



DEVELOPPEMENT MOBILE

1 Présentation du TP : Prise en main de l'environnement de développement d'Android

1.1 Objectif

L'objectif de ce TP est de vous permettre d'exploiter les outils permettant le développement d'application pour Android. L'existence d'un IDE (**integrated development environment**) dédié permet que l'installation des outils de développement pour Android soit beaucoup plus facile qu'il y a quelques années. Ce premier TP n'a pas pour but de vous donner une description détaillée de ces outils, mais de vous proposer un fil conducteur vous permettant de vous familiariser avec le monde du développement sous Android.

1.2 Exercice 1 : Premier application android

Une version de Android studio est installée sur les ordinateurs de l'ENSICAEN sous Linux.

Premier lancement de l'IDE

Créer une votre première application « **hello world** ».

1.3 Calculatrice scientifique

L'objectif de cet exercice est de créer une calculatrice permettant de faire des calculs que dans la base décimale.

Pour sa réalisation vous devez suivre les étapes suivantes :

1. Changer le nom de l'application « **hello world** »,
2. Ajouter un bouton avec comme libellé "**calculatrice**",
3. Créer l'interface **calculatrice**,
4. Lancer l'interface calculatrice à partir d'un clic sur le bouton déjà créé,
5. Ajouter le menu vous permettant de changer la base de calcul.



1.4 Flux RSS


L'objectif principal de cet exercice est de comprendre les principes permettant la réalisation d'une application Android exploitant les données fournies par un flux RSS pour les afficher directement dans une vue à partir des possibilités offertes par le SDK d'Android.

Afin de pouvoir réaliser le chargement des données XML à partir d'un flux RSS, nous exploiterons un parseur XML de type SAX. De plus, Android n'autorisant plus le chargement de données provenant d'Internet à partir du thread principal de l'interface graphique, nous exploiterons l'interface **ExecutorService** pour réaliser ce chargement.

Exploitation d'un flux RSS

Nous allons exploiter les flux RSS fournis par le journal « **le Monde** ». La page d'accueil¹ du journal propose plusieurs rubriques :

- ✓ **Actualités** [A la une](#) | [En continu](#) | [Vidéos](#) | [Portfolios](#) | [Les plus lus](#) | [Les plus partagés](#)
- ✓ **International** [La une International](#) | [Europe](#) | [Amériques](#) | [Afrique](#) | [Asie Pacifique](#) | [Proche-Orient](#) | [Royaume-Uni](#) | [Etats-Unis](#)
- ✓ **France** [Politique](#) | [Société](#) | [Les décodeurs](#) | [Justice](#) | [Police](#) | [Campus](#) | [Education](#)
- ✓ **Economie** [La une Economie](#) | [Entreprises](#) | [Argent](#) | [Économie française](#) | [Industrie](#) | [Emploi](#) | [Immobilier](#) | [Médias](#)
- ✓ **Culture** [La une Culture](#) | [Cinéma](#) | [Musiques](#) | [Télévision et radio](#) | [Le Monde des livres](#) | [Arts](#) | [Scènes](#)
- ✓ **Sport** [La une Sport](#) | [Football](#) | [Rugby](#) | [Tennis](#) | [Cyclisme](#) | [Basket](#)
- ✓ **Planète** [La une Planète](#) | [Climat](#) | [Agriculture](#) | [Environnement](#)
- ✓

Le Monde |  ACTUALITÉS ▾ ÉCONOMIE ▾ VIDÉOS ▾ OPINIONS ▾ CULTURE ▾ M LE MAG ▾ SERVICES ▾ 

Les flux RSS du Monde.fr

Publié le 12 août 2019 à 15h23 - Mis à jour le 12 août 2019 à 17h19

Un flux RSS correspond à un dialecte **XML** dont les tags sont normalisés. Dans notre cas, nous allons nous intéresser aux contenus des **<item>**, en particulier aux tags : **<title>**, **<description>**, **<pubDate>** pour les contenus textuels et au tag **<enclosure>** pour l'url de l'image.

Voici un extrait du flux RSS correspondant à l'url https://www.lemonde.fr/international/rss_full.xml:

¹ https://www.lemonde.fr/actualite-medias/article/2019/08/12/les-flux-rss-du-monde-fr_5498778_3236.html



<item>

<title>

<![CDATA[Ukraine : la stratégie occidentale validée par la contre-offensive]]>

</title>

<pubDate>Tue, 13 Sep 2022 12:31:15 +0200**</pubDate>**

<description><![CDATA[La percée de l'armée ukrainienne sur le Donbass et dans la région de Kherson est un revers puissant pour Vladimir Poutine et témoigne d'un renversement du rapport de force entre Kiev et Moscou.]]> **</description>**

<guid isPermaLink="true"><https://www.lemonde.fr/idees/article/2022/09/13/ukraine- la-strategie- occidentale-validee-par-la-contre-offensive 6141421 3232.html>

</guid>

<link><https://www.lemonde.fr/idees/article/2022/09/13/ukraine-la-strategie-occidentale-validee-par-la- contre-offensive 6141421 3232.html>

</link>

<media:content

url="https://img.lemde.fr/2022/09/12/111/0/2500/1250/644/322/60/0/eee07b9_aaeb79f20a674887a685899 11a1fad84-aaeb79f20a674887a68589911a1fad84-0.jpg**" width="644" height="322"/>**

</item>

Pour afficher les données extraites nous prévoyons une vue de conception minimaliste. La description XML étant la suivante :

Code XML 1



```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
```

```
<LinearLayout
    android:id="@+id/linearLayout2"
    android:layout_width="match_parent"
    android:layout_height="48dp"
    android:layout_gravity="center_horizontal"
    android:layout_marginTop="16dp"
    android:layout_marginEnd="6dp"
    android:orientation="horizontal"
    android:weightSum="1"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent">
```

```
<Button
    android:id="@+id/buttonPrev"
    style="?android:attr/buttonStyleSmall"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="0.5"
    android:onClick="previousItem"
    android:text="@string/image_precedente" />
```

```
<Button
    android:id="@+id/buttonNext"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="0.5"
    android:onClick="nextItem"
    android:text="@string/image_suivante" />
```

```
</LinearLayout>
```

```
<ScrollView
    android:id="@+id/scrollView"
    android:layout_width="fill_parent"
    android:layout_height="0dp"
    android:layout_marginStart="6dp"
    android:layout_marginTop="16dp"
    android:tag="MyScrollView"
```



```
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@+id/linearLayout2"
app:layout_constraintVertical_bias="0.0"
tools:context=".MainActivity"
tools:ignore="MissingConstraints">
```

```
<LinearLayout
    android:id="@+id/linearLayout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:tag="MyLinearLayout">
```

```
<TextView
    android:id="@+id/imageTitle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="10dp"
    android:tag="MyImageTitle"
    android:text="@string/test_image_title" />
```

```
<TextView
    android:id="@+id/imageDate"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="10dp"
    android:tag="MyImageDate"
    android:text="@string/test_image_date" />
```

```
<ImageView
    android:id="@+id/imageDisplay"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:adjustViewBounds="true"
    android:contentDescription="@string/description_image"
    android:src="@mipmap/ic_launcher"
    android:tag="MyImageDisplay" />
```

```
<TextView
    android:id="@+id/imageDescription"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp"
    android:layout_marginBottom="10dp"
    android:tag="MyImageDescription"
    android:text="@string/test_image_description" />
```



```
</LinearLayout>
</ScrollView>
```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

Les trois **TextViews** sont pour le titre, la date de parution et la description et l'**ImageView** pour l'image. Nous englobons le tout dans un **ScrollView** afin que si la quantité de données affichées dépasse la taille de l'écran, il soit possible de faire scroller cet affichage pour en voir la totalité. Nous créons ensuite une classe **MyRSSsaxHandler** qui hérite de la classe **DefaultHandler**². Cette classe fait partie du paquetage classique **org.xml.sax**. Un ensemble d'attributs sont définis pour cette classe afin de savoir où on est situé dans le flux RSS et sauvegarder les données lues :

```
private String url = null ;// l'URL du flux RSS à parser
// Ensemble de drapeau permettant de savoir où en est le parseur dans le flux XML
private boolean inTitle = false ;
private boolean inDescription = false ;
private boolean inItem = false ;
private boolean inDate = false ;
// L'image référencée par l'attribut url du tag <enclosure>
private Bitmap image = null ;
private String imageURL = null ;
// Le titre, la description et la date extraits du flux RSS
private StringBuffer title = new StringBuffer();
private StringBuffer description = new StringBuffer();
private StringBuffer date = new StringBuffer();
private int numItem = 0 ;// Le numéro de l'item à extraire du flux RSS
private int numItemMax = - 1 ;// Le nombre total d'items dans le flux RSS
```

La méthode **setUrl** permet de fixer l'URL à parser.

```
public void setUrl(String url){
    this.url= url;
}
```

La méthode **processFeed** initialise le parser XML, ouvre un flux d'entrée à partir de l'adresse Internet du flux RSS (**new URL(url).openStream()**), réalise sa lecture (**reader.parse(...)**), puis charge l'image correspondante (**getBitmap**).

²<https://developer.android.com/reference/org/xml/sax/helpers/DefaultHandler>



```
public void processFeed(){
    try {
        numItem = 0; //A modifier pour visualiser un autre item

        SAXParserFactory factory = SAXParserFactory.newInstance();
        SAXParser parser = factory.newSAXParser();
        XMLReader reader = parser.getXMLReader();
        reader.setContentHandler(this);
        InputStream inputStream = new URL(url).openStream();
        reader.parse(new InputSource(inputStream));
        image = getBitmap(imageURL);
        numItemMax = numItem;
    } catch (Exception e) {
        Log.e("smb116rssview", "processFeed Exception" + e.getMessage());
    }
}
```

Pour extraire les informations du flux XML (i.e. RSS) nous utilisons la méthode **startElement** qui est appelée à chaque rencontre de tag XML ouvrant et la méthode **characters** qui permet d'extraire les données textuelles entre deux tags XML.

Donnez le code correspondant à l'extraction des données du flux RSS pour les éléments : **title**, **description**, **pubDate** et **media:content** (pour ce dernier, il faut réaliser l'extraction de l'URL de l'image, si disponible) pour un item donné (correspondant à numItem). Pour vérifier le bon fonctionnement de l'extraction, vous afficherez les données extraites dans le LogCat (voir suite pour la problématique d'un chargement asynchrone). Le prototype de startElement étant le suivant :

```
public void startElement(String uri, String localName, String qName, Attributes
    attributes) throws SAXException {

    //...
}
```

Vous devrez également définir la méthode **characters** pour récupérer les contenus textuels correspondant à un tag:

```
public void characters(char ch[], int start, int length){
    String chars = new String(ch).substring(start, start+length);
    //...
}
```

La lecture d'une image à partir de son URL sera réalisée en exploitant la classe **HttpURLConnection**. L'extraction d'un bitmap à partir d'un stream sera réalisée à partir de la méthode **BitmapFactory.decodeStream()**. Compléter le code de la méthode suivante pour réaliser ce travail :



```
public Bitmap getBitmap(String imageURL){
{
    //...
}
}
```

Afin de pouvoir télécharger des données à partir d'Internet, il faut donner à l'application la permission de réaliser ce type de tâche. Pour cela, il faut modifier le fichier AndroidManifest.xml en ajoutant la permission internet :

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

Nous pouvons cependant ajouter dans la méthode **onCreate** de l'activité de notre application le chargement direct des données à partir du flux RSS de la façon suivante :

```
MyRSSsaxHandler handler = new MyRSSsaxHandler();
handler.setUrl("https://www.lemonde.fr/international/rss_full.xml");
handler.processFeed();
```

L'interface **java.util.concurrent.Executor**, créée depuis le **JDK 1.5**, décrit les fonctionnalités permettant l'exécution différée de tâches implémentées sous la forme de **Runnable**.

Elle ne définit qu'une seule méthode : void **execute(Runnable command)**.

Cette dernière permet d'exécuter la tâche fournie en paramètre et elle peut être exécutée dans un thread dédié ou dans le thread courant.

L'interface **Executor** possède deux interfaces filles :

- **ExecutorService** : elle définit les fonctionnalités d'un service permettant l'exécution de tâches de type **Runnable** ou **Callable**.
- **ScheduledExecutorService** qui hérite de l'interface **ExecutorService** : elle définit les fonctionnalités d'un service pour l'exécution de tâches planifiées et/ou répétées

La classe **Executors** permet de créer des réserves de threads de différentes façons. Chaque méthode crée sa réserve suivant une certaine sémantique :

- **newCachedThreadPool()** : crée une réserve capable de créer des nouveaux threads à la demande, sans limite. Lorsqu'un thread n'est plus utilisé, cette réserve est capable de lui confier de nouvelles tâches. Lorsqu'une nouvelle tâche est confiée à cet executor, elle est lancée immédiatement.
- **newFixedThreadPool(int nThreads)** : crée le même type de réserve que la méthode précédente. La différence est que le nombre maximal de threads est fixé à la construction de cet objet. Dans ce cas, il est devenu possible qu'une tâche confiée à cette réserve ne puisse pas être exécutée immédiatement. Cette réserve gère donc une file d'attente dans laquelle ces tâches sont placées. Cette file d'attente ne peut pas saturer.
- **newSingleThreadExecutor()** : crée le même type de réserve que la méthode précédente. Cette fois, la réserve de threads ne comporte qu'un unique thread. Les tâches qui ne peuvent être immédiatement exécutées sont également placées dans une file d'attente.



La méthode **resetDisplay** doit être ajoutée à la classe **MainActivity**. Elle affichera les informations lues en modifiant le text (**setText**) ou l'image (**setImageBitmap**) des éléments de type View correspondant au layout XML de l'activité. Vous devrez exploiter la méthode **findViewById** afin de pouvoir réaliser ces modifications. Complétez le prototype ci-dessous

```
public void resetDisplay(String title, String date, Bitmap image, String description)
{
    // ...
}
```

La **ScrollView** vous permet de vous déplacer verticalement dans la vue.

Travail :

1. Ajoutez deux boutons en début de vue, alignés horizontalement, permettant le passage d'une image à la suivante (ou précédente). Les boutons feront parties de la **ScrollView**. Ils pourront donc disparaître lorsque l'on fait scroller le contenu du flux affiché vers le bas.
2. On souhaite que les deux boutons soit toujours visibles en haut de l'écran, tout en pouvant scroller sur les données affichées pour un item du flux RSS. Proposez une solution permettant d'avoir ces deux modalités.
3. Ajouter un menu contenant trois items (**Actualités, Economie, Sport,...**) permettant de charger les données du flux RSS de la une de chacun de ces trois rubriques.
4. Actuellement le passage d'un item à un autre, nécessite le rechargement du flux XML afin d'extraire le contenu de l'item n°i. Proposez une nouvelle version dans laquelle l'ensemble des données est chargé en mémoire lors du premier parcours du flux RSS. Le changement d'item en appuyant sur un bouton ne nécessitera alors plus de faire de rechargement. Le flux RSS pouvant avoir été modifié lors d'une longue utilisation de l'application, vous ajouterez au menu la possibilité de faire un rafraîchissement du contenu sauvegarder en mémoire.
5. Pensez à stocker les données dans une base de données qui sera utilisée lorsque la connexion internet sera coupée.

