**Animation**

Property Animation API was introduced by Google in Android 3.0 which gives us the flexibility to change object properties over a certain time interval. The Animations Framework allows us to create visually attractive animations and transitions in our apps. Using the animations, one can turn their good looking app into an excellent and highly usable app.

Animations are useful when the screen changes state, i.e when the content loads or new actions become available.

Animation is the process of creating motion and shape change. In this Tutorial we will show how simple animations encourage you to use them more freely and confidently.

Animations notify users about what’s going on in your app and improve the mental model of your app’s interface.

**Android Defines Three Types Of Animations:**

**View Animation**:

This is the simplest animation used in Android. It define the properties of our Views that should be animated using a technique called Tween Animation .It take the following parameters i.e. size, time duration , rotation angle, start value , end value, and perform the required animation on that object. You can  execute the animation by specifying transformations on your View. This can be done in XML resource files or programmatically.

Android View animation can make animation on any View objects, such as ImageView, TextView or Button objects. View animation can only animate simple properties like position, size, rotation, and the alpha property that allows you animate the transparency of a View.

The drawback of this mechanism is that it can only be applied to Views.

**Property Animation:**

This animation was introduced in Android 3.0 (API level 11). It allows the user to animate anything.

Property animations are highly customizable, you can specify the duration, the number of repeats, the type of interpolation, and the frame rate of the animation. The Property Animation system is always preferred for more complex animations.

Property animations allow us to animate any property of any object from one value to another over a specified duration. Let us take an example, imagine you wanted to animate the 3d rotation using the rotationX or rotationY properties that were introduced in API 11. Or maybe you want to animate a color change or animate the change of a drawable. This is not possible by using View Animations. For this purpose Property Animation is used .

You are not limited by predefined animation types or by the type of object that you want to animate. At first the usage of the Property Animation System might seem a little complicated. However, in most cases you can use the very versatile ObjectAnimator that uses reflection. The ObjectAnimator makes creating Property Animations almost as easy as creating View Animations.

Common properties commonly animated on views include:

| **Property** | **Description** |
| --- | --- |
| Alpha | Fade in or out |
| rotation, rotationX, rotationY | Spin or flip |
| scaleX ,scaleY | Grow or shrink |
| x,y,z | Position |
| translationX, translationY, translationZ  (API 21+) | Offset from Position |

**Drawable Animation:**

This animation allows the user to load drawable resources and display them one frame after another. This method of animation is useful when user wants to animate things that are easier to represent with Drawable resources.

To use Animations in Android , Android has provided us a class called Animation.

To perform animation in android, we will  call a static function **loadAnimation()**(this method is used to load animation) of the class **AnimationUtils.** We are going to receive the result in an instance of Animation Object. Its syntax is as follows −

Animation animation =AnimationUtils.loadAnimation(getApplicationContext(), R.anim.myanimation);

Second parameter refers to our animation.xml file.It is created under res directory**(res->anim->myanimation.xml)**

**The Animation class has many methods given below:**

1.**start():**This method will start the animation.

2.**setDuration(long duration):**This method sets the duration of an animation.

3.**getDuration()**: This method gets the duration.

4.**end():** This method ends the animation.

5.**cancel()**: This method cancels the animation.

**Setting The Animation Listeners (Optional)**

Animation listeners can be set by implementing AnimationListener in our Activity. If you will use AnimationListener, then  you  will have to override following methods.

**onAnimationStart** –

@Override

public void onAnimationStart(Animation animation) {

}

**onAnimationEnd**–

@Override

public void onAnimationEnd(Animation animation) {

}

**onAnimationRepeat**–

@Override

public void onAnimationRepeat(Animation animation) {

}

***Scale Animation***

Scale Animation is used to make a smaller or larger view either on x axis or y axis. **Pivot point** can also be specified around which we want the animation to take place.

***Rotate Animation***

Rotate Animation is used to rotate a view around a pivot point by a certain number of degrees.

***Translate Animation***

Translate Animation is used to move a view along the x or y axis.

***Alpha Animation***

Transparency of a view can be changed by Alpha Animation

**Interpolator:**

The dictionary meaning of Interpolate is to alter, intercept or insert in between two things or events.

It is used to insert or apply an additional animation effects between the start and the end value of the property of an object. It is also used to define the rate of change of an animation.

**Types of Interpolator:**

1**.Linear Interpolator:**It is the default interpolator of all animations. It defines that the rate of the change of the animation is constant.

2.**Accelerate Interpolator:** Accelerates the moving of the view, it starts out slowly and then accelerates until it reaches the final position.

3.**Decelerate Interpolator:** Does the opposite of the accelerate interpolator. It starts out fast and the slows down.

4.**Accelerate Decelerate Interpolator:** Makes the moving go slow at the beginning and the end, but fast in the middle.

5.**Anticipate Interpolator:** Moves the animation backwards before it starts

6.**Overshoot Interpolator:** Does the opposite of the anticipate interpolator. It make the animation to go further than the defined destintion, and then get back to the defined destination.

7.**Anticipate Overshoot Interpolator:** Combines the anticipate and the overshoot interpoplator. The change starts backward then flings forward and overshoots the target value and finally goes back to the final value.

8.**Bounce Interpolator:** Makes a bounce animation before the desired animation ends.

9.**Cycle Interpolator:**Repeats the animation for a specified number of cycles.

**Important XML Animation Attributes In Android:**

**1. android:duration:** The duration in which animation is completed is referred to as duration attribute. It refers to the ideal duration to show the transition on the screen.

android:duration="1500"

**2. android:fillAfter:** This attribute defines whether the view should be visible or not at the end of the animation.We have set its value true in our animation. If it would set to false then element changes comes to its previous state after the animation.

android:fillAfter="true"

**3. android:interpolator:** This property refers to the rate of change in animation. You can define your own interpolators using the time as the constraint. In this animation we have used an inbuilt interpolator i.e.  linear interpolator.

android:interpolator="@android:anim/linear\_interpolator"

**4. android:startOffset:**It refers to  the waiting time before an animation starts.

android:startOffset="2000"

**5. android:repeatMode:**When user wants the animation to be repeated then this attribute is used

android:repeatMode="restart"

**6. android:repeatCount:**This attribute defines the number of repetitions on animation.

android:repeatCount="infinite"