Laboratoire 6

Les fragments

Objectifs d'apprentissage

- Créer et utiliser les fragments
- Comprendre les fragments statiques
- Comprendre les fragments dynamiques
- Comprendre le cycle de vie des fragments
- Comprendre la communication inter-fragments
- Cibler différents périphériques dans une même application
  (Réf. https://developer.android.com/guide/components/fragments.html)

Un fragment représente une portion d'interface que vous pouvez attacher à une activité, il a son propre cycle de vie, gère ses propres événements. Ils peuvent être ajoutés aux activités de façon statique ou de façon dynamique. Vous pouvez combiner plusieurs fragments dans une activité. Les fragments ont été introduits sur Android avec l'arrivée des tablettes sur le marché. Ils permettent de rendre les applications compatibles à la fois sur les smartphones et sur les tablettes. Ils ont permis de mieux exploiter la grande surface offerte par les tablettes.

Un fragment peut être créé en dehors de toute activité mais lui est étroitement lié une fois qu'il y est attaché à cette activité. Son cycle de vie devient dépendant du cycle de vie de l'activité hôte. Le schéma ci-dessous définit le cycle de vie d'un fragment que vous pouvez comparer au cycle de vie d'une activité.

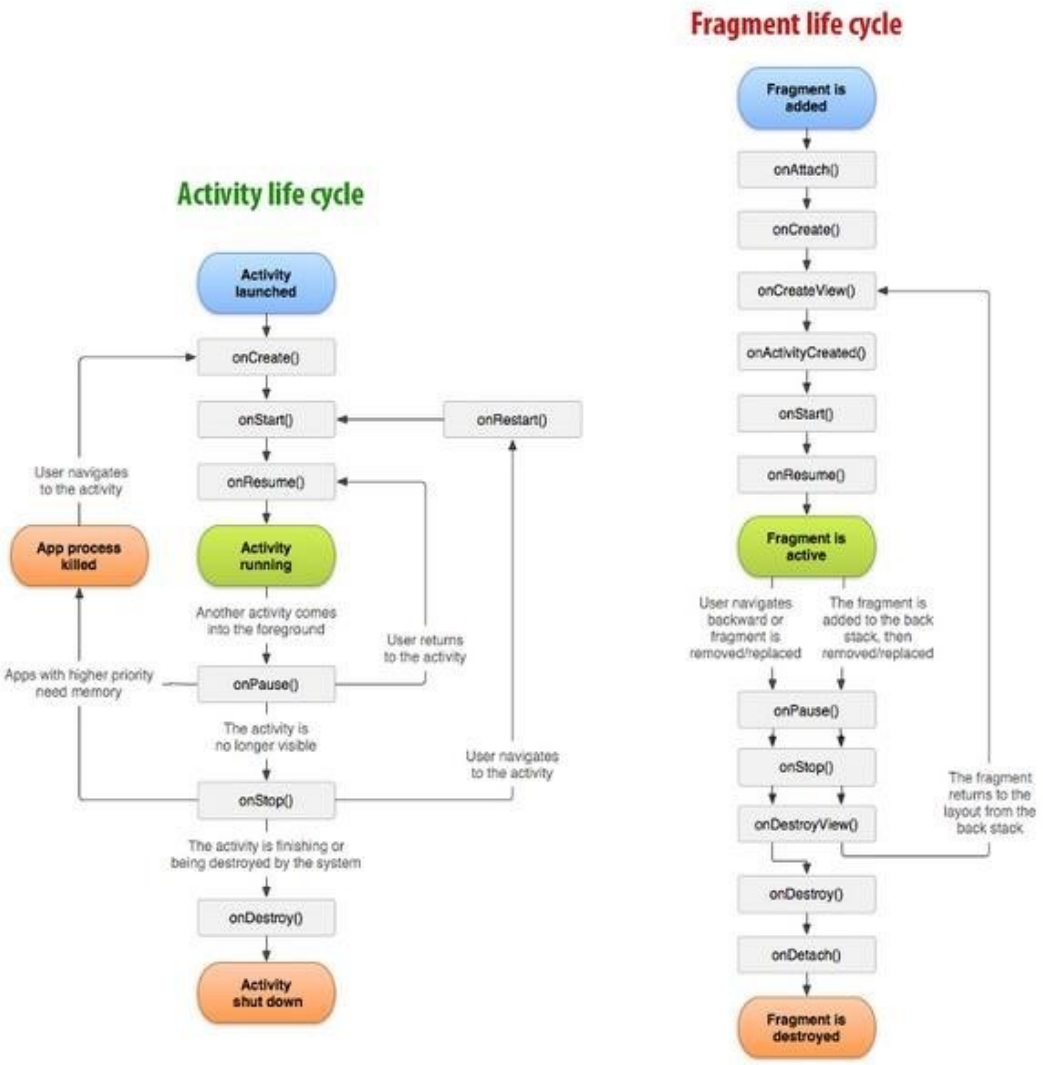
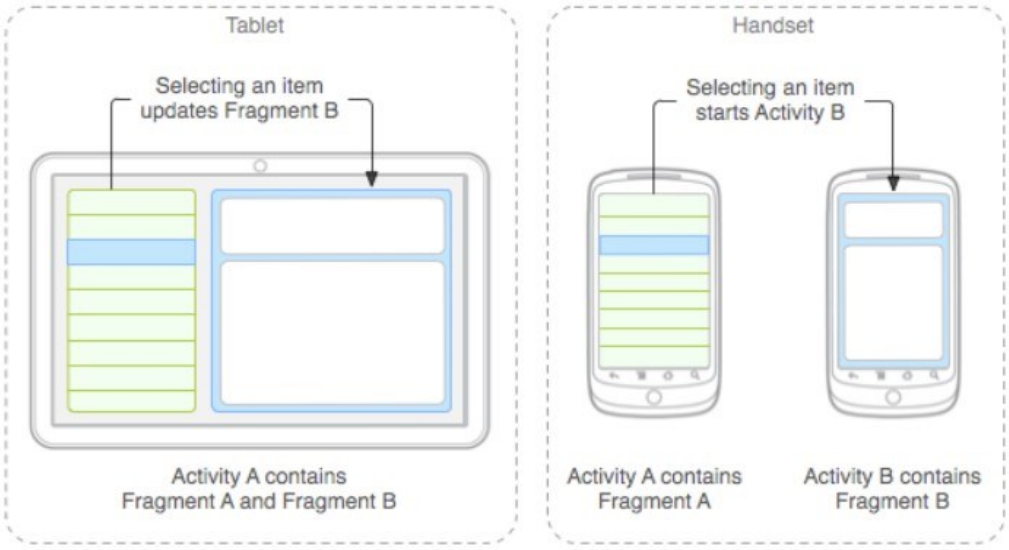


Figure 1: Cycle de vie des Fragments et des activités

Les fragments permettent une réutilisation plus facile dans différentes mises en page. Par exemple, vous pouvez construire des mises en page avec un seul panneau pour des téléphones et plusieurs panneaux pour des tablettes. Ce n'est pas limité aux tablettes, vous pouvez aussi supporter les différentes mises en page suivant l'orientation (portrait ou paysage) du téléphone. Sur une tablette, l'activité principale contient les deux fragments dans sa mise en page, sur un téléphone, elle ne contient que l'activité principale.



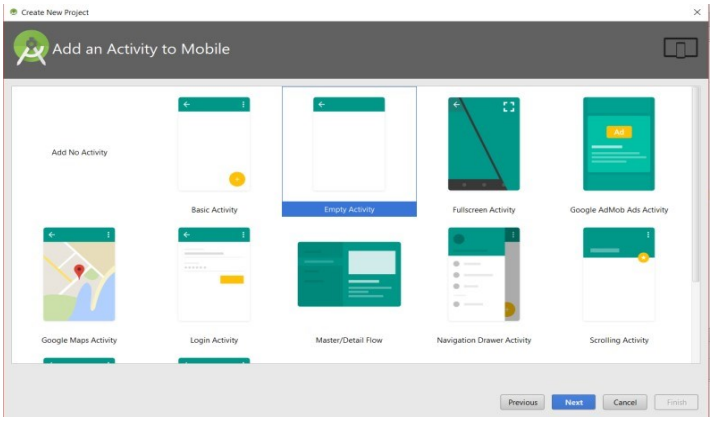
Création d'un Fragment

Pour créer un fragment, vous devez créer une classe héritant de la classe Fragment. La classe Fragment a un ensemble de méthodes de callbacks similaires à la classe Activity, telles que onCreate(), onStart(), onPause() et onStop(). Généralement, vous n'aurez qu'à redéfinir les méthodes onCreate(), onCreateView(), onPause(), onStop(). Il existe d'autres classes héritant de la classe Fragment comme DialogFragment, ListFragment, PreferenceFragment que vous pouvez utiliser aussi suivant vos besoins.

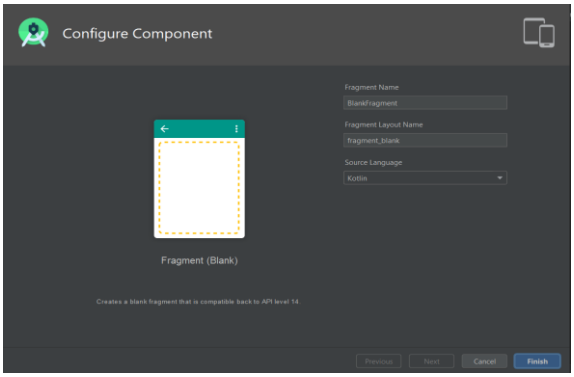
Fragment statique

Dans ce premier exemple que vous allez mettre en place, vous allez utiliser les fragments de façon statique. Vous allez en ajouter deux à votre activité MainActivity.

Créez un projet Laboratoire 6 dans Android Studio en choisissant l'option "Empty Activity".



Créez deux fragments (ColoringFragment et ColoredFragment) en cliquant sur File -> New -> Fragment (Blank) pour ce faire. Vous allez les ajouter de façon statique plus tard à votre activité.



Modifiez le fichier de layout de l'activité en lui ajoutant deux fragments (ColoringFragment et ColoredFragment) dans le sens de la verticale. Le premier occupant la première partie de l'écran(haut) et le deuxième occupant la deuxième partie (bas). Donnez un **identifiant *coloringFragment*** au fragment du haut et ***coloredFragment*** au fragment du bas. Vos fragments doivent remplir la largeur de l'écran. Votre fichier de Layout ressemblera au contenu ci-dessous.

```
<?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">

    <fragment
        android:id="@+id/coloringFragment"
        android:name="ca.qc.cgodin.laboratoire6.ColoringFragment"
        android:layout_width="0dp"
        android:layout_height="300dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <fragment
        android:id="@+id/coloredFragment"
        android:name="ca.qc.cgodin.laboratoire6.ColoredFragment"
        android:layout_width="0dp"
        android:layout_height="300dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/coloringFragment" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Ajoutez les méthodes de rappel additionnelles aux deux fragments et à l'activité hôte MainActivity que vous observez sur la figure 1 (Cycle de vie des fragments et des activités).
Ajoutez des Logs dans les méthodes de rappel des fragments ainsi que l'activité hôte pour afficher les noms des méthodes appelées avec les noms de classes correspondantes comme aux labos précédents.
Lancez l'application et observez les appels aux méthodes de callback.

Communication entre les fragments

Pour permettre à deux fragments de pouvoir communiquer entre eux, il est très important de passer par l'activité hôte.

Vous allez dans cette partie modifier le fragment du haut que nous appelons fragment colorant (coloringFragment) pour envoyer un message au fragment du bas que nous appelons fragment coloré (coloredFragment). Quand l'utilisateur va cliquer sur le layout racine (ConstraintLayout) du **fragment colorant**, vous allez envoyer un message au fragment coloré pour lui appliquer une couleur aléatoire provenant du **fragment colorant**. Pour appliquer le principe du couplage faible, vous allez passer par l'activité hôte pour envoyer un message du **fragment colorant** au **fragment coloré**. En vertu de ce principe, vous ne devez **jamais** créer un objet fragment à l'intérieur d'un autre fragment pour lui envoyer un message.

Pour ce faire, vous devez créer une interface dans le fragment colorant qui devra être implémentée par toute activité hébergeant ce fragment pour permettre à un autre fragment de lui envoyer un message comme suit :

```
interface OnColoringFragmentInteractionListener{
    fun onSendColorFragmentInteraction()
}
```

Ajoutez les variables suivantes au début de la classe ***ColoringFragment***

```
private lateinit var mListener: OnColoringFragmentInteractionListener
private val TAG = ColoringFragment::class.java.canonicalName
```

Pour obliger l'activité hôte à implémenter cette interface, redéfinissez la méthode de Callback ***onAttach*** comme suit dans le fragment colorant. Le casting impose sa volonté d'implémentation de l'interface à l'activité.

```
override fun onAttach(context: Context) {
    super.onAttach(context)
    Log.i(TAG, "${javaClass.simpleName}: entered ${object
{}.javaClass.enclosingMethod.name}")
    try {
        // Set the OnColoringFragmentInteractionListener for communicating with
the hosting Activity
        mListener = context as OnColoringFragmentInteractionListener

    } catch (e: ClassCastException) {
        throw ClassCastException("$context must implement
OnColoringFragmentInteractionListener")
    }
}
```

Créez les variables sivantes dans ColoringFragment:

```
private lateinit var mListener: OnColoringFragmentInteractionListener
private val TAG = ColoringFragment::class.java.canonicalName
private lateinit var coloringFragmentLayout: FrameLayout
```

Votre objectif premier est de colorer le fragment de bas à partir du fragment de haut en cliquant sur le layout racine de ce dernier. Vous devez pour ce faire ajouter un listener sur le layout racine du fragment colorant pour envoyer un message au fragment coloré. Dans la fonction onCreateView, vous devez récupérer le layout attaché au fragment et ensuite lui ajouter un listener pour envoyer un message au fragment coloré. Modifiez la fonction onCreateView comme suit:

```
override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?
): View? {
    // Inflate the layout for this fragment
    val coloringFragmentLayout = inflater.inflate(R.layout.fragment_coloring, container,
false)

    coloringFragmentLayout.setOnClickListener {
        mListener.onSendColorFragmentInteraction(Random.nextInt())
    }
    return coloringFragmentLayout
}
```

Dans la classe MainActivity représentant l'activité hôte dans ce cas précis, créez un champ coloringFragment de type ColoringFragment et un champ coloredFragment de type ColoredFragment.

```
private lateinit var coloringFragment: ColoringFragment
private lateinit var coloredFragment: ColoredFragment
```

La classe MainActivity doit implémenter l'interface OnColoringFragmentInteractionListener de coloringFragment. Modifiez la déclaration de la classe MainActivity comme suit :

```
class MainActivity : AppCompatActivity(),
ColoringFragment.OnColoringFragmentInteractionListener {
```

Dans la classe MainActivity, implémentez la méthode onSendColorFragmentInteraction comme suit :

```
override fun onSendColorFragmentInteraction(bgColor: Int) {
    Log.i(javaClass.simpleName, "onSendColorFragmentInteraction")
    coloredFragment.setLayoutBackgroundColor(bgColor)
}
```

Initialisez les objets coloringFragment et coloredFragment en ajoutant les deux lignes suivantes dans la méthode onCreate:

```
coloringFragment = supportFragmentManager.findFragmentById(R.id.coloringFragment)
as ColoringFragment
coloredFragment = supportFragmentManager.findFragmentById(R.id.coloredFragment)
as ColoredFragment
```

Dans le fragment coloré (ColoredFragment), implémentez la méthode ***setLayoutBackgroundColor***:

```
fun setLayoutBackgroundColor(color: Int) {
    coloredFragmentLayout.setBackgroundColor(color)
}
```

Dans la classe ColoredFragment, créez le champ coloredFragmentLayout

```
private lateinit var coloredFragmentLayout: View
```

Initialisez l'objet coloredFragmentLayout dans la méthode onCreateView en la modifiant comme suit

```
override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?
): View? {
    // Inflate the layout for this fragment
    coloredFragmentLayout = inflater.inflate(R.layout.fragment_colored,
container, false)
    return coloredFragmentLayout
}
```

Placez un bouton "Remerciements" dans le fragment coloré pour envoyer un message au fragment colorant.

En vous inspirant de l'approche de communication du fragment colorant au fragment coloré, faites l'inverse en envoyant un message du fragment coloré au fragment colorant affichant dans ce dernier un message "Merci de m'avoir coloré".

Modifier le programme pour que ce bouton soit actif dans le fragment coloré après avoir été effectivement coloré par le fragment colorant.

Ajoutez une liste de 7 couleurs au fragment de haut en utilisant un spinner. Modifiez le programme pour que quand l'utilisateur sélectionne une couleur que le fragment de bas prenne cette couleur.

II. Fragment dynamique

Ajoutez au projet une autre activité ***DynamicHostActivity*** en utilisant le ***template Empty Activity***.

Dans le layout de cette nouvelle activité, ajoutez deux layout linéaires partageant l'écran en deux parties égales, un en haut et un en bas en mode portrait (une à droite et une à gauche en mode paysage), l'un ayant une couleur de fonds différente de l'autre. Le xml résultant s'apparente à ce qui suit:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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"
    android:orientation="vertical">

    <LinearLayout
        android:id="@+id/topLinearLayout"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_gravity="center_horizontal"
        android:layout_weight="1"
        android:background="@color/colorAccent"
        android:orientation="vertical">
    </LinearLayout>

    <LinearLayout
        android:id="@+id/bottomLinearLayout"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_gravity="center_horizontal"
        android:layout_weight="1"
        android:background="@android:color/holo_orange_dark"
        android:orientation="vertical">
    </LinearLayout>
</LinearLayout>
```

Ajoutez ensuite 2 objets fragments à votre activité:

```
private lateinit var dynamicColoredFragment: ColoredFragment
private lateinit var dynamicColoringFragment: ColoringFragment
```

Initialisez les fragments comme suit à la fin de la méthode onCreate():

```
dynamicColoringFragment = ColoringFragment.newInstance("", "")
dynamicColoredFragment = ColoredFragment.newInstance("", "")
```

Utilisez l'objet supportFragmentManager pour créer un objet de type FragmentTransation vous permettant de manipuler les fragments (ajout, remplacement, suppression, etc.). Ajoutez les lignes suivantes à la fin de la fonction onCreate() pour ajouter les deux fragments de façon dynamique à votre interface.

```
val fragmentTransaction =  supportFragmentManager.beginTransaction()
fragmentTransaction.add(R.id.topLinearLayout, dynamicColoringFragment)
fragmentTransaction.add(R.id.bottomLinearLayout, dynamicColoredFragment)
fragmentTransaction.commit()
```

Modifiez l'entête de la classe DynamicHostActivity comme suit:

```
class DynamicHostActivity : AppCompatActivity(),
ColoringFragment.OnColoringFragmentInteractionListener {
```

Le fragment colorant va passer aussi par cette activité pour pouvoir communiquer avec le fragment coloré comme dans le cas des fragments ajoutés statiquement. Vous devez pour ce faire implémenter aussi la méthode onSendFragmentInterAction dans DynamicHostActivity comme suit:

```
override fun onSendColorFragmentInteraction(color: Int) {
    dynamicColoredFragment.setLayoutBackgroundColor(color)
}
```

Pour pouvoir accéder aux deux interfaces statiques et dynamiques, on va se créer une activité d'accueil qui va servir comme point d'entrée à notre application en lieu et place du ***MainActivity*** utilisé actuellement.

Changez le nom de l'activité ***MainActivity*** en ***StaticHostActivity*** pour être plus significatif de ce qu'on est en train de faire. Faites un refactor.

Créez une nouvelle activité ***WelcomeActivity*** en utilisant le ***template Empty Activity***. Ajoutez-lui deux boutons pour vous permettre d'accéder aux interfaces ***StaticHostActivity*** et ***DynamicHostActivity*** respectivement. Ajoutez les listeners aux boutons pour accéder aux activités susmentionnées.

Modifiez le fichier ***AndroidManifest*** en faisant déplacer le filtre d'intention de ***StaticHostActivity*** à ***WelcomeActivity*** pour que ***WelcomeActivity*** soit lancé au démarrage de votre application. Vous aurez ce qui suit:

```
 <activity android:name=".WelcomeActivity" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity android:name=".DynamicHostActivity" />
<activity android:name=".StaticHostActivity">
</activity>
```

Lancez votre application.

Les deux fragments doivent s'afficher correctement quand vous cliquez sur les boutons vous amenant aux fragments créés de façon statique et dynamique.

Remplacement d'un fragment par un autre fragment

Ajoutez un troisième fragment ***ReplacingFragment*** en utilisant le ***template Blank Fragment*** au programme. Ce fragment va remplacer le fragment coloré quand vous cliquez sur un bouton que vous allez ajouter au fragment colorant.

Modifiez le fichier coloring_fragment pour lui ajouter un nouveau bouton btnReplaceFragment vous permettant de remplacer le fragment du bas par un objet de type ReplacingFragment

```
<Button
    android:id="@+id/btnReplaceFragment"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:visibility="gone"
    android:text="@string/btn_change_fragment_txt" />
```

Ajoutez la variable suivante à la classe ColoringFragment:

```
private lateinit var btnReplace: Button
```

Ajoutez la fonction suivante à la classe ColoringFragment:

```
fun enableReplaceFragmentButton() {
    btnReplace.visibility = View.VISIBLE
}
```

Dans la fonction ***onCreateView*** de la classe ***ColoringFragment***, ajoutez le code suivant pour initialiser le bouton btnReplace et lui ajouter un listener qui vous permettra d'afficher le bouton dans l'interface ***ColoringFragment*** seulement dans le cas des fragments ajoutés dynamiquement.

```
btnReplace = coloringFragmentLayout.findViewById<Button>(R.id.btnReplaceFragment)
btnReplace.setOnClickListener {
    mListener.onChangeFragment()
}
```

Ajoutez la fonction onChangeFragment à l'interface ***OnColoringFragmentInteractionListener*** créé dans la classe ***ColoringFragment*** pour avoir ce qui suit:

```
interface OnColoringFragmentInteractionListener{
    fun onSendColorFragmentInteraction(nextInt: Int)
    fun onChangeFragment()
}
```

Les activités hôtes ***StaticHostActivity*** et ***DynamicHostActivity*** doivent désormais implémenter la fonction ***onChangeFragment***.

Ajoutez la fonction ***onChangeFragment*** comme suit à la classe ***DynamicHostActivity***:

```
override fun onChangeFragment() {
    val replacingFragment = ReplacingFragment.newInstance("", "")
    val fragmentTransaction = supportFragmentManager.beginTransaction()
    fragmentTransaction.replace(R.id.bottomLinearLayout, replacingFragment)
    fragmentTransaction.addToBackStack(null)
    fragmentTransaction.commit()
}
```

Ajoutez la fonction onChangeFragment comme suit à la classe ***StaticHostActivity***:

```
override fun onChangeFragment() {

}
```

Ajoutez la fonction de rappel ***onResume*** à la classe ***DynamicHostActivity*** et modifiez la comme suit:

```
override fun onResume() {
    super.onResume()
    dynamicColoringFragment.enableReplaceFragmentButton()
}
```

Remarquez que l'appel à la fonction ***enableReplaceFragmentButton*** est bien faite dans onResume et non pas dans onCreate. Faites un test en essayant de le placer dans la fonction ***onCreate*** et observez ce qui se passe.

Remarquez aussi l'utilisation de la fonction ***addToBackStack*** à la fonction ***onChangeFragment***. La fonction ***addToBackStack*** vous permet d'ajouter une transaction à la pile (stack) de transaction.

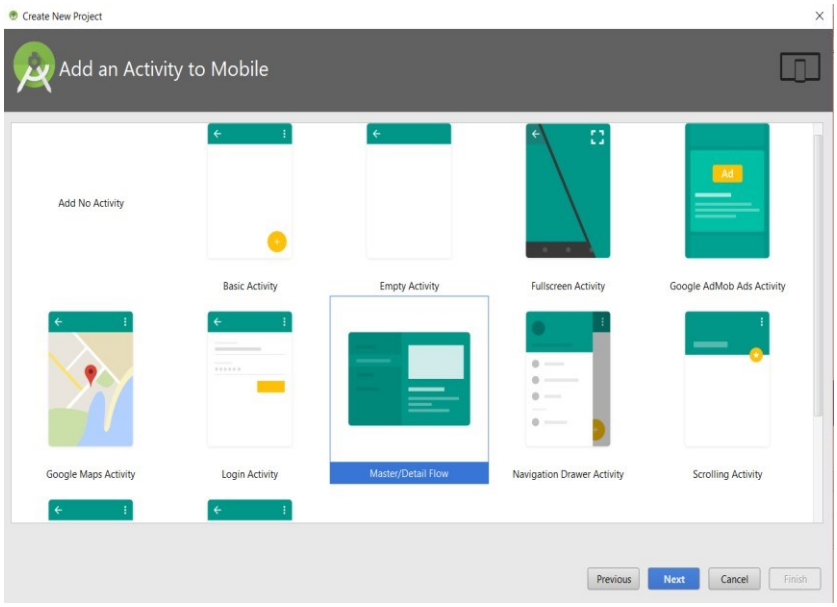
Mettez cette ligne en commentaire, relancez l'application. Testez en cliquant sur le bouton ***Change Fragment*** pour remplacer le fragment du bas (***DynamicColoredFragment***) par le fragment remplaçant (***ReplacingFragment***) et ensuite cliquez sur le bouton ***Retour***. Remarquez la différence par rapport à l'utilisation de la fonction ***addToBackStack***. Remarquez aussi que si vous cliquez plusieurs fois sur le bouton ***Change Fragment***, il vous ajoute le fragment à plusieurs reprises.

Modifiez l'application pour empêcher cet ajout multiple d'un même fragment ReplacingFragment.


18:37
GG5PAFragmentsLabo7
CHANGE FRAGMENT
Hello from Replacing Fragment

III. Fragment Master-Détail

Dans cette troisième partie du labo, vous allez créer une application qui va vous permettre de faire la communication entre un RecyclerView (un widget vous permettant de manipuler des listes d'éléments) et un fragment. Dans ce cas-ci, vous allez utiliser le template master-detail. Les RecyclerViews sont une version un peu plus évoluée et plus performante des ListView que vous pouvez encore voir dans la littérature ou d'anciennes applications. Créez un projet Labo6MasterDetail en choisissant le template master-detail



Laissez les options par défaut et cliquez sur Finish pour créer l'application.

Le programme, tel qu'il est créé au départ, vous permet de manipuler des objets bidons appelés Item. Exécutez le programme sur une tablette et cliquez sur les items à gauche et les informations s'afficheront dans la partie de droite. Exécutez l'application sur un téléphone et remarquez que vous aurez une nouvelle fenêtre pour afficher les mêmes détails qui étaient affichés à droite sur la tablette.

Dans ce programme, vous avez trois classes avec pour activité hôte ***ItemListActivity***. Vous avez un fragment qui s'affiche soit dans la partie droite dans le cas d'une tablette et dans l'activité ***ItemDetailActivity*** dans le cas d'un smartphone.

Les items qui sont manipulés sont placés dans le sous-package ***dummy***. Une classe ***DummyContent*** et une classse interne ***DummyItem***. Les informations de la classe DummyItem sont les suivantes : id, content, details.

Essayez de comprendre l'application construite en utilisant le template.

Créez un sous package encapsulant une classe de données (data class) Person vous permettant de manipuler des personnes à la place des ***DummyItems***. Vous pouvez vous inspirer de l'approche de manipulation des ***DummyItem***. Les champs de la classe Person sont les suivants : id, firstName, lastName, phoneNumber, email, address, city, country.

Dans le fragment de gauche, affichez seulement les noms et prénoms des personnes. Dans le fragment de détail, affichez toutes les informations sur la personne sélectionnée dans le fragment de gauche dans des EditText.

Bon travail !