WaffleStudio Android Seminar - 4

이승민 (안드로이드 세미나장)

2021.10.09.(토) 11:30 ~

What we will learn in Seminar 4

- Dependency Injection
  - Dagger Hilt
- Fragment

Dependency Injection

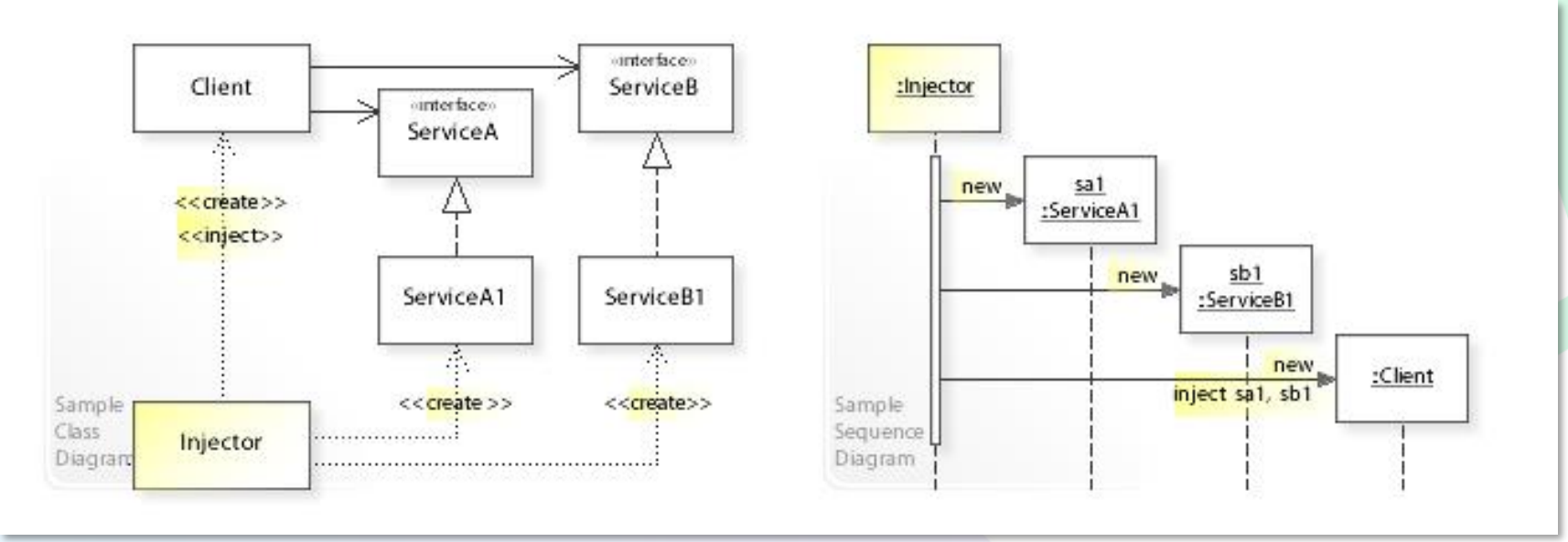
- 기존
  - 어떤 Class를 사용하려면…
  - Object를 생성하여 사용
    - -> Object를 생성한 클래스 내에 종속
- Dependency Injection
  - Object를 클래스 내에 종속시키지 않고 외부에서 Injection 받는다.

Dependency Injection

- Why?
  - 클라이언트 코드 구성을 유연하게 해줌
  - 외부에서 주입해주므로 테스트 작성이 편리

Dependency Injection

- 클라이언트가 사용할 서비스 객체
- 클라이언트 객체
- 서비스를 클라이언트에 주입시킬 주입자



Dependency Injection

- Dagger (Hilt)
  - Compile Time 에 작업 처리
- Koin
  - Run Time 에 작업 처리

Dagger Hilt

- Dagger
  - Dependency Injection Library
  - Compile Time에 작업을 처리하여 Performance에 영향 X
- Dagger Hilt
  - 범용적인 Dagger을 Android에 맞게 만든 Library
  - Android 에서 사용하기 편함

Dagger Hilt

- 기존…
  - App에 선언하여 끌어다가 썼음
  - Scope 처리가 불완전

```
private val moshi by lazy {
    Moshi.Builder()
        .add(KotlinJsonAdapterFactory())
        .build()
}

private val httpClient by lazy {
    OkHttpClient.Builder()
        .addInterceptor(
            HttpLoggingInterceptor().apply { this: HttpLoggingInterceptor
                level =
                    if (BuildConfig.DEBUG) HttpLoggingInterceptor.Level.BODY
                    else HttpLoggingInterceptor.Level.NONE
            }
        )
        .build()
}

private val retrofit by lazy {
    Retrofit.Builder()
        .client(httpClient)
        .baseUrl( baseUrl: "https://jsonplaceholder.typicode.com/")
        .addConverterFactory(MoshiConverterFactory.create(moshi))
        .build()
}

private val postService by lazy {
    retrofit.create(PostService::class.java)
}

val postRepository by lazy {
    PostRepository(postService)
}
```

Dagger Hilt

- 사용법
  - Application 클래스에 @HiltAndroidApp
  - Activity, Fragment에 @AndroidEntryPoint
  - ViewModel에 @HiltViewModel

Dagger Hilt

- Module
  - 의존성 주입을 관리하는 객체
  - 앱 Scope에 Install (SingletonComponent)
  - Singleton 패턴을 쉽게 구현 (@Singleton)

```kotlin
@Module
@InstallIn(SingletonComponent::class)
object NetworkModule {

    @Provides
    @Singleton
    fun provideMoshi(): Moshi {
        return Moshi.Builder()
            .add(KotlinJsonAdapterFactory())
            .build()
    }

    @Provides
    @Singleton
    fun provideHttpClient(): OkHttpClient {
        return OkHttpClient.Builder()
            .addInterceptor(
                HttpLoggingInterceptor().apply { this: HttpLoggingInterceptor
                    level =
                        if (BuildConfig.DEBUG) HttpLoggingInterceptor.Level.BODY
                        else HttpLoggingInterceptor.Level.NONE
                }
            ).build()
    }

    @Provides
    @Singleton
    fun provideRetrofit(httpClient: OkHttpClient, moshi: Moshi): Retrofit {
        return Retrofit.Builder()
            .client(httpClient)
            .baseUrl( baseUrl: "https://jsonplaceholder.typicode.com/")
            .addConverterFactory(MoshiConverterFactory.create(moshi))
            .build()
    }

    @Provides
    @Singleton
    fun providePostService(retrofit: Retrofit): PostService {
        return retrofit.create(PostService::class.java)
    }

}
```

Dagger Hilt

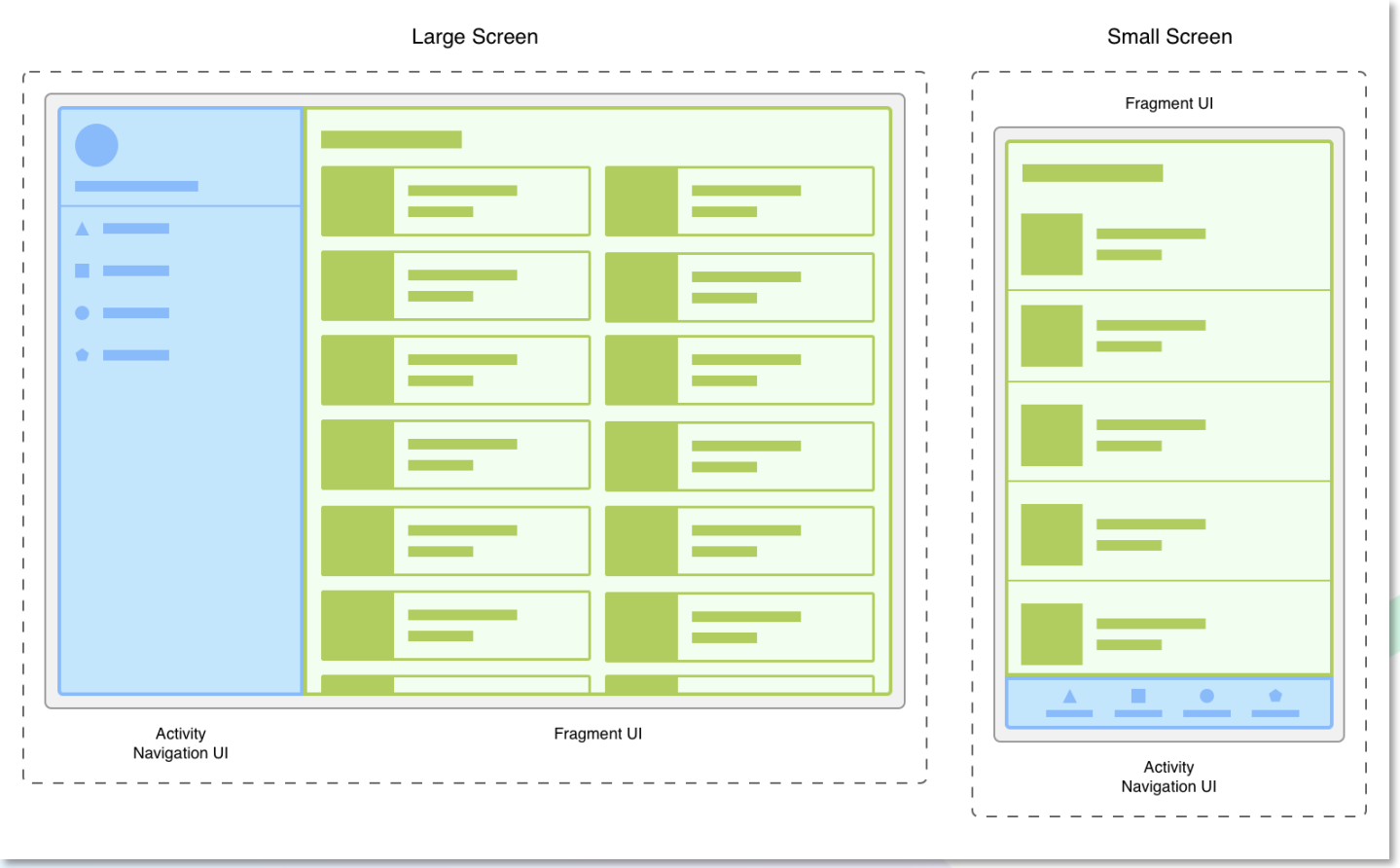
- Repository도 Singleton하게
- 더 이상 ActivityViewModel을 써서 application context를 불러올 필요가 없음

```
@Singleton
class PostRepository @Inject constructor(private val postService: PostService) {
    suspend fun getAllPost(): List<Post> {
        return postService.getAllPost()
    }
}
```

```
@HiltViewModel
class MainViewModel @Inject constructor(
    private val postRepository: PostRepository
) : ViewModel() {
```

Fragment

- Activity의 한계를 넘어서게 해주는 UI 요소 – 우리가 사용하는 대부분의 앱은 Fragment로 이루어져 있음
- 재사용 가능



Fragment

- UI를 다양하게 활용 가능
- 단순히 화면을 전환하는 기능부터
- ViewPager을 활용한 스와이프
- navGraph를 활용한 직관적인 UI graph 구성
- 화면 비율에 따른 동적인 화면 구성

Fragment

- 주의점
  - Activity에서는 onCreate에서 UI 구현
  - Fragment에서는
    - onCreateView에서 binding 구성
    - onViewCreated에서 UI 구현

QA