android

An Open Handset Alliance Project

Cycle de vie d'une application

O.Legrand
G.Seront

Cycle de vie d'une application

- La durée de vie d'un process n'est pas contrôlée uniquement par l'application ou l'utilisateur, mais aussi par le système (qui gère la mémoire)
- Avant d'arrêter un process, le système évalue son importance pour l'utilisateur et la place mémoire encore disponible.
- Les composants applicatifs (*Activity, Service, BroadcastReceiver*) n'ont pas la même importance, la même priorité pour le système.

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

Cycle de vie d'une application

- Ne pas utiliser ces composants applicatifs correctement pourraient induire le système à arrêter le process, alors que celui-ci réalise peut-être une tâche importante.
- Pour déterminer quels process arrêter, le système évalue leur importance.
- Cette importance est déterminée par :
  - le type de composant applicatif du process;
  - l'état dans lequel il se trouve.

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

Types de process

- Le système considère 5 types de process selon leur importance :
  - foreground process;
  - visible process;
  - service process;
  - background process;
  - empty process.

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

Foreground process

- Un process est qualifié de *foreground process* si c'est *:*
  - une *Activity* avec laquelle l'utilisateur est en train d'interagir:
    - sa méthode *onResume()* a été appelée;
  - un *BroadcastReceiver* qui s'exécute :
    - sa méthode *BroadcastReceiver.onReceive(…)* a été appelée;
  - un *Service* dont une des méthodes suivantes a été appelée :
    - *onCreate(), onStart(), onDestroyed()*
- Ce type de process ne sera en principe pas arrêté par le système.

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

Visible process

- Un process est qualifié de *visible process* si c'est une *Activity* qui est visible à l'écran, mais pas à l'avant-plan:
  - car une boîte de dialogue est affichée devant elle par exemple;
  - sa méthode *onPause()* a été appelée.
- Ce type de process ne sera en principe pas arrêté par le système, sauf si un *foreground process* nécessite de la mémoire et qu'il n'est pas possible d'en libérer autrement.

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

Service process

- Un process est qualifié de *service process* si c'est un *Service* qui a été lancé par l'appel de la méthode *context.startService()*.
- Exemples :
  - morceau mp3 en tâche de fond;
  - upload ou download de données.
- Le système peut arrêter un *service process* si un *foreground* ou *visible process* nécessite de la mémoire et qu'il n'est pas possible d'en libérer autrement.

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

Background process

- Un process est qualifié de *background process* si c'est une *Activity* qui n'est plus visible à l'écran:
  - sa méthode *onStop()* a été appelée.
- Ce type de process peut être arrêté par le système lorsqu'un process de plus grande importance nécessite de la mémoire et qu'il n'est pas possible d'en libérer en supprimant un *empty process*.

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

Empty process

- Un process est qualifié d'*empty process* s'il ne contient plus de composant applicatif actif.
- Ce type de process est gardé en mémoire afin d'accélérer son redémarrage éventuel.
- Si un process de plus grande importance requière de la mémoire, l'*empty process* le plus ancien sera supprimé par le système.

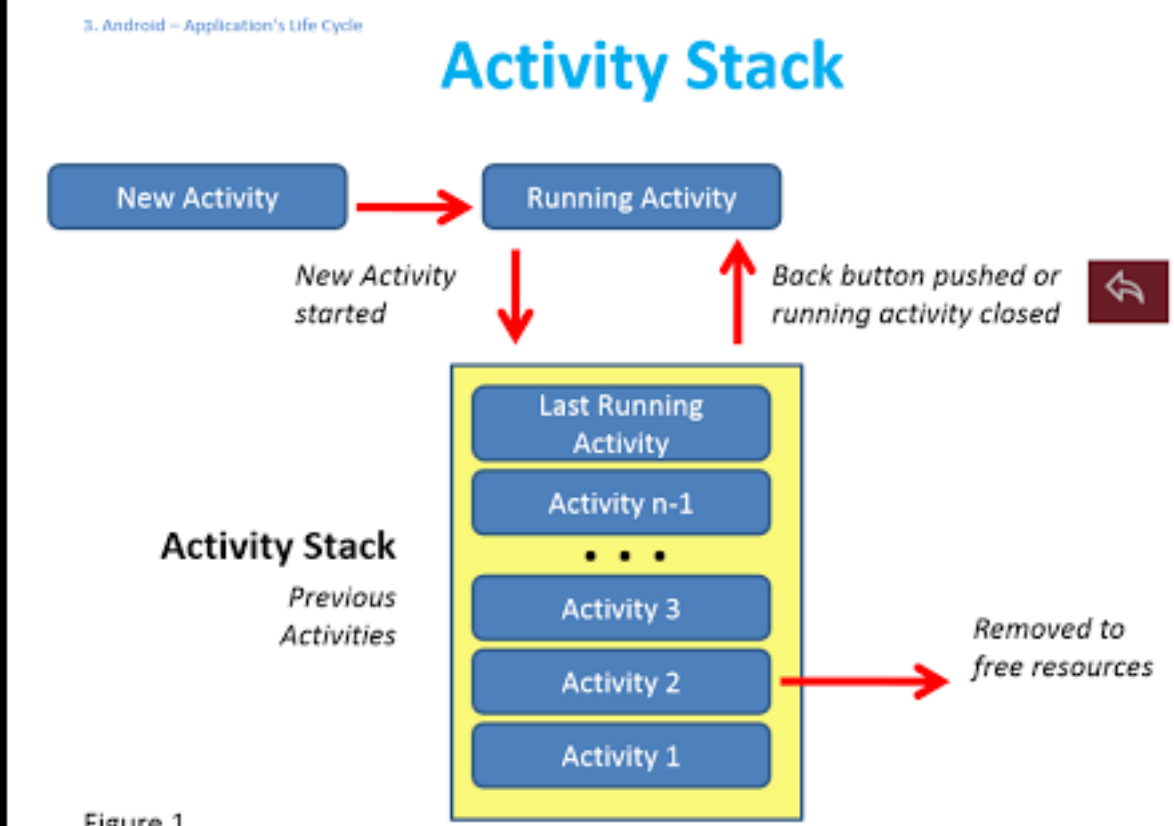
http://developer.android.com/reference/android/app/Activity.html

Cycle de vie d'une application

- Une application comprend un ou plusieurs composants applicatifs de type *Activity, BroadcastReceiver, Services* ou *ContentProvider.*
- A chaque activité est associé un écran (*View*).
- L'exécution d'une application est une suite d'écrans qui s'enchaînent.
- Chaque écran étant associé à une activité, passer à un autre écran, fait démarrer l'activité associée à cet écran.

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

Activity Stack



http://sosimpleandroid.blogspot.be/2013/05/activity-stack-in-android.html

Activity Stack

- Le système gère les activités à l'aide d'une pile d'activités (*activity stack*);
- Au démarrage d'une activité, il la place au sommet de la pile et la considère comme active.
- L'activité précédente se trouve en dessous dans la pile et ne reviendra à l'avant-plan que lorsque l'activité active est quittée.

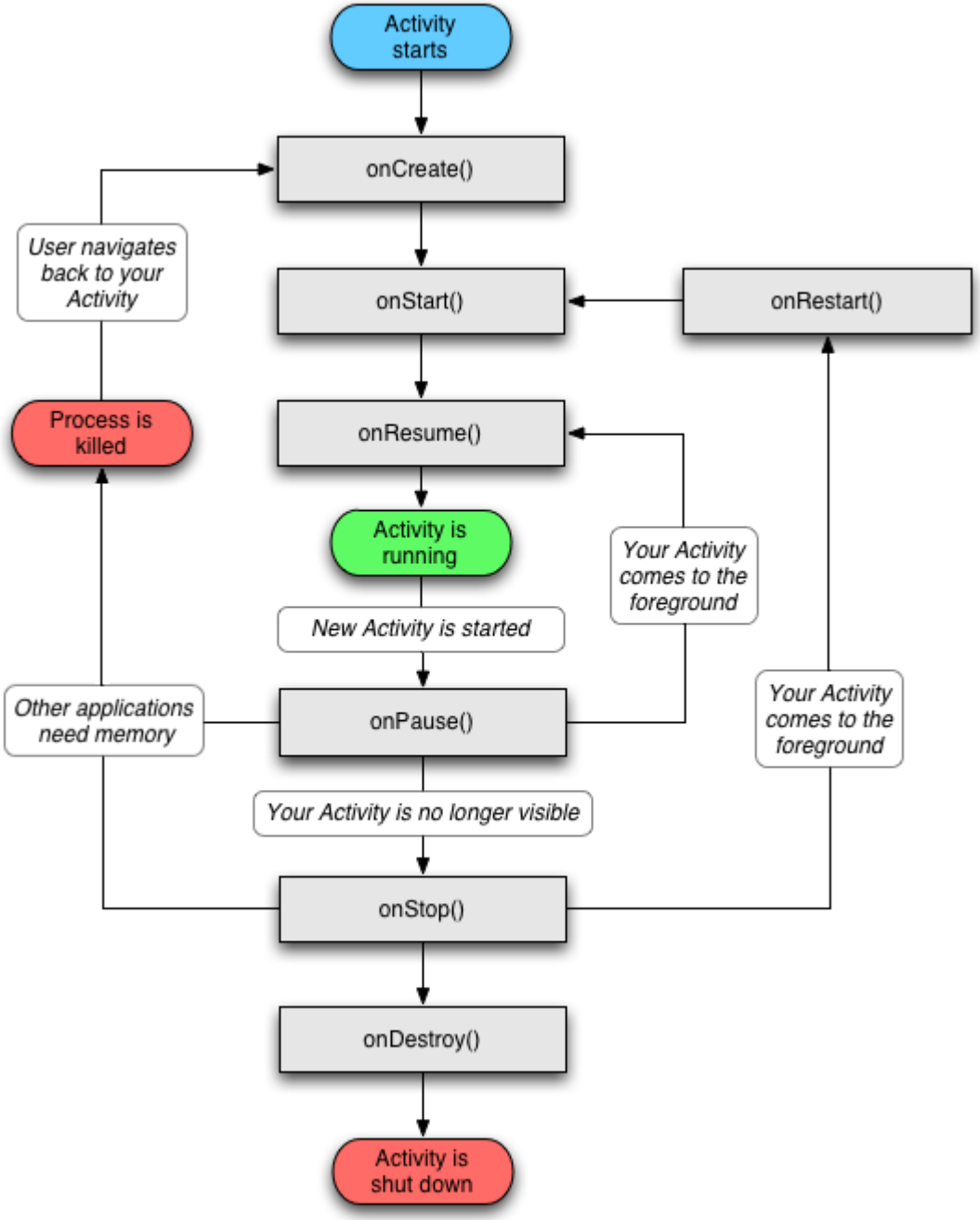
http://developer.android.com/reference/android/app/Activity.html

Cycle de vie d'une *Activity*

- Une activité peut se trouver dans un des 4 états suivants :
  - active (*running*):
    - visible à l'avant-plan de l'écran;
  - en pause:
    - a perdu le focus, mais encore visible;
  - stoppée:
    - n'est plus visible;
  - détruite:
    - le système l'a supprimée de la mémoire.

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

Cycle de vie d'une *Activity*



http://developer.android.com/reference/a
l

Activity - onCreate(Bundle)

- Méthode appelée à la création de l'activité.
- Généralement :
  - on y initialise les variables;
  - on y associe l'écran
    - setContentView(View);
  - on y obtient les références des composants graphiques (*widgets*) avec lesquels l'activité va interagir;
    - findViewById(int);

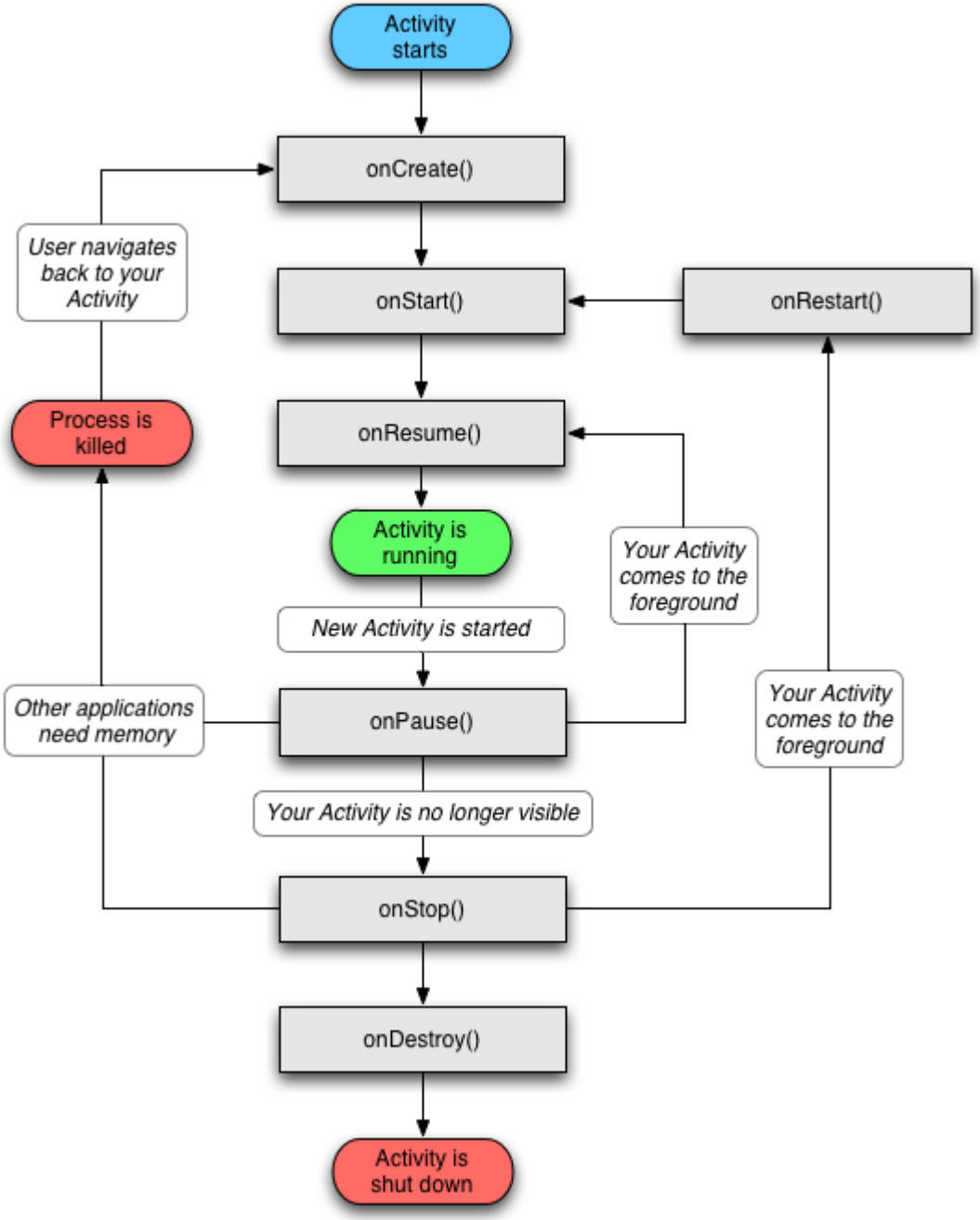
http://developer.android.com/reference/android/app/Activity.html

Activity - onCreate(Bundle)

- Appelée
  - à la création de l'activité,
  - ou suite à un redémarrage après avoir été stoppée (plus visible) et supprimée de la mémoire.
- On récupère, éventuellement, l'état précédent de l'activité (*Bundle*), s'il a été sauvé par la méthode *onSaveInstanceState(Bundle).*
- Suivie de *onStart();*

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

Cycle de vie d'une *Activity*



http://developer.android.com/reference/a
l

Activity – onRestart()

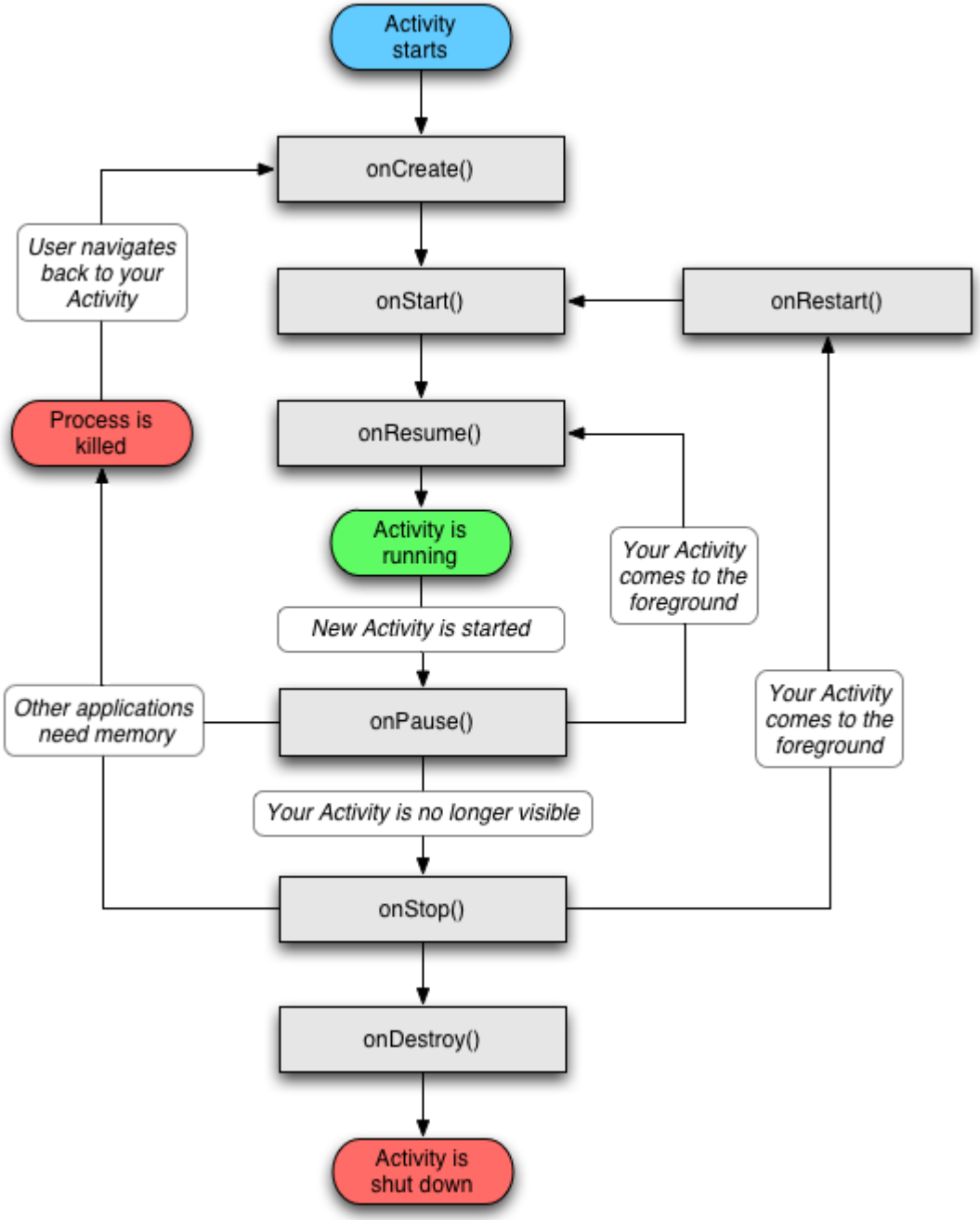
- Méthode appelée quand l'activité redevient visible.
- Toujours suivie de *onStart();*

Activity – onStart()

- Méthode appelée quand l'activité devient visible.
- Toujours suivie de *onResume();*

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

Cycle de vie d'une *Activity*



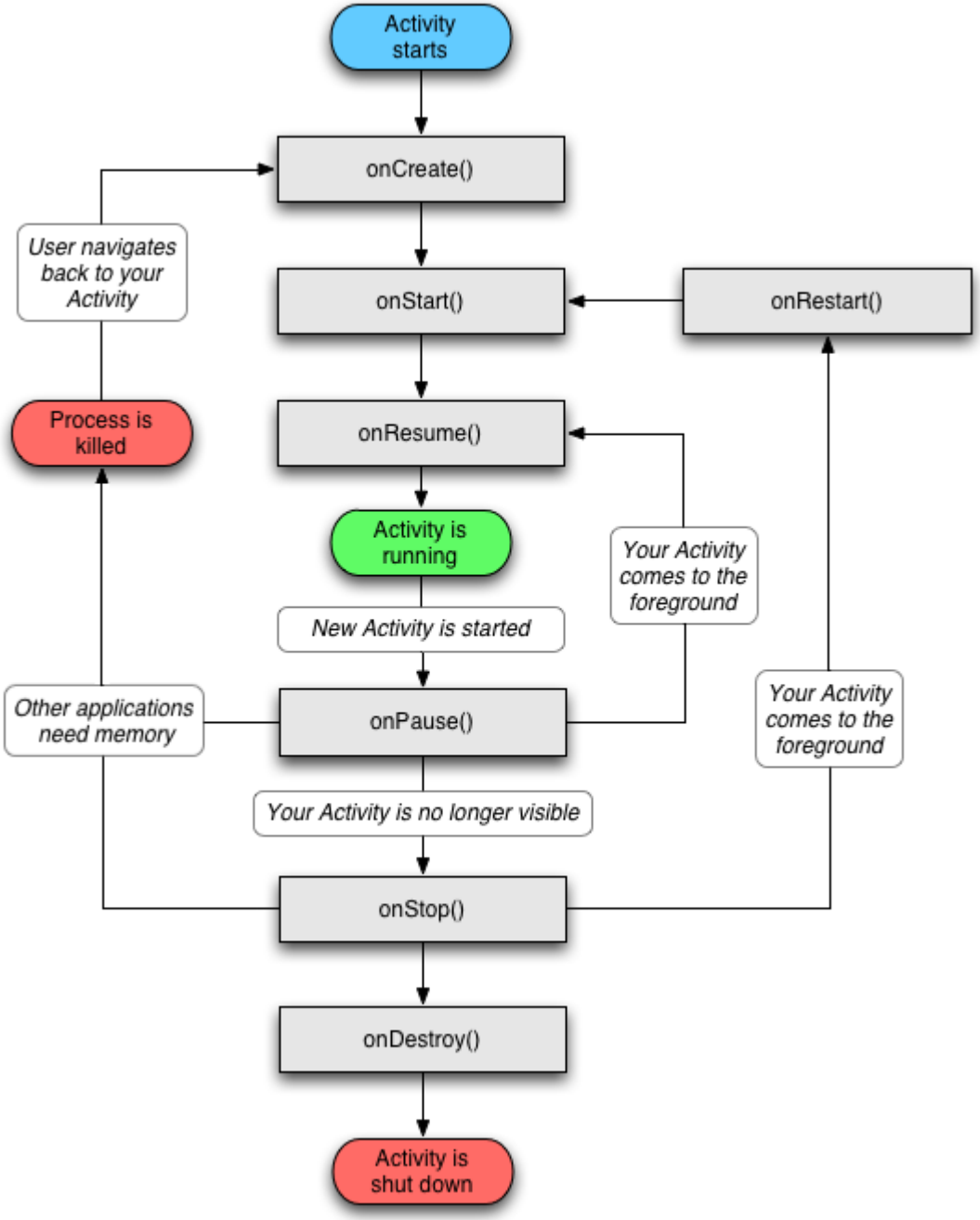
http://developer.android.com/reference/a
l

Activity – onResume()

- Méthode appelée juste avant que l'utilisateur ne puisse interagir avec l'activité.
- L'activité devient active.
- Elle est placée au sommet de la pile d'activités.
- Suivie de *onPause();*

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

Cycle de vie d'une *Activity*



http://developer.android.com/reference/a
l

Activity – onPause()

- Méthode appelée après *onResume()*.
- Méthode appelée quand l'activité n'est plus active car une autre activité passe à l'avant-plan.
- L'activité est toujours visible, mais n'a plus le focus.
- Il faut :
  - sauver les données non encore sauvegardées;
  - stopper les animations et autres composants applicatifs qui consomment du temps cpu.

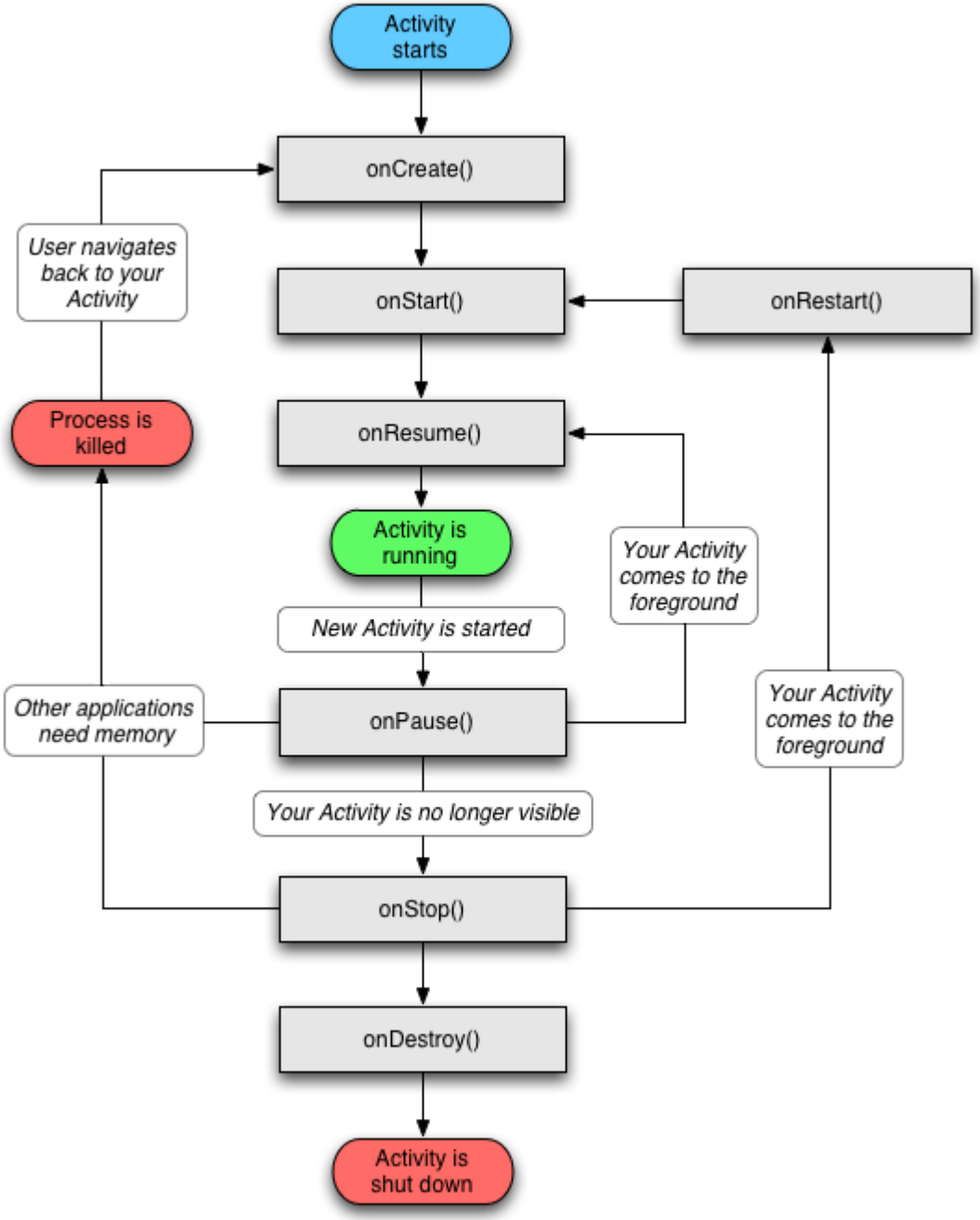
http://developer.android.com/reference/android/app/Activity.html

Activity – onPause()

- Il faut quitter rapidement cette méthode car l'activité qui devient active attend;
- Suivie de :
  - *onResume()* si elle repasse à l'avant-plan;
  - *onStop()* si elle devient totalement invisible.

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

Cycle de vie d'une *Activity*



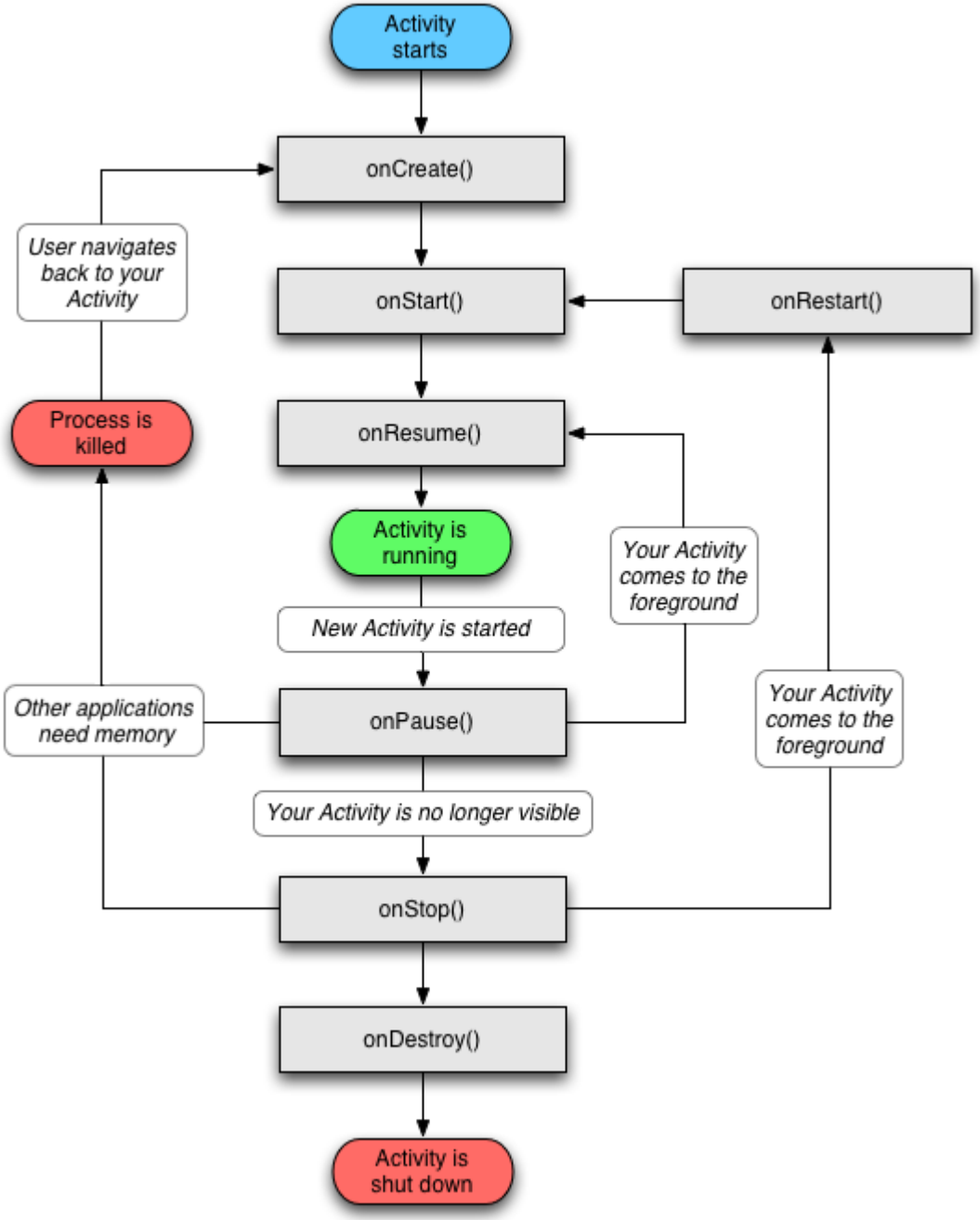
http://developer.android.com/reference/a
l

Activity – onStop()

- Méthode appelée quand l'activité n'est plus visible :
  - soit, une autre activité est devenue active (à l'avant-plan) et la recouvre entièrement;
  - soit, le process est arrêté par le système qui requière de l'espace mémoire.
- Suivie de :
  - *onRestart()* si l'activité revient à l'avant-plan;
  - *onDestroy()* si l'activité est arrêtée;

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

Cycle de vie d'une *Activity*



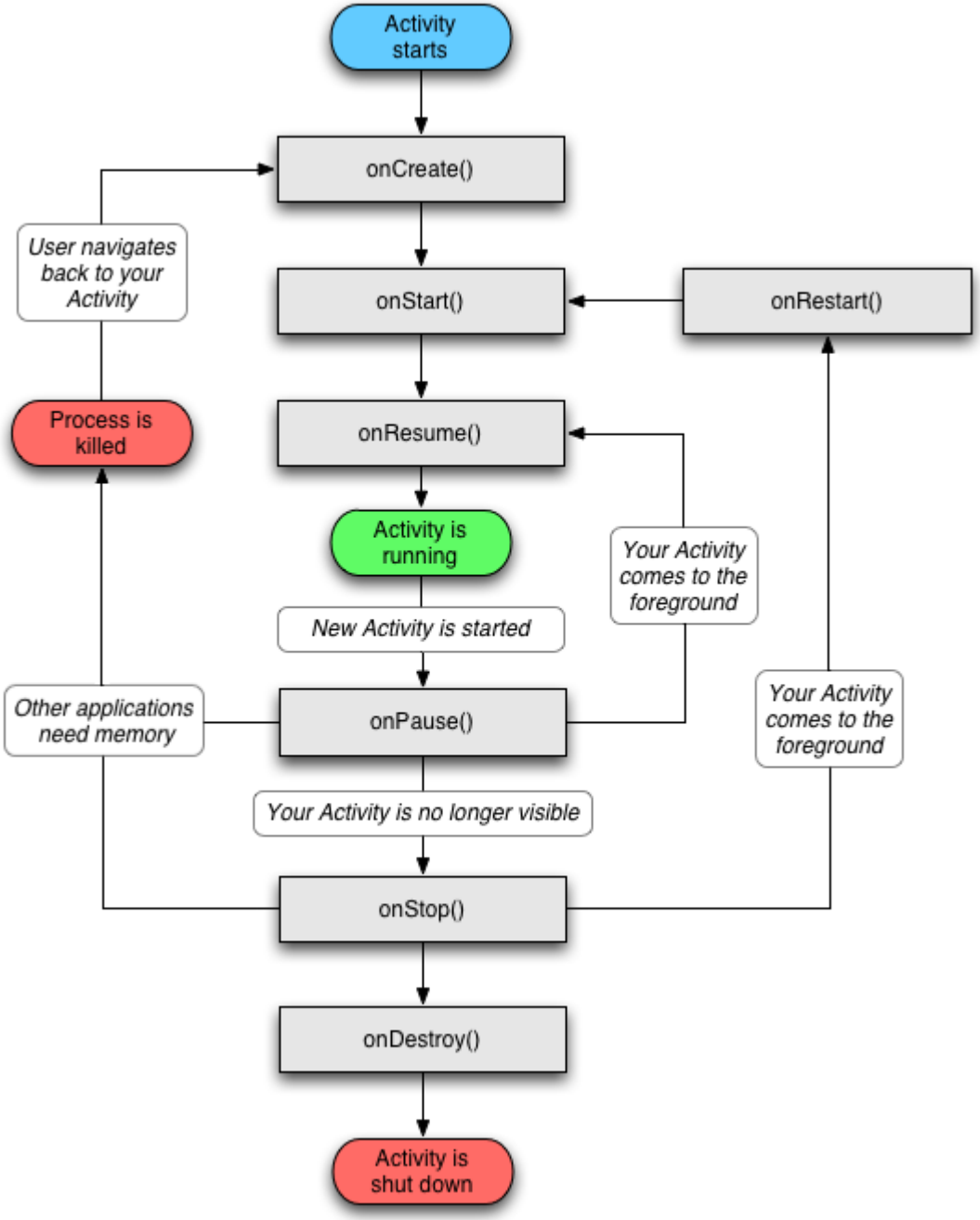
http://developer.android.com/reference/a
l

Activity – onDestroy()

- Dernière méthode de l'activité appelée :
  - quand le système arrête le process afin de libérer de l'espace mémoire;
  - ou quand la méthode *finish()* a été appelée;
- La méthode *isFinishing()* permet de distinger l'un ou l'autre scénario.

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

Cycle de vie d'une *Activity*



http://developer.android.com/reference/a
l

Cycle de vie d'une application

- Pour lancer une activité :
  - *startActivity(Intent);*
- Pour lancer une activité dont on attend un résultat :
  - *startActivityForResult(Intent, int);*
  - lors du retour, la méthode *onActivityResult(int,int,Intent)* de l'activité appelante sera appelée;
- Pour renvoyer un résultat :
  - *setResult(int)* ou *setResult(int, Intent);*
- Pour arrêter l'activité : *finish();*

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

Fragment

- Une activité peut-être subdivisée en plusieurs parties (depuis Android 3.0)
- Chaque partie est appelée un *Fragment*
- Chaque *Fragment possède :*
  - *sa propre UI (mais pas obligatoire)*
  - *son propre cycle de vie*
  - *gère lui-même les inputs de l'utilisateur*
- Un *Fragment* peut être utilisé dans différentes activités

developer.android.com/reference/android/app/Fragment.html

Vidéo

- Vidéos concernant les applications:
  - Androidology - Application Lifecycle
  - A first hand look at building an Android application

Shared Preferences

- Les *Shared Preferences* permettent de sauver d'une session à l'autre :
  - les préférences des utilisateurs;
  - les paramètres de l'application;
  - l'état de l'interface utilisateur;
- Par défaut, elles ne sont partagées que par les composants de l'application *(MODE_PRIVATE)*;
- Elles seront accessibles par les composants des autres applications si : *MODE_WORLD_READABLE* ou *MODE_WORLD_WRITEABLE* ou *MODE_MULTI_PROCESS.*

developer.android.com/guide/topics/data/data-storage.html#pref

Sauvegarder les préférences

- Pour créer ou modifier une *Shared Preferences* :
  - appelez la méthode *getSharedPreferences()* sur le context en lui passant le nom de la préférence en paramètre;
  - demandez la référence de l'éditeur : *edit()* ;
  - appelez les méthodes *put<type>* sur l'éditeur pour ajouter les données (clef-valeur);
  - exécutez un *commit()* pour sauvegarder.

developer.android.com/guide/topics/data/data-storage.html#pref

Sauvegarder les préférences

- Exemple :

```
private void sauverLesPreferences() {
    // obtenir la référence des préférences partagées
    SharedPreferences preferences = this.getSharedPreferences(
            "mesPreferences", MODE_PRIVATE);

    // récupère un éditeur pour modifier les préférences
    Editor editor = preferences.edit();

    // modifie les préférences
    editor.putBoolean("unBooléen", true);
    editor.putFloat("unRéel", 4.5f);
    editor.putInt("unEntier", 8);
    editor.putString("uneChaine", "Salut toi!");

    // enregistre les préférences
    editor.commit();
}
```

developer.android.com/guide/topics/data/data-storage.html#pref

Récupérer les préférences

- Pour récupérer les préférences sauvegardées :
  - appelez la méthode *getSharedPreferences()* sur le context en lui passant le nom de la préférence en paramètre;
  - appelez les méthodes *get<type>* pour extraire les valeurs sauvegardées;
  - Passez aux getters 2 paramètres :
    - une clef et une valeur par défaut

developer.android.com/guide/topics/data/data-storage.html#pref

Récupérer les préférences

- Exemple :

```
private void chargerLesPreferences() {
    // obtenir la référence des préférences partagées
    SharedPreferences preferences = this.getSharedPreferences(
            "mesPreferences", MODE_PRIVATE);

    // récupère les valeurs sauvegardées
    boolean unBooléen = preferences.getBoolean("unBooléen", false);
    float unRéel = preferences.getFloat("unRéel", 0);
    int unEntier = preferences.getInt("unEntier", 0);
    String uneChaine = preferences.getString("uneChaine", "vide");

    Log.i("MonLog", "" + unBooléen + "   " + unRéel + "   " + unEntier
            + "  " + uneChaine);
}
```

developer.android.com/guide/topics/data/data-storage.html#pref

Préférences d'une activité

- Toute activité possède sa propre *Shared Preferences* unique qui porte son nom;
- Par défaut, elle ne la partage qu'avec les autres composants de l'application *(MODE_PRIVATE)*;
- Elle sera accessible par les composants des autres applications si : *MODE_WORLD_READABLE ou MODE_WORLD_WRITEABLE ou MODE_MULTI_PROCESS.*
- Pour y accéder : *getPreferences() ;*
- Elle s'utilise comme toute préférence partagée.

developer.android.com/guide/topics/data/data-storage.html#pref

Préférences d'une activité

```
private void sauverLesPreferencesDeActivité() {
    // obtenir la référence des préférences de l'activité
    SharedPreferences preferences = this.getPreferences(MODE_PRIVATE);

    // récupère un éditeur pour modifier les préférences
    Editor editor = preferences.edit();

    // modifie les préférences
    editor.putString("uneChaine", "préférence de l'activité");

    // enregistre les préférences
    editor.commit();
}

private void chargerLesPreferencesDeActivité() {
    // obtenir la référence des préférences de l'activité
    SharedPreferences preferences = this.getPreferences(MODE_PRIVATE);

    // récupère la valeur sauvegardée
    String uneChaine = preferences.getString("uneChaine", "vide");

    Log.i("MonLog", "" + uneChaine);
}
```

developer.android.com/guide/topics/data/data-storage.html#pref