Android UIThread, Thread, Handler et AsyncTask

jean-michel Douin, douin au cnam point fr
version : 21 septembre 2012

Notes de cours

Bibliographie utilisée

http://developer.android.com/resources/index.html
http://developer.android.com/reference/android/os/AsyncTask.html

Un ensemble de tutoriels à lire
http://www.vogella.de/android.html
http://www.vogella.de/articles/AndroidPerformance/article.html

StrictMode (intégré en 4.0)
http://android-developers.blogspot.com/2010/12/new-gingerbread-api-strictmode.html

http://blog.akquinet.de/2010/02/17/android-activities-the-predominance-of-the-ui-thread/

JSON
http://www.tutos-android.com/parsing-json-jackson-android
http://www.vineetgupta.com/2011/03/mobile-platforms-part-1-android/

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*
- **StrictMode**
  - **Aide au développement « correct »**

UIThread

- **Gère l'affichage de l'*Activity* :**
  - **Prise en compte des évènements de l'utilisateur**
    - **Clic, déplacements, …**
    - **Class Looper**
  - **Exécution des ***Listeners*** associés**
    - **…**
- **L'UIThread est le seul Thread « agréé »**
  - **Pour l'affichage au sein d'une activité**
    - **Les évènements 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*

Exemples, et démonstration

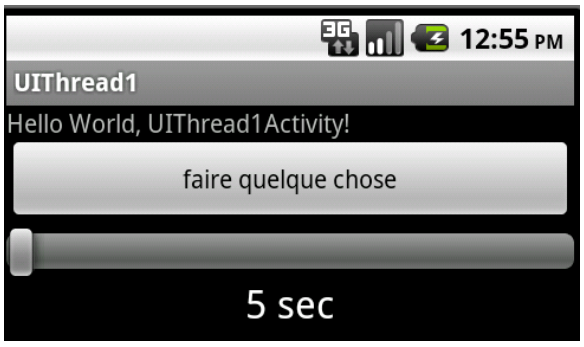
- **Exemples à ne pas suivre …**
  - **La méthode onCreate contient un appel à Thread.sleep(30000) !**
    - **Un affichage est souhaité**
    - **Thread.sleep(30000);**
    - **-> ce n'est qu' au bout de 30000ms que l'affichage se produit !!**
  - **Au clic sur un bouton :**
    - **Une modification de l'IHM est souhaité**
    - **Et Thread.sleep est appelée …**

Exemple à ne pas suivre : une IHM au ralenti

- **Cette IHM gère un clic toutes les 5 secondes**
  - **L'UIThread s'endort …**

```
 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     }
```

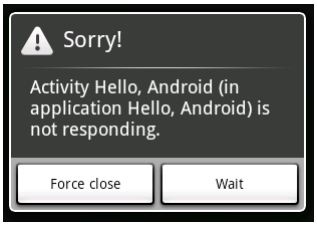

IHM au ralenti



- **C'est seulement au bout de 5 secondes**
  - **Qu'un nouveau clic devient possible,**
  - **Que la jauge est accessible (répond pas 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 !**

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)**

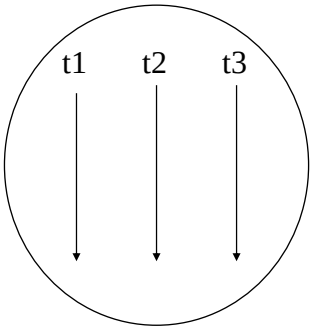


- **Ce qui arrive quand l'UIThread est bloqué plus de 5 sec.,**
- **Android considère que votre application ne répond plus**
  cf. ANR, Application Not Responding

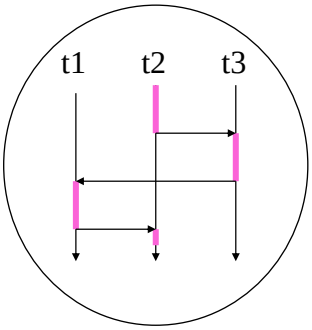
Rappel : les Threads en java

- **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();}`

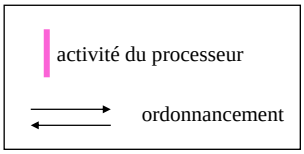
Contexte : Quasi-parallèle



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

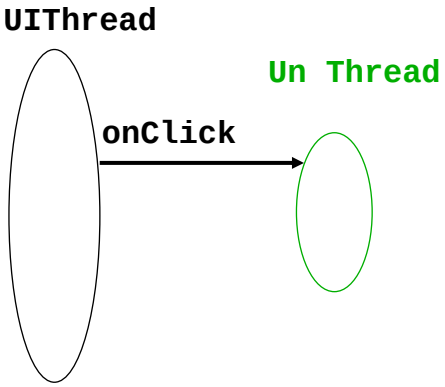
```
public class T extends Thread{
  public void run(){
    // traitement
  }
}
Thread t = new T();
t.start();


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


Runnable, autre syntaxe

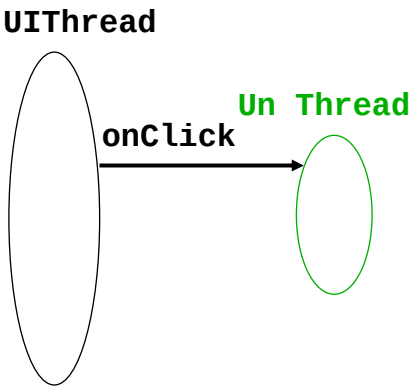
```
Runnable r = new Runnable(){
               public void run(){
                 // traitement
               }
             };

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

Reprenons l'exemple à ne pas suivre

- **Une solution serait de créer un Thread qui s'endort pendant 5 * 1sec**
- **Au clic un Thread est créé**



Au clic un thread est créé



- **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         }
```

- **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 …

Une façon de faire :

Création d'un Thread

+

Accès prédéfinis à l'UIThread

- envoi de séquences d'affichage
- envoi de messages

Accès à l'UIThread

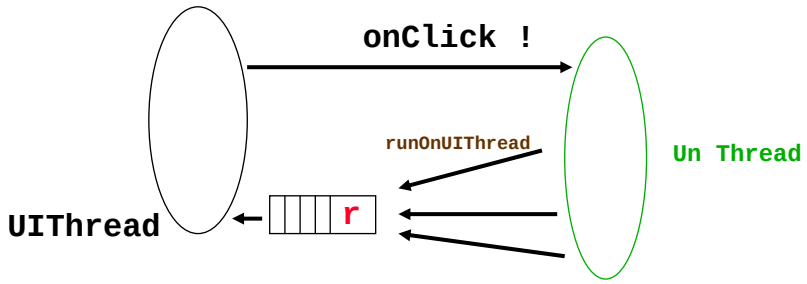
- **Envois de séquences d'affichage**
  - runOnUiThread (Runnable r){…
    - Méthode héritée de la classe Activity
  - Ou bien une instance de la classe Handler
    - unHandler.post(Runnable r){ …
- **Envois de messages**
  - Avec une instance de la classe Handler
    - unHandler.send(Message m){ …

Ou le cumul des deux

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

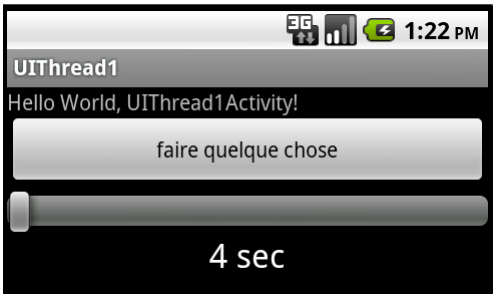


runOnUIThread

```
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
34                      }catch(Exception e){
35                      }
36                  }
37              });
38              t.start();
39          }
```

- **Tout est correct**
  - **Peu lisible,** *mais avec l'habitude…*

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



- **Ok**
- *Démonstration*

Une autre façon de faire

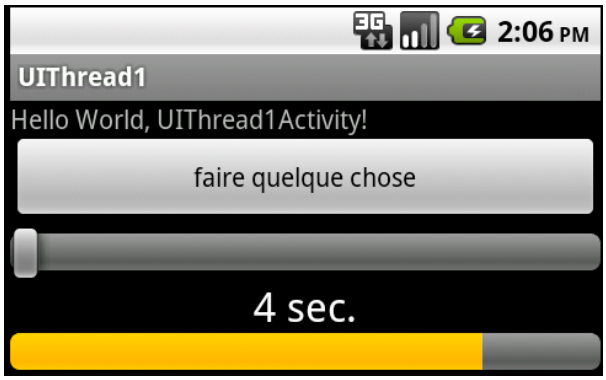
- **android.os.Handler**
  - **Permet de poster un thread dans la file de l'UIThread**
  - **Permet aussi de poster des messages à l'attention de l'UIThread**
  - **Analogue à l'usage de runOnUIThread**

Handler … idem à runOnUIThread

- **android.os.Handler**
  - **Cf. runOnUIThread**
  - **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     }
```

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



- **L'interface s'est enrichie d'un « ProgressBar**
- **<ProgressBar**
  - **ProgressBar p = (ProgressBar) find….**
  - **p.setprogress(value)**

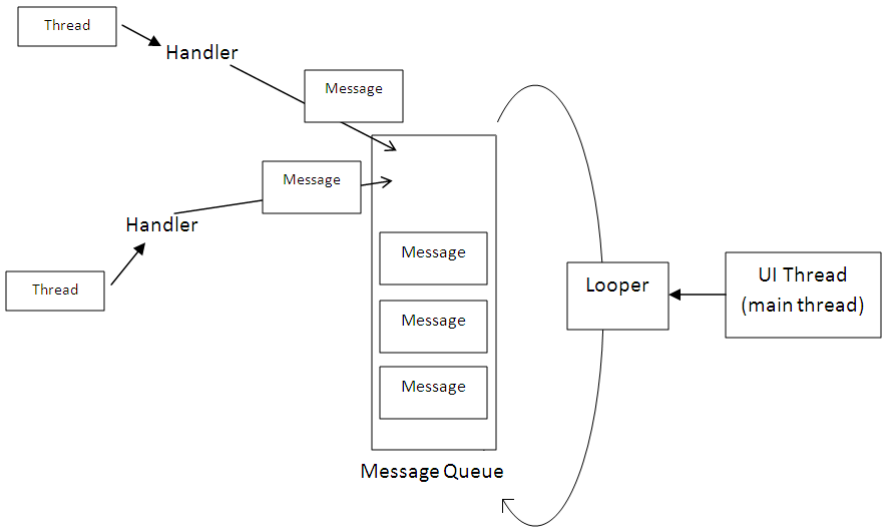
En progress, mais de moins en moins lisible…

```
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
```

Une autre façon de faire

- **Poster un message à destination de l'UIThread**
- **Par exemple**
  - **informer l'utilisateur du déroulement d'un traitement**
  - **Il faut**
    - **Obtenir un exemplaire de message**
    - **Proposer la gestion du message par une méthode du handler**
      - **handleMessage est redéfinie**

Handler et Message

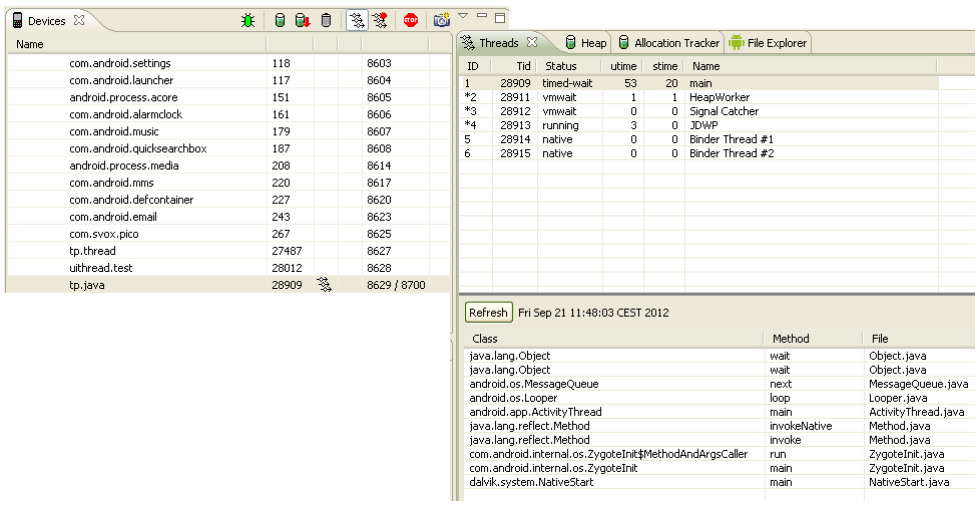


- http://www.aviyehuda.com/2010/12/android-multithreading-in-a-ui-environment/

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



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

Messages à destination de l'UIThread

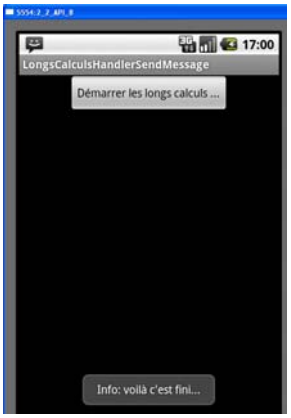
- Exemple extrait de http://www.tutomobile.fr/utiliser-une-progressdialog-dans-ses-applications-android-tutoriel-android-n%C2%B022/03/02/2011/
- Une Activity avec deux longs, très longs calculs
- Un Thread se charge des deux calculs
- Comment informer l'utilisateur tout au long des calculs ?
  - Au démarrage,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

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();
      }
}
```

Copie d'écran de ce que l'on souhaiterait



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

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 = voilàaaaaa c'est fini
// envoi d'un message à destination du handler

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

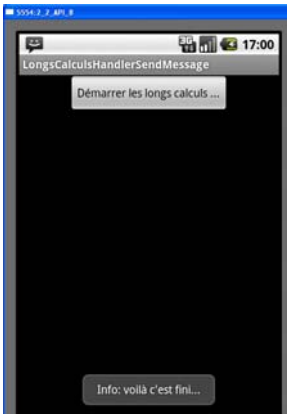
onClickStart est décoré

```
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(getBaseContext(),"Info:" +
                                   (String)msg.obj,
                                   Toast.LENGTH_LONG
               ).show();

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

Copie d'écran obtenu …



- **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 ←→ avec l'UIThread**
    - **post(un thread**
    - **sendMessage(un message**
- **Existe-t-il une classe toute prête et simplificatrice ?**
  - **UI !**
  - AsyncTask<Params, Progress, Result>

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
  - **Progress** type des paramètres en cours de traitement transmis au Handler
  - **Result** type du résultat pour l'appelant

AsyncTask< Params, Progress, Result>

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

- **Les méthodes publiques pour les clients**
  - **Une méthode pour l'utilisateur execute(param1, param2, param3)**
    - **D'autres méthodes publiques existent**
      - **cancel, get, executeOnExecutor, …**

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

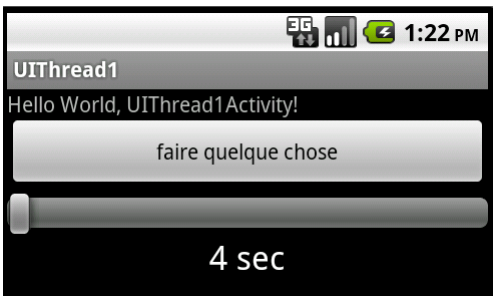
```
public void onStart(){
  …
    WorkAsyncTask wt = new WorkAsyncTask();
    wt.execute(string1, string2, string3);
  …
}


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,…

  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
}
```


class MaClasse extends AsyncTask<Params, Progress, Result>

Protected Methods	
abstract Result	doInBackground (Params... params) Override this method to perform a computation on a background thread.
void	onCancelled (Result result) Runs on the UI thread after cancel(boolean) is invoked and doInBackground(Object[]) has finished.
void	onCancelled () Applications should preferably override onCancelled(Object).
void	onPostExecute (Result result) Runs on the UI thread after doInBackground(Params...).
void	onPreExecute () Runs on the UI thread before doInBackground(Params...).
void	onProgressUpdate (Progress... values) Runs on the UI thread after publishProgress(Progress...) is invoked.
final void	publishProgress (Progress... values) This method can be invoked from doInBackground(Params...) 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 MaClasse**
    - http://developer.android.com/reference/android/os/AsyncTask.html
  - **publishProgress déclenche, appelle onProgressUpdate**
    - **La quantité de sable du sablier …**

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 prend la main 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
```

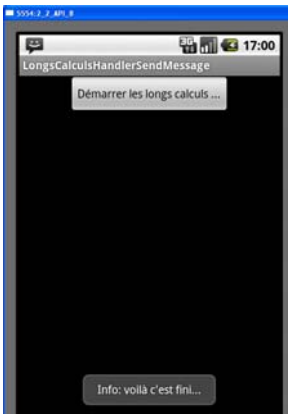
L'exemple avec AsyncTask, moins simple, en 5 Threads

```
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    }
```

Un autre découpage en tâche élémentaire de 1 sec…

Discussion ...

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>{

 // transparent suivant

}
```


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 WebAsyncTask().execute(url1, url2, url3);**

- **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*

Résumé d'étape

- **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 préfèrerons

- **La suite : deux utilisations d'AsyncTask**
  - **accès au web** *(avec pannes possibles … mobilité oblige …)*

AsyncTask et chargement d'une liste d'item

```
public class ListeActivity extends ListActivity {

 private static final String[] items=  une liste avec bcp d'éléments

 @Override
 public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main);

   setListAdapter(new ArrayAdapter<String>(
                                this,
                                android.R.layout.simple_list_item_1,
                                new ArrayList())
    );


    new AddStringTask().execute();
   }
```


AsyncTask et chargement des items d'une liste en tache de fond

```
class AddStringTask extends AsyncTask<Void, String, Void> {

protected Void doInBackground(Void... unused) {
  for (String item : items) {
    publishProgress(item);
    SystemClock.sleep(200);
  }

  return(null);
}



protected void onProgressUpdate(String... item) {
   ((ArrayAdapter)getListAdapter()).add(item[0]);
}


protected void onPostExecute(Void unused) {
   Toast.makeText(ListeActivity.this, "Done!", Toast.LENGTH_SHORT)
   .show();
}
```

AsyncTask et réseau, deux exemples

- **Ouverture d'une connexion en mode TCP**
  - Au sein de l'activity
  - private class Connexion extends AsyncTask<String,Void,Boolean>{
- **Lire une page sur le web HTTP, requête GET**
  - private class LirePageHTML extends AsyncTask<String,Void,String>{

Schéma commun

onPreExecute
    Afficher une fenêtre d'informations, ProgressDialog
doInBackGround
    Ouvrir une connexion, avec un échec éventuel
onPostExecute
    Informer l'utilisateur

Mode TCP : AsyncTask<String,Void,Boolean>{

```
protected void onPreExecute() {
       dialog = ProgressDialog.show( « Patientez »);
}

protected Boolean doInBackground(String... args) {
boolean result = true;
try{
    InetAddress addr = InetAddress.getByName(args[0]); // adresse
    int port = Integer.parseInt(args[1]);                  // port
    int timeout = Integer.parseInt(args[2]);    // délai de garde
    SocketAddress sockaddr = new InetSocketAddress(addr, port);
    this.socket = new Socket();
    socket.connect(sockaddr, timeout);
    // le flux out est une variable d'instance de l'activité
    out = new DataOutputStream(socket.getOutputStream());
}catch(Exception e){
  erreur = e.getMessage();
  result= false;
}
return result;
}
```

Mode TCP : AsyncTask<String,Void,Boolean>{

```
protected void onPostExecute(Boolean result) {
 if(!result)
        alerte(erreur);

 dialog .dismiss();
}


private AlertDialog alerte(String message){
     AlertDialog.Builder builder = new AlertDialog.Builder();

    builder.setMessage(message)
           .setCancelable(false)
           .setPositiveButton("ok", new DialogInterface.OnClickListener() {
               public void onClick(DialogInterface dialog, int id) {
                 // actualiser l'interface
               }
           });
    AlertDialog alert = builder.create();
    alert.show();
    return alert;
  }
```


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

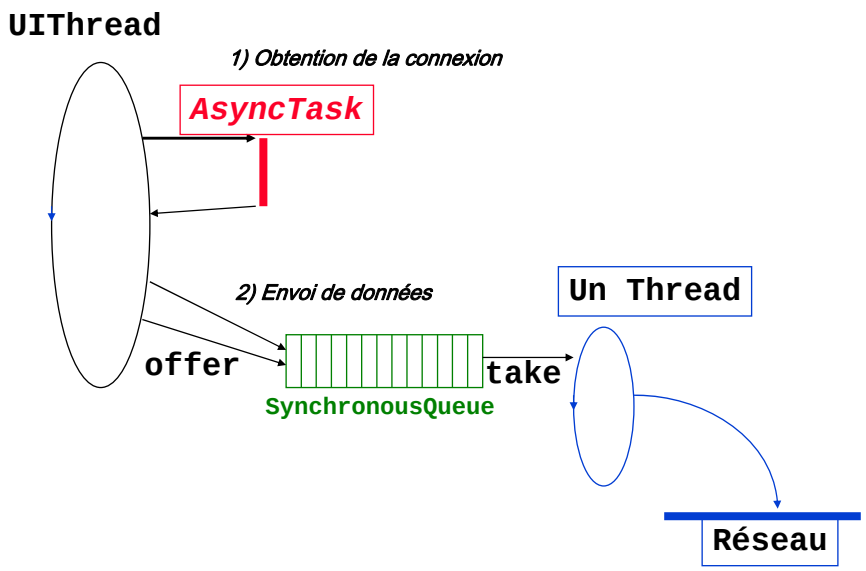
```
protected String doInBackground(String... args) {
  builder = new StringBuilder();
  try {
    HttpClient client = new DefaultHttpClient();
    HttpGet httpGet = new HttpGet(args[0]);

    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();}
```


Encore un autre exemple, essai d'architecture

- **1) Ouverture d'une connexion TCP**
  - **Obtention d'un flux (OutputStream)**
- **2) Envois de données sur le flux**
  - **En fonction des opérations de l'utilisateur**
- **Règle :**

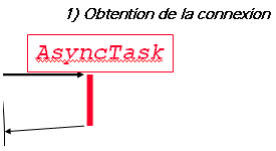
  l'ouverture de la connexion et l'envoi de données se font sur des threads
- **Une solution :**
  - Ouverture d'une connexion TCP : dans une sous classe d'AsyncTask
  - Envoi de données : dans un thread en attente sur une file (SynchronousQueue)

Un Schéma d'une architecture possible



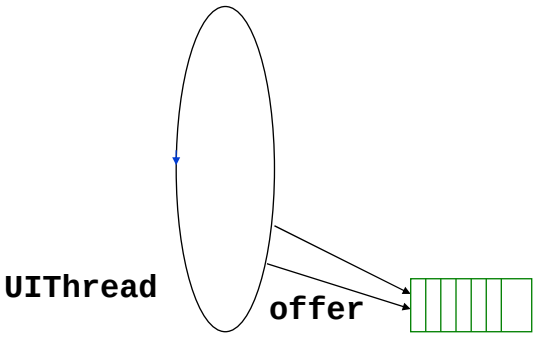
Obtention de la connexion, AsyncTask

```
protected void onPreExecute() {
   dialog = ....);
}

protected Boolean doInBackground(String... args) {
boolean result = true;
try{
   InetAddress addr = InetAddress.getByName(args[0]);
   int port = Integer.parseInt(args[1]);
   int timeout = Integer.parseInt(args[2]);
   SocketAddress sockaddr = new InetSocketAddress(addr, port);
   this.socket = new Socket();
   socket.connect(sockaddr, timeout);
   out= new DataOutputStream(socket.getOutputStream());

}catch(Exception e){ erreur = e.getMessage();result= false;}
return result;
}

@Override
protected void onPostExecute(Boolean result) {
 if(!result) alerte(erreur);
 else  // réussite
```



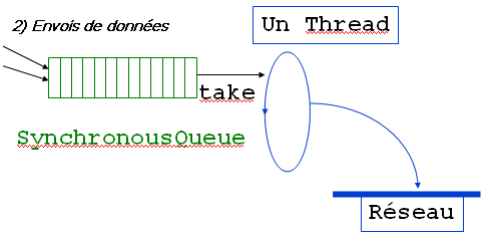
Envoi de données depuis l'UIThread

```
// ici à chaque clic des données sont envoyées vers la file

public void onClickCommand(View v){
    String cmd = v.getContentDescription().toString() + "\n";
    try {
      sender.offer(cmd.getBytes());
     } catch (Exception e) {
    }
  }
```



Envois de données, vers la file

```
public class Sender extends Thread{
  private BlockingQueue<byte[]> queue;

  public Sender(){
    queue = new SynchronousQueue<byte[]>();
    this.start();
   }

  public boolean offer(byte[] cmd){
     return queue.offer(cmd);
   }

   public void close(){
     this.interrupt();
   }

   public void run(){
     while(!isInterrupted()){
       try {
         byte[] cmd = queue.take(); // lecture bloquante
          out.write(cmd);
        }catch (Exception e) {
      }}
    }
```



Une connexion Bluetooth

- **Recherche d'un périphérique bluetooth aux alentours**
  - Hypothèse : Nous connaissons l'adresse physique du périphérique
    - 00-19-EF-01-17-9C (obtenu ipconfig /all sous windows)
  - Cette recherche doit s'effectuer dans un thread
    - Alors héritons de AsyncTask
    - Au clic
      - new ConnexionBT().execute("00:19:EF:01:17:9C");

```
private class ConnexionBT extends AsyncTask<String,String,BluetoothSocket>{

  protected void onPreExecute() {
  protected BluetoothSocket doInBackground(String... args) {
  protected void onPostExecute(BluetoothSocket btSocket) {
```


onPreExecute : Patience, doInBackground : Recherche

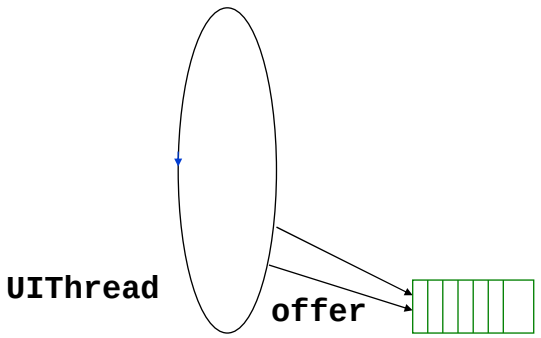
```
protected void onPreExecute() {
        dialog = ProgressDialog.show(BTClientActivity.this,
  "connexion Bluetooth", " patientez ", true);
  }

protected BluetoothSocket doInBackground(String... args) {
  try{
    this.btDevice = btAdapter.getRemoteDevice(args[0]);
    btSocket =
  btDevice.createRfcommSocketToServiceRecord(MY_UUID);
    btAdapter.cancelDiscovery();
    btSocket.connect();

  }catch(Exception e){
    erreur = e.getMessage();
    btSocket= null;
  }
  return btSocket;
}
```

onPostExecute(BluetoothSocket btSocket)

```
protected void onPostExecute(BluetoothSocket btSocket) {

  try {

     os = btSocket.getOutputStream();
     //
  } catch (IOException e) {
        erreur = e.getMessage();
        e.printStackTrace();
  }finally{
    dialog.dismiss();
  }

}
```

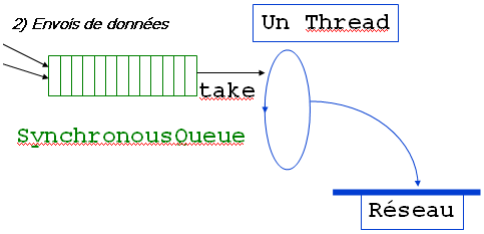

Envoi de données depuis l'UIThread, idem TCP

```
// ici à chaque click des données sont envoyées vers la file

public void onClickCommand(View v){
    String cmd = v.getContentDescription().toString() + "\n";
    try {
      sender.offer(cmd.getBytes());
     } catch (Exception e) {
    }
  }
```



Envois de données, vers la file, idem TCP

```
public class Sender extends Thread{
  private BlockingQueue<byte[]> queue;

  public Sender(){
    queue = new SynchronousQueue<byte[]>();
    this.start();
   }

   public boolean offer(byte[] cmd){
      return queue.offer(cmd);
   }

   public void close(){
     this.interrupt();
   }

   public void run(){
     while(!isInterrupted()){
       try {
          byte[] cmd = queue.take();
           out.write(cmd);
        }catch (Exception e) {
      }}
    }
```



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.
  - Et 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
  - Classe interne et statique + Variable de classe + Thread
    - Un thread, indépendant du cycle de vie
- **trés 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();
     }
   });
   }}
```

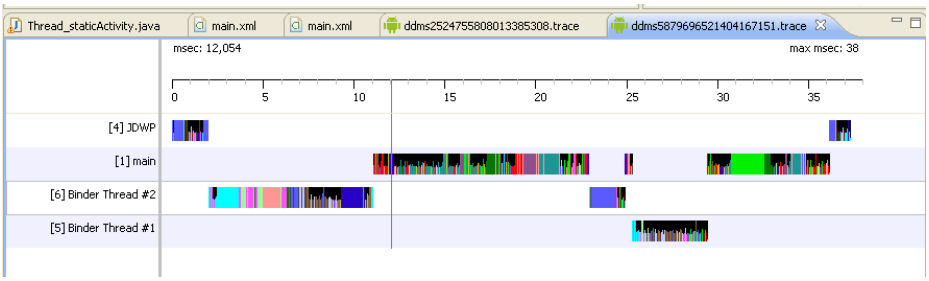

Copies d'écran

- **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

Conclusion

- **Discussions**

Annexes possibles

- **En mode TCP,comme client**
  - **XML**
    - **Usage de SAX, les stations velib**
  - **JSON**
    - **Usage org.json ou …, twitter android**
- **Comme serveur**
  - **Un serveur web sur votre mobile …**
- **Déploiement & Développement**
  - **Pannes possibles, le wifi disparaît, le tunnel apparaît,** *(le prévoir…)*
  - **StrictMode**

XML, SAX

JSON

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");