[Initiation à Android]

Télécom Bretagne
Décembre 2017

Horacio Gonzalez

Google Developers Experts

Finist Devs

Horacio Gonzalez

@LostInBrittany

Spaniard lost in Brittany,
developer, dreamer and
all-around geek

Cityzen Data

http://cityzendata.com

WARP 10

Finist Devs

Breizh C@mp
Mix de technologies

Google Developers Experts 2015
Web Technologies GDE

{ Service }

Services

- Composants tournant en tâche de fond
  - Sans interface utilisateur
  - Pouvant réagir aux événements, mettre à jour des contenus, effectuer des traitements…
  - Pouvant notifier de leur état à l'utilisateur
  - Avec un cycle de vie similaire aux activités
  - Contrôlables depuis d'autres applications
    - (activité, service et Broadcast Receiver)
- Beaucoup d'applications ont activités et services
  - L'activité permet à l'utilisateur de contrôler le service
  - Le service tourne en permanence
    - Par exemple pour aller chercher des informations

Code d'un service

Les services peuvent redéfinir quatre méthodes : `onCreate()`, `onStart()`, `onBind()` et `onDestroy()`

- Les services démarrés doivent implémenter `onStart()`
- Les services liées doivent implémenter `onBind()`
  - Cette méthode retourne un objet IBinder qui permet au mécanisme IPC de fonctionner

IPC : communication inter-processus, permettant d'appeler des méthodes distantes

```
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
public class MonService extends Service {

	@Override
	public void onCreate() {
		// Placez ici le code qui sera exécuté lors
		// de la création de ce service
	}

	@Override
	public void onStart(Intent intent, int startId) {
		// Placez ici le code qui sera exécuté à chaque
		// démarrage du service
	}

	@Override
	public void onDestroy() {
		// Placez ici le code qui sera exécuté lors
		// de la destruction de ce service
	}

	@Override
	public IBinder onBind(Intent arg0) {
		return null;
	}
}
```

{Threads}

Programmation concurrente

- Par défaut chaque application s'exécute dans un seul processus
  - Cela peut-être modifié dans le Manifeste

```
<application>
  <activity>.../activity>
  <service>...</service>
  <service android:name=".MyService" android:process=":p2">
</application>
```

- Le processus peut créer des threads différents

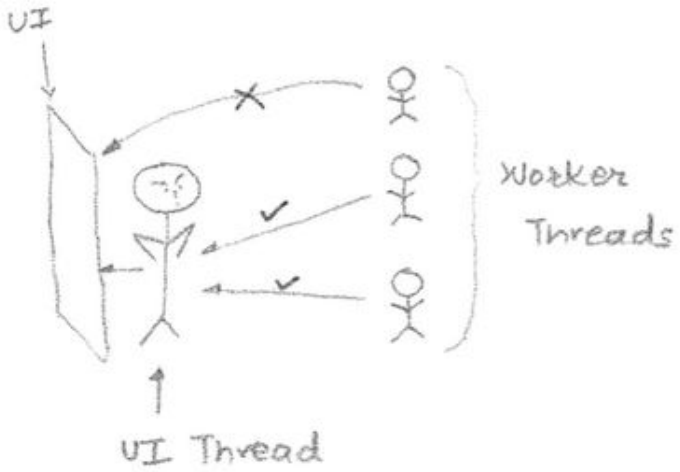
Le Thread UI

Thread unique sur lequel l'application tourne au démarrage

- Si ce thread reste bloqué
  - L'interface d'utilisateur de l'appli reste bloqué
  - Le système peut arrêter l'application

Toute opération potentiellement bloquante ou simplement lente doit se faire dans d'autre thread

- Le Thread UI doit servir qu'à la gestion des composants graphiques
- Appels réseau, consultation données, logique métier... sur d'autres threads



Multi-threading : les *Async Tasks*

Depuis le Thread UI

- On peut créer des threads traditionnels Java (implements Runnable)
- On peut utiliser des tâches asynchrones (*AsyncTask*)

Async Task

- L'activité appelle la méthode `execute()` de l'*AsyncTask*
- Le code à exécuter dans le nouveau thread est à mettre dans la méthode `doInBackground(Object... params)`

Async Tasks

Ca cache la compléxité des threads

- Pour chaque action asynchrone on crée une classe heritant d'`AsyncTask`
  - Surchargeant au moins `doInBackground(Object... params)`
  - Souvent surchageant aussi `onPostExecute(Result)`

- La classe a trois paramètres `<U,V,W>`
  - `U` : le type du paramètre envoyé à l'exécution
  - `V` : le type de l'objet permettant de notifier de la progression
  - `W` : le type du résultat de l'exécution

Async Tasks

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            publishProgress((int) ((i / (float) count) * 100));
            // Escape early if cancel() is called
            if (isCancelled()) break;
        }
        return totalSize;
    }

    protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }

    protected void onPostExecute(Long result) {
        showDialog("Downloaded " + result + " bytes");
    }
}
```


{Réseau}

Permissions et état du réseau

- Pour pouvoir accéder au réseau l'appli doit demander ces permissions
  - Dans le Manifeste

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

- Elle doit ensuite vérifier que le réseau est disponible
  - Dans l'Activité qui veut se connecter

```
ConnectivityManager connMgr =
      (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
if (networkInfo != null && networkInfo.isConnected()) {
        // fetch data
} else {
        // display error
}
```

[LiveCoding]

[Hello RESTful World!]

LiveCoding : Hello RESTFful World

- Objectif : Première application réseau
  - Un premier écran avec un champ de texte pour saisir le prénom et un bouton pour l'envoyer vers le serveur
  - Suite à la réponse du serveur, un deuxième écran qui affiche cette réponse

- Concepts à voir
  - Utilisation des capacités réseau du terminal
  - Permissions dans l'AndroidManifest.xml
  - Envoi et réception de requêtes/réponses REST

REST, REST... c'est quoi déjà REST ?

REST (REpresentational State Transfer)

- Une manière de construire une application pour le web
  - Pas un protocole, pas un format
  - Le style architectural original du Web.

- Quelques principes simples :
  - Des URI qui permettent de nommer et identifier les ressources
  - Le protocole HTTP qui fournit les méthodes pour accéder, modifier, créer ou effacer des contenus dans les ressources
    - Méthodes GET, POST, PUT et DELETE, essentiellement
  - Pas de session, pas d'état, chaque opération est auto-suffisante
  - Des formats de transfert standard
    - HTML, XML et JSON principalement.

Et JSON ?

JSON (JavaScript Object Notation)

- Format de données textuel et générique
  - dérivé de la notation des objets du langage JavaScript
- Plus simple, léger et facile à traiter que le XML
- Évaluation native en JavaScript
  - Bibliothèques très optimisés dans les autres langages

```
{
   "menu": {
       "id": "file",
       "value": "File",
       "popup": {
           "menuitem": [
               { "value": "New", "onclick": "CreateNewDoc()" },
               { "value": "Open", "onclick": "OpenDoc()" },
               { "value": "Close", "onclick": "CloseDoc()" }
           ]
       }
   }
}
```

LiveCoding : Hello RESTFful World

- URI à appeler

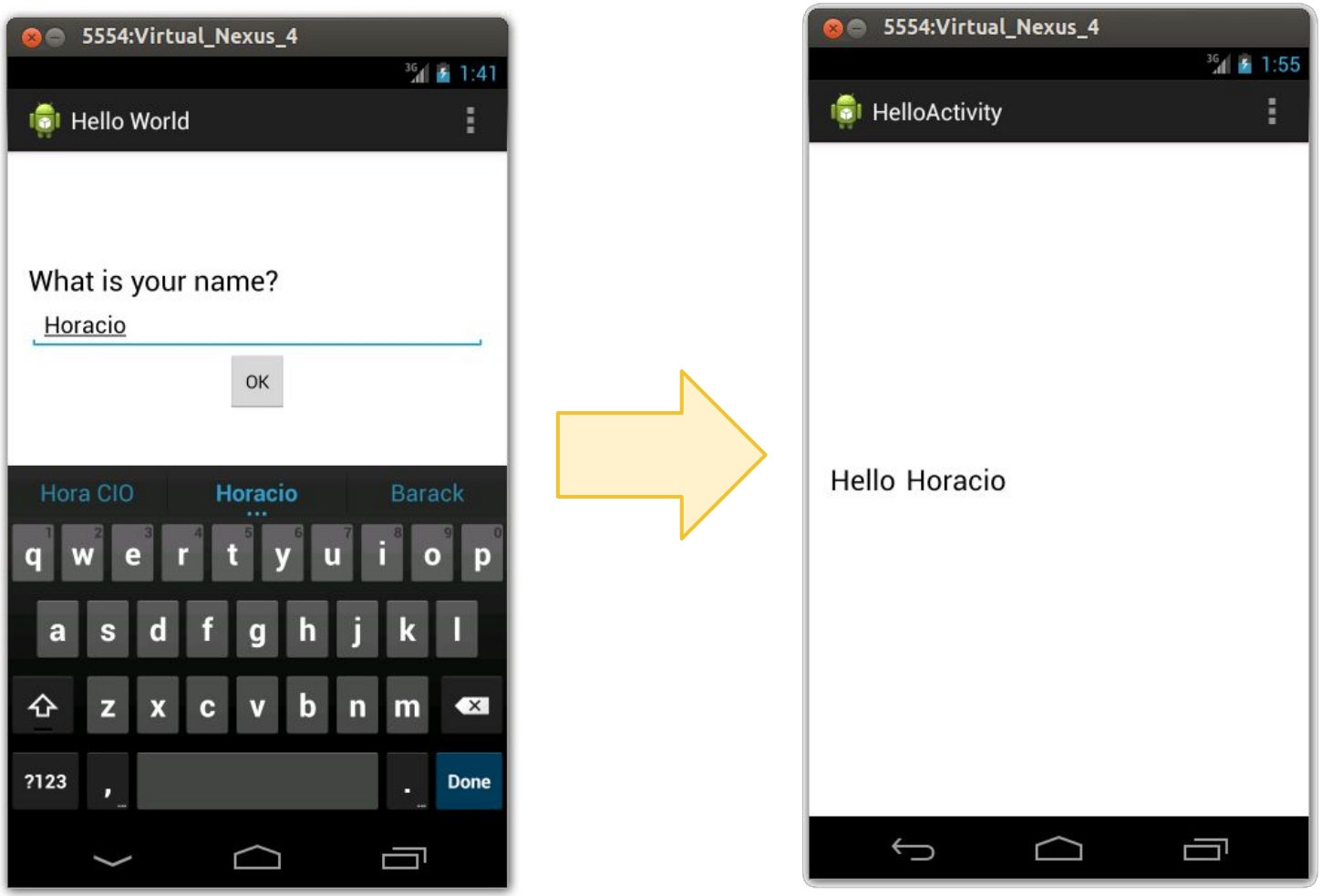
  http://lostinbrittany-chat-server.cleverapps.io/hello/PRENOM

  - avec à la place de PRENOM le prénom saisi dans le champ de texte

- Méthode HTTP à utiliser : GET
  - On cherche à obtenir une réponse du serveur (GET), pas à modifier du contenu (POST), le créer (PUT) ou effacer (DELETE)

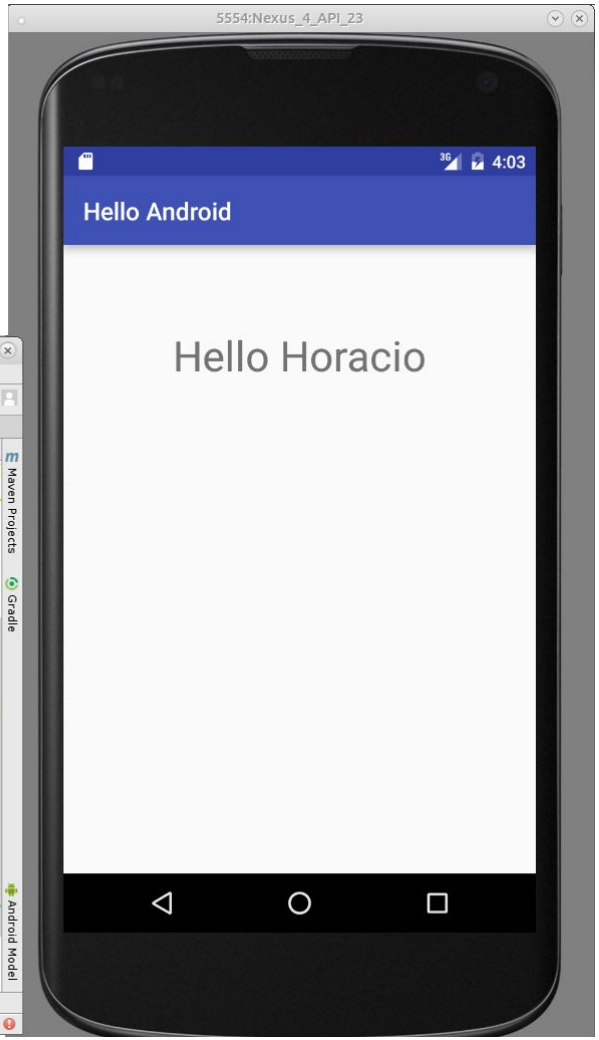
- On peut partir d'une copie du projet précédent

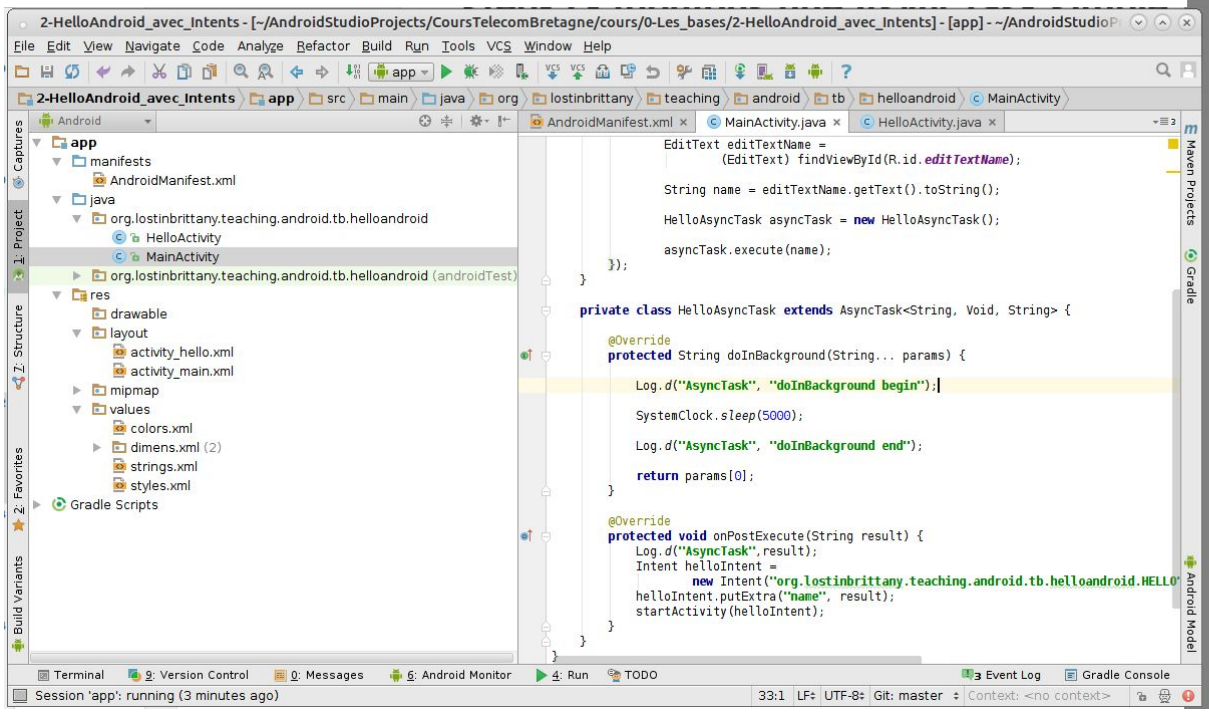
On part du dernier step du précédant



step-5 - Ajoutons une *AsyncTask* simple

- `onClick()` crée une *AsyncTask*
  - qui ne va faire qu'attendre
  - et lancer *l'Intent* à la fin





step-6 - Et les permissions qui vont bien...

- Dans le Manifeste

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.lostinbrittany.teaching.android.tb.helloandroid" >

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" >

        <activity android:name="org.lostinbrittany.teaching.android.tb.helloandroid.MainActivity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".HelloActivity" >
            <intent-filter>
            <action android:name="org.lostinbrittany.teaching.android.tb.helloandroid.HELLO" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
        </activity>
    </application>

</manifest>
```

step-7 - Outillage

Ajoutons un NetworkHelper

```
public class NetworkHelper {

   public static boolean isInternetAvailable(Context context) {
       try {
           ConnectivityManager cm = (ConnectivityManager)
                context.getSystemService(Context.CONNECTIVITY_SERVICE);
           NetworkInfo activeNetwork = cm.getActiveNetworkInfo();
           return null != activeNetwork &&
                activeNetwork.isConnectedOrConnecting();
       } catch (Exception ex) {
           Log.d("NetworkHelper", "Error in checking internet");
           return false;
       }
   }
}
```

step-7 - Outillage

Ajoutons un InputStreamReader pour lire les réponses réseau

```
// Reads an InputStream and converts it to a String.
private static String readIt(InputStream stream)
      throws IOException, UnsupportedEncodingException {
   int ch;
   StringBuffer sb = new StringBuffer();

   Reader reader = null;
   reader = new InputStreamReader(stream, "UTF-8");

   while ((ch = reader.read()) != -1) {
       sb.append((char) ch);
   }
   Log.d("Read Response", "Begin");
   if (sb.length() > 0) {
       return sb.toString();
   }
   return null;
}
```

step-7 - Outillage

```
public static String callHello(String name) {
    try {
        URL url = new URL(BASE_URL+HELLO_SERVICE+"/"+name);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setReadTimeout(10000); conn.setConnectTimeout(15000);
        conn.setRequestMethod("GET");
        conn.setDoInput(true); conn.setDoOutput(true);
        conn.connect();

        int responseCode = conn.getResponseCode();

        if (responseCode >= 400) {
            return readIt(conn.getErrorStream());
        }
        return readIt(conn.getInputStream());

    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return null;
}
```

step-7 - Outillage

Vérifions d'abord le réseau, appelons ensuite.

```
protected String doInBackground(String... strings) {
   Log.d("HelloAsyncTask", "doInBackground begin");
   // SystemClock.sleep(5000);

   boolean networkAvailable =
NetworkHelper.isInternetAvailable(getApplicationContext());
   Log.d("Available network?", Boolean.toString(networkAvailable));
   if (!networkAvailable) { return null; }

   String result = NetworkHelper.callHello(strings[0]);
   Log.d("HelloAsyncTask", "doInBackground end");
   return result;
}
```

Méthode HTTPConnection, bas niveau

[La suite]

Bilan

- Vous avez vu aujourd'hui :
  - Un aperçu de la plate-forme Android
  - Les concepts basiques d'une application Android
    - Activités, Services, Intents, AndroidManifest.xml
    - Layouts, widgets, strings.xml
    - Communications réseau, appels REST
    - AsyncTasks, TimerTask

[MERCI]