REPUBLIQUE DU CAMEROUN
- - - - - - - - - - - - - - - - -
UNIVERSITE DE DSCHANG
- - - - - - - - - - - - - - - - -
Faculté des Sciences
- - - - - - - - - - - - - - - - -
Département de mathématiques et d'Informatique

UE: **MAM416 MC Applications mobiles**
Enseignant : **Miguel Landry FOKO SINDJOUNG**
Contact : _____
E-mail : miguelfoko@gmail.com
Apprenant : Essowèmlou NDANATCHE
E-mail1 : ndanager@yahoo.fr

RAPPORT de TP

Rapport thème :

RAPPORT DE TP-MODELISATION ET CONCEPTION D'UNE APPLICATION CALCULATRICE SOUS ANDROID.

Réalisé par:

- Essowèmlou NDANATCHE (E-mail : ndanager@yahoo.fr Contact : 00228 90 75 67 45)

UNIVERSITE DE DSCHANG

TABLE DE MATIERE

I. TRAVAIL A FAIRE

Il s'agit de réaliser un TP pour modéliser et concevoir une application mobile (**Calculatrice**) en général qui nous permettra :

- ✓ de revoir le principe des évènements, et du positionnement des Buttons principalement,
- ✓ de faire de la programmation hybride, soit de la programmation native,
- ✓ de produire le fichier **apk** installable sous android.

Par conséquent, voici la démarche à suivre :

- Créer un nouveau projet et le nommer,
- Ensuite, créer un **projet Github** portant notre nom,
- Envoyer le code source de notre projet en se rassurant d'ajouter le fichier **apk** installable,
- Envoyer finalement l'invitation au Professeur à notre projet Github (id github : **miguelfoko ou** **miguelfoko@gmail.com**),
- Produire un fichier global de la classe où on y retrouve deux colonnes : **Les identifiants github et les noms des étudiants correspondants.**
- **Obtenir enfin le résultat suivant en capture d'écran :**

II. CONTEXTE DU PROJET

Ce TP s'inscrit dans le cadre de la familiarisation et de l'appropriation des bonnes pratiques et maîtrise de la conception et modélisation des applications mobiles hybrides ou natives.

III. PLAN D'ILLUSTRATION DU RAPPORT.

A. ANDROID-STUDIO

1. INSTALLATION

ETAPE 1 : CREATION D'UNE MACHINE VIRTUELLE AVEC OS UBUNTU 20.04

ETAPE 2 : INSTALLATION D'ANDROID STUDIO

B. CREATION DU PROJET MONCALC

ETAPE 3 : PRODUCTION DU VISUEL DE L'APPLICATION

ETAPE 3 : PRODUCTION DU CODE SOURCE (DEFINITION DU CODE JAVA)

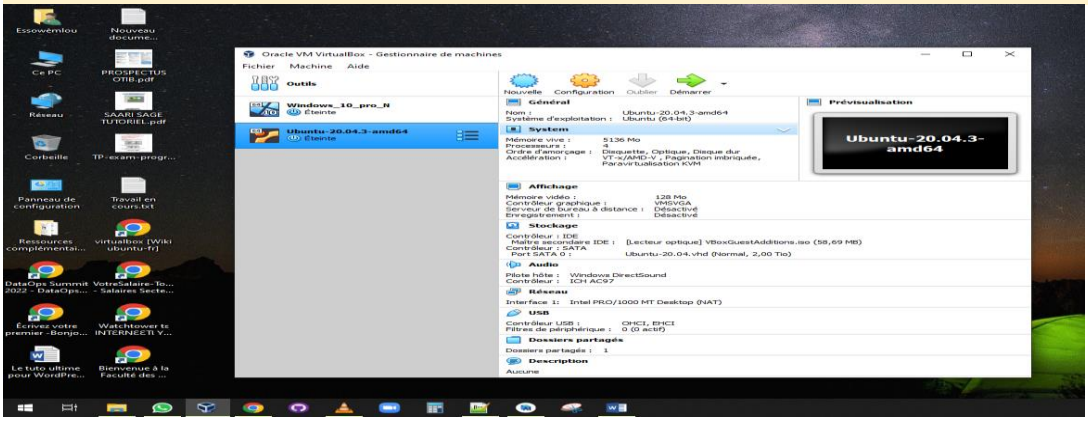
IV. RESULTATS DU TP

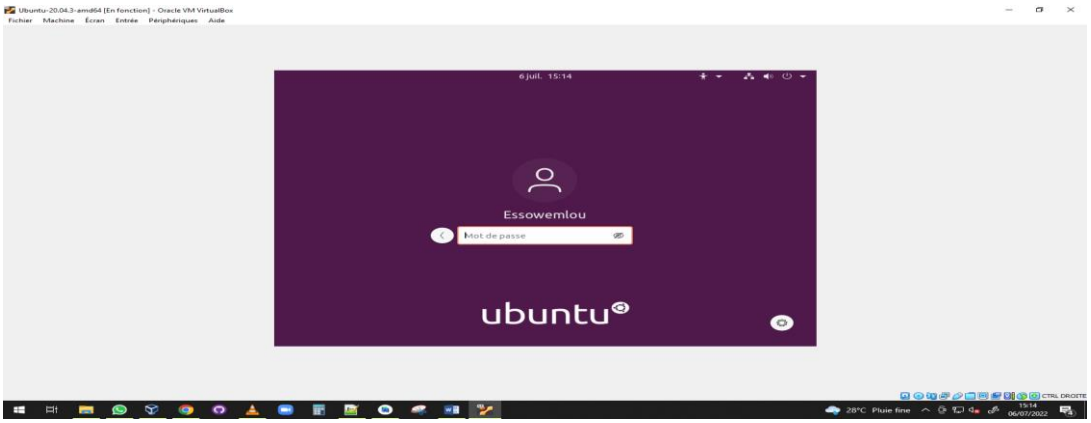
Le présent TP nous a permis de s'initier au framework hadoop et au patron MapReduce en utilisant le docker pour lancer un cluster hadoop de 3 noeuds. Nous avons utilisé pour la circonstance l'OS **ubuntu 20.04-amd64**.

A. ANDROID-STUDIO

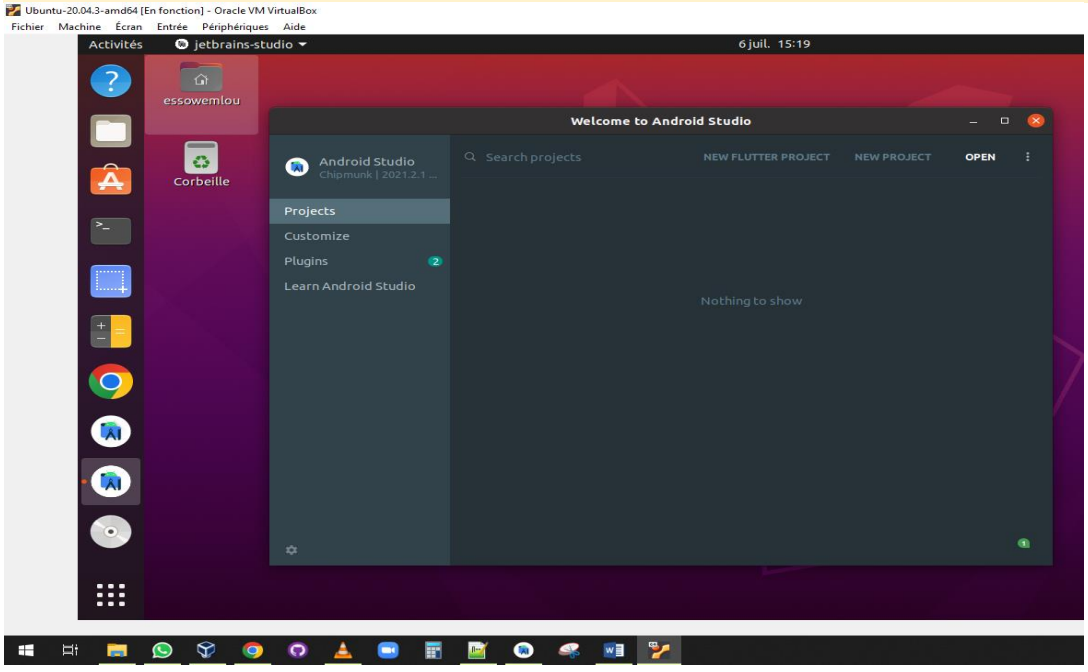
1. INSTALLATION

ETAPE 1 : CREATION D'UNE MACHINE VIRTUELLE AVEC OS UBUNTU 20.04





ETAPE 2 : INSTALLATION D'ANDROID STUDIO



B. CREATION DU PROJET MONCALC

ETAPE 3 : PRODUCTION DU VISUEL DE L'APPLICATION

L'organisation générale demandée peut se décomposer en un texte en haut de l'écran et un tableau de boutons en bas de l'écran. En utilisant le ConstraintLayout fourni par défaut, on peut donc placer un TextView en haut de l'écran sur l'intégralité de la largeur avec un texte aligné à droite :
Voici le résultat en capture d'écran :

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://s
    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"
    tools:context=".MainActivity">

    <DigitalClock
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="25sp"
        android:textStyle="bold"
        tools:layout_editor_absoluteX="627dp"
        tools:layout_editor_absoluteY="-1dp"
        tools:ignore="MissingConstraints" />

    <TextView
        android:id="@+id/screen2"
        android:layout_width="730dp"
        android:layout_height="20dp"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="false"
        android:layout_alignParentEnd="true"
        android:layout_marginTop="28dp"
        android:background="#807979"
        android:fontFamily="monospace"
        android:gravity="end"
        android:text=" Ma Calculatrice"
        android:textAlignment="viewStart"
```

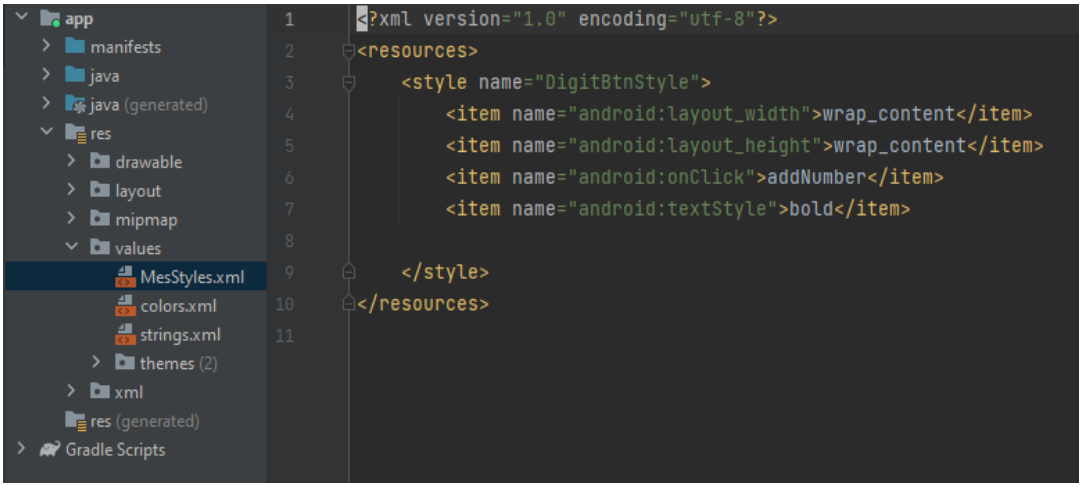
Pour le tableau de boutons en bas d'écran, il y a plusieurs manières d'obtenir cette organisation. Par exemple, avec des LinearLayout imbriqués et un jeu de poids (layout_weight) pour que le bouton = soit plus grand. On aurait aussi pu utiliser un Grid_Layout. Je vous propose d'utiliser un TableLayout qui permettra de garantir l'égalité des tailles des boutons où on pourra utiliser la notion de colspan pour que le bouton = soit sur exactement 2 cases.
On déclare donc un TableLayout aligné sur le bas de l'écran, à qui l'on donne l'intégralité de la largeur de la fenêtre et à qui on dit que les cases sont redimensionnables si nécessaire (stretchColumnsetshrinkColumns). Chaque ligne du tableau doit ensuite être dans une balise TableRow.

Enfin, le bouton = devra simplement avoir une propriété en plus pour lui dire d'occuper 4 cases :android:layout_span="4".

```
        tools:ignore="HardcodedText" />

    <TableLayout
        android:id="@+id/tableLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_alignParentEnd="true"
        android:layout_alignParentBottom="true"
        android:layout_marginLeft="8dp"
        android:layout_marginRight="8dp"
        android:layout_marginBottom="8dp"
        android:background="#252424"
        android:shrinkColumns="*"
        android:stretchColumns="*"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        tools:ignore="UselessLeaf">

        <TableRow
            android:layout_width="match_parent"
            android:layout_height="match_parent">

            <Button
                android:id="@+id/btn7"
                style="@style/DigitBtnStyle"
                android:text="7"
                android:textSize="28sp" />
```

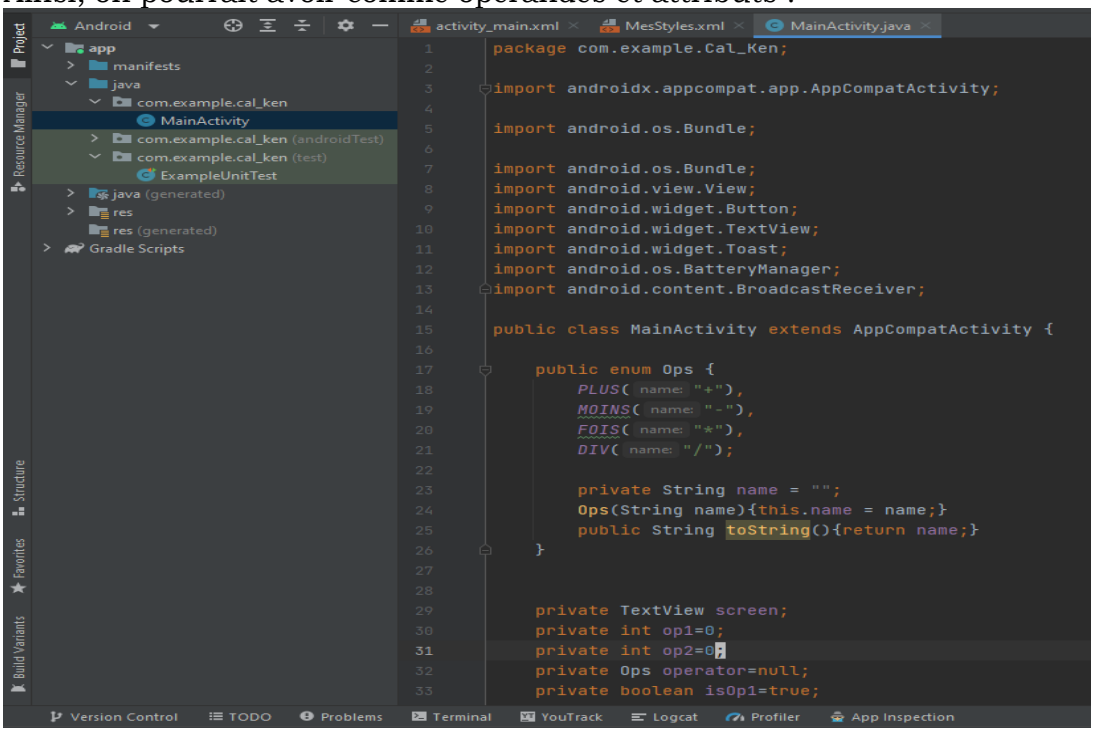
Afin de réduire les copiés/collés et de faciliter le maintenabilité du code, il est possible de définir une feuille de style définissant ce qu'est un bouton

comme le montre bien notre fichier de style nommé MesStyles.xml que l'on aurait pu créer dans le dossier values :



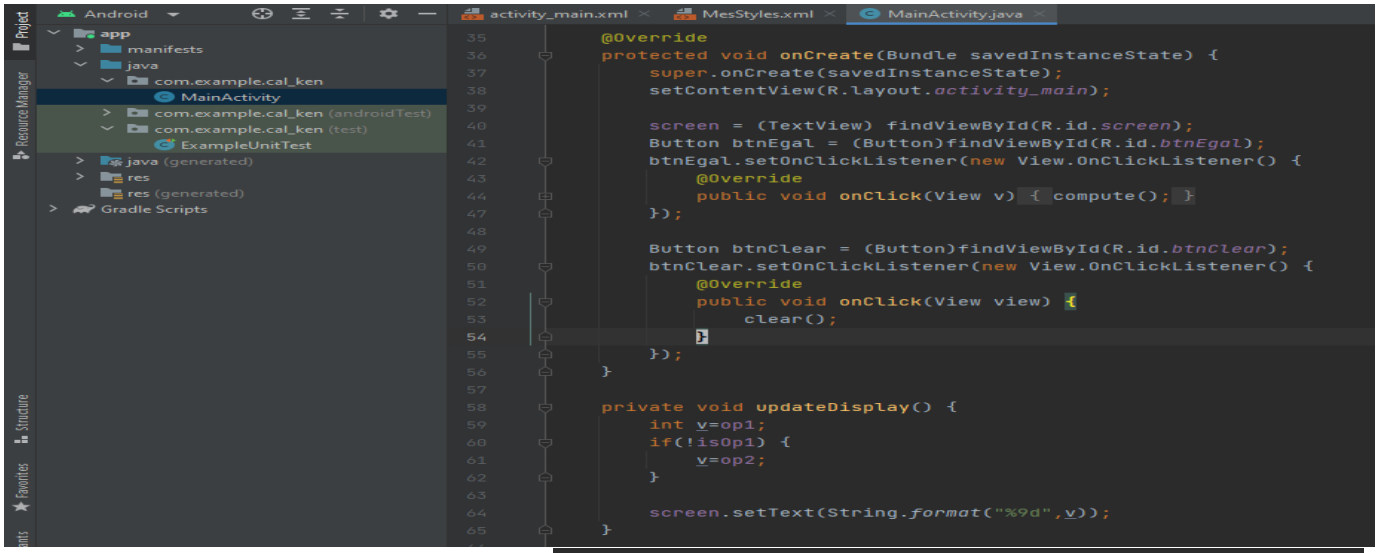
ETAPE 3 : PRODUCTION DU CODE SOURCE (DEFINITION DU CODE JAVA)

Au niveau du comportement on se rend compte que pour faire des opérations binaires (donc avec 2 opérandes), il faudra mémoriser 2 opérandes et un opérateur. L'action de `=` sera celle qui fera le calcul. Comme il n'y a qu'un seul affichage, il faut aussi mémoriser si on est en train de saisir le premier ou le second opérande. Dans ces attributs, j'ai fait un choix de représentation pour l'opérateur. On pourrait utiliser simplement un caractère (`'+', '-', '*'  et '/'`), mais je vous propose plutôt de faire une énumération, qui permet de caractériser par null, l'absence d'opérateur. Ainsi, on pourrait avoir comme opérandes et attributs :



Ainsi, lancer le calcul ne correspondrait qu'à faire l'opération demandée entre les 2 opérandes en mémoire, stocker le résultat en premier opérande et mettre à jour l'affichage :

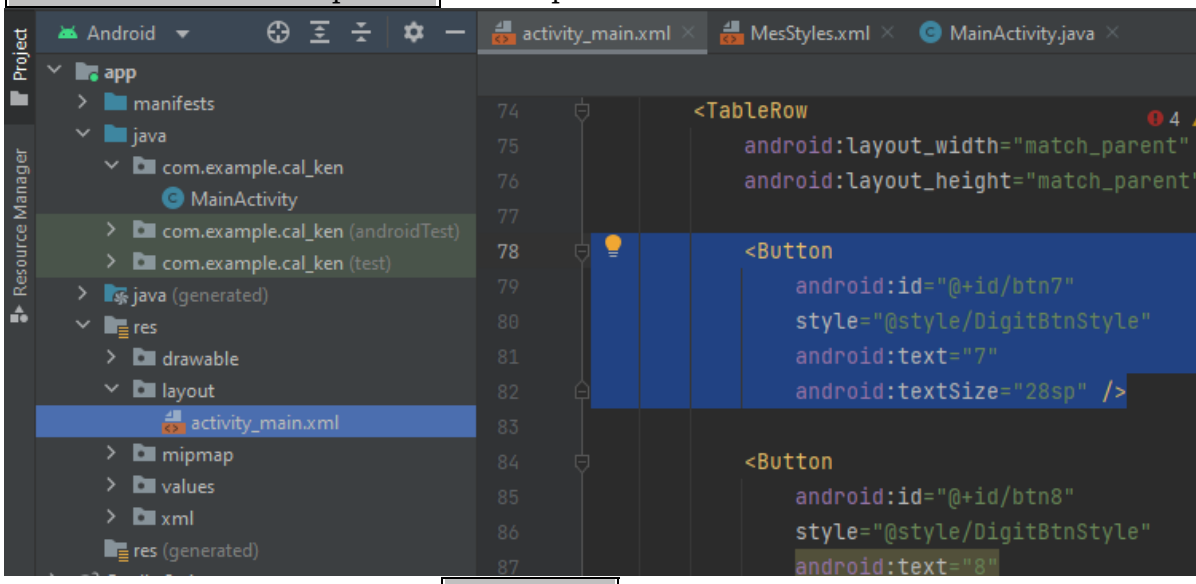
Cette action serait à appeler lorsque le bouton `btnEgal` serait pressé. Il faudrait donc appeler cette méthode dans un écouteur associé à `btnEgal`. En choisissant de le faire via une classe anonyme, on pourrait écrire dans `onCreate` :




```
public void compute() {
    if(isOp1) {
        // do nothing
    } else {
        switch(operator) {
            case PLUS  : op1 = op1 + op2; break;
            case MOINS : op1 = op1 - op2; break;
            case FOIS  : op1 = op1 * op2; break;
            case DIV   : op1 = op1 / op2; break;
            default : return; // do nothing if no operator
        }

        op2 = 0;
        isOp1 = true;
        updateDisplay();
    }
}

private void clear() {
    op1 = 0;
    op2 = 0;
    operator = null;
    isOp1 = true;
    updateDisplay();
}
```

```
public void setOperator(View v) {
    switch (v.getId()) {
        case R.id.btnPlus  : operator=Ops.PLUS;  break;
        case R.id.btnMoins : operator=Ops.MOINS; break;
        case R.id.btnFois  : operator=Ops.FOIS;  break;
        case R.id.btnDiv   : operator=Ops.DIV;   break;
        default :
            Toast.makeText( context: this, text: "Opérateur non reconnu",Toast.LENGTH_LONG);
            return; // do nothing if no operator
    }
    isOp1=false;
    updateDisplay();
}


public void addNumber(View v){
    try {
        int val = Integer.parseInt(((Button)v).getText().toString());
        if (isOp1) {
            op1 = op1 * 10 + val;
            updateDisplay();
        } else {
            op2 = op2 * 10 + val;
            updateDisplay();
        }
    }catch (NumberFormatException | ClassCastException e) {
        Toast.makeText( context: this, text: "Valeur erronée",Toast.LENGTH_LONG);
    }
}
```

Pour les opérateurs, on souhaiterait aussi faire une unique méthode gérant les 4 cas, mais on voudrait éviter de faire 4 fois ce gros bloc de lignes pour associer le bouton à la méthode. Je vous propose de varier et de le faire via le *XML*. À chacun des boutons d'opérateur, ajoutez la propriété android:onClick="setOperator". Exemple :

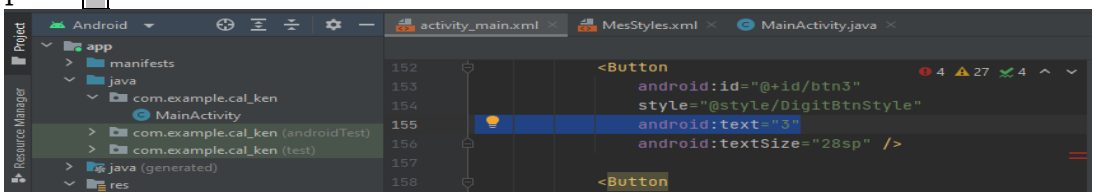


Définissez aussi la méthode setOperator. Cette méthode devra prendre une View en paramètre et retrouver le bouton qui a été pressé pour agir en conséquence. Pour ce faire, nous pouvons utiliser la méthode getId() présente dans toutes les vues et qui permet de récupérer l'identifiant de la vue qui a généré l'événement. En comparant cette valeur aux 4 *id* possibles, on obtient le script comme le montre la capture du fichier .Java plus haut.

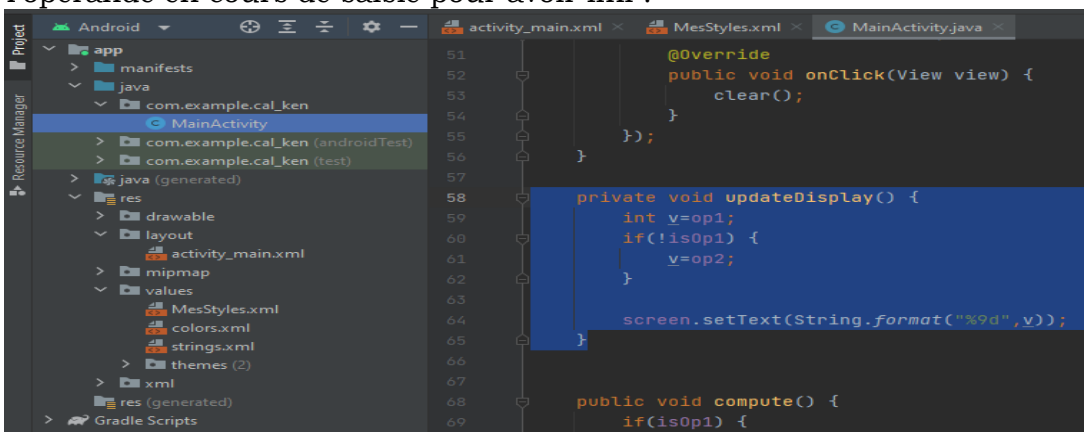
Enfin, pour les boutons associés à des chiffres, on pourrait pratiquer de la même manière. Cependant, plutôt que de passer par une énumération de choix, je vous propose de le faire plutôt par conversion numérique de la valeur présente dans le texte du bouton. Ajouter un digit à un opérande revient ensuite à multiplier la valeur actuelle de l'opérande par 10 et à y ajouter la valeur du nouveau digit.

Pour varier, pour associer l'action du bouton aux 10 boutons de chiffre, je vous propose de passer par une feuille de style. En effet, comme pour les autres propriétés, il est possible de préciser la propriété onClick dans la feuille de style. En définissant le style suivant :

On pourrait alors avoir une définition très légère des boutons. Exemple pour 3 :



Il ne manque alors qu'à écrire la méthode de publication de la valeur de l'opérande en cours de saisie pour avoir fini :



V. CONCLUSION

Ce projet nous a permis de mettre en œuvre toutes nos facultés pour nous permettre d'appréhender le concept de modélisation et conception des applications mobiles.

VI. BIBLIOGRAPHIE