Vision d'ensemble

Chantal Keller

25 septembre 2015

Plan

Projet

- Voir le sujet en ligne
- **Binômes ?**

Planning

Aujourd'hui :

- cours : pas de nouveaux concepts, mais un autre angle de vue
  - vision d'ensemble (recul)
  - sur Android en général
  - sur les tâches de fond
- TD : serveur pour consolider

Semaine prochaine :

- changement d'air (probablement : applications web)

Rendu du client du TD4

L'application crashe lorsqu'on clique sur `LOGOUT`

- ordre des opérations : `logout` appelle `finish` en premier puis arrête la boucle infinie seulement après
- respect du protocole : envoi de `"LOGOUT" + login` au lieu de `LOGOUT`

Le serveur envoie `Unknown command`

- respect du protocole : envoi de lignes vides (`writer.println(text + "\n")`)

↪ Autres questions dans la suite de ces transparents

Plan

La surcharge...

Ingrédient fondamental d'Android :

- L'API d'Android liste des méthodes (parfois sans contenu)
- Android appelle ces méthodes à des moments précis de l'exécution d'une application
- Au développeur de les **surcharger** pour effectuer les actions qu'il souhaite à chacun de ces moments
- Dans des **classes héritières**

Exemples :

- `Activity` : `onCreate`, `onStart`, `onStop`, ..., `onActivityResult`
- `AsyncTask` : `doInBackground`, `onProgressUpdate`, `onPostExecute`, ...

↪ commencent souvent par `on` ou `do`

... mais pas que

Méthodes à utiliser (*alias* appeler) :

- L'API d'Android fournit également des méthodes pour effectuer des actions
- Au développeur de les **utiliser** pour effectuer les actions correspondantes
- Sur des **objets de la classe**

Exemples :

- `Activity` : `findViewById`, `setContentView`, `startActivity`, ...
- `TextView` : `setText`, `setTextSize`, ...
- `Intent` : `putExtra`, `getIntExtra`, ...
- `AsyncTask` : `execute`, `cancel`, `isCancelled`, ...

Syntaxe (par l'exemple)

Surcharge :

```
class MaTache extends AsyncTask {
  ...
  @Override        // facultatif
  protected void onPreExecute () {
    ...
  }
  ...
}
```

Utilisation :

```
...
void maMethode() {
  ...
  maZoneDeTexte.setText("Hello world!");
  ...
}
...
```

Plan

La classe AsyncTask

But de AsyncTask :

- programmer des **tâches de fond**
- sans avoir le niveau de généralité des threads

Avantages et inconvénients

Avantages :

- interface haut niveau : quelques méthodes à surcharger
- points de synchronisation : `onPreExecute` puis `doInBackground` puis `onPostExecute`

Inconvénients :

- pas le niveau de détail des threads
- pas de parallélisme :

```
maTache1.execute();
maTache2.execute();
```

`maTache2.doInBackground` appelée que lorsque `maTache1.doInBackground` a fini

Calculs longs en tâche de fond

Exemples de calculs longs (voire... très longs) :

- ouverture de sockets réseau
- boucle infinie pour écouter sur un réseau
- actions bloquantes : attente d'un évènement extérieur

Exemples d'actions bloquantes :

- `reader.readLine()` : attend qu'une ligne de texte arrive dans le flux entrant
- `serverSocket.accept()` : attend qu'un client se connecte à la socket serveur

Interruption d'une tâche

Lancer une interruption quand :

- l'utilisateur quitte l'activité
- l'utilisateur appuie sur un bouton
- ...
- méthode : `maTache.cancel(true)` (`true` indique qu'on arrête et la tâche, et le thread sous-jacent)

Lorsqu'une interruption a été lancée :

- la tâche doit terminer son long calcul
- elle peut ensuite encore effectuer quelques opérations (ex : fermer une socket)
- ex dans le corps d'une boucle infinie : `if (isCancelled()) { break; }`

Plan

TD

Serveur de chat :

- point de vue inverse du client
- attend qu'un client se connecte, puis attend ses messages, et agit en conséquence
- mêmes outils : flux entrant et sortant d'une socket, tâches de fond, boucles infinies...

↪ pour mercredi 30 septembre 23h59 : "À vous de jouer" au moins jusqu'à la question 4 incluse