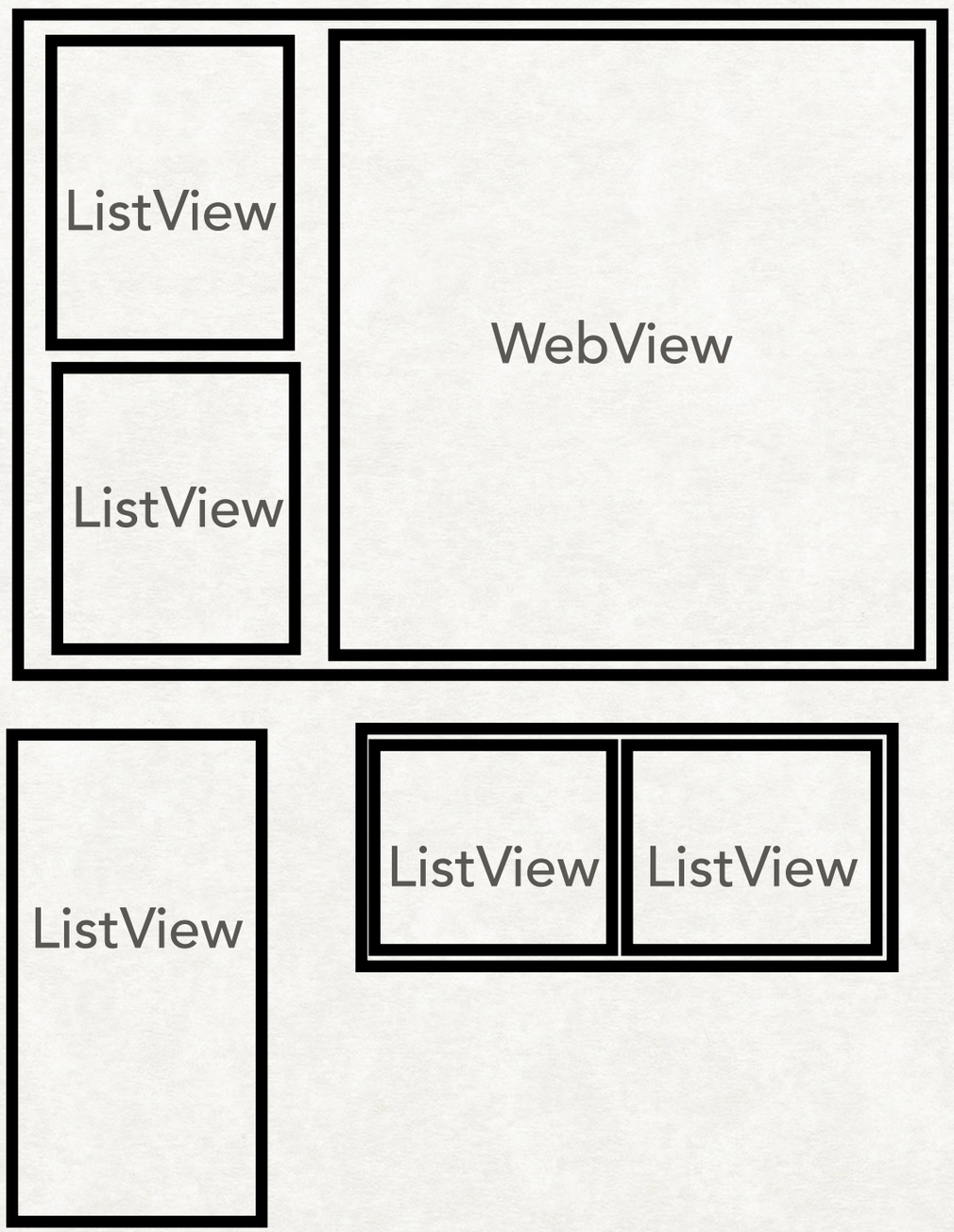
PROGRAMMATION DE COMPOSANTS MOBILES (ANDROID)

WIESLAW ZIELONKA

WWW.IRIF.UNIV-PARIS-DIDEROT.FR/~ZIELONKA

Pourquoi les fragments ?



Pour la même application :

sur une tablette on peut voir en même temps deux ListView et un WebView, trois éléments à la fois

Sur un téléphone mobile en position verticale une seule liste ou un WebView

Sur le même mobile en position horizontale deux ListViews mais pas WebView

Eviter la redondance

Problème : nous ne voulons pas de reimplémenter les mêmes comportements plusieurs fois.

Solution :
Implémenter trois fragments chacun avec sa propre interface graphique et sa logique (fonctionnalité).

Combiner les fragments dans les activités en fonction de la taille de l'écran, de la position de l'appareil (verticale, horizontale).

Possibilité de modifier l'apparence de l'activité à la volée : enlever un fragment (add), supprimer un fragment (remove), remplacer un fragment par un autre (replace)

Fragment

Les fragments

- permettent de décomposer une activité en des parties plus ou moins indépendantes
- implémentent un comportement
- peuvent être réutilisés dans des différentes activités

ajouter les dépendances

Dans build.gradle (module) dans la section dependencies:

```
def fragment_version  = "1.5.3"

implementation "androidx.fragment:fragment-ktx:$fragment_version"
```

créer un fragment

```
class ButtonFragment : Fragment(R.layout.main_fragment) {
```

le fichier layout du fragment. Si nous donnons le fichier layout au constructeur nous n'aurons plus à faire inflate pour créer la View de fragment à partir de layout, Android le fait pour nous.

layout du fragment

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/fragment"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    tools:context=".ui.main.ButtonFragment">

    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:textAlignment="center"
        android:background="#F00"
        android:textSize="@dimen/textSize"
        android:textStyle="italic" />

    <TextView
        android:id="@+id/compteur"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/button"
        android:layout_marginTop="5dp"
        android:gravity="center_horizontal"
        android:layout_alignRight="@id/button"
        android:layout_alignLeft="@id/button"
        android:text="0"
        android:textSize="@dimen/textSize" />

</RelativeLayout>
```

la classe qui implémente le fragment

le fragment composé d'un bouton et d'un TextView

fichier main_fragment.xml

le fichier layout de fragment s'écrit comme layout de l'activité

créer un fragment

chaque fragment possède un bundle **arguments** où on passe des informations qui servent à initialiser le fragment. Pour utiliser ce bundle il est commode de créer le fragment avec une méthode de type factory (static en java) :

```
class ButtonFragment : Fragment(R.layout.main_fragment) {

  companion object {
    @JvmStatic
    fun newInstance(buttonText: String, numeroFragment: Int): Fragment {
        return ButtonFragment().apply {
            arguments = Bundle().apply {
                putString("buttonText", buttonText)
                putInt("numeroFragment", numeroFragment)
            }
        }
    }
  }
```

deux extras mis dans le bundle arguments

arguments : bundle pour stocker l'info qui permet d'initialiser une instance de fragment

```
.....
On va créer les instances de ce fragment avec :

val fragmentA = ButtonFragement.newInstance( "A", 0 )
val fragmentB = ButtonFragement.newInstance( "B", 1 )
```

le texte à afficher dans le bouton du fragment

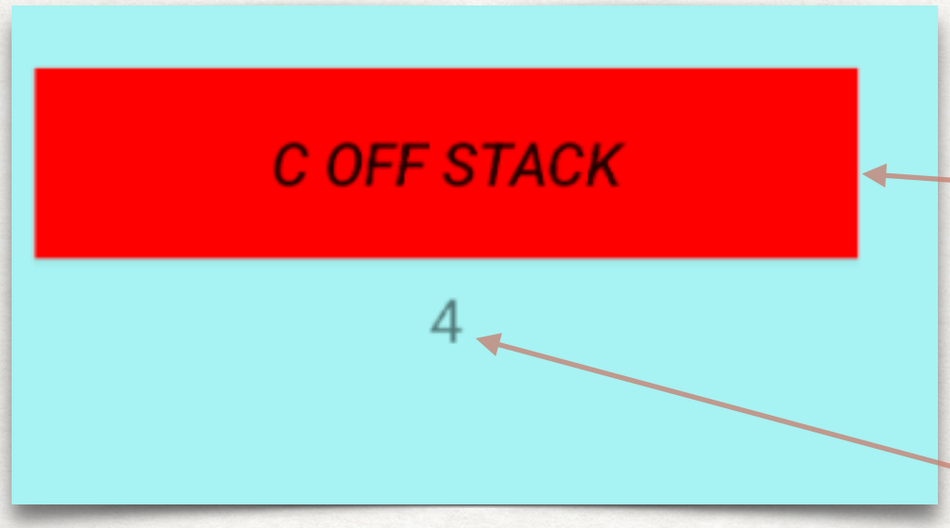
créer un fragment

Récupérer les valeurs stocker dans le bundle **arguments.**
Je suppose que view binding est activé (voir le cours sur view binding)

```
class ButtonFragment : Fragment(R.layout.main_fragment) {

    private lateinit var binding : MainFragmentBinding

    override fun onViewCreated(view: View,
                               savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        /* création de binding avec view à la racine, bind à
         * la place de inflate */
        binding = MainFragmentBinding.bind( view )

        binding.button.text =
              requireArguments().getString("buttonText", "invalide")
        /* requireArguments() retourne le bundle arguments */

        binding.button.setOnClickListener(listener)

        value = savedInstanceState?.getInt("value") ?: 0
        binding.compteurValue.text="$value"
}
```

view créée par android à partir de fichier xml

le fragment



button Le texte fixe
à la création de fragment et récupéré depuis le bundle arguments

compteur, la valeur préservée dans
le bundle paramètre de onViewCreated() et onSaveInstanceState()

```
class ButtonFragment : Fragment(R.layout.main_fragment) {

    /* la valeur de compteur affichée dans TextView */
    private  var vompteurValue: Int = 0

    override fun onSaveInstanceState(outState: Bundle) {
        super.onSaveInstanceState(outState)
        outState.putInt("value", value)
    }
```

Comme pour les activités, il existe un bundle qui est préservé pendant le changement de configuration. On sauvegarde les valeurs dans ce bundle dans la méthode onSaveInstanceState() et on les récupère dans onViewCreated()

communication entre l'activité et son fragment

L'activité connaît son fragment donc elle peut librement appeler les méthodes du fragments.

Le fragment est sensé d'être utilisé dans plusieurs activités et le fragment ne "sait" rien sur l'activité qui l'utilise. Comment le fragment peut savoir quelles méthodes de l'activité peut-il appeler?

Il y a deux solutions possibles :

1. le fragment définit une interface de communication, c'est-à-dire le fragment définit quelle méthodes doivent être implémentées par l'activité. Le fragment utilisera uniquement ces méthodes de l'activité et rien d'autre.
2. Le fragment utilise le ViewModel de l'activité pour passer les valeurs vers l'activité

communication entre l'activité et son fragment solution 1

.

```
class ButtonFragment : Fragment(R.layout.main_fragment) {

    /* interface à implementer par chaque activité qui utilise
     * ce fragment */
    interface FragmentInteractionListener {
        fun valueSend(value: Int, numeroFragment: Int )
    }
```

Dans cette exemple l'interface de communication se réduit à une fonction. Mais il est possible de définir l'interface avec autant de fonctions que nécessaire.

L'activité doit implémenter cette interface : le fragment va appeler cette méthode de l'activité. Si l'activité n'implémente cette interface alors l'activité ne pourra pas utiliser ce fragment.

(En occurrence le fragment utilisera cette fonction pour informer l'activité de la valeur de son compteur.)

communication entre l'activité et son fragment solution 1

Le fragment doit vérifier si l'activité implémente l'interface de communication définie sur la page précédente :

```
class ButtonFragment : Fragment(R.layout.main_fragment) {

  private lateinit var activity: FragmentInteractionListener

  override fun onAttach(context: Context) {
      super.onAttach(context)
      /* vérifier si l'activité implemente
       * l'interface FragmentInteractionListener */
      activity = context as FragmentInteractionListener
  }
```

onAttach() est exécuté quand le fragment devient attaché à l'activité, le paramètre c'est rien d'autre que l'activité qui veut utiliser le fragment.

context as FragmentInteractionListener : si l'activité n'implémente pas l'interface FragmentInteractionListener alors cette instruction lancera ClassCastException qui terminera le programme.

communication entre l'activité et son fragment solution 1

Le fragment de l'exemple contient un bouton. Donc il faut un listener qui sera activé quand l'utilisateur tape sur le bouton :

```
class ButtonFragment : Fragment(R.layout.main_fragment) {


  val listener = View.OnClickListener{
    compteurValue++
    binding.compteur.text = "$compteurValue"

    /* appeler l'activité pour lui communiquer
     * la valeur du compteur local */
    activity.valueSend(compteurValue,
          requireArguments().getInt("numeroFragment") ?: 0)
}
```

le fragment -- changement de configuration solution 1

Préserver les données du fragment par une sauvegarde dans un bundle :

```
class ButtonFragment : Fragment(R.layout.main_fragment) {

/* sauvegarde dans un bundle */
 override fun onSaveInstanceState(outState: Bundle) {
    super.onSaveInstanceState(outState)
    outState.putInt("value", compteurValue)
}

override fun onViewCreated(view: View, savedInstanceState: Bundle?)
{
   ......
   /* récupère le bundle de sauvegarde */
   compteurValue = savedInstanceState?.getInt("value") ?: 0

}
```

communication fragment -- activité deuxième solution

Cependant on peut régler deux problèmes :

1. le problème de préservation de données pendant le changement de configuration et
2. le problème de communication entre les fragment et l'activité

d'un seul coup en utilisant le ViewModel.

L'idée : les fragments utilisent le ViewModel de l'activité pour stocker ses propres données. L'activité installe les observer pour réagir sur le changement de données effectués par les fragments.

communication fragment -activité deuxième solution

Préserver ls données -- le ViewModel partagé avec l'activité :

```
class MainActivity : AppCompatActivity() {

    /* récupérer le ViewModel  */
    val model by lazy{ ViewModelProvider(this).get(ActivityModel::class.java)  }

}
--------
class ButtonFragment : Fragment(R.layout.main_fragment) {

     /* on récupère le ViewModel de l'activité */
     private  val activityViewModel : ActivityModel by
lazy{ ViewModelProvider(requireActivity()).get(ActivityModel::class.java)}

}
```

Notez que le paramètre de ViewModelProvider différent dans l'activité et dans le fragment

```
-------
le ViewModel :
class ActivityModel :  ViewModel() {

    val compteurs = MutableLiveData( mutableListOf(0,0,0)) /* les compteurs de trois
fragments*/
    var indexOnFrameLayout : Int = 0
    var indexOffFrameLayout : Int = 0

}
```

le fragment -- changement de configuration

L'activité observe les changement de compteurs de trois fragments :

```
class MainActivity : AppCompatActivity() {

   val model : ActivityModel by
lazy{ ViewModelProvider(this).get(ActivityModel::class.java)  }


    override fun onCreate(savedInstanceState: Bundle?) {

        model.compteurs.observe(this){
           val v = it[0] + it[1] + it[2]
           binding.somme.text = "$v"
      /* le compteur global est la somme de trois compteurs */
     }
.......
}
```

---

communication fragment - activité deuxième solution

Le fragment met à jour la valeur de son compteurs dans le ModelView de l'activité :

```
class ButtonFragment : Fragment(R.layout.main_fragment) {

private lateinit var binding : MainFragmentBinding

private  val activityViewModel : ActivityModel by
lazy{ ViewModelProvider(requireActivity()).get(ActivityModel::class.java
)}

private var numeroFragment : Int = 0
```

communication fragment - activité deuxième solution

dans le fragment :

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    /* créer le binding */
    binding = MainFragmentBinding.bind( view )
    binding.button.text = requireArguments()
                .getString("buttonText", "invalide") //?: "empty"
    numeroFragment = requireArguments().getInt("numeroFragment")

    var l = activityViewModel.compteurs.value![numeroFragment]
    binding.compteur.text="$l"

    /* installer le listener sur le bouton */
    binding.button.setOnClickListener{
        /* récupérer la liste de valeurs de compteurs */
        val list = activityViewModel.compteurs.value
        var l = ++ list![numeroFragment]

         activityViewModel.compteurs.value = list.toMutableList()
         binding.compteur.text = "$l"
    }
}
```

communication fragment - activité deuxième solution

dans le fragment :

```
override fun onDetach() {
    super.onDetach()
    binding.button.setOnClickListener(null)
}

/* la fonction utilisée pour créer le fragment */
companion object {
    fun newInstance(buttonText: String, numeroFragment: Int): Fragment {
        return ButtonFragment().apply {
            arguments = Bundle().apply {
                putString("buttonText", buttonText)
                putInt("numeroFragment", numeroFragment)
            }
        }
    }
}
```

A retenir : le constructeur ButtonFragment sans paramètre.
Les données pour initialiser le fragment sont mises dans un Bundle *arguments*

où l'activité place le fragment ?

Dans le fichier **layout de l'activité prévoir l'emplacement pour les fragments dynamiques :**

```
<androidx.fragment.app.FragmentContainerView
    android:id="@+id/five"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    ....
/>
```

On place le fragment dans un container spécialisé FragmentContainerView.

Si on veut qu'un fragment spécifique soit mis dans ce container on peut le spécifier avec l'attribut name :

```
<androidx.fragment.app.FragmentContainerView
    android:id="@+id/five"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:name="fr.irif.zielonka.MainFragment"
    ....
/>
```

Layout de l'activité

Google recommande de placer le fragment dans

`androidx.fragment.app.FragmentContainerView`

Mais si on utilise un composant de android dans ce layout il vaut mieux remplacer d'autres composants de layout par ceux de androidx.
sous peine de problèmes.

androidx.appcompat.widget.LinearLayoutCompat	**à la place de LinearLayout**
androidx.gridlayout.widget.GridLayout	**à la place de GridLayot**
androidx.appcompat.widget.AppCompatRadioButton	à la place de RadioButton etc

Mais il y a de views qui n'ont pas de correspondant dans androidx et dans ce cas on utilise les views habituels.

FragmentManager (dans Activity)

FragmentManager gère les opérations sur les fragments : ajouter, remplacer, supprimer. **supportFragmentManager** est le FragmentManager de AppCompatActivity.

Dans l'activité on regroupe plusieurs opérations sur les fragments à l'aide d'une transaction.

```
val transaction = supportFragmentManager.beginTransaction()
                  .add( R.id.five, /* id de FragmentContainerView
                                    * où on place le fragment */
                        ButtonFragment.newInstance( "button A", 5), /* le
fragment */
                        "A" ) /* un String,  le tag du fragment */
```

Le tag est facultatif, mettre null s'il n'est pas utilisé.

On peut ensuite ajouter la transaction dans le backstack :

`transaction.addToBackStack( tag_transaction )` tag_transaction est un String qui identifie la transaction, null si pas utilisé

`transaction.commit()` soumettre la transaction au TransactionManager

FragmentManager

Une transition peut enchaîner plusieurs opérations :

```
supportFragmentManager.beginTransaction()
                      .add(...).replace(...).add(...).remove(...)
                      .addToBackStack(null)
                      .commit()
```

FragmentManager

il existe aussi une nouvelle syntaxe de transaction :

```
supportFragmentManager
    .commit{
    val fragment = ButtonFragment.newInstance( label, numeroBouton )
    val id = R.id.two /* id dun FragmentContainerView */
    val tag = "tag2"  /* me tag de fragment */
    add( id ,  fragment, tag )
    setReorderingAllowed(true)
    addToBackStack(null)
}
```

setReorderingAllowed(true) est important s'il y a des animations.

FragmentTransaction possède les méthodes **show(fragment : Fragment)**
et **hide(fragment: Fragment)** qui rendent les fragments (le View du fragment) visible/non-visible

FragmentManager

Le tag d'un fragment permet de tester si le fragment est dans FragmentManager :

```
val fragment = supportFragmentManager.findFragmentByTag( "A")
if( fragment == null ){
    /* pas de fragment dont le taf est "A" dans
     *supportFragmentManager */
else{
    /*Le fragment recherché est dans supportFragmentManager */

}
```

Dans mon code ja lance un dialogue si l'utilisateur essaie d'installer un fragment avec le même tag :

```
val fragment = supportFragmentManager.findFragmentByTag( "tag5")

if(fragment != null ){
    AlertDialog.Builder(this)
        .setPositiveButton("OK"){ d, _ -> d.dismiss() }
        .setMessage("le fragment est déjà affiché").show()
    return@OnClickListener
}

/* ce fragment de code se trouve dans OnClickListener */
```