

Interfaces avancées

Chantal Keller

Interfaces, jusqu'à présent

Interfaces "figées"

- pas d'apparition dynamique de widgets
- pas de paramétrisation de l'utilisateur ou l'utilisatrice

Pas de factorisation de code interface

- si interfaces similaires \Rightarrow copier-coller le XML

Plan

- 1 Fragments
- 2 Utilisation des fragments
- 3 Préférences
- 4 Persistence longue
- 5 Conclusion

Fragment : portion modulaire d'une activité

Possède :

- sa propre interface
- ses propres données
- son propre cycle de vie

On peut :

- combiner des fragments au sein d'une même activité
- utiliser le même fragment dans plusieurs activités
- ajouter/retirer des fragments lorsqu'une activité tourne

Principe

Flexibilité de l'interface :

- combiner et échanger des morceaux d'interface
- l'activité va gérer ses différents fragments
- très important pour gérer plusieurs tailles d'écran

Bonnes pratiques :

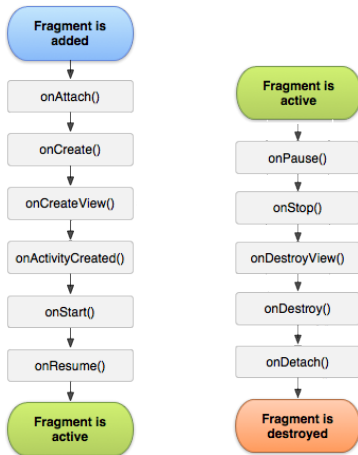
- chaque morceau d'interface doit être décrit par un fragment
- les fragments doivent être modulaires et réutilisables

Programmation d'un fragment

Programmation similaire à une activité :

- vue en XML
- contrôleur en Java, dans une classe héritant cette fois de `Fragment`

La classe Fragment



©Android

↪ comme d'habitude, redéfinir ces méthodes pour contrôler le déroulement du fragment

La méthode onCreateView

```
public static class DetailsFragment extends Fragment {  
    @Override  
    public View onCreateView(LayoutInflater inflater,  
                             ViewGroup container,  
                             Bundle savedInstanceState) {  
        ...  
        // Initialisation de l'interface  
        View v = inflater.inflate(R.layout.fragment_details,  
                                   container,  
                                   false);  
        ...  
        // Récupération des objets de l'interface, écouteurs...  
        detailsTextView = v.findViewById(R.id.details);  
        ...  
        return v;  
    }  
}
```


La méthode onCreateView

```
public static class DetailsFragment extends Fragment {  
    @Override  
    public View onCreateView(LayoutInflater inflater,  
                             ViewGroup container,  
                             Bundle savedInstanceState) {  
        ...  
        // Initialisation de l'interface  
        View v = inflater.inflate(R.layout.fragment_details,  
                                  container, ← fichier XML  
                                  false);  
        ...  
        // Récupération des objets de l'interface, écouteurs...  
        detailsTextView = v.findViewById(R.id.details);  
        ...  
        return v;  
    }  
}
```

La méthode onCreateView

```
public static class DetailsFragment extends Fragment {  
    @Override  
    public View onCreateView(LayoutInflater inflater,  
                             ViewGroup container,  
                             Bundle savedInstanceState) {  
        ...  
        // Initialisation de l'interface  
        View v = inflater.inflate(R.layout.fragment_details,  
                                   container,  
                                   false);  
        ...  
        // Récupération des objets de l'interface, écouteurs...  
        detailsTextView = v.findViewById(R.id.details);  
        ...  
        return v;  
    }  
}
```

bundle → `Bundle savedInstanceState`

fichier XML → `R.layout.fragment_details`

La méthode onCreateView

```
public static class DetailsFragment extends Fragment {  
    @Override  
    public View onCreateView(LayoutInflater inflater,  
                             ViewGroup container,  
                             Bundle savedInstanceState) {  
        ...  
        // Initialisation de l'interface  
        View v = inflater.inflate(R.layout.fragment_details,  
                                  container,  
                                  false);  
        ...  
        // Récupération des objets de l'interface, écouteurs...  
        detailsTextView = v.findViewById(R.id.details);  
        ...  
        return v;  
    }  
}
```

vue créée →

bundle →

fichier XML →

Plan

- 1 Fragments
- 2 Utilisation des fragments
- 3 Préférences
- 4 Persistence longue
- 5 Conclusion

Affichage non dynamique

Dans l'interface XML de l'activité :

```
<...ConstraintLayout ...>
    ...

    <fragment
        android:name="com.example.DetailsFragment"
        android:id="@+id/fragment"
        android:layout_width="..."
        android:layout_height="..."
        app:layout_constraintStart_toStartOf="..."
        ... />

    ...
</...ConstraintLayout>
```

↪ impossible de détacher ou de remplacer le fragment

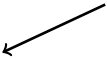
Affichage non dynamique

Dans l'interface XML de l'activité :

```
<...ConstraintLayout ...>  
...
```

nom de la classe
dans son package

```
<fragment  
    android:name="com.example.DetailsFragment"  
    android:id="@+id/fragment"  
    android:layout_width="..."  
    android:layout_height="..."  
    app:layout_constraintStart_toStartOf="..."  
    ... />
```



```
...  
</...ConstraintLayout>
```

↪ impossible de détacher ou de remplacer le fragment

Affichage dynamique

Dans l'interface XML de l'activité (on réserve de la place) :

```
<FrameLayout
    android:id="@+id/fragment"
    android:layout_width="..."
    android:layout_height="..."
    app:layout_constraintStart_toStartOf="..."
    ... />
```

Dans le code Java de l'activité, à l'endroit souhaité :

```
DetailsFragment frag = new DetailsFragment();
FragmentManager transaction = getSupportFragmentManager().beginTransaction();
transaction.add(R.id.fragment, frag);
transaction.commit();
```

Gestionnaire de fragments et transactions

3 étapes :

■ Début :

■ dans une activité :

```
FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();
```

■ dans un fragment :

```
FragmentTransaction transaction = getParentFragmentManager().beginTransaction();
```

■ Opérations (autant que l'on souhaite) :

■ attachement : `transaction.add(R.id.fragment, newFrag);`

■ détachement : `transaction.remove(oldFrag);`

■ remplacement : `transaction.replace(R.id.fragment, newFrag);`

■ Fin (ici que ça prend effet) : `transaction.commit();`

Le gestionnaire de fragments

La classe `FragmentManager` permet notamment de :

- créer des transactions : méthode `beginTransaction`
- trouver un fragment attaché : méthode `findFragmentById`

Remarques :

- l'appel à `getSupportFragmentManager()` ou `getParentFragmentManager()` peut être fait une fois pour toutes, et le résultat mis dans une variable
- en revanche, on crée une transaction à chaque fois qu'on veut effectuer des modifications
- possible de faire des animations

Interaction : le fragment veut accéder aux objets de l'activité

La méthode `getActivity`

- méthode de la classe `Fragment`
- renvoie l'activité à laquelle le fragment est **attaché**, de type `Activity`
- on a ensuite accès à toutes les méthodes de la classe `Activity`, notamment `findViewById`
- si on veut avoir accès aux méthodes spécifiques de l'activité :
caster du type `Activity` vers le type de la classe
ex : `MCCActivity ma = (MCCActivity) getActivity();`

Interaction : l'activité veut accéder aux objets du fragment

Aussi en appelant des méthodes :

- l'activité connaît le fragment frag (ou peut le retrouver à l'aide de `findFragmentById`)
- elle peut appeler les méthodes qu'elle souhaite dessus
- ex :

- dans l'activité : `frag.changerAffichage("Bonjour !")`
- dans le fragment :

```
public void changerAffichage(String s) {  
    affichage.setText(s);  
}
```

Plan

- 1 Fragments
- 2 Utilisation des fragments
- 3 Préférences
- 4 Persistence longue
- 5 Conclusion

But des préférences

Permettre à l'utilisateur ou l'utilisatrice :

- de personnaliser l'utilisation de l'appareil ou de l'application
- une fois pour toutes
- de fournir un certain nombre de données (ex : login, clé du wifi, ...)

de manière **uniforme** entre applications

Deux types de préférences

Les préférences système :

- s'appliquent à toutes les applications
- transparent pour le programmeur
- mais faire attention à ne pas les “contourner” (ex. taille du texte)

Les préférences d'une application :

- s'appliquent uniquement à l'application
- à la charge du programmeur

Deux types de préférences

Les préférences système :

- s'appliquent à toutes les applications
- transparent pour le programmeur
- mais faire attention à ne pas les “contourner” (ex. taille du texte)

Les préférences d'une application :

- s'appliquent uniquement à l'application
- à la charge du programmeur

La classe PreferenceFragmentCompat

Le SDK Android propose une classe prédéfinie :

- facile à mettre en œuvre ; s'utilise comme un fragment
- interface similaire entre toutes les applications d'un même appareil
- conservation des données
- signalement lors d'une modification
- possibilité de mettre des valeurs par défaut

Rappel

On distingue :

- la **valeur** des préférences, sauvegardées dans la mémoire de l'appareil
- l'**application** des préférences, par ex. pour réaliser l'affichage selon le choix de l'utilisateur ou l'utilisatrice

Sauvegarder les préférences ne les applique pas magiquement !

- sauvegarde : gérée automatiquement par la classe `PreferenceFragmentCompat`
- application : travail du programmeur

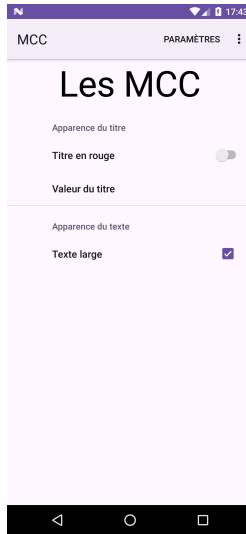
Principe

Le cahier des charges indique les éléments qui doivent être paramétrés, et les choix possibles.

Travail du programmeur :

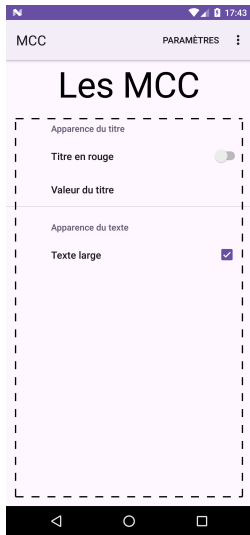
- 1 décrire l'interface du menu de paramètres
- 2 permettre d'y accéder
- 3 appliquer les paramètres sauvegardés à chaque lancement
- 4 [réagir lors d'une modification]
- 5 [restaurer les valeurs par défaut]

1. Interface



1. Interface

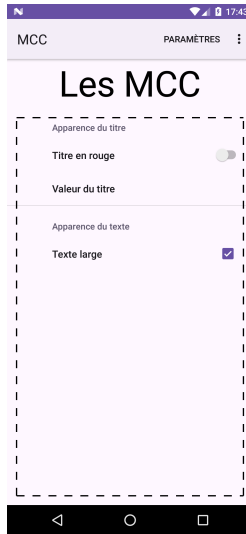
PreferenceScreen



1. Interface

PreferenceCategory

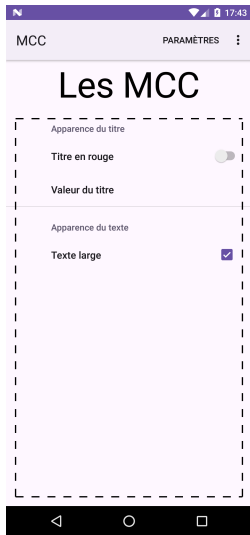
PreferenceScreen



1. Interface

PreferenceCategory

PreferenceScreen

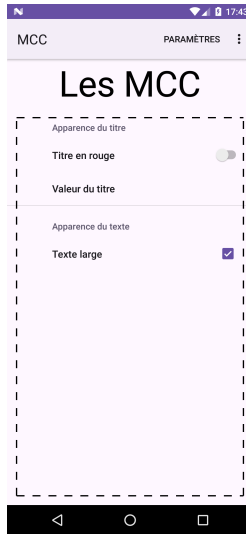


SwitchPreference

1. Interface

PreferenceCategory

PreferenceScreen



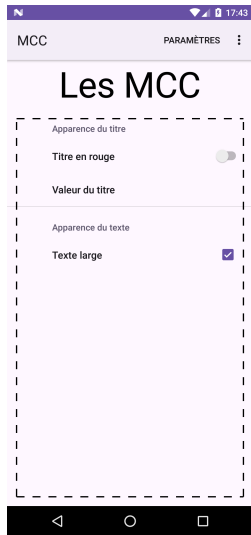
← SwitchPreference

← EditTextPreference

1. Interface

PreferenceCategory

PreferenceScreen

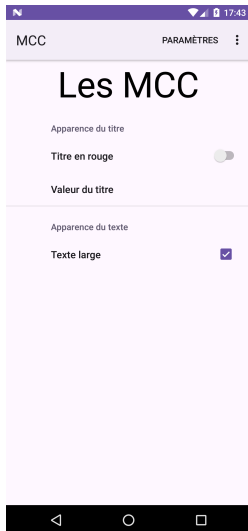


← SwitchPreference

← EditTextPreference

← CheckBoxPreference

Description en XML



```
<PreferenceScreen>

    <PreferenceCategory
        app:title="Apparence du titre">

        <SwitchPreference
            app:title="Titre en rouge"
            app:defaultValue="false"
            app:key="RED_TITLE"/>

        <EditTextPreference
            app:title="Valeur du titre"
            app:defaultValue="@string/mcc"
            app:key="TITLE_VALUE"/>
    </PreferenceCategory>

    <PreferenceCategory
        app:title="Apparence du texte">

        <CheckBoxPreference
            app:title="Texte large"
            app:defaultValue="true"
            app:key="LARGE_TEXT"/>
    </PreferenceCategory>

</PreferenceScreen>
```

Les trois attributs essentiels

```
<CheckBoxPreference
    android:title="Texte large"
    android:defaultValue="false"
    android:key="LARGE_TEXT"/>
```

- `title` : description de la préférence affichée à l'utilisateur ou l'utilisatrice
- `defaultValue` : valeur par défaut (utilisée au premier lancement et en cas de restauration des paramètres par défaut)
- `key` : identifiant unique de la préférence (stocke la valeur suivant un principe clé/valeur)

Remarques

⚠ Pas un fichier de ressource de type `layout` mais de type `xml`
(car pas les mêmes “boîtes”)

Si beaucoup de niveaux, possibilité d'imbriquer des
`PreferenceScreen` (⇒ ouvre un nouveau menu)

Programmation

```
public class SettingsFragment extends PreferenceFragmentCompat {  
  
    @Override  
    public void onCreatePreferences(Bundle savedInstanceState, String rootKey) {  
        setPreferencesFromResource(R.xml.root_preferences, rootKey);  
    }  
  
}
```



L'affichage est lancé dans onCreatePreference

2. Accès aux préférences

C'est un fragment : soit

- le mettre statiquement dans une activité et lancer cette activité
- l'attacher dynamiquement à la place du fragment courant

⚠ Les paramètres doivent être accessibles depuis **toutes** les activités de l'application (bouton ou menu)

Aparté : menu d'une activité

Description en XML (ressource de type menu) :



```
<menu>

    <item
        android:title="Paramètres"
        android:id="@+id/settings"
        android:orderInCategory="100"
        app:showAsAction="ifRoom" />

    <item
        android:title="Réinitialisation"
        android:id="@+id/init_settings"
        android:orderInCategory="200"
        app:showAsAction="never" />

</menu>
```

Aparté : menu d'une activité

Dans la classe de l'activité :

```
// Création du menu
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu, menu);
    return true;
}

// Écouteur sur le menu
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // L'item sur lequel l'utilisateur a cliqué
    int id = item.getItemId();

    // Action choisie selon l'item
    if (id == R.id.action_settings) {
        // Attachement du fragment de préférences
        getSupportFragmentManager().beginTransaction().addToBackStack("pref")
            .replace(R.id.frag, prefFrag).commit();

        return true;
    }

    if (id == R.id.init_settings) {
        initPref();
        return true;
    }

    return super.onOptionsItemSelected(item);
}
```

Aparté : menu d'une activité

Dans la classe de l'activité :

```
// Création du menu
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu, menu);
    return true;
}

// Écouteur sur le menu
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // L'item sur lequel l'utilisateur a cliqué
    int id = item.getItemId();


    // Action choisie selon l'item
    if (id == R.id.action_settings) {
        // Attachement du fragment de préférences
        getSupportFragmentManager().beginTransaction().addToBackStack("pref")
            .replace(R.id.frag, prefFrag).commit();

        return true;
    }

    if (id == R.id.init_settings) {
        initPref();
        return true;
    }

    return super.onOptionsItemSelected(item);
}
```

permet de quitter les préférences avec le bouton arrière



3. Au lancement, application des paramètres sauvegardés

Lorsque le fragment ou l'activité devient visible :

```
// Application des préférences lorsque le fragment devient visible
@Override
public void onResume() {
    super.onResume();
    // - récupérer les valeurs choisies par l'utilisateurice
    SharedPreferences sharedPref =
        PreferenceManager.getDefaultSharedPreferences(getActivity());
    boolean largeText = sharedPref.getBoolean("LARGE_TEXT", false);
    ...

    // - les appliquer
    text.setTextSize(largeText ? 40 : 20);
    ...
}
```


3. Au lancement, application des paramètres sauvegardés

Lorsque le fragment ou l'activité devient visible :

```
// Application des préférences lorsque le fragment devient visible
@Override
public void onResume() {
    super.onResume();
    // - récupérer les valeurs choisies par l'utilisateurice
    SharedPreferences sharedPref =
        PreferenceManager.getDefaultSharedPreferences(getActivity());
    boolean largeText = sharedPref.getBoolean("LARGE_TEXT", false);
    ...

    // - les appliquer
    text.setTextSize(largeText ? 40 : 20);
    ...
}
```

activité courante :
getActivity() si
dans un fragment,
this si dans une
activité




3. Au lancement, application des paramètres sauvegardés

Lorsque le fragment ou l'activité devient visible :

```
// Application des préférences lorsque le fragment devient visible
@Override
public void onResume() {
    super.onResume();
    // - récupérer les valeurs choisies par l'utilisateur
    SharedPreferences sharedPref =
        PreferenceManager.getDefaultSharedPreferences(getActivity());
    boolean largeText = sharedPref.getBoolean("LARGE_TEXT", false);
    ...

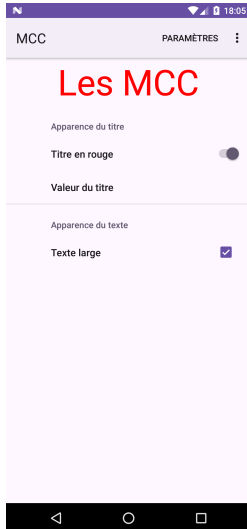
    // - les appliquer
    text.setTextSize(largeText ? 40 : 20);
    ...
}
```

activité courante :
getActivity() si
dans un fragment,
this si dans une
activité



⚠ À faire dans **tous** les fragments et activités dépendant des paramètres (factoriser le code !)

4. Réaction instantanée à une modification



Utile si modifications visibles
au moment où elles sont faites
(ici, utile pour le titre, inutile
pour le texte)

Principe habituel

- on associe un écouteur au fragment de préférences
- l'écouteur est appelé dès que l'utilisateur ou l'utilisatrice modifie l'une des préférences
- on peut alors savoir quelle préférence a été modifiée et quelle est sa nouvelle valeur pour réagir en conséquence

Écouteur sur le fragment de préférences

```
public class PrefFragment extends PreferenceFragmentCompat
    implements SharedPreferences.OnSharedPreferenceChangeListener {

    ...

    // Écouteur
    @Override
    public void onSharedPreferenceChanged(SharedPreferences sharedPreferences, String key) {
        // On applique tous les paramètres utiles :
        ((MCCActivity) getActivity()).applyPref();
    }

    // Lancement de l'écouteur
    @Override
    public void onResume() {
        super.onResume();
        getPreferenceScreen().getSharedPreferences()
            .registerOnSharedPreferenceChangeListener(this);
    }

    // Arrêt de l'écouteur
    @Override
    public void onPause() {
        getPreferenceScreen().getSharedPreferences()
            .unregisterOnSharedPreferenceChangeListener(this);
        super.onPause();
    }
}
```

Écouteur sur le fragment de préférences

```
public class PrefFragment extends PreferenceFragmentCompat
    implements SharedPreferences.OnSharedPreferenceChangeListener {

    ...

    // Écouteur
    @Override
    public void onSharedPreferenceChanged(SharedPreferences sharedPreferences, String key) {
        // On applique tous les paramètres utiles :
        ((MCCActivity) getActivity()).applyPref();
    }

    // Lancement de l'écouteur
    @Override
    public void onResume() {
        super.onResume();
        getPreferenceScreen().getSharedPreferences()
            .registerOnSharedPreferenceChangeListener(this);
    }

    // Arrêt de l'écouteur
    @Override
    public void onPause() {
        getPreferenceScreen().getSharedPreferences()
            .unregisterOnSharedPreferenceChangeListener(this);
        super.onPause();
    }
}
```

la clé du
paramètre
modifié



5. Restauration des valeurs par défaut


Rappel : définies dans le fichier XML

À l'endroit voulu :

```
// Réinitialisation de la sauvegarde
PreferenceManager.getDefaultSharedPreferences(this).edit().clear().commit();

// Lecture des valeurs par défaut
PreferenceManager.setDefaultValues(this, R.xml.root_preferences, true);

// Les appliquer ! (comme avant)
...
```

 Ne pas oublier de les appliquer ! (même technique que précédemment)

5. Restauration des valeurs par défaut

Rappel : définies dans le fichier XML

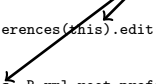
là encore,
l'activité
courante

À l'endroit voulu :

```
// Réinitialisation de la sauvegarde
PreferenceManager.getDefaultSharedPreferences(this).edit().clear().commit();

// Lecture des valeurs par défaut
PreferenceManager.setDefaultValues(this, R.xml.root_preferences, true);

// Les appliquer ! (comme avant)
...
```



⚠ Ne pas oublier de les appliquer ! (même technique que précédemment)

Plan

- 1 Fragments
- 2 Utilisation des fragments
- 3 Préférences
- 4 Persistance longue
- 5 Conclusion

Utilisation pour la persistance longue des données

Indépendemment de l'interface :

- on peut sauvegarder des couples clé/valeur :

```
// Récupération des préférences partagées
SharedPreferences.Editor editor =
    PreferenceManager.getDefaultSharedPreferences(this).edit();
// Données à sauvegarder (autant qu'on veut avec des clés différentes)
editor.putInt("SCORE", score);
// Sauvegarde
editor.commit();
```

- on peut les lire de même que précédemment
- approche préconisée pour la persistance de données simples (types de base), petites, et **non confidentielles**

Autres approches selon les besoins

| Moyen | Type de données | Taille des données | Lieu |
|---------------------|-------------------|--------------------|--------|
| Préférences | publiques | petites | local |
| Système de fichiers | privées | petites/grandes | local |
| Base de données | privées | grandes/organisées | local |
| Cloud | publiques/privées | petites/grandes | réseau |
| Serveur personnel | privées | petites/grandes | réseau |

Plan

- 1 Fragments
- 2 Utilisation des fragments
- 3 Préférences
- 4 Persistance longue
- 5 Conclusion

Conclusion

Fragments :

- factorisation de code qui peut être utilisé plusieurs fois (au sein d'une même activité ou non)
- dynamisme
- fragment de préférences