TP : `ListView` et compagnie !

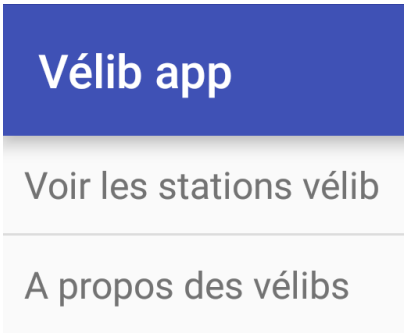
Vous allez, dans ce TP, construire une application s'exécutant sous Android. L'application présente tout d'abord une fenêtre avec deux items dans une liste , "Voir les stations Vélib" et "A propos des Velib". L'item "A propos des Vélib" amène un nouvel écran de renseignements sur les Vélibs, l'item "Voir les stations Vélib" amène un écran proposant la liste des stations Vélib. On interrogera pour cela l'URL http://www.velib.paris.fr/service/carto.

L'interface graphique

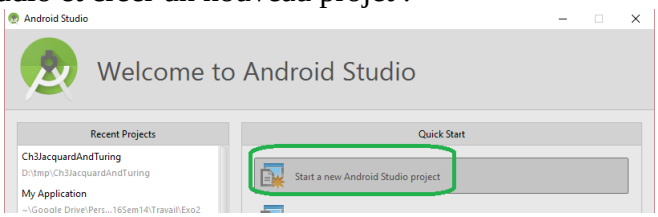
Première étape : Ecrire une `Activity` qui présente une `ListView` avec 2 items "Voir les stations Vélib" et "A propos des Vélib"

Première étape : Construire une IHM affichant une `ListView`

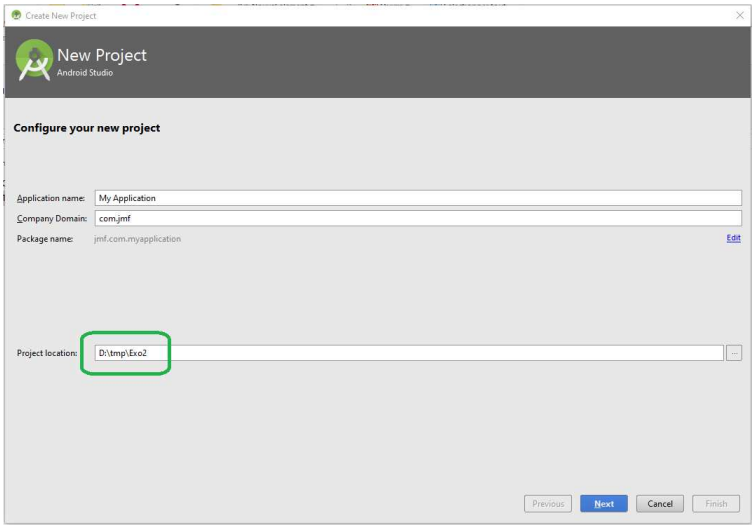
On veut construire l'IHM suivante :



Les principales étapes de développement sont décrites ci-dessous.
1°) Lancer Android Studio et créer un nouveau projet :

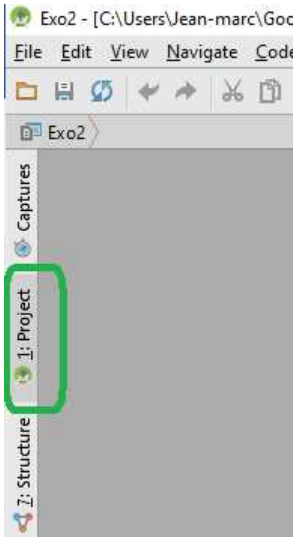


Préciser, dans la fenêtre suivante, un nom de nouveau répertoire :



Cliquez le bouton Next.

Dans la fenêtre suivante, cliquez sur l'onglet 1: Project



Vous allez suivre les étapes du cours pour créer la `ListView` avec ses 2 items.

2°) Créer le fichier `res/layout/list_item.xml` représentant l'IHM des items de la liste (voir le cours).

3°) Dans le fichier `res/layout/activity_main.xml` représentant le premier écran de lancement de l'application, indiquer que vous devez avoir une `ListView`. Ce fichier `res/layout/activity_main.xml` doit donc avoir pour contenu :

---
fichier `res/layout/activity_main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
	android:orientation="vertical" android:layout_width="fill_parent"
	android:layout_height="fill_parent">

	<ListView android:layout_height="wrap_content"
		android:layout_width="match_parent" android:id="@+id/listMenu">
	</ListView>
</LinearLayout>
```

Remarque : sous Android Studio, pour indenter correctement (un fichier XML, ou un code Java), utiliser CTRL+ALT+l. Voir aussi le menu Code.

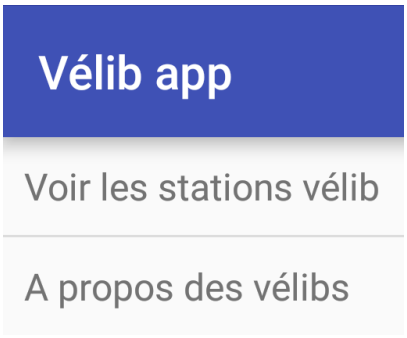
Par quel identifiant dans votre code Java, pourrez-vous manipuler cette `ListView` ?

4°) Ecrire, dans l'`Activity` de lancement de votre application, le code Java qui récupère la `ListView` et qui l'alimente des deux items "Voir les stations vélib" et "A propos des vélibs". On repèrera ces deux items par les identifiants `seeAllStations` et `aboutUs`.
Donne le titre "Vélib app" à votre application.

Remarque : sous Android Studio, pour importer les bonnes classes et interfaces Java, utiliser la combinaison de touche ALT+Return

5°) Lancer l'exécution de votre application.

Vous devez obtenir :



Seconde étape : Prendre en compte les interactions utilisateurs

1°) On veut désormais que :
- lorsque l'utilisateur choisit l'item "Voir les stations vélib", un `Toast` affiche "En cours de développement"

- lorsque l'utilisateur choisit l'item "A propos des vélibs", une nouvelle activité contenant un `TextView` qui affiche des renseignements sur les Vélibs est lancée.

Ecrire le code nécessaire au traitement des événements. Vérifier que la nouvelle activity est bien déclarée dans l'`AndroidManifest.xml`.

Troisième étape : Récupérer des informations réseau

On s'intéresse désormais au résultat à obtenir lorsqu'on sélectionne le premier item. La liste des stations Vélib devra être affichée :

Pour cela, on contactera le site `http://www.velib.paris.fr/service/carto`

1°) Connecter vous à ce site. Quel type de données retourne ce site ?

On va donc faire un parsing (une analyse) de ces données XML. Le résultat sera donc une liste de Stations Vélib.

La classe `StationVelib`

2°) Construire une classe `StationVelib` qui modélise une station vélib. Ce pourra être :

```
public class StationVelib {
	private String nom;
	private String numero;

	public String getNom() {
		return nom;
	}
	public void setNom(String nom) {
		this.nom = nom;
	}
	public String getNumero() {
		return numero;
	}
	public void setNumero(String numero) {
		this.numero = numero;
	}
}
```

Le but sera d'obtenir un `ArrayList<StationVelib>` après l'analyse XML. On a donc besoin de lancer la construction d'une `StationVelib` après lecture de certaines balises XML et d'enregistrer cette `StationVelib` après construction, dans un `ArrayList<StationVelib>`.

La classe `StationsParser`

3°) Définir la classe `StationsParser` qui va être un "handler XML" pour analyser un flux XML. Un "handler XML" est un traitant de fichier XML, objet d'une sous classe de la classe `org.xml.sax.helpers.DefaultHandler`. Il décrit les actions à effectuer lorsqu'on traite le fichier XML. Essentiellement ici, il construit un `ArrayList<StationVelib>`, qui sera ensuite passée à l'adapteur associé à la `ListView` affichant toutes les stations.
Cette classe commence par :

```
public class StationsParser extends DefaultHandler {
	private ArrayList<StationVelib> arrList;
	private XMLReader xr;

	public StationsParser() throws Exception {
		URL urlBase = new URL("http://www.velib.paris.fr/service/carto");
		InputSource is = new InputSource(urlBase.openStream());

		// traitement du parsing XML : factory XML, ...
		SAXParserFactory spf = SAXParserFactory.newInstance();
		SAXParser sp = spf.newSAXParser();
		xr = sp.getXMLReader();
		xr.setContentHandler(this);

		xr.parse(is);
	}
```

Par la suite des actions sont déclenchées sous certains événements de traitement du fichier XML. Ce sont, par exemple, les lectures de balises d'ouverture (`startElement()`), lecture de balise de fermeture (`endElement()`), etc.

4°) Compléter cette classe par :

```
	public ArrayList<StationVelib> getArrList() {
		return arrList;
	}

	@Override
	public void endDocument() throws SAXException {
		super.endDocument();
	}

	@Override
	public void startDocument() throws SAXException {
		super.startDocument();
		arrList = new ArrayList<StationVelib>();
	}

	@Override
	public void startElement(String uri, String localName, String qName,
Attributes attributes) throws SAXException {
		super.startElement(uri, localName, qName, attributes);
		if (localName.equals("marker")) {
			StationVelib s = new StationVelib();

			s.setNom(attributes.getValue("name"));
			s.setNumero(attributes.getValue("number"));

			arrList.add(s);
		}
	}
}
```

Les méthodes `startElement()`, `startDocument()`, `endDocument()` étant lancées lors du traitement du fichier XML, on comprend qu'à la fin de la lecture (et du traitement du flux XML), on obtienne dans `arrList`, un ensemble d'objets de la classe `StationVelib` (c'est à dire un `ArrayList<StationVelib>`).

La classe `StationsAdapter`

On va écrire l'adapteur de la liste des stations vélib. En effet, ce sera une `ListView` qui va afficher la liste des stations Velib. Il lui faut donc un adaptateur.
5°) Cet adaptateur reçoit ses données de l'`ArrayList<StationVelib>` du `StationsParser` (cf. question précédente) d'où le début du code de cette classe :

```
public class StationsAdapter extends ArrayAdapter<StationVelib> {
	private ArrayList<StationVelib> arrList;

	public StationsAdapter(Context ctx, ArrayList<StationVelib> arrList) {
		super(ctx, R.layout.ligne_station, arrList);
		this.arrList = arrList;
```

```
    }
```

6°) Essentiellement, il doit fabriquer la `View` qui sera insérée à la position `position` de la `ListView` associée, c'est à dire un item de la `ListView`. C'est évidemment ce qui correspond à l'objet `arrList` à son indice `position`. De plus ce qui est fabriqué est un `TextView`. On décrit tout d'abord les caractéristiques de ce `TextView` dans le fichier `res/layout/ligne_station.xml`. Voici ce fichier :

---

fichier `res/layout/ligne_station.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/item1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="TextView"
        android:textAppearance="?android:attr/textAppearanceLarge" >
    </TextView>

</LinearLayout>
```

7°) Compléter la classe `StationsAdapter` par le code ci dessous :

```
    /**
     * La méthode ci dessous construit l'item de la ListView,
     * qui sera à l'indice position
     * et retourne cet item
     */
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        // Un LayoutInflater est un constructeur de View à partir d'une source XML
        // On l'obtient (et on ne le construit pas par new !)
        // par exemple par LayoutInflater.from(getContext());
        LayoutInflater inflater = LayoutInflater.from(getContext());
        View row = inflater.inflate(R.layout.ligne_station, null);
        // le second argument est l'éventuel parent de la vue récupérée

        TextView label = (TextView) row.findViewById(R.id.item1);
        label.setText(arrList.get(position).getNom());

        return row;
    }
}
```

Ce peut paraître compliqué, mais les commentaires donnent les explications.

La classe `ListingDesStationsActivity`

Il faut désormais écrire la dernière classe qui affichera la liste des stations. C'est la classe `ListingDesStationsActivity` (qui est donc une `Activity`).

8°) Ecrire le fichier `filtre_stations.xml` IHM de cette activity.

---

fichier `res/layout/filtre_stations.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <ListView
        android:id="@+id/listViewFiltreStations"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
    </ListView>

</LinearLayout>
```

9°) Ecrire la classe `ListingDesStationsActivity` qui est une `Activity`. Cette Activity va présenter une fenêtre contenant la liste des stations vélibs, les données de cette liste ayant été fabriquées par les questions précédentes. Cette classe commence par :

```
public class ListingDesStationsActivity extends AppCompatActivity {

	private ListView listing;
	private StationsParser sp;
	private ProgressDialog progress;
	private StationsAdapter leStationsAdapter;

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.filtre_stations);

		listing = (ListView) findViewById(R.id.listViewFiltreStations);

	// a compléter par la définition de la classe interne à cette méthode
	// class ChargementDesStationsTache extends AsyncTask<Void, Void, Void>
	// ainsi que l'instanciation et le lancement de la méthode execute()
	// sur cette instance

	}
}
```

10°) Déclarer, si ce n'est déjà fait, cette activity dans l'`AndroidManifest.xml`.

11°) Indiquer aussi, dans ce fichier `AndroidManifest.xml`, que l'application doit utiliser internet par la ligne :

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

comme fille de l'élément `manifest`.

12°) Ecrire (ou récupérer) le code qui, à l'aide de la classe `AsyncTask`, permet de construire l'adapteur de la `ListView`, après avoir fait le parsing XML d'une donnée lointaine ;-)

13°) Modifier l'activité initiale (= principale) de votre application de sorte à faire afficher les stations vélib lorsqu'on clique sur le premier item de la `ListView` de cette activité initiale.