



# Dependency Injection

---

PRAKTIKUM PEMROGRAMAN MOBILE #5

Wanda Gusdya Purnama  
TEKNIK INFORMATIKA UNIVERSITAS PASUNDAN  
2025

# Pendahuluan

Dependency injection adalah sebuah teknik dalam pemrograman yang bertujuan untuk memisahkan penciptaan objek dengan penggunaannya. Hal ini memungkinkan pengembang untuk membuat kode yang lebih modular, mudah diuji, dan mudah dikelola.

Pada dasarnya, dependency injection bekerja dengan cara menyuntikkan (inject) dependensi yang dibutuhkan oleh sebuah objek dari luar. Sehingga, sebuah objek tidak perlu membuat atau mencari dependensi tersebut. Ada tiga jenis utama dependency injection: constructor injection, setter injection, dan interface injection.

Constructor injection menyuntikkan dependensi melalui konstruktor objek, sementara setter injection menggunakan metode setter untuk menyuntikkan dependensi. Interface injection, meskipun jarang digunakan, melibatkan implementasi antarmuka khusus yang menyuntikkan dependensi ke dalam objek. Modul ini akan menggunakan dependency injection dengan jenis constructor injection.

Dengan menggunakan dependency injection, kita dapat mengurangi tight coupling (kepaduan erat) antar komponen dalam sistem, sehingga perubahan pada satu komponen tidak mempengaruhi komponen lainnya secara signifikan. Hal ini juga memudahkan pengujian unit (unit testing), karena kita dapat dengan mudah mengganti dependensi tertentu dengan mock object atau stub, yang memungkinkan kita untuk mengisolasi bagian kode yang ingin diuji.

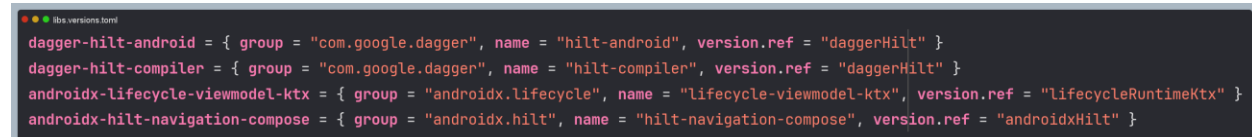
## Latihan 1

Pada latihan ini, kita akan menggunakan project MyNote dari modul 4. Silakan buka project MyNote pada Android Studio. Pada bagian [versions] di file libs.versions.toml, tambahkan kode berikut



```
libs.versions.toml
daggerHilt = "2.56.1"
androidxHilt = "1.2.0"
```

Kemudian, di bagian [libraries], tambahkan kode berikut



```
libs.versions.toml
dagger-hilt-android = { group = "com.google.dagger", name = "hilt-android", version.ref = "daggerHilt" }
dagger-hilt-compiler = { group = "com.google.dagger", name = "hilt-compiler", version.ref = "daggerHilt" }
androidx-lifecycle-viewmodel-ktx = { group = "androidx.lifecycle", name = "lifecycle-viewmodel-ktx", version.ref = "lifecycleRuntimeKtx" }
androidx-hilt-navigation-compose = { group = "androidx.hilt", name = "hilt-navigation-compose", version.ref = "androidxHilt" }
```

Kemudian, di bagian [plugins], tambahkan kode berikut

libs.versions.toml

```
hilt-android = { id = "com.google.dagger.hilt.android", version.ref = "daggerHilt" }
```

Sekarang, file libs.versions.toml akan terlihat seperti berikut

libs.versions.toml

```
[versions]
agp = "8.9.2"
kotlin = "2.1.20"
coreKtx = "1.16.0"
junit = "4.13.2"
junitVersion = "1.2.1"
espressoCore = "3.6.1"
lifecycleRuntimeKtx = "2.8.7"
activityCompose = "1.10.1"
composeBom = "2025.04.01"
room = "2.7.1"
ksp = "2.1.20-2.0.0"
benasher44Uuid = "0.8.4"
daggerHilt = "2.56.1"
androidxHilt = "1.2.0"

[libraries]
androidx-core-ktx = { group = "androidx.core", name = "core-ktx", version.ref = "coreKtx" }
junit = { group = "junit", name = "junit", version.ref = "junit" }
androidx-junit = { group = "androidx.test.ext", name = "junit", version.ref = "junitVersion" }
androidx-espresso-core = { group = "androidx.test.espresso", name = "espresso-core", version.ref = "espressoCore" }
androidx-lifecycle-runtime-ktx = { group = "androidx.lifecycle", name = "lifecycle-runtime-ktx", version.ref = "lifecycleRuntimeKtx" }
androidx-activity-compose = { group = "androidx.activity", name = "activity-compose", version.ref = "activityCompose" }
androidx-compose-bom = { group = "androidx.compose", name = "compose-bom", version.ref = "composeBom" }
androidx-ui = { group = "androidx.compose.ui", name = "ui" }
androidx-ui-graphics = { group = "androidx.compose.ui", name = "ui-graphics" }
androidx-ui-tooling = { group = "androidx.compose.ui", name = "ui-tooling" }
androidx-ui-tooling-preview = { group = "androidx.compose.ui", name = "ui-tooling-preview" }
androidx-ui-test-manifest = { group = "androidx.compose.ui", name = "ui-test-manifest" }
androidx-ui-test-junit4 = { group = "androidx.compose.ui", name = "ui-test-junit4" }
androidx-material3 = { group = "androidx.compose.material3", name = "material3" }
androidx-room-runtime = { group = "androidx.room", name = "room-runtime", version.ref = "room" }
androidx-room-ktx = { group = "androidx.room", name = "room-ktx", version.ref = "room" }
androidx-room-compiler = { group = "androidx.room", name = "room-compiler", version.ref = "room" }
androidx-lifecycle-livedata-ktx = { group = "androidx.lifecycle", name = "lifecycle-livedata-ktx", version.ref = "lifecycleRuntimeKtx" }
androidx-compose-runtime-livedata = { group = "androidx.compose.runtime", name = "runtime-livedata" }
benasher44-uuid = { group = "com.benasher44", name = "uuid", version.ref = "benasher44Uuid" }
dagger-hilt-android = { group = "com.google.dagger", name = "hilt-android", version.ref = "daggerHilt" }
dagger-hilt-compiler = { group = "com.google.dagger", name = "hilt-compiler", version.ref = "daggerHilt" }
androidx-lifecycle-viewmodel-ktx = { group = "androidx.lifecycle", name = "lifecycle-viewmodel-ktx", version.ref = "lifecycleRuntimeKtx" }
androidx-hilt-navigation-compose = { group = "androidx.hilt", name = "hilt-navigation-compose", version.ref = "androidxHilt" }

[plugins]
android-application = { id = "com.android.application", version.ref = "agp" }
kotlin-android = { id = "org.jetbrains.kotlin.android", version.ref = "kotlin" }
kotlin-compose = { id = "org.jetbrains.kotlin.plugin.compose", version.ref = "kotlin" }
room = { id = "androidx.room", version.ref = "room" }
ksp = { id = "com.google.devtools.ksp", version.ref = "ksp" }
hilt-android = { id = "com.google.dagger.hilt.android", version.ref = "daggerHilt" }
```

Lakukan sinkronisasi project gradle, kemudian buka file build.gradle.kts (Project: MyNote) dan tambahkan kode berikut di bagian plugins

```
build.gradle.kts

alias(libs.plugins.hilt.android, com.google.dagger.hilt.android:2.56.1) apply false
```

Sehingga file build.gradle.kts (Project: MyNote) akan terlihat seperti berikut

```
build.gradle.kts

// Top-level build file where you can add configuration options common to all sub-projects/modules.
plugins {
    alias(libs.plugins.android.application, com.android.application:8.9.2) apply false
    alias(libs.plugins.kotlin.android, org.jetbrains.kotlin.android:2.1.20) apply false
    alias(libs.plugins.kotlin.compose, org.jetbrains.kotlin.plugin.compose:2.1.20) apply false
    alias(libs.plugins.ksp, com.google.devtools.ksp:2.1.20-2.0.0) apply false
    alias(libs.plugins.room, androidx.room:2.7.1) apply false
    alias(libs.plugins.hilt.android, com.google.dagger.hilt.android:2.56.1) apply false
}
```

Selanjutnya, buka file build.gradle.kts (Module: app) dan tambahkan kode berikut pada bagian plugins

```
build.gradle.kts

alias(libs.plugins.hilt.android, com.google.dagger.hilt.android:2.56.1)
```

Kemudian di bagian dependencies, tambahkan kode berikut

```
build.gradle.kts

implementation(libs.dagger.hilt.android, com.google.dagger:hilt-android:2.56.1)
ksp(libs.dagger.hilt.compiler, com.google.dagger:hilt-compiler:2.56.1)
implementation(libs.androidx.lifecycle.viewmodel.ktx, androidx.lifecycle:lifecycle-viewmodel-ktx:2.8.7)
implementation(libs.androidx.hilt.navigation.compose, androidx.hilt:hilt-navigation-compose:1.2.0)
```

Sehingga keseluruhan file build.gradle akan terlihat seperti berikut

● ● ● build.gradle.kts

```
plugins {  
    alias(libs.plugins.android.application) com.android.application:8.9.2 )  
    alias(libs.plugins.kotlin.android) org.jetbrains.kotlin.android:2.1.20 )  
    alias(libs.plugins.kotlin.compose) org.jetbrains.kotlin.plugin.compose:2.1.20 )  
    alias(libs.plugins.ksp) com.google.devtools.ksp:2.1.20-2.0.0 )  
    alias(libs.plugins.room) androidx.room:2.7.1 )  
    alias(libs.plugins.hilt.android) com.google.dagger.hilt.android:2.56.1 )  
}  
  
android {  
    namespace = "id.ac.unpas.mynote"  
    compileSdk = 36  
  
    defaultConfig {  
        applicationId = "id.ac.unpas.mynote"  
        minSdk = 24  
        targetSdk = 36  
        versionCode = 1  
        versionName = "1.0"  
  
        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"  
    }  
  
    buildTypes {  
        release {  
            isMinifyEnabled = false  
            proguardFiles(  
                getDefaultProguardFile("proguard-android-optimize.txt"),  
                "proguard-rules.pro"  
            )  
        }  
    }  
  
    compileOptions {  
        sourceCompatibility = JavaVersion.VERSION_11  
        targetCompatibility = JavaVersion.VERSION_11  
    }  
}
```

```

kotlinOptions {
    jvmTarget = "11"
}
buildFeatures {
    compose = true
}
room {
    schemaDirectory("$projectDir/schemas")
}
}

dependencies {

    implementation(libs.androidx.core.ktx androidx.core:core-ktx:1.16.0 )
    implementation(libs.androidx.lifecycle.runtime.ktx androidx.lifecycle:lifecycle-runtime-ktx:2.8.7 )
    implementation(libs.androidx.activity.compose androidx.activity:activity-compose:1.10.1 )
    implementation(platform(libs.androidx.compose.bom androidx.compose:compose-bom:2025.04.01 ))
    implementation(libs.androidx.ui)
    implementation(libs.androidx.ui.graphics)
    implementation(libs.androidx.ui.tooling.preview)
    implementation(libs.androidx.material3)
    testImplementation(libs.junit junit:junit:4.13.2 )
    androidTestImplementation(libs.androidx.junit androidx.test.ext:junit:1.2.1 )
    androidTestImplementation(libs.androidx.espresso.core androidx.test.espresso:espresso-core:3.6.1 )
    androidTestImplementation(platform(libs.androidx.compose.bom androidx.compose:compose-bom:2025.04.01 ))
    androidTestImplementation(libs.androidx.ui.test.junit4)
    debugImplementation(libs.androidx.ui.tooling)
    debugImplementation(libs.androidx.ui.test.manifest)

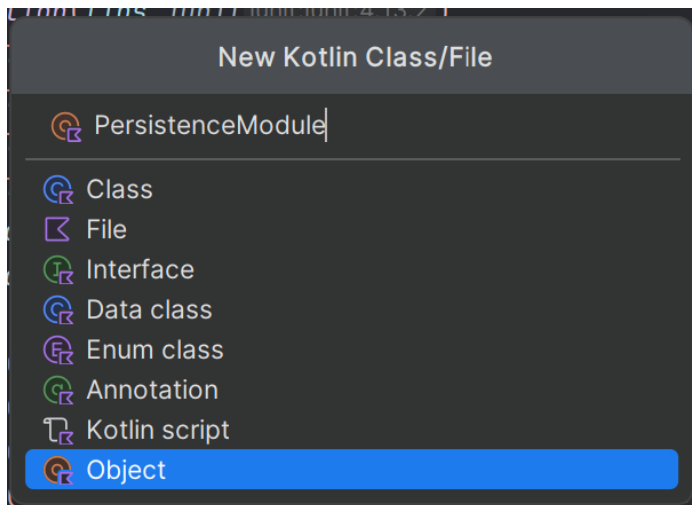
    implementation(libs.androidx.lifecycle.livedata.ktx androidx.lifecycle:lifecycle-livedata-ktx:2.8.7 )
    implementation(libs.androidx.compose.runtime.livedata)
    implementation(libs.androidx.room.runtime androidx.room:room-runtime:2.7.1 )
    implementation(libs.androidx.room.ktx androidx.room:room-ktx:2.7.1 )
    ksp(libs.androidx.room.compiler androidx.room:room-compiler:2.7.1 )
    implementation(libs.benasher44.uuid com.benasher44:uuid:0.8.4 )

    implementation(libs.dagger.hilt.android com.google.dagger:hilt-android:2.56.1 )
    ksp(libs.dagger.hilt.compiler com.google.dagger:hilt-compiler:2.56.1 )
    implementation(libs.androidx.lifecycle.viewmodel.ktx androidx.lifecycle:lifecycle-viewmodel-ktx:2.8.7 )
    implementation(libs.androidx.hilt.navigation.compose androidx.hilt:hilt-navigation-compose:1.2.0 )
}

```

Lakukan sinkronisasi project gradle sekali lagi.

Setelah proses sinkronisasi selesai, buat paket baru dengan nama di, kemudian buat kelas kotlin baru dengan nama PersistenceModule dan jenis Object.



Kelas PersistenceObject adalah kelas yang akan kita gunakan untuk mendefinisikan objek-objek yang akan digunakan berkali-kali pada aplikasi. Sebagai contoh, perhatikan kode berikut.

```
NoteScreen.kt
@Composable
fun NoteScreen(modifier: Modifier) {
    val context = LocalContext.current
    val dao = NoteDatabase.getDatabase(context).noteDao()
    val list : LiveData<List<Note>> = dao.getAllNotes()
    val notes: List<Note> by list.observeAsState(initial = listOf())
}
```

Dengan pendekatan modul 4, setiap kali kita perlu mengakses objek Dao, kita perlu memanggil fungsi `getDatabase` dari kelas `NoteDatabase`. Fungsi tersebut akan mencoba menghubungkan aplikasi dengan database setiap kali dipanggil. Penggunaan dependency injection dapat menyederhanakan pemanggilannya, sehingga koneksi hanya akan terjadi sekali ketika aplikasi dijalankan yang dapat membuat aplikasi menjadi efisien. Pada kasus lain, dependency injection juga dapat mengurangi kode boilerplate dari project sehingga project jadi lebih mudah untuk dirawat.

Langkah berikutnya, tambahkan kode berikut untuk kelas `PersistenceModule`.

```

PersistenceModule.kt

package id.ac.unpas.mynote.di

import android.app.Application
import androidx.room.Room
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.components.SingletonComponent
import id.ac.unpas.mynote.NoteDatabase
import id.ac.unpas.mynote.dao.NoteDao
import javax.inject.Singleton

@Module
@InstallIn(SingletonComponent::class)
object PersistenceModule {

    @Provides
    @Singleton
    fun provideAppDatabase(application: Application): NoteDatabase {
        return Room.databaseBuilder(
            application.applicationContext,
            NoteDatabase::class.java,
            "note_database"
        ).build()
    }

    @Provides
    @Singleton
    fun provideNoteDao(db: NoteDatabase): NoteDao {
        return db.noteDao()
    }
}

```



Selanjutnya, kita dapat menghapus fungsi `getDatabase` di kelas `NoteDatabase`. Sehingga, kelas `NoteDatabase` akan terlihat seperti berikut.

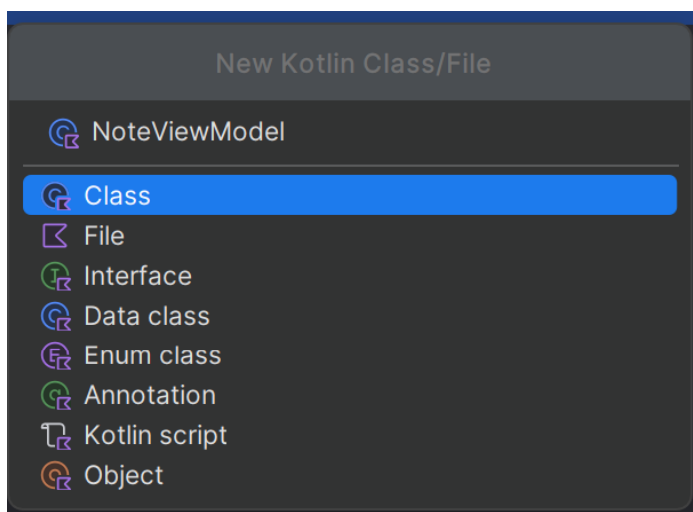
```
NoteDatabase.kt

package id.ac.unpas.mynote

import androidx.room.Database
import androidx.room.RoomDatabase
import id.ac.unpas.mynote.dao.NoteDao
import id.ac.unpas.mynote.models.Note

@Database(entities = [Note::class], version = 1)
abstract class NoteDatabase : RoomDatabase() {
    abstract fun noteDao(): NoteDao
}
```

Selanjutnya, kita akan memisahkan logika bisnis dengan logika antarmuka pengguna dengan menggunakan `ViewModel`. Buat kelas `NoteViewModel` di package utama.



ViewModel adalah komponen utama dalam arsitektur perangkat lunak yang digunakan untuk memisahkan logika bisnis dari logika antarmuka pengguna. Dalam konteks pengembangan aplikasi, ViewModel berfungsi sebagai jembatan antara UI (User Interface) dan data aplikasi. Dengan menggunakan ViewModel, logika bisnis dapat diisolasi dari aktivitas antarmuka pengguna sehingga meningkatkan keterbacaan dan pemeliharaan kode.

ViewModel menyimpan dan mengelola data yang dibutuhkan untuk antarmuka pengguna, serta bertanggung jawab untuk menyiapkan data tersebut agar dapat ditampilkan dengan benar. ViewModel juga mampu bertahan dari perubahan konfigurasi, seperti rotasi layar, yang membuatnya sangat berguna dalam pengembangan aplikasi Android.

Dalam pendekatan ini, ViewModel bekerja bersama LiveData untuk mengelola perubahan data secara efisien. LiveData memungkinkan UI untuk mengamati data yang dikelola oleh ViewModel, sehingga setiap perubahan data dapat langsung ditampilkan tanpa memerlukan permintaan data ulang.

Dengan menggunakan ViewModel, kita dapat meminimalkan kode boilerplate, mengurangi kompleksitas kode, dan memastikan bahwa aplikasi tetap responsif meskipun terjadi perubahan konfigurasi. Hal ini membuat pengembangan aplikasi menjadi lebih efisien dan mudah untuk dirawat.

Berikan kode berikut untuk kelas NoteViewModel yang baru kita buat.

```

NoteViewModel.kt

package id.ac.unpas.mynote

import androidx.lifecycle.LiveData
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import dagger.hilt.android.lifecycle.HiltViewModel
import id.ac.unpas.mynote.dao.NoteDao
import id.ac.unpas.mynote.models.Note
import kotlinx.coroutines.launch
import javax.inject.Inject

@HiltViewModel
class NoteViewModel @Inject constructor(private val noteDao: NoteDao) : ViewModel() {
    val list : LiveData<List<Note>> = noteDao.getAllNotes()

    fun insertNote(note: Note) {
        viewModelScope.launch {
            noteDao.insertNote(note)
        }
    }

    fun deleteNote(note: Note) {
        viewModelScope.launch {
            noteDao.deleteNote(note)
        }
    }
}

```

Selanjutnya, ubah Composable NoteScreen untuk mengambil dan mengubah data melalui ViewModel.

NoteScreen.kt

```
package id.ac.unpas.mynote

import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material3.Button
import androidx.compose.material3.Card
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.material3.TextField
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.livedata.observeAsState
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import com.benasher44.uuid.uuid4
import id.ac.unpas.mynote.models.Note

@Composable
fun NoteScreen(modifier: Modifier) {
    val viewModel = hiltViewModel<NoteViewModel>()
    val notes: List<Note> by viewModel.list.observeAsState(initial = listOf())

    var title by remember { mutableStateOf("") }
    var description by remember { mutableStateOf("") }

    Column(modifier = modifier.padding(16.dp)) {
        TextField(
            value = title,
            onChange = { title = it },
            label = { Text("Title") },
            modifier = Modifier.fillMaxWidth()
        )
        Spacer(modifier = Modifier.height(8.dp))
    }
}
```

```

NoteScreen.kt

TextField(
    value = description,
    onValueChange = { description = it },
    label = { Text("Description") },
    modifier = Modifier.fillMaxWidth()
)
Spacer(Modifier.height(8.dp))
Button(
    onClick = {
        if (title.isNotEmpty() && description.isNotEmpty()) {
            viewModel.insertNote(Note(uuid4().toString(), title, description))
            title = ""
            description = ""
        }
    },
    modifier = Modifier.fillMaxWidth()
) {
    Text("Save Note")
}

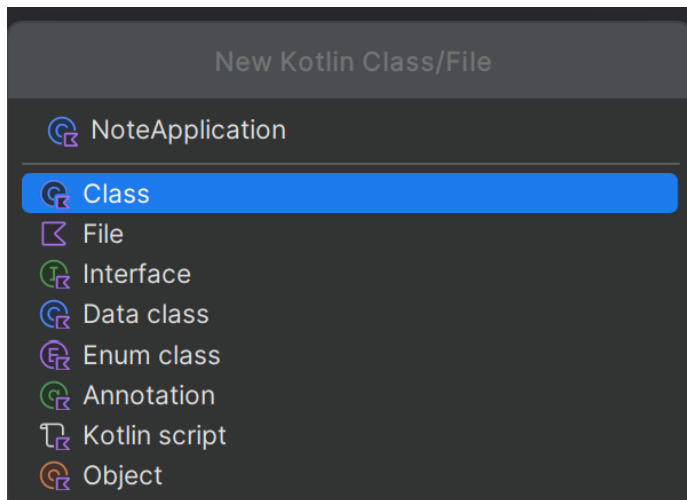
Spacer(Modifier.height(16.dp))

LazyColumn {
    items(notes) { note ->
        Card(
            modifier = Modifier
                .fillMaxWidth()
                .padding(vertical = 4.dp),
            onClick = {
                viewModel.deleteNote(note)
            }
        ) {
            Column(modifier = Modifier.padding(16.dp)) {
                Text(note.title, style = MaterialTheme.typography.titleMedium)
                Text(note.description, style = MaterialTheme.typography.bodyMedium)
            }
        }
    }
}
}
}

```

Perhatikan, kita tidak perlu lagi mengakses data secara langsung ke Database dan DAO, sehingga tidak perlu menggunakan variable context dan scope pada User Interface (UI). Hal ini memungkinkan UI untuk fokus pada logika UI.

Kemudian, buatlah kelas baru bernama NoteApplication pada package utama.



Berikut ini adalah kode untuk kelas tersebut.

```
NoteApplication.kt
package id.ac.unpas.mynote

import android.app.Application
import dagger.hilt.android.HiltAndroidApp

@HiltAndroidApp
class NoteApplication: Application()
```

Kemudian, bukalah file AndroidManifest.xml dan tambahkan atribut berikut pada tag application.

```
AndroidManifest.xml
android:name=".NoteApplication"
```

Sehingga, file AndroidManifest.xml akan terlihat seperti berikut.

```

AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="MyNote"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.MyNote"
        android:name=".NoteApplication"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="MyNote"
            android:theme="@style/Theme.MyNote">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>

```

Selanjutnya, tambahkan anotasi berikut pada kelas MainActivity.

```

MainActivity.kt
@AndroidEntryPoint

```

Sehingga, kelas MainActivity akan terlihat seperti berikut.

```

● ● ● MainActivity.kt
package id.ac.unpas.mynote

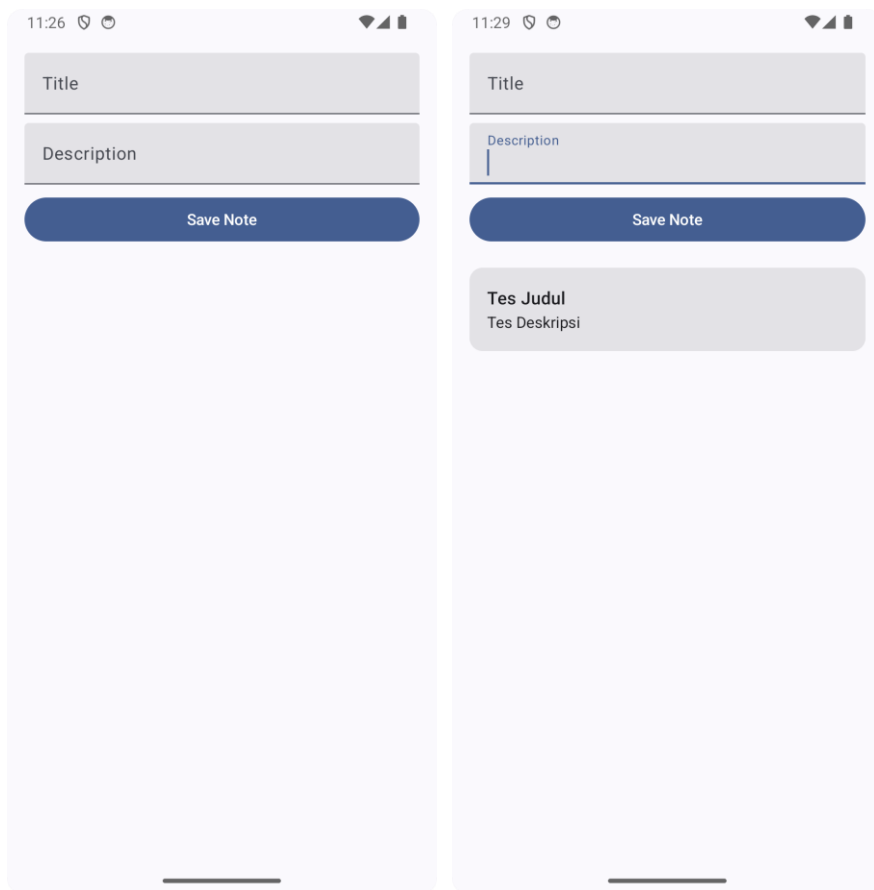
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.activity.enableEdgeToEdge
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Scaffold
import androidx.compose.ui.Modifier
import dagger.hilt.android.AndroidEntryPoint
import id.ac.unpas.mynote.ui.theme.MyNoteTheme

@AndroidEntryPoint
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            MyNoteTheme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    NoteScreen(Modifier.padding(innerPadding))
                }
            }
        }
    }
}

```

Jalankan aplikasi dan amati hasilnya. Seharusnya, tidak ada perbedaan apapun dari modul 4, namun sekarang pengembangan dan perawatan perangkat lunak akan menjadi lebih baik.





## Latihan 2

Modifikasi fitur update untuk aplikasi MyNote agar menggunakan ViewModel seperti fitur insert dan delete.