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
- Stockage de données SQLite
- Multithreading
- ListView

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 vélo star, 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 (EmptyActivity)
- Télécharger les dernières données statiques disponibles sur le site open-data de keolis
- A l'intérieur du zip téléchargé, ouvrir le fichier routes.txt. Que contient t-il ?
- Inclure ce fichier dans les ressources de l'application
- Créer un objet métier, Route, dont les attributs reflètent le contenu du fichier routes

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 bibliothèque disponibles : commons-csv de la fondation apache.

- Télécharger cette bibliothèque à partir du site officiel : X
- Ajouter cette bibliothèque à l'application

Le code suivant permet d'itérer sur chaque ligne du fichier :

```
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)
- Optionnel : personnaliser l'affichage en utilisant par exemple le code couleur de chaque ligne fourni dans les données
- Créer une nouvelle activity qui contiendra le détails des prochains des bus de cette ligne
- 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 et 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

Ainsi, un appel au webservice 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/

4.5 Intégration dans l'application

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

- 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 appellée. 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 :

```
public String readStream(InputStream inputStream) throws IOException {
    BufferedReader reader = new BufferedReader(new
    InputStreamReader(inputStream));
    String ligne = null;
    String contenu = "";
    while ((ligne = reader.readLine()) != null) {
        contenu += ligne;
    }
    return contenu;
}
```

- 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 la méthode 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 une classe Java Arret
- Donner à cette classe 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 Stockage des données

L'objectif est de stocker les données chargées pour éviter de devoir les retélécharger à chaque fois (gain de temps, accès hors connexion, économie de batterie ...)

1. Quelle solution de stockage vous parait la plus adaptée ?

- En utilisant SQLiteOpenHelper, implémenter le nécessaire pour créer une base de données SQLite
- Dans onCreate, éxecuter une requête de création de table pour créer une table Arret avec tous les attributs de la classe Arret
- Exécuter une insertion avec des données fictives
- Exécuter une "query" pour récupérer les données stockées et instancier les objets Arret correspondant aux données, les afficher dans la log
- Insérer les données récupérées via le webservice

2. Imaginons qu'un jour les données renvoyées par le webservice changent.
   Comment modifier la base de données SQLite si elle a déjà été crée sur l'appareil ?

7 Il est temps de faire les choses proprement

3. 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

```
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 (utiliser new Thread(Runnable)), sourire.

8 Afficher les données

8.1 Afficher la liste des arrêts

4. Quel view / viewgroup vous parait le plus adapté pour afficher les cours ?

- Ajouter cet élément au layout en lui faisant prendre toute la place en largeur / hauteur
- 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 position (il suffit que l'id de chaque item 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.

```
public View getView(int position, View convertView, ViewGroup parent) {
        View view;

        if (convertView == null) {
            view = ((LayoutInflater)
            context.getSystemService(Context.LAYOUT_INFLATER_SERVICE)).inflate(R.layout.
                layoutarret,
            parent, false);
        }
        else {
            view = convertView;
        }
        Arret arret = (Arret) getItem(position);

        TextView idArret = (TextView) view.findViewById(R.id.arret);
        idArret.setText(arret.getIdArret());

        return view;
    }
```

- Instancier un ArretAdapter et l'affecter à la ListView via la méthode setAdapter
- Il est possible d'affiner le layout de chaque item en rajoutant des champs et d'autres informations. Rajouter la date de prochain passage.

9 Pour aller plus loin