Développement Android

Jean-François COUCHOT
couchot[arobase]femto-st[point]fr

16 octobre 2017

Table des matières

Chapitre 1

Projet sous Android Studio et interfaces utilisateur

1.1 Projet Android. Organisation d'interface

Exercice 1.1. *Organisation d'un projet Android.*

1. *Construire un projet Android pour téléphone et tablette à base de « Navigation Drawer » et comprendre les options proposées.*
2. *Dans la vue « Project » d'Android Studio, étudier chaque dossier et sous-dossier.*
3. *Comprendre le chaîne de compilation d'un projet Android présentée à l'adresse :* `https://developer.android.com/studio/build/index.html`.
4. *Représenter graphiquement à l'aide d'un arbre la hiérarchie des composants de l'interface utilisateur à partir de* `setContentView(R.layout.activity_main)` *du fichier* `MainActivity`.

A la fin de cette section vous devez avoir compris :
— ce que contient un projet Android notamment, le manifest, les fichiers gradle, les ressources, les sources ;
— comment est compilée un projet Android ;
— comment est organisée le projet basé sur « Navigation Drawer ».

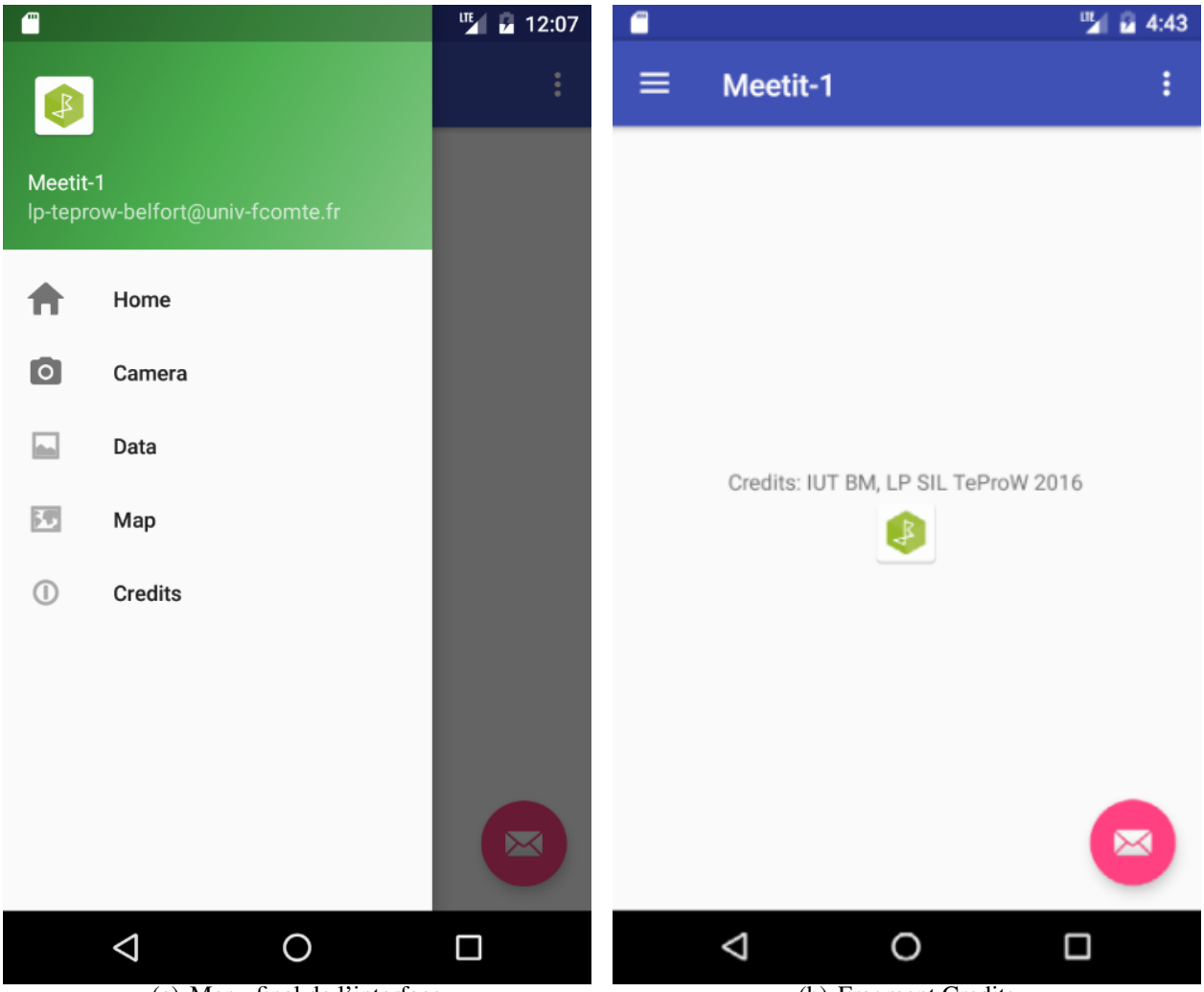
1.2 Ressources

Exercice 1.2. *Modification de l'entête du menu de navigation. Tout ce qui suit concerne l'entête du menu de navigation, c'est à dire la partie supérieure gauche de la figure 1.1(a).*

1. *Repérer le fichier xml qui définit ceci.*
2. *Modifier les fichiers* `strings.xml` *et* `nav_header_main.xml` *pour faire en sorte que le nom de l'application et votre adresse mail apparaissent en lieu et place de l'existant.*
3. *Pour modifier le logo (et d'une manière générale ajouter une image) :*
   (a) *Récupérer un logo en haute définition.*
   (b) *Demander la création d'une nouvelle* `Image Asset`.
   (c) *Préciser que c'est une icône de lancement, construite à partir d'une image dont on dira où elle est enregistrée et lui donner le nom* `ic_launcher`.
   (d) *Vérifier que l'image produite a été ajoutée dans le dossier mipmap sous différentes tailles.*
   (e) *Dans l'élément* `ImageView`, *renseigner* `@mipmap/ic_launcher`.

Exercice 1.3. *Modification du contenu du menu de navigation. Tout ce qui suit concerne le contenu du menu de navigation, représenté dans la partie gauche de la figure 1.1(a).*

1. *Repérer le fichier xml qui définit ceci.*
2. *Modifier le contenu du menu de navigation pour avoir le même menu que celui présenté à la figure 1.1(a). Les identifiants des menus doivent être respectivement :* `nav_home`, `nav_camera`, `nav_data`, `nav_map` *et* `nav_credits`.
3. *Modifier l'activité* `MainActivity` *pour qu'elle accepte ces identifiants : ceci se fait dans la méthode* `onNavigationItemSelected`.



(a) Menu final de l'interface (b) Fragment Credits

FIGURE 1.1 – Premières interfaces de l'application

A la fin de cette section vous devez avoir compris :
— où sont stockées les ressources d'un projet Android et sous quelles formes,
— comment en ajouter, comment les modifier.

1.3 Activités et Fragments

Exercice 1.4. *Migration vers les fragments.*

L'objectif de cet exercice est ne plus avoir de TextView affichant "Hello World", mais d'avoir un fragment qui effectue ceci dynamiquement.

1. *Dans le layout contenant le TextView affichant "Hello World", supprimer ce TextView.*

2. *Repérer l'identifiant* `content_main` *dans le RelativeLayout de ce fichier ; s'il n'y est pas (dépend de la version de Studio) préciser que ce RelativeLayout a pour identifiant* `@+id/content_main` *puis réexécuter le programme. Que constatez-vous ?*
3. *Demander à Studio de créer un nouveau fragment blanc, nommé HomeFragment. Constater la création de la classe HomeFragment et du layout* `fragment_home.xml`.
4. *On va demander qu'à la création l'activité principale MainActivity, le fragment précédemment défini soit positionné à la place de l'élément identifié par* `content_main` *vu à la question 2. Pour cela :*
   (a) *Préciser que l'activité MainActivity interagit avec ce fragment*

```
public class MainActivity extends AppCompatActivity
  implements NavigationView.OnNavigationItemSelectedListener,
   HomeFragment.OnFragmentInteractionListener
```

   et demander à Studio d'implanter la méthode requise.

   (b) *Créer les deux attributs privés de MainActivity après avoir compris leur sens :*

```
private FragmentManager fm = null;
private Fragment fragment = null;
```

   (c) *Dans sa méthode* `onCreate`, *après* `setContentView(R.layout.activity_main)`, *ajouter les lignes suivantes après avoir compris leur sens :*

```
fm = getSupportFragmentManager();
fragment = new HomeFragment();
fm.beginTransaction().replace(R.id.content_main,fragment).commit();
```

   Constater que l'objectif est atteint.

Exercice 1.5. *Remplacement d'un fragment par un autre.*

L'objectif de cet exercice est de faire remplacer le fragment `HomeFragment` *par le fragment nommé* `CreditsFragment` *lorsque l'utilisateur valide l'item « Credits » du menu latéral.*

1. *Le fragment* `CreditsFragment` *contient les éléments graphiques représentés à la figure 1.1(b).*
2. *Reprendre partiellement l'exercice précédent pour réaliser ceci. Le code à exécuter lorsque l'utilisateur choisit un item du menu latéral se trouve dans la méthode* `MainActivity.onNavigationItemSelected`.
3. *Améliorer votre code en remplaçant la cascade de if dans cette méthode par un* `switch....case`.
4. *Faire de même avec* `MyCameraFragment`, `MyDataFragment` *et* `MyMapFragment`.

A la fin de cette section vous devez avoir compris :
— ce qu'est une activité, ce qu'est un fragment, leurs points communs, leurs différences ;
— comment ajouter une activité et ce que cela implique ;
— comment ajouter un fragment et comment le charger dans une activité.

1.4 Paramètres

Exercice 1.6. *introduction aux paramètres de l'application. L'objectif est de construire une interface permettant à l'utilisateur de saisir ses préférences vis à vis de l'application. Ceci se fait classiquement en construisant une activité de paramètres (Settings), comme représentée à la figure 1.2(b) et qui s'affiche lorsque l'utilisateur clique sur les ":" en haut à droite de l'application.*

1. *Demander à Studio de créer une nouvelle activité de type* `Settings Activity`. *Constater l'ajout :*
   — *de la classe abstraite* `AppCompatPreferenceActivity` *qu'on ne modifiera pas ;*
   — *de la classe* `SettingsActivity` *; constater que cette classe définit aussi les classes* `GeneralPreferenceFragment`, `NotificationPreferenceFragment` *et* `DataSyncPreferenceFragment` *;*
   — *du dossier* `xml` *de ressources ;*

— *des layout* `pref_general.xml`, `pref_data_sync.xml`, `pref_notification.xml` *et* `pref_ headers.xml` *enregistrés dans le dossier* `xml`*; ces gabarit définissent l'affichage des fragments présentés ci-avant.*

2. *Modifier la méthode* `MainActivity.onOptionsItemSelected` *pour qu'elle demande à Android le démarrage de l'activité* `SettingsActivity`*. Il suffit pour cela d'adapter le code comme suit, qu'on doit comprendre :*

```
if (id == R.id.action_settings) {
  Intent intent = new Intent(this, SettingsActivity.class);
  startActivity(intent);
  return true;
}
```

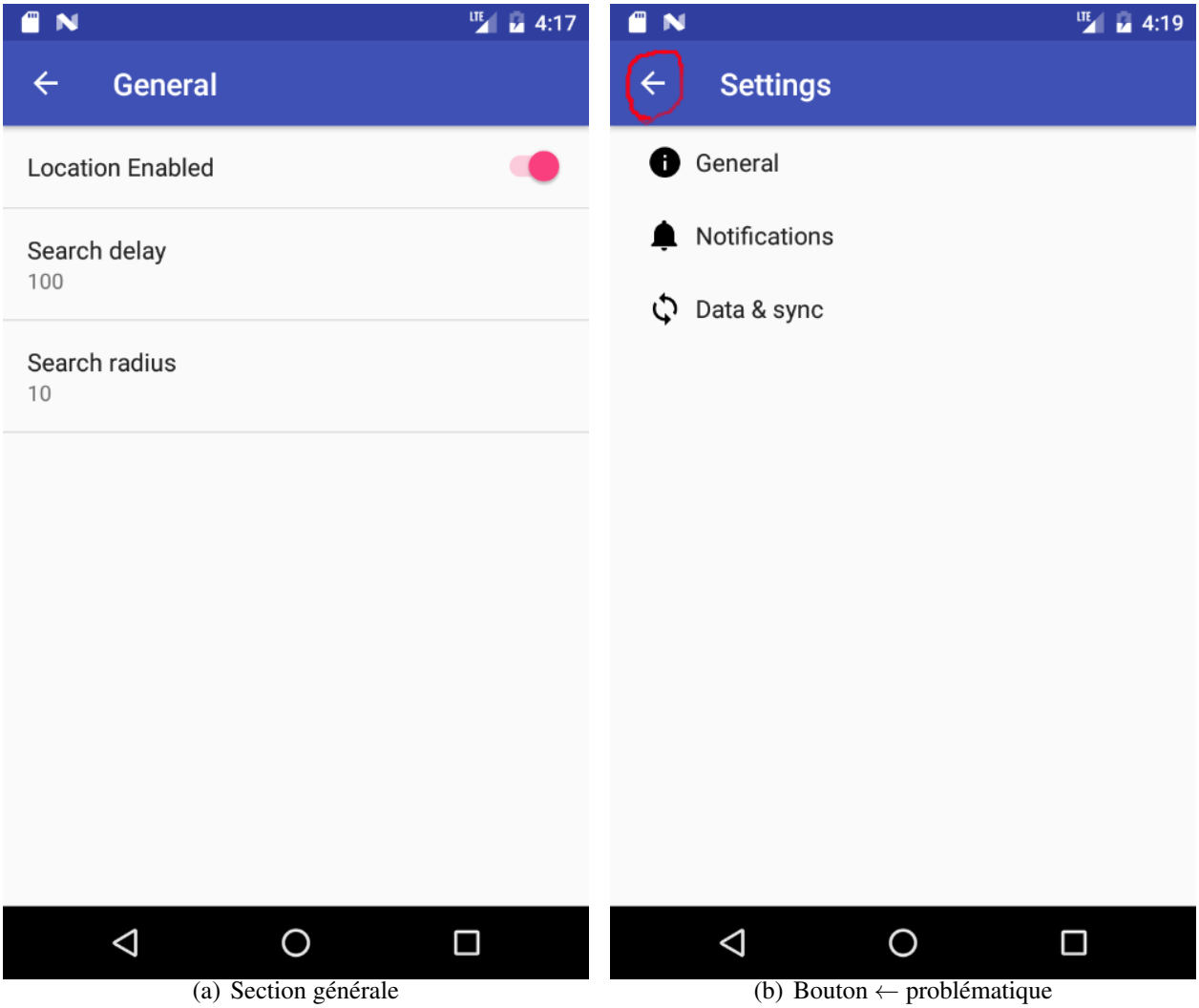
Exercice 1.7. *Modification des préférences par défaut.*

1. *A l'aide de Studio, remplir l'élément* `PreferenceScreen` *du fichier* `pref_general.xml` *comme présenté à la figure figure 1.2. Les libellés seront stockés dans le fichier* `strings.xml` *et les valeurs par défaut des éléments présentés sont* `false`*, 100 et 10 respectivement.*
2. *Faire en sorte que l'interrupteur* `Location enabled` *active la possibilité de saisir le rayon de recherche ainsi que le délai entre deux demandes de localisation.*
3. *L'invocation de* `bindPreferenceSummaryToValue` *dans la méthode* `onCreate` *de la classe* `GeneralPreferenceFragment` *génère des erreurs. Corriger le code pour les supprimer.*

Exercice 1.8. *Correction d'un bug dans les préférences.*
Constater qu'appuyer sur le bouton ← de la barre de `Settings` *ne permet pas de revenir à l'activité principale (cf figure 1.2(b)). Ajouter la méthode* `SettingsActivity.onOptionsItemSelected` *comme suit (et la comprendre) :*

```
public boolean onOptionsItemSelected(MenuItem item) {
  int id = item.getItemId();
  if (id == android.R.id.home) {
    this.finish();
    return true;
  }
  return super.onOptionsItemSelected(item);
}
```

A la fin de cette section vous devez avoir compris :
— ce qu'est une activité de type Settings,
— comment la démarrer, comment l'arrêter,
— comment modifier les paramètres à mémoriser, particulièrement leurs types, leurs valeurs.



(a) Section générale (b) Bouton ← problématique

FIGURE 1.2 – Interfaces de Paramètres

Chapitre 2

Sauvegarde de données

Tout ce qui suit démarre avec l'archive `Meetit-1.tgz`. Archivez votre travail précédent, décompressez l'archive `Meetit-1.tgz` et ouvrez le projet Android `Meetit-1`.

2.1 Les chaînes de caractères constantes

Dans un projet Android, on peut déclarer des chaînes de caractères constantes de deux manières.

— Soit comme un attribut d'une classe `MaClasse` de la forme
`public final static String MA_CLEF="la clef finale";`
puis y accéder directement avec `MaClasse.MA_CLEF` comme dans l'exemple

```
Toast.makeText(getApplicationContext(),
               MaClasse.MA_CLEF,
               Toast.LENGTH_SHORT).show();
```

— Soit comme un élément du fichier `strings.xml` (qu'on ne traduira pas dans une autre langue) :
`<string name="key_search_delay">key_search_delay</string>`
puis y accéder via `getResources().getString(R.string.id)` comme dans

```
Toast.makeText(getApplicationContext(),
               getResources().getString(R.string.key_search_delay),
               Toast.LENGTH_SHORT).show();
```

Exercice 2.1.
1. *Repérer dans le fichier* `strings.xml` *les chaînes de caractères constantes.*
2. *Repérer leur utilisation dans* `pref_general.xml` *à la fois dans les attributs* `android:key`, `android:title` *et* `android:dependency`. *Comprendre ces attributs.*
3. *Repérer leur utilisation à la ligne 192 du fichier* `SettingsActivity.java`.

A la fin de cette section, vous devez avoir compris comment fixer des constantes de type `string` dans `strings.xml` et comment y accéder, à la fois dans les ressources et dans du code java.

2.2 Les préférences

La valeur de chaque préférence déclarée dans un élément xml `PreferenceScreen` est toujours enregistrée dans un fichier dit de `SharedPreferences`. Ce fichier de préférences partagées contient toutes les paires (LaClef,SaValeur) qui sont déclarées dans un des éléments de `PreferenceScreen` et modifiées éventuellement par l'utilisateur. Ce fichier est accessible en invoquant, dans n'importe quelle classe, la méthode statique `PreferenceManager.getDefaultSharedPreferences()`.

2.2.1 Les types des valeurs saisies dans les Settings

Un `SwitchPreference` ou un `CheckBoxPreference` a deux états possibles : activé ou non. L'état est enregistré sous la forme d'un Boolean. La valeur d'un `EditTextPreference` ou d'une

`ListPreference` est enregistrée sous la forme d'une chaîne de caractères.

2.2.2 Extraction des valeurs des Settings depuis des SharedPreferences

Exercice 2.2. *Récupérer les données enregistrées dans les Settings.*

1. *Le gabarit* `fragment_my_data.xml` *du fragment* `MyDataFragment.java` *a été modifié. Le comprendre et repérer les trois* `TextView` *qui ont un identifiant. A quoi ces* `TextView` *vont-ils servir ?*
2. *Dans ce qui suit, l'objectif est de remplir chacun de ces* `TextView` *avec la donnée saisie par l'utilisateur dans les Settings.*
   (a) *Déclarer les deux attributs suivants dans la classe* `MyDataFragment` *:*

```
private SharedPreferences sharedPref;
private TextView tv_loc_enabled_out = null;
View root = null;
```

   (b) *A quoi sert le premier attribut de la question précédente ? Le second ?*
   (c) *Remplacer la méthode* `onCreateView` *par le code suivant :*

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
  // Inflate the layout for this fragment
  root =  inflater.inflate(R.layout.fragment_my_data, container, false);
  return root;
}

public void updateUI(){
  sharedPref = PreferenceManager.getDefaultSharedPreferences(getActivity());
  Boolean locationEnabled =
      sharedPref.getBoolean(getResources().getString(R.string.key_location_switch), false);
  String isLocationEnable = ": ";
  isLocationEnable += locationEnabled ? "True" : "False";

  tv_loc_enabled_out = (TextView) root.findViewById(R.id.text_location_switch_out);
  tv_loc_enabled_out.setText(isLocationEnable);
}

@Override
public void onResume(){
  super.onResume();
  updateUI();
}
```

   (d) *Comprendre chaque ligne du code précédent.*
   (e) *Exécuter le code et constater que l'on atteint partiellement l'objectif.*
   (f) *Compléter le code de* `updateUI` *pour que l'application affiche aussi le délai entre deux recherches ainsi que le rayon de recherche dans les* `TextView` *prévus à cet effet.*

A la fin de cette section, vous devez avoir compris
— où une `Settings Activity` enregistre les paramètres modifiés par l'utilisateur ;
— comment récupérer la valeur d'un paramètre de l'application ;
— comment afficher cette valeur dans une `TextView`.

2.3 Les bases de données

Les personnages de l'application seront mémorisés dans une base de données. Android permet de gérer en interne des bases de données de type SQLite.

2.3.1 Introduction aux bases de données

Exercice 2.3. *Introduction à partir du tutoriel* [1].

1. *Comprendre pourquoi on construit une classe* `Contact`.
2. *La classe qui étend* `SQLiteOpenHelper` *contient de nombreuses constantes statiques. Expliquer pourquoi celles-ci seront enregistrées dans cette classe et non pas dans le fichier* `strings.xml`.
3. *Expliquer pourquoi l'élément* `KEY_ID` *est défini par* `INTEGER PRIMARY KEY`, *sans* `AUTOINCREMENT`.
4. *Dans les méthodes CRUD se concentrer d'abord sur* `addContact(Contact contact)` :
   (a) *Pourquoi passe-t-on comme paramètre :* `Contact contact` *?*
   (b) *Comment explique-t-on* `getWritableDatabase()` *?*
   (c) *Comprendre les lignes restantes.*
5. *Dans les méthode CRUD se concentrer ensuite sur* `getAllContacts()` :
   (a) *Pourquoi retourne-t-on un objet de la classe* `List<Contact>` *?*
   (b) *Que pensez-vous de* `SQLiteDatabase db = this.getWritableDatabase()` *?*
   (c) *Comprendre les lignes restantes.*

Exercice 2.4. *Organisation en packages. Dès que le projet grandit, il est temps de s'organiser dans le package principal.*

1. *Construire les packages* `views`, `models` *et* `managers`.
2. *Déplacer toutes les classes existantes dans* `views`.
3. *Où sera enregistrée la classe qui gérera la base de données ?*

2.3.2 La base characterManager

Exercice 2.5. *Une base de données. Les personnages de l'application seront mémorisés dans la base de données "characterManager" contenant la table "characters" définie par*

```
id INTEGER PRIMARY KEY, firstname TEXT, familyname TEXT,
weburl TEXT, latitude REAL, longitude REAL, image BLOB.
```

1. *Implanter la classe* `Character` *possédant les attributs correspondants à cette définition. Ceci devrait être fait dans le package* `models`. *Prévoir une méthode* `toString()` *qui retourne l'identifiant, le prénom et le nom du personnage.*
2. *Implanter la classe* `CharactersDatabaseHandler` *qui permet de construire la base de données. Ceci devrait être fait dans le package* `managers`.
3. *A l'initialisation, ajouter dans la table* `characters` *les données qui suivent. On remarquera que l'on insère pas d'image dans la base à cette étape.*

```
Jean-François Couchot
  http://members.femto-st.fr/jf-couchot/fr  47.642900  6.840027
Raphaël Couturier
  http://members.femto-st.fr/raphael-couturier/fr 47.659518, 6.813337
Stéphane Domas
  http://info.iut-bm.univ-fcomte.fr/staff/sdomas/ 47.6387143 6.8370225
Abdallah Makhoul
  http://members.femto-st.fr/abdallah-makhoul/fr 47.638114 6.862139
```

4. *Compléter cette classe pour qu'on puisse :*
   (a) *afficher dans le logcat les informations sur les personnages classés par ordre alphabétique en se limitant aux* `n` *premiers résultats :* `List<Character> getCharacters(int n);`
   (b) *y ajouter le personnage passé en paramètre :* `addCharacter(Character char)`.

1. http://www.androidhive.info/2011/11/android-sqlite-database-tutorial

2.3.3 Gestion des images dans la base

Exercice 2.6. *Un dossier Assets pour des données initiales. Les quatre personnages initiaux n'ont pour l'instant pas de données dans le champs* `image`*. L'objectif est d'intégrer celles-ci à l'initialisation de la base de données avec des images brutes intégrées à l'application.*

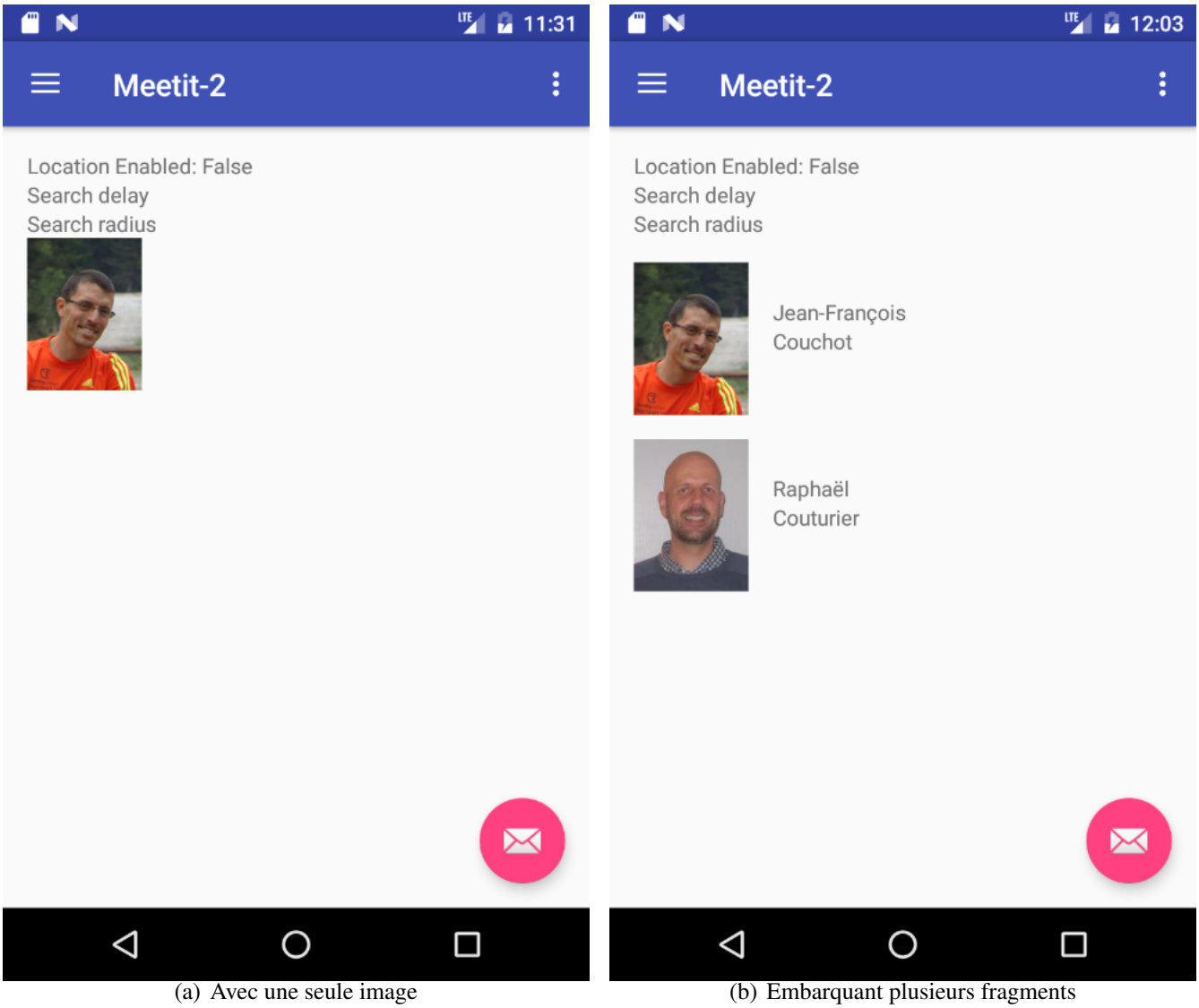
1. *Demander à Studio de construire un nouveau dossier de données (Assets Folder).*
2. *Récupérer les photos des personnages et les enregistrer dans ce dossier. Constater leur présence dans l'arborescence de Studio.*
3. *Dans la classe* `CharactersDatabaseHandler` *ajouter la méthode*

```
private void update_image(SQLiteDatabase db, String s, int id) {
  try {
    # translates an asset image into an array of bytes
    InputStream inputStream  = this.mContext.getAssets().open(s);
    byte[] bitmapdata = new byte[inputStream.available()];
    inputStream.read(bitmapdata);
    inputStream.close();

    # updates the database
    ContentValues values = new ContentValues();
    values.put(KEY_IMAGE, bitmapdata);
    db.update(TABLE_CHARACTERS, values, KEY_ID + " = ?",
      new String[]{String.valueOf(id)});

  } catch (Exception e) {
    e.printStackTrace();
  }
}
```

4. *Comprendre le code présenté à la question précédente. On commencera par s'intéresser aux paramètres de la méthode.*
5. *De quel type semble être l'attribut* `mContext`*? Ajouter cet attribut à la classe et modifier le constructeur* `CharactersDatabaseHandler(Context context)` *pour l'instancier.*
6. *Modifier la méthode* `onCreate` *pour qu'elle mette à jour la photo de chaque personnage. Ce code devrait être placé à la suite de ce qui a été fait à la question 3 de l'exercice 2.5.*
7. *Modifier la méthode* `MyDataFragment.updateUI` *pour qu'elle affiche la photo d'un des personnages dans l'ImageView identifiée par* `image_view_test` *du gabarit* `fragment_my_data`*. Vous devriez obtenir l'interface représentée à la figure 2.1(a).*

Exercice 2.7. *Synthèse. Cet exercice combine beaucoup éléments vus jusqu'à présent : la lecture dans une base, l'insertion dynamique de fragments, la mise en forme selon des gabarits xml... Construire l'interface représentée à la figure 2.1(b), sachant tout est dynamique.*

A la fin de cette section, vous devez avoir compris
— comment construire une base de données SQLite ;
— comment effectuer des requètes CRUD sur cette base ;
— comment raisonner objet avec cette base comme dans un ORM ;
— comment gérer des images dans une base ;
— comment exploiter des fragments liés à une base de données.



(a) Avec une seule image

(b) Embarquant plusieurs fragments

FIGURE 2.1 – Le fragment de données

Chapitre 3

Cartographie et localisation

Tout ce qui suit démarre avec l'archive `Meetit-2.tgz`. Archivez votre travail précédent, décompresser l'archive `Meetit-2.tgz` et ouvrir le projet Android `Meetit-2`.

3.1 Affichage d'une carte Google Map

Exercice 3.1. *Préalable pour l'utilisation de Google Map V2.*

1. *Installer dans le sdk manager l'élément suivant dans sa dernière version (si ce n'est pas fait) :* `extras> Google Play Services`
2. *Ajouter la ligne suivante dans la section "dependencies" du fichier* `build.gradle` *du dossier* `app` *:* `compile 'com.google.android.gms:play-services:11.4.2'`
3. *Synchroniser votre projet pour avoir toutes les bibliothèques correctement installées.*

Exercice 3.2. *Une clef pour l'API Google Map.*
L'objectif de cet exercice est de construire la clef permettant d'exploiter la cartographie Google.
A l'URL https://console.developers.google.com/iam-admin/projects *:*

1. *Créer un projet avec comme nom « Meetit-VOTRE-NOM ».*
2. *Activer l'API « Google Map Android API ».*
3. *Créer la clef d'API sans aucune restriction.*
4. *Ajouter l'élément suivant dans l'élément* `application` *du fichier* `AndroidManifest.xml` *et remplacer* `VOTRE_CLEF_D_API` *par celle générée par Google.*

```
<meta-data
   android:name="com.google.android.maps.v2.API_KEY"
   android:value="VOTRE_CLEF_D_API" />
```

Exercice 3.3. *Modification du fragment d'affichage de carte.*

1. *Dans le gabarit* `fragment_my_map.xml` *remplacer l'élément* `textView` *par ce qui suit et en comprendre le sens.*

```
<fragment
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:id="@+id/map"
  android:name="com.google.android.gms.maps.SupportMapFragment" />
```

2. *Constater le bon fonctionnement du chargement de la carte dans l'émulateur.*

3.2 Modifier quelques paramètres de la cartes

Exercice 3.4. *Ajout de marqueurs statiques. L'objectif de cet exercice est de contrôler quelques éléments dans l'affichage de la carte affichée par défaut.*

1. *Commencer par dire que le Fragment implémente* `OnMapReadyCallback` *et laisser Studio implanter la méthode* `onMapReady`.
2. *Remplacer la méthode* `onCreateView` *par ce qui suit et en comprendre le sens.*

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
  View root = inflater.inflate(R.layout.fragment_my_map, container, false);
  SupportMapFragment mapFragment  =
     (SupportMapFragment)   getChildFragmentManager().findFragmentById(R.id.map);
  mapFragment.getMapAsync(this);
  return root;
}
```

3. *Remplacer la méthode* `onMapReady` *ajoutée par Studio par ce qui suit, et en comprendre le sens.*

```
@Override
public void onMapReady(GoogleMap googleMap) {
  LatLng sydney = new LatLng(-34, 151);
  googleMap.addMarker(new MarkerOptions().position(sydney).title("Marker in Sydney"));
  googleMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
  googleMap.moveCamera(CameraUpdateFactory.zoomTo(10));
}
```

Exercice 3.5. *Affichage sur une carte de données issues d'une base. L'objectif de cet exercice est d'extraire des données spatiales de la base de données et de les afficher sur la carte. Le rendu de cet exercice est représenté à la figure 3.1(a).*

1. *Pour chaque personnage de la base de données, afficher un marqueur en sa position.*
2. *Chaque marqueur aura un libellé explicite : le nom et le prénom du personnage.*
3. *Chaque marqueur aura une transparence de 50% ($\alpha = 0.5$).*
4. *Faire en sorte que la carte affiche tous les personnages, à l'aide du plus grand zoom possible. On peut exploiter pour cela la méthode* `CameraUpdateFactory.newLatLngBounds`.

3.3 Les autorisations

Exercice 3.6. *Permissions nécessaires L'objectif de cet exercice est de gérer les* permissions modifiables à l'exécution *introduites à partir de la version 23 d'Android.*

1. *Dans l'élément* `<manifest>` *du fichier* `manifest.xml`*, ajouter l'élément suivant :*

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

2. *Préciser que la classe* `MyMapFragment` *implante* `GoogleApiClient.ConnectionCallbacks` *et* `GoogleApiClient.OnConnectionFailedListener` *et demander à Studio d'ajouter les méthodes nécessaires.*
3. *Déclarer l'attribut* `mGoogleApiClient` *de type* `GoogleApiClient` *et l'initialiser à null.*
4. *Dans la méthode* `OnCreate` *de ce framgent, ajouter le code suivant et le comprendre.*

```
if (mGoogleApiClient == null) {
  mGoogleApiClient = new GoogleApiClient.Builder(getActivity())
                     .addConnectionCallbacks(this)
                     .addOnConnectionFailedListener(this)
                     .addApi(LocationServices.API)
                     .build();
}
```

5. *Ajouter les deux méthodes suivantes dans le fragment et les comprendre.*

```
@Override
public void onStart() {
  super.onStart();
  mGoogleApiClient.connect();
}

@Override
public void onStop() {
  super.onStop();
  mGoogleApiClient.disconnect();
}
```

6. *Ajouter une constante statique entière* `REQUEST_LOCATION` *initialisée à 1.*
7. *Remplacer la méthode* `onConnected` *par ce qui suit et en comprendre le sens.*

```
@Override
public void onConnected(@Nullable Bundle bundle) {
  if (ActivityCompat.checkSelfPermission(getActivity(), Manifest.permission.ACCESS_FINE_LOCATION)
          != PackageManager.PERMISSION_GRANTED) {
     requestLocationPermission();

  } else {
    mLastLocation = LocationServices.FusedLocationApi.getLastLocation(
    mGoogleApiClient);
    if (mLastLocation != null) {
      String lats = "" + mLastLocation.getLatitude();
      String longs = "" + mLastLocation.getLongitude();
      Toast.makeText(getActivity(), lats + " " + longs, Toast.LENGTH_LONG).show();
    }
  }
}

private void requestLocationPermission(){
  ActivityCompat.requestPermissions(
                getActivity(),
                new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
                REQUEST_LOCATION);
}
```

8. *Exécuter votre application dans l'émulateur. Constater que votre autorisation est demandée à la volée par l'application. Affiche-t-elle cependant la dernière position connue ?*

Exercice 3.7. *Évaluer un projet avec de la localisation L'objectif de cet exercice est d'évaluer l'application en utilisant des coordonnées GPS virtuelles. Pour cela on exécute une application (Maps) qui met à jour la dernière localisation connue et on travaille ensuite avec l'application Meetit-2.*

1. *Exécuter l'application Maps dans l'émulateur.*
2. *Ouvrir le panneau des « extended controls », représenté par « ... » dans la barre d'outils de l'émulateur. et envoyer la paire de coordonnées GPS correspondant à une longitude de 6.82813 et une latitude de 47.637942. Votre émulateur a ainsi une dernière position connue.*
3. *Exécuter votre application dans l'émulateur. Constater que la dernière position connue s'affiche avec un Toast.*

3.4 Récupérer régulièrement des coordonnées GPS

Exercice 3.8. *Récupérer les préférences de localisation.*

L'objectif de cet exercice est de ne récupérer les informations de localisation que si l'interrupteur « Location » a été activé dans les préférences. Dans le cas positif, on souhaite que le délai entre deux récupérations d'informations de localisation soit celui renseigné dans les préférences.

Créer une méthode privée `setLocationParmeters` *qui mémorise :*

1. *dans l'attribut booléen* `mRequestingLocationUpdates` *le fait que la localisation doit être tracée ou non (en fonction de l'interrupteur « Location »);*
2. *dans l'attribut* `mLocationRequest` *de type* `LocationRequest` *l'intervalle entre deux récupérations d'information de localisation saisi dans les préférences ; On pourra regarder la page* [1].

Exercice 3.9. *Mise à jour régulière des données de localisation. L'objectif de cet exercice est de récupérer les données de localisation selon la fréquence définie à l'exercice précédent.*

1. *Préciser que la classe* `MyMapFragment` *implante* `LocationListener` *et demander à Studio d'ajouter les méthodes nécessaires.*
2. *Ajouter les méthodes suivantes après les avoir comprises :*

1. https://developer.android.com/training/location/change-location-settings.html

```
@Override
public void onResume() {
  super.onResume();
  setLocationParmeters();
  if (mGoogleApiClient.isConnected()){
    if (mRequestingLocationUpdates) {
      startLocationUpdates();
    } else {
      stopLocationUpdates();
    }
  }
}

protected void startLocationUpdates() {
  if (ActivityCompat.checkSelfPermission(getActivity(), Manifest.permission.ACCESS_FINE_LOCATION)
        != PackageManager.PERMISSION_GRANTED) {
    requestLocationPermission();
  } else {
    LocationServices.FusedLocationApi.requestLocationUpdates(
                    mGoogleApiClient, mLocationRequest, this);
  }
}

protected void stopLocationUpdates() {
  LocationServices.FusedLocationApi.removeLocationUpdates(
  mGoogleApiClient, this);
}
```

3. *Ajouter la méthode suivante après l'avoir comprise :*

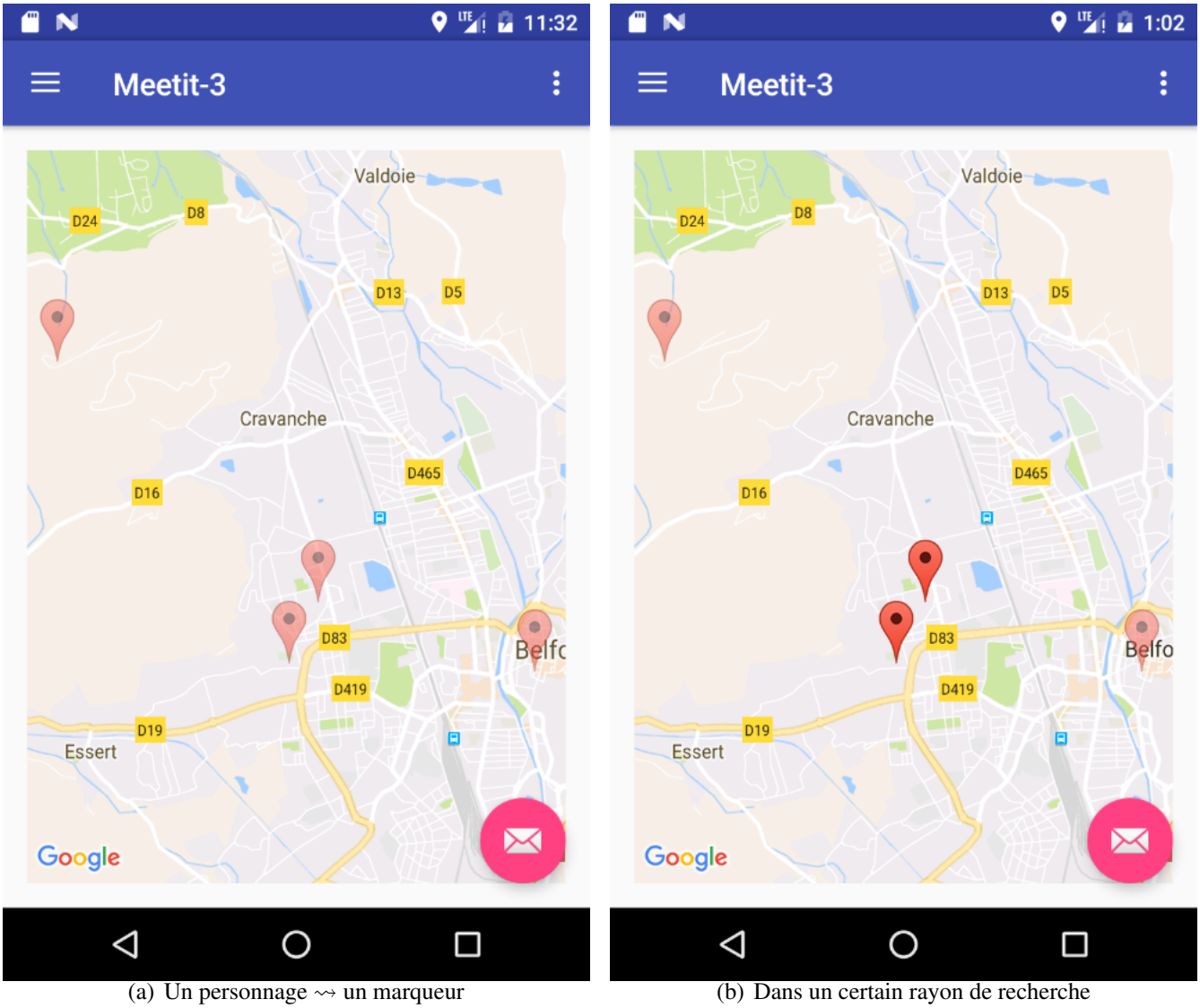
```
@Override
public void onPause() {
  super.onPause();
  stopLocationUpdates();
}
```

4. *Dans la méthode* `onConnected`*,*
   (a) *démarrer les demandes de mise à jour de la localisation (*`startLocationUpdates`*) si l'attribut booléen* `mRequestingLocationUpdates` *est vrai ;*
   (b) *arrêter les demandes de mise à jour de la localisation (*`stopLocationUpdates`*) si l'attribut booléen* `mRequestingLocationUpdates` *est faux.*
5. *Dans la méthode* `onLocationChanged`*, faire en sorte que la localisation s'affiche comme un Toast si celle-ci n'est pas nulle.*
6. *Activer la localisation et un délai de 10s dans les préférences de l'application. Changer les coordonnées GPS régulièrement dans la fenêtre de contrôles et constater le bon respect du délai.*
7. *Désactiver la localisation. Changer les coordonnées GPS régulièrement dans la fenêtre de contrôles et constater le bon respect du délai.*

Exercice 3.10. *Les personnages dans un certain rayon.*

L'objectif de cet exercice est d'informer au mieux l'utilisateur de la présence de personnage(s) dans le rayon précisé dans les préférences de l'application.

1. *Lorsque la localisation change, afficher dans un Toast les informations sur les personnages présents dans ce rayon.*
2. *Modifier le code précédent pour que les personnages soient représentés en rouge seulement lorsqu'il sont à l'intérieur de ce rayon et à 50% de transparence sinon, comme illustré à la figure 3.1(b).*



(a) Un personnage ⇝ un marqueur

(b) Dans un certain rayon de recherche

FIGURE 3.1 – Interfaces avec la carte

Chapitre 4

Web et tâches asynchrones

Tout ce qui suit démarre avec l'archive `Meetit-3.tgz`. Archivez votre travail précédent, décompresser l'archive `Meetit-3.tgz` et ouvrir le projet Android `Meetit-3`.

4.1 Exécuter une requête

Exercice 4.1. *Afficher la page professionnelle du personnage cliqué. Dans le modèle de données, chaque personnage possède un attribut mémorisant l'URL de sa page web professionnelle. L'objectif de cet exercice est d'abord de capturer l'action de cliquer sur le marqueur d'un personnage puis d'ouvrir un navigateur web chargeant la page web professionnelle associée à celui-ci.*

1. *Comprendre code suivant.*

```
Uri uriUrl = Uri.parse("http://www.google.com");
Intent launchBrowser = new Intent(Intent.ACTION_VIEW, uriUrl);
startActivity(launchBrowser);
```

2. *Préciser que le fragment* `MyMapFragment` *implante* `GoogleMap.OnMarkerClickListener`.
3. *Une fois que l'attribut* `mGoogleMap` *est instancié, ajouter l'instruction suivante, après en avoir compris le sens.*

```
mGoogleMap.setOnMarkerClickListener(this);
```

4. *Modifier le code de la méthode* `onMarkerClick` *pour que s'affiche la page web* `http://www.google.com` *lorsque l'utilisateur clique sur un marqueur.*
5. *Modifier les propriétés des marqueurs et le code de la méthode* `onMarkerClick` *pour que s'affiche la page professionnelle du personnage correspondant au marqueur.*

4.2 Exploiter un service web

Pour exploiter un service web il est nécessaire que ce service s'exécute en arrière plan, c'est à dire dans un autre thread que celui de l'interface utilisateur.

On utilise pour cela la classe abstraite `AsyncTask`. Sa méthode `doInBackground` contiendra le code pour effectuer ce travail en arrière plan. Cette méthode communiquera avec les interfaces (Fragment, Activity) à l'aide d'intentions qu'elle lancera et qui seront récupérées par celles-ci.

Exercice 4.2. *Création d'une classe Asynchrone.*

1. *Créer la classe* `NeighborAsyncTask` *qui étends* `AsyncTask` *et qui contient le code suivant.*

```
public class NeighborAsycncTask extends AsyncTask {
    Context mContext;
    Location lo = null;

    public NeighborAsycncTask(Context mContext, Location lo) {
        super();
```

```
        this.mContext = mContext;
        this.lo = lo;
    }

    @Override
    protected Object doInBackground(Object[] params) {
        String st = "json data";
        Intent i = new Intent(mContext.getResources().getString(R.string.key_neighbor_intent));
        i.putExtra(mContext.getResources().getString(R.string.key_neighbor_path),st);
        mContext.sendBroadcast(i);
        return null;
    }
}
```

— *Pourquoi avoir passé un paramètre de type* `Location` *au constructeur ?*
— *Pour l'instant que réalise la méthode* `doInBackground` *?*
— *Où allez vous placer le code dont l'exécution nécessite des ressources importantes ?*
— *Pour que le code précédent puisse être compilé, quelles chaînes de caractères statiques faut-il ajouter (et dans quel fichier) ?*

2. *Dans le Fragment* `MyMapFragment`*, déclarer l'attribut suivant* `neighborBR` *de type* `BroadcastRecei` *C'est lui qui recevra les éléments envoyés par* `sendBroadcast` *dans le code précédent.*
3. *Dans la méthode* `onCreate` *instancier* `neighborBR` *et demander à Studio d'implanter la méthode* `onReceive`*. Compléter cette méthode pour qu'elle affiche dans un Toast la chaîne écrite dans l'intention.*
4. *Dans la méthode* `onResume`*, activer le* `BroadcastReceiver` *comme suit.*

```
getContext().registerReceiver(neighborBR,
                              new IntentFilter(getResources().getString(R.string.key_neighbor_intent)));
```

5. *Dans la méthode* `onPause`*, désactiver le* `BroadcastReceiver` *comme suit.*

```
getContext().unregisterReceiver(neighborBR);
```

6. *Pourquoi l'activation et la desactivation n'ont pas le même nombre de paramètres.*
7. *Demander l'exécution de la tâche asynchrone à chaque mise à jour de la carte comme suite*

```
new NeighborAsycncTask(this.getContext().getApplicationContext(),null).execute();
```

Exercice 4.3. *Interroger un service web. L'objectif de cet exercice est de demander au service web « directions » de Google, quel est le plus court chemin entre deux points dont les coordonnées GPS sont fournies.*

1. *Comprendre la requête suivante et la saisir dans un navigateur.*

```
https://maps.googleapis.com/maps/api/directions/json?
  &mode=walking
  &origin=47.6421675,6.8365264
  &destination=47.643393,6.845571
  &key=%20AIzaSyDr9mrOmk2ObslA4Z0ntoG9kfO_DdofJiE
```

2. *Comprendre la réponse retournée, particulièrement la distance.*
3. *Ajouter les deux méthodes suivantes dans la classe* `NeighborAsycncTask`*.*

```
    private String getPathLength(Location lo,Location ld) {
        String r=null;
        try {
            String urls = "https://maps.googleapis.com/maps/api/directions/json?"
                    + "&mode=walking"
                    + "&origin=47.6421675,6.8365264"
                    + "&destination=47.643393,6.845571"
                    + "&key= AIzaSyDr9mrOmk2ObslA4Z0ntoG9kfO_DdofJiE";
            URL url = new URL(urls);
            URLConnection conn = url.openConnection();
            conn.setDoOutput(true);
            OutputStreamWriter wr = new OutputStreamWriter(conn.getOutputStream());
            wr.flush();
            r = convertStreamToString(conn.getInputStream());
        } catch (Exception e) {
        }
        return r;
    }

    private String convertStreamToString(InputStream in){
```

```
        String reponse =null;
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        byte[] buffer = new byte[1024];
        try {
            for (int count; (count = in.read(buffer)) != -1; ) {
                out.write(buffer, 0, count);
            }
            byte[] response = out.toByteArray();
            reponse= new String(response, "UTF-8");
        } catch (Exception e){
        }
        return  reponse;
    }
```

4. *Comprendre les instructions de* `getPathLength` *et uniquement l'objectif de* `convertStreamToStrin` *Constater notamment qu'on ne se sert pas du paramètre de* `getPathLength`*.*
5. *Modifier la méthode de* `doInBackground` *pour qu'elle appelle* `getPathLength` *et qu'elle retourne non plus la chaîne "json data", mais celle renvoyée par* `getPathLength`*.*
6. *Modifier le code pour que la localisation de départ (*`origin`*) soit celle de l'utilisateur.*

Exercice 4.4. *Classer les personnage en fonction de leur distance à l'utilisateur. L'objectif de cet exercice est de dire à l'utilisateur quel personnage est le plus proche de lui (par l'itinéraire piéton le plus court). Pour cela, l'application doit interpréter les fichiers JSON renvoyés par le service "directions" de Google.*

1. *Ajouter la méthode suivante à la classe* `NeighborAsycncTask`*. Elle interprète la chaîne de caractères passée en paramètre correspondant au fichier JSON et extrait de celle-ci la longueur (en mètres) du chemin le plus court. Comprendre toutes les lignes de cette méthode en analysant le fichier JSON correspondant.*

```
private int interprete_json_file(String jsontext){
        int dist=-1;
        try {
            JSONObject racine = new JSONObject(jsontext);
            JSONArray routes = racine.getJSONArray("routes");
            JSONObject ob1 = routes.getJSONObject(0);
            JSONArray legs = ob1.getJSONArray("legs");
            JSONObject ob2 = legs.getJSONObject(0);
            JSONObject distance = ob2.getJSONObject("distance");
            dist = distance.getInt("value");

        }catch (JSONException e) {
            e.printStackTrace();
        }
        return dist;
}
```

2. *Utiliser cette méthode pour évaluer la distance entre deux points quelconques.*
3. *Dans la méthodes* `doInBackground`*, de quelles variables a-t-on besoin pour trouver le chemin dont la longueur est la plus petite possible ? Modifier la méthode en conséquence ainsi que la ligne qui invoque l'exécution de la tâche asynchrone.*