**Activity**  
Est une fenêtre pour les appareils Android.  
  
**XML**Est similaire au HTML, mais on choisit le nom de nos balises.  
  
-Si on ne veut pas hardcode du texte dans les widgets de notre layout, on va dans le fichier /values/strings.xml et on créer notre texte selon la syntaxe <string name=’’string\_name’’>My String</string>. On peut ensuite l’utiliser dans le champ texte de notre widget en utilisant la syntaxe @string/string\_name.   
  
-Dans /values/themes/themes.xml on peut changer notre thème ainsi que les couleurs de notre thème.   
  
**Fichiers importants**-/manifests/AndroidManifest.xml Fichier de publication de l’application, signature de l’application  
-/java/my.package.project/MainActivity Fichier java qui contient la logique de l’application. Contient l’équivalent de notre classe main sous la forme de onCreate();.   
-/java/my.package.project(android test)/ExempleInstrumentedTest et /java/my.package.project(test)/ExempleUnitTest Permet de faire des tests junit.   
-/drawable/ C’est ici que l’on va mettre toutes nos ressources visuelles pour pouvoir les utiliser dans notre code plus tard.   
-/layout/activity\_main.xml C’est le fichier où on dessine l’application. On peut le faire visuellement ou bien avec du code.   
-/mipmap/ Dossier qui contient les icônes qui représentent notre application.   
-/values/colors.xml Contient les couleurs de notre thème. Les couleurs sont définies en hexadécimal avec le standard RGBA mais le canal alpha est représenté par la première valeur.   
-/values/strings/ Contient les différentes langues et traduction pour notre application. Endroit où on doit écrire le texte statique pour les différentes langues supportées par notre application.   
-/values/themes/ Contient un peu l’équivalent du css pour android/java.   
-/Gradle Scripts/build.gradle (Modele : project) Contient les différentes dépendances et version nécessaire pour le fonctionnement de notre projet. (Si jamais la version de material n’est pas 1.8.0, la mettre à ça et faire sync now en haut à droite)  
  
**Android Studio**Pour créer une deuxième activité dans notre projet, il faut faire un clique droit sur le package contenant notre main activity New🡪Activity🡪TypeOfActivity.   
***Layout***On utilise les fichiers xml qui sont dans /res/layout/ pour modifier le layout de notre application. Le layout par défaut est un constraint layout. Lorsque nos widgets sont présents dans un constraint layout, on le perçois par les petites boules autour de nos widgets, il faut au moins en connecter 2 aux bordures de notre écran, sinon à d’autres widgets.  
Pour avoir accès à un widget spécifique dans notre code et à ses méthodes, il nous faut vérifier qu’elle est son ID, chose que l’on on peut changer dans les détails de notre widget.   
Dans la bar au-dessus de notre layout, il est possible de changer le téléphone qui est utilisé comme étant un preview.   
Dans les attributs de nos éléments, on peut changer la largeur ainsi que la hauteur de nos éléments avec les options layout\_width et layout\_height. Ils existent deux options par défaut soit match\_parent et wrap\_content. On peut aussi mettre nos propres valeurs en mettant dp suite à un chiffre tel que 48dp. Dp signifie density independant pixel.   
Dans les attributs de nos éléments, on peut aussi changer la taille de notre texte, cette valeur est en sp tel que 18sp. Sp signifie scale independent pixel.   
Dans les attributs des éléments qui sont normalement modifiable par l’utilisateur, il existe un champ nommé editable que l’on peut mettre à false si on ne veut pas que l’utilisateur puisse le changer.   
-Common/TextView contient du texte qui ne pourra pas changer pendant que l’application tourne.   
  
-Text/Numbers contient un champ dans lequel l’utilisateur pourra rentrer de l’information. Le nom de la classe dans le code est EditText de type Number.   
  
-Text/Password contient un champ dans lequel l’utilisateur pourra rentrer son mot de passe. Le nom de la classe dans le code est EditText de type Password.   
  
-Buttons/Switch est un bouton à deux états avec lequel l’utilisateur peut interagir. Le nom de la classe dans le code est Switch.   
  
-Common/Button est un bouton sur lequel l’utilisateur peut cliquer. Le nom de la classe dans le code est Button.   
  
***Gestion des évènements***Il existe 3 grands concepts :  
-La source : La composante avec laquelle l’usager interagit. Elle lance un objet évènementiel lorsqu’un évènement survient. Elle comprend des méthodes afin de s’inscrire à un écouteur (Ex : setOnClickListener).   
-Objet évènementiel (Event) : Il renferme des informations sur l’évènement qui vient de survenir (Ex : l’endroit, la source, etc…). En java du moins, les noms des évènements finissent toujours en Event (Ex : TouchEvent, MenuEvent, etc…).   
-Écouteur(Listener) : Un objet (pas une composante) provenant d’une classe qui implémente une ou plusieurs interfaces-écouteurs. La réaction à l’évènement sera le contenu d’une méthode d’une interface-écouteur. Le même écouteur peut écouter plusieurs sources et plusieurs types d’évènements. (Exemple listeners : OnClickListener, OnLongClickListener, OnDragListener, OnTouchListener, OnHoverListener, OnKeyListener, OnAttachStateChangeListener, OnLayoutChangeListener, OnCreateContextMenuListener, OnFocusChangerListener, OnGenericMotionListener, OnSystemUiVisibilityChangeListener, etc…)  
  
Donc quand on clique sur une source, un objet évènementiel est lancé et cet objet sera capté par un écouteur qui exécutera une action.   
  
Il existe 3 étapes pour gérer un évènement :  
1. Créer un objet écouteur. Cela doit être fait dans la fonction onCreate. (La variable peut être déclaré avant la fonction, mais elle doit être initialisé dans la fonction)  
2. Il faut inscrire la/les source(s) à notre écouteur.   
3. Coder une classe interne de l’écouteur en y codant les méthodes de l’interface-écouteur qu’elle met en œuvre.   
  
**Code**  
-@Overrideprotected void onCreate(Bundle savedInstanceState{super.onCreate(savedInstanceState); setContentView(R.layout.activity\_main);} Est l’équivalent de notre main pour une application Android. Est la première fonction qui sera exécuté lors du démarrage de l’application.   
  
-findViewById(anIntId) nous retourne un élément de notre layout selon le Id donnée.   
  
-R.id.IdOfWidget nous retourne un int selon le Id qu’on avait donnée à notre widget. Souvent utilisé avec la fonction findViewById().   
  
-AWidget.setOnClickListener(AListener) nous permet d’inscrire une source à un listener/écouteur.   
  
-implements View.OnClickListener nous permet d’implémenter l’interface du listener pour les actions on click. Sa méthode est public void onClick(View v){}, celle-ci doit être override. Le v représente la source dans notre layout qui a appelé la fonction.  
  
-AWidgetThatContainsText.setText(AString) nous permet de changer le texte de notre widget.  
  
-AWidget.getText().toString() nous retourne le string présent dans notre widget. On doit faire toString suite au getText car sinon on obtient un objet Editable.   
  
***Vector***  
-Vector<AType> nameOfVector = new Vector();   
Un vecteur est un tableau qui a une allocation dynamique de la mémoire. On peut donc rajouter des valeurs au fur et à mesure du programme dans notre vecteur. On peut limiter les types qui seront présent dans notre vecteur en spécifiant le type voulu à l’intérieur des <>. Un vecteur ne peut pas avoir de champ vide, on ne peut donc pas placer un élément à la position 10 si je n’ai que 3 éléments dans mon vecteur. On n’utilise pas de [] avec un vecteur. On peut uniquement mettre des objets dans un vecteur, les types primitifs sont converties en objet.   
-nameOfVector.add(AValue) Nous permet d’ajouter un élément à notre vecteur à la dernière position possible. Si j’ai 2 éléments, le nouvel élément sera placer à la position 2.   
-nameOfVector.insertElementAt(NewValue, int index) nous permet de placer une valeur à l’index voulu. Il faut placer l’objet à une position qui ne créera pas d’espace vide dans le vecteur. Si j’ai déjà un élément à la position 1 et que je place le nouvel élément à la position 1, l’ancienne valeur sera déplacé à la position 2 et ainsi de suite pour les autres valeurs aux indexes plus grand.   
-nameOfVector.contains(AValue) Nous permet de vérifier si un vecteur contient la valeur mis en paramètre.   
-nameOfVector.get(int index) Nous retourne la valeur à l’index indiqué.  
-nameOfVector.set(int index, AValue) Remplace la valeur à l’index indiqué.