ANDROID - TP3

Source, pdf et corrigé de ce TP :
http://tiny.cc/techmob

L'objectif de ce TP est de réaliser une application utilisant les données publiques des transports en commun de rennes afin d'afficher le temps d'attente des bus.
Ce TP fera appel aux notions suivantes :

- Réseau / requêtes HTTP (webservice)
- Parsing JSON
- Multithreading
- Localisation GPS

Il est fortement conseillé d'avoir le cours à portée de main et à ne pas hésiter à se référer à la documentation officielle http://d.android.com ainsi qu'à la multitude de tutoriaux disponibles.

1 Présentation des données STAR

1.1 Contexte

Rennes a fait partie des premières villes à libérer ses données publiques concernant les transports en commun. Ainsi, la STAR met à disposition des développeurs une grande partie des données concernant son réseau (stations, horaires, prochains passages ...).
Les données sont disponibles pour tous, gratuitement.

1.2 Contenu des données

Les données disponibles sont réparties en deux catégories : les données statiques et les données temps réel.

1.2.1 Les données statiques

Les données statiques contiennent les informations sur le réseau STAR : les stations et leurs emplacements, les lignes et leurs horaires, les équipements disponibles en station ...
Ces données n'évoluent que quelques fois par an.
La STAR publie ses données au standard GTFS (General Transit Feed Specification). Concretement, il s'agit simplement de plusieurs fichiers csv.
La documentation sur ces données et la dernière version des données sont disponibles ici : http://data.keolis-rennes.com/fr/les-donnees/donnees-telechargeables.html)

1.2.2 Les données temps réel

La STAR met aussi à disposition des données temps réel. On y trouve le nombre de vélos et de places disponibles à une station vélostar, les heures de prochain passage des bus et métro ...
La liste des données disponibles est disponible ici : http://data.keolis-rennes.com/fr/les-donnees/donnees-et-api.html
L'accès à ces données se fait via un webservice.
Dans un premier temps, nous ne nous occuperons que des données statiques.

2 Hello World !

— Créer un nouveau projet
— Télécharger les dernières données statiques disponibles sur le site open-data de keolis
— A l'intérieur du zip téléchargé, ouvrir les fichiers routes.txt et stops.txt. Que contiennent t-ils ?
— Inclure ces deux fichiers dans les ressources de l'application
— Créer un objet métier, Route, dont les attributs reflètent le contenu du fichier routes (le fichier stops sera exploité plus tard)

On souhaite maintenant écrire le code permettant d'instancier les objets Route à partir des informations présentes dans le fichier.
L'analyse (parsing) d'un fichier CSV est relativement facile mais peut se réveler fastidieuse. Pour éviter de réécrire à chaque application le même code, on peut utiliser une des multiples bibliothèques disponibles : commons-csv de la fondation apache.
La documentation est disponible sur ce site : https://commons.apache.org/proper/commons-csv/

— Ajouter cette bibliothèque à l'application en utilisant la dépendance gradle : 'org.apache.commons :commons-csv :1.2'

Pour itérer sur chaque ligne du fichier, on pourra s'inspirer du code suivant :

```
1 InputStream stream = context.getResources().openRawResource(R.raw.routes);
2 Iterator<CSVRecord> iterateur = new CSVParser(new InputStreamReader(stream), CSVFormat.DEFAULT)
      .iterator();
3 CSVRecord enregistrementCourant = null;
4 while (iterateur.hasNext()) {
5     enregistrementCourant = iterateur.next();
6 }
7 stream.close();
```

— Créer une classe DAO, RouteDAO, qui instancie une liste de Route à partir du contenu du fichier
— Afficher un toast, sur l'activity principale, indiquant le nombre de lignes présentes sur le réseau STAR

3 Un peu d'interface

— Ajouter une ListView prenant toute la taille de l'écran à l'activity principale
— Afficher, dans cette ListView, la liste des lignes (au moins le nom)
— Créer une nouvelle activity qui contiendra le détail des prochains passages des bus de cette ligne (on implémentera son contenu plus tard)
— Lors d'un click sur un élément de la liste des lignes, lancer l'activity en passant en paramètre l'identifiant de la route choisie

4 Utilisation des données temps réel

On va maintenant utiliser les données temps réel pour récupérer les horaires des prochains passages pour la ligne choisie.

4.1 Principe d'un webservice

Un webservice est, comme son nom l'indique, un service accessible sur le web.
Techniquement, il s'agit d'un programme tournant sur un serveur et déstiné à diffuser des données brutes (contrairement au HTML, il n'y a pas d'informations sur la façon d'afficher ces données) via le protocole HTTP.
Toutes les requêtes sont testables directement en tapant l'URL dans votre navigateur préféré.

4.2 Appel au webservice

Un webservice est toujours constitué d'une adresse de base suivi d'un paramètre indiquant le type d'information demandée et éventuellement de paramètres précisant cette information.

Pour le webservice STAR, l'URL de base est la suivante : http://data.keolis-rennes.com/json/
Il y a ensuite 3 paramètres obligatoires :

- cmd : la commande demandée. C'est à dire, l'information que l'on cherche. Par exemple, pour obtenir le prochain départ d'un bus, on a $cmd = getbusnextdepartures$
- version : la version du webservice que l'on utilise. Se référer à la documentation pour savoir quelle version utiliser pour chaque commande.
- key : la clé. Afin de savoir qui utilise le webservice, la STAR fournit aux développeurs des clés correspondant à chaque application déclarée. Pour obtenir une clé, il faut s'inscrire sur le site. Pour tout ce TP, vous pouvez utiliser la clé suivante : 1RJLZ38TUFZSWTW

Les autres paramètres (subparams) doivent être entourés par param[].
Par exemple : param[direction]=1

Ainsi, un appel au webservice pour récupérer l'état du métro (panne, interruption, fermeture ...) peut se faire directement à l'URL suivante :
http://data.keolis-rennes.com/json/?cmd=getmetrostatus&version=2.2&key=1RJLZ38TUFZSWTW

4.3 Résultat du webservice

Le webservice renvoie les données au format JSON ou au format XML (changer JSON en XML dans l'URL).
Pour simplifier, dans ce TP, on ne manipulera que les données JSON.

4.4 Visualiser le JSON

Même si le format JSON est relativement clair et simple, il reste assez difficile d'obtenir une vision globale des données présentes dans un résultat JSON d'un seul coup d'oeil.
Des outils de visualisation JSON existent et sont très utiles dès lors que l'on manipule des fichiers JSON.
Vous pouvez par exemple utiliser http://jsonviewer.stack.hu/ ou la multitude d'extensions chrome / firefox disponibles.

4.5 Intégration dans l'application

On souhaite afficher, dans la nouvelle Activity, la liste des arrêts de la ligne choisie.

- Dans la nouvelle activity, récupérer le numéro de la ligne concernée. Si ce numéro n'est pas défini, afficher un toast et fermer l'activity.
- Ajouter la permission (uses-permission) INTERNET au manifest
- Dans le onCreate de votre Activity, ajouter la ligne suivante

```
1 StrictMode.setThreadPolicy(new ThreadPolicy.Builder().permitAll().build());
```

  Cette ligne sera expliquée plus tard dans le TP
- Construire l'adresse qui sera appelée. La commande utilisée est "getbusnextdepartures" en mode line.
- Tester l'adresse dans un navigateur.
- Utiliser HttpURLConnection pour faire une requête HTTP vers le webservice

Pour lire le contenu d'un InputStream (flux entrant), on peut s'inspirer du code suivant :

```
1  public String readStream(InputStream inputStream) throws IOException {
2      BufferedReader reader = new BufferedReader(new
3      InputStreamReader(inputStream));
4      String ligne = null;
5      String contenu = "";
6      while ((ligne = reader.readLine()) != null) {
7          contenu += ligne;
8      }
9      return contenu;
10 }
```

- Afficher dans la log le contenu de la réponse.

5 Parsing des données

5.1 Lire le JSON

— Créer un objet JSONObject à partir des données récupérées précedemment
— En utilisant une combinaison des méthodes getJSONObject et getJSONArray, récupérer le tableau des arrêts déservis par la ligne
— Afficher, dans un Toast, le nombre d'arrêts

5.2 Instancier les objets correspondants

— Créer des classes Java "Arret" et "Departure"
— Donner à ces classes les mêmes attributs que les données disponibles dans la réponse JSON
— Instancier, à partir du JSONObject, une liste (ou un tableau) d'objet Arret

6 Il est temps de faire les choses proprement

1. Quel point, vu dans les bonus, a été complétement ignoré depuis le début de ce TP ?

— Remplacer la ligne StrictMode ajoutée au début du TP par celle ci

```
1  StrictMode.setThreadPolicy(new ThreadPolicy.Builder().detectNetwork().penaltyDeathOnNetwork
       ().build());
```

— Relancer l'application, pleurer
— StrictMode est un outil d'aide au développement permettant de détecter les opérations longues susceptibles de bloquer un Thread.
— Le code ci-dessus indique à StrictMode de faire crasher l'application dès qu'un appel réseau est fait dans le Thread actif (ici, le Thread principal : main / UI). Depuis le niveau d'API 11, c'est le comportement par défaut.
La ligne ajoutée en début de TP disait à StrictMode de tout laisser passer.
— Déplacer le bloc de code correspondant à la requête HTTP dans un Thread séparé, sourire.

7 Afficher les données

7.1 Afficher la liste des arrêts

— Ajouter une ListView au layout
— Il nous faut maintenant un adapter : créer une nouvelle classe "ArretAdapter" qui extends BaseAdapter
Hériter de BaseAdapter implique d'écrire 4 méthodes
— Avant d'implémenter ces méthodes, construire un constructeur prenant un Context et une liste d'Arret et stocker ces valeurs dans des attributs de classe

getCount : renvoyer le nombre d'éléments de la liste

getItem : renvoyer le n-ième élément de la liste

getItemId : renvoyer la position (la contrainte sur ItemId est que chaque id soit unique)

getView : ci-dessous un squelette d'une méthode getView. Il faut au préalable créer un layout "layoutarret" représentant un élement de la liste.

```
1  public View getView(int position, View convertView, ViewGroup parent) {
2      View view;
3
4      if (convertView == null) {
5          view = ((LayoutInflater)
6          context.getSystemService(Context.LAYOUT_INFLATER_SERVICE)).inflate(R.layout.layoutarret
7          ,parent, false);
8      }
9      else {
10         view = convertView;
```

```
11      }
12      Arret arret = (Arret) getItem(position);
13
14      TextView idArret = (TextView) view.findViewById(R.id.arret);
15      idArret.setText(arret.getIdArret());
16
17      return view;
18  }
```

— Utiliser cet ArretAdapter pour afficher les données venant du webservice (identifiant de l'arrêt, heure de prochain passage)

Optionnel : les informations dont le nom des arrêts sont disponibles dans le fichier stops.txt.

8 Station de métro la plus proche

Le webservice donne accès à bien plus d'informations, on peut par exemple trouver toutes les stations de métro ou de vélostar à proximité à partir de la latitude et la longitude.

Pendant les tests, il est possible de truquer la position du GPS de l'émulateur ou d'un appareil à partir de la vue DDMS.
Une multitude de sites permettent de récupérer la latitude / longitude d'un endroit dont par exemple google maps.

— Utiliser la commande *getmetrostations* pour récupérer les arrêts de métro les plus proches d'une position de votre choix
— Ajouter la permission ACCESS_FINE_LOCATION au manifest

En s'inspirant du code suivant, récupérer la position GPS et chercher les stations de métro les plus proches.

```
1  LocationManager locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
2  LocationListener locationListener = new LocationListener() {
3
4          @Override
5          public void onStatusChanged(String provider, int status, Bundle extras) {
6
7          }
8
9          @Override
10         public void onProviderEnabled(String provider) {
11
12         }
13
14         @Override
15         public void onProviderDisabled(String provider) {
16
17         }
18
19         @Override
20         public void onLocationChanged(Location location) {
21             //La localisation a chang
22         }
23     };
24 locationManager.requestLocationUpdates(
25 LocationManager.GPS_PROVIDER, 5000, 10, locationListener);
```

Une icône dans la barre d'état indique quand le service de localisation est utilisé. Afin de réduire au minimum la consommation de batterie, il est important de libérer les ressources quand elles ne sont plus utilisées.

— En utilisant le cycle de vie d'une activity et la méthode removeUpdates, libérer les ressources lors de la fermeture de l'activity

9 Pour aller plus loin

— Les chargements de données pouvant être longs, rajouter des popups ou des icônes indiquant que l'application charge des données. Eventuellement, afficher l'avancement du chargement.
— Plutôt que de lire à chaque fois les fichiers de données statiques, il est préférable, au lancement de l'application, de créer une base de données avec le contenu des fichiers routes et stops. Créer cette base et en profiter pour afficher le nom des arrêts dans l'activity des horaires.