# Layouts

On les appelle en anglais des layouts et en français des gabarits. Un layout est donc une vue spéciale qui peut contenir d'autres vues et qui n'est pas destinée à fournir du contenu ou des contrôles à l'utilisateur. Les layouts se contentent de disposer les vues d'une certaine façon. Les vues contenues sont les enfants, la vue englobante est le parent, comme en XML. Une vue qui ne peut pas en englober d'autres est appelée un widget (composant, en français).

# Inflate et setContentView

En fait, à chaque fois qu'on récupère un objet depuis un fichier XML dans notre code Java, on procède à une opération qui s'appelle la désérialisation. Concrètement, la désérialisation, c'est transformer un objet qui n'est pas décrit en Java − dans notre cas l'objet est décrit en XML − en un objet Java réel et concret. C'est à cela que sert la fonction View findViewById (int id). Le problème est que cette méthode va aller chercher dans un arbre de vues, qui est créé automatiquement par l'activité. Or, cet arbre ne sera créé qu'après le setContentView ! Donc le findViewById retournera null puisque l'arbre n'existera pas et l'objet ne sera donc pas dans l'arbre. On va à la place utiliser la méthode static View inflate (Context context, int id, ViewGroup parent). Cette méthode va désérialiser l'arbre XML au lieu de l'arbre de vues qui sera créé par l'activité.

On ne peut pas utiliser des setContentView à chaque fois car à la compilation, si on détecte qu'il y a deux setContentView dans onCreate, eh bien on ne prendra en compte que la dernière ! Ainsi, toutes les instances de setContentView précédant la dernière sont rendues caduques.

# Unité de taille et autre

Unité de taille : dp (density pixel)

Unité de police : sp (dp avec en plus un respect des prefs de l’utilisateur)

match\_parent : signifie qu'elle prendra autant de place que son parent sur l'axe concerné ;

wrap\_content : signifie qu'elle prendra le moins de place possible sur l'axe concerné. Par exemple si votre vue affiche une image, elle prendra à peine la taille de l'image, si elle affiche un texte, elle prendra juste la taille suffisante pour écrire le texte ;

# Les 3 façons de déclarer un listener

## Héritage (ou interfacage plutôt) : c’est la vue qui implémente le listener

public class Main extends Activity implements View.OnTouchListener, View.OnClickListener {

## Classe anonyme

widget.setTouchListener(new View.OnTouchListener() {

/\*\*

\* Contenu de ma classe

\* Comme on implémente une interface, il y aura des méthodes à implémenter, dans ce cas-ci

\* « public boolean onTouch(View v, MotionEvent event) »

\*/

});

## Attribut

private OnClickListener clickListenerBoutons = new View.OnClickListener() {

@Override

public void onClick(View v) {

/\* Réagir au clic pour les boutons 1 et 2\*/

}

};

# Trucs interesants que j’ai repéré

tools:showIn="@layout/activity\_main" dans le XML pour dire que le layout appartient à une activité donnée qu’on écrit <include layout="@layout/content\_main"/> dans le fichier XML qui a le nom de l’activité. Ce que je sais c’est qu’il permet d’afficher le bon design dans android studio quand on selectionne le fichier XML car android studio va checker le XML parent.

Pas oublier les Toast (messages ponctuels), le principe du TextWatcher