

Développement Android

Jean-François COUCHOT

couchot@femto-st.fr

18 janvier 2019

Table des matières

1	Projet Android et IHM	2
1.1	Projet Android. Organisation d'interface	2
1.2	Ressources	2
1.3	Activités et Fragments	3
1.4	Une liste de Fragments avec RecyclerView	4
1.4.1	Un objet Character et son fragment pour chaque personnage	5
1.4.2	Des objets pour afficher ces vues	5
1.4.3	Le Fragment MyDataFragment	6
1.5	Paramètres	7
2	Sauvegarde de données	9
2.1	Les chaînes de caractères constantes	9
2.2	Les préférences	9
2.2.1	Les types des valeurs saisies dans les Settings	9
2.2.2	Extraction des valeurs des Settings depuis des SharedPreferences	10
2.2.3	Modification d'une valeur stockée dans des SharedPreferences	10
2.3	Les bases de données	11
2.3.1	Configuration du projet pour utiliser Room	11
2.3.2	Générer les éléments de la base de données à partir d'annotations	11
2.3.3	Générer la base de données	12
2.3.4	Exploiter la base de données dans MyDataFragment	12
2.3.5	Gestion des images dans la base	13
3	Cartographie et localisation	15
3.1	Affichage d'une carte Google Map	15
3.2	Modifier quelques paramètres de la cartes	15
3.3	Les autorisations	16
3.4	Récupérer régulièrement des coordonnées GPS	17
4	Web et tâches asynchrones	20
4.1	Exécuter une requête	20
4.2	Exploiter un service web	20

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 ».

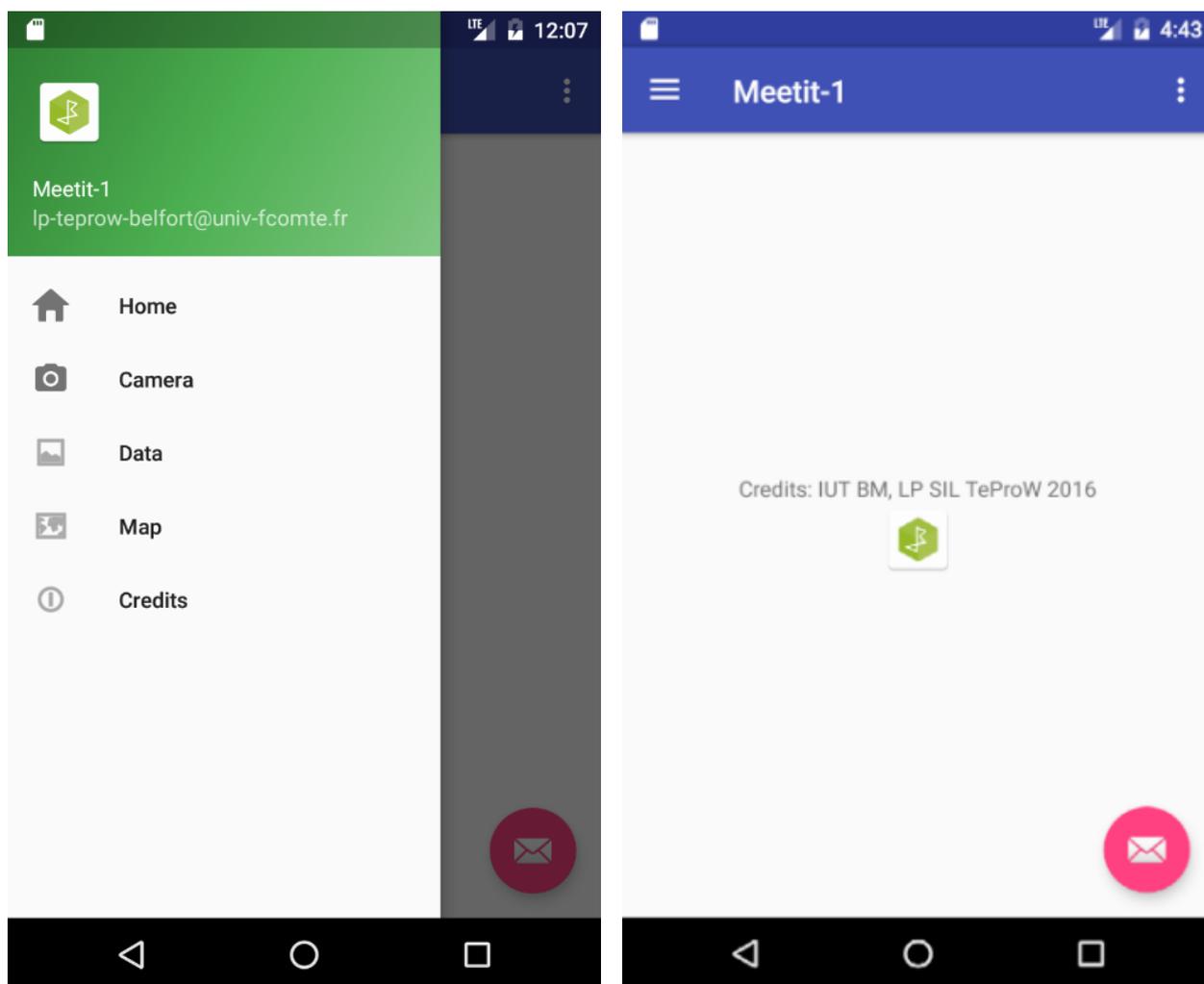
1.2 Ressources

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 `onNavigationItemSelectedListener`.



(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`. Constaté 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.onNavigationItemSelectedListener`.
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 Une liste de Fragments avec `RecyclerView`

Il est souvent intéressant d'afficher des données sous la forme d'une liste. Lorsque la liste est longue, pour des raisons d'efficacité, le système ne devrait créer que les vues visibles à l'écran. Lorsque que l'on fait défiler les vues, celles qui disparaissent devraient être « recyclées » pour être réutilisées. C'est le principe des `RecyclerView`.

Les personnages de l'application vont être affichés dans le fragment `MyDataFragment`. Ils seront extraits d'une base de données au chapitre suivant. Dans cette section on ne va s'intéresser qu'à la partie affichage pour laisser le stockage des données au prochain chapitre.

1.4.1 Un objet `Character` et son fragment pour chaque personnage

Exercice 1.6. Construction du modèle et de sa vue

1. Construire la classe `Character` avec les attributs privés suivants :
 - `Int uid` //ce sera la clef primaire de la base de données,
 - `String firstname`,
 - `String familyname`,
 - `String weblink`,
 - `float latitude`,
 - `float longitude`,
 - `String bmpath`.
2. Générer les getters, les setters et 2 constructeur : le premier prendra tous les attributs tandis que le second les prendra tous, sauf `uid`.
3. Construire un gabarit nommé `fragment_character` permettant l’affichage d’un personnage comme donné à la figure 1.2.
4. Fixer les identifiants des vues comme suit :
 - `@+id/image_view_character_picture` pour l’`ImageView` qui contient la photo du personnage ;
 - `@+id/text_view_character_first_name` pour le `TextView` contenant le prénom ;
 - `@+id/text_view_character_familiy_name` pour le `TextView` contenant le nom ;
 - `@+id/text_view_character_latitude` pour le `TextView` contenant la latitude ;
 - `@+id/text_view_character_longitude` pour le `TextView` contenant la longitude.



FIGURE 1.2 – Le gabarit des personnages

1.4.2 Des objets pour afficher ces vues

Chaque personnage va être affiché au moyen d’un `RecyclerView.ViewHolder`.

Exercice 1.7. Un conteneur de vue dédié pour chaque personnage

1. Construire la classe `CharacterViewHolder` qui étend `RecyclerView.ViewHolder` en insérant le code suivant :

```
public class CharacterViewHolder extends RecyclerView.ViewHolder {
    public TextView firstNameView, familyNameView, latitudeView, longitudeView;
    public ImageView pictureView;

    public CharacterViewHolder(View itemView) {
```

```

    super(itemView);
    firstNameView = (TextView) itemView.findViewById(R.id.text_view_character_first_name);
    familyNameView = (TextView) itemView.findViewById(R.id.text_view_character_familiy_name);
    latitudeView = (TextView) itemView.findViewById(R.id.text_view_character_latitude);
    longitudeView = (TextView) itemView.findViewById(R.id.text_view_character_longitude);
    pictureView = (ImageView) itemView.findViewById(R.id.image_view_character_picture);
}
}

```

2. Quel est l'objectif d'un tel objet ?

Il faut construire un adaptateur (`RecyclerView.Adapter`) pour afficher uniquement les vues visibles de la liste de tous les personnages.

Exercice 1.8. Un adaptateur pour la liste des personnage

1. Construire la classe `ChararacterListAdapter` qui étend `RecyclerView.Adapter<CharacterViewHolder>` et demander à studio d'implanter les méthodes requises.
2. ajouter l'attribut `List<Character> characterList` et un constructeur permettant d'initialiser cet attribut.
3. Que contiendra cet attribut ?
4. Remplacer la méthode vide `onCreateViewHolder(ViewGroup parent, int viewType)` par le code

```

public CharacterViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    View itemView = LayoutInflater.from(parent.getContext())
        .inflate(R.layout.fragment_character, parent, false);
    return new CharacterViewHolder(itemView);
}

```

Cette méthode construit le conteneur de la vue du personnage et le retourne.

5. Modifier le code de la méthode `getItemCount()` pour qu'il retourne la taille de la liste des personnages.
6. Dans la méthode `onBindViewHolder(CharacterViewHolder holder, int position)`
 - (a) Récupérer le personnage `Character teacher` de la liste `characterList` situé à l'indice `position`
 - (b) Modifier le conteneur reçu en paramètre en fixant son prénom avec celui du personnage récupéré à la question précédente :

```
holder.firstNameView.setText(teacher.getFirstname());
```
 - (c) Faire de même avec toutes les autres vues du conteneur.

1.4.3 Le Fragment MyDataFragment

Dans le Fragment `MyDataFragment` il ne reste plus qu'à

- créer une `RecyclerView` dans le gabarit qui contiendra la liste déroulante et
- modifier le créateur de vue de ce fragment pour lui dire comment remplir correctement cette vue.

Exercice 1.9. Une RecyclerView en plus

1. Ajouter le code suivant dans le gabarit de `MyDataFragment`.

```

<android.support.v7.widget.RecyclerView
    android:id="@+id/character_recycler_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />

```

2. Laisser Studio modifier le `build.gradle` pour importer la bibliothèque `com.android.support:recyclerview-v7`.

Exercice 1.10. Paraméter la RecyclerView de MyDataFragment

1. Ajouter les attributs suivants dans MyDataFragment

```
— private List<Character> characterList = new ArrayList<>();  
— private RecyclerView recyclerView;
```

2. Modifier la méthode onCreateView comme suit :

```
@Override  
public View onCreateView(LayoutInflater inflater, ViewGroup container,  
                          Bundle savedInstanceState) {  
    View v = inflater.inflate(R.layout.fragment_my_data, container, false);  
  
    recyclerView = v.findViewById(R.id.character_recycler_view);  
    recyclerView.setLayoutManager(new LinearLayoutManager(getActivity().getApplicationContext()));  
    recyclerView.setItemAnimator(new DefaultItemAnimator());  
    recyclerView.addItemDecoration(new DividerItemDecoration(getActivity(),  
                                                              LinearLayoutManager.VERTICAL));  
    recyclerView.setAdapter(new CharacterListAdapter(characterList));  
  
    prepareCharacterData();  
  
    return v;  
}
```

3. Comprendre chaque ligne du code précédent.

4. Développer la méthode prepareCharacterData qui ajoute trois personnages dans la liste characterList selon le modèles suivant :

```
characterList.add(new Character("Stéphane", "Domas",  
                               "http://info.iut-bm.univ-fcomte.fr/staff/sdomas/",  
                               47.6387143f, 6.8370225f,  
                               "domas.png");
```

A la fin de cette section, vous devriez avoir compris les bases de RecyclerView avec ses deux parties RecyclerView.ViewHolder et RecyclerView.Adapter.

1.5 Paramètres

Exercice 1.11. *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 1.3(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. Constaté l'ajout :

```
— de la classe abstraite AppCompatActivity 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;  
}
```

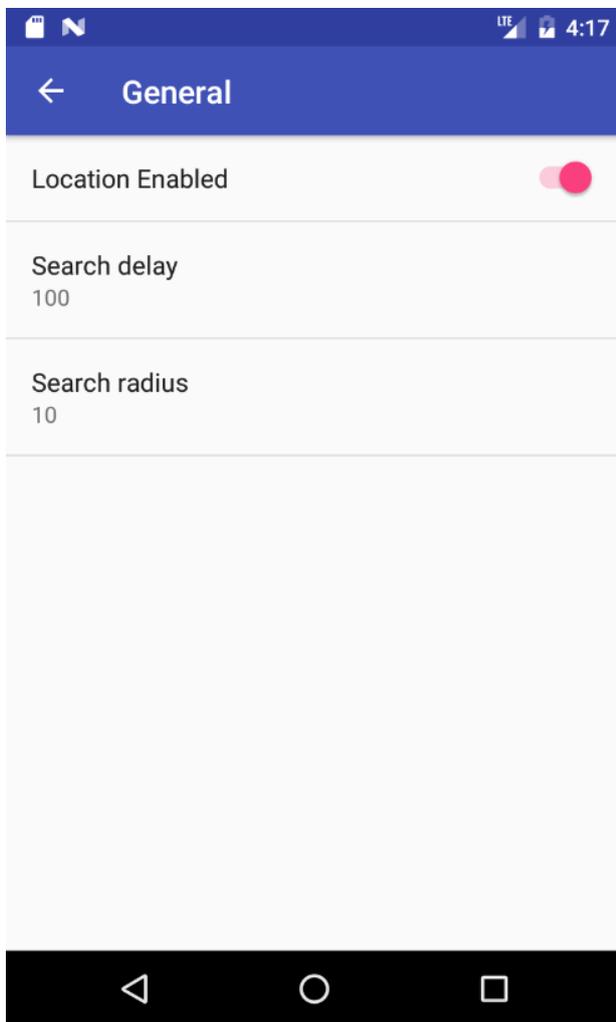
Exercice 1.12. *Modification des préférences par défaut.*

1. A l'aide de Studio, remplir l'élément PreferenceScreen du fichier `pref_general.xml` comme présenté à la figure figure 1.3. 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.

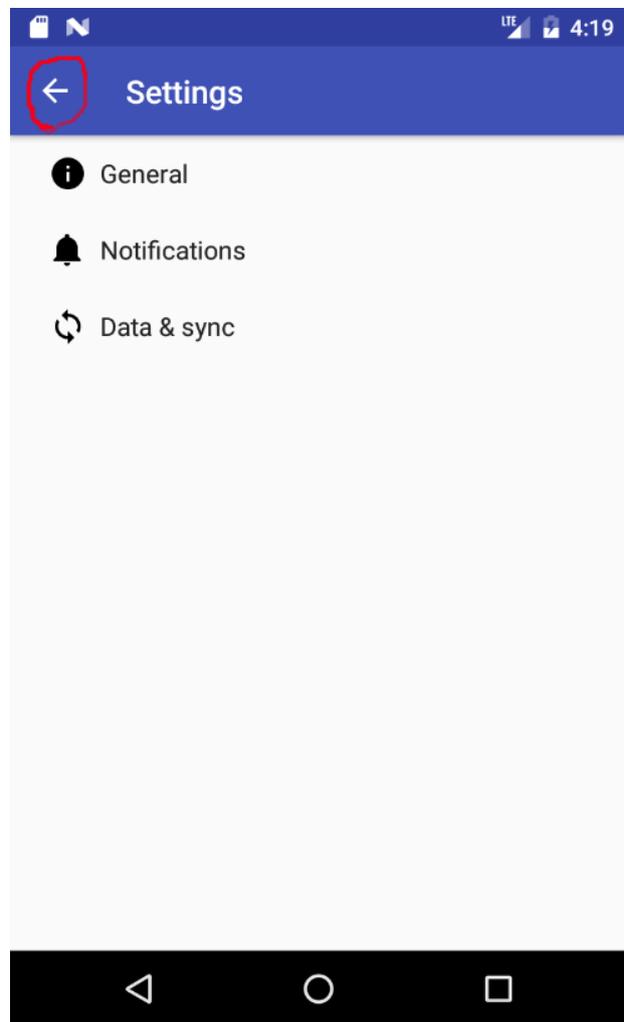
Exercice 1.13. Correction d'un bug dans les préférences.

Constater qu'appuyer sur le bouton ← de la barre de Settings ne permet pas de revenir à l'activité principale (cf figure 1.3(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) Section générale



(b) Bouton ← problématique

FIGURE 1.3 – Interfaces de Paramètres

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.

Chapitre 2

Sauvegarde de données

Tout ce qui suit démarre avec l'archive `Meetit-1.zip`. Archivez votre travail précédent, décompressez l'archive `Meetit-1.zip` 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 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 booléen. 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.

2.2.3 Modification d'une valeur stockée dans des SharedPreferences

On peut modifier toutes les valeurs qui sont stockées dans des préférences partagées. Pour cela il suffit

1. Récupérer les préférences partagées :

```
SharedPreferences sp = PreferenceManager.getDefaultSharedPreferences(getActivity());
```

2. Récupérer un éditeur pour préférences, de définir la valeur associée à la clef "MA_CLEF" à l'aide d'un `putX` et de valider comme suit :

```
sp.edit().putBoolean("MA_CLEF", true).commit();
```

Naturellement la clef devrait être mémorisée dans le fichier `strings.xml` et récupérée à l'aide de `getResources().getString(R.id.XXXXX)` plutôt qu'en dur ("MA_CLEF") comme ci-dessus.

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` ;
- comment modifier la valeur d'une préférence partagée.

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. Aujourd'hui, on ne définit plus la base de données à l'aide de scripts SQL, mais on utilise Room, qui, à partir d'annotations, va construire la base, les requêtes et retourner des objets.

2.3.1 Configuration du projet pour utiliser Room

Exercice 2.3. *Mise à jour des gradle pour intégrer Room.*

1. *Insérer le code suivant dans le fichier build.gradle (Module: app) à la fin du bloc de dépendances.*

```
implementation "android.arch.persistence.room:runtime:$rootProject.roomVersion"
annotationProcessor "android.arch.persistence.room:compiler:$rootProject.roomVersion"
```

La première ligne dit qu'on va utiliser la bibliothèque Room, tandis que la seconde précise qu'on va utiliser le générateur de code utilisant les annotations.

2. *Pour corriger l'erreur engendrée par l'insertion précédente, ajouter le code suivant dans le fichier build.gradle (Project: Meetit...) tout à la fin.*

```
ext { roomVersion = '1.1.1' }
```

3. *Pour pouvoir voir comment les tables sont construites par Room, insérer le code suivant dans le fichier build.gradle (Module: app) dans la partie DefaultConfig.*

```
javaCompileOptions {
    annotationProcessorOptions {
        arguments = ["room.schemaLocation": "$projectDir/schemas.toString()"]
    }
}
```

2.3.2 Générer les éléments de la base de données à partir d'annotations

Dans cette application, nous n'aurons qu'une seule table nommée characters. Celle-ci va être générée automatiquement à partir de la classe Character déjà construite, mais qu'on va modifier.

Exercice 2.4. *Création de la table characters à partir de la classe Character.*

1. *Ajouter l'annotation @Entity devant la déclaration de la classe Character comme suit :*

```
@Entity(tableName = "characters")
public class Character {
```

2. *Spécifier que l'attribut uid est la clef primaire et qu'elle sera générée automatiquement comme suit :*

```
@PrimaryKey(autoGenerate = true)
private int uid;
```

3. *Pour chacun des attributs, demander à ce qu'une colonne dans la table soit générée. Par exemple, pour le prénom, cela donnerait :*

```
@ColumnInfo(name = "firstname")
private String firstname;
```

4. *L'uid ne sert que pour définir une clef primaire dans la base de données. Il n'est donc pas utile que les objets que l'on construira à partir de cette base de données fixent cet attribut. On mettra donc @Ignore devant ce constructeur*

```
@Ignore
public Character(int uid, String firstname, String familyname, ..., String bmpath) {
...}
```

On construit ensuite une interface qui effectue les requêtes SQL en invoquant des méthodes et qui retourne les résultats sous la forme d'objets. C'est un accès aux données au travers d'objet DAO (data access object).

Exercice 2.5. *Création de l'interface DAO effectuant les requêtes.*

1. *Créer l'interface CharacterDAO comme suit.*

```

@Dao
public interface CharacterDAO {
    @Query("SELECT * FROM characters")
    List<Character> getAllCharaters();

    @Insert
    void insertCharacters(Character... characters);
}

```

2. *Que va-t-il se passer lorsque la méthode `getAllCharaters()` va être invoquée ?*
3. *Combien de paramètres prend la méthode `insertCharacters(Character... characters)` ? Que va-t-il se passer si l'on exécute `characterDAO.insertCharacters(c1, c2, c3)` ?*

2.3.3 Générer la base de données

On va engendrer automatiquement une unique instance de la base de données ne contenant que la table définie par la classe `Character` et contenant un attribut de type `CharacterDAO` pour réaliser les requêtes.

Exercice 2.6. *La classe `AppDatabase` qui étend `RoomDatabase`.*

1. *Construire la classe `AppDatabase` qui étend `RoomDatabase`. La rendre abstraite pour qu'elle ne soit pas instanciée.*
2. *Ajouter l'annotation suivante à cette classe qui stipule qu'elle contient une unique table engendrée par la classe `Character` :*

```

@Database(entities = {Character.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {

```

3. *Dans l'application, il n'y aura qu'une seule instance de la base. Pour cela, le patron de Singleton va être utilisé. Ajouter le code suivant et le comprendre.*

```

private static volatile AppDatabase INSTANCE=null;
static AppDatabase getDatabase(final Context context) {
    if (INSTANCE == null) {
        synchronized (AppDatabase.class) {
            if (INSTANCE == null) {
                INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                    AppDatabase.class, "character_database")
                    .build();
            }
        }
    }
    return INSTANCE;
}

```

4. *Ajouter dans cette classe la méthode abstraite*

```

public abstract CharacterDAO characterDAO()

```

5. *Que se passerait-il si l'on exécutait*

```

((AppDatabase.getDatabase(getContext())).characterDAO()).getAllCharaters();

```

dans un fragment ou une activité ?

2.3.4 Exploiter la base de données dans `MyDataFragment`

L'objectif va être de remplir la liste `characterList` (après une insertion dans la base et une extraction de celle-ci) et de mettre à jour la `recyclerView` avec cette nouvelle liste. Les accès à la base de données peuvent bloquer l'interface utilisateur. Ils ne peuvent pas être exécuté dans le thread de celle-ci. On va utiliser pour cela des tâches asynchrones (`AsyncTask`) pour que ceci soit exécuté en arrière-plan.

Commencer par commenter la ligne

```

//recyclerView.setAdapter(new ChracterListAdapter(characterList));

```

de la méthode `MyDataFragment.onCreateView(...)`.

Exercice 2.7. *Insérer des données à l'initialisation avec une tâche asynchrone.*

1. Ajouter les attributs privés `AppDatabase appdb` et `CharacterDAO characterDAO` dans `MyDataFragment`. A quoi ces deux attributs vont-ils servir ?
2. Créer la classe privée `InsertAsyncTask` interne à `MyDataFragment` en copiant le code ci-dessous.

```
private class InsertAsyncTask extends AsyncTask<Character, Void, Void> {
    private CharacterDAO dao;
    InsertAsyncTask(CharacterDAO dao) {
        this.dao = dao;
    }

    @Override
    protected Void doInBackground(final Character... params) {
        for (Character c : params) {
            this.dao.insertCharacters(c);
        }
        return null;
    }
}
```

Cette classe prend en paramètre de construction un objet de type `CharacterDAO` qui est utilisé dans la méthode `doInBackground` pour effectuer la requête d'insertion en arrière-plan. Cette méthode reçoit un tableau de `Character`.

3. Remplacer les trois instructions `characterList.add(c1); characterList.add(c2); characterList.add(c3);` de la méthode `prepareCharacterData()` par la demande d'exécution de la tâche asynchrone d'insertion.

```
(new InsertAsyncTask(characterDAO)).execute(c1, c2, c3);
```

4. Mettre un point d'arrêt sur l'instruction `this.dao.insertCharacters(c)` et constater que celle-ci est bien invoquée à chaque chargement du fragment `MyDataFragment`. Cela est problématique car à chaque appel, on ajoute à nouveau les trois mêmes personnages dans la base.
5. Utiliser les préférences partagées pour n'invoquer la méthode `prepareCharacterData()` que la première fois que l'application est exécutée.

Exercice 2.8. Récupérer les personnages de la base et mettre à jour la `recyclerView`.

1. Créer une tâche asynchrone nommée `GetAllCharactersAsyncTask` (comme une classe privée à la classe `MyDataFragment`) qui récupère sous la forme de liste tous les personnages dans sa méthode `doInBackground`.
2. Ajouter à cette classe la méthode `onPostExecute(Void voids)` qui va s'exécuter une fois la méthode `doInBackground` terminée. Dans cette méthode, mettre à jour la `characterList` et la `recyclerView`.
3. Dans la méthode `MyDataFragment.onCreateView`, immédiatement après avoir demandé l'initialisation de la base de données, demander l'exécution de la tâche asynchrone `GetAllCharactersAsyncTask`.

2.3.5 Gestion des images dans la base

Exercice 2.9. Un dossier `Assets` pour des données initiales. Les trois 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.

Exercice 2.10. Modifier l'adaptateur pour qu'il aille chercher les images.

1. Quelle classe et quelle méthode faut-il principalement modifier pour définir quelle image charger pour chaque personnage ?

2. *A la fin de la méthode `CharacterListAdapter.onBindViewHolder`, ajouter le code erroné suivant :*

```
try {
    String imageName=teacher.getBmppath();           //1
    InputStream ims = ctxt.getAssets().open(imageName); //2
    Drawable d = Drawable.createFromStream(ims, null); //3
    holder.imageView.setImageDrawable(d);           //4
    ims .close();                                   //5
}
catch(IOException ex) { return;}

```

3. *Comprendre les 5 instructions.*
4. *De quel type semble être l'attribut `ctxt` ? Ajouter cet attribut à la classe et modifier le constructeur `CharacterListAdapter(List<Character> characterList, Context ctxt)` pour l'instancier.*
5. *Modifier l'unique appel à ce constructeur en conséquence.*

Chapitre 3

Cartographie et localisation

Tout ce qui suit démarre avec l'archive `Meetit-2.tgz`. Archivez votre travail précédent, décompressez l'archive `Meetit-2.tgz` et ouvrez 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 fragment, 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. Constaté 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. Constaté 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 setLocationParameters 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();
    setLocationParameters();
    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.

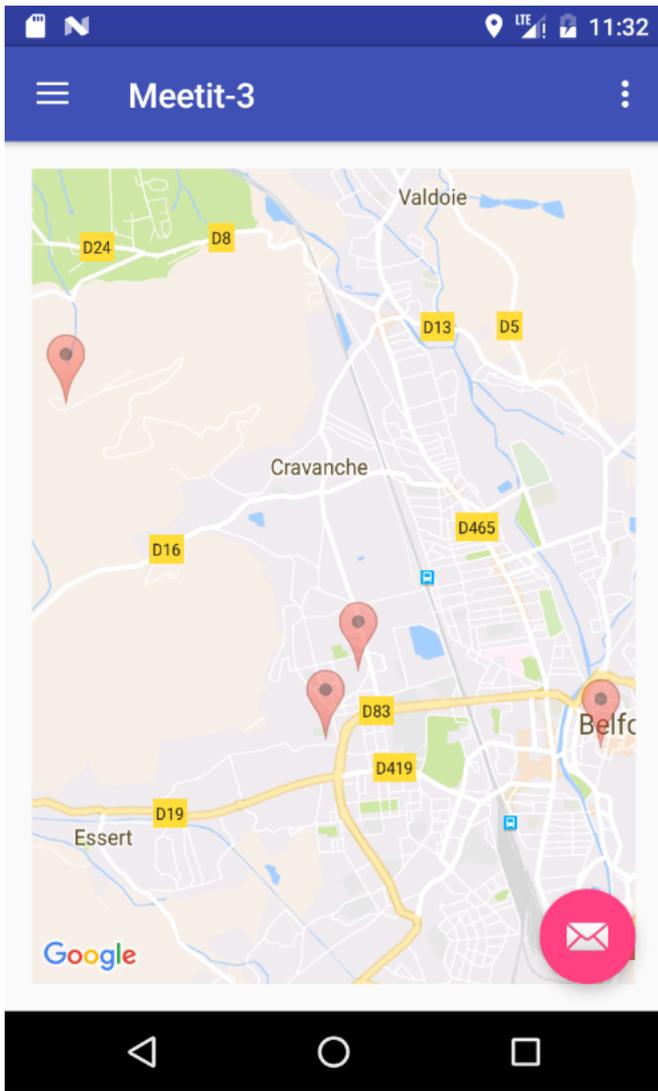
7. Désactiver la localisation. Changer les coordonnées GPS régulièrement dans la fenêtre de contrôles et constater le bon respect du délai.

Exercice 3.10. Les personnages dans un certain rayon.

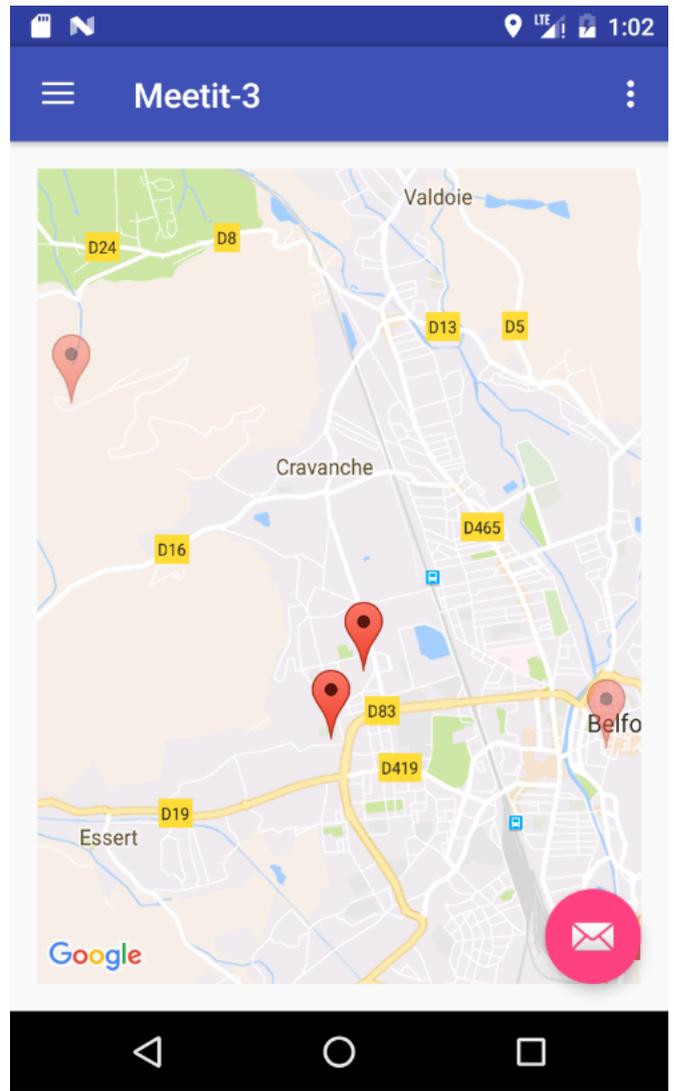
L'objectif de cet exercice est d'informer au mieux l'utilisateur de la présence de personnage(s) dans le rayon précisé dans les préférences de l'application.

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'ils sont à l'intérieur de ce rayon et à 50% de transparence sinon, comme illustré à la figure 3.1(b).



(a) Un personnage \rightsquigarrow 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, décompressez l'archive `Meetit-3.tgz` et ouvrez 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 étend `AsyncTask` et qui contient le code suivant.

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

    public NeighborAsyncTask(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 `BroadcastReceiver`. 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 désactivation 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 NeighborAsyncTask(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 `NeighborAsyncTask`.

```

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 `convertStreamToString`. Constaté 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 `NeighborAsyncTask`. 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 jsoncontext){
    int dist=-1;
    try {
        JSONObject racine = new JSONObject(jsoncontext);
        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.*