Android TD 6
AsyncTasks

Par défaut une `Activity` réside dans un thread et si ce thread effectue une opération longue telle qu'un calcul ou le chargement de données, cela bloque la totalité de l'interface. Pour tester ce problème il suffit de simuler une attente à l'aide de la méthode `sleep` qui provoque une pause du thread courant :

```
try {
  Thread.sleep( 5000 );
}
catch( Exception e ) {
  Log.e( "AsyncTaskTest", "Exception␣" + e );
}
```

Le thread principal est celui qui s'occupe de l'interface graphique. Afin de pouvoir par exemple charger des données et en même temps afficher des informations sans bloquer l'interface il est possible d'utiliser des tâches asynchrones `AsyncTasks` :

— `http://developer.android.com/guide/components/processes-and-threads.html`
— `http://developer.android.com/reference/android/os/AsyncTask.html`

Les `AsyncTasks` permettent de lancer un thread indépendant du thread principal et de communiquer avec le thread principal lorsque la tâche est terminée, ou pendant la progression de la tâche, par exemple pour afficher une barre de progression ou mettre à jour une `ListView`.

1 Principe

Cet exercice présente le principe de fonctionnement d'une `AsyncTask`.

1. Créer un nouveau projet avec une interface comportant 2 `Buttons` et un `TextView`. Le premier bouton doit afficher un `Toast` et le second doit lancer l'`AsyncTask`. Une fois terminée, l'`AsyncTask` modifiera le texte du `TextView`.
2. Dans la classe de votre `Activity` ajouter une classe fille héritant de la classe `AsyncTask` :

```
private class AsyncTaskTest extends AsyncTask<Void, Void, Void>
{
  protected Void doInBackground(Void... params)
  {
    try {
      Thread.sleep(5000);
    }
    catch(Exception e) {
      Log.e("AsyncTaskTest", "InterruptedException" + e );
    }
    return null;
  }

  protected void onPostExecute( Void result )
  {
    ( ( TextView )findViewById( R.id.tv0 ) )
                  .setText("AsyncTask␣terminated");
    findViewById( R.id.bt0 ).setEnabled( true );
  }
}
```

Lors du lancement de la tâche, la méthode `doInBackground` est lancée dans un autre thread qui est mis en pause pendant 5 secondes. Lorsque la tâche est terminée, la méthode `onPostExecute` est appelée et met à jour le texte affiché et réactive le bouton.

3. Lors du clic sur le bouton de lancement on désactive le bouton, on affiche un message indiquant le lancement, et on lance l'AsyncThread à l'aide de la méthode execute :

```
public void onClick( View view )
{
  view.setEnabled(false);
  ( ( TextView )findViewById( R.id.tv0 ) )
                  .setText("AsyncTask terminated");
  new AsyncTaskTest().execute();
}
```

La méthode execute n'est pas bloquante et on peut ajouter du code ensuite.

2 onProgressUpdate

En suivant l'exemple de la documentation http://developer.android.com/reference/android/os/AsyncTask.html, modifier votre code et remplacer l'attente de 5 secondes par une boucle effectuant des attentes de 1 seconde. Dans cette boucle, placer également un appel à la méthode publishProgress qui prendra en paramètre la valeur de votre compteur de boucle. Cette méthode provoque l'appel à la méthode onProgressUpdate par le thread principal qui peut mettre à jour le message afin d'afficher la progression.

```
private class AsyncTaskTest extends AsyncTask<Void, Integer, Void>
{
  AsyncTaskTest()
  {
    ( ( TextView )findViewById( R.id.tv0 ) ).setText("AsyncTask started...");
  }
  protected Void doInBackground(Void... params)
  {
    try {
      for( int i = 1 ; i < 6 ; ++i ) {
        Thread.sleep(1000);
        publishProgress(i);
      }
    }
    catch(Exception e) {
      Log.e("AsyncTaskTest", "InterruptedException" + e );
    }
    return null;
  }

  protected void onProgressUpdate( Integer... progress )
  {
    ( ( TextView )findViewById( R.id.tv0 ) ).setText("Progress = " + progress[0] );
  }

  protected void onPostExecute( Void result )
  {
    ( ( TextView )findViewById( R.id.tv0 ) ).setText("AsyncTask terminated");
    findViewById( R.id.bt0 ).setEnabled( true );
  }
}
```