Travaux pratiques

Formation Android

Pierre-Yves Ricau (Excilys)

Preface

Ces TP ont été générés avec Wikbook. T'écris du XWiki dans des documents pur texte, puis tu lances un petit :

```
mvn package
```

Et hop t'as un beau PDF !

Chaque TP est prévu pour durer 1h, en travaillant en pair programming. Les positions (clavier / cerveau) doivent s'inverser toutes les 30 minutes.

A la fin de chaque TP, des questions Bonus sont disponible pour ceux qui ont le temps d'aller plus loin.

Les corrigés vous seront fournis, y compris pour les questions bonus.

Pour vous faire gagner du temps, la plateforme est déjà installée. Pour en savoir plus, RTFM : http://developer.android.com/sdk/index.html

Chapter 1. Fondamentaux

1.1. Premiers pas (15 min)

- Lancez Eclipse (non, oubliez Netbeans, c'est mort)

1.1.1. La simulation, ya que ça de bon

1.1.1.1. Un Emulateur aux petits oignons

- Démarrez le gestionnaire Android depuis Eclipse : **Window > Android SDK And AVD Manager**.
- Pour créer un nouvel émulateur : **Virtual devices > New**, puis jouez avec les différents paramètres pour créer l'émulateur de vos rêves.
- Pour la suite des TP, il nous faut un Emulateur dont la *Target* est : **Google APIs (Google Inc.) - API Level 8**.

> **Note**
>
> Google APIs (Google Inc.) - API Level 8 correspond à Android 2.2 (Froyo), avec les extensions Google (Google Maps, Gmail, etc)

- Démarrez l'émulateur, et allez sur Google depuis le navigateur de l'émulateur.

> **Note**
>
> Oui, le premier démarrage de l'émulateur est long, c'est noooormal.

1.1.1.2. Prenez le contrôle

- Dans **Eclipse**, ouvrez la vue **Emulateur Control**, et envoyez un SMS à l'émulateur.

1.1.2. Snoop doggy log

- Dans **Eclipse**, ouvrez la vue **Logcat**
- Dans votre projet, ouvrez la seule classe Java se trouvant dans src (hint: elle étend Activity, et implémente onCreate())
- Dans la méthode **onCreate()**, ajoutez une ligne de log, en vous basant sur l'exemple suivant :

```
[...]
import android.util.Log;

public class HelloAndroidWorld extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.main);
        Log.i("HelloAndroidWorld", "You don't understand, I want a sheet on the bed");
    }
}
```

- Lancez l'application et vérifiez que le log est présent.
- N'oubliez pas que vous pouvez éliminer le bruit en filtrant les logs!

1.1.3. Hello, Android World!

- Dans **Eclipse**, créez votre première application : **File** > **New** > **Project** > **Android** > **Android Project** > **Next** > **Remplir les champs** > **Finish**

> **Note**
>
> Le champ **Min SDK Version** doit correspondre à l'**API Level** de la **Build Target** (mékeskidit ? posez la question si c'est pas clair)

- Démarrez l'application : **Run As** > **Android Application**

1.2. Cycle de vie (30 min)

1.2.1. Commençons par tout péter

- Dans la méthode onCreate() de votre activity, jetez une RuntimeException.
- Lancez l'application. Kaboum ? Ne cliquez pas sur Force Close immédiatement
- Visualisez la stacktrace dans les logs
- Cliquez sur Force Close, et retournez aux logs. Si tout va bien, vous devriez voir quelque chose comme :

```
I/Process (  531): Sending signal. PID: 531 SIG: 9
I/ActivityManager(   59): Process com.excilys.formation.HelloAndroidWorld (pid 531) has died.
```

- Que signifie donc ce mystérieux chiffre 9 ?

1.2.2. N'appelle pas papa

- Dans la méthode onCreate(), supprimez le code qui lance la RuntimeException, et supprimez l'appel à **super.onCreate()**.
- Lancez l'application. Kaboum ?
- Visualisez la stack dans les logs. Explicite, n'est-ce pas ?

1.2.3. Les callbacks

1.2.3.1. Tracking de l'activité

- Dans l'activité, overridez les méthodes suivantes : **onCreate(), onStart(), onRestart(), onResume(), onPause(), onStop(), onDestroy(), onSaveInstanceState(), onRestoreInstanceState()**.
- Placez un log dans chaque méthode, portant le nom de la méthode en question, afin de suivre l'ordre d'appel des différentes callbacks. Ex :

```
@Override
protected void onResume() {
    super.onResume();
    Log.d("HelloAndroidWorld", "onResume");
}
```

1.2.3.2. Tracking de l'application

- Créez une classe étendant de android.app.Application
- Overridez les méthodes suivantes : **onCreate(), onTerminate(), onLowMemory()**.
- Placez un log dans chaque méthode, portant le nom de la méthode en question, afin de suivre l'ordre d'appel des différentes callbacks. Ex :

```
package com.excilys.formation.HelloAndroidWorld;

import android.app.Application;
import android.util.Log;

public class HelloApplication extends Application{
        @Override
        public void onCreate() {
                super.onCreate();
                Log.d("HelloApplication", "onCreate");
        }

        @Override
        public void onTerminate() {
                super.onTerminate();
                Log.d("HelloApplication", "onTerminate");
        }

        @Override
        public void onLowMemory() {
                super.onLowMemory();
                Log.d("HelloApplication", "onLowMemory");
        }
}
```

- Dans le fichier **AndroidManifest.xml**, spécifiez l'attribut **android:name** du noeud **application**, en lui donnant comme valeur le nom complet de la classe d'application. Ex :

```
<application
    android:name="com.excilys.formation.HelloAndroidWorld.HelloApplication"
    android:icon="@drawable/icon"
    android:label="@string/app_name">
```

1.2.3.3. Jouons avec la vie de l'application

- Ouvrez LogCat, puis lancez l'application et testez l'impact des cas d'utilisation suivants sur les appels de

callbacks :

- Appuie sur le bouton **Retour arrière**
- Appuie sur le bouton **Home** puis relancement de l'application via un appui long sur **Home**
- Changement d'orientation de l'écran (Ctrl+F11 pour l'émulateur)
- Interruption de l'activité en cours par un appel (à simuler via la vue **Emulateur Control**), puis réaffichage quand l'appel est terminé.

- Ajoutez un Thread.sleep de 5 secondes dans onPause(), et refaites vos tests.

1.3. S'inspirer de l'existant (15 min)

1.3.1. Retour aux sources

- Le package fournit contient les sources d'Android pour les versions 1.5, 1.6, 2.1 et 2.2
- Dans Eclipse, allez donc regarder les sources de la méthode onCreate() de la classe Activity (hint : F3 sur **super.onCreate()** dans votre activité).
- Saurez-vous trouver par quel technique fantastique Android parvient à savoir si vous avez appelé ou non **super.onCreate()** ?
- Pour disposer des sources Android lors de vos développements, il vous faudra lire ceci : http://android.opensourceror.org/2010/01/18/android-source/

1.3.2. Des samples pas si simples

- Créez un projet Eclipse à partir d'un sample : **File > New > Project > Android > Android Project > Next > Sélectionnez la Build Target 1.6> Create Project From Existing sample > ApiDemos**
- Attendez un peu, faut que ça importe et que ça build ;-) .
- Lancez l'application, and have fun !

1.4. Bonus

1.4.1. Les logs du geek

- Le programme **adb** se trouve dans **[...]/android-sdk-linux_x86/platform-tools**.
- Affichez les logs en utilisant adb directement plutôt que la vue **LogCat** d'Eclipse (hint : **adb logcat**)
- Essayez de filtrer les logs, de façon à n'afficher que les logs de niveau INFO (hint : **adb logcat -h**)
- Quels sont les avantages et inconvénients de cette commande par rapport à la vue Eclipse ?

1.4.2. On va gratter DDMS

- Le programme **ddms** se trouve dans **[...]/android-sdk-linux_x86/tools**.
- Lancez ddms.
- Prenez un screenshot de l'émulateur via DDMS (Device > Screen Capture)

1.4.3. N'appelle pas papa (bis repetita)

- Supprimez tout appel à **super.onCreate()** depuis la méthode onCreate() d'une activité.
- Essayez de feinter Android pour lui faire croire que l'appel à **super.onCreate()** à bien eu lieu (hint : mCalled + Reflection API).

1.4.4. APK, installation et désinstallation

- Lancez votre application sur l'émulateur à partir d'Eclipse (Run As)
- Créez un APK à partir de votre application : **Clic droit sur projet > Android Tools > Export Signed Application Package**

  > **Note**
  >
  > Le wizard vous permet de créer le keystore très simplement. Pensez à toujours spécifier une validité d'au moins 30 ans.

- Essayez d'installer l'application sur l'émulateur : **adb install nomDuFichier.apk**
- Normalement, l'installation échoue. Analysez pourquoi, et suivez les instructions du message d'erreur, puis installez pour de bon l'application.

1.4.5. Galaxy Tab

- Créez un émulateur de type GALAXY Tab Addon, et vérifiez que vos applications fonctionnent sur celui-ci.

Chapter 2. UI

The chapter 1 talks about the Bar class

2.1. The Bar class

Example 2.1. The full Bar class with its javadoc

Example 2.2. The Bar juu method

Example 2.3. The Bar juu method excerpt

> **Note**
>
> The source code shown in those examples is part of the maven project and is included in the documentation when the project is build.

```
public void foo()
{
   System.out.println("This is going to the output"); ❶
}
```

❶ A callout

Chapter 3. UI Avancée

The chapter 1 talks about the Bar class

3.1. The Bar class

Example 3.1. The full Bar class with its javadoc

Example 3.2. The Bar juu method

Example 3.3. The Bar juu method excerpt

> **Note**
>
> The source code shown in those examples is part of the maven project and is included in the documentation when the project is build.

```
public void foo()
{
   System.out.println("This is going to the output"); ❶
}
```

❶ A callout

Chapter 4. Au delà de la UI

The chapter 1 talks about the Bar class

4.1. The Bar class

Example 4.1. The full Bar class with its javadoc

Example 4.2. The Bar juu method

Example 4.3. The Bar juu method excerpt

Note

The source code shown in those examples is part of the maven project and is included in the documentation when the project is build.

```
public void foo()
{
   System.out.println("This is going to the output"); ❶
}
```

❶ A callout

Chapter 5. Mise en application

Ici une liste des TP proposés

5.1. App 1

5.2. App 2