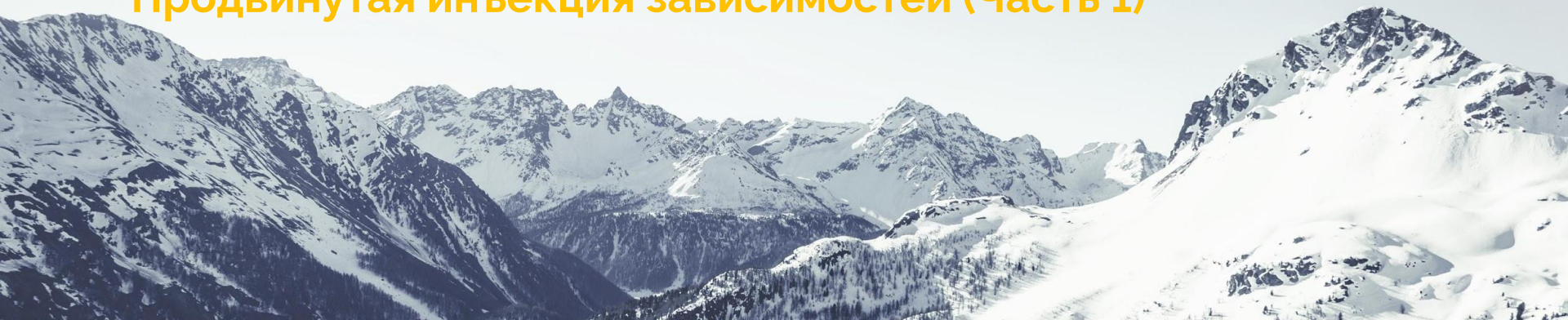
<TeachMeSkills/>

курс

Android разработчик

Продвинутая инъекция зависимостей (Часть 1)



Агенда занятия:

Dagger2

@Provides

@Inject

@Component

@Module

- **Dagger2**
  - @Provides
  - @Inject
  - @Component
  - @Module

Dagger2

Dagger2 — это один из самых популярных фреймворков для DI в Android. Он генерирует код на этапе компиляции, что делает DI быстрым и безопасным. Позволяет уменьшить “ручную” работу с зависимостями. Статически генерируемый фреймворк для внедрения зависимостей от Google. Использует аннотации и compile-time генерацию кода.

Аннотации в Dagger2

Это специальные метки в коде, которые Dagger использует, чтобы понять:

- какие зависимости тебе нужны;
- как их создать;
- где и как их "вколоть" (внедрить).

Как работает Dagger2 внутри

Этап 1. Ты используешь аннотации.

Dagger2 анализирует эти аннотации на этапе компиляции.

Этап 2. Генерация кода (compile-time).

Когда ты собираешь проект, Dagger:

- Создаёт классы с префиксом Dagger, например: DaggerAppComponent;
- Генерирует реализацию всех зависимостей, описанных в @Module, @Provides, @Inject;
- Собирает граф зависимостей (кто от кого зависит).

Как работает Dagger2 внутри

Этап 3. Время выполнения (runtime).

В приложении ты вызываешь DaggerAppComponent.create().inject(this)

И всё, зависимости "вкалываются" в поля, конструкторы или параметры методов.

Ключевые отличия Dagger2 от Hilt

	Dagger2	**Hilt**
Контроль	Максимальный — ты контролируешь всё	Много скрыто под капотом
Boilerplate-код	Больше	Меньше
Архитектура	Можно использовать в любом Java/Kotlin-проекте	Только Android
Лучший выбор для	Крупных, кастомных решений	Быстрый старт и Android-приложения

Аннотация @Inject

Применяется к **конструкторам,** полям, методам.

@Inject говорит Dagger'у: "можно создавать этот объект и внедрять зависимости в него".

```
class Engine @Inject constructor()

class Car @Inject constructor(private val engine: Engine)
```

Аннотация @Inject

Применяется к конструкторам, **полям**, методам.

Но это работает только если потом вызвать component.inject(this) — Dagger не может сам "пройтись" по классу, его надо попросить.

```
class Car {
    @Inject
    lateinit var engine: Engine
}
```

Аннотация @Provides

Обозначает метод, который создает объект (предоставляет зависимость)
Это как сказать Dagger: «Вот, если тебе нужен Engine, зови этот метод — я скажу, как его собрать».

```
@Provides
fun provideEngine(): Engine {
    return Engine()
}
```

Аннотация @Module

Обозначает класс, где ты сам говоришь, как создавать зависимости.

Почему нужен:

Когда не можешь поставить @Inject на конструктор (например, класс сторонний).

```
@Module
class AppModule {

    @Provides
    fun provideLogger(): Logger {
        return Logger("debug")
    }
}
```

Аннотация @Component

Связывает все: модули, @Inject, @Provides, и объекты, куда нужно внедрить.

Это "точка входа" для всей инъекции.

Dagger сам сгенерирует реализацию AppComponent, например DaggerAppComponent.

```
@Component(modules = [AppModule::class])
interface AppComponent {
    fun inject(activity: MainActivity)
}
```

Как это всё работает вместе

Смотрим проект

Практика

Задача 1. @Inject

Создай класс Battery, внедри его в Phone с помощью @Inject.

Задача 2. @Provides + @Module

Используй @Provides, чтобы внедрить Logger в Repository.

Задача 3. Связь с Activity

Внедри зависимость в MainActivity через компонент.

Задача 4. @Singleton

Добавь скоуп и проверь, что Logger создается один раз.

Задача 5. Комбинированное задание

Построй граф: Car → Engine → FuelPump, используя @Module и @Inject.

Домашнее задание

Задача 1.

Переписать DI, реализованный в теме занятия 30 “Инъекция зависимостей”, заменив Hilt на Dagger2.

Q&A

Ваши вопросы

Спасибо

<TeachMeSkills/>