Les animations

Sommaire

1

Animation par le code

Animation par le code

Les animations permettent :

- D’effectuer une rotation
- De modifier la position
- De modifier la taille
- De modifier l’opacité

Android se chargera de définir les étapes intermédiaires par interpolation en fonction du temps et des états d’origine et de fin définis

Animation par le code

Les animations sont souvent utilisées pour donner plus de feedback à l'utilisateur.

Ex : transition entre activités, mauvaise saisie de l'utilisateur, ...

Les animations possibles sous Android :

- **Alpha** (opacité) : modifie la transparence de la vue
- **Scale** (échelle) : permet de spécifier l'agrandissement/réduction sur les axes X et Y d'une vue
- **Translation** : permet de spécifier la translation / le déplacement d'une vue
- **Rotation** : permet d'effectuer une rotation selon un angle en degré et un point de pivot

Pour démarrer une animation sur une vue, il faut utiliser la méthode startAnimation

Animation d’alpha

Paramètres constructeur :

- Valeur de départ de l’opacité
- Valeur de fin de l’opacité

Propriété :

- Duration (durée de l’animation en millisecondes)

```
val alphaAnimation = AlphaAnimation(
    0f,
    1f
).apply {  this: AlphaAnimation
    duration = 2000
}

binding?.topImageView?.startAnimation(alphaAnimation)
```

Animation d’échelle

Paramètres constructeur :

- Agrandissement axe des x départ
- Agrandissement axe des x fin
- Agrandissement axe des y départ
- Agrandissement axe des y fin

Propriété :

- Duration (durée de l’animation en millisecondes)

```
val scaleAnimation = TranslateAnimation(
        fromXDelta: 1f,
        toXDelta: 2f,
        fromYDelta: 1f,
        toYDelta: 2f).apply {  this: TranslateAnimation
    duration = 2000
}
binding?.imageView?.startAnimation(scaleAnimation)
```

Animation de translation

Paramètres constructeur :

- Déplacement des x départ
- Déplacement des x fin
- Déplacement des y départ
- Déplacement des y fin

Propriété :

- Duration (durée de l'animation en millisecondes)

```
val translateAnimation = TranslateAnimation(
        fromXDelta: 0f,
        toXDelta: -150f,
        fromYDelta: 0f,
        toYDelta: 150f).apply {  this: TranslateAnimation
    duration = 2000
}
binding?.imageView?.startAnimation(translateAnimation)
```

Animation de rotation

Paramètres constructeur :

- Angle départ
- Angle fin
- Pivot x
- Pivot y

Propriété :

- Duration (durée de l'animation en millisecondes)

```
val pivotX = binding?.imageView?.pivotX ?: 0
val pivotY = binding?.imageView?.pivotY ?: 0
val rotateAnimation = RotateAnimation(
        fromDegrees: 0.0f,
        toDegrees: 180.0f,
        pivotX.toFloat(),
        pivotY.toFloat()).apply {  this: RotateAnimation
    duration = 2000
}
binding?.imageView?.startAnimation(rotateAnimation)
```

Animation de rotation

Astuces :

- Conserver l'état de la vue à la fin de l'animation en précisant true avec la propriété **fillAfter**
- Pour répéter une animation, vous pouvez utiliser la propriété **repeatCount** (entier ou INFINITE)
- Il est possible aussi de définir le mode de répétition d'une animation grâce à la méthode setRepeatMode :
- Animation.REVERSE : l'animation est jouée à l'envers à la fin de la première itération
- Animation.RESTART : l'animation redémarre

```
val rotateAnimation = RotateAnimation(
        fromDegrees: 0.0f,
        toDegrees: 360.0f,
        pivotX.toFloat(),
        pivotY.toFloat()).apply {  this:

    duration = 2000
    fillAfter = true
    repeatCount = Animation.INFINITE
}
```

2

Animation par fichiers XML

Animation par fichiers XML

Avantage animation fichier XML :

- Externalisé et ré-utilisable
- Peut-être défini par une autre personne que le développeur

Possibilité de définir les animations suivantes :

- ScaleAnimation
- AlphaAnimation
- TranslateAnimation
- RotateAnimation

Animation par fichiers XML

Les fichiers d'animation déclarés en XML doivent se trouver dans le dossier res/anim

Pour créer un nouveau fichier XML d'animation :

- New > Android XML File.
- Resource Type : Animation
- Root Element : ex: scale

```xml
<?xml version="1.0" encoding="utf-8"?>
<scale
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/accelerate_interpolator"
    android:fromXScale="1.0"
    android:toXScale="2.0"
    android:fromYScale="1.0"
    android:toYScale="2.0"
    android:duration="1000" />
```

Animation par fichiers XML

Pour charger une animation depuis un fichier XML il faut utiliser la méthode loadAnimation de la classe AnimationUtils

```
val scaleAnimation = AnimationUtils.loadAnimation(
        context: this,
       R.anim.scale_animation)

binding?.imageView?.startAnimation(scaleAnimation)
```

Animation par fichiers XML

Paramètres animations (**alpha**) :

- fromAlpha : valeur opacité de départ (entre 0.0 et 1.0)
- toAlpha : valeur opacité de fin (entre 0.0 et 1.0)

Animation par fichiers XML

Paramètres animations (**scale**) :

- fromXScale : float
- toXScale : float
- fromYScale : float
- toYScale : float
- pivotX, pivotY = pourcentage pour le point de pivot (entre 0 et 100)

Animation par fichiers XML

Paramètres animations (**translate**) :

- fromX : float
- toX : float

Animation par fichiers XML

Paramètres animations (**rotate**) :

- fromDegrees : orientation de départ en degrés (entre 0 et 360)
- toDegrees : orientation de fin en degrés (entre 0 et 360)
- pivotX, pivotY : pourcentage pour le point de pivot (entre 0 et 100)

3

Evénements liés aux animations

Evénements liés aux animations

Les animations sous Android possèdent un listener (AnimationListener) permettant de gérer les événements du cycle de vie d'une animation.

Grâce à ces événements il va être possible d'exécuter des actions :

- Avant le démarrage de l'animation
- À la fin de l'exécution de l'animation
- Entre chaque répétition de l'animation

Evénements liés aux animations

Méthodes interface AnimationListener :

- onAnimationStart : notifie le début de l'animation
- onAnimationEnd : notifie la fin de l'animation
- onAnimationRepeat : notifie la répétition de l'animation

```
val scaleAnimation = AnimationUtils.loadAnimation(
        context: this,
       R.anim.scale_animation)

scaleAnimation.setAnimationListener(object: Animation.AnimationListener {
    override fun onAnimationStart(animation: Animation?) {
        TODO( reason: "Not yet implemented")
    }

    override fun onAnimationEnd(animation: Animation?) {
        TODO( reason: "Not yet implemented")
    }

    override fun onAnimationRepeat(animation: Animation?) {
        TODO( reason: "Not yet implemented")
    }

})
```

4

Gérer une série d’animations

Gérer une série d’animations

Possibilité de combiner plusieurs animations grâce à la classe AnimationSet (attribut set XML)

Les animations peuvent être exécutées simultanément ou décalées dans le temps

Gérer une série d’animations

Propriétés communes aux animations pour l’exécution planifiée :

- duration : durée (milli-secondes)
- fillAfter : applique la transformation après l’animation
- fillBefore : applique la transformation avant l’animation
- startOffset : délai en milli-secondes pour démarrer l’animation

Gérer une série d’animations

Propriétés communes aux animations pour l’exécution planifiée :

- repeatCount : nombre de fois où l’animation devra être jouée
- repeatMode : mode de répétition
- interpolator : interpolator utilisé pour gérer l’animation

Gérer une série d'animations

Remarque :

Le paramètre constructeur **shareInterpolator** permet de partager le même interpolateur entre les animation de l'animationSet

```
val alphaAnimation = AlphaAnimation(
        1.0f,
        0.0f).apply {  this: AlphaAnimation
    duration = 1000
}

val translateAnimation = TranslateAnimation(
        fromXDelta: 0.0f,
        toXDelta: 100.0f,
        fromYDelta: 0.0f,
        toYDelta: 0.0f).apply {  this: TranslateAnimation
    duration = 900
    startOffset = 100
}

val animationSet = AnimationSet( shareInterpolator: true)
animationSet.addAnimation(alphaAnimation)
animationSet.addAnimation(translateAnimation)

binding?.imageView?.startAnimation(animationSet)
```

Gérer une série d’animations

Il est aussi possible de définir un animationSet par fichier XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<set
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/accelerate_interpolator">

    <alpha
        android:fromAlpha="1.0"
        android:toAlpha="0.0"
        android:duration="1000"/>

    <translate
        android:fromXDelta="0.0"
        android:toXDelta="10%p"
        android:startOffset="100"
        android:duration="900" />
</set>
```

Gérer une série d'animations

Possibilité d'appliquer une animation à une hiérarchie de vue complexe (ex : Layout)

Grâce à la classe **LayoutAnimationController**

```
val alphaAnimation = AlphaAnimation(
        1.0f,
        0.0f).apply {  this: AlphaAnimation
    duration = 1000
}

val layoutAnimationController = LayoutAnimationController(
        alphaAnimation,
         delay: 0.25f)

binding?.root?.layoutAnimation = layoutAnimationController
binding?.root?.startLayoutAnimation()
```

Gérer une série d'animations

Possibilité aussi avec un animationSet

Toujours grâce à la classe **LayoutAnimationController**

```
val animationSet = AnimationUtils.loadAnimation(
        context: this,
        R.anim.multi_animation
)

val layoutAnimationController = LayoutAnimationController(
        animationSet,
        delay: 0.25f)

binding?.root?.layoutAnimation = layoutAnimationController
binding?.root?.startLayoutAnimation()
```

5

Les interpolateurs

Les interpolateurs

L’interpolateur permet de spécifier une accélération ou une décélération.

L’interpolateur va servir à calculer la position de l’objet en fonction du temps et de la durée de l’animation.

Par défaut interpolateur linéaire (positionnement de la vue proportionnelle au temps d’exécution de l’animation).

Les interpolateurs

AccelerateDecelerateInterpolator :

- Permet à une animation de démarrer et de s'arrêter lentement avec une accélération entre ces deux phases.

AccelerateInterpolator :

- Permet à une animation de démarrer lentement et d'accélérer jusqu'à la fin.

AnticipateInterpolator :

- Permet à une animation de démarrer à l'envers puis d'accélérer jusqu'à la fin.

Les interpolateurs

AnticipateOvershootInterpolator :

- Permet à l'animation de démarrer à l'envers puis de revenir à l'endroit sans passer par la valeur cible et en allant directement à la valeur finale

BounceInterpolator :

- L'animation se terminera avec des "sauts" (comme si l'objet rebondissait)

CycleInterpolator :

- Répète l'animation le nombre de fois spécifié selon un modèle sinusoïdal

```
val animation = AnimationUtils.loadAnimation(
        context: this,
        R.anim.translate_animation
)

animation.interpolator = BounceInterpolator()

binding?.imageView?.startAnimation(animation)
```

Les interpolateurs

DecelerateInterpolator :

- Permet à l'animation de démarrer rapidement puis de décélérer jusqu'à la fin

OvershootInterpolator :

- L'animation dépasse la valeur cible avant de revenir à la valeur finale

LinearInterpolator :

- Animation homogène dans l'espace et le temps

6

Les animations par propriété

Les animations par propriété

Les animations par propriétés ont été introduites à partir d'Honeycomb (v3.0+ API v11).

Les animations par propriétés permettent d'animer n'importe quels objets.

Changement d'une propriété au cours du temps.

Les animations par propriété

Le système d'animation par propriétés permet de définir les caractéristiques suivantes d'une animation :

- Durée (par défaut 300 ms)
- Interpolateur de temps
- Compteur de répétition et comportement : possibilité de jouer une animation à l'envers puis la rejouer à l'endroit à l'infini
- Ensemble d'animations : possibilité de regrouper des animations et de les jouer ensemble
- Retard de rafraîchissement

Les animations par propriété

Classe ValueAnimator assure le calcul des valeurs d'une animation et les applique sur l'objet cible.

- Possibilité de récupérer la valeur en cours.
- Encapsule un objet TimeInterpolator qui modifie la fréquence de changement des valeurs (ex : Accélération/Décélération) et un objet TypeEvaluator qui permet de traiter des valeurs non reconnues par le système (ex : ArgbEvaluator)
- Pour démarrer une animation il faut appeler la méthode start sur une instance de ValueAnimator.

Les animations par propriété

Différences avec les animations de View (ancienne API) :

- Possibilité d'animer autre chose que des vues
- De plus grandes possibilités d'animation (ex : background color, ...), n'importe quelle propriété est animable.
- La propriété de l'objet est réellement modifiée

Les animations par propriété

Classe ObjectAnimator

- Permet d'animer plus facilement une propriété d'un objet cible.
- Animation liée à un accesseur dans l'objet cible

```
val animator =
    ObjectAnimator.ofFloat(
            imageView,
            propertyName: "y",
            ...values: 0f, 200f
    ).apply {  this: ObjectAnimator!
        duration = 2000
    }

animator.start()
```

Les animations par propriété

Utilisation d'un évaluateur

Ex: pour effectuer une animation de couleur

```
val animator = ObjectAnimator.ofObject(
        imageView,
        propertyName: "backgroundColor",
        ArgbEvaluator(),
        Color.BLUE, Color.RED)

animator.interpolator = AccelerateInterpolator()
animator.repeatCount = INFINITE
animator.repeatMode = REVERSE
animator.duration = 2000
animator.start()
```

Les animations par propriété

Utilisation d’un évaluateur

Autre exemple, animer la valeur text d’une TextView avec un effet de compteur

```
val animator = ofInt( ...values: 0, 6000)
animator.duration = 2000
animator.addUpdateListener { animation ->
    textView.text = animation.animatedValue.toString()
}
animator.start()
```

Les animations par propriété

Classe AnimatorSet

- Permet de regrouper des animations et de les jouer ensemble
- Possibilité de chorégraphier de manière fine les animations
- Méthodes disponibles : play, playSequentially, playTogether

Les animations par propriété

AnimatorSet.Builder

- Classe utilitaire permettant de chorégraphier plus facilement les animations entre elles.
- Créé depuis la méthode play sur une instance d'Animator.
- Méthode after, before, with.

Les animations par propriété

AnimatorSet.Builder

Utilisation de **playTogether**(anim1, anim2, anim3)
ou
play(anim1).**with**(anim2).with(anim3)

```
val scaleXAnimator = ObjectAnimator.ofFloat(
        imageView,
        propertyName: "scaleX",
        ...values: 1f, 2f
).apply {  this: ObjectAnimator!
    duration = 1000
}

val scaleYAnimator = ObjectAnimator.ofFloat(
        imageView,
        propertyName: "scaleY",
        ...values: 1f, 2f
).apply {  this: ObjectAnimator!
    duration = 1000
}

val animatorSet = AnimatorSet()
animatorSet.play(scaleXAnimator).with(scaleYAnimator)
animatorSet.start()
```

Les animations par propriété

Deux listeners disponibles

AnimatorListener :

- Permet de connaître le cycle de vie d'une animation.
- Méthodes : onAnimationStart, onAnimationRepeat, onAnimationEnd, onAnimationCancel
- Pour s'enregistrer : méthode addListener sur une instance de ValueAnimator, ObjectAnimator.

AnimatorUpdateListener :

- Permet d'être notifié pour chaque modification de frame.
- Méthode : onAnimationUpdate
- Pour s'enregistrer : méthode addUpdateListener

Les animations par propriété

Possibilité d'appliquer des transformations géométriques plus complexes :

- Translation : translationX, translationY
- Rotation : rotation, rotationX, rotationY
- Scale : scaleX, scaleY
- Pivot : pivotX, pivotY (par défaut au centre de la vue)
- Position : x, y
- Alpha : 0 = transparent -> 1 = opaque.

```
val rotationAnimator = ObjectAnimator.ofFloat(
        imageView,
        propertyName: "rotationY",
        ...values: 0f, 360f)

rotationAnimator.start()
```

Les animations par propriété

Possibilité de créer des animations combinant plusieurs propriétés grâce aux:

- **PropertyValuesHolder**
- Méthode **ofPropertyValuesHolder** de la classe ObjectAnimator.

```
val valueYChange = PropertyValuesHolder.ofFloat( propertyName: "y",  ...values: 100f)
val opacityChange = PropertyValuesHolder.ofFloat( propertyName: "alpha",  ...values: 0f)
val disappearAnimation = ObjectAnimator.ofPropertyValuesHolder(imageView, valueYChange,opacityChange)
disappearAnimation.duration = 600
disappearAnimation.start()
```

Les animations par propriété

Les interpolateurs disponibles :

- AccelerateDecelerateInterpolator
- AnticipateInterpolator
- AnticipateOvershootInterpolator
- BounceInterpolator
- CycleInterpolator
- DecelerateInterpolator
- LinearInterpolator
- OvershootInterpolator

Les animations par propriété

Exemple d'utilisation d'un interpolateur

```
val valueYChange = PropertyValuesHolder.ofFloat( propertyName: "y",  ...values: 100f)
val opacityChange = PropertyValuesHolder.ofFloat( propertyName: "alpha",  ...values: 0f)
val disappearAnimation = ObjectAnimator.ofPropertyValuesHolder(imageView, valueYChange,opacityChange)
disappearAnimation.interpolator = OvershootInterpolator()
disappearAnimation.duration = 600
disappearAnimation.start()
```

7

Spring animation

Spring animation

Disponible via la library : androidx.dynamicanimation

- Compatible jusqu'à Android 16 (4.1 - Jelly Bean)
- Deux classes :
  - SpringAnimation
  - SpringForce

```
implementation 'androidx.dynamicanimation:dynamicanimation:1.0.0'
```

Spring animation

Création d'une animation

```
val springAnimation = SpringAnimation(imageView, DynamicAnimation.TRANSLATION_Y,  finalPosition: 500f)
springAnimation.start()
```

Spring animation

Les différentes propriétés animables :

- ALPHA
- TRANSLATION_X, TRANSLATION_Y, TRANSLATION_Z
- ROTATION, ROTATION_X, ROTATION_Y
- SCROLL_X, SCROLL_Y
- SCALE_X, SCALE_Y

Spring animation

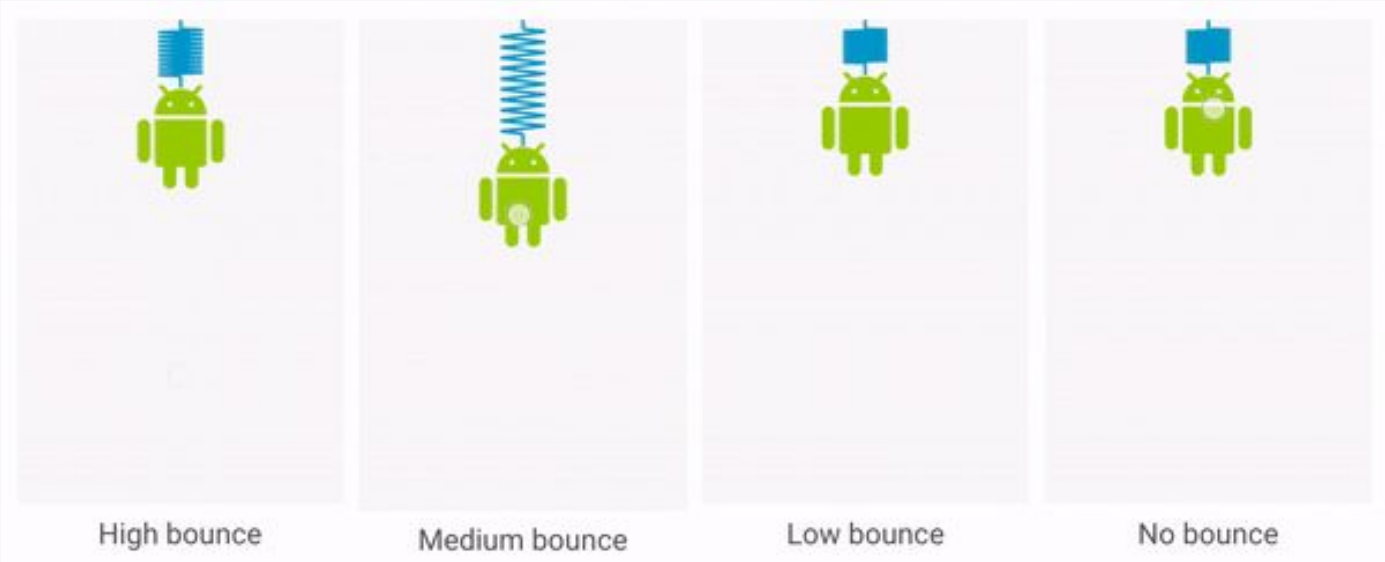
Paramètre supplémentaire via la classe **SpringForce**.

Deux propriétés :

- Damping (Amortissement) va influencer le “rebond”
- Stiffness (Raideur) va influencer la “vitesse”

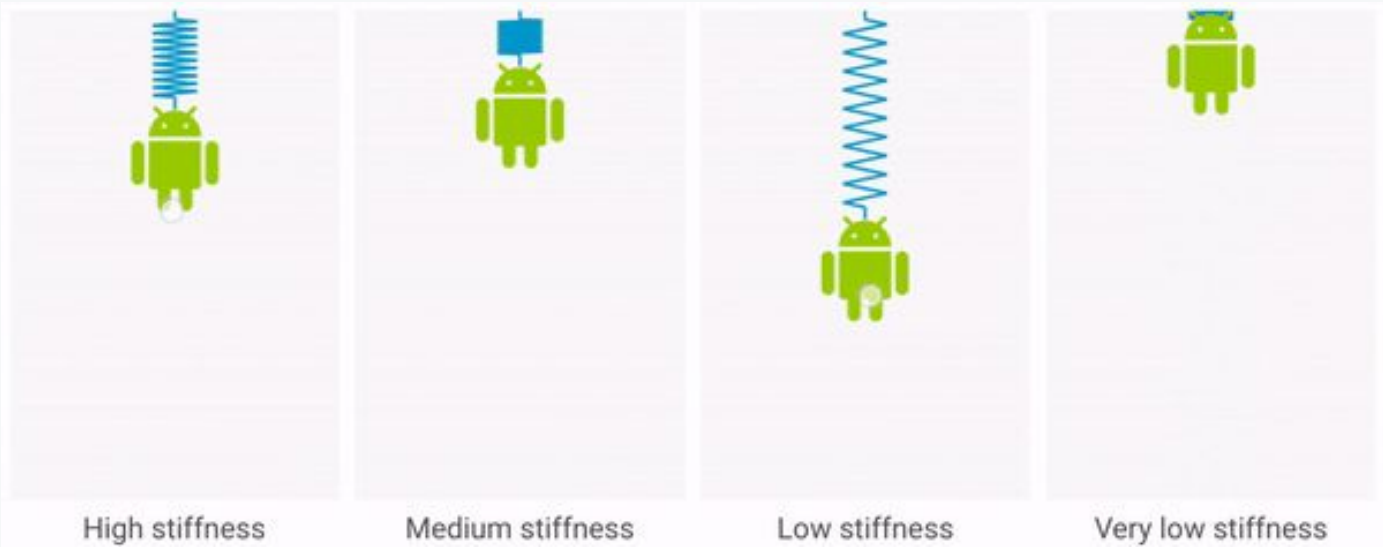
Spring animation

Damping (Amortissement) va influencer le “rebond”



Spring animation

Stiffness (Raideur) va influencer la “vitesse”



Spring animation

Modification de l'objet SpringForce:

- Propriété **dampingRatio**
- Propriété **stiffness**

```
val springAnimation = SpringAnimation(imageView, DynamicAnimation.TRANSLATION_Y,  finalPosition: 500f)
springAnimation.spring.dampingRatio = SpringForce.DAMPING_RATIO_HIGH_BOUNCY
springAnimation.spring.stiffness = SpringForce.STIFFNESS_MEDIUM
springAnimation.start()
```

ex1

Exercice

Exercice 1

Créez un nouveau projet dans Android Studio

De type Empty Activity

Nom: Exo1

Package name: com.technipixl.exo1

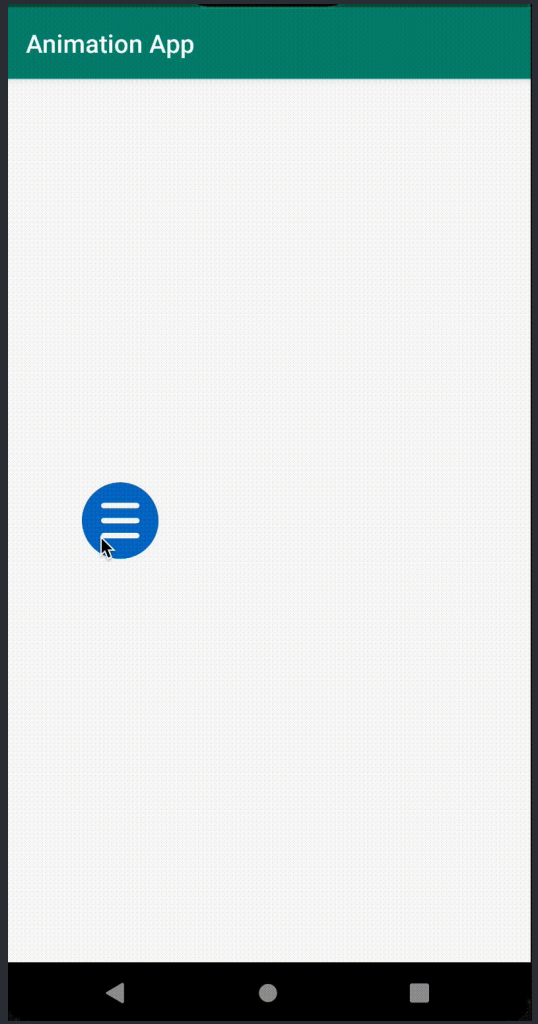
Placez le dans un dossier TECHNIFUTUR-AND16-EXO1

Exercice

Nous allons implémenter un menu circulaire animé

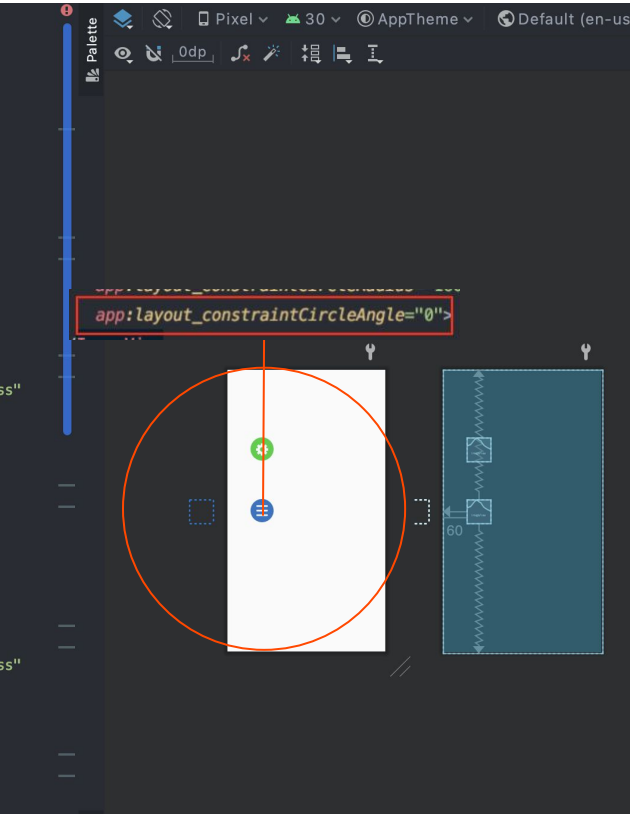
Afin d'y parvenir nous allons utiliser des fonctionnalités des ConstraintLayout:

- app:**layout_constraintCircle** :
  Contrainte circulaire de votre composant par-rapport à un autre composant
- app:**layout_constraintCircleRadius** :
  Distance de votre composant par-rapport à son centre
- app:**layout_constraintCircleAngle** :
  Angle à appliquer à votre composant par-rapport à sa contrainte circulaire



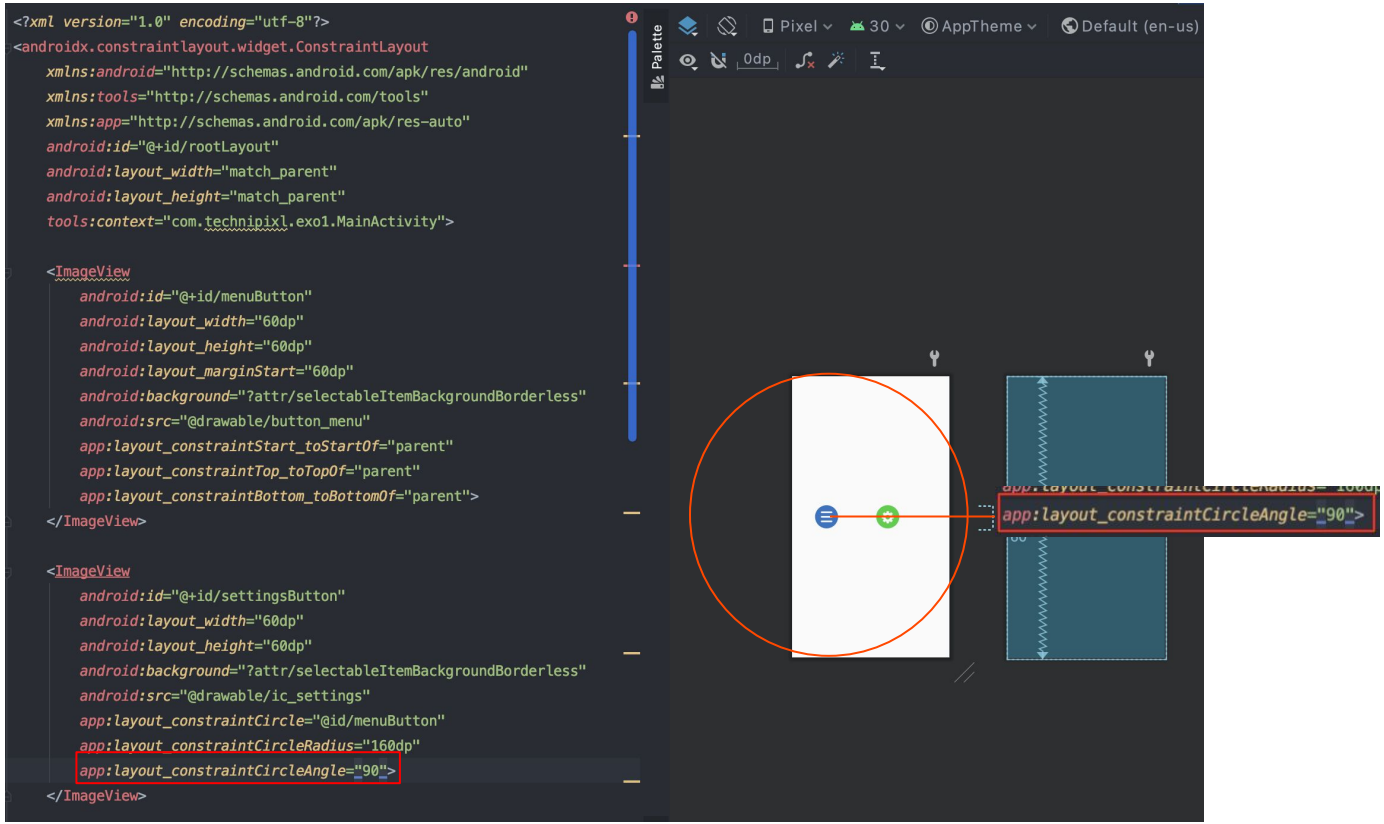
Exercice

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/rootLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.technipixl.exo1.MainActivity">

    <ImageView
        android:id="@+id/menuButton"
        android:layout_width="60dp"
        android:layout_height="60dp"
        android:layout_marginStart="60dp"
        android:background="?attr/selectableItemBackgroundBorderless"
        android:src="@drawable/button_menu"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent">
    </ImageView>

    <ImageView
        android:id="@+id/settingsButton"
        android:layout_width="60dp"
        android:layout_height="60dp"
        android:background="?attr/selectableItemBackgroundBorderless"
        android:src="@drawable/ic_settings"
        app:layout_constraintCircle="@id/menuButton"
        app:layout_constraintCircleRadius="160dp"
        app:layout_constraintCircleAngle="0">
    </ImageView>
```



neopixl. A SMILE GROUP COMPANY

Exercice




```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/rootLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.technipixl.exo1.MainActivity">

    <ImageView
        android:id="@+id/menuButton"
        android:layout_width="60dp"
        android:layout_height="60dp"
        android:layout_marginStart="60dp"
        android:background="?attr/selectableItemBackgroundBorderless"
        android:src="@drawable/button_menu"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent">
    </ImageView>

    <ImageView
        android:id="@+id/settingsButton"
        android:layout_width="60dp"
        android:layout_height="60dp"
        android:background="?attr/selectableItemBackgroundBorderless"
        android:src="@drawable/ic_settings"
        app:layout_constraintCircle="@id/menuButton"
        app:layout_constraintCircleRadius="160dp"
        app:layout_constraintCircleAngle="90">
    </ImageView>
```

Exercice

Vous l'aurez compris, vou devrez définir des valeurs correctes pour constraintCircle, constraintCircleRadius et constraintCircleAngle

Ensuite vous devrez **animer** la propriété **constraintCircleAngle** pour ouvrir le menu et pour le masque suite au clic sur le bouton

Exercice

Afin d'y parvenir vous devrez utiliser:

- AnimatorSet
- ValueAnimator et son UpdateListener

Remarques:

- Vous ne pourrez pas directement modifier la valeur de la propriété **circleAngle**
- Vous devrez intercepter les valeurs intermédiaires dans l'UpdateListener afin de mettre à jour le layout de vos boutons

```
val animator = ValueAnimator.ofFloat(startAngle, endAngle).apply {  this: ValueAnimator!
        this.addUpdateListener { valueAnimator ->
            val value = valueAnimator.animatedValue as Float
            val layoutParams = menuButton?.layoutParams as ConstraintLayout.LayoutParams
            layoutParams.circleAngle = value
            menuButton.layoutParams = layoutParams
        }
```

Avez-vous des questions?

neopixl.

A SMILE GROUP COMPANY

Restons en contact

neopixl.

A SMILE GROUP COMPANY

115A, Rue Emile Mark
L-4620 Differdange

(+352) 26 58 06 03
contact@neopixl.com