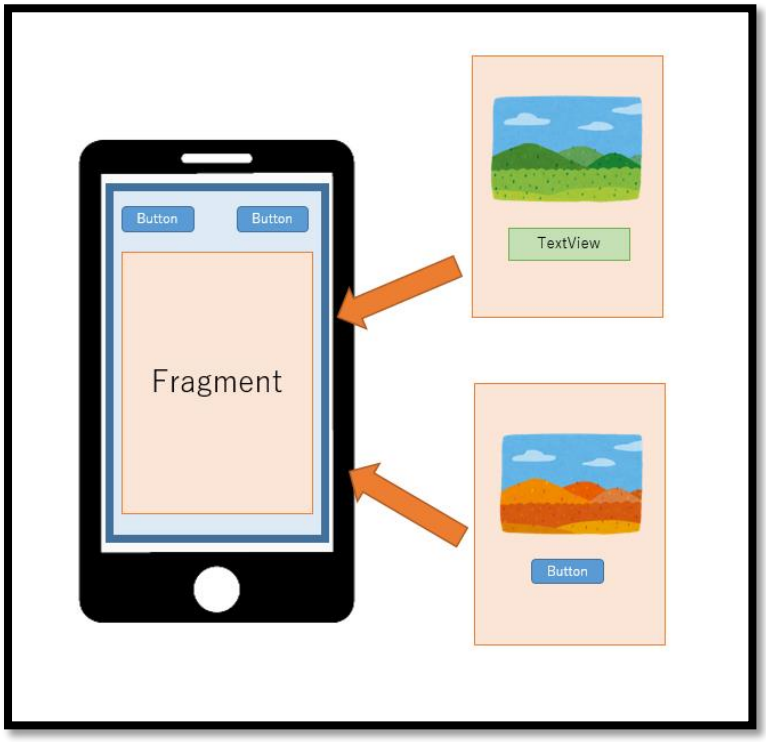
本日の内容

- Fragment
- Fragment のライフサイクル

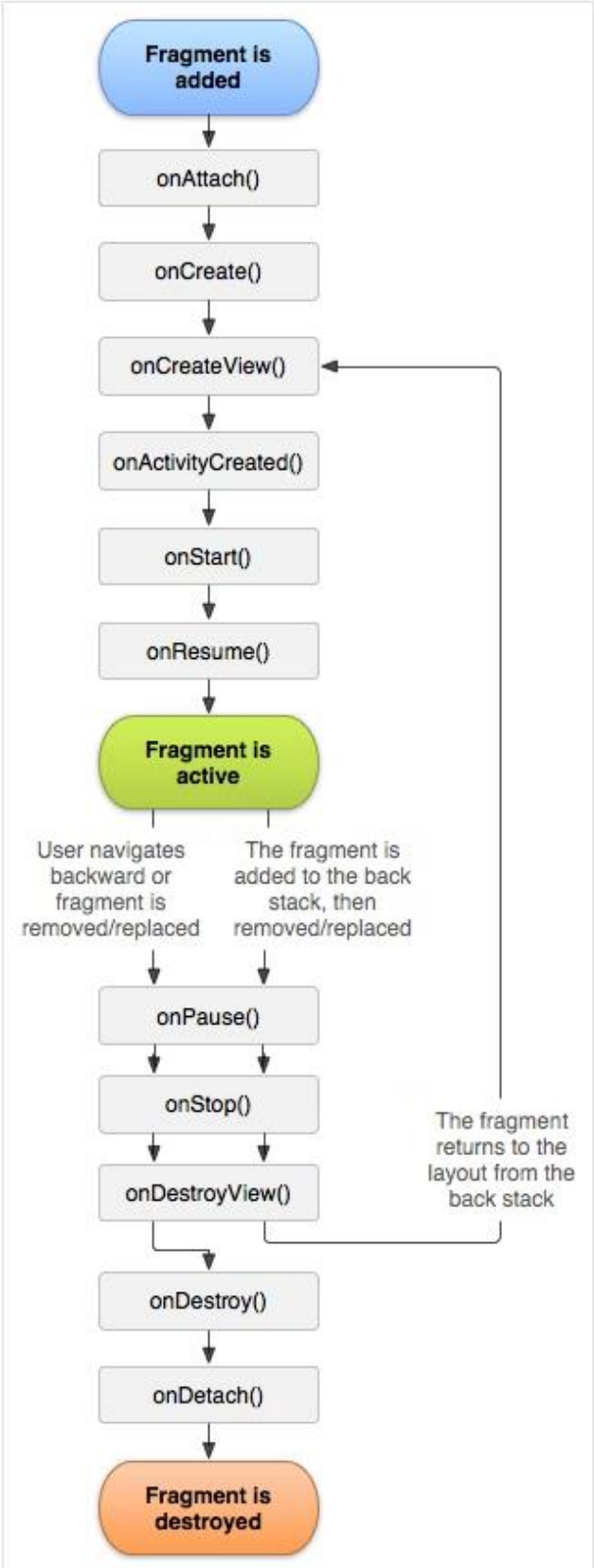
公式ドキュメント(Fragment)
https://developer.android.com/guide/components/fragments?hl=ja

◆Fragment とは
簡単に言うと「Activity の部品」です
1 画面を部品として扱える為、1Activity 内で複数の画面に切り替えることが可能となります。



◆Fragment の仕組み
Fragment は Activity と同じように、「レイアウトの xml ファイル」と「制御の kt ファイル」が必要です。

◆Fragment のライフサイクル



◆onAttach(Context context)

Fragment が Activity と関連付けられたときに呼ばれる

◆onCreate()

Fragment が作成されたときに呼び出されます

◆onCreateView()

Fragment に関連付けられた View を作成する際に呼ばれる

◆onActivityCreated()

Activity の onCreate()の完了後に呼ばれる

◆onStart()

Fragment がユーザに見えるようになった時呼ばれる

◆onResume()

Fragment がユーザ操作の受付開始時に呼ばれる

◆onPause()

Fragment がフォアグラウンドで亡くなった場合呼ばれる

◆onStop()

Fragment がユーザに表示されなくなったら呼ばれる

◆onDestroyView()

Activity 破棄前に呼ばれる

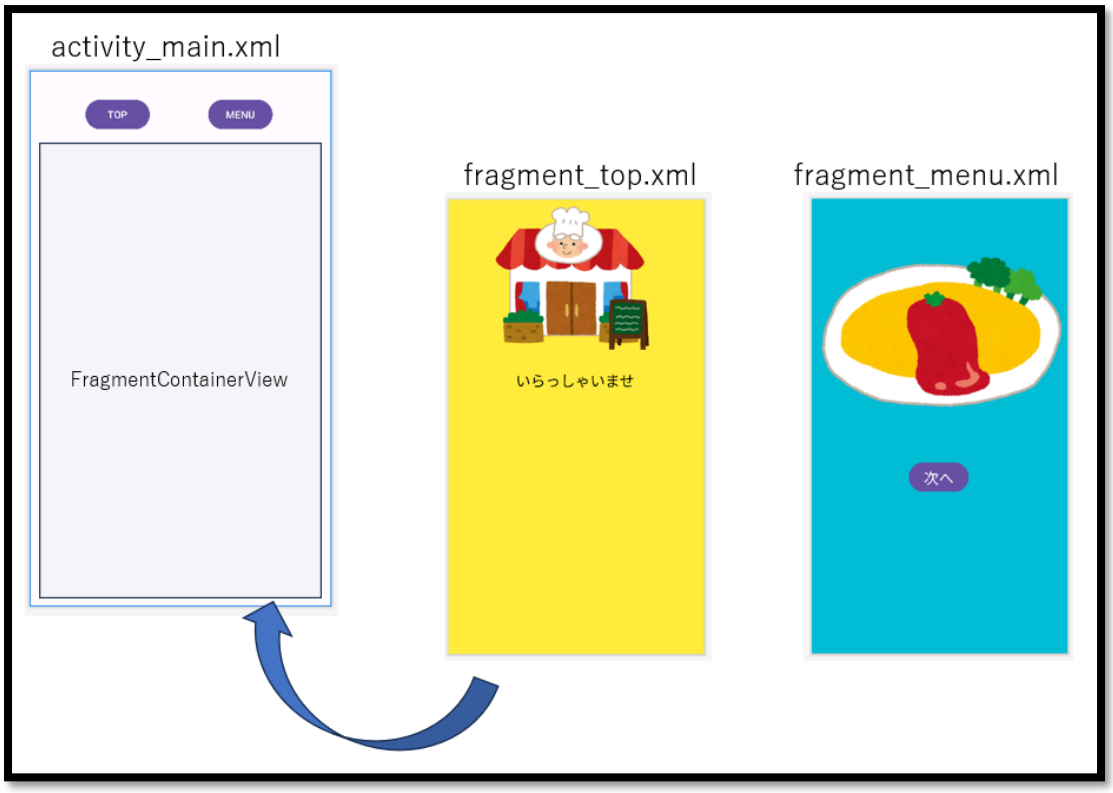
Fragment に関連付けられた View の削除時に呼ばれる

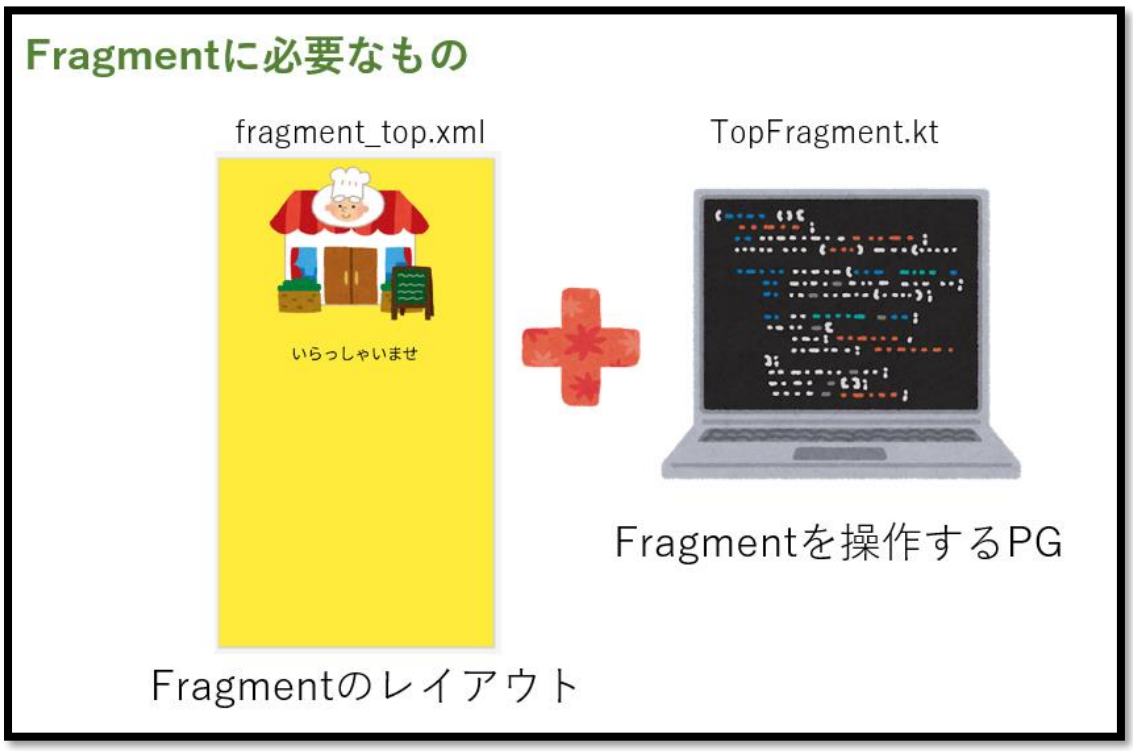
◆onDestroy()

Fragment 破棄前に呼ばれる

◆onDetach()

Fragment がユーザに表示されなくなったら呼ばれる


activity_main.xml
TOP
MENU
FragmentContainerView
fragment_top.xml
いらっしゃいませ
fragment_menu.xml
次へ

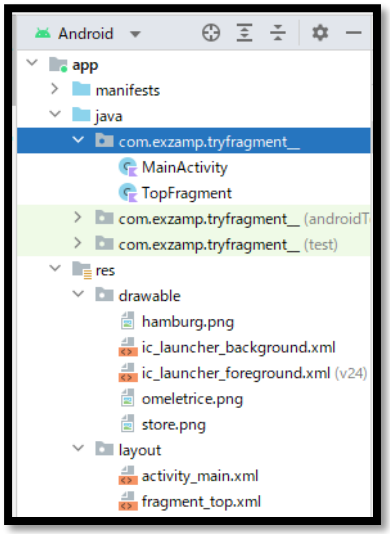

Fragmentに必要なもの
fragment_top.xml
いらっしゃいませ
TopFragment.kt
Fragmentを操作するPG
Fragmentのレイアウト

■Fragment の動作確認
では実際に Fragment を活用したアプリを作成し
作成方法と動作確認を行っていきます。

■プロジェクト作成
プロジェクト名：FragmentMenu
==========

◆① ：Fragment1(TopFragment)の作成

①-1： MainActivity と同じパッケージを右クリック>new>Fragment>Fragment(Blank)
FragmentName:TopFragment



①-2： デフォルトで記載されている、今回は不要な関数や処理を下記画像となるように削除する

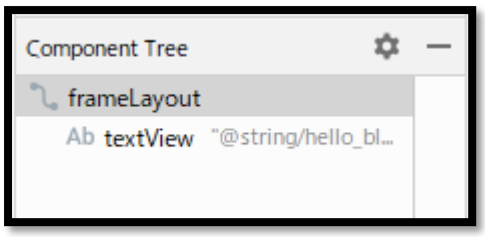
```
class TopFragment : Fragment() {
    // Fragmentに関連付けられたViewを作成する際に呼ばれる
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_top, container, attachToRoot: false)
    }
    // Singletonで作成するデータ
    companion object {
        @JvmStatic
        fun newInstance() = TopFragment()
    }
}
```

●LayoutInflater とは
指定した xml レイアウト(View)リソースを
利用出来る仕組み

●attachToRoot
レイアウトから溢れた場合、
自動で子の要素として追加する(true) またはしない

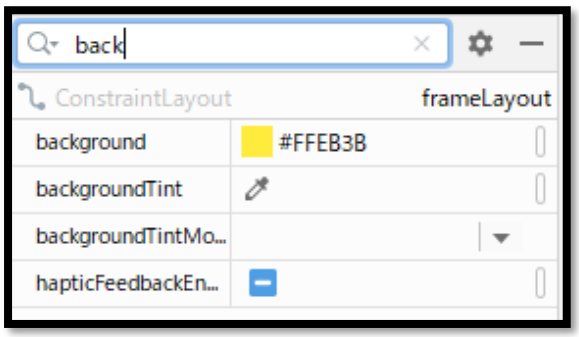
◆② ：fragment_top.xml の編集

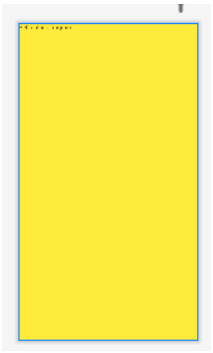
②-1： ルートタグを ConstraintLayout に変更
- compornentTree の[FrameLayout]を右クリック>Convert FrameLayout to ConstraintLayout
- OK



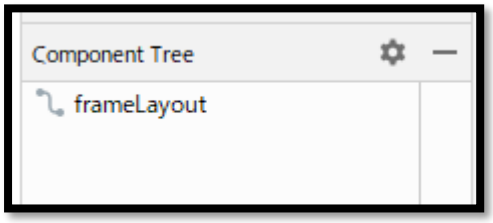
②-2:　ルートタグの背景色を黄色に変更

- background の値を黄色っぽい色で選択(自由)



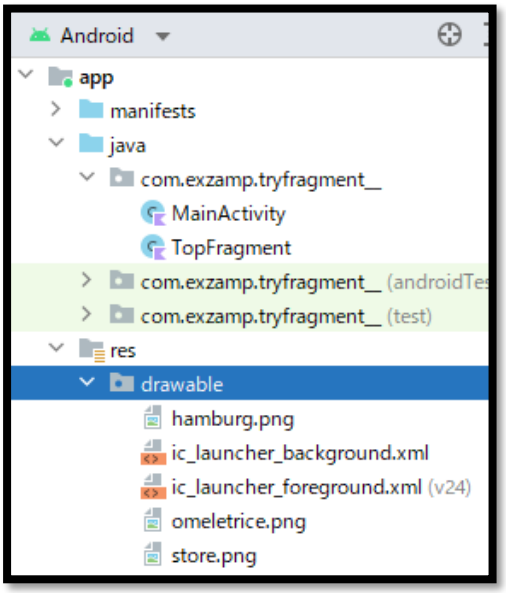


②-3 :　デフォルトで配置されている TextView を削除

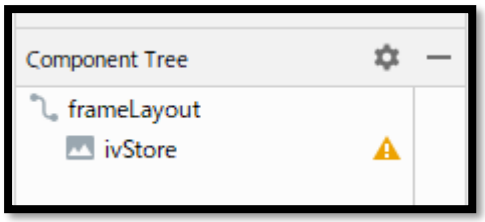


②-4 :　ImageView の配置

- res>drawble フォルダに各画像をコピー&ペースト(-v24 は選ばない)



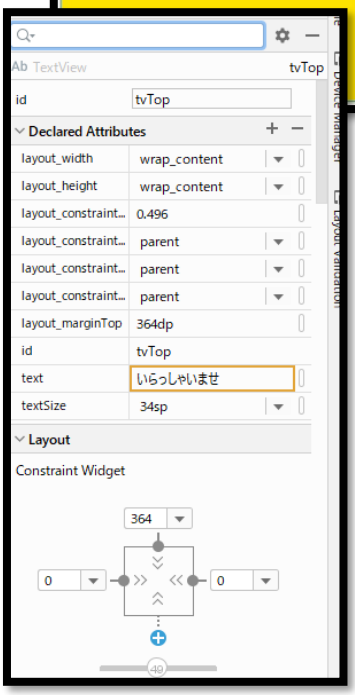
- ImageView を fragment_top.xml 上に配置:画像は store.png（位置の細かい指定は無し）
- id を ivStore に変更

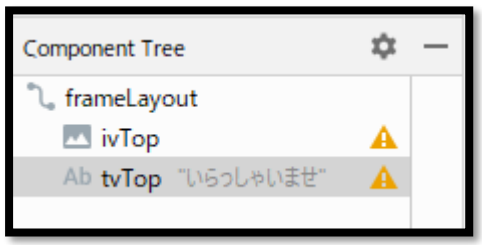


- 位置制約を設定

②-5：　TextView の配置

- TextView を fragment_top.xmlxml 上に配置
  （細かい指定は無し）
- Id を tvTop に変更
- Text を”いらっしゃいませ！”に変更
- textSize を 34sp に変更
- 位置制約を設定





◆③ ：Fragmet2(MenuFragment)の作成

③-1：　MainActivity と同じパッケージを右クリック>new>Fragment>Fragment(Blank)
FragmentName:MenuFragment

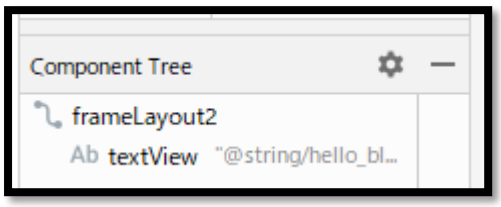
③-2：　MenuFragment.kt 内にデフォルトで記載されている、(今回は)不要な関数や処理を削除

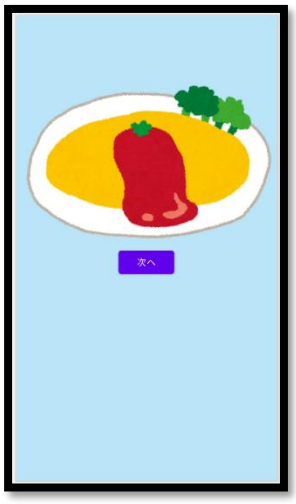
```
class MenuFragment : Fragment() {
    // Fragmentに関連付けられたViewを作成する際に呼ばれる
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_menu, container, attachToRoot: false)
    }
    // Singletonで作成するデータ
    companion object {
        @JvmStatic
        fun newInstance() = MenuFragment()
    }
}
```

◆④ : fragment_menu.xml の編集

④-1 : ルートタグを ConstraintLayout に変更

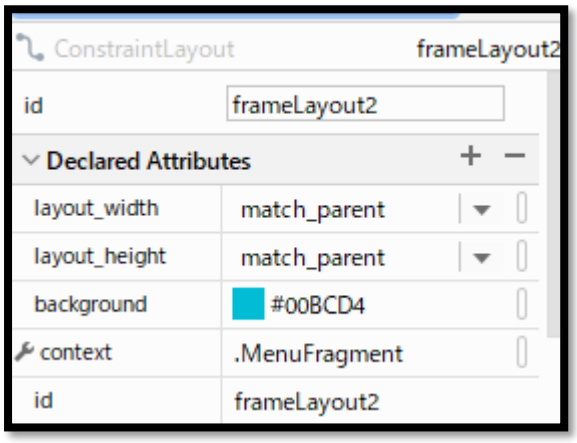
- compornentTree の[FrameLayout]を右クリック>Convert FrameLayout to ConstraintLayout
- OK



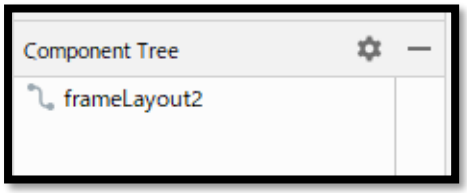


④-2 : ルートタグの背景色を水色に変更

- background の値を黄色っぽい色で選択(自由)

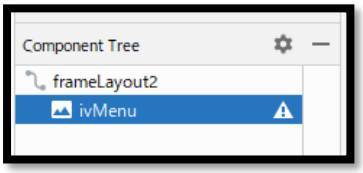


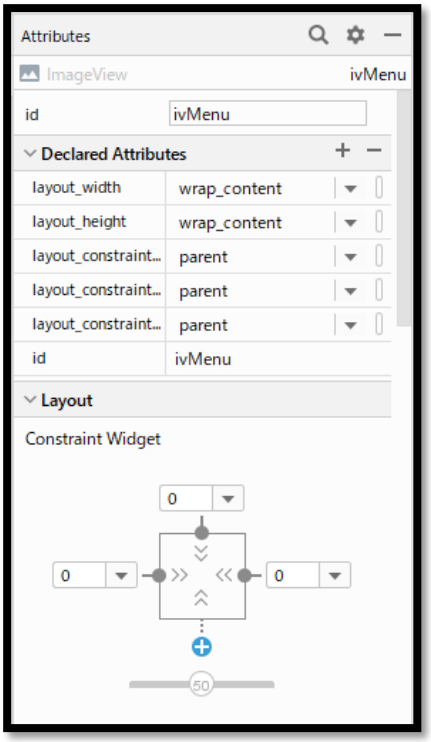
④-3 : デフォルトで配置されている TextView を削除

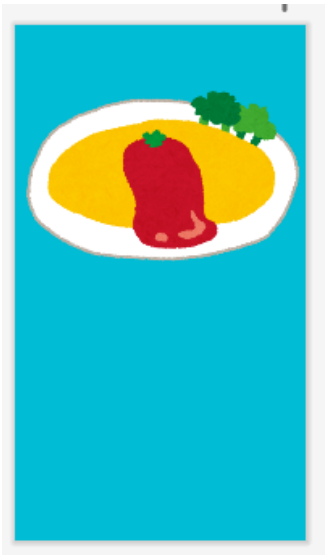


④-4：　ImageView の配置

- ImageView を xml 上に配置:画像は omeletrice.png
- id を ivMenu に変更
- 位置制約を設定

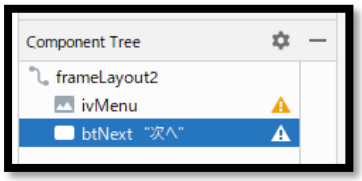


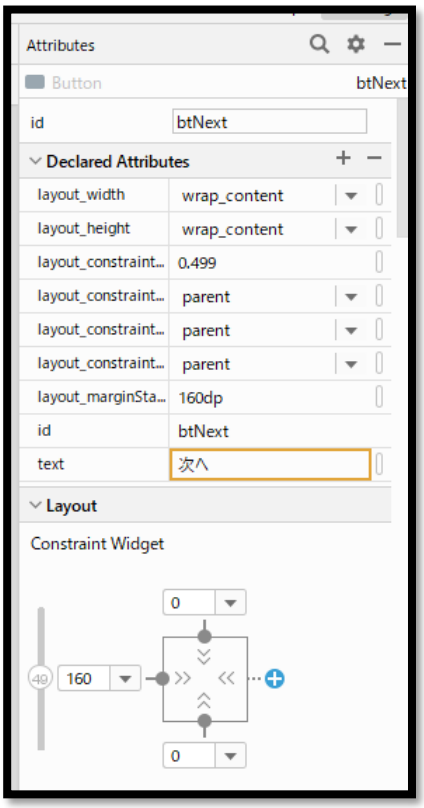


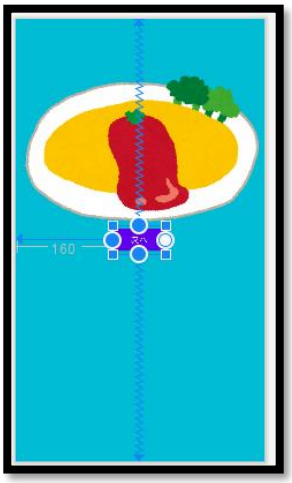


④-5：　Button の配置

- Button を xml 上に配置
- id を btNext に変更
- text を「次へ」に変更







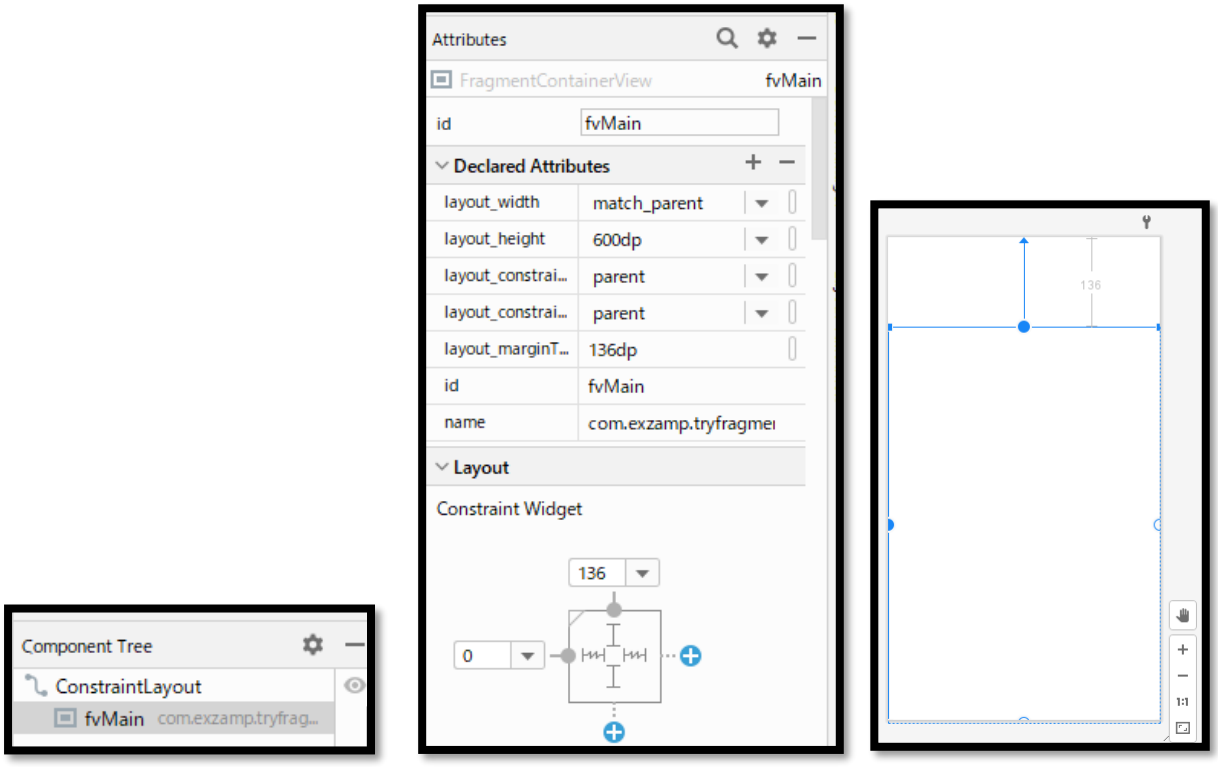
作成した Fragment を mainActivity で表示出来るように配置していきます。

◆⑤ : activity_main.xml

⑤-0:　デフォルトの TextView を削除

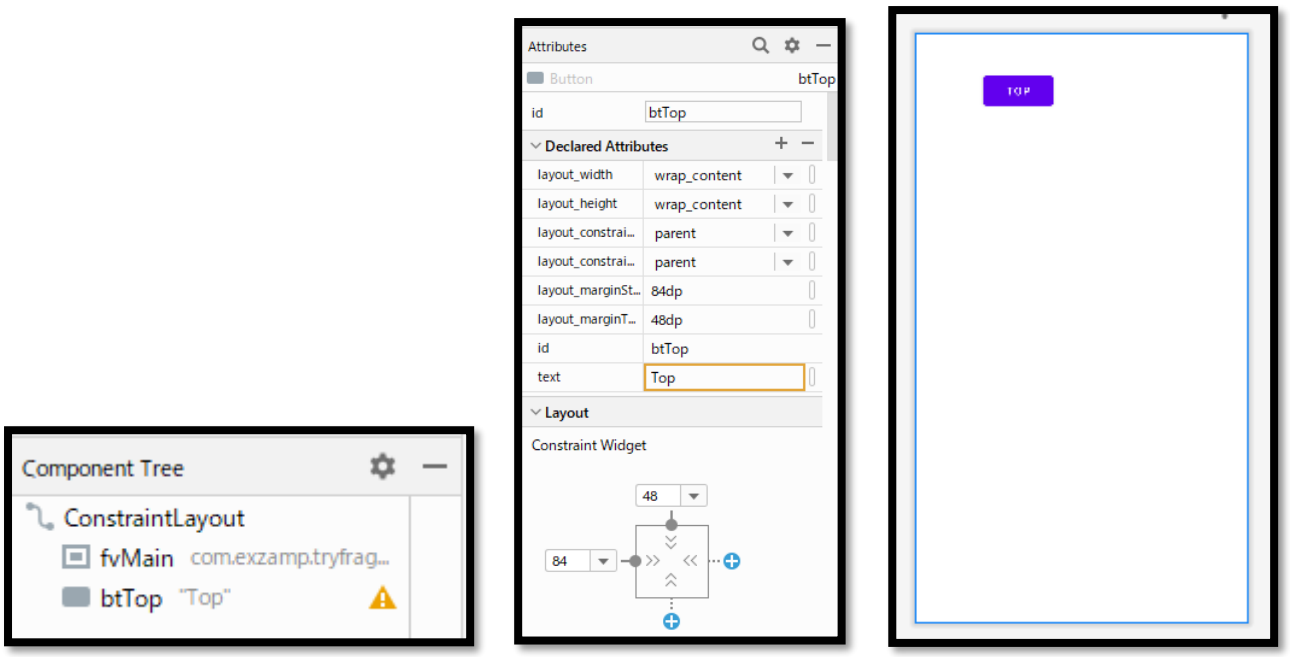
⑤-1 :　FragmentContainerView の配置

- xml 上に FragmentContainerView を配置
    - 最初の fragent に TopFragment を指定
- layout_width を match_parent に変更
- layout_height を 600dp に変更
- 位置制約を設定
- 下詰めの位置に移動
- id を fvMain に変更



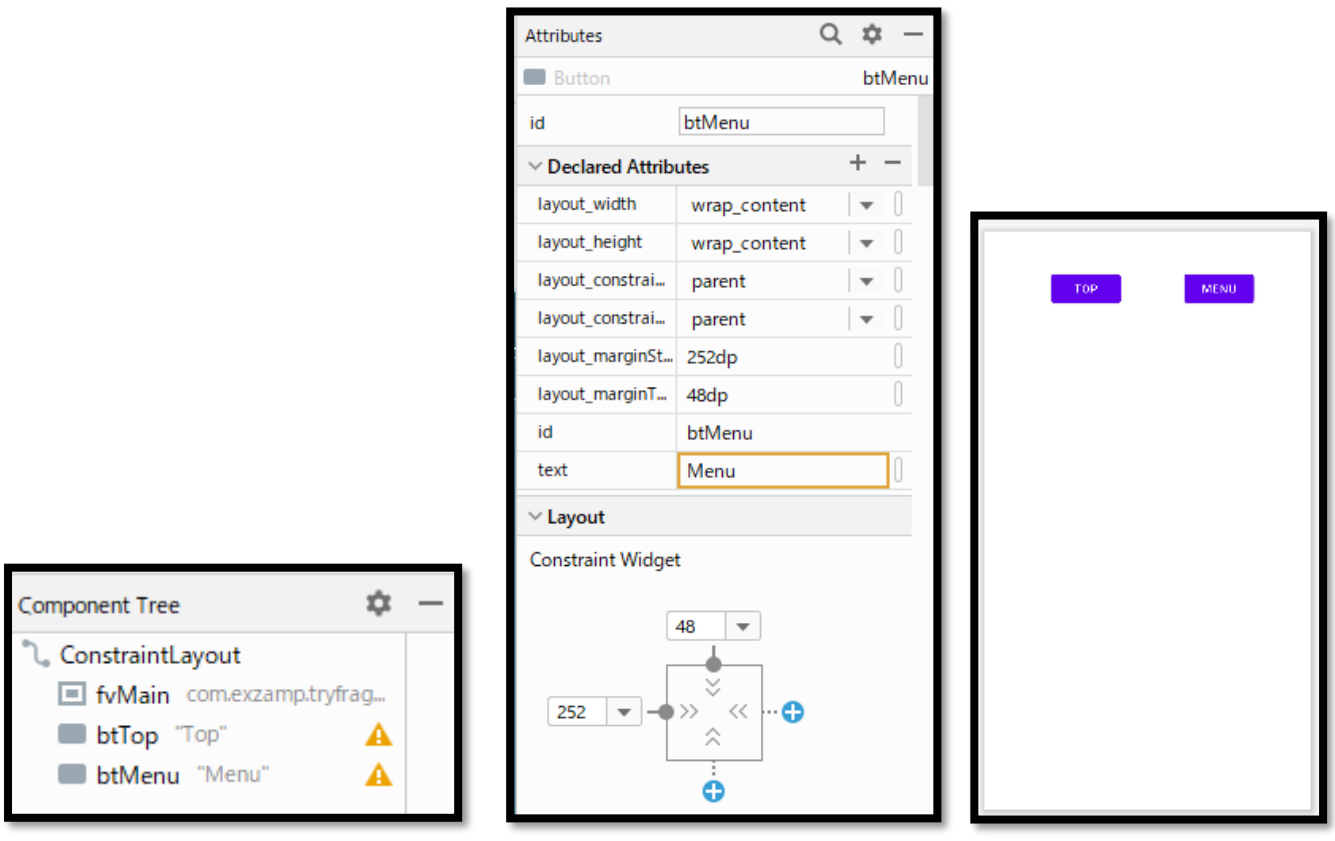
⑤-2 :　1 つ目の Button の配置

- xml 上に Button を配置
- id を btTop に変更
- text を「Top」に変更
- 位置制約を設定



⑤-3： 2 つ目の Button の配置

- xml 上に Button を配置
- id を btMenu に変更
- text を「Menu」に変更
- 位置制約を設定



レイアウトの準備は出来たので、
プログラム上で Fragment を切り替える処理を追加しましょう

◆⑥ : ActivityMain.kt

⑥-1 : View の取得

- btTop と xml の btTop を関連付ける
- mtMenu と xml の btMenu を関連付ける

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Viewの取得
        val btTop: Button = findViewById(R.id.btTop)
        val btMenu: Button = findViewById(R.id.btMenu)
    }
}
```

⑥-2 : btTop のクリック時の処理を追加

- btTop の setOnClickListener を定義
- supportFragmentManager を活用し Fragment の内容を TopFragment に置き換える

```
supportFragmentManager.beginTransaction()
                .replace(置き換え元フラグメントの ID , 置き換えるフラグメントのインスタンス)
                .commit()
```

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Viewの取得
        val btTop: Button = findViewById(R.id.btTop)
        val btMenu: Button = findViewById(R.id.btMenu)

        // btTopのクリックイベント
        btTop.setOnClickListener { it: View!
            // TopFragmentに置き換える
            supportFragmentManager.beginTransaction()
                .replace(R.id.fvMain,TopFragment.newInstance())
                .commit()
        }
    }
}
```

⑥-3：　btMenu のクリック時の処理を追加

- btMenu の setOnClickListener を定義
- supportFragmentManager を活用し Fragment の内容を TopFragment に置き換える

```
supportFragmentManager.beginTransaction()
                .replace(置き換え元フラグメントの ID,置き換えるフラグメントのインスタンス)
                .commit()
```

```
        // btTopのクリックイベント
        btTop.setOnClickListener { it: View!
            // TopFragmentに置き換える
            supportFragmentManager.beginTransaction()
                .replace(R.id.fvMain,TopFragment.newInstance())
                .commit()
        }

        // btMenuのクリックイベント
        btMenu.setOnClickListener { it: View!
            // MenuFragmentに置き換える
            supportFragmentManager.beginTransaction()
                .replace(R.id.fvMain, MenuFragment.newInstance())
                .commit()
        }
    }
}
```

実行し、ボタン押下で fragment が切り替わっていることを確認

◆⑦Fragment ライフサイクルの確認

⑦-1 : TopFragment.kt に各コールバックの処理を実装し確認

- 上部の code タブ>Override methods を選択し以下のコールバックを実装
- Log.d()の処理を追加しログを確認

◆onAttach(Context)

Fragment が Activity と関連付けられたときに呼ばれる

◆onCreate()

Fragment が作成されたときに呼び出されます

◆onCreateView()

Fragment に関連付けられた View を作詞絵する際に呼ばれる

◆onActivityCreated()

Activity の onCreate()の完了後に呼ばれる

◆onStart()

Fragment がユーザに見えるようになった時呼ばれる

◆onResume()

Fragment がユーザ操作の受付開始時に呼ばれる

◆onPause()

Fragment がフォアグラウンドで亡くなった場合呼ばれる

◆onStop()

Fragment がユーザに表示されなくなったら呼ばれる

◆onDestroyView()

Activity 破棄前に呼ばれる

Fragment に関連付けられた View の削除時に呼ばれる

◆onDestroy()

Fragment 破棄前に呼ばれる

◆onDetach()

Fragment がユーザに表示されなくなったら呼ばれる