PROVISIÓN DE DEPENDENCIAS

```
viewModel = getViewModel {
    val moviesRepository = MoviesRepository(
        RoomDataSource(app.db),
        TheMovieDbDataSource(),
        RegionRepository(
            PlayServicesLocationDataSource(app),
            AndroidPermissionChecker(app)
        ),
        app.getString(R.string.api_key)
    )

    DetailViewModel(
        intent.getIntExtra(MOVIE, -1),
        FindMovieById(moviesRepository),
        ToggleMovieFavorite(moviesRepository)
    )
}
```

PROVISIÓN DE DEPENDENCIAS

```
viewModel = getViewModel { viewModelProvider.get() }
```

¿QUÉ OPCIONES TENEMOS?

- Usar los valores por defecto

¿QUÉ OPCIONES TENEMOS?

- Usar los valores por defecto
- Service Locator

¿QUÉ OPCIONES TENEMOS?

- Usar los valores por defecto
- Service Locator
- Inyector de dependencias

Valores por defecto

- Funcionalidad de Kotlin
- Nos permite hacer inyecciones muy sencillas
- No requiere de mucho aprendizaje así que es fácil empezar a usarlo
- Problemas principales
  - Si no podemos proveer una dependencia transitiva, estamos en la misma situación
  - Solo nos vale para Unit Testing

VALORES POR DEFECTO - EJEMPLO

```
class MoviesRepository(
    private val localDataSource: LocalDataSource = RoomDataSource(),
    private val remoteDataSource: RemoteDataSource = TheMovieDbDataSource(),
    private val regionRepository: RegionRepository = PlayServicesLocationDataSource(),
    private val apiKey: String = MoviesApp.instance.getString(R.string.api_key)
)
```

Service locator

- Componente que provee los servicios a la aplicación
- Todos estos servicios viven a nivel de aplicación
- Se crean en el Application
- Se accede al locator desde cualquier parte

SERVICE LOCATOR - EJEMPLO

```
SL.put<LocalDataSource>(RoomDataSource(db))
SL.put<RemoteDataSource>(TheMovieDbDataSource())
SL.put<LocationDataSource>(PlayServicesLocationDataSource(this))
SL.put<PermissionChecker>(AndroidPermissionChecker(this))
SL.put(RegionRepository(SL.get(), SL.get()))
SL.put(MoviesRepository(SL.get(), SL.get(), SL.get()))
```

Inyector de dependencias

- Las dependencias se proveen en vez de tener que ir nosotros a buscarlas
- Se suelen basar en un grafo de dependencias
- Se pueden crear grafos locales que mueren con el objeto asociado.
- Solución más compleja pero más flexible.

INYECTOR DE DEPENDENCIAS - EJEMPLO

```
@Inject
lateinit var viewModel: MainViewModel
```

INYECTOR DE DEPENDENCIAS

- Hay 2 inyectores principales en Android

INYECTOR DE DEPENDENCIAS

- Hay 2 inyectores principales en Android
  - Dagger
    - La opción clásica, algo más eficiente
    - Más complejo de entender y configurar

INYECTOR DE DEPENDENCIAS

- Hay 2 inyectores principales en Android
  - Dagger
    - La opción clásica, algo más eficiente
    - Más complejo de entender y configurar
  - Koin
    - Mucho más sencillo
    - Aprovecha el potencial de Kotlin
    - Aún no está muy demostrado su potencial en producción