**Dagger Hilt**

Hilt merupakan library Jetpack terbaru yang direkomendasikan dalam dependency injection. Ia dibuat di atas Dagger yang sudah teruji dan sangat kokoh (robust). Bedanya yaitu langkah yang dilakukan lebih simpel dan dinilai lebih mudah dipelajari. Jadi dulu Dagger dikenal sebagai konsep yang cukup sulit dipelajari, selain itu proses ritual yang panjang membuat kurva belajar mempelajari Dagger ini lumayan tajam. Nah harapannya dengan adanya Hilt proses tersebut menjadi lebih mudah.

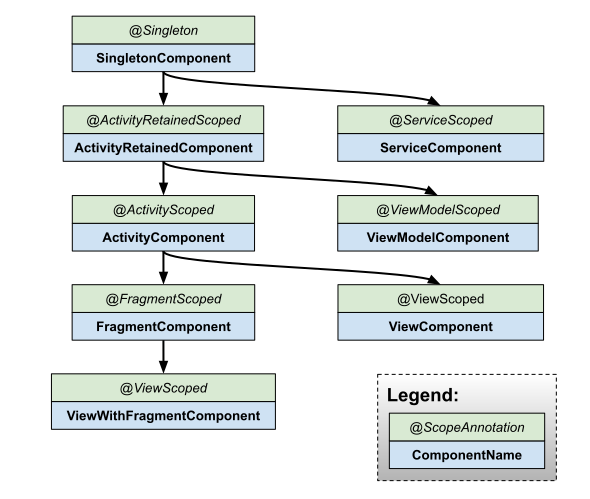
**Dependency**

Berikut ini adalah dependency yang perlu ditambahkan untuk menggunakan Dagger Hilt.

* Pada  **build.gradle(project)**
  1. buildscript {
  2. ...
  3. ext.hilt\_version = '2.36'
  5. dependencies {
  6. ...
  7. classpath 'com.google.dagger:hilt-android-gradle-plugin:$hilt\_version'
  8. }
  9. }
* Pada **build.gradle(module: app)**
  1. plugns {
  2. ...
  3. id 'kotlin-kapt'
  4. id 'dagger.hilt.android.plugin'
  5. }
  6. ...
  7. dependencies {
  8. ...
  9. implementation "com.google.dagger:hilt-android:$hilt\_version"
  10. kapt "com.google.dagger:hilt-android-compiler:$hilt\_version"
  12. }

**Catatan:**Saat ini Hilt masih dalam tahap beta, sehingga mungkin masih sering terjadi perubahan.

**Konsep Dasar**

Perbedaan dasar antara Dagger dan Hilt adalah Hilt sudah menyediakan component dan scope yang bisa langsung Anda gunakan. Berbeda dengan Dagger yang mengharuskan Anda membuatnya secara manual. Yang menarik yaitu Component ini sudah terintegrasi dengan lifecycle di dalam Android, sehingga Anda tak perlu mengaturnya secara manual. Berikut ini adalah hirarki dari beberapa Component beserta Scope yang bisa Anda gunakan:  
[](https://www.dicoding.com/academies/165/tutorials/10404?from=10399)

**Catatan:**  
Pastikan gunakan annotation yang sesuai pada diagram ini ya, misalnya jika Anda menggunakan **ActivityComponent**maka Anda hanya bisa menggunakan scope **@ActivityScoped**.

Baik supaya lebih paham mari simak kode yang sama pada modul Dagger dan kita ubah dengan menggunakan Hilt sehingga hasilnya menjadi seperti ini:

1. **@Module**
2. **@InstallIn(SingletonComponent::class)**
3. **class AppModule {**
4. **@Singleton**
5. **@Provides**
6. **fun provideEngine(@ApplicationContext context: Context): Engine = Engine(context)**
7. **}**
8. **@HiltAndroidApp**
9. **open class MyApplication : Application() {**
10. **}**
11. **@AndroidEntryPoint**
12. class MainActivity : AppCompatActivity() {
14. **@Inject**
15. lateinit var car: Car
17. override fun onCreate(savedInstanceState: Bundle?) {
18. super.onCreate(savedInstanceState)
19. setContentView(R.layout.activity\_main)
21. car.start()
22. }
23. }
25. **@Singleton**
26. class Car **@Inject** constructor(private val engine: Engine) {
27. fun start() {
28. engine.start()
29. }
30. }
32. class Engine(val context: Context) {
33. fun start() {
34. println(context.applicationContext.getString(R.string.engine\_start))
35. }
36. }

**Catatan:**  
Anda juga dapat mencoba dan menjalankan kode di atas pada Android Studio, dengan syarat menambahkan library Dagger Hilt terlebih dahulu.

Nah mari kita simak bagian bagian tersebut satu per satu.

**@Module dan @InstallIn**

1. @Module
2. @InstallIn(SingletonComponent::class)
3. class AppModule {
4. ...
5. }

Sama seperti Dagger, untuk memberitahu suatu kelas adalah Module Anda cukup menambahkan annotation @Module. Bedanya yaitu di sini terdapat anotasi @InstallIn untuk memasukkan module ini ke Component yang diinginkan. Berbeda dengan Dagger yang merupakan kebalikannya, yaitu Component menentukan Module apa saja yang dimasukkan.

**Context Injection dengan @ApplicationContext**

1. @Singleton
2. @Provides
3. fun provideEngine(@ApplicationContext context: Context): Engine = Engine(context)

Jika pada Dagger Anda harus menggunakan @BindInstance dan memasukkannya dari luar, maka pada Hilt Anda cukup menggunakan annotation @ApplicationContext saja pada saat membutuhkan Context. Selain @ApplicationContext, qualifier yang lainnya yang dapat Anda pakai yaitu @ActivityContext.  
  
Kemudian fungsi untuk mensuplai Object masih sama dengan menggunakan @Provides. Selain itu juga bisa menggunakan @Binds untuk yang abstract. Selanjutnya perhatikan juga pada scope yang digunakan, karena Anda menggunakan SingletonComponent maka scope yang digunakan yaitu @Singleton.

Selain itu Anda juga mensuplai melalui constructor Injection sama seperti pada Dagger dan tidak ada perbedaan dalam hal ini.

**Inject ke Application dengan @HiltAndroidApp**

Supaya Anda bisa menggunakan Hilt di semua kelas, Anda dapat membuat Custom Application untuk memanggil Hilt. Kelas ini akan dijadikan sebagai base Application dengan menambahkan name pada AndroidManifest.

1. **@HiltAndroidApp**
2. open class MyApplication : Application() {
3. }

Bedanya dengan Dagger yaitu Anda tidak perlu *create component* di sini karena component pada Hilt sudah disediakan. Anda cukup menambahkan anotasi @HiltAndroidApp untuk men-*trigger*Hilt men-*generate*kode untuk dependency yang didefinisikan.

**@AndroidEntryPoint**

1. @AndroidEntryPoint
2. class MainActivity : AppCompatActivity() {
4. @Inject
5. lateinit var engine: Engine
7. override fun onCreate(savedInstanceState: Bundle?) {
8. super.onCreate(savedInstanceState)
9. setContentView(R.layout.activity\_main)
11. val car = Car(engine)
12. car.start()
13. }
14. }

Berbeda dengan Dagger, Anda tak perlu memanggil Component yang ada di MyApplication untuk melakukan injection ke suatu class. Namun cukup dengan menambahkan anotasi @AndroidEntryPoint pada kelas yang ingin di-*inject*. Selain pada Activity, Anda juga dapat menambahkannya pada Fragment, View, Service, dan juga BroadcastReceiver.

Yang perlu diperhatikan yaitu Anda jika menambahkan anotasi tersebut pada suatu class, maka Anda juga perlu menambahkan anotasi pada kelas yang menaunginya juga. Misalnya jika Anda menambahkan anotasi pada Fragment, maka kamu harus menambahkan anotasi pada Activity yang menggunakan Fragment tersebut juga.

**Integrasi Hilt dengan Jetpack**

Karena Hilt sudah terintegrasi dengan Jetpack Anda dapat meng-*inject*beberapa komponen dalam Jetpack dengan lebih mudah. Untuk saat ini komponen yang di-*support*yaitu ViewModel dan WorkManager.

**- Injection pada ViewModel**Untuk melakaukan injection ViewModel, Anda bisa mem-*provide*-nya cukup hanya dengan menambahkan anotasi @HiltViewModel pada class dan @Inject pada constructor ViewModel seperti berikut:

1. @HiltViewModel
2. class HomeViewModel @Inject constructor(private val tourismUseCase: TourismUseCase) : ViewModel() {
3. ...
4. }

Kemudian pada activity/fragment cukup menambahkan properties delegation by viewModels untuk melakukan injection seperti berikut:

1. @AndroidEntryPoint
2. class HomeFragment : Fragment() {
3. private val viewModel: HomeViewModel by viewModels()
4. }

Simpel bukan? Di sini Anda tidak perlu menggunakan ViewModelFactory lagi. Bahkan Anda juga tidak perlu membuat multi-bindings seperti yang ada di Dagger, seperti mulai dari membuat ViewModelKey sampai menggunakan IntoMap pada Module. Semua ini sudah diatur oleh Dagger Hilt. Mantap!

**- Injection pada WorkManager**  
Sedangkan untuk Injection pada WorkManager dapat Anda lakukan dengan menggunakan anotasi @HiltWorker pada class dan @AssistedInject pada constructor. Selain itu juga anotasi @Assisted untuk masing-masing parameternya.

1. @HiltWorker
2. class ExampleWorker @AssistedInject constructor(
3. @Assisted appContext: Context,
4. @Assisted workerParams: WorkerParameters,
5. workerDependency: WorkerDependency
6. ) : Worker(appContext, workerParams) { ... }

Kasusnya hampir sama dengan ViewModel, di sini Anda tidak perlu membuat Worker Factory dan assisted inject secara manual lagi. Semuanya sudah disediakan oleh Hilt yang sudah terintegrasi dengan Jetpack ini.

Selamat! Anda telah selangkah lebih maju untuk belajar menggunakan library Dagger Hilt untuk mempermudah proses Anda dalam membuat dependency injection. Seperti biasa kami menyarankan Anda mendalami lebih jauh tentang Hilt melalui tautan di bawah ini:

* [Documentation Dagger Hilt](https://dagger.dev/hilt/)
* [Documentation Android : Dependency injection with Hilt](https://developer.android.com/training/dependency-injection/hilt-android)
* [Dagger Hilt Cheat sheet](https://developer.android.com/images/training/dependency-injection/hilt-annotations.pdf)