Android TP 3 :
Les fragments

Dans ce TP nous allons apprendre à créer une application contenant plusieurs écrans et à passer d'un écran à l'autre. Nous verrons, pour cela, la notion de fragment.

Les fragments permettent de scinder une activité en composants encapsulés dans l'activité, réutilisables qui possèdent leur propre cycle de vie et leur propre interface graphique. Cela permet de mettre en place une I.H.M. évoluée munie d'un menu et qui s'adapte aux différents écrans et à leur orientation.

Quelques notions élémentaires concernant les fragments :

- Ils déportent une partie du traitement de l'activité en leur sein,
- Ils sont liés à une activité (ils n'existent pas sans elle),
- Ils définissent la plupart du temps une interface graphique,
- Ils peuvent être statiques (définis une fois pour toutes dans le fichier de layout) ou dynamiques (créés, supprimés, ajoutés dynamiquement) ;

Les classes fondamentales pour la gestion des fragments sont : Fragment, FragmentManager et FragmentTransaction.

Source : https://mathias-seguy.developpez.com/tutoriels/android/comprendre-fragments/

La méthode de développement proposée ici n'est pas la plus à jour, mais elle est relativement accessible. ANDROID STUDIO recommande de passer au « JetPacks », qui modifie considérablement le codage des applications.

1 Création de fragments

1.1 Création du projet

Nous allons créer un projet, dont l'interface comprend deux pages. On passera d'une page à l'autre en cliquant sur deux boutons en bas de l'écran. Le projet sera constitué d'une activité et de deux fragments (Figure 1)

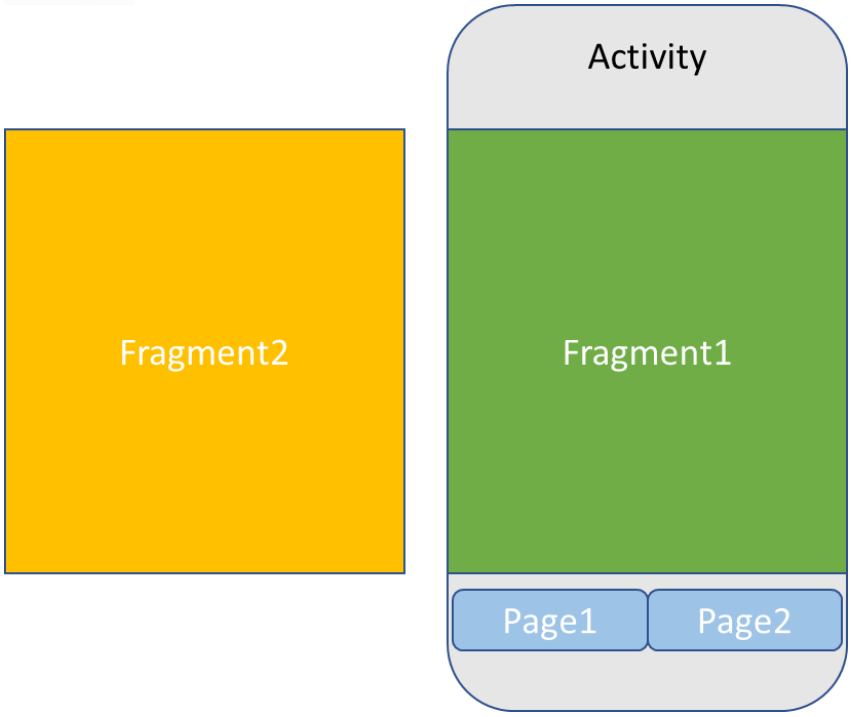

Figure 1 - Cahier des charges

→ Créer un projet nomme TP3FragmentBase avec une Activité vide. Le ranger dans un dossier convenable.

Nous allons utiliser le « package » le plus récent des API d'Android Studio, nommé AndroidX.

→ Pour cela, aller dans le menu « Refactor » → « Migrate to AndroidX ». Vérifier que le fichier de compilation a été mis à jour, c'est-à-dire que le package AndroidX est bien déclaré (Figure 2) :

Figure 2 - Mise à jour du compilateur

→ Créer deux boutons « page1 » et « page2 » en bas de l'écran, de sorte d'obtenir ceci (Figure 3):

Figure 3 - Deux boutons

→ Dans le fichier JAVA, déclarer les deux objets « button » ainsi que les deux « Listener » qui seront exécutés en cas de « clique » sur un des boutons.

→ Compiler et exécuter pour vérifier qu'il n'y a pas d'erreur.

→ Le petit truc en plus, si on en a envie : pour retirer la barre de titre, ajouter les deux lignes ci-dessous dans le fichier XML de « theme » (Figure 4):

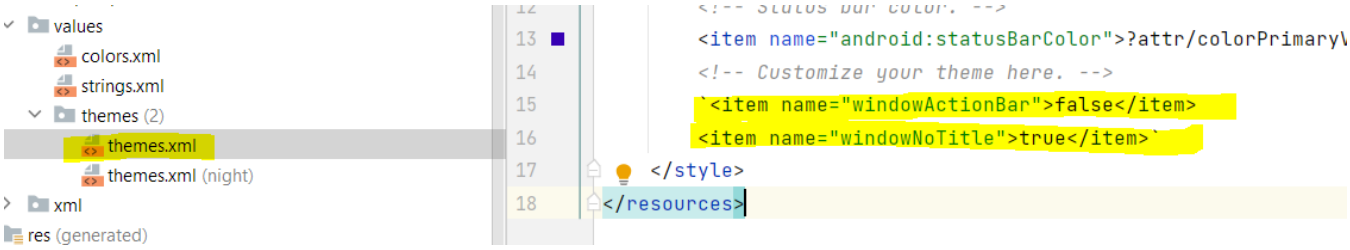

Figure 4 - Suppression de la barre de titre

→ Compiler et vérifier. Jeter un coup d'œil au fichier de style, cela peut donner des idées…

1.2 Création des fragments

Un fragment est une vue, autonome, notamment quand à son cycle de vie (la façon dont elle s'exécute, lancement, arrêt, relancement…). Elle est indépendante de l'activité, mais pourra s'afficher dans celle-ci. Dans notre application, nous allons utiliser deux fragments qui porteront chacun une page de celle-ci.

→ Créer un 1er fragment comme ci-dessous (Figure 5) :

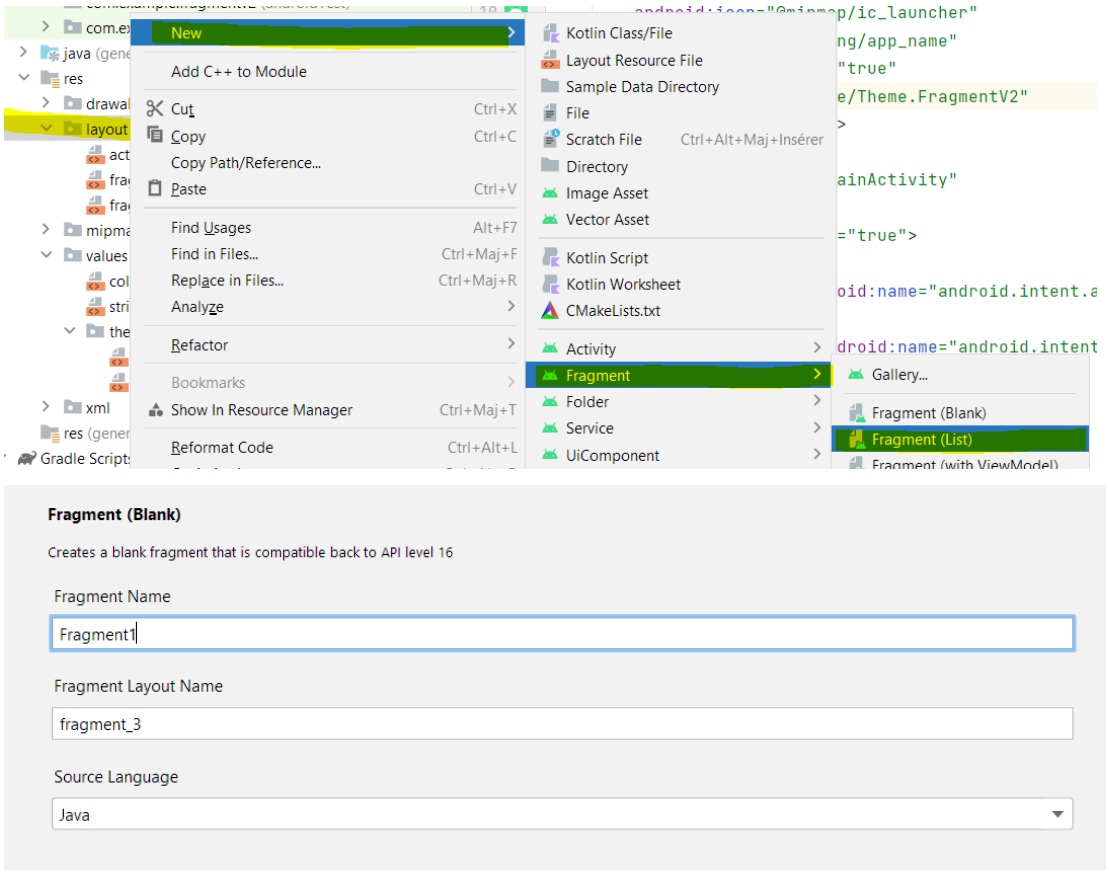

Figure 5 - Création du fragment

Deux fichiers sont créés, un fichier XML pour décrire le contenu de la page, un fichier JAVA pour l'animer (Figure6).

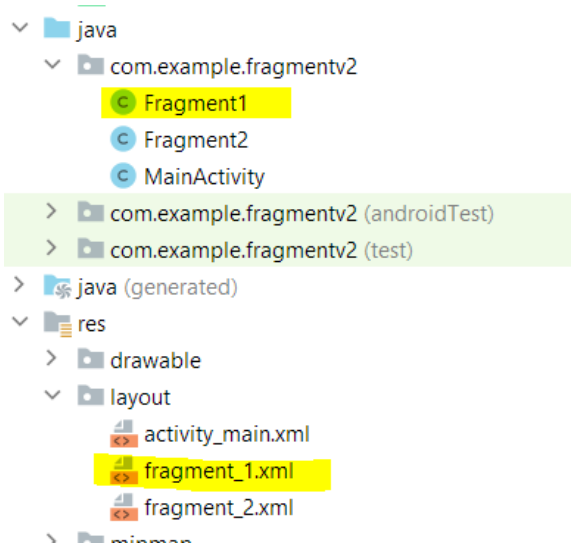

Figure 6 - Structure du projet

→ Créer un 2ème fragment comme nommé « Fragment2 ».

→ Modifier les fichiers XML de chaque fragment pour ajouter une couleur de fond, au choix, mais différente pour chaque page (`android:background="#FFC107"`, pour la couleur orange), et pour que le texte qui apparait au milieu soit respectivement « Page1 » et « Page2 placé à 1 tiers en haut et 1 tiers en bas de la page. Convertir d'abord le « Layout » en « vue contrainte » avec le bouton droit de la souris et utiliser les attributs « habituels » (Figure 7)

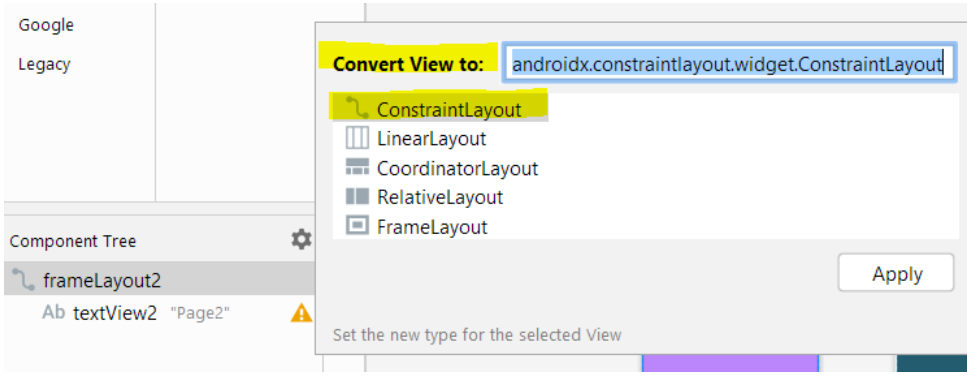

Figure 7 - Conversion de la vue

On obtient ceci (Figure 8):

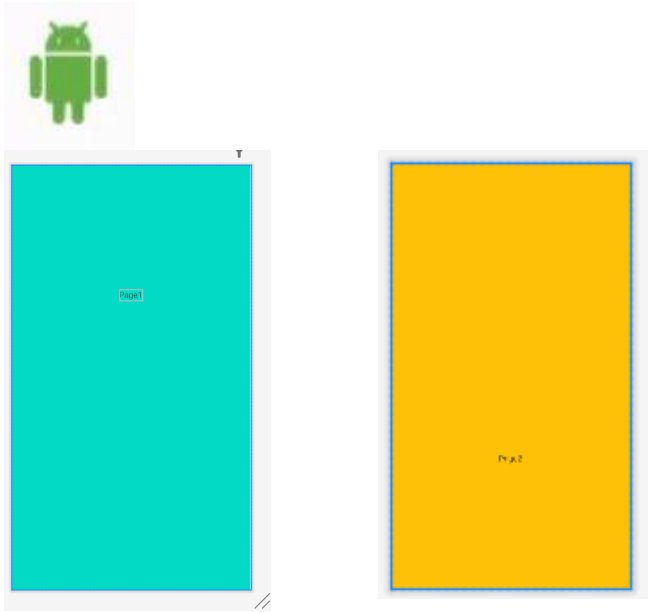

Figure 8 - Résultat...

1.3 Création du « container » dans l'activité

Nous allons maintenant prévoir le « cadre » des fragments dans l'activité. Pour cela on crée une vue dans le fichier XML de l'activité, un des fragments utilisera cette vue lors de son appel. Le widget utilisé ici est le « frameLayout », modèle simple, un peu dépassé…. On remarquera que des règles de placement ont été retenues ici pour bien placer la vue par rapport aux deux boutons.

→ Ajouter le code ci-dessous dans le fichier Activite.XML…. et le comprendre. Un seul cadre suffit.

```
<FrameLayout
    android:id="@+id/frame"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    app:layout_constraintBottom_toTopOf="@+id/bouton2"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

1.4 Appel des pages dans l'activité

Nous ajoutons maintenant le code JAVA dans l'activité pour passer d'une page à l'autre quand on clique sur un des boutons.

→ Ajouter la méthode ci-dessous dans la classe « Main Activity » :

```
private void lanceFragment(Fragment fragment) {
    FragmentManager fm =getSupportFragmentManager();
    FragmentTransaction ft= fm.beginTransaction();
    ft.replace(R.id.frame,fragment);
    ft.addToBackStack(null);
    ft.commit();
}
```

On remarquera que le nom de cette méthode a été librement choisi…, ainsi que le nome des objets mis en œuvre (« fm », « ft »).

Le « FragmentManager » est la classe d'Android Studio qui gère l'ordonnancement des fragments.

Plus d'infos ici : https://developer.android.com/guide/fragments/fragmentmanager?hl=fr

→ Questions à se poser (et y répondre…)

1- Quelle ligne de code « installe » le fragment dans la vue principale ?
2- Une ligne permet le bon fonctionnement du bouton de retour du téléphone, laquelle est-ce ?

→ Déclarer un nouveau fragment dans l'activité :

```
Fragment1 frag1;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    frag1 = new Fragment1();
```

→ Ajouter le code ci-dessous dans chaque « listener » des boutons. Ici le « button » se nomme « page1 ».

```
page1.setOnClickListener(new View.OnClickListener() {
  @Override
    public void onClick(View view) {
    lanceFragment(frag1) ;
   }
});
```

→ Compiler, lancer l'application. Vérifier que la navigation entre les pages fonctionne ainsi que le bouton de retour. Vérifier aussi que la rotation de l'écran s'effectue correctement.

2 Atelier : programmation des fragments

Le but de cet atelier est d'ajouter du code dans les fragments.

Le cahier des charges est le suivant :

Fragment 1 : un bouton sur l'écran incrémente un compteur. La valeur du compteur s'affiche.

Fragment 2 : on saisit un texte qui s'affiche en lettres majuscules en dessous une fois qu'on a validé avec un bouton.

La programmation s'effectue de façon classique. Le code est à placer dans la méthode « OnCreateView » du fragment, que l'on modifiera ainsi.

```
private View vue;
private Button boutonValid;
private Integer compteur;
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    // Inflate the layout for this fragment
   vue= inflater.inflate(R.layout.fragment_1, container, false);

   // code à placer ici
   boutonValid = (Button) vue.findViewById(R.id.validation);

   boutonValid.setOnClickListener(new View.OnClickListener() {
     @Override
```

```
    public void onClick(View view) {
        // ici code à ajouter
 });
   return vue;
}
```

→ Commencer par le fragment 1, vérifier puis par le fragment2 et vérifier !

3 Outils pour la suite … Au choix, la classe Toast ou bien la classe Snack Bar

Android Studio propose deux outils pour afficher des messages temporaires sur l'écran, faciles à mettre en œuvre. Les objets de la classe « Toast », remplacés aujourd'hui (recommandation 2021) par ceux de la classe « SnackBar », plus sophistiqués car le message peut être interactif.

→ On choisira l'une des classes. Le clic sur le bouton du fragment1 déclenche l'affichage d'un message de longue durée au milieu de l'écran. Faire valider par le professeur. Concernant l'objet SnackBar on essaiera d'obtenir une barre interactive.

3.1 Coder la classe « Toast »

Les objets de la classe « Toast » permettent d'afficher des messages temporaires sur l'écran, pour indiquer à l'utilisateur, par exemple, qu'une action vient de se terminer (Figure 9).

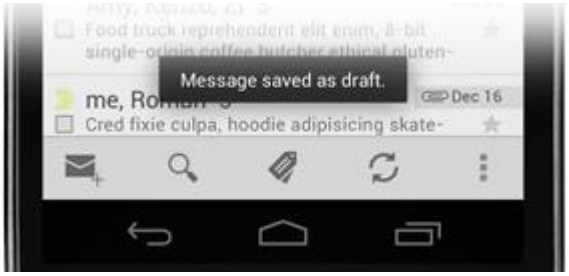

Figure 9 - Message affiché via un objet "Toast"

Figure 10 - Message affiché via un objet "SnackBar"

La manipulation de l'objet « Toast » est assez simple. Elle est bien expliquée dans la documentation d'ANDROID STUDIO (https://developer.android.com/guide/topics/ui/notifiers/toasts.html).

3.2 Coder le SnackBar

La documentation officielle se trouve ici :

https://developer.android.com/training/snackbar/showing

Les tutoriels sont nombreux sur Internet, attention, ils ne sont pas forcément justes et à jour. En voici tout de même un :

https://devstory.net/12707/android-snackbar

1ère étape : la barre doit être affichée dans un layout particulier, ce qui permet d'ailleurs de la positionner où l'on veut sur la vue, contrairement au « Toast » . Ce layout est de la classe «CoordinatorLayout ». Il faut donc, dans le fichier XML déclarer un objet de cette classe et le positionner sur la vue.

L'auto-complétion et le copier/coller aident à ne pas faire d'erreur et à ne par perdre trop de temps.

Par exemple :

```
<androidx.coordinatorlayout.widget.CoordinatorLayout
    android:id="@+id/vueSnackBar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintVertical_bias="0.9"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    />
```

2ème étape : dans le code, là où l'on veut déclencher l'affichage de la barre

On crée un objet de type « SnackBar » en appelant la méthode « make ». On passe en paramètre, la vue, le message à afficher, et la durée d'affichage (LENGTH_SHORT possible).

```
Snackbar snackbar = Snackbar.make(findViewById( R.id.vueSnackBar ), "SnackBar
simple", Snackbar.LENGTH_LONG);
```

3ème étape : à la suite dans le code, on utilise la méthode « show » pour déclencher l'affichage du message.

```
snackbar.show();
```

Encore plus fort : on peut rendre la barre interactive en lui ajoutant un bouton. Pour cela, il faut ajouter un « listener » dans la code et utiliser la méthode « Action » comme par ci-dessous, où l'appui sur le bouton déclenche un message dans le logcat.

```
Snackbar snackbar = Snackbar.make(findViewById( R.id.vueSnackBar ), "SnackBar Avec
Bouton", Snackbar.LENGTH_LONG);

snackbar.setAction("Reset", new View.OnClickListener() {
        @Override
        public void onClick(View view) {

        // ici juste une vérification que cela fonctionne
        Log.i("Appui Bouton SnackBar","Ca Marche");
        }

});
    snackbar.show();
```

4 Atelier : créer une barre de navigation

Nous allons ici créer un projet avec une barre de navigation. Dans un second atelier, nous utiliserons des fragments pour accéder à chaque page.

Cet atelier a été rédigé à l'aide de ce tutoriel :

https://www.youtube.com/watch?v=zQh-QGGKPw0

L'objectif est d'obtenir la barre de menu de la Figure11. Le modèle proposé sur le tutoriel est un peu plus élaboré mais les ajustements à faire ne sont pas très compliqués.

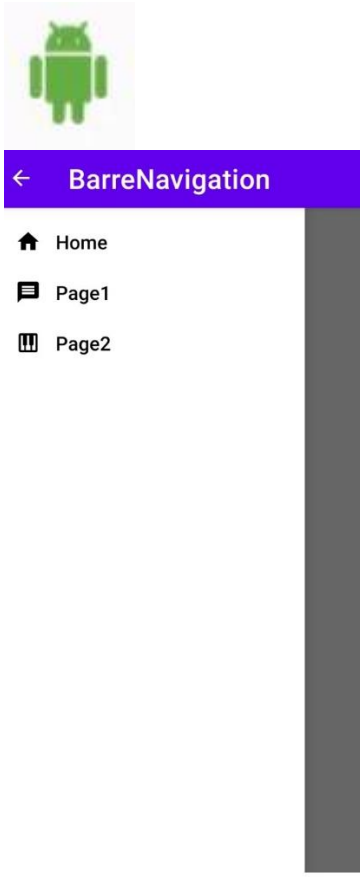

Figure 11 - Menu à obtenir

4.1 Création de l'activité

→ Créer sur le modèle habituel un projet, à partir d'une Activity vide, rangé dans un dossier bien choisi et nomme « BarreNavigation »

→ Ouvrir le fichier Activity.XML et le modifier comme ci-dessous. Les parties à modifier sont en rouge.

```
<?xml version="1.0" encoding="utf-8"?>

<androidx.drawerlayout.widget.DrawerLayout

    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"

    android:id="@+id/drawerLayout"

    tools:context=".MainActivity">

<androidx.constraintlayout.widget.ConstraintLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Activite1"
```

```
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>

    </androidx.constraintlayout.widget.ConstraintLayout>
</androidx.drawerlayout.widget.DrawerLayout>
```

Un peu d'explications :

1- on a modifié le « widget » de la vue de l'activité principale, en remplaçant la vue contrainte par le widget « DrawerLayout »
2- On a ajouté une vue contrainte à l'intérieur de cette vue
3- On a modifié le texte du message de base.

→ Ajouter une vue de Navigation (« Navigation View) à la suite et lui donner un nom (Figure 12)

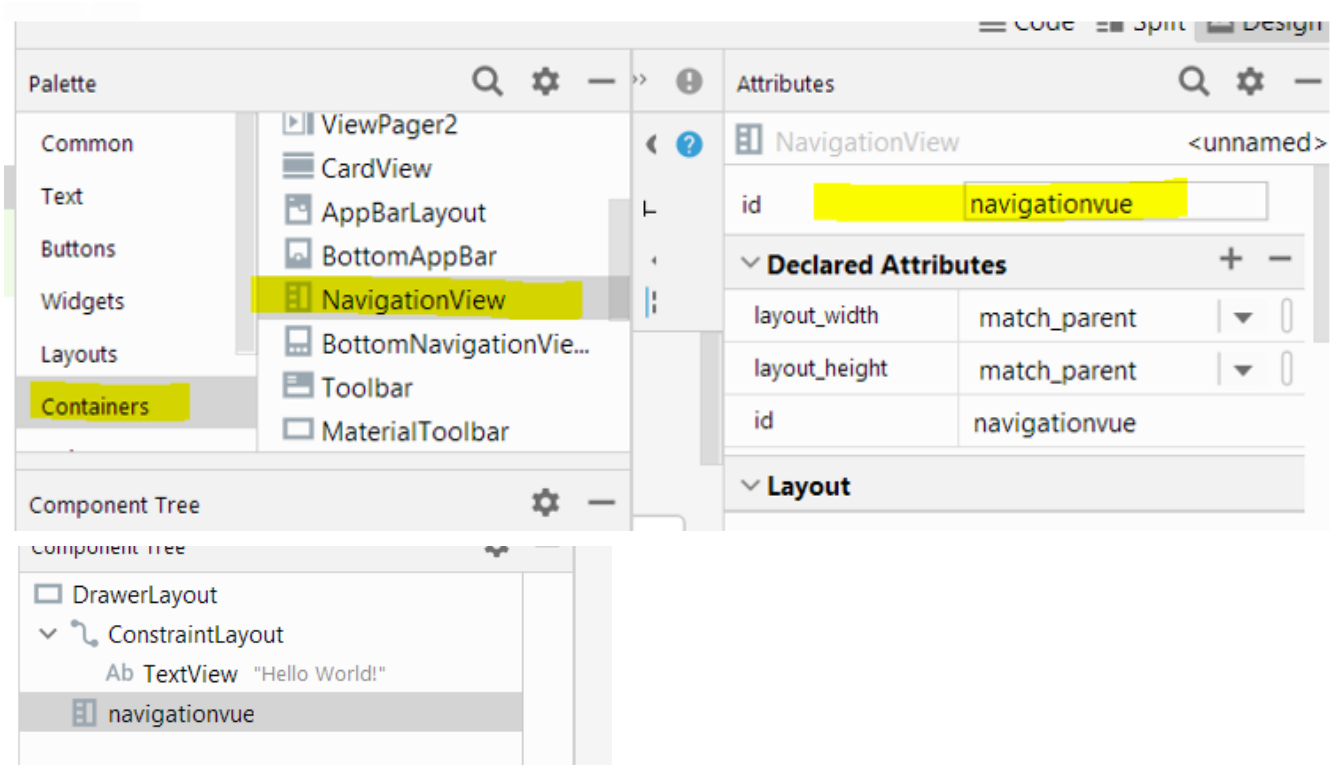

Figure 12 - vue de navigation

→ Compléter le code XML ainsi :

```
    </androidx.constraintlayout.widget.ConstraintLayout>

    <com.google.android.material.navigation.NavigationView
        android:id="@+id/navigationvue"
        android:layout_width="match_parent"
        android:layout_height="match_parent"

        android:layout_gravity="start"
        android:fitsSystemWindows="true" />
```

```
</androidx.drawerlayout.widget.DrawerLayout>
```

La gravité permet d'orienter la barre de navigation dans la vue (ici à gauche), l'attribut « fistSystemWidows » …….

→ Compiler et tester l'application. L'écran s'affiche en blanc, si on fait glisser le doigt de la gauche vers la droite, la barre de navigation apparait.

4.2 Conception du menu

→ S'il n'existe pas, créer un sous-dossier de ressources « menu » dans le dossier « res » (Figure 13)

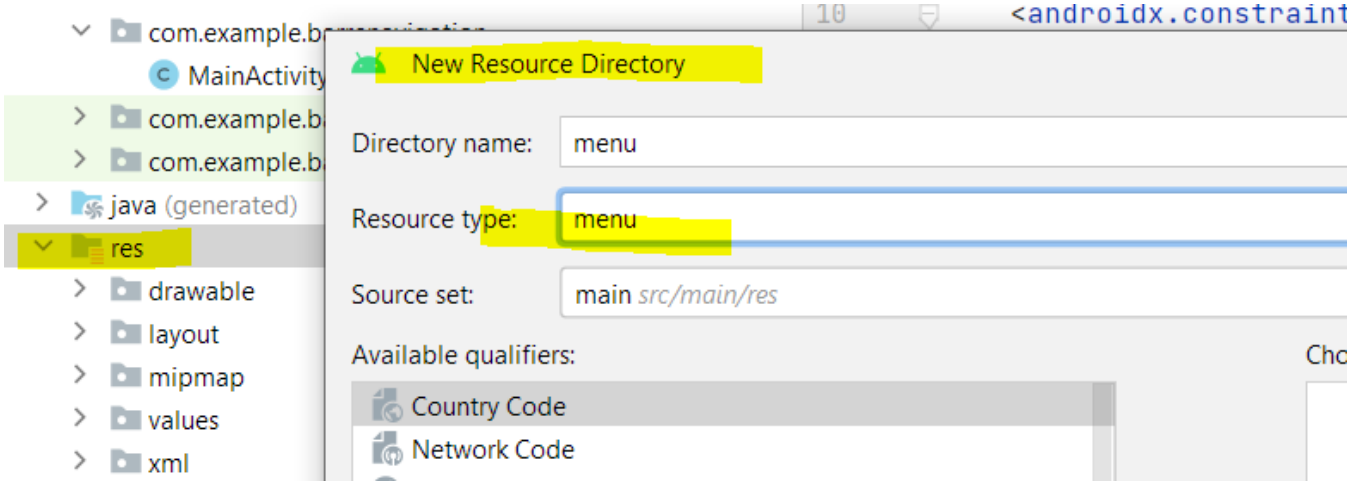

Figure 13 - Sous-dossier de ressources

→ Dans ce dossier, y ajouter un fichier de ressources, nommé, par exemple, menu.XML (Figure 14)

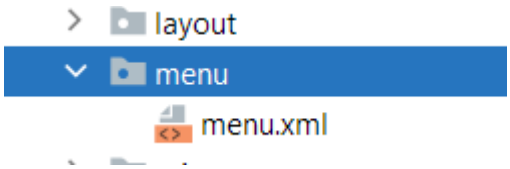

Figure 14 - Ressource "menu"

C'est dans ce fichier XML que l'on va créer l'aspect graphique du menu. Les lignes ci-dessous indiquent comme procéder :

Ajouter une icône : drawable → new → Vector Asset, puis rechercher l'icône dans le « clipart » , ici l'icône « home ». Celle-ci doit apparaitre dans les ressources (Figure 15)

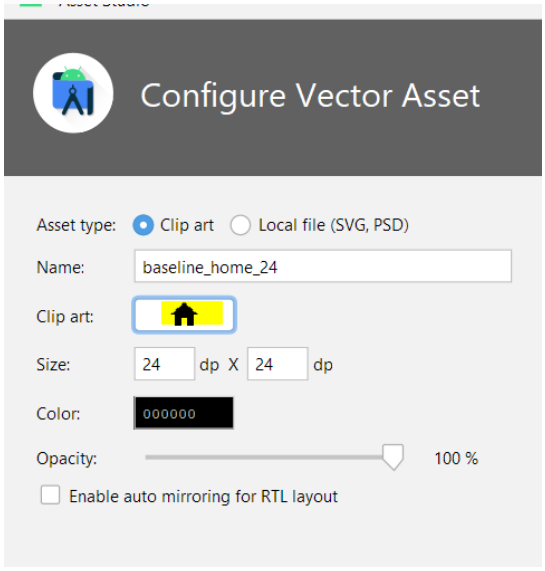

Figure 15 - Utiliser un "Vector Asset"

On code le fichier menu. XML

```
<item
    android:id="@+id/item1"
    android:title="Home"
    android:icon="@drawable/baseline_home_24">
</item>
```

Remarque : on peut créer facilement ses propres images au format SVG via un convertisseur de fichier JPG vers SVG ou bien un logiciel d'édition de fichiers « assets ».

Ajouter les lignes ci-dessous au début du fichier menu.XML :

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

xmlns:tools=http://schemas.android.com/tools
android:id="@+id/menuItem"
tools:showIn="NavigationView">
```

et ajouter maintenant la ligne ci-dessous dans le code XML du « NavigationView » situé dans le fichier activity_main.XML

```
app:menu="@menu/menu"
```

4.3 Coder l'activité principale

On ajoute maintenant le code permettant d'afficher le menu dans l'activité principale et de provoquer une action quand on sélectionne une des rubriques du menu.

L'action sera ici de simplement provoquer l'affichage d'un message de type « Toast ».

→ Ajouter le code ci-dessous à l'activité principale. Il est conseillé d'utiliser l'autocomplétion !

Il faudra créer deux « string » dans le fichier string.XML

```
<string name="close">Close</string>
<string name="open">Open</string>
```

```
public class MainActivity extends AppCompatActivity {

// declaration des objets necessaires
    private DrawerLayout tiroir;
    private NavigationView vueNav;
    private ActionBarDrawerToggle toggle;

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

// Association des objets avec les vues
        vueNav = (NavigationView) findViewById(R.id.navigationvue);
        tiroir = (DrawerLayout) findViewById(R.id.drawerLayout);
        toggle = new ActionBarDrawerToggle(this, tiroir, R.string.open,
R.string.close);


// Configuration du « listener » d'une action sur le menu
        tiroir.addDrawerListener(toggle);
        toggle.syncState();
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);


// Lancement du « listener » d'une action sur le menu

        vueNav.setNavigationItemSelectedListener(new
NavigationView.OnNavigationItemSelectedListener() {


// balayage des rubriques du menu
            @Override
            public boolean onNavigationItemSelected(MenuItem item) {
                switch (item.getItemId()) {
                    case R.id.item1:
                        Toast toast = Toast.makeText(getApplicationContext(),
"Home", Toast.LENGTH_SHORT);
                        toast.show();
                        break;
                }
                return true;
            }

        });

    }
}
```

→ Compiler et tester. Quand cela fonctionne, ajouter les 2 autres rubriques du menu… et faire fonctionner !

5 Atelier – mini projet

Faire la synthèse du travail réalisé. Ajouter 3 fragments au projet, pour intégrer les pages du paragraphe 2. La page « Home » affichera simplement un message. Tout est dans le TP !