**Hilt**

Sadrzaj

[Osnovne informacije o Hiltu i ukljucivanje u projekat 2](#_Toc121493798)

[Dodavanje dependency-ja 2](#_Toc121493799)

[Hilt application klasa **(@HiltAndroidApp)** 3](#_Toc121493800)

[Definisanje Hilt veza **(@Inject)** 4](#_Toc121493801)

[Hilt moduli (**@Module**) 5](#_Toc121493802)

[**@Binds** 5](#_Toc121493803)

[**@Provides** 6](#_Toc121493804)

[Dobavljanje vise instanci istog tipa **(@Qualifier)** 6](#_Toc121493805)

[Generisane komponente za Android klase 7](#_Toc121493806)

[Hijerarhija komponenti 10](#_Toc121493807)

[Podrazumevana vezivanja komponenti (Component Default Bindings) 10](#_Toc121493808)

[Ubacivanje veza u klase koje nisu podrzane od strane Hilt-a **(@EntryPoint)** 11](#_Toc121493809)

[Hilt I Dagger 12](#_Toc121493810)

# Osnovne informacije o Hiltu i ukljucivanje u projekat

**Hilt** predstavlja Android Dependency Injection (skr. DI) biblioteku koja smanjuje sablonski kod koji se javlja manuelnom implementacijom unutar projekta. Koristeci manuelni DI, zahteva se kreiranje svake klase I njenih zavisnih klasa rucno i koriscenje container-a za ponovno koriscenje I upravljanje zavisnim (dependency) klasama.

Hilt pruza standardan nacin za koriscenje DI unutar aplikacije time sto pruza container objekat za svaku Android klasu u projektu I upravlja njihovim zivotnim ciklusima automatski. Hilt je napravljen na osnovu popularne biblioteke **Dagger** u cilju iskoriscenja benefita koje on nudi:

* poput compile-time korektnosti
* runtime perfomansi
* podrske koje Android Studio nudi za Dagger komponentu.

## Dodavanje dependency-ja

Prvi korak jeste dodavanje plugina u **build.gradle** aplikacije koja se razvija:

|  |
| --- |
| *plugins {   ...   id("com.google.dagger.hilt.android") version "2.44" apply false }* |

Nakon toga, sledeci korak je ukljucivanje dependency-ja I aktiviranje plugina koji je dodat u prethodnom koraku:

|  |
| --- |
| *plugins {   kotlin("kapt")   id("com.google.dagger.hilt.android") }  android {   ... }  dependencies {   implementation("com.google.dagger:hilt-android:2.44")   kapt("com.google.dagger:hilt-android-compiler:2.44") }  // Allow references to generated code kapt {   correctErrorTypes = true }* |

Hilt koristi Java 8 funkcionalnosti i da bi se omogucila Java 8 u projektu, potrebno je dodati sledece linije u **app/build.gradle** fajlu:

|  |
| --- |
| *android {   ...   compileOptions {     sourceCompatibility = JavaVersion.VERSION\_1\_8     targetCompatibility = JavaVersion.VERSION\_1\_8   } }* |

# Hilt application klasa **(@HiltAndroidApp)**

Sve aplikacije koje koriste Hilt moraju imati Application klasu koja je anotirana sa **@HiltAndroidApp** anotacijom. Ova anotacija izaziva generisanje Hilt koda, ukljucujuci I osnovnu klasu za aplikaciju koja ce sluziti kao **dependency container** na nivou aplikacije.

|  |
| --- |
| ***@HiltAndroidApp*** *class ExampleApplication : Application() { ... }* |

Jednom kada je Hilt podesen unutar Application klase I application level komponenta je dostupna, Hilt moze pruziti dependency klase drugim Android klasa koje imaju **@AndroidEntryPoint** anotaciju.

|  |
| --- |
| ***@AndroidEntryPoint*** *public class ExampleActivity extends AppCompatActivity { ... }* |

Hilt trenutno podrzava sledece Android klase:

* Application (koristeci **@HiltAndroidApp**)
* ViewModel (koristeci **@HiltViewModel**)
* Activity
* Fragment
* View
* Service
* BroadcastReceiver

Ako anotiramo Android klase sa **@AndroidEntryPoint**, takodje moramo anotirati Android klase koje zavise od njega. Npr. ako anotiramo fragment, takodje moramo anotirati sve activity-je koji koriste taj fragment.

**@AndroidEntryPoint** generise individualnu Hilt komponentu za svaku Android klasu u projektu. Ove komponente mogu primati dependency-je od njihove odredjene parent klase kao sto je opisano u hijerarhiji komponenti. Da bi se dobio objekat odredjene komponente, potrebno je koristiti **@Inject** anotaciju da bi se izvrsila field injection.

|  |
| --- |
| ***@AndroidEntryPoint*** *public class ExampleActivity extends AppCompatActivity {* ***@Inject*** *AnalyticsAdapter analytics;   ... }* |

Klase koje Hilt inject-uje mogu imati druge base klase koje takodje koriste injection. Ove klase ne trebaju anotaciju **@AndroidEntryPoint** ukoliko su apstraktne.

## Definisanje Hilt veza **(@Inject)**

Da bi se izvrsio **field injection**, Hilt mora da zna kako da obezbedi instance nepotrebnih dependency-ja iz odgovarajuce komponente. Veza (**binding**) sadrzi informacije potrebne za ovo obezbedjivanje instance odredjenog tipa dependency-ja.

Jedan od nacina da se ovo postigne jeste **constructor injection**. Koriscenje **@Inject** anotacije nad konstruktorom klase govori Hilt kako da obezbedi instancu te klase:

|  |
| --- |
| *public class AnalyticsAdapter {    private final AnalyticsService service;* ***@Inject*** *AnalyticsAdapter(AnalyticsService service) {     this.service = service;   } }* |

Parametri anotiranog konstruktora klase su “dependency” za tu klasu. U gornjem primeru, AnalyticsAdapter ima AnalyticsService kao dependency I samim tim Hilt mora znati kako da obezbedi instancu tog servisa.

# Hilt moduli (**@Module**)

U nekim slucajevima **constructor injection** nije moguc. Ovo se moze desiti iz razlicitih razloga: npr ne mozemo odraditi constructor injection interfejsa. Takodje ne mozemo odraditi ovaj tip injection-a nad klasama koje dolaze iz eksterne biblioteke ili druge aplikacije. U ovim slucajevima se koriste Hilt moduli za povezivanje informacija.

**Hilt modul** je klasa koja je anotirana sa **@Module**. Kao kod dagger modula, ona informise Hilt kako da obezbedi instance odredjenog tipa. Za razliku od Dagger modula, Hilt moduli moraju da budu anotirani sa **@InstallIn** u cilju obavestavanja Hilt koje Android klase svaki modul mogu koristiti.

## **@Binds**

Uzmimo u obzir predjasnji primer sa AnalyticsService interfejsom. Ukoliko je u pitanju interfejs, koriscenje constructor injectiona nije moguce. Umesto toga potrebno je Hiltu obezbediti informaciju o vezi kreiranjem abstraktne funkcije anotirane sa @Binds anotacijom unutar Hilt modula.

**@Binds** anotacija govori Hiltu koju implementaciju da koristi kada je potrebna instance odredjenog interfejsa.

|  |
| --- |
| *public interface AnalyticsService {   void analyticsMethods(); }  // Constructor-injected, because Hilt needs to know how to // provide instances of AnalyticsServiceImpl, too. public class AnalyticsServiceImpl implements AnalyticsService {   ...* ***@Inject*** *AnalyticsServiceImpl(...) {     ...   } }* ***@Module @InstallIn(ActivityComponent.class)*** *public abstract class AnalyticsModule {* ***@Binds*** *public abstract AnalyticsService bindAnalyticsService(     AnalyticsServiceImpl analyticsServiceImpl   ); }* |

Anotirana funkcija pruza sledece informacije Hiltu:

* Povratni tip funkcije govori Hiltu kog interfejsa funkcija pruza instancu
* Parametri funkcije govore Hiltu koju implementaciju treba obezbediti.

Hilt modul AnalyticsModule je anotiran sa **@InstallIn**(ActivityComponent::class) zato sto zelimo da Hilt inject-uje ovaj depenendcy unutar ExampleActivity-ja. Ova anotacija znaci da su svi dependency-ji iz modula AnalyticsModule dostupni svim aktivitijima u aplikaciji.

## **@Provides**

Interfejsi nisu jedini slucaj kada ne mozemo koristiti constructor injection, vec se ta situacija desava I kada ukljucujemo odredjene klase iz eksterne biblioteke (poput Retrofita, OkHttpClient, RoomDatabase) ili u slucaju kada instance pravimo uz pomoc BuilderPattern-a.

Razmatrajuci prethodni primer, ukoliko nemamo direktno vlasnistvo nad AnalyticsService klasom, jedini nacin da kazemo Hiltu da treba da obezbedi instancu ovog tipa jeste kreiranje funkcije unutar Hilt modula I anotiranje sa **@Provides.**

Anotirana funkcija pruza sledece informacije Hiltu:

* Povratni tip funkcije govori Hiltu koji tip instance funkcija kreira.
* Parametri funkcije predstavljaju spisak dependency-ja koje su toj funkciji potrebne
* Telo funkcije govori Hiltu kako da obezbedi instancu odgovarajuceg tipa. Hilt izvrsava telo funkcije svaki put kada zeli da obezbedi instancu tog tipa

|  |
| --- |
| ***@Module @InstallIn(ActivityComponent::class)*** *object AnalyticsModule {* ***@Provides*** *fun provideAnalyticsService(     // Potential dependencies of this type   ): AnalyticsService {       return Retrofit.Builder()                .baseUrl("https://example.com")                .build()                .create(AnalyticsService::class.java)   }* } |

## Dobavljanje vise instanci istog tipa **(@Qualifier)**

U situacijama kada je potrebno da Hilt kreira instance razlicitih implementacija istog tipa kao dependency, Hiltu moramo dostaviti vise razlicitih veza. Ovo je moguce sa **Qualifiers** anotacijom. Ovu anotaciju koristimo kada zelimo da specificiramo vezu za tip instance koja ima vise razlicitih definisanih implementacija.

Kao primer mozemo uzeti sledece: Ukoliko zelimo da intercept-ujemo pozive ka AnalyticsService-u, mozemo koristiti OkHttpClient objekat sa interceptorom. Za ostale service, mozda cemo morati da koristimo druge interceptere na drugaciji nacin. U ovom slucaju moramo reci Hiltu kako da pruzi dve razlicite implementacije OkHttpClient-a.

|  |
| --- |
| **@Qualifier** @Retention(AnnotationRetention.BINARY) annotation class AuthInterceptorOkHttpClient  **@Qualifier** @Retention(AnnotationRetention.BINARY) annotation class OtherInterceptorOkHttpClient  **@Module @InstallIn(SingletonComponent::class)** object NetworkModule {  **@AuthInterceptorOkHttpClient   @Provides**   fun provideAuthInterceptorOkHttpClient(     authInterceptor: AuthInterceptor   ): OkHttpClient {       return OkHttpClient.Builder()                .addInterceptor(authInterceptor)                .build()   }  **@OtherInterceptorOkHttpClient   @Provides**   fun provideOtherInterceptorOkHttpClient(     otherInterceptor: OtherInterceptor   ): OkHttpClient {       return OkHttpClient.Builder()                .addInterceptor(otherInterceptor)                .build()   } } |

# Generisane komponente za Android klase

Za svaku Android klasu u kojoj izvrsavamo field injection, posto odredjena Hilt komponenta na koju se referenciramo unutar **@InstallIn** anotaciji. Svaka Hilt komponenta je zaduzena za injectovanje svojih veza u odgovarajucu Android klasu.

|  |  |
| --- | --- |
| SingletonComponent | Application |
| ActivityRetainedComponent |  |
| ViewModelComponent | ViewModel |
| ActivityComponent | Activity |
| FragmentComponent | Fragment |
| ViewComponent | View |
| ViewWithFragmentComponent | View annotated with @WithFragmentBindings |
| ServiceComponent | Service |

Hilt automatski kreira ili unistava instance generisane komponente prateci lifecycle odgovarajuce Android klase:

|  |  |  |
| --- | --- | --- |
| SingletonComponent | Application#onCreate() | Application#onDestroy() |
| ActivityRetainedComponent | Activity#onCreate() | Activity#onDestroy() |
| ViewModelComponent | ViewModel created | ViewModel destroyed |
| ActivityComponent | Activity#onCreate() | Activity#onDestroy() |
| FragmentComponent | Fragment#onAttach() | Fragment#onDestroy() |
| ViewComponent | View#super() | View destroyed |
| ViewWithFragmentComponent | View#super() | View destroyed |
| ServiceComponent | Service#onCreate() | Service#onDestroy() |

Po default, sve veze u Hiltu su unscoped. Ovo znaci da svaki put kada aplikacija trazi neku vezu, Hilt kreira novu instancu odgovarajuceg tipa.

U prethodnom primeru, svaki put kada se od Hilta zatrazi AnalyticsAdapter kao veza od strane neke Android klase, ta klasa dobija novu instancu tog adaptera.

Ipak, Hilt takodje omogucava da veza ima svoj opseg trajanja (scope) ka odrdjenoj komponenti. Hilt kreira scoped vezu jednom po instanci komponente koja tu vezu trazi, I svi zahtevi ka toj vezi ce biti deljeni izmedju komponenti.

|  |  |  |
| --- | --- | --- |
| Android class | Generated component | Scope |
| Application | SingletonComponent | @Singleton |
| Activity | ActivityRetainedComponent | @ActivityRetainedScoped |
| ViewModel | ViewModelComponent | @ViewModelScoped |
| Activity | ActivityComponent | @ActivityScoped |
| Fragment | FragmentComponent | @FragmentScoped |
| View | ViewComponent | @ViewScoped |
| View annotated with @WithFragmentBindings | ViewWithFragmentComponent | @ViewScoped |
| Service | ServiceComponent | @ServiceScoped |

U prethodnom primeru, ukoliko definisemo scope AnalyticsAdapteru ka ActivityComponent koristeci **@ActivityScoped**, Hilt ce pruziti istu instancu AnalyticsAdapter-a kroz kompletan zivotni ciklus odgovarajuceg activity-ja.

|  |
| --- |
| ***@ActivityScoped*** *class AnalyticsAdapter @Inject constructor(   private val service: AnalyticsService ) { ... }* |

Ukoliko imamo situaciju da odredju vezu treba da koristi vise komponenti (npr vise activity-ja), pametnije je toj vezi definisati scope na SingletonComponent I na taj nacin ce ta veza biti dostupna svim komponentima u trajanju rada aplikacije.

|  |
| --- |
| *// If AnalyticsService is an interface.* ***@Module @InstallIn(SingletonComponent::class)*** *abstract class AnalyticsModule {* ***@Singleton   @Binds*** *abstract fun bindAnalyticsService(     analyticsServiceImpl: AnalyticsServiceImpl   ): AnalyticsService }  // If you don't own AnalyticsService.* ***@Module @InstallIn(SingletonComponent::class)*** *object AnalyticsModule {* ***@Singleton   @Provides*** *fun provideAnalyticsService(): AnalyticsService {       return Retrofit.Builder()                .baseUrl("https://example.com")                .build()                .create(AnalyticsService::class.java)   } }* |

# Hijerarhija komponenti

Diagram

Description automatically generated

Instaliranje modula u odredjenu komponentu omogucava da njegove veze budu dostupne za pristup I u komponentama koje su predstavljene kao child komponente u datoj hijerarhiji komponenata. Po default, ukoliko izvrsimo field injection odredjenog view-a, ViewComponent moze koristiti sve veze definisane u ActivityComponent. Isto tako, ako zelimo da koristimo veze koje su definisane u FragmentComponent-u I view pripada tom fragment, potrebno je koristiti **@WithFragmentBindings** anotaciju u kombinaciji sa **@AndroidEntryPoint**

# Podrazumevana vezivanja komponenti (Component Default Bindings)

Svaka Hilt komponenta dolazi sa odredjenom skupom podrazumevanih veza koje Hilt moze ubaciti kao zavisnost u vase sopstvene prilagodjene veze. Treba imati na umu da ove veze odgovaraju generalno activity-jima I fragment-ima a ne bilo kojoj specificnoj klasi. To je zato sto Hilt koristi jednu definiciju activity komponente za ubacivanje svih aktivitija. Svaki aktiviti ima razlicitu instancu ove komponente.

|  |  |
| --- | --- |
| Android component | Podrazumevane veze |
| Singleton component | Application |
| ActivityRetainedComponent | Application |
| ViewModelComponent | SavedStateHandle |
| ActivityComponent | Application, Activity |
| FragmentComponent | Application, Activity, Fragment |
| ViewComponent | Application, Activity, View |
| ViewWithFragmentComponent | Application, Activity, Fragment, View |
| ServiceComponent | Application, Service |

Application context (ili Activity context) je takodje uvek dostupan uz pomoc @ApplicationContext (@ActivityContext) anotacije. Npr:

|  |
| --- |
| *class AnalyticsServiceImpl @Inject constructor(* ***@ApplicationContext*** *context: Context ) : AnalyticsService { ... }  // The Application binding is available without qualifiers. class AnalyticsServiceImpl @Inject constructor(* ***application: Application*** *) : AnalyticsService { ... }*  *class AnalyticsAdapter @Inject constructor(* ***@ActivityContext*** *context: Context ) { ... }  // The Activity binding is available without qualifiers. class AnalyticsAdapter @Inject constructor(* ***activity: FragmentActivity*** *) { ... }* |

# Ubacivanje veza u klase koje nisu podrzane od strane Hilt-a **(@EntryPoint)**

Hilt dolazi sa podrskom za vecinu Android klasa. Ipak, nekada je potrebno izvrsiti field injection u klasama koje Hilt ne podrzava. U tom slucaju, moze se kreirati pocetna tacka koristeci @EntryPoint anotaciju. Pocetna tacka je veza izmedju koda koji je upravljan od strane Hilta I koda koji nije. To je tacka gde kod prvi put ulazi u graph objekata koji je koriscen od strane Hilta.

Npr. Hilt ne podrzava direktno ContentProvider-e. Ukoliko zelimo da implementacija content provider-a koristi hilt za dobijanje veza ka odredjenim objektima, potrebno je definisati interfejs koji ce biti anotiran sa @EntryPoint za svaki tip veze koji zelimo. Tada je potrebno dodati @InstallIn za specifikaciju komponente u koju zelimo da instaliramo taj entry point.

|  |
| --- |
| *class ExampleContentProvider : ContentProvider() {* ***@EntryPoint   @InstallIn(SingletonComponent::class)*** *interface ExampleContentProviderEntryPoint {     fun analyticsService(): AnalyticsService   } }* |

Da bi se pristupilo entry point-u, potrebno je koristiti odgovarajucu staticku metodu iz EntryPointAccessors klase. Parametar treba da bude ili instanca komponente ili **@AndroidEntryPoint** objekat koji se ponasa kao component holder. Obratiti paznju da se komponenta koju prosledjujemo kao parameter I EntryPointAccessors staticka metoda podudaraju sa Android klasom u anotaciji **@InstallIn** anotacijom na **@EntryPoint** interfejsu:

|  |
| --- |
| *class ExampleContentProvider: ContentProvider() {     ...    override fun query(...): Cursor {     val appContext = context?.applicationContext ?: throw IllegalStateException()     val hiltEntryPoint =       EntryPointAccessors.fromApplication(appContext, ExampleContentProviderEntryPoint::class.java)      val analyticsService = hiltEntryPoint.analyticsService()     ...   } }* |

U gornjem primeru, potrebno je koristiti **@ApplicationContext** za dobijanje entry point-a zato sto je entry point instaliran u SingletonComponent. Ukoliko je binding koji zelimo da dobijemo smesten u ActivityComponent-u, potrebno je koristiti **@ActivityContext**.

# Hilt I Dagger

Hilt je napravljen na osnovu Dagger DI biblioteke, omogucavajuci standardan nacin za inkorporaciju Dagger-a u Android aplikacije.

Ciljevi Hilt-a u odnosu na Dagger su:

* Pojednostavljenje Dagger infrastructure za Android aplikacije.
* Kreiranje standardnog seta komponenti I njihovog scope-a za lako podesavanje, citljivost I kodiranje izmedju aplikacija.
* Omogucavanje lakog nacina za dobavljanje razlicitih veza za razlicite build tipove, poput testiranja, debugginga ili generisanje release.

Posto Android operativni system instancira mnoge sopstvene framework klase, koriscenje Dagger-a u Android aplikaciji zahteva od developera da pise znacajnu kolicinu sablona (boilerplate kod). Hilt smanjuje ovaj sablonski kod koji je ukljucen prilikom koriscenja Daggera u Android aplikaciji. Hilt automatski generise I obezbedjuje sledece:

* **Komponente za integraciju Android framework klasa** koje Dagger automatski generise I koje bismo morali da pisemo sami.
* **Scope annotations** za koriscenje komponenti koje Hilt generise automatski
* **Predefinisane veze** koje reprezentuju Android klase poput **Application** ili **Activity**
* **Predefinisani kvalifikatori** koji reprezentuju @ApplicationContext ili @ActivityContext