Hilt & Dagger Annotations

v2.33 - @AndroidDev

Annotation	Usage	Code Sample
@HiltAndroidApp	Kicks off Hilt code generation. Must annotate the Application class.	`@HiltAndroidApp` `class MyApplication : Application() { ... }`
@AndroidEntryPoint	Adds a DI container to the Android class annotated with it. This requires using Hilt's Gradle Plugin.	`@AndroidEntryPoint` `class MyActivity : AppCompatActivity() { ... }`

@Inject	Constructor Injection. Tells which constructor to use to provide instances and which dependencies the type has.	`class AnalyticsAdapter @Inject constructor(`
		`  private val service: AnalyticsService`
		`) { ... }`

	Field injection. Populates fields in @AndroidEntryPoint annotated classes.
	Fields cannot be private.
		`@AndroidEntryPoint`
		`class MyActivity : AppCompatActivity() {`
		`  @Inject lateinit var adapter: AnalyticsAdapter`
		`  ...`
		`}`

@HiltViewModel	Tells Hilt how to provide instances of an Architecture Component ViewModel.	`@HiltViewModel`
		`class MyViewModel @Inject constructor(`
		`  private val adapter: AnalyticsAdapter,`
		`  private val state: SavedStateHandle`
		`): ViewModel() { ... }`

@Module	Class in which you can add bindings for types that cannot be constructor injected.	`@InstallIn(SingletonComponent::class)`
		`@Module`
		`class AnalyticsModule { ... }`

@InstallIn	Indicates in which Hilt-generated DI containers (`SingletonComponent` in the code) module bindings must be available.	`@InstallIn(SingletonComponent::class)`
		`@Module`
		`class AnalyticsModule { ... }`

@Provides	Adds a binding for a type that cannot be constructor injected:
	- Return type is the binding type.
	- Parameters are dependencies.
	- Every time an instance is needed, the function body is executed if the type is not scoped.
		`@InstallIn(SingletonComponent::class)`
		`@Module`
		`class AnalyticsModule {`
		
		`  @Provides`
		`  fun providesAnalyticsService(`
		`  converterFactory: GsonConverterFactory`
		`  ): AnalyticsService {`
		`  return Retrofit.Builder()`
		`  .baseUrl("https://example.com")`
		`  .addConverterFactory(converterFactory)`
		`  .build()`
		`  .create(AnalyticsService::class.java)`
		`  }`
		`}`

@Binds	Shorthand for binding an interface type:
	- Methods must be in a module.
	- @Binds annotated methods must be abstract.
	- Return type is the binding type.
	- Parameter is the implementation type.
		`@InstallIn(SingletonComponent::class)`
		`@Module`
		`abstract class AnalyticsModule {`
		
		`  @Binds`
		`  abstract fun bindsAnalyticsService(`
		`  analyticsServiceImpl: AnalyticsServiceImpl`
		`  ): AnalyticsService`
		`}`

Scope Annotations:
@Singleton
@ActivityScoped
...
	Scoping object to a container.
	The same instance of a type will be provided by a container when using that type as a dependency, for field injection, or when needed by containers below in the hierarchy.
		`@Singleton`
		`class AnalyticsAdapter @Inject constructor(`
		`  private val service: AnalyticsService`
		`) { ... }`

Qualifiers for predefined Bindings:
@ApplicationContext
@ActivityContext
	Predefined bindings you can use as dependencies in the corresponding container.
	These are qualifier annotations.
		`@Singleton`
		`class AnalyticsAdapter @Inject constructor(`
		`  @ApplicationContext val context: Context`
		`  private val service: AnalyticsService`
		`) { ... }`

@Entry Point	Obtain dependencies in classes that are either not supported directly by Hilt or cannot use Hilt.
	The interface annotated with @EntryPoint must also be annotated with @InstallIn passing in the Hilt predefined component from which that dependency is taken from.
	Access the bindings using the appropriate static method from EntryPointAccessors passing in an instance of the class with the DI container (which matches the value in the @InstallIn annotation) and the entry
		`class MyContentProvider(): ContentProvider {`
		
		`  @InstallIn(SingletonComponent::class)`
		`  @EntryPoint`
		`  interface MyContentProviderEntryPoint {`
		`  fun analyticsService(): AnalyticsService`
		`  }`
		
		`  override fun query(...): Cursor {`
		`  val appContext =`
		`  context?.applicationContext`
		`  ?: throw IllegalStateException()`
		`  val entryPoint =`
		`  EntryPointAccessors.fromApplication(`
		`  appContext,`
		`  MyContentProviderEntryPoint::class.java)`
		`  val analyticsService =`
		`  entryPoint.analyticsService()`
		`  ...`
		`  }`
		`}`

For more information about DI, Hilt and Dagger: https://d.android.com/dependency-injection