Jake Taggart

Research

1. What is Dependency Injection?

   Dependency injection is a way to create objects via an external entity. Put simply, it’s a way to use objects to supply other objects’ dependencies (arguments/fields). Similar to prototyping, it is a way to use higher level code to call upon lower level code to function. This is known as inversion of control and greatly improves the testability and oftentimes maintenance of code. Most dependency injection frameworks require reflection from the JVM. Because of this, errors that used to be thrown at compile time are now thrown at runtime. Dependency injection helps improve the use of loose coupling which is to say that separate objects now depend upon each other much less.

2. What is Dagger?

   Dagger is a compile-time dependency injection framework for java and Android. It is maintained by Google and is a great way to use dependency injection with less performance overload than most injection dependency frameworks. Dagger differs from other frameworks as it does not use reflection, but instead throws errors at compile time.

3. What if the difference in Dagger 1 vs Dagger 2?

   While Dagger 1 did not support method injection, Dagger 2 does. Dagger 2 does not support static injection while Dagger 1 does. Dagger 2 reduces the amount of configuration properties from Dagger 1 in modules. Dagger 2 does not support overrides. In Dagger 1, the dependency injection graph was built statically using reflection by ObjectGraph. In Dagger 2, the dependency injection graph is built at compile time using @Component annotated, developer-defined types. While Dagger 1 only supported @Singleton for scope, Dagger 2 allows for developers to customize scopes. Unlike Dagger 1, Dagger 2 does an implicit null check unless the @Nullable annotation is used.

4. What is a dependency graph?

   A dependency graph is a graph representing the various dependencies a collection of objects has amongst one another. Based on the mapping of the dependencies, an evaluation order can be deduced or - depending on the dependencies - not. One can actually print the dependency graph in his or her terminal if he/she so pleases.

5. What are the following annotations for in Dagger 2:

a) Inject

Inject is an annotation to denote the construction of an object and also to instantiate fields. Dagger 2 also supports method injection, but constructor and field injection are preferred.

b) Component

Component is an annotation used to group modules together. Injects and Provides together form dependencies and use an interface to form them. By using the Component annotation with the passed modules with the interface below that annotation, the entire desired object can be made from the interface in one fell swoop.

c) Module

The Module annotation is used to denote a class that is provided objects by those defined inside its class via Provides annotation. It is a way to group any of the objects denoted inside its class as an instantiable class unto itself.

d) Provides

The Provide annotation is what tells the class which objects comprise the module. Provides is always called inside a module and also allows the Inject annotation to instantiate its contents.

6. What are the main types of dependency injection and what is the difference in them?

There are three main types of dependency injection. These three types include constructor injection, setter injection, and interface injection. Constructor injection makes it so that dependencies amongst objects are provided through a class constructor. Setter injection requires the developer to exploit the setter methods of a class to inject a dependency amongst objects. Interface injection requires the developer to implement an interface which uses the setters to accept a dependency. Any dependency comes with an injector that injects that dependency into any client/object passed in. Thus interface injection is a way to inject dependency from a passed object via setter injection using an interface.