



Android Developer

Hilt - управление зависимостями



Проверить, идет ли запись

```
if (видно && слышно) {  
    chat.print("+")  
}
```



Ставим "+", если все хорошо
"-", если есть проблемы



Тема вебинара

StateMachine

Hilt - управление зависимостями



Николай Кочетков

Руководитель Android разработки FlyXO

Об опыте (например):

22 года в IT, 7 лет - играющий тренер Android

Телефон / эл. почта / соц. сети:

LinkedIn: <https://www.linkedin.com/in/motorro/>

GitHub: <https://github.com/motorro/>

Medium: <https://medium.com/@motorro>

Правила вебинара



Активно
участвуем



Off-topic обсуждаем
в Slack **#канал группы**
или **#general**



Задаем вопрос
в чат или голосом



Вопросы вижу в чате,
могу ответить не сразу

Условные обозначения



Индивидуально



Время, необходимое
на активность



Пишем в чат



Говорим голосом



Документ



Ответьте себе или
задайте вопрос

Маршрут вебинара

Архитектурное ИМНО

Что где лежит: модель, данные, сеть

Узнаем, зачем нам DI

Scopes. Зачем они нужны (и нет)

Hilt - простой Dagger

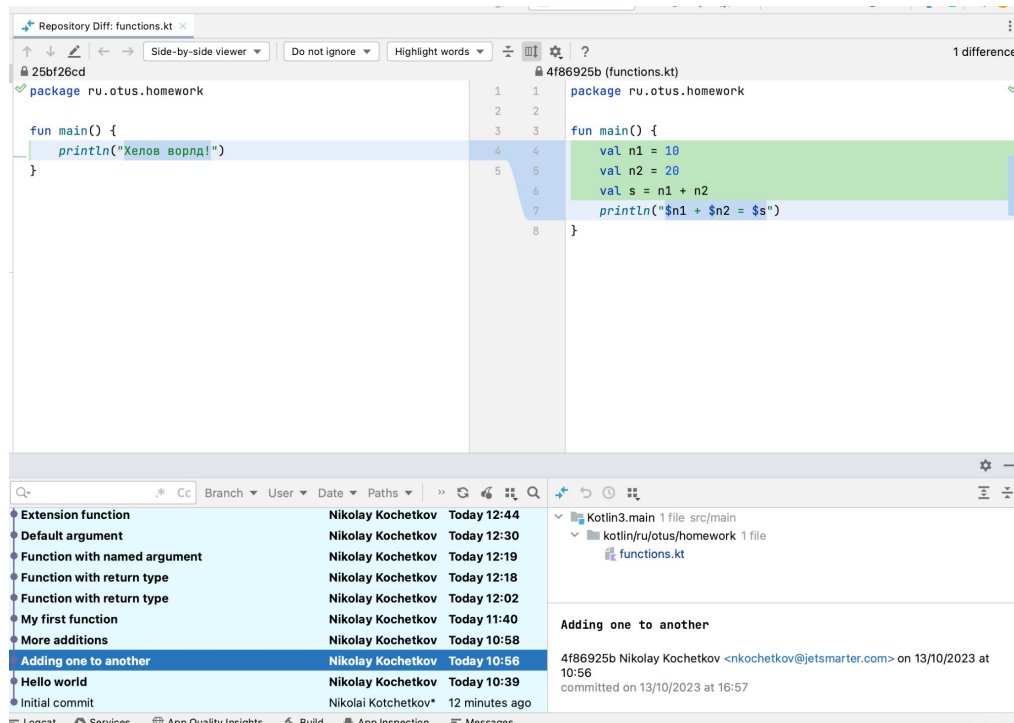


Репозиторий к занятию

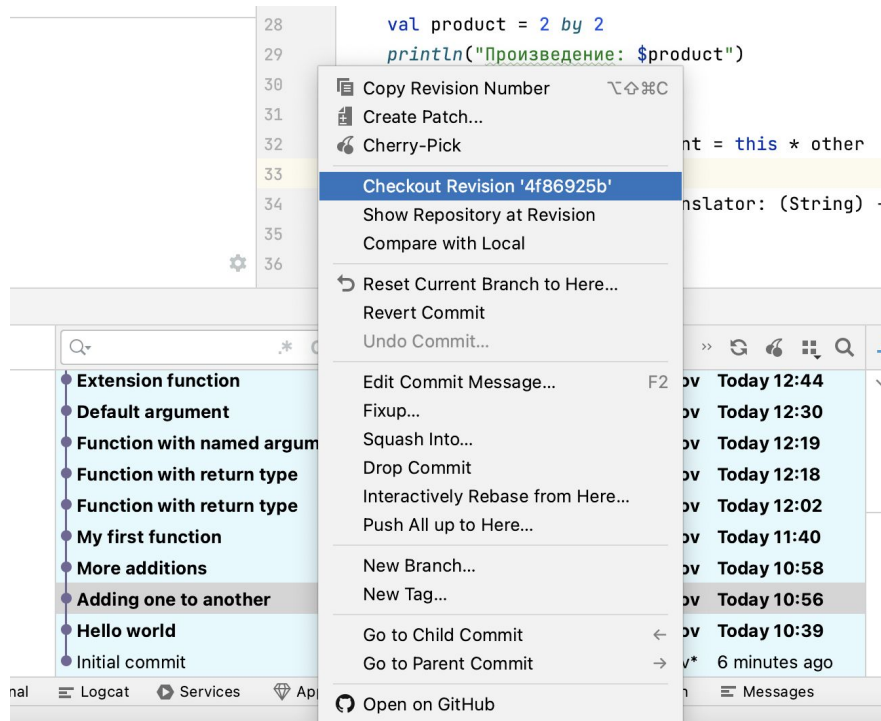


<https://github.com/Android-Developer-Basic/Hilt>

Репозиторий к занятию - по шагам

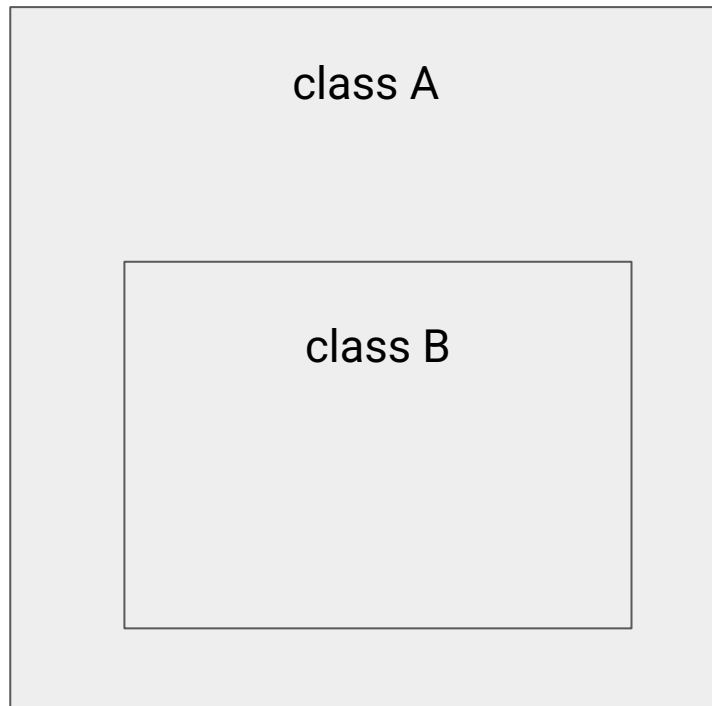


Репозиторий к занятию - по шагам



Зависимости

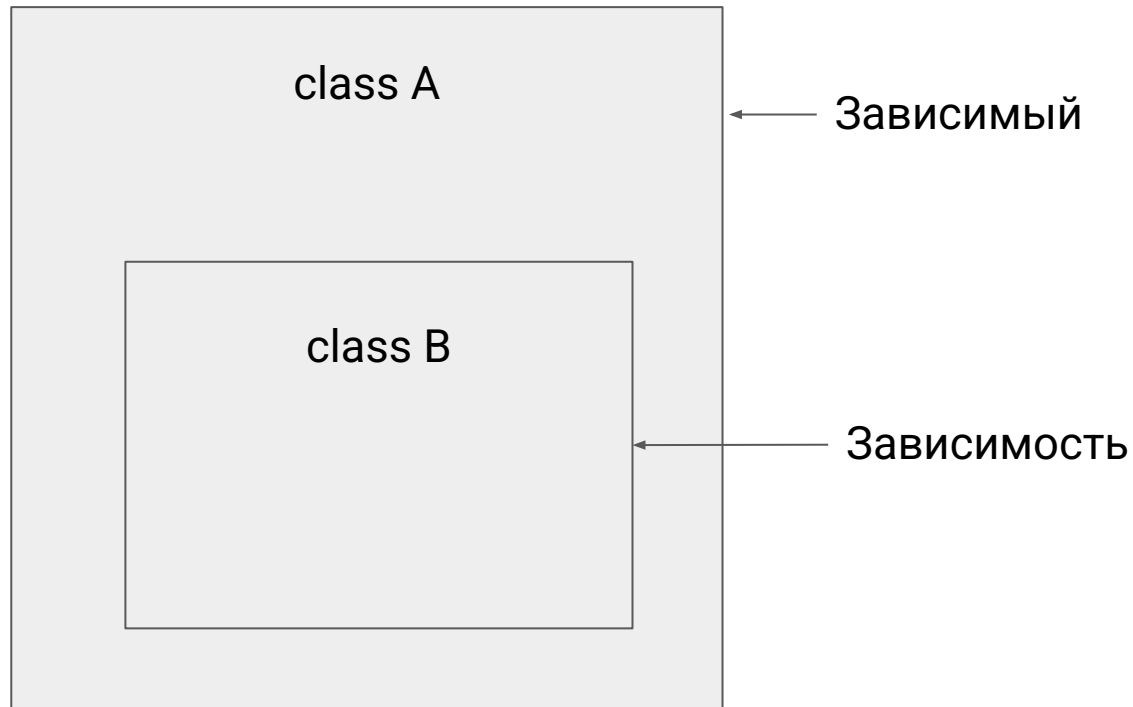
Зависимость



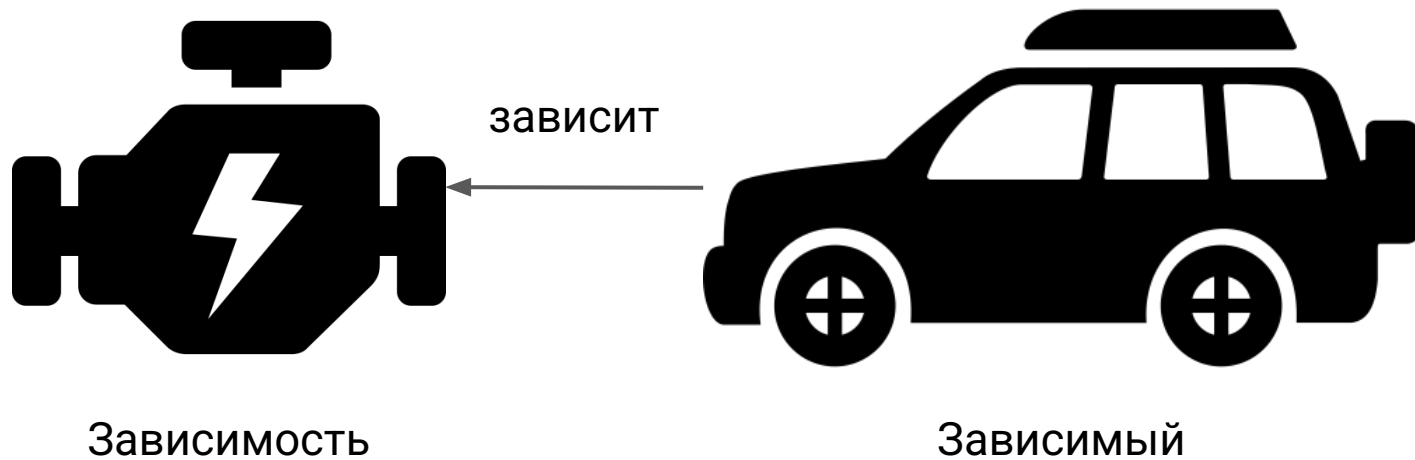
Зависимость

- Два класса, которые используют друг друга, называют “связанными”
- Когда класс А использует В, тогда А зависит от В
- А не может выполнить работу без В
- А не может быть переиспользован без В
- А - “зависимый”
- В - “зависимость”

Зависимость



Зависимость



Вопросы?

Engine

```
3  class Engine(private val power: Int) {  
4      fun start() {  
5          println("Engine started with power $power")  
6      }  
7      fun stop() {  
8          println("Engine stopped")  
9      }  
10 }
```



Engine

```
9      class Car {  
10         private val engine = Engine( power: 100)  
11  
12         fun start() {  
13             engine.start()  
14         }  
15         fun stop() {  
16             engine.stop()  
17         }  
18     }
```


Engine

```
9      class Car {  
10         private val engine = Engine( power: 100)  
11  
12         fun start() {  
13             engine.start()  
14         }  
15         fun stop() {  
16             engine.stop()  
17         }  
18     }
```

↑
Только такой мотор

Проблемы

1. Как нам построить машину с другим объемом двигателя?
2. Что если мы хотим сделать электромобиль?
3. Как нам организовать Unit-тестирование машины?
4. Кто определяет время жизни объектов?

Вопросы?

Рефакторинг - минимальный интерфейс

```
3  ⓘ↓ interface Engine {  
4  ⓘ↓     fun start()  
5  ⓘ↓     fun stop()  
6      }
```

Реализация

```
8      class V4Engine(private val power: Int) : Engine {  
9          override fun start() {  
10              println("Engine started with power $power")  
11          }  
12          override fun stop() {  
13              println("Engine stopped")  
14          }  
15      }
```

Машина лучше не стала

```
9      class Car {  
10         private val engine = V4Engine(power: 100)  
11  
12         fun start() {  
13             engine.start()  
14         }  
15         fun stop() {  
16             engine.stop()  
17         }  
18     }
```

↑
Только такой мотор

Предоставление зависимости

Любой двигатель



```
9      class Car(private val engine: Engine) {  
10          fun start() {  
11              engine.start()  
12          }  
13          fun stop() {  
14              engine.stop()  
15          }  
16      }
```



Предоставление зависимости

```
3  ▶ fun main() {  
4      val car1 = Car(V4Engine( power: 200))  
5      car1.start()  
6      car1.stop()  
7  
8      val car2 = Car(V8Engine( power: 400))  
9      car2.start()  
10     car2.stop()  
11 }
```





Тестирование

```
private val fakeEngine = object : Engine {  
    var started = false  
  
    override fun start() {  
        started = true  
    }  
    override fun stop() {  
        started = false  
    }  
}
```

Тестирование

```
@Test
fun `should start engine`() {
    val car = Car(fakeEngine)
    car.start()
    assertTrue { fakeEngine.started }
}
```

Проблемы

-  1. Как нам построить машину с другим объемом двигателя?
-  2. Что если мы хотим сделать электромобиль?
-  3. Как нам организовать Unit-тестирование машины?
- 4. Кто определяет время жизни объектов?

Вопросы?

Фабрика машин

```
interface CarFactory {  
    fun buildCar(): Car  
}
```

Фабрика машин

```
class CarFactoryV4 : CarFactory {  
    override fun buildCar(): Car {  
        return Car(V4Engine( power: 200))  
    }  
}
```



Фабрика машин

```
fun main() {  
    val car = carFactory.buildCar()  
    car.start()  
    car.stop()  
}
```

Фабрика машин





```
fun buildWithFactory(factory: CarFactory) {  
    val car = factory.buildCar()  
    car.start()  
    car.stop()  
}
```


Время жизни

```
fun main() {  
    // 2025 car factory - V4 engine  
    val factory2025 = CarFactoryV4()  
    buildWithFactory(factory2025)  
    // 2026 car factory - V8 engine  
    val factory2026 = CarFactoryV8()  
    buildWithFactory(factory2026)  
}
```



Проблемы

-  1. Как нам построить машину с другим объемом двигателя?
-  2. Что если мы хотим сделать электромобиль?
-  3. Как нам организовать Unit-тестирование машины?
-  4. Кто определяет время жизни объектов?

Вопросы?

Архитектура ПРОЕКТА

Зачем оно вообще нужно (IMHO)

Для чего она нужна?

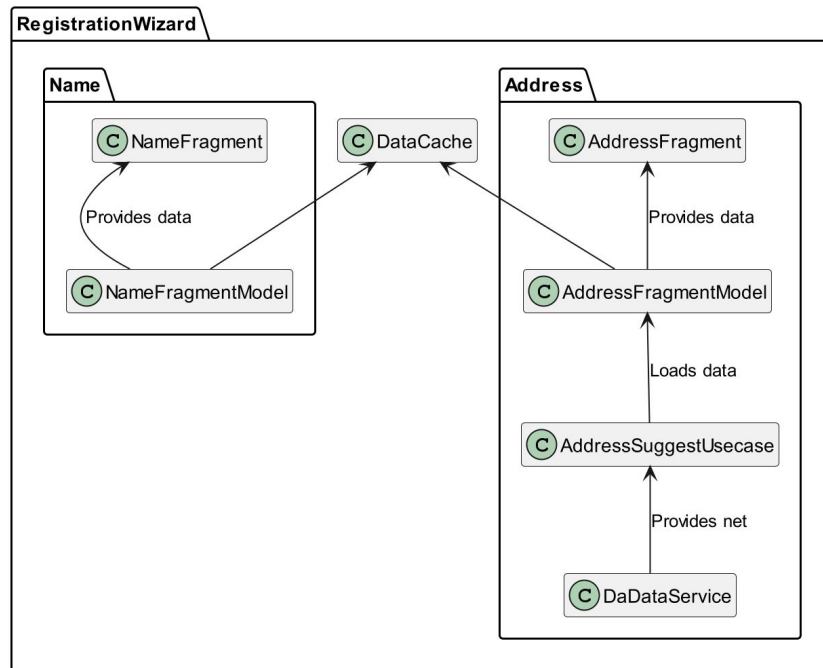
На самом деле (IMHO)

1. КОМАНДНАЯ РАБОТА
2. ПОНЯТНЫЙ ПУТЬ ИЗМЕНЕНИЯ КОДА
3. УВЕРЕННОСТЬ, ЧТО НЕ СЛОМАЕШЬ

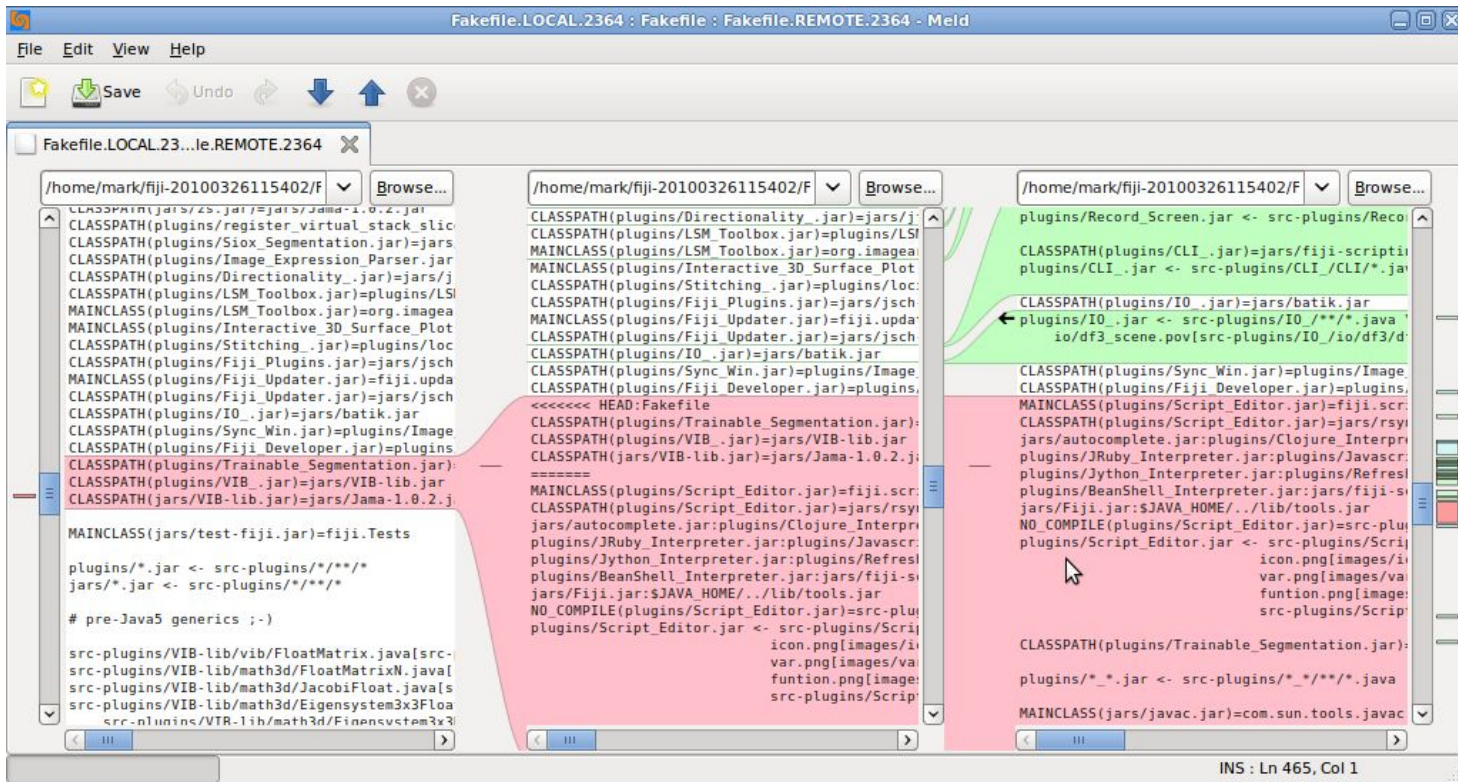
Что включает

1. Структура приложения (архитектура кода)
2. Соглашения о расположении файлов
3. Тестируемость

Структура приложения



Соглашения о расположении файлов



Тестируемость = не бояться сломать



Вопросы?

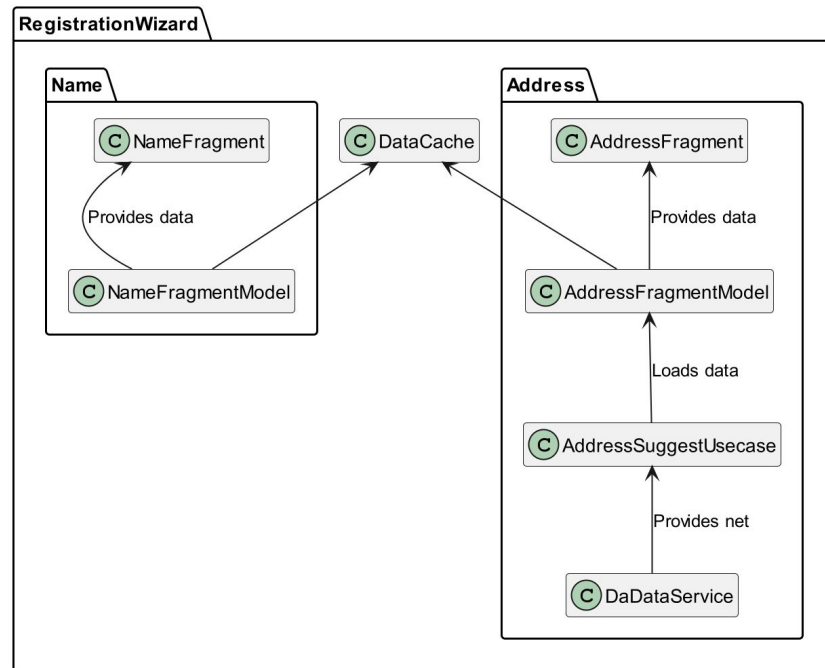
Архитектура приложения

Что где лежит?

Разделение зон ответственности

- View - показывает данные
- ViewModel - содержит локальное состояние и запускает UseCase
- UseCase - выполняет операции
- Service - абстрагирует реализацию (сеть, диск)

Разделение зон ответственности



Разделение зон ответственности

- Разные компоненты пишут разные члены команды
- Лучше понимаешь, где искать баг
- Простота тестирования

Мнение Google

Guide to app architecture

<https://developer.android.com/topic/architecture>

Вопросы?

Как нам поможет DI?

Что нам нужно, чтобы запустить проект?

Собрать все вместе

Что мы должны уметь?

- Создать объекты
- Знать, кому что нужно (зависимости)
- Предоставить зависимости клиентам
- Структурировать время жизни объектов
- Проверить целостность набора зависимостей

Dagger2 DI

- Компонент - хранит зависимости
- Модуль - создает зависимости

Компонентный DI

- Гранулированное время жизни
- Динамические зависимости
- Модуляризация

Что такое компонент?

**Компонент определяет время жизни
scored-объектов в нем!**

Вопросы?

Dagger Hilt

Рецепты на каждый день...

Тестовый проект

<https://github.com/Android-Developer-Basic/Hilt>

Dagger components - application

Живет
пока
жив
Application

```
19
20     @Singleton
21     @Component(modules = [AppModule::class])
22     interface AppComponent {
23         fun num(): Int
24
25         @Component.Factory
26         interface Factory {
27             fun create(@BindsInstance app: Application) : AppComponent
28         }
29     }
30
31     @Module
32     class AppModule {
33         @Provides
34         @Singleton
35         fun num(): Int = 5
36     }
```

Dagger components - application

```
10      class DaggerApp : Application() {  
11  
12          lateinit var appComponent: AppComponent  
13  
14          override fun onCreate() {  
15              super.onCreate()  
16              appComponent = DaggerAppComponent.factory().create(this)  
17          }  
18      }
```

Dagger components - activity

Живет
пока
жива
активности

```
27     @Scope
28     @Retention(AnnotationRetention.RUNTIME)
29     annotation class
30     ActivityScope
31
32     @ActivityScope
33     @Component(dependencies = [AppComponent::class])
34     interface MainActivityComponent {
35         fun activity(): Activity
36
37         fun inject(activity: MainActivity)
38
39         @Component.Factory
40         interface Factory {
41             fun create(
42                 appComponent: AppComponent,
43                 @BindsInstance activity: Activity
44             ) : MainActivityComponent
45         }
46     }
```

Dagger components - activity

```
11 class MainActivity : ComponentActivity() {  
12  
13     @set:Inject  
14     var num: Int = 0  
15  
16     override fun onCreate(savedInstanceState: Bundle?) {  
17         super.onCreate(savedInstanceState)  
18  
19         DaggerMainActivityComponent  
20             .factory()  
21             .create((application as DaggerApp).appComponent, activity: this)  
22             .inject(activity: this)  
23     }  
24 }  
25 }
```

Hilt - application

```
11  @HiltAndroidApp
12  class HiltApp : Application()
13
14  @Module
15  @InstallIn(SingletonComponent::class)
16  class AppModule {
17      @Provides
18      @Singleton
19      fun num(): Int = 5
20  }
```

Hilt - activity

```
36     @Module
37     @InstallIn(ActivityComponent::class)
38     class ActivityModule {
39         @Provides
40         @ActivityScoped
41         fun str(): String = "String"
42     }
```


Hilt - activity

```
14  @AndroidEntryPoint
15  class MainActivity : ComponentActivity() {
16
17      @set:Inject
18      var num: Int = 0
19
20      @set:Inject
21      var str: String = ""
22
23      override fun onCreate(savedInstanceState: Bundle?) {
24          super.onCreate(savedInstanceState)
25
26          val view = ActivityMainBinding.inflate(layoutInflater)
27          setContentView(view.root)
28
29          with(view) { this: ActivityMainBinding
30              numView.text = num.toString()
31              strView.text = str
32          }
33      }
34  }
```

Hilt - activity module

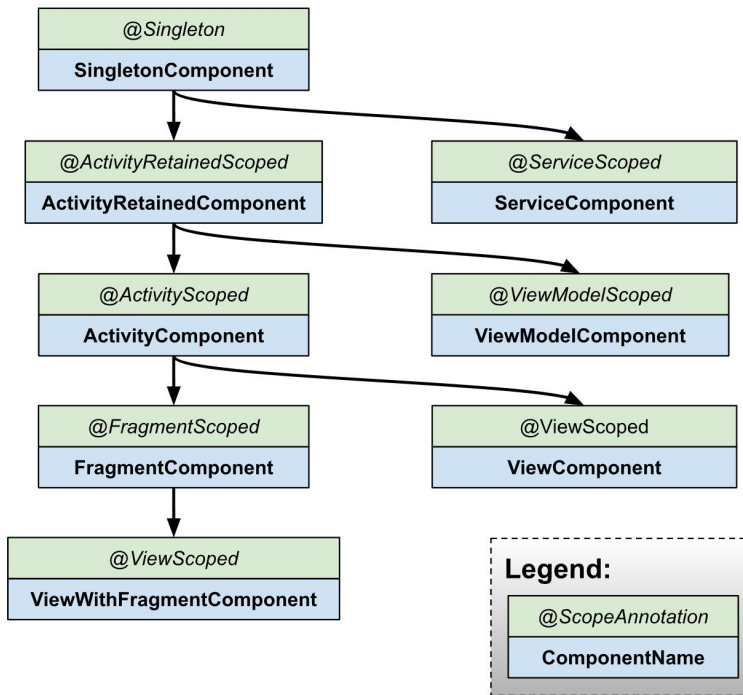
```
36     @Module
37     @InstallIn(ActivityComponent::class)
38     class ActivityModule {
39         @Provides
40         @ActivityScoped
41         fun str(): String = "String"
42     }
43
```



Hilt использует subcomponents

- Подкомпонент наследует и расширяет родительский компонент
- Любая зависимость подкомпонента может зависеть от объекта в родительском графе
- Родительский граф не может зависеть от объектов дочерних компонентов

Hilt components



Hilt components

Component	Scope	Created at	Destroyed at
SingletonComponent	@Singleton	Application#onCreate()	Application process is destroyed
ActivityRetainedComponent	@ActivityRetainedScoped	Activity#onCreate() ¹	Activity#onDestroy() ¹
ViewModelComponent	@ViewModelScoped	ViewModel created	ViewModel destroyed
ActivityComponent	@ActivityScoped	Activity#onCreate()	Activity#onDestroy()
FragmentComponent	@FragmentScoped	Fragment#onAttach()	Fragment#onDestroy()
ViewComponent	@ViewScoped	View#super()	View destroyed
ViewWithFragmentComponent	@ViewScoped	View#super()	View destroyed
ServiceComponent	@ServiceScoped	Service#onCreate()	Service#onDestroy()

From Hilt website

Важно! Каждому своё!

- У каждой активити - свой экземпляр компонента
- У каждого фрагмента - свой экземпляр компонента
- У каждой модели - свой экземпляр компонента

Вопросы?

Пример работы с компонентами

Score - что это?

Score определяет в каком компоненте экземпляр будет синглтоном

Scope - что это?

```
// This binding is "unscoped".  
// Each request for this binding will get a new instance.  
class UnscopedBinding @Inject constructor() {  
}  
  
// This binding is "scoped".  
// Each request from the same component instance for this binding will  
// get the same instance. Since this is the fragment component, this means  
// each request from the same fragment.  
@FragmentScoped  
class ScopedBinding @Inject constructor() {  
}
```

From Hilt website



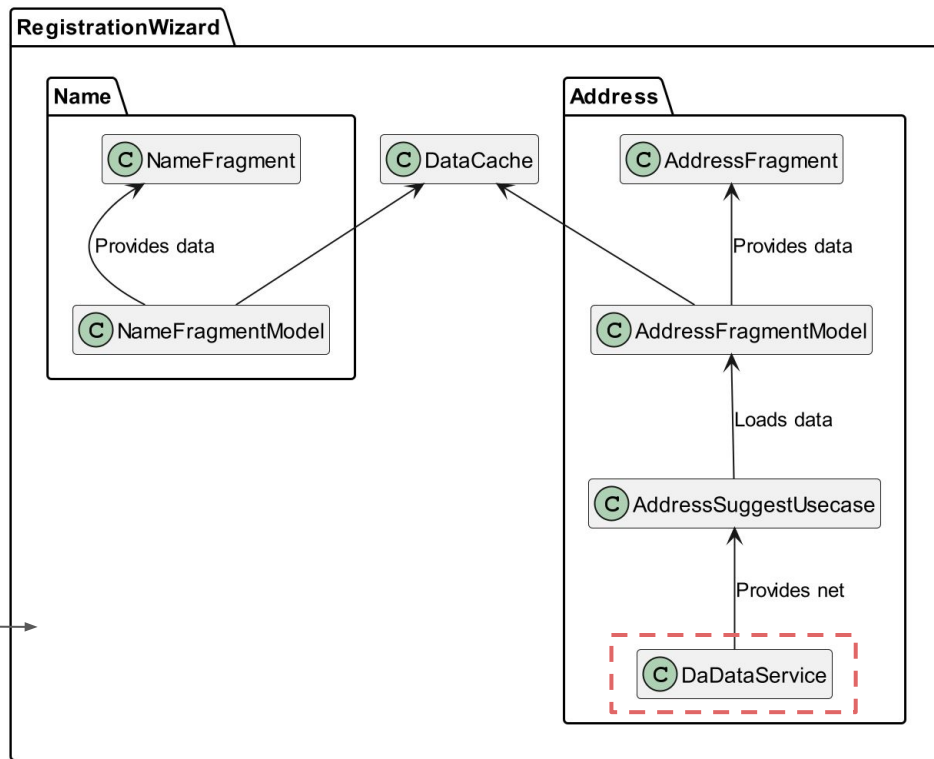
Пример работы scoped/unscoped

Вопросы?

Кто куда?

DaDataService у нас много где
нужен и тяжелый

Всё приложение:
@Singleton

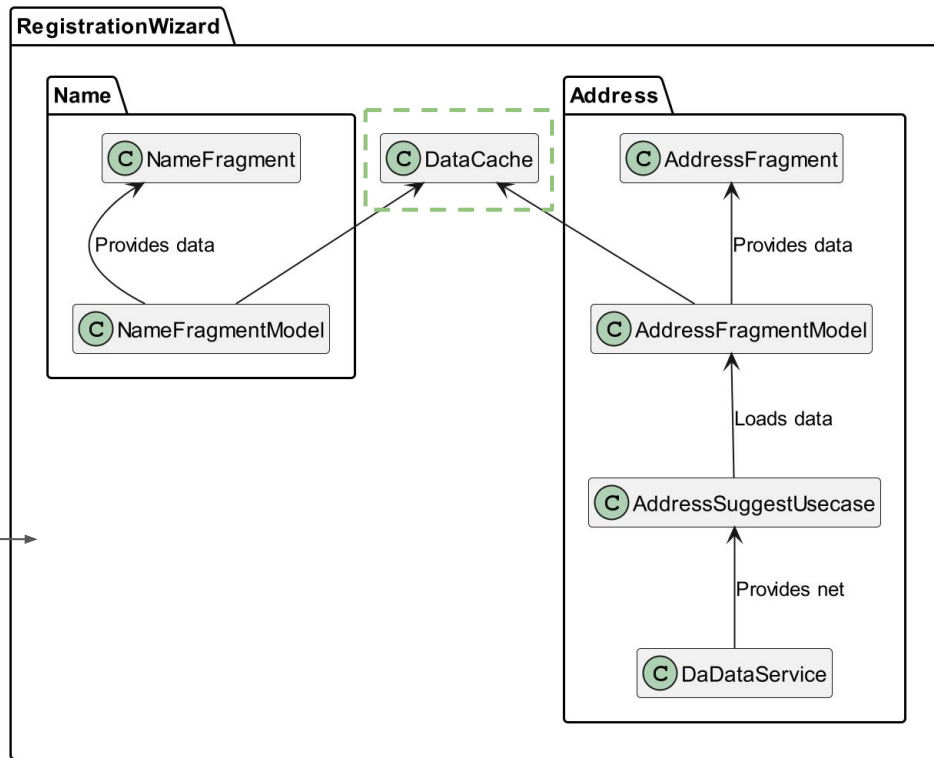


Кто куда?

DataCache нам нужен только для регистрации, между экранами нужен синглтон!

Activity:

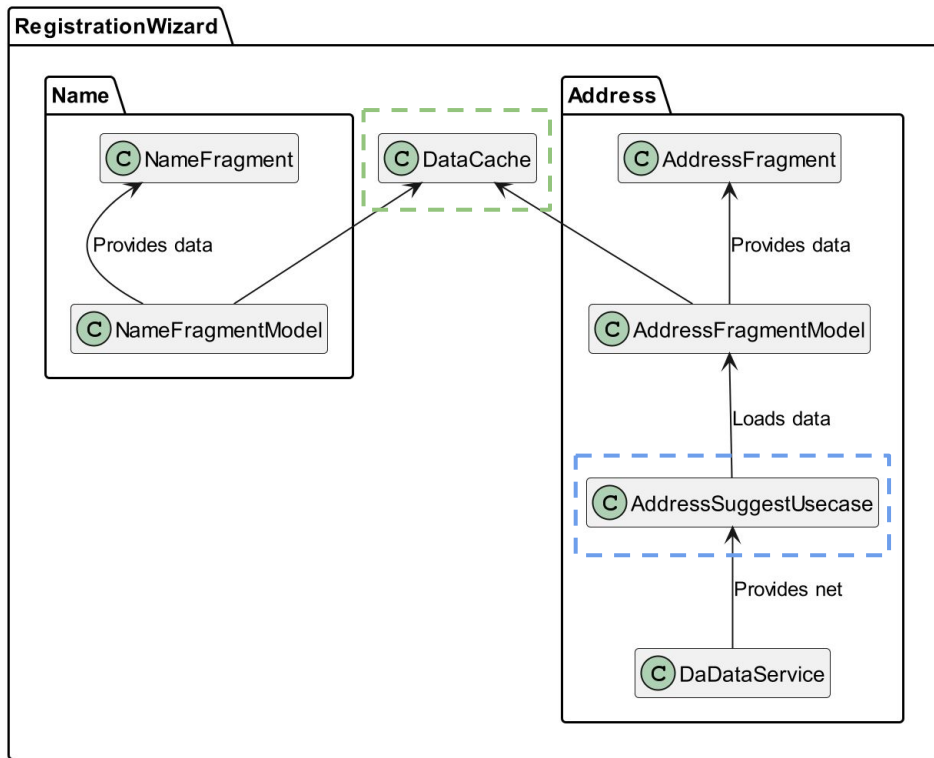
`@ActivityRetainedScoped`



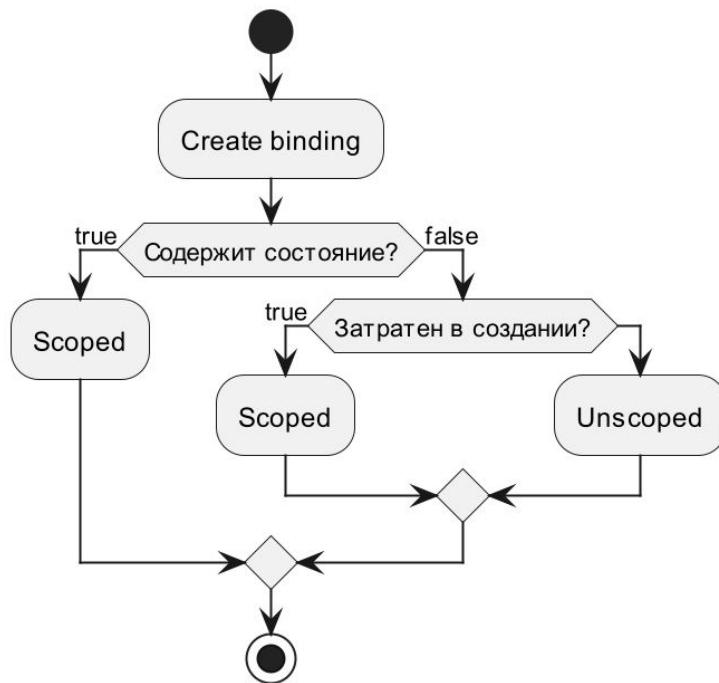
Кто куда?

AddressSuggestUsecase нам нужен только для **AddressFrament**. И нам все равно, сколько их будет - они не содержат состояния

UNSCOPED



Кто куда?



Вопросы?

Dagger Hilt

Поддержка JetPack из коробки



@AndroidEntryPoint для компонентов Android

- Activity
- Fragment
- View
- Service
- BroadcastReceiver

@HiltViewModel для ViewModel

```
8
9  @HiltViewModel
10 class MainActivityViewModel @Inject constructor(usecase: MainActivityUsecase) : ViewModel() {
11     val data = MutableLiveData<String>()
12
13     init {
14         data.value = usecase.getData()
15     }
16 }
17
18 @ViewModelScoped
19 class MainActivityUsecase @Inject constructor() {
20     fun getData(): String = "Data from view-model"
21 }
```

@HiltViewModel для ViewModel

```
private val viewModel: MainActivityViewModel by viewModels()
```

Передача параметров ViewModel

```
29 companion object {  
30  
31     const val PARAM = "param"  
32  
33     fun getStartIntent(context: Context, param: String): Intent {  
34         val intent = Intent(  
35             context,  
36             ParamsActivity::class.java  
37         )  
38         intent.putExtra(PARAM, param)  
39  
40         return intent  
41     }  
42 }  
43
```

Передача параметров ViewModel

```
10      @HiltViewModel
11      class ParamsActivityViewModel @Inject constructor(
12          savedStateHandle: SavedStateHandle
13      ) : ViewModel() {
14          val data = MutableLiveData<String>(savedStateHandle[PARAM])
15      }
```

SaveStateHandle: [документация](#)

Вопросы?

Рефлексия

Тестовый проект

<https://github.com/Android-Developer-Basic/Hilt>

Маршрут вебинара

Архитектурное ИМНО

Что где лежит: модель, данные, сеть

Узнаем, зачем нам DI

Scopes. Зачем они нужны (и нет)

Hilt - простой Dagger

Вопросы для проверки

По пройденному материалу всего вебинара

1. Какие задачи решает архитектура проекта?
2. Чем нам помогает DI?
3. Что такое компонент? Какое у него время жизни?
4. Для чего нужен Scope? В каких случаях он нужен, в каких не нужен?
5. Как получить зависимости не в Android компоненте?
6. Как передать параметр из интента во ViewModel?



Рефлексия



С какими впечатлениями уходите с вебинара?



Как будете применять на практике то, что узнали на вебинаре?

**Заполните, пожалуйста,
опрос о занятии
по ссылке в чате**

Спасибо за внимание!

Приходите на следующие вебинары



Николай Кочетков

Руководитель Андроид разработки FlyXO

Об опыте (например):

22 года в IT, 7 лет - играющий тренер Андроид

Телефон / эл. почта / соц. сети:

LinkedIn: <https://www.linkedin.com/in/motorro/>

GitHub: <https://github.com/motorro/>

Medium: <https://medium.com/@motorro>