# **Dagger – 2**

[Story 16796:](http://tfsemea1.ta.philips.com:8080/tfs/TPC_Region24/CDP2/_workitems/edit/16796)[Spike] Value of using Dagger Tool - Dependency injection framework in Reference App.

Author: Ritesh Jha

Date: 26th Sep 2017

Note: This spike covers the implementation. One separate spike is needed to achieve code coverage and testing options.

# **What is Dependency Injection**

Dependency injection means a class should not configure its dependencies statically but should be configured from the outside. The general concept behind dependency injection is called *Inversion of Control.*

# **Daager 2 Introduction:**

Dagger 2 is a dependency injection framework that is built on the standard javax.inject annotations (JSR 330)

# **Daager Terminologies**

Following are the minimum annotations required for getting started.

1. **@Module** for the classes whose methods provide dependencies
2. **@Provides** for the methods within @Module classes
3. **@Inject** to request a dependency (a constructor, a field, or a method)
4. **@Component** is a bridge interface between modules and injection

# **Benefits of using Dependency Injection:**

1. Testable classes
2. Re-usable and interchangeable components
3. Based on Code-generation and not on reflection.
4. Code becomes more reusable
5. Code becomes more cleaner
6. Code becomes independent of Object creation
7. Deeper dependencies can be avoided
8. Code is generated at compile time. Code generation is optimized and happens very fast.

# **Disadvantages**

1. Every-time dependency changes, a clean build has to be done (the whole project should be compiled again)
2. No Dynamism as everything should be decided at compile time.

# **Memory**

1. The Daager-2 generates .class files that are stored just like Android .dex classes. It occupies negligible space.
2. The Dependency object injected by Daager can be scoped variables. Need not be application scoped Singleton’s always, hence efficient in terms of memory.
3. Multiple Objects can have same dependency. In case, the dependency is already created and kept ready, the object need not be created again.
4. Daager provides an option – weather singleton objects should be used or always new Objects should be created. This gives flexible control in terms of memory.
5. Local scoped Singletons can be used in Daager. As-in, the Singletons are scoped to Activity Life-Cycle. This gives better memory management.

# **Performance:**

1. Daager 2 is designed keeping Android low-end phone in consideration. Few things like battery, memory and execution speed are well addressed with Daager 2.
2. Dependency Graphs are drawn at compile time and hence the whole dependency is predicted at compile time. Runtime decisions are avoided because of this.
3. Multiple Objects can have same dependency. In case, the dependency is already created and kept ready, the execution speed increases. Again and again, same dependency requirements need not asked.
4. Since everything happens at compile time, the performance is much better compared to other Dependency frameworks.
5. The below URL provides some statistics about the performance between various Dependency Engine frameworks
   1. <http://blog.nimbledroid.com/2016/03/07/performance-of-dependency-injection-libraries.html>

# **Implementation Details**

# **Gradle dependency**

annotationProcessor **'com.google.dagger:dagger-compiler:2.10'**compile **'com.google.dagger:dagger:2.10'**

# **Feature branch created**

origin/feature/16796\_Dagger\_Spike feature

I have done the implementation for Application class and push notification module. Which covers below part:

1. RefApp component: Application class covers the construction of below:
   1. UR State
   2. AppInfra State
   3. PR State
   4. IAP State
2. Push Notification component:
   1. Construction of Push Notification.

# **Conclusion**

1. Dependency Injection is a powerful design-pattern which enforces a very good base design for any App
2. Daager-2 was specifically designed for addressing Android and works great with Android.
3. Dagger2 improves the performance and minimizes the memory footprint
4. Code organization and modularization in a better way.
5. We can remove boilerplate code from our RefApp Application class.
6. Separating our object construction and place to use it.