Programmation Mobile et Réalité Augmentée
FAN Shiyu
TANG Qing

Compte-Rendu du TP1
Todo List

Introduction:

Les travails déjà réalisés:

- MainActivity
    1. permet de **saisir notre pseudo**. Si le pseudo est déjà saisi avant, on peut accéder à nos listes liés à ce pseudo, sinon ce pseudo sera créé et stocké, on va accéder à la page de listes.
    2. **Le dernier pseudo** saisi est automatiquement renseigné dans le champ de saisie.
    3. Un bouton de "**préférences**" nous permet d'accéder aux "préférences".
    4. Un bouton **OK** nous permet d'accéder à des listes-todo liés à pseudo choisi.

- SettingActivity
    1. **Afficher des pseudos** saisis auparavant
    2. Si on clique sur un de ces pseudos, on va pouvoir accéder à son compte et **afficher ses listes-todo**
    3. Permet de **supprimer un pseudo** par glissant l'écran à gauche

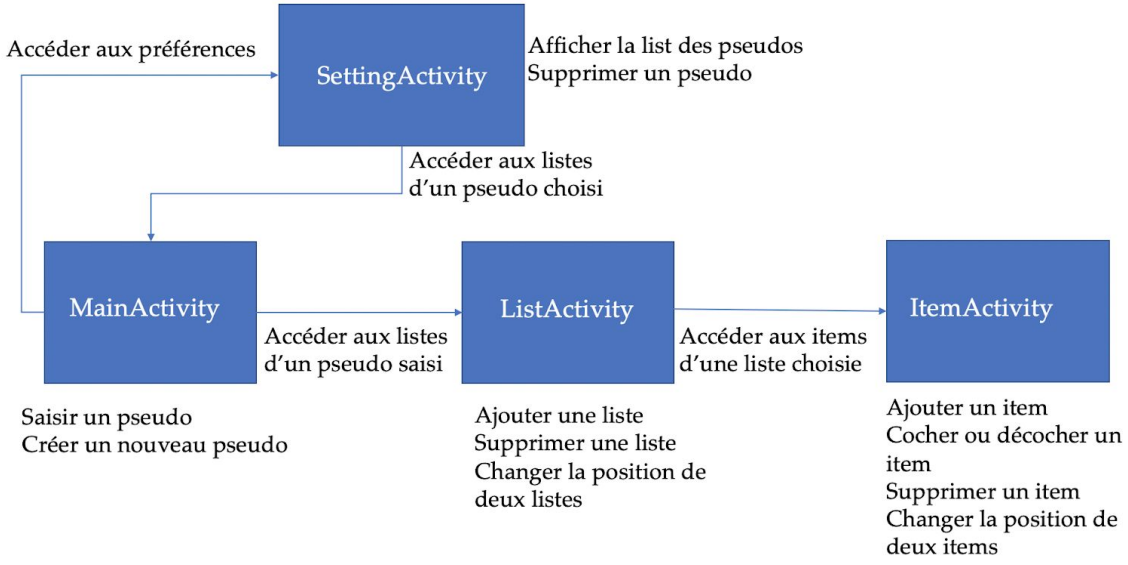
- ListActivity
    1. **Afficher la list des listes-todo** de l'utilisateur dont le pseudo a été saisi
    2. Lors du clic sur une liste, des **items** de cette liste **s'affichent**.
    3. Permet d'**ajouter une nouvelle liste** pour cet utilisateur
    4. Permet de **supprimer une liste** par glissant l'écran à gauche
    5. Permet de **changer la position** entre deux listes

- ItemActivity
    1. **Afficher les items des listes** de l'utilisateur dont le pseudo a été saisi
    2. Permet de **cocher ou décocher** un item
    3. permet d'**ajouter un nouvel item** dans cette liste
    4. Permet de **supprimer un item** par glissant l'écran à gauche
    5. Permet de **changer la position** entre deux listes

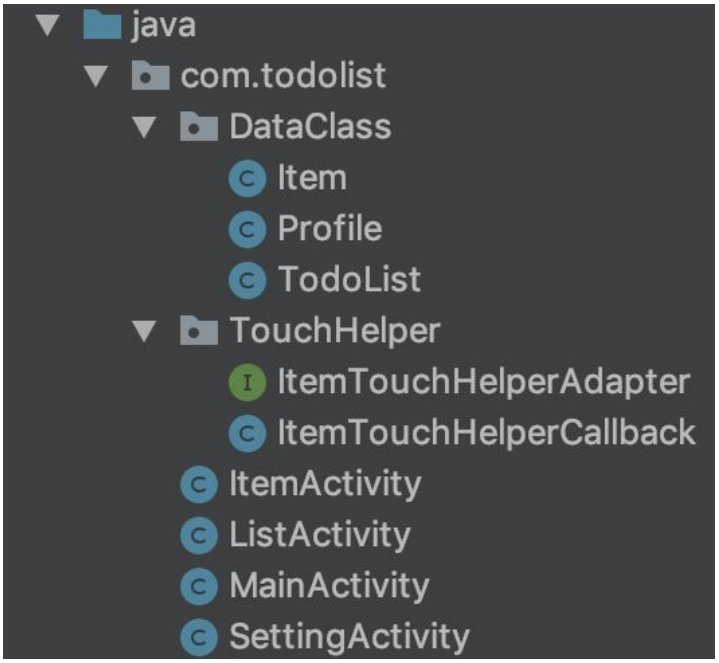
Analyses:

1. Analyse de TP:

On analyse d'abord des relations et connections entre chaque activité, et on fait une figure comme montré ci-dessous. Pour réaliser chaque travail demandé, on va devoir écrire et faire fonctionner plusieurs fonctions.



2. Analyse de fonction:



Structure de class:

Activities :
- MainActivity
- ListActivity
- ItemActivity
- SettingActivity

Data :
- Profile
- TodoList
- Item

TouchHelper :
- ItemTouchHelperAdapter
- ItemTouchHelperCallback

Les activity class gèrent les 4 écrans, les data class construisent la structure de donnée, et la class TouchHelper sert à supprimer et ré-ordonner les items.

MainActivity

```
public class MainActivity extends AppCompatActivity {
    EditText edt_pseudo;
    Button btn_pseudo;
    List<String> profiles;

    @Override
    protected void onCreate(Bundle savedInstanceState) {...}

    // Initialize the preference menu
    public boolean onCreateOptionsMenu(Menu menu) {...}

    // Show the profile selected by user when redirect to this activity
    @Override
    protected void onStart() {...}

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {...}

    // Open an existing list activity or create a new one
    // regarding if the pseudo exist or not in the preference
    public void openListActivity(View v) {...}

    // Function to save data as json file
    // Each Profile class has its own json file named by its login
    public void saveProfilData(Profile profile, String pseudo) {...}

    // Read data and return a Profile class by using its login
    public Profile readProfilData(String filename) {...}

    // Save logins and number as preference
    public void savePreference(List<String> profiles) {...}

    // Return an array list of profiles' logins;
    public List<String> readPreference() {...}
}
```

onCreate initialize l'activité et met la dernier pseudo dans le champs texte.
onCreateOptionsMenu et onOptionsItemSelected initialize le menu de préférence et permet de passer à l'activité setting quand le menu est cliqué.
onStart permet de vérifier si l'intent passé par setting activité existe ou pas, et le mettre dans le champs texte.
openListActivity est attaché avec le bouton "OK" pour ouvrir l'activité List et passer le profile sélectionné.

```java
// Function to save data as json file
// Each Profile class has its own json file named by its login
public void saveProfilData(Profile profile, String pseudo) {
    final GsonBuilder builder = new GsonBuilder();
    final Gson gson = builder.create();
    String fileContents = gson.toJson(profile);
    FileOutputStream fileOutputStream;

    try {
        fileOutputStream = openFileOutput(pseudo, Context.MODE_PRIVATE);
        fileOutputStream.write(fileContents.getBytes());
        fileOutputStream.close();
    } catch (FileNotFoundException e) {...} catch (IOException e) {...}
}

// Read data and return a Profile class by using its login
public Profile readProfilData(String filename) {
    StringBuilder jsonRead = new StringBuilder();
    Profile profile;
    final GsonBuilder builder = new GsonBuilder();
    final Gson gson = builder.create();
    try {
        FileInputStream inputStream;
        inputStream = openFileInput(filename);
        int content;
        while ((content = inputStream.read()) != -1) {...}
        inputStream.close();

        profile = gson.fromJson(jsonRead.toString(), Profile.class); // cast Profile

        return profile;
    } catch (FileNotFoundException e) {...} catch (IOException e) {...}
    return new Profile();
}
```

saveProfilData et readProfilData utilise les classes FileInputStream et FileOutputStream et gson pour écrire et lire les données stocké dans le format json. La structure de donnée est Profile -> List -> Item, les données sont stockées par profile, chaque profile est stocké dans un fichier dont le nom est son pseudo.

```java
// Save logins and number as preference
public void savePreference(List<String> profiles) {
    SharedPreferences preferences = getSharedPreferences( name: "profile", MODE_PRIVATE);
    SharedPreferences.Editor editor = preferences.edit();

    editor.putInt("profile_number", profiles.size());
    for (int i = 0; i < profiles.size(); i++) {
        editor.putString("" + i, profiles.get(i));
    }

    editor.apply();
    editor.commit();
}

// Return an array list of profiles' logins;
public List<String> readPreference() {
    SharedPreferences preferences = getSharedPreferences( name: "profile", MODE_PRIVATE);
    List<String> profiles = new ArrayList<>();
    int profile_number = preferences.getInt( key: "profile_number",  defValue: 0);
    for (int i = 0; i < profile_number; i++) {
        profiles.add(preferences.getString( key: "" + i,  defValue: ""));
    }
    return profiles;
}
```

savePreference et readPreference servent à stocker les données de comptes dans un fichier xml en utilisant la classe SharedPreferences. Le fichier xml contient les noms de fichier json et le nombre de compte.

ListActivity

```java
public class ListActivity extends AppCompatActivity {
    private Profile profile;
    EditText edt_list;
    Button btn_list;
    RecyclerView recyclerView;
    ListAdapter listAdapter;
    List<String> data_list;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {...}

    // Return a list of todolists' names in a profile class
    // to dispaly in the Recycler view
    private List<String> data_list(Profile profile) {...}

    // Function bind to the "OK" button as onClick event for a new list
    public void newList(View v) {...}

    class ListAdapter extends RecyclerView.Adapter<ListAdapter.ListViewHolder> implements ItemTouchHelperAdapter {...}

    public void saveProfilData(Profile profile, String pseudo) {...}

    public Profile readProfilData(String filename) {...}
}
```

data_list sert à lire les données et faire passer à l'adapteur.
newList est attaché avec le bouton "OK". Elle crée une nouvelle classe de TodoList à partir de l'entrée d'utilisateur et puis sauvegarde les données.
ListAdapter est une sous-class dans l'activité ListActivity qui gère l'affichage et l'événement de RecyclerView. Et elle hérite RecyclerView.Adapter et implémente l'interface ItemTouchHelperAdapter pour gérer la suppression et le déplacement. Chaque list est attaché avec la onClickListener et la fonction onClick qui ouvre l'activité Item.

ItemActivity

```
public class ItemActivity extends AppCompatActivity {
    private Profile profile;
    String list_name_selected;
    EditText edt_item;
    Button btn_item;
    RecyclerView recyclerView;
    ItemAdapter itemAdapter;
    List<DataEntity> data;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {...}

    // Class contains item name and check status
    class DataEntity {...}

    // Return a list of DataEntity in a profile.list class
    // to dispaly in the Recycler view
    private List<DataEntity> data(Profile profile, String list_name_selected) {...}

    // Function bind to the "OK" button as onClick event for a new item
    public void newItem(View v) {...}

    class ItemAdapter extends RecyclerView.Adapter<ItemAdapter.ItemViewHolder> implements ItemTouchHelperAdapter {...}

    public void saveProfilData(Profile profile, String pseudo) {...}

    public Profile readProfilData(String filename) {...}
}
```

DataEntity est une sous-classe qui contient le nom de item et l'état de checked.
ItemAdapter est une sous-classe qui ressemble à la classe ListAdapter. Chaque item est attaché avec la fonction onCheckedChanged qui permet de sauvegarder les modifications d'états de checked.

SettingActivity

```
public class SettingActivity extends AppCompatActivity {
    List<String> profiles;
    RecyclerView recyclerView;
    SettingAdapter settingAdapter;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {...}

    public void savePreference(List<String> profiles) {...}

    public List<String> readPreference() {...}

    class SettingAdapter extends RecyclerView.Adapter<SettingAdapter.SettingViewHolder> implements ItemTouchHelperAdapter {...}
}
```

SettingAdapter est une sous-classe qui ressemble aux autres adapteurs. Chaque profile est attaché avec la fonction onClick qui permet de passer le profile sélectionné à l'activité Main.

ItemTouchHelperCallback

```
public class ItemTouchHelperCallback extends ItemTouchHelper.Callback {
    private ItemTouchHelperAdapter myAdapter;

    public ItemTouchHelperCallback(ItemTouchHelperAdapter adapter) { myAdapter = adapter; }

    @Override
    public int getMovementFlags(@NonNull RecyclerView recyclerView,
                                @NonNull RecyclerView.ViewHolder viewHolder) {
        int dragFlags = ItemTouchHelper.UP | ItemTouchHelper.DOWN;
        int swipeFlags = ItemTouchHelper.LEFT;
        return makeMovementFlags(dragFlags, swipeFlags);
    }
```

```
    @Override
    public boolean onMove(@NonNull RecyclerView recyclerView,
                          @NonNull RecyclerView.ViewHolder viewHolder,
                          @NonNull RecyclerView.ViewHolder target) {
        myAdapter.onItemMove(viewHolder.getAdapterPosition(),target.getAdapterPosition());
        return true;
    }

    @Override
    public void onSwiped(@NonNull RecyclerView.ViewHolder viewHolder, int direction) {
        myAdapter.onItemDismiss(viewHolder.getAdapterPosition());
    }

    @Override
    public boolean isLongPressDragEnabled() { return super.isLongPressDragEnabled(); }

    @Override
    public boolean isItemViewSwipeEnabled() { return super.isItemViewSwipeEnabled(); }
}
```

Cette classe gère le mouvement de item dans le recyclerview.
getMovementFlags retourne la direction du mouvement, UP et DOWN servent au réordonancement de item, LEFT sert au glissement de item.

3. Analyse de disposition:
Pour afficher une liste de chose, il nous faut utiliser le “RecyclerView”. Comme montré dans le code ci-après:

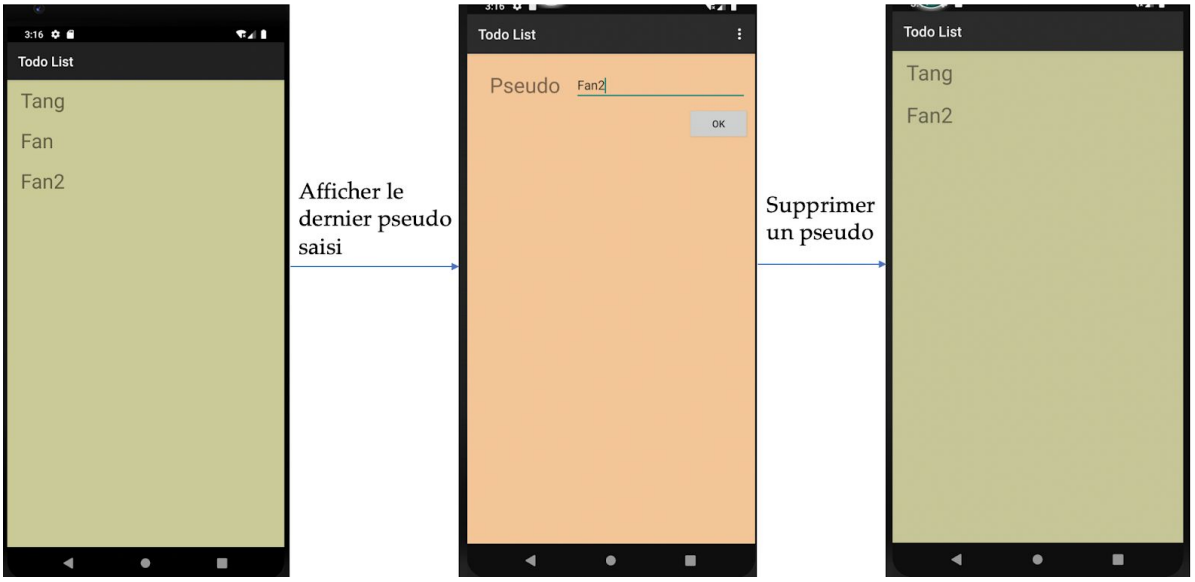
```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/list_activity"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="9"
    android:background="#FFCCCC"
    ></androidx.recyclerview.widget.RecyclerView>
```

Conclusion:

Avec notre travail réalisé, on a réussi à créer l’application “ToDo List”. Voici quelques images pour montrer le processus de l’application.

- Pour MainActivity et SettingActivity, on peut accéder aux préférences, choisir ou supprimer un pseudo. Le dernier pseudo saisi s’affiche lorsque l’on ouvre l’application automatiquement.

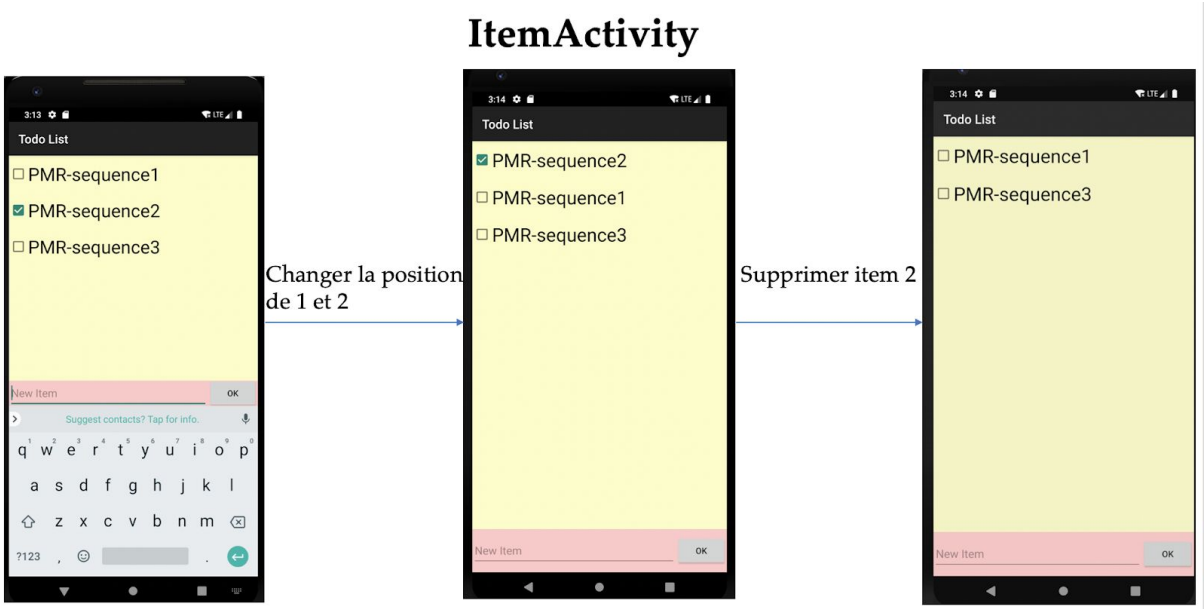
MainActivity et SettingActivity



- Pour ListActivity, on peut ajouter, supprimer une liste et on peut changer la position de deux listes.

ListActivity

- Pour ItemActivity, on peut ajouter ou supprimer un item. On peut aussi cocher ou décocher un item.


ItemActivity
Todo List
PMR-sequence1
PMR-sequence2
PMR-sequence3
New Item
OK
Changer la position de 1 et 2
Todo List
PMR-sequence2
PMR-sequence1
PMR-sequence3
New Item
OK
Supprimer item 2
Todo List
PMR-sequence1
PMR-sequence3
New Item
OK