IUT - Departement Informatique

Android - 2014-2015

2eme séance - Cycle de vie

Android TD2

1 Découverte du cycle de vie et gestion des événements

Le but de cet exercice est de vous faire découvrir manuellement le cycle de vie d'une activité, et de comprendre le fonctionnement de la console `LogCat`. L'application développée dans cette section sera également utilisée dans la section suivante.

1. Créez un nouveau projet `CycleVie`
   (a) Lancez l'assistant de nouveau projet Android (New Project).
   (b) Nommez le projet `CycleVie`, puis faîtes Next jusqu'à la fin de l'assistant, et enfin Finish. Vous choisirez comme SDK minimum l'API 16 (Jelly Bean) et Blank Activity lorsque cela est proposé.
   (c) Vous disposez maintenant d'une application avec une activité de base.
2. Ajouter à la classe MyActivity une variable `TAG` de type `String`, valant "AppelCV" (CV=Cycle de Vie). Nous l'utiliserons pour identifier nos messages parmi les autres messages de log.
3. Redéfinissez les méthodes évènementielles afin de capturer les changements d'états de l'activité :
   (a) Code -> Override Methods
   (b) Sélectionnez les méthodes `onDestroy`, `onPause`, `onRestart`, `onRestoreInstanceState`, `onResume`, `onSaveInstanceState`, `onStart`, `onStop`.
4. Pour chacune des méthodes redéfinies, écrivez une trace dans le journal d'évènements (avant d'appeler la méthode parente) :
   (a) `Log.e(TAG, "onDestroy")` ; Remplacez `onDestroy` par le nom de la méthode en cours
5. Filtrage des évènements. La console LogCat affiche les évènements de différentes sources (identifiées par un tag). Il est possible de ne visualiser que ceux qui nous intéressent :
   (a) Après avoir lancé l'application, sélectionnez la perspective LogCat dans Android Studio.
   (b) Saisissez "AppelCV" dans le champs de recherche. De cette façon, seuls les évènements avec notre tag seront visibles
6. Analysez l'ordre d'appel des fonctions redéfinies en fonction des actions effectuées sur la machine :
   (a) Effectuez différentes actions (ouvrir l'application, la mettre en fond pour en lancer une autre, la quitter, changer l'orientation de l'écran, attendre l'apparition de l'écran de veille ...).
   (b) Vérifiez les évènements à partir des Logs affichés dans la console Logcat.
   (c) À partir de vos observations, retracez le cycle de vie d'une activité Android. Vous allez pouvoir le vérifier dans la section suivante.

2 Gérer le cycle de vie d'une application

Vous venez de découvrir que le cycle de vie d'une application Android est particulièrement complexe. Il faut comprendre et gérer ce cycle pour avoir des applications fonctionnelles qui ne perdent pas d'informations au fil des évènements. Dans cette section, vous allez apprendre à prendre en compte les changements d'état des activités.

1. **Création du projet**. Rendez-vous sur la page `http://developer.android.com/training/basics/activity-lifecycle/` et cliquez sur `Starting an Activity`. Le but de la leçon est d'expliquer comment fonctionne ce cycle de vie et le prendre en compte dans le développement des applications. Nous allons suivre cette leçon pas à pas en s'assurant que l'on a bien compris les concepts importants au fur et à mesure. Nous vous invitons donc à revenir à cette feuille de TD à chaque fois que vous atteignez un bas de la page du tutoriel en ligne afin de faire le point et appliquer ce qui a été vu dans l'application développée précédemment.
   ⇒ Une fois en bas de page, vous devez avoir pris connaissance des points suivants :

(a) Une activité peut évoluer suivant différents états principaux : `Resumed` (l'activité est au premier plan, l'utilisateur peut interagir avec) , `Paused` (l'activité est partiellement masquée par une autre activité et ne peut pas recevoir d'informations de l'utilisateur ou exécuter du code), `Stopped` (l'activité n'est pas visible par l'utilisateur). Il existe des états temporaire dans lesquels l'activité ne reste qu'une fraction de temps (`Created`, `Started`).

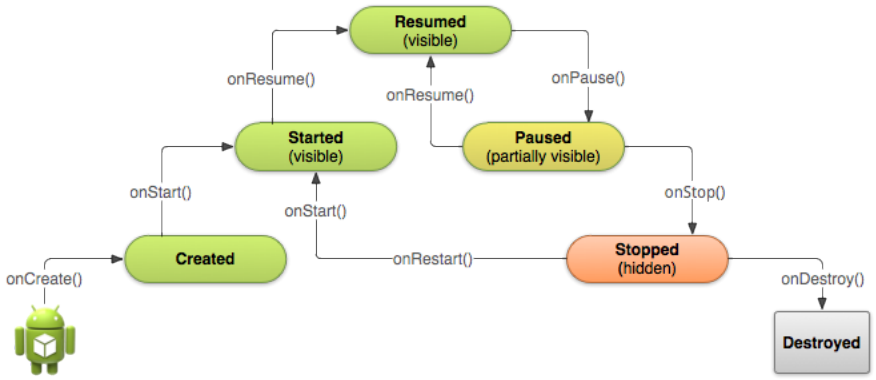
(b) L'activité principale (celle qui est executée au lancement) d'une application doit être définie dans une balise `<intent-filter>` en incluant l'action `MAIN` et la catégorie `LAUNCHER` :

```
  ...
<activity android:name=".MyActivity" android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
  ...
```

(c) La méthode `onCreate()` est appelée à la création de l'activité. Vous devez y mettre le code qui ne doit être exécuté qu'une fois dans l'application (création de l'interface, instantiation de variables, ...).

(d) La méthode `onDestroy()` est le dernier callback de l'activité avant qu'elle ne soit supprimée de la mémoire. Elle est appelée après `onPause`, puis `onStop`. Cependant, le système n'est pas obligé de l'appeler. Il n'y a donc aucune garantie qu'elle le sera !

(e) La chaîne d'événements/état est la suivante (vous pouvez comparer avec ce que vous avez obtenu en première section) :



2. **Modifiez l'application de la première section** pour ajouter un formulaire simple demandant le nom et le prénom (l'identifiant des deux champs texte doit être `nom` et `prenom`) [1]

   (a) Ouvrez le fichier `res/layout/activity_my.xml`

   (b) Modifiez le de façon à ajouter un formulaire demandant le nom et prénom d'une personne. Layout avec `LinearLayout`, orientation des layouts avec `android:orientation="horizontal"` et `android:orientation="vertical"`, utilisation de `Textview` et `EditText`.

   Rendez-vous à la page suivante de la leçon.

3. **Gestion des états** `Paused` **and** `Resumed`

   ⇒ Une fois en bas de page, vous devez avoir pris connaissance des points suivants :

   (a) Le système appelle la méthode `onPause()` de votre activité dès quelle n'est plus pleinement visible. La méthode est également appelée avant que l'utilisateur ne quitte l'activité.

   (b) Il faut libérer les ressources qui ne sont pas utilisées (capteurs, algorithmes couteux en batterie, ...) ou stopper temporairement certaines actions (animation, lecture vidéo,...)

   (c) Le temps d'exécution de `onPause()` doit être relativement court. Les opérations couteuses doivent être retardées à `onStop()`.

   (d) Le système appelle la méthode `onResume()` lorsque l'utilisateur revient sur l'activité.

4. Après avoir rempli le formulaire, analysez les évènements et le visuel lorsque vous effectuez les actions suivantes :

1. Si besoin, un modèle est fourni en annexe

(a) Que ce passe-t-il lorsque vous changez l'orientation du matériel ? [2]

(b) Lorsque vous affichez le menu de configuration de la machine ?

Rendez-vous à la page suivante de la leçon.

5. **Gestion de l'arrêt et du redémarrage d'une activité**

   ⇒ Une fois en bas de page, vous devez avoir pris connaissance des points suivants :

   (a) La méthode `onStop()` est appelée lorsque l'activité n'est plus visible du tout. La méthode `onRestart()` est ensuite appelée lorsque l'activité est à nouveau affichée.

   (b) Lorsqu'une activité est stoppée, le système peut la détruire à tout moment pour libérer la mémoire.

   (c) `onStop()` est appelé après `onPause()` et sert à effectuer les opérations de libération les plus lentes.

   (d) De base, le système sauvegarde automatiquement les informations de la plupart des objets de type `View` [3]

6. Analyse sur notre application :

   (a) Mêmes questions que précédemment (changement d'orientation, analyse, ...) lorsque vous commentez l'appel `super.onRestoreInstanceState(savedInstanceState)` dans `onRestoreInstanceState`

   (b) Comment expliquez vous ce changements ?

   (c) Afin de visualiser la pile d'appel des méthodes, modifiez l'appel de log dans `onSaveInstanceState` par

```
Log.e(TAG, "onSaveInstance␣" + Log.getStackTraceString(new Exception()));
```

   (d) Relancez l'application et changez l'orientation. Au vu de ce qui s'affiche à l'écran à quel moment est appelé `onSaveInstanceState()` ?

   (e) Par analogie, où doit être appelé onRestoreInstanceState ? Vérifiez le en modifiant son appel à Log.

Rendez-vous à la page suivante de la leçon.

7. **Recréer une activité** ⇒ Une fois en bas de page, vous devez avoir pris connaissance des points suivants :

   (a) Il est nécessaire de fournir un identifiant à chaque `View` pour permettre au système de sauvegarder automatiquement son état.

   (b) Il est nécessaire de sauvegarder manuellement les autres informations lors de l'invocation de `onSaveInstanceState()` au travers d'un objet de type `Bundle` (`http://developer.android.com/reference/android/os/Bundle.html`).

   (c) La restauration se fait à l'appel de la méthode `onRestoreInstanceState()`.

Nous allons modifier l'activité de telle façon à ce qu'il soit nécessaire d'effectuer la sauvegarde de certaines données. L'activité va contenir un bouton et un compteur. À chaque fois que l'on appuye sur le bouton, le compteur est incrémenté.

(a) Ajouter dans `activity_my.xml` une nouvelle ligne avec un bouton (`Button`) d'identifiant `bouton` et un champs texte (`TextView`) d'identifiant `compteur` .

```
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <Button android:id="@+id/bouton"  />
    <TextView android:id="@+id/compteur" android:text="0" />
</LinearLayout>
```

(b) Ajoutez à la classe `MyActivity` une variable `mCompteur` de type `int` et initialisez la à 0 dans `onCreate()`.

(c) Toujours dans `onCreate()`, ajoutez un gestionnaire sur le clic du bouton afin d'incrémenter `mCompteur` et d'afficher la valeur dans `compteur`.

2. `[CTRL]+F12` avec l'émulateur
3. C'est pour cette raison que le contenu des formulaires était conservé dans la question précédente

```
bouton = (Button) findViewById(R.id.bouton);
compteur = (TextView) findViewById(R.id.compteur);
bouton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        mCompteur++;
        compteur.setText("*"+mCompteur+"*");
    }
  });
```

(d) Lancer l'application. Cliquez plusieurs fois sur le bouton.

(e) Changez l'orientation du matériel. Que ce passe-t-il ?

Il est donc nécessaire de mettre en place une sauvegarde du compteur ;

(a) Sauvegardez la valeur de `mCompteur` dans la méthode `onSaveInstanceState()` (indice, regardez la documentation de `Bundle`).

(b) Restaurez la valeur de `mCompteur` (et de l'interface graphique) dans `onRestoreInstanceState()`.

A activity_my.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    tools:context=".MyActivity" >
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Nom" />
        <EditText
            android:id="@+id/nom"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:hint="Votre␣nom" />
    </LinearLayout>
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Prénom" />
        <EditText
            android:id="@+id/prenom"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:hint="Votre␣prénom" />
    </LinearLayout>
</LinearLayout>
```