Compte rendu :
Application ToDoList

Introduction

L'objectif de cette séquence est de créer une application de ToDoList, permettant de relier un utilisateur à l'ensemble de ses todo listes, définies elles-mêmes par une suite d'items à effectuer. Cela implique de créer à la fois des activités, des layout et des classes destinées au traitement des informations.

Analyse

Description de la structure

Pour cette application, j'ai donc utilisé 3 activités principales :

- **MainActivity** : l'utilisateur entre son pseudo et peut avoir accès à ses listes
- **CheckListActivity** : l'utilisateur voit ses listes, peut choisir d'en ajouter ou supprimer une, un clic sur une liste donne accès à la liste des items
- **ShowListActivity** : l'utilisateur peut voir les items de sa liste ainsi que leur état (fait ou non), il peut ajouter ou supprimer une liste, ou bien mettre à jour son état

Quatre autres activités ont également été implémentées :

- **SettingsActivity** : accessible depuis la **MainActivity**, l'utilisateur peut choisir la langue de l'application, modifier le pseudo par défaut ou supprimer l'historique des profils créés.
- **GenericActivity** : hérite de **AppCompatActivty**, cette activité regroupe toutes les fonctions utilisées dans les trois activités principales afin de simplifier la lisibilité du code (cette activité ne possède aucun l'atout associé). Les trois activités principales héritent de cette classe
- **ListAdapter** : permet de gérer le RecyclerView associé à l'affichage des listes todo, contient une classe ListViewHolder
- **ItemAdapter** : permet de gérer le RecyclerView associé à l'affichage des items, contient une classe ItemViewHolder

Afin de gérer le traitement des informations, j'ai également crée trois classes Java, suivant le diagramme UML donné dans l'énoncé :

- **ProfilListeToDo** : création et gestion des profils, définis par un pseudo et une liste de **ListeToDo**
- **ListeToDo** : création et gestion des listes todo, définies par un titre et une liste d'**ItemToDo**
- **ItemToDo** : création et gestion des items, définis par une description et un booléen représentant leur état (fait ou non fait)

L'activité **GenericActivity** contient également une classe de traitement **ListeDesProfils** possédant un unique attribut correspondant à la liste de tous les profils gardés en mémoire par l'application.

Points particuliers

1. GenericActivity

L'objectif de GenericActivity est de centraliser les fonctions de traitement dont ont besoin les trois activités principales. Cela permet de réduire la quantité de code en ne créant pas de doublons de fonctions.

GenericActivity possède différentes fonctions :

- les fonctions associées au fichier JSON : serialiser() et désérialiser()
- les fonctions associées à la recherche d'une instance de **ProfilListeToDo** ou **ListeToDo** à partir d'un pseudo ou d'un identifiant
- les fonctions d'actualisation après modification : actualisation de la langue de l'application, de l'instance de **ListeDesProfils** en cours en cas de modification d'un profil, ou de la liste des pseudos utilisés pour l'affichages dans **SettingsActivity**

GenericActivity possède également la classe interne **ListeDesProfils**. Cette classe a été crée pour être utilisée dans la désérialisation du fichier JSON : la fonction fromJson() transforme la chaine de caractère passée en paramètre en une instance de la classe **ListeDesProfils**, dont on peut ensuite récupérer l'attribut correspondant à la liste des ProfilListeToDo créés.

2. La gestion de la transmission d'information : JSON et SharedPreferences

Pour transmettre les informations d'une activité à une autre j'ai utilisé les SharedPreferences comme nous l'avions vu dans le cours. A chaque début d'activité, celle ci récupère les préférences qui lui sont utiles. A chaque changement d'activité, on met à jour les préférences.

```
pseudo = settings.getString( key: "pseudo", defValue: "");
titre = settings.getString( key: "titre", defValue: "");
idListe = settings.getInt( key: "idListe", defValue: -1);
JSONFile = settings.getString( key: "JSONFile", defValue: "");
```

```
editor.clear();
editor.putString("historique", afficherPseudos(JSONFile));
editor.putString("titre", titre);
editor.putString("pseudo", pseudo);
editor.putString("JSONFile", JSONFile);
editor.putString("langue", languageToLoad);
editor.commit();
```

Les préférences sont presque toutes des chaines de caractères. Pour transformer les informations d'instances Java à des chaines de caractères, j'ai implémenté deux fonctions de la **GenericActivity** permettant de serialiser ou désérialiser la classe **ListeDesProfils** en chaine de caractères au format JSON. De cette façon, on peut récupérer à chaque création d'activité les informations sous forme d'une liste des Profils afin de traiter plus facilement les modifications que fera l'utilisateur. Cette méthode m'a semblé être la plus simple avec la contrainte d'utiliser des SharedPreferences.

```
public String serialiserJSON(ListeDesProfils listeProfils) {
    GsonBuilder builder = new GsonBuilder();
    Gson gson = builder.create();
    String sJson = gson.toJson(listeProfils);
    return sJson;
}
```

```
public ListeDesProfils deserialiserJSON(String JSONFile) {
    final GsonBuilder builder = new GsonBuilder();
    final Gson gson = builder.create();
    ListeDesProfils listeProfils = gson.fromJson(JSONFile, ListeDesProfils.class);
    return listeProfils;
}
```

3. L'affichage et la gestion des RecyclerView

Comme nous l'avions vu avec M. Boukadir, j'ai mis au point des **RecyclerView** pour l'affichage des listes de todolistes et d'items. Pour cela, j'ai crée deux **Adapter** : un pour les listes et un pour les items. La séparation des deux permet de gérer plus facilement les actions sur le vues.

Pour la modification de la liste globale des profil en cas d'action de l'utilisateur (ajout, suppression, modification de la checkbox), j'ai implémenté des interfaces dans chaque **Adapter**. Ainsi, CheckListActivity et ShowListActivity implémentent elles-même les interfaces et peuvent agir directement sur la liste des profils en cas de clic.

```
interface ActionListenerListe {
    void onItemClickedListe(ListeToDo listeClicked);
    void onClickDeleteButtonListe(ListeToDo listeToDelete);
}
```

```
interface ActionListenerItem {
    void onItemClickedItem(ItemToDo itemClicked);
    void onClickDeleteButtonItem(ItemToDo itemToDelete);
    void onClickCheckButtonItem(ItemToDo itemToCheck);
}
```

4. SettingsActivity

L'activité **SettingsActivity** contient trois fonctionnalités principales. La première est de modifier le pseudo par défaut, c'est à dire celui qui sera proposé à l'utilisateur lors de son arrivée dans la **MainActivity**. Sans précision particulière de la part de l'utilisateur, il s'agira du dernier pseudo entré dans l'application.

La deuxième fonctionnalité est la modification de la langue de l'application. Pour cela j'ai crée une ListPreference (boutons de type radio), qui, lors d'un changement, actualise la langue de l'activité en cours selon la nouvelle valeur indiquée. Lors du clic, on envoie les nouvelles informations et on recharge la page SettingsActivity avec les nouveaux paramètres.

```
//Modification de la langue, lors du clic sur l'objet ListPrefrence
final ListPreference listeLangues = (ListPreference) findPreference( key: "langues");
listeLangues.setOnPreferenceChangeListener((preference, newValue) → {
        langue = newValue.toString();

        //Mise à jour des préférences avec la nouvelle langue choisie
        SharedPreferences newSettings = PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
        SharedPreferences.Editor newEditor = newSettings.edit();
        newEditor.clear();
        newEditor.putString("pseudo", pseudo);
        newEditor.putString("JSONFile", JSONFile);
        newEditor.putString("historique", historique);
        newEditor.putString("langue", langue);
        newEditor.commit();

        //Vers SettingsActivity
        Intent toSettingsActivity = new Intent( packageContext: SettingsActivity.this, SettingsActivity.class);
        startActivity(toSettingsActivity);
        return true;
});
```

A chaque ouverture d'activité, on appelle la fonction actualiserLangue() (définie à la fois dans **SettingsActivity** et dans **GenericActivity**) avant d'appeler le l'atout pour s'assurer que la langue affichée est bien celle qui a été placée en paramètre. Comme la langue choisie est contenue à tout moment dans le SharedPreferences, la langue appliquée lors du redémarrage de l'application correspond à la dernière qui avait été choisie.

```
public void actualiserLangue(String langue) {
    String languageToLoad = langue; //
    Locale locale = new Locale(languageToLoad);
    Locale.setDefault(locale);
    Configuration config = new Configuration();
    config.locale = locale;
    getBaseContext().getResources().updateConfiguration(config,
            getBaseContext().getResources().getDisplayMetrics());
}
```

La troisième fonctionnalité est l'affichage de la liste des pseudos utilisés et la suppression de l'historique, c'est à dire de l'ensemble des informations que l'application retenait en mémoire (profils créés et leurs listes associées). La liste des pseudos est récupérée via les SharedPreferences, et avait précédemment été crée avec la fonction afficherPseudos() qui se trouve dans **GenericActivity**.

```
public String afficherPseudos(String JSONFile) {
    ListeDesProfils contenuJSON = deserialiserJSON(JSONFile);
    if (! JSONFile.equals("")) {
        ArrayList<ProfilListeToDo> listeProfils = contenuJSON.getListeProfils();
        String listePseudos = "";
        for (int i =0 ; i < listeProfils.size() ; i++){
            listePseudos += listeProfils.get(i).getLogin() + " ; ";
        }
        return listePseudos;
    }
    else {
        return "Aucun pseudo";
    }
}
```

Pour la suppression de l'historique, on place un **Listener** sur la PreferenceView appelée « Supprimer l'historique ». Dans ce **Listener**, on remplace les valeurs des paramètres (pseudo, JSONFile, historique) par des chaines de caractères vides. On enregistre ces nouvelles préférences dans le SharedPréférence, afin qu'elles soient reconnues comme nouvelles préférences, même en cas de redémarrage de l'application par l'utilisateur.

```
Preference suppressHistoric = findPreference( key: "suppress_historic");
suppressHistoric.setOnPreferenceClickListener((preference) → {
        historique = "Historique vide";
        JSONFile = "";
        pseudo = "";

        //Mise à jour des préférences
        SharedPreferences newSettings = PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
        SharedPreferences.Editor newEditor = newSettings.edit();
        newEditor.clear();
        newEditor.putString("pseudo", pseudo);
        newEditor.putString("JSONFile", JSONFile);
        newEditor.putString("historique", historique);
        newEditor.putString("langue", langue);
        newEditor.commit();
        afficherHistoric.setSummary("");
        return true;
});
```

Conclusion

L'application développée permet d'accomplir des actions de base comme ajouter, mettre à jour ou supprimer des listes mais n'est pas optimisée en terme d'expérience utilisateur. De plus, le code lui même ou la gestion de préférences par exemple ne sont pas complètement optimisés : PreferenceActivity est déprécié depuis l'API 16, au profit de l'utilisation de Fragments.

Perspectives

Plusieurs améliorations sont possibles pour cette application. En effet, elle aurait tout d'abord pu être implémentée avec des fragments et des bundle et non pas une PreferenceActivity.
On aurait également pu sauvegarder les informations dans un fichier au lieu d'utiliser uniquement une chaine de caractère, cela aurait permis d'y avoir accès en dehors de l'application.

Ensuite, la gestion du mode landscape du téléphone aurait pu être controlée.
De plus, étant donné que l'utilisateur entre son pseudo, on aurait également pu imaginer un onglet « Compte » dans le menu, ou l'utilisateur aurait accès à plusieurs informations comme une photo de profil, son nombre de listes total, la part d'items réalisés sur le nombre total d'items entrés, la langue par défaut de l'utilisateur, la dernière modification de ses listes, etc.
D'autres possibilités d'amélioration d'expérience utilisateur auraient pu être l'autocomplétion des pseudos, l'affichage d'un message de bienvenue à l'utilisateur qui entre son pseudo, ou encore la mise à jour automatique de l'affichage des listes selon leur dernière modification.

Bibliographie

Site : developer.android.com

- ListPreference : *https://developer.android.com/reference/android/preference/ListPreference*
- PreferenceActivity : *https://developer.android.com/reference/android/preference/PreferenceActivity*
- PreferenceScreen : *https://developer.android.com/reference/android/preference/PreferenceScreen*
- Settings : *https://developer.android.com/guide/topics/ui/settings*
- JSONObject : *https://developer.android.com/reference/org/json/JSONObject*

Site : stackoverflow.com

- OnClickListener : *https://stackoverflow.com/questions/26619044/listpreference-onclicklistener-takes-twice-to-execute*
- Manual language changes : *https://stackoverflow.com/questions/2900023/change-app-language-programmatically-in-android*

Site : developpez.net

- getApplicationContext() : *https://www.developpez.net/forums/d1526332/java/general-java/java-mobiles/android/getcontext-getbasecontext-getapplicationcontext/*
- SharedPreferences : *https://vogella.developpez.com/tutoriels/android/persitance-preferences-fichiers/*

Autres

- Internationalisation d'application : *https://www.supinfo.com/articles/single/460-internationalisation-une-application-mobile-android*
- toJSON, fromJSON : *https://www.rdocumentation.org/packages/jsonlite/versions/1.6/topics/toJSON%2C%20fromJSON* ; *https://www.mkyong.com/java/how-do-convert-java-object-to-from-json-format-gson-api/*
- Event Handler dans l'Adapter : *https://guides.codepath.com/android/Using-an-ArrayAdapter-with-ListView* ; *https://openclassrooms.com/fr/courses/4568576-recuperez-et-affichez-des-donnees-distantes/4893791-interagissez-avec-la-recyclerview*