образование ПОЛИТЕХ одноклассники экосистема

Custom View. Animations. Touches

Дорофеев Юрий

Agenda

Custom view. Что такое measure и layout

Пример: Wave View

Анимации

Обработка тачей

Пример: свайп для удаления

образование ПОЛИТЕХ одноклассники экосистема

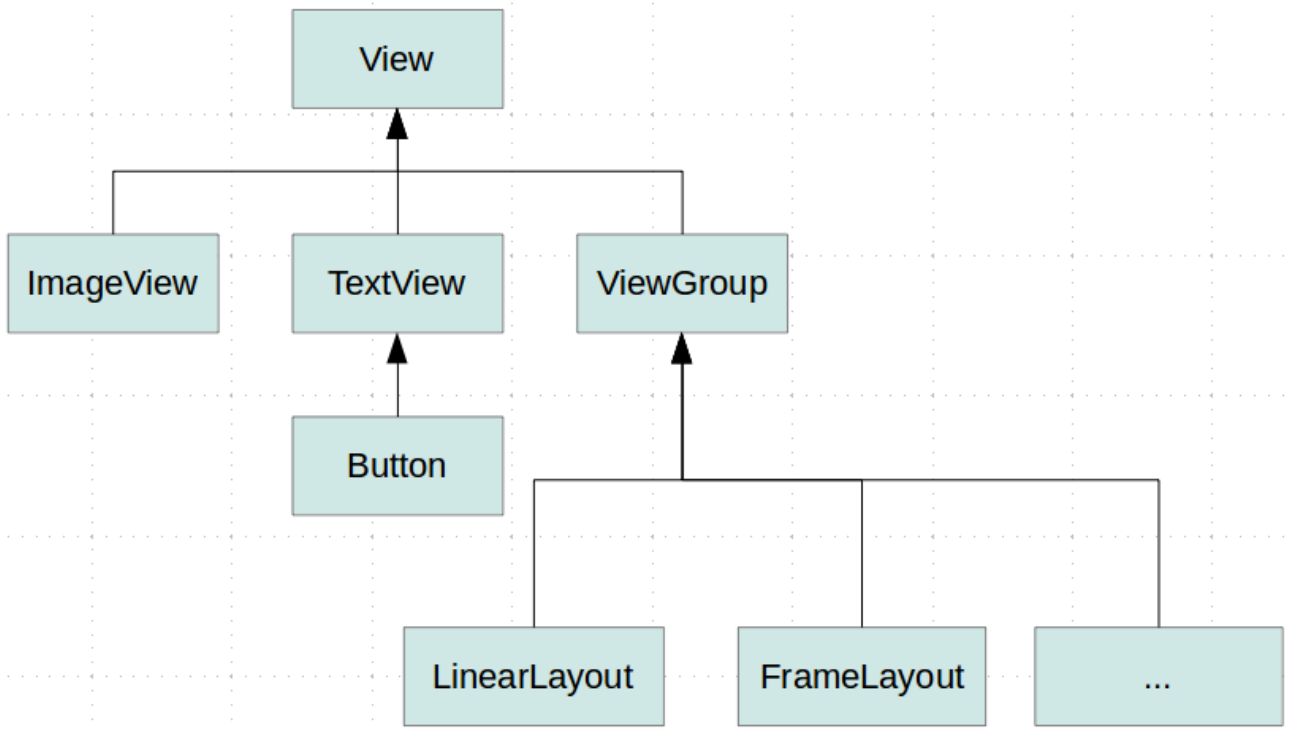
Custom View

1. специфичная отрисовка
2. специфичная обработка жестов
3. оптимизация существующих элементов
4. правка багов в существующем элементе

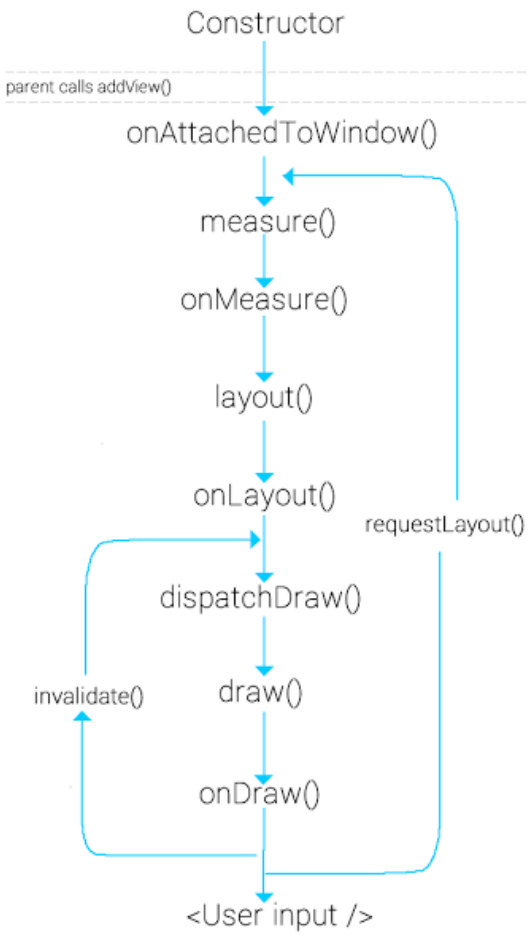
образование ПОЛИТЕХ одноклассники экосистема

НЕ ПИШИТЕ КАСТОМНЫЕ VIEW

Наследники View



Жизненный цикл View



Constructor

Создание View из кода:

```
public CustomView(Context context);
```

Создание View из XML:

```
public CustomView(Context context, @Nullable AttributeSet attrs);
```

Constructor

Создание View из XML со стилем из темы:

```
public CustomView(Context context, @Nullable AttributeSet attrs, int defStyleAttr);
```

Создание View из XML со стилем из темы и/или с ресурсом стиля

```
public CustomView(Context context, @Nullable AttributeSet attrs, int defStyleAttr, int defStyleRes);
```

Последний конструктор добавлен в sdk версии 21

Constructor

Kotlin

```
class CustomView @JvmOverloads constructor(
    context: Context,
    attrs: AttributeSet? = null,
    defStyleAttr: Int = 0,
) : View(context, attrs, defStyleAttr)
```

onAttachedToWindow

После того как родитель View вызовет метод **addView(View)**, наш **View** будет прикреплён к окну.
На этой стадии наш **View**-компонент попадает в иерархию родителя

onMeasure

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

int spec = mode | size

MeasureSpec.getMode(widthMeasureSpec)
MeasureSpec.getSize(widthMeasureSpec)
```

onMeasure

mode может принимать следующие значения:

MeasureSpec.EXACTLY;
MeasureSpec.AT_MOST;
MeasureSpec.UNSPECIFIED;

onMeasure

```
if (widthMode === MeasureSpec.EXACTLY) {
    width = widthSize
} else if (widthMode === MeasureSpec.AT_MOST) {
    width = Math.min(wrapWidth, widthSize)
} else {
    width = wrapWidth
}
```

onMeasure

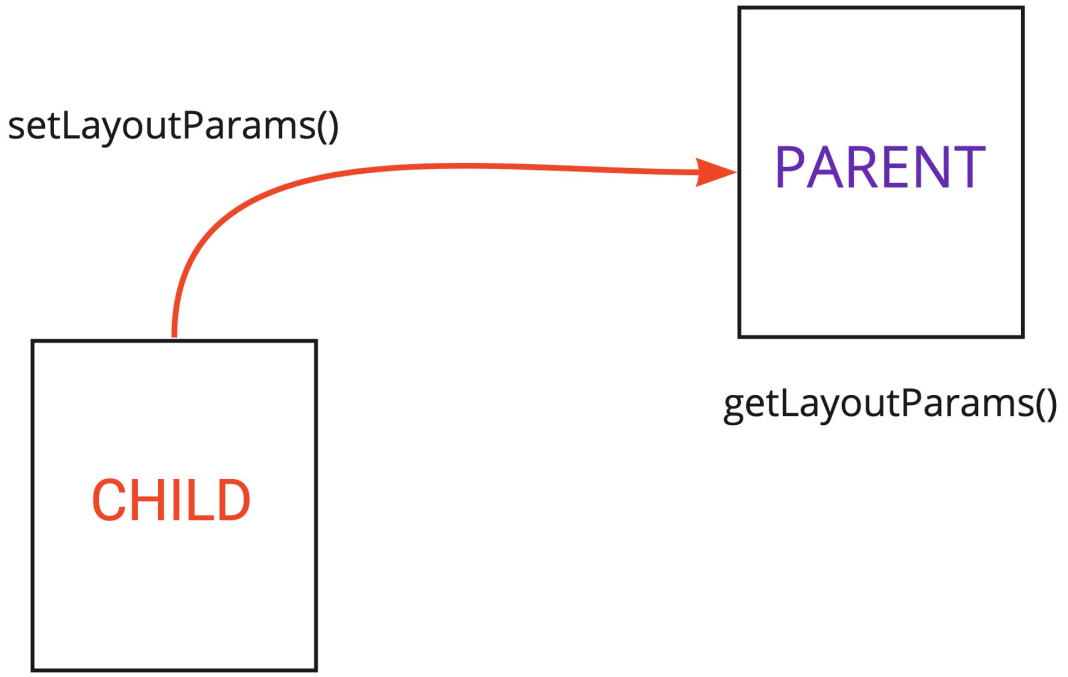
```
if (widthMode === MeasureSpec.EXACTLY) {
    width = widthSize
} else if (widthMode === MeasureSpec.AT_MOST) {
    width = Math.min(wrapWidth, widthSize)
} else {
    width = wrapWidth
}

public static int resolveSize(int size, int measureSpec)
```

Не забудь!

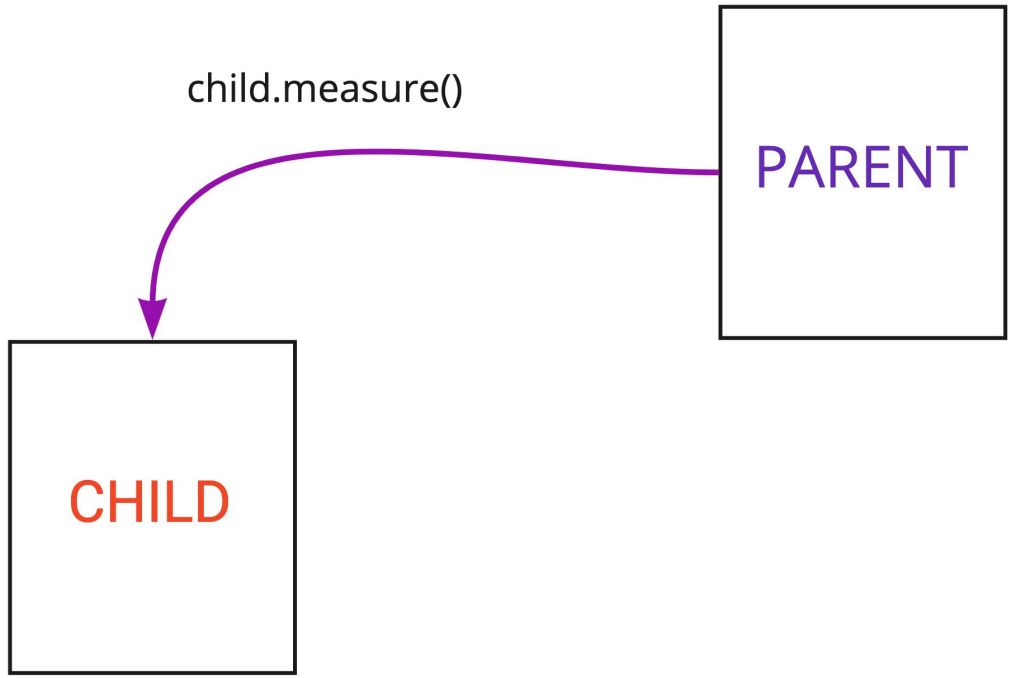
```
protected final void setMeasuredDimension(int measuredWidth, int measuredHeight)
```

образование ПОЛИТЕХ одноклассники экосистема

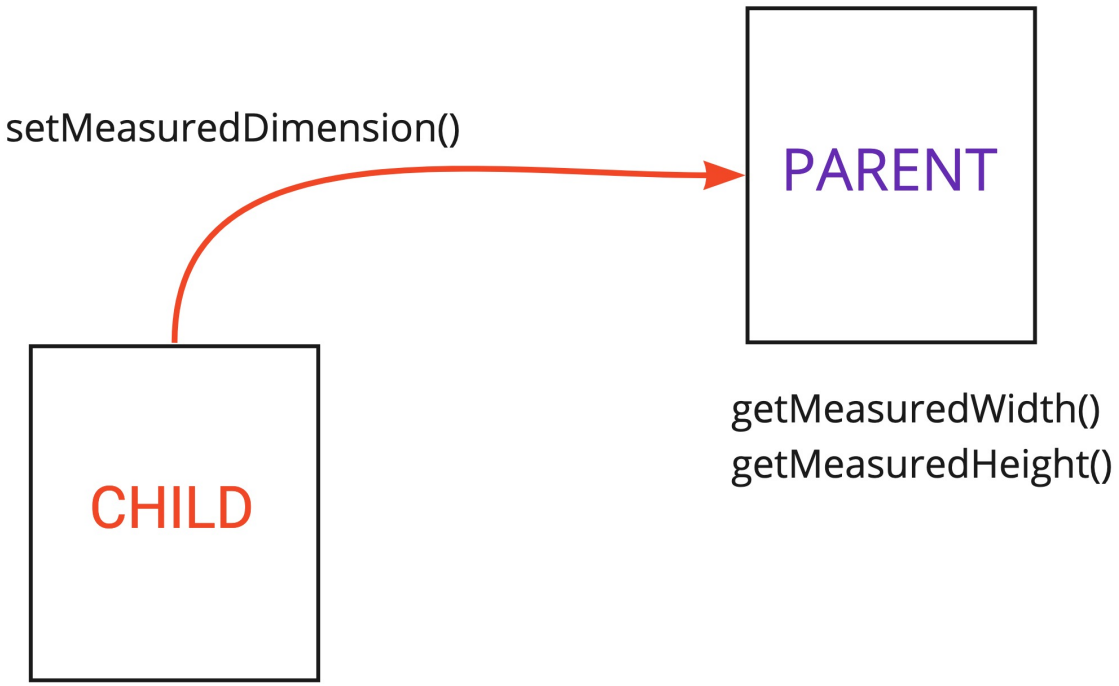
Measure и layout



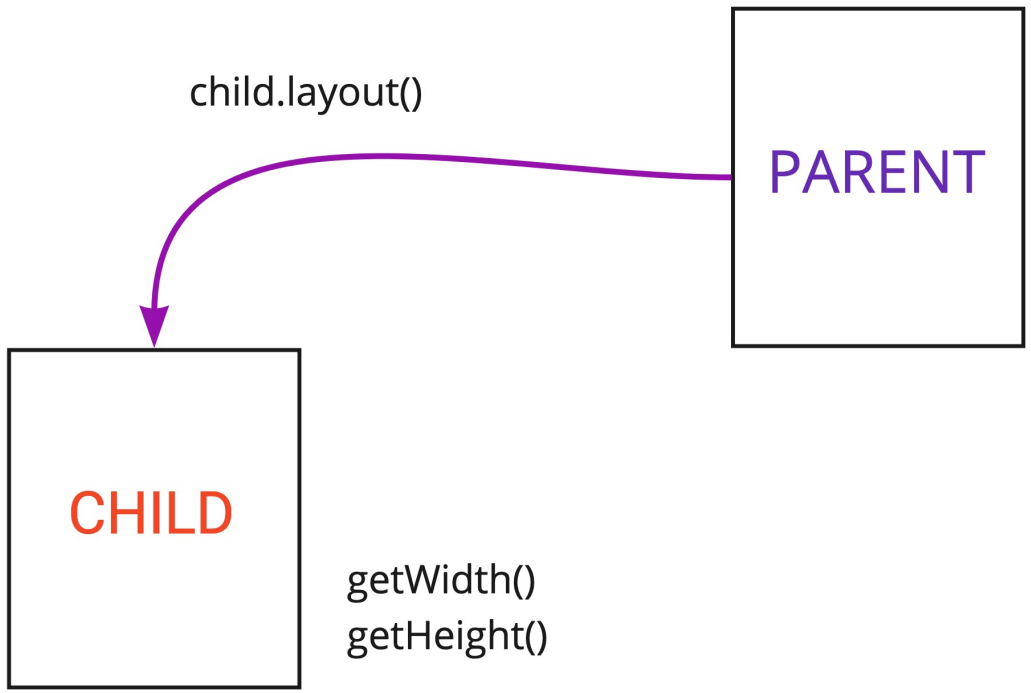
Measure и layout



Measure и layout



Measure и layout



onLayout

Этот метод позволяет присваивать позицию и размер дочерним элементам **ViewGroup**. В случае, если мы наследовались от **View**, нам не нужно переопределять этот метод.

onDraw

```
protected void onDraw(Canvas canvas)
```

onDraw

```
protected void onDraw(Canvas canvas)
```

- Не создавать объекты!

onDraw

```
protected void onDraw(Canvas canvas)
```

- Не создавать объекты!
- Не делать сложные операции!

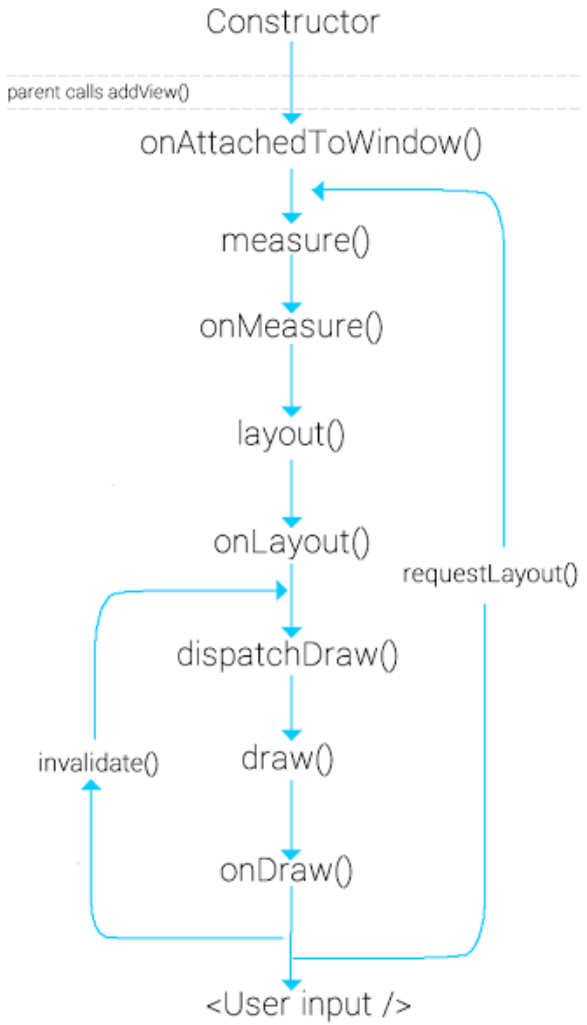
onDraw

```
protected void onDraw(Canvas canvas)
```

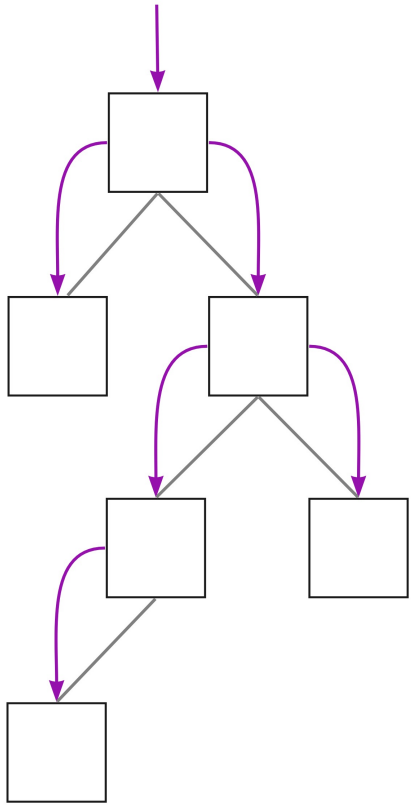
- Не создавать объекты!
- Не делать сложные операции!
- Не пишите кастомные View!

Обновление View

- invalidate();
- requestLayout()



Measure и layout



Шпаргалка

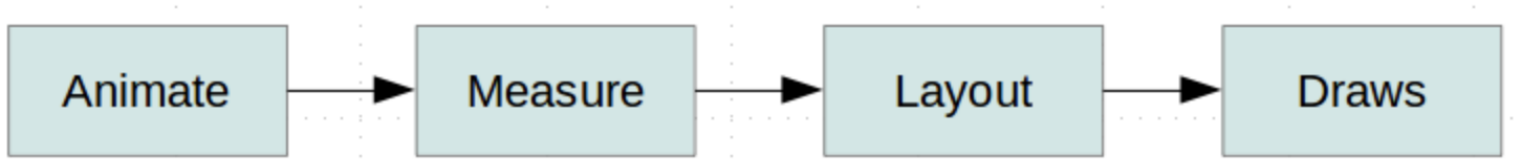
- Если есть возможность обойтись без кастомных View, воспользуйтесь ей;
- Внимательно отнеситесь к расчету размера View, это поможет избежать множества проблем;
- Во время отрисовки не создавайте лишних объектов и не делайте лишней работы;
- Вызывайте метод invalidate только когда это действительно нужно;
- Отрисовка происходит в px, но оперировать нужно в терминах dp и sp;
- Используйте методы, которые могут упростить ваши расчеты, такие как resolveSize;

Пример: Wave View

Анимации

образование ПОЛИТЕХ одноклассники экосистема

Жизненный цикл



Способы анимации

- View animation;
- Drawable animation;
- ValueAnimator;
- ObjectAnimator;
- ViewPropertyAnimator;
- Layout transition;
- Transition Framework;
- Dynamic Animation.

View animation

- Устарел!
- Был до Android 2.3;
- alpha, rotate, scale, translate;

Drawable animation

Drawable animation

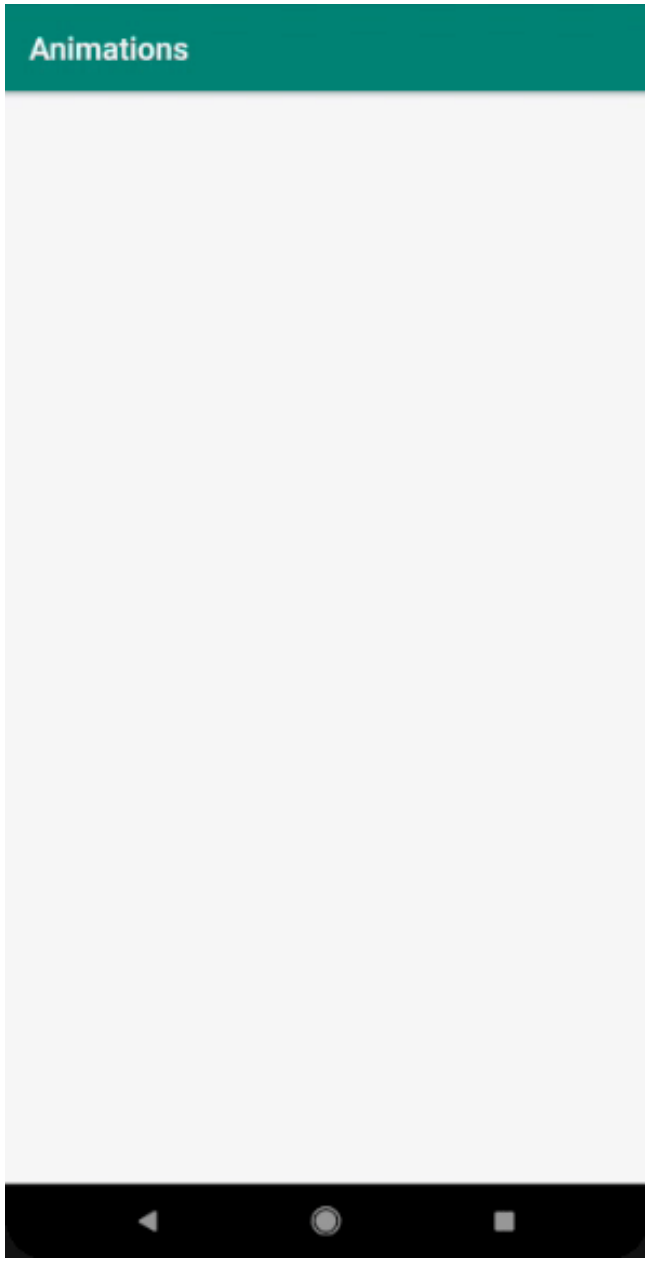
```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="true">
    <item android:drawable="@drawable/rocket_thrust1" android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust2" android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust3"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

ValueAnimator

- Появился в Android 3.0
- Базовый движок

ValueAnimator

```
private fun animate() {
    val alphaAnimator = ValueAnimator.ofFloat(0f, 1f)
    alphaAnimator.addUpdateListener { animation -> circle.alpha = animation.animatedValue as Float }

    val yAnimator = ValueAnimator.ofFloat(0f, (root.height - circle.height).toFloat())
    yAnimator.addUpdateListener { circle.y = yAnimator.animatedValue as Float }

    val set = AnimatorSet()
    set.playTogether(alphaAnimator, yAnimator)
    set.duration = ANIMATION_DURATION
    set.start()
}
```



Движок аниматора

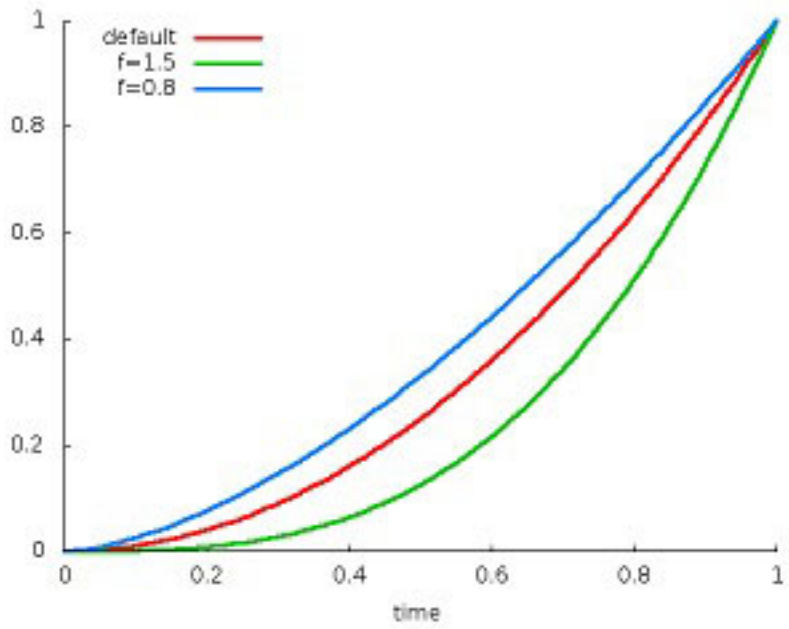
- Время в **Animator** представляется как значение от 0 до 1. Время обновления экрана в Android это 16ms. Допустим, анимация длится 160 мс, тогда, для отрисовки нам понадобится 10 фреймов. Для примера рассмотрим что происходит во время отрисовки 5го фрейма:

Движок аниматора

Сначала значение 0.5 попадает в **TimeInterpolator**, это функция, которая показывает как должна изменяться скорость анимации. Можно, например, сделать что бы анимация разгонялась со временем или наоборот тормозила. Воспользуемся **AccelerateInterpolator**:

f(x) = x^2

0.5^2 = 0.25



Движок аниматора

Рассчитанное значение подается на вход **TypeEvaluator**. **TypeEvaluator** определяет как должен меняться объект в процессе анимации относительно времени. В простых случаях это разница конечного и начального значения умноженная на время:

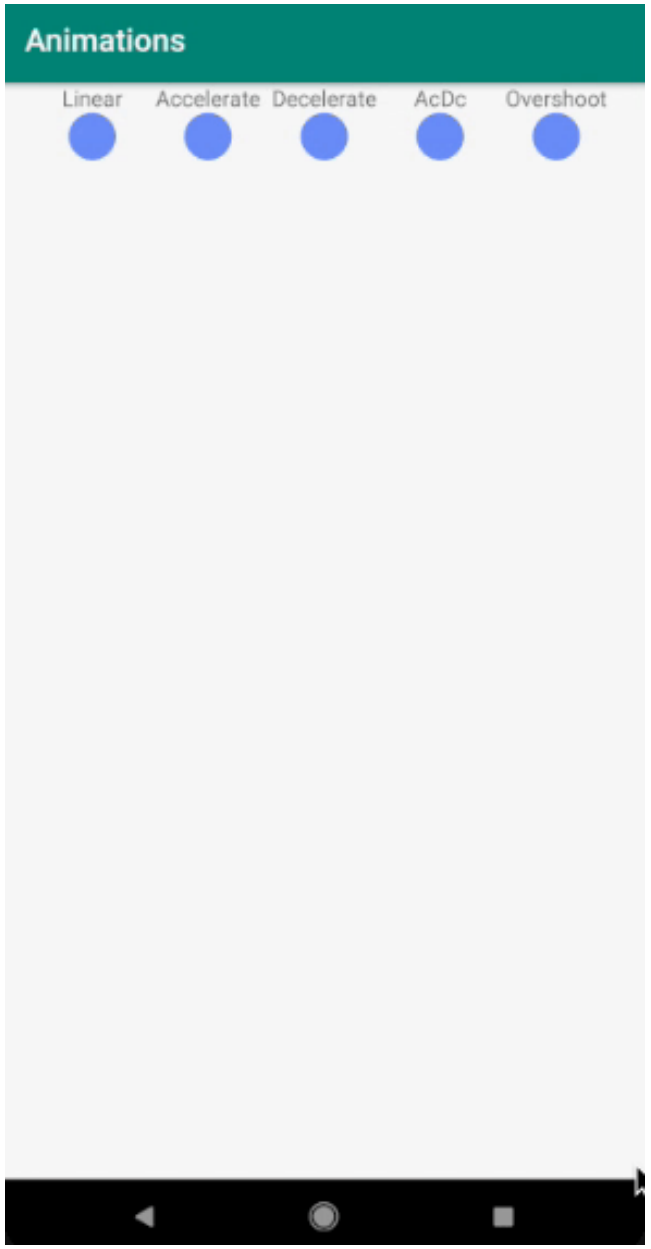
(x2 - x1) * t

(255-0) * 0.25 = 63.75

Получившееся значение записывается в **Animator** и передается в **Listener**.

Интерполяторы

- **AccelerateInterpolator**. Определяет функцию с увеличивающейся скоростью. (Исчезновение)
- **DecelerateInterpolator**. Определяет функцию с уменьшающейся скоростью. (Появление)
- **AccelerateDecelerateInterpolator**. Определяет функцию, скорость которой сначала увеличивается, а потом снижается.



Интерполяторы

В Android 5 появились новые интерполяторы, аналогичные перечисленным выше. Считается, что они работают более плавно и
естественно: **FastOutLinearInInterpolator**, **LinearOutSlowInInterpolator**, **FastOutSlowInInterpolator**.

ObjectAnimator

ObjectAnimator является расширением **ValueAnimator**. Все свойства **ValueAnimator** также применимы и к **ObjectAnimator**. Главным отличием является то, что нам нет необходимости задавать Listener, вместо этого в **ObjectAnimator** представлено понятие **Property**, которое отвечает за изменяемый параметр.

```
val yAnimator: ObjectAnimator = ObjectAnimator.ofFloat(circle, Y, circle.y, root.height - circle.height)
```

Property

```
public static final Property<View, Float> Y = new FloatProperty<View>("y") {
    @Override
    public void setValue(View object, float value) {
        object.setY(value);
    }

    @Override
    public Float get(View object) {
        return object.getY();
    }
};
```

Property

с помощью наследника Property:

ObjectAnimator.ofFloat(circle, Y, circle.y, root.height - circle.height)

с помощью строки:

ObjectAnimator.ofFloat(circle, “y”, circle.y, root.height - circle.height)

Property

- ALPHA;
- TRANSLATION_X;
- TRANSLATION_Y;
- TRANSLATION_Z;
- X;
- Y;
- Z;
- ROTATION;
- ROTATION_X;
- ROTATION_Y;
- SCALE_X;
- SCALE_Y.

ViewPropertyAnimator

- **ViewPorpertyAnimator** также работает на основе **ValueAnimator**. Главный плюс **ViewPorpertyAnimator** состоит в том, что он предоставляет удобный API для анимации **View**. В случае анимирования нескольких значений, он может быть незначительно быстрее **ObjectAnimator**. Минусом является то, что нам нельзя анимировать какие-то свои параметры, можно использовать только те, которые есть в стандартном наборе.

```
val yAnimator: ViewPropertyAnimator = circle.animate()
                    .x(root.height - circle.height)
                    .setDuration(ANIMATION_DURATION)
                    .setInterpolator(interpolator)
```

Какой аниматор выбрать?

- **ViewAnimation**. Устаревший. Лучше не использовать;
- **DrawableAnimation**. Использовать только в очень сложных случаях;
- **ValueAnimator**. Стоит использовать только в тех случаях, когда нам не удается написать Property;
- **ObjectAnimator**. Стоит использовать по возможности всегда;
- **ViewPropertyAnimator**. Стоит использовать для простых случаев.

Завершение анимации

- view.clearAnimation(). Для **View animation;**
- animator.cancel(). Для **ValueAnimator** и **ObjectAnimator;**
- view.animate().cancel(). Для **ViewPropertyAnimator.**

- Activity.onStop();
- Fragment.onStop();
- View.onDetachFromWidow().

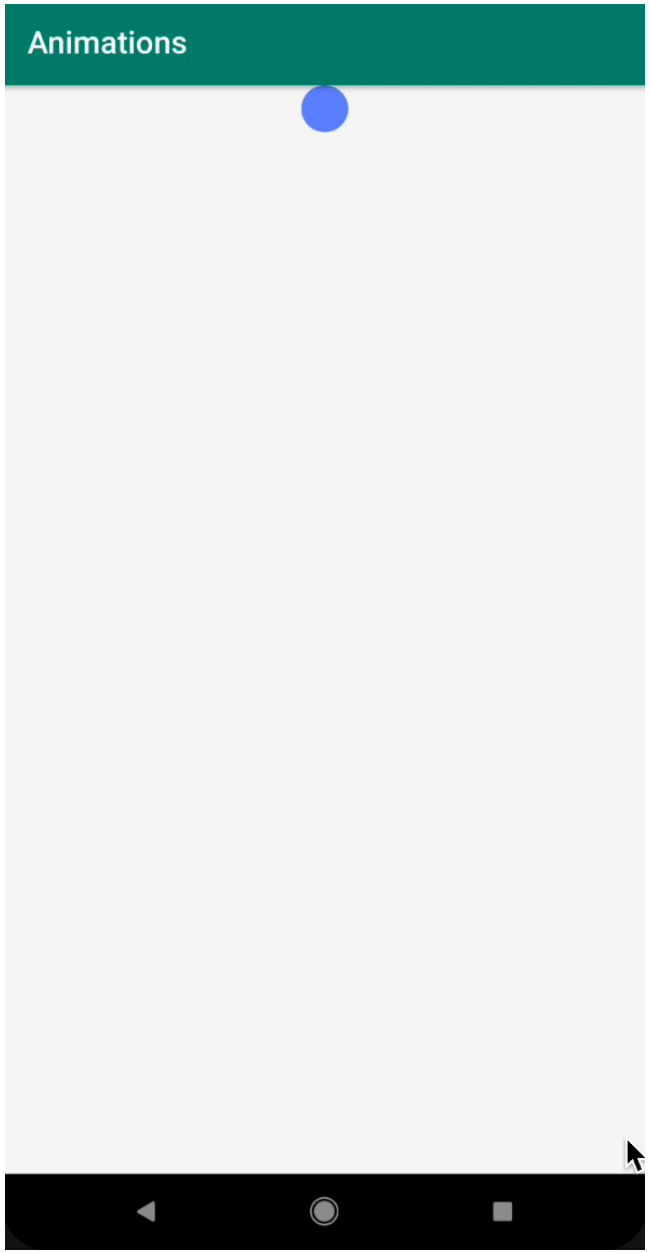
LayoutTransition

LayoutTransition(флаг animateLayoutChanges) работает на основе **ValueAnimator**. Позволяет при изменениях внутри дочерних **View** анимировать родительский **View**.

Подходит для простых случаев:

- Появление;
- Исчезновение;
- Изменение размеров.

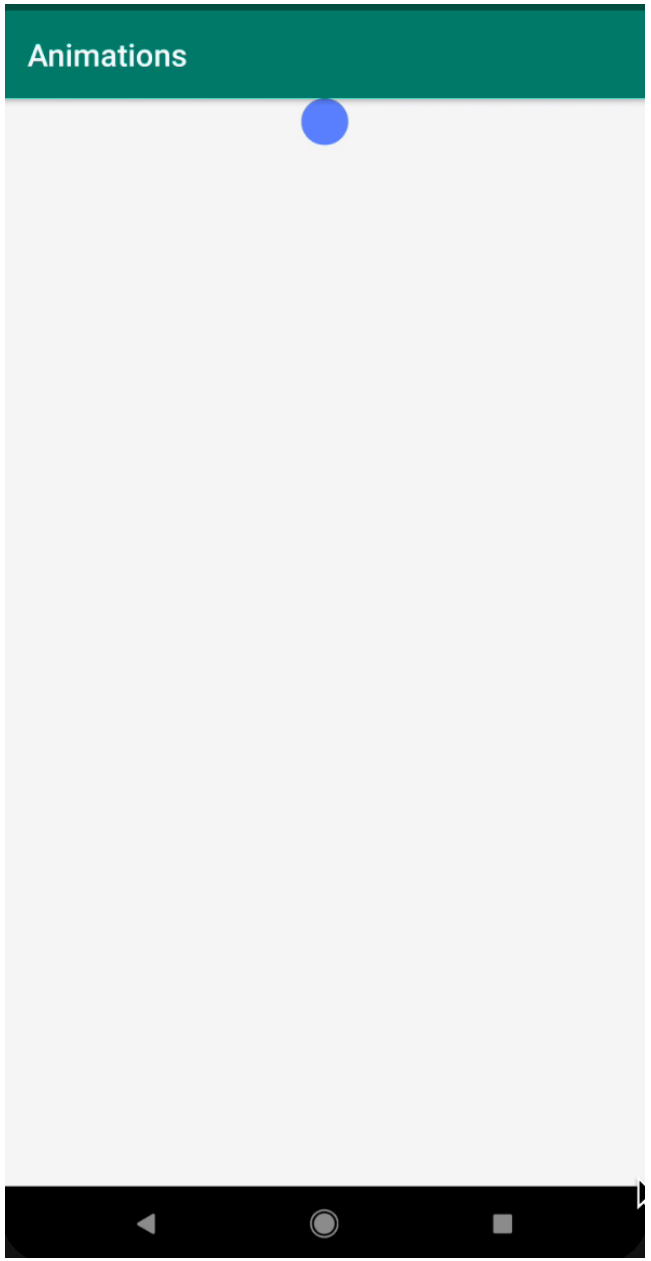
```
root.setLayoutTransition(new LayoutTransition());
```



Transition Framework

Кроме базовых анимаций есть несколько необычных, например **Slide**. Этот способ анимации заставляет **View** "прилететь" в заданную позицию:

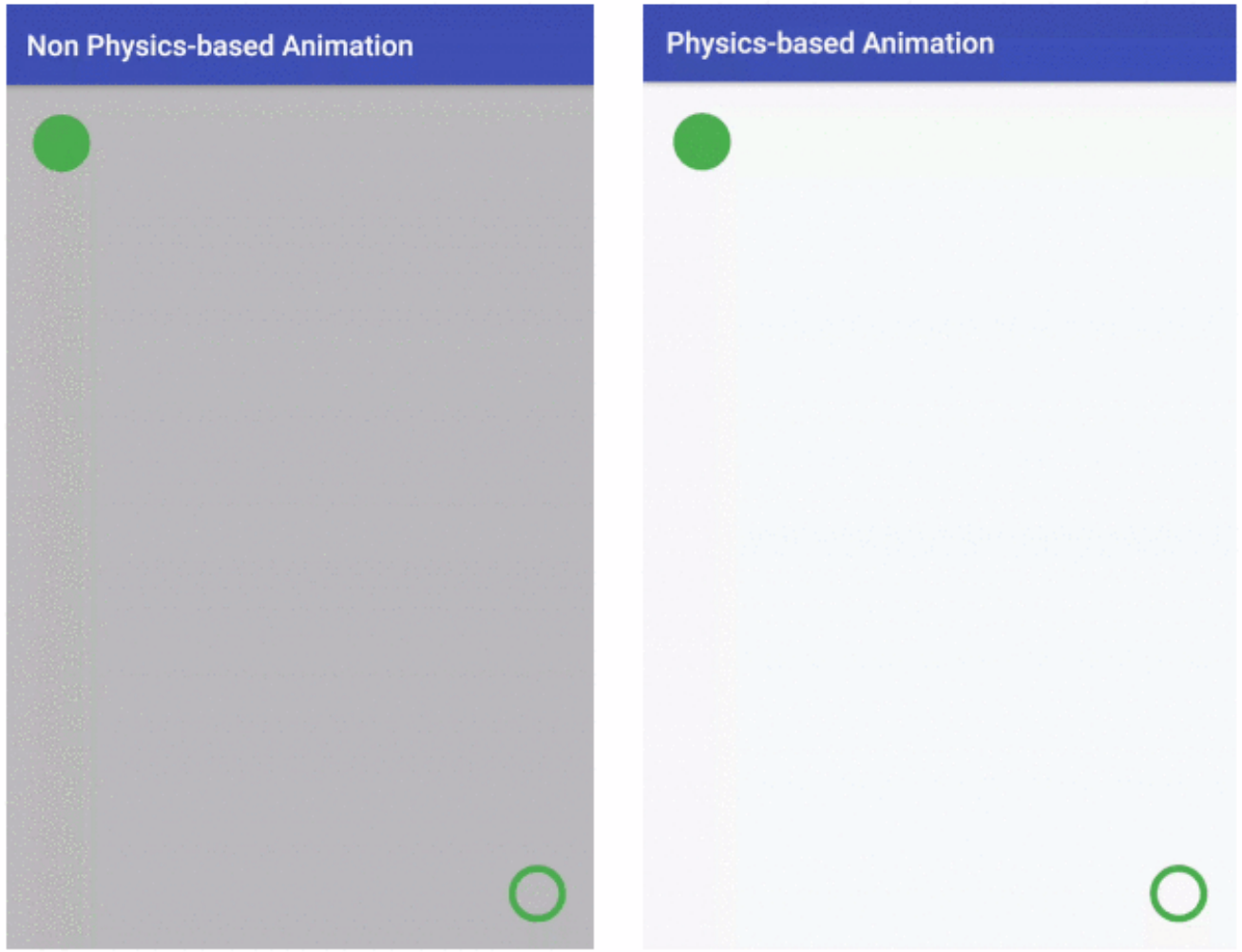
```
TransitionSet transitionSet = new TransitionSet()
.addTransition(new Fade())
.addTransition(new Slide();

TransitionManager.beginDelayedTransition(root, transitionSet);
circle2.setVisibility(View.VISIBLE);
```



Что лучше?

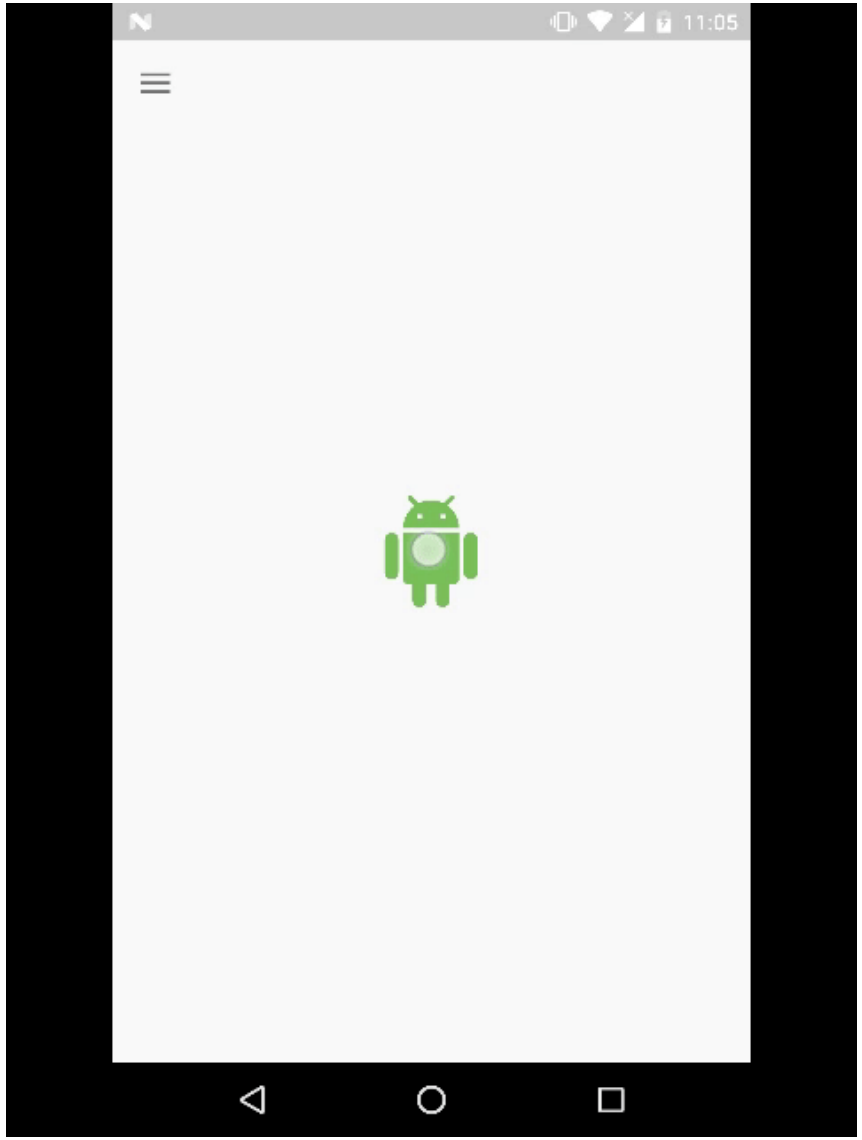
LayoutTransition стоит использовать для простых иерархий, где анимируется не большое количество элементов и/или для старых версий Android. **Transition Framework** следует использовать для более сложных случаях и/или для новых версий Android.

Dynamic Animation



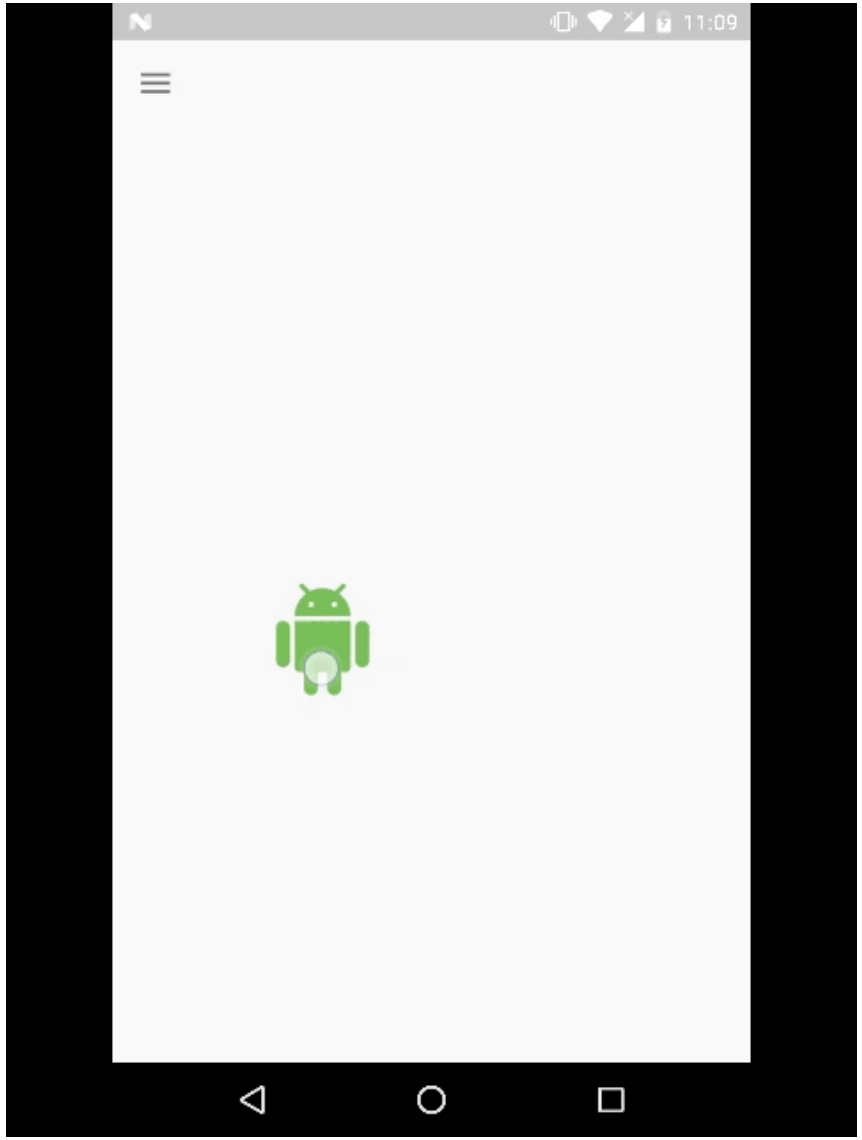
FlingAnimation

FlingAnimation. Предназначена для тех случаев, когда пользователь своими жестами инициирует какую-то анимацию, например свайп: когда пользователь поднимает палец, элемент должен сдвинуться примерно с той же самой скоростью, с которой пользователь двигал палец.



SpringAnimation

SpringAnimation. Анимация отмены действия или возврата к начальному состоянию. Чем-то похожа на пружину.



Прочие анимации

Кроме анимаций, которые могут применяться к любым **View**, есть анимации, которые предназначены для каких-то определенных ситуаций. Например:

- **ItemAnimator**. Этот аниматор работает только в **RecyclerView**
- **AnimatedVectorDrawable**. Аниматор для работы с векторными изображениями;
- **Activity Transition**. Способ анимации между **Activity**;
- **Fragment Transition**. Способ анимации между фрагментами.

Android постоянно обновляется и вместе с тем добавляются новые способы анимации объектов. С умом подойдите в выбору способа анимации и следите за обновлениями.

Обработка тачей

образование ПОЛИТЕХ одноклассники экосистема

Обработка тачей

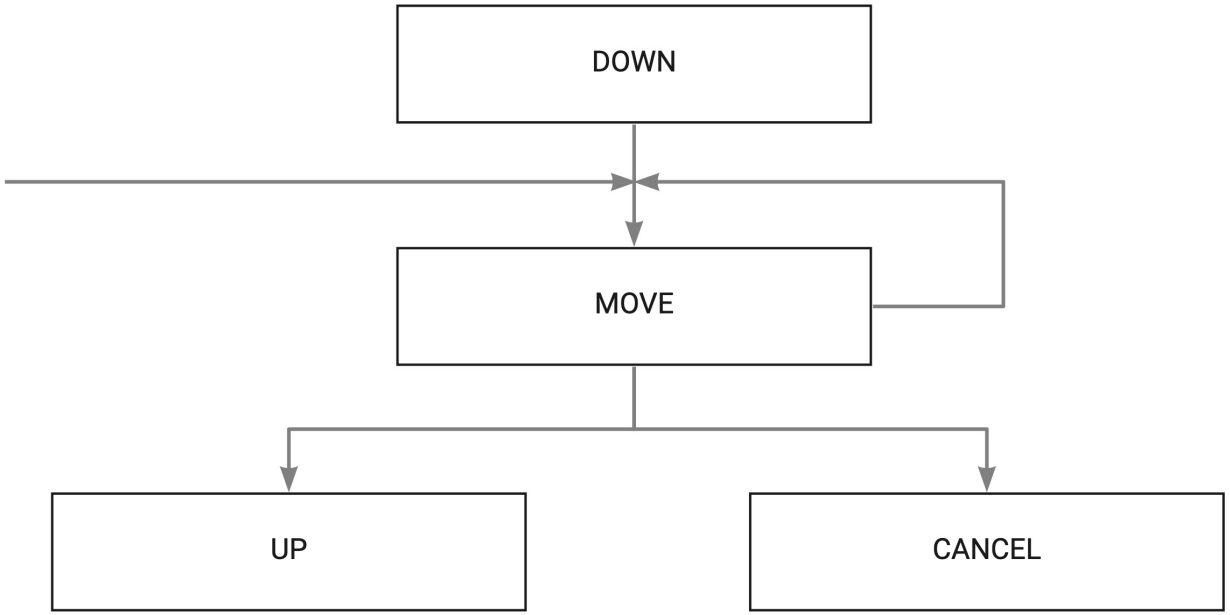
MotionEvent - это событие касания экрана. **MotionEvent** работает с любыми способами ввода:

- палец;
- стилус;
- мышь.

В рамках этого урока мы будет рассматривать только касания пальцем.

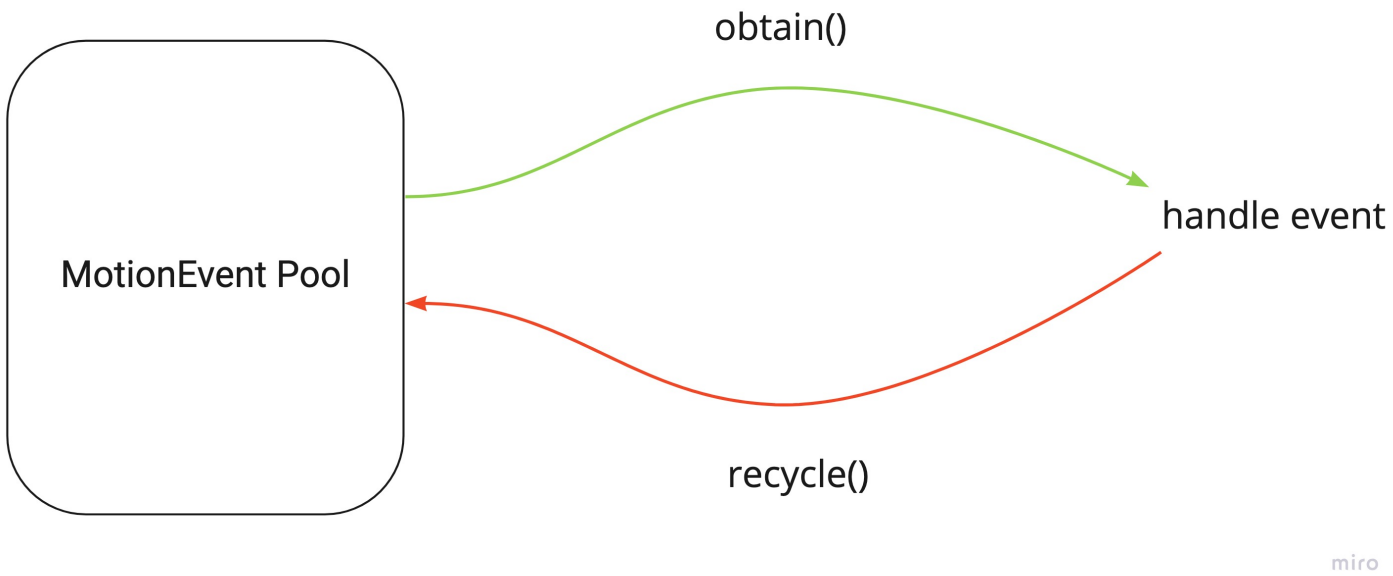
образование ПОЛИТЕХ одноклассники экосистема

Обработка тачей



образование ПОЛИТЕХ одноклассники экосистема

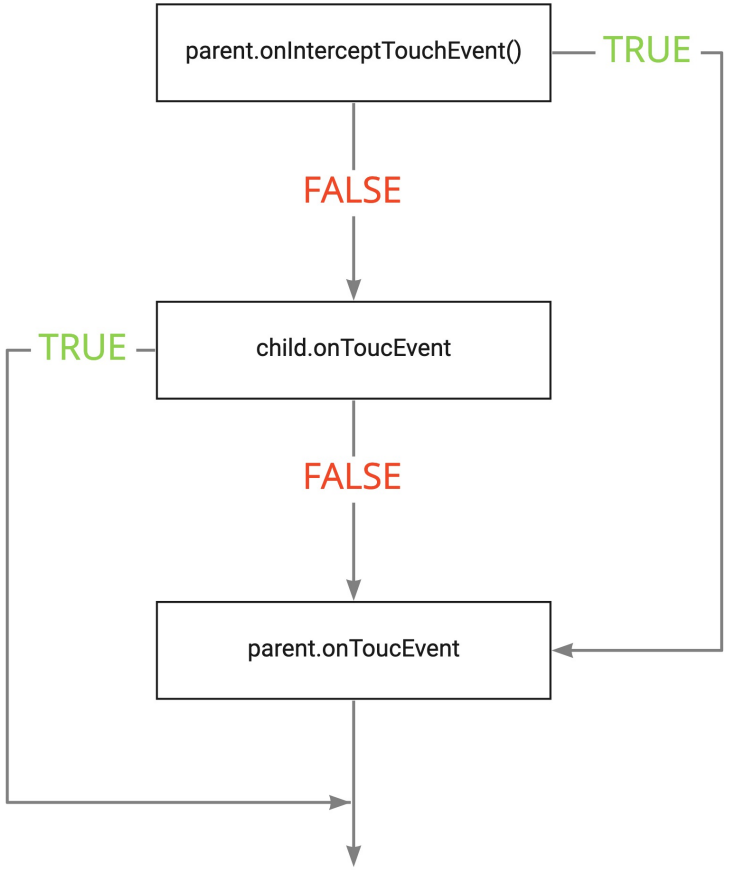
Обработка тачей



miro

образование ПОЛИТЕХ одноклассники экосистема

Диспетчеризация нажатий



Multi touch

Multi touch это события множественного касания. Когда человек использует несколько пальцев одновременно. Для Multi тачей вводятся два новых события:

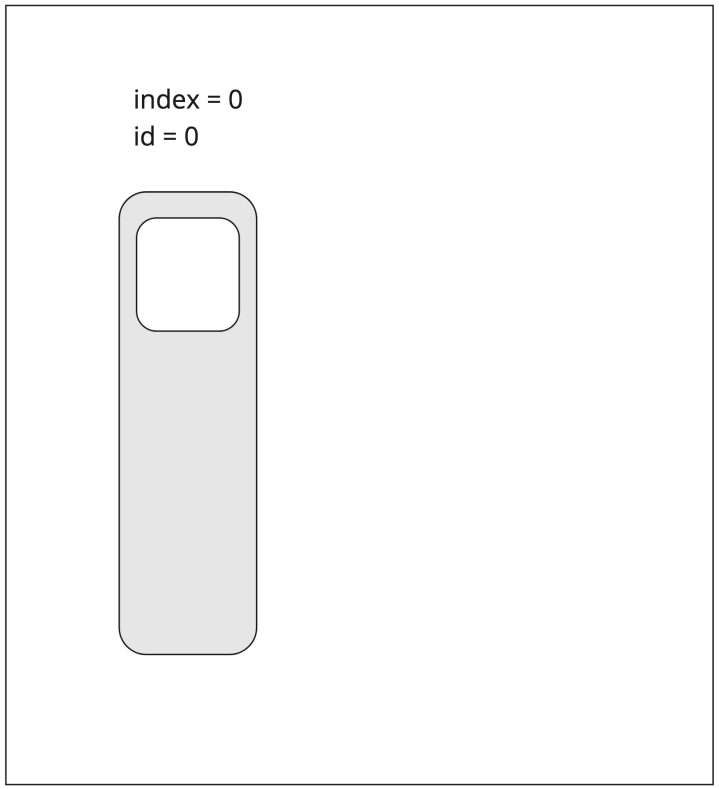
- **POINTER_DOWN**. Прикосновение пальцем;
- **POINTER_UP**. Поднятие пальца.

Каждому пальцу назначаются два параметра:

- **index**. Порядковый номер пальца. Не привязан к пальцу, один палец может иметь разные индексы в течение одного касания.
- **id**. Идентификатор пальца. Привязан к конкретному пальцу от начала и до конца касания

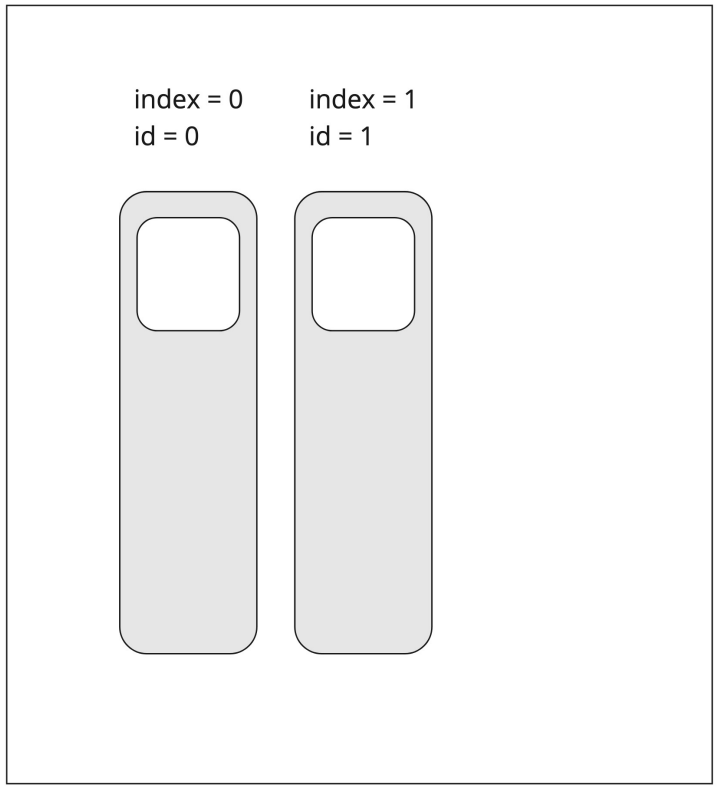
образование ПОЛИТЕХ одноклассники экосистема

Multi touch

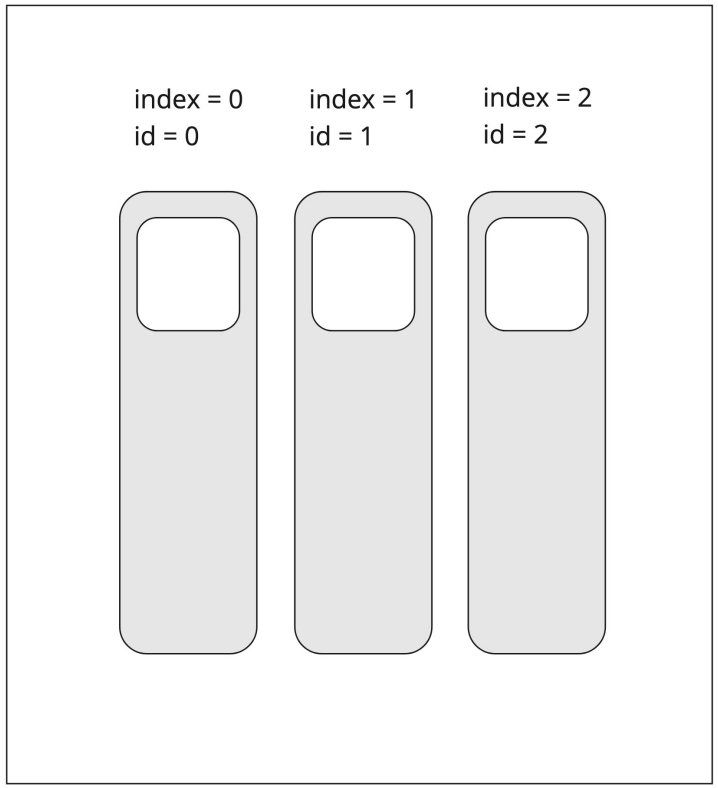


образование ПОЛИТЕХ одноклассники экосистема

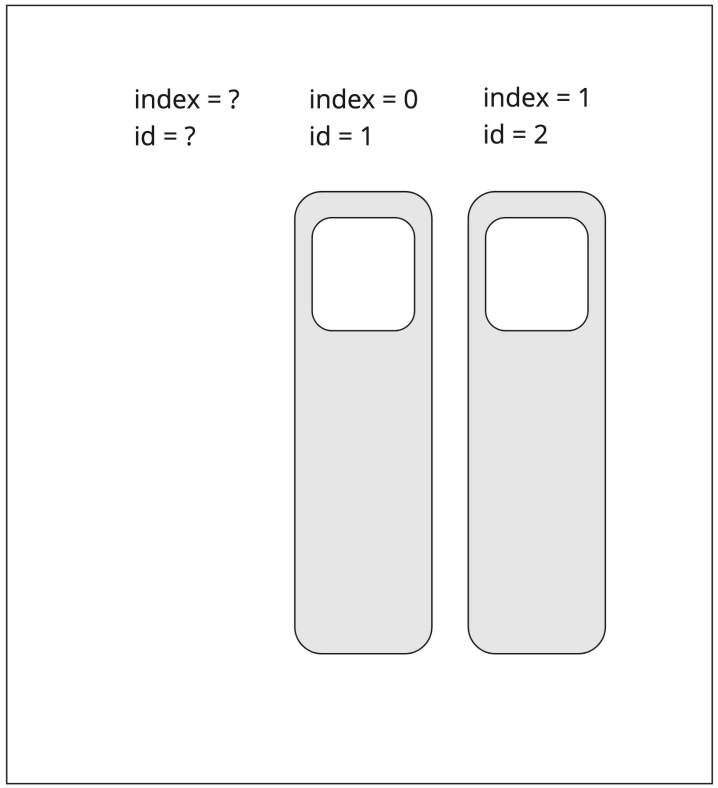
Multi touch



Multi touch

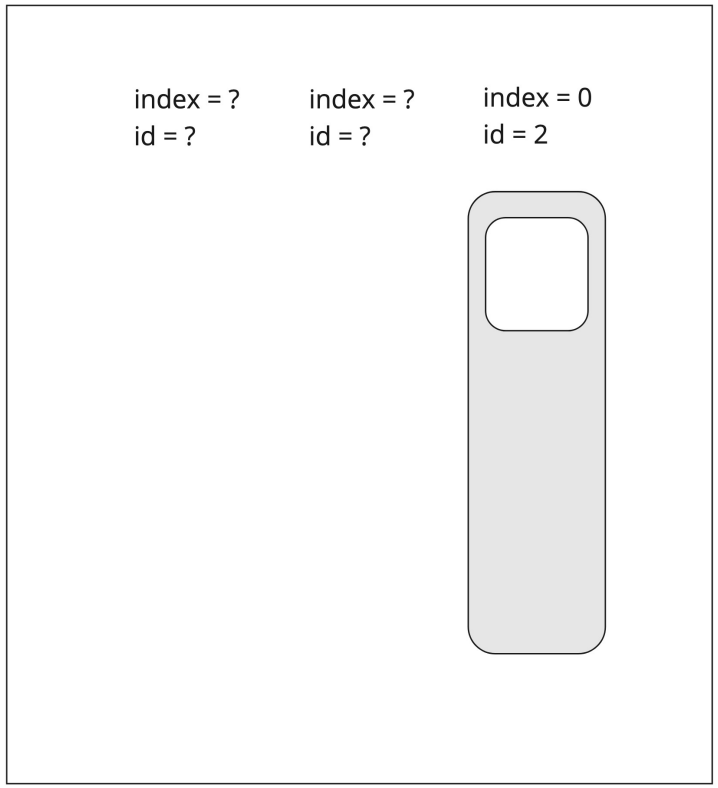


Multi touch



образование ПОЛИТЕХ одноклассники экосистема

Multi touch



Отслеживание скорости

Для того, что бы отслеживать скорость перемещения пальцев в Android существует класс **VelocityTracker**. Работа с ним чем-то похожа на работу с **MotionEvent** - объекты тоже находятся в пуле и для получения **VelocityTracker** необходимо вызвать метод **obtain**:

```
vt = VelocityTracker.obtain()
```

Что бы **VelocityTracker** понимал как пользователь передвигает пальцы, необходимо все **MotionEvent** передавать в метод **addMovement**:

```
vt.addMovement(event)
```

Отслеживание скорости

Когда мы захотим получить текущую скорость перемещения, нужно вызвать метод:

```
vt.computeCurrentVelocity(VELOCITY_UNITS)
```

Что такое **VELOCITY_UNITS**? Это то количество пикселей, относительно которых будет считаться скорость. Например, если **VELOCITY_UNITS = 1000**, то скорость будет измеряться в том, сколько тысяч пикселей в секунду проходит палец на экране.

После работы, нужно не забывать утилизировать объект вызовом:

```
vt.recycle()
```

Отслеживание скорости

- OnDown
- OnFling
- OnLongPress
- OnScroll
- onShowPress
- onSingleTapUp
- ...

Пример: свайп для удаления

Спасибо!

образование

ПОЛИТЕХ

одноклассники
экосистема