ТЕХНОТРЕК

Занятие №9

Разработка приложений на Android

Кирилл Филимонов
Юрий Береза

Напоминание

А ты отметился о присутствии на занятии?

Agenda

- Custom View
- Анимация

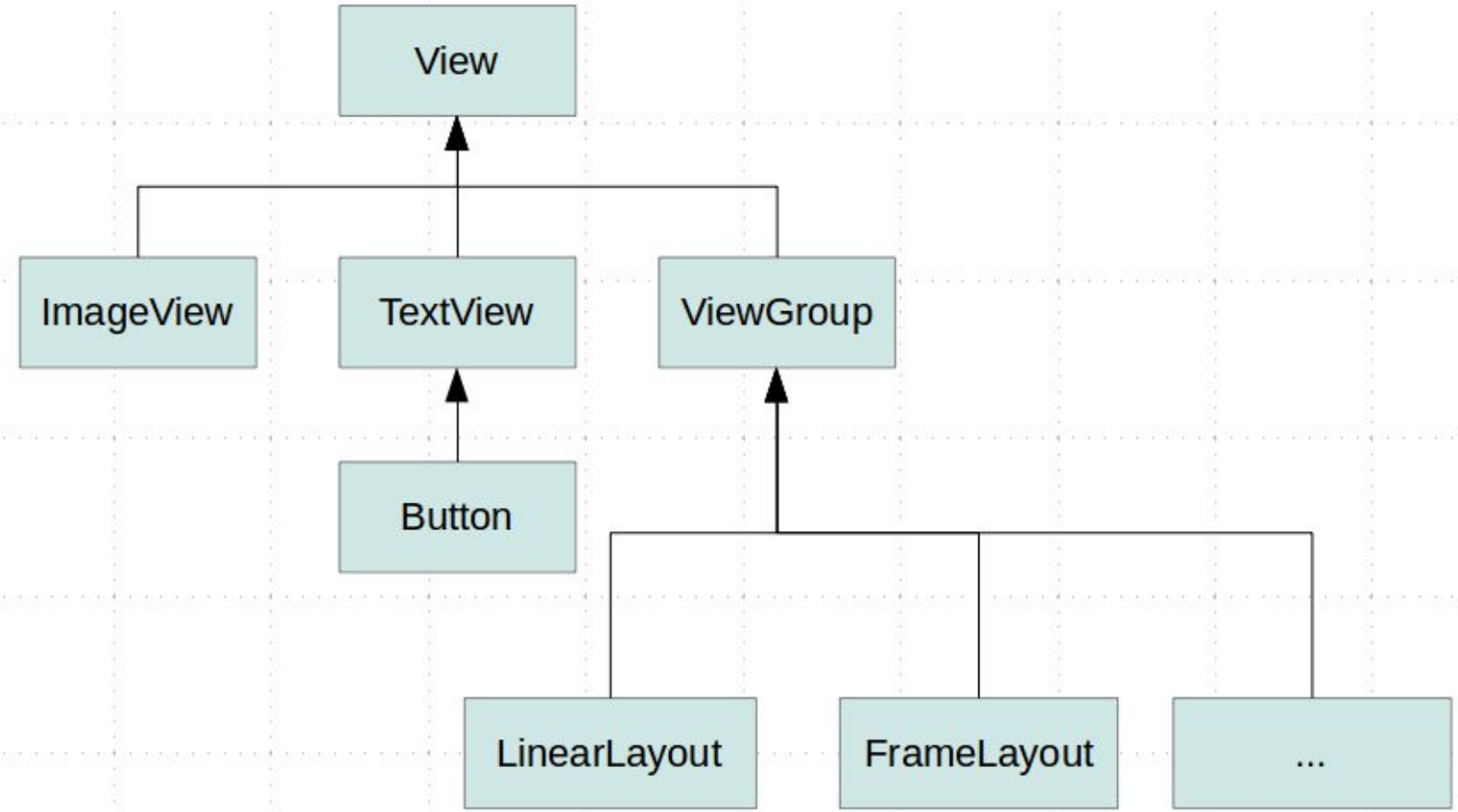
Свой View

- Зачем может понадобится
- Как создать
- Как использовать
- Какие функции надо реализовать
- Собственные атрибуты xml
- Собственная отрисовка
- Вычисление размеров
- Оптимизация

Зачем может понадобится View

- Нестандартное поведение
- Нестандартная отрисовка
- Все остальное вписывается в эти два пункта
  - Выравнивание
  - Оптимизация
  - Жесты
  - Обработка ввода

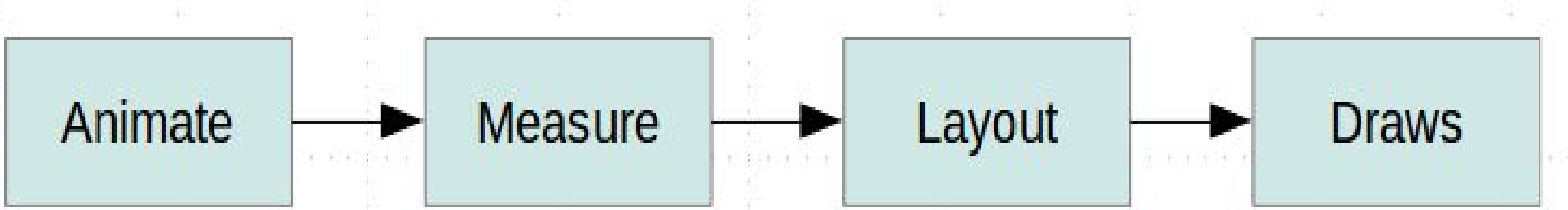
Что у вас есть



Как создать

- Наследуетесь от любого View чей функционал вам ближе
- Определяете нужные вам атрибуты
- Применяете атрибуты в классе
- Добавляете нужные вам свойства и события
- Реализуете весь нужный вам функционал
- Если нужно, описываете все в документации
- Обрабатываете все виды ввода
- Предоставляете возможности для людей с ограниченными возможностями

Жизненный цикл View



Наследование

- Лучший выбор - обычный View
- Надо делать лишнюю работу по копированию вставке кода
- Не надо выкручиваться с видимостью функций и данных
- Прежде чем наследоваться от другого класса проверьте что вы сможете воспользоваться его преимуществами

Важные функции

- onFinishInflate()
- onMeasure(int, int)
- onLayout(boolean, int, int, int, int)
- onSizeChanged(int, int, int, int)
- onDraw(Canvas)
- onTouchEvent(MotionEvent)
- onFocusChanged(boolean, int, Rect)

Атрибуты

- В файл res/values/attrs.xml надо добавить раздел <declare-styleable> с именем вашего view
- Внутри надо прописать нужные вам атрибуты
- Атрибуты бывают:
  - reference - ("@color/my_color", "@layout/my_layout")
  - color
  - boolean
  - dimension
  - float
  - integer
  - string
  - fraction
  - enum
  - flag

```
<resources>
    <declare-styleable name="PieChart">
        <attr name="showText" format="boolean" />
        <attr name="labelPosition" format="enum">
            <enum name="left" value="0"/>
            <enum name="right" value="1"/>
        </attr>
    </declare-styleable>
</resources>
```

Использование атрибутов

Пример использования атрибутов

- Вы создаете xml namespace - *xmlns:your_text*
- Используете атрибуты через созданный namespace - *your_text:your_attr=“value”*

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:custom="http://schemas.android.com/apk/res/com.example.customviews">
 <com.example.customviews.charting.PieChart
     custom:showText="true"
     custom:labelPosition="left" />
</LinearLayout>
```

Загрузка значений атрибутов

- Делается в конструкторе View
- Используется AttributeSet для доступа к значениям атрибутов
- Используется obtainStyledAttributes для матчинга значений с атрибутами

```
public PieChart(Context context, AttributeSet attrs) {
    super(context, attrs);
    TypedArray a = context.getTheme().obtainStyledAttributes(
         attrs,
         R.styleable.PieChart,
         0, 0);

    try {
        mShowText = a.getBoolean(R.styleable.PieChart_showText, false);
        mTextPos = a.getInteger(R.styleable.PieChart_labelPosition, 0);
    } finally {
        a.recycle();
    }
}
```

Свойства и события

- Обычно дублируют задание и получение атрибутов
- Не забывайте вызывать invalidate() если нужно визуальное обновление
- И requestLayout() если меняется размер

```
public boolean isShowText() {
    return mShowText;
}

public void setShowText(boolean showText) {
    mShowText = showText;
    invalidate();
    requestLayout();
}
```

Своя отрисовка

Своя отрисовка реализуется в функции *onDraw(Canvas)*

- Canvas – Холст, на котором рисуем
- Paint - Чем рисуем (цвет, толщина и пр)
  - Canvas дает метод для отрисовки линии, Paint – определяет цвет линии
  - Canvas имеет метод для отрисовки прямоугольника, Paint – позволяет определить заливать его или оставить контуром

Своя отрисовка

- Текст рисуйте с помощью drawText
- Примитивы рисуйте
  - drawRect
  - drawOval
  - drawArc
  - drawLines
  - drawPoints
- Картинки с помощью drawBitmap

Размеры и альфа

1. void onSizeChanged(int w, int h, int oldw, int oldh)
   - Будет вызвано на изменения размера
   - w, h – новая ширина высота
   - oldw, oldh – предыдущие высота и ширина
2. boolean onSetAlpha(int alpha)
   - Вызывается на изменения прозрачности
   - alpha значение от 0 до 255
   - возвращает true если view может быть отрисовано с установленным alpha

onMeasure

- Должен рассчитать размеры View с учетом детей и значений аргументов и передавать рассчитанные значения в метод setMeasuredDimension
- Если не вызвать setMeasureDimension будет брошено исключение при размещении элемента
- ОС будет знать сколько места на экране отвести под конкретное View
- Узнать рассчитанные высоту и ширину View можно с помощью функций getMeasuredHeight и getMeasuredWidth

onMeasure

- Аргументы это размер и спецификация, упакованные вместе
- Как с ними работать

```
protected void onMeasure(int widthMeasureSpec,
                         int heightMeasureSpec) {

    int widthMode = MeasureSpec.getMode(widthMeasureSpec);

    int widthSize = MeasureSpec.getSize(widthMeasureSpec);
```

onMeasure

Mode может быть 3 значений:

- EXACTLY соотвествует точным значениям в layout_width или layout_height. Вы должны уложиться в эти рамеры и вернуть их. Это также иногда используется в случае match_parent (зависит от фреймворка).
- AT_MOST обычно значит match_parent или wrap_content в layout_width или layout_height. Вам нужно установить размер не больше указанного.
- UNSPECIFIED обчно значит wrap_content в layout_width или layout_height. Можете указывать что угодно. Иногда вызывается просто чтобы оценить размер (первый элемент списка)

Оптимизация

- Не создавать свой View
- Не аллоцируйте в нем объекты
- Ничего не делать в нем
- Не рисовать в нем

Анимации

- Анимации
- Анимации свойств
- Анимации View
- Анимации через Transitions API
- GIF анимации

Анимации

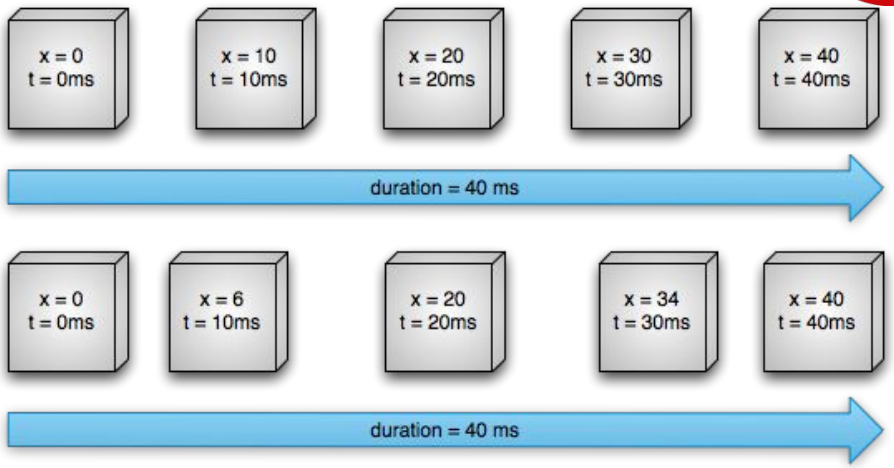
- Позволяет сделать красиво
- Позволяет скрыть работу приложения
- Делает ваше приложение визуально более отзывчивым
- Позволяет выделить ваше приложение
- Можно анимировать свойства
- Можно анимировать окна
- Можно анимировать через onDraw
- Можно анимировать через Transitions API
- Теоретически можно анимировать через GIF

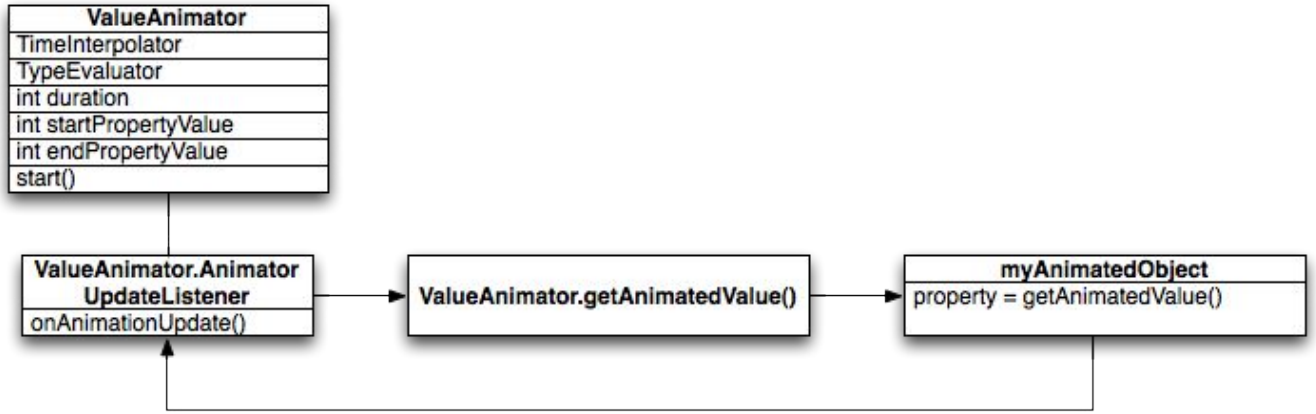
Анимации свойств

- Продолжительность
- Управление интерполяцией
- Повтор и поведение повтора
- Наборы анимации
- Время обновления фрейма
- Можно анимировать не только элементы UI

Как это работает

- Линейная анимация
- Нелинейная анимация



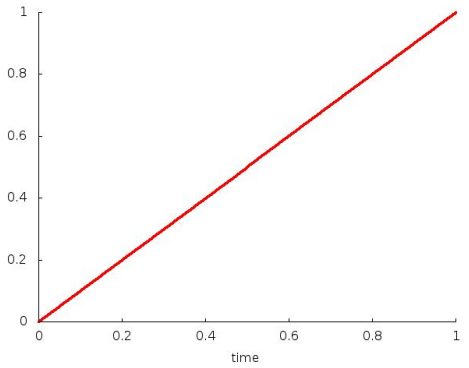


Общий обзор классов

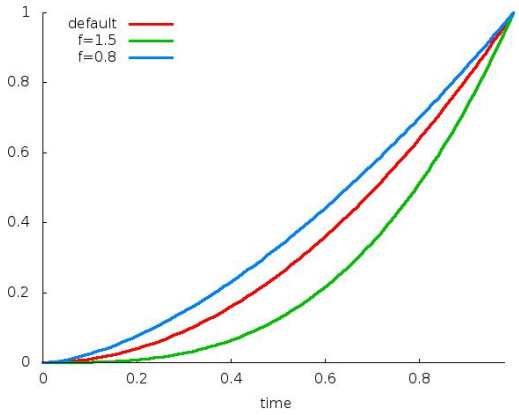
- Аниматоры – структура для создания анимации
  - ValueAnimator – анимации значений
  - ObjectAnimator – анимации параметров объекта
  - AnimatorSet – набор анимаций
- Evaluator – класс для работы со значением
  - IntEvaluator
  - FloatEvaluator
  - ArgbEvaluator
  - TypeEvaluator – базовый класс для пользовательского evaluator
- Интерполятор
  - LinearInterpolator
  - AccelerateDecelerateInterpolator
  - AccelerateInterpolator
  - AnticipateInterpolator
  - AnticipateOvershootInterpolator
  - BounceInterpolator
  - CycleInterpolator
  - DecelerateInterpolator
  - OvershootInterpolator
  - TimeInterpolator – базовый класс для пользовательского интерполятора

Интерполяторы

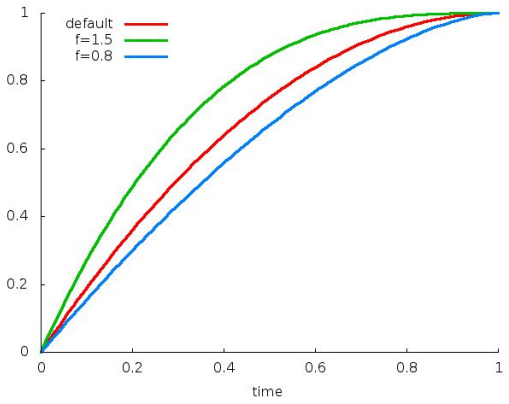
- LinearInterpolator



- AccelerateInterpolator

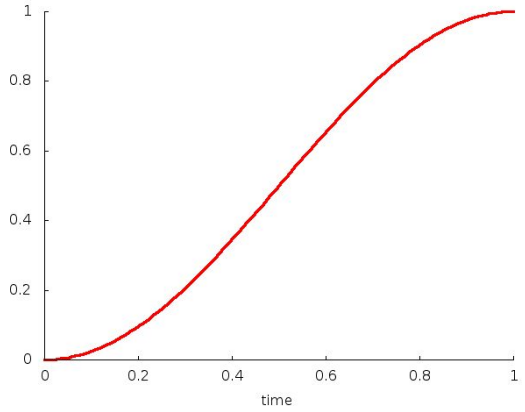


- DecelerateInterpolator

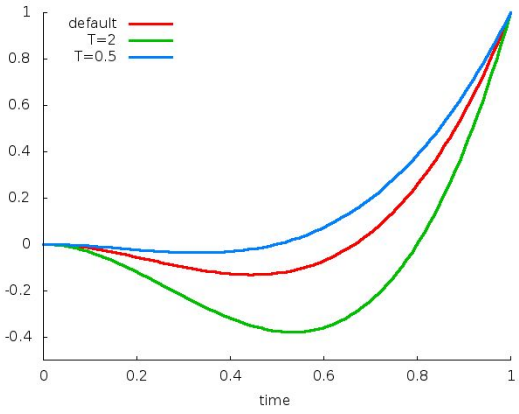


Интерполяторы

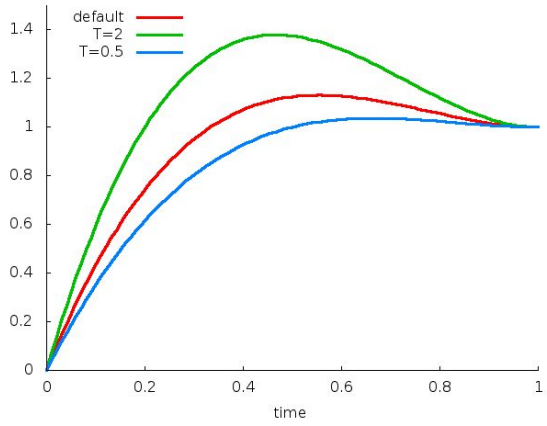
- AccelerateDecelerateInterpolator



- AnticipateInterpolator

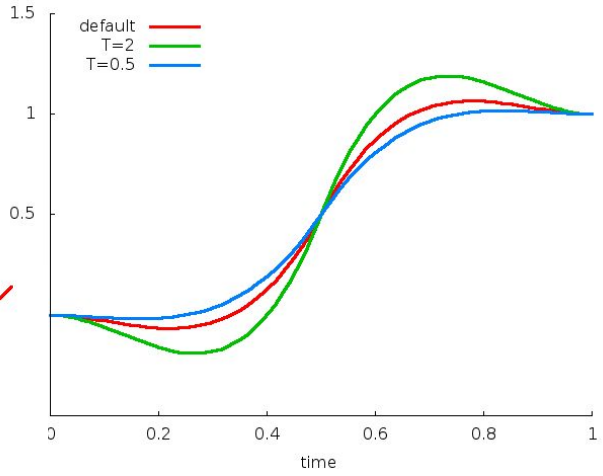


- OvershootInterpolator

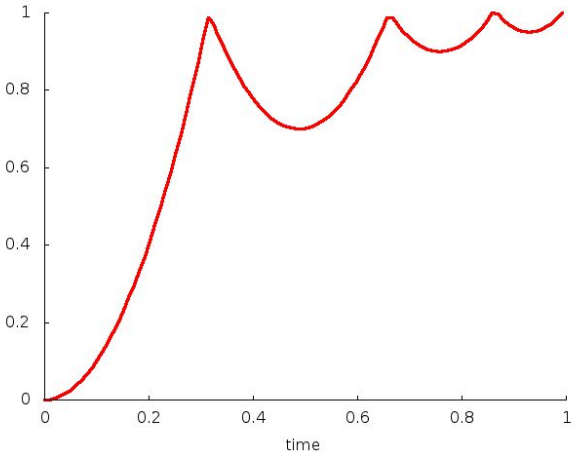


Интерполяторы

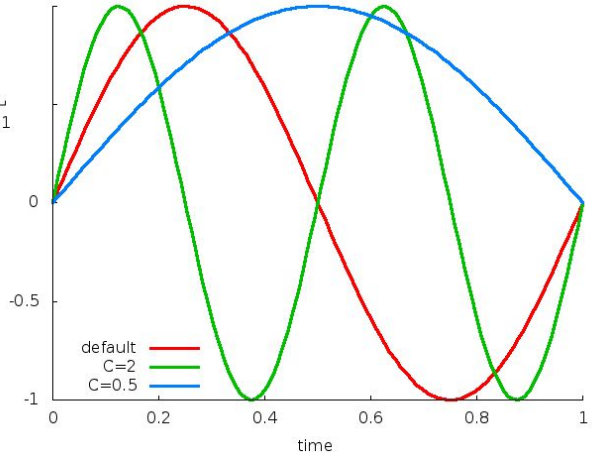
- AnticipateOvershootInterpolator



- BounceInterpolato



- CycleInterpolator



ValueAnimator и ObjectAnimator

- Создаем класс
- Задаем длительность (прочие свойства)
- Запускаем

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

- Все тоже самое с ObjectAnimation
- Задаем параметр объекта который анимируется

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

AnimationSet

- Контейнер анимаций
  - Пусть играем сначала bounceAnim
  - Потом играем все squashAnim* и stretchAnim*
  - Потом bounceBackAnim
  - Затем fadeAnim

```
AnimatorSet bouncer = new AnimatorSet();
bouncer.play(bounceAnim).before(squashAnim1);
bouncer.play(squashAnim1).with(squashAnim2);
bouncer.play(squashAnim1).with(stretchAnim1);
bouncer.play(squashAnim1).with(stretchAnim2);
bouncer.play(bounceBackAnim).after(stretchAnim2);
ValueAnimator fadeAnim = ObjectAnimator.ofFloat(newBall, "alpha", 1f, 0f);
fadeAnim.setDuration(250);
AnimatorSet animatorSet = new AnimatorSet();
animatorSet.play(bouncer).before(fadeAnim);
animatorSet.start();
```

Animation Listeners

- Нужны для обработки анимации
- Animator.AnimatorListener
  - onAnimationStart() – начало анимации
  - onAnimationEnd() – окончание анимации
  - onAnimationRepeat() – когда анимация повторяется
  - onAnimationCancel() – на отмену анимации
- ValueAnimator.AnimatorUpdateListener
  - onAnimationUpdate() – на каждый фрейм
- AnimatorListenerAdapter – имеет реализацию ф-ци по умолчанию

```
ValueAnimatorAnimator fadeAnim = ObjectAnimator.ofFloat(newBall, "alpha", 1f, 0f);
fadeAnim.setDuration(250);
fadeAnim.addListener(new AnimatorListenerAdapter() {
public void onAnimationEnd(Animator animation) {
    balls.remove(((ObjectAnimator)animation).getTarget());
}
```

TypeEvaluator

- Позволяет работать с пользовательскими типами
- Работает через метод evaluate
- Рассчитывается от доли (от 0 до 1)

```
public class FloatEvaluator implements TypeEvaluator<Float> {

    @Override
    public Float evaluate(float fraction, Float startValue, Float endValue) {
        return startValue + fraction * (endValue - startValue);
    }
}
```

Собственный интерполятор

AccelerateDecelerateInterpolator

```
public float getInterpolation(float input) {
    return (float)(Math.cos((input + 1) * Math.PI) / 2.0f) + 0.5f;
}
```

LinearInterpolator

```
public float getInterpolation(float input) {
    return input;
}
```

Анимации View

Поддерживаются следующие анимации

- translationX, translationY – перемещение
- rotation, rotationX, rotationY – вращение
- scaleX, scaleY – скалирование
- pivotX, pivotY – изменение точки опоры (вращения)
- x, y – задание позиции
- alpha - прозрачность

```
ObjectAnimator.ofFloat(myView, "rotation", 0f, 360f);
```

ViewPropertyAnimation

- Позволяет быстро запустить анимацию с длительностью по умолчанию
- Сменить длительность можно через myView.animate().setDuration(..)

Multiple ObjectAnimator objects

```
ObjectAnimator animX = ObjectAnimator.ofFloat(myView, "x", 50f);
ObjectAnimator animY = ObjectAnimator.ofFloat(myView, "y", 100f);
AnimatorSet animSetXY = new AnimatorSet();
animSetXY.playTogether(animX, animY);
animSetXY.start();
```

One ObjectAnimator

```
PropertyValuesHolder pvhX = PropertyValuesHolder.ofFloat("x", 50f);
PropertyValuesHolder pvhY = PropertyValuesHolder.ofFloat("y", 100f);
ObjectAnimator.ofPropertyValuesHolder(myView, pvhX, pvyY).start();
```

ViewPropertyAnimator

```
myView.animate().x(50f).y(100f);
```

Настройка анимации через xml

- Анимации свойств хранятся в ресурсах по пути /res/animator
- Анимации преобразований в /res/anim можно и в animator, но лучше не путать
- Задаются анимации которые обсуждались раньше через теги
  - ValueAnimator - <animator>
  - ObjectAnimator - <objectAnimator>
  - AnimatorSet - <set>
- Соответственно при загрузке получаем указатели на эти объекты

Пример анимации через xml

```xml
<set android:ordering="sequentially">
    <set>
        <objectAnimator
            android:propertyName="x"
            android:duration="500"
            android:valueTo="400"
            android:valueType="intType"/>
        <objectAnimator
            android:propertyName="y"
            android:duration="500"
            android:valueTo="300"
            android:valueType="intType"/>
    </set>
    <objectAnimator
        android:propertyName="alpha"
        android:duration="500"
        android:valueTo="1f"/>
</set>
```

```
AnimatorSet set = (AnimatorSet) AnimatorInflater.loadAnimator(myContext,
    R.anim.property_animator);
set.setTarget(myObject);
set.start();
```

Настройка анимации View через xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@[package:]anim/interpolator_resource"
    android:shareInterpolator=["true" | "false"] >
    <alpha
        android:fromAlpha="float"
        android:toAlpha="float" />
    <scale
        android:fromXScale="float"
        android:toXScale="float"
        android:fromYScale="float"
        android:toYScale="float"
        android:pivotX="float"
        android:pivotY="float" />
    <translate
        android:fromXDelta="float"
        android:toXDelta="float"
        android:fromYDelta="float"
        android:toYDelta="float" />
    <rotate
        android:fromDegrees="float"
        android:toDegrees="float"
        android:pivotX="float"
        android:pivotY="float" />
    <set>
        ...
    </set>
</set>
```

- хранятся в ресурсах по пути /res/anim
- при загрузке получаем ссылку на объект Animation

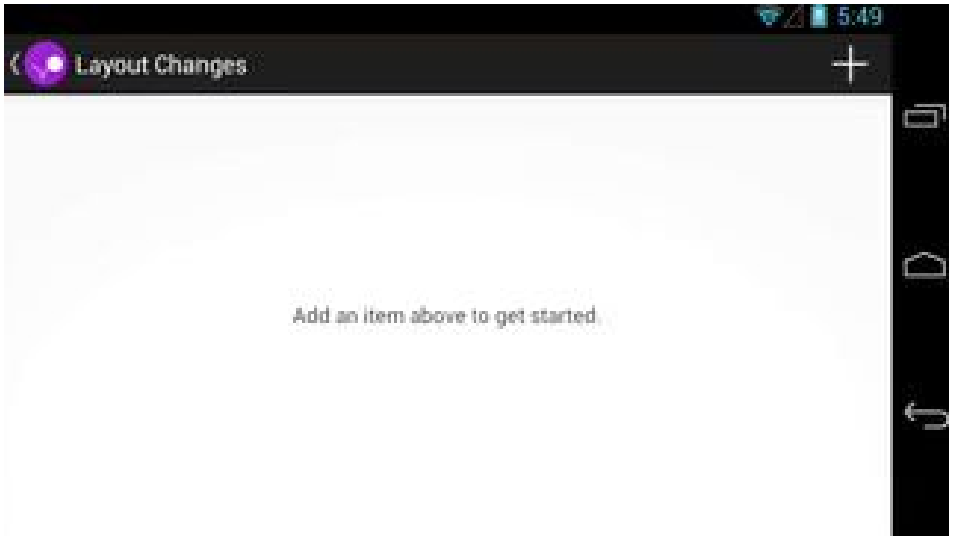
Пример

```xml
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false">
    <scale
        android:interpolator="@android:anim/accelerate_decelerate_interpolator"
        android:fromXScale="1.0"
        android:toXScale="1.4"
        android:fromYScale="1.0"
        android:toYScale="0.6"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fillAfter="false"
        android:duration="700" />
    <set
        android:interpolator="@android:anim/accelerate_interpolator"
        android:startOffset="700">
        <scale
            android:fromXScale="1.4"
            android:toXScale="0.0"
            android:fromYScale="0.6"
            android:toYScale="0.0"
            android:pivotX="50%"
            android:pivotY="50%"
            android:duration="400" />
        <rotate
            android:fromDegrees="0"
            android:toDegrees="-45"
            android:toYScale="0.0"
            android:pivotX="50%"
            android:pivotY="50%"
            android:duration="400" />
    </set>
</set>
```

```
ImageView image = (ImageView) findViewById(R.id.image);
Animation hyperspaceJump = AnimationUtils.loadAnimation(this,
R.anim.hyperspace_jump);
image.startAnimation(hyperspaceJump);
```

Transition API

- Анимация изменений в layout
- Добавлена в 4.4
- Начиная с 4.0 добавился флаг animateLayoutChange в ViewGroup
- Основные понятия
  - Scene – сцена, состояние layout
  - Transition – переход между сценами
  - Scene root – корневой layout в котором меняются сцены

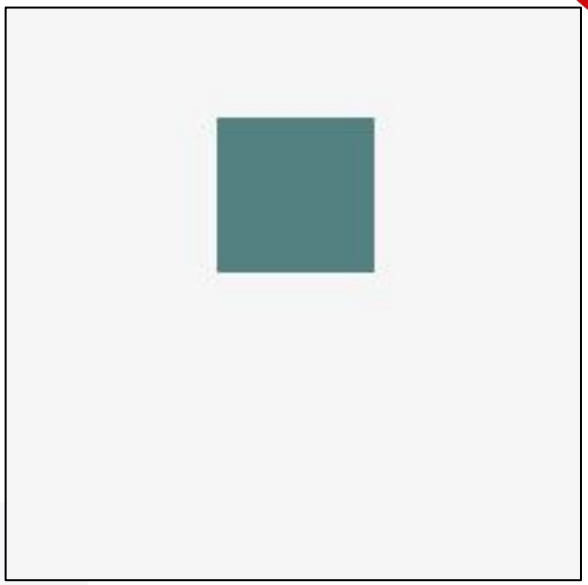


Простой пример

```
<FrameLayout
    android:id="@+id/container"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <View
        android:id="@+id/transition_square"
        android:layout_width="@dimen/square_size_normal"
        android:layout_height="@dimen/square_size_normal"
        android:background="@color/square_bg"/>

</FrameLayout>
```



```
ViewGroup sceneRoot = (ViewGroup) findViewById(R.id.container);
View square = mSceneRoot.findViewById(R.id.transition_square);
int newSquareSize = getResources().getDimensionPixelSize(R.dimen.square_size_expanded);

// вызываем метод, говорящий о том, что мы хотим анимировать следующие изменения внутри sceneRoot
TransitionManager.beginDelayedTransition(sceneRoot);

// и применим сами изменения
ViewGroup.LayoutParams params = square.getLayoutParams();
params.width = newSquareSize;
params.height = newSquareSize;
square.setLayoutParams(params);
```

Типы Transition

- **ChangeBounds** - это Transition, который отвечает за изменение координат View внутри layout и его размеров.
- **Fade** - объединяет в себе анимации fade in и fade out.
- **TransitionSet** - представляет из себя всего лишь набор какого-либо количества других Transition, которые выполняются по очереди или одновременно.
- **AutoTransition** - выполняется, когда мы не указали никакого конкретного

Пример с двумя сценами

```xml
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    android:id="@+id/container"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <View
        android:id="@+id/transition_square"
        android:layout_width="@dimen/square_size_normal"
        android:layout_height="@dimen/square_size_normal"
        android:background="@color/square_bg"
        android:layout_gravity="top|left"/>

</FrameLayout>
```

```xml
<FrameLayout
    android:id="@+id/container"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <View
        android:id="@+id/transition_square"
        android:layout_width="@dimen/square_size_expanded"
        android:layout_height="@dimen/square_size_expanded"
        android:background="@color/square_bg"
        android:layout_gravity="top|right"/>

    <TextView
        android:id="@+id/transition_title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="This text fades in."
        android:textAppearance="?android:attr/textAppearanceLarge"/>/>

</FrameLayout>
```

Во второй сцене добавили текст и изменили положение квадрата

Пример с двумя сценами

```
ViewGroup sceneRoot = (ViewGroup) findViewById(R.id.scene_root);

// You can also inflate a generate a Scene from a layout resource file.
final Scene scene2 = Scene.getSceneForLayout(sceneRoot, R.layout.scene3, this);

findViewById(R.id.btn_change_scene).setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // опишем свой аналог AutoTransition
        TransitionSet set = new TransitionSet();
        set.addTransition(new Fade());
        set.addTransition(new ChangeBounds());
        // выполняться они будут одновременно
        set.setOrdering(TransitionSet.ORDERING_TOGETHER);
        // уставим свою длительность анимации
        set.setDuration(500);
        // и изменим Interpolator
        set.setInterpolator(new AccelerateInterpolator());
        TransitionManager.go(scene2, set);
    }
});
```

Минусы

- Минимальная версия Android 4.4
- Даже для промежуточного варианта 4.0
- Если вы не хотите исключить большинство устройств придется использовать сторонние библиотеки реализующие похожее поведение
- Или отказаться от этого

Анимации кадрами

- Описываются в xml файле – объект AnimationDrawable
- Лежит в папке /res/anim
- Корневой объект - <animation-list>
- Описываются кадры и длительность

```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="true">
    <item android:drawable="@drawable/rocket_thrust1" android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust2" android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust3" android:duration="200" />
</animation-list>
```

```
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  ImageView rocketImage = (ImageView) findViewById(R.id.rocket_image);
  rocketImage.setBackgroundResource(R.drawable.rocket_thrust);
  rocketAnimation = (AnimationDrawable) rocketImage.getBackground();
}

public boolean onTouchEvent(MotionEvent event) {
  if (event.getAction() == MotionEvent.ACTION_DOWN) {
    rocketAnimation.start();
    return true;
  }
  return super.onTouchEvent(event);
}
```

Аппаратное ускорение

- Начиная с Android 3.0 отрисовка может поддерживать аппаратное ускорение
- Может быть на уровне
  - Application
  - Activity
  - Window
  - View
- Много операций в младших версиях Android не могут быть отрисованы с поддержкой видеокарты

Разница между программной и аппаратной отрисовкой

- Программная отрисовка
  - Рассчитываем иерархию
  - Рисуем иерархию
- Аппаратная отрисовка
  - Рассчитываем иерархию
  - Компонуем и создаем буферы на отрисовку (индексный, вертексный и пр) чтобы уменьшить количество вызовов draw
  - Рисуем собранные списки на отрисовку

Некоторые советы по оптимизации

- Сокращайте количество элементов для рисования
- Избегайте перерисовок – много объектов в одном месте
- Не создавайте объектов рендера в момент рисования
- Не модифицируйте часто фигуры (размеры и пр объектов)
- Не модифицируйте часто картинки
- Осторожно используйте прозрачку

GIF анимации

- Это боль!
- По умолчанию GIF поддерживается как статическая картинка
- Можно использовать Movie
- Можно вытаскивать из GIF покадрово Bitmap
- Можно использовать WebView
- Все три варианта имеют свои недостатки

ТЕХНОТРЕК

Спасибо за внимание!

Кирилл Филимонов – Kirill.Filimonov@gmail.com

Юрий Береза – ybereza@gmail.com