En el `build.gradle.kts` raíz del proyecto:

```
plugins {
    ...
    id("com.google.devtools.ksp") version "1.9.0-1.0.12" apply false
    id("com.google.dagger.hilt.android") version "2.48" apply false
}
```

En el `build.gradle.kts` del la app:

```
plugins {
    ...
    id("com.google.devtools.ksp")
    id ("com.google.dagger.hilt.android")
}
```

En las dependencias de `build.gradle.kts` del la app:

```
android {
    ...
dependencies {
    ...
    implementation("com.google.dagger:hilt-android:2.44")
    implementation("androidx.hilt:hilt-navigation-compose:1.0.0")
    ksp("com.google.dagger:hilt-compiler:2.48")
}
```

IMPORTANTE!! **No olvides sincronizar**

1. Deberemos añadir la clase `Application` como ya hemos hecho en otras ocasiones, pero esta vez le tendremos que agregar la anotación `@HiltAndroidApp`

```
// Indicamos que es la clase principal de la aplicación
// Esta clase se ejecutará antes que cualquier otra clase de la aplicación
// y se encargará de proveer las dependencias a los módulos que las requieran
@HiltAndroidApp
class MiApplication : Application() { ... }
```

Recuerda a continuación, en el `AndroidManifest.xml` añadir la clase que acabamos de crear como la clase principal de la aplicación si no la habías hecho ya:

```
<application
  android:name=".MiApplication"
  android:allowBackup="true"
  ...>

  ...

</application>
```

2. Añadiremos la anotación `@AndroidEntryPoint` a la MainActivity.

```
@AndroidEntryPoint
class MainActivity : ComponentActivity() { ... }
```

3. El siguiente paso sería **exponer la inyección de dependencia** de los objetos que se inyectan en los constructores donde las necesitemos, en nuestro caso en el repositorio UsuarioRepository del objeto de tipo UsuarioDaoMock.

```
class UsuarioRepository @Inject constructor(
private val proveedorUsuarios: UsuarioDaoMock)
```

NOTA: Creamos una Injección de constructor en claseRepository del objeto tipo claseDaoMock

4. Ahora pasaremos a definir la clase con los métodos proveedores de las instancias, para ello crearemos el paquete **di** en la raiz de nuestro proyecto y dentro crearemos la clase **AppModule**, como se explica en los apuntes, no olvides etiquetarla como `@Module InstallIn(SingletonComponent::class)` :

```
@Module
@InstallIn(SingletonComponent::class)
class AppModule {
    ...
}
```

NOTA: Crear package “di” en “com.pmdm.ejemplo” y dentro la clase AppModule
// Mirar arquitectura para entender mejor //

Dentro de esta clase crearemos los **métodos proveedores de dependencias**, en nuestro caso solo tenemos un repositorio que debe ser inyectado con un UsuarioDaoMock, por lo que definiremos el método que provee la instancia a inyectar. Y el proveedor de repositorio al que se le inyecta la instancia del UsuarioDaoMock

```
@Provides
@Singleton
fun provideUsuarioDaoMock():UsuarioDaoMock= UsuarioDaoMock()

@Provides
@Singleton
fun provideUsuarioRepository(
        usuarioDaoMock: UsuarioDaoMock
    ): UsuarioRepository =
        UsuarioRepository(usuarioDaoMock)
```

NOTA: Crear un Provides Singleton por cada DaoMock y por cada Repository
// El Repository necesita el DaoMock y el Dao Mock tendrá un constructor vacío //

5. Para finalizar nos faltaría etiquetar con `@HiltViewModel` los ViewModel y en el constructor de estos inyectaremos mediante Hilt las instancias de los **objetos colaboradores** (repositorios), *ya **no** crearemos las instancias de los repositorios dentro del ViewModel.*

```
@HiltViewModel
class LoginViewModel  @Inject constructor(private  val usuarioRepository: UsuarioReposi
```

NOTA: Quiere decir que en el ViewModel tendríamos un objeto claseRepository, ya no está iniciado dentro, sino en el contructor inyectado

NOTA SOBRE PREVIEW:

```
@Composable
fun MiScreen(miVm: MiViewModel = hiltViewModel()) { ... }
```

SI NECESITAMOS UN @Preview, tenemos que crearle el view model así
Otro ejemplo debajo

```
@Preview(showBackground = true)
@Composable
fun LoginScreenPreview(miVm: LoginViewModel = hiltViewModel()) {
    LoginScreen(loginUiState = miVm.loginUiState, eventos = miVm::onLoginEvent)
}
```