
Android UIThread, Thread, Handler et AsyncTask

jean-michel Douin, douin au cnam point fr
version : 13 Mars 2020

Notes de cours

Bibliographie utilisée

<http://developer.android.com/reference/android/os/AsyncTask.html>

- Added in [API level 3](#)
Deprecated in [API level R](#) ...

Le site de Lars Vogella

<https://www.vogella.com/tutorials/AndroidBackgroundProcessing/article.html>

Pré-requis

- **Les threads en java**
 - **java.lang.Thread**
 - **java.lang.Runnable**

Sommaire

- **UiThread déclenche une activity et gère l'IHM**
 - Les appels des « onXXXX » c'est lui
- **Thread, Handler,** *du java traditionnel*
- **AsyncTask,** *adapté et créé pour se passer du traditionnel*

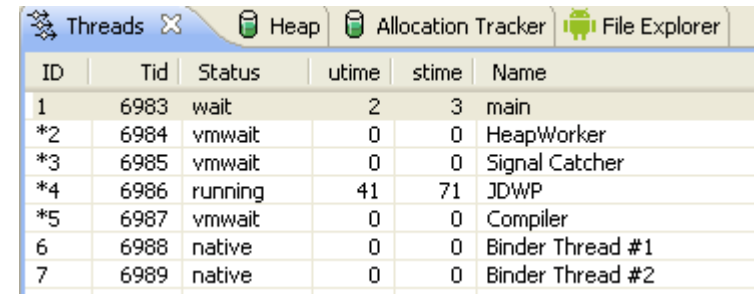
UIThread

- **Gère l'affichage de l'*Activity* :**
 - **Prise en compte des événements de l'utilisateur**
 - **Clic, déplacements, ...**
 - **...**
 - **Exécution des Listeners associés**
 - **...**
- **L'UIThread est le seul Thread « agréé »**
 - **Pour l'affichage au sein d'une activité**
 - **Accès à l'un des composants graphiques**
 - **Les événements liés à l'affichage sont gérés par une file**
- **Alors**
 - *Toutes les opérations de mises à jour, modifications de l'IHM doivent s'effectuer depuis cet UIThread*

En « rappel »

- Une application engendre un processus linux

à ce processus est associée une DVM
cette DVM installe des Threads



ID	Tid	Status	utime	stime	Name
1	6983	wait	2	3	main
*2	6984	vmwait	0	0	HeapWorker
*3	6985	vmwait	0	0	Signal Catcher
*4	6986	running	41	71	JDWP
*5	6987	vmwait	0	0	Compiler
6	6988	native	0	0	Binder Thread #1
7	6989	native	0	0	Binder Thread #2

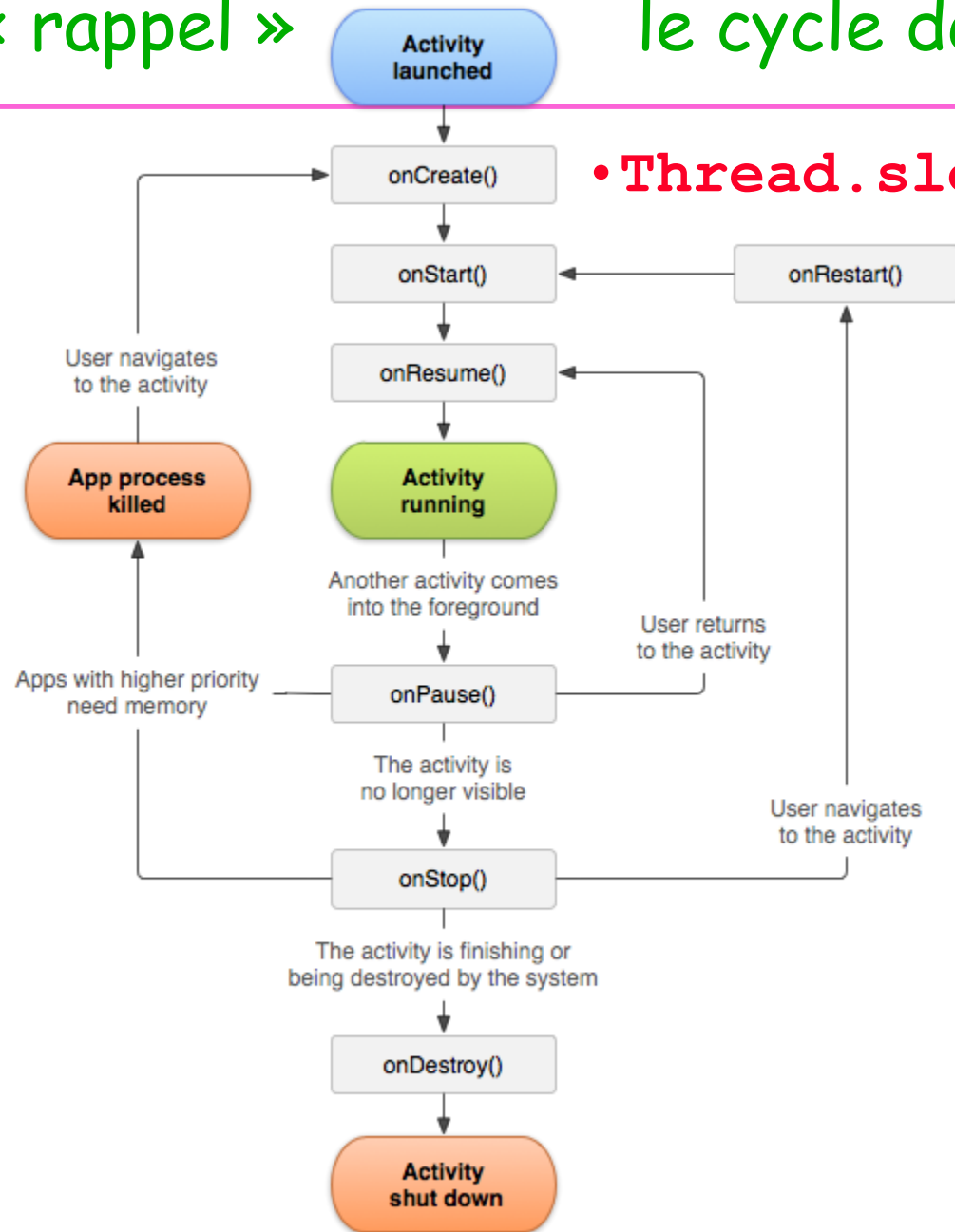
- Parmi ces Threads,
 - Un thread est nommé **main** alias **UIThread**
- Le thread **main** alias **UIThread**
 - Crée, initialise l'activité,
 - Déclenche les méthodes selon le cycle de vie,
 - Et est chargé de l'affichage,
 - De la prise en charge des actions de l'utilisateur

Exemples, et démonstration

- Exemples à ne pas suivre ...
- Exemple 1
 - La méthode *onCreate* contient un appel à **Thread.sleep(30000) !**
 - Un affichage est souhaité
 - **Thread.sleep(30000);**
 - Ou *SystemClock.sleep(30000);*
 - -> ce n'est qu'au bout de 30000ms que l'affichage se produit !!
 - Cycle de vie ... *rappel*

En « rappel »

le cycle de vie



Exemples suite, et démonstrations

- **Exemple 2**
 - **Au clic sur un bouton :**
 - **Thread.sleep est appelée ...**
 - *Ou bien SystemClock.sleep*
 - **Un affichage est souhaité ...**

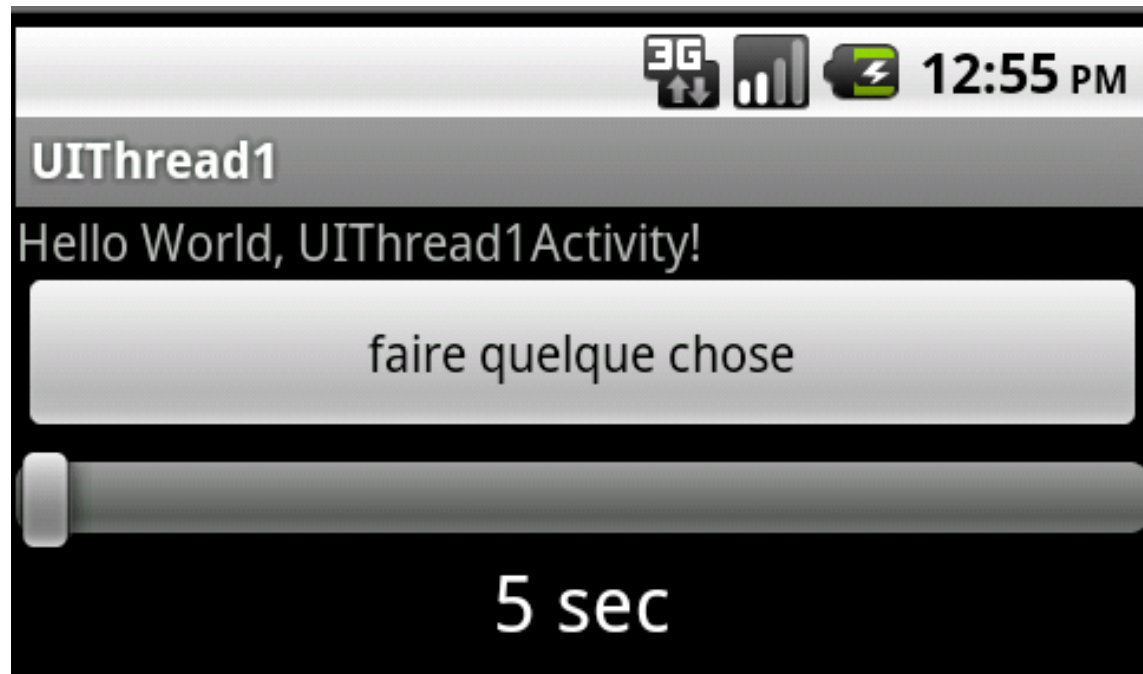
Exemple à ne pas suivre : une IHM au ralenti

- Un clic déclenche la méthode *faireQuelqueChose*
 - L'UIThread s'endort ... une seconde, deux secondes ... **et ...**

```
8 public class UIThread1Activity extends Activity {
9     private TextView tv;
10
11     @Override
12     public void onCreate(Bundle savedInstanceState) {
13         super.onCreate(savedInstanceState);
14         setContentView(R.layout.main);
15         this.tv = (TextView)findViewById(R.id.label);
16     }
17
18     public void faireQuelqueChose(View v){ // android:onClick
19         try{
20             for(int i=1;i<=5;i++){
21                 Thread.sleep(1000L);
22                 tv.setText(i + " sec");
23             }
24         }catch(Exception e){
25         }
26     }
```

et pas d'affichage...

IHM: affichage seulement au bout de 5 sec.



- **C'est seulement au bout de 5 secondes**
 - Qu'un nouveau clic devient possible,
 - Que la jauge est accessible (répond aux sollicitations de l'utilisateur)
 - **L'affichage n'est pas effectué**
 - toutes les secondes comme la lecture du source le laisse supposer
- **Démonstration de l'exemple à ne pas suivre**
 - Un seul affichage (visible) au temps écoulé : 5 sec !

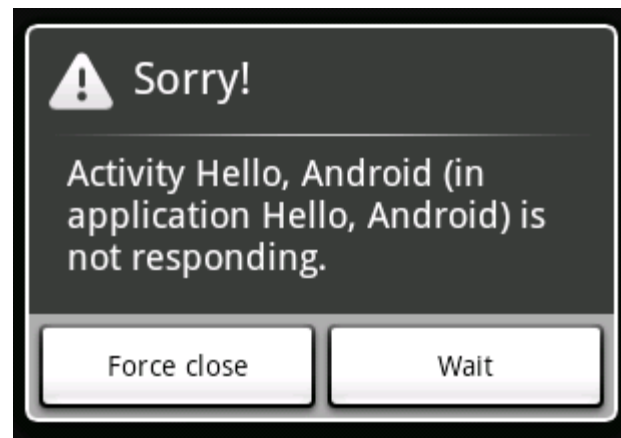
Démonstration

```
public void faireQuelqueChose(View v){ // android:onClick
    try{
        for(int i=1;i<=5;i++){
            Thread.sleep(5000L);
            Log.i("UiThread1Activity","apres 5 sec.");
            tv.setText(i + " sec");
            Log.i("UiThread1Activity","apres tv.setText");
        }
    } catch (Exception e){
    }
}
```

- **Ici 5 secondes et avec appels de Log.i**
 - Toujours rien à l'écran, sauf au bout de 25 sec.
 - Les affichages ne sont pas pris en compte, l'UiThread passe son temps à s'endormir et en conséquence ne permet pas la prise en compte des souhaits d'affichage (setText)

L'UIThread gère l'IHM et seulement

- **Une première conclusion**, *par la force des choses...*
 - **Toutes les opérations coûteuses en temps d'exécution doivent être placées dans un autre Thread**,
 - **mais pas n'importe lequel...**(l'UIThread est réservé à l'IHM)

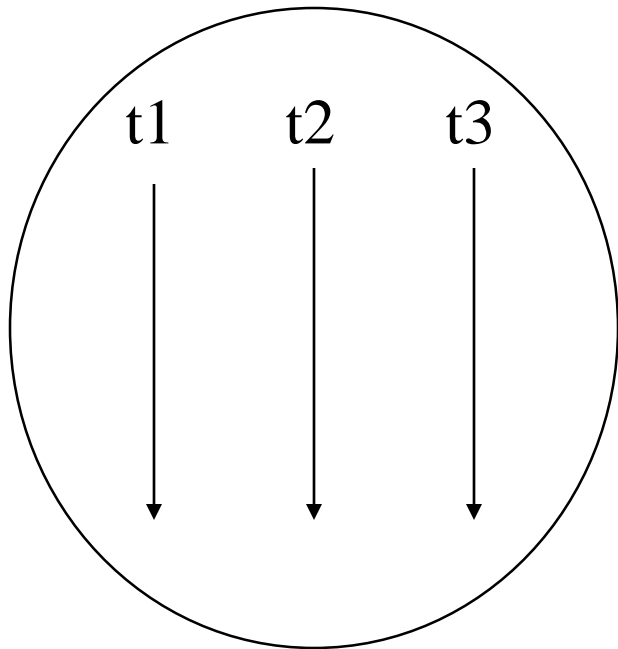


- **Ce qui arrive quand l'UIThread est bloqué plus de 5 sec.,**
- **Android considère que votre application ne répond plus**
cf. ANR, Application Not Responding

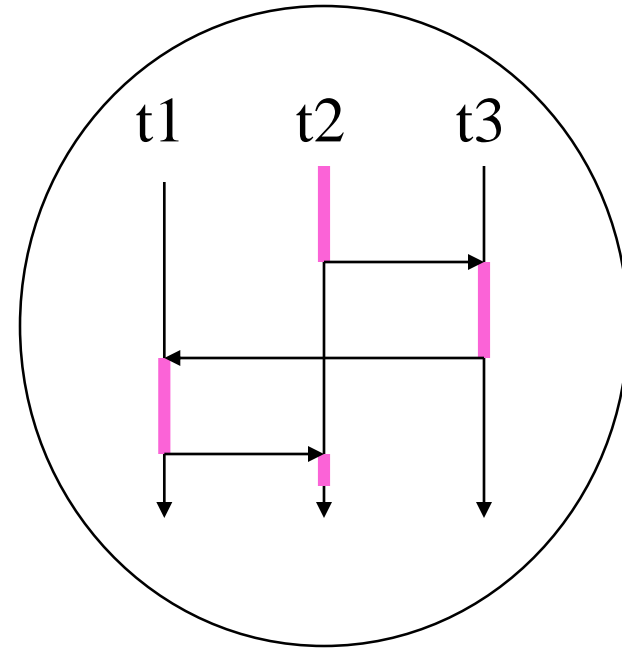
Rappel : les Threads en java

- *Rappels: les 5 prochaines diapositives peuvent être optionnelles*
- **Un Thread en java,**
 - une classe, une interface, deux méthodes essentielles
 - **La classe** `java.lang.Thread`
 - **L'interface** `java.lang.Runnable`
 - **Les deux méthodes**
 - **start** éligibilité du Thread,
 - **run** le code du Thread
 - » issue de `public interface Runnable{void run();}`

Contexte : Quasi-parallèle

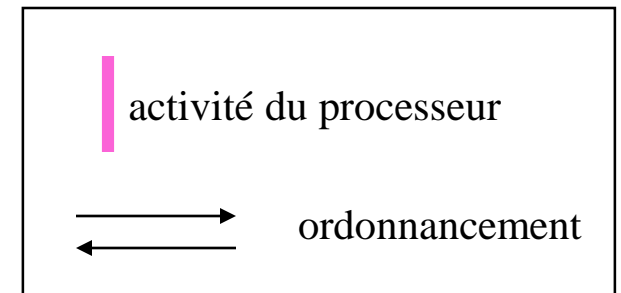


vue logique



vue du processeur

t1, t2, t3 sont des « Threads »



Rappel : La classe java.lang.Thread

Syntaxe : Création d'une nouvelle instance

Thread unThread = new T (); ... T extends Thread

- *(un(e) Thread pour processus allégé...)*

- « Démarrage » du *thread*

- **unThread.start();**
 - **éligibilité de UnThread**

- « Exécution » du *thread*

- *L 'ordonnanceur interne déclenche la méthode run() (unThread.run();)*

Rappel : java.lang.Thread, syntaxe

```
public class T extends Thread{  
    public void run(){  
        // traitement  
    }  
}  
Thread t = new T();  
t.start();
```

```
    // autre syntaxe, à l'aide d'une classe anonyme  
Thread t = new Thread(  
    new Runnable(){  
        public void run(){  
            // traitement  
        }  
    });  
t.start();
```

Runnable, autre syntaxe

```
Runnable r = new Runnable() {  
    public void run() {  
        // traitement  
    }  
};
```

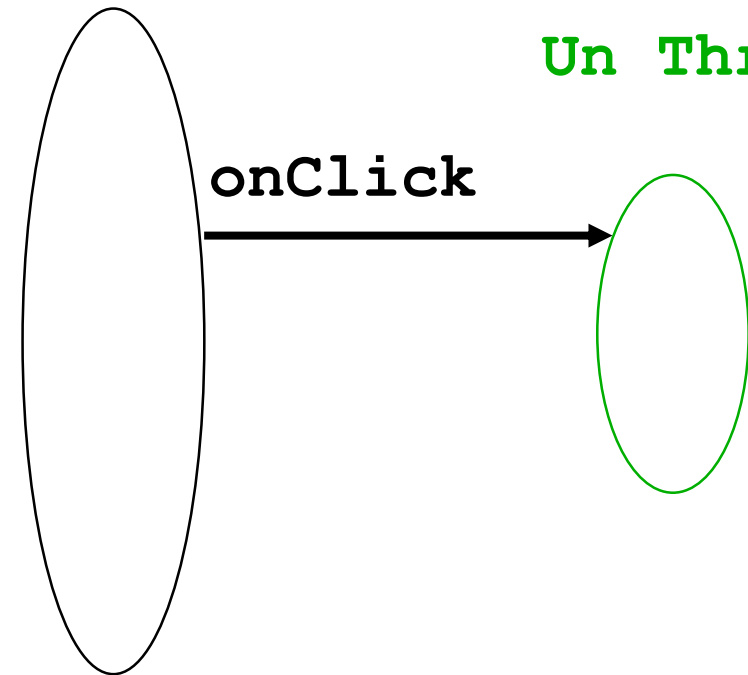
```
new Thread(r).start();
```

Reprenons l'exemple à ne pas suivre

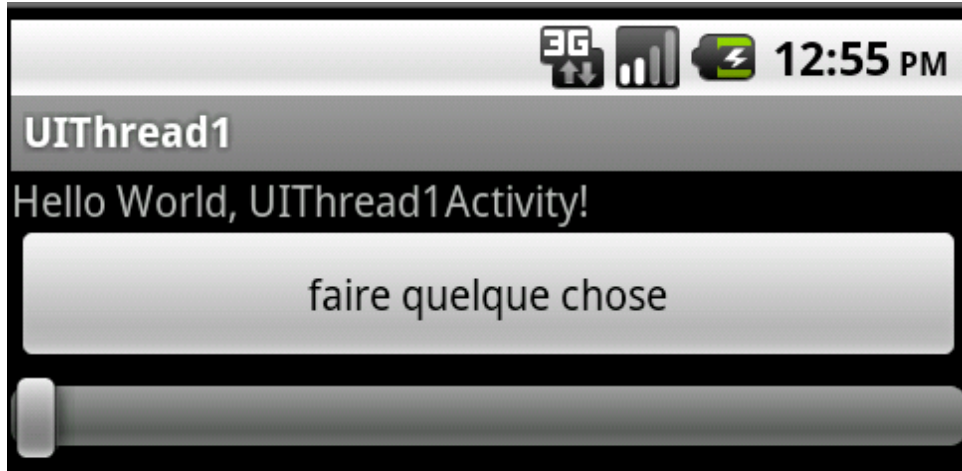
- Une « solution » serait de créer un **Thread** qui s'endort pendant $5 * 1\text{sec}$
- Au clic un **Thread** est créé

UIThread

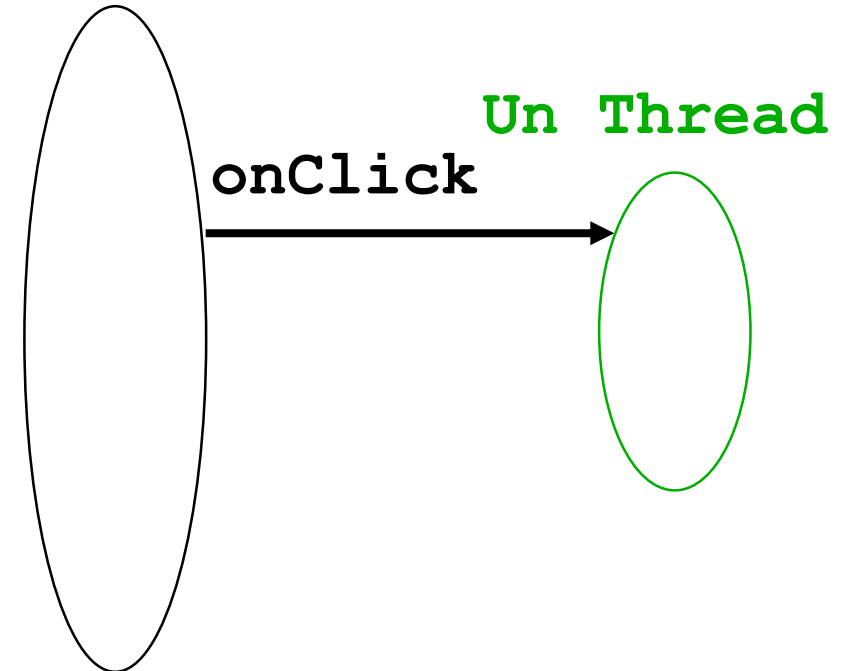
Un Thread



Au clic un thread est créé



UiThread







- **faire quelque chose** engendre un thread
 - UITHread devient disponible pour les sollicitations de l'utilisateur, Sur cet exemple le curseur de la jauge devient accessible

S'endormir dans un autre Thread

```
9 public class UIThread1Activity extends Activity {
10     private TextView tv;
11
12     @Override
13     public void onCreate(Bundle savedInstanceState) {
14         super.onCreate(savedInstanceState);
15         setContentView(R.layout.main);
16         this.tv = (TextView)findViewById(R.id.label);
17     }
18
19     public void faireQuelqueChose(View v) {
20         Thread t= new Thread(new Runnable() {
21             public void run() {
22                 try{
23                     for(int i=1;i<=5;i++){
24                         Thread.sleep(1000L);
25                         tv.setText(i + " sec");
26                     }
27                 } catch (Exception e) {
28                 }
29             }
30         });
31         t.start();
32     }
33 }
```

Démonstration

 Threads  Heap  Allocation Tracker  File Explorer					
ID	Tid	Status	utime	stime	Name
1	8025	wait	10	3	main
*2	8026	vmwait	0	0	HeapWorker
*3	8027	vmwait	0	0	Signal Catcher
*4	8028	running	0	3	JDWP
*5	8029	vmwait	2	0	Compiler
6	8030	native	0	0	Binder Thread #1
7	8031	native	0	0	Binder Thread #2
8	8036	timed-wait	0	0	Thread-10

- C'est mieux,
 - Le clic devient possible,
 - La jauge répond aux sollicitations avant les 5 secondes
- Mais

Mais

– Plus d’affichage du tout ...

– Cet autre Thread ne doit pas tenter de modifier l’IHM,
» Nul ne peut se substituer à l’UIThread ...

» Comportement incertain ...

Création d'un Thread

+

Accès prédéfinis à l'UIThread

- envois de séquences d'affichage
 - **runOnUiThread**
- envois de messages
 - **android.os.Handler**,
 - Méthodes **post, send, handleMessage ...**

Accès à l'UIThread

- **Envois de séquences d'affichage**
 - **runOnUiThread (Runnable r){...**
 - Méthode héritée de la classe Activity
 - Ou bien une instance de la classe Handler
 - **unHandler.post(Runnable r){ ...**
- **Envois de messages**
 - Avec une instance de la classe Handler
 - **unHandler.send(Message m){ ...**

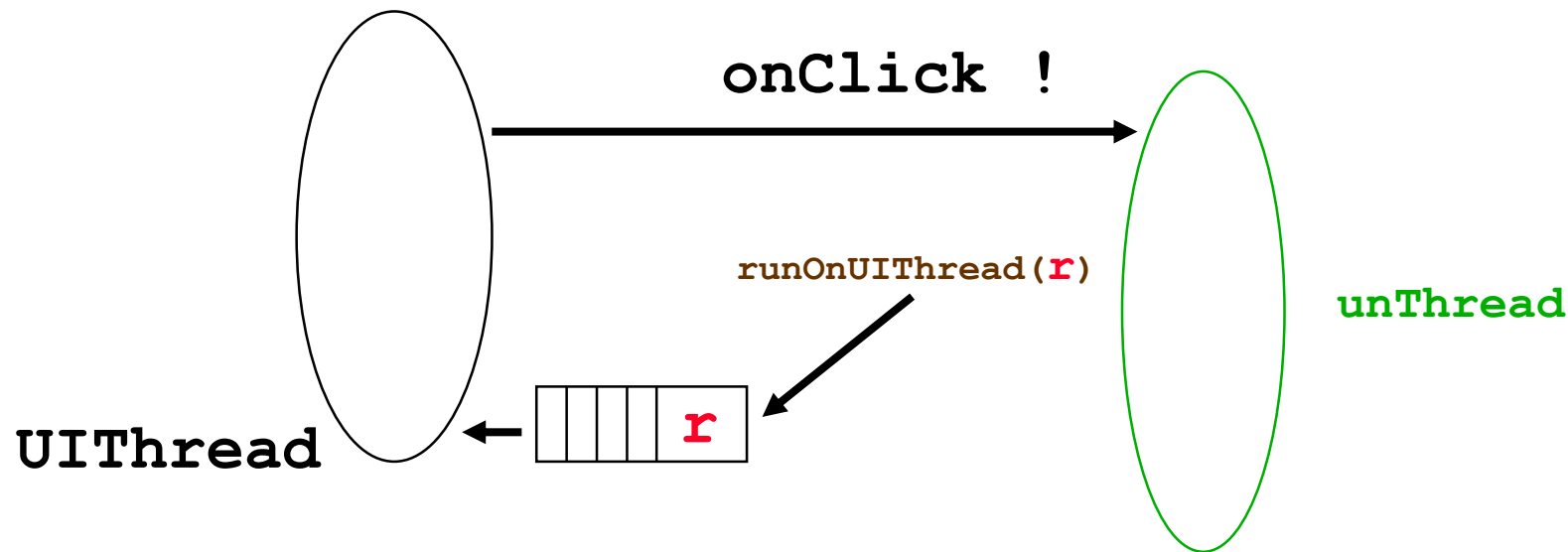
Ou le cumul des deux fonctionnalités

*Ce sera la classe **AsyncTask***

Envoi de séquences d'affichage

Dépôt dans la file d'attente de l'UIThread

- `runOnUiThread(Runnable r)`,
 - C'est une méthode de la classe Activity
- `r` est placé dans la file de l'UIThread,
 - Avec une exécution immédiate si l'appelant est l'UIThread



runOnUiThread

```
19
20 public void faireQuelqueChose(View v){
21     Thread t= new Thread(new Runnable(){
22         public void run(){
23             try{
24                 for(int i=1;i<=5;i++){
25                     Thread.sleep(1000L);
26                     final int value = i;
27                     runOnUiThread(new Runnable(){
28                         public void run(){
29                             tv.setText(value + " sec");
30                         }
31                     });
32                 }
33             } catch (Exception e){
34             }
35         }
36     });
37     t.start();
38 }
39
```

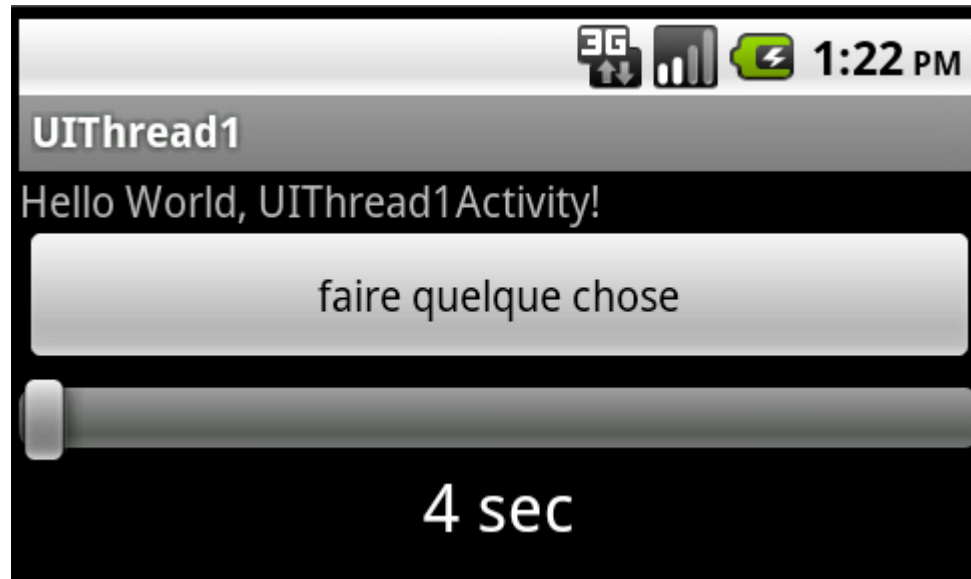
- Tout semble correct

runOnUiThread est héritée

```
public void faireQuelqueChose /*_1_5*/(View v){ // android:onClick
    try{
        for(int i=1;i<=5;i++){
            Thread.sleep(1000L);
            final int value = i;
            UIThread1Activity.this.runOnUiThread(new Runnable(){
                public void run(){
                    tv.setText(value + " sec");
                }
            });
        }
    } catch (Exception e){
    }
}
```

- **Tout semble correct**
 - **runOnUiThread est bien héritée de la classe Activity**

Affichage de 1,2,3,4 sec, enfin



- **Ok**
- *Démonstration*

Une autre façon de faire

- **android.os.Handler**
 - Permet de poster des instructions dans la file de l'UIThread
 - Analogue à l'usage de `runOnUiThread`
 - Permet aussi de poster des messages à l'attention de l'UIThread

Handler ... idem à runOnUiThreadThread

- **android.os.Handler**

- Cf. runOnUiThreadThread

- **handler** comme variable d'instance de l'activity

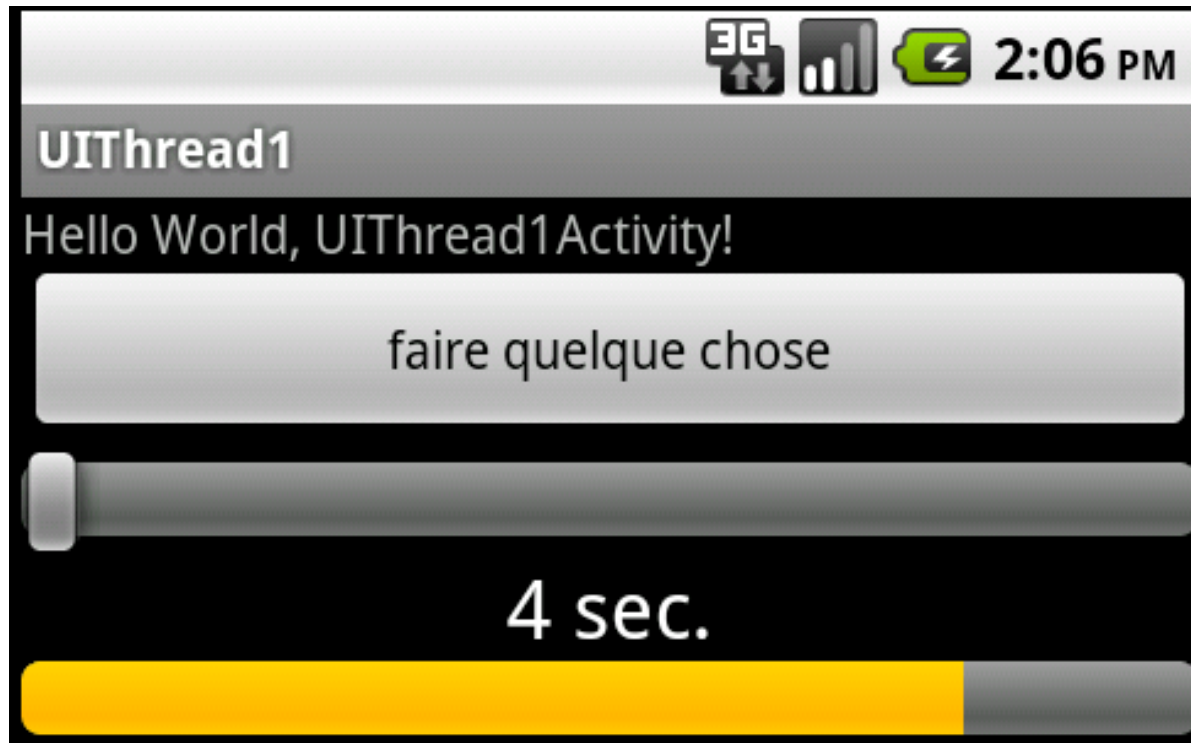
```
10 public class UIThread1Activity extends Activity {
11     private TextView tv;
12     private Handler handler;
13
14     @Override
15     public void onCreate(Bundle savedInstanceState) {
16         super.onCreate(savedInstanceState);
17         setContentView(R.layout.main);
18         this.tv = (TextView)findViewById(R.id.label);
19         this.handler = new Handler();
20     }
21
22     public void faireQuelqueChose(View v){
23         Thread t= new Thread(new Runnable(){
24             public void run(){
25                 try{
26                     for(int i=1;i<=5;i++){
27                         Thread.sleep(1000L);
28                         final int value = i;
29                         handler.post(new Runnable(){
30                             public void run(){
31                                 tv.setText(value + " sec");
32                             }
33                         });
34                     }
35                 }catch(Exception e){
36                 }
37             }
38         });
39         t.start();
40     }
41 }
```

runOnUiThread / Handler.post

```
public final void runOnUiThread(Runnable action) {  
    if (Thread.currentThread() != mUiThread) {  
        mHandler.post(action);  
    } else {  
        action.run();  
    }  
}
```

- Sans commentaire ...

Handler en Progress : il n'y a qu'un pas



- L'interface s'est enrichie d'un « ProgressBar
- <ProgressBar
 - `ProgressBar p = (ProgressBar) findViewById(....`
 - `p.setProgress(value)`

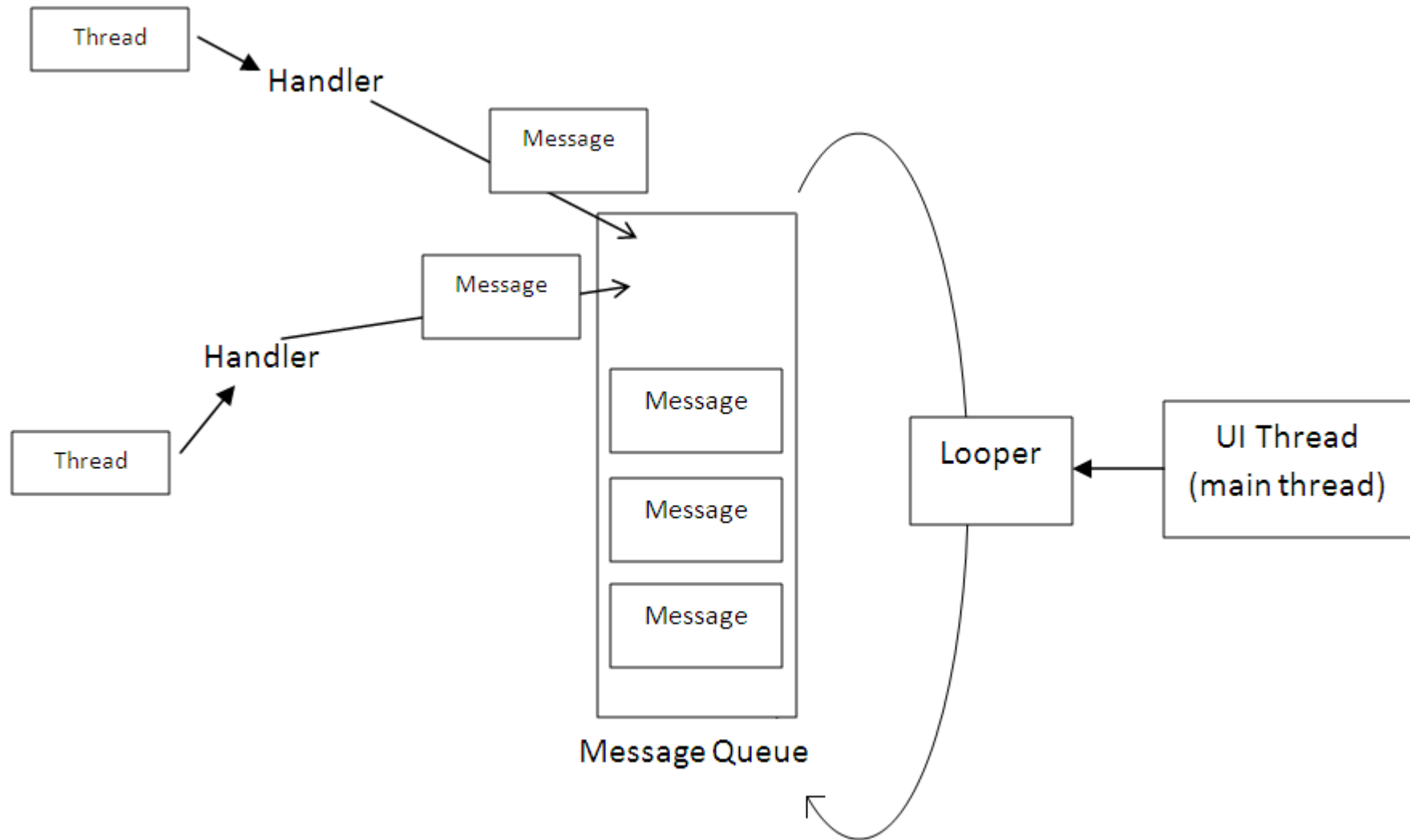
En progress, mais de moins en moins lisible...

```
11 public class UIThread1Activity extends Activity {
12     private TextView tv;
13     private ProgressBar progress;
14     private Handler handler;
15
16     @Override
17     public void onCreate(Bundle savedInstanceState) {
18         super.onCreate(savedInstanceState);
19         setContentView(R.layout.main);
20         this.tv = (TextView) findViewById(R.id.label);
21         this.progress = (ProgressBar) findViewById(R.id.progress);
22         this.handler = new Handler();
23     }
24
25
26     public void faireQuelqueChose(View v) {
27         Thread t= new Thread(new Runnable() {
28             public void run() {
29                 try{
30                     for(int i=1;i<=5;i++){
31                         Thread.sleep(1000L);
32                         final int value = i;
33                         handler.post(new Runnable() {
34                             public void run() {
35                                 tv.setText(value + " sec.");
36                                 progress.setProgress(value);
37                             }
38                         });
39                     }
40                 } catch (InterruptedException e) {
41                     // TODO Auto-generated catch block
42                     e.printStackTrace();
43                 }
44             }
45         });
46         t.start();
47     }
48 }
```

Une autre façon de faire, le retour

- Poster un **message** à destination de l'UIThread
- Par exemple
 - informer l'utilisateur du déroulement d'un traitement
 - Un top toutes les secondes
 - Il faut
 - Obtenir une instance de message
 - Installer au sein de l'activité,
 - la gestion du message par une méthode du handler
 - » handleMessage est redéfinie

Handler, Message et la classe Looper



- <http://www.aviyehuda.com/2010/12/android-multithreading-in-a-ui-environment/>

Réception d'un message toutes les sec.

```
public class UIThread1Activity extends Activity {  
    private TextView tv;  
    private ProgressBar progress;  
  
    private Handler handler = new Handler() {  
        public void handleMessage(Message msg) {  
            tv.setText(msg.what + " sec");  
            progress.setProgress(msg.what);  
        }  
    };  
}
```

- **Un handler comme variable d'instance de l'activité**
 - Redéfinition de la méthode **handleMessage**

L'Activity

```
public void faireQuelqueChose(View v){ // android:onClick
    new Thread(new Runnable(){
        public void run(){
            for(int i=1;i<=5;i++){
                try {
                    Thread.sleep(1000L);
                } catch (InterruptedException e) {
                }
                Message msg = handler.obtainMessage(i);
                handler.sendMessage(msg);
            }
        }
    }).start();
}
```

- **Un Thread**

- **Qui à chaque seconde,**

- 1. Obtient un message identifié par un paramètre (i)**
- 2. Envoie ce message**
 - *Le paramètre i correspond au champ `msg.what`*

Au sein de l'Activity

```
public void faireQuelqueChose(View v){ // android:onClick
    new Thread(new Runnable(){
        public void run(){
            for(int i=1;i<=5;i++){
                try {
                    Thread.sleep(1000L);
                } catch (InterruptedException e) {
                }
                Message msg = handler.obtainMessage(i);
                handler.sendMessage(msg);
            }
        }
    }).start();
}
```

Android se charge
de déclencher
handleMessage

```
public class UIThread1Activity extends Activity {
    private TextView tv;
    private ProgressBar progress;

    private Handler handler = new Handler() {
        public void handleMessage(Message msg) {
            tv.setText(msg.what + " sec");
            progress.setProgress(msg.what);
        }
    };
};
```

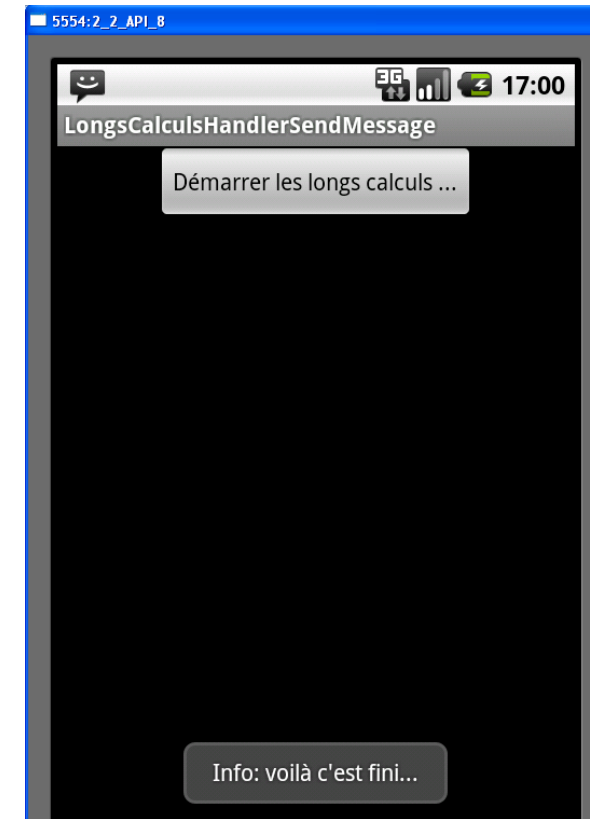
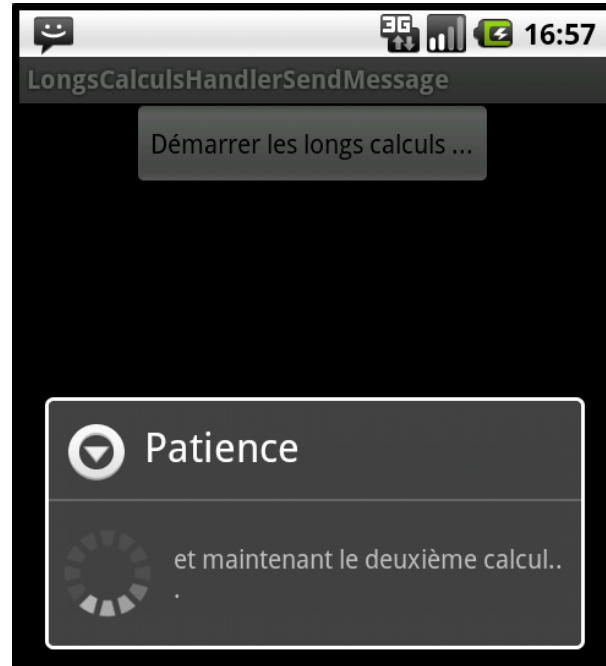
- Envoi + réception du message
 - `i == msg.what`

Messages à destination de l'UIThread

- **Un exemple classique**

- **Extrait de** <http://www.tutomobile.fr/utiliser-une-progressdialog-dans-ses-applications-android-tutoriel-android-n%C2%B022/03/02/2011/>
- **Une Activity avec deux longs, très longs calculs**
 - **Un Thread se charge des deux calculs**
- **Comment informer l'utilisateur tout au long des calculs ?**
 - **Au démarrage,**
 - **Entre deux calculs,**
 - **D'une erreur,**
 - **De la fin**

Copie d'écran de ce que l'on souhaiterait



- Deux ProgressDialog et pour terminer un toast

Comment informer l'utilisateur ?

- **Attribut onClick ...** de l'unique bouton de l'interface (start)
 - Méthode **onClickStart**

```
public void onClickStart(View v){
    mProgressDialog = ProgressDialog.show(this, "Patience",
        "de longs calculs commencent...", true);

    Runnable r = new Runnable(){
        public void run(){
            // le premier calcul débute
            doLongOperation1();
            // et maintenant le deuxième calcul
            doLongOperation2();
            // voilà c'est fini
        };
        new Thread(r).start();
    }
}
```

Comment informer l'utilisateur ?

- Un handler est créé
- Des messages lui sont envoyés, *Looper* s'en charge

```
public void onClickStart(View v){
    mProgressDialog = ProgressDialog.show(...);

    Runnable r = new Runnable(){
        public void run(){
            // message = le premier calcul débute
            // envoi du message à destination du handler

            doLongOperation1();
            // message = maintenant le deuxième calcul
            // envoi du message à destination du handler

            doLongOperation2();
            // message = voilà c'est fini
            // envoi d'un message à destination du handler

        }
    };
    new Thread(r).start();
}
```

onClickStart est décoré, MSG_IND==1, MSG_END==2

```
public void onClickStart(View v){
    mProgressDialog = ProgressDialog.show(this, "Patience",
        "de longs calculs commencent...", true);
    Runnable r = new Runnable(){
        public void run(){

            String progressBarData = "le premier calcul débute...";
            Message msg = mHandler.obtainMessage(MSG_IND, (Object) progressBarData);
            mHandler.sendMessage(msg);
            doLongOperation1();

            progressBarData = "et maintenant le deuxième calcul...";
            msg = mHandler.obtainMessage(MSG_IND, (Object) progressBarData);
            mHandler.sendMessage(msg);
            doLongOperation2();

            progressBarData = "voilà c'est fini...";
            msg = mHandler.obtainMessage(MSG_END, (Object) progressBarData);
            mHandler.sendMessage(msg);
        }
    };
    new Thread(r).start();
}
```

Un Handler est créé, redéfinition de handleMessage

```
final Handler mHandler = new Handler() {  
  
    public void handleMessage(Message msg) {  
        if (mProgressDialog.isShowing()) {  
            if (msg.what==MSG_IND)  
                mProgressDialog.setMessage(((String) msg.obj));  
  
            if (msg.what==MSG_END) {  
                Toast.makeText(getApplicationContext(), "Info:" +  
                    (String)msg.obj,  
                    Toast.LENGTH_LONG  
                ).show();  
  
                mProgressDialog.dismiss();  
            }  
        }  
    }  
};
```

Envoi et réception

```
public void onClickStart(View v){
    ProgressDialog = ProgressDialog.show(this, "Patience",
        "de longs calculs commencent...", true);

    Runnable r = new Runnable(){
        public void run(){

            String progressBarData = "le premier calcul débute...";
            Message msg = mHandler.obtainMessage(MSG_IND, (Object) progressBarData);
            mHandler.sendMessage(msg);
            doLongOperation1();

            progressBarData = "et maintenant le deuxième calcul...";
            msg = mHandler.obtainMessage(MSG_IND, (Object) progressBarData);
            mHandler.sendMessage(msg);
            doLongOperation2();

            progressBarData = "voilà c'est fini...";
            msg = mHandler.obtainMessage(MSG_END, (Object) progressBarData);
            mHandler.sendMessage(msg);
        }
    };
    new Thread(r).start();
}
```

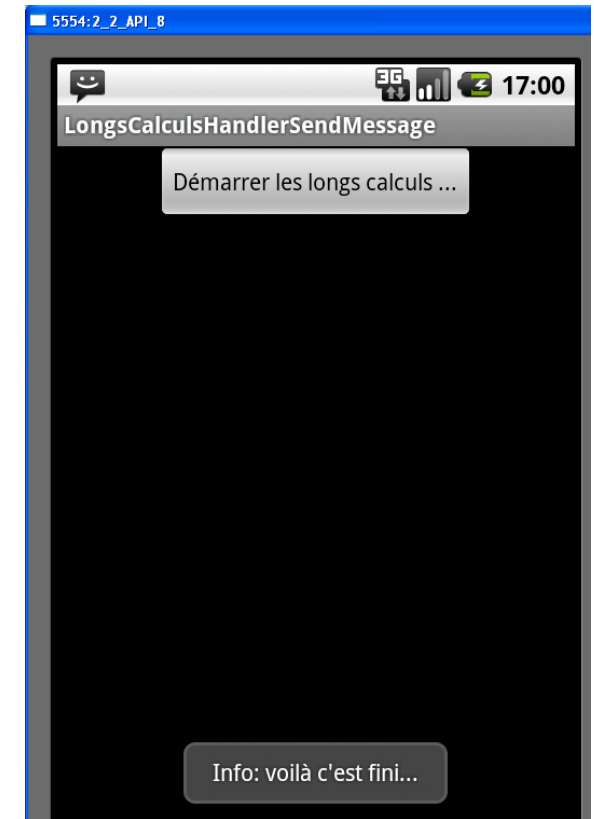
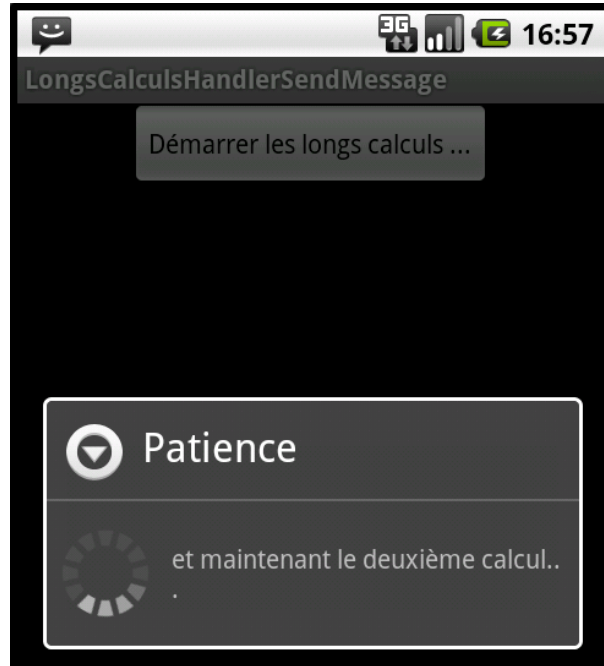
1 msg.what==MSG_IND
2 msg.what==MSG_IND
3 msg.what==MSG_END

```
final Handler mHandler = new Handler() {
    public void handleMessage(Message msg) {
        if (mProgressDialog.isShowing()) {
            if(msg.what==MSG_IND)
                mProgressDialog.setMessage(((String) msg.obj));

            if(msg.what==MSG_END){
                Toast.makeText(getApplicationContext(), "Info:" +
                    (String)msg.obj,
                    Toast.LENGTH_LONG
                ).show();

                mProgressDialog.dismiss();
            }
        }
    }
};
```

Copie d'écran obtenue ...



- à quel prix ... *mais avec l'habitude ?*

Un premier résumé

- **UiThread** : à préserver
- **Thread pas facile et ne suffit pas toujours**
 - Rappel : Pas de thread autre que l'UiThread pour gérer l'affichage
- **Handler**
 - Une interface \leftrightarrow avec l'UiThread
 - **post()** une instance de Runnable
 - **sendMessage()** un message
 - Message obtenu depuis un handler
- **Existe-t-il une classe toute prête et simplificatrice ?**
 - **oUI !**
 - `public class AsyncTask<Params, Progress, Result>`

Sommaire

- **UiThread** déclenche une activity et gère l'IHM
 - Les appels des « onXXXX » c'est lui
- **Thread, Handler**, *du java traditionnel*
- **AsyncTask**, *adaptée et créée pour se passer du traditionnel*
 - *Contient Un Thread + Handler*
- **Mais...**

AsyncTask<Params, Progress, Result>

- Avec la classe,
AsyncTask<Params, Progress, Result>
 - <http://developer.android.com/reference/android/os/AsyncTask.html>
- Nous avons un thread et un handler créés en interne
 - Un thread : pour le traitement en tâche de fond
 - Un handler : pour la mise à jour de l'UI
- **Params** type des paramètres transmis au Thread (*tâche de fond*)
- **Progress** type des paramètres en cours de traitement transmis au Handler/UIThread
- **Result** type du résultat pour l'appelant

Schéma de programme, syntaxe, AsyncTask<Params, Progress, Result>

```
public class MonActivity extends Activity{
```

```
    public void onClickStart(){
```

```
        ...
```

```
        WorkAsyncTask wt = new WorkAsyncTask();
```

```
        ...
```

```
    }
```

Classe interne

```
private class WorkAsyncTask
```

```
// ou private static class WorkAsyncTask
```

```
    extends AsyncTask<String, Long, Boolean>{
```

```
    }
```

```
}
```

Schéma de programme, syntaxe, AsyncTask<Params, Progress, Result>

```
public void onClickStart(){
```

```
...
```

```
    WorkAsyncTask wt = new WorkAsyncTask() ;
```

```
    // appels de méthodes publiques
```

```
...
```

```
}
```

```
private class WorkAsyncTask
```

```
    // ou private static class WorkAsyncTask
```

```
        extends AsyncTask<String, Long, Boolean>{
```

```
            // méthodes redéfinies, public, protected
```

```
}
```

AsyncTask< Params, Progress, Result >

Public Methods	
final boolean	<code>cancel</code> (boolean mayInterruptIfRunning) Attempts to cancel execution of this task.
static void	<code>execute</code> (Runnable runnable) Convenience version of <code>execute (Object)</code> for use with a simple Runnable object.
final AsyncTask<Params, Progress, Result>	<code>execute</code> (Params... params) Executes the task with the specified parameters.
final AsyncTask<Params, Progress, Result>	<code>executeOnExecutor</code> (Executor exec, Params... params) Executes the task with the specified parameters.
final Result	<code>get</code> (long timeout, TimeUnit unit) Waits if necessary for at most the given time for the computation to complete, and then retrieves its result.
final Result	<code>get</code> () Waits if necessary for the computation to complete, and then retrieves its result.
final AsyncTask.Status	<code>getStatus</code> () Returns the current status of this task.
final boolean	<code>isCancelled</code> () Returns <code>true</code> if this task was cancelled before it completed normally.

- Une des méthodes publiques pour les clients

- `execute(param1, param2, param3)` // cf. Params...

Schéma de programme, syntaxe, AsyncTask<Params, Progress, Result>

```
public void onClickStart(){
```

```
...  
    WorkAsyncTask wt = new WorkAsyncTask();  
    wt.execute(string1, string2, string3);  
...  
}
```

```
private class WorkAsyncTask
```

```
// ou private static class WorkAsyncTask
```

```
    extends AsyncTask<String, Long, Boolean>{
```

```
        // méthodes redéfinies (déclarées protected)
```

```
    }
```

AsyncTask< Params, Progress, Result >

Public Methods		
	final boolean	<code>cancel</code> (boolean mayInterruptIfRunning) Attempts to cancel execution of this task.
	static void	<code>execute</code> (Runnable runnable) Convenience version of <code>execute (Object)</code> for use with a simple Runnable object.
	final AsyncTask<Params, Progress, Result>	<code>execute</code> (Params... params) Executes the task with the specified parameters.
	final AsyncTask<Params, Progress, Result>	<code>executeOnExecutor</code> (Executor exec, Params... params) Executes the task with the specified parameters.
	final Result	<code>get</code> (long timeout, TimeUnit unit) Waits if necessary for at most the given time for the computation to complete, and then retrieves its result.
	final Result	<code>get</code> () Waits if necessary for the computation to complete, and then retrieves its result.
	final AsyncTask.Status	<code>getStatus</code> () Returns the current status of this task.
	final boolean	<code>isCancelled</code> () Returns true if this task was cancelled before it completed normally.

- **Autres méthodes publiques : *pour les clients***
 - **cancel, get, executeOnExecutor, ...**

Schéma de programme, syntaxe, AsyncTask<Params, Progress, Result>

```
public void onClickStart(){
    ...
    this.wt = new WorkAsyncTask();
    wt.execute(string1, string2, string3);
    ...
}

public void onClickResult(){
    ...
    Boolean result = wt.get();
    ...
}

private class WorkAsyncTask
// ou private static class WorkAsyncTask

    extends AsyncTask<String, Long, Boolean>{

    // méthodes redéfinies, protected

}
```


Schéma de programme, syntaxe, AsyncTask<Params, Progress, Result>

```
public void onClickStart(){  
    ...  
    wt = new WorkAsyncTask();  
    wt.execute(string1, string2, string3);  
    ...  
}
```

```
private class WorkAsyncTask  
    extends AsyncTask<String, Long, Boolean>{
```

// Quelles sont les méthodes à redéfinir

```
    public void onPreExecute() ...  
    public Boolean doInBackground(String... s) ...  
    public void onProgressUpdate(Long... l) ...  
    public void onPostExecute(Boolean ) ...  
}
```

class `WorkAsyncTask` extends `AsyncTask<Params, Progress, Result>`

Protected Methods	
abstract Result	<code>doInBackground (Params... params)</code> Override this method to perform a computation on a background thread.
void	<code>onCancelled (Result result)</code> Runs on the UI thread after <code>cancel (boolean)</code> is invoked and <code>doInBackground (Object[])</code> has finished.
void	<code>onCancelled ()</code> Applications should preferably override <code>onCancelled (Object)</code> .
void	<code>onPostExecute (Result result)</code> Runs on the UI thread after <code>doInBackground (Params...)</code> .
void	<code>onPreExecute ()</code> Runs on the UI thread before <code>doInBackground (Params...)</code> .
void	<code>onProgressUpdate (Progress... values)</code> Runs on the UI thread after <code>publishProgress (Progress...)</code> is invoked.
final void	<code>publishProgress (Progress... values)</code> This method can be invoked from <code>doInBackground (Params...)</code> to publish updates on the UI thread while the background computation is still running.

- **Les méthodes héritées**

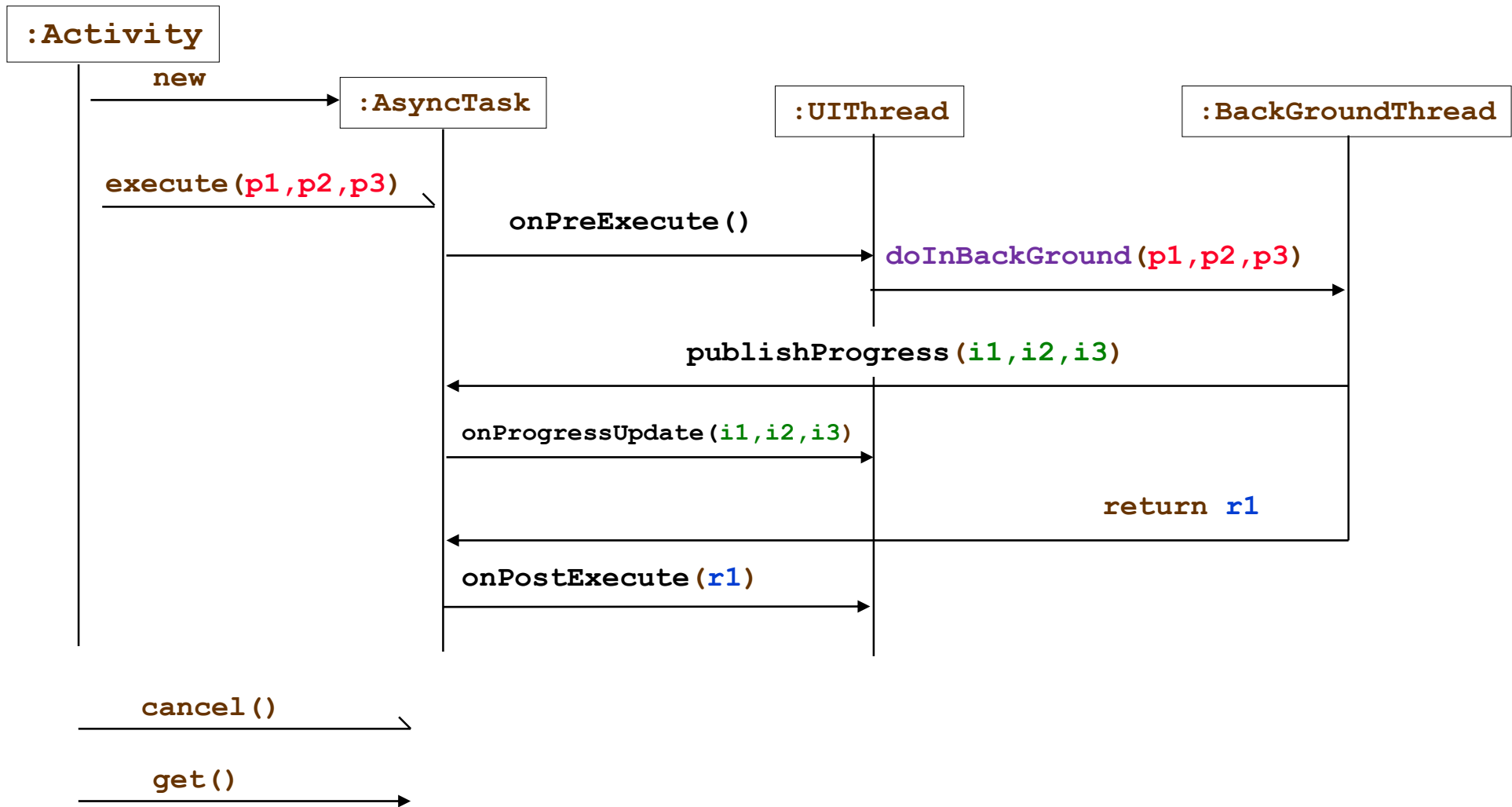
- **En liaison directe avec l'UI Thread**
- **`doInBackground` doit être redéfinie dans `WorkAsyncTask`**
 - <http://developer.android.com/reference/android/os/AsyncTask.html>
- **`publishProgress` déclenche, appelle `onProgressUpdate`**
 - **La quantité de sable du sablier ...**

Schéma de programme, syntaxe, AsyncTask<Params, Progress, Result>

```
public void onStart() {  
    ...  
    this.wt = new WorkAsyncTask();  
    wt.execute( string0, string1, string2 );  
    ...  
}  
  
private class WorkAsyncTask extends AsyncTask<String,Long,Boolean>{  
  
    void onPreExecute() { // faire patienter l'utilisateur,  
                          // affichage d'un sablier...  
  
    Boolean doInBackground(String... t){ // effectuer la tâche coûteuse en temps  
                                          // t[0]/string0, t[1]/string1,...  
        publishProgress( 10, 11, 12 ); // déclenche onProgressUpdate  
  
    void onProgressUpdate(Long... v) { // informer l'utilisateur que le  
                                         traitement est en cours  
                                         // t[0]/10, t[1]/11,...  
  
    void onPostExecute(Boolean b) { // le sablier disparaît,  
                                     une éventuelle erreur est affichée  
    }  
}
```

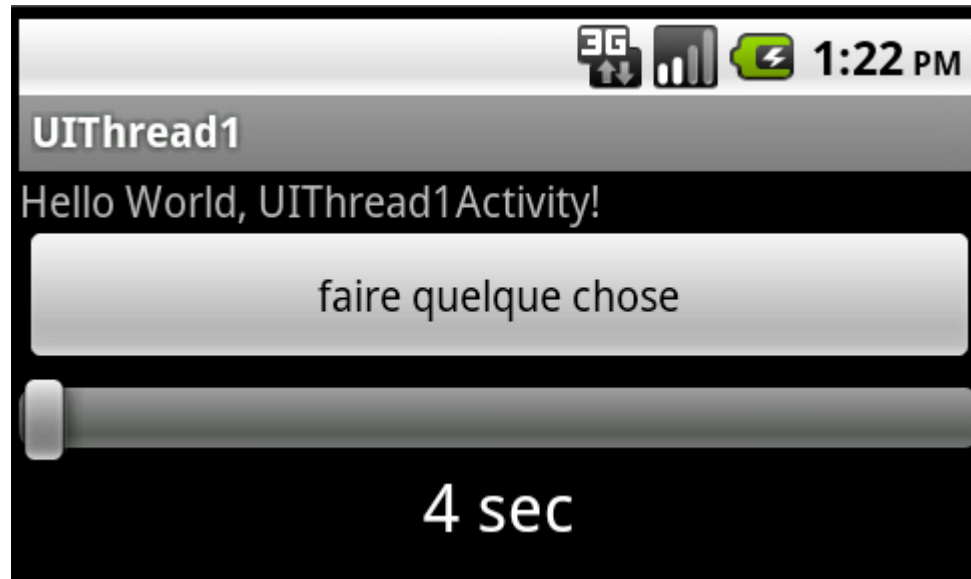
Diagramme de séquences simplifié

<Params, Progress, Result>



• Discussions

Exemple initial revu : affichage de 1,2,3,4 sec



- **AsyncTask**<Void, Integer, Integer>

AsyncTask<Void, Integer, Integer>

À chaque clic : `new ProgressBarTask().execute();`

```
private class ProgressBarTask extends  
    AsyncTask<Void, Integer, Integer>{
```

```
    protected Integer doInBackground(Void... v) {
```

Pendant 5 fois

s'endormir une seconde

prévenir l'affichage à chaque seconde écoulée

```
    void onProgressUpdate(Integer... v) {
```

afficher la seconde écoulée

```
}
```

L'exemple initial avec AsyncTask, plus simple... ?

```
26 public void faireQuelqueChose(View v) {  
27     new ProgressBarTask().execute();  
28 }  
29  
30 private class ProgressBarTask extends AsyncTask<Void, Integer, Integer>(  
31     @Override  
32     protected Integer doInBackground(Void... params) {  
33         try{  
34             for(int i=1;i<=5;i++){  
35                 Thread.sleep(1000);  
36                 publishProgress(i);  
37             }  
38         } catch (Exception e) {}  
39         return null;  
40     }  
41  
42     @Override  
43     protected void onProgressUpdate(Integer... result) {  
44         tv.setText(result[0] + " sec.");  
45         progress.setProgress(result[0]);  
46     }  
47 }
```

- Tout est bien qui finit bien,
 - L'UIThread gère l'IHM
 - ProgressBar progresse, *(c'est lisible, nous sommes en progrès (facile...))*
 - à chaque appel de publishProgress (l'UIThread est réactualisé avec onProgressUpdate)

Une autre version, 5 Thread * 1 sec

À chaque clic :

```
for(int i=1; i<=5; i++)  
    new ProgressBarTask().execute(i);
```

```
private class ProgressBarTask extends  
    AsyncTask<Integer,Void,Integer>{
```

```
protected Integer doInBackground(Integer... t) {  
    s'endormir une seconde  
    prévenir l'affichage t[0] à chaque seconde écoulée
```

```
void onPostExecute(Integer v) {  
    afficher la seconde écoulée  
    est appelée avec le résultat retourné par doInBackground
```

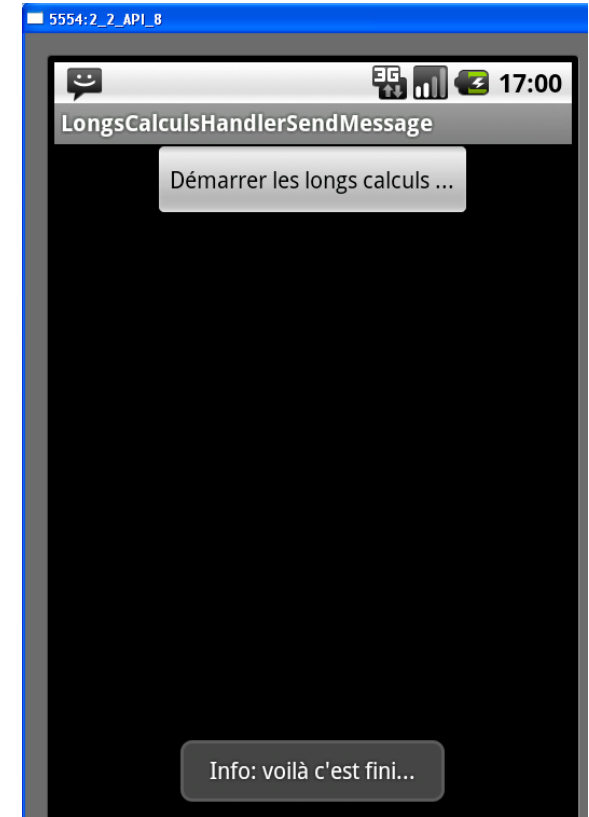
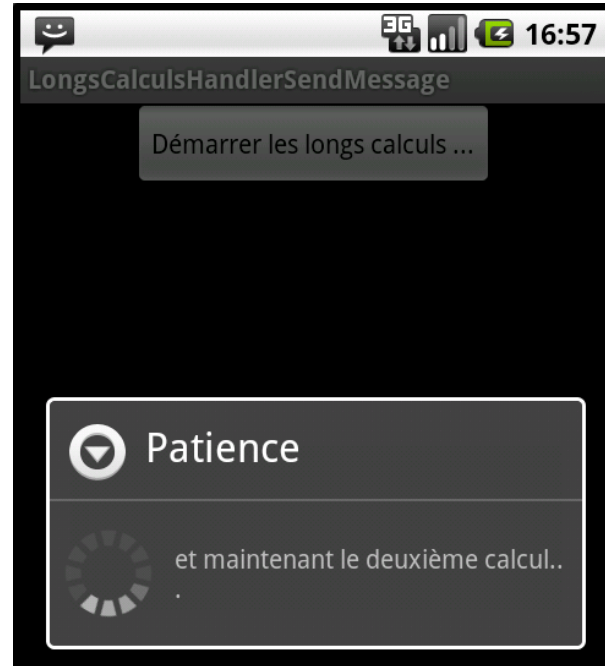
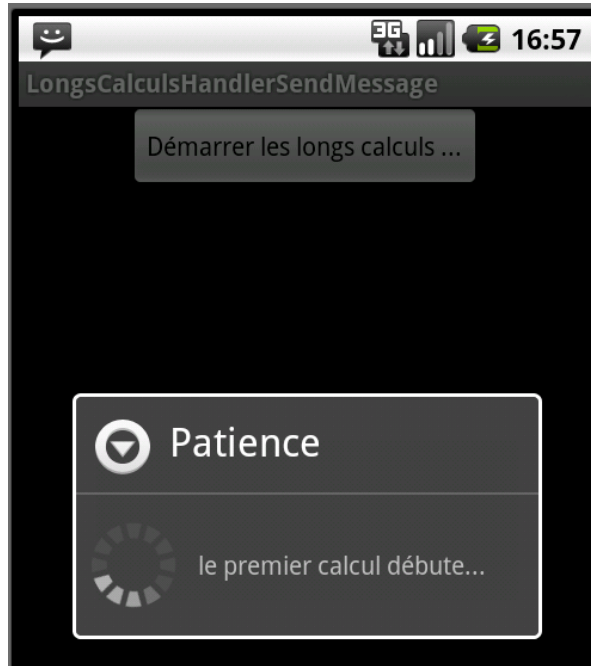

L'exemple avec AsyncTask, moins simple, en 5 Threads

```
26 public void faireQuelqueChose(View v){
27     for(int i = 1;i<=5;i++){
28         new ProgressBarTask().execute(i);
29     }
30
31 private class ProgressBarTask extends AsyncTask<Integer,Void,Integer>{
32     @Override
33     protected Integer doInBackground(Integer... params) {
34         try{
35             Thread.sleep(params[0]*1000);
36         }catch(Exception e){}
37         return params[0];
38     }
39
40     @Override
41     protected void onPostExecute(Integer result) {
42         tv.setText(result + " sec.");
43         progress.setProgress(result);
44     }
45 }
46
```

Un autre découpage en tâche élémentaire de 1 sec...

Discussion ... Démonstration

Reprenons l'exemple des longs calculs...



- **ProgressDialog et pour terminer portons un toast**

Simplissime ...

- Pas de Handler, pas de messages, ...

```
public void onClickStart(View v){  
    new LongsCalculs().execute();  
}
```

```
private class LongsCalculs extends AsyncTask<Void,String,String>{
```

```
    // diapositive suivante
```

```
}
```

AsyncTask<Void,String,String>

```
private class LongsCalculs extends AsyncTask<Void,String,String>{
    private ProgressDialog mProgressDialog;
    private Context thiss = LongsCalculsActivity.this;

    protected void onPreExecute() {
        mProgressDialog = ProgressDialog.show(thiss, "Patience",
            "de longs calculs commencent...", true);
    }

    protected String doInBackground(Void... inutilisé) {
        publishProgress("le premier calcul débute...");
        doLongOperation1();
        publishProgress("et maintenant le deuxième calcul...");
        doLongOperation2();
        return "voilà c'est fini";
    }

    protected void onProgressUpdate(String... result){
        mProgressDialog.setMessage(result[0]);
    }

    protected void onPostExecute(String s){
        Toast.makeText(getBaseContext(), "Info: " + s,Toast.LENGTH_LONG).show();
        mProgressDialog.dismiss();}}}
```

En résumé AsyncTask<Params, Progress, Result>

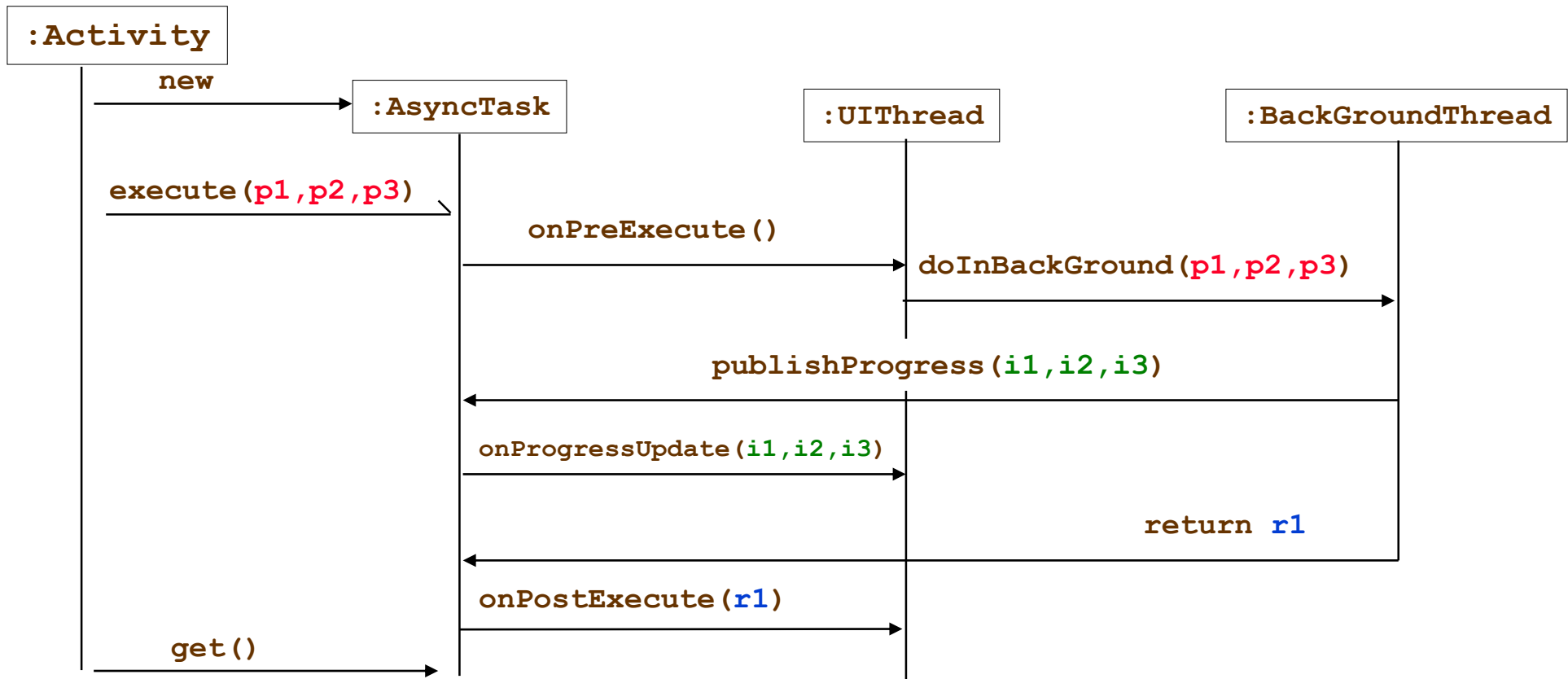
- Depuis l'UIThread
 - création d'une instance et appel de la méthode execute
 - Exemple `new WorkAsyncTask().execute(str1, str2, str3);`
- AsyncTask<Params, Progress, Result>
 - Réalise une encapsulation d'un Thread et d'un Handler

Méthodes

- onPreExecute()
 - Préambule, l'UI exécute cette méthode
- Result doInBackground(Params...p)
 - Le contenu de cette méthode s'exécute dans un autre Thread
- onProgressUpdate(Progress...p)
 - Mise à jour de l'UI à la suite de l'appel de *publishProgress*
- onPostExecute(Result)
 - Mise à jour de l'UI à la fin de la méthode *doInBackground*

<http://developer.android.com/reference/android/os/AsyncTask.html>

Demande de résultat: appel de get



- Discussions

Schéma de programme, syntaxe, AsyncTask<Params, Progress, Result>

```
public void onResult(){
```

```
...
```

```
    Boolean r = wt.get();
```

```
}
```

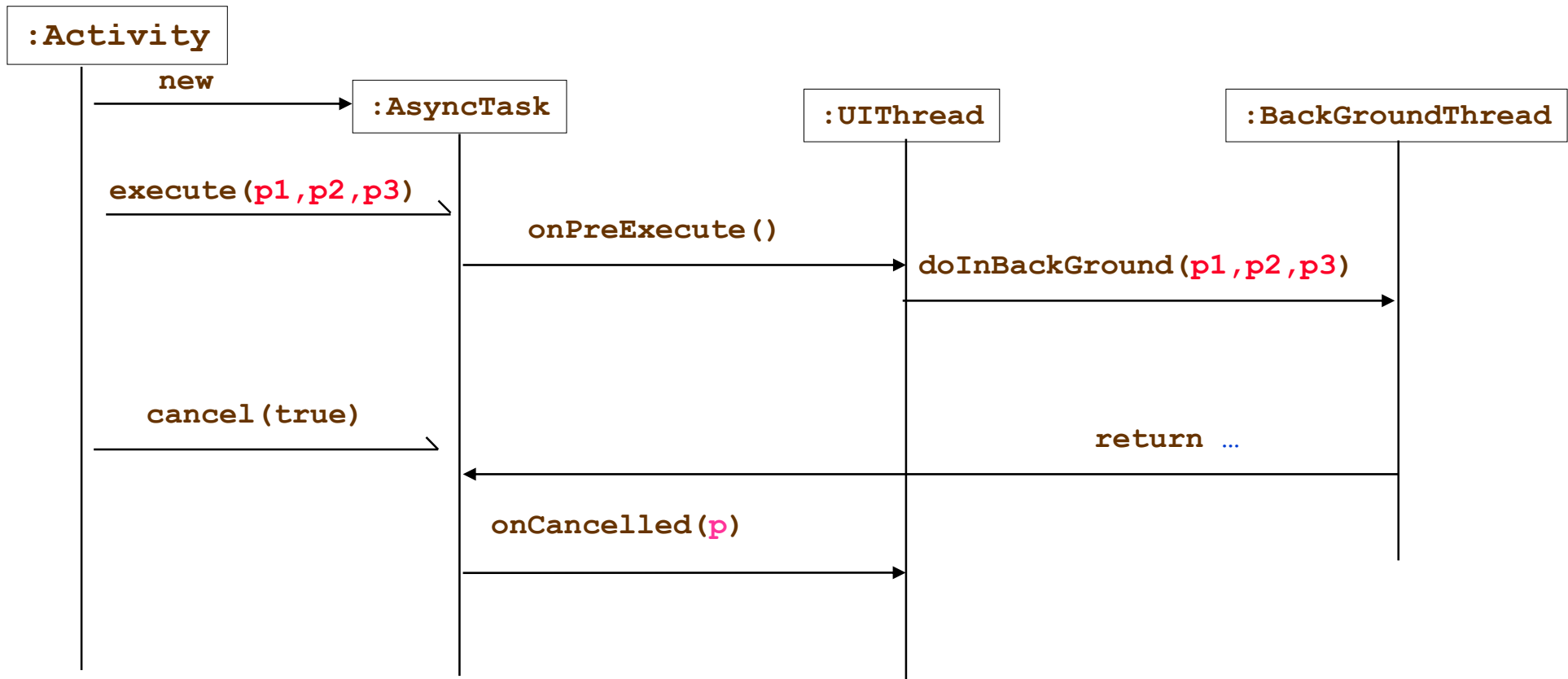
```
private class WorkAsyncTask
```

```
    extends AsyncTask<String, Long, Boolean>{
```

```
    public Boolean doInBackground(String... s) ...
```

```
}
```

Arrêt : appel de cancel(true)



- Discussions

Schéma de programme, syntaxe, AsyncTask<Params, Progress, Result>

```
public void onStop() {
    ...
    boolean mayInterruptingIfRunning = true;
    boolean r = wt.cancel(mayInterruptingIfRunning);
}

private class WorkAsyncTask
    extends AsyncTask<String, Long, Boolean>{

    public Boolean doInBackground(String... s) ...
        while(!isCancelled()) {
            ...
        }

    public void onCancelled(String s){
        // est appelée au return de doInBackground
    }
}
```

Schémas de programme

- Séquences et appels de méthodes
- `execute`
- `publishProgress`
- `cancel(true)`

Schéma de programme, syntaxe, AsyncTask<Params, Progress, Result>

```
execute(string0,string1,string2)
onPreExecute est appelée, puis
doInBackground({string0,string1,string2})
```

```
public void onStart(){
    ...
    this.wt = new WorkAsyncTask();
    wt.execute(string0, string1, string2);
}
```

```
private class WorkAsyncTask
    extends AsyncTask<String, Long, Boolean>{

    void onPreExecute() { // affichage d'un sablier...

    Boolean doInBackground(String... t){
        // t[0]/string0, t[1]/string1,...
```

Schéma de programme, syntaxe, AsyncTask<Params, Progress, Result>

```
public void onStart(){  
    ...  
    this.wt = new WorkAsyncTask();  
    wt.execute(string0, string1, string2);  
    ...  
}
```

```
private class WorkAsyncTask extends AsyncTask<String, Long, Boolean>{
```

```
    Boolean doInBackground(String... t){
```

```
        boolean resultat..  
        while(condition){ ...
```

doInBackground

publishProgress déclenche onProgressUpdate

```
            publishProgress(valeur)
```

```
        }
```

```
        return resultat;
```

```
    }
```

```
    void onProgressUpdate(Long... v) { // informer l'utilisateur que le  
                                        traitement est en cours  
                                        v[0] == valeur
```

Schéma de programme, syntaxe, AsyncTask<Params, Progress, Result>

```
public void onStart() {  
    ...  
    this.wt = new WorkAsyncTask();  
    wt.execute(string0, string1, string2);  
    ...  
}
```

```
private class WorkAsyncTask extends AsyncTask<String, Long, Boolean>{
```

```
    Boolean doInBackground(String... t) {
```

```
        return resultat;  
    }
```

*doInBackground se termine
onPostExecute est appelée*

```
    void onPostExecute(Boolean b) {
```

```
}
```

Schéma de programme, syntaxe, AsyncTask<Params, Progress, Result>

```
public void onStop(){  
    ...  
    wt.cancel(true); // true: le thread (doInBackground) peut être interrompu  
    ...  
}
```

```
private class WorkAsyncTask extends AsyncTask<String, Long, Boolean>{
```

```
    Boolean doInBackground(String... t){  
        while(!isCancelled()){  
            //  
        }  
        return resultat;  
    }
```

*onCancelled est appelée
doInBackground se termine*

```
    void onCancelled(Boolean result) {
```

```
}
```

Schéma de programme, syntaxe, AsyncTask<Params, Progress, Result>

Attention aux accès concurrents, exemple

```
private int compteur;
```

```
private class WorkAsyncTask extends AsyncTask<String, Long, Boolean>{
```

```
    Boolean doInBackground(String... t){
```

```
        while(!isCancelled()){
```

```
            compteur++;
```

si

```
        }
```

```
        return resultat;
```

```
    }
```

Alors int en AtomicInteger

```
private AtomicInteger compteur;
```

```
compteur.incrementAndGet();
```

Utilisation de
`java.util.concurrent.atomic.*`

Précautions ... AsyncTask<Params...

- Attention aux références transmises : **Params**

Méthodes exécutées dans un autre Thread

- **Result** doInBackground(**Params...** p)
- Alors un accès concurrent est possible à p[0], p[1]
 - L'idéal serait de ne transmettre que des instances immutables
 - **String, Integer, ...**
 - Ou bien utiliser java.util.concurrent.atomic.* **AtomicInteger, ...**
 - Ou encore Collections.synchronizedMap ...

Mais...



This class was deprecated in API level R.

Use the standard `java.util.concurrent` or [Kotlin concurrency utilities](#) instead.

`AsyncTask` was intended to enable proper and easy use of the UI thread. However, the most common use case was for integrating into UI, and that would cause Context leaks, missed callbacks, or crashes on configuration changes. It also has inconsistent behavior on different versions of the platform, swallows exceptions from `doInBackground`, and does not provide much utility over using `Executor` s directly.

`AsyncTask` is designed to be a helper class around `Thread` and `Handler` and does not constitute a generic threading framework. `AsyncTasks` should ideally be used for short operations (a few seconds at the most.) If you need to keep threads running for long periods of time, it is highly recommended you use the various APIs provided by the `java.util.concurrent` package such as `Executor`, `ThreadPoolExecutor` and `FutureTask`.

- <https://developer.android.com/reference/android/os/AsyncTask>

Une proposition à tester...

- Une version simplifiée d'AsyncTask vous est proposée

- Une classe apparentée AsyncTask développée pour l'occasion,
- Proposition perfectible et testée partiellement.

- Basée sur `java.util.concurrent.*`; (issue du cours NSY102)

- Schéma

```
ExecutorService executor = Executors.newSingleThreadExecutor();
```

```
Future<Result> future = executor.submit(new Background(params))
```

```
private class Background implements Callable<Result>{
```

```
    public Result call() {
```

```
        return ...
```

- Le projet AndroidStudio pour une explication

- http://douin.free.fr/AsyncTask_OS.zip

Démonstration

- **Le projet AndroidStudio pour une démonstration**
 - http://douin.free.fr/AsyncTask_OS.zip

```
private static class AsyncTask1  
extends fr.cnam.asynctask_os.AsyncTask<Integer, Integer, Float> {
```

```
//private static class AsyncTask1  
extends android.os.AsyncTask<Integer, Integer, Float> {
```

- **Le projet AndroidStudio pour les corrections, améliorations ...**
 - http://douin.free.fr/AsyncTask_OS.zip

Résumé, conclusion

- **Attention à la gestion de l'écran**
 - Laissons cet UIThread gérer tous les évènements de l'utilisateur
 - **Alors**
 - Chaque opération coûteuse en temps d'exécution se fera dans un autre Thread
 - Si ce Thread doit interagir avec l'écran (ProgressBar) alors
 - runOnUiThread, Handler représentent une solution
 - **AsyncTask<Params, Progress, Result>**
 - » Est une autre solution avec une encapsulation (réussie)
d'un thread et d'un handler
- Alors AsyncTask<Params, Progress, Result> nous préférons**
- **En annexe : Accès au Web, utilisation d'AsyncTask**

execute et THREAD_POOL_EXECUTOR

```
public void onStart(){
    ...
    this.wt = new WorkAsyncTask();
    //wt.execute( string1, string2, string3 );

    wt.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR,
        string1, string2, string3 );
    ...
}
```

Par défaut: les Threads internes créés ne sont pas en « parallèle »

```
private class WorkAsyncTask extends AsyncTask<String,Long,Boolean>{

    void onPreExecute() { // faire patienter l'utilisateur,
                          // affichage d'un sablier...

    Boolean doInBackground(String... t){ // effectuer la tâche coûteuse en temps
                                      // t[0]/string1, t[1]/string2,...
        publishProgress( 11, 12, 13 ); // déclenche onProgressUpdate

    void onProgressUpdate(Long... v) { // informer l'utilisateur que le
                                      // traitement est en cours

    void onPostExecute(Boolean b) { // le sablier disparaît,
                                    // une éventuelle erreur est affichée
    }
```

ModernAsyncTask ?

- **android-support-v4.lib**
- **android.support.v4.content.ModernAsyncTask**
 - https://github.com/android/platform_frameworks_support/blob/master/v4/java/android/support/v4/content/ModernAsyncTask.java
- À lire ce chapitre
 - <https://books.google.fr/books?id=nGzWDQAAQBAJ&lpg=PA732&ots=S7wC12Jzo1&dq=modern%20asynctask&hl=fr&pg=PA743#v=onepage&q=modern%20asynctask&f=false>

Annexe

- **Lecture d'une page HTML**
- **L'activité est détruite mais le Thread continue**
- **Outil de supervision**

Le patron Command/AsyncTask

- **Cf. Command**
- **Discussion**

AsyncTask et réseau, un exemple

- Lire une page sur le web HTTP, requête GET
 - `private class LirePageHTML extends AsyncTask<String, Void, String>{`

Schéma

`onPreExecute`

Afficher une fenêtre d'informations, `ProgressDialog`

`doInBackground`

Ouvrir une connexion, avec un échec éventuel

`onPostExecute`

Informar l'utilisateur

Lire la page www.cnam.fr

- Si j'ai la permission ... de naviguer sur le web

- `<uses-permission android:name="android.permission.INTERNET"></uses-permission>`
- Une IHM simple



- L'accès au web est une opération coûteuse alors héritons de `AsyncTask`

Une classe interne héritant de AsyncTask

```
protected String doInBackground(String... args) {
    builder = new StringBuilder();
    try {
        HttpClient client = new DefaultHttpClient();
        HttpGet httpGet = new HttpGet(args[0]);

        HttpResponse response = client.execute(httpGet);
        StatusLine statusLine = response.getStatusLine();
        int statusCode = statusLine.getStatusCode();
        if (statusCode == 200) {
            HttpEntity entity = response.getEntity();
            InputStream content = entity.getContent();
            BufferedReader reader = new BufferedReader( new InputStreamReader(content));
            String line;
            while ((line = reader.readLine()) != null) {
                builder.append(line);
            }
        } else {error = "Failed to download file";}
        } catch (Exception e) {error = e.getMessage();}

    return builder.toString();}
}
```

- <http://som-itsolutions.blogspot.fr/2011/10/android-async-task-demystified.html>
- **Le patron Command**
- **L'activité est détruite,**
 - **Que devient le Thread ?**
- **Mise au point, déploiement**

L'activité est détruite

- **Un thread est créé**
- **L'activité s'arrête, moult raisons**
- **Le thread est encore là...**

Un autre exemple, avec sauvegarde d'un thread

- **Cf. Cycle de vie d'une activité**
 - Suspension pendant une communication ...
- **Rappel**
 - Une rotation de l'écran entraîne un arrêt puis un nouveau démarrage de l'activité,
 - ce qui nécessite une sauvegarde des données de l'activité et sa restitution.
 - Il existe bien d'autres possibilités d'arrêter l'activité en cours
- **ET si un thread était en pleine activité...**
 - Lecture d'une page sur le web, calculs intenses, ...

Une solution

- Une solution
 - Classe interne et statique + Variable de classe + Thread
 - Un thread, indépendant du cycle de vie
 - Nous préférons un service
- très inspiré de
<http://www.vogella.de/articles/AndroidPerformance/article.html>

A la création

```
public class ThreadStaticActivity extends Activity {  
    private static ProgressDialog dialog;  
    private static Handler handler;  
    private static Thread horloge;
```

```
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
        handler = new Handler();
```

```
        // restitution éventuelle
```

```
        horloge = (Thread) getLastNonConfigurationInstance();  
        if (horloge != null && horloge.isAlive()) {  
            dialog = ProgressDialog.show(this, "horloge", horloge.toString());  
        }  
    }
```

Démarrage et Sauvegarde au cas où

```
public void onClickStart(View v) {  
    horloge = new Horloge();  
    horloge.start();  
    dialog = ProgressDialog.show(this, "horloge", horloge.toString());  
  
}
```

@Override

```
public Object onRetainNonConfigurationInstance() {  
    return horloge;  
}
```

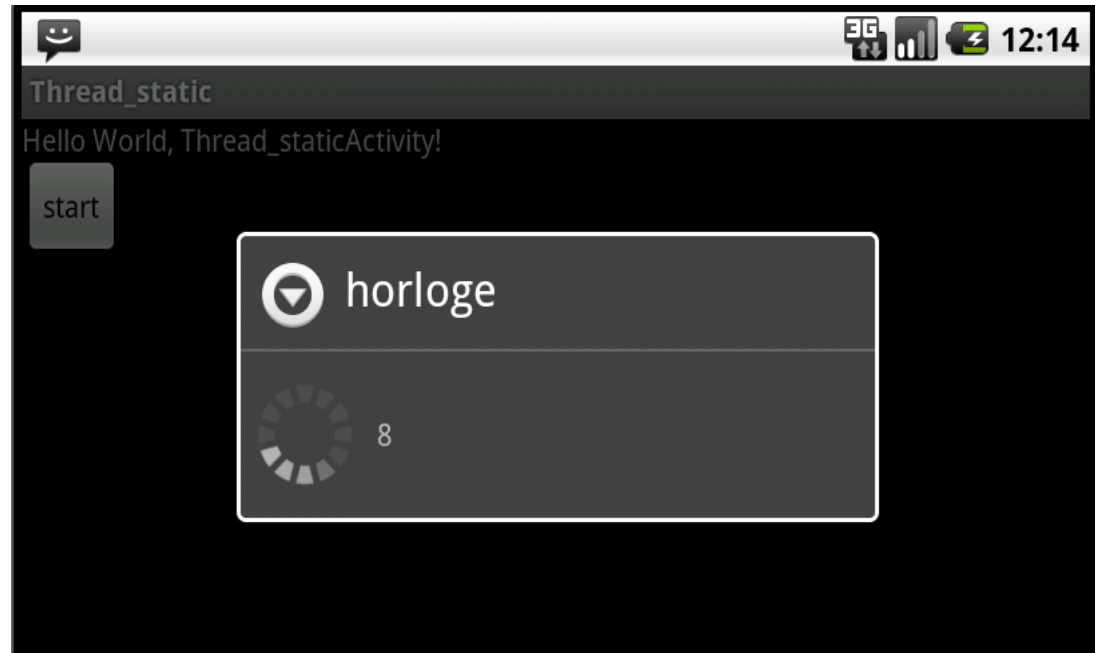
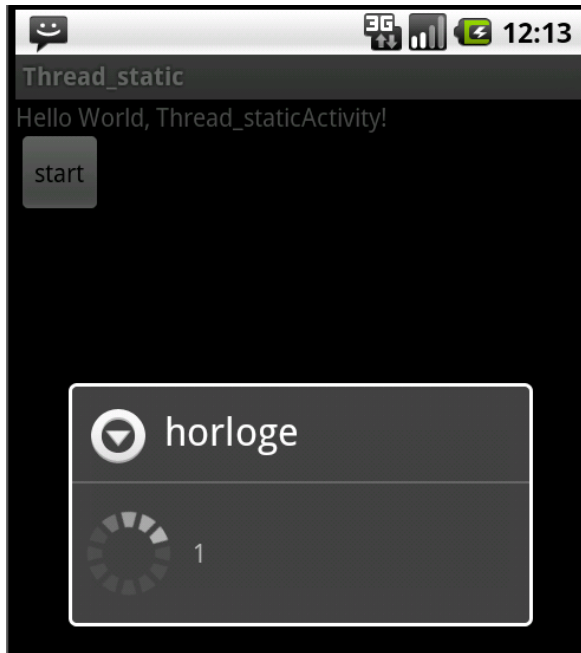
Horloge interne et statique ... discussions

```
public static class Horloge extends Thread{
    private final static int MN = 60*1000;
    private final static int PERIODE = 10*1000;
    private int compteur = 0;

    public String toString(){
        return Integer.toString(compteur);
    }
    public void run(){

        while(compteur < (3*MN) / PERIODE) {           // arrêt au bout de 3 mn
            try{
                Thread.sleep(PERIODE);
            } catch (Exception e) {}
            compteur++;
        }
        handler.post(new Runnable() {                 // dismiss dans l'UIThread
            public void run() {
                dialog.dismiss();
            }
        });
    }
}
```

Copies d'écran

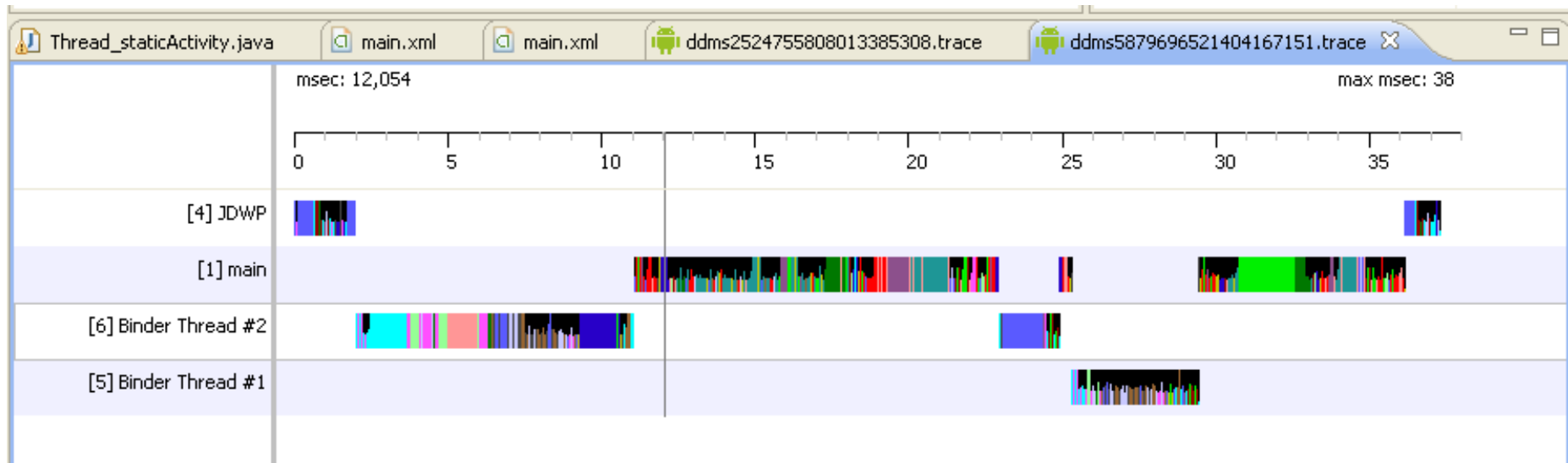


- **L'IHM est réactualisée à chaque rotation de l'écran...**
 - À chaque rotation sauvegarde du Thread ...

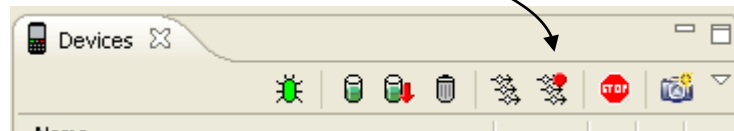
Mise au point, déploiement

- **Outils graphiques prédéfinis**
- **StrictMode**

Déploiement, débogage



- **Chaque appel de méthode est recensé**
 - **Outil traceview**



StrictMode, se souvenir

- **Se souvenir des règles d'utilisation de l'UIThread et pas seulement, StrictMode le détecte**
 - <http://android-developers.blogspot.com/2010/12/new-gingerbread-api-strictmode.html>
- **En cours de développement seulement**
 - L'application s'arrête, avec une explication trace LogCat D Debug

```
if (DEVELOPER_MODE) {  
    StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder()  
        .detectAll().penaltyLog().penaltyDeath().build());  
    StrictMode.setVmPolicy(new  
        StrictMode.VmPolicy.Builder().detectAll()  
        .penaltyLog().penaltyDeath().build());  
}
```

Ou

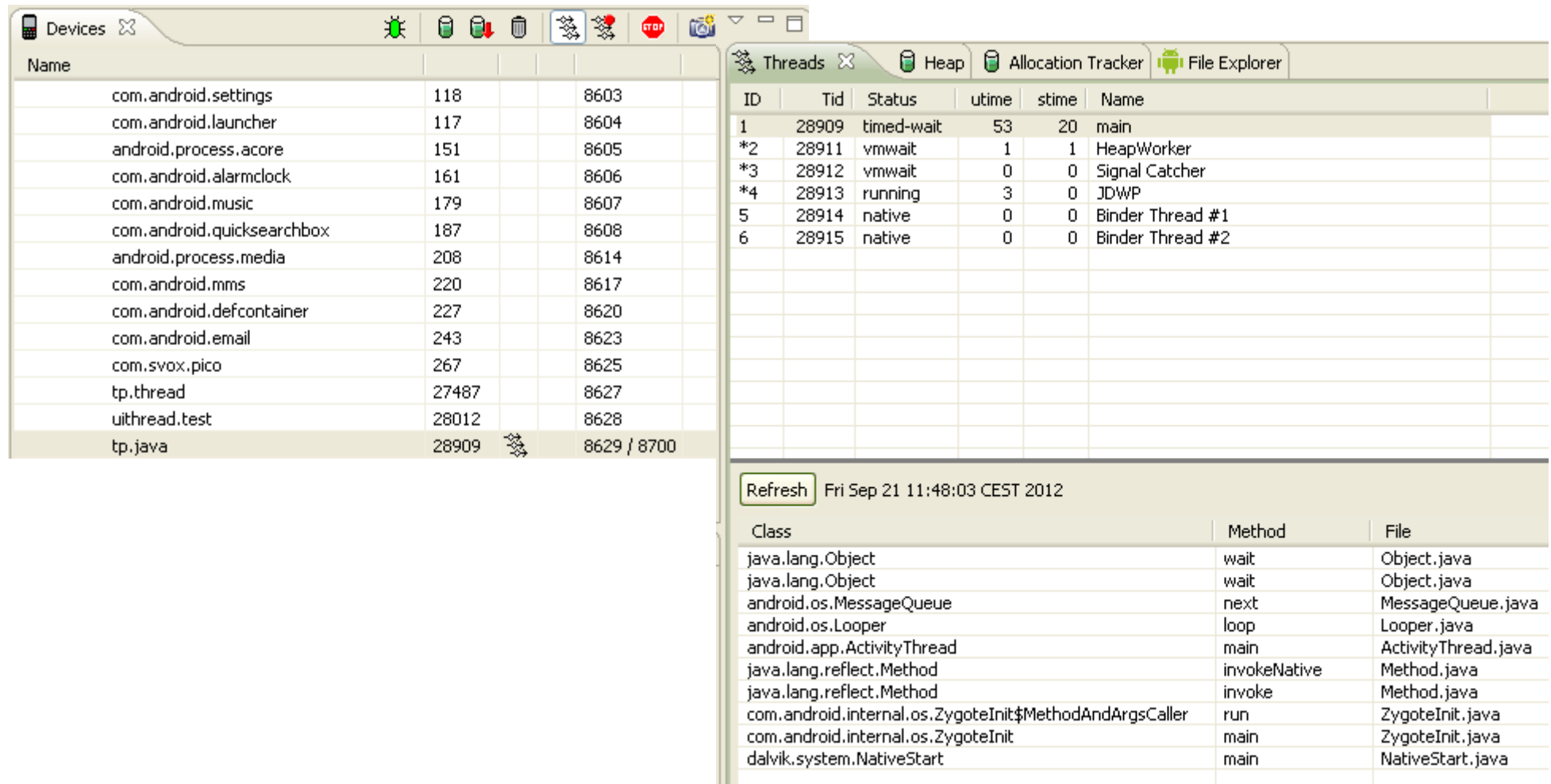
```
StrictMode.enableDefaults();
```

API_10 2.3.3

La classe Looper, en détail

- **Looper.loop()**
 - Retrait des messages de la file d'attente
 - Chaque message possède son Handler
 - Dont la méthode `handleMessage(Message msg)` est appelée
- **Suis-je dans l'UIThread ?**
 - `Looper.getMainLooper().getThread().equals(Thread.currentThread())`
 - `Looper.getMainLooper().equals(Looper.myLooper())`
- **DDMS pour en savoir plus**

DDMS



Name			
com.android.settings	118		8603
com.android.launcher	117		8604
android.process.acore	151		8605
com.android.alarmclock	161		8606
com.android.music	179		8607
com.android.quicksearchbox	187		8608
android.process.media	208		8614
com.android.mms	220		8617
com.android.defcontainer	227		8620
com.android.email	243		8623
com.svox.pico	267		8625
tp.thread	27487		8627
uithread.test	28012		8628
tp.java	28909		8629 / 8700

ID	Tid	Status	utime	stime	Name
1	28909	timed-wait	53	20	main
*2	28911	vmwait	1	1	HeapWorker
*3	28912	vmwait	0	0	Signal Catcher
*4	28913	running	3	0	JDWP
5	28914	native	0	0	Binder Thread #1
6	28915	native	0	0	Binder Thread #2

Refresh Fri Sep 21 11:48:03 CEST 2012

Class	Method	File
java.lang.Object	wait	Object.java
java.lang.Object	wait	Object.java
android.os.MessageQueue	next	MessageQueue.java
android.os.Looper	loop	Looper.java
android.app.ActivityThread	main	ActivityThread.java
java.lang.reflect.Method	invokeNative	Method.java
java.lang.reflect.Method	invoke	Method.java
com.android.internal.os.ZygoteInit\$MethodAndArgsCaller	run	ZygoteInit.java
com.android.internal.os.ZygoteInit	main	ZygoteInit.java
dalvik.system.NativeStart	main	NativeStart.java

- Méthode `Looper.loop` appelée depuis le thread `main`

extensions possibles

- **Comme serveur**
 - Un serveur web sur votre mobile ...
 - Cf. l'application PawServer
- **Déploiement & Développement**
 - Pannes possibles, le wifi disparaît, le tunnel apparaît, (*le prévoir...*)
 - StrictMode

Un serveur web sur votre mobile (wifi)

- **Attention,**
 - Lancer l'émulateur une fois connecté wifi (et non 3G)
 - depuis l'émulateur accéder au serveur installé sur le PC hôte
 - n'est pas localhost mais 10.0.2.2
 - <http://developer.android.com/guide/developing/tools/emulator.html>
 - Proxy : `Settings.System.putString(getContentResolver(), Settings.System.HTTP_PROXY, "myproxy:8080");`