**\*\* 서론**

Property Animation은 거의 모든 것을 animate 하게 해주는 강력한 프레임워크다. 스크린에 그려지는지 여부와 상관없이 오브젝트를 변화시킬 수 있다. 또한 View Animation보다 더 사용하기 flexible 하다.

뷰애니는 뷰오브젝트에만 적용할 수 있다. 그래서 뷰 오브젝트 아닌거에 하려면 코드를 새로 써야 했다. 그리고 animate할 오브젝트의 일부분만을 노출할 수 있었다. 예를 들어 뷰의 scaling, rotation.

또 뷰가 그려지는 곳에서만 수정할 수 있었다. 그리고 이건 실제 뷰도 아니었다. 예를 들어, 버튼이 스크린을 가로지르게 하고 싶으면 그건 잘 되지만 실제로 버튼을 클릭하는 위치는 변하지 않는다. 이걸 할려면 따로 구현해야 됨. 프로퍼티애니는 이런 제약 하나도 없고 어떤 오브젝트의 프로퍼티도 animate 할 수 있다. 그리고 오브젝트는 실제로 수정된다.

하지만 뷰애니가 셋업하는데 시간이 적게 들고 코드를 적게 써도 된다. 뷰에서 다 할 수 있으면 프로퍼티 애니를 쓸 필요가 없다.

**\*\* What is property animation?**

프로퍼티애니는 오브젝트의 필드값(프로퍼티)를 변경할 수 있다.

[변경시킬 수 있는 특성들 리스트]

duration, Time interpolation, Repeat count and behavior, Animator sets, Frame refresh delay.

**\*\* How the property animation system works**

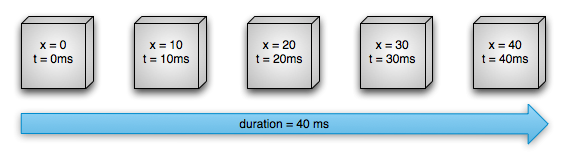


figure1은 엑스좌표만 변화시키는 보통의 애니매이션 예

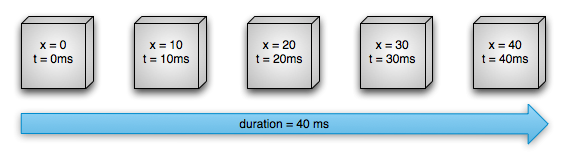
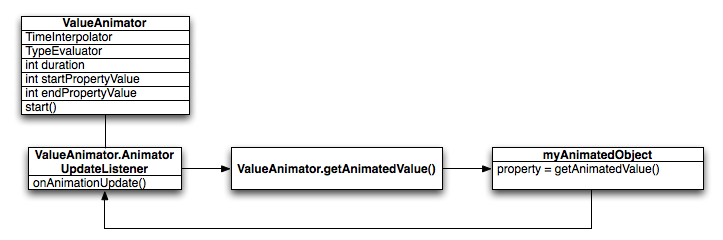


figure2는 리니어 interpolation 말고 가속됐다 감속되는거 예

프로퍼티 애니매이션 시스템의 컴포넌트들이 어떻게 애니매이션을 계산하는지 봅시다. figure3은 main class 들이 다른 것들과 어떻게 동작하는지 보여준다.



ValueAnimator 오브젝트는 애니매이션의 타이밍을 주시한다. (타이밍을 관리한다.) 예를 들어 애니매이션이 얼마나 재생되고 있었는지, 재생되고 있는 프로퍼티의 현재 값은 뭔지.

ValueAnimator는 애니매이션 interpolation을 정의하는 TimeInterpolator를 캡슐화 한다. 그리고 TypeEvaluator도 캡슐화 하는데, 이것은 재생되는 속성의 값을 계산하는 방법을 정의한다. 그림 2의 예를 들자면, TimeInterpolator는 ‘AccelerateDecelerateInterpolator’ 고, TypeEvaluator 는 ‘IntEvaluator’ 였을 거다.

애니매이션을 하기 위해서는 ValueAnimator를 생성하고, duration과, 재생할 시작값과 종료값을 지정해야 된다. start()를 콜 하면 애니매이션이 시작한다. 전체 애니매이션에서 ValueAnimator는 0과 1사이의 *elapsed fraction*을 계산한다. (애니매이션의 duration을 베이스로 시간이 얼마나 경과되었는지를 계산한다.) 그림1을 예로 들면, 10ms 후의 elapsed fraction은 0.25이다.

ValueAnimator는 elapsed fraction을 계산하면, *interpolated fraction*을 계산하기 위해, 현재 설정되어 있는 TimeInterpolator를 호출한다.) interpolated fraction은 elapsed fraction을 설정된 time interpolation을 참조하는 새로운 fraction값과 매칭시킨다. 그림 2를 예로 들면, 애니매이션이 천천히 가속되기 때문에, t=10ms일 때, interpolated fraction은 elapsed fraction보다 적다. (interpolated fraction은 0.15정도, elapsed fraction은 0.25정도) 그림1은 계속 두 값이 같다.

\*\*TimeInterpolator는 애니매이션의 변화 정도(rate of change)를 정의한다. 애니매이션이 non-linear로 동작할 수 있도록 해준다. getInterpolation() 메소드를 가지는데, 이것은 0~1사이의 현재 포인트를 나타내는 값을 파라미터로 받아서 interpolation value를 리턴한다. 이 값은 애니매이션 시간보다 더 지났으면 1을 넘을 거고, undershoot of their target 일 경우는 0보다 적을 것이다.

interpolated fraction이 계산되면, ValueAnimator는 재생하는 프로퍼티의 값을 계산하기 위해 적절한 TypeEvaluator를 호출한다. (애니매이션의 interpolated fraction, 시작 값, 종료 값을 베이스로 계산) 예를 들어, 그림 2에서, t=10ms일 때, interpolated fraction은 0.15였다. 그래서 이때의 프로퍼티 값은 0.15 \* (40 – 0) , 6이다.

**\*\* API Overview**

뷰애니매이션 시스템이 이미 interpolator를 많이 정의해 놨기 때문에, 이걸 프로퍼티 애니매이션 시스템에서도 사용할 수 있다. 아래는 프로퍼티 애니매이션 시스템의 main component들이다.

*Animator* 클래스는 애니매이션을 생성하는 기본 구조를 제공한다. 보통은 이 클래스를 직접 사용하지는 않는다. 왜냐면 이것은 확장되어야 하는 최소한의 기능들만 제공하기 때문이다. 아래는 Animator클래스의 서브 클래스 들이다.

1. ValueAnimator: 메인 타이밍 엔진이다. 재생되는 프로퍼티의 값을 계산한다. 애니매이션 값들을 계산하는 핵심 기능을 포함한다. 그리고 각 애니매이션의 timing detail을 포함한다. (애니매이션이 반복되었는지 아닌지 같은 정보, 업데이트 이벤트를 처리하는 리스너, 커스텀 타입을 설정하는 )

2. ObjectAnimator: ValueAnimator의 서브클래스로 재생할 타겟 오브젝트와 오브젝트 프로퍼티를 설정할 수 있게 해준다. 언제 새로운 값을 계산했는지에 맞춰 프로퍼티를 갱신한다. 이 클래스를 주로 사용하게 될 것이다. 왜냐하면 타겟오브젝트에서 재생값을 처리하는 걸 쉽게 만들어 주기 때문이다. 그러나, 이 클래스는 몇가지 제한이 있어서 가끔은 ValueAnimator를 직접 사용할 수도 있다. 타겟 오브젝트에 존재해야 하는 특정한 acessor 메소드를 요구하는 등의 제한이 있다.

3. AnimatorSet : 그룹 애니매이션의 매커니즘을 제공한다. 애니매이션을 함께 재생할 수도 있고, 순차적으로 재생할 수도 있다.

*Evaluator*는 프로퍼티 애니매이션 시스템에게 주어진 프로퍼티에 대한 값을 계산하는 방식을 알려준다. Animator 클래스에게서 제공받는 타이밍 데이터(시작, 종료 값)을 가지고 프로퍼티의 재생값을 계산한다. 다음과 같은 evaluator들이 있다.

1. IntEvaluator : int 프로퍼티들을 계산하는 디폴트 evaluator

2. FloatEvlauator: float 프로퍼티들을 계산하는 디폴트 evaluator

3. argbEvaluator : 16진수 값으로 표현되는 컬러 프로퍼티를 계산하는 디폴트 evaluator

4. TypeEvaluator: 사용자 정의 evaluator를 갖게 하는 인터페이스. int, float, color가 아닌 오브젝트 속성을 재생하고 싶을 때, 오브젝트 속성의 값들을 계산하기 위해서, TypeEvaluator 인터페이스를 구현해야 된다. 물론 int, float, color 값을 좀 다르게 프로세싱하고 싶으면, 이 값들의 커스텀 TypeEvaluator도 만들 수 있다.

*time interpolator*는 애니매이션에 속한 특정 값들이 어떻게 계산되는지를 정의한다. 예를 들어, 애니매이션을 리니어하게 움직이게 할 수도 있고, non-linear하게, 즉 가속, 감속 사용해서 애니매이션 할 수도 있다. Table3는 android.view.animatio이 포함하는 interpolator를 설명한다. 제공되는 interpolator중에 맘에 드는게 없으면, TimeInterpolator 인터페이스를 구현해서 쓰면된다.

1. AccelerateDecelerateInterpolator : 시작과 끝의 변화율은 천천히 하고 중간에선 가속하는 interpolator

2. Accelerateinterpolator : 시작할땐 천천히 시작해서 점점 가속하는 interpolator

3. AnticipateInterpolator : 첨엔 뒤에서 시작하다가 앞쪽으로 나오는 interpolator

4.. AnticipateovershootInterpolator : 시작은 뒤에서, 그다음 앞쪽으로 나오고, 그리고 나서 target value를 초과하는거, 마지막에는 final value로 돌아감

5. BounceInterpolator: 마지막에 반사되는거

6. CycleInterpolator : 지정된 반복 수만큼 애니매이션을 반복

7. DecelerateInterpolator : 첨에 빨리 시작하고 감속

8. LinearInterpolator : 변화율이 일정한거

9. OvershootInterpolaotr: 앞으로 튀어나오면서 시작하고 last value를 초과했다가 뒤로 다시 옴

10. TimeInterpolator: 커스텀 interpolator를 만들게 하는 인터페이스

**\*\*Animating with ValueAnimator**

*ValueAnimator* 클래스는 int, float, 컬러값 등을 명시해서, 애니매이션의 duration의 몇가지 타입의 값을 재생시킬 수 있게 한다. ofInt(), ofFloat(), ofObject()등의 factory method를 호출해서 ValueAnimator를 가질수 있다.

예)

ValueAnimator animation = ValueAnimator.ofFloat(0f, 1f);  
animation.setDuration(1000);  
animation.start();

이 코드에서 ValueAnimator는 start() 메소드가 시작되면 1000ms 동안 0과1 사이의 애니매이션 values의 계산을 시작한다.

커스텀 타입을 작성할 수도 있다.

ValueAnimator animation = ValueAnimator.ofObject(new MyTypeEvaluator(), startPropertyValue, endPropertyValue);  
animation.setDuration(1000);  
animation.start();

이 코드에서, Valueanimator는 start()메소드가 시작하면, 1000ms 동안 MyTypeEvaluator의 로직을 사용하는 startPropertyValue와 endPropertyValue 사이의 애니매이션 values의 계산을 시작한다.

그러나, 앞의 코드는 오브젝트에 어떤 효과도 주지 않는다. 왜나하면 Valueanimator가 오브젝트나 프로퍼티에서 직접 작동된 것이 아니기 때문이다. 사용자들이 주로 원하는 것은 이런 계산된 값들을 이용해서, 재생시키기를 원하는 오브젝트를 수정하는 것이다. 이것은 ValueAnimator 안의 리스너를 정의해서 할 수 있다. 리스너는 애니매이션이 라이프사이클동안 프레임 업데이트와 같은 중요한 이벤트를 적절하게 핸들하는 역할을 한다. 리스너를 작성할 때, getAnimatedValue()를 콜해서 프레임을 refresh 할 계산된 값들을 얻을 수 있다.

**\*\* Animating with ObjectAnimator**

ObjectAnimaotr는 ValueAnimator의 서브 클래스이다. 그리고 타이밍 엔진과 ValueAnimator의 값 계산 방식을 포함한다. 또한 타겟 오브젝트의 명명된 속성들을 재생할 수 있는 능력도 있다. 이것은 어떤 오브젝트도 재생하기 쉽게 해준다. 더 이상 ValueAnimator.AnimatorUpdateListener를 구현할 필요가 없다. 왜냐하면 재생되는 속성을 자동으로 업그레이드 하기 때문이다.

ObjectAnimaotr의 객체를 만드는 것은 Valueanimator와 비슷하다. 그러나 오브젝트와 오브젝트의 속성이름과 재생값들을 명시해야 한다.

ObjectAnimator anim = ObjectAnimator.ofFloat(foo, "alpha", 0f, 1f);  
anim.setDuration(1000);  
anim.start();

ObjectAnimator의 업데이트 속성을 정확하게 하려면 다음을 수행해야 한다.

-> 재생시킬 오브젝트의 속성은 반드시 setter function을 set<propertyaName>()형식으로 가져야 한다. 왜냐하면 ObjectAnmator는 애니매이션 중에 속성을 자동 업데이트 하는데, 이 메소드로 속성 값에 접근할 수 있기 때문이다. 예를 들어, 속성 이름이 foo 라면 setFoo()메소드를 가져야 한다. 이 메소드가 없으면 세가지 옵션이 있다.

1. setter 메소드를 클래스 안에 더한다. 2. wrapper 클래스를 사용한다. 3. ValueAnimaotr를 사용한다.

->values..에 하나의 값만 작성한다면 그것은 종료값으로 인식된다. 그러므로, 오브젝트의 속성은 시작 값을 얻는 getter function을 가져야 한다. get<propertyName>의 형식으로 만든다. 예를 들어 프로퍼티 이름이 foo라면 getFoo() 메소드를 작성한다.

-> get, set 메소드는 시작, 종료값이 같은 타입이어야 한다. 예를 들어

• ObjectAnimator.ofFloat(targetObject, "propName", 1f)

이것처럼 ObjectAnimator를 생성하면, targetObject.setPropName(float), targetObject.getProName(float) 를 가져야 한다.

-> 어떤 오브젝트와 속성을 재생할 것인지에 따라서, invalidate()를 호출해야 할 수도 있다. 애니매이션 벨류를 업데이트 해서 강제적으로 스크린을 다시 그리는 메소드다. onAnimationUpdat() 콜백 함수에서 해야 된다. 예를 들어, 배경색은 자동으로 업데이트 하니까 안 불러도 된다

**\*\* Choreographing Multiple Animations with AnimatorSet**

많은 상황에서, 다른 애니매이션의 상황에 따라 애니매이션을 실행하고 싶을 것이다. 안드로이드 시스템은 AnimatorSet에서 여러 애니매이션을 같이 재생할 수 있게 해준다. 애니매이션이 같이 시작할지, 순차적으로 할지, 그런거만 작성하면 된다.

\*\* Animation Listeners

리스너로 중요한 이벤트를 애니매이션이 재생되는 도중에 받을 수 있다.

Animator.AnimatorListener

onAnimationStart()

onAnimationEnd()

onAnimationRepeat()

onAnimationCancel() : onAnimationEnd()를 콜한다.

ValueAnimator.AnimationUpdateListener

onAnimationUpdat() : 모든 애니매이션 프레임 마다 불린다. 재샞 웅에 ValueAnimator에 의해 만들어진 계산된 값들을 사용하기 위해서 불린다. getAnimatedValue()메소드로 현재의 재생 값들을 얻기 위해 ValueAnimator 오브젝트가 이벤트로 들어간다. ValueAnimator를 쓸 때 작성한다.

어떤 오브젝트와 프로퍼티를 애니매이팅하는지에 따라 ,invalidate()를 호출해야 할지도 모른다. 예를 들어, 드러워블 오브젝트의 컬러속성을 애니매이팅 하는거는 오브젝트가 새로 그려질 때 스크린을 업데이트 하는 것만 유발한다. setAlpha()나 setTraslationX() 같은 걸로 뷰의 속성을 세팅하는 거는 뷰 프로퍼티를 invalidate 한다. 그래서 이런 메소드들을 새로운 값을 인자로 해서 호출할 때는 invalidate를 할 필요가 없다.

Animator.AnimatorListener 인터페이스의 모든 메소드를 구현하고 싶지 않으면, Animator.AnimatorListener 인터페이스를 구현하는 대신에 AnimatorListenerAdapter클래스를 확장할 수 있다. AnimatorListenerAdapter 클래스는 오버라이드 할 수 있는 빈 메소드들을 제공한다.

\*\* Animating Layout Changes to viewGroups

프로퍼티 애니매이션 시스템은 뷰 오브젝트를 쉽게 애니매이트 하는 방법을 제공할 뿐 아니라, 뷰그룹 오브젝트를 애니매이트 할 수도 있다.

LayoutTransition 클래스로 뷰 그룹 안의 레이아웃 변경하는 것을 애니매이트 할 수 있다. 뷰그룹 안의 뷰를 사라지게 하거나 생기게 하는 애니매이션을 할 수 있다. (View의 setVisibility() 메소드를 VISIBLE로 하는거 등을 통해서), 뷰 그룹 안에 새로운 뷰를 추가, 삭제 할 때 다른 뷰들의 포지션 변경도 애니매이트 할 수 있다. LayoutTransition 오브젝트의 다음과 같은 애니매이션들은 setAnimator()를 호출하고, Animator오브젝트에 LayoutTransition constant를 보내서 정의할 수 있다.

예) LayoutTransition.setAnimator(layoutTransition.APPEARING, ~~~~);

1. APPEARING – 뷰가 새로 생길때의 애니매이션을 나타내는 flag

2. CHANGE\_APPEARING – 새로운 뷰가 생기면서 다른 뷰들의 이동을 나타내는 flag

3. DISAPPEARING – 뷰가 삭제될때의 애니매이션을 나타내는 flag

4. CHANGE\_DISAPPEARING – 뷰가 삭제될 때 다른 아이템들의 이동을 나타내는 flag

이 네가지 타입을 위해서 커스텀 애니매이션을 작성할 수 있다. 레이아웃을 커스터마이즈 하거나, 애니매이션 시스템에 디폴트 애니매이션으로 얘기하면 가능하다.

후자의 방법은 xml 파일의 레이아웃 태그에 android:animateLayoutchanges 속성을 true로만 하면 된다.

**\*\* Using a TypeEvaluator**

안드로이드 시스템에 알려져 있지 않은 타입을 애니매이트 하고 싶을 때, TypeEvaluator 인터페이스를 구현해서 커스텀 evaluator를 만들수 있다. int, float, color는 각각, IntEvaluator, FloatEvaluator, ArgbEvaluator 의 evaluator가 지원한다.

TypeEvaluator 인퍼페이스에는 구현할 메소드가 evaluate() 하나밖에 없다. 이것은 애니매이션의 현재 포인트에서 재생될 속성에 알맞은 값을 리턴할 수 있도록 해준다.

FloatEvaluator 클래스의 예를 보면

public class FloatEvaluator implements TypeEvaluator {  
  
    public Object evaluate(float fraction, Object startValue, Object endValue) {  
        float startFloat = ((Number) startValue).floatValue();  
        return startFloat + fraction \* (((Number) endValue).floatValue() - startFloat);  
    }  
}

ValueAnimator나 ObjectAnimator가 시작될 때, 애니매이션의 현재 elapsed fraction을 계산한다. 그리고 사용하고 있는 interpolator에 따라 interpolated version을 계산한다. interpolated fraction은 TypeEvaluator가 fraction 파라미터를 통해 받는 값이다. 그래서 재생되는 값을 계산할 때 interpolator를 고려할 필요가 없다.

**\*\*Using Interpolators**

interpolator는 애니매이션의 특정 값들이 어떻게 계산되는지를 정의한다. 예를 들어, … 앞에 나왔던거,

interpolators 는 Animator로부터 fraction을 받는다. interpolators는 이 fraction을 일치하는 애니매이션의 타입으로 수정한다. 안드로이드 시스템은 common interpolator를 android.view.animation 패키지에서 제공한다. 이중에서 적당한게 없으면 TimeInterpolator 인터페이스로 커스텀 객체를 만든다.

예를 들어, 디폴트 interpolator인 AccelerateDecelateInterpolator와, LinearInterpolator가 어떻게 interpolated fraction을 계산하는 지를 비교해 보았다. LinearInterpolator는 elapsed fraction에 영향을 주지 않는다. AccelearateDecelerateInterpolator는 시작할 때 가속하고 끝날 때 감소한다.

**\*\*Specifying Keyframes**

Keyframe 오브젝트는 시간/값 쌍을 포함한다. 이 쌍은 애니매이션의 특정한 시간에서의 특정한 상태를 정의할 수 있게 해준다. 각 Keyframe은 이전 keyframe 시간과 자신의 시간 사이의 구간에서의 애니매이션의 행동을 조절하기 위해, 자신의 interpolator를 갖는다

Keyframe 오브젝트를 초기화 하기 위해서는, 적절한 Keyframe 타입을 얻기 위해 ofInt(). ofFloat(), ofObject()같은 factory method를 사용해야 한다. 그런다음 ProperfyValuesHolder 오브젝트를 얻기 위해 ofKeyframe() 메소드를 호출한다. 오브젝트를 한번 갖고 나면, PropertyValueholder 오브젝트와 애니매이트 할 오브젝트를 보내서 animator 를 얻을 수 있다.