# Annexe 8 – Animations

*Threads ( semaine dernière ) : On a vu qu’on pouvait animer des ImageViews avec un Thread :*

* Soit un Thread qui roule sur son propre fil d’exécution et on utilise un Handler pour communiquer avec le thread principal ( UIThread ). Une boucle avec une condition de fin de boucle en boolean fait en sorte que l’animation se poursuit.
* Soit en travaillant directement avec le Handler et un Thread anonyme roulant sur le thread principal ( UIThread ) en utilisant postDelayed dans la méthode run pour que l’animation continue.

Principes derrière les animations « utiles » dans une interface graphique :

* Informative : permet de montrer les liens entre les informations 🡪 glisser « swipe » pour voir plus d’informations
* Attractive : une animation peut permettre d’attirer l’attention sur ce qui est important 🡪 animation sur un bouton pour indiquer qu’on doit le cliquer.
* Originale : donne du style à une application si revient à plusieurs reprises

1. ObjectAnimator ( 1h )

* Permet d’animer certaines propriétés d’un objet graphique
* Des méthodes de la classe permettent de définir la ou les propriétés qui seront soumises à l’animation
* On peut établir ces animations en XML ou en Java/Kotlin
* La méthode invalidate est automatiquement appelée afin que l’apparence visuelle soit actualisée
* Roule sur le thread principal ( celui qui s’occupe des éléments graphiques, entre autres )

Exemples :

ObjectAnimator anim = ObjectAnimator.ofFloat(auto, View.X ou ‘’x’’, 200, 500);

anim.start();

ObjectAnimator anim2 = ObjectAnimator.ofFloat(objet, “scaleY”, 3);

anim2.start();

Propriétés pouvant être utilisées :

[Property Animation Overview  |  Android Developers](https://developer.android.com/develop/ui/views/animations/prop-animation#views)

entres autres…

* View. X : la position finale par rapport au coin supérieur gauche de l’écran
* View.TRANSLATION\_X : en fonction de la position au départ ( nonobstant d’autres translations qui auraient eu lieu )
* Toute autre propriété qui a une méthode set codée ( backgroundColor par exemple, la passer en String )

Exercices à l’ordi :

* durée
* interpolations
* animation suivant un Path ( rappel tp Paint C44 )
* AnimatorSet : C’est un ensemble d’animations qu’on peut exécuter en séquence (playSequentially) ou en même temps (playTogether)

\*\*\* Les unités sont en pixels. Si besoin est vous pouvez faire les équivalences en dp à l’aide des formules suivantes :

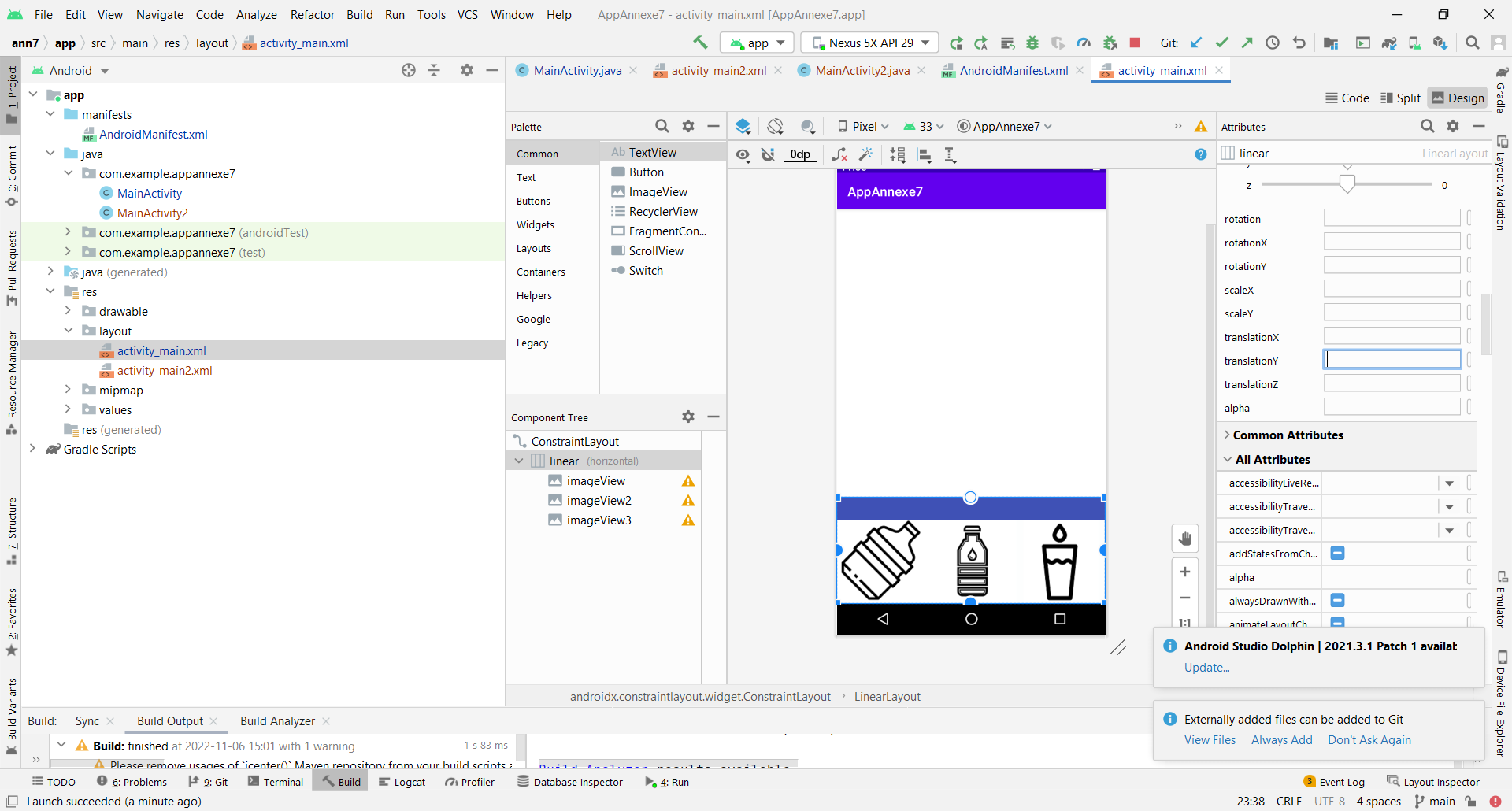
**public static int** pxToDp(**int** px) {  
 **return** (**int**) (px / Resources.*getSystem*().getDisplayMetrics().**density**);

}  
  
**public static int** dpToPx(**int** dp) {  
 **return** (**int**) (dp \* Resources.*getSystem*().getDisplayMetrics().**density**);  
}

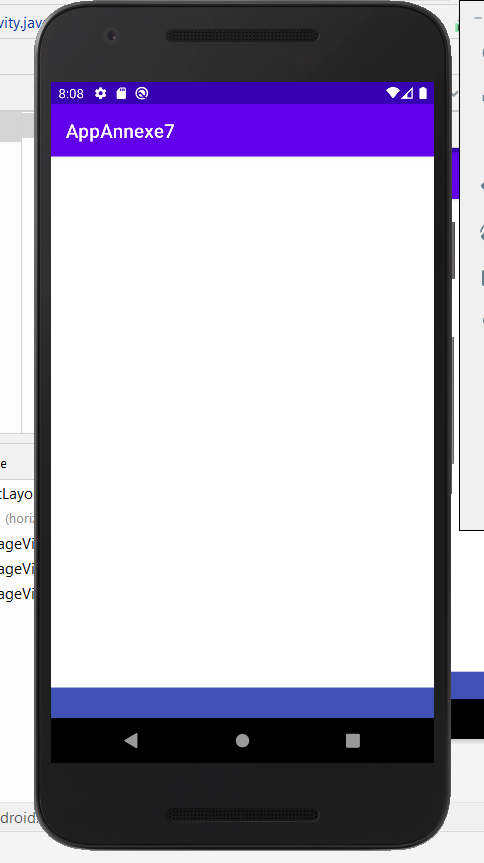
Resources.*getSystem*().getDisplayMetrics().**widthPixels // pour connaître la largeur de l’écran en pixels**

Exercice 1 ( 30 min )

Dessinez un conteneur qui apparaitra lorsqu’on le cliquera :



1. Dans une nouvelle activité ( Empty Views Activity ) , dans le fichier .xml de positionnement, utilisez un LinearLayout que vous accrocherez au ConstraintLayout d’origine et placez-y ces trois images connues.
2. Donnez un fond blanc aux ImageViews et un fond bleu au LinearLayout. Donnez un padding de manière à ce qu’on voie une bande bleue en haut des images.
3. Dans l’onglet design, utilisez la propriété translationY sur le LinearLayout de manière à cacher tout son contenu sauf la bande bleue.



D) Utilisez un ObjectAnimator avec une propriété adéquate de manière à ce que le LinearLayout apparaisse progressivement jusqu’à s’afficher au complet quand on clique sur la bande bleue :



E) Trouvez une méthode dans la classe ObjectAnimator afin de rabaisser le menu une fois ouvert

F) Codez votre gestion d’événements en utilisant une expression lambda. Faites en sorte que le LinearLayout apparaisse lorsqu’il est caché et qu’il disparaisse lorsqu’il est déjà visible. ( en alternance )