Polycopié de cours pour le module Applications mobiles

Chapitre IV

interface utilisateur et navigation

La navigation désigne les interactions qui permettent aux utilisateurs de naviguer à travers, dans et hors les différents éléments de contenu de l' application. L'API android expose une multitude de composants et de modules permettant l'implémentation d'une navigation cohérente avec l'expérience utilisateur. En particulier, l'API suggère l'emploi du composant navigation `NavHostFragment` utilisé en conjonction avec les éléments : `NavGraph` , `NavController` et `NavDestination` . Aussi, l'ensemble des widgets et modules incluant la Toolbar, les menus, les dialogues et les toasts est impliqué couramment dans une implémentation de navigation. Dans ce qui suit sont discutés au préalable cet ensemble de widgets et modules. L'utilisation du composant navigation `NavHostFragment` est illustrée via un exemple d'implémentation.

IV.1 Utilisation d'une App bar

L'application bar (ou barre d'application) contribue à la convivialité de l'application en assurant :

- Un espace dédié pour donner une identité à votre application et indiquer l'emplacement de l'utilisateur dans l'application.
- Lancement des actions considérées comme plus importantes de manière prévisible ou directe, telle que la recherche.
- Support pour la navigation et le changement des vues (avec des onglets ou des listes déroulantes).

OOn doit utiliser la classe Toolbar (issue de la librairie) pour implémenter les barres d'application. Dans le manifeste de l'application, définir l'élément <application> pour qu'il utilise l'un des thèmes NoActionBar de appcompat. L'utilisation de l'un de ces thèmes empêche l'application d'utiliser la classe ActionBar native pour fournir la barre d'applications.

```
<application
    android:theme="@style/Theme.AppCompat.Light.NoActionBar"
    />
```

Ajouter un Toolbar dans le layout

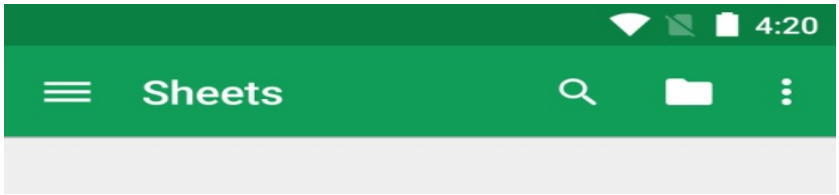

Figure IV.1: "Barre d'application "

```
<android.support.v7.widget.Toolbar
    android:id="@+id/my_toolbar"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:background="?attr/colorPrimary"
    android:elevation="4dp"
    android:theme="@style/ThemeOverlay.AppCompat.ActionBar"
    app:popupTheme="@style/ThemeOverlay.AppCompat.Light"/>
```

Dans la callback méthode onCreate() de l'activité, faire appel à la méthode `setSupportActionBar().`

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.my_activity);
    Toolbar tool_br = (Toolbar)findViewById(R.id.toolbr_id);
    setSupportActionBar(tool_br);
}
```

Pour utiliser les méthodes de `ActionBar` , faire appel à la méthode `getSupportActionBar().` Par exemple `ActionBar.hide().`

La barre d'applications vous permet d'ajouter des boutons pour les actions de l'utilisateur. Cette fonctionnalité vous permet de placer les actions les plus importantes pour le contexte actuel en haut de l'application. Par exemple, une application de navigation dans les photos peut afficher les partages et créer des boutons d'album en haut lorsque l'utilisateur regarde son rouleau de photos.

L'espace disponible dans la barre d'applications est limité. Si une application déclare plus d'actions que prévues, les actions en excès sont poussé sur l' overflow menu. L'application peut également spécifier qu'une action doit toujours être affichée dans le menu de débordement au lieu d'être affichée dans la barre d'applications.

- **exemple**

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <!-- should appear as action button if possible -->
    <item android:id="@+id/action_favorite"
        android:icon="@drawable/ic_favorite_black_48dp"
        android:title="@string/action_favorite"
        app:showAsAction="ifRoom"/>

    <item android:id="@+id/action_favorite"
        android:icon="@drawable/ic_favorite_black_48dp"
        android:title="@string/action_favorite"
        app:showAsAction="ifRoom"/>

</menu>
```

Lorsque l'utilisateur sélectionne l'un des éléments de la barre d'application, le système appelle la méthode de rappel `onOptionsItemSelected()` de l'activité et transmet un objet `MenuItem` pour indiquer l'élément sur lequel l'utilisateur a cliqué. Dans cette implémentation de `onOptionsItemSelected()` , on appelle la méthode `MenuItem.getItemId()` pour déterminer quel élément a été utilisé.

```
@Override
public boolean onOptionsItemSelected(MenuItem item){

    switch (item.getItemId()) {
        case R.id.action_settings :
            // User chose the "Settings" item,
            // show the app settings UI...
            return true;

        case R.id.action_favorites :
            // User chose the "favorites" action,
            return true;

        default :
            // the user's action was not recognized
            // Invoke the superclass to handle it.
            super.onOptionsItemSelected(item);
     }

}
```

IV.1.1 Définir une action Up

L'application devrait permettre aux utilisateurs de retrouver facilement leur écran principal. Un moyen simple de le faire est de fournir un bouton "Haut" dans la barre d'applications pour toutes les activités, à l'exception de la principale. Lorsque l'utilisateur sélectionne le bouton "Haut" , l'application accède à l'activité parent.

```
<application ...  >
    ...

    <!-- The main/home activity (has no parent activity) -->
    <activity
        android:name="com.exmpl.frstapp.MainActvty" ...>
        ...  ...
    </activity>

    <!-- A child of the main activity -->
    <activity
        android:name="com.exmpl.frstapp.ChildActivity"
        android:label="@string/title_activity_child"
        android:parentActivityName="com.exmpl.frstapp.MainActvty">
        ...  ...
    </activity>

</application>
```

IV.1.2 Activer l'action up

Pour activer le bouton Vers le haut d'une activité ayant une activité parent, faire appel à la méthode **`setDisplayHomeAsUpEnabled()`** de la barre d'applications.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.chld_activity);

     // child toolbar is defined in the layout file
     Toolbar ChldTlbar = (Toolbar)findViewById(R.id.chld_tlbar_id);
     setSupportActionBar(myChildToolbar);
     // Get a support ActionBar corresponding to this toolbar
     ActionBar  a_b = getSupportActionBar();
     //Enable the Up button
     a_b.setDisplayHomeAsUpEnabled(true);

}
```

IV.2 Les Menus

Les types :

- Les options menus : C'est l'endroit où sont placées des actions ayant un impact global sur l'application, telles que "Rechercher", "email" et "Paramètres". Sont Associés souvent à une App bar.
- Les menus contextuels : Un menu contextuel est un menu flottant qui apparaît lorsque l'utilisateur clique longuement sur un élément. Le mode d'action contextuelle affiche les éléments d'action qui affectent le contenu sélectionné dans une barre en haut de l'écran et permet à l'utilisateur de sélectionner un élément parmi plusieurs.
- Les popup menus : Un menu contextuel affiche une liste verticale d'éléments ancrée à la vue qui a appelé le menu. C'est utile pour fournir un débordement d'actions liées à un contenu spécifique ou pour fournir des options pour une deuxième partie d'une commande.

IV.2.1 Ajouter un menu option

Pour permettre un accès rapide aux actions importantes, on peut faire apparaitre les éléments correspondants à ces actions dans la barre des applications en ajoutant l'attribue `android:showAsAction = "ifRoom"` au tag <menu>.

Il est possible de déclarer des éléments pour le menu d'options à partir de la sous-classe d'activité ou de la sous-classe de fragment. Si l'activité et les fragments déclarent des éléments pour le menu d'options, ils sont combinés dans l'interface utilisateur. Les éléments de l'activité apparaissent en premier, suivis de ceux de chaque fragment dans l'ordre dans lequel chaque fragment est ajouté à l'activité. Si nécessaire, vous pouvez réorganiser les éléments de menu avec l'attribut `android:orderInCategory` .

Pour spécifier le menu d'options d'une activité, faire @override de `onCreateOptionsMenu()`, (les fragments fournissent leur propre rappel `onCreateOptionsMenu()`). Dans cette méthode, vous pouvez inflater la ressource menu (définie en XML).

```
@Override
public void onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.game_menu, menu);
    return true;
}
```

Il est possible également d'ajouter des éléments (items) de menu à l'aide de **add()** ou bien récupérer d'aitres avec **findItem()** pour éventuellement réviser leurs propriétés et ce moyennant les API **MenuItem** .

IV.2.2 Définir des actions pour les sélections

Lorsque l'utilisateur sélectionne un élément dans le menu des options (y compris les actions dans la barre d'applications), le système appelle la méthode **onOptionsItemSelected()** de l'activité.

```
@Override
public boolean onOptionsItemSelected(MenuItem item){
    // Handle item selection
    switch (item.getItemId()) {
        caseR.id.new_game :
            newGame();
            return true;

        caseR.id.help :
            showHelp();
            return true;

        default :
            super.onOptionsItemSelected(item);
     }
}
```

IV.2.3 Le MenuItem search et l'interface search

En portant un MenuItem search dans le menu de l'application résulte en une implémentation de l'interface Search.

L'interface search :

il est possible d'implémenter une interface `Search` comme un `Dialog` ou autant que widget comme une instance `SearchView.`
A part une légère différence Les deux variantes ont presque les mêmes propriétés .
Utilisant un `Dialog` l'interface s'affiche au top de l'activité.
Alors qu'en implémentant le Search comme widget on peut choisir l'endroit de l'affichage dans le layout de l'activité au même titre que les autres widgets.
Dans les deux cas, le système crée un intent et enregistre dans cet intent la requête de recherche et lance l'activité éligible qui est normalement due être déjà déclarée `SEARCHABLE.`

Pour pouvoir implémenter cette forme de Search doivent être réuni les éléments suivants.

- Une configuration dite searchable : un fichier xml pour configurer les propriétés désirées.
- L'activité Searchable : c'est l'activité cible au quelle l'intent est émis.
- L'interface Search comme Dialog ou comme widget.

IV.2.4 Changer les éléments du menu durant l'exécution

Une fois que le système a appelé `onCreateOptionsMenu()` , il conserve une instance du menu et n'appelle plus `onCreateOptionsMenu()` à moins que le menu ne soit invalidé pour une raison quelconque. Toutefois, on ne doit utiliser `onCreateOptionsMenu()` que pour créer l'état de menu initial et ne pas apporter de modifications au cours du cycle de vie de l'activité.
Si on souhaite modifier le menu d'options en fonction des événements qui se produisent pendant le cycle de vie de l'activité, on peut le faire dans la méthode `onPrepareOptionsMenu()` . Cette méthode transmet l'objet Menu tel qu'il existe actuellement afin que vous puissiez le modifier, tel que l'ajout, la suppression ou la désactivation d'éléments. (Les fragments fournissent également un callback `onPrepareOptionsMenu()`).

Sur Android 3.0 et les versions ultérieures, le menu des options est considéré comme toujours ouvert lorsque des éléments de menu sont présentés dans la barre des applications. Lorsqu'un événement se produit et qu'on souhaite effectuer une mise à jour de menu, on doit appeler `invalidateOptionsMenu()` pour demander l'appel système `onPrepareOptionsMenu()` .