Quand la base de données est complètement installée, il est alors tentant de proposer une mise à jour des données par le réseau. On suppose qu'un serveur fournit à l'application la liste des devises et des cours à appliquer sous forme d'un fichier JSON. obtenu par une requête http.

1 Mise en place

Le rechargement sera demandé par l'utilisateur par appui sur un nouveau bouton étiqueté "RECHARGER" et qui sera associé à la méthode `void onReload(View view)` que nous allons écrire plus loin. Placez le bouton sur l'interface est insérez le squelette de la méthode `onReload`.

2 Utilisation d'un second thread

Les accès réseaux peuvent fréquemment être une occasion de ralentissement surout dans un contexte mobile. Pour préserver une expérience utilisateur acceptable, il est déconseillé (et même interdit sur les version récentes de l'OS) de réaliser une opération réseau dans le thread principal. La première étape sera de mettre en œuvre un second thread dans votre application.
Il existe deux outils pour utiliser des threads secondaires sous Android :
- Un thread Java traditionnel (utilisation de la classe `Runnable`) assez lourd mais toujours applicable.
- L'utilisation d'une classe spécifique `AsyncTask` qui masque les détails d'implémentation mais qui ne peut s'utiliser qu'en parrallèle avec le thread principal UI d'une application. C'est exactement notre cas d'utilisation. Nous préférerons donc cette seconde technique.

2.1 Description

La définition du thread à réaliser est faite en surclassant la classe de base `AsyncTask<T1,T2,T3>`.La nouvelle classe sera impliquée dans le calcul de trois façons différentes. Pour chacune de ces trois modalités, une information d'un type générique (nous noterons ces types *T1*,*T2*et *T3* et le thread secondaire.
- Au lancement.Un appel à la méthode `execute(T1 param)` demande le lancement du thread. Ceci appellera les méthodes `onPreExecute()` puis `doInBackground(T1 param)` qui lancera le calcul proprement dit. Ce calcul peut être paramètré par une valeur de type T1.
- à tout moment au cours du calcul, la tache secondaire peut appeler `publishProgress(T2 progress)` pour signaler à l'utilisateur la progression de la tâche (barre de progression, compteur...). Ceci appellera la méthode `onProgressUpdate(T2 progress)` qui est la seule méthode de la nouvelle classe habilitée à faire des opérations sur l'interface utilisateur. Etant donné que cela demande un accès au contexte, je vous propose de faire une déclaration interne (inner class) à la classe `MainActivity` pour la nouvelle classe.
- En fin de calcul, la méthode `doInBackgroud` retourne une valeur de type T3. au thread principal.

L'ensemble de ces opérations est résumé figure 1.

2.2 En pratique

Les information à échanger sont les suivantes :
- L'url à charger (`String`).
- L'indicateur de progression (aucun, `Integer` par défaut).
- la réponse du serveur (`String`)

Il faudra donc définir une classe (appelée `AsyncGet` qui surcharge `AsyncTask<String, Integer, String>`
A titre d'essai, je vous propose d'écrire une classe qui ne fait aucun calcul mais effectue un décompte (en cinq étapes de 2 secondes). Nous réaliserons la classe qui fait la requête http proprement dite dans une section ultérieure. Voici les étapes à appliquer :

1. Création de la classe interne (à `MainActivity`) `AsyncGet` qui dérive de `AsyncTask<String, Integer, String>`
2. Surcharge des méthodes suivantes :
   (a) `void onPreExecute()` inutile dans ce cas.
   (b) `void on PostExecute()` inutile dans ce cas.
   (c) `void onProgressUpdate(Integer progres)` permet d'afficher la progression. Pour notre exeample un simple appel à `Toast.makeText(context, message, LENGTH_SHORT).show()` affiche un pop-up. Faites ce qu'il faut pour que le message donne la valeur du compteur (progres). Le contexte est obtenu par `MainActivity.this`.
   (d) `T3 doInBackGround` est constituée d'une boucle de 5 itérations. à chaque itération un appel à la mthode `publishProgress(Integer)` est réalisé de même qu'une pause de 2 secondes (`THread.sleep(2.0)`.

5-1.png

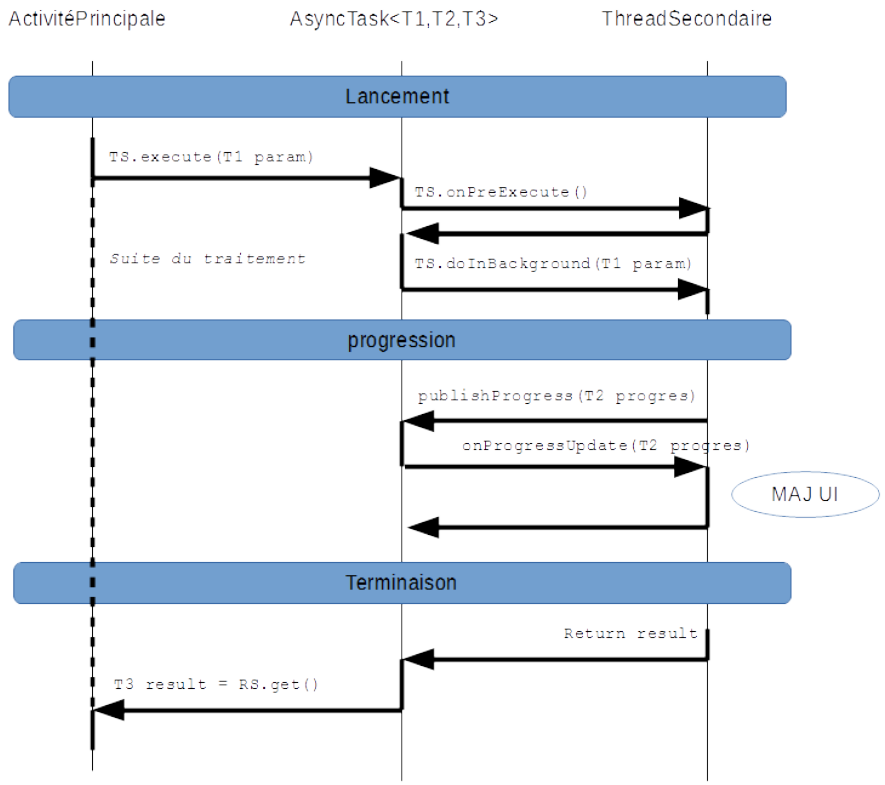

FIGURE 1 – diagramme d'activité lors de l'appel d'un thread secondaire via AsyncTask

3. Dans le Thread principal (dans notre cas, dans la méthode `onReload` :
   - Créez un objet de la classe `AsyncGet`.
   - Appelez la méthode `execute(null)` de cet objet.
   - Si un résultat devait être récupéré, un appel à la méthode `get` retourne le résultat du calcul secondaire (attention une exception peut être générée par cet appel. Il conviendra de la traiter).

2.3 Réalisation et tests

Vérifiez que lors de l'appui sur le bouton 'Recharger', un décompte s'affiche. Vérifiez que ce décompte n'empêche pas le fonctionnement habituel de l'interface (choix des devises, conversion,...) par l'utilisateur.

3 Récupération du fichier

Nous allons compléter le dispositif existant par l'accès proprement dit au serveur. Celui-ci se caratèrise par la fourniture d'une chaîne de caractère (l'url du serveur) et la récupération de la réponse (également de type `String`) qui devrait être du JSON.

3.1 Mise en place du serveur

Nous allons simuler l'existence du serveur par la création d'un fichier de nom `devises.json` dans votre répertoire personel `public_html`. Dans ces conditions, une requête http GET vers http ://*votrepc*/ *votrelogin*/devises.json doit retourner le contenu du fichier.
Note : Pour les appareils émulés, l'ordinateur hôte est accesible via le numéro ip 10.0.2.2
Le contenu du fichier est le suivant (vous constaterez le changement des taux et l apparition de nouvelles devises) :

```
{"devises":[
{ "nom" : "EUR", "valeur" : 1.7},
{ "nom" : "USD", "valeur" : 6.0},
{ "nom" : "GPB", "valeur" : 6.0},
{ "nom" : "JPY", "valeur" : 6.0},
{ "nom" : "CAD", "valeur" : 7.0},
{ "nom" : "AUD", "valeur" : 8.0},
```

```
{ "nom" : "RUB", "valeur" : 9.0}
]}
```

3.2 Réalisation de l'objet `Downloader`

La première chose à faire lorsque l'on veut qu'une application accède au réseau est d'inscrire la demande d'autorisation dans le manifeste (`AndroidManifest.xml`. La ligne correspondantes est :

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

à insérer juste avant l'entité `<application...`
La totalité de l'accès sera encapsulé dans un objet de la classe `MyDownloader` que nous allons créer. Cette classe utilisera un attribue de type `AndroidHttpClient` (là aussi, il existe plusieurs manière d'aboutir au résultat en fonction des version d'OS. Nous utiliserons la plus directe). Attention, l'instance de client devra être obtenue par un appel à la méthode statique `newInstance` plutôt que par l'appel d'un constructeur. L'accés sera déclenché par un appel à la méthode `execute` qui demande un paramètre de classe `HttpGet` (c'est là que seront placé les paramètres de la requête http) :

```
AndroidHttpClient client = AndroidHttpClient.newInstance(Build.MODEL) ;
HttpGet requete = new HttpGet(url) ;
...
reponse = client.execute(requete, new BasicResponseHandler());
...
```

3.3 Intégration et tests

Tout en gardant le squelette de la section précédente, remplacer les instructions existantes par celle qui permettent d'appeler la requête http.
Pour tester, je vous propose d'inclure dans le thread principal (donc après la récupération du résultat de l'`AsyncGet`) la reponse obtenue qui devrait être le contenu du fichier.

4 Analyse JSON et mise à jour de la base

4.1 Objets JSON

Une fois la chaîne de caractère recueillie, il faut l'analyser pour en extraire les informations à inclure dans la base. Nous disposoins pour cela des classes JSONObject et JSONArray qui disposent d'un constructeur qui effectue cette analyse. On récupérera ainsi un Objet JSON qui contiendra un tableau JSON sur lequel nous pourrons effectuer les itérations adéquates. Nous aurons donc la construction suivantes :

```
JSONObject data ;
try {
   data = new JSONObject(json);
   / json est la chaîne de caractère obtenue par la requete http
   JSONArray devises = data.getJSONArray("devises") ;
   for (int i = 0 ; i < devises.length() ; i++) {
       JSONObject devise = (JSONObject) devises.get(i) ;
       String nom = devise.getString("nom") ;
       double valeur = devise.getDouble("valeur") ;
//                INSERT INTO devises VALUES (devise.nom, devise.valeur) ;
...
   }
   } catch (JSONException e) {
...
}
return ;
```

Dans la structure précédente, chaque itération sur une devise aura pour objet de remplacer dans la b

4.2 Mise à jour de la base

Je vous propose d'effacer complètement la tabel devises, puis d'ajouter chaque devise obtenue par l'objet JSON comme une nouvelle ligne.

4.3 Intégration

On n'oubliera pas de mettre à jour l'affichage des spinner, soit en reconstruisant complètement les ou les adapters, soit par appel à la méthode `notifyDataSetChanged`