Développement Android Fondamentaux (partie 1)

v1.0 - 06.11.2014

@ Olivier Denier - odenier@myges.fr @

Plan

- **AndroidManifest**
- **Permissions**
- **Application**
- **Composants**
  - Activity
  - Broadcast Receiver
  - Intent

AndroidManifest

L'AndroidManifest est un fichier XML qui décrit l'ensemble des caractéristiques d'une application Android.

Un seul fichier AndroidManifest par Application Android.

L'AndroidManifest d'une application permet d'enregistrer cette application et ses composants au système.

Les caractéristiques de l'application sont :

- Son nom (package name), sa version
- Les versions du SDK Android supportées
- Ses composants
  - Les Activity
  - Les Service
  - Les Content Provider
  - Les Broadcast Receiver
- Les permissions
- Les tailles des écrans supportées
- Des configurations matérielles spécifiques
- ...

Source : http://developer.android.com/guide/topics/manifest/manifest-intro.html

AndroidManifest

Sa structure

```
<?xml version="1.0" encoding="utf-8"?>

<manifest>

    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />
    <compatible-screens />
    <supports-gl-texture />

    <application>

        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>

        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>

        <service>
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>

        <receiver>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>

        <provider>
            <grant-uri-permission />
            <meta-data />
            <path-permission />
        </provider>

        <uses-library />

    </application>

</manifest>
```

AndroidManifest

L'AndroidManifest peut contenir plus ou moins d'éléments (Tags) pour spécifier l'application qu'il définit.

L'ordre des éléments au sein du noeud <manifest></manifest> n'a pas d'importance.

Un même élément (par exemple **<activity>**) peut en général être défini plusieurs fois indiquant ainsi que l'application contient plusieurs de ces éléments (plusieurs **Activity** dans notre exemple).

Les éléments ont des attributs définis qui leurs sont spécifiques à chacun.

AndroidManifest

Les principaux éléments et leurs attributs en détail

- **<manifest>**

  Tag racine du fichier AndroidManifest. Obligatoire.

  - **xmlns:android**
    Définit le namespace des attributs Android
    sa valeur vaut "http://schemas.android.com/apk/res/android"

  - **package**
    Définit le package racine de l'application.
    C'est aussi l'identifiant de l'application. Il ne doit pas changer pour sa publication !

- **android:versionCode**
  Définit le numéro de version (entier) de l'application.
  A incrémenter à chaque nouvelle version à publier !

- **android:versionName**
  Définit le "nom" de version (chaine de caractères) de l'application.
  Peut prendre n'importe quelle valeur; c'est cette valeur que voit l'utilisateur.

- **android:installLocation**
  Définit où l'application peut être installer sur le device.
  Doit avoir une des valeurs : auto, internalOnly ou preferExternal

AndroidManifest

- **<uses-permission>**
  Définit une permission nécessaire au bon fonctionnement de l'application
  - **android:name**
    Définit le nom de la permission.
    Cette permission peut :
    - avoir été définie par l'application.
    - avoir été définie par une autre application
    - être une permission système

- **<uses-sdk>**
  Définit les versions SDK compatibles à l'application via l'API Level
  - **android:minSdkVersion**
    Définit l'API Level minimum nécessaire pour exécuter l'application
  - **android:targetSdkVersion**
    Définit l'API Level cible (pour lequel l'application a été testée) pour exécuter l'application

AndroidManifest

- **<supports-screens>**
  Définit les tailles d'écran supportées par l'application
  - android:smallScreens
  - android:normalScreens
  - android:largeScreens
  - android:xlargeScreens
  - android:anyDensity
- **<application>**
  Définit les propriétés de l'application (son nom, son icône, sa classe Java, ...)
  Cf. Détails dans le paragraphe **Application**
- **<activity>**
  Définit une Activity (contrôleur d'écran) de l'application. Une Activity (classe Java) non déclarée dans l'AndroidManifest, ne peut être lancée.
  Cf. Détails dans le paragraphe **Activity**
- **<service>**
  Définit un Service (traitement exécuté en tâche de fond sans IHM) de l'application. Un Service (classe Java) non déclaré dans l'AndroidManifest, ne peut être exécuté.
  Cf. Détails dans le paragraphe **Service** du cours **Fondamentaux (partie 2)**

AndroidManifest

- **<receiver>**
  Définit un Broadcast Receiver (composant capable de se déclencher sur des Intents émis par le système ou d'autres applications) de l'application.
  Cf. Détails dans le paragraphe **Broadcast Receiver**

- **<intent-filter>**
  Définit le type d'Intent auquel le composant (Activity, Service ou Broadcast Receiver) peut répondre.
  Cf. Détails dans le paragraphe **Intent**

- **<provider>**
  Définit un Content Provider (composant permettant de partager les données de l'application à d'autres applications) de l'application. Un Content Provider (classe Java) non déclaré dans l'AndroidManifest, ne peut être exécuté.
  Cf. Détails dans le paragraphe **Content Provider** du cours **Fondamentaux (partie 2)**

Permissions

Qu'est-ce ?

Il s'agit de restrictions données à l'application afin de limiter ses possibilités d'action (accès au mobile et aux données).

Objectif ?

Cela permet à l'utilisateur de "savoir" ce que l'application nécessite comme permissions pour fonctionner.

Quand ?

C'est **uniquement** à l'installation que l'utilisateur sera informé des permissions dont a besoin l'application et il pourra ou non accepter l'ensemble.

S'il les accepte, l'application pourra donc à "son insu", réaliser des actions sans limitation mais **à la hauteur des permissions accordées**.

Sinon, l'application **ne sera tout simplement pas installée**.

Source : http://developer.android.com/guide/topics/security/permissions.html

Permissions

Définition type dans le fichier AndroidManifest

```
<manifest . . . >
   <permission android:name="com.example.project.DEBIT_ACCT" . . . />
   <uses-permission android:name="com.example.project.DEBIT_ACCT" />
   . . .
   <application . . .>
      <activity android:name="com.example.project.FreneticActivity"
            android:permission="com.example.project.DEBIT_ACCT"
            . . . >
         . . .
      </activity>
   </application>
</manifest>
```

- **<permission>**
  Déclare une permission spécifique à l'application.
  - **android:name**
    Définit le nom de la permission en respectant la nomenclature de packaging Java (ex: fr.esgi.android.MY_PERMISSION). C'est ce nom qui sera utilisé par l'élément <uses-permission>.
- **<uses-permission>**
  Définit une permission nécessaire au bon fonctionnement de l'application
  - **android:name**
    Définit le nom de la permission.

Permissions

Permissions systèmes courantes :

android.permission.

- RECEIVE_SMS
- SEND_SMS
- ACCESS_COARSE_LOCATION
- ACCESS_FINE_LOCATION
- ACCESS_NETWORK_STATE
- BLUETOOTH
- CAMERA
- CHANGE_CONFIGURATION
- GET_TASKS
- INTERNET
- MANAGE_ACCOUNTS
- READ_CONTACTS
- WRITE_CONTACTS
- READ_EXTERNAL_STORAGE
- WRITE_EXTERNAL_STORAGE (disparu en API Level 19)
- ...

Plus de 130 permissions systèmes !

Source : http://developer.android.com/reference/android/Manifest.permission.html

Application

Qu’est-ce ? (définition fonctionnelle)

L’application est le “composant” qui regroupe l’ensemble des composants (Activity, Service, Broadcast Receiver, Content Provider) constituants une application Android.

Qu’est-ce ? (définition technique)

Une classe Java qui étend la classe **android.app.Application**

Cette classe doit être déclarée dans l’**AndroidManifest**

Application

Déclaration dans l'AndroidManifest

- **<application>**
  Définit les propriétés de l'application (son nom, son icône, sa classe Java, ...) et contient la déclaration des composants (<activity>, <service>, <receiver> et <provider>).

    - **android:icon**
      Définit l'icône de l'application en faisant référence à une ressource drawable

    - **android:label**
      Définit le nom de l'application

    - **android:name**
      Définit la classe Java Application associée c'est à dire qui "*extends android.app.Application*"

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="fr.esgi.android.activitytutorial"
   android:versionCode="1" android:versionName="1.0" >
  ...

   <application android:icon="@drawable/ic_launcher" android:label="MonActivityTuto"  android:name=".MyApp">
...
   </application>

</manifest>
```

Source : http://developer.android.com/guide/topics/manifest/application-element.html

Application

L'intérêt principal de cette classe Application, c'est qu'elle **permet de conserver des informations globales** à l'application.

Elle est chargée au tout début du lancement de l'application. Et ainsi, elle peut permettre d'initialiser l'application.

Les informations mémorisées par cette classe Application sont accessibles à tous les composants (Activity, Service, ...) de l'application dont ils font partis.

Remarques :

- Cette classe Application casse un peu l'approche modulaire d'Android en conservant des informations
- Cette classe étant chargée en mémoire durant toute la vie de l'application, ATTENTION au volume de données qu'elle conservera

TD Application

Exo 1

Créez un projet “TDApplication” avec le wizard ADT (avec Eclipse) pour qu’une Activity “MainActivity” soit créée avec sa méthode “onCreate”.
Implémentez une classe “MonApplication” de type Application en surchargeant sa méthode “onCreate”.

- ✓ Vérifier qu’au lancement de l’application, le chargement de “MonApplication” précède bien celui de la “MainActivity”
- ✓ Initialiser une variable au sein de “MonApplication” et récupérez sa valeur dans “MainActivity”

Composants

5 composants forment le socle d'une application Android :

- Activity
- Broadcast Receiver
- Intent et Intent-filter
- Service (étudié dans la partie 2)
- Content Provider (étudié dans la partie 2)

Tous ne seront pas forcément utiles dans votre application.

Source : http://developer.android.com/guide/components/fundamentals.html

Activity

Qu’est-ce ? (définition fonctionnelle)

C’est un écran affiché sur le mobile

Une application est donc constituée d’une ou plusieurs Activity

Qu’est-ce ? (définition technique)

Une classe Java qui étend la classe **android.app.Activity (ou ses sous-classes)**

On peut assimiler l’**Activity comme un contrôleur d’écran**

Source : http://developer.android.com/guide/components/activities.html

Activity

Plusieurs Activity spécifiques prédéfinies :

- **AccountAuthenticatorActivity** : Activité utilisée par un AbstractAccountAuthenticator
- **ActivityGroup** (déprécié en API 13)
- **AliasActivity** : Référence une Activity existante pour en redéfinir les caractéristiques (intent-filter)
- **ExpandableListActivity** : Activity spécifique pour afficher une liste arborescente d'éléments
- **FragmentActivity** : Activity spécifique pour gérer des Fragments et des Loaders (android-support v4)
- **ListActivity** : Activity spécifique pour afficher une liste d'éléments
- **NativeActivity**
- **ActionBarActivity** : Activity spécifique pour gérer l'ActionBar (android-support v7)
- **LauncherActivity**
- **PreferenceActivity** : Activity spécifique pour afficher les "settings" d'une application via ses SharedPreferences
- **TabActivity** (déprécié en API 13)

Activity

Déclaration dans l'AndroidManifest

- **<activity>**
  Définit une Activity (contrôleur d'écran) de l'application. Une Activity (classe Java) non déclarée dans l'AndroidManifest, ne peut être lancée.
    - android:name
      Définit la classe Java Activity associée c'est à dire qui "*extends android.app.Activity*" ou ses sous-classes

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="fr.esgi.android.activitytutorial"
   android:versionCode="1" android:versionName="1.0" >

  ...

   <application android:icon="@drawable/ic_launcher" android:label="@string/app_name">

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

</manifest>
```

Activity

Déclaration dans l'AndroidManifest

Plusieurs autres attributs permettent de spécifier l'Activity.

L'Activity peut également déclarée des Intent-filter qui permettront de la lancer

Source : http://developer.android.com/guide/topics/manifest/activity-element.html

L'Activity de démarrage de votre application doit être déclarée avec l'Intent-filter :

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

Activity

Le Context

Le Context modélise les informations globales de l'application qui sont accessibles à ses composants (Activity, ...).

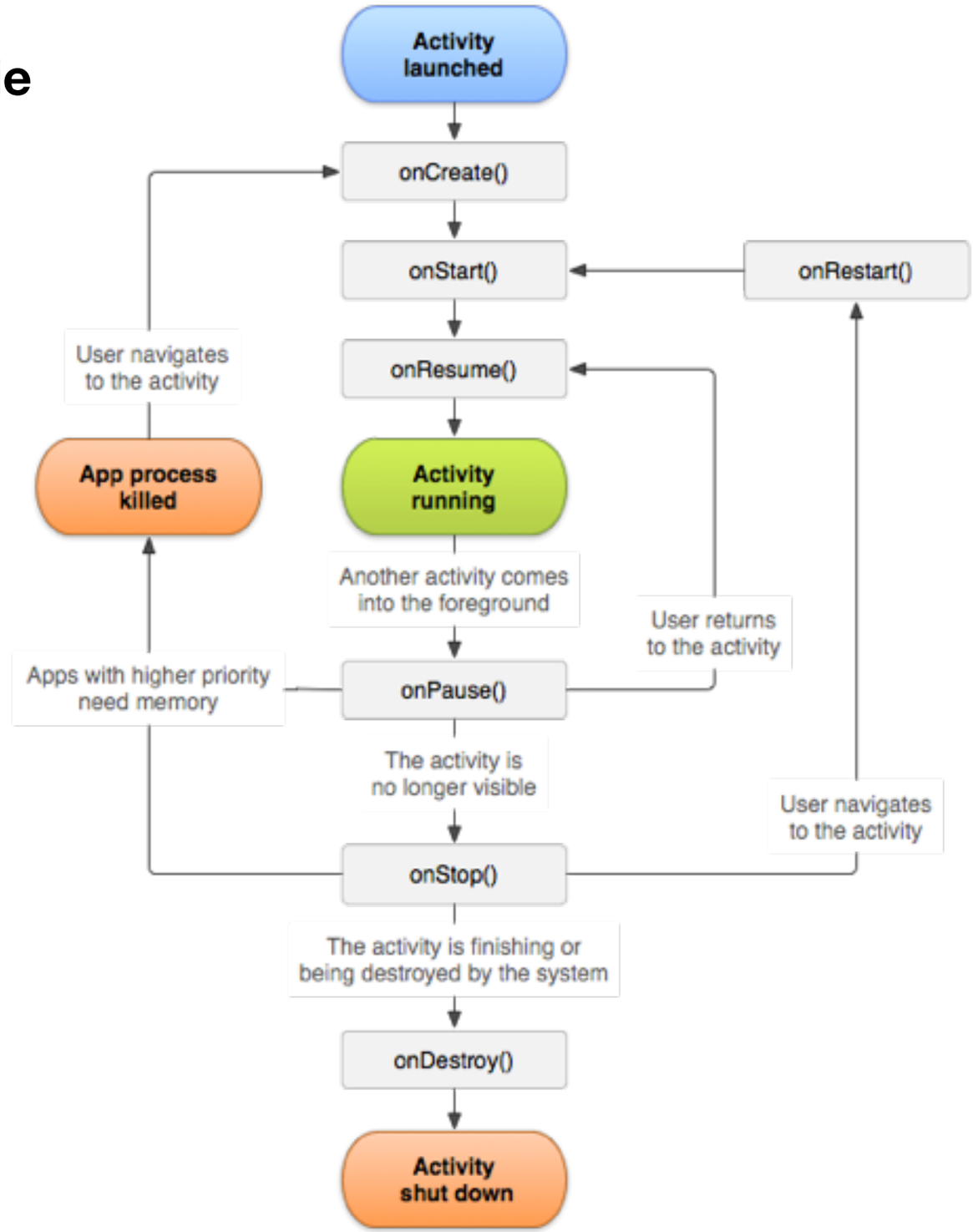
Il permet par exemple :

- d'avoir accès aux ressources de l'application (getRessources), aux services systèmes (getServices), ...
- de démarrer une Activity (startActivity), un service (startService), ...
- de faire des accès au système de fichiers (openFileInput), à la base de données (openOrCreateDatabase), aux préférences (getSharedPreferences), ...
- ...

Une Activity étend un Context !

Activity

Cycle de vie



Activity

Cycle de vie

Le cycle de vie d'une (instance d') Activity est **géré par le système** !

7 méthodes sont déclenchées suivant l'état de l'Activity :

- **onCreate**
  Méthode appelée à la création de l'Activity pour l'initialiser et définir son layout via la méthode "setContentView"
- **onRestart**
  Méthode appelée après un onStop et juste avant un onStart
- **onStart**
  Méthode appelée dés que l'Activity devient visible à l'utilisateur
- **onResume**
  Méthode appelée à chaque fois que l'Activity passe au 1er plan et que l'utilisateur a le focus dessus
- **onPause**
  Méthode appelée si une autre Activity passe au premier plan et a le focus (onResume), mais qu'elle est toujours partiellement visible. C'est la première méthode appelée dés qu'on quitte cette Activity (qui ne sera pas forcément détruite)
- **onStop**
  Méthode appelée dés que l'Activity n'est plus visible (pas rattachée au Window Manager mais encore "en vie").
- **onDestroy**
  Méthode appelée à la destruction de l'Activity (par le système ou via la méthode finish())

Activity

Configuration

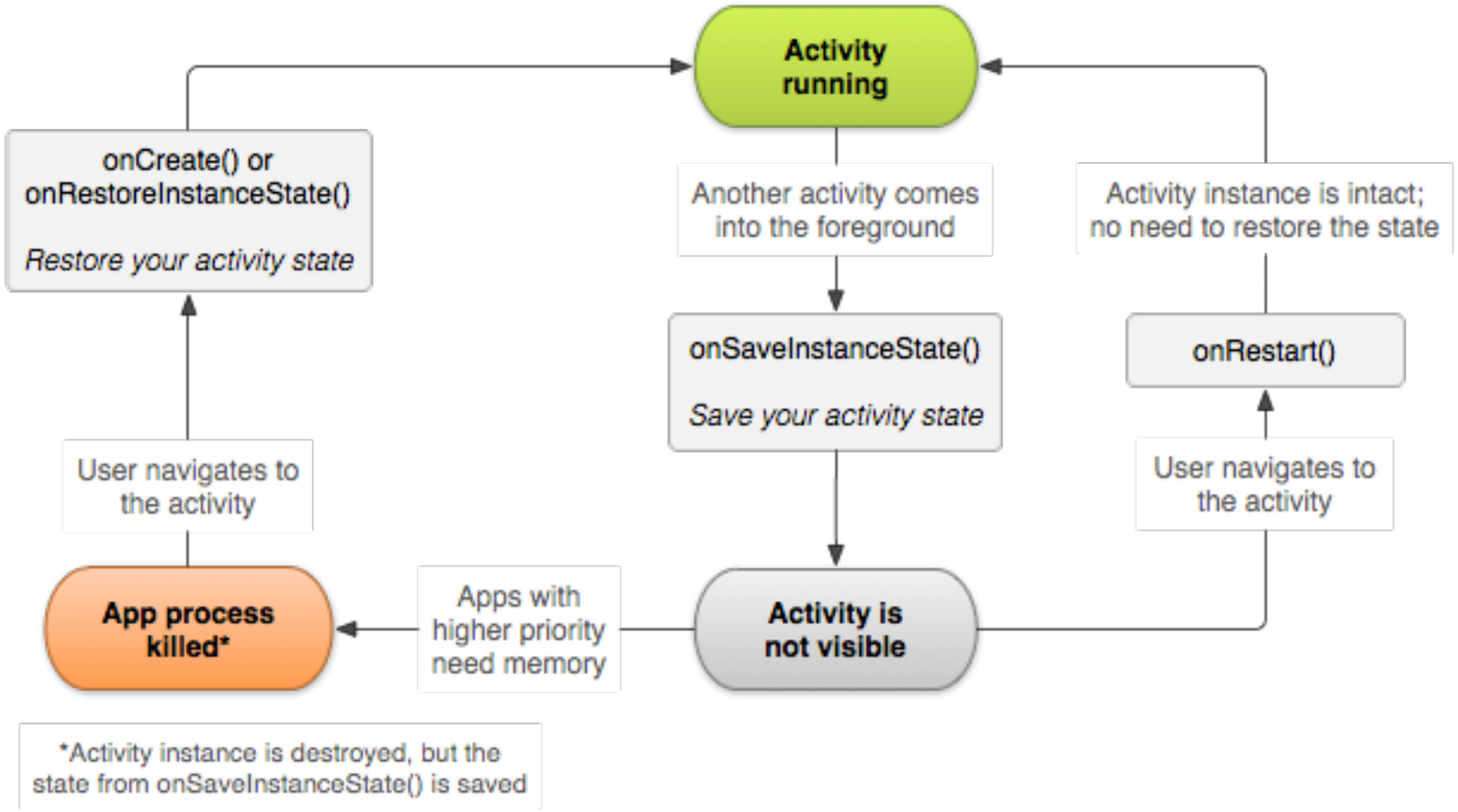
Une activity redémarre automatiquement dés lors que la configuration du device change :

- mcc/mnc : mobile country code/mobile network change
- locale : changement de langue
- *touchscreen*
- keyboard : branchement d'un nouveau clavier
- keyboardHidden
- *navigation*
- screenLayout : affichage sur un autre écran
- fontScale : changement de fonte globale
- uiMode : dock ou night mode
- orientation : passage de portrait/paysage
- screenSize (API Level 13)
- smallestScreenSize (API Level 13)
- layoutDirection (API Level 17)

Activity

Sauvegarde et restauration d'une Activity

Une Activity détruite va être désallouée de la mémoire et par conséquent, les informations qu'elle gérait seront donc définitivement perdues !
D'où le besoin de sauvegarder ses informations afin de la recharger par la suite "dans le même état".



Source : http://developer.android.com/training/basics/activity-lifecycle/recreating.html

Activity

Quoi sauvegarder et comment ?

L'état de l'Activity
Son état doit être sauvegardé en surchargeant la méthode **onSaveInstanceState(Bundle outState)**.
Un Bundle (variable outState ici) va permettre de sauvegarder les informations souhaitées.
Remarque : L'état des Views de l'Activity ayant un ID sont automatiquement sauvegardées

Des données
Elles devront être sauvegardées (dans les préférences, dans un fichier, en base de données) plutôt au sein de la méthode onPause(). Ainsi, dés la disparition de l'Activity, les données seront sauvegardées.

La restauration de l'état de l'Activity

Deux méthodes permettent de restaurer l'état stocké dans un Bundle :

- **onCreate(Bundle savedInstanceState)**
- **onRestoreInstanceState(Bundle savedInstanceState)**

Activity

Cycle de vie de 2 Activity où l'ActivityA lance l'ActivityB

L'ActivityA est passée dans ses méthodes onCreate(), puis onStart() et enfin onResume(). On lance l'ActivityB.

1. ActivityA onPause()
2. ActivityB onCreate() puis onStart() puis onResume()
3. ActivityA onStop()

Important :
Si ActivityB doit utiliser des données persistantes de ActivityA, celles-ci doivent être sauvegardées dans **ActivityA onPause()**

Remarque :
L'ActivityB n'est pas nécessairement une Activity de notre application. Ce peut être l'Activity de réception d'un appel téléphonique par exemple. Dans ce cas, votre application est interrompue et ce n'est qu'à la fin de l'appel que vous retournerez dans votre application.

Activity

Lancement d'une Activity

```
// Démarrage explicite d'une Activity
Intent intent = new Intent(this, OtherActivity.class);
startActivity(intent);
```

```
// Démarrage d'une Activity répondant à une Action
Intent intent = new Intent(Intent.ACTION_SEND);
intent.putExtra(Intent.EXTRA_EMAIL, "my_email@email.com");
startActivity(intent);
```

Arrêt d'une Activity

```
// Arrêt de l'Activity
finish();
```

Activity

D'une Activity **FirstActivity**, lancement d'une autre, **SecondActivity** dont **FirstActivity** souhaite récupérer un résultat

1. Lancement de **SecondActivity**

```
// De FirstActivity, lancement de SecondActivity
Intent intent = new Intent(this, SecondActivity.class);
startActivityForResult(intent, REQUEST_CODE);
```

2. Arrêt de **SecondActivity**

```
// Arrêt de SecondActivity
Intent data = new Intent();
data.putExtra(FirstActivity.INPUT, myInput.getText().toString());
setResult(RESULT_OK, data);
finish();
```

3. Au sein de **FirstActivity**, récupération du résultat de **SecondActivity**

```
// Retour dans FirstActivity et récupération du résultat dans l'intent de la méthode onActivityResult
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (resultCode == RESULT_OK) {
                myText.setText(data.getStringExtra(FirstActivity.INPUT));
        }
        else if (resultCode == RESULT_CANCELED) {
        }
}
```

Activity

Tasks et Back Stack

Le workflow d'une application Android est basé sur une approche modulaire où les Activity sont indépendantes les unes des autres.

L'enchainement des Activity d'une application forme une Task

Boutons Back et Home

- Le bouton Back dépile l'activité courante (elle va être détruite)
- Le bouton Home ne dépile pas l'activité courante (elle passe juste en arrière plan)

Source : http://developer.android.com/guide/components/tasks-and-back-stack.html

TD Activity

Exo 1
Créez une application avec une Activity nommée “MainActivity” et vérifiez le cycle de vie de l’Activity.

Exo 2
Créez une 2ème Activity nommée “OtherActivity” appelée à partir de la précédente et vérifiez le cycle de vie des 2 Activity.

Exo 3
Au sein d’une Activity “SavedStateActivity”, définissez une variable d’instance et faites en sorte que celle-ci soit sauvegardée et restaurée à la recréation de l’Activity

Exo 4
Au sein d’une Activity “FormActivity”, définissez une zone de saisie (EditText avec un ID) et testez si la valeur saisie est bien sauvegardée. Enlevez la déclaration de l’ID et retestez.

TD Activity

Exo 5

Dans l'activity nommée "MainActivity", définissez un Button qui permettra de lancer l'Activity d'appel téléphonique (Intent.ACTION_DIAL).

Exo 6

Créez 2 Activity nommées respectivement "FirstActivity" et "SecondActivity".
"FirstActivity" contient un composant "TextView" permettant d'afficher une valeur et un "Button" permettant de lancer "SecondActivity".
"SecondActivity" contient un composant "EditText" permettant de saisir une valeur et de la valider avec un "Button" pour revenir sur "FirstActivity".
De "FirstActivity", lancez "SecondActivity", saisissez une valeur et faites en sorte qu'en revenant sur "FirstActivity", le composant "TextView" affiche bien la valeur saisie et validée et qu'il n'affiche rien si la valeur n'a pas été validée (bouton BACK).

Exo 7

Créez une nouvelle Activity nommée "DialogActivity". Faites en sorte qu'on puisse la lancer de "MainActivity", mais sans que "MainActivity" passe dans l'état "onStop".

Broadcast Receiver

Qu’est-ce ? (définition fonctionnelle)

Un Broadcast Receiver est un composant capable d’intercepter des messages c’est à dire se déclencher sur des évènements qui peuvent être issus du système.

A l’interception d’un message, le Broadcast Receiver va permettre de lancer une application, déclencher un service ou bien d'appeler le système de notification Android (NotificationManager) qui gère la barre de notification pour informer l’utilisateur qu’un évènement s’est produit en lui transmettant le message.

Qu’est-ce ? (définition technique)

Une classe Java qui étend la classe **android.content.BroadcastReceiver**

Le message transmis est fourni par un **Intent**

Source : http://developer.android.com/reference/android/content/BroadcastReceiver.html

Broadcast Receiver

Déclaration dans l'AndroidManifest

- **<receiver>**
  Définit un Broadcast Receiver de l'application.
  - android:name
    Définit la classe Java Broadcast Receiver associée c'est à dire qui "*extends android.content.BroadcastReceiver*" ou ses sous-classes

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="fr.esgi.android.activitytutorial"
   android:versionCode="1" android:versionName="1.0" >
   ...
   <application android:icon="@drawable/ic_launcher" android:label="@string/app_name">
       ...
       <receiver android:name=".MyReceiver">
         <intent-filter>
            <action android:name="android.intent.action.AIRPLANE_MODE"/>
         </intent-filter>
       </receiver>
   </application>

</manifest>
```

Source : http://developer.android.com/guide/topics/manifest/receiver-element.html

Broadcast Receiver

Le message (type d'évènement)

Il peut être :

- Système (environ 60 évènements)
  - android.intent.action.AIRPLANE_MODE
  - android.intent.action.BATTERY_CHANGED
  - android.intent.action.BATTERY_LOW
  - android.intent.action.BOOT_COMPLETED
  - android.intent.action.MEDIA_XXXXX
  - android.intent.action.PACKAGE_XXXXX
  - android.intent.action.POWER_[CONNECTED|DISCONNECTED]
  - android.intent.action.REBOOT
  - android.intent.action.SCREEN_[ON|OFF]
  - android.intent.action.SHUTDOWN
  - …
- Utilisateur : Le développeur défini son propre évènement au sein de son application

Broadcast Receiver

Abonnement

Un Broadcast Receiver peut s'abonner ou se désabonner à la réception de messages :

- soit au sein de l'AndroidManifest (abonnement uniquement) à l'aide d'un intent-filter qui définit l'action pour laquelle le Broadcast Receiver va se déclencher
- soit par programmation

```
public class OtherActivity extends Activity {
	MyReceiver myReceiver;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_other); // définit le layout de cette activity

		// Abonnement d'un BroadcastReceiver
		myReceiver = new MyReceiver();
		registerReceiver(myReceiver, new IntentFilter(Intent.ACTION_SCREEN_OFF));
	}

	@Override
	protected void onDestroy() {
		super.onDestroy();

		// Désabonnement du BroadcastReceiver
		unregisterReceiver(myReceiver);
	}
}
```

Broadcast Receiver

Déclenchement

```
// Déclenchement explicite d'un Broadcast Receiver
Intent intent = new Intent(this, MyReceiver.class);
sendBroadcast(intent);
```

```
// Déclenchement d'un Broadcast Receiver répondant à une Action
Intent intent = new Intent("MyBroadcastEvent");
sendBroadcast(intent);
```

Réception d'un message

Méthode appelée dés la réception d'un message par un Broadcast Receiver :

- **onReceive(Context context, Intent intent)**

TD Receiver

Exo 1

Créez une application avec une Activity et un Broadcast Receiver.
Au sein de l'Activity, déclenchez le Broadcast Receiver (à l'aide d'un Button par exemple) et vérifiez (surchargez la méthode onReceive) cela avec un Log.

Exo 2

Dans cette même application, faites en sorte que le Broadcast Receiver soit invoqué par un évènement du système (device mis en mode avion par exemple).

Exo 3

Dans cette même application, faire en sorte que le Broadcast Receiver soit déclaré dans l'Activity et pour cette Activity uniquement et invoqué par le déclenchement d'un évènement.

Intent

Qu’est-ce ? (définition fonctionnelle)

Une Intent est un message capable de déclencher une action et plus spécifiquement de :

- lancer une Activity
- démarrer un Service
- déclencher un Broadcast Receiver

Qu’est-ce ? (définition technique)

Une instance de la classe Java **android.content.Intent**

Source : http://developer.android.com/guide/components/intents-filters.html

Intent

Utilisation dans l'AndroidManifest

Les Intents ne sont pas déclarés dans l'AndroidManifest

Ce sont les intent-filter qui y sont déclarés pour les composants (au sein des tags des composants) et qui permettent de les activer.

Intent

Lancement d'une Activity

```
Intent intent = new Intent(this, OtherActivity.class);
startActivity(intent);
// ou
startActivityForResult(intent, requestCode);
```

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
}
```

Démarrage d'un Service

```
Intent serviceIntent = new Intent(this, MyService.class);
startService(serviceIntent);
// ou
bindService(serviceIntent, mServiceConnection, Context.BIND_AUTO_CREATE);
```

Déclenchement d'un BroadcastReceiver

```
Intent intent = new Intent("MyBroadcastEvent");
sendBroadcast(intent);
```

Intent

Informations d'une Intent

- Component Name : package name et class name d'un composant
- Action : Nom de l'action du composant

  Par exemple : ACTION_CALL, ACTION_EDIT, **ACTION_MAIN**, ACTION_SYNC, ACTION_BATTERY_LOW, ACTION_HEADSET_PLUG, ACTION_SCREEN_ON, ...
- Data : URI d'une donnée et son type MIME
- Category : Catégorie du composant

  Par exemple : CATEGORY_BROWSABLE, CATEGORY_GADGET, CATEGORY_HOME, **CATEGORY_LAUNCHER**, CATEGORY_PREFERENCE, ...
- Extras : Dictionnaire Clé/Valeur fourni au composant

  Les méthodes get...Extra() et putExtra() permettent de lire ou écrire une valeur
- Flags

Source : http://developer.android.com/reference/android/content/Intent.html

Intent

Résolution d'Intent

- **Explicite**
  Le composant cible est explicitement nommé

```
Intent intent = new Intent(this, OtherActivity.class);
```

- **Implicite (intent-filter)**
  Le composant cible n'est pas nommé, mais c'est une action qui est demandée et celui-ci saura ou non y répondre

```
Intent intent = new Intent("MyBroadcastEvent");
```

C'est l'**IntentResolver** (composant système) qui va se charger d'attribuer l'Intent au bon composant suivant le paramétrage de :

- L'action
- La data (URI et type MIME)
- La category

Intent

Résolution d'Intent

- **Implicite (intent-filter)**
  Exemple :

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
...
        <activity android:name="NotesList" android:label="@string/title_notes_list">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.VIEW" />
                <action android:name="android.intent.action.EDIT" />
                <action android:name="android.intent.action.PICK" />
                <category android:name="android.intent.category.DEFAULT" />
                <data android:mimeType="vnd.android.cursor.dir/vnd.google.note" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.GET_CONTENT" />
                <category android:name="android.intent.category.DEFAULT" />
                <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
            </intent-filter>
        </activity>
...
</manifest>
```

Intent

Intent-filter

Un intent-filter définit comment un composant (Activity, Service ou Broadcast Receiver) peut être activé.

Plusieurs intent-filter peuvent être définis pour un composant.

Un intent-filter est constitué de tags :

- <action />
- <category /> (optionnel)
- <data /> (optionnel)

Intent

Utilisation dans l'AndroidManifest

- **<intent-filter>**
  Définit un filtre sur les Intents qu'un composant pourra ou non traiter.
  Le filtre doit être défini par au moins une **action**, mais peut aussi des **category** et **data**.

Exemple :

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

Sources :

http://developer.android.com/guide/topics/manifest/action-element.html
http://developer.android.com/guide/topics/manifest/category-element.html
http://developer.android.com/guide/topics/manifest/data-element.html

Intent

Types de données embarquées dans l'Intent (Extras)

- Une donnée typée et identifiée par une clé

  Ecriture : putExtra(String name, <un_type> value)

  Lecture : get<un_type>Extra(String name)

```
Intent intent = new Intent(this, MyActivity.class);
intent.putExtra("monID", 123);
...
int value = intent.getIntExtra("monID", -1);
```

- Un objet complexe (Parcelable)

```
Intent intent = new Intent(this, MyActivity.class);
// myObject doit être Parcelable !!
intent.putExtra("monObjet", myObject);
...
// Le type TypedObject doit être Parcelable !!
<TypedObject> value = intent.getParcelableExtra("monObjet");
```

Intent

Types de données embarquées dans l'Intent (Extras)

- Un ensemble de données regroupées dans un Bundle

```
Intent intent = new Intent();
Bundle sendBundle = new Bundle();
sendBundle.putFloat("floatKey", (float) 3.14);
sendBundle.putString("stringKey", "Lorem Ipsum");
intent.putExtra("monDico", sendBundle);
...
Bundle receiveBundle = intent.getBundleExtra("monDico");
```

Intent

Actions natives/Category natives/Extra natives

Les actions natives, les category natives, les extras natives Android sont définies par des constantes de la classe **android.content.Intent**

Quelques actions natives d'Activity :
ACTION_MAIN, ACTION_VIEW, ACTION_DIAL, ACTION_CALL, ACTION_SEND, ...

Quelques actions natives de Broadcast :
ACTION_BOOT_COMPLETED, ACTION_POWER_CONNECTED, ACTION_POWER_DISCONNECTED, ACTION_SHUTDOWN, ...

Quelques category natives :
CATEGORY_DEFAULT, CATEGORY_BROWSABLE, CATEGORY_LAUNCHER, CATEGORY_HOME, ...

Quelques extra natives :
EXTRA_BCC, EXTRA_CC, EXTRA_EMAIL, EXTRA_PHONE_NUMBER, EXTRA_REFERRER, EXTRA_SUBJECT, EXTRA_TEXT, …

Intents de lancement des applications Google

http://developer.android.com/guide/appendix/g-app-intents.html

TD Intent

Exo 1
Créez une application avec 2 Activity. La 1ère va lancer la 2ème en lui passant un paramètre simple qu'elle récupérera (à l'aide de la méthode getIntent) et qu'elle affichera dans un Log.

Exo 2
Dans cette même application, la 1ère Activity va lancer la 2ème en lui passant une valeur saisie au sein du formulaire composé d'un composant EditText et d'un Button. Celle-ci devra l'afficher à l'aide d'un Log.

Exo 3
Dans cette même application, remplacez la valeur simple transmise d'une Activity à l'autre par un objet complexe, à savoir un objet "Car" (Parcelable) composé de 2 propriétés "year" de type entier et "model" de type chaine de caractères. On récupérera dans la 2ème Activity cet objet dont on affichera les caractéristiques au moyen d'un Log.

Q&A