Chapitre 5
Création d'interfaces simples

1. Les vues

La création d'une interface sous Android peut s'effectuer de deux manières :

– **La création statique**, qui s'effectue en XML.
– **La création dynamique**, qui s'effectue en Java.

Remarque

On peut combiner ces deux méthodes pour créer des interfaces plus complexes (cf. chapitre Création d'interfaces avancées - Interfaces dynamiques).

Une interface se compose :

– **D'un ou plusieurs fichiers XML** : ils représentent la partie statique d'une interface. Elle est constituée de différents éléments (bouton, texte, zone d'édition, etc.).
– **D'un fichier JAVA (Activité)** : il représente la partie dynamique d'une interface, les interactions utilisateur et les traitements à effectuer, etc.

Remarque

*Tous les éléments basiques d'une vue (bouton, zone de texte...) héritent de la classe **View**.*

Modifier une vue peut s'effectuer de deux manières :

– Mettre à jour le code XML de l'interface (onglet **Text** sous Android Studio).
– Mettre à jour la vue à l'aide de l'éditeur d'interface (onglet **Design** sous Android Studio).

1.1 Déclarer des identifiants

Un identifiant correspond à un nom unique affecté à un élément d'une vue. Grâce à cet identifiant, vous pouvez mettre en place les interactions et les traitements pour l'élément possédant cet identifiant.

Pour associer un identifiant à un élément d'une vue, il faut utiliser l'attribut suivant :

```
android:id="@+id/nom_identifiant"
```

La déclaration d'un identifiant se compose de plusieurs éléments :

– **android:id** : nom de l'attribut.
– **@+** : indique la déclaration d'un nouvel identifiant.
– **id** : correspond à la catégorie « l'identifiant ».
– **nom_identifiant** : correspond à l'identifiant d'un élément.

La syntaxe suivante permet d'accéder à l'identifiant d'un élément depuis un fichier Java :

```
R.id.nom_identifiant
```

ou depuis un fichier XML :

```
@id/nom_identifiant
```

© Editions ENI - All rights reserved

1.2 Spécifier la taille des éléments

À chaque déclaration d'un élément d'une vue (conteneur ou composant), vous devez spécifier sa hauteur et sa largeur (**android:layout_height** et **android:layout_width**).

Vous pouvez spécifier ces valeurs de plusieurs manières :

– **match_parent** : signifie que la taille de l'élément est égale à celle de l'élément parent.

  Par exemple, un bouton possédant une largeur définie à match_parent occupera le même espace que son conteneur.

– **wrap_content** : signifie que la taille de l'élément est égale à celle de son contenu.

  Par exemple, un bouton possédant une largeur définie à wrap_content aura pour taille la somme de la taille de son contenu et des différents espacements internes (padding).

– **en spécifiant une valeur** : vous pouvez définir la taille d'un élément à l'aide de valeurs fixes.

Remarque

Il faut spécifier la taille des éléments en dp (density-independent pixels) et non en px. Les tailles spécifiées en dp conservent les mêmes proportions quelle que soit la densité de l'écran.

1.3 Combiner avec les activités

Une fois la partie statique (xml) d'une interface déclarée, il faut créer une classe Java représentant votre activité.

Remarque

Chaque nouvelle activité créée doit être déclarée dans le manifeste de l'application.

Cette classe doit :

– Hériter de la classe **AppCompatActivity**.
– Surcharger au minimum la méthode **onCreate** (cf. chapitre Principes de programmation - Cycle de vie d'une activité).
– Lier l'activité à l'interface à l'aide de la méthode **setContentView**.

▶ Pour créer une nouvelle activité, faites un clic droit sur le dossier **src** de votre projet, puis sélectionnez l'option **New - Activity** et choisissez le type d'activité que vous voulez créer (**Blank Activity**, **Login Activity**, etc.).

Prenons l'exemple d'une interface créée dans le fichier **home.xml**. Pour pouvoir la lier à une activité, la méthode **onCreate** doit contenir au minimum le code ci-dessous :

```
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.home);
}
```

Remarque

Vous pouvez remarquer l'utilisation du fichier R.java pour récupérer le layout voulu.

▶ N'oubliez pas de déclarer votre activité dans le fichier manifeste de votre application. La déclaration de nouveaux composants (activité, service...) s'effectue entre les balises <**application**>.

```
<activity android:name="chemin.package.MyNewActivity"
android:label="@string/activity_title">
```

▶ Vous pouvez lui spécifier des propriétés ou comportements à l'aide des filtres d'intention. Les filtres d'intention se divisent en plusieurs catégories :
  – **Les actions** (balise **action**) : permettent de spécifier des actions (comportements) à un composant, par exemple : **ACTION_CALL** (pour passer un appel téléphonique), **ACTION_MAIN** (activité principale de l'application), **ACTION_SEND** (utilisé pour le partage de données), etc.
  – **Les données** (balise **data**) : permettent de spécifier le type de données traité par le composant.

© Editions ENI - All rights reserved

- **Les catégories** (balise **category**) : permet de spécifier la catégorie du composant, par exemple **CATEGORY_BROWSABLE** (peut être invoqué par le navigateur pour afficher des données), **CATEGORY_LAUNCHER** (l'activité sera disponible depuis le lanceur d'application), etc.
- **Les extras** : représente des données additionnelles qui seront fournies à l'activité. Par exemple, pour l'envoi d'un e-mail, utiliser la clé **EXTRA_EMAIL** pour spécifier le destinataire du mail.
- Différents flags utiles à l'activité.

Par exemple :

```
<activity android:name=".MyActivity"
          android:label="@string/activity_title">

<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>

</activity>
```

2. Les layouts

Les layouts facilitent l'organisation des différents éléments qui composent une interface. Ils servent de conteneur aux composantes d'une vue. Tous les layouts Android héritent de la classe **ViewGroup**.

Remarque

*La classe **ViewGroup** hérite de la classe **View**.*

2.1 FrameLayout

Le Framelayout est le conteneur le plus simple, il représente un espace qui affiche l'objet de votre choix.

Un élément ajouté à un FrameLayout se positionne en haut à gauche du layout. Vous pouvez changer cette position à l'aide de l'attribut **android:gravity**.

Vous avez la possibilité d'ajouter plusieurs éléments dans un même Framelayout et de modifier la visibilité de ces éléments pour afficher ou cacher plusieurs éléments au même emplacement.

2.2 LinearLayout

Le LinearLayout permet d'aligner des éléments (dans l'ordre des déclarations) dans une direction (verticale ou horizontale).

Vous pouvez définir les attributs suivants :

– L'orientation du layout.
– La gravité des éléments.
– Le poids des éléments.

Orientation

À la création d'un LinearLayout, vous devez préciser son orientation (horizontale ou verticale) à l'aide de l'attribut **android:orientation**.

Remarque

L'orientation possède par défaut la valeur horizontale.

Positionnement d'un élément

Pour définir le positionnement d'un élément dans un LinearLayout, deux attributs sont disponibles :

– **layout_gravity** : spécifie le positionnement d'un élément dans son conteneur.
– **gravity** : spécifie le positionnement du contenu d'un élément (par exemple, on peut spécifier la position d'un texte dans un bouton).

© Editions ENI - All rights reserved