`

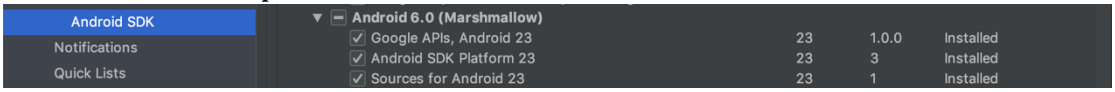
Tutoriel *Avancé*
User Interface ANDROID

Objectifs visés par ce tutoriel

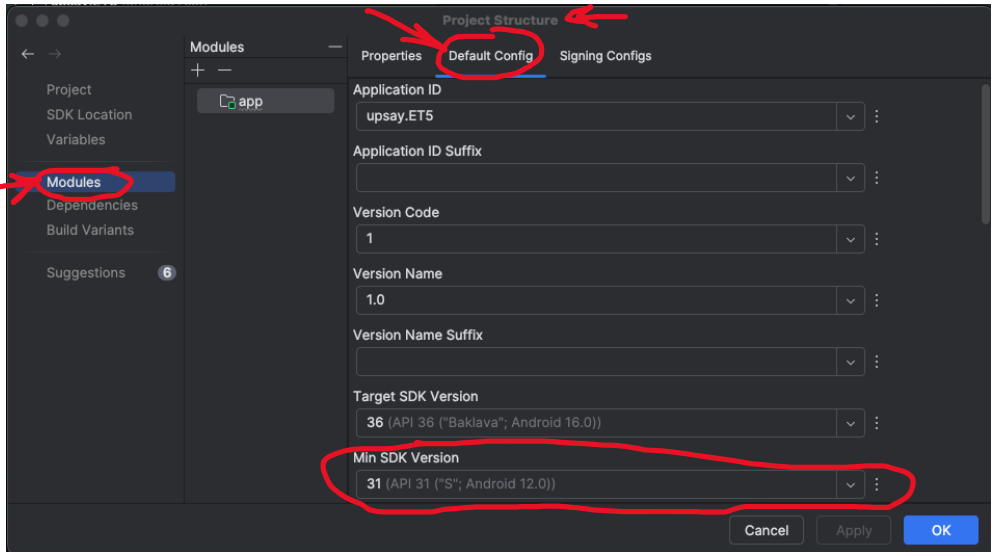
- Utilisation du GitHub intégré.
- RAD avancée
- Utilisation élémentaire de plusieurs classes Activity dans une UI et échange de données entre elles.
- Utilisation de la classe Fragment pour limiter une UI à une seule Activity et simplifier les échanges de données dans l'application.

Versions utilisées pour ce tutoriel

- Version en date de l'environnement JAVA : **il est important de s'assurer avant l'installation de Androïd Studio que votre version de Java est à jour**, en cas de doute désinstallez la version actuelle et installez la dernière en date.
  le **Software Development Kit** est la bibliothèque Java utilisée pour votre développement, le choix de sa version est critique. Une version trop ancienne limiterait les fonctionnalités, une version trop récente le nombre de terminaux sur lesquels votre application s'exécuterait.
  Important : pour du code utilisant le BLE il y a eu un changement majeur dans la gestion des droits à partir de la version 12 (API 31). En fonction de votre matériel il faut choisir la bonne version.
- Le **SDK Manager** vous permet de savoir ce que vous avez comme versions installées pour votre Androïd Studio, et d'en installer d'autres



- Pour le projet ouvert c'est la fenêtre file/**Project Structure** qui vous permet de savoir comment est configuré votre projet.



Bibliographie d'accompagnement conseillée

Les références de toutes les API des SDK

- https://developer.android.com/docs
- https://developer.android.com/reference/packages
- https://developer.android.com/reference/classes

Tout pour se former du débutant à l'expert

- https://developer.android.com/courses/fundamentals-training/toc-v2
- https://developer.android.com/courses/fundamentals-training/overview-v2
- https://drive.google.com/drive/folders/1MRqvBGEDtNtpDyKd8sulMJreFCz1JxgC
- https://github.com/google-developer-training/android-advanced
- https://firebase.google.com/docs/android/setup?authuser=0
- https://www.udacity.com/course/android-basics-user-interface--ud834
- https://classroom.udacity.com/me

A. Utilisation de GitHub avec Androïd Studio

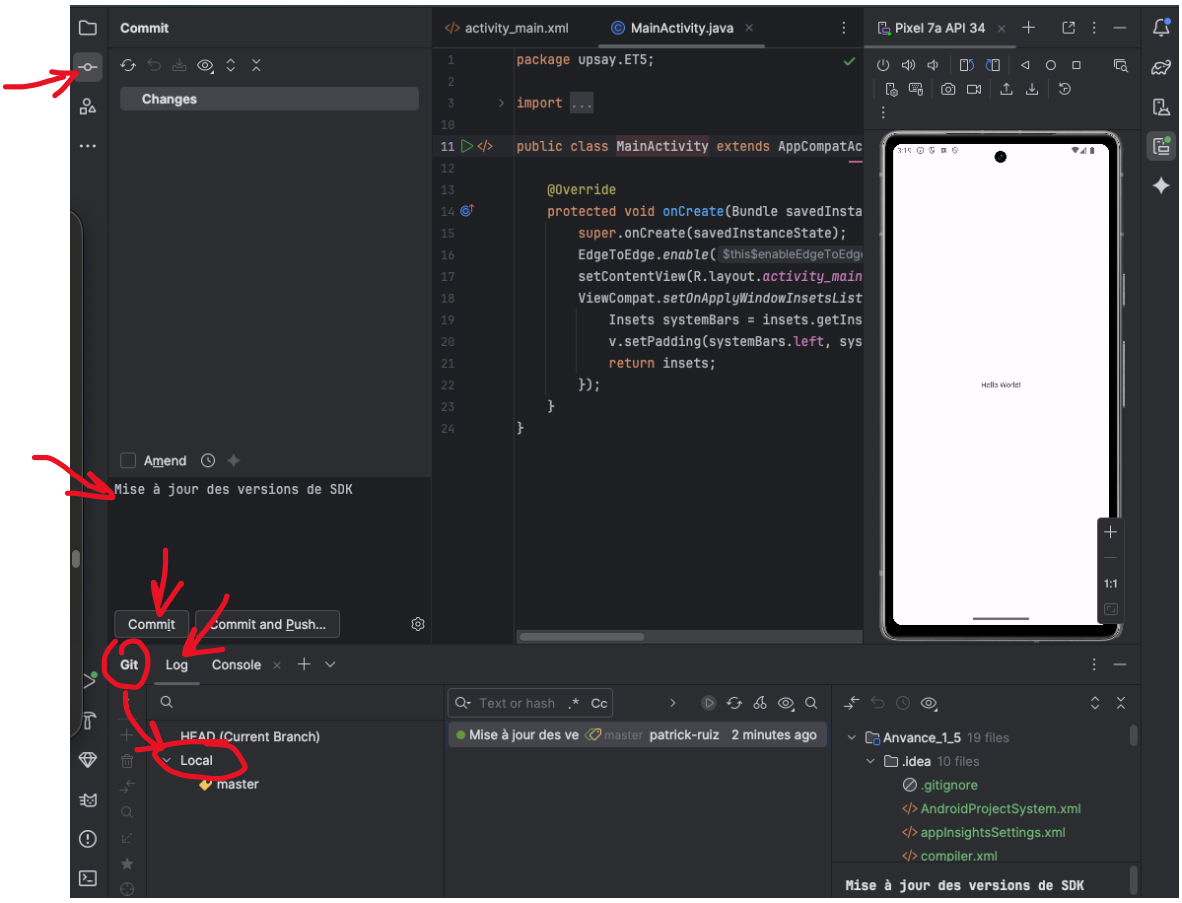
Notions générales sur Git :

- **Git** (Global Information Tracker, Linus Torvalds) : système de gestion de versions décentralisé (version control system).
- **GitHub** : plateforme **collaborative** hébergeant des dépôts Git, avec outils de gestion de projets.
- **Collaborators** : contributeurs ayant des droits sur le dépôt.
- **Repository (français: dépôt)** : enregistrement **distant** (**sur le cloud ou un serveur privé dans le cas de GIT)** contenant le code, la documentation, et l'historique des versions.
- **Clone** : copie **locale** complète d'un dépôt distant.
- **Commit sans "push"** : enregistrement **local** d'une modification dans l'historique du **clone**.
  Message de commit : description courte et claire du changement apporté.
- **Revert / Reset** : annulation ou retour à un état antérieur du code.
- **Push** : envoi des commits locaux vers le dépôt distant.
- **Pull** : synchronisation locale en récupérant les mises à jour distantes.
- **Master** : branche principale du projet.
- **Branch (français: branche)** : ligne de développement parallèle pour isoler ou tester des modifications (isole un développement temporaire).
- **Merge** : intégration des changements d'une branche **dans une autre branche**.
  Merge conflict : conflit entre deux versions d'un même fichier lors d'une fusion.
- **Fork**: copie complète d'un dépôt GitHub dans un nouveau dépôt distinct, avec son propre espace d'historique (copie open-source sans être collaborateur).
- **Pull Request (PR)** : proposition de **fusion** d'un fork vers vers le dépôt d'orgine (souvent pour contribution open-source).

Quelques outils avancés sur Git:

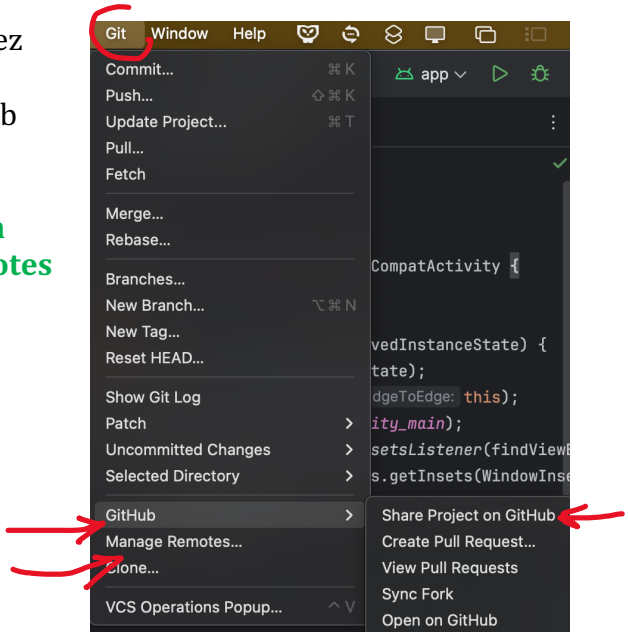
- **Historique Git** : suivi de tous les commits (qui, quoi, quand).
- **Tag** : marque un point particulier dans l'historique (souvent pour une version stable).
- **Issues** : signalement de bugs, demandes de fonctionnalités ou tâches à suivre
- **Milestones** : regroupement d'issues en objectifs intermédiaires.
- **Projects (tableaux Kanban)** : suivi visuel de l'avancement d'un projet.
- **Wiki** : documentation du projet directement intégrée dans GitHub.

Vue dans Androïd Studio après avoir effectué un commit local (le dépôt local n'a pas encore été rattaché au repository GitHub):

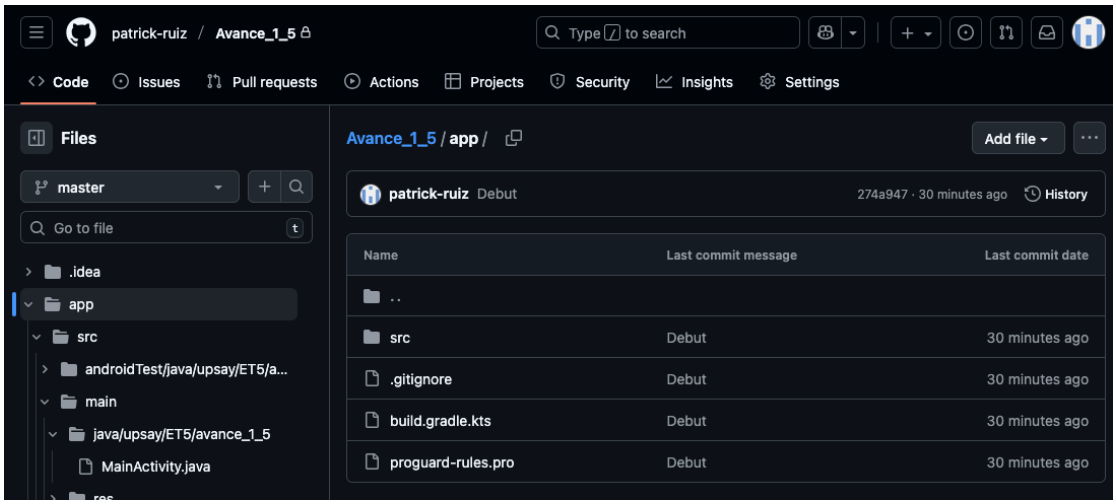


Pour le lier à votre GitHub vous pouvez procéder ainsi :

1- Créer un nouveau dépôt GitHub
2- Copier son URL
3- Utiliser la commande **Git/GitHub/Share Project on GitHub** OU **Git/Manage Remotes**
4- Coller l'URL du dépôt



Résultat dans GitHub (distant) :

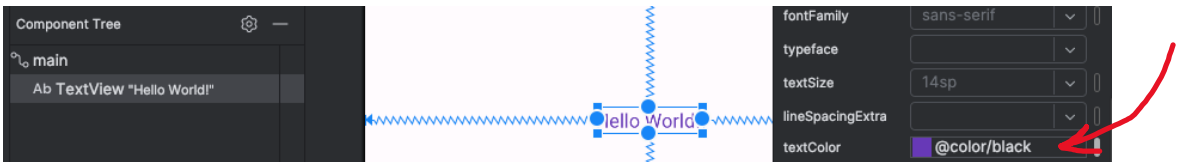


B. Quelques outils avancés de RAD

1. ***Si vous travaillez à la suite du tutoriel bases passez directement au 2.***
   Dans le menu ***Quick Start***, ou ***File/new…*** sélectionner ***Start a new Android Studio Project***.
   - Choisir ***Empty Views Activity*** ATTENTION pas Empty Activity !
   - Donner le nom de votre choix à l'application.
   - Modifier le nom du package par défaut en par exemple ***UPsay.ET5***.
   - Notez bien l'endroit où sera enregistré le projet dans la zone ***Save Location***.
   - **Assurez-vous que le langage est bien Java** !
   - Regarder la version d'Androïd sur votre matériel et choisir la version correspondante (notez le pourcentage de systèmes sur lesquels s'exécutera votre application). Ne rien sélectionner de plus, puis cliquer sur ***finish***.

2. Nous allons maintenant **naviguer dans les ressources, c'est-à-dire les éléments « statiques » (xml, images…)** de cette application Androïd en ouvrant la commande ***View/Tool Windows/Ressource Manager***
   Cliquer par curiosité sur **les trois points à droite** de la barre de sélection qui permet d'ouvrir les onglets qui n'apparaissent pas pour l'instant.

   - Ouvrir l'onglet ***Layout***, et double cliquer sur une des ressources. Que se passe-t-il ?
     A partir de la fenêtre de RAD qui s'est ouverte modifier la propriété *textColor* du *TextView* en saisissant ***@color/black*** de votre application comme ci-dessous :



- Regarder dans l'onglet ***Color***, et cliquer sur une des couleurs. Dans la fenêtre qui affiche le code XML décrivant la couleur. Cliquer sur la couleur apparaissant dans la marge. Modifier la couleur : transformez le noir en violet par exemple et lancer l'application pour voir la conséquence. Ceci sert surtout à créer une palette de couleurs propre à votre application.

- Regarder dans l'onglet ***Drawable***, qu'est ce qui apparaît ?

- Regarder dans l'onglet ***Mip Map***, puis cliquer sur une des icones. Pourquoi l'image apparait-elle en plusieurs version ? En cliquant sur les versions il apparait des dimensions en haut à droite. Quelle est cette unité ? Quel est l'intérêt de cette fonction « automatique »?

Provide alternative bitmaps

To provide good graphical qualities on devices with different pixel densities, you should provide multiple versions of each bitmap in your app—one for each density bucket, at a corresponding resolution. Otherwise, Android must scale your bitmap so it occupies the same visible space on each screen, resulting in scaling artifacts such as blurring.

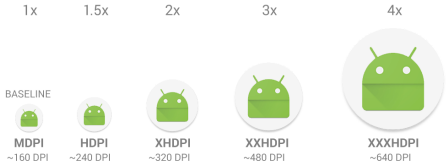


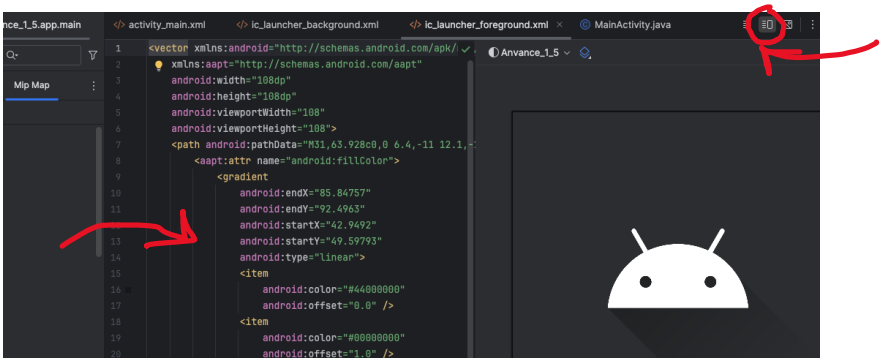
Figure 2. Relative sizes for bitmaps at different density sizes

There are several density buckets available for use in your apps. Table 1 describes the different configuration qualifiers available and what screen types they apply to.

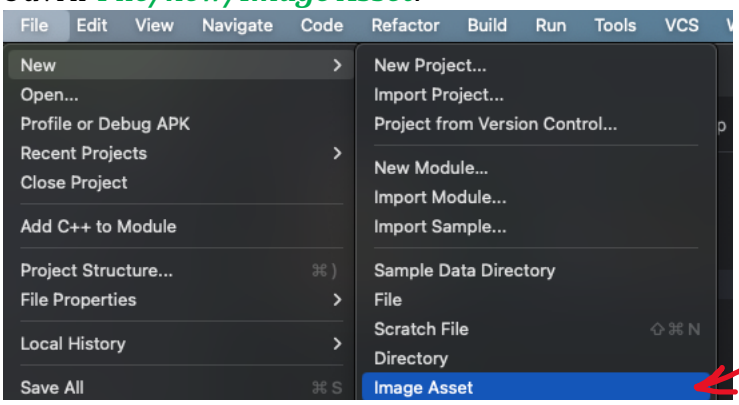
Table 1. Configuration qualifiers for different pixel densities.

Density qualifier	Description
`ldpi`	Resources for low-density (*ldpi*) screens (~120dpi).
`mdpi`	Resources for medium-density (*mdpi*) screens (~160dpi). (This is the baseline density.)

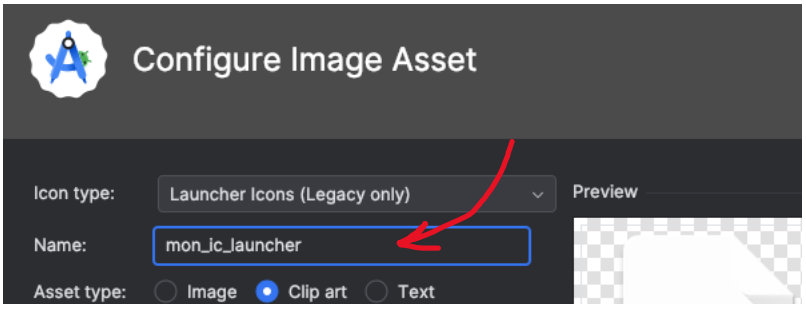
- Pour remplacer l'icône de départ, toujours à partir de l'onglet ***Mip Map***, cliquer dans la **zone du code XML**.



Ouvrir ***File/new/Image Asset***.

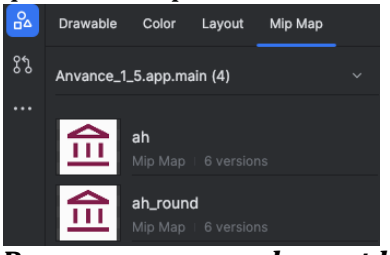


En cliquant dessus s'ouvre la fenêtre de design. Choisir un nom différent, comme `ah`



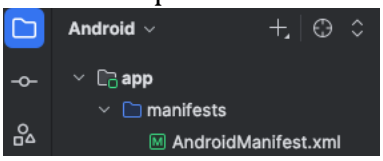
Cliquer sur **Clip art** et choisir une autre icone, changer la couleur, modifier le background etc....

En allant jusqu'au bout et en remontant dans le menu Mip Map vous verrez que vous disposez d'un nouveau jeu d'images

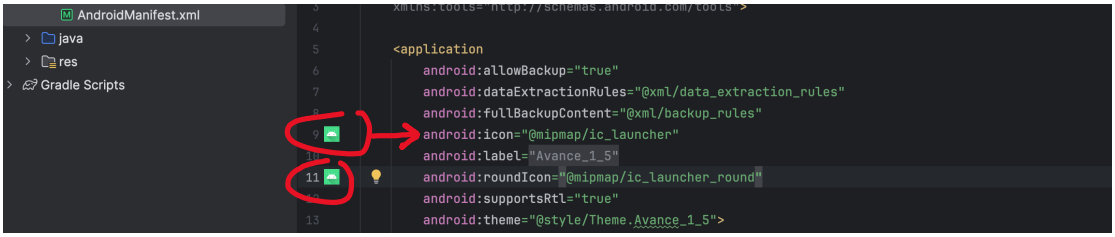


Remarque : normalement l'automatisme de GitHub vous propose aussi de rajouter le code xml correspondant au dépôt.

Ouvrir à partir du menu ***Android*** le fichier AndroidManitest.xml.

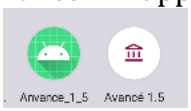


Remplacer la ligne qui déclare l'icône de votre application en cliquant sur l'icône !



- Ouvrir l'onglet ***String***, et modifiez le texte indiqué en gras de la ligne reproduite ci-dessous avec par exemple « mon app »:
  <string name="title_activity_main">**MainActivity**</string>
  Remarque: Androïd prévoir aussi du multilingue si vous regardez le message qui s'affiche au moment de la modification.

Lancer l'application et la fermer. Regarder maintenant l'icône.

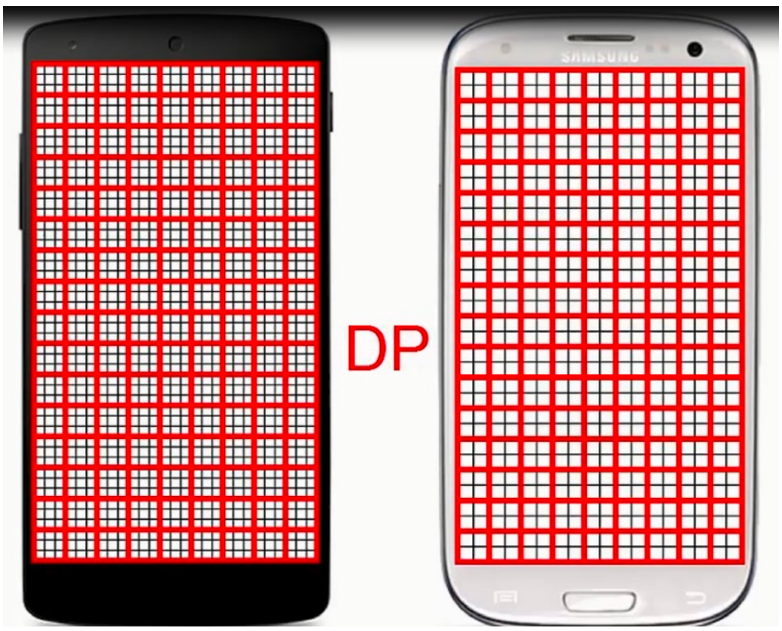


C. Organiser précisément les composants graphiques grâce à la classe *ConstraintLayout*

Le **ConstraintLayout** *est le gestionnaire de disposition le plus flexible, mais il permet aussi une adaptation automatique à différentes tailles d'écran. C'est celui utilisé le plus largement par les applications professionnelles.*

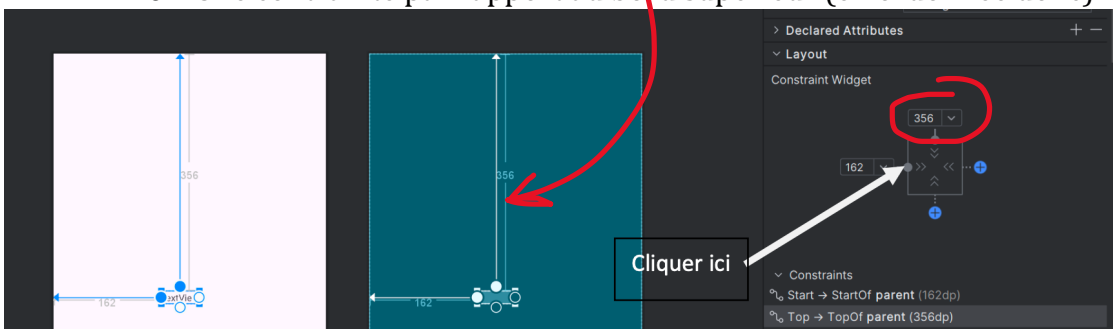
Choisir l'application possédant seulement un bouton et une zone de texte.

- Convertir le gestionnaire de disposition en ***ConstraintLayout*** (c'est celui par défaut à la création de l'application)
- L'écran est repéré en XY, avec le point de coordonnées (0,0) en haut à gauche. L'axe des X est l'axe horizontal orienté vers la droite, l'axe des Y est l'axe vertical, orienté vers le bas. Pour tenir compte de la diversité des résolutions des écrans l'unité n'est le pixel mais le **DP**, ceci explique des incohérences entre la taille en pixel des images dans les ressources et leur représentation à l'écran. La mesure en DP correspondent à « la taille en pixels qu'aurait l'interface sur un écran de résolution 160 pixels par pouce ». Ci-dessous les DP en rouge par rapport aux pixels réels.

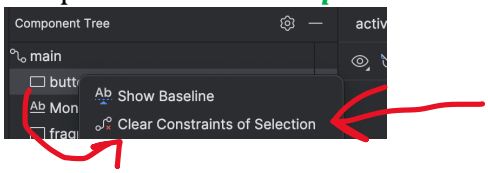


- Il existe pour les texte une unité comparable, le SP (Scale-independent Pixel). Comme le dp, il est indépendant de la densité, mais **il prend en compte en plus les préférences d'accessibilité de l'utilisateur** (taille de police personnalisée dans les paramètres Android). Si par exemple l'utilisateur choisit "grande police", les valeurs en sp s'adaptent automatiquement.

- Chaque objet peut présenter **4 contraintes** avec un ***ConstraintLayout*** mais il est recommandé de commencer par en définir, par exemple :
    - Une contrainte par rapport au bord gauche (en abscisse donc) : **remarquez la flèche « lisse ».**
    - Une contrainte par rapport au bord supérieur (en ordonnée donc)



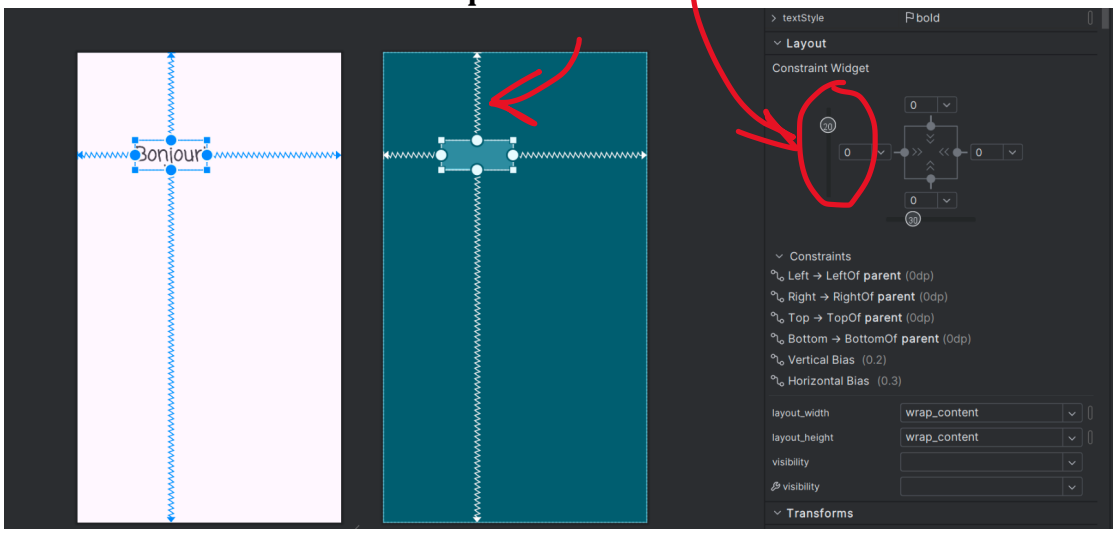
    - Détruire les 2 contraintes inutiles **en les effaçant directement en les sélectionnant grâce à la touche « sup ».**
      On peut aussi effacer toutes les contrraintes un clic droit sur le composant dans le ***component tree*** (voir ci-dessous).



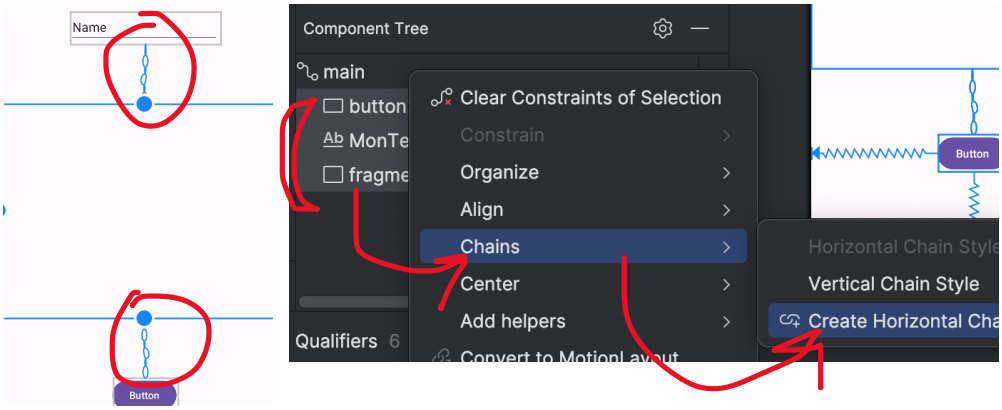
- Prendre le temps aussi de réfléchir aussi aux différents paramètres proposés pour les paramètres ***layout_width*** et ***layout_height.***

- Une autre façon de positionner les objets sur l'interface permet de rendre les applications plus facilement indépendantes de la taille de l'écran de la cible en parametrant des pourcentages. D'abord rajouter les contraintes manquantes grâce à un clic droit dans le ***component tree*** comme ci-dessous :

On positionne un objet selon **un pourcentage** de la largeur et de la hauteur de l'écran. **Remarquez la flèche « ressort ».**



- Les composants graphiques peuvent aussi être **espacés proportionnellement les uns par rapport aux autres** par exemple **grâce aux chaines verticales** ou horizontales : sélectionner **plusieurs** composants, puis avec le bouton droit de la souris, de choisir « Create Vertical Chain» puis « Spread » (= « Etendre »).



D. Collaboration entre deux classes héritant de *Activity*

Une deuxième[1] *technique pour créer une interface utilisateur Androïd Studio élaborée consiste à utiliser plusieurs activités (classe Activity).* ***Cette méthode ne doit être utilisée que s'il y a peu de données à transmettre d'une activité à l'autre***[2] ***.*** *Dans ce cas, comment transmettre les informations de l'interface utilisateur d'une activité à l'autre ? C'est ce que nous verrons dans cette partie.*

3. Rajouter une nouvelle Activity à votre projet:
   - Sélectionner l'onglet de gauche fichier et le sous-onglet ***Android***.

   - Puis cliquer sur ***File/new/Activity/Empty Views Activity***. Nommer la nouvelle activité comme vous le souhaitez (dans ce tutoriel c'est Activity2).
     Remarque : Comment a évolué l'onglet Layout ?

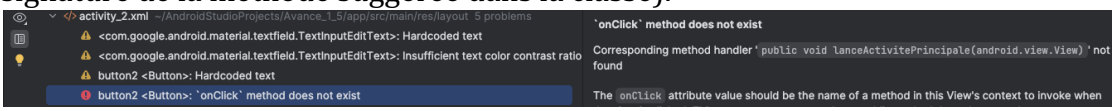
4. Pour savoir quelle est l'activité lancée il faut regarder le fichier ***AndroidManifest.xml*** pour voir celle qui possède les lignes ci-dessous :

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

```
<activity
    android:name=".Activity2"
    android:exported="false" />
<activity
    android:name=".MainActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

5. Nous allons maintenant **faire collaborer les deux activités** de ce programme.
   - Commencer par rajouter dans ***chaque*** activité ***Button*** et une zone de texte ***TextView*** pour l'activité principale et une zone de saisie ***TextInputLayout*** contenant un ***TextInputEditText*** pour la secondaire.
   - Pour l'activité principale, modifier le texte sur le bouton pour donner un sens facile à comprendre et rajouter un événement ***onClick***, en créant la méthode associée qu'on appellera ***lanceActivite2*** (voir tutoriel de découverte en cas de difficultés, mais l'aide fournie par Android studio rend le codage quasiment automatique… il suffit de copier-coller la signature de la méthode suggérée dans la classe).



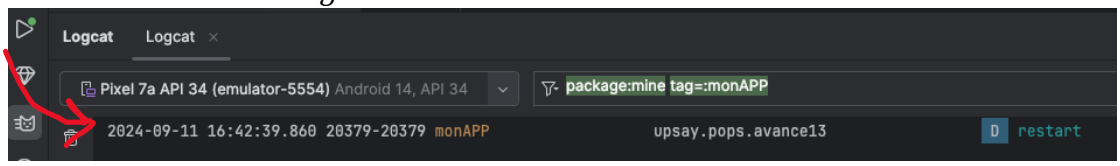
[1] La première méthode consiste à changer de layout comme vu à la fin de la partie A du tutoriel de base.

[2] Les deux cas de figure courants pour une telle utilisation sont : l'utilisation d'un splash screen, ou l'utilisation d'une base de données comme Firebase qui sert d'intermédiaire entre les Activity.

- **Voici la nouveauté : chaque activité est lancée ou activée avec un *Intent*,** qui est un objet *message* qui demande au *runtime* Android de lancer une activité ou un autre composant logiciel contrôlable par l'application (drivers de périphériques du téléphone par exemple).
  Rajouter l'attribut suivant :
  ```
  private ActivityResultLauncher<Intent>
  lanceActivity2;
  ```
  Il faut rajouter
  ```
  lanceActivity2 = registerForActivityResult(
          new ActivityResultContracts.StartActivityForResult(),
          result -> {});
  ```
  le code suivant dans la méthode qui lance la seconde activité :
  ```
  Intent intent = new Intent(this, Activity2.class);
  lanceActivity2.launch(intent);
  ```
- Utiliser le bouton de l'activité 2 pour revenir à l'activité précédente de la même façon que vous l'avez fait pour l'activité 1.
  Voici le code correspondant :
  ```
  Intent messageIntent = new Intent();
  setResult(Activity.RESULT_OK, messageIntent);
  finish();
  ```

Conseils pour faciliter le débogage : deux façons de placer des marqueurs

1. *Pour un affichage de courte durée sur l'application, placer où vous le souhaitez :*
   ```
   Toast.makeText(getApplicationContext(), "mon message",
   Toast.LENGTH_SHORT).show();
   ```
2. *Rajouter l'attribut*
   ```
   private static final String LOG_TAG ="monAPP" ;
   ```
   et le texte qui doit s'afficher dans la fenêtre logcat (en bas d'Androïd Studio)
   ```
   Log.d(LOG_TAG, "restart");
   ```
   Vous obtiendrez dans logcat :



- **Comment passer des *données* de l'activité 2 vers l'activité principale ?**
  On utilise également un ***Intent*** pour transmettre des informations d'une activité à une autre. L'objet ***Intent*** que vous utilisez pour démarrer une activité peut inclure des ***extras*** d'*Intent*, qui sont des données supplémentaires dont l'activité pourrait avoir besoin. Les ***extras*** d'*Intent* sont des paires clé/valeur rajoutées à un Bundle. Un Bundle est dictionnaire de données, stockées sous forme de paires clé/valeur. Puis vous les récupérez dans l'activité réceptrice.

  Voici le code que vous pouvez rajouter au bundle de l'activité 2 un ***extras*** d'*Intent*:
  - Un nouvel **attribut** dans la classe :
    ```
    public static final String EXTRA = "message";
    ```

```
EditText monTexte;
```

- Et **avant** le `setResult(…)`:

```
monTexte = findViewById(R.id.MonTexte);
String message = monTexte.getText().toString();
messageIntent.putExtra(EXTRA, message);
```

- **Comment récupérer des données provenant de l'activité 2 dans l'activité principale ?**

  Dans l'activité principale rajouter le code suivant en gras :

```
lanceActivity2 = registerForActivityResult(
        new ActivityResultContracts.StartActivityForResult(),
        result -> {
             if (result.getResultCode() == Activity.RESULT_OK) {
                   Intent data = result.getData();
                   if (data != null) {
                         String message =
                         data.getStringExtra(Activity2.EXTRA);
                         TextView textView =
                         findViewById(R.id.textView);
                         textView.setText(message);
                         }
                    }
               });
```

E. Utiliser des *Fragment* pour faciliter la communication entre classes

Une troisième technique pour créer une interface utilisateur Androïd Studio élaborée consiste à utiliser une seule Activity mais plusieurs fragments (classe Fragment). ***Cette méthode est la plus souple****, mais une quatrième méthode appelée Jetpack associée à Kotlin est maintenant recommandée, elle est pourvue d'un autre outil de RAD.*

Il s'agit d'utiliser une seule ***Activity*** en changeant son apparence en fonction des besoins grâce à des classes filles de la classe ***Fragment.*** **L'avantage par rapport à la multiplication d'activités, c'est que les Fragments peuvent facilement accéder à l'espace des données de l'*Activity* à laquelle ils appartiennent,** alors que seuls les Intents permettent de passer des données d'une Activity à une autre. Un ***Fragment*** est comme une Activité miniature. Bien qu'il doive être hébergé par une ***Activity***, un ***Fragment*** est une classe qui **a son propre cycle de vie.** Un ***Fragment*** peut rester à l'écran pendant tout le cycle de vie de l'activité, ou **il peut être une partie dynamique de l'interface utilisateur, ajoutée et retirée pendant que l'activité est en cours.**

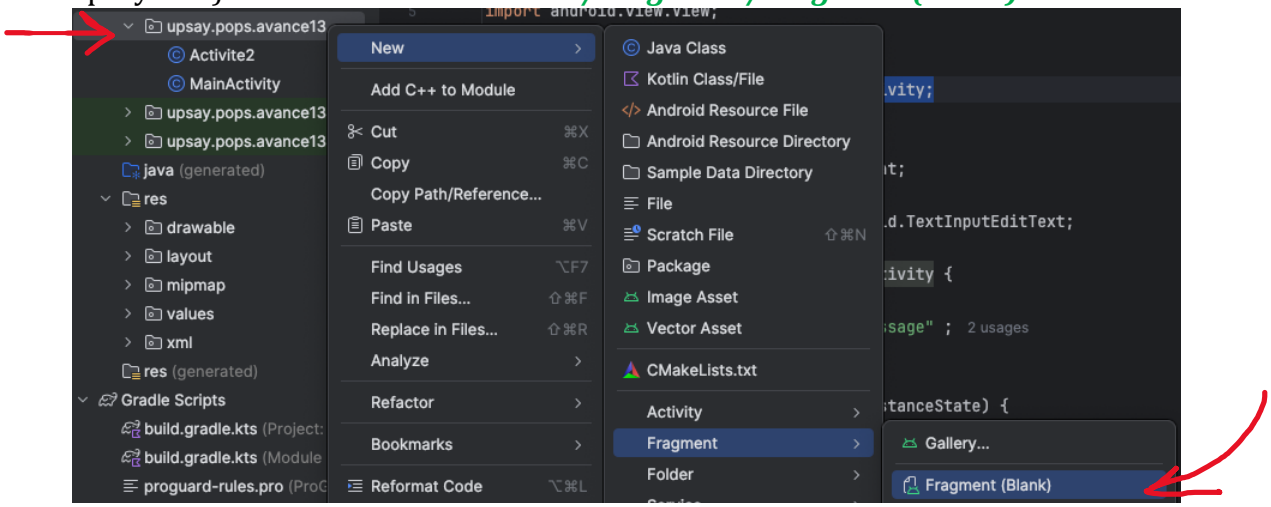
Les classes fondamentales pour la gestion des fragments sont : ***Fragment, FragmentManager*** et ***FragmentTransaction.***

6. Avertissement : pour utiliser les Fragments nous utilisons un « package » récent des API d'Android Studio, nommé AndroidX. Vérifier dans les imports que le package AndroidX est bien déclaré dans la ligne :

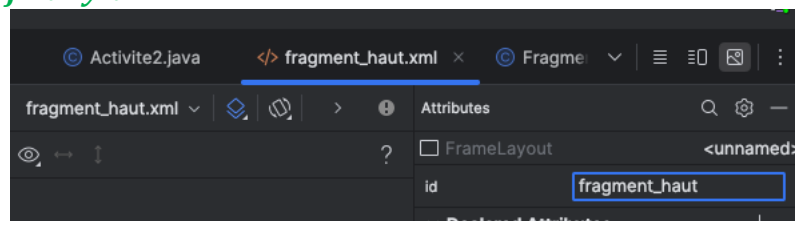
   import androidx.appcompat.app.AppCompatActivity;

Sinon cela devrait être proposé dans la suite ou il faudra utiliser la commande ***Refactor / Migrate to Android X***

7. Création d'un fragment
   - Dans la vue Project développer ***app/java.***
   - En faisant un clic droit sur le nom de votre package (normalement upsay.XXX) choisir la commande ***New/Fragment /Fragment (blank)***.



   - Dans la boîte de dialogue Configure Component, nommez le Fragment **FragmentHaut** (toujours en langage Java !).
   - L'option Create layout XML devrait déjà être sélectionnée, et le layout du fragment sera nommé ***fragment_haut***.
   - Cliquer sur Terminer pour créer le fragment.
   - Ouvrir ***fragment_haut.xml*** à partir du ***Ressource Manager*** et en mode Design donner un identifiant XML (propriété ***id***) au layout du fragment pour pouvoir le récupérer à partir de ***MainActivity*** grâce à la méthode ***findById***.



   - Ajouter une couleur de fond au layout, au choix (mais différente pour chaque fragment) et le texte « Page1 » dans un TextView centré horizontalement.
   - Recommencer avec un fragment nommé ***FragmentBas*** avec une couleur différente et le texte « Page2 »

8. Créer deux boutons ***page1*** et ***page2*** en bas de l'activité secondaire déjà créée dans la partie C que vous avez peut-être appelé ***Activity2.***

9. Nous allons maintenant prévoir le « cadre » des fragments dans le fichier xml de ***Activity2.*** Pour cela on rajoute un ***`FragmentContainerView`*** au-dessus des deux boutons de l'activité secondaire.

- Normalement à sa création le container vous demande de rajouter le fragment qui apparaitra au départ en cliquant dessus (c'est importat d'avoir créé les classes Fragment avant d'avoir créer le container). Après avoir cliqué, vérifier dans les propriétés du container que l'attribut **name** contient bien le fragment choisi, dans notre exemple ce devrait être ***upsay.ET5.avance_1_5.FragmentHaut***
- Pour le besoin de la prévisualisation un message d'erreur vous demande rajouter la ligne suivante dans le xml : il suffit de cliquer sur la ligne apparaissant dans le message, et vérifier le fichier xml est la suivante :

  tools:layout="@layout/fragment_haut"
- Donner un identifiant à celui-ci, par exemple **frame**.
- L'ensemble ressemble à ça (remarquez qu'il n'y a plus de retour possible vers MainActivity).

10. Pour chaque bouton rajouter sur la propriété ***onClick*** respectivement la méthode **`lancePage1`** et **`lancePage2`**. Comme d'habitude, créer ces méthodes dans l'activité secondaire.

11. **Pour utiliser les deux fragments à partir de *l'activité secondaire* rajouter le code suivant** :
    - Créer les fragments et leurs références en tant qu'**attributs** de classe :

```
Fragment page1 = new FragmentHaut() ;
Fragment page2 = new FragmentBas() ;
```

    - Déclarer comme **attribut** de la classe Activity2 le **fragment manager** :

```
FragmentManager fm = getSupportFragmentManager();
```

- Copier dans chaque méthode appelée par les boutons, le code pour activer le fragment correspondant, par exemple :

```
FragmentTransaction ft = fm.beginTransaction();
ft.replace(R.id.frame, page1);
ft.addToBackStack(null);
ft.commit();
```

En guise conclusion de ce tutoriel, vous avez actuellement une application qui peut servir de point de départ pour la suite du cours car elle a un format assez standard. Elle présente d'abord un écran d'accueil qui une fois passé permet d'accéder au cœur de l'application. Cette seconde Activity possède deux fragments différents qui pourraient correspondre à des configurations différentes des fenêtres de l'application qui seraient deux cas d'utilisation différents de celle-ci. Le but de ces deux tutoriels est de donner des outils avec le minimum de code à saisir permettant de réaliser des applications très variées. Évidemment on peut faire beaucoup plus beau et il y a une énorme variété de possibilités grâce aux bibliothèques Androïd qui restent à découvrir. Les liens mentionnés en première page vous renvoient vers quelques formations de qualité.