TP Android

Exercice 1 : Construction d'une IHM avec des composants Android

Le but de cet exercice est de construire une interface humain-machine (IHM) présentant plusieurs composants graphiques Android (`View`). Cette IHM permet de faire une authentification en demandant un couple (nom de login, un mot de passe). Après appui sur le bouton Connecter, un `Toast` indique si le couple donné convient ou pas.

1°) Indiquez les composants graphiques Android qui interviennent dans cette IHM.
Remarque : le troisième (si, si !) composant n'affiche pas l'écho des caractères.
2°) Construire cette IHM avec l'environnement Eclipse et le plug-in Android. On demande d'écrire le moins de code Java possible. Les chaînes de caractères seront plutôt définies dans le fichier `strings.xml`.

Réaction aux interactions utilisateur

3°) Ecrire le code Java qui, lorsque le bouton Connecter est actionné par l'utilisateur, une vérification d'authentification est faite et le `Toast` affiche le résultat (bon couple login, mot de passe ou pas).
4°) Ecrire le code Java qui, lorsque le bouton Connecter est actionné par l'utilisateur, si le couple (login, mot de passe) est correct, une nouvelle activité est affichée indiquant une bonne connexion.

Exercice 2 : `ListView`, Velibs and Cie !

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`

Les principales étapes de développement sont décrites ci dessous.
1°) Lancer Eclipse et choisir un répertoire de travail (workspace) par exemple ....\TP.
Créer un nouveau projet Android (File | New | Project | Android | Android Project) en utilisant l'API 8 Android APIs 2.2.

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

---
fichier `res/layout/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>
```

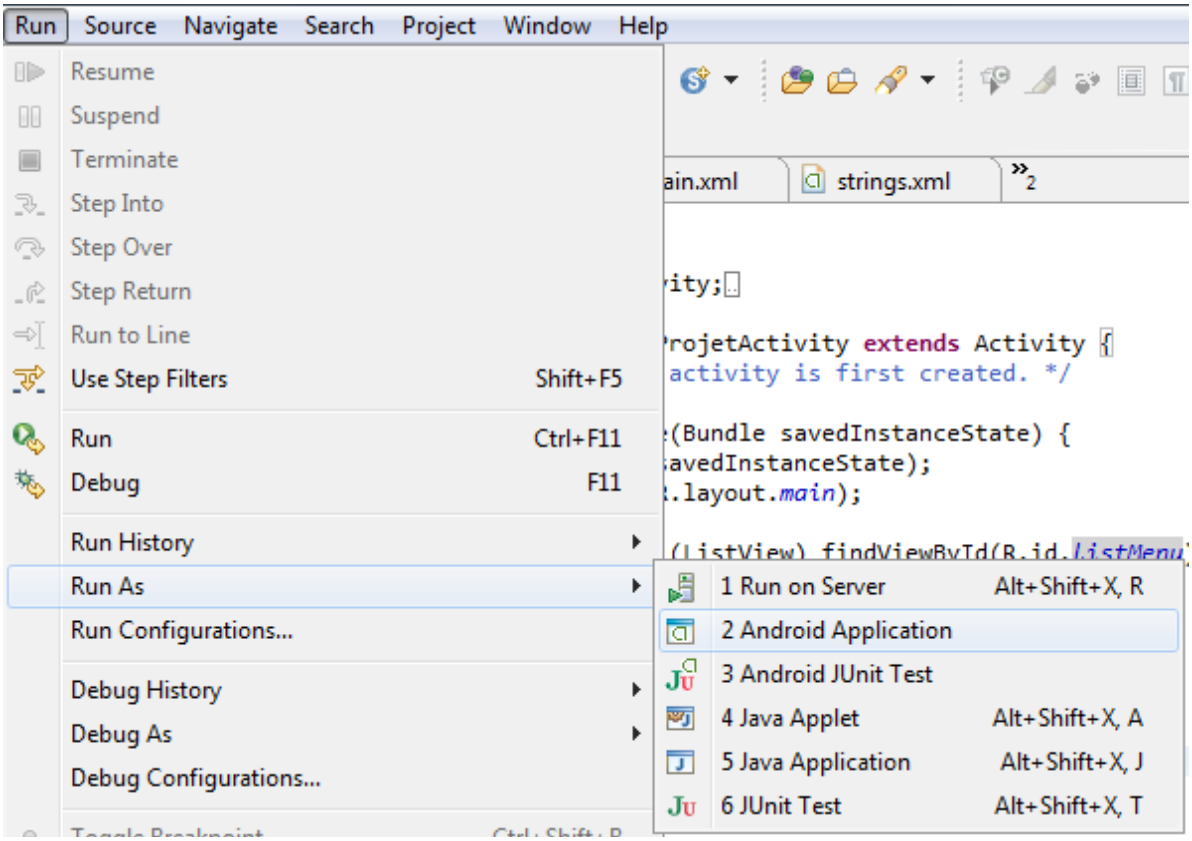
Remarque : sous Eclipse pour indenter correctement (un fichier XML), utiliser CTRL+a (= tout sélectionner) suivi de CTRL+i (indenter).

Par quel identifiant dans votre code Java, pourrez vous manipuler ce `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 Vélib". On repèrera ces deux items par les identifiants `seeAllStations` et `aboutUs`.

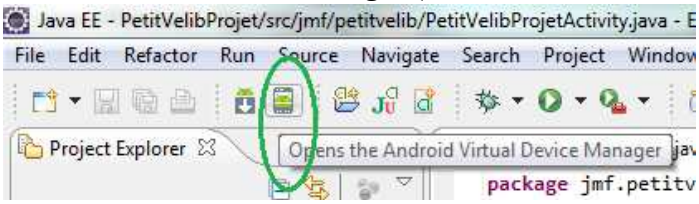
Remarque : sous Eclipse, pour importer les bonnes classes et interfaces Java, utiliser la combinaison de touche SHIFT+CTRL+O

5°) Lancer l'exécution de votre application en sélectionnant votre projet : c'est important car sélectionner autre chose amène parfois des problèmes !
Puis menu Run | Run As | 2 Android Application

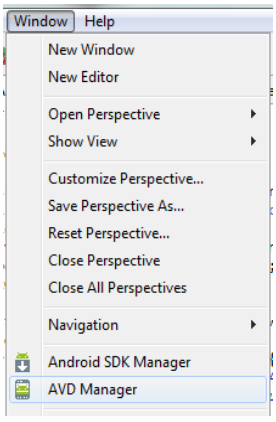


Remarque sur le AVD

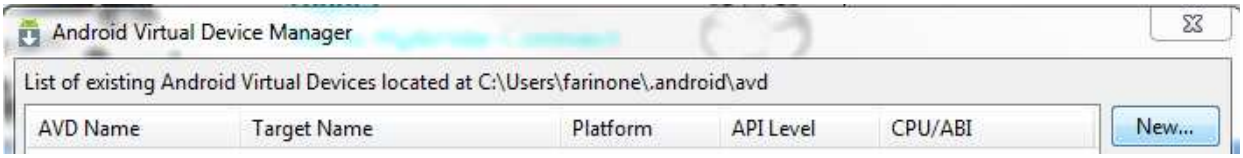
Si vous avez besoin d'un Android Virtual Device (= AVD = bref un émulateur de téléphone ou tablette), en créez un.
Pour cela lancer l'Android Virtual Manager (bouton sous la barre de menu)



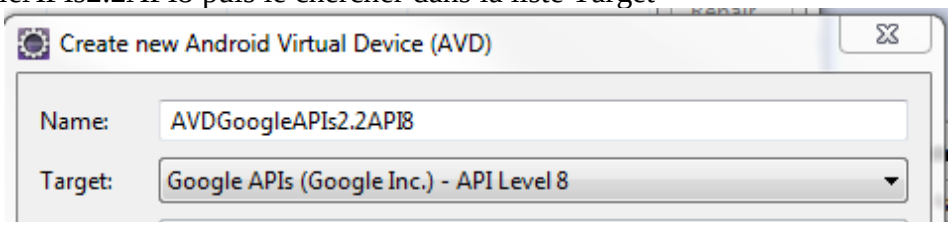
ou Window | AVD Manager



Dans la fenêtre Android Virtual Device Manager cliquer New...



Dans la fenêtre Create new Android Virtual Device (AVD), appelez le AVDGoogleAPIs2.2API8 puis le chercher dans la liste Target



Cliquez Create AVD (bouton en bas de cette fenêtre).

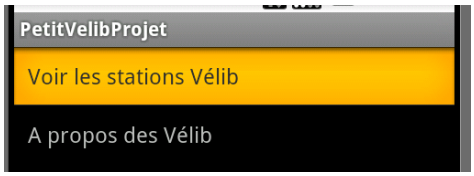
Cet AVD apparaît alors dans la liste des AVD.

Lancer votre application. Vous devez obtenir :

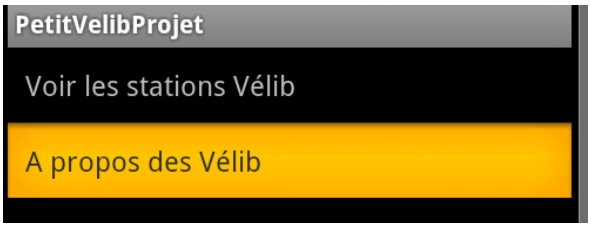
Seconde étape : Prendre en compte les interactions utilisateurs

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 Vélib", une nouvelle activité contenant un `TextView` qui affiche des renseignements sur les Vélibs est lancée.

Bref



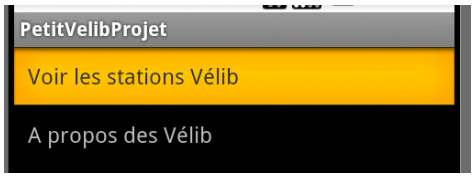
et



Ecrire le code nécessaire au traitement des événements. Ne pas oublier de déclarer la nouvelle activity 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 Velib 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 donc 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 balise 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'adaptateur associé à la `ListView` affichant toutes les stations.

Cette classe commence par :

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

	public StationsParser(Context ctx) throws Exception {
		this.ctx = ctx;
		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;
	}

	public void setArrList(ArrayList<StationVelib> arrList) {
		this.arrList = 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, Activity activity,
ArrayList<StationVelib> arrList) {
            super(ctx, 1, 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"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true" >
    </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 donc évidemment par :

```
public class ListingDesStationsActivity extends Activity {

	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<String, String, String>
	// ainsi que l'instanciation et le lancement de la méthode execute()
	// sur cette instance

	}
}
```

10°) Déclarer 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 le code, à l'aide de la classe `AsyncTask`, qui permet de visionner la liste des stations lorsque l'utilisateur sélectionne le premier item (voir le cours).