Navigation interne

1. Objectifs

* Comprendre ce qu’est Android Jetpack’s Navigation
* Savoir utiliser Android Jetpack’s Navigation.

1. Présentation

Android Jetpack’s Navigation met à disposition des outils pour simplifier la navigation. Le principe est simple : une activité servant de conteneur et des fragments qui alternent leur affichage au sein de ce conteneur. Pour définir la cinématique d'affichage des différents fragments Android Jetpack’s Navigation met à disposition un designer.

1. Avant

La navigation dans une application Android AVANT l’arrivée d’Android 3.0 : c’était le passage d’un objet Activity à un autre.

Le passage d’informations entre deux entités pouvait s’effectuer par l’ajout de paramètres dans le bundle de l’objet Intent et à un écran de l’application correspondait une classe Activity : La navigation était plutôt simple à gérer !

Avec l’apparition des tablettes et leurs écrans plus larges, de nouveaux besoins sont apparus. A partir d’Android 3.0 (API 11), la notion de fragments apporte davantage de possibilités de navigation au sein d’une application. Une activité pouvant être composée de plusieurs fragments : composants encapsulés et réutilisables qui possèdent leur propre cycle de vie et leur propre interface graphique.

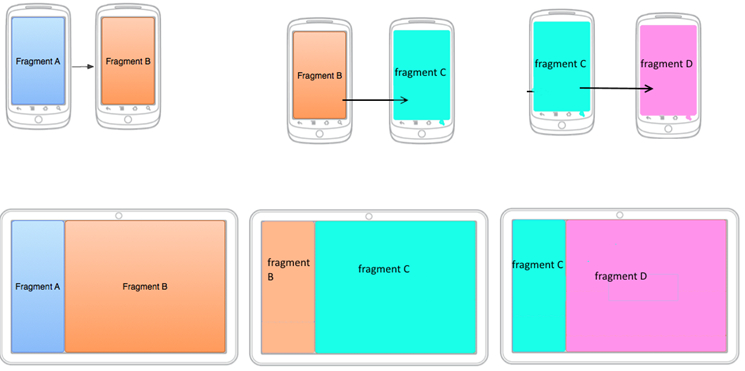
​​​​​​​De nouvelles possibilités, mais plus de complexité dans la gestion de la navigation :  
passage d’informations entre les différents éléments (Activity et Fragment), gestion des transactions d’un fragment à l’autre, gestion de la pile des appels et du bouton retour etc…

Le dessin ci-dessous illustre l'ancienne méthode de navigation ou les développements étaient basés sur une navigation de passages d'activités en activités.

Une image contenant texte, tableau blanc

Description générée automatiquement

Les deux schémas ci-dessous illustre la nouvelle méthode qui est conseillée et qui incite les développeurs et développeuses à utiliser les fragments. On voit que l'ihm peut dépend de la taille de l'écran, lors d'une consultation de mail sur un petit écran l'écran principal n'affiche que la liste de mails alors que si j'utilise un grand écran alors il y a la possibilité d'afficher la liste de mails ainsi que le contenu des mails et il serait dommage d'être obligé de développer l'entièreté de chaque écran pour petit écran ou grand écran, l'utilisation des fragment permet la réutilisation de parties d'ihm et ainsi, lorsque l'application est consulté sur un petit écran alors il n'y a qu'un fragment d'affiché et si l'application est consulté sur un grand écran alors il y a deux écrans qui sont affichés.



Cette facon de faire était très fastidieuse par le passé mais depuis l'apparition d'Android JetPack's navigation la mise en place et l'utilisation des fragment est devenu très simple.

1. Aujourd'hui : Android Jetpack's Navigation
   1. Présentation

La navigation fait référence aux interactions qui permettent aux utilisateurs de parcourir, entrer et sortir des différents éléments de contenu de votre application. Le composant de navigation d'Android Jetpack aide à mettre en œuvre la navigation, des simples clics sur les boutons aux modèles plus complexes, tels que les barres d'applications et le tiroir de navigation. Le composant de navigation garantit également une expérience utilisateur cohérente et prévisible en adhérant à un ensemble de principes établis.

* 1. Les composantes d'Android Jetpack's Navigation

Les grandes composantes de cette librairie sont :

* Navigation Graph : Fichier XML avec son éditeur
* Composant NavHost est un conteneur
* Composant NavController est le chef d’orchestre

Avec Android Jetpack’s Navigation et sa collection d’outil, Google propose une prise en charge rapide et plus aisée de la navigation, le Navigation Architecture Component se compose de 3 parties :

Le Navigation graph : un fichier resource XML de type Navigation qui contient les éléments du graphe de navigation de l’application.

Le Composant NavHost : un conteneur qui délègue la navigation du graphe à son NavController. Par défaut, ce fragment hérite de la classe NavHostFragment et servira de conteneur où les fragments seront placés.

Le composant NavController : un objet qui gère et orchestre la navigation. Chaque NavHost détient un NavController, dont le rôle est de gérer la navigation dans le le NavHost.

Avec une version d’Android Studio supérieure ou égale à 3.3, le graphe de navigation peut facilement être édité avec l’outil qui lui est associé : le Navigation Editor. Ce dernier, permet de créer simplement des destinations reliées par des actions et même paramétrer les données qui seront transmises et les animations jouées lors des transitions.

Cet éditeur est également en lien avec les assistants qui permettront de créer le code rapidement de l’application.

* 1. Mise en place

Pour expliquer comment mettre en place Android navigation, un nouveau projet va être créé, les étapes suivantes seront présentées dans cette mise en place.

* Avoir des fragments
* Créer un graphe de navigation et le configurer
* Mettre en place une activité faisant conteneur de fragment
* Définir les actions de navigation
  + 1. Configuration du projet

La méthode de viewBinding est définie pour gérer les ihms de cette démonstration.

android {

...

buildFeatures {

viewBinding = true

}

...

}

* + 1. Création de fragments

Pour créer un fragment il faut faire un clic droit sur le package dans lequel le fragment doit être positionné puis NEW -> Fragment -> Empty Fragment cela à pour effet de créer le contrôleur, c'est à dore une class héritant de la classe fragment ainsi qu'un fichier xml représentant l'ihm du fragment.

Trois fragments vont être créés ci-après, le fragment principal sera nommé "PrincipalFragment" il contient 2 boutons, 1 pour aller vers le fragment nommé BlueFragment et un autre bouton pour aller vers GreenFragment.

Code source de l'IHM de PrincipalFragment

Le code ci-dessous montre comment définir l'ihm du fragment principal de la démonstration :

<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 tools:context=".principal\_fragment.PrincipalFragment">  
  
 <TextView  
 android:id="@+id/tv\_title"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Fragment principal"  
 app:layout\_constraintLeft\_toLeftOf="parent"  
 app:layout\_constraintRight\_toRightOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 android:layout\_marginTop="100dp"  
 />  
  
 <Button  
 android:id="@+id/btn\_blue"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:backgroundTint="#2196F3"  
 android:text="Blue"  
 app:layout\_constraintTop\_toBottomOf="@id/tv\_title"  
 app:layout\_constraintLeft\_toLeftOf="parent"  
 app:layout\_constraintRight\_toRightOf="parent"  
 />  
  
 <Button  
 android:id="@+id/btn\_green"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:backgroundTint="#8BC34A"  
 android:text="Green"  
 app:layout\_constraintTop\_toBottomOf="@id/btn\_blue"  
 app:layout\_constraintLeft\_toLeftOf="parent"  
 app:layout\_constraintRight\_toRightOf="parent"  
 />  
  
</androidx.constraintlayout.widget.ConstraintLayout>

Code source de l'IHM de BlueFragment

Le code ci-dessous montre comment définir un fragment qui affiche un écran tout bleu.

<?xml version="1.0" encoding="utf-8"?>  
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".blue\_fragment.BlueFragment"  
 android:background="#03A9F4"  
 >  
 <TextView  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:text="Blue"  
 android:textSize="75sp"  
 />  
</FrameLayout>

* L'attribut android:background avec la valeur "#03A9F4" permet de rendre l'écran bleu.

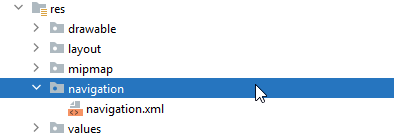
Code source de l'IHM de GreenFragment

Le code ci-dessous montre comment définir un fragment qui affiche un écran tout vert.

<?xml version="1.0" encoding="utf-8"?>  
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".green\_fragment.GreenFragment"  
 android:background="#8BC34A"  
 >  
 <TextView  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:text="Green"  
 android:textSize="75sp"  
 />  
</FrameLayout>

* + 1. Création et configuration du graphe de navigation

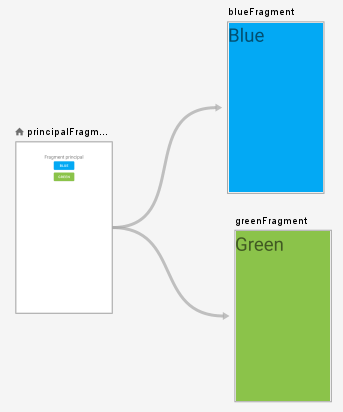
Pour créer le graphe de navigation il faut simplement créer un fichier nommé navigation.xml dans le dossier navigation du dossier res.



Pour la configuration du graphe de navigation il est nécessaire d'utiliser le designer qui doit apparaitre lors d'un double clic sur le fichier navigation.xml. Les ressources devant être utilisées doivent être ajoutées grâce à ce bouton :



Une fois les ressources ajoutées il suffit de définir la cinématique de votre application :



Les flèches entre les différents fragments sont des actions de navigation, les objets les représentant permettent de naviguer entre les différents fragments.

* + 1. Création de l'activité conteneur de fragments

Il n'y a rien à faire dans le contrôleur de votre activité principale, seule l'ihm de l'activité principale doit être modifiée comme ceci :

Activity\_main.xml

Le code ci-dessous montre comment définir le conteneur de fragments.

<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout  
 xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".MainActivity">  
  
 <fragment  
 android:id="@+id/notreNavHostFragment"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:name="androidx.navigation.fragment.NavHostFragment"  
 app:defaultNavHost="true"  
 app:navGraph="@navigation/navigation"/>  
  
</androidx.constraintlayout.widget.ConstraintLayout>

C'est la balise fragment qui fait office de conteneur de fragment, il est possible d'avoir plusieurs balises fragment au sein d'une même activité.

* + 1. Utilisation des actions de navigation

Utiliser les actions de navigation c'est naviguer entre les différents fragments.

Le code ci-dessous montre comment un clic sur le bouton nommé bleu du fragment principal permet de redirigé l'utilisateur vers le fragment nommé blueFragment et comment un clic sur le bouton nommé vert du fragment principal permet de redirigé l'utilisateur vers le fragment nommé greenFragment.

class PrincipalFragment : Fragment() {  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View? {  
 var vb = FragmentPrincipalBinding.inflate(layoutInflater)  
  
 vb.btnBlue.setOnClickListener {  
 Navigation.createNavigateOnClickListener(R.id.action\_principalFragment\_to\_blueFragment).onClick(it)  
 }  
 vb.btnGreen.setOnClickListener {  
 Navigation.createNavigateOnClickListener(R.id.action\_principalFragment\_to\_greenFragment).onClick(it)  
 }  
 return vb.root  
 }  
}

* 1. Conclusion

Il a été vu comment définir la cinématique d'affichage des différents fragments dans une application. La pièce centrale permettant de mettre en place ce système est le fichier navigation.xml éditable via le designer. Cela apporte un certain confort aux développeuses et développeurs et apporte aussi énormément de simplification par rapport aux anciennes méthodes. Il est ainsi recommandé d'utiliser Android JetPack Navigation.

1. SafeArgs
   1. Présentation

Le composant Navigation a un plugin Gradle appelé Safe Args qui génère des objets simples et des classes de générateur pour une navigation de type sécurisé et l'accès à tous les arguments associés. Safe Args est fortement recommandé pour naviguer et transmettre des données, car il garantit la sécurité de type.

* 1. Mise en place
* Définir les dépendances et le plugin
* Définir les besoins d’informations dans le navigations Graph
* Fournir les paramètres
* Récupérer les paramètres
  + 1. Définir les dépendances et le plugin

Dans le build.gradle du projet, avant "plugins".

buildscript {

dependencies {

classpath "androidx.navigation:navigation-safe-args-gradle-plugin:2.3.5"

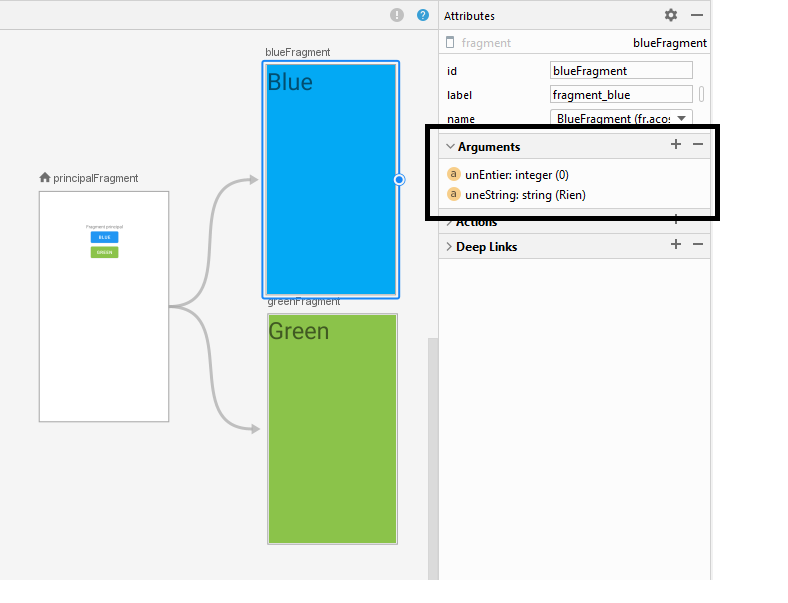
}

}

Plugin à ajouter dans le build.gradle du Module.

'androidx.navigation.safeargs'

* + 1. Définir les besoins d’informations dans le navigation Graph



* + 1. Fournir les paramètres

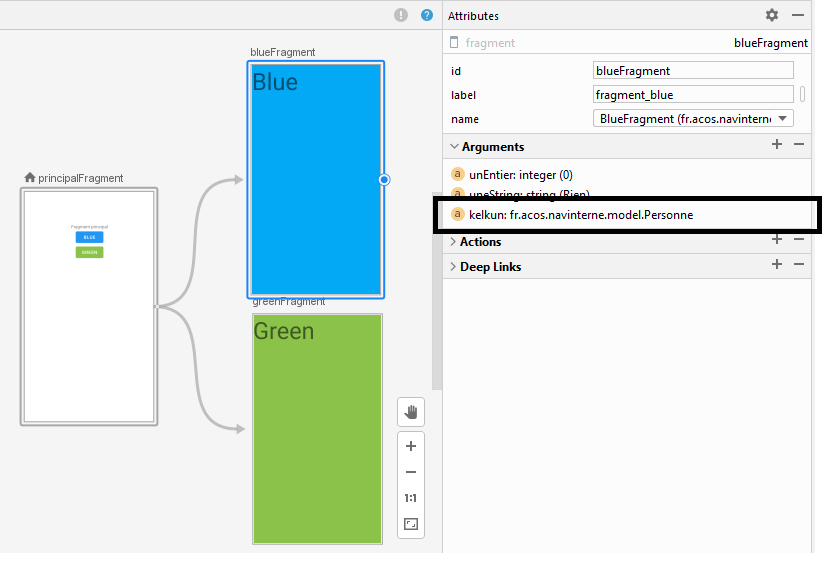
class PrincipalFragment : Fragment() {  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View? {  
 var vb = FragmentPrincipalBinding.inflate(layoutInflater)  
  
 vb.btnBlue.setOnClickListener {  
 val action = val action = PrincipalFragmentDirections.actionPrincipalFragmentToBlueFragment(29,"QCDA01 En force")

Navigation.findNavController(it).navigate(action)  
 }  
 vb.btnGreen.setOnClickListener {  
 Navigation.createNavigateOnClickListener(R.id.action\_principalFragment\_to\_greenFragment).onClick(it)  
 }  
 return vb.root  
 }  
}

* + 1. Récupérer les paramètres

class BlueFragment : Fragment() {  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View? {  
  
 var args = arguments?.let {  
 BlueFragmentArgs.fromBundle(it)  
 }  
  
 Log.i("WOLOLOBZH","Info : ${args?.unEntier} ${args?.uneString}" )  
  
 return inflater.inflate(R.layout.fragment\_blue, container, false)  
 }  
}

* 1. Parcelable
     1. Présentation



* + 1. Définir le plugin

id 'kotlin-parcelize'

* + 1. Rendre la classe parcelable

@Parcelize  
data class Personne(var id:Int,var nom:String,var prenom:String) : Parcelable