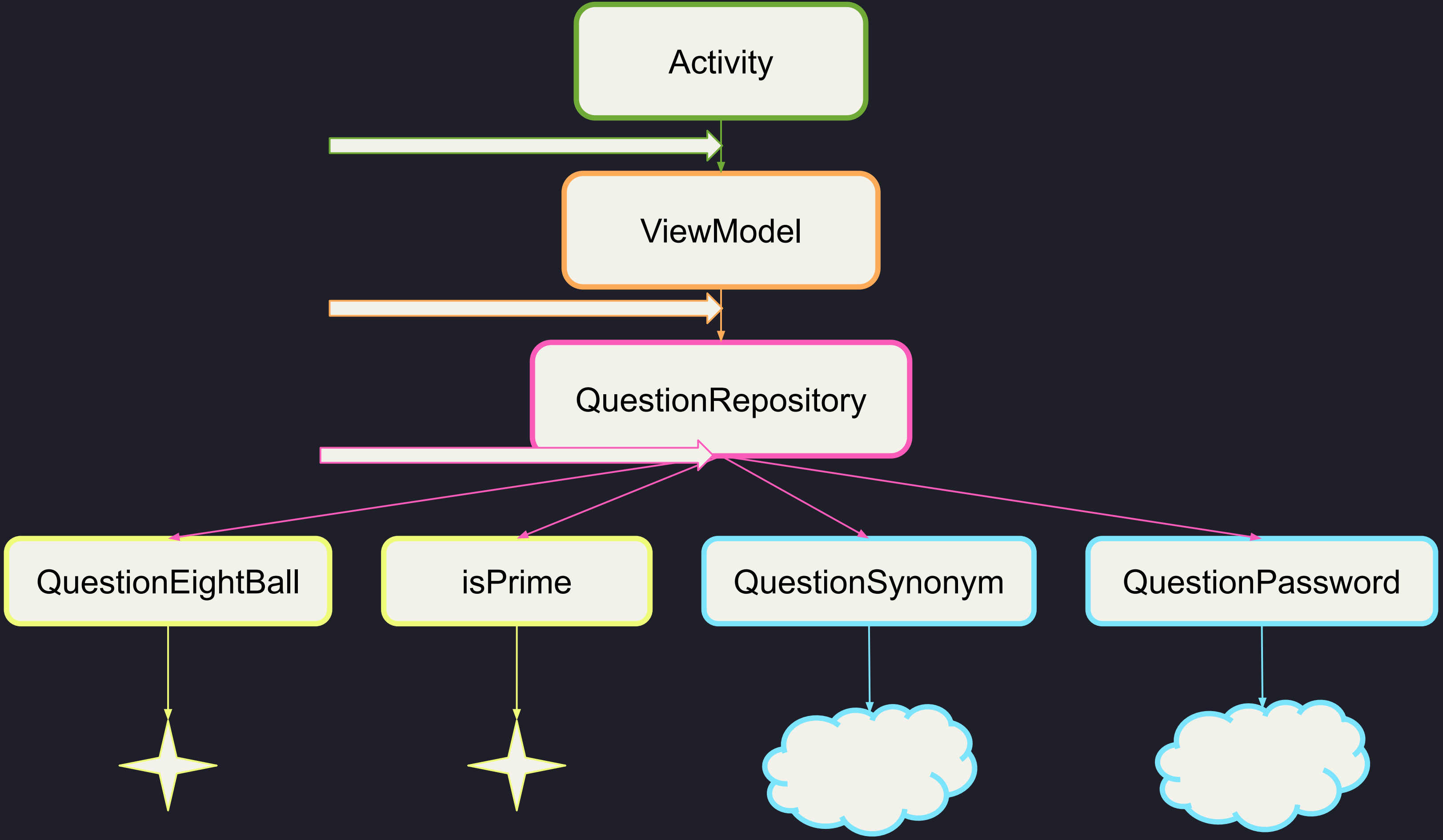
NuuBU4]72ch

password



1777 is Prime

1777


Activity
ViewModel
QuestionRepository
QuestionEightBall
isPrime
QuestionSynonym
QuestionPassword

Use constructor injection

💡 Top testing tip 💡

How to do it yourself

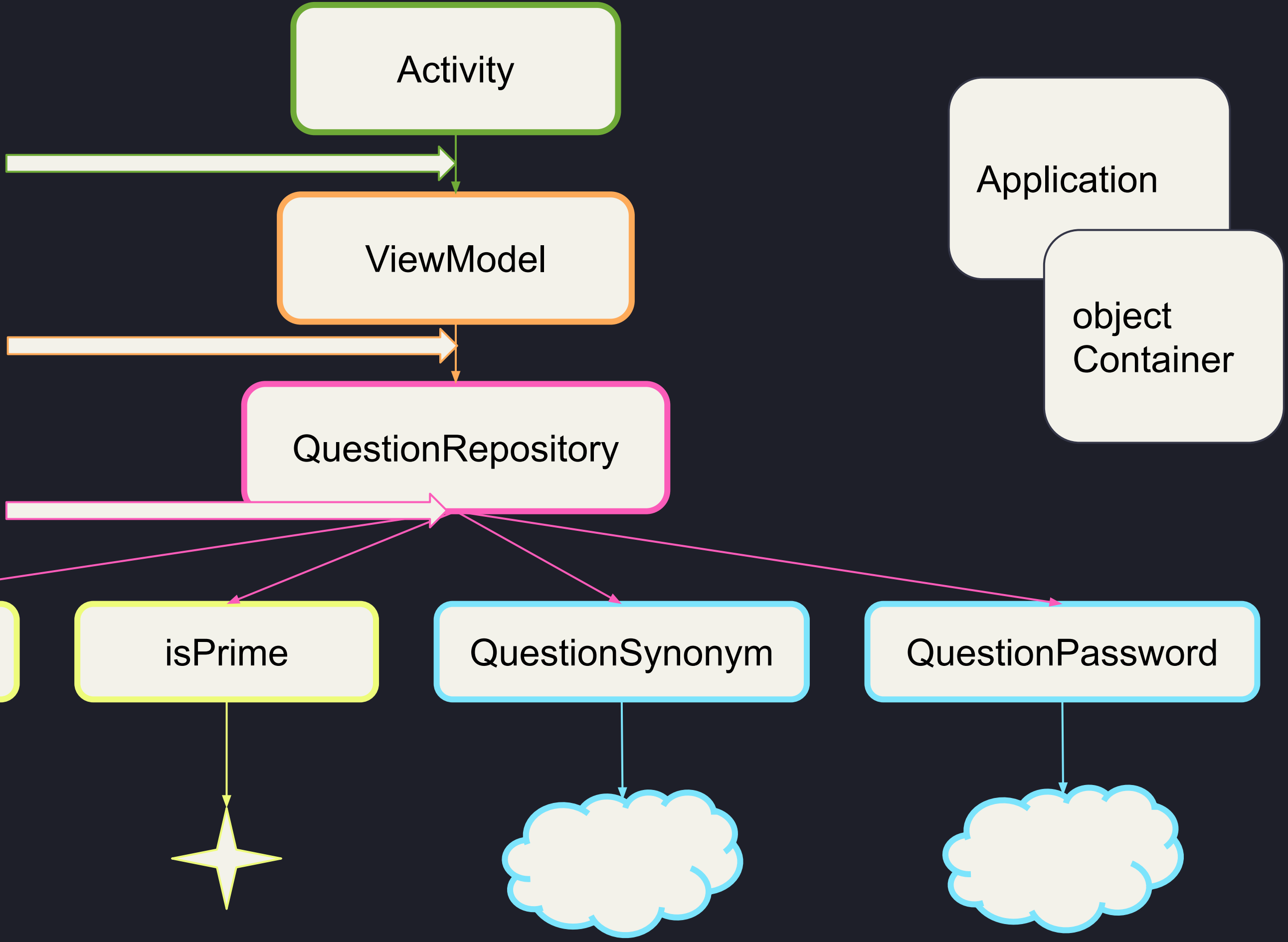
Constructor injection:

- Constructor parameters
- Use Kotlin Default parameters

Activity- field injection:

- Container Object / Container in application

Manual



Activity

ViewModel

QuestionRepository

QuestionEightBall

isPrime

QuestionSynonym

QuestionPassword

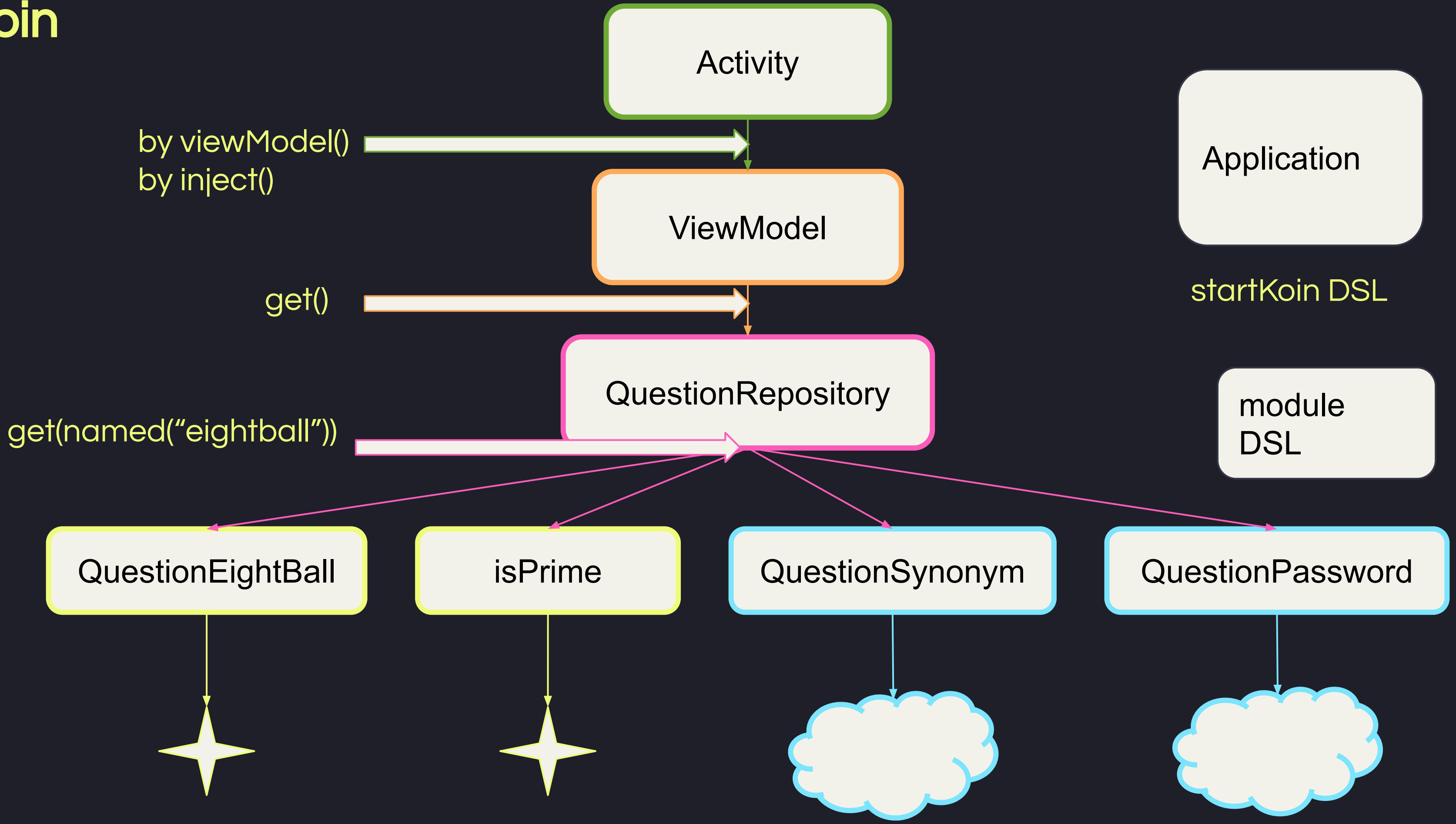
Application

object
Container

Koin - Main features

- Service locator for field injection in a library
- No annotation or code generation - build speed
- Easy to grasp, similar concepts to manual
- All Kotlin
- DSL
- Knows about view models and context
- Start koin with test modules for integration tests

Koin



Activity

by viewModel()
by inject()

ViewModel

get()

QuestionRepository

get(named("eightball"))

QuestionEightBall

isPrime

QuestionSynonym

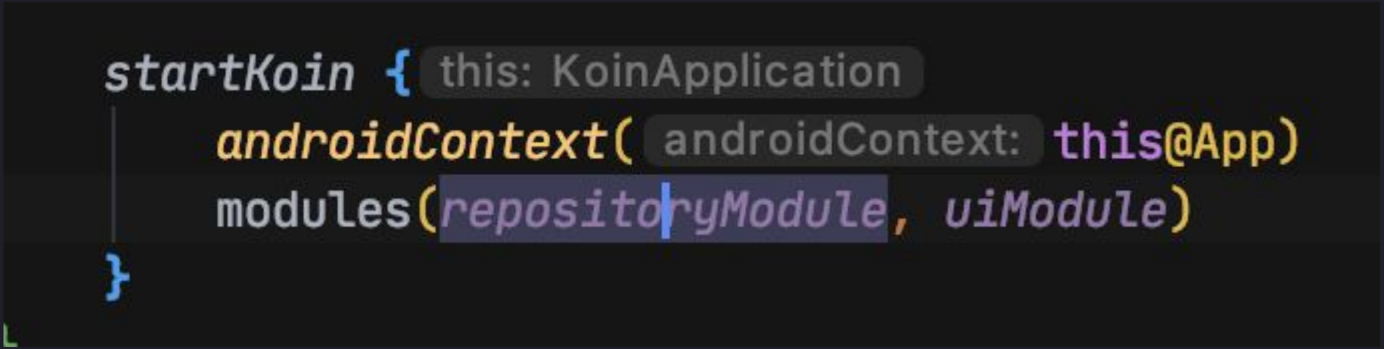
QuestionPassword

Application

startKoin DSL

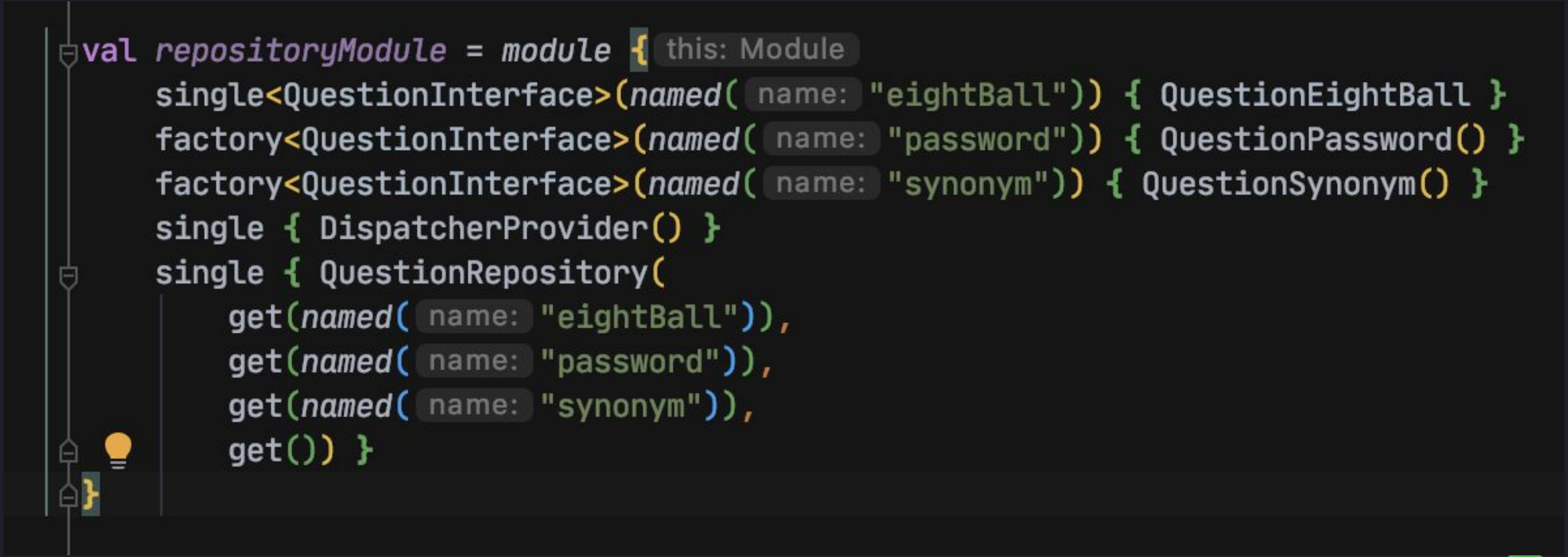
module
DSL

Koin - Application



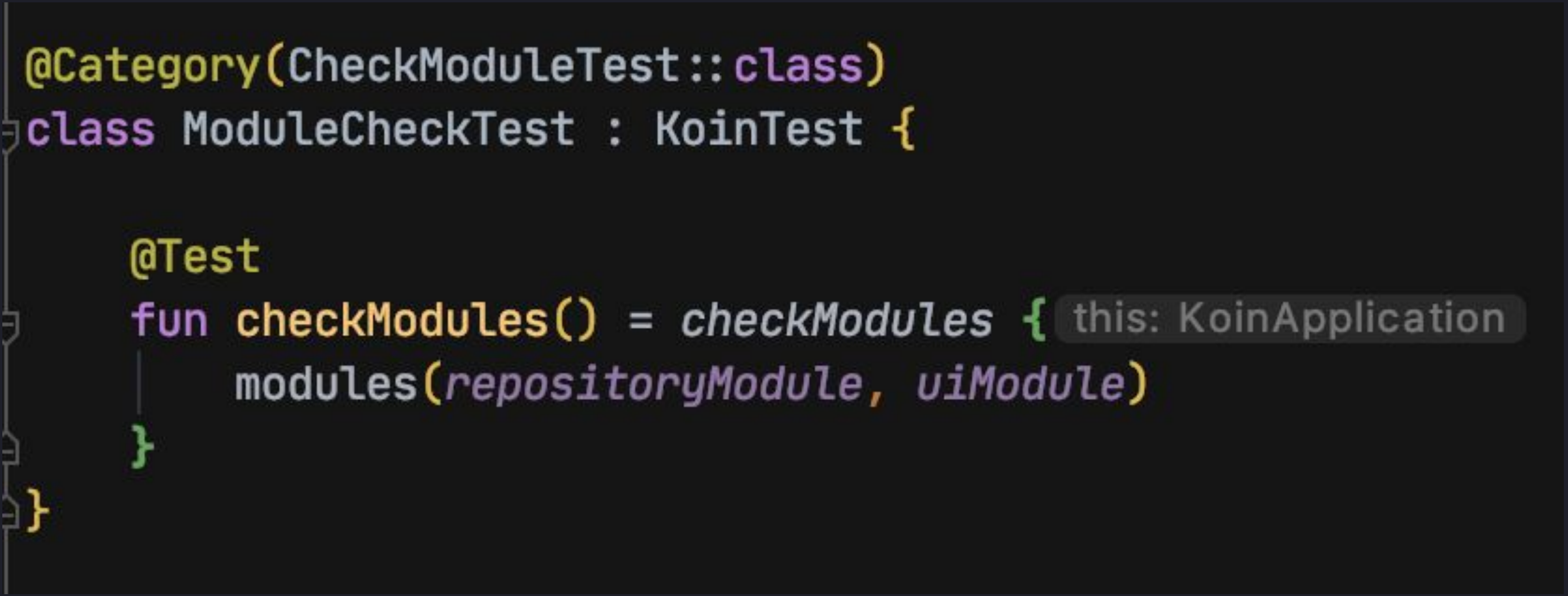
```
startKoin { this: KoinApplication
    androidContext( androidContext: this@App)
    modules(repositoryModule, uiModule)
}
```

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()) }
}
```

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 as necessary

Hilt - main features

- Built on Dagger
- Official - consistent - opinionated
- 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

One step back - to dagger

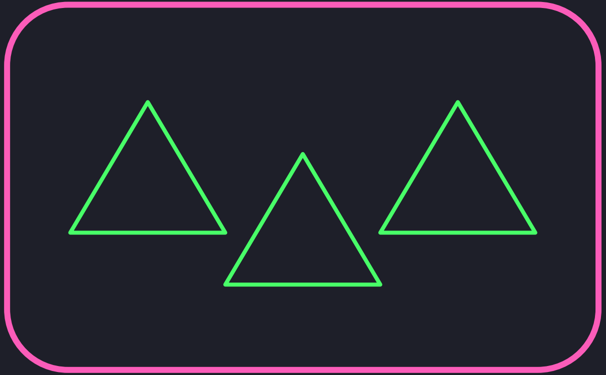
Component

object that knows what dependencies should be bundled together

AppComponent? FooComponent
BarComponent

Modules

object that knows how to instantiate objects



One step forward - to Hilt

Component

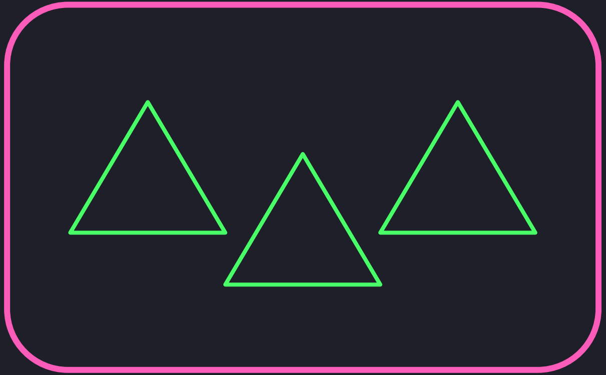
Prebuilt

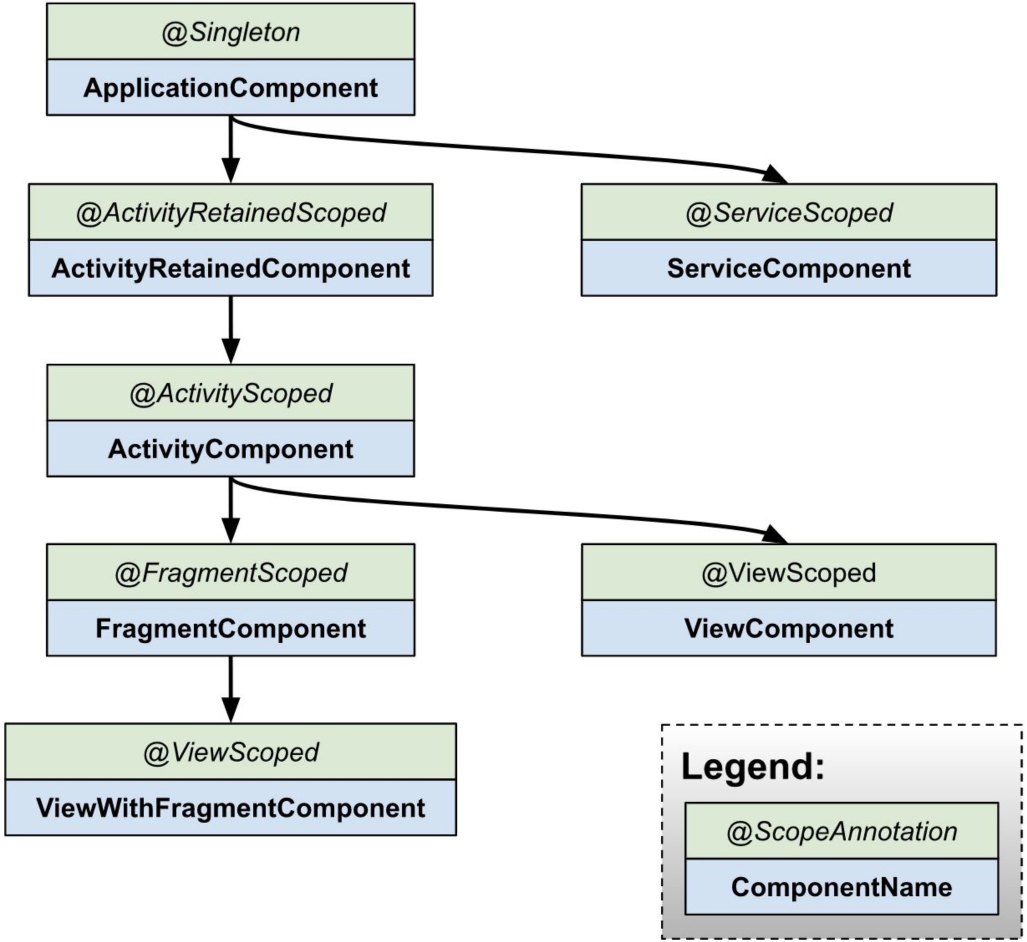
Scoped

Or custom

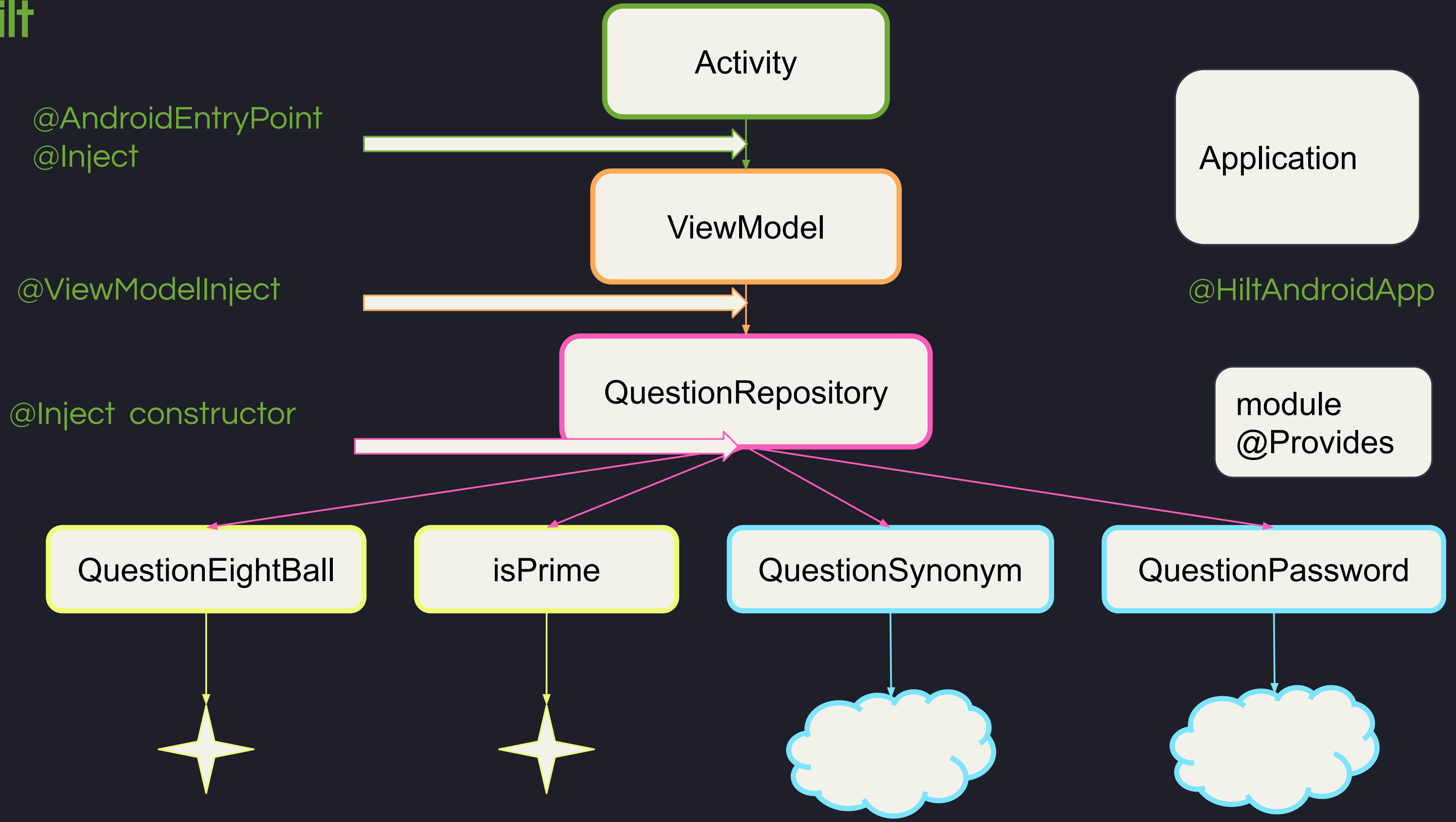
Modules

Same as dagger



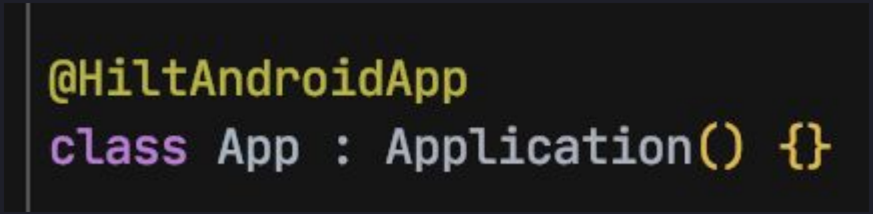

@Singleton
ApplicationComponent
@ActivityRetainedScoped
ActivityRetainedComponent
@ServiceScoped
ServiceComponent
@ActivityScoped
ActivityComponent
@FragmentScoped
FragmentComponent
@ViewScoped
ViewComponent
@ViewScoped
ViewWithFragmentComponent
Legend:
@ScopeAnnotation
ComponentName

Hilt



Activity
@AndroidEntryPoint
@Inject
ViewModel
@ViewModelInject
QuestionRepository
@Inject constructor
Application
@HiltAndroidApp
module
@Provides
QuestionEightBall
isPrime
QuestionSynonym
QuestionPassword

Hilt - Application

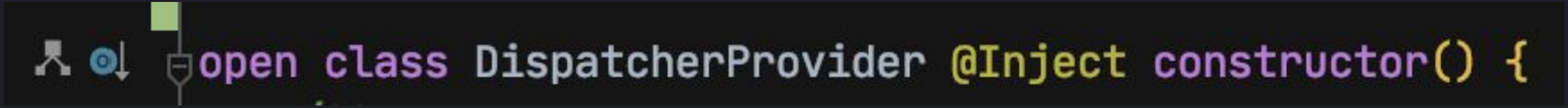


```
@HiltAndroidApp
class App : Application() {}
```

Hilt Module

When?

- Interface
- 3rd party
- Configuration

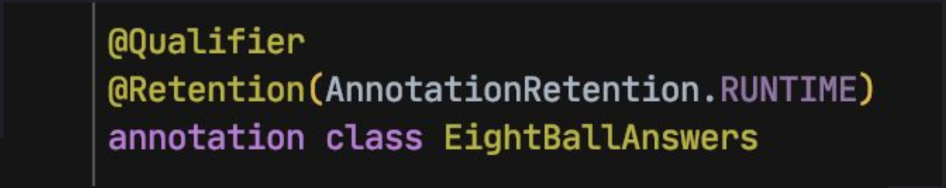


```kotlin
open class DispatcherProvider @Inject constructor() {
```

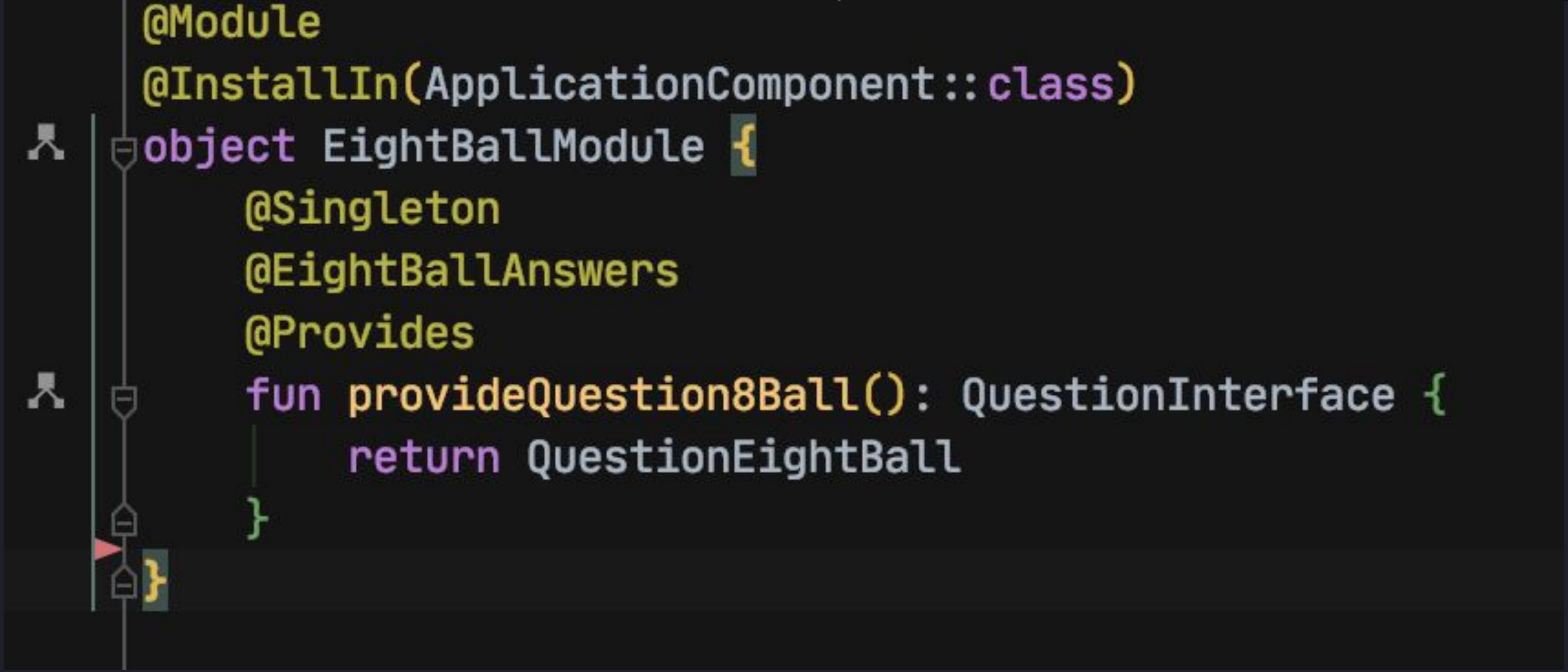
Split modules for swaps

💡 Top testing tip 💡

Hilt - Module

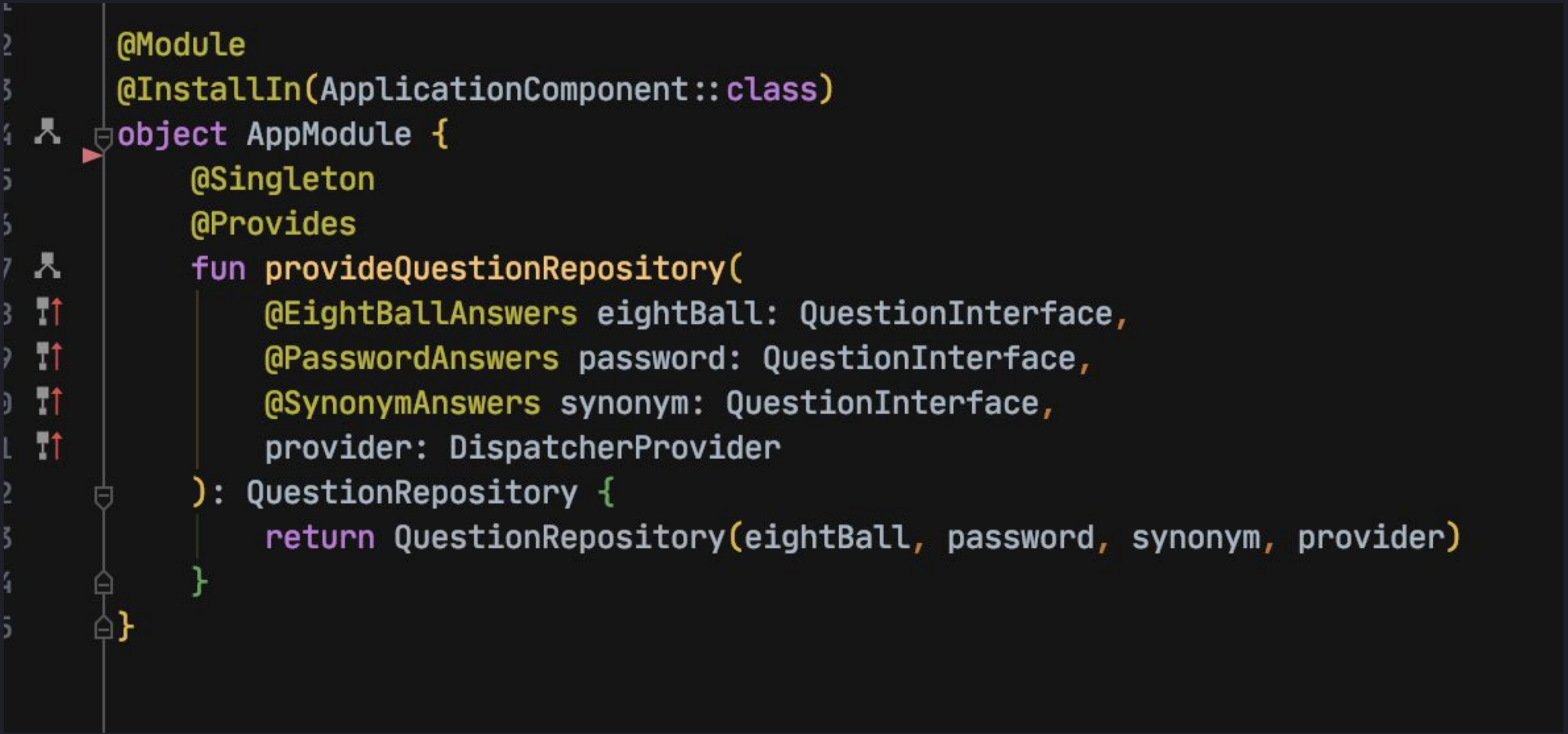


```
@Qualifier
@Retention(AnnotationRetention.RUNTIME)
annotation class EightBallAnswers
```



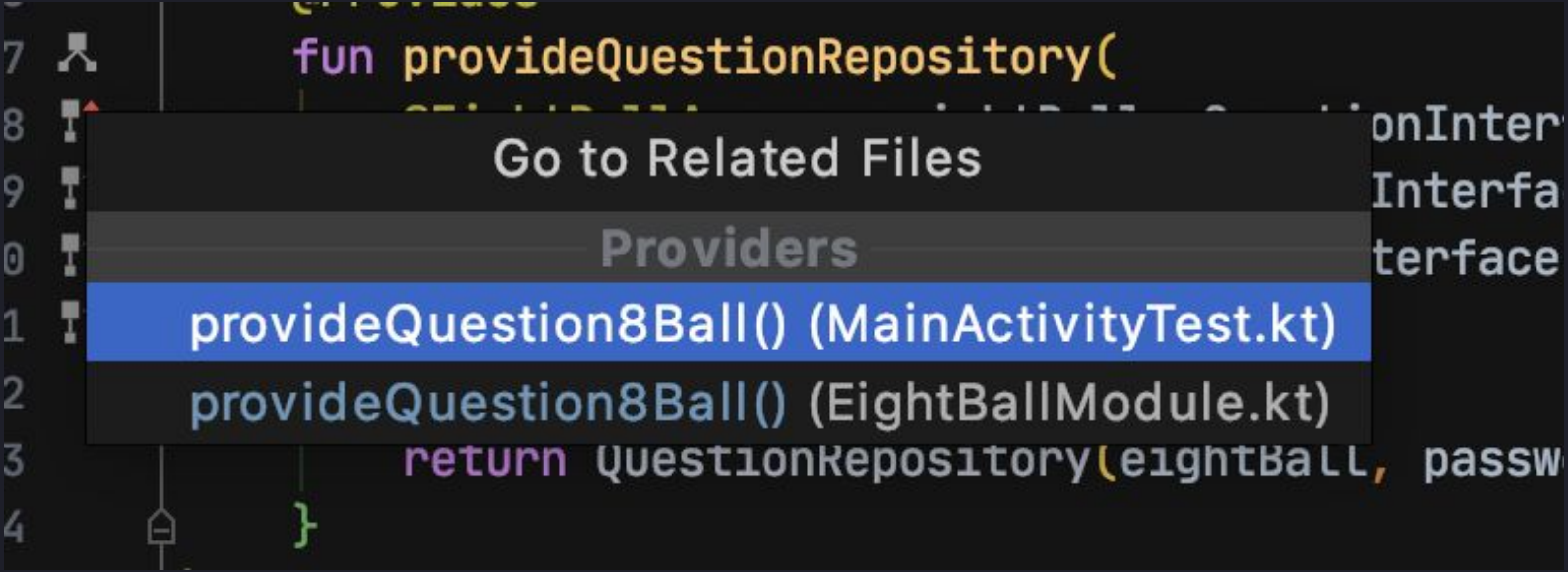
```
@Module
@InstallIn(ApplicationComponent::class)
object EightBallModule {
    @Singleton
    @EightBallAnswers
    @Provides
    fun provideQuestion8Ball(): QuestionInterface {
        return QuestionEightBall
    }
}
```

Hilt - Module



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

Hilt - IDE support



```
    fun provideQuestionRepository(
                                                              onInter
                                                              Interfa
                                                              terface
        return QuestionRepository(eightBall, passw
    }
```

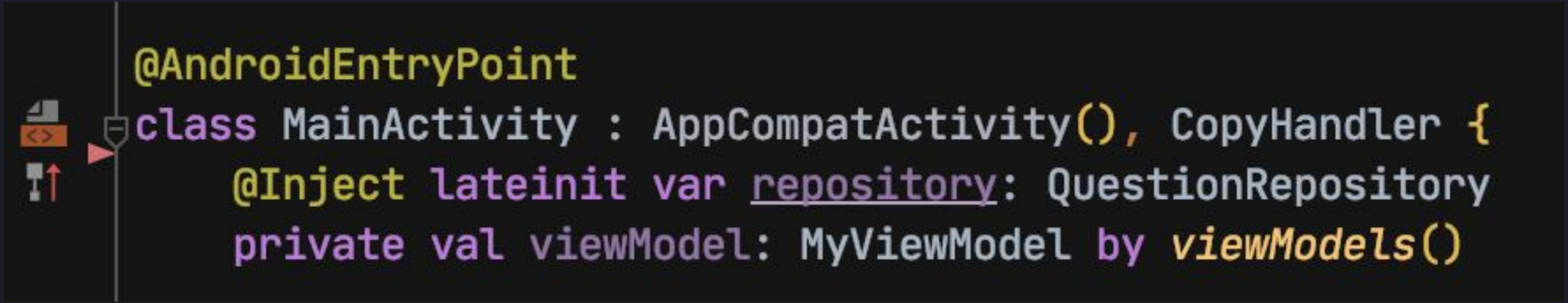
Go to Related Files

Providers

provideQuestion8Ball() (MainActivityTest.kt)

provideQuestion8Ball() (EightBallModule.kt)

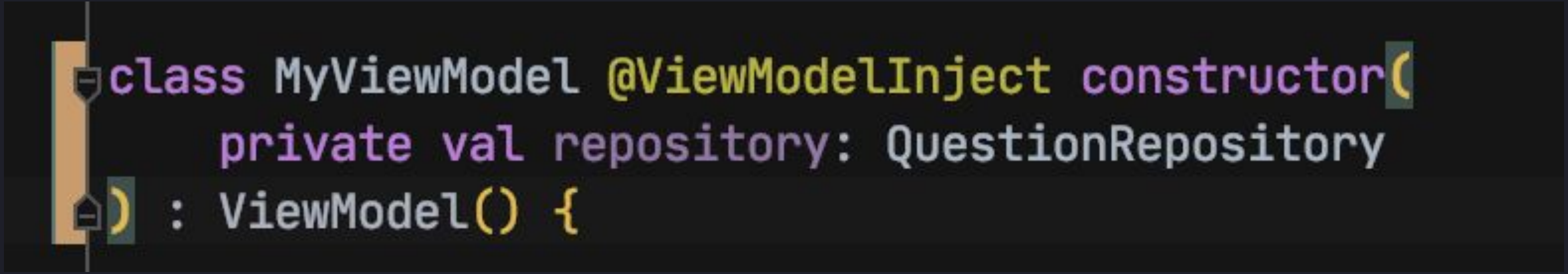
Hilt - Activity



```kotlin
@AndroidEntryPoint
class MainActivity : AppCompatActivity(), CopyHandler {
    @Inject lateinit var repository: QuestionRepository
    private val viewModel: MyViewModel by viewModels()
```

onCreate code is generated

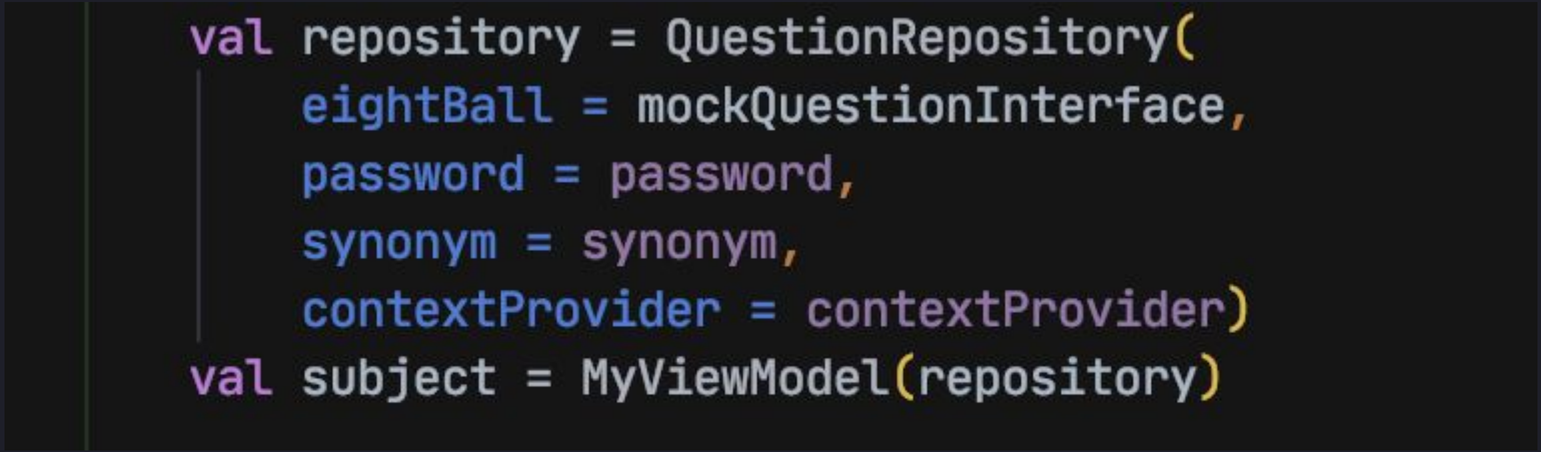
Hilt - ViewModel



```
class MyViewModel @ViewModelInject constructor(
    private val repository: QuestionRepository
) : ViewModel() {
```

Unit test

No Hilt needed constructor injection



```
val repository = QuestionRepository(
    eightBall = mockQuestionInterface,
    password = password,
    synonym = synonym,
    contextProvider = contextProvider)
val subject = MyViewModel(repository)
```

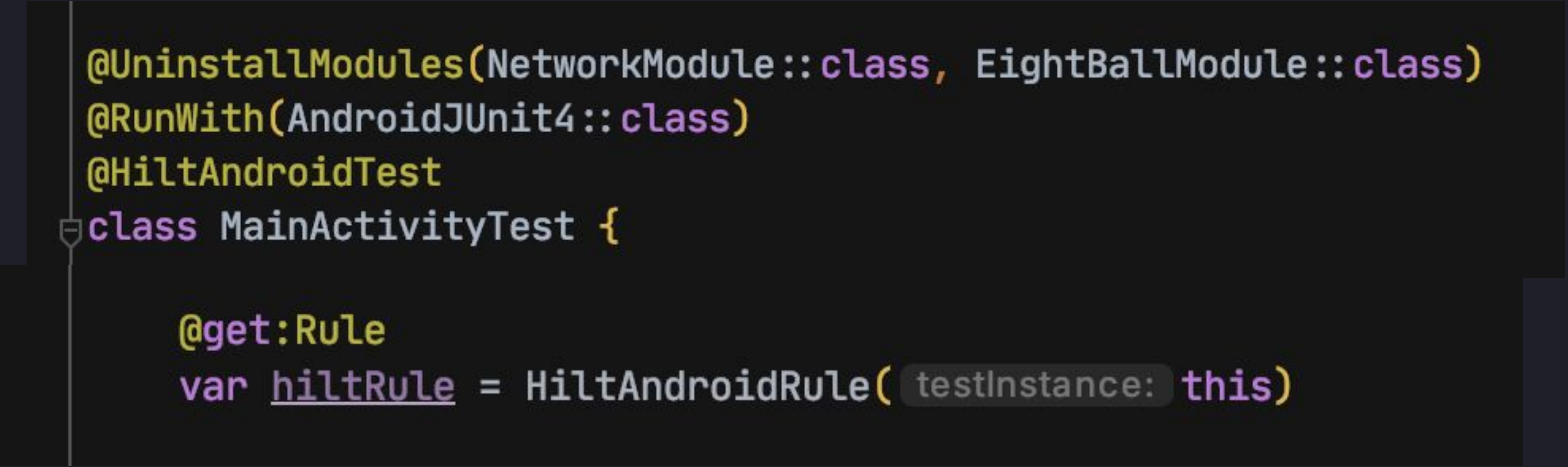
Hilt integration test

Dependencies - build.gradle

CustomTestRunner - build.gradle

See docs, codelab, gitrepo

Hilt - Integration test - annotate

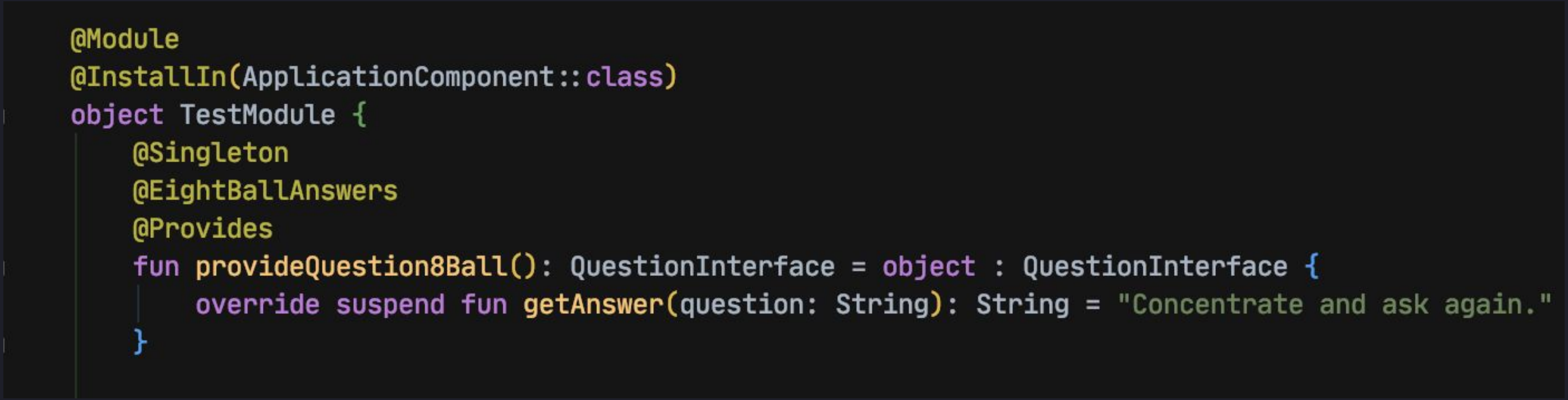


```
@UninstallModules(NetworkModule::class, EightBallModule::class)
@RunWith(AndroidJUnit4::class)
@HiltAndroidTest
class MainActivityTest {

    @get:Rule
    var hiltRule = HiltAndroidRule( testInstance: this)

```

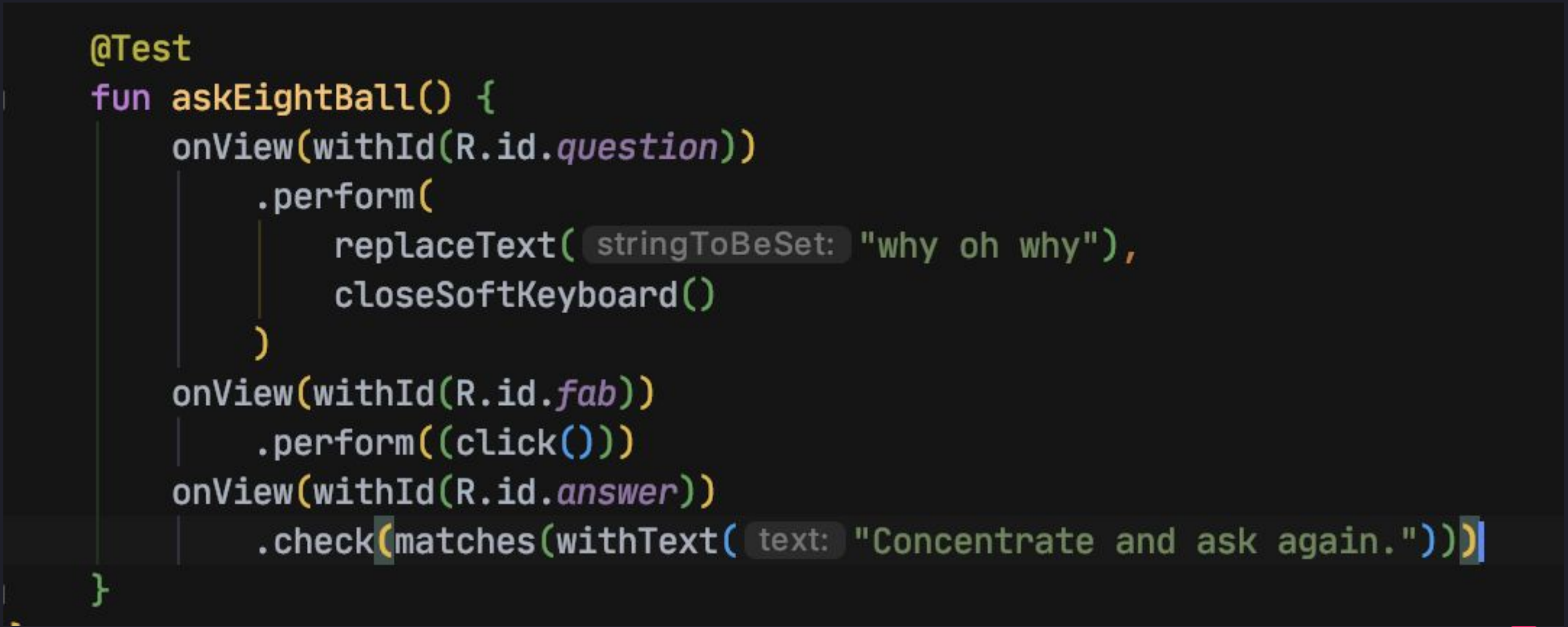
Hilt - Integration test - test module



```
@Module
@InstallIn(ApplicationComponent::class)
object TestModule {
    @Singleton
    @EightBallAnswers
    @Provides
    fun provideQuestion8Ball(): QuestionInterface = object : QuestionInterface {
        override suspend fun getAnswer(question: String): String = "Concentrate and ask again."
    }
```

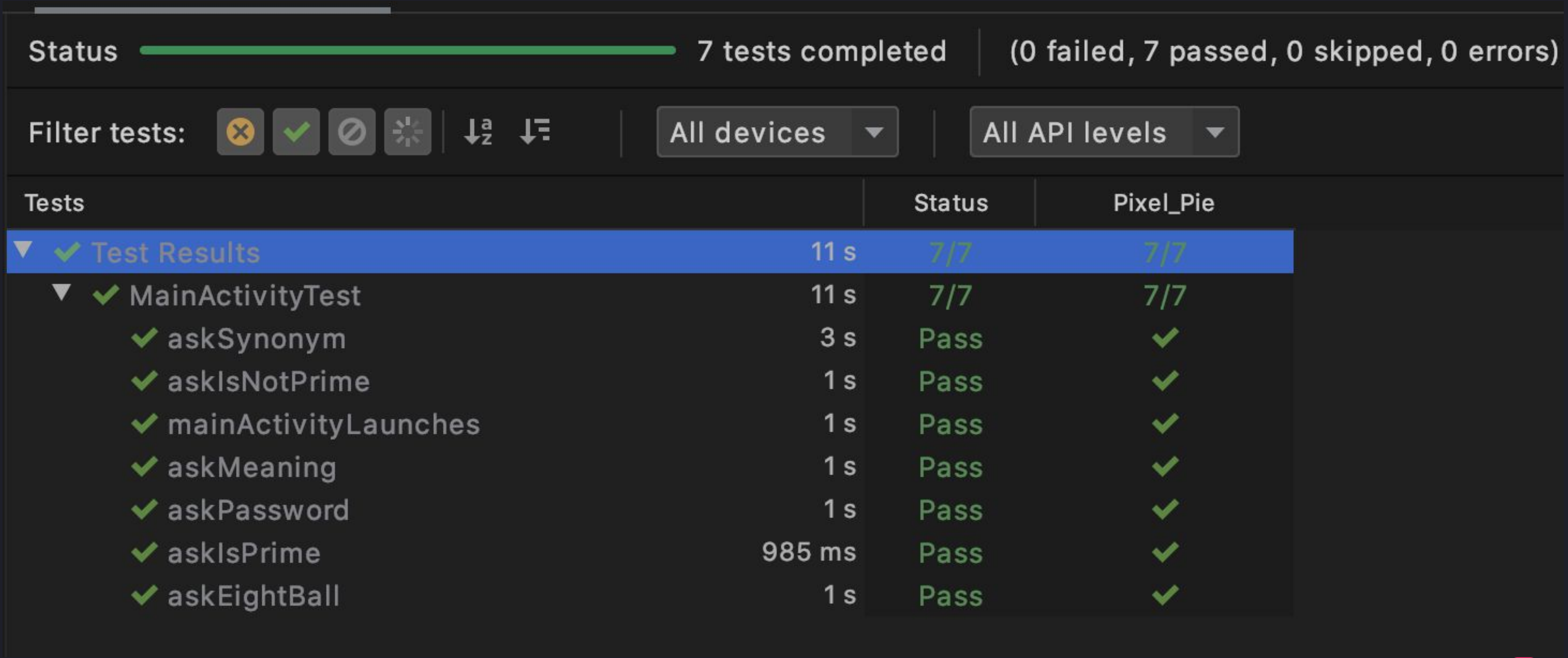
… add any @provides needed

Hilt - Integration test



```
@Test
fun askEightBall() {
    onView(withId(R.id.question))
        .perform(
            replaceText( stringToBeSet: "why oh why"),
            closeSoftKeyboard()
        )
    onView(withId(R.id.fab))
        .perform((click()))
    onView(withId(R.id.answer))
        .check(matches(withText( text: "Concentrate and ask again.")))
}
```

Hilt - Integration test



Status 7 tests completed (0 failed, 7 passed, 0 skipped, 0 errors)

Tests		Status	Pixel_Pie
▼ ✓ Test Results	11 s	7/7	7/7
▼ ✓ MainActivityTest	11 s	7/7	7/7
✓ askSynonym	3 s	Pass	✓
✓ askIsNotPrime	1 s	Pass	✓
✓ mainActivityLaunches	1 s	Pass	✓
✓ askMeaning	1 s	Pass	✓
✓ askPassword	1 s	Pass	✓
✓ askIsPrime	985 ms	Pass	✓
✓ askEightBall	1 s	Pass	✓

Hilt - Recipe

1. Add libraries
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 as necessary

Contrast and Compare

SL vs DI

Jake says… but Martin says… then reddit said

https://www.reddit.com/r/androiddev/comments/8ch4cg/dagger2_vs_koin_for_dependency_injection/

Inversion of control and complexity

vs

All dependent on the service locator object that hides how it constructs but easier to understand

And all because we can’t pass anything in the constructor of the Activity

Contrast and Compare

Manual - 0B - sortof

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

Koin - 600B

Pros

Easy to get started - easy to understand

Simple DSL

Multiple modules

No code generation = quick build

No reflection

Cons

Start up time

Run time errors

Hilt - 41B - sortof

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 - good docs - code labs - videos

Cons

Alpha

Annotation processors and code generation could make builds slower

Some limitations but active development

https://github.com/google/dagger/issues?q=is%3Aissue+is%3Aopen+label%3A%22area%3A+hilt%22

! ✓ ☺

Which one?

Really small simple app - Roll your own

Med to large app -

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

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/

References

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

https://dagger.dev/hilt/