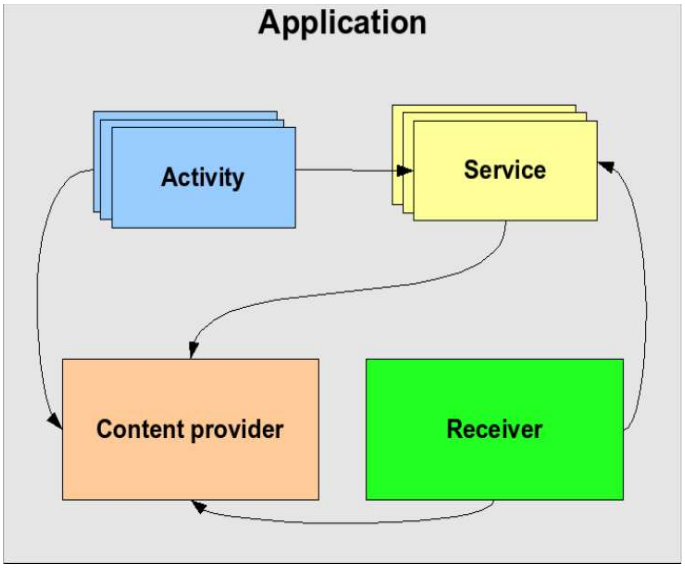
Chapitre 2
Cycle de vie d'une application

Une application Android est composée d'un ensemble de 4 éléments qui interagissent entre eux.

Ces éléments sont : « Activité », « Service », « Broadcast Reciever » et « Content Provider ».



1. Activité

- L'activité est le premier composant essentiel permettant de gérer le cycle de vie d'une application et l'interactivité avec l'utilisateur moyennant une interface graphique.

- Par extension, on dit que l'activité est grosso modo la forme disponible sur l'écran.

- Elle offre à l'utilisateur l'interface (écran) pour interagir avec l'application.

- Chaque écran de l'application représente une activité.

- Une application Android est composée d'une ou de plusieurs activités (un jeu sur plusieurs écrans).

- Une seule activité est lancée à la fois.

- Dans « Java », chaque activité est une classe qui étend par héritage la classe « Activity ».

- Lors du démarrage d'une activité, 3 méthodes sont appelées automatiquement : « onCreate », « onStart » et « onResume ».

- Lorsque l'on quitte l'activité, 3 méthodes sont appelées automatiquement : « onPause », « onStop » et « onDestroy ».

- Ces méthodes sont définies dans la classe « Activity » et devront être redéfinies si c'est nécessaire.

- L'ensemble du code « Java » est généralement écrit dans la méthode « onCreate ».

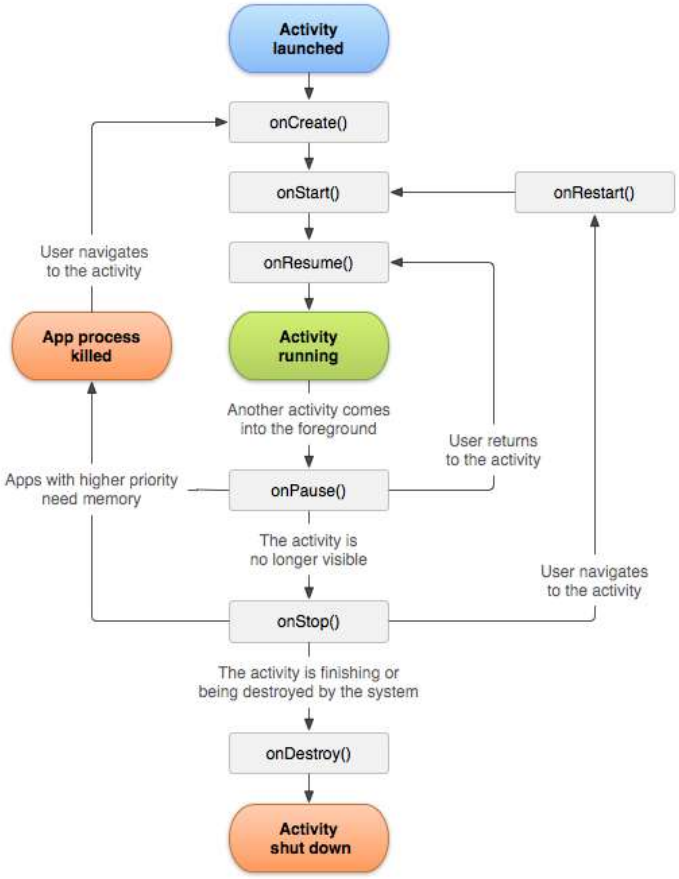
- La sauvegarde des données importantes quant à elle est effectuée dans la méthode « onPause ».

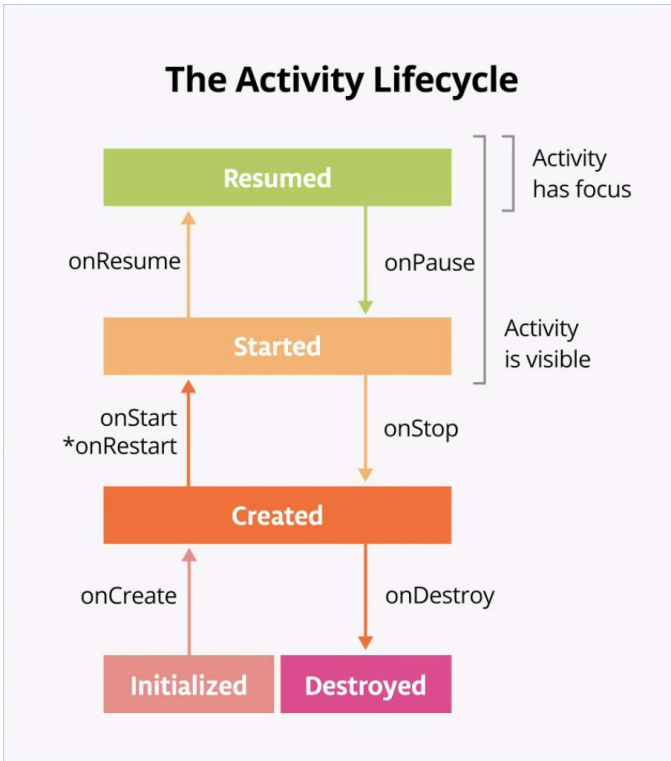
- En effet quand une activité n'a plus « le focus », elle est mise en pause d'où l'appel à la méthode « onPause ». Il est donc préférable de sauvegarder les informations critiques avant de se mettre en pause!

- Quand l'activité « onPause » termine sa période de repos, le système appelle la méthode « onResume ».

- La figure qui suit est disponible à cette adresse :

  http://developer.android.com/reference/android/app/Activity.html





- Examiner l'exemple « **C02 : Activite** ».
- Redéfinir les 6 méthodes. Ajouter un Log.i(TAG, « x ») pour chaque méthode.
- Interagir (envoyer un gsm) avec l'AVD et examiner le comportement de l'application et les différents appels aux 6 méthodes.

Description des états

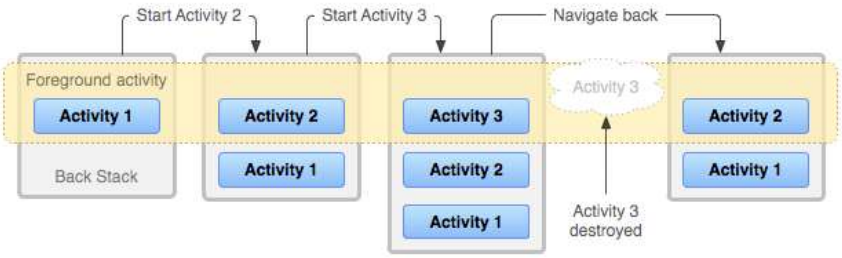
- État « start » : quand l'activité n'est pas encore référencée en mémoire, elle est dans un état « start ».
- État « Resume » / « Running » : une activité qui a le focus est dans un état « Running » à un instant « t ». Cette activité se trouve au sommet de la pile.
- État « Pause » : une activité qui n'a pas le focus (donc elle n'interagit pas avec l'utilisateur), mais reste visible à l'écran, est dans un état « Pause ». Elle maintient son état et reste attachée au gestionnaire de fenêtres. Cette activité peut quand même être détruite par le système si ce dernier se trouve en manque de mémoire par exemple.
- État « Stopped » : une activité qui n'est pas visible à l'écran, mais existe quand même en mémoire, est dans un état « Stopped ».
- État « Destroyed » : quand l'activité n'est plus en mémoire, elle est détruite. Ce phénomène arrive, car le gestionnaire d'activités décide qu'une telle activité n'est plus utilisée.

Pile

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

- Les activités sont ordonnées en utilisant la notion de pile.

- La pile est du type « LIFO » (Last In First Out).



- Quand une activité est dans un des états "onPause", "onStop" et "onDestroy", elle est dans une situation où elle peut être détruite si le système est en manque de mémoire.

- L'état « onPause » est le seul qui a la possibilité de s'exécuter complètement avant que l'activité ne soit détruite complètement.

- Ainsi donc, s'il y a des données critiques à écrire, c'est dans cet état qu'il faudra le faire.

2. Service

- Un service est une composante qui fonctionne en arrière-plan (tâche de fond).

- Un service n'est pas un composant graphique.

- Ce fonctionnement lui permet de réaliser une tâche d'une très longue durée sans une interaction avec l'utilisateur : écouter de la musique en arrière-plan, sauvegarder des données sur le réseau.

- Un service s'exécutant en arrière-plan est prioritaire qu'une application exécutée dans le même plan.

Un service peut avoir deux états :

- Started (Démarré) : quand une composante d'une application, par exemple une activité, a démarré le service en faisant appel à la méthode « startService() ».

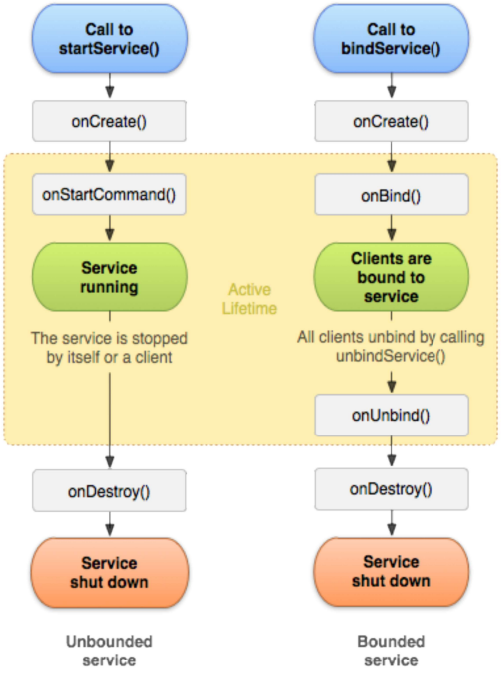
  Quand un service est démarré, il peut fonctionner en arrière-plan indéfiniment même si la composante qui s'est chargée de le démarrer a été détruite.

- Bound (Lié/Associé) : un service est dans cet état, quand un composant d'une application s'est lié à lui en appelant la méthode « bindService() ».

  Cet état offre une interface client-serveur permettant aux composantes d'interagir avec le service, envoyer des requêtes, obtenir des résultats, etc.

  Le diagramme suivant explique le cycle de vie d'un service et les différentes méthodes nécessaires pour interagir avec lui.

http://developer.android.com/guide/components/services.html



onStartCommand() : cette méthode est appelée par le système quand une activité a démarré le service en faisant appel à « startService() ». Cette méthode sert à exécuter le traitement que doit effectuer votre service. Si cette méthode est implémentée, il faut arrêter le service quand la tâche est terminée avec « *stopSelf()* » (commande interne au service) ou « *stopService()* » *(commande externe)*.

onBind() : cette méthode est appelée par le système quand une autre composante veut se lier à ce service en faisant appel à « bindService() ». Si cette méthode est implémentée, vous devez prévoir pour le client une interface lui permettant de communiquer avec le service en retournant un objet « IBinder ». Vous pouvez retourner une valeur « null » si vous voulez refuser toute liaison possible.

onUnbind() : cette méthode est appelée quand un client s'est déconnecté de l'interface fournie par le service.

onRebind() : cette méthode est appelée quand un nouveau client s'est connecté au service après que ce dernier ait reçu l'information comme quoi tous les clients se sont déconnectés de lui en utilisant la méthode « onUnbind() ».

onCreate() : cette méthode est appelée quand un service est créé ou qu'une liaison a été établie avec le service, pour la première fois. Cet appel permet d'exécuter une seule fois une série de commandes.

onDestroy() : cette méthode est appelée quand un service n'est plus utilisé. Cette méthode doit-être implémentée pour faire le ménage correctement (threads, écouteurs, etc.).

- Examiner l'exemple « **C02 : MonService** ».

3. BroadcastReceiver (Récepteur d'événements)

https://developer.android.com/guide/components/broadcasts.html

- C'est un récepteur spécialisé dont la tâche est d'écouter les messages transmis.

- Il va écouter certaines intentions en particulier (grâce à des filtres d'intention).

- Il est conçu pour recevoir des intentions afin d'effectuer une action.

- La tablette a une batterie faible. Android va envoyer un message « interne » pour informer les « BroadcastReceiver » associés à diverses applications de cet état. À eux d'effectuer les tâches nécessaires suite à ce message.

- Un « BroadcastReceiver » n'est pas un composant graphique.
- Pour qu'un « BroadcastReceiver » puisse fonctionner, vous devez d'abord le créer puis l'enregistrer.

Création :

- Un « BroadcastReceiver » que vous allez définir doit-être implémenté comme une sous-classe de « BroadcastReceiver ».
- Vous devez redéfinir la seule méthode disponible dans cette classe, il s'agit de « onReceive ».
- Cette méthode est appelée en cas de réception d'une intention.

Enregistrement :

- Un récepteur est déclaré par l'ajout d'un élément « receiver » dans le fichier « AndroidManifest.xml ».
- Les principaux « broadcast » sont définis dans la classe « intent ». Parmi eux « android.intent.action.airplane_mode ».
- Examiner les exemples « **C02 : ModeLocale** », « **C02 : AirplaneMode** » et « **C02 : MyPhoneReceiver** ».

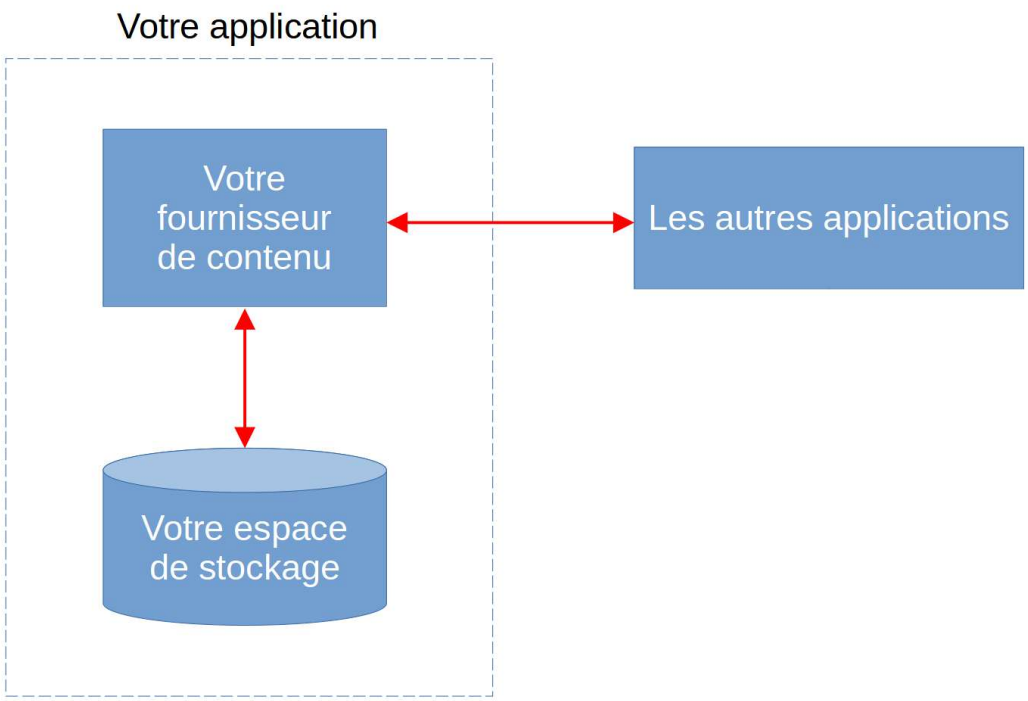
Un récepteur d'événements sur mesure

- Pour qu'une application génère et envoie par elle-même une intention (« intent »), il faut utiliser la méthode « sendBroadcast() ».
- Examiner l'exemple « **C02 : SimpleBroadcastReceiverApp** ».

4. Les Content Providers (fournisseurs de contenu)

https://developer.android.com/guide/topics/providers/content-providers.html

- Une base de données « SQLite » est embarquée dans Android.
- Une application peut créer une base de données pour regrouper un ensemble d'informations (exemple les contacts, une banque de photos, etc.).
- Si cette application veut rendre ses données accessibles, elle le fait à travers un mécanisme bien défini.
- Elle utilise pour cela un « fournisseur de contenu » en précisant le mode d'accès « lecture/écriture ».
- Les fournisseurs de contenu gèrent les données partagées par une ou plusieurs applications.
- Ils ont la charge de fournir ces informations ou bien de les modifier à la demande.



- C'est le seul moyen disponible pour partager des données entre différentes applications.

- La notion de fournisseurs de contenu est un concept évolué sur la gestion de données communes basée sur le principe de permissions d'accès.

- Ainsi donc, Android a créé une séparation claire entre votre application et les données exposées.

- Exemple : en installant une application Android sur votre appareil, ce dernier vous informe des accès réclamés par cette application, et sollicite votre accord avant de le faire.

- Nous allons revenir sur ces notions dans le chapitre « Persistance et partage des données ».