TP5 : Éléments graphiques avancés : Listes et Adaptateurs

Partie 1 : ListView

1) Créez un nouveau projet **TP5**, contenant une activité appelée « **NotesActivity** »
2) Insérer dans l'interface une widget ListView. Vous lui affecterez l'id : **simpleListView**

```
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="...">

    <TextView
        android:id="@+id/nomMatiere"
        android:layout_width="match_parent"
        android:layout_height="74dp"
        android:text="Matiere"
        android:layout_gravity="center"
        android:textStyle="bold"
        />

    <ListView
        android:id="@+id/simpleListView"
        android:layout_width="..."
        android:layout_height="..."
        android:divider="..."
        android:dividerHeight="..."
        android:listSelector="...">

    </ListView>
```

3) Ajoutez une ressource de type tableau au fichier **res/values/arrays.xml**:

   Nom : notes

   Type : String

   Contenu : {"12.5", "15", "7","12","10","18","8"};

   NB: Voir annexe pour l'utilisation d'une ressource tableau

4) Remplir la ListeView avec ce tableau en utilisant un ArrayAdapter.

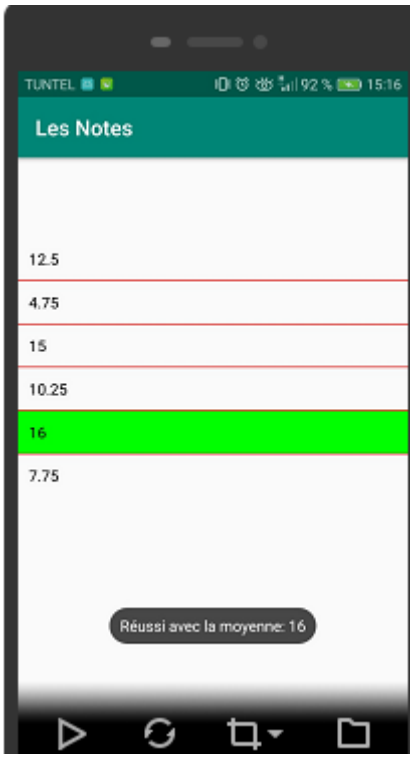
```
ArrayAdapter<String> arrayAdapter;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_noteslist);

     //Completer l'initialisation
```

```
ArrayAdapter<String>  arrayAdapter = . . .
//Lier l'adaptateur à la liste
```

Le résultat que vous allez obtenir doit ressembler à l'interface suivante :



5) On veut implémenter l'événement onClick sur un Item de la liste.
Pour se faire, il faut surcharger la méthode onItemClick, comme suit :

```
listeview.setOnItemClickListener(new AdapterView.OnItemClickListener() {
@Override
    public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
     // behavior when you click on an item
     // Récupérer le contenu de l'Item sélectionné
     Object  o = liste.getItemAtPosition(i);
     Toast.makeText(Note.this, "Bonjour" + o, Toast.LENGTH_SHORT).show();
   }
 });
```

Si on clique sur un élément de la liste élément, afficher dans un toast « Réussi » si la note est supérieure à 10, et « Échoué.. » sinon.

6) Vérifiez le bon fonctionnement de votre application

Partie 2 : Auto-Complet TextView

Il est parfois utile, pour faciliter la saisie de données, de fournir à l'utilisateur des propositions suite à un début de saisie dans un champs. Pour cela, un widget particulier est fourni, appelé « **AutoCompleteTextView** ».

Cet élément est une sous-classe de EditText, on peut donc le paramétrer de la même manière, mis à part un attribut supplémentaire : **android :completionThreshold**, qui indique le nombre minimum de caractères qu'un utilisateur doit entrer pour que les suggestions apparaissent.

Pour l'utiliser, suivre les étapes suivantes :

1) Dans une autre activité « **MatiereActivity** », insérez un AutoCompleteTextView (id = « matiere » et completionThreshold = "2") .

```
<AutoCompleteTextView
    android:id="@+id/..."
    android:layout_width="match_parent"
    android:layout_height="123dp"
    android:completionThreshold="..."
    android:hint="Matiere"
    android:layout_margin="10dp"/>
```

2) Créez un nouveau tableau :

```
String matieres[] = {"Algoritmique", "Android","Programmation C",
                     "Programmation OO"}
```

3) Associer un adaptateur à ce widget, de type ArrayAdapter.

   Notez que le layout utilisé comme type pour un élément de la liste est un **simple_dropdown_item_1line.**

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_matiere);

    //Initialisation

    //Creation d'un adaptateur

   //Lier l'adaptateur à la liste
```

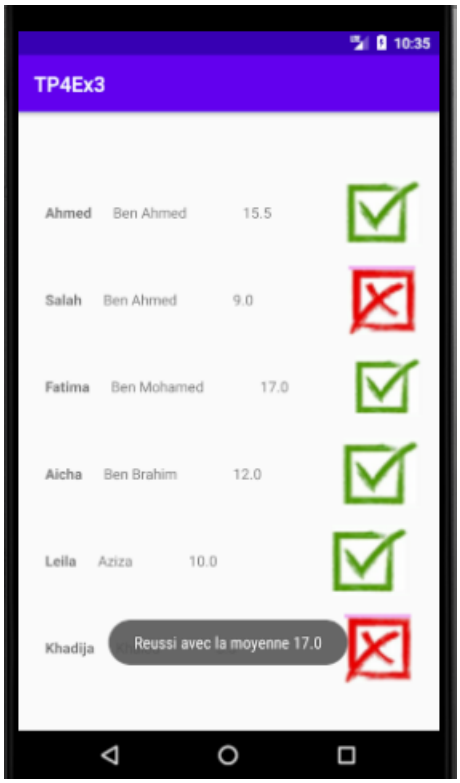
Le résultat obtenu aura l'allure suivante :

4) Implémentez l'événement setOnItemClickListener relatif à l'AutoCompleteTextView

Définissez son comportement de manière à ce qu'on affiche le nom de la matière dans un toast.

Partie 3 : RecyclerView

Dans le cas où on aimerait que le contenu de la liste (activité NotesActivity) soit plus complexe qu'un simple TextView, et qu'il puisse afficher plus qu'un élément.

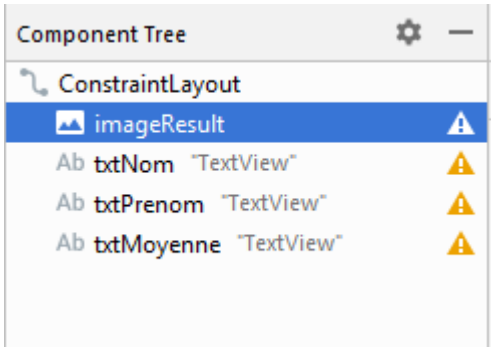


Pour cela :

1) Créer un nouveau projet nommé **"Tp5Ex3"**, ajouter les dépendances de RecyclerView dans le script gradle
2) Ajoutez à vos ressources les images suivantes : « **ok.jpg** » et « **ko.jpg** ».

Modéliser une ligne de la liste

3) Ajoutez une ressource de type layout qu'on appellera « **row.xml »** et implémentez son code source

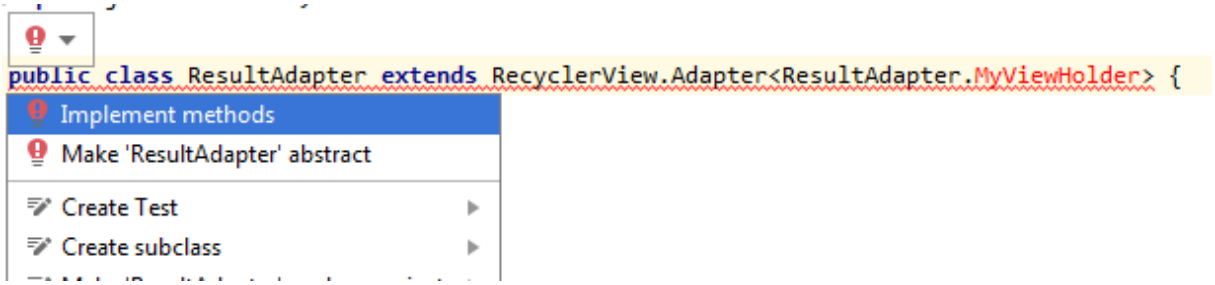


4) Ajouter une classe **Resultat** qui représente le "Data Model" du résultat d'un étudiant définit par un nom et prénom de l'étudiant, sa moyenne et une image dépendante de la moyenne.

Créer l'Adaptateur

1) Créez un adaptateur qu'on nommera MyAdapter et qui héritera de la classe **RecyclerView.Adapter<MyAdapter.MyViewHolder>**

Corriger les erreurs en implémentant le constructeur et les méthodes proposées. Ajouter la classe **MyViewHolder**




```
      public class MyAdapter extends RecyclerView.Adapter< . . .> {

    List<ResultatEdut> resultatList;


    public MyAdapter(List<ResultatEdut> liste) {
        //Initialisation
    }


@NonNull
    @Override
    public MyViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int
viewType) {
        View v =  . . .
        return new MyViewHolder(v);
    }


    @Override
    public void onBindViewHolder(@NonNull MyViewHolder holder, int
position) {
```

```
        . . .
    }

    @Override
    public int getItemCount() {
        return . . .;
    }

    public class MyViewHolder extends RecyclerView.ViewHolder {
        . . . //Déclaration
        public MyViewHolder(@NonNull final View itemView) {
            super(itemView);
            . . .
        }
    }
}
```

Activité principale

1) Ajouter une liste RecyclerView à votre interface principale.
2) Dans le fichier MainActivity.java, déclarer la liste RecyclerView
3) Déclarer la liste des résultats

```
List<Resultat> lesResultats = new ArrayList<Resultat>();
. . .
lesResultats.add(new Resultat("Ahmed","Ben Ahmed" , (float) 15.5,R.drawable.ok));
lesResultats.add(new Resultat("Salah","Ben Ahmed" , (float) 9,R.drawable.ko));
lesResultats.add(new Resultat("Fatima","Ben Mohamed" , (float) 17,R.drawable.ok));
lesResultats.add(new Resultat("Aicha","Ben Brahim" , (float) 12,R.drawable.ok));
lesResultats.add(new Resultat("Leila","Aziza" , (float) 10,R.drawable.ok));
lesResultats.add(new Resultat("Khadija","Khaled" , (float) 5.5,R.drawable.ko));
```

4) Attachez votre adaptateur à cette liste
5) Ajouter un LayoutManager à votre liste;
6) Ajouter un événement lors du clique sur un élément de la liste (dans la classe MyViewHolder), afficher dans un toast un message "Reussi/Echec avec la moyenne"

7) Refaire le même projet en affichant uniquement la moyenne et une image, tester le gestionnaire d'affichage GridLayout /StraggeredLayout

Annexes : Ressources de type tableau

Fichier **res/values/arrays.xml**

La syntaxe est :

```
<array name="mon_tableau">
    <item>Element 1 </item>
    <item>Element 2</item>
</string-array>
```

Fichier **java**

Pour assigner cette valeur à un champ de votre interface :

```
String[] nom-tableau-java =
getResources().getStringArray(R.array.mon_tableau_res);
```

Annexes : CircleImage

Il est possible d'utiliser des ImageView circulaires sous Android. Pour se faire :

Gradle

```
dependencies {
            ...
            implementation 'de.hdodenhof:circleimageview:2.2.0'
}
```

Utilisation

```
<de.hdodenhof.circleimageview.CircleImageView
…/>
```

Au lieu de la balise **ImageView**