



Android

Développement d'applications mobiles

Formation avancée





Développement d'applications mobiles avec Android (avancé)

- Interfaces utilisateur avancées
 - Styles et thèmes
 - Menus d'options
 - Fragments
 - Activités de préférences
 - Toasts
 - Notifications
 - App Widgets
 - Internationalisation
- Multithreading
- Les Content Providers
- Réseau, connexion et services web
- Publier son application



Interfaces utilisateur avancées



Interfaces utilisateur avancées

Styles et thèmes

- Un style est comparable à une classe CSS : il définit une série de propriétés qui seront partagées par plusieurs contrôles.
- Un thème est un style appliqué à toute une application ou à une activité.
- Les styles, comme les thèmes, peuvent hériter d'autres styles/thèmes, ce qui évite d'avoir à reconfigurer un style existant.
- Avantage :
 - Externalisation
 - Réutilisabilité

Interfaces utilisateur avancées

Styles : Définition

- Un style est une ressource, et se définit dans un fichier .xml à part.
- L'attribut « name » du style doit être l'attribut à appliquer, et « value » la valeur.

```
<style name="CustomText">
    <item name="android:padding">16dp</item>
    <item name="android:textAppearance">@android:style/TextAppearance.Large</item>
    <item name="android:textColor">#000000</item>
</style>
```

Interfaces utilisateur avancées

Thèmes : Définition

- Même procédure que pour les styles :

```
<!-- Base application theme. -->
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->
    <item name="android:textColor">#FFDDDD</item>
    <item name="android:background">#550000</item>
</style>
```

- Par convention, on utilise NomDuParent.NomDuTheme.
- Un nouvel outil existe également dans Android Studio : le Theme Editor.

Interfaces utilisateur avancées

Styles et thèmes : Utilisation

- Les styles sont référencés dans les propriétés :

```
<TextView
    android:text="Hello world!"
    style="@style/CustomText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

- Les thèmes sont définis dans le manifest (application, ou activity) :

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="AppWidgetTest"
    android:theme="@style/AppTheme" >
```



Interfaces utilisateur avancées

Menu d'options

- Les systèmes Android possèdent souvent un bouton « hardware » de menu. Celui-ci est utile afin de proposer des fonctionnalités supplémentaires sans encombrer l'interface graphique.
- Alternativement, ce menu peut aussi s'ouvrir grâce à l'ActionBar.
- Un menu est propre à une activité.



Interfaces utilisateur avancées

Menu d'options : création

- Surcharge de la méthode `onCreateOptionsMenu()` de l'activité
 - Cette méthode est appelée lorsque l'utilisateur clique sur le bouton menu. Typiquement, on y spécifie le fichier de ressource xml qui servira à créer le menu.
- Surcharge de la méthode `onOptionsItemSelected()`
 - Cette méthode est appelée lorsque l'utilisateur sélectionne un des items du menu, et permet d'effectuer une action propre à cet item.



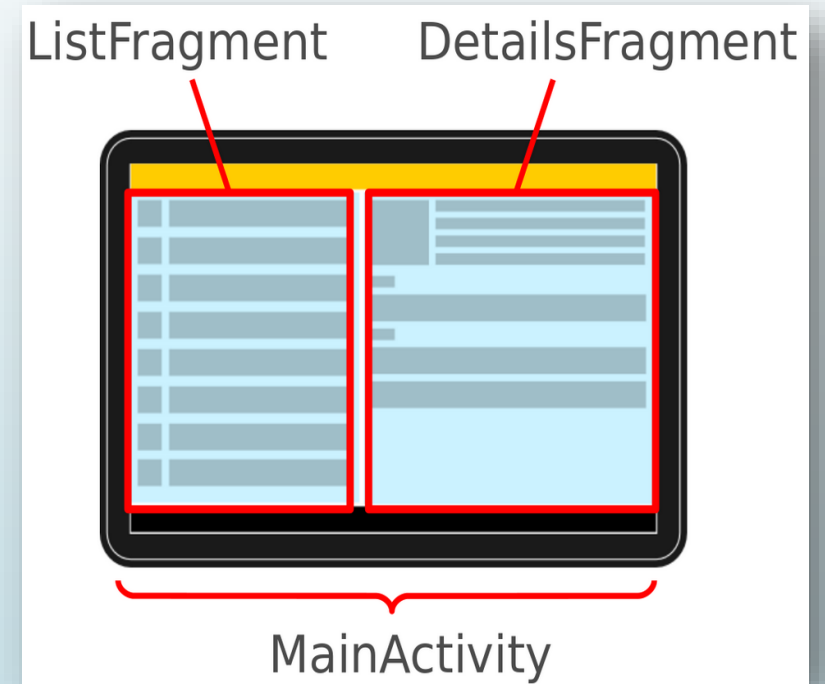
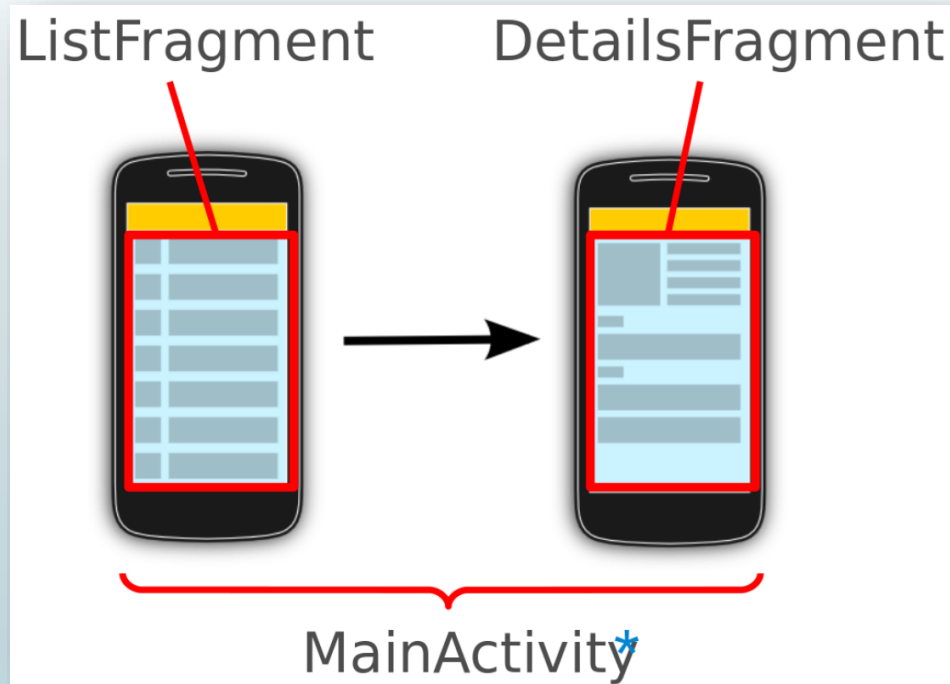
Interfaces utilisateur avancées

Les fragments : Introduction

- Un fragment est un composant réutilisable possédant sa propre interface et sa propre logique.
- Un fragment ne peut jamais vivre « seul », il s'exécute uniquement dans le contexte d'une activité.
- Son cycle de vie diffère un peu, mais est lié au cycle de vie de son activité.
- Buts :
 - Réutilisabilité
 - Possibilité d'agencements différents selon la taille de l'écran
 - Séparation de logiques

Interfaces utilisateur avancées

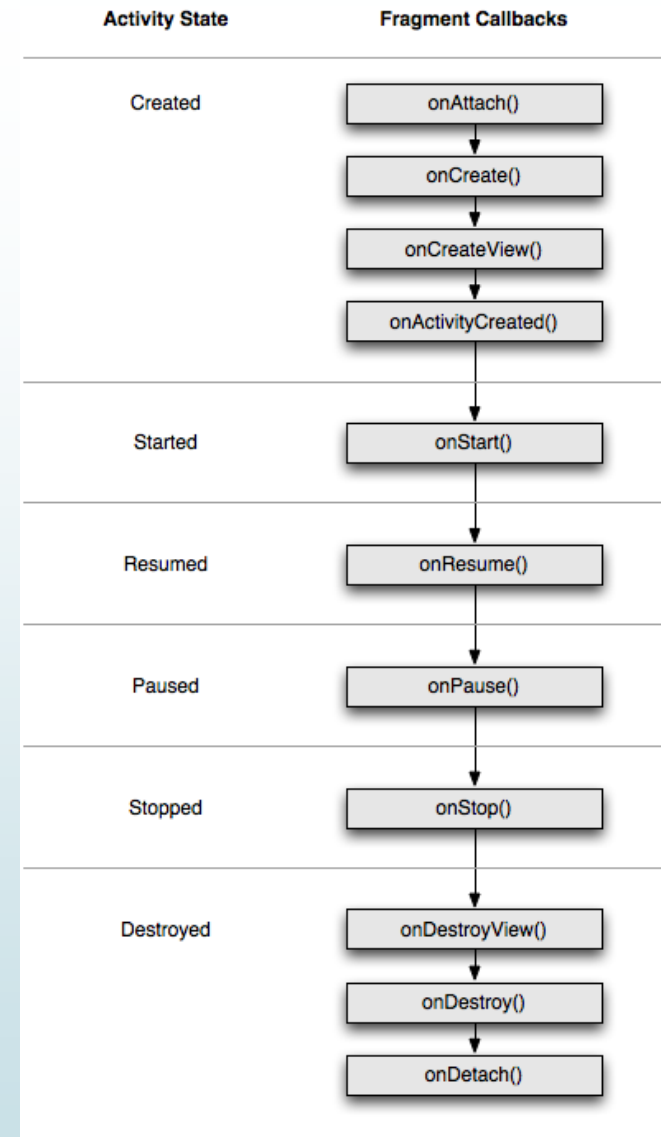
Les fragments : Exemple



Interfaces utilisateur avancées

Les fragments : Cycle de vie

- `onAttach()` est appelée lorsque le fragment est associé à l'activité.
- `onCreateView()` est appelée lorsque la vue associée au fragment est créée.
- `onActivityCreated()` est appelée lorsque la méthode `onCreate()` de l'activité a renvoyé une valeur de retour.
- `onDestroyView()` est appelée lorsque la vue associée au fragment est détruite.
- `onDetach()` est appelée lorsque le fragment est dissocié de l'activité.





Interfaces utilisateur avancées

Les fragments : Quelques classes

- Pour utiliser des fragments, il faut tout d'abord utiliser une activité qui hérite de `FragmentActivity`, ou d'une de ses sous-classes (`AppCompatActivity` par exemple).
- Les autres classes que nous utiliserons sont notamment :
 - `Android.app.Fragment`
 - `Android.app.FragmentManager`
 - `Android.app.FragmentTransaction`
- **Attention !** Il existe également des classes dites de « support », qui permettent une meilleure compatibilité avec les anciens appareils. Ces classes ne sont pas interchangeables, dans le sens où un `Fragment` de support ne fonctionnera pas avec un `FragmentManager` « normal ».

Interfaces utilisateur avancées

Les fragments : Création

- L'interface d'un fragment se déclare de manière similaire à celle d'une activité, c'est-à-dire dans un fichier XML.
- La logique d'un fragment se déclare dans un fichier java. La méthode qui doit être implémentée est la méthode `onCreateView()`, dans laquelle on déclare le fichier XML qui sera utilisé, ainsi que les éléments de l'interface grâce à la méthode `View.findViewById()`.

```
public HomeFragment() {}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                        Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_home, container, false);
    v.findViewById(R.id.menu_list);
    return v;
}
```

Interfaces utilisateur avancées

Les fragments : Création

- Souvent, on va également utiliser pour ces fragments le design pattern « singleton ». Cela permettra d'avoir une seule instance de chaque fragment, afin de conserver les données entre les changements de pages, et d'éviter de devoir instancier un nouveau fragment à chaque navigation.

```
public class HomeFragment extends Fragment {  
  
    // Singleton  
    private static HomeFragment instance;  
  
    public static HomeFragment getInstance() {  
        if (instance == null) {  
            instance = new HomeFragment();  
        }  
        return instance;  
    }  
}
```

Interfaces utilisateur avancées

Les fragments : Création

- Pour afficher ce fragment à l'écran, on a besoin également d'un *container* placé dans l'activité. Ce *container* peut être n'importe quel *layout*, auquel on donne un *id*.
- On utilise ensuite le `FragmentManager` pour ajouter le fragment choisi au *container* :

```
FragmentManager fm = getSupportFragmentManager();  
FragmentTransaction transaction = fm.beginTransaction();  
transaction.add(R.id.drawer_content_main, HomeFragment.getInstance());  
transaction.commit();
```




Interfaces utilisateur avancées

Les fragments : Création

- Pour ajouter un fragment à un container :
`transaction.add(container, fragment)`
- Pour remplacer le contenu d'un container par un fragment :
`transaction.replace(container, fragment)`
- Pour ajouter une transaction au « backstack » et permettre le retour en arrière :
`transaction.addToBackStack(identifiant)`
- Pour afficher / masquer un fragment :
`transaction.show()` / `transaction.hide()`
- **Attention !** Ne pas oublier le `transaction.commit()` à la fin !



Interfaces utilisateur avancées

Les fragments : Communication

- La communication ne s'effectue **jamais** directement entre deux fragments, elle passe toujours par l'activité.
- La façon la plus élégante de communiquer entre une activité et un fragment est via une interface de callback.
- L'activité donne en fait au fragment la possibilité de lui renvoyer des informations grâce à une interface spécifique au fragment, et définie dans celui-ci.

Interfaces utilisateur avancées

Les fragments : Communication

- Imaginons une application dans laquelle un texte entré par l'utilisateur dans un fragment doit remonter au niveau de l'activité :
 - Le fragment :

```
public class ActionFragment extends Fragment {  
  
    private ActionFragmentCallback callback;  
  
    public void setCallback(ActionFragmentCallback callback) {  
        this.callback = callback;  
    }  
  
    public interface ActionFragmentCallback {  
        void changeTextOnClick(String text);  
    }  
}
```

Interfaces utilisateur avancées

Les fragments : Communication

► L'activité :

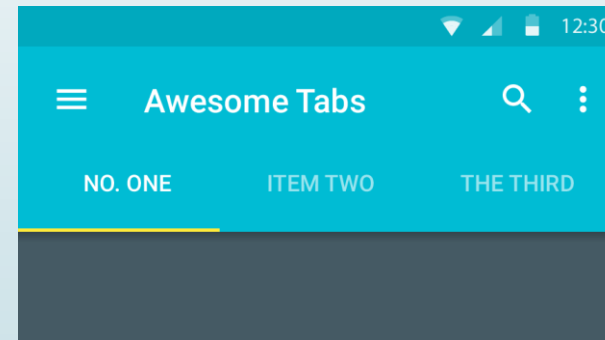
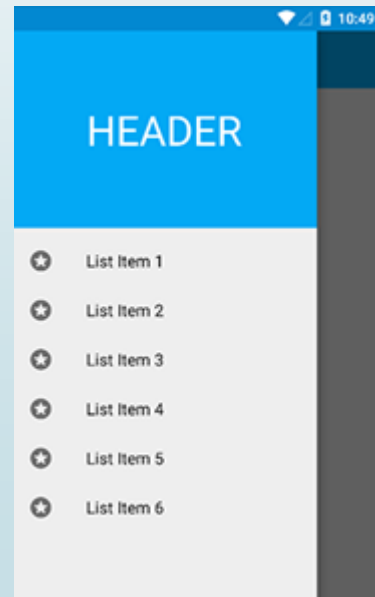
```
public class MainActivity extends AppCompatActivity implements ActionFragmentCallback {  
  
    TextView tv_main_text;  
  
    @Override  
    public void changeTextOnClick(String text) {  
        tv_main_text.setText(text);  
    }  
}
```

```
FragmentManager fm = getSupportFragmentManager();  
FragmentTransaction transaction = fm.beginTransaction();  
ActionFragment actionFragment = new ActionFragment();  
actionFragment.setCallback(this);  
transaction.add(R.id.fl_main_container, actionFragment);  
transaction.commit();
```

Interfaces utilisateur avancées

Les fragments : Utilisation

- Les fragments sont rarement utilisés sans faire appel à d'autres éléments tels que les menus de navigation ou les onglets. Ces éléments peuvent facilement être introduits dans une application grâce à la Support Design Library :





Interfaces utilisateur avancées

Les activités de préférences

- Une activité de préférence est une activité particulière en Android, qui permet de créer facilement des menus d'options.
- Le but est d'avoir des menus d'apparence similaires entre les différentes applications. Cela signifie que nous n'aurons pas à définir nous-mêmes l'interface graphique de cette activité.
- À la place, nous allons définir un fichier XML qui définira un PreferenceScreen.

Interfaces utilisateur avancées

Les activités de préférences

► XML

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
    <CheckBoxPreference
        android:key="newsletter"
        android:title="Newsletter"
        android:summary="S'abonner à la newsletter ?"
        android:summaryOn="Vous recevrez notre newsletter"
        android:summaryOff="Vous ne recevrez pas notre newsletter" />
    <EditTextPreference
        android:key="email"
        android:title="Email"
        android:summary="Votre adresse email"
        android:dialogTitle="Saisissez votre adresse email : " />
</PreferenceScreen>
```

Interfaces utilisateur avancées

Les activités de préférences

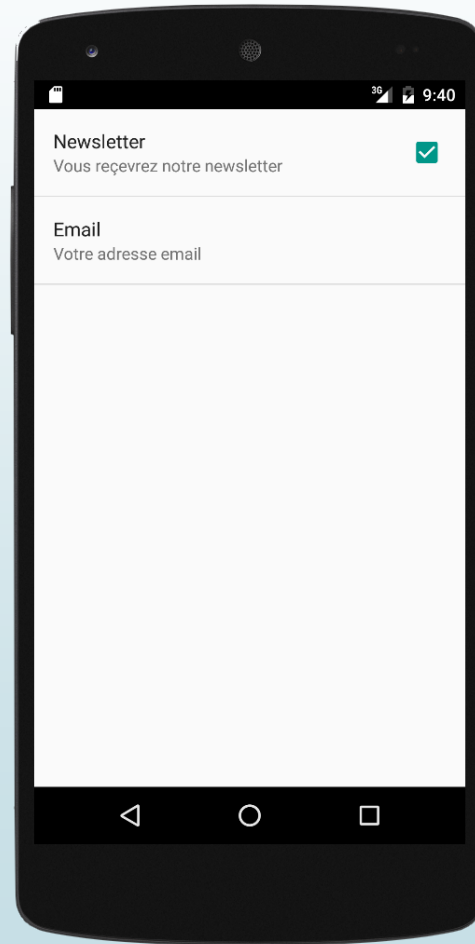
► Java

```
public class ConfigActivity extends PreferenceActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        addPreferencesFromResource(R.xml.preferences);  
    }  
}
```


Interfaces utilisateur avancées

Les activités de préférences

➡ Résultat :



Interfaces utilisateur avancées

Les toasts

- Les Toasts sont des petits messages destinés à l'utilisateur qui s'affichent, généralement en bas de l'écran.
- Leur utilisation est très simple :

```
Toast.makeText(getApplicationContext(), "Mon texte", Toast.LENGTH_SHORT).show();
```

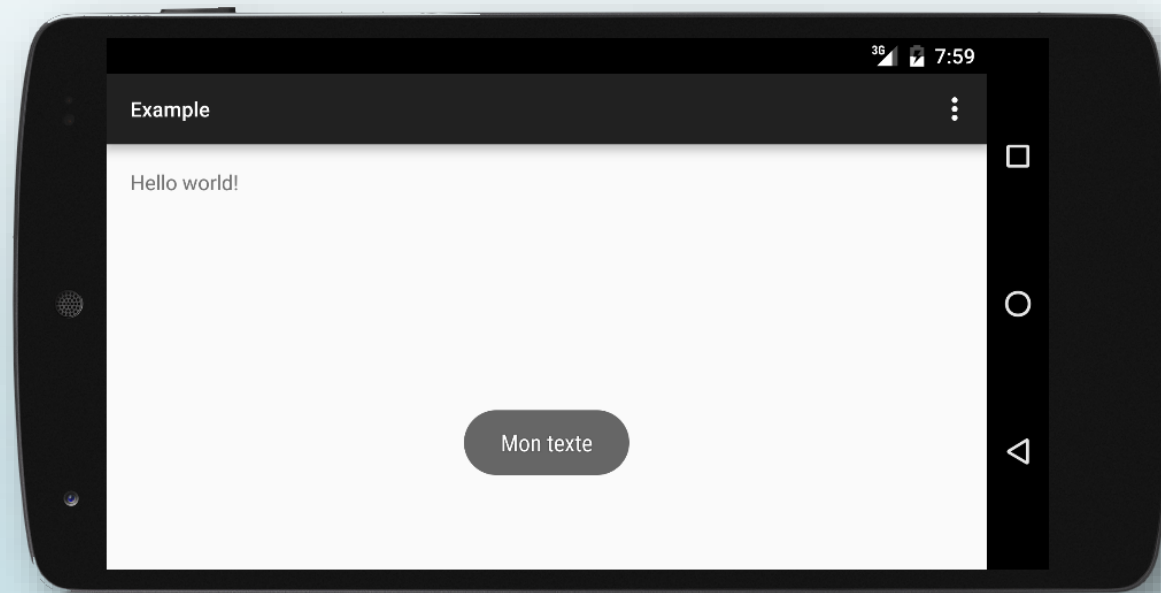
- Ou, alternativement :

```
Toast t = Toast.makeText(getApplicationContext(), "Mon texte", Toast.LENGTH_SHORT);  
t.show();
```

Interfaces utilisateur avancées

Les toasts

► Voici le résultat attendu :





Interfaces utilisateur avancées

Les notifications

- Les notifications sont des messages destinés à l'utilisateur, mais plus évolués que les Toasts :
 - Elles s'affichent dans la barre de statut tant que l'utilisateur n'a pas signifié leur lecture.
 - Elles peuvent émettre un son ou utiliser la vibration pour avertir l'utilisateur de leur réception.
 - Elles peuvent être configurées afin d'utiliser un *PendingIntent* (un *Intent* en attente d'être utilisé).

Interfaces utilisateur avancées

Les notifications : Utilisation

- Pour construire nos notifications, on utilise le *Notification.Builder* (ou *NotificationCompat.Builder*).
 - **Attention !** *setSmallIcon()*, *setContentTitle()* et *setContentText* sont obligatoires !

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(getApplicationContext());
builder.setSmallIcon(R.mipmap.ic_launcher);
builder.setContentTitle("My notification title");
builder.setContentText("My notification text");
builder.setSound(Settings.System.getUriFor(Settings.System.NOTIFICATION_SOUND));
builder.setVibrate(new long[]{200, 200, 200, 200, 200});
Notification notification = builder.build();

NotificationManager manager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
manager.notify(ID, notification);
```

Interfaces utilisateur avancées

Les notifications : Utilisation

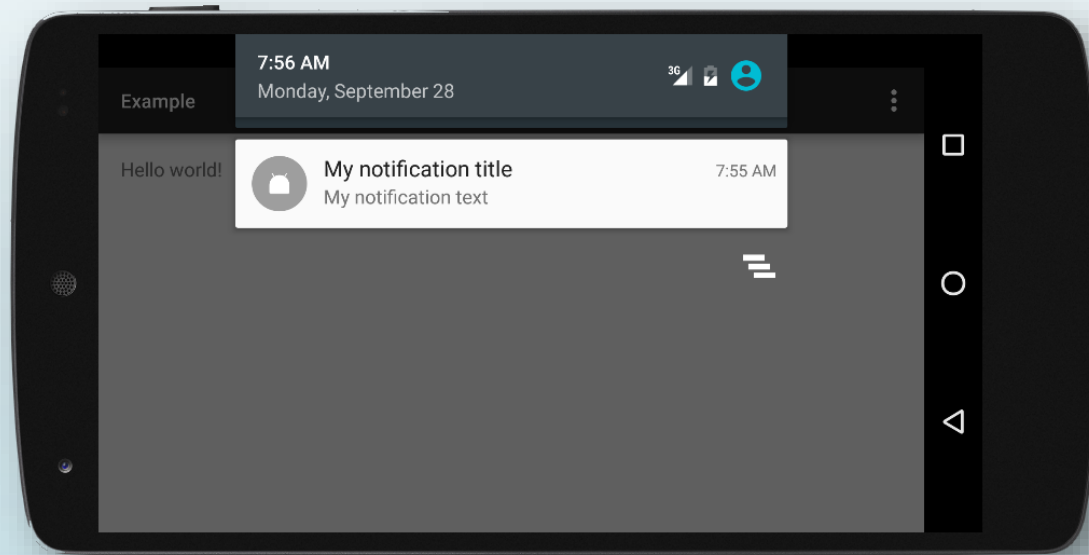
- Pour spécifier l'*Intent* qui soit être effectuée au clique, on utilise un *PendingIntent* :
 - Exemple avec `PendingIntent.getActivity(context, requestCode, intent, flag)`

```
PendingIntent intent = PendingIntent.getActivity(getApplicationContext(),  
    5,  
    new Intent(getApplicationContext(), MainActivity.class),  
    0);  
builder.setContentIntent(intent);
```

Interfaces utilisateur avancées

Les notifications : Résultat

► Et voici le résultat attendu :





Interfaces utilisateur avancées

Les App Widgets

- Les App Widgets sont des petits composants applicatifs qui peuvent être affichés sur le Home Screen, ou un autre "App Widget Host".
- Pour publier un App Widget, on utilise un App Widget Provider.

Interfaces utilisateur avancées

Les App Widgets

- Comme tous vos composants applicatifs, votre *AppWidgetProvider* doit être déclaré dans le manifeste, au niveau de l'application :

```
<receiver
    android:name=".MyAppWidget" >
    <intent-filter>
        <action
            android:name="android.appwidget.action.APPWIDGET_UPDATE" />
    </intent-filter>
    <meta-data
        android:name="android.appwidget.provider"
        android:resource="@xml/my_app_widget_info" />
</receiver>
```

Interfaces utilisateur avancées

Les App Widgets

- Il faudra ensuite définir dans un fichier XML de ressources différents attributs, comme ses dimensions minimums, son intervalle de mise à jour, ou encore une éventuelle application de configuration.

```
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="56dp"
    android:minHeight="56dp"
    android:updatePeriodMillis="3600000"
    android:previewImage="@drawable/ic_cloud_white_24dp"
    android:initialLayout="@layout/widget_main"
    android:resizeMode="horizontal|vertical">
</appwidget-provider>
```



Interfaces utilisateur avancées

Les App Widgets

- L'étape suivante est de créer un Layout pour notre widget. Cette étape n'est pas très différente de la configuration d'un layout pour une activité ou un fragment.



Interfaces utilisateur avancées

Les App Widgets

- Pour échanger des données entre votre widget et votre application, on pourra utiliser les méthodes suivantes de notre App Widget Provider :
 - onUpdate
 - onAppWidgetOptionsChanged
 - onDeleted
 - onEnabled / onDisabled

Interfaces utilisateur avancées

Les App Widgets

```
public class MyAppWidget extends AppWidgetProvider {

    public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {

        for (int appWidgetId : appWidgetIds) {
            // Layout
            RemoteViews views = new RemoteViews(context.getPackageName(), R.layout.widget_main);

            // Create an Intent to launch MainActivity
            Intent intent = new Intent(context, MainActivity.class);
            PendingIntent pendingIntent = PendingIntent.getActivity(context, 0, intent, 0);
            views.setOnClickPendingIntent(R.id.ib_widget_more, pendingIntent);

            // ON REFRESH CLICK
            Intent refreshIntent = new Intent(context, MyAppWidget.class);
            refreshIntent.setAction(AppWidgetManager.ACTION_APPWIDGET_UPDATE);
            refreshIntent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_IDS, appWidgetIds);
            PendingIntent pIntent = PendingIntent.getBroadcast(context, 0, refreshIntent, PendingIntent.FLAG_UPDATE_CURRENT);
            views.setOnClickPendingIntent(R.id.ib_widget_refresh, pIntent);

            // UPDATE TEXT & ...
            views.setTextViewText(R.id.tv_widget_last_update, new Date().toString());

            // Tell the AppWidgetManager to perform an update on the current app widget
            appWidgetManager.updateAppWidget(appWidgetId, views);
        }
    }
}
```



Interfaces utilisateur avancées

Les App Widgets

► Plus d'informations :

<http://developer.android.com/guide/topics/appwidgets/index.html>



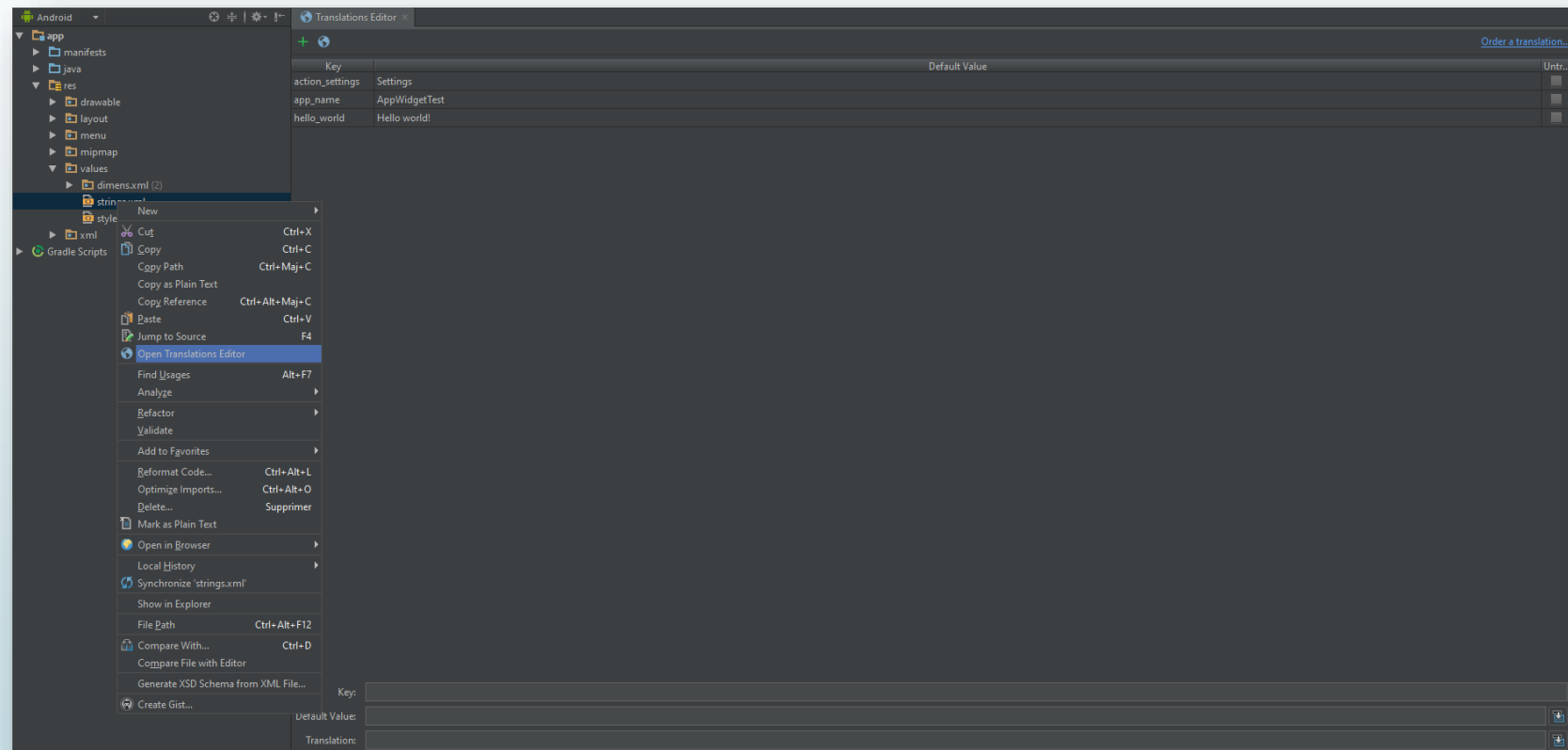
Interfaces utilisateur avancées

Internationalisation

- Nous utilisons depuis le début des fichiers de ressources pour stocker nos chaînes de caractères (*strings.xml*). Ce n'est pas qu'une bonne pratique, c'est aussi un prérequis à l'internationalisation de notre application !
- Pour ajouter de nouvelles langues, il suffit de faire un clic droit sur le fichier *strings.xml*, et de sélectionner « Open Translation Editor ».
- L'icône « + » permet d'ajouter de nouvelles ressources, l'icône « planète terre » permet d'ajouter de nouvelles langues.

Interfaces utilisateur avancées

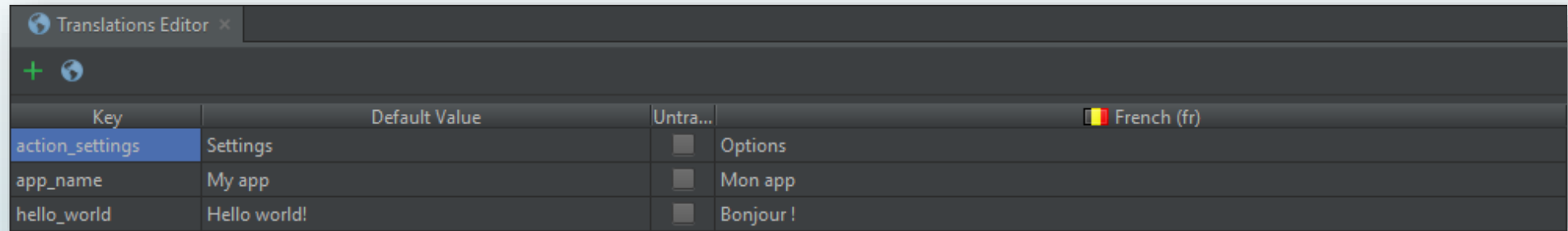
Internationalisation



Interfaces utilisateur avancées

Internationalisation

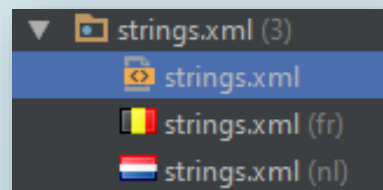
➤ Exemple :



The screenshot shows the 'Translations Editor' window in Android Studio. It features a table with columns for 'Key', 'Default Value', 'Untra...', and a language selector. The language is set to 'French (fr)'. The table contains three rows of data.

Key	Default Value	Untra...	French (fr)
action_settings	Settings	<input type="checkbox"/>	Options
app_name	My app	<input type="checkbox"/>	Mon app
hello_world	Hello world!	<input type="checkbox"/>	Bonjour !

➤ En réalité, un fichier différent est créé par langue :





Interfaces utilisateur avancées

Conclusion

- Vous êtes maintenant capables de proposer des interfaces riches et dynamiques à vos utilisateurs.
 - En utilisant les styles et thèmes.
 - En créant des menus, des sous-menus et des menus contextuels.
 - En proposant des interfaces basées sur des fragments
 - En créant des app widgets
- Vous pouvez également ouvrir vos applications au monde entier grâce à l'internationalisation.



Multithreading

Introduction

- Il peut arriver qu'on veuille effectuer une tâche sans que celle-ci ne soit visible par l'utilisateur.
- Le chargement de données peut parfois prendre un certain temps avant que notre application ne dispose des informations à afficher.
- Le temps de réponse d'une application est un critère essentiel à son succès. Après quelques secondes d'attente, votre application est considérée comme ne répondant plus.
- Heureusement, il existe plusieurs solutions à ces problématiques.

Multithreading

Threads et Runnable

- En java, le multithreading est géré grâce aux Threads et aux Runnables. Cela fonctionne également en Android :

```
new Thread(new Runnable() {  
    @Override  
    public void run() {  
        Log.i("Thread 2", "Background thread");  
    }  
}).start();
```

- Par contre, impossible de modifier les éléments de l'interface graphique dans ce code, car ils sont gérés uniquement par le Thread principal !



Multithreading

AsyncTask

- Afin d'effectuer facilement des tâches en arrière-plan, tout en ayant la possibilité d'interagir avec l'utilisateur, on peut utiliser l'*AsyncTask*.
- Afin de créer une *AsyncTask*, il suffit d'hériter de cette classe, et d'implémenter les méthodes nécessaires.
- Vu que l'*AsyncTask* est une classe générique, il faut également spécifier les types qui seront utilisés en entrée, sortie, et à chaque progression de notre tâche.



Multithreading AsyncTask

- Une *AsyncTask* se structure autour de plusieurs méthodes :
 - *onPreExecute()*
 - *doInBackground()*
 - *onProgressUpdate()*
 - *onPostExecute()*
- La méthode *doInBackground()* est la seule obligatoire, les autres sont facultatives. Par contre, on peut accéder au Thread principal dans ces autres méthodes, pas dans le *doInBackground()* !



Multithreading AsyncTask

```
public class MyAsyncTask extends AsyncTask<String, Integer, String> {  
  
    @Override  
    protected void onPreExecute() {  
    }  
  
    @Override  
    protected String doInBackground(String... params) {  
        return "";  
    }  
  
    @Override  
    protected void onProgressUpdate(Integer... values) {  
    }  
  
    @Override  
    protected void onPostExecute(String s) {  
    }  
}
```

Multithreading

AsyncTask

- Pour exécuter une *AsyncTask*, on appelle **jamais** nous-même ces méthodes ! À la place, on appelle la méthode *execute()*.

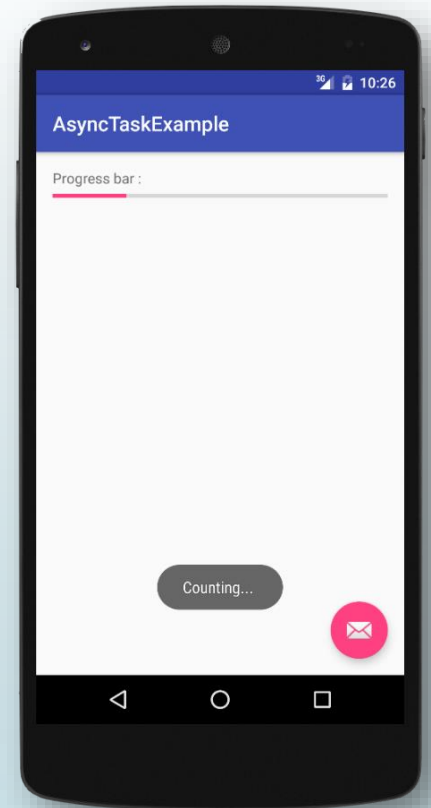
```
MyAsyncTask task = new MyAsyncTask();  
task.execute("MonUrl");
```

- Si l'on souhaite accéder à les éléments de notre activité dans notre *AsyncTask* (une barre de progression, un texte, le contexte,...), il faut les passer dans le constructeur, ou, bonne pratique, utiliser une interface de callback.

Multithreading

AsyncTask : Exemple complet

- Le but est d'arriver à une application comme celle-ci :
 - Notre activité lance une tâche de compteur de 1 à 100.
 - Notre AsyncTask va informer l'utilisateur tout en réalisant la tâche demandée.



Multithreading

AsyncTask : Exemple complet

- D'abord, l'AsyncTask et son interface de callback :

```
public class CustomAsyncTask extends AsyncTask<Integer, Integer, Integer> {  
  
    public interface CustomInterface {  
        void toastMessage(String message);  
        void updateProgressBar(int progress);  
    }  
  
    private CustomInterface callback;  
  
    public CustomAsyncTask(CustomInterface callback) {  
        this.callback = callback;  
    }  
}
```

Multithreading

AsyncTask : Exemple complet

► Ensuite, les méthodes qui ont accès à l'UI :

```
@Override
protected void onPreExecute() {
    callback.toastMessage("Counting... ");
}

@Override
protected void onProgressUpdate(Integer... values) {
    callback.updateProgressBar(values[0]);
}

@Override
protected void onPostExecute(Integer integer) {
    callback.toastMessage("Reached " + integer + " ! ");
}
```

Multithreading

AsyncTask : Exemple complet

► La méthode `doInBackground()` :

```
@Override
protected Integer doInBackground(Integer... params) {
    int i = params[0];
    while (i < params[1]) {
        i++;
        publishProgress(i);
        try {
            Thread.sleep(50);
        } catch (InterruptedException e) {
            Log.e("CAT", e.getLocalizedMessage());
        }
    }
    return i;
}
```

Multithreading

AsyncTask : Exemple complet

- Et enfin, l'activité, qui implémente l'interface définie plus tôt...

```
public class MainActivity extends AppCompatActivity implements CustomInterface {  
  
    ProgressBar pb_main_progress;  
  
    @Override  
    public void toastMessage(String message) {  
        Toast.makeText(getApplicationContext(), message, Toast.LENGTH_SHORT).show();  
    }  
  
    @Override  
    public void updateProgressBar(int progress) {  
        pb_main_progress.setProgress(progress);  
    }  
}
```

Multithreading

AsyncTask : Exemple complet

► ...et qui lance l'exécution de la tâche !

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    pb_main_progress = (ProgressBar) findViewById(R.id.pb_main_progress);

    CustomAsyncTask task = new CustomAsyncTask(this);
    task.execute(1, 100);
}
```



Multithreading Service

- Les services sont des **composants** logiciels Android, tout comme les activités.
 - Ils doivent être déclarés dans le manifeste.
 - Ils possèdent un cycle de vie.
 - Ils effectuent un traitement spécifique.
- Par contre, le service tourne en **arrière-plan**, et ne possède **pas** d'interface graphique.
- Exemple : un lecteur MP3 possède une activité (avec les contrôles), ainsi qu'un service qui joue la musique.



Multithreading IntentService

- Les IntentServices sont similaires aux Services, sauf qu'ils effectuent un traitement de manière asynchrone suite aux Intents qu'ils reçoivent.
- Ils sont plus simples à implémenter que les services.



Les Content Providers



Les Content Providers

Introduction

- Sur Android, le stockage de grandes quantités de données se fait grâce à une base de données SQLite.
- Cependant, il n'est pas possible d'accéder aux données stockées par une autre application !
- Cela pose un problème : comment gérer des données communes à plusieurs applications, ou qui peuvent être partagées ?
 - Répertoire, Agenda, SMS, Médias,...



Les Content Providers

Introduction

- Afin de permettre le **partage** des données, Android propose la notion de **fournisseur de contenu** (Content Provider).
 - On peut donc accéder aux données partagées par d'autres applications.
 - On peut aussi rendre disponibles nos données !
- **Attention !** Il est possible de donner la possibilité aux autres applications de **modifier** les données, même si c'est rarement le cas. Par contre, les applications systèmes le permettent !



Les Content Providers

Mise en place

- Pour accéder à un fournisseur de contenu, on utilise un **ContentResolver**.
 - On peut obtenir un grâce à la méthode **getContentResolver()**.
- Chaque fournisseur de contenu possède une ou plusieurs **URI** qui permet(tent) de l'identifier, cette propriété s'appelle **CONTENT_URI**.
- Comme en base de données, nos requêtes sur un fournisseur de contenu retournent un objet **Cursor**.

Les Content Providers

Exemple de Query

- Pour utiliser certains *ContentProviders*, il faut demander une **permission** :

```
<uses-permission android:name="android.permission.READ_CONTACTS" />  
<uses-permission android:name="android.permission.WRITE_CONTACTS" />
```

- Ensuite, on récupère l'URI à utiliser, et on exécute une *query()* :

```
Uri uri = ContactsContract.Contacts.CONTENT_URI;  
Cursor c = getContentResolver().query(uri, null, null, null, null);
```



Les Content Providers

Insertions et modifications

- Si le fournisseur de contenu permet la modification des données, on peut utiliser d'autres méthodes du ContentResolver :
 - insert(), qui prend en paramètres l'URI visée, ainsi d'un objet ContentValues des valeurs à ajouter.
 - update(), qui prend en paramètres l'URI, l'objet ContentValues qui contient les données modifiées, ainsi qu'un filtre permettant de spécifier quelle entrée sera affectée par la modification.
 - delete(), qui prend en paramètres l'URI, ainsi qu'un filtre.



Les Content Providers

Conclusion

- Les **fournisseurs de contenus** sont une fonctionnalité puissante d'Android, qui permet le **partage** des **données** entre plusieurs applications. C'est grâce à eux que vous pouvez importer vos contacts dans une application, ou encore remplacer l'application de SMS native !
- Un fournisseur de contenu est une **couche d'abstraction** supplémentaire pour accéder aux données de l'application.
- Souvent, une application ne permet que la **lecture** des données, cependant, les **modifications** peuvent parfois être aussi autorisées.

Réseau, connexion et services web

Introduction

- Depuis plusieurs versions, Android interdit les communications réseau dans le Thread principal !
- La façon la plus simple de travailler est, le plus souvent, d'utiliser une AsyncTask.
- Dès qu'on utilise le réseau, il faut demander certaines permissions :

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




Réseau, connexion et services web

Connexion et requête GET

- D'abord, on commence par créer une *URL* :

```
URL url = new URL("http://www.google.be");
```

- Ensuite, on récupère une *URLConnection*, et on se connecte :

```
URLConnection connection = (URLConnection)url.openConnection();  
connection.setRequestMethod("GET");  
connection.connect();
```

- Finalement, on peut obtenir un *InputStream* grâce à :

```
InputStream inputStream = connection.getInputStream();
```



Réseau, connexion et services web

Connexion et requête GET

- Il faut ensuite traiter cet InputStream, par exemple grâce à un InputStreamReader, un BufferedReader et un StringBuilder, afin d'obtenir une chaîne de caractère qui contiendra tout ce qui a été renvoyé par le serveur.
- Si ce type de connexions peut paraître peu utile (le but n'est pas d'afficher tout l'HTML d'un site web !), elle peut déjà nous permettre d'interagir avec des webservices.



Réseau, connexion et services web

Web API : Introduction

- Pour utiliser une web API avec Android, il faut d'abord comprendre ces différents concepts :
 - RESTful Web API
 - Client
 - JSON



Réseau, connexion et services web

Web API : Concepts

► RESTful Web API

- Une API web RESTful est un service web basé sur l'architecture REST. Cela signifie que c'est un service accessible via le protocole HTTP et qui renvoie les informations demandées en JSON ou en XML.

► Client

- Le client d'une web API est une application qui utilise des informations qu'elle a récupérée depuis cette API. Concrètement, l'application fera des requêtes pour obtenir des données, puis traitera ces données afin de les proposer à l'utilisateur.



Réseau, connexion et services web

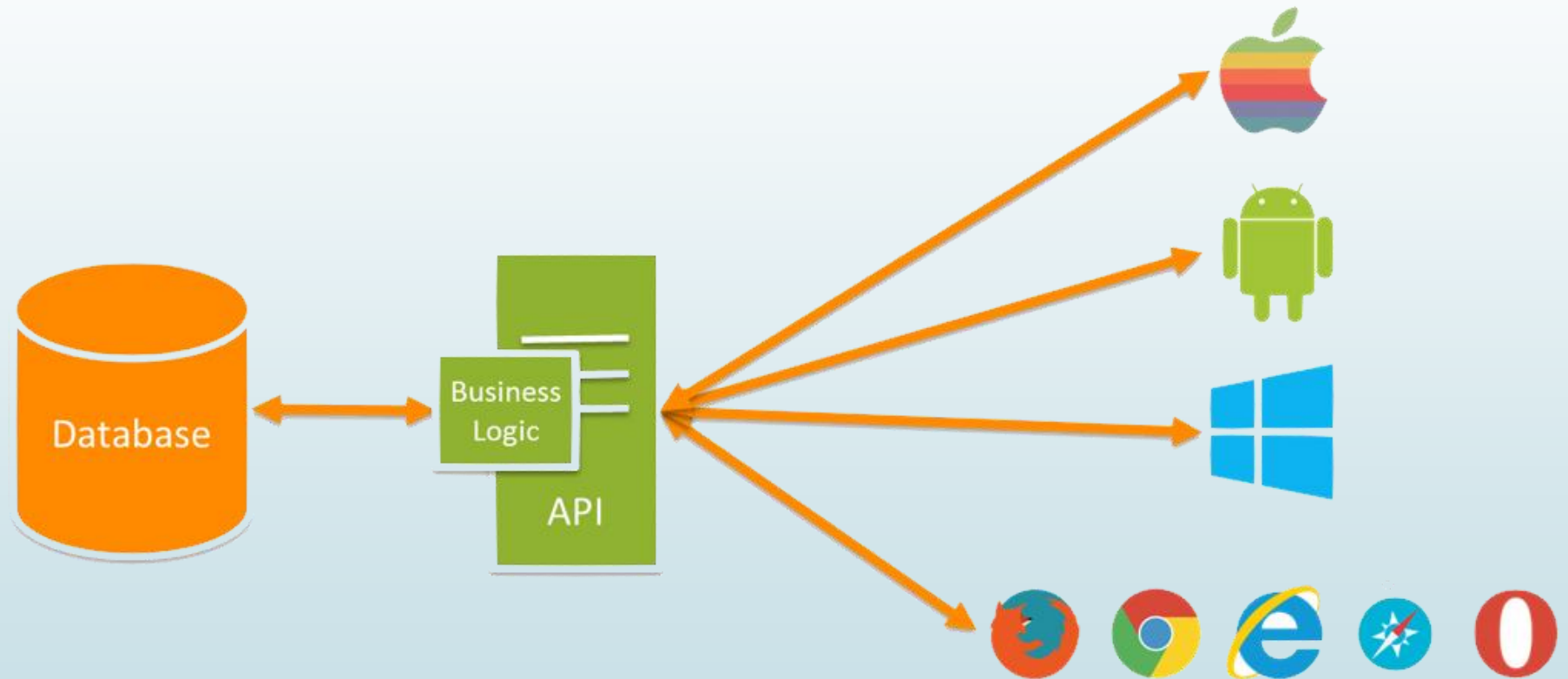
Web API : Concepts

► JSON

- Le JSON (pour JavaScript Object Notation) est un format de données qui permet, comme le XML, de structurer des données. Sa puissance vient du fait qu'il n'est composé que de texte, et qu'il est donc très léger, et facilement utilisable par de nombreux langages.

Réseau, connexion et services web

Web API : Concepts





Réseau, connexion et services web

Web API : Outils

- Pour plus de facilité, on peut utiliser des outils ainsi que des libraires :

- <http://www.jsonschema2pojo.org>

- Ce site créera pour vous les classes équivalentes au JSON qui vous est rendu.

- GSON

- Cette librairie, créée par Google, permet de convertir le JSON en objets java, et inversement.



Réseau, connexion et services web

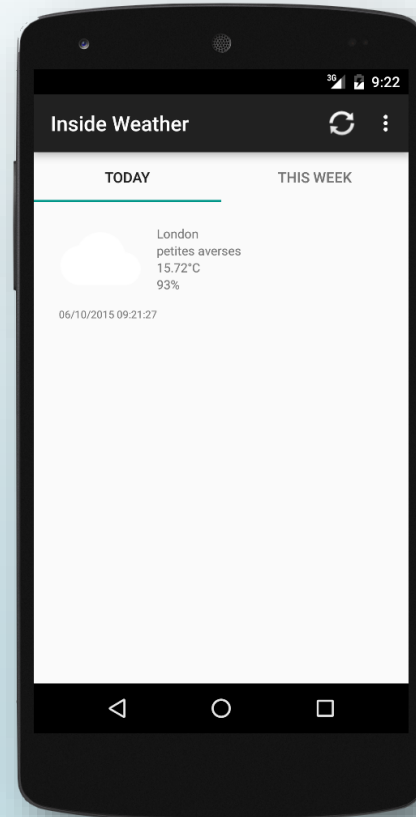
Web API : Exemples

- Il existe des web APIs pour de nombreuses utilisations :
 - Musique (Spotify,...)
 - Météo (OpenWeatherMap, AccuWeather,...)
 - Réseaux sociaux (Facebook, Twitter, Google+, LinkedIn,...)
 - Gaming (Steam, GiantBomb,...)
 - Vidéo (Youtube,...)
 - Photos (Flickr,...)
- Google propose d'ailleurs de nombreux services :
- Maps, StreetView, Places, Calendar, Gmail,...

Réseau, connexion et services web

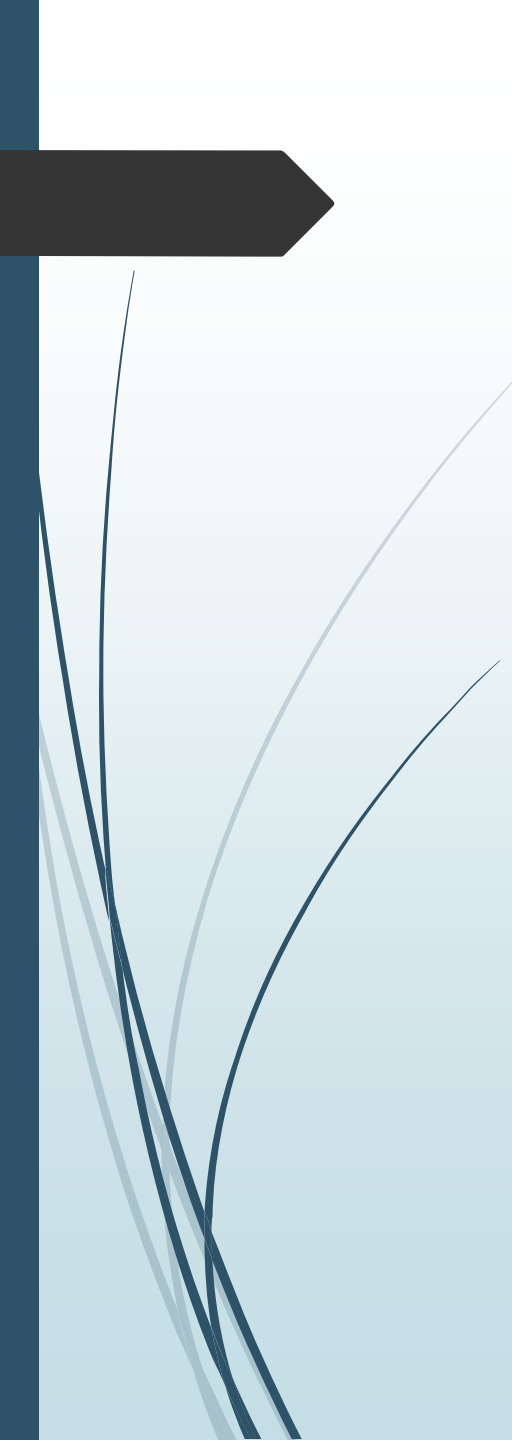
Exercice

1. Utilisez l'API d'OpenWeatherMap pour créer une application Météo





Publier son application



Publier son application

Introduction

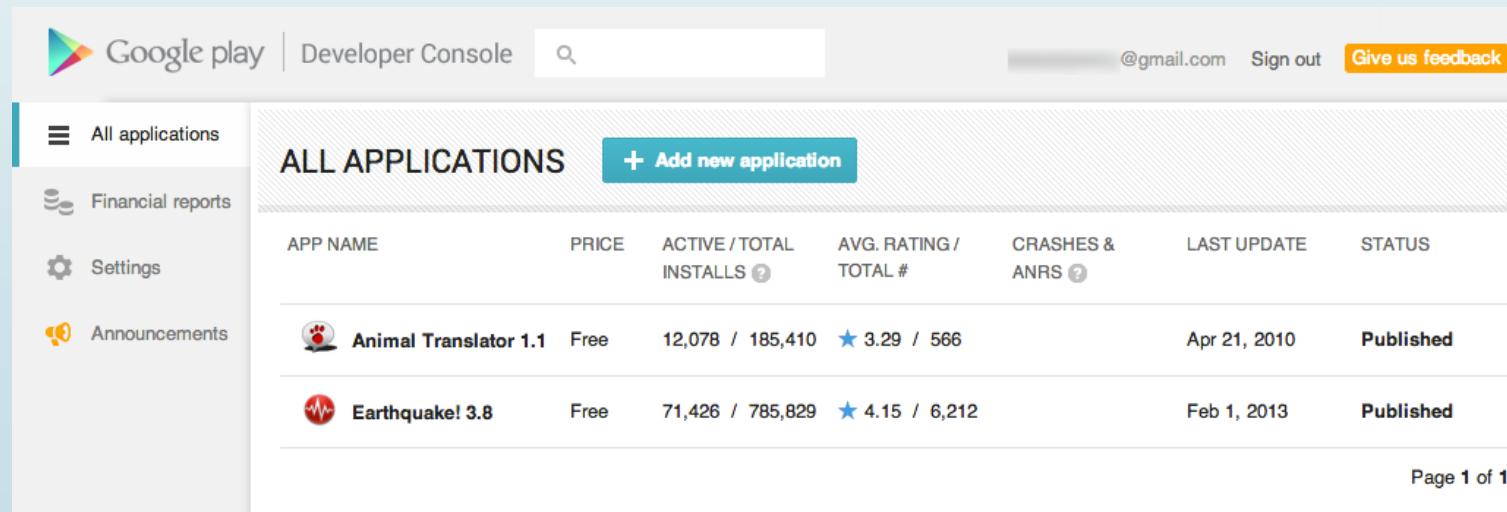
- La publication est la dernière étape du développement d'une application.
- Une application peut être distribuée par n'importe quel moyen (il ne s'agit que d'un fichier .apk), mais le plus souvent, il est mis en ligne sur Google Play.
- Google propose une documentation abondante afin de publier son application :

http://developer.android.com/tools/publishing/publishing_overview.html



Publier son application

Création du compte

- La création d'un compte développeur sur Google Play est payante (25€ à vie).
- Ce compte permet d'accéder à la Developer Console, où vous pouvez gérer vos publications et vos outils.



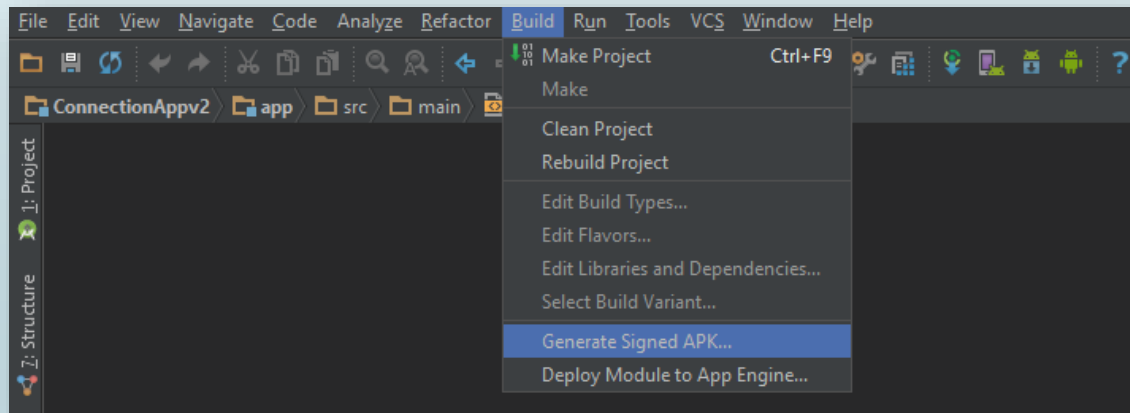
The screenshot displays the Google Play Developer Console interface. At the top, there's a header with the Google Play logo, 'Developer Console', a search bar, and user information including an email address and a 'Sign out' button. Below the header, a sidebar on the left contains navigation links: 'All applications', 'Financial reports', 'Settings', and 'Announcements'. The main content area is titled 'ALL APPLICATIONS' and includes a '+ Add new application' button. It features a table with columns for APP NAME, PRICE, ACTIVE / TOTAL INSTALLS, AVG. RATING / TOTAL #, CRASHES & ANRS, LAST UPDATE, and STATUS. Two applications are listed: 'Animal Translator 1.1' and 'Earthquake! 3.8', both marked as 'Published'. The bottom right corner shows 'Page 1 of 1'.

APP NAME	PRICE	ACTIVE / TOTAL INSTALLS	AVG. RATING / TOTAL #	CRASHES & ANRS	LAST UPDATE	STATUS
 Animal Translator 1.1	Free	12,078 / 185,410	★ 3.29 / 566		Apr 21, 2010	Published
 Earthquake! 3.8	Free	71,426 / 785,829	★ 4.15 / 6,212		Feb 1, 2013	Published

Publier son application

Signer son application

- Pour pouvoir être publiée, une application doit être signée. Cette signature est généralement gardée par le développeur de l'application. Elle ne doit pas être certifiée par un organisme externe, et sert simplement à identifier l'auteur de l'application.
- Le plus simple est de le faire directement dans Android Studio





La géolocalisation



La géolocalisation

Introduction

- Deux moyens de localisation
 - Par GPS
 - Par triangulation
- Géolocalisation par GPS plus gourmande
- Géolocalisation par triangulation est moins précise mais moins consommatrice

La géolocalisation

Permission

- ACCESS_FINE_LOCATION est utilisé pour le GPS

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

- ACCESS_COARSE_LOCATION est utilisé par la triangulation

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


La géolocalisation

Utilisation

- La géolocalisation possède un service Android: **LocationManager**
- Récupération des fournisseurs

```
//NAME OF PROVIDER = LocationManager.GPS_PROVIDER et LocationManager.NETWORK_PROVIDER
ArrayList<LocationProvider> listProvider = new ArrayList<LocationProvider>();

List<String> names = locationManager.getProviders(true);

for (String name : names) {
    listProvider.add(locationManager.getProvider(name));
}
Log.i("PROVIDERS", listProvider.toString());
```

La géolocalisation

Utilisation

- Il est possible d'utiliser des critères permettant de sélectionner un fournisseur bien précis

```
Criteria criteria = new Criteria();

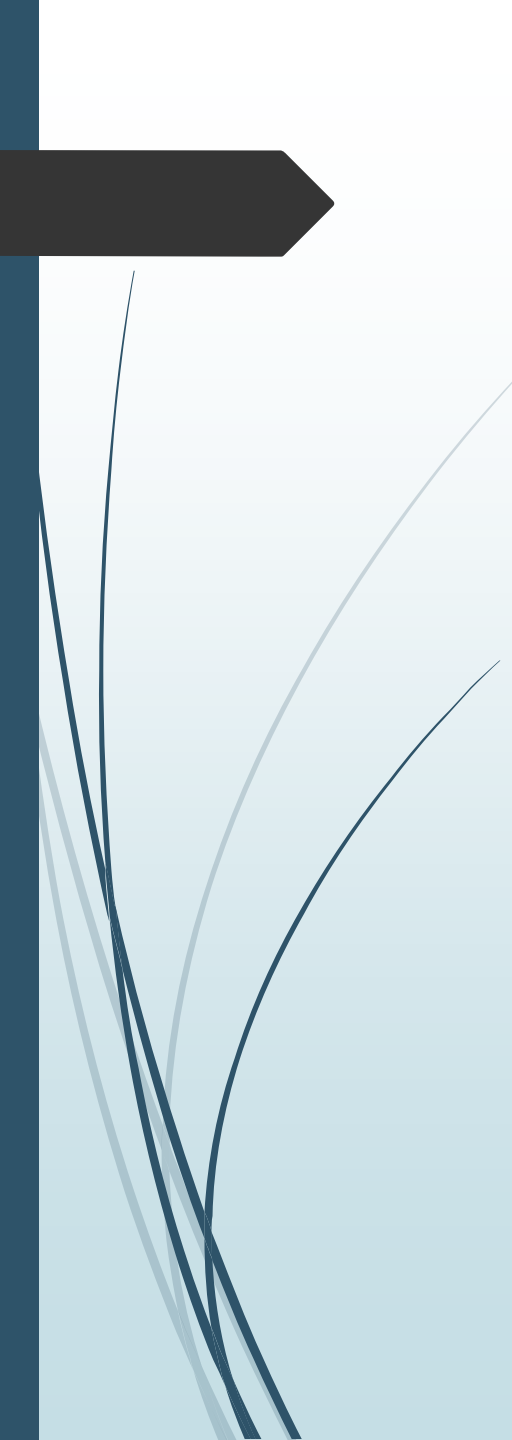
//ACCURACY_FINE, ACCURACY_LOW, ACCURACY_MEDIUM, ACCURACY_HIGH
criteria.setAccuracy(Criteria.ACCURACY_COARSE);
criteria.setSpeedAccuracy(Criteria.ACCURACY_COARSE);

criteria.setAltitudeRequired(false); // Donne l'altitude ?
criteria.setBearingRequired(true); // Donne une direction ?
criteria.setCostAllowed(false); // Fournisseur payant ?
criteria.setSpeedRequired(false); // Donne la vitesse ?

//POWER_MEDIUM, POWER_HIGH, POWER_LOW
criteria.setPowerRequirement(Criteria.POWER_MEDIUM);

// Fourni le meilleur provider répondant au critère
nameProvider = locationManager.getBestProvider(criteria, true);

locationProvider = locationManager.getProvider(nameProvider);
```



La géolocalisation

Utilisation (2)

- Mettre à jours la localisation
 - Utilisation d'un évènement
- Ces mises à jours peuvent s'exécuter seulement si votre application est en cours d'exécution
 - Utilisation d'un PendingIntent pour l'activé et d'un BroadcastReceiver
- Il existe un système d'alerte de proximité



Gestion des mouvements



Gestion des mouvements

- Utilisation des événements onTouch
 - Méthode onTouchEvent(MotionEvent event)
 - Possède des constantes permettant de connaître l'action effectuée.
- Deux interfaces
 - OnGestureListener
 - OnDoubleTapListener



Gestion des mouvements

Traquer la vitesse

- Utilisation de VelocityTracker
- Cette vitesse peut être obtenue par la méthode `static obtain()`