Chapitre 3
Les bases de Compose UI

1. Premiers pas

Avant de découvrir la bibliothèque Compose UI plus en détail, voyons pour commencer comment intégrer Jetpack Compose dans un projet Android.

1.1 Les dépendances Jetpack Compose

Faisons d'abord une mise au point sur les dépendances **Gradle** nécessaires pour utiliser Jetpack Compose. Nous avons vu dans le chapitre La genèse de Jetpack Compose que Jetpack Compose se découpe en trois parties principales : le **Compiler**, le **Runtime** et **Compose UI**.

Derrière ces trois parties se cachent en réalité plusieurs dépendances issues du repository Maven de Google. Actuellement, elles se distinguent en sept Groupes ID Maven principaux :

– `compose.compiler`
– `compose.runtime`
– `compose.ui`
– `compose.foundation`
– `compose.material`

– `compose.material3`

– `compose.animation`

Ces différentes dépendances ont un calendrier d'évolution de versions indépendantes les unes des autres. Cependant, pour éviter de déclarer chaque version de chaque dépendance, il existe le **BoM Compose**. Un BoM étant un module Maven regroupant un ensemble de bibliothèques et leurs versions. Il n'est pas obligatoire d'utiliser le BoM même s'il simplifie la gestion des dépendances. Nous utiliserons donc le BoM dans ce livre.

Remarque

*Pour rappel, un **Group ID** Maven rassemble une ou plusieurs dépendances identifiées unitairement par leur **Artifact ID**. On peut par exemple simplifier la nomenclature d'une dépendance par : **groupId:artifactId:version**. Ce qui donne par exemple **androidx.compose.ui:ui-tooling:1.1.1** où **androidx.compose.ui** est le groupId, **ui-tooling** est l'artifactId et 1.1.1 est la version de la dépendance.*

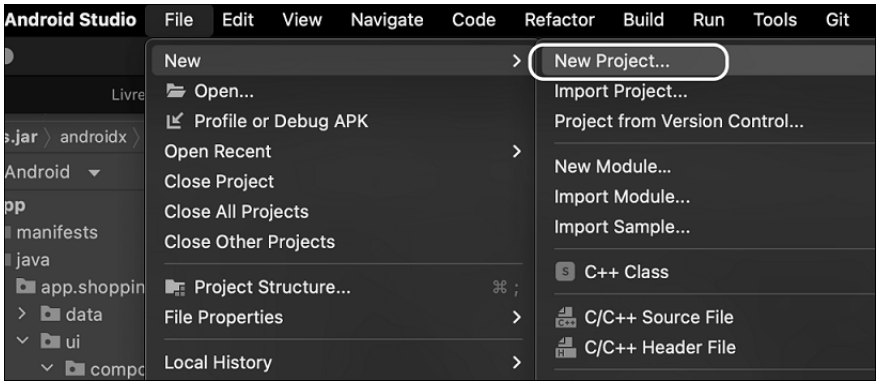
Remarque

Le groupID `compose.compiler` suit le rythme des évolutions de version du langage Kotlin. En effet, ce groupe correspond au compilateur de Compose qui n'est autre qu'un Kotlin Compiler Plugin. Son utilisation est donc étroitement liée à la version de Kotlin utilisée sur le projet. C'est pourquoi le cycle d'évolution des versions de celui-ci évolue conjointement aux évolutions de versions de Kotlin.

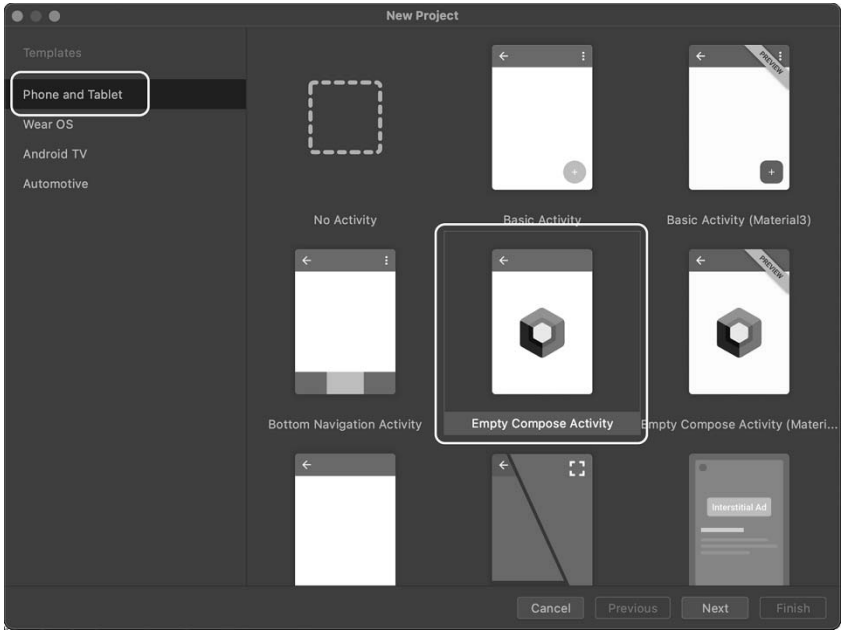
1.2 Créer un nouveau projet Android avec Jetpack Compose

Pour créer un nouveau projet Android avec Jetpack Compose, le plus simple est d'utiliser le créateur de projet Android Studio. Pour cela :

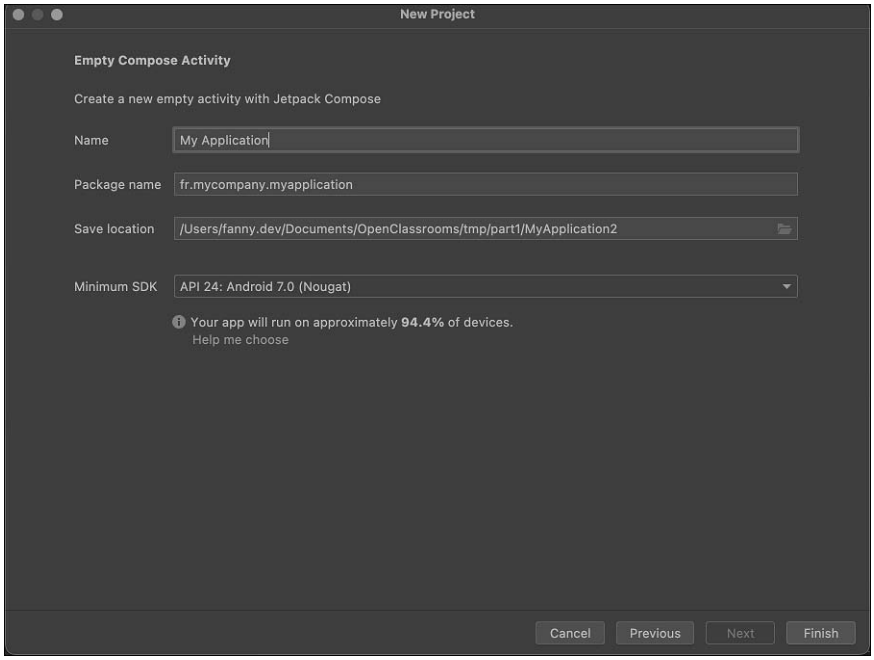
- Utilisez une version récente d'Android Studio.
- Cliquez sur **File - New - New Project...**.



- Dans la catégorie **Phone and Tablet**, cliquez sur **Empty Compose Activity**.



- Après avoir cliqué sur **Next**, vous êtes invité à nommer votre projet.



- Cliquez enfin sur **Finish**. Le projet est maintenant créé et sa compilation est en cours.

1.3 Ajouter Jetpack Compose à un projet existant

Pour ajouter Jetpack Compose à un projet existant :

Utilisez une version récente d'Android Studio (dans ce livre, nous utilisons la version Android Studio Electric Eel | 2022.1.1 RC 1).

- Dans le fichier `build.gradle` se trouvant à la racine du projet, au sein du bloc `buildscript`, configurez deux variables contenant la dernière version du BoM Compose et la dernière version du Compiler Compose.

```
buildscript {
    // Début du code à rajouter
    ext {
        // TODO remplacer les versions par les dernières versions disponibles
        compose_bom_version = '2022.10.00'
        compose_compiler_version = '1.2.0'
        kotlin_versin = '1.7.10'
    }
    // Fin du code à rajouter
}
```

Assurez-vous d'utiliser une version de Kotlin compatible avec la version du Compiler Compose. Pour vous aider vous pouvez retrouver dans la documentation officielle un tableau listant les versions du Compiler et de Kotlin compatibles (à l'adresse https://developer.android.com/jetpack/androidx/releases/compose-kotlin).

- Dans le fichier `build.gradle` de votre module au sein du bloc `android {}`, ajoutez la version du Compose Compiler.

```
android{
    // Début du code à rajouter
    buildFeatures {
        compose true
    }

    composeOptions {
        kotlinCompilerExtensionVersion compose_compiler_version
    }
    // Fin du code à rajouter
}
```

- Enfin, dans le fichier `build.gradle`, de votre module au sein du bloc `dependencies {}`, ajoutez les dépendances suivantes :

```
dependencies {
    // Début du code à rajouter
    def composeBom = platform("androidx.compose:
compose-   bom:$compose_bom_version")
    implementation composeBom

    // Material 2
    implementation 'androidx.compose.material:material'
    // Facultatif : Icones Material Design supplémentaires
    implementation 'androidx.compose.material:material-icons-extended'
    // Animations
```

```
    implementation "androidx.compose.animation:animation"
    // Outillage
    implementation 'androidx.compose.ui:ui-tooling-preview'
    debugImplementation 'androidx.compose.ui:ui-tooling'
    // Intégration avec les activités
    implementation "androidx.activity:activity-compose:1.6.1"

    // UI Tests
    androidTestImplementation composeBom
    androidTestImplementation 'androidx.compose.ui:ui-test-junit4'
    debugImplementation 'androidx.compose.ui:ui-test-manifest'
    // Fin du code à rajouter
}
```

▶Compilez le projet.

1.4 Le composant racine

Jetpack Compose est conçu pour être utilisé, dans l'idéal, avec une architecture contenant une seule activité et aucun fragment. Pour autant, Jetpack Compose inclut des API permettant de s'intégrer dans une application existante utilisant le UI Toolkit original avec plusieurs activités et plusieurs fragments. L'objectif de cette section est de découvrir les points d'entrée possibles au sein d'une application pour utiliser Jetpack Compose.

1.4.1 Une activité Jetpack Compose

C'est grâce à la fonction `setContent()` que nous allons pouvoir déclarer notre composant racine au sein d'une activité. Cette fonction succède à la fonction `setContentView()` utilisée dans le UI Toolkit original qui servait à lier le XML représentant notre vue à notre activité, comme ci-dessous :

```
class OriginalMainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main_original)
}
```

Mais revenons un instant au fonctionnement de base de Jetpack Compose. Lors de son exécution, une fonction Composable va émettre une représentation de son contenu qui sera alors stockée en mémoire dans l'arbre de composition. C'est en fait grâce à la fonction `setContent()` que cet arbre va pouvoir être initialisé. En effet, c'est cette fonction qui va créer la racine de l'arbre et qui va fournir implicitement le contexte de l'application à tous les enfants de l'arbre de composition. C'est également cette fonction qui fait le lien entre la plateforme Android et Compose Runtime.

```
class ComposeMainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            AppTheme {
                MainScreen()
            }
        }
    }
}
```

`setContent()` est une extension de fonction qui s'applique à la classe `ComponentActivity` et à ses sous-classes comme les classes `AppCompatActivity` ou `FragmentActivity`.

Remarque

Chaque fois que la fonction setContent est appelée, un nouvel arbre de composition est créé. Ainsi, si dans une application nous avons plusieurs fragments et activités qui cohabitent, alors plusieurs arbres de composition sont créés en parallèle et sont indépendants les uns des autres.

Remarque

Dans l'exemple ci-dessus, AppTheme permet de définir le style des composants. Nous verrons dans la suite de ce chapitre comment appliquer un thème à une application.

1.4.2 Jetpack Compose au sein d'une vue XML

L'intérêt de Jetpack Compose est qu'il est interopérable avec le UI Toolkit original. Il est donc possible de migrer progressivement vers Jetpack Compose une application existante utilisant le système de vue traditionnel ou alors de développer uniquement les nouveaux écrans en Compose. Pour cela, il existe un point d'entrée permettant d'initialiser un nouvel arbre de composition : `ComposeView`. C'est une classe de type `View` issue du UIToolkit original. Cette classe permet d'intégrer un composant Jetpack Compose dans une vue définie en XML.

Pour cela, suivez les étapes suivantes :

- Ajoutez une vue `ComposeView` dans votre code XML. Par exemple :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textview"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="XML TextView!" />

    <androidx.compose.ui.platform.ComposeView
        android:id="@+id/compose_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

- Dans le code source Kotlin associé à votre XML, définissez le contenu de votre `ComposeView`. Par exemple :

```
binding.compose_view.setContent {
    Text("Compose Text !")
}
```