Android: Getting Started with Dagger2

INTRODUCTION TO DEPENDENCY INJECTION & DAGGER2

Justin Inácio
ANDROID DEVELOPER

https://moducode.com

What is dependency injection?

Supplying the dependencies an object requires when creating it.

Without Dependency Injection

```kotlin
class Car {
    private val frame: Frame = Frame()
    private val wheels: Wheels = Wheels()
    private val engine: Engine = RocketEngine()

    private fun startEngine() = engine.start()
}
```

With Dependency Injection

```kotlin
class Car(private val frame: Frame,
    private val wheels: Wheels,
    private val engine: Engine){

    private fun startEngine = engine.start()
}
```

Instantiation Example

Without

```
val car: Car = Car()
```

With

```
val car: Car =
    Car(Frame(),
    Engine(), Wheels())
```

Why Use Dependency Injection?

- Class reusability
- Easier to maintain
- Easier to test
- Cleaner architecture

What is Dagger2?

Dagger2 is a dependency injection library;
a fork of the original Dagger developed by Square.

It is open source and maintained by Google.

Why Use Dagger2?

Dagger allows us to easily scope dependencies; binding single instances to certain lifecycles of our application

When building our dependencies, we only need to construct them once with Dagger. Dagger also resolves large dependency graphs easily.

Unlike it's predecessor and other dependency injection libraries, Dagger2 will generate all of its code at compile time

Constructor Injection
With and Without Dagger

Without

```kotlin
fun buildCar(): Car =
    Car(SturdyFrame(),
    Wheels(),
    RocketEngine())
```

With

```kotlin
fun buildCar(): Car =
    DaggerAppComponent
    .builder()
    .build()
    .buildCar()
```

Field Injection Without Dagger

```kotlin
class SomeActivity : AppCompatActivity() {

    private val db: DbRepo = DbRepository.get()
    private val client: OkHttpClient = OkHttpClient
            .Builder()
            .cache(Cache(cacheDir, 5 * 1024 * 1024))
            .build()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
    }
}
```
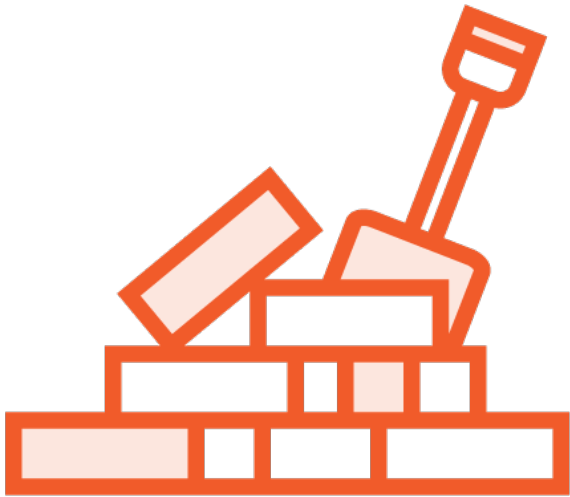
Field Injection With Dagger

```kotlin
class SomeActivity : AppCompatActivity() {

    @Inject lateinit var db: DbRepo
    @Inject lateinit var client: OkHttpClient

    override fun onCreate(savedInstanceState: Bundle?) {
        DaggerAppComponent
                .builder()
                .build()
                .injectActivity(this)
        super.onCreate(savedInstanceState)
    }
}
```
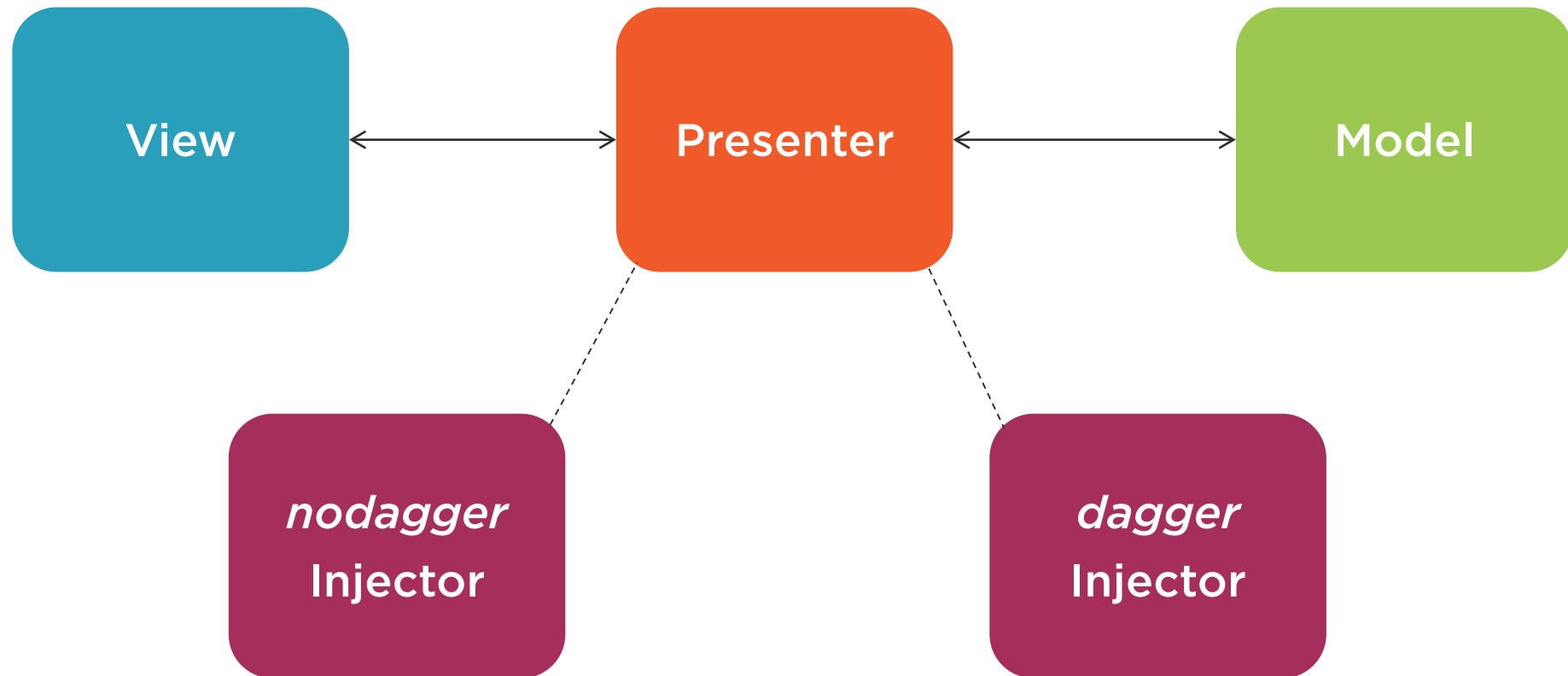
The Two Building Blocks of Dagger2

Modules
Responsible for building dependencies

Components
Responsible for holding our dependencies (modules)

Demo

Demonstration of Injector files

Summary

Dependency Injection

- How it works
- Field vs Constructor injection

Dagger2

- Helps with DI

What is a module?

In simple terms, Modules in Dagger are responsible for *providing* objects which we want to inject. They contain methods which return said objects

Visualizing Modules

Network Module

RetrofitService
Retrofit
OkHttp

Context Module

Context

Car Module

Car
Wheels
Frame
Engine

Visualizing Modules

Network Module

RetrofitService

Retrofit

OkHttp

These dependencies can now be injected...

...into other classes in our project.

Your First Module

```
class EngineModule {


}
```

Your First Module

```
@Module
class EngineModule {



}
```

Your First Module

```
@Module
class EngineModule {


    fun provideEngine(): Engine = Engine()
}
```

Your First Module

```
@Module
class EngineModule {

    @Provides
    fun provideEngine(): Engine = Engine()
}
```

A More Complex Module

```kotlin
@Module
class CarModule {

    @Provides
    fun provideEngine(): Engine = Engine()

    @Provides
    fun provideFrame(): Frame = Frame()

    @Provides
    fun provideWheels(): Wheels = Wheels()



    }
}
```

A More Complex Module

```kotlin
@Module
class CarModule {

    @Provides
    fun provideEngine(): Engine = Engine()

    @Provides
    fun provideFrame(): Frame = Frame()

    @Provides
    fun provideWheels(): Wheels = Wheels()

    @Provides
    fun provideCar(engine: Engine, wheels: Wheels, frame: Frame): Car{
        return Car(frame, wheels, engine)
    }
}
```

A *Lesser* Complex Module

```kotlin
@Module
class CarModule {

    @Provides
    fun provideCar(): Car{
      return Car(Frame(), Wheels(), Engine())
    }
}
```

Including Modules

```kotlin
@Module
class CarModule {

    @Provides
    fun provideFrame(): Frame = Frame()

    @Provides
    fun provideWheels(): Wheels = Wheels()

    @Provides
    fun provideCar(engine: Engine, wheels: Wheels, frame: Frame): Car{
        return Car(frame, wheels, engine)
    }
}
```

Including Modules

```kotlin
@Module(includes = [EngineModule::class])
class CarModule {

    @Provides
    fun provideFrame(): Frame = Frame()

    @Provides
    fun provideWheels(): Wheels = Wheels()

    @Provides
    fun provideCar(engine: Engine, wheels: Wheels, frame: Frame): Car{
        return Car(frame, wheels, engine)
    }
}
```

External Dependencies

```kotlin
@Module
class CacheModule {

    @Provides
    fun provideCache(context: Context): Cache =
            Cache(context.cacheDir, 5 * 5 * 1014)

}
```
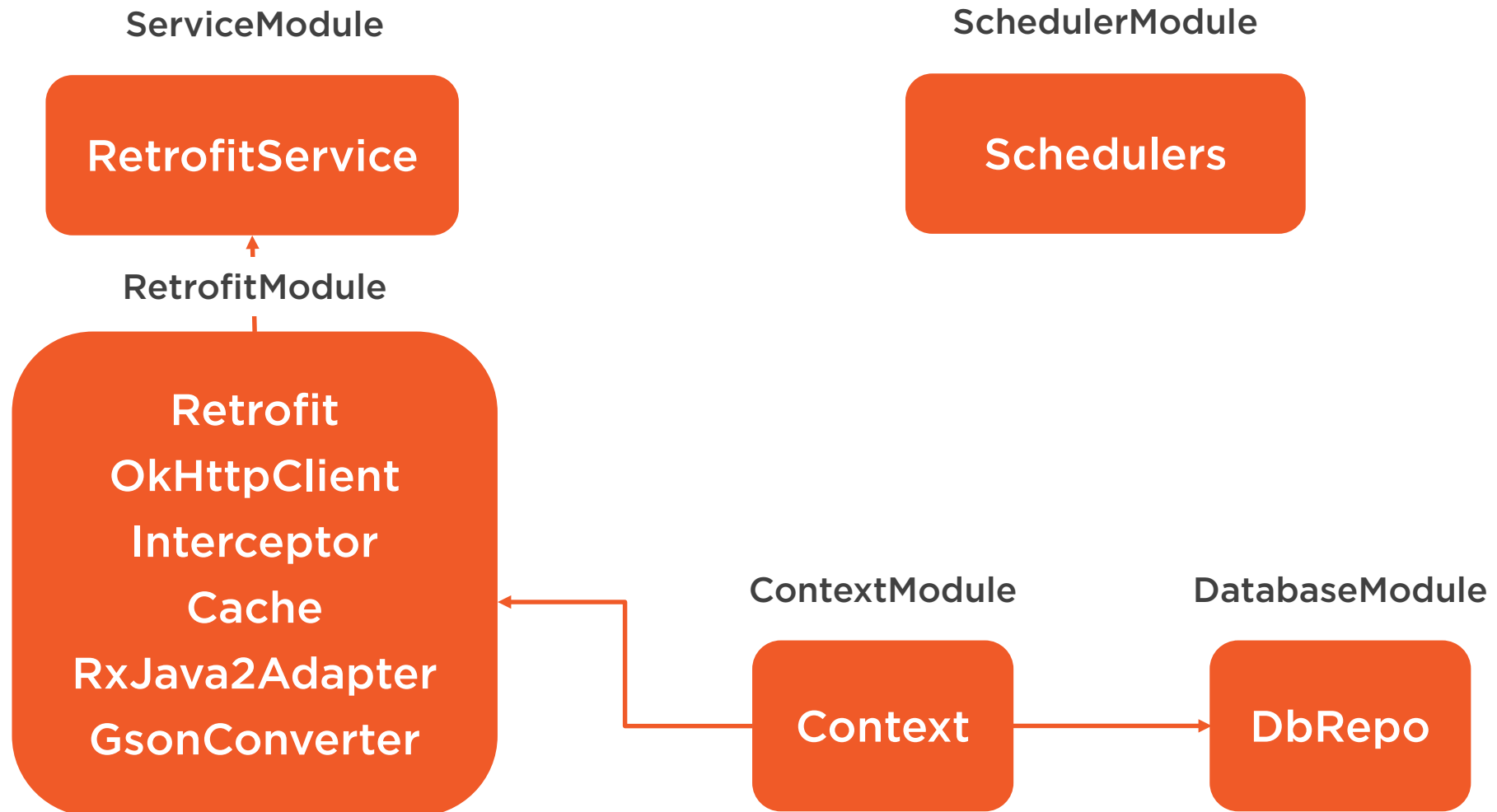
External Dependencies

```kotlin
@Module
class ContextModule(val context: Context) {

    @Provides
    fun provideContext(): Context = context
}
```

Building Our Modules

ServiceModule

RetrofitService

SchedulerModule

Schedulers

RetrofitModule

**Retrofit
OkHttpClient
Interceptor
Cache
RxJava2Adapter
GsonConverter**

ContextModule

Context

DatabaseModule

DbRepo

Demo

Demonstration of module construction

Summary

Modules

- What they are

- How to create them

- Including modules in modules

What is a Component

A component is used for *holding* our modules. We include modules, which provide dependencies, inside a component. Doing this makes our dependencies accessible via our component.

Visualizing a Component

Network Module

RetrofitService
Retrofit
OkHttp

Context Module

Context

Car Module

Car
Wheels
Frame
Engine

Visualizing a Component

AppComponent

Network Module	Context Module	Car Module
RetrofitService **Retrofit** **OkHttp**	**Context**	**Car** **Wheels** **Frame** **Engine**

Building a Component

```
interface AppComponent {


}
```

Building a Component

```
@Component
interface AppComponent {


}
```

Building a Component

```
@Component(modules = [NetworkModule::class, ContextModule::class,
CarModule::class])
interface AppComponent {


}
```

Returning Single Dependencies

```kotlin
@Component(modules = [NetworkModule::class, ContextModule::class,
CarModule::class])
interface AppComponent {

    fun okHttpClient(): OkHttpClient

}
```

Returning Single Dependencies

```kotlin
@Component(modules = [NetworkModule::class, ContextModule::class,
CarModule::class])
interface AppComponent {

    fun okHttpClient(): OkHttpClient

    fun retrofitService(): RetrofitService

    fun car(): Car
}
```

Returning Single Dependencies

```kotlin
class SomeActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

    }
}
```

Returning Single Dependencies

```kotlin
class SomeActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val component = DaggerAppComponent
                .builder()
                .build()
    }
}
```

AppComponent in Action

```kotlin
class SomeActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val component = DaggerAppComponent
                    .builder()
                    .build()


        val car = component.car()
        val retrofitService = component.retrofitService()
        val client = component.okHttpClient()
    }
}
```

AppComponent in Action

```kotlin
class SomeActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val component = DaggerAppComponent
                    .builder()
                    .networkModule(NetworkModule())

                    …
                    .build()

        val car = component.car()
        val episodeService = component.episodeService()
        val client = component.okHttpClient()
    }
}
```

AppComponent in Action

```kotlin
class SomeActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val component = DaggerAppComponent
                .builder()
                .contextModule(ContextModule(context!!))
                .build()


        val car = component.car()
        val retrofitService = component.retrofitService()
        val client = component.okHttpClient()
    }
}
```

Summary

Components
- How to include modules
- Function definitions
- Building and supplying modules
- Retrieve dependencies

Demo

Build a component with modules

Building Our App's Component