POLYTECH NICE-SOPHIA — Université Nice Sophia Antipolis

Membre de UNIVERSITÉ CÔTE D'AZUR

Nom.....

Prénom.....

2022-2023 – 2ème semestre

N° carte étudiant.....

NOTE / 20

Polytech SI3 – sEIIN625B - ECUE Interfaces Homme Machine

Partie 1 (récupération des données)

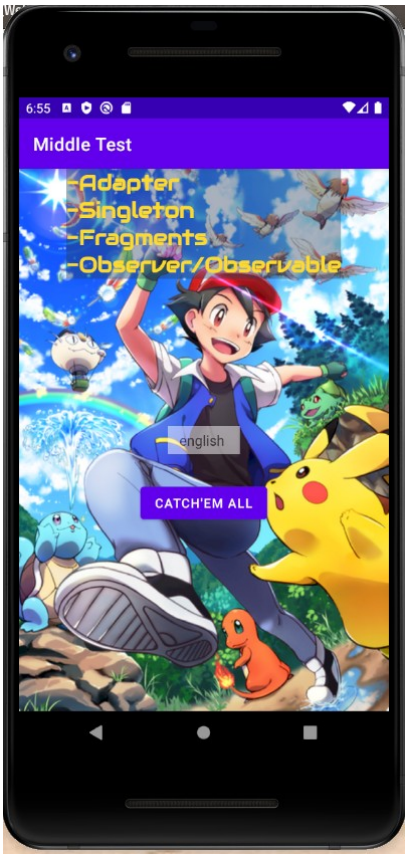
Reprenez les éléments donnés en annexe et vérifiez que vous disposez de tous les éléments.

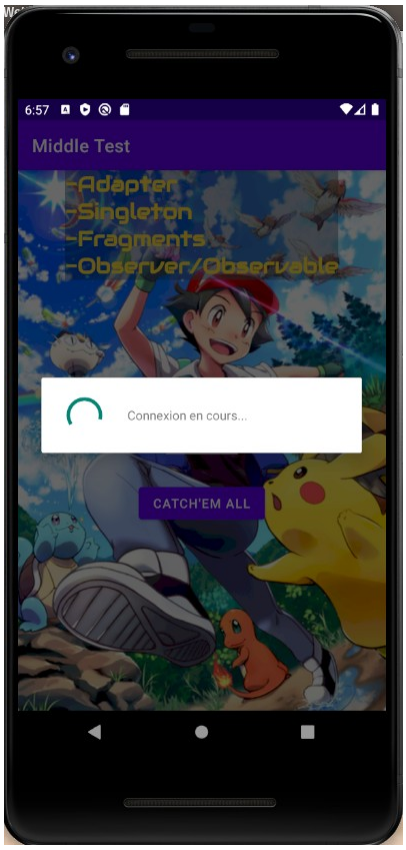
Ressources :

- Les images fond.jpg et forest.jpg à placer en arrière-plan.
- L'image podium.png.
- Un fichier JSON hébergé sur https://raw.githubusercontent.com/fanzeyi/pokemon.json/17d33dc111ffcc12b016d6485152aa3b1939c214/pokedex.json.
- La classe générique HttpAsyncGet<T>.
- L'interface Clickable.
- L'interface FragmentNotifiable.
- L'interface PostExecuteActivity<T>.
- La classe Pokemon à compléter.

Partie 2 (Vue)

Créez la vue qui ressemble à l'illustration ci-contre. Les langues possibles sont celles définies dans le fichier JSON. Les langues possibles seront décrites de façon statique à l'aide d'un tableau de chaînes de caractères (**string-array**) dans le fichier XML.





Partie 3 (MainActivity)

Créez votre MainActivity. Lorsqu'on clique sur le bouton, l'activité doit récupérer la liste des Pokemons depuis le fichier JSON en ligne et récupérer la langue choisie depuis le spinner.

Pour tester le bon fonctionnement du système, à la fin de l'activité, vous devez afficher le premier Pokemon de la liste ainsi que la langue choisie dans un journal de débogage (Log).

L'application doit être utilisable en mode portrait pour la première activité. Vous n'avez pas besoin de créer les vues pour l'autre orientation dans le cadre de cet exercice, ni d'interdire la rotation : cela reste la responsabilité de l'utilisateur.

ASTUCES :

Pensez à donner une valeur à l'attribut de classe public langage (Pokemon.language)

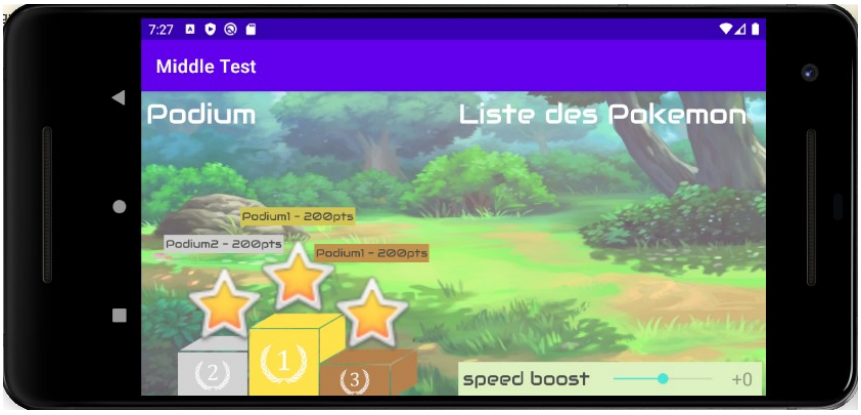
La liste de Pokemon finalement créée est un singleton

```
fredrallo MainActivity  edu.frallo.myapplication          D  First pokemon = Bulbasaur
```

Partie 4 (Vue avec fragments)

Créez une nouvelle activité : ResultActivity. Cette nouvelle activité doit être utilisée en mode paysage, mais dans le cadre de cet exercice, vous n'avez pas à vous préoccuper d'interdire la rotation : cela reste la responsabilité de l'utilisateur. Modifiez votre code pour que lorsque les Pokemons soient chargés, l'application accède automatiquement à ResultActivity.

Créez la vue de ResultActivity qui ressemble à l'illustration ci-contre. À droite, vous devez afficher **de manière statique** le Fragment ListPokemonFragment, et à gauche le Fragment PodiumPokemonFragment. Créez également les vues des Fragments. Le Fragment ListPokemonFragment doit contenir une ListView.



Partie 5 (ListPokemonFragments)

Ce fragment n'a besoin d'aucune information préalable pour fonctionner. La classe ResultActivity, qui héberge ce fragment, implémente désormais l'interface FragmentNotifiable.
Le Fragment ListPokemonFragment s'attend maintenant à rendre des comptes à l'activité qui l'héberge en appelant la méthode ((FragmentNotifiable) activity).onFragmentNotify().

Astuce :

Lorsqu'il est attaché à une activité, le fragment s'assure que celle-ci implémente l'interface FragmentNotifiable :

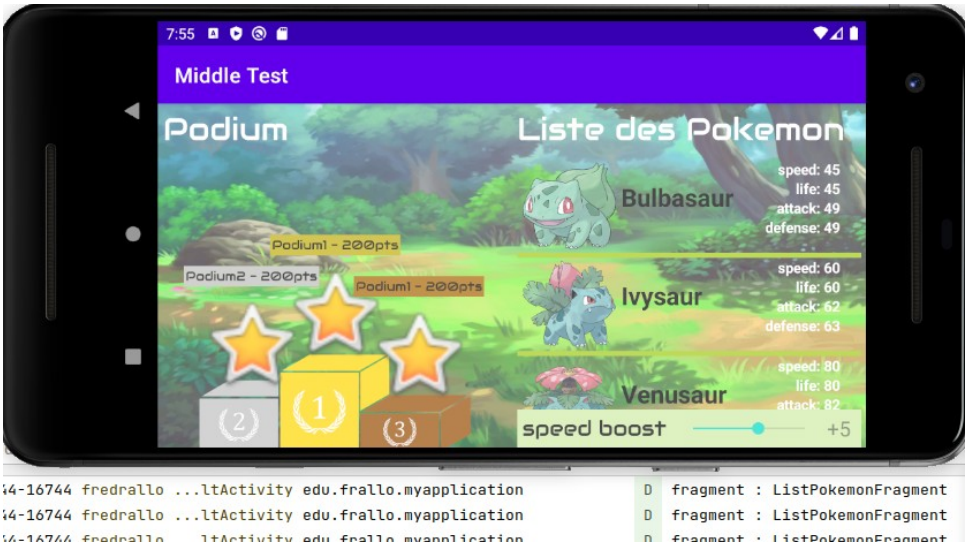
```
private FragmentNotifiable mCallback;  //attribut

...

@Override
public void onAttach(Context context) {
   super.onAttach(context);

   if (context instanceof FragmentNotifiable)
      mCallback = (FragmentNotifiable) context;
   else
      throw new RuntimeException("FragmentNotifiable not implemented in context");
}
```

Le rôle de ce fragment est le suivant :

- afficher une liste de Pokemon comme dans l'illustration ci-dessous (en utilisant un adaptateur personnalisé PokemonAdapter à créer).
- Lorsqu'on clique sur l'un des Pokemons de la liste, les informations relatives à ce Pokemon doivent être enregistrées dans un Log.

```
fredrallo ...onFragment edu.frallo.myapplication          D  Ivysaur (60) - [GRASS, POISON]
```

- Lorsqu'on déplace la SeekBar, la valeur comprise entre -30 et +30 doit être affichée.
- Lorsqu'on déplace la SeekBar, on appelle la méthode statique Pokemon.boost() qui recalcule le rang de chaque Pokémon. Ensuite, on met à jour les données affichées dans l'adaptateur avec adapter.notifyDataSetChanged(). Eh oui, un adaptateur est un observateur ;-)
- Informer l'activité qui héberge ce fragment que la valeur de la SeekBar a changé (affichage dans un Log le nom du fragment qui prévient).



Partie 6 (Podium)

Il nous faut à présent être en mesure de comparer les Pokémon. Pour cela, nous allons faire en sorte que la classe Pokemon implémente l'interface Comparable. De plus, nous avons besoin d'une nouvelle classe Podium (à ajouter dans le fichier Pokemon qui contient tout le modèle de données). Il est important de noter que Podium est Observable (par PodiumFragment... nous verrons cela à l'étape suivante).

- un Podium contient une List<Pokemon> (il nous en faut 3 mais on peut imaginer plus)
- A la création, le podium contient les 3 meilleurs de la liste des Pokemon
- à la demande on peut mettre à jour le podium.

Partie 7 (PodiumFragment)

Ce fragment n'a besoin d'aucune information préalable pour fonctionner.

- Le constructeur vérifie que la liste de Pokemon existe sinon il ne peut pas afficher de Podium → on peut alors faire new Throwable("Pokemon's completeList is null");
- Il observe le Podium
- Lorsqu'il est informé que le podium a changé, il affiche les 3 nouveaux meilleurs Pokemon

ASTUCE :

Depuis une activité, pour trouver la référence d'un fragment en cours d'exécution, on peut écrire PodiumPokemonFragment podiumPokemonFragment = (PodiumPokemonFragment) getSupportFragmentManager().findFragmentById(R.id.fragment_podium)