Android – Gestion des Threads

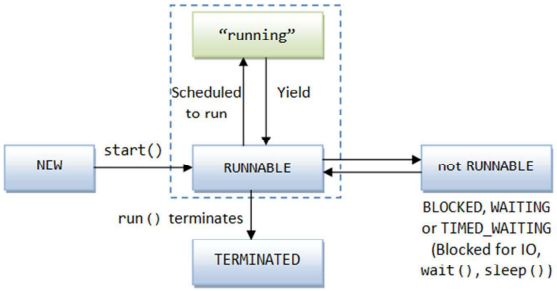
Emmanuel Conchon
(emmanuel.conchon@unilim.fr)

Introduction et rappels

- Les threads sont souvent appelés des processus légers
  - Comme les processus ils proposent un environnement d'exécution mais demandent moins de ressources que les processus
- Les threads existent à l'intérieur d'un processus
  - Tous les threads ont au moins un processus auquel ils sont rattachés
  - Si plusieurs threads sont attachés au même processus, ils vont alors en partager l'environnement (mémoire, processeur etc.)
    - Utile mais peut entraîner quelques difficultés pour l'accès aux ressources

Le cycle de vie des Threads

- Un thread va donc suivre un cycle de vie relativement simple



Rappels Java - Création de threads

- Pour créer un thread il suffit d'instancier la classe `Thread`
- La classe Thread implémente `Runnable` et fournit une méthode `run()` vide. Il faut donc définir le fonctionnement de la méthode `run()` lorsque l'on créé un thread
- Deux solutions:
  - Hériter de Thread et surcharger la méthode `run()` (méthode simple)

```
public class ThreadCoucou extends Thread {

   public void run() {
      System.out.println("Bonjour de la part du thread");
   }

   public static void main(String[] args) {
      Thread monThread = new ThreadCoucou();
       monThread.start();
   }
}
```

Rappels Java - Création de thread

- Fournir un objet implémentant `Runnable` à votre objet `Thread`

```
public class CoucouRunnable {

   public static void main(String[] args){
      Thread monThread = new Thread(new Runnable(){
         public void run(){
            System.out.println("Bonjour de la part du thread");
         }
      });
      monThread.start();
   }
}
```

```
public class CoucouRunnable {

   public static void main(String[] args){
      Thread monThread = new Thread( () -> {
         System.out.println("Bonjour de la part du thread");
      });
      monThread.start();
   }
}
```

- Quelle méthode choisir ?
  - Il n'y a pas d'héritage multiple en JAVA donc il n'est pas toujours possible d'hériter de Thread
  - En général la solution `Runnable` est privilégiée

Rappels Java - Suspendre l'exécution d'un thread

- `Thread.sleep(…)` permet de suspendre l'exécution du thread courant pendant une durée spécifiée en paramètre de la méthode
  - Le fait de suspendre l'exécution libère des ressources pour d'autre thread par exemple

```
public class TestSleep {

   public static void main(String[] args) throws InterruptedException {
      for (int i=0; i<10; i++){
         // Attente de 1s
         Thread.sleep(1000);
         System.out.println("Programme lancé depuis "+i+" secondes");
      }
   }
}
```

- Depuis Java 5 cette instruction peut être replacée par
  - `TimeUnit.SECONDS.sleep(1);`

Rappels Java - Les interruptions

- Une interruption est un événement qui va avoir pour conséquence d'interrompre l'activité du thread pour qu'il puisse faire autre chose

```
public class TestSleep {

   public static void main(String[] args) {
      for (int i=0; i<10; i++){
         // Attente de 1s
         try{
            Thread.sleep(1000);
         }
         catch (InterruptedException e){
            // on est interrompu donc on stop le programme
            return;
         }
         System.out.println("Programme lancé depuis "+i+" secondes");
      }
   }
}
```


Rappels Java - Les interruptions & les verrous

- Les interruptions ne sont pas forcément des exceptions
- Java dispose d'un mécanisme permettant à un Thread d'en interrompre un autre
  - La méthode `interrupt()` de la classe `Thread` :
    - `public void interrupt()`
- Possibilité de connaître l'état du Thread avec la méthode statique `interrupted()`
  - `public static boolean interrupted()`
- Les verrous sont réalisables à l'aide des objets de type Lock ou de sémaphores
  - Les Locks doivent être privilégiés par rapport aux blocs synchronized()
- Les threads peuvent également être synchronisés

Introduction et rappels

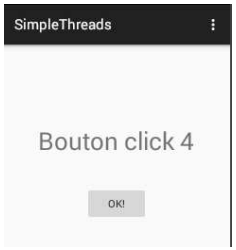
- Android repose sur un noyau Linux
  - Chaque application fonctionne dans un processus dédié
  - Chaque application dispose de ses propres ressources en terme de mémoire et de processus
- Un « garbage collector » similaire à celui de Java fonctionne en arrière plan
  - Toute activité inactive depuis 5s est éliminée de la mémoire
- Un Thread est dédié à l'interface graphique: l'UIThread
  - Il gère les rafraichissements d'écran et les évènements
  - La principale difficulté va donc consister à permettre l'exécution en parallèle de plusieurs taches sans bloquer ce thread graphique

Exemple

- Prenons une application simple permettant d'afficher un message toute les secondes à l'écran
  - Une solution serait:

```
try{
    for (int i=0; i<5;i++){
        Thread.sleep(1000);
        TextView textView = (TextView) findViewById(R.id.textView);
        textView.setText("Bouton click "+i);
    }
}catch(InterruptedException e){
    e.printStackTrace();
}
```

  - Pbm: L'application ne répond plus pendant 5s et affiche directement le dernier message

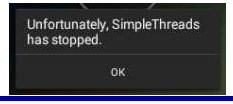


Exemple

- Autre solution: embarquer le tout dans un thread dédié

```
Thread thread = new Thread(new Runnable() {
    @Override
    public void run() {
        try{
            for (int i=0; i<5;i++){
                Thread.sleep(1000);
                TextView textView = (TextView) findViewById(R.id.textView);
                textView.setText("Bouton click " + i);
            }
        }catch(InterruptedException e){
            e.printStackTrace();
        }
    }
});
thread.start();
```

- Bonne nouvelle, l'application répond bien une fois que le thread est lancé
- Pbm: Il ne s'affiche rien à l'écran, le TextView n'est pas modifié…et nous avons un beau message d'erreur qui s'affiche

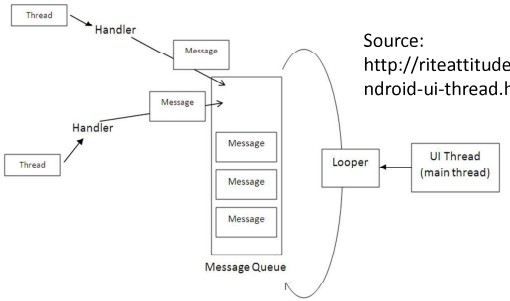


Dialoguer avec l'UIThread

- L'UIThread est dédié à l'affichage des éléments et à la gestion des interactions et il ne doit faire que ça !
  - Le corollaire étant qu'aucun autre thread ne peut intervenir sur l'affichage
- Toute autre opération coûteuse en temps d'exécution doit être placée dans un thread distinct
  - Si cet autre Thread a besoin d'interagir avec l'UIThread, il doit utiliser des accès prédéfinis
- Dans Android, la communication avec l'UIThread a été implémentée de la manière suivante:
  - Une file de messages contenant les instructions à exécuter par l'UIThread est proposée
  - Les autre threads peuvent ajouter des messages dans la file
  - Seul l'UIThread peut extraire les messages pour les traiter

Dialoguer avec l'UIThread

- Les activités proposent deux méthodes pour interagir avec l'UIThread
  - En utilisant une séquence d'action avec runOnUIThread(…) qui prend en paramètre un Runnable
    - Si le thread courant est l'UIThread, l'action est exécutée immédiatement sinon elle est mise dans la file d'évènements à traiter



Source: http://riteattitude.blogspot.fr/2011/11/android-ui-thread.html

    - En ajoutant explicitement un Runnable à la file de messages à travers la méthode post d'une vue (View) ou d'un Handler

Exemple

- Si nous reprenons notre exemple précédent avec runOnUiThread()

```
Thread thread = new Thread(new Runnable() {
    @Override
    public void run() {
        try{
            for (int i=0; i<5;i++){
                Thread.sleep(1000);
                final int val = i;
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        TextView textView = (TextView) findViewById(R.id.textView);
                        textView.setText("Bouton click " + val);
                    }
                });
            }
        }catch(InterruptedException e){
            e.printStackTrace();
        }
    }
});
thread.start();
```


Exemple

- Avec un Handler

```
Thread thread = new Thread(new Runnable() {
    @Override
    public void run() {
        try{
            for (int i=0; i<5;i++){
                Thread.sleep(1000);
                final int val = i;
                handler.post(new Runnable() {
                    @Override
                    public void run() {
                        TextView textView = (TextView) findViewById(R.id.textView);
                        textView.setText("Bouton click " + val);
                    }
                });
            }
        }catch(InterruptedException e){
            e.printStackTrace();
        }
    }
});
thread.start();
```

- Le handler est déclaré en variable d'instance et instancié dans onCreate()

Dialoguer avec l'UIThread - Asynctask

- Les solutions précédentes sont pratiques tant que le code à exécuter par le thread est simple
- Lorsque les interactions avec l'UIThread augmentent il faut mettre en place un Handler au niveau du thread pour recevoir les messages de l'UIThread
- Lorsque la tâche devient plus complexe, il vaut mieux passer par Asynctask
  - Tache asynchrone en arrière plan qui publie ses résultats dans l'UIThread
    - A privilégier pour les taches devant durer quelques secondes
  - Elle est définie par trois paramètres génériques
    - Params
    - Progress
    - Results

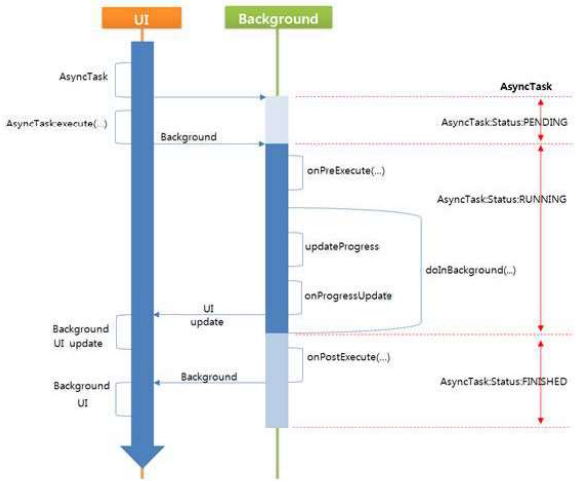
Dialoguer avec l'UIThread - Asynctask

- Elle repose essentiellement sur 4 étapes
  - onPreExecute() qui est invoquée sur l'UIThread avant que la tache ne soit exécutée
    - ★ *Permet d'initialiser des composants graphiques comme une barre de progression par exemple*
  - doInBackground(Params…) qui est invoquée dans un thread en arrière plan et qui permet d'effectuer des traitements longs
    - ★ *Le résultat qui est retourné à cette étape est ensuite fourni à l'étape suivante*
  - onProgressUpdate(Progress…) traite le retour de l'étape précédente et permet de fournir un feedback à l'utilisateur par exemple
  - onPostExecute(Result) qui est invoquée à la fin du traitement en tache de fond
    - ★ *Le résultat de l'exécution est passé en paramètre de cette méthode*

Dialoguer avec l'UIThread - Asynctask

- Il faut donc hériter d'Asynctask et redéfinir les méthodes précédentes pour réaliser une tâche asynchrone
  - Au minimum, il faut surcharger doInBackground()
- Utilisation d'une Asynctask
  - Une instance Asynctask doit être créée depuis l'UIThread
  - Utilisation de la méthode execute(Params…) qui permet de déclencher les 4 étapes précédentes
    - Il n'y a jamais d'appel explicite à onPreExecute() , doInBackground(Params…), onProgressUpdate(Progress…) ou onPostExecute(Result)

Dialoguer avec l'UIThread - Asynctask



Exemple

- Toujours le même exemple avec une AsyncTask

```
private class MyAsyncTask extends AsyncTask<Void, Integer, Integer>{
    @Override
    protected Integer doInBackground(Void... params) {
        try{
                for (int i=0; i<5;i++){
                    Thread.sleep(1000);
                    publishProgress(i);
                }
            }catch(InterruptedException e){
                e.printStackTrace();
            }
        return null;
    }

    protected void onProgressUpdate(Integer... result){
        TextView textView = (TextView) findViewById(R.id.textView);
        textView.setText("Bouton click "+result[0]);
    }
}

public void generateToast(View view) {
    new MyAsyncTask().execute();
}
```

Signifie qu'il y a entre 0 et n entiers