Travaux pratiques

Formation Android

Pierre-Yves Ricau (Excilys)

Preface

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

```
mvn package
```

Et hop t'as un beau PDF !

Pour vous faire gagner du temps, le SDK est déjà installé. Pour en savoir plus, RTFM : http://developer.android.com/sdk/index.html

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

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

Les informations nécessaires vous sont fournies dans les TP, sans pour autant vous mâcher le travail. N'hésitez surtout pas à poser des questions !

Chapter 1. La base de la base

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"); ❶
    }
}
```

❶ Log au niveau info

- 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"); ❶
}
```

❶ Log de passage dans la méthode

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. Dessinez c'est gagné

2.1. Composants graphiques (30 min)

2.1.1. L'approche programmatique

- Petit rappel : au fur et à mesure de l'avancement, pensez à lancer l'application sur l'émulateur pour voir le résultat.
- Créez un nouveau projet Android
- Dans l'activité créée par défaut, modifiez la méthode **onCreate()**, créez une instance de LinearLayout et configurez la pour avoir une orientation verticale et remplir son parent horizontalement et verticalement.

```
LayoutParams linearLayoutParams = new LayoutParams(LayoutParams.FILL_PARENT, LayoutParams.FILL_PARENT);
layout.setLayoutParams(linearLayoutParams);
layout.setOrientation(LinearLayout.VERTICAL);
```

- Passez l'instance de LinearLayout à la méthode **setContentView()** :

```
setContentView(layout); ❶
```

❶ Pensez à supprimer setContentView(R.layout.main); !

- Créez une instance de TextView, qui remplisse son parent en largeur et soit à la taille du contenu en hauteur.

```
LayoutParams textViewParams = new LayoutParams(LayoutParams.FILL_PARENT, LayoutParams.WRAP_CONTENT);
```

- Définissez le texte de la TextView, puis ajoutez là au layout

```
textView.setText("I've got a big monkey");
layout.addView(textView);
```

- Sur le même principe, ajoutez un Button au layout.
- Un bouton, c'est fait pour être cliqué. Quand on clique sur le bouton, le contenu de la textview doit changer.

```
button.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
                ❶
        }
});
```

❶ Vous savez déjà comment (re)définir le texte de la textView, n'est-ce pas ?

- Ajoutez un nouveau bouton. Quand on clique sur celui-ci, le titre de la fenêtre change.

```
setTitle("I'm a monkey lover");
```

2.1.2. L'approche déclarative

Plus de 30 lignes de Java pour un petit texte et deux boutons, trouvez-vous cette approche pratique ?

- Créez une nouvelle activité
- Enregistrez-la dans le fichier **AndroidManifest.xml**. Pensez à déplacer l'IntentFilter de lancement de l'ancienne activité à la nouvelle.

```
<activity
        android:name="LeNomDeLaClasseDeMonActivite"
        android:label="Le titre">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

- Créez une nouvelle resource de type layout. Pour cela, dupliquez le fichier **main.xml** dans **res/layout**.
- Implémentez la méthode **onCreate()** dans cette nouvelle activité, et appelez **setContentView()** en lui donnant l'id la nouvelle ressource layout.

```
setContentView(R.layout.declarative); ❶
```

❶ si votre fichier s'appelle toto.xml, l'id vaut R.layout.toto

- Le fichier XML de layout contient un LinearLayout qui contient une textview.

> **Note**
>
> Avez vous remarqué l'attribut **android:text="@string/hello"** ? Il s'agit d'une référence vers la chaîne de caractère dont l'identifiant est **hello**. Celle-ci est définit dans le fichier **res/values/strings.xml**.

- Modifiez le fichier de layout XML pour qu'il soit identique à l'exercice précédent (en gros, ajoutez deux Button), en prenant soin d'externaliser les chaînes de caractères de la même manière que pour la TextView.
- Ajoutez des identifiants à vos composants dans le fichier XMl, grâce à l'attribut id. **@+id** permet de créer un nouvel identifiant.

```
<Button
    android:id="@+id/sexyButton"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Click me, I'm sexy!"
    />
```

- Une fois **setContentView()** appelé, vous pouvez récupérer les composants créés grâce à la méthode **findViewById()**.

```
setContentView(R.layout.declarative);
// [...]
Button sexyButton = (Button) findViewById(R.id.sexyButton);
```

- Reproduisez le comportement des boutons de l'exercice précédent.

2.2. Navigation inter activités (30 min)

- Créez une nouvelle activité (activité 1) avec un champ texte (EditText) et un bouton (Button), et modifiez le fichier AndroidManifest.xml afin que l'application se lance sur cette activité.
- Créez une seconde activité comportant une TextView.
- Quand on clique sur le bouton, le texte de l'EditText doit être envoyé à la seconde activité qui l'affiche dans la TextView.

Pour démarrer une activité :

```
Intent intent = new Intent(context, ActiviteADemarrer.class);
intent.putExtra("paramKey", paramValue); ❶
startActivity(intent);
```

❶ Ajout d'un paramètre (on ajoute des Extras à l'Intent utilisé pour démarrer l'activité)

Et pour récupérer les extras envoyés à une activité :

```
Intent intent = getIntent();
String param = intent.getStringExtra("paramKey");
```

- Faites en sorte que l'EditText de l'activité 1 ne puisse contenir que des chiffres

```
android:inputType="number"
```

- Ajoutez un bouton à l'activité 2. Quand on clique dessus, la fenêtre de composition d'appel téléphonique s'ouvre, avec comme numéro précomposé le paramètre envoyé par l'activité 1.

> **Note**
>
> La fenêtre de composition d'appel téléphonique est une activité, située dans une autre application que la votre.

```
Uri intentUrl = Uri.parse("tel:0642693313");
Intent intent = new Intent(Intent.ACTION_DIAL, intentUrl);
```

2.3. Bonus

2.3.1. Linkify

- Ajoutez une TextView affichant un texte comportant plusieurs liens HTTP. Ça serait cool qu'ils soient cliquables, non ? Linkify est là pour ça !

```
Linkify.addLinks(textView, Linkify.ALL); ❶
```

❶ RTFAB (Read The Fine Android Blog) : http://android-developers.blogspot.com/2008/03/linkify-your-text.html

2.3.2. Un layout tout relatif

- Comment feriez vous pour créer un layout composé de plusieurs champs textes les uns en dessous des autres, avec à droite de chaque champ texte un bouton ?

> **Note**
>
> Mac Gyver a la solution : imbriquer des LinearLayout horizontaux dans un LinearLayout verticaux. Oui, mais vous n'êtes pas Mac Gyver ;-)

- La solution ? Le RelativeLayout, qui permet à chaque composant de préciser comment il se positionne par rapport à d'autres composants.
- Essayez de mettre en oeuvre cette solution, en vous basant sur ce tutoriel : http://developer.android.com/resources/tutorials/views/hello-relativelayout.html

2.3.3. Ayez de la ressource !

- Créez un layout alternatif pour le mode paysage (Ctrl+F11 pour basculer l'émulateur). Pour cela, il suffit de créer un dossier *res/layout-land* et d'y placer une autre version de votre fichier de layout.
- Créez une version anglaise et une version française de votre application, en dupliquant strings.xml dans un dossier values-fr et un dossier values-en. Ensuite, changez la langue de l'émulateur dans la configuration (Home > Menu > Settings) et vérifiez que votre application a bien changé de langue.

2.3.4. Des Intents sous amphets

- Ajoutez un bouton qui déclenche l'ouverture d'une activité de composition d'un sms, avec le message pré rempli.

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.putExtra("sms_body", "Je suis ton père");
intent.setData(Uri.parse("sms:"));
```

- Faites de même avec un mail :

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.setType("plain/text");
intent.putExtra(Intent.EXTRA_EMAIL, new String[]{"you@duckmywife.com"});
intent.putExtra(Intent.EXTRA_SUBJECT, "Sujet");
intent.putExtra(Intent.EXTRA_TEXT, "Le corps du mail");
startActivity(Intent.createChooser(intent, "Quelle appli ??")); ❶
```

❶ Android demande à l'utilisateur de choisir l'application à utiliser pour envoyer le mail

2.3.5. Retour vers le passé

- Reprenez l'exercice sur la navigation inter activités.
- Ajoutez un champ texte à la l'activité 2.
- Faites en sorte que chaque fois qu'à chaque caractère modifié dans le champ texte, la TextView de l'activité 2 soit mis à jour avec le nouveau contenu.

```
editText.addTextChangedListener(new TextWatcher() {
        public void onTextChanged(CharSequence s, int start, int before, int count) {}
        public void beforeTextChanged(CharSequence s, int start, int count, int after) {}
        public void afterTextChanged(Editable s) {
                String content = s.toString();
                ❶
        }
});
```

❶ C'est là que ça se passe !

- Ajoutez un bouton à l'activité 2. Quand on clique dessus, l'activité se termine et renvoie le contenu du champ texte numérique à l'activité 1.

```
startActivityForResult(intent, 42); ❶
// [...]
Intent data = new Intent();
data.putExtra("myResult", someValue); ❷
setResult(RESULT_OK, data);
finish(); ❸
// [...]
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) { ❹
        if (requestCode==42) {
                if (resultCode==RESULT_OK) {
                        ❺
                }
        }
}
```

❶ Remplace startActivity(), à utiliser quand on attend un résultat de la part de l'activité appelée.
❷ On utilise un Intent comme conteneur pour passer des données en retour
❸ Termine l'activité courante
❹ A implémenter dans l'activité appelante.
❺ Faire quelque chose avec data

- Quand elle reçoit le résultat, l'activité 1 demande l'ouverte d'une activité de composition d'un sms, avec comme message pré rempli le résultat reçu.

Chapter 3. Un Toast vaut mieux que deux Dialogs

3.1. Les Toast, sautés à point mais sans intéraction svp (10 min)

- Créez une activité contenant un champ texte et un bouton.
- Lorsque l'utilisateur clique sur le bouton, le contenu du champ texte est affiché dans un Toast.

```
import android.content.Context;
import android.widget.Toast;
// [...]
Context context = this; ❶
Toast toast = Toast.makeText(context, "Hello World !", Toast.LENGTH_LONG); ❷
toast.show();
```

❶ Une Activity est un Context, de même qu'une Application ou un Service.
❷ Les deux seules durées possibles sont Toast.LENGTH_SHORT et Toast.LENGTH_LONG.

Le code est souvent raccourci en une seule ligne :

```
Toast.makeText(this, "Hello World !", Toast.LENGTH_LONG).show();❶
```

❶ Une des erreurs les plus communes est d'oublier l'appel à la méthode show().

3.2. Un menu à la carte (15 min)

3.2.1. Plain old Java

- Ajoutez deux éléments au menu de votre activité. Il suffit d'implémenter **onCreateMenu()** (méthode appelée la première fois que l'utilisateur presse le bouton Menu).

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
        menu.add("Yeah");
        return true;
}
```

- Affichez un Toast lorsque qu'un élément du menu est sélectionné.

```
MenuItem yeah = menu.add("Yeah");
yeah.setOnMenuItemClickListener(new OnMenuItemClickListener() {
        @Override
        public boolean onMenuItemClick(MenuItem item) {
                ❶
                return true;
        }
});
```

❶ Implémentez ici l'affichage du Toast.

3.2.2. Via une ressource XML

- Créez un fichier XML dans le répertoire **res/menu**, sur ce modèle :

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/yeah"
          android:title="Yeah" />
</menu>
```

- Utiliser un MenuInflater pour créer les éléments de menu à partir de la ressource XML.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.nomDuFichierXml, menu); ❶
    return true;
}
```

❶ Si votre menu est décrit dans res/menu/toto.xml, la ressource sera R.menu.toto

- Modifiez le fichier XML de description du menu, afin d'ajouter plusieurs éléments au menu.
- Implémentez **onOptionsItemSelected()** pour afficher un Toast lorsqu'un élément du menu est sélectionné.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        case R.id.yeah: ❶
                ❷
                return true;
        default:
                return super.onOptionsItemSelected(item);
        }
}
```

❶ R.id.yeah fait référence à l'élément définit par @+id/yeah dans le fichier XML
❷ Implémentez ici l'affichage du Toast.

3.3. Boîtes de dialogue modales (20 min)

- Dans l'activité de votre choix, affichez un Toast lorsque l'utilisateur appuie sur le bouton *Recherche* (la loupe). Hint : implémentez **onSearchRequested()**.
- Remplacez le Toast par une boîte de dialogue avec un titre, un message, un bouton OK et un bouton Annuler. Pour demander l'affichage d'une dialog, on utilise "showDialog(id)" en donnant l'int de son choix, puis on implémente **onCreateDialog(int id, Bundle args)**.

```
import android.app.AlertDialog;
import android.app.Dialog;
// [...]
showDialog(42);
// [...]
@Override
protected Dialog onCreateDialog(int id, Bundle args) {
        switch (id) {
        case 42:
                AlertDialog.Builder builder = new AlertDialog.Builder(this);
```

```
                builder.setTitle("test");
                ❶
                return builder.create();
        default:
                return null;
        }
}
```

❶ Le builder propose de nombreux setters, à vous d'en faire bon usage

- Ajoutez un champ texte à la dialog (indice : **builder.setView(unEditText))**).
- Lorsque l'utilisateur clique sur **OK**, affichez un Toast avec le contenu du champ texte. Lorsqu'il clique sur **Annuler**... ne faites rien ;-).

3.4. Adapters (15 min)

- Créez une nouvelle activité, dont le layout comporte une ListView
- Dans la méthode **onCreate()**, liez un ArrayAdapter à la ListView.

```
List<String> values = Arrays.asList("Who", "Will Move", "The Fruit Juice");
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, values); ❶
listView.setAdapter(adapter);
```

❶ android.R.layout.simple_list_item_1 désigne un layout mis à disposition par Android.

- Lorsque l'utilisateur clique sur un élément de la liste, affichez l'élément sélectionné dans une boîte de dialogue.

```
listView.setOnItemClickListener(new OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
                String item = (String) parent.getItemAtPosition(position);
                ❶
        }
});
```

❶ Mais comment passer l'item sélectionné à une boîte de dialogue ? Hint : showDialog a deux paramètres ;-)

3.5. Bonus

3.5.1. Notifications

- Créez une activité contenant un bouton.
- Lorsque ce bouton est pressé, une notification est affichée. Lorsque la notification est cliquée, elle lance l'activité de votre choix. Voici comment faire :

```
NotificationManager notificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
```

❶ On demande au Context une référence vers le NotificationManager

```
int icon = R.drawable.icon; ❶
String statusText = "Hello"; ❷
long when = System.currentTimeMillis(); ❸
Notification notification = new Notification(icon, statusText, when);
```

❶ L'icône à utiliser pour la notification
❷ Le texte qui s'affiche dans la barre de statut
❸ L'heure de l'évènement, qui s'affichera à côté de la notification

```
Context context = getApplicationContext(); ❶
String contentTitle = "My notification"; ❷
String contentText = "Hello World!";
Intent notificationIntent = new Intent(this, MyActivity.class); ❸
PendingIntent contentIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0); ❹

notification.setLatestEventInfo(context, contentTitle, contentText, contentIntent);

notificationManager.notify(42, notification); ❺
```

❶ C'est important : n'utilisez pas le context d'une activité, qui ne sera plus valable une fois l'activité détruite.
❷ Le contentTitle et le contentText sont visibles lorsque l'utilisateur ouvre la barre de notification.
❸ MyActivity est l'activité qui sera lancée lorsque la notification sera cliquée.
❹ Un PendingIntent est un intent qui peut s'exécuter dans le futur.
❺ C'est parti mon kiki !

- Créez une notification qui soit affichée comme "En cours" dans la barre de statut. Pour en savoir plus sur les notifications, RTFM : http://developer.android.com/guide/topics/ui/notifiers/notifications.html

3.5.2. Un menu dynamique

- Reprenez votre activité avec le menu définit via un fichier XML
- Affichez un nombre aléatoire entre 1 et 42 dans le titre d'un des éléments du menu, changeant à chaque fois que le menu s'affiche.

```
@Override
public boolean onPrepareOptionsMenu(Menu menu) {
        MenuItem yeah = menu.findItem(R.id.yeah);
        ❶
        return super.onPrepareOptionsMenu(menu);
}
```

❶ Modifiez l'élément yeah.

- Tous les secrets du fonctionnement des menus : http://developer.android.com/guide/topics/ui/menus.html

Note

Comment génère t'on un nombre aléatoire en Java ? On lit la SCJP.

3.5.3. Des dialogs de sourd

- Créez une activité avec un fond transparent et sans titre qui affiche une boîte de dialogue, de manière à donner l'impression que la boîte de dialogue fait partie de l'activité précédente. Allez, je vous aide :

```
<activity
  android:name=".MyDialogActivity"
  android:theme="@android:style/Theme.Translucent.NoTitleBar"
/>
```

- Créez une boîte de dialog comportant une liste d'éléments à sélectionner via des checkboxs dans la dialog (hint : setMultiChoiceItems()) et affichez dans un Toast les éléments sélectionnés.

- Ajoutez l'heure courante au titre de la dialog. Le titre doit être à jour chaque fois qu'on ferme puis qu'on affiche la dialog (hint : onPrepareDialog() à implémenter)

3.5.4. Adapter

- Reprenez l'exercice sur les adapters, en remplaçant l'ArrayAdapter par un adapteur créé pour l'occasion, étendant BaseAdapter. Attention, il faut recycler les vues dans **getView()** !

Note

Si vous avez lu ce TP jusqu'au bout et que vous demandez ce que c'est que cette histoire de recyclage des vues... il suffit de lever la main ;-)

Chapter 4. Le web, les threads, et toussa toussa

4.1. Oueb et raies zoo (15 min)

- Créez un nouveau projet **Android 2.3** (API 9)
- Ajoutez un bouton à l'activité créée.
- Ajoutez la permission d'accès à internet au fichier **AndroidManifest.xml**

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

- Quand l'utilisateur clique sur le bouton, téléchargez une image disponible sur le web

```
String url = "https://ecoles.excilys.com/lms/secure/downloadAvatar.htm?username=jbriche";
HttpUriRequest request = new HttpGet(url);
HttpClient client = new DefaultHttpClient();
HttpResponse response = client.execute(request);
InputStream inputStream = response.getEntity().getContent();
```

- Créez une ImageView à partir de l'image

```
Context context = getApplicationContext();
ImageView imageView = new ImageView(context);
Bitmap bitmap = BitmapFactory.decodeStream(inputStream);
imageView.setImageBitmap(bitmap);
```

- Affichez l'ImageView dans un Toast

```
Toast toast = new Toast(context);
toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0);
toast.setDuration(Toast.LENGTH_LONG);
toast.setView(imageView);
toast.show();
```

- Est-ce que tout vous paraît fonctionner correctement ?
- Ajoutez au début de la méthode **onCreate()** le code suivant, qui permet de vérifier que vous faites bien votre boulot :

```
StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder() ❶
                .detectNetwork() ❷
                .build());
```

❶ StrictMode n'est disponible qu'à partir d'Android 2.3
❷ Les autres options disponibles sont décrites dans la doc : http://developer.android.com/reference/android/os/StrictMode.html

- Puis lancez l'application et surveillez les logs.
- Quel est le problème ? Comment le corriger ?

4.2. Threads (15 min)

- Reprenez le code précédent, mais en exécutant le code de téléchargement dans un nouveau thread.
- Que se passe t'il si vous essayez d'afficher le Toast depuis ce thread ?
- Utilisez la méthode **runOnUiThread()** pour afficher le Toast depuis le thread de UI.

```
runOnUiThread(new Runnable() {
        @Override
        public void run() {
                ❶
        }
});
```

❶ Ce code s'exécute dans le thread de UI.

4.3. Services inclus (15 min)

- Créez une classe étendant IntentService.
- Implémentez la méthode **onhandleIntent()** et faites la écrire un log chaque fois qu'elle est appelée.
- Enregistrez le service dans le fichier **AndroidManifest.xml**.

```
  <service android:name="MyService" />
</application>
```

- Ajoutez un bouton dans une activité. Le click sur le bouton doit démarrer le service.

```
Intent intent = new Intent(this, Myservice.class)
startService(intent);
```

- Vérifiez que le clic sur le bouton entraîne bien l'écriture d'un log.
- Faites en sorte que le service télécharge l'image de l'exercice précédent, et l'affiche dans un Toast. Attention : la méthode **onHandleIntent()** ne **s'exécute pas** dans le thread de UI. Pas de méthode runOnUiThread ? En regardant dans le code source de cette méthode, vous découvriez qu'elle utilise un Handler.

```
Handler handler = new Handler(); ❶
handler.post(new Runnable() {
        @Override
        public void run() {
        }
});
```

❶ Attention : le thread associé au Handler est celui dans lequel il a été instancié. Notez qu'un IntentService est instancié dans le thread de UI ;-)

4.4. Broadcast receivers (20 min)

- Créez une classe étendant BroadcastReceiver.
- Enregistrer le receiver dans le fichier **AndroidManifest.xml**, de façon à ce qu'il soit appelé chaque fois qu'une nouvelle application est installée.

```
<receiver android:name="MyInstallReceiver">
        <intent-filter>
                <action android:name="android.intent.action.PACKAGE_ADDED" />
                <data android:scheme="package"/>
        </intent-filter>
</receiver>
```

- Dans **onReceive()**, créez une notification indiquant le package de l'application installée, et lançant l'activité de votre choix. Indice : regardez ce que contient l'intent passé au broadcast receiver, par exemple en utilisant le débogueur.
- Dans **onReceive()**, ajoutez un Thread.sleep() de 15 secondes, puis affichez un Toast. Que se passe t'il ? Pourquoi ?

4.5. Bonus

4.5.1. ASyncTask

- Reprenez l'exercice sur les threads, mais plutôt que **runOnUiThread()**, utilisez une ASyncTask : http://developer.android.com/reference/android/os/AsyncTask.html

4.5.2. Secret Receiver

- Créez un BroadcastReceiver capable de recevoir un code secret composé dans le dialer. Il suffit ainsi que composer *#*#1337#*#* pour déclencher le broadcast **android_secret_code://1337**. <receiver android:name="MySecretReceiver"> <intent-filter> <action android:name="android.provider.Telephony.SECRET_CODE" /> <data android:scheme="android_secret_code"/> <data android:host="1337"/> </intent-filter> </receiver>**

Chapter 5. Mise en application

Réalisez l'application de votre choix ! Vous avez une heure :-)

Voici une liste de sujets "réalisables", vous pouvez aussi proposer les votres et on étudiera ensemble la faisabilité dans le temps imparti.

5.1. Babel fish

L'application se compose d'un champs texte et d'un bouton. Quand on clique sur le bouton, le contenu du champs texte est envoyé à Google Translate afin d'être traduit (dans la langue de votre choix). Une fois la traduction effectuée, celle-ci est lue à voix haute par le téléphone grâce au Text-To-Speech

- TTS : http://android-developers.blogspot.com/2009/09/introduction-to-text-to-speech-in.html
- Google Translate API : http://code.google.com/p/apps-for-android/source/browse/trunk/Translate/src/com/beust/android/translate/Translate.java

5.2. Maps to kill

L'application est un Tap Taupe like, mais avec une carte Google Maps en fond. Un "ennemi" apparaît à une position donnée sur la carte (pas forcément visible), et il faut cliquer dessus le plus vite possible.

- Maps : http://code.google.com/android/add-ons/google-apis/

5.3. Uninstall Notifications

Chaque fois qu'une nouvelle application est installée, une notification apparaît 5 minutes après l'installation. Quand il clique sur la notification, l'utilisateur est redirigé vers l'activité de désinstallation de l'application. L'icône de la notification est celle de l'application à désinstaller.

- Uninstall intent : http://android.amberfog.com/?p=98
- Alarm : http://developer.android.com/resources/samples/ApiDemos/src/com/example/android/apis/app/AlarmService.html

5.4. Devinez le nombre

Il s'agit d'un classique jeu de devinette d'un nombre par proposition successives. L'utilisateur propose un nombre, l'application indique si le nombre à trouver est inférieur ou supérieur. L'interface doit se faire uniquement par des dialogs, le fond est transparent (ie on doit voir la home), et des notifications : l'utilisateur rentre un nombre dans une dialog, celle-ci disparaît et 30 secondes plus tard une notification lui indique si le nombre est supérieur ou inférieur. Un clic sur la notification permet de proposer un nouveau nombre.