

Images lazy loading

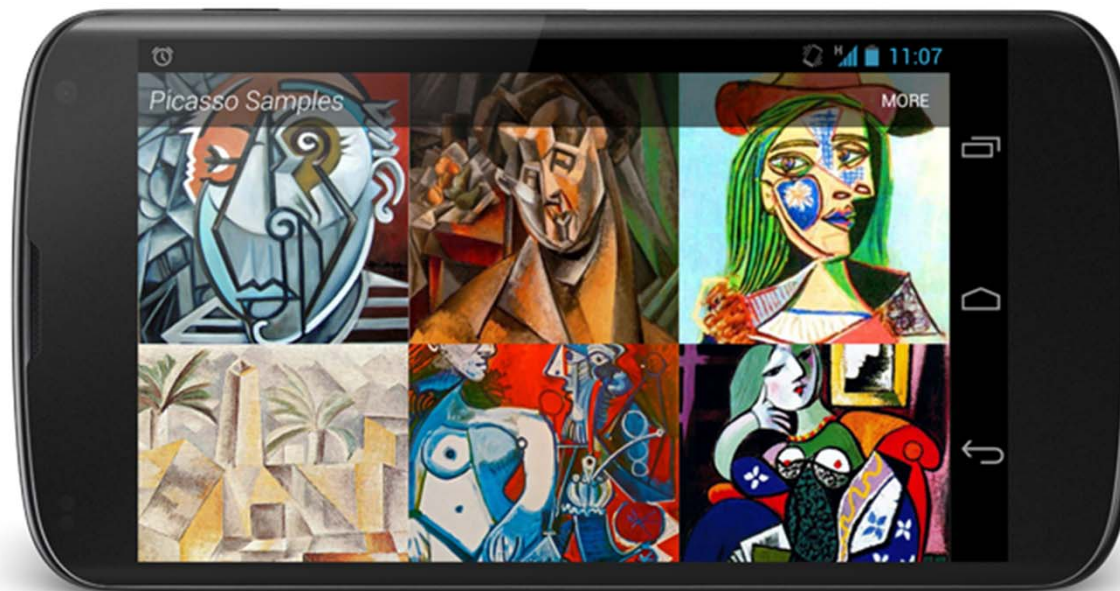
Série d'*Exercices* 2

Fabien Dutoit

SYM – Systèmes mobiles

Problématique

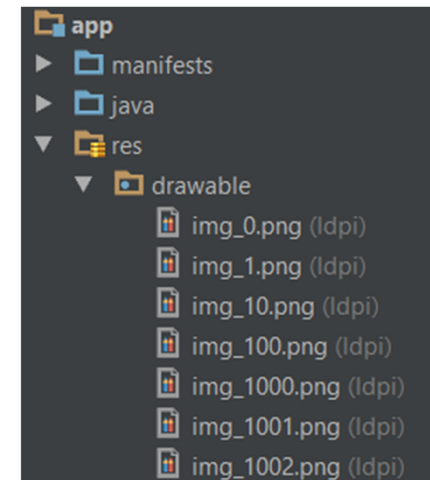
- *On souhaite réaliser une application permettant de consulter une série d'images*
- *On veut donner la possibilité à l'utilisateur de consulter la collection complète dans une ListView (ou une GridView)*



<http://square.github.io/picasso>

Solution «triviale»

- *Les images sont disponibles dans les ressources de l'application*
- *La seule «difficulté» sera de pouvoir retrouver l'identifiant de la ressource depuis son nom:*



```
int resId = getResources().getIdentifier(imageName, "drawable", getPackageName());
```

- *On peut ensuite attribuer la ressource à l'ImageView*

```
img.setImageResource(resId);
```

Solution «triviale»

Les principaux défauts de cette solution sont que toutes les images sont contenues dans l'apk (poids...) et que si l'on souhaite changer une seule image dans la collection, il faut distribuer une nouvelle version de l'application (validation de plusieurs jours chez Apple, tous vos utilisateurs ne vont pas mettre à jour, nécessitera le re-téléchargement de la collection complète, etc.)*

Images sur un serveur

- *Nous allons stocker les images sur un serveur*
Par ex: <http://sym.iict.ch/img/2345.png> 2345
- *Ce serveur démo va générer les images nécessaires entre 0 et 10'000*
- *L'application téléchargera les images dont elle a besoin – On va voir 3 manières de le réaliser*
- *L'application gardera un cache local des images pour ne pas avoir à les re-télécharger à chaque affichage*

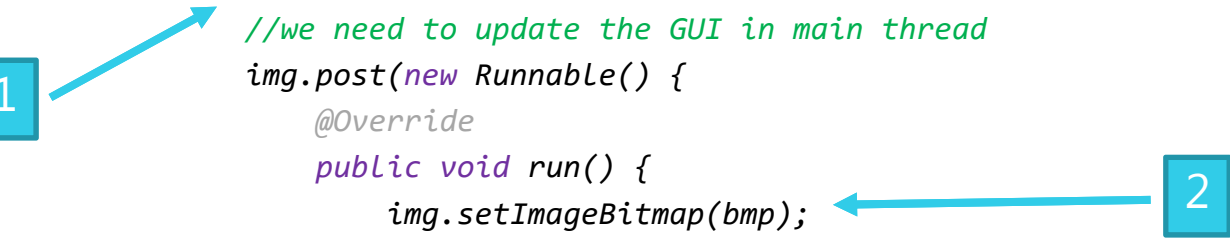
Threads

- *La méthode getView de l'Adapter s'exécutant dans le UIThread, nous ne devons pas y réaliser des accès réseaux ou de lecture/écriture de fichiers*
- *Nous devons utiliser un thread séparé*
- *La mise à jour de l'interface graphique (l'affichage de l'image) devra en revanche être réalisée dans le UIThread...*

Threads

1. Est exécuté dans le nouveau Thread
2. Sera exécuté dans le UIThread

```
new Thread(){  
    @Override  
    public void run() {  
        try {  
            final Bitmap bmp = BitmapFactory.decodeStream(url.openConnection().getInputStream());  
            //we need to update the GUI in main thread  
            img.post(new Runnable() {  
                @Override  
                public void run() {  
                    img.setImageBitmap(bmp);  
                }  
            });  
        } catch (Exception e) {  
            Log.e(TAG, "Error while loading image", e);  
        }  
    }  
}.start();
```



Threads – Problèmes

- *Au premier affichage, un thread est lancé pour chaque image qui sera visible à l'écran - 27 dans notre cas*
- *L'affichage lag si on scroll rapidement*
- *Les vues de la GridView étant recyclées, on peut se retrouver avec plusieurs Thread en cours qui tenteront de modifier la même vue...*

SYM_Exercise2		
Simple thread		
0	1	2
0	1	2
3	4	5
3	4	5
6	7	8
6	7	8
9	10	11
9	10	11
12	13	14
12	13	14
15	16	17
15	16	17
18	19	20
18	19	20
21	22	23
21	22	23
24	25	26
24	25	26

960	961	962
882	883	884

Threads – Solutions ?

- *On peut être tenté de «tuer» les threads qui ne sont plus nécessaires. En Java cela n'est pas possible directement, il faut les interrompre. Une InterruptedException est lancée si le thread était en pleine opération de communication HTTP ou de lecture/écriture disque. Cela laissera le cache dans un état imprévisible. Il faut gérer le cas (supprimer le fichier du cache). C'est à nous d'attraper l'exception dans la méthode run() et de la terminer proprement*
- *Il faudra aussi s'assurer que l'UI ne sera pas mise-à-jour après l'interruption...*

Asynctask

- *Toutes les AsyncTask sont exécutées par le système dans un thread unique*
- *l'UI aura moins tendance à «laguer»*
- *Les méthodes onPreExecute() et onPostExecute() sont exécutées dans le UIThread, on peut mettre les vues à jour. La méthode doInBackground() est exécutée dans un thread séparé, on peut effectuer des opérations I/O*
- *Il faudra aussi un mécanisme nous permettant de stopper les tâches «obsolètes» (tâche lancée pour mettre à jour une vue qui a déjà été recyclée)*

AsyncTask - Annulation

- *On ne peut pas interrompre une AsyncTask, on peut l'annuler avec la méthode `cancel()`, ce qui a pour effet principal de mettre un flag à true*
- *Vous devrez donc vérifier régulièrement dans votre méthode `doInBackground()` si la tâche a été annulée et traiter le cas proprement*

```
if(isCancelled()) { ...; return; }
```
- *On est confronté aux mêmes problèmes qu'avec les Threads, mais c'est certainement plus simple de les gérer*

Threads – AsyncTasks - Comparaison

- *Les AsyncTask sont plus simples à prendre en main*
- *Les AsyncTask étant exécutées les unes après les autres, on verra apparaître les images une à une. On aimerait sans doute pouvoir paralléliser le téléchargement des images...*
- *Les threads s'exécutent en parallèle, on ne peut en revanche pas contrôler combien sont en cours d'exécution (pas idéal, car N threads vont être en concurrence avec l'UIThread → possibilité de lag de l'UI...)*
- *Troisième méthode avec un ThreadPoolExecutor*

ThreadPoolExecutor

- *Service permettant de gérer l'exécution de threads dans un pool prédéfini:*

```
exec = Executors.newFixedThreadPool(5);
```

- *On peut ajouter des tâches au pool:*

```
Future<?> f = exec.submit(new Thread(){...});
```

dans notre exemple, uniquement 5 threads seront exécutés en parallèles, les autres sont mis dans une file d'attente

- *f.cancel(false), permet de sortir un thread de la file d'attente ou alors de l'interrompre s'il est en cours d'exécution*
f.cancel(true)

Exercices

- *Compléter les Adapters de l'application fournie:*
 - *AdapterAsyncTasks utilisation d'AsyncTasks*
 - *AdapterExecutorPool utilisation d'un ThreadPoolExecutor*
- *Vous rafraichirez les images du cache si elles sont plus âgées que 1 minute, pour l'exercice*
- *Deux questions :*
 - *Est-ce qu'il existe des librairies permettant de résoudre notre problématique ? Présentent-elles des limitations ?*
 - *Est-il possible, tout du moins théoriquement, de modifier notre problème pour le simplifier ? Développez*