INFO0306 - Programmation mobile

sommaire

- Rappels
- Composants d'une applications Android
- Types d'applications
- Cycle de vie d'une application

Rappels

- Architecture en 5 couches :
  - Noyau
  - Bibliothèques natives
  - Runtime
  - Framework
  - Application

Composants d'une Applications Android

Généralités

- Les applications sont écrites en Java
- Le code compilé ”dex” ainsi que les ressources (images, layout...) sont regroupés dans une archive au format ”apk” par les outils du SDK
- Cette archive ”apk” est un tout permettant la distribution et l’installation de l’application sur n’importe quelle plateforme Android

Indépendance des Applis

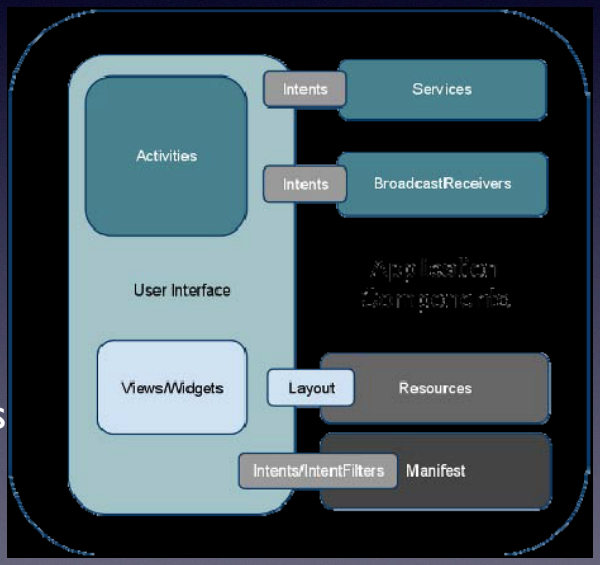
- Chaque application Android est isolée des autres à plusieurs
  - Chaque application tourne sur son propre process Linux :
    - Ce processus est lancé par Android dès qu'une partie du code nécessite une exécution
    - Inversement tue les processus dont il n'a plus d'utilité.
  - Chaque process utilise sa propre machine virtuelle Dalvik :
    - Chaque application possède son propre environnement.
  - Chaque application est associée à un unique Linux User Id :
    - Les fichiers d'une application ne sont pas visibles par les autres applications.
    - Mais, il existe des moyens de partager ces ressources, par exemple via les Content Provider
  - Forcer le partage :
    - Il est possible de forcer deux application de partager le même user ID (et donc de partager des fichiers nativement).
    - Il est possible donc d'utiliser la même VM et le même processus Linux.

Indépendance des Applis

- Un des aspects les plus important d'Android est la réutilisabilité :
  - Chaque application peut utiliser des "morceaux d'autres applications" (si elle le permettent)
  - Par exemple si votre application permet de retoucher des photos et que vous désirez publier cette photo vous pouvez utiliser toutes les applications déjà présentes pour réaliser cette tâche (facebook, picasa, mail ...). Et sans utiliser le code de cette application tierce juste en appeler la partie intéressante.
- Ainsi le système doit être capable :
  - De lancer n'importe quelle partie exposée d'une application sans en lancer la totalité
  - Donc les applications Andoid n'ont pas de point d'entrée global (méthode main()).
  - Elles sont composées d'éléments indépendants ou chacun peut être lancé individuellement.

Eléments Fondamentaux des Applis Android

- Les applications Android sont constitués de composants à couplage.
- Les composants sont liés par un Manifest d'application qui décrit chacun d'eux et comment ils interagissent.
- Les composants suivants sont en quelque sorte les briques élémentaires de vos applications :
- Activity : qui est la couche de présentation de l'application
- Service : ces composants tournent en arrière plan
- BroadcastReceiver : Consommateurs des messages diffusés par les intents
- ContentProvider : Sources de données partageables
- Intent : Framework de communication inter-applications
- Widgets : Composant d'application visuels
- Notifications : Framework de notifications utilisateurs



Les types d'application Android

- La plupart des applications Android appartiennent à l'une des catégories suivantes :
  - Application de premier plan :
    - utilisable uniquement lorsqu'elle est visible
    - mise en suspens lorsqu'elle ne l'est pas;
  - Application d'arrière plan :
    - une application à interaction limitée
    - reste la plupart du temps cachée;
  - Intermittente :
    - présente une certaine interactivité mais effectue l'essentiel de sa tâche en arrière plan.
    - notifie l'utilisateur lorsque cela est nécessaire;
  - Widget :
    - représentée que sous la forme d'un widget de l'écran d'accueil;

Les Activity

- Une activité (”Activity”) = une IHM pour une action utilisateur précise :
  - Liste d’éléments parmi lesquels l’utilisateur peut choisir
  - Affichage d’une image avec un titre
  - Affichage d’un calendrier pour choisir une date
- Exemple d’une application de SMS :
  - Une activité pour choisir un contact
  - Une autre pour écrire le message
  - Une autre pour afficher un historique d’échanges.
- Chaque activité est indépendante des autres
- Une activité doit hériter de la classe : android.app.Activity

Les Activity

- Une application est donc un ensemble d'activités
- On doit définir quelle est la première activité à exécuter lors du lancement de l'application
- Pour naviguer dans l'application chaque activité doit elle-même lancer l'activité suivante.
- Chaque activité est assignée à une fenêtre :
  - Plein écran
  - Fenêtre flottante
- Une activité peux aussi posséder des sous fenêtres
  - Pop-up ...

Les Activity

- Le rendu d'une activité est défini par :
  - Une ou un ensemble de vues
  - Les vues héritent de la classe android.view.View
  - Chaque vue contrôle une zone rectangulaire de l'activité
  - L'organisation est défini par un arbre de "Layout" ou chaque feuille est une vue.
  - Un grand nombre de vues standards sont proposées (combobox, zone de texte, bouton...)
  - Possibilité de définir des vues personnalisées
- Les layouts :
  - Agents de placement
  - Plusieurs layouts sont proposés en standard
- Possibilité de définir ses propres Layout
- Les layout sont utilisable via des fichier XML ou via le code Java

Service

- Un service ne possède pas d'interface
- Tourne en arrière plan en continue (ou presque)
- Exemple :
  - Lecture de musique
  - Collecte de données affichables dans une activité
  - Suivi GPS
  - Vérification de mise à jour
- Lancement d'une application musicale :
  - Démarrage de l'activité de "choix de chanson"
  - L'utilisateur lance la musique
  - Le service diffuse cette musique
  - L'utilisateur peut quitter l'application en fermant l'activité
  - La musique continue à être diffusée!

Service

- Pour communiquer avec un service il faut :
  - S'y connecter (il se lance s'il était arrêté)
  - Utiliser l'interface que présente ce service
  - Exemple : Play(), Pause(), next() ...
- Un service s'exécute dans un Thread et donc ne bloque pas le reste du terminal quand il tourne en fond.

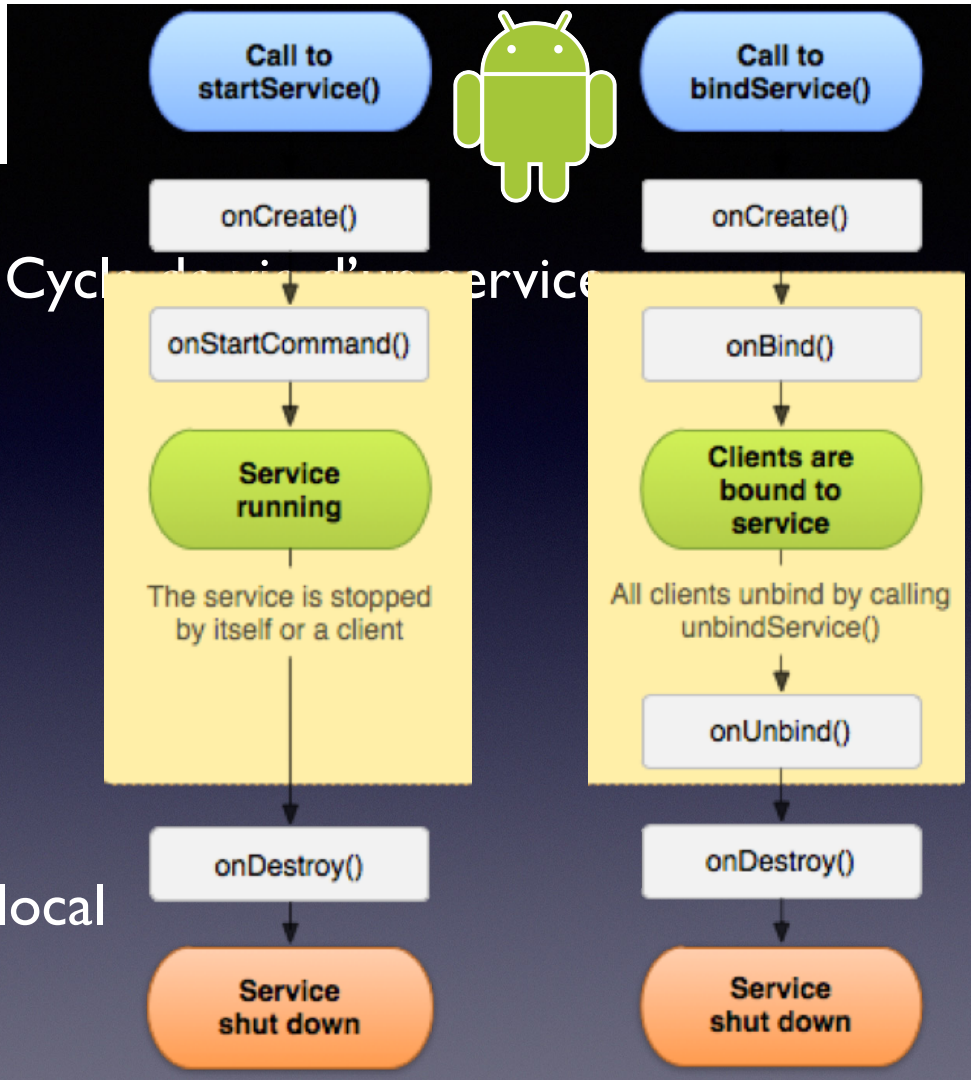
Types de service

- Service local (démarré)
  - Il doit être explicitement appelé par la méthode startService().
  - Une fois en cours d'exécution, ce service continue de tourner en arrière-plan (même si le composant l'ayant créé est détruit).
  - De façon générale, le service doit s'arrêter de lui-même lorsque son opération est terminée (téléchargement d'un fichier par exemple).
- Le service est dit distant (lié)
  - Il est appelé avec la méthode bindService().
  - Dans ce cas, le service dispose d'une interface client-serveur :
    - permettant au composant l'ayant démarré d'interagir avec lui.
    - Plusieurs composants peuvent se lier à un même service,
    - De ce fait, un service reste actif tant qu'il est attaché à un composant.
    - Il est détruit lorsque plus aucun composant ne s'y attache.

Les classes implémentant un service

- La classe Service :
  - la plus générique
  - elle permet de créer un service standard
  - son contenu sera exécuté dans le thread principal de l'application (UIThread)
  - il est possible de créer un nouveau thread pour éviter les ANR (Application Not Responding).
- La classe IntentService :
  - la plus pratique
  - elle exécute le code dans son propre thread en respectant une file d'attente.
  - à chaque utilisation de startService(), la nouvelle demande sera ajoutée à la file d'attente
  - toutes les requêtes seront gérées par ordre d'arrivée.

Cycle de vie d'un service



Service local

Service distant

BroadcastReceiver

- Les broadcast receiver sont :
  - Des éléments inactifs qui attendent un évènement
  - Il y a des évènements système :
    - Batterie faible
    - Changement de langue du système
    - L'utilisateur a pris une photo
- Il est possible de définir ses propres évènements
- Ils héritent de la classe android.content.BroadcastReceiver
- Une application peux contenir plusieurs receiver : un par évènement important
- Les receiver n'ont évidemment pas d'interface
- Ils peuvent lancer des activités en cas de besoin
- Ils peuvent également utiliser le NotificationManager pour signaler quelque chose à l'utilisateur
  - Exemple : Icone, vibration, alerte sonore, clignotement diode...

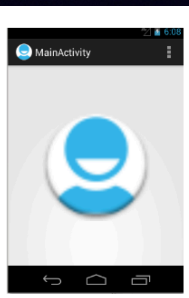
Content Provider

- Les content provider permettent de partager du contenu entre les applications
- Une application s'en sert pour rendre public certaines de ses données
- Le données sont donc exposées dans une classe héritant de android.content.ContentProvider
- Méthodes utilisées : Query(), Insert(), Update(), Delete()…
- Les autres applications n'accèdent pas directement à la classe de ContentProvider
- Utilisation d'un ContentResolver qui va rediriger les requêtes vers le provider voulu
- Si l'on tente d'accéder à une ressource d'une application n'étant pas en cours d'exécution le système Android se charge de la lancer avant.

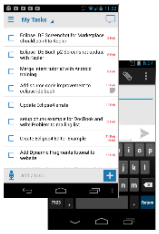
Intent

- Les content providers sont activés par une requête d'un content resolver
- Mais les 3 autres systèmes (Activity, Service, BroadCast Receiver) sont activés par des messages asynchrone appelés "Intent"
- Un intent dérive de android.content.Intent
- Un intent possède une action et un contenu particulier
- Pour les activités et les services il nomme l'action désirée et précise l'URI (Identifiants Uniformes de Ressources) des données sur lesquelles agir.
  - Exemple :
    - Afficher / image,
    - Editer / texte, etc.
- Pour les broadcast receivers il se contente de nommer l'action à annoncer
  - Exemple : Batterie faible, etc.

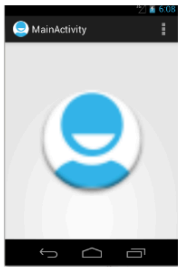
Intent



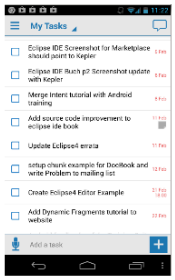
Intent resolution by the Android system



One activity is started



Intent + resultCode provided by called activity



onActivityResult(requestCode, resultCode, intent)

RequestCode provided by Android to identify which activity type was started

Intent

- Les Intents et les activités :
  - Lancement en passant un Intent en paramètre à une des méthodes suivantes :
    - Context.startActivity(intent)
    - Activity.startActivityForResult(intent)
  - L'activity peut accéder à celui ci avec :
    - getIntent()
  - Si le système doit envoyer des nouveaux intent :
    - Appel de onNewIntent(intent) sur l'activité
  - En cas de résultat attendu
    - Appel de onActivityResult() sur l'activité appelante

Intent

- Les Intents et les services :
  - Lancement en passant un Intent en paramètre à la méthode suivante :
    - Context.startService(intent)
    - Le système appellera ensuite la méthode onStart(intent) en précisant cet Intent en paramètre
  - Connexion en passant un Intent en paramètre à la méthode suivante :
    - Context.bindService(intent)
    - Le Système appellera ensuite la méthode onBind(intent) en précisant cet Intent en paramètre, en le créant s'il le faut

Intent

- Les Intents et les Broadcast Receiver :
  - Une application voulant envoyer un évènement va utiliser une des méthodes suivantes :
    - Context.sendBroadcast(intent) —> Broadcast du BR à tous
    - Context.sendOrderedBroadcast() —> Broadcast ordonné selon priorités dans intent-filter
    - Context.sendStickyBroadcast() —> Broadcast valable pour les futurs BR dynamiques lancés.
  - Le système va alors appeler la méthode onReceive(intent) sur tous les broadcast receivers intéressés en passant en paramètre l'Intent.

Intent

- La catégorie :
  - Une chaine de caractère précisant quel type de composant peut gérer l'intent.
  - Plusieurs catégories peuvent être précisées.
  - Exemples :
    - CATEGORY_BROWSABLE : Le contenu peut être affiché dans le navigateur
    - CATEGORY_HOME : L'activité est de type Home
    - CATEGORY_LAUNCHER : L'activité est lançable par le launcher et donc doit y être présente
    - CATEGORY_PREFERENCE : l'activité est un panneau de préférences

Intent

- Quelques exemples :
  - ACTION VIEW content://contacts/people/1 – Affiche les information sur le contact 1
  - ACTION DIAL content://contacts/people/1 – Affiche le mode d'appel rempli avec les informations du contact 1
  - ACTION VIEW tel:123 – Affiche le mode d'appel rempli avec "123". (ACTION VIEW s'adapte donc au contenu)
  - ACTION DIAL tel:123 – Idem
  - ACTION EDIT content://contacts/people/1 – Permet de modifier les informations du contact 1
  - ACTION VIEW content://contacts/people/ – Affiche la liste des contacts (le choix d'un de ces contact génèrera un Intent pour afficher ce contact)

Manifest

- Fichier XML
- Chaque projet Android inclut un fichier aux format XML nommé Manifest (AndroidManifest.xml)
- Le Manifest permet de définir la structure et les métadonnées de votre application, ses composants et ses pré-requis
- Précise l'architecture de l'application
- Il contient des balises pour chacun des composants qui constituent votre application (Activities, Services, Content Providers et Broadcast Receivers)
- Le Manifest fournit aussi des attributs permettant de spécifier les métadonnées d'une application (Exemple son icône, son thème).
- Chaque application doit en avoir un
- AndroidManifest.xml à la racine du projet

Manifest

- Contenu :
  - Précise le nom du package java utilisant l'application : Cela sert d'identifiant unique !
  - Il décrit les composants de l'application
    - Liste des activités, services, broadcast receivers
    - Précise les classes qui les implémentent
    - Précise leurs capacités (à quels intents ils réagissent)
    - Ceci permet au système de savoir comment lancer chaque partie de l'application afin de satisfaire au principe de la réutilisabilité.
  - Définit les permissions de l'application :
    - Droit de passer des appels
    - Droit d'accéder à Internet
    - Droit d'accéder au GPS, etc.
  - Précise la version d'Android minimum nécessaire
  - Déclare les librairies utilisées
  - Déclare des outils d'Instrumentation/déboggage (en développement)

Manifest

- Conventions :
  - Seuls deux éléments sont obligatoires :
    - < manifest > : contient le package, la version... Englobe tout le fichier
    - < application > : décrit l'application et contiendra la liste de ses composants.
  - Les données sont passées en tant qu'attribut et non en tant que contenu
  - Tous les attributs commencent par "android:" (sauf quelques un dans < manifest >)
- Les ressources
  - Au lieu de contenir les données en tant que tel le fichier manifest peut faire appel à des ressources
  - < activityandroid : icon = "@drawable/smallPic"... >
  - Ces ressources sont définies dans le répertoire "res" de l'application.

Manifest

- Permissions :
  - Une application ne peut pas utiliser certaines fonctionnalités sauf si elle le précise dans le fichier Manifest
  - Il faut donc préciser les permissions nécessaires grace à :
    - < uses-permission >
  - Il existe des permission standard :
    - android.permission.CALL_EMERGENCY_NUMBERS
    - android.permission.READ_OWNER_DATA
    - android.permission.SET_WALLPAPER
    - android.permission.DEVICE_POWER
  - Il est possible de définir ses propres permissions

Manifest

- Intent Filter :
  - Ils informent le système à quelle intent les composants peuvent réagir
  - Un composant peut avoir plusieurs filtres :
    - Exemple : Editeur de texte
      - Filtre pour éditer un document existant
      - Filtre pour initier un nouveau document
  - Un filtre doit posséder une ”action” qui définit à quoi il correspond

Manifest : Exemple

```
<?xml version="1.0" encoding="utf-8" ?>
- <manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.kahriboo.menu3d" android:versionCode="1" android:versionName="1.0">
  - <application android:icon="@drawable/icon1" android:label="@string/app_name"
      android:debuggable="true" android:theme="@android:style/Theme.Translucent.NoTitleBar">
    - <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>
    </application>
    <uses-sdk android:minSdkVersion="6" />
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.VIBRATE" />
  </manifest>
```

Manifest & Eclipse

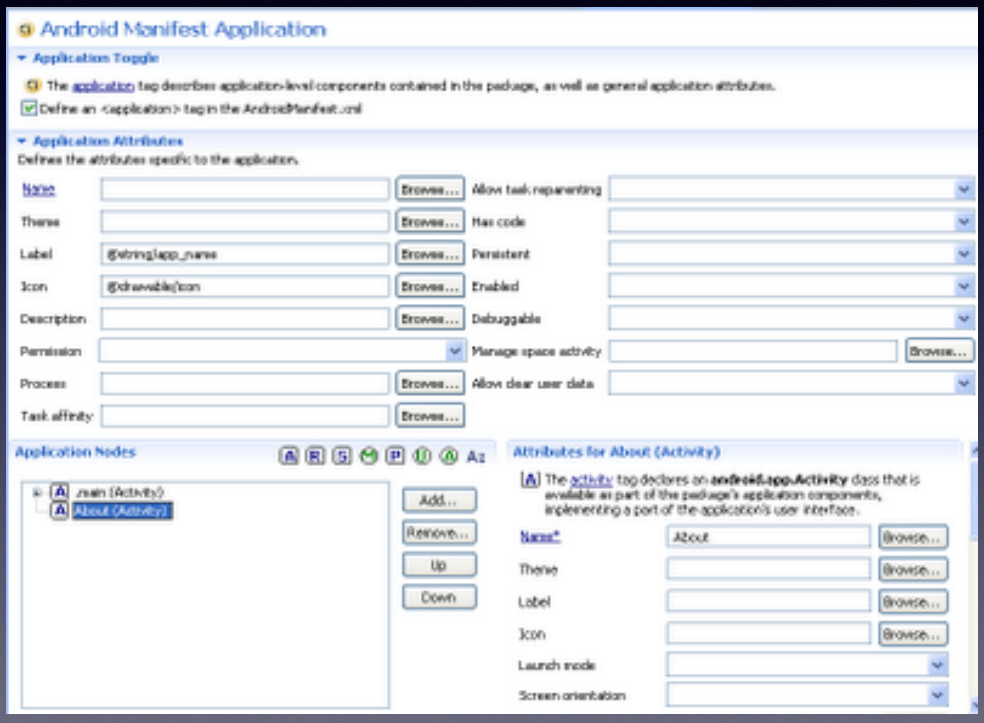
- Le plugin ADT d'Eclipse comprend un éditeur visuel qui évite de manipuler directement le XML.

- Pour y accéder :
  - Clic droit sur le fichier AndroidManifest.xml situé dans votre dossier de projet
  - Sélectionnez Ouvrir Avec —> Android Manifest Editor.

- Cet éditeur permet ainsi de spécifier les attributs d'une application (icône, label, thème) dans le panneau Application Attributes.

- Le panneau Application Nodes permet de gérer les composants de l'application.

Manifest : Editeur sous Eclipse



Activity & Task

- Résumé d'une situation :
  - L'application A doit afficher une carte
  - A prépare l'intent avec les données nécessaires (GPS par exemple)
  - A appelle startActivity() avec cet intent
  - Le système trouve l'application B qui sait gérer cet Intent
  - L'application B affiche la carte
  - L'utilisateur ferme cette carte (bouton back)
  - L'application A reprends la main

Activity & Task

- Conclusion pour cette situation :
  - Du point de vue de l'utilisateur :
    - 1 seule application (A et B sont confondues)
  - Du point de vue du système :
    - 2 applications
    - 2 DVM
    - 2 process
- 1 Tâche = 1 Application au sens utilisateur.

Activity & Task

- Une tâche :
  - Est une pile d'activités
  - La première est celle qui a été initiée par l'utilisateur
  - Les activités peuvent provenir de différentes applications
  - L'ensemble forme un tout
    - Mis en arrière plan en même temps
    - Remise au premier plan dans son ensemble
- Comportement par défaut modifiable via le manifest et le tag "< activity >" et ses flags

Processus & Threads

- Quand le premier composant d'une application nécessite une exécution, Android démarre un nouveau processus Linux pour gérer ce composant
- Chaque composant peut préciser dans le Manifest (via l'attribut "process") s'il doit s'exécuter dans un nouveau processus ou s'il doit partager un processus existant
- Deux composants de deux applications peuvent aussi partager le même processus si :
  - Elles utilisent le même Linux User ID
  - Elles sont signées par la même autorité
- Attention pour les composants utilisés dans le même processus!
  - Ne pas faire de longues opérations lors des appels par le Système sinon cela bloquera tout le reste des composants.
  - Pensez à utiliser des Threads pour les traitements longs.
  - Utilisez la classe classique Java de Threads
  - Android fournit aussi des classes utilitaires pour simplifier l'utilisation des Threads

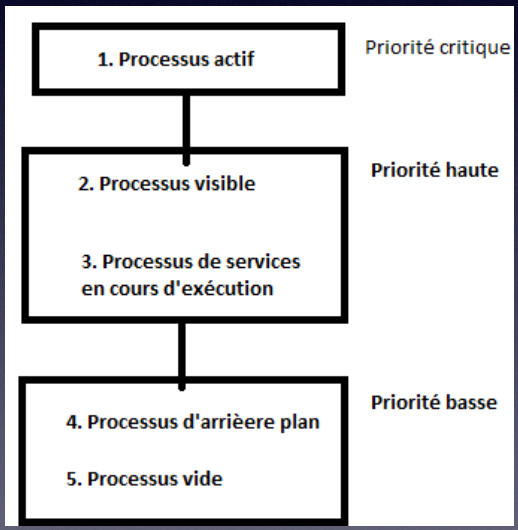
Cycle de vie d'une Applications Android

Cycle de Vie des Activity

• Les applications Android ont un contrôle limité sur leur propre cycle de vie.
• Leurs composants doivent être à l'écoute des changements d'état de l'application et réagir en conséquence pour une fin intempestive.
• Par défaut, chaque application Android est exécutée dans son propre processus, exécutant une instance distincte de la machine Dalvik.
• Celle-ci se charge de la mémoire et la gestion de processus.
• Les processus sont tués sans avertissement pour libérer des ressources pour d'autres applications

Priorités des applications

- L'ordre dans lequel les processus sont tués pour libérer les ressources est déterminé par la priorité de leur application.
- La priorité d'une application est celle du composant de plus haute priorité.
- Les priorités peuvent se classer en 3 types:
  - Priorité critique: le processus est actif;
  - Priorité haute: le processus est visible ou processus de service en cours d'exécution;
  - Priorité basse: processus d'arrière plan ou processus vide;
- L'arbre des priorités utilisé pour déterminer l'ordre de fin des applications



Etat des processus

- Processus actifs :
  - Sont les processus au premier plan, les composants qui interagissent avec l'utilisateur.
  - Android les garde à l'état actif en récupérant des ressources.
- Processus visible :
  - Sont les processus qui hébergent des Activities "visible" mais à l'état inactif.
  - Ils sont tués aux cas extrêmes pour permettre aux processus actifs de continuer à s'exécuter.
- Processus de services en cours d'exécution :
  - Sont les processus hébergeant des services ayant été démarrés. Ils s'exécutent sans interface utilisateur et en continue.
  - Ils sont tués que si des ressources sont nécessaires aux processus actifs ou visibles.
- Processus d'arrière-plan:
  - Sont les processus hébergeant des Activités non visibles et n'ayant aucun service en cours d'exécution.
  - Ils sont tués pour récupérer des ressources au profit des processus de premier plan.
- Processus vide :
  - Pour des raisons de performance, Android garde des applications en mémoire une fois celles-ci terminées.
  - Ce cache réduit le temps de relance des ces applications. Cependant ces processus sont tués si nécéssaire

Cycle de Vie des Activity

- Une activité possède trois états :
  - Active (running) : Quand l'activité est au premier plan et reçoit les actions utilisateur.
  - Paused : Quand elle est toujours visible mais n'a pas le focus (autre activité transparente par dessus ou activité ne prenant pas tout l'écran)
    - Toujours vivante
    - Mais peut être tuée en cas de ressources très limitées
  - Stopped : Quand elle n'est plus visible
    - Toujours vivante
    - Mais sera tuée dès que des ressources seront nécessaires.
- Le système tue les activités en état "stopped" ("paused") de deux manières :
  - En appelant la méthode finish()
  - En tuant le processus tout simplement
- Quand l'activité sera a nouveau demandée :
  - Doit être complètement reconstruite
  - Doit Potentiellement recharger son dernier état

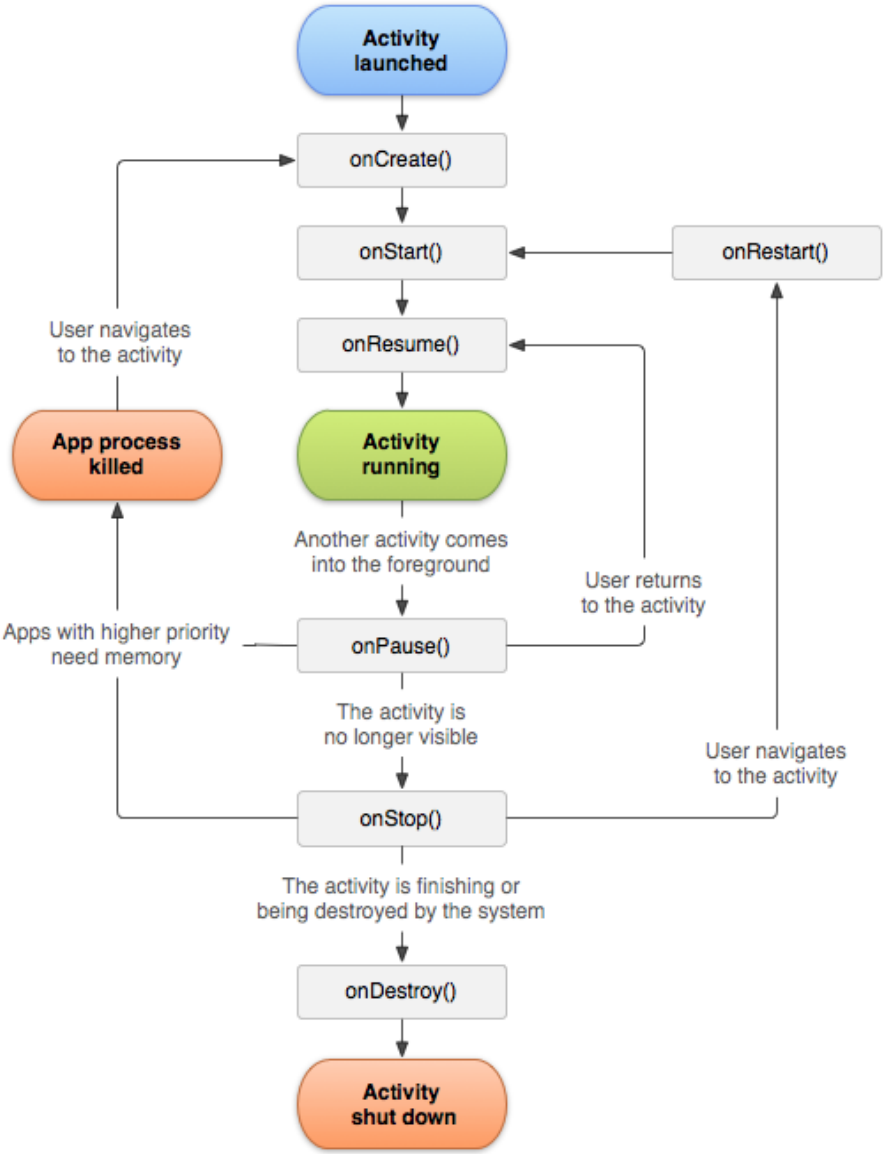
Cycle de Vie des Activity

- Une activité est notifiée de ses changement d'état par l'appel à ses méthodes :
  - void onCreate (Bundle savedInstanceState)
  - void onStart()
  - void onRestart()
  - void onResume()
  - void onPause()
  - void onStop()
  - void onDestroy()

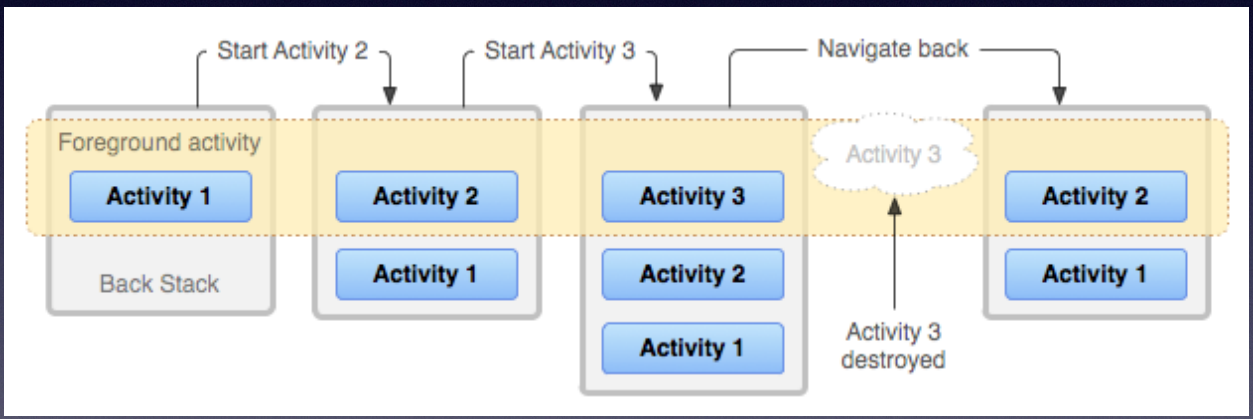
Cycle de Vie des Activity

- Afin de sauvegarder le contexte le système appelle ”onSaveInstanceState()” avant de rendre l’application potentiellement tuable (paused...)
- Cet appel fournit un bundle ”clé/valeurs” pour que le développeur puisse sauvegarder l’état
- Au prochain appel de ”onCreate()” ce bundle sera fournit
- Il est également fournit via un appel à ”onRestoreInstanceState()”
- L’appel à la sauvegarde n’est fait qu’en cas de risque de terminaison de l’activité par le système et non si cela vient d’une action utilisateur (back)

Cyc


Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Back Stack


Start Activity 2
Start Activity 3
Navigate back
Foreground activity
Activity 1
Back Stack
Activity 2
Activity 1
Activity 3
Activity 2
Activity 1
Activity 3
Activity 3
destroyed
Activity 2
Activity 1

Développement d'Applications sous Android

Composants et Cycles de vie d'Applications d'Android

Post-it

- L'application Android peut englober plusieurs composants : Activity, Service, Broadcast Receiver, ContentProvider, Intent, Widgets, Notifications.

- Les activités communiquent grâce à des Intent

- Dans Android, l'accent est mis sur la réutilisation des composants avec les Intent

- Il existe trois états pour les activités : Active, Paused et Stopped

- Chaque activité possède son propre Processus