TP3 : Localiser ses amis…

Objectifs :

- utiliser le capteur GPS,
- utiliser le fournisseur de contenu « **Contacts** »,
- enregistrer son nom de paquetage sur Google pour obtenir une clé d'usage **Google MAP**,
- utiliser la librairie **Google Map Api V2** pour Android,
- positionner deux punaises sur une carte, tracer la route, calculer le temps et la distance,
- utilisation des concepts déjà traités (changement d'activité avec paramètres, tâche de fond, persistance…).

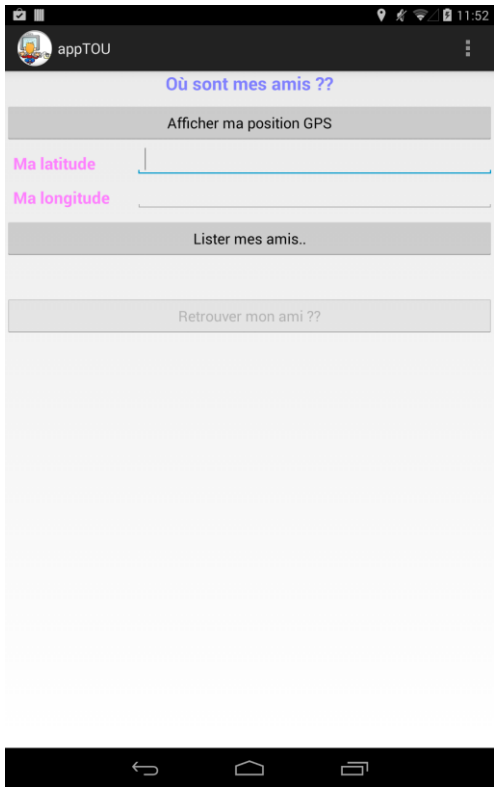
L'application devra permettre :

- ✓ de se géo-localiser via la fonction GPS et d'afficher sa latitude et sa longitude,
- ✓ d'afficher sa liste de contacts afin de sélectionner un « ami » (ou une amie…) que l'on désire rejoindre (pas obligatoirement en scooter !!),
- ✓ d'afficher des punaises pour sa position et celle de son ami€ sur une carte **Google Map**,
- ✓ de tracer le parcours entre les deux punaises pour un trajet pédestre ou routier,
- ✓ d'afficher la distance et la durée de ce parcours.

Mise en place de l'activité principale

1. Créer un nouveau projet **Android Studio Project** nommé **appTOU**, l'application portera le nom du projet et le paquetage sera nommé **net.votrenom.votreprénom.appTOU** :
   - ✓ utiliser **l'API 18** comme sdk mini et choisir une « **Blank Activity** » comme activité de départ du projet,
   - ✓ vous pouvez utiliser l'icône fournie dans le dossier **Android_Resources** pour « agrémenter » votre application (voir TP1 pour la méthode),
   - ✓ conserver les noms par défaut de l'activité (**MainActivity.java**), du layout (**activity_main.xml**) et du menu et donner comme titre de l'activité **AppTOU**.

Voici le layout que vous devez obtenir :



Vous aurez reconnu en partant du haut :

- ✓ un contrôle de type « **TextView** » pour le titre de la vue,
- ✓ un contrôle de type « **Button** » qui servira à démarrer/arrêter la fonction GPS,
- ✓ 2 couples « **TextView** »/« **EditText** » pour éditer les informations GPS, latitude et longitude,
- ✓ un contrôle de type « **Button** » qui servira à lister et choisir un ami parmi le carnet d'adresses,
- ✓ deux contrôle de type « **TextView** » (vides et côte à côte) pour l'affichage du nom et de l'adresse de l'ami(e) à rejoindre,
- ✓ un contrôle de type « **Button** » qui servira à basculer vers l'activité d'affichage de la carte **Google Map**.

2. A vous de jouer, faîtes en sorte d'obtenir la vue présentée ci-dessus.

Gestion de la fonction GPS, codage d'une classe LocationListener

3. Ajouter à votre classe **MainActivity** les données membres privées suivantes pour la gestion du GPS :
   - ✓ **btnStartGPS** de type **Button**, objet bouton pour le démarrage et l'arrêt du GPS,
   - ✓ **edtLat** de type **EditText**, objet affichage de la latitude,
   - ✓ **edtLon** de type **EditText**, objet affichage de la longitude,
   - ✓ **gps** de type **LocationManager**, pour l'utilisation du GPS.

Avant d'envisager le démarrage du GPS vous allez devoir coder la classe « **Listener** » pour la capture des événements GPS.

4. Ajouter une nouvelle classe nommée **myGPSListener** au projet.
5. Faire en sorte qu'elle implémente l'interface **LocationListener** et coder les méthodes suivantes :
   - ✓ Le constructeur qui recevra l'identificateur des contrôles affichant la latitude et la longitude ainsi que la référence à l'activité en charge de ces contrôles. Il recopiera ces données dans des attributs.
   - ✓ La méthode **onProviderDisable()** qui signalera par un « Toast » la désactivation du GPS.
   - ✓ La méthode **onProviderEnable()** qui signalera par un « Toast » l'activation du GPS.
   - ✓ La méthode **onStatusChanged()** qui signalera par un Toast la perte et la récupération du signal GPS.
   - ✓ La méthode **onLocationChanged()** qui mettra à jour la latitude et la longitude dans les contrôles prévus à cet effet.
6. Ajouter à la classe **MainActivity** une donnée membre privée **gpsListener** de type **myGPSlistener**.
7. Compléter la méthode surchargée **Activity**.**onCreate()** afin :
   - ✓ d'instancier les objets **btnStartGPS**, **edtLat** et **edtLon**,
   - ✓ d'instancier l'objet **myGPSListener** associé à l'activité et aux contrôles d'affichage de la latitude et de la longitude et gps,
   - ✓ d'instancier l'objet **gps**.
   - ✓ d'associer un écouteur sur le bouton « Afficher ma position GPS » (objet **btnStartGPS**) pour capturer l'événement **onClick** et déclencher la méthode associée.

8. Coder la méthode **onClick()** afin qu'elle démarre ou arrête le GPS en modifiant le texte du bouton en conséquence. On vérifiera que la fonction GPS est validée avant de tenter de le démarrer, si ce n'est pas le cas on pourra le signaler à l'aide d'une « **AlertDialog** ».
9. Vérifier le bon fonctionnement de l'ensemble, soyez patient, la fonction GPS peut mettre un certain temps à se synchroniser avant de récupérer une position.

Utilisation du fournisseur de contenu « Contacts »

Le clic sur le bouton « Lister mes amis » devra afficher la liste des contacts afin de pouvoir choisir le contact que l'on souhaite rejoindre. L'utilisateur du fournisseur de contenu des contacts nécessite la permission Android **READ_CONTACTS**.

10. Ajouter cette permission au manifeste.
11. Ajouter à votre classe **MainActivity** les données membres privées suivantes pour la gestion des contacts :
    - ✓ **btnFindFriends** de type **Button**, objet bouton pour le choix d'un(e) ami(e),
    - ✓ **CODE_REQUETE_AMIS**, attribut **final static**, code de requête pour l'intention d'interrogation du fournisseur de contenu des contacts.
12. Compléter la méthode surchargée **Activity**.**onCreate()** afin :
    - ✓ d'instancier l'objet **btnFindFriends**,
    - ✓ d'associer un écouteur sur le bouton « Lister mes amis » pour capturer l'événement **onClick** et déclencher la méthode associée.
13. Coder la méthode **onClick()** afin qu'elle crée une intention d'accès au fournisseur de contenu des contacts en vue de sélectionner un contact pour récupérer son nom et son adresse postale.
14. Surcharger et coder la méthode **onActivityResult()** afin de récupérer les champs « **DISPLAY_NAME** » et « **FORMATTED_ADDRESS** » du contact sélectionné. On affichera le nom du contact et son adresse dans les contrôles **TextView** prévus à cet effet.
15. Vérifier le bon fonctionnement de l'ensemble.

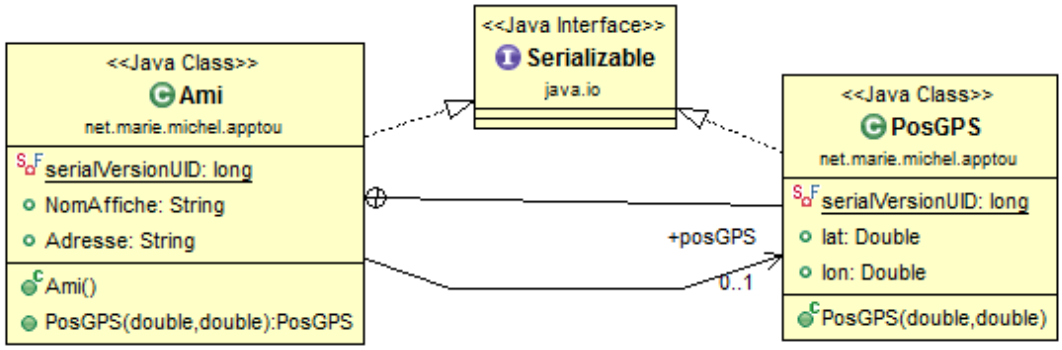
Remarque : *penser à ajouter au moins deux contacts sur votre matériel avant de tester…*

Création d'une classe Ami « serializable »

Vous allez coder une classe **Ami** permettant le stockage du nom, de l'adresse et de la position GPS (une fois résolue). Cette classe servira pour passer comme paramètres les deux individus à localiser sur la carte, vous et votre ami(e) à retrouver. Ces objets doivent donc être « **serializable** ».

Rien de nouveau à ce sujet, le principe de la « serialization » ayant déjà été traité dans le TP1. Je vous propose donc la classe suivante que vous trouverez dans le dossier **Android_Resources**.

On peut en discuter si vous le souhaitez, et si elle ne vous convient pas vous pouvez bien entendu la coder « à votre sauce ».



16. Ajouter à la classe **MainActivity** les données membres privées suivantes pour la sauvegarde des deux objets **Ami** :
    - ✓ **monPote** de type **Ami**, objet représentatif de votre ami(e) sélectionné(e) dans les contacts,
    - ✓ **moi de type Ami**, objet représentatif de l'utilisateur du périphérique, vous…
17. Compléter la méthode **onActivityResult()** afin d'instancier l'objet **monPote** de type **Ami** et de l'initialiser (nom et adresse). Faire également en sorte que le bouton « Retrouver mon ami ?? » soit actif si le contact sélectionné contenait une adresse et inactif autrement.

Gestion des positions GPS des deux amis, utilisation du « Geocoder »

18. Vérifier que la bibliothèque **Google Play Service** et bien installée, sinon ajoutez là. Nous en aurons besoin pour utiliser la classe **Geocoder** ainsi que pour les fonctions **Google Map**.
19. Ajouter la méthode de prototype ci-dessous à la classe **MainActivity** et coder celle-ci afin qu'elle réalise la résolution inverse de l'adresse de votre ami, et qu'elle retourne la position GPS correspondante dans un élément **Ami.PosGPS**. (si besoin voir ***solutionTP3_q19.txt***)

    private Ami.PosGPS getPositionGPSFromAddress(String adresse)
20. Ajouter à la classe **MainActivity** une donnée membre privée **btnRefindFriend** de type **Button**.
21. Compléter la méthode surchargée **Activity.onCreate()** afin :
    - ✓ d'instancier l'objet **btnRefindFriend**,
    - ✓ d'associer un écouteur sur ce bouton pour capturer l'événement **onClick** et déclencher la méthode associée.
22. Coder la méthode **onClick()** afin qu'elle :
    - ✓ réalise la résolution inverse de l'adresse de votre ami et complète l'objet **monPote** du résultat obtenu,
    - ✓ vérifie qu'une position GPS est affichée dans les contrôles **edtLat** et/ou **edtLon** et l'utilise pour initialiser l'objet **moi**. Dans le cas contraire utiliser une position par défaut pour vous localiser et la définir comme constante finale,
    - ✓ affiche dans un Toast l'adresse et la position GPS de votre ami.

      (si besoin voir ***solutionTP3_q22.txt***)
23. Vérifier le bon fonctionnement de cette partie avant de poursuivre.

Ajouter l'activité d'affichage des « punaises » sur la carte

Vous allez à présent créer une seconde activité visant à afficher sur la carte deux punaises correspondant à votre position et celle de votre ami.

24. Ajouter une **Google Map Activity** nommé **activity_map.xml** avec layout relatif pour le design XML et **MapActivity.java** pour le code java.
25. Vérifier la déclaration de cette nouvelle activité dans le manifeste.
26. Faire ce qui est demandé dans le fichier **google_maps_api.xml** (présenté en cours) afin d'obtenir la clé d'utilisation de la Google Map API V2.

    Pour vous authentifier sur le site Google vous pouvez utiliser le compte Gmail suivant ou votre compte perso si vous le souhaitez :

    - ✓ Username = **androidN.iris@gmail.com** (avec N = numéro du device utilisé)
    - ✓ Password = **Password1234@il**
27. Vérifier (et compléter si nécessaire) toutes les définitions nécessaires dans le manifeste de l'application.
28. Revenir sur la classe **MainActivity** afin de compléter la méthode associée au clic sur le bouton « Retrouver mon ami(e) ?? » pour qu'elle charge l'activité d'affichage de la carte.
29. Vérifier le bon fonctionnement de cette partie avant de poursuivre.
30. Compléter la méthode associée au clic sur le bouton « Retrouver mon ami(e) ?? » pour qu'elle passe à l'activité **carte** les paramètres suivants afin de permettre l'ajout des punaises :
    - ✓ l'objet **moi** pour vous localiser sur la carte,
    - ✓ l'objet **monPote** pour localiser votre ami(e) sur la carte.
31. Compléter la méthode **onCreate()** de la classe **MapActivity** afin de créer et d'ajouter les punaises à la carte puis de redessiner la carte en la limitant au cadre contenant les punaises.
32. Vérifier le bon fonctionnement de l'ensemble.

Extension possible : le tracé de la route

Ajouter le tracé de la route (à pied ou en voiture) entre vous et votre ami(e)

La récupération des différents segments de la route entre deux points se fait en interrogeant le serveur **Google Map** via une URI de type GET permettant le passage de paramètres tels que les positions de départ et d'arrivée, le système d'unités à utiliser et le type de transport.

Exemple :

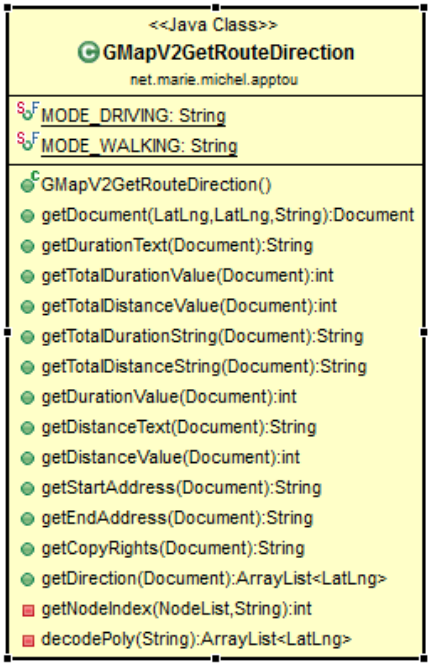
```
String url = "http://maps.googleapis.com/maps/api/directions/xml?"
        + "origin=" + start.latitude + "," + start.longitude
        + "&destination=" + end.latitude + "," + end.longitude
        + "&sensor=false&units=metric&mode="+mode;
```

La réponse HTML est au format XML, son décodage afin de récupérer les segments de route à tracer, les distances des segments, et le temps de parcours des segments, nécessite le codage d'une classe « parseur » XML. Ci-dessous un exemple (incomplet) de réponse XML faisant apparaître le schéma XML de la réponse :

```
<DirectionsResponse>
 <status>OK</status>
 <route>
  <summary>I-40 W</summary>
  <leg>
```

```
<step>
 <travel_mode>DRIVING</travel_mode>
 <start_location>
  <lat>41.8507300</lat>
  <lng>-87.6512600</lng>
 </start_location>
 <end_location>
  <lat>41.8525800</lat>
  <lng>-87.6514100</lng>
 </end_location>
 <polyline>
  <points>a~l~Fjk~uOwHJy@P</points>
 </polyline>
 <duration>
  <value>19</value>
  <text>1 min</text>
 </duration>
 <html_instructions>Head <b>north</b> on <b>S Morgan St</b> toward <b>W Cermak
                    Rd</b></html_instructions>
 <distance>
  <value>207</value>
  <text>0.1 mi</text>
 </distance>
</step>
...
... additional steps of this leg
```

Aussi la classe **GapV2GetRouteDirection.java** que vous trouverez dans le dossier **Android_Resources** réalise cette fonction de parseur. C'est une classe que j'ai pu récupérer sur les nombreux exemples fournis par Google et que j'ai complétée de deux méthodes pour obtenir la distance totale et la durée totale d'un trajet. Ci-dessous le diagramme de cette classe :



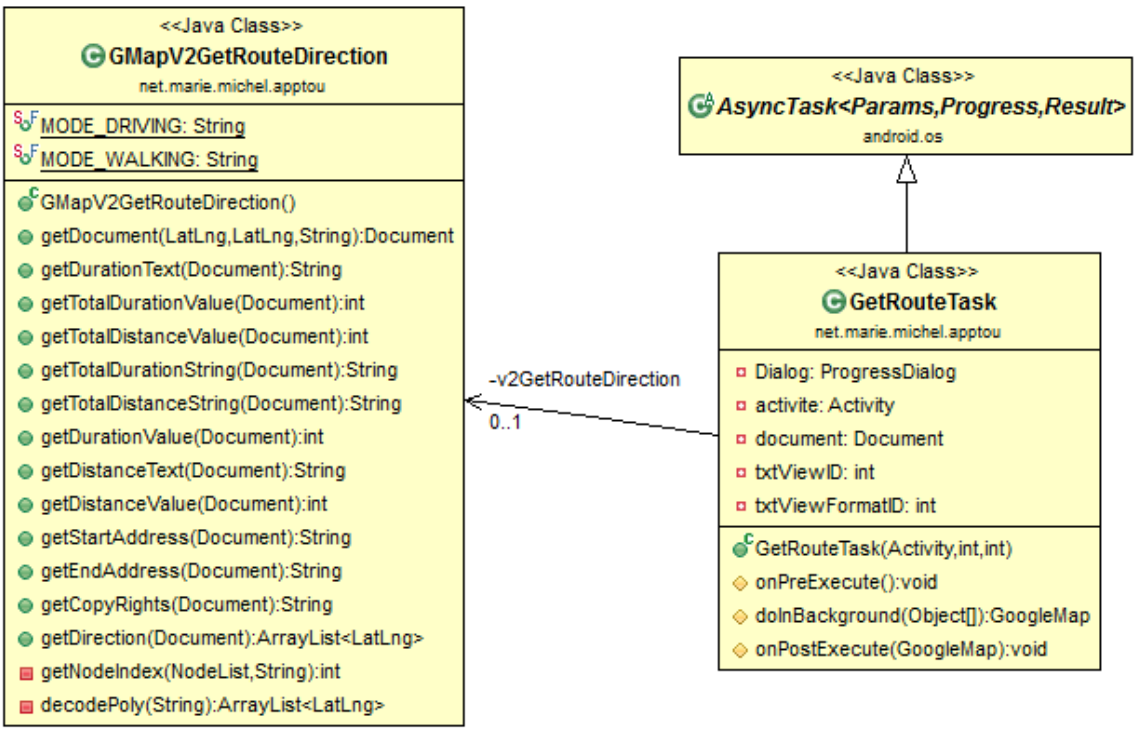
Quelques explications sur les méthodes utiles à cette partie…

- ✓ **getDocument()** : retourne dans un objet **Document** la réponse XML à la requête constituée d'un point de départ (position GPS), d'un point d'arrivée et d'un mode de transport (**MODE_DRIVING** ou **MODE WALKING**),
- ✓ **getDirection()** : retourne une liste des points de passages (type **LatLng**) qui constituent la route décrite par un objet Document,
- ✓ **getTotalDurationString()** : retourne la durée totale de la route décrite par un objet Document sous la forme d'une chaîne « heures minutes secondes »,

- ✓ **getTotalDistanceString()** : retourne la distance totale de la route décrite par un objet Document sous la forme d'une chaîne « en km ou en m si la distance est inférieure à 1km ».

Enfin comme je pressens bien que ce n'est pas suffisant, voici encore un cadeau maison, une classe **GetRouteTask** qui permet « en tâche de fond » d'effectuer la requête Google Map pour obtenir la description XML de la route, puis une fois la réponse obtenue, en « post traitement » de tracer la route sur la carte et de retourner la durée et la distance totale du trajet. En bonus elle affiche un message « Chargement de la route » sur l'activité courante (l'activité carte) pendant que la tâche de fond récupère le flux XML.

Ci-dessous le diagramme de classe faisant apparaître la classe **GetRouteTask** héritant de la classe **AsyncTask** déjà vue et utilisée dans le TP1. Elle utilise la classe **GMapV2GetRouteDirection** pour récupérer et « parser » les éléments de la route **Google Map**.



Le constructeur de cette classe **GetRouteTask** attend comme paramètres :

- ✓ la référence à l'activité en charge de l'affichage de la carte,
- ✓ l'identificateur du contrôle « **TextView** » en charge de l'affichage de l'information distance et durée du trajet,
- ✓ l'identificateur de la chaîne formatée à utiliser pour afficher la durée et la distance.

La méthode **execute()** héritée de la classe **AsyncTask** attend un tableau de type **Object** qui doit contenir dans l'ordre :

- ✓ une chaîne spécifiant le type de transport à utiliser pour la route (**MODE_DRIVING** ou **MODE_WALKING**),
- ✓ un objet **LatLng** position origine du trajet,
- ✓ un objet **LatLng** position finale du trajet,
- ✓ un objet **GoogleMap** référence à la carte **Google Map** affichée dans l'activité.

33. Compléter le fichier **activity_map.xml** en lui ajoutant un contrôle **TextView** pour l'affichage de la durée et la distance du trajet.

    Exemple de configuration du contrôle **TextView** :

    <TextView

```
android:id="@+id/txtDureeDistance"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/txtDureeDistance"
        android:textColor="#0000FF"
        android:textSize="20sp"
        android:textStyle="bold"
         android:layout_marginTop="2dp"
        android:gravity="center"
/>
```

34. Compléter le fichier **strings.xml** avec l'élément texte du contrôle précédent par défaut et ajouter un élément pour l'affichage du texte qui sera formaté avec l'information durée et distance du trajet.

    Exemple de configuration des chaînes du contrôle **TextView** :

```
<string name="txtDureeDistance">Durée trajet = ?? ; Distance = ??</string>
<string name="txtDureeDistanceParams">Durée trajet = %1$s ; Distance = %2$s</string>
```

35. Compléter la méthode **onCreate()** de la classe **MapActivity** afin d'utiliser la classe **GetRouteTask** pour ajouter le tracé de la route entre vous et votre ami(e) sur la carte. Utiliser un trajet à pied ou en voiture, selon votre goût, défini « en dur » dans un premier temps.
36. Vérifier le bon fonctionnement de l'ensemble.
37. Revenir sur la classe **MainActivity** afin de compléter la méthode associée au clic sur le bouton « Retrouver mon ami(e) ?? » pour qu'avant de charger la carte elle demande via une **AlertDialog,** de choisir entre un trajet à pied ou en voiture. L'information sur le type de trajet retenu sera transmise à l'activité carte à l'aide d'un paramètre supplémentaire de l'intention.
38. Compléter la méthode **onCreate()** de la classe **MapActivity** afin de prendre en compte le type de trajet sélectionné.
39. Vérifier le bon fonctionnement de l'ensemble.

Sujet, ressources et corrigé :