

Databinding

Le Databinding ou liaison de donnée permet de déléguer au framework le fait de remplir des données dans la vue en utilisant des mécanismes de génération de code, afin de limiter le code répétitif.

Databinding dans la RecyclerView

Dans un premier temps, voyons comment utiliser le databinding pour les éléments de la RecyclerView.

Layout

Commençons par le Layout, `todo_list_item`, dans lequel nous allons rajouter une balise qui va englober tout notre Layout :

```
<layout xmlns:android="http://schemas.android.com/apk/res/android">

....

</layout>
```

(n'oubliez pas de déplacer le XMLNS Android de votre LinearLayout à cette nouvelle balise.

Nous devons ensuite déclarer le contexte de donnée du layout :

```
<layout xmlns:android="http://schemas.android.com/apk/res/android">

  <data>
    <variable
      name="todo"
      type="fr.arsenelapostolet.jetpacktuto.model.TODO" />
  </data>

  ....

</layout>
```

Ces balises permettent de déclarer que ce Layout possède un contexte de donnée qui dépend d'une variable de la classe Todo et qui s'appelle todo. On va donc pouvoir utiliser cette variable dans les éléments de notre Layout :

```
...  
  
<TextView  
    android:id="@+id/name"  
    style="@style/Title"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@{todo.title}"></TextView>  
  
<TextView  
    android:id="@+id/lifespan"  
    style="@style/Text"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@{todo.description}"></TextView>  
  
...
```

En utilise la syntaxe `@{variable.champs}` on peut lier des données dans notre Layout.

Adapteur

Voyons maintenant du coté de notre adapteur de liste, `TodoListAdapter`, qui doit être légèrement modifié pour supporter la liaison de données.

Tout d'abord, notre classe interne ViewHolder ne va plus détenir une View mais un `TodoListItem`, une classe générée par le framework pour opérer notre liaison de donnée.

```
class TodoViewHolder extends RecyclerView.ViewHolder {  
  
    public TodoListItem itemView;  
  
    public DogViewHolder(@NonNull TodoListItem binding) {  
        super(binding.getRoot());  
        this.itemView = binding;  
    }  
}
```

Ensuite, dans la méthode `onCreateViewHolder`, nous devons légèrement modifier la façon dont on *inflate* notre View :

```
public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {  
    LayoutInflater inflater = LayoutInflater.from(parent.getContext());  
    TodoListItem view = DataBindingUtil.inflate(inflater, R.layout.todo_list_item, parent, false);  
    return new ViewHolder(view);  
}
```

Cela permet de faire appel au mécanisme de Databinding pour *inflate* notre View.

Enfin, nous allons remplacer le contenu de la méthode `onBindViewHolder` par une seule ligne, l'affectation de l'objet du modèle à la liaison de donnée :

```
public void onBindViewHolder(@NonNull ViewHolder holder, int position) {  
    holder.itemView.setTodo(todoList.get(position));  
}
```

Lier des gestionnaires d'événement

On peut également utiliser le Databinding pour faire des liaisons de gestionnaire d'événements. Faisons ça pour implémenter la navigation au clic d'un élément de la RecyclerView.

...

Dans votre contexte de donnée, ajoutez une variable de type `OnClickListener`. Il s'agit d'une interface fonctionnelle qui définit un contrat de service pour la gestion d'un événement de clique.

Ensuite dans notre adaptateur, dans la méthode `onBindViewHolder`, rajoutons le binding en passant une :

```
holder.itemView.setOnTodoClicked(v -> {  
    ListFragmentDirections.ActionDetail action = ListFragmentDirections.actionDetail();  
    action.setDogUuid(Integer.parseInt(todo.getId()));  
    Navigation.findNavController(v).navigate(action);  
});
```

Databinding dans un Fragment

Dans le Layout, on va faire exactement la même chose :

- Rajouter un élément `layout` à la racine
- Rajouter une élément `data` pour le contexte donnée
- Dans l'élément `data`, créer avec élément `variable` avec un nom et comme type `Todo`
- Déclarer nos liaisons de données avec la syntaxe `@{ }`

Pour le Code Behind du fragment, on va pouvoir supprimer tous nos `@BindView` et remplacer le contenu notre méthode `onCreateView` :

```
private FragmentDetailBinding binding;

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                        Bundle savedInstanceState) {
    this.binding = DataBindingUtil.inflate(inflater, R.layout.fragment_detail, container, false);
    return binding.getRoot();
}
```

Revision #5

Created 31 January 2021 14:44:27 by Arsène Lapostolet

Updated 2 April 2021 12:28:48 by Arsène Lapostolet