



Applications Mobiles

420-5GM-BB - Cours 2

Pierre Prades & Mathieu Brodeur-Béliveau





Jetpack Compose

Jetpack Compose est un framework de **UI déclarative** développé par Google pour créer des interfaces Android plus rapidement et plus simplement.

Philosophie compose

- Dans compose les éléments graphiques ne sont pas des objets mais des fonctions.
- Composable
 - Une fonction Kotlin annotée avec **@Composable**
 - Elle définit une **partie de l'interface utilisateur**
- Preview (Aperçu)
 - Permet de voir le rendu de votre composable sans lancer l'app :

```
@Composable
```

```
fun Greeting(name: String, modifier: Modifier = Modifier) {  
    Text(  
        text = "Hello $name!",  
        modifier = modifier  
    )  
}
```

```
@Preview(showBackground = true)
```

```
@Composable
```

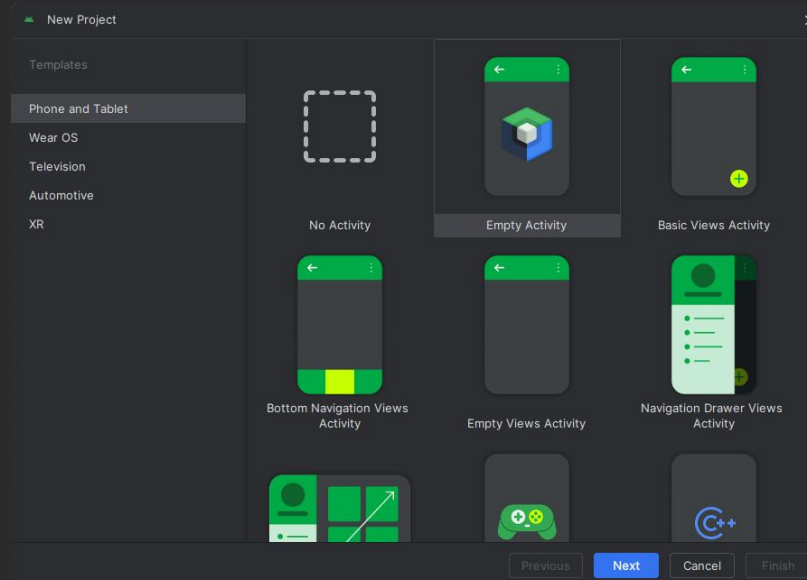
```
fun GreetingPreview() {  
    HappyBirthdayTheme {  
        Greeting("Android")  
    }  
}
```

Contrôle des éléments

- Les éléments sont contrôlés directement par les paramètres de la fonction.
- La modification des éléments graphiques est faite par l'intermédiaire d'états. Un événement (par exemple un click) va modifier l'état ce qui va causer la modification de l'IU. Lorsqu'un changement d'état cause une modification de l'interface, on parle alors de **recomposition**.

HappyBirthday

- Nous allons créer ensemble une nouvelle application intitulée « **HappyBirthday** », qui sera par la suite exécutée sur l'émulateur Android.
- Dans le menu File/New/New Project...
- Choisir « Empty Activity » puis « Next ». Appelez le projet « **HappyBirthday** ».



Happy Birthday

New Project

Empty Activity

Create a new empty activity with Jetpack Compose

Name

Package name

Save location

Minimum SDK

i Your app will run on approximately 93,4% of devices.
[Help me choose](#)

Build configuration language

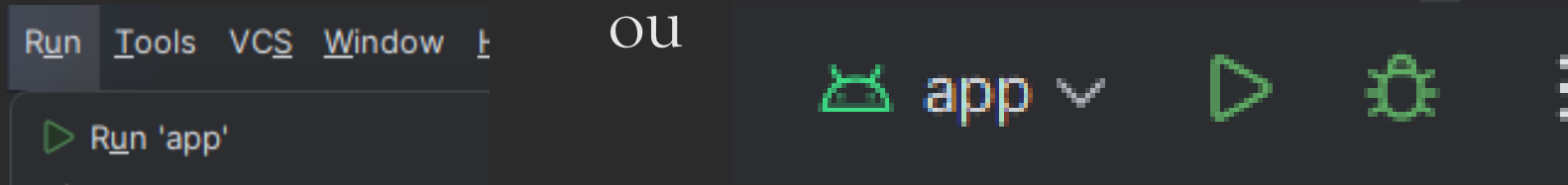
Previous Next Cancel Finish

HappyBirthday

- Lorsque vous ouvrez votre projet vous devriez avoir votre fichier MainActivity.kt ouvert.
- Dans ce fichier vous trouverez un exemple de code avec un composant graphique (un composable).
- Dans la fonction onCreate la fonction composable est appelée ce qui déclenche son intégration à la page.

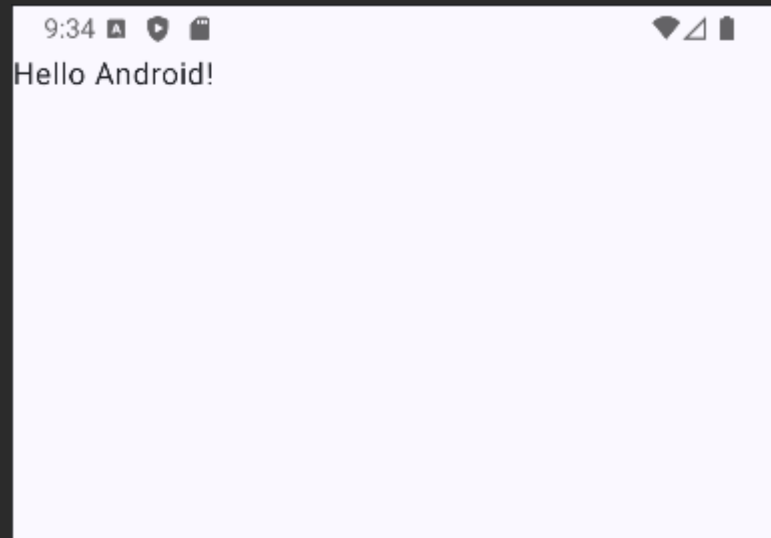
Lancer une application sur l'émulateur Android

- Si vous avez bien configuré vos machines virtuelles avec les bonnes versions de SDK d'Android, nous pouvons à partir d'Android Studio spécifier sur quelle machine virtuelle exécuter notre application. Assurez-vous de choisir une version égale ou supérieure à la version minimale supportée (tel que défini lors de la création du projet).
- Pour lancer l'application sur une machine virtuelle, il suffit d'appeler le menu:



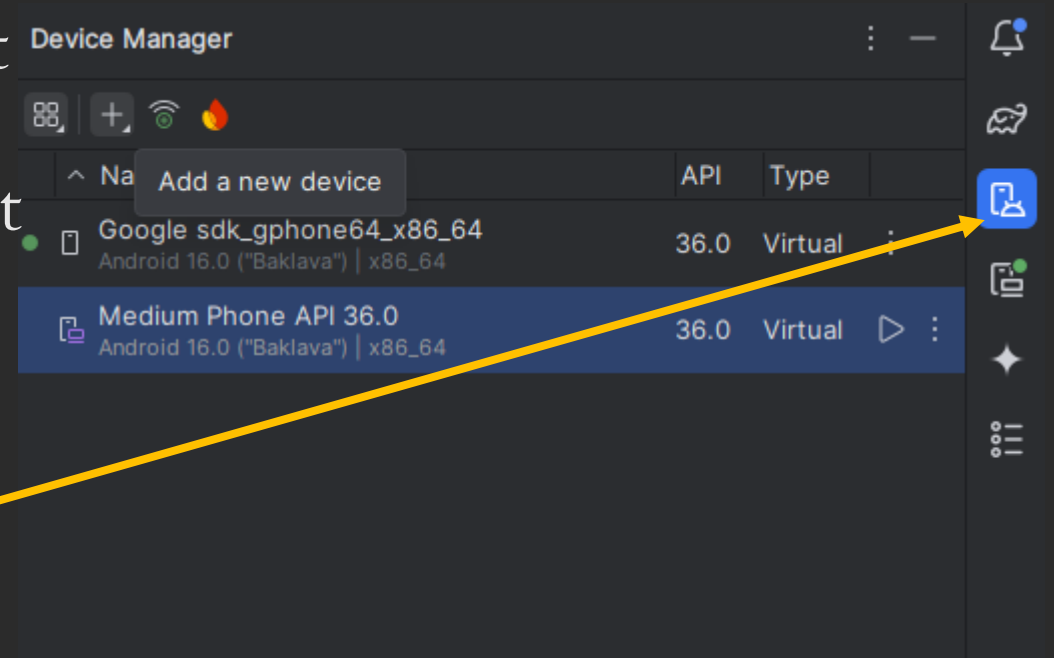
- Félicitations, vous avez créé votre première application Android!

- L'application s'exécute dans l'émulateur.

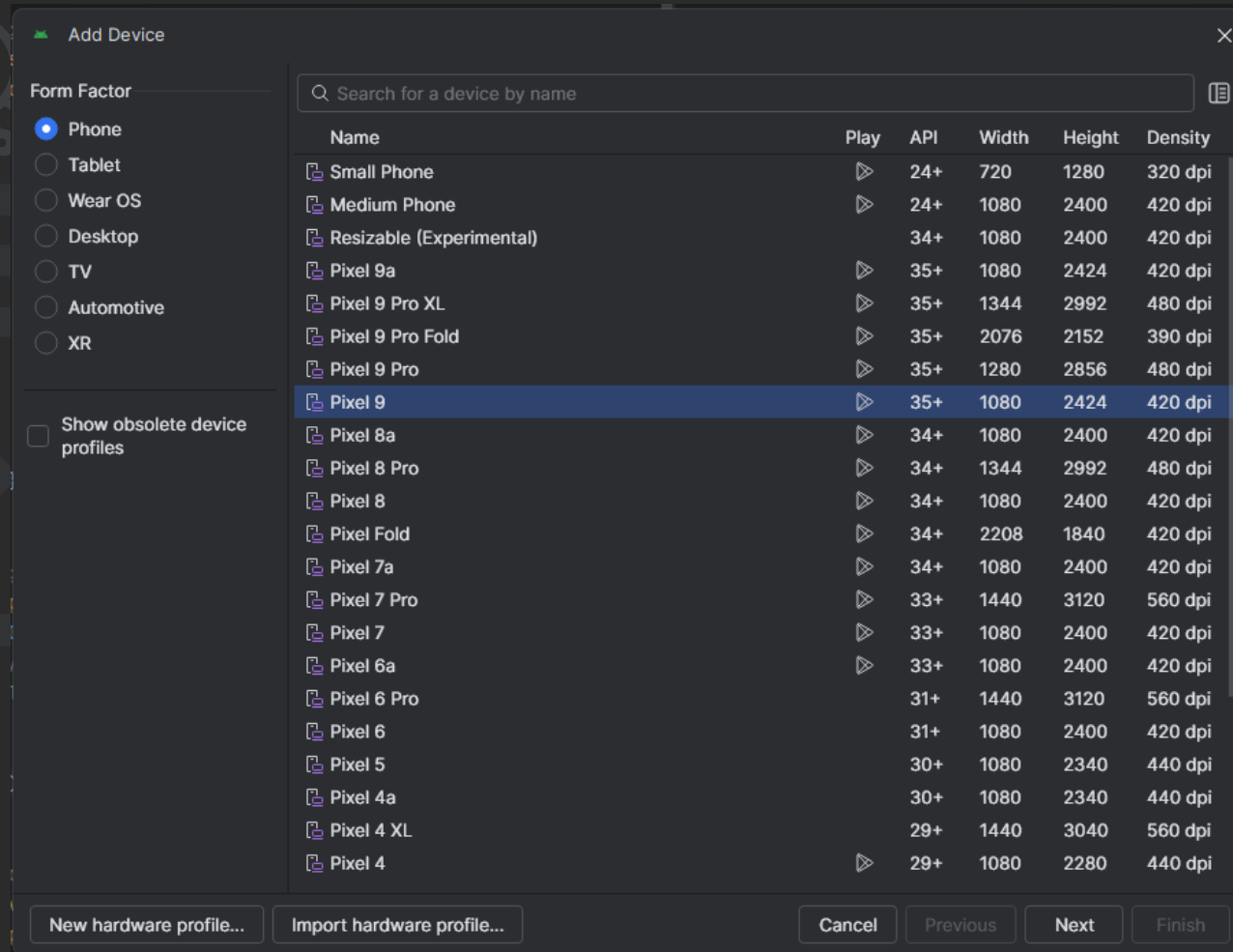


Création de machines virtuelles

- On pourrait même vouloir créer plusieurs machines virtuelles qui roulent sur la même version de SDK d'Android, pour simuler par exemple un cellulaire et une tablette ayant des écrans de tailles différentes.
- Ouvrez l'onglet Device manager pour accéder aux différentes machines virtuelles installées présentement sur votre ordinateur :



- Pour créer votre machine virtuelle vous allez devoir télécharger une image système en cliquant sur le + dans le Device Manager



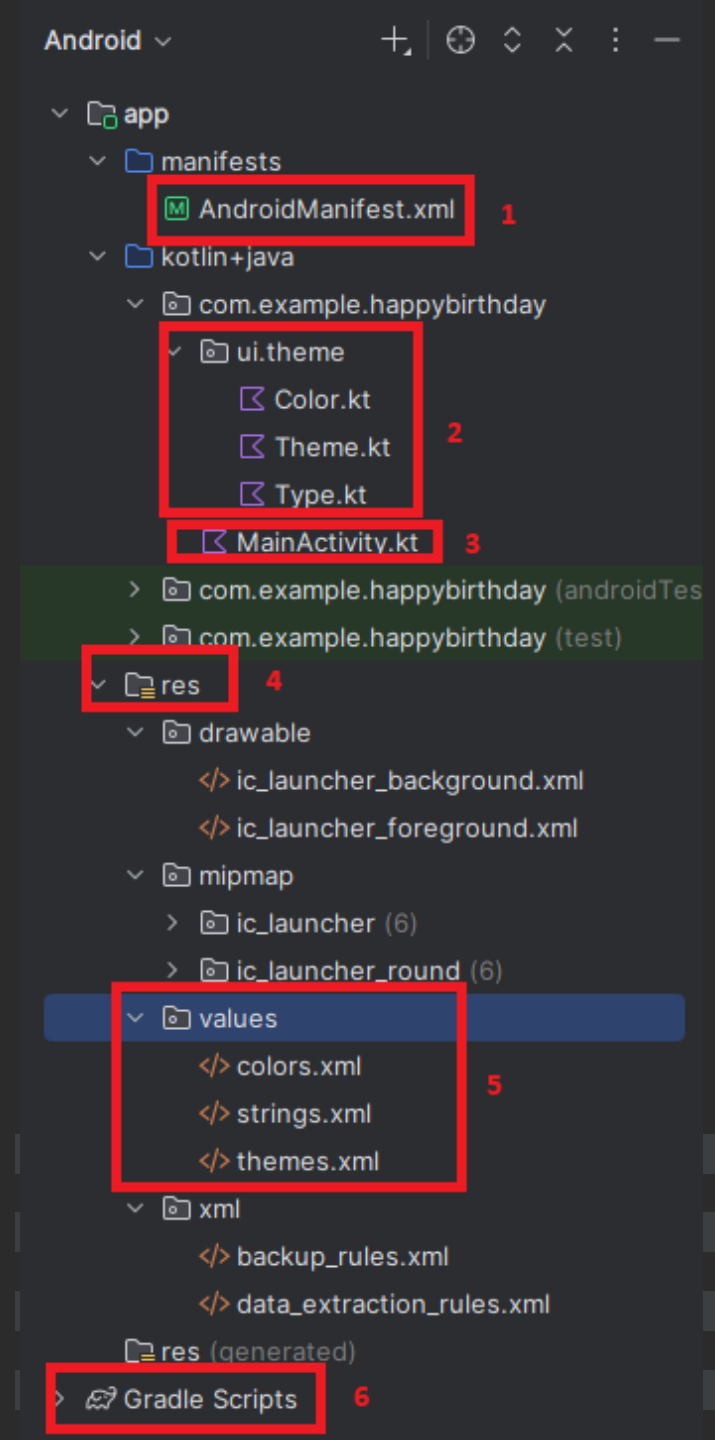
- La machine sera ajoutée à la liste des machines virtuelles.

Structure d'application Android

- Maintenant que nous avons une première application Android fonctionnelle, il faut comprendre comment ça marche en arrière pour pouvoir créer de nouvelles applications plus intéressantes et de plus grande envergure que Happy Birthday.
- Un simple programme Happy Birthday génère une multitude de dossiers et de fichiers, que l'on va expliquer brièvement :

Structure d'application Android

1. Le fichier « Manifest » est généré automatiquement pour vous. Il contient l'information générale de votre application. (API, Icones, Thème, activités, etc.)
2. Le répertoire ui.theme contient la définition du thème de l'application
3. Le fichier « MainActivity » contient le code source (Kotlin) pour l'activité principale.
4. Le répertoire «res» contient les ressources de l'app (images, icônes, ...).
5. Dans « values », comme on peut le voir, on va retrouver les strings, les couleurs, etc.
6. Fichiers Gradle, qui décrit comment l'application va compiler, se tester et se déployer.



Activity

- L'interface graphique d'une Activity peut être construite de deux façons, soit par programmation (directement dans le code du fichier .kt) soit par fichier XML dans un fichier de Layout associé.
- L'ancien mode de programmation forçait le découplage VUE et MODÈLE aux développeurs. Tout ce qui concerne le visuel était stocké dans un fichier XML de layout, et tout ce qui concerne la programmation était stocké dans un fichier .kt.
- Maintenant avec Compose on crée des application en intégrant directement les composants graphiques dans le code de la même manière qu'une page web.

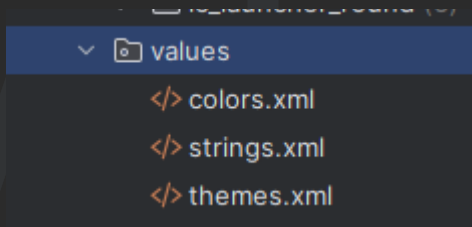


Introduction au dossier « res » (Ressources)



Le dossier Values

- Le dossier « values » contient les valeurs statiques propres à un ou plusieurs environnements.
- Dans le dossier « values » est contenu tout ce qui est par défaut :
 - Chaînes de caractères par défaut
 - Styles par défaut
 - Dimensions par défaut
 - Etc.
- Comme pour les layout, les fichiers de même nom représentent des caractéristiques différentes en fonction de:
 - la version du SDK
 - La taille de l'affichage
 - la langue de l'appareil
 - l'orientation de l'appareil (portrait vs paysage)
 - Etc.



Les chaînes de caractères (strings.xml)

- Lorsque vous créez un composant graphique de type text, et que vous voulez modifier ou ajouter du texte, il faut s'y prendre correctement. Si vous voulez supporter plusieurs langues vous devez récupérer vos chaînes depuis string.xml
- La valeur du texte ne sera pas hardcodée, elle sera plutôt stockée dans un fichier « strings.xml » sous le dossier « Values ». Toutes les chaînes de caractères devront obligatoirement être stockées à cet endroit.
- Par exemple, dans notre programme « Happy Birthday », si on veut afficher le nom de l'application on peut le récupérer en utilisant:

```
name = getString(R.string.app_name)
```

Dossier drawable

- Le dossier drawable contient toutes les images de l'application pour les différentes tailles d'écran.
- La densité des pixels est aussi considérée lorsqu'une taille d'image est choisie à l'exécution :
 - xxlarge (xxhdpi) l'écran a au moins 960dp x 720dp
 - xlarge (xhdpi) l'écran a au moins 960dp x 720dp
 - large (hdpi) l'écran a au moins 640dp x 480dp
 - normal (mdpi) l'écran a au moins 470dp x 320dp
 - small (ldpi) l'écran a au moins 426dp x 320dp
- Les mêmes noms de fichiers doivent se trouver dans chacun des dossiers dont la résolution est supportée, et Android va prendre le bon fichier selon l'écran de l'appareil utilisé par l'utilisateur.
- Ça veut dire que pour une application donnée, vous devez avoir plusieurs fichiers image identiques en aspect, mais avec plusieurs résolutions différentes.

- Pour une image donnée, voici les ratios que vous devez respecter pour garder une belle application peu importe l'écran utilisé:

drawable	Ratio à respecter
ldpi	1 : 0.75
mdpi	1 : 1
hdpi	1 : 1.5
xhdpi	1 : 2
xxhdpi	1 : 3

- Ça vaut autant pour les icônes de l'application que les images utilisées à l'intérieur de l'application.



Configure Image Asset

Icon Type: Launcher Icons (Legacy only) ▼

Name: ic_launcher

Asset Type: ☐ Image ☒ Clip Art ☐ TextClip Art: Trim: ☐ Yes ☒ NoPadding:  0 %

Foreground: 000000


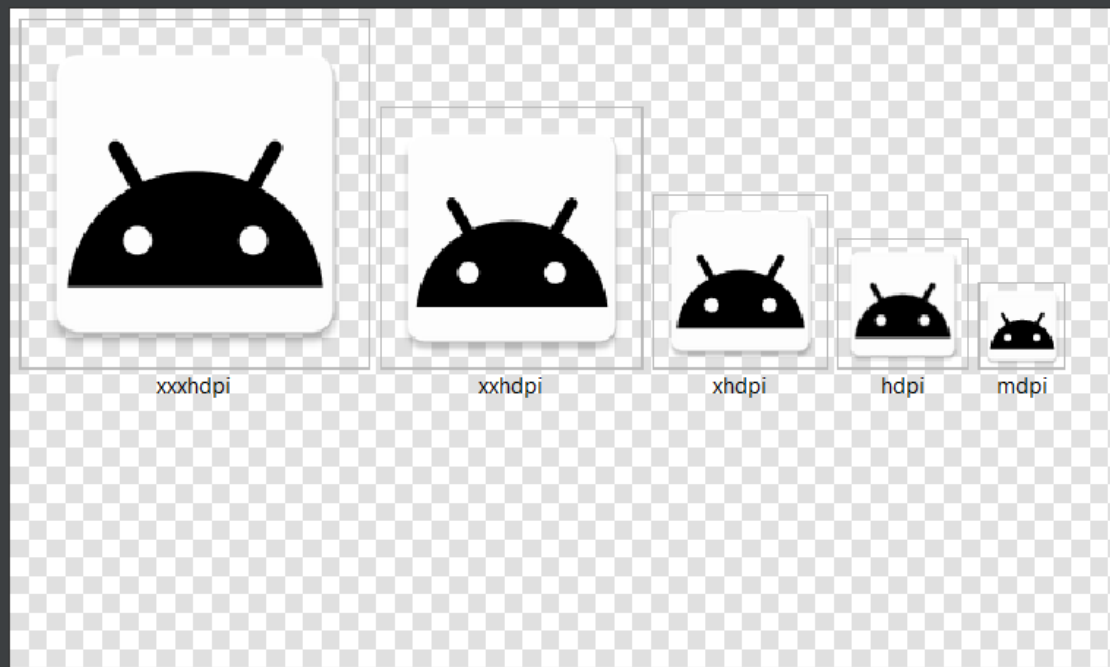
Background: FFFFFFFF

Scaling: ☐ Crop ☒ Shrink to Fit

Shape: Square ▼

Effect: ☒ None ☐ DogEar

Preview

 An icon with the same name already exists and will be overwritten.

Previous

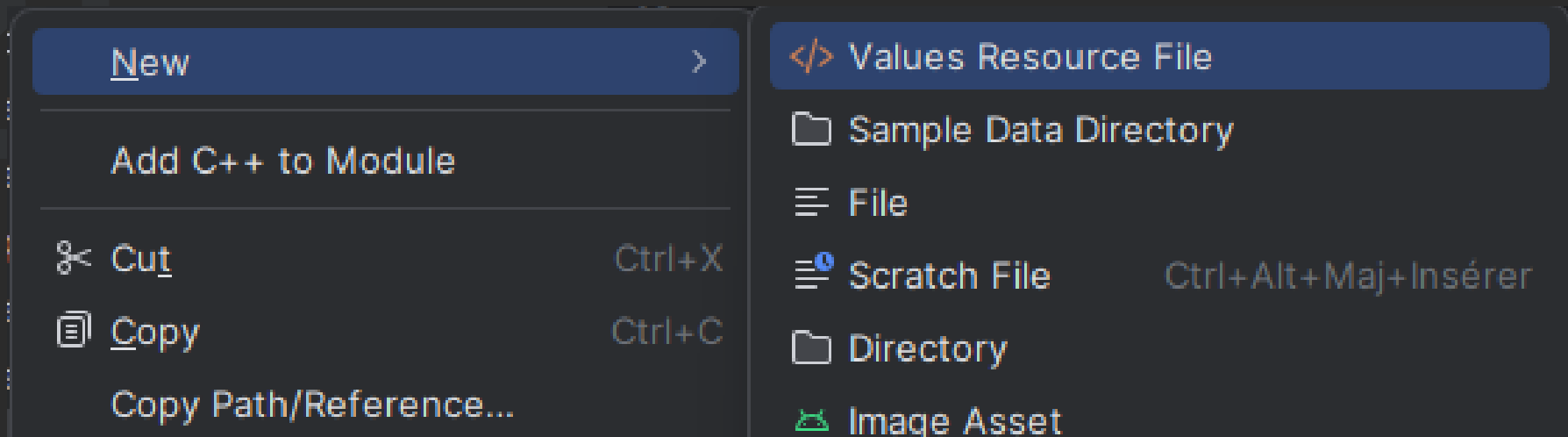
Next

Cancel

Finish

Gestion des types et caractéristiques d'appareils dans le dossier /res

- Pour ajouter une nouvelle ressource (fichier de values XML, images, etc.) dans un dossier avec différentes caractéristiques, il suffit de cliquer avec le bouton droit dans le dossier désiré, et choisir le menu « XYZ Resource file », où XYZ définit le type de ressource :



- Ensuite on spécifie le type de données à ajouter et les caractéristiques pour lesquelles le fichier s'applique :

New Resource File

File name: strings.xml

Root element: resources

Source set: main src/main/res

Directory name: values-en

Available qualifiers:

- Country Code
- Network Code
- Layout Direction
- Smallest Screen Width
- Screen Width
- Screen Height
- Size
- Ratio
- Roundness

Chosen qualifiers:

- en

Language:

- el: Greek
- en: English
- eo: Esperanto
- es: Spanish
- et: Estonian
- eu: Basque
- fa: Persian
- ff: Fulah

Tip: Type in list to filter

Specific Region Only:

- Any Region
- AE: United Arab Emirates
- AG: Antigua & Barbuda
- AI: Anguilla
- AS: American Samoa
- AT: Austria
- AU: Australia
- BB: Barbados

☐ Show All Regions

OK Cancel

- Attention : C'est fortement possible que votre nom de fichier existe déjà, et c'est normal car on veut en spécifier une nouvelle version.

Atelier guidé

- Nous allons maintenant modifier notre application pour en apprendre plus sur Compose.
- Pour débuter nous allons ajouter des paramètres à notre preview comme ceci:

```
@Preview(showBackground = true,  
showSystemUi = true,  
name = "B-Day app Preview")
```

- Supprimer maintenant la fonction Greeting() ainsi que toutes ses références.
- Source: [Créer une application simple avec des composables textuels](#)

Atelier guidé

- Ajouter un élément composable de type Text, nous allons l'appeler TexteAccueil.
- Il est recommandé d'ajouter un paramètre Modifier et de le transmettre à son premier enfant:

```
@Composable  
fun TexteAccueil(modifier: Modifier = Modifier) {}
```

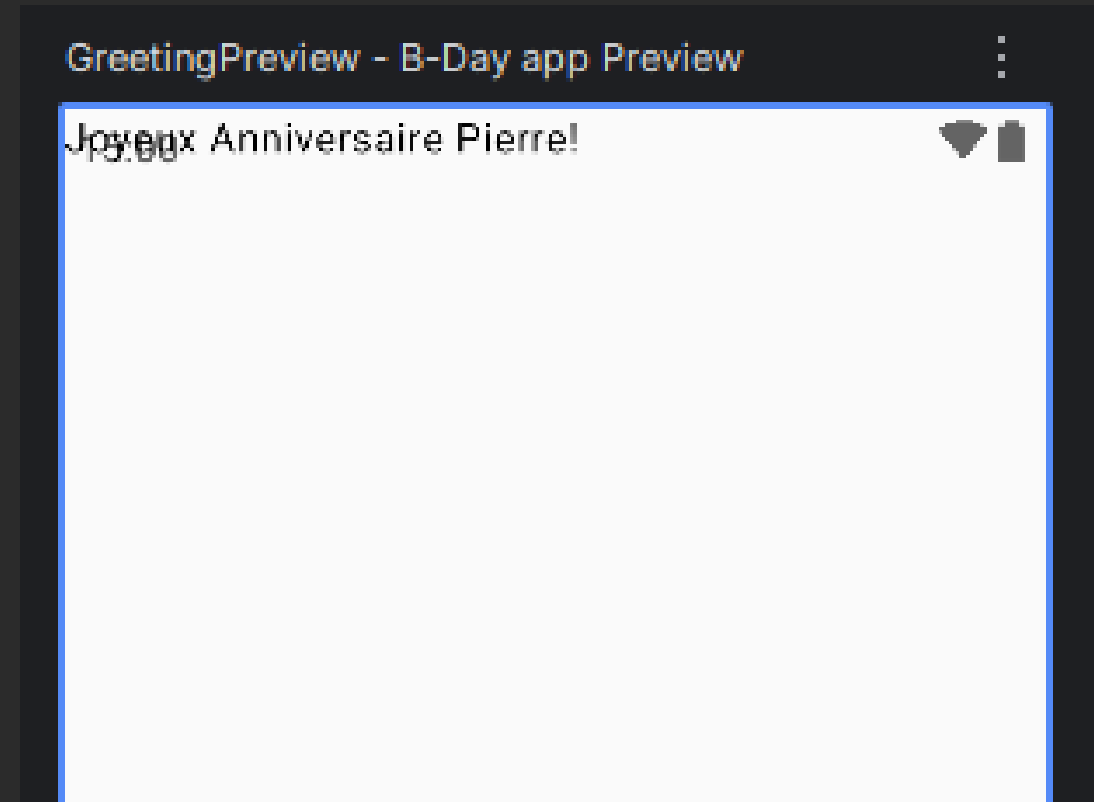
- On ajoute ensuite un paramètre message de type String que l'on va afficher dans une zone de texte:

```
@Composable  
fun TexteAccueil(message: String, modifier: Modifier = Modifier) {  
    Text(  
        text = message  
    )  
}
```


Afficher un aperçu des modifications

- Pour afficher un aperçu de votre application, appelez TexteAccueil dans la fonction GreetingPreview

```
fun GreetingPreview() {  
    HappyBirthdayTheme {  
        TexteAccueil(message = "Joyeux Anniversaire Pierre!")  
    }  
}
```

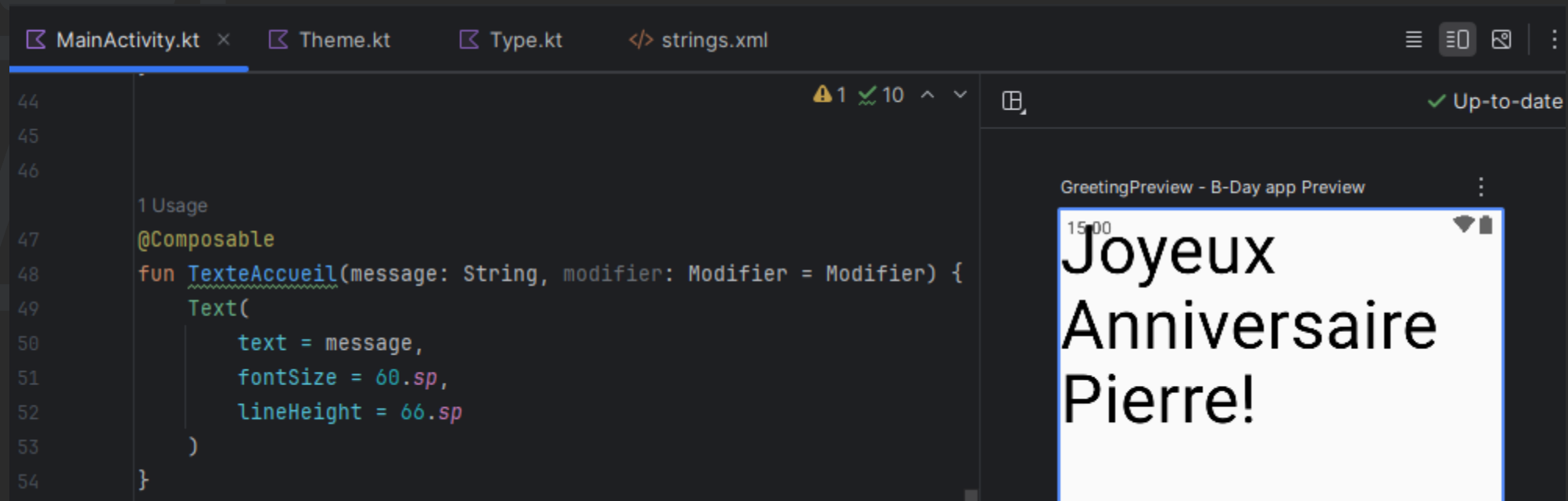


Modifier la taille de la police

- Pour modifier la taille de la police nous allons transmettre un attribut `fontSize` à la fonction `Text` attribuons-lui la valeur `60.sp`
- `.sp` génère une erreur car nous devons importer des classes ou propriétés pour pouvoir compiler.
- Ajouter l'import `androidx.compose.ui.unit.sp`
- Dans l'aperçu les lignes se chevauchent car nous n'avons pas spécifié la hauteur des lignes.
- Ajoutons l'attribut `lineHeight` avec une valeur de `66.sp`



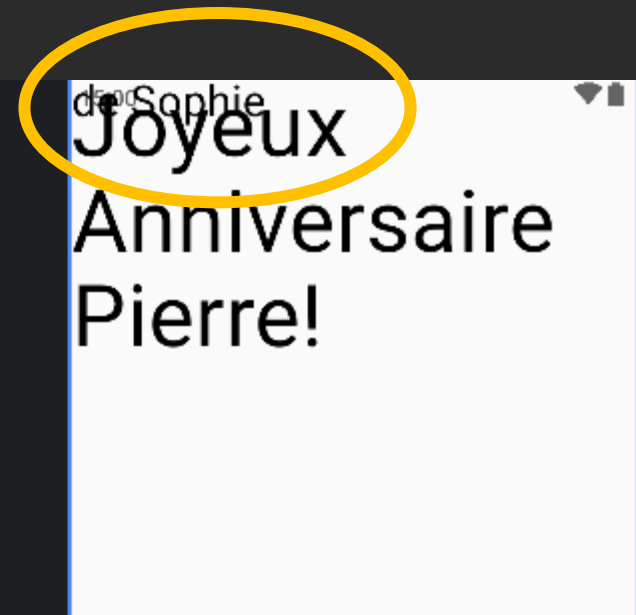
Résultat



Ajout d'un autre élément de texte

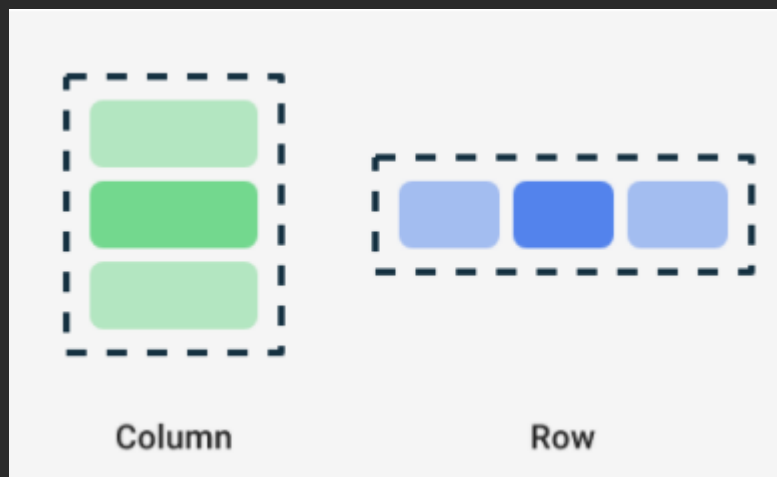
- Ajoutons maintenant l'auteur du message avec le champ auteur puis un composable Text qui affiche ce paramètre.
- Modifiez ensuite l'appel de la fonction pour spécifier l'auteur, par exemple: « de Sophie »

```
@Composable
fun TexteAccueil(message: String, auteur: String, modifier: Modifier) {
    Text(
        text = message,
        fontSize = 60.sp,
        lineHeight = 66.sp
    )
    Text(
        text = auteur,
        fontSize = 30.sp
    )
}
```



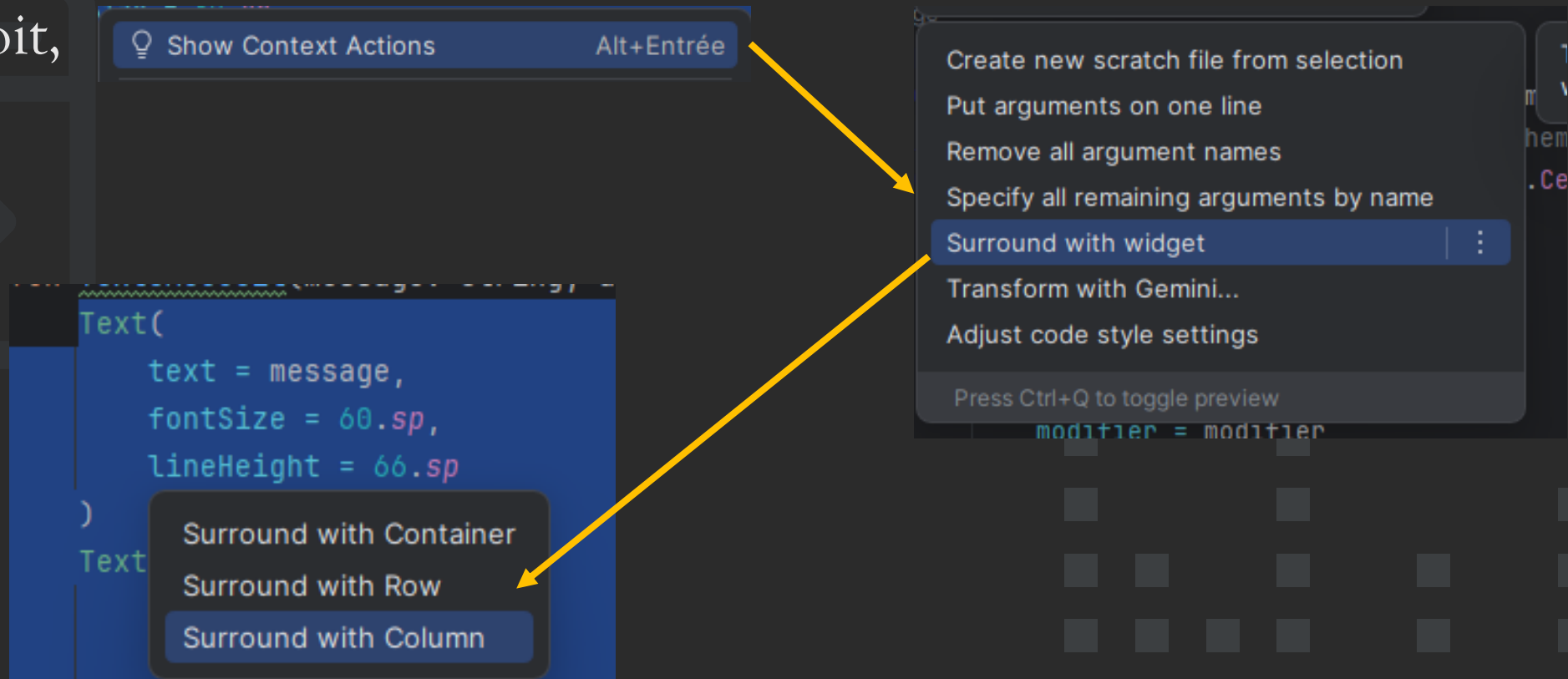
Disposer les éléments sur des lignes et des colonnes

- La hiérarchie de l'UI est basée sur des conteneurs. En d'autres termes, un composant peut **contenir un ou plusieurs autres composants**, et les termes "**parent**" et "**enfant**" sont parfois utilisés.
- Dans Compose, les trois éléments de mise en page standard les plus simples sont les composables **Column**, **Row** et **Box**.



Ajouter un conteneur

- Sélectionnez les 2 composables Text dans la fonction TexteAccueil, puis clic droit,



Résultat

- N'oubliez pas de transmettre le modifier au composant Column (l'enfant de TexteAccueil)

```
@Composable
fun TexteAccueil(message: String, auteur: String, modifier: Modifier = Modifier) {
    Column(modifier = modifier) {
        Text(
            text = message,
            fontSize = 60.sp,
            lineHeight = 66.sp
        )
        Text(
            text = auteur,
            fontSize = 30.sp
        )
    }
}
```



Ajout en production

- Une fois satisfait de l'aperçu ajoutez votre composant à votre application en appelant la fonction dans la fonction onCreate() dans le bloc Surface:

```
class MainActivity : ComponentActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        enableEdgeToEdge()  
        setContent {  
            HappyBirthdayTheme {  
                Surface(  
                    modifier = Modifier.fillMaxSize(),  
                    color = MaterialTheme.colorScheme.background  
                ) {  
                    TexteAccueil( message = "Joyeux Anniversaire Pierre!", auteur = "de Sophie")  
                }  
            }  
        }  
    }  
}
```


ASTUCE

- Pour éviter d'avoir à copier/coller du code entre la prévisualisation et la production, créez une fonction composable MonAppli qui contiendra vos composants UI.

```
@Composable
fun MonAppli(modifier: Modifier = Modifier) {
    Surface(
        modifier = Modifier.fillMaxSize(),
        color = MaterialTheme.colorScheme.background
    ) {
        TexteAccueil("Joyeux Anniversaire Pierre!", "de Sophie")
    }
}
```

- Ensuite appelez MonAppli dans Preview et OnCreate

ASTUCE

```
class MainActivity : ComponentActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        enableEdgeToEdge()  
        setContent {  
            HappyBirthdayTheme {  
                MonAppli()  
            }  
        }  
    }  
}  
  
//...  
  
@Preview(  
    showBackground = true,  
    showSystemUi = true,  
    name = "Happy Birthday"  
)  
@Composable  
fun GreetingPreview() {  
    HappyBirthdayTheme {  
        MonAppli()  
    }  
}
```

Derniers ajustements

- Centrons le texte verticalement en ajoutant un paramètre `verticalArrangement = Arrangement.Center` au conteneur `Column`
- Ajoutons-lui aussi une marge intérieure de `8.dp`

```
Column(  
    verticalArrangement = Arrangement.Center,  
    modifier = modifier.padding(8.dp)  
)
```

- Alignons maintenant le texte du message au centre avec le paramètre `textAlign = TextAlign.Center`

```
Text(  
    text = message,  
    fontSize = 60.sp,  
    lineHeight = 66.sp,  
    textAlign = TextAlign.Center  
)
```

Derniers ajustements

- Ajoutons un alignement à droite et une marge intérieure à la signature.

```
Text(  
    text = auteur,  
    fontSize = 30.sp,  
    modifier = Modifier  
        .padding(16.dp)  
        .align(alignment = Alignment.End)  
)
```

- Compilez et exécutez votre application pour voir le résultat final!

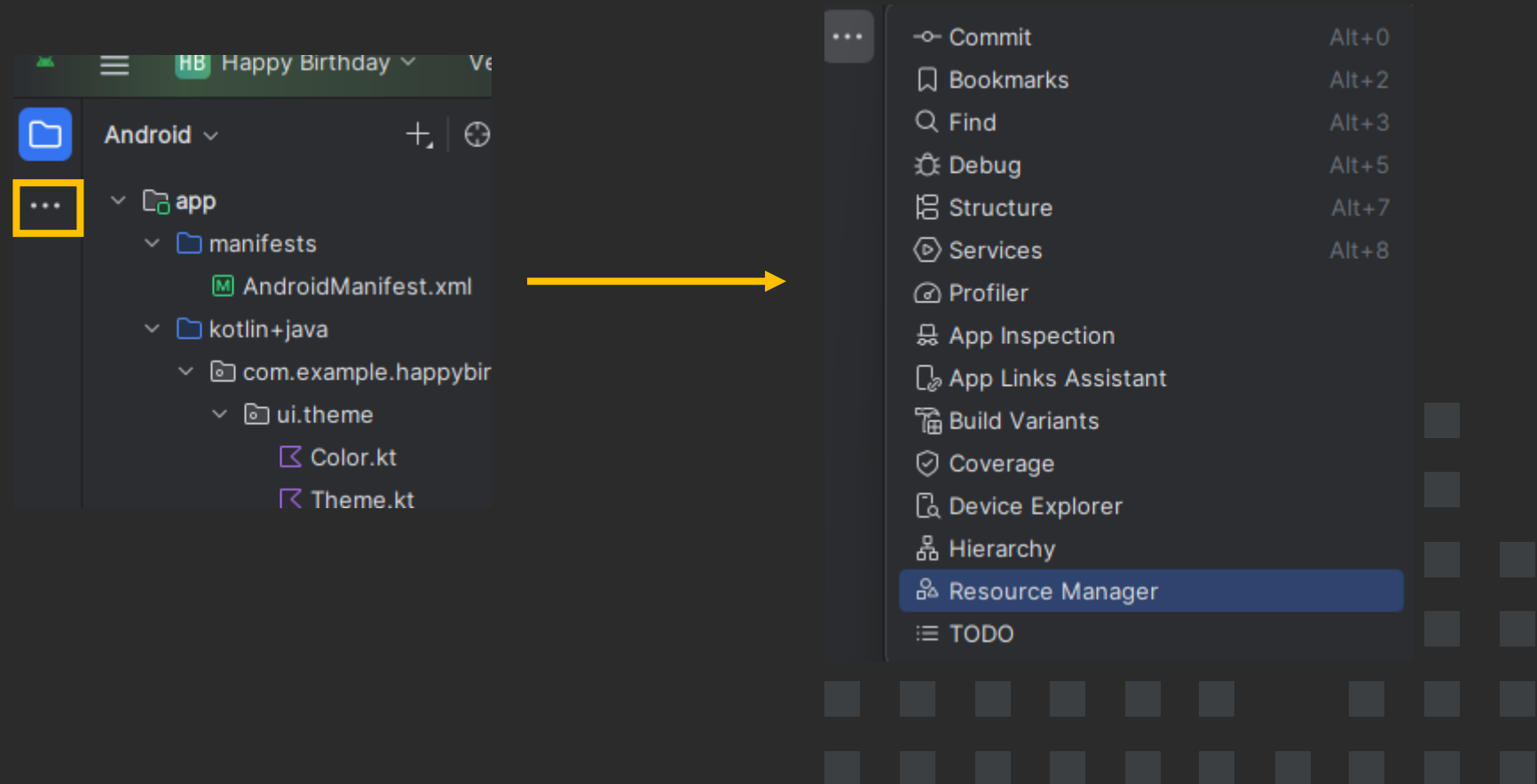


Ajout d'image



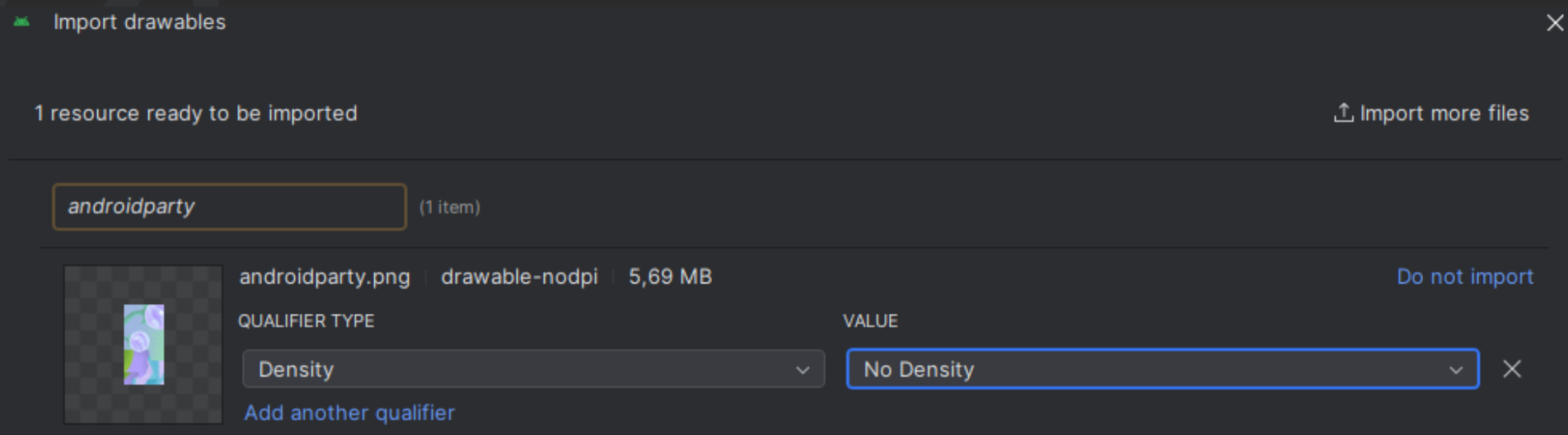
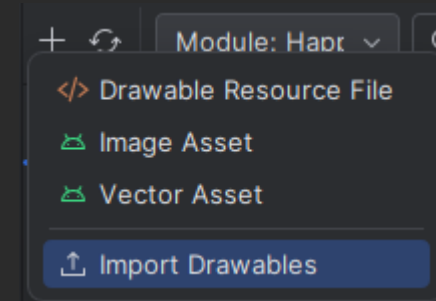
Gestionnaire de ressources

- Pour afficher le gestionnaire de ressources, on clique sur les ... dans la marge à gauche



Importation d'image

- Cliquer sur le +, puis Import Drawables
- Sélectionner l'image puis modifier le qualifieur type et la valeur:



Accéder aux ressources

- Les ressources de l'application sont accessibles via la classe générée R qui contient tous les ID de ressource.
- On y accède de la façon suivante:
`R.drawable.resID`
- Créez une fonction composable `ImageAccueil` avec les mêmes paramètres que `TexteAccueil`, déclarez une variable image puis chargez la ressource `androidparty` à l'aide de la fonction `painterResource()`
- Ajoutez un composable `Image` et attribuez la valeur image au paramètre `painter`
- Prévisualisez en appelant la fonction `ImageAccueil()`

Solution

- Vous obtenez un code ressemblant à ça:

```
@Composable
fun ImageAccueil(message: String, auteur: String, modifier: Modifier = Modifier) {
    val image = painterResource(R.drawable.androidparty)
    Image(
        painter = image,
        contentDescription = null
    )
}
```



Conteneur Box

- Pour combiner notre texte et notre image nous allons utiliser le conteneur Box
- Ajouter un conteneur Box autour de l'image, n'oubliez pas le paramètre modifier.
- Appeler la fonction TexteAccueil dans le Box, transmettez-lui les paramètres comme indiqué ci-dessous:

```
TexteAccueil(  
    message = message, auteur = auteur, modifier = Modifier.fillMaxSize().padding(8.dp)  
)
```

Modification de l'image

- On va adapter l'image à la taille de l'écran:
 - Ajoutons le paramètre `contentScale = ContentScale.Crop`
- Modifions l'opacité de l'image en modifiant son `alpha` à 0,5F:

```
Image(  
    painter = image,  
    contentDescription = null,  
    contentScale = ContentScale.Crop,  
    alpha = 0.5F  
)
```

- L'image occupe maintenant tout l'écran et est plus pâle.