Chapitre 05

Les threads

1 Thread et Java

- Examiner le chapitre relatif aux Threads du cours IFT1176 :

http://www.iro.umontreal.ca/~lokbani/cours/ift1176/communs/Cours/PDF/threads.pdf

- Examiner les exemples associés au cours en question :

http://www.iro.umontreal.ca/~lokbani/cours/ift1176/communs/Cours/ZIP/threads.zip

- Plus particulièrement les 3 exemples suivants :

Programme	Description
course.java	Thread suite à un héritage
courseapp.java	Thread suite à une implémentation
synchtest.java	Accès à une zone partagée

2 Thread et Android

- Quand une application est démarrée, un nouveau processus est créé afin de l'héberger. Ce processus contient un seul thread d'exécution.
- En procédant ainsi, on s'assure que si l'application plante, on ne fera pas planter le système. De même si une machine virtuelle associée à un processus plante, elle fait planter le processus en question et non pas le système en entier.
- Le système est ainsi protégé des soubresauts de ses applications.
- Tout ce mécanisme nécessite une efficacité dans l'exécution concurrente de plusieurs machines virtuelles et une rapidité lors du lancement d'une nouvelle instance de la machine virtuelle.
- L'application a par ailleurs droit aussi à un seul thread, appelé thread principal (« User Interface/Main Thread »). Par défaut, toutes les composantes d'une même application s'exécutent dans ce thread.

- Ce thread est responsable de l'affichage et de la mise à jour. Il supervise les échanges des messages, des notifications, ou des événements entre des composants graphiques.

  Pour ce qui suit, nous allons utiliser l'abréviation « UI » pour désigner « User Interface ».

- L'utilisation des threads sous Android est balisée par 2 règles majeures :

Ne pas bloquer le « UI Thread »

« L' UI thread » est le seul thread habilité à effectuer des modifications sur l'affichage

Ne pas bloquer le « UI Thread »

- Tout travail s'effectuant sur le thread principal ne doit pas s'éterniser.
- Il est donc important que ce thread ne se retrouve pas dérangé dans sa tâche par manque de ressources.
- Le système Android est responsable de préserver les ressources du système.
- Ainsi donc, si une application tarde à répondre à une requête de l'utilisateur, Android prend les choses en main et affiche à l'utilisateur le message « Application Not Responding » communément appelé « ANR ».
- Le système offre à l'utilisateur deux possibilités : arrêter l'exécution de l'application ou bien continuer l'exécution de l'application en espérant qu'elle va se débloquer par elle-même.
- Ce message s'affiche après un délai d'attente de 5 secondes dans le cas des applications et, 10 secondes pour le « Broadcast Receiver ».

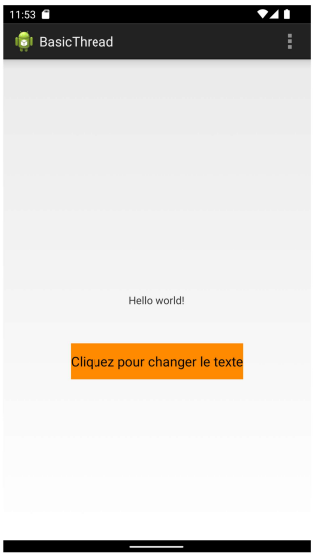
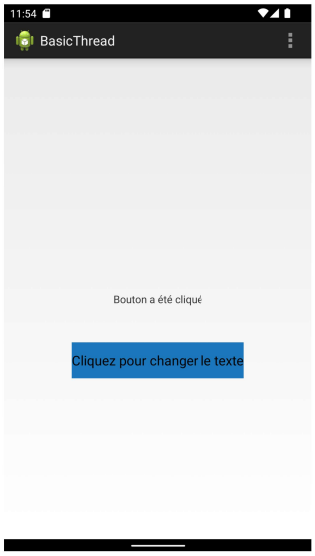
- Sur un émulateur, il faut activer l'affichage « ANR » via le mode de développement.

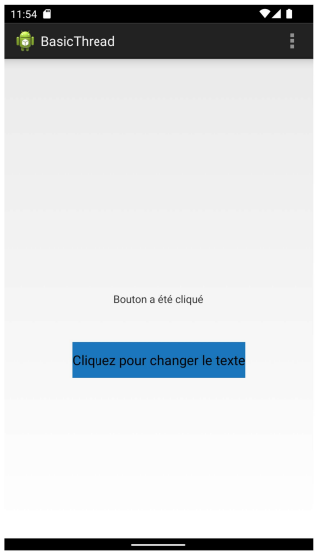
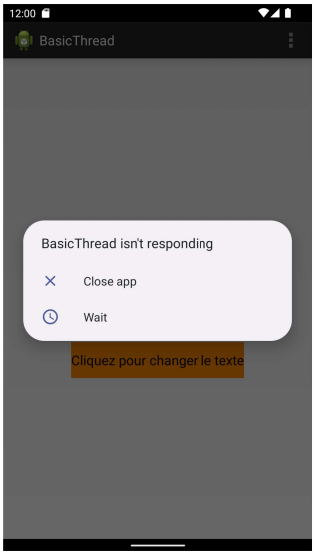
Voir l'exemple « **C05 : BasicThread** »

En cliquant sur le bouton, on met en veille le thread un certain temps. Dès que le temps est écoulé, on change le contenu du texte.

Nous sommes en présence de deux cas de figure : un temps d'attente court et un temps d'attente très long.

On peut provoquer « l'ANR » sans à avoir à modifier le temps d'attente. On n'a qu'à cliquer plusieurs fois sur l'écran. Le système se retrouve à gérer deux événements qui réclament son attention. Il passe à la trappe, celui qui ralentit son intervention.

Temps d'attente de « 20x1000 »	
	
On remarque que le bouton reste figé et de couleur bleue (ou grise foncée). On ne peut pas utiliser autre chose sur l'application.	Le texte a bien changé.

Temps d'attente de « 20x10000 »	
	
On remarque que le bouton reste figé et de couleur bleue. On ne peut pas utiliser autre chose sur l'application.	Après un long moment d'attente, le système a généré un message d'erreur.

Cette erreur va générer un fichier de traces. Ce fichier est disponible dans le répertoire « /data/anr » de l'appareil.

On peut le sauvegarder localement sur l'ordinateur connecté à l'appareil Android.

Ouvrez une fenêtre de commandes (DOS ou autre), et tapez la commande :

```
adb pull /data/anr/traces.txt DESTINATION
```

À noter que:

Le chemin vers la commande « adb » doit-être connu par le système, dans le cas contraire inclure le chemin complet.

« DESTINATION » est le répertoire dans lequel vous voulez sauvegarder le fichier, par exemple « . » pour indiquer le répertoire courant.

Il faut avoir les droits « root » pour avoir accès à ce fichier.

On peut récupérer aussi le fichier à travers l'onglet « Device File Explorer ».

« L' UI thread » est le seul thread habilité à effectuer des modifications sur l'affichage

- Votre application utilise plusieurs threads, et vous avez eu l'ingénieuse idée d'aller modifier l'interface graphique de l'application sans en référer au thread principal, Android vous rattrape et génère ce type exception :

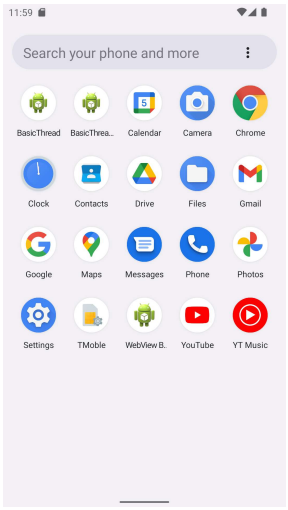
```
2022-02-04 21:14:25.130 8055-8123/ca.umontreal.iro.ift1155.basicthread02
E/AndroidRuntime: FATAL EXCEPTION: Thread-2
   Process: ca.umontreal.iro.ift1155.basicthread02, PID: 8055
   android.view.ViewRootImpl$CalledFromWrongThreadException: Only the
original thread that created a view hierarchy can touch its views.
```

- Tous les autres threads doivent se relayer par le thread principal pour pouvoir communiquer avec l'interface graphique (affichage).
- En centralisant la gestion, on s'assure ainsi de la cohérence de ce qui est affiché.

Voir l'exemple « **C05 : BasicThread02** »

Dans cet exemple, nous tentons de modifier à partir d'un second thread l'affichage qui, rappelons-le, est géré par le thread principal.

L'application plante et le système revient à la vue d'avant :




```
2022-02-04
21:17:49.670
8292-8321/ca.umontreal.iro.ift1155.basicthread
02 I/Process: Sending signal. PID: 8292 SIG: 9
```


3 Création de threads sous Android

- Pour éviter de priver le thread principal des ressources nécessaires pour son bon fonctionnement, nous pouvons déléguer les tâches lourdes à des threads qui vont s'exécuter en arrière-plan. En faisant cela, nous allons nous ajuster à la première contrainte « ne pas bloquer le UI Thread ».

- Nous allons examiner plusieurs manières de créer un thread à travers l'exemple des deux coureurs d'une course de 1000 mètres.

Héritage

Il n'est pas possible d'utiliser le principe d'héritage de classes pour réaliser cet exemple. En effet, la classe qui génère une activité a besoin d'hériter de la classe « Activity ». L'héritage est donc déjà utilisé.

L'activité implémente « Runnable »

Voir l'exemple « **C05 : Course01** »

```
public class MainActivity extends Activity implements Runnable { … }
```

Nous avons besoin de redéfinir dans la classe « MainActivity » la méthode « run » associée :

```
@Override
public void run() { … }
```

Dans la méthode associée au clic sur un bouton, nous avons implanté les instructions nécessaires pour démarrer les threads en question, un par coureur. L'appel « this » fait référence à une instance de la classe « main » (principale).

```
public void buttonClick(View view){
     Thread jthread = new Thread(this,"Jean");
     jthread.start();
     Thread pthread = new Thread(this,"Paul");
     pthread.start();

}
```


Avantages

Accès à toutes les méthodes de la classe « MainActivity », vu que la méthode « run » fait partie de la classe.

Inconvénients

Partager les données n'est pas trivial vu que tout le monde se connaît!
Passer un argument à la méthode « run » n'est pas possible. Tous les threads exécutent la même tâche.
Cette approche est utilisée pour la facilité d'accès aux données dans la classe principale. Si ces données tendent à être modifiées, assurez-vous de la synchronisation de ces données.

Une classe séparée implémente « Runnable »

Nous allons définir dans une classe à part le code nécessaire pour le bon fonctionnement du thread associé à l'application.

Voir l'exemple « **C05 : Course02** »

Nous avons défini une nouvelle classe « Coureur », cette dernière implémente « Runnable ».

Nous avons repris exactement la même méthode associée au clic sur un bouton.

Avantages

Il n'y a pas d'attaches à la classe mère.

Il est possible d'utiliser la classe dans d'autres applications.

Le passage par le constructeur de la classe est possible.

Vous utilisez cette approche quand il n'y a pas de données à partager (pas de synchronisation).

Inconvénients

Accéder à la classe « MainActivity » n'est pas une opération triviale. Une petite gymnastique s'inscrit en perspective.

Une classe interne

La classe « Coureur » fait maintenant partie de la classe « MainActivity ».

Voir l'exemple « **C05 : Course03** »

Avantages

Accès à toutes les méthodes de la classe « MainActivity », vu que la méthode « run » fait partie de la classe.
Vous pouvez passer des arguments à la méthode « run » à travers le constructeur de la classe qui va les transmettre aux instances qui feront exécuter la méthode « run ».

Inconvénients

Cette approche est utilisée pour la facilité d'accès aux données dans la classe principale. Si ces données tendent à être modifiées, assurez-vous de la synchronisation de ces données.

En résumé

	Activité implémente « Runnable »	Classe externe	Classe interne
Passer des arguments à « run »	Non	Oui	Oui
Accéder aux données dans la classe principale est facile	Oui	Non	Oui
Accès concurrent (situation de compétition / race condition)	Danger	Sans danger	Danger

4 Threads et UI

- Comme nous l'avons déjà mentionné, la 2e règle imposée par Android est de toujours passer par le thread principal pour modifier l'interface utilisateur.

- La solution a été implémentée sous cette forme :

  - Une file de messages, chaque message est un ensemble d'instructions à exécuter.
  - Des threads peuvent ajouter des messages dans la file.
  - Un seul thread extrait les messages un par un afin de les traiter.
  - Ce thread est dans un mode d'écoute infini dans l'attente d'un nouveau message.
  - C'est ce thread qui est associé à l'activité (interface graphique).

Android offre plusieurs manières d'accéder au thread UI :

- « runOnUiThread(Runnable action) » : l'action doit-être exécutée dans le thread UI.

  Reprendre l'exemple « **C05 : BasicThread02** » et corriger les lignes en question pour permettre la mise à jour de l'interface graphique. Le résultat est dans le projet « **C05 : TestThreadUI** ».

```
runOnUiThread (new Runnable () {
     public void run () {
          TextView myTextView = findViewById(R.id.myTextView);
          myTextView.setText("Bouton a été cliqué");
     }
});
```

- On peut ajouter un « Runnable » à la file des messages.

```
view.post (new Runnable () {
     public void run () {
          TextView myTextView = findViewById(R.id.myTextView);
          myTextView.setText("Bouton a été cliqué");
     }
});
```

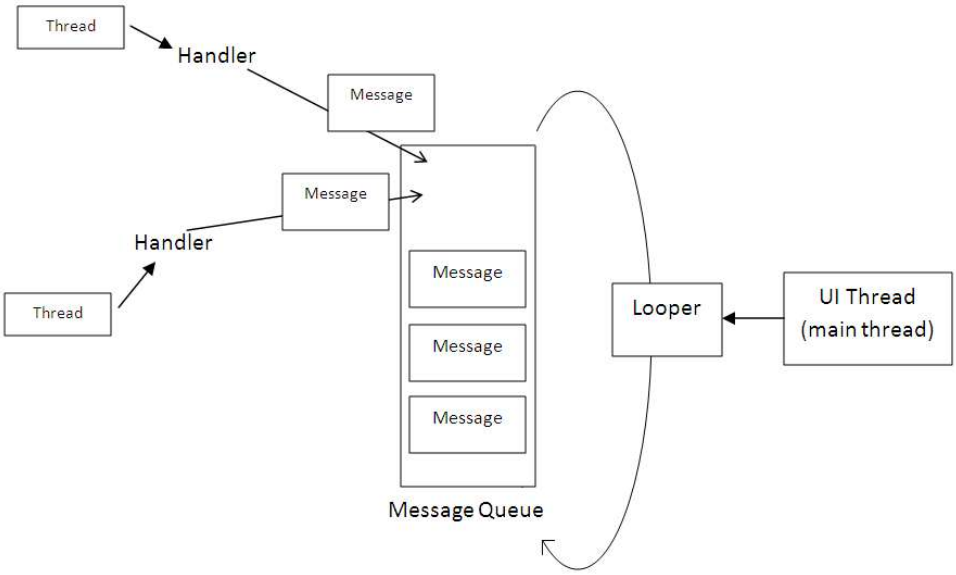
Examiner l'exemple « **C05 : PostThread** ».

- On peut définir un « handler »

Un handler permet d'ajouter des messages ou des « Runnable » à une file de messages.

Lors de la création d'un handler ce dernier est lié à un thread. Les messages sont ajoutés dans la file de ce thread.

Un « handler » est donc positionné entre le thread et la file des messages.



http://www.aviyehuda.com/images/AndroidMultithreading/android_threading.png

Les messages transmis par les handlers sont des instances de la classe « Message ».

https://developer.android.com/reference/android/os/Message.html

Pour construire un message :

```
Message msg = handler.obtainMessage();
```

« handler.obtainMessage » permet de récupérer d'anciens objets qui se trouvent dans le pool des messages au lieu d'en créer un nouveau.
boolean post (Runnable r) : ajoute r à la file des messages.
boolean sendEmptyMessage (int what) : envoie un message qui ne contient que la valeur « what », utilisée comme un identifiant.
boolean sendMessage(Message msg) : poste un message complet dans la file des messages.

Tous les messages seront reçus dans la méthode « handleMessage(Message msg) » du handle attaché à un thread.

Examiner les exemples « **C05 : Handler01** », « **C05 : Handler02** » et « **C05 : Handler03** ».

- Handler01 : un handler simpliste. Il envoie un message vide, mais il se charge de modifier la chaîne localisée dans l'interface graphique (UI Thread).
- Handler02 : un handler plus complexe qui se charge d'envoyer des données au Thread UI.
- Handler03 : la problématique de la déclaration du handler.

Pour l'exemple « Handler03 », nous avons cet avertissement :

```
This Handler class should be static or leaks might occur.
```

Ce document explique en détail la problématique :

http://www.androiddesignpatterns.com/2013/01/inner-class-handler-memory-leak.html

Nous avons introduit les corrections nécessaires dans « **C05 : Handler01** » et « **C05 : Handler02** ».

Cette problématique ne se pose plus depuis l'API 30+. Examiner l'exemple « **C05 : Handler03** » pour plus de détails.

Bibliographies

Processes and Threads

https://developer.android.com/guide/components/processes-and-threads.html

Sending Operations to Multiple Threads

https://developer.android.com/develop/background-work/background-tasks

Keeping Your App Responsive

https://developer.android.com/topic/performance/anrs/keep-your-app-responsive

Thread, Handler, AsyncTask et fuites mémoires

(attention, le document date un peu)

https://mathias-seguy.developpez.com/tutoriels/android/comprendre-thread-handler-asynctask-fuites-memoires/