`

Tutoriel *Bases*
User Interface ANDROID

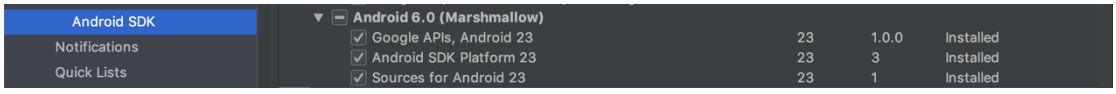
Objectifs visés par ce tutoriel

- Utilisation de boutons, de zones de texte et de layouts (canevas ou gestionnaire de disposition des composants graphiques)
- Utilisation de fonctions graphiques de base (formes, texte, incrustation d'images)
- Utilisation d'un timer
- Utilisation des évènements utilisateurs courants : touchpad (clic) et accéléromètre

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.
- Android Studio 2021.1.1
- Gradle 7.2 Plugin 7.1.3 (à vérifier en cas de problèmes de compilation voir project struture)
- Android 6.0 marshmallow ; SDK 6 with Application Programming Interface API 23 ; revision 3 : 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.
- S'il y a un second SDK installé, assurez-vous qu'il ne soit pas postérieur à la version du Androïd Studio utilisé.

  Le **SDK Manager** vous permet de savoir ce que vous avez comme versions installées, et d'en installer d'autres.



  Une fois le projet ouvert c'est la fenêtre **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

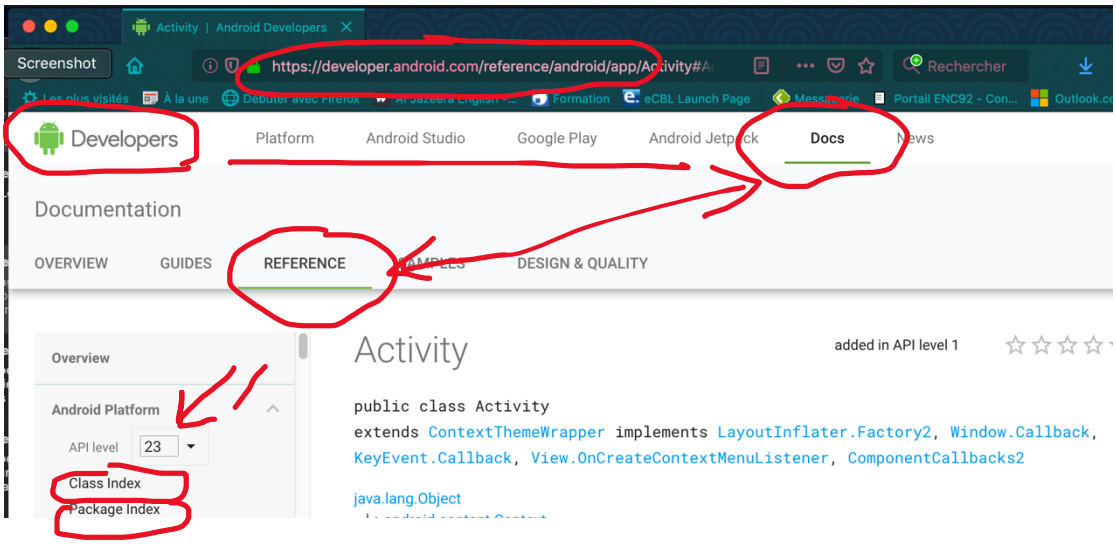


Figure 1

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--cd0342
- https://classroom.udacity.com/me

A. Utilisation de boutons, de zones de texte et de layouts (canevas)

Android Studio *intègre une interface permettant le « Rapid Application Development », c'est-à-dire la génération de code sans saisie de script via une palette de composants graphiques. Sous Androïd Studio seule la création de l'interface utilisateur bénéficie de cette possibilité. Le code généré est à la fois du code XML et code Java.*

1. Dans le menu ***Quick Start***, ou ***File/new…*** sélectionner ***Start a new Android Studio Project***.
   - Choisir ***Empty View Activity***.
   - Donner le nom de votre choix à l'application.
   - Modifier le nom du package par défaut en *UPsay.decouverteAndroid*.
   - Notez bien l'endroit où sera enregistré le projet dans la zone ***Save Location***.
   - **Assurez-vous** que le langage est bien Java.
   - Choisir la version 6 d'Androïd (notez au passage le pourcentage de systèmes sur lesquels s'exécutera votre application). Ne rien sélectionner de plus, puis cliquer sur ***finish***.

L'interface utilisateur des applications Androïd utilisent les notions (ou classes) suivantes

*- **Activity** : élément graphique complexe remplissant tout l'écran.*

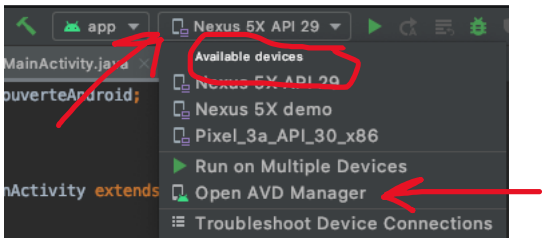
*- **View :** tout composants graphiques. Le seul moyen d'être exhaustif pour présenter les Views est de vous renvoyer à la documentation en ligne de Android Studio.* ***Regardez comment y accéder sur la figure 1 (plus haut)****.*

*- **ViewGroup :** un groupe de Views, souvent un **Layout**.*

*- **Layout** : en français « gestionnaire de disposition », permet de placer automatiquement les différents composants graphiques dans l'Activity. Cet élément a énormément de conséquences sur l'apparence de l'application lorsque la taille de l'écran est différente, d'un terminal à un autre.*

---

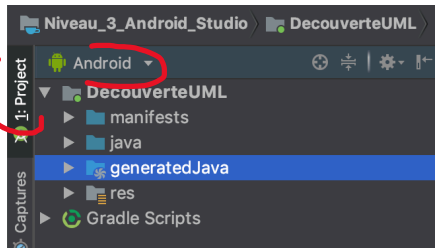
2. Android Studio propose des émulateurs couvrant une large palette d'appareils standard afin de valider en simulation le programme. Pour voir ce qu'il en est, cliquer ***à gauche*** de la flèche de lancement du programme. Alors vous verrez si vous avez connecté votre téléphone dans ***Connected Devices*** (ici none) et dans ***Available Devices*** les appareils virtuels que vous avez peut-être créé.



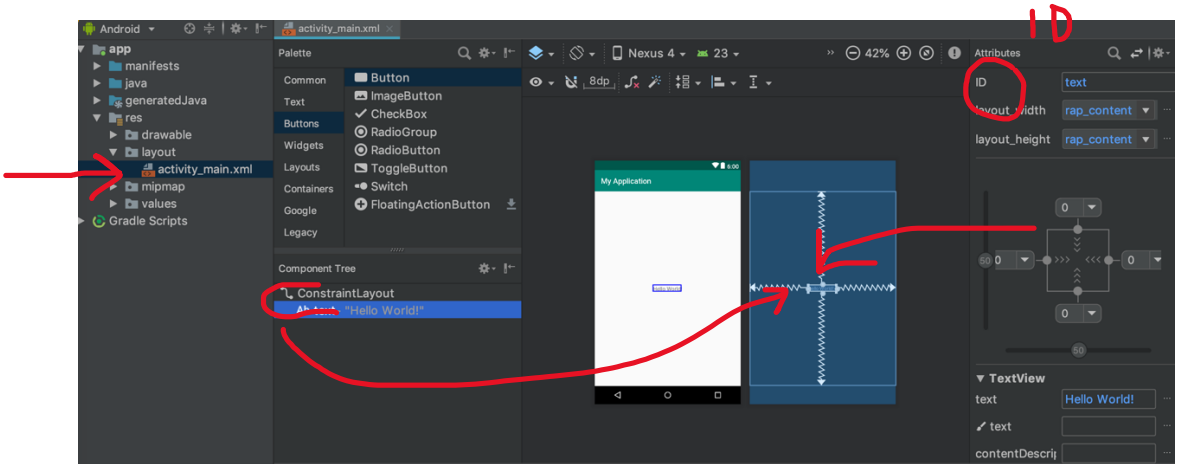
Le gestionnaire est aussi accessible dans le menu ***Tools/ Device Manager*** . Sur le plus en haut à gauche *Create a virtual device*.

Remarque : pour les cibles non physiques (émulateur) HAMX permet l'utilisation des ressources physiques mais exige l'autorisation de virtualisation. Si ceci n'est pas autorisé par le BIOS il y a blocage de HAMX. Avec Windows 10 il faut installer le Media Feature Package. Avec Windows 11 il faut cocher la fonctionnalité Plateforme de machine virtuelle.

3. Vous pouvez maintenant lancer l'application « vide ».
   - Notez que cette phase peut-être très longue à cause des ressources importantes requises par l'émulateur. Ne vous impatientez pas. Ne cliquez pas frénétiquement ! Laisser **Gradle** (éditeur de lien) faire son travail. Vous devez obtenir un écran de téléphone standard, puis l'application s'ouvre.
   - **Pendant ce temps** remarquez à droite du téléphone virtuel une colonne permettant d'utiliser les commandes du téléphone habituellement présentes sur la tranche de celui-ci. Et si vous cliquez sur les '…' en bas vous obtiendrez d'autre paramètres qui simulent certaines fonctions non présentes : GPS, microphone, accéléromètres etc…
   - Une fois cela terminé, assurez vous que la fenêtre ***1 :Project*** est paramétrée en mode ***Android*** comme ci-contre:



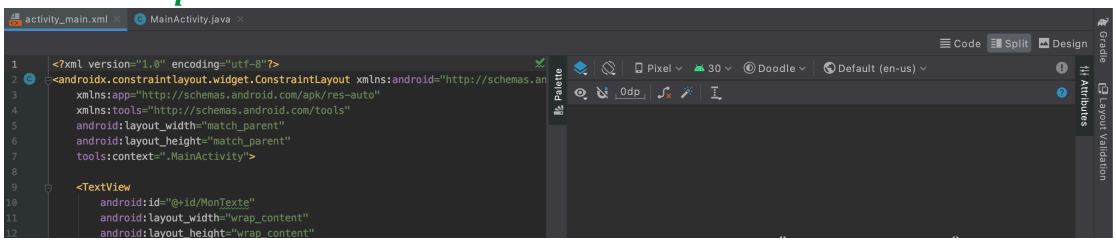
4. Dans la fenêtre 1 : Project vous trouverez le répertoire ***res*** (ressources). Développez ce répertoire et ouvrez le répertoire *layout*. Double cliquez sur le fichier *.xml* qu'il contient. Vous devez obtenir ce qui suit.



- Quels sont les éléments de la fenêtre ci-dessus caractéristiques d'un environnement de RAD ?

- Cliquez sur le TextView « Hello world » placé sous le *ConstraintLayout* de l'arborescence des composants (Component Tree). A droite apparaissent les attributs de ce composant graphique. **Modifiez l'identifiant désigné par *ID* en « monTexte ».**

- Pour voir le code *source* du fichier xml qui génèrera sur le téléphone l'apparence graphique de cette fénêtre, cliquez en haut à droite de la fenêtre sur ***Split***.



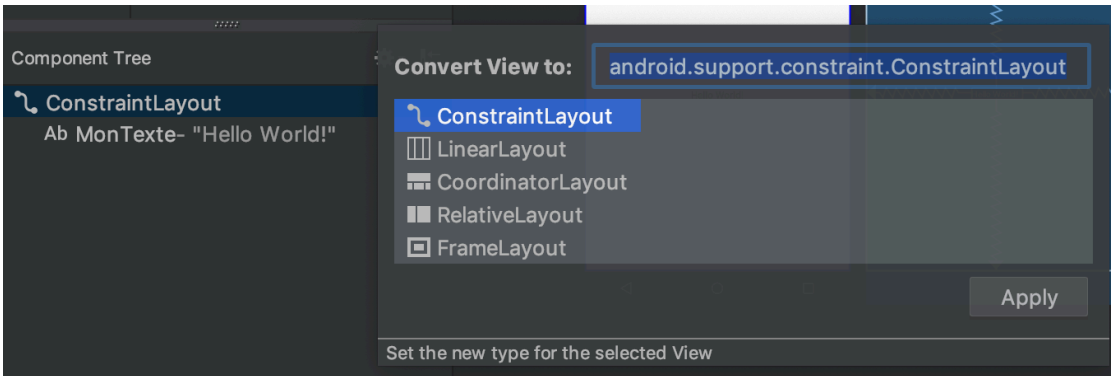
- Retrouvez l'identifiant du TextView décrit dans le code xml. Où apparaît cet identifiant ? Recopiez ci-dessous précisément la ligne.

- Pour information, toujours dans le code xml vous pouvez en profiter pour changer le nom de l'application à votre guise en vous inspirant de ce qui s'est passé pour le TextView. Pour cela ouvrez le fichier ***manifest.xml***.

  RETENEZ QUE TOUTE MODIFICATION RAD GENERE UN CODE XML EN COHERENCE.

5. Modifions l'interface utilisateur de cette application en revenant dans le mode RAD : cliquer sur *Design* en haut à droite.

- Modification du Layout : avec un clic droit sur le Layout (ici un *ConstraintLayout* sous la partie *Component Tree*) vous pouvez le modifier en cliquant sur *Convert View*. Choisir un ***LinearLayout (vertical)*** (en deux temps)



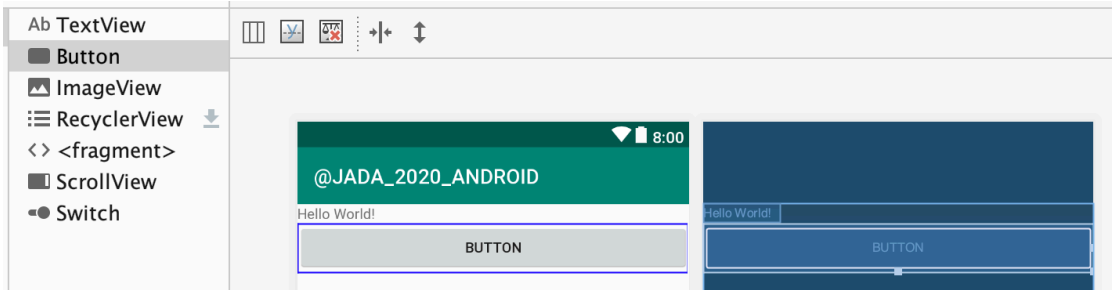
Le choix et le réglage du Layout est important puisqu'il fixe la disposition définitive des éléments graphiques. Voici quelques exemples de Layouts courants :

- ***Constraint Layout***
- ***Linear Layout***
- ***Coordinator Layout***
- ***Relative Layout***
- ***Frame Layout***
- ***Table Layout***

---

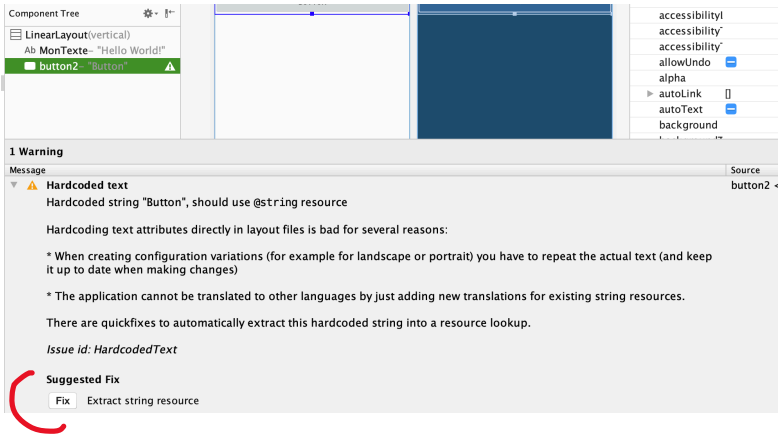
6. Rajouter un bouton

- Dans Palette choisir *Button* et le placer



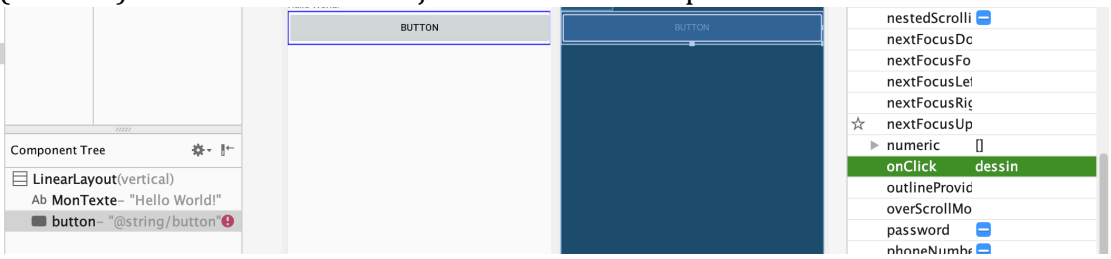
- Le point d'exclamation qui apparaît sous le ***Component Tree*** vous aide à transformer le code comme conseillé en cliquant sur ***fix***. La ligne suivante est modifiée avec le **@string**

android:text="@string/textview" />

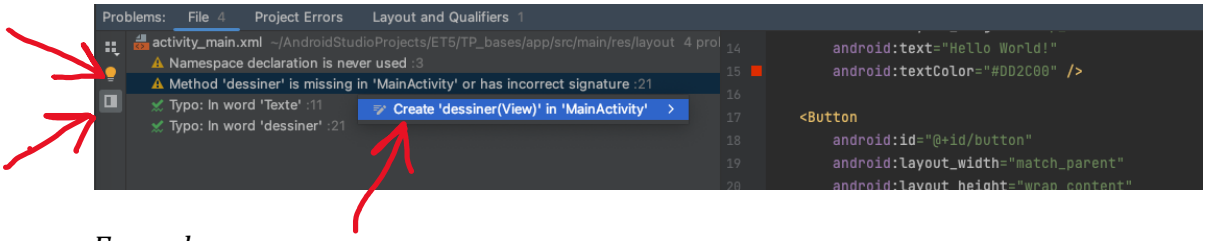


- Dans l'arborescence du projet dérouler le répertoire ***RES/values*** et ouvrir ***string.xml***. Vous devez y retrouver les ressources texte de votre application, par exemple la ressource *textview* qui contient le texte que vous avez prévu d'afficher pour le bouton.
  Remarque : L'expression @string traite le texte qui suit comme un élément référencé dans string.xml, fichier permettant de donner des équivalents d'un libellé dans plusieurs langues.

7. **Répondre aux évènements**
   - Sélectionner le bouton *button* puis en regardant dans ***Common Attributes*** (à droite) trouver ***onClick*** et rajouter dans le champ à côté le mot ***dessiner***



   - Après compilation un point d'exclamation apparaît : **Android Studio explique qu'il faut rajouter une méthode** (nom des fonctions en programmation orientée objet) dont la signature (nom du prototype en programmation orientée objet) vous est donnée ci-dessous. Recopiez le code dans la classe Java **correspondant à l'activité (MainActivity)**.



Exemple :

```
public void dessin(android.view.View view){

}
```

   - Si le View est en rouge comme ci-dessous, placez le curseur dessus. Android Studio vous recommande alors de rajouter le chemin au package contenant la classe *View*. Faire le raccourci clavier qu'il préconise

```
package fr.iutcachan.jada_2020_android;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class Pacpacman extends AppCompatActivity {

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

    public void dessin(View view) {
    }

}
```

Verifiez que la ligne suivante s'est alors rajoutée automatiquement :

import android.view.View;

8. Nous allons apprendre à utiliser le Debugger (c'est très simple).
   - Mettre un point d'arrêt dans la colonne à gauche de la méthode *dessin*. Pour cela il suffit de cliquer et un point rouge apparaît. **Lancer en mode debugger**  . Puis cliquez dans l'application (sur l émulateur ou le téléphone) sur le bouton… le programme s'arrête ! C'est bon signe, la méthode répond bien à l'événement « clic » !
   - Arrêter le débogueur en cliquant sur le carré rouge et enlevez le point d'arrêt.
   - Pour qu'il se passe quelque chose lorsqu'on clique sur le bouton **il faut maintenant écrire du code Java à l'intérieur de la méthode *dessin***. A partir de là plus de développement rapide… il faut écrire du code Java ! Essayez le code ci-dessous dans lequel on suppose que *MonTexte* est l'identifiant de votre objet *TextView*.

```
TextView editText = (TextView) findViewById(R.id.MonTexte);
editText.setText("YES !!!");
```

   - Vérifiez le bon fonctionnement.

IMPORTANT: à quoi sert la méthode ***findViewById*** ?

IMPORTANT : quel est l'argument qu'on doit donner à cette méthode ?

A RETENIR: POUR UTILISER UN OBJET GRAPHIQUE ON PEUT TROUVER DIRECTEMENT SA REFERENCE PAR LES IDENTIFIANTS GRACE A LA METHODE findViewById qui fouille la classe R.java pour trouver la référence de l'objet graphique qu'on souhaite manipuler.

Voici un principe fondamental de la programmation Androïd : il existe un fichier caché important, accessible uniquement grâce à un navigateur dans le répertoire ***GeneratedJava/UPSay/decouverteAndroid…*** le fichier ***R.java*** (R comme référence). Ce fichier est mis-à-jour chaque fois qu'une modification est apportée à la description de l'interface utilisateur dans les fichiers XML. Ce fichier contient la classe *id* qui **contient les références** (adresses où se trouve l'objet correspondant dans la mémoire de la machine virtuelle Java) **de tous les éléments graphiques de votre application**.

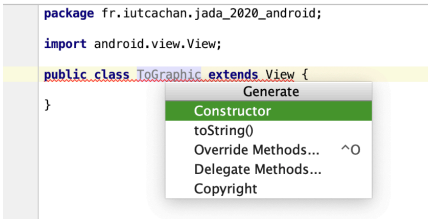
9. **Le plus simple pour avoir une nouvelle apparence d'une Activity est de changer de gestionnaire de disposition,** pour cela il suffit de suivre les étapes suivantes :
   - **Créer un nouveau gestionnaire** : faire un clic droit sur ***res/layout*** et ***new Layout Resource File***, nommé ici gestionaire2
   - Organiser le nouveau gestionnaire

- Pour l'afficher :
  Rajouter l'instruction suivante dans la méthode appelée par le click sur le bouton :
  setContentView(R.layout.gestionaire2);

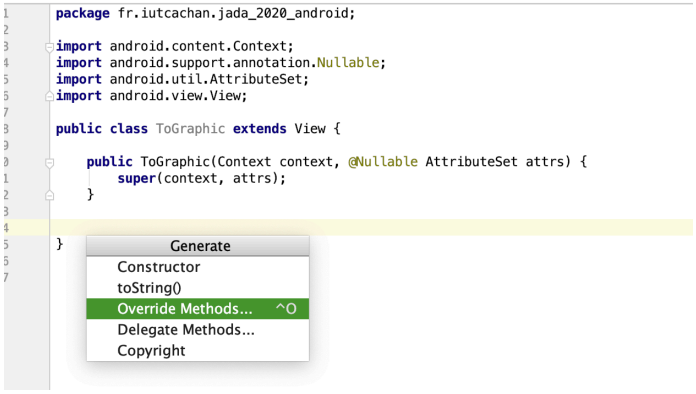
  Remarques :
  - Ceci permet de rester dans l'Activity et de conserver l'environnement de données de celle-ci.
  - Ce layout ne contient pas l'entête nécessaire pour servir de layout de lancement de l'Actity (autrement dut vous ne pouvez remplacer le layout activity_main.xml par celui-ci).

B. Représentations graphiques élémentaires

10. Pour faire quelques dessins on se propose maintenant de créer une nouvelle classe dans le répertoire Java (clic droit, new Class etc…).
    - Vous pouvez choisir librement son nom.
    - Faites hériter cette classe de la classe View en rajoutant ***extends View***
      Un truc : ne saisissez pas les imports, Android Studio les rajoutera automatiquement au fur et à mesure qu'il reconnaitra les classes que vous souhaitez utiliser.
    - Voici deux trucs Android Studio permettant d'accélérer la saisie :
      - **Un truc :** Utilisez le clic droit et cliquez sur *Generate*. Puis choisissez *Constructor*. Choisissez celui qui convient, c'est-à-dire **avec 2 arguments (context et attr)** (celui avec un argument est interdit).



    - **Un autre truc :** Utilisez à nouveau generate avec Override Methods, et saisissez onDraw



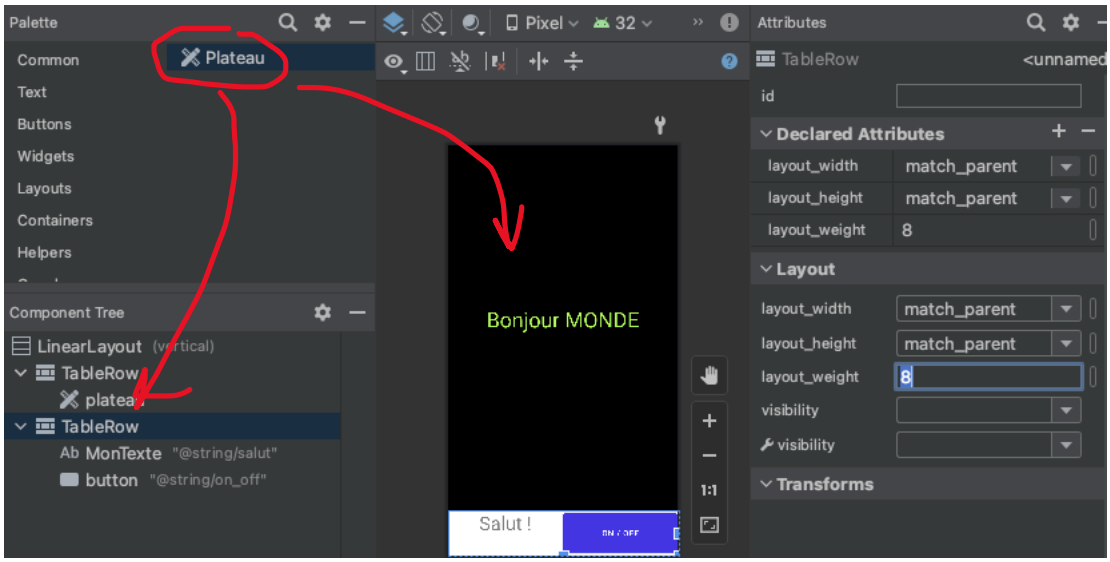
Voici le code à saisir dans la classe :

```
package com.example.myapplication;
```

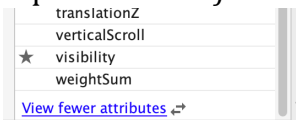
```
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.util.AttributeSet;
import android.view.View;

public class MonView extends View {

    public MonView(Context context, AttributeSet attrs) {
        super(context,attrs);
    }

@Override
public void onDraw (Canvas canvas)      {
        Paint p = new Paint();
        /*définir la couleur de l'objet de dessin */
        p.setColor(Color.BLACK);
        /*définir son style en remplissage*/
        p.setStyle(android.graphics.Paint.Style.FILL);
        /*dessiner un rectangle qui occupe la totalité du View*/
        canvas.drawRect(0,0,getWidth(),getHeight(), p);
        /*définir une autre couleur pour dessiner un texte*/
        p.setColor(Color.GREEN);
        /*définir la taille du texte*/
        p.setTextSize(100);
        /*définir le centre du texte comme étant son origine*/
        p.setTextAlign(android.graphics.Paint.Align.CENTER);
        /*dessiner le texte en positionnant son origine au centre
du
            View */
        String texte = "Bonjour MONDE";
        canvas.drawText(texte, getWidth()/2, getHeight()/2, p);
    }
}
```

Remarque : l'argument de la classe ***Context*** dans le constructeur vous donnera la possibilité d'accéder à plein d'éléments mis à disposition par Android studio en fonction des possibilités du téléphone.

10. Une fois le code saisi, lancer **l'exécution du programme.** Rien ne change mais maintenant **la classe devrait apparaît maintenant dans la palette de composants de la RAD sous *project*.**
    Si vous ne le trouvez pas, utilisez l'outil de recherche (loupe) de la PALETTE de composant. Si la loupe n'apparait pas faites glisser **la classe** dans l'arborescence des composants *Component tree* au bon endroit.
    - Rajouter sous le *LinearLayout* deux *TableRow* comme sur l'image ci-dessus.



- Du côté des attributs, s'il y en a trop, faire défiler la liste vers le bas et cliquer sur *View fewer attributs*



- Paramétrer l'ensemble avec des *matchparent* et des *weight* de 1 (sauf pour le deuxième TableRow, prendre *weight* à 8).
  - Le paramètre *weight* indique la proportion de l'espace **supplémentaire** de la mise en page qui doit être allouée au composant graphique. Pour information, 0 signifie donc que le composant est réduit strictement à la taille dont il a besoin en fonction des paramètres *height* et *width*.

  - *wrapcontent* est utilisé lorsque nous voulons que la vue n'occupe **que l'espace dont elle a besoin**.
- Lancer l'application
  Remarque : les dessins sont effectués sur un objet de la classe ***Canvas*** ! Pour connaître toutes les possibilités en termes de dessin 2D **consultez la documentation officielle** de *cette* classe (attention : API 23 !). Notez ci-dessous 3 de celles qui vous intéressent le plus.

-
-
-

12. Voici comment afficher simplement une portion d'image d'un fichier graphique.

- Placer le fichier ***nom.png*** dans le répertoire res/drawable (https://emojipedia.org/; click droit, enregistrez l'image sous, taille 50x50 si possible)
- Créer l'objet affichable par la commande suivante :
```
Bitmap b = BitmapFactory.decodeResource(getResources(), R.drawable.nom);
```
- Découpe un rectangle dans le bitmap positionné en 20,705 et de dimensions 50,65
```
b = Bitmap.createBitmap(b,20,705,50,65);
```
- Dans une méthode où l'objet canvas est disponible, comme onDraw par exemple, pour affiche cette image à la taille 200 x 200
```
canvas.drawBitmap(b, 200,200,paint);
```
- Lancer l'application

  Remarque : les classes Canvas et Bitmap possèdent de nombreuses méthodes pour modifier les images, voir la documentation pour en savoir plus.

13. Comment afficher simplement une information ?

    Toast.makeText(getContext(), "Il faut saisir une heure", Toast.LENGTH_SHORT).show();

14. On veut maintenant déplacer le texte à l'endroit où nous avons cliqué. Ecrire le code correspondant. N'oubliez pas de *forcer le rafraichissement de l'écran* chaque fois par la méthode ***invalidate*** ();
    - Rajouter les attributs suivants à la classe MonView et modifie

```
float xText, yText;

public MonView(Context context, AttributeSet attrs) {
    super(context, attrs);
    setXYText (600,600);
}


public void setXYText (float x, float y){
    xText = x;
    yText = y;
}
```

    - Modifier en conséquence la méthode onDraw du MonView

```
canvas.drawText(texte, xText, yText, p);
```

    - Ecrire complétez comme suit MonView

```
    @Override
    public boolean onTouchEvent(MotionEvent event){
        xText = event.getX();
        yText = event.getY();
        invalidate();
        return false;
    }
}
```

- Lancer l'application

C. Timer

14. Nous allons maintenant créer un timer parmis les attributs du View :
    - Importer la bonne classe :

```
15.     import android.os.Handler;
```

    - Rajouter le code suivant dans la classe :

```
Handler timerHandler = new Handler();

Runnable updateTimerThread = new Runnable() {
    public void run() {
        timerHandler.postDelayed(this,100);
        invalidate();     // appel de onDraw pour redessiner
    }
};
```

Et dans le constructeur:

```
timerHandler.postDelayed(updateTimerThread, 10);
```

La méthode **run** sera appelée toutes les 100 millisecondes.
Imaginez une action à réaliser (par exemple faire grandir la taille du texte à partir de 0) à chaque appel.

D. Utilisation de différents types d'évènements courants

15. On veut maintenant **déplacer le texte à l'endroit où nous avons cliqué.** Ecrire le code ci-dessous dans la classe MonView. N'oubliez pas de *forcer le rafraichissement de l'écran* chaque fois par la méthode ***invalidate*** ();
    - Rajouter les attributs suivants à la classe MonView et modifie

```
float xText, yText;

public MonView(Context context, AttributeSet attrs) {
    super(context, attrs);
    setXYText (600,600);
}


public void setXYText (float x, float y){
    xText = x;
    yText = y;
}
```

    - Modifier en conséquence la méthode onDraw du MonView

```
canvas.drawText(texte, xText, yText, p);
```

    - Pour pouvoir réagir à une action sur le pad il est recommandé de mettre en œuvre un OnTouchListener bien qu'on puisse aussi utiliser directement la méthode onTouchEvent vue plus haut. Complétez comme suit MonView

```
OnTouchListener onTouchListener = new OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
  return true;
    }
};

setOnTouchListener(onTouchListener);
```

- `return true` indique que l'évènement a été traité

```
        xText = event.getX();
        yText = event.getY();
        invalidate();
```

15. Utilisation du touchpad pour determiner un mouvement rajouter à l'intérieur de la méthode onTouch le code suivant :

    - Créer des attributs pour conserver les valeurs du point de contact et du point de relache.

```
float x1, x2, y1, y2;
```

    - Exemple de code de la méthode

```
@Override
public boolean onTouch(MotionEvent event){
    float dx, dy;
    String direction;
    switch(event.getAction()) {
        case(MotionEvent.ACTION_DOWN):
            x1 = event.getX();
            y1 = event.getY();
            Log.i("pacman", "appuyé");
            break;

        case(MotionEvent.ACTION_UP): {
            x2 = event.getX();
            y2 = event.getY();
            dx = x2-x1;
            dy = y2-y1;
            // Use dx and dy to determine the direction of the move
            if(Math.abs(dx) > Math.abs(dy)) {
                if(dx>0)
                    direction = "right";
                else
                    direction = "left";
            } else {
                if(dy>0)
                    direction = "down";
                else
                    direction = "up";
            }
            Log.i("pacman", "laché " + direction);
            Log.i("pacman", "dx = " + dx +"; dy = " + dy);
            break;
        }
    }
    invalidate();
    return true;
}
```

16. Utilisation de l'**accéléromètre** du téléphone.
    Puis rajouter dans la classe concernée le code ci-dessous.

Créer l'objet représentant l'accéléromètre dans le constructeur. Attention celui-ci doit fournir un objet de la classe Context.

```
/* accelerometre */
Sensor accelerometre;
SensorManager m = (SensorManager) context.getSystemService
(Context.SENSOR_SERVICE);
accelerometre = m.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

Et ajouter la méthode de réaction aux évènements sur l'accéléromètre :

```
        final SensorEventListener mSensorEventListener = new
SensorEventListener() {
            public void onAccuracyChanged(Sensor sensor, int accuracy) {
            }

            public void onSensorChanged(SensorEvent sensorEvent) {
                // Que faire en cas d'évènements sur le capteur ?
                terrain.pencheH = -(int)(sensorEvent.values[0]);
                terrain.pencheV = (int)(sensorEvent.values[1]);
                terrain.penche =
                terrain.pencheH*terrain.pencheH+terrain.pencheV*terrain.
                pencheV;
            }
  };
 m.registerListener(mSensorEventListener, accelerometre,
SensorManager.SENSOR_DELAY_UI);
```

E. Intégration d'un code « pur Java » préexistant (c'est-à-dire pour la console) à une UI Android

1. Copier/coller les classes de l'autre projet dans le même répertoire que la classe MainActivity
    - *Si jamais les classes ne sont pas dans le même package* :
        - il faut les y placer en le modifiant dans le code après le mot clé package,
        - ou bien, comme Android Studio le surligne en rouge, en plaçant le curseur dessus il doit proposer une ampoule rouge. *En cliquant sur l'ampoule* il vous propose *plusieurs options dont celle de la placer dans le même package* que le nom du répertoire.
2. Puisque le programme ne commence plus par public static void main (), il faut déplacer ce code dans la méthode *onCreate* de MainActivity puisque c'est le début du code.
    - A ce stade vous pouvez tenter d'exécuter l'application Android, mais bien sûr elle sera toujours vide
    - Vous pouvez tout de même écrire dans un label les contenus des objets.

Pour en savoir plus, suivre les nombreux et excellents tutoriels Android Studio à l'adresse :
https://classroom.udacity.com/courses/ud834