OTUS

Android Developer

Animations

otus.ru


```
если (видно &
    слышно &
    записьИдёт) {
   чат.написать(“+”)
} иначе {
   чат.написать(“$чтоНеТак”)
}
```

Тема вебинара

Android
Анимации

Александр Аникин

Руководитель команды Андроид-разработки

- Пришёл в IT в 34 года из маркетинга;
- Опыт разработки под Android — 9 лет;
- Опыт преподавания — 8 лет.

Правила вебинара

Активно
участвуем

Off-topic обсуждаем в
канале группы

Задаем вопрос
в чат

Вопросы вижу в чате,
могу ответить не сразу

Условные обозначения

Индивидуально

Время, необходимое
на активность

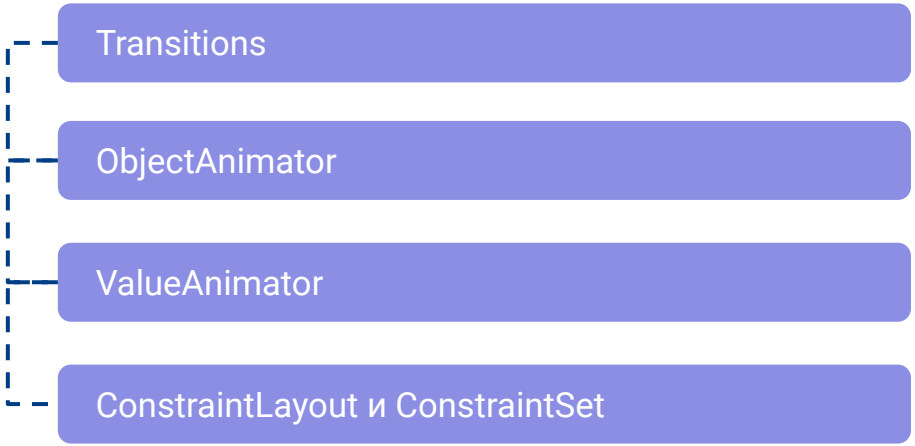
Пишем в чат

Говорим голосом

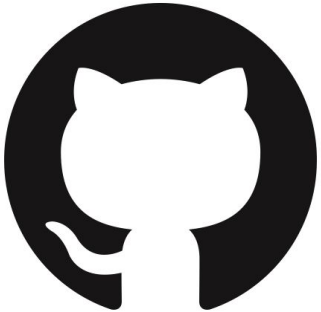
Документ

Ответьте себе или
задайте вопрос

Маршрут вебинара

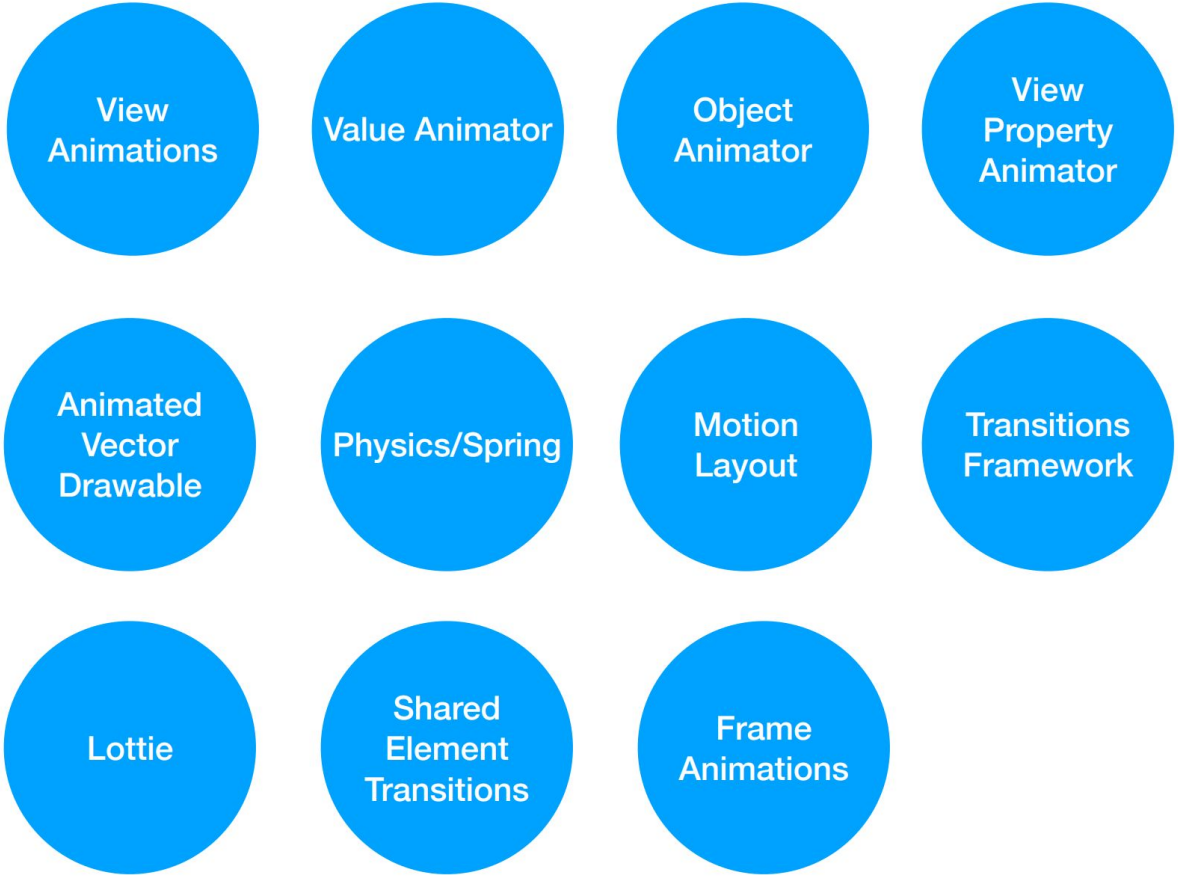


Репозиторий к занятию



https://github.com/Android-Developer-Basic/AnimationsWebinar

Обзор


View Animations
Value Animator
Object Animator
View Property Animator
Animated Vector Drawable
Physics/Spring
Motion Layout
Transitions Framework
Lottie
Shared Element Transitions
Frame Animations

Transitions

TransitionManager

«Движение передаёт смысл. Объекты представляются пользователю, не ломая последовательности его положительного опыта взаимодействия с приложением, даже если эти объекты трансформируются или перегруппировываются. Движение в мире материального дизайна используется для описания пространственных связей объектов, функциональности и намерений вкупе с красотой и плавностью» (Material Design guidelines).

TransitionManager

`TransitionManager` принимает в качестве аргумента какой-то ViewGroup и автоматически (плавно!) меняет видимость текста в нём, при этом сдвигая кнопку выше и давая место появившемуся тексту. Более того, анимация автоматически прерывается и запускается в обратную сторону, если вы нажали на кнопку в процессе анимации. И всё это делается без вашего участия. Виды Transition можно менять: они определяются вторым аргументом для функции `beginDelayedTransition`

1. `ChangeBounds` меняет расположение элемента и его размеры, двигает нашу кнопку по умолчанию.
2. `Fade` — медленное проявление и затухание элемента. Применяется для отображения нашего текста по умолчанию.
3. `TransitionSet` — это набор других Transitions, таких как `ChangeBounds` или `Fade`.
4. `AutoTransition` — TransitionSet, содержащий `Fade out`, `ChangeBounds` и `Fade` в таком порядке. То есть сначала исчезают view, которые должны исчезнуть, затем изменяется расположение остальных view, затем появляются новые. `AutoTransition` применяется по умолчанию, если вы не передаёте никаких параметров в качестве второго аргумента в функцию `beginDelayedTransition`

TransitionManager

Примеры анимаций

```
TransitionSet().apply {
    ordering = TransitionSet.ORDERING_SEQUENTIAL
    addTransition(ChangeBounds())
    addTransition(Fade( Fade.OUT))
}
```

```
TransitionManager.beginDelayedTransition( binding.transitionsContainer, Slide(Gravity.END))
```

TransitionManager

Увеличение/уменьшение

```
setOnClickListener {
    isExpanded = !isExpanded
    TransitionManager.beginDelayedTransition(
        binding.transitionsContainer, TransitionSet()
            .addTransition(ChangeBounds())
            .addTransition(ChangeImageTransform())
    )

    val params: ViewGroup.LayoutParams = layoutParams
    params.height = if (isExpanded) ViewGroup.LayoutParams.MATCH_PARENT else
ViewGroup.LayoutParams.WRAP_CONTENT
    layoutParams = params
    scaleType = if (isExpanded) ImageView.ScaleType.CENTER_CROP else
ImageView.ScaleType.FIT_CENTER
}
```

Вопросы?

ObjectAnimator +
View.animate

ObjectAnimator + View.animate

`ObjectAnimator` занимается анимациями объектов, а метод `animate` — изменением свойств объектов.
С помощью первого виджеты можно двигать, с помощью второго — менять их цвет, прозрачность и т. д.

Последовательно сверху вниз в самых простых контейнерах мы размещаем:

- задний фон;
- фон, который будет менять прозрачность;
- контейнер для опции 1;
- контейнер для опции 2;
- FAB;
- иконку «+», которую мы будем анимировать.

Обратите внимание на атрибуты `elevation` у FAB и иконки. Они должны быть такими, чтобы FAB не находился выше иконки и не перекрывал её.

ObjectAnimator + View.animate

Несмотря на многословность кода, всё достаточно просто. Метод `setInitialState`задаёт начальные параметры для наших view:

- никакого затенения нет;
- опции сделаны прозрачными и некликабельными.

Что мы делаем:

1. Вызываем метод `ObjectAnimator ofFloat`, который позволяет изменять состояние виджета от одного значения (во `float`) до другого.
2. Передаём в этот метод иконку с параметром `"rotation"`. В следующем параметре указываем, на сколько градусов по часовой стрелке мы хотим провернуть иконку;
3. Передаём контейнер с опциями и параметрами `"translationY"`— это движение по вертикали вверх;
4. Изменяем свойства контейнера с помощью `animate`: меняем прозрачность 0 на 1 с длительностью в 300 миллисекунд и вешаем слушатель конца анимации. Когда анимация закончится, мы сделаем контейнеры кликабельными и повесим соответствующие слушатели нажатий;
5. Изменяем свойство затенения: делаем его непрозрачным и кликабельным. Это позволяет нам эффективно запретить взаимодействие с любыми элементами, находящимися под затенением;
6. При повторном нажатии осуществляем все анимации с точностью до наоборот.

Вопросы?

ValueAnimator/ObjectAnimator

PropertyValuesHolder/AnimatorSet

ValueAnimator + Canvas

`PropertyValuesHolder` нужен для того, чтобы иметь возможность запустить несколько анимаций параллельно.

ObjectAnimator + AnimatorSet

`AnimatorSet` нужен для того, чтобы иметь возможность запустить несколько анимаций параллельно.

Вопросы?

ConstraintLayout +
ConstraintSet

ConstraintLayout + ConstraintSet

Есть ещё один способ сложной анимации множества объектов на экране: с помощью начального и конечного `ConstraintLayout` с участием ConstraintSet. Создадим анимацию по тапу на фото: будем отображать и скрывать дополнительную информацию.

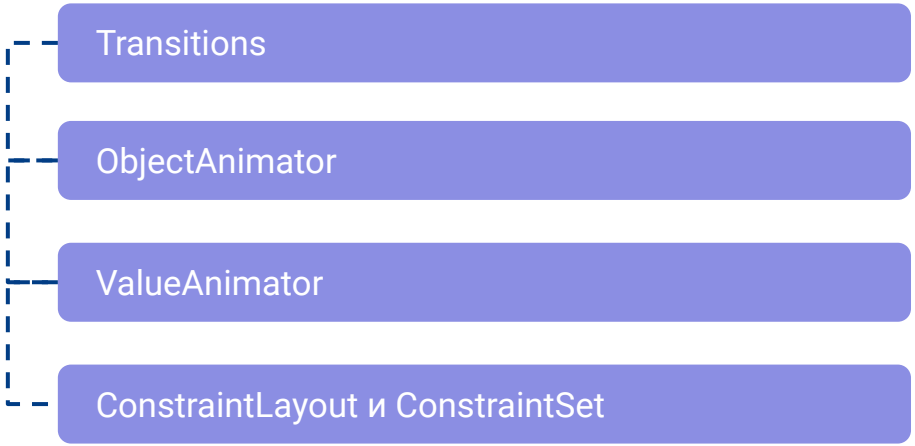
- **Layouts** идентичны, за исключением того, что все элементы, которые мы будем анимировать, находятся за пределами начального экрана;
- Создаём экземпляр класса `ConstraintSet` — он позволяет программно создавать constraints для вашего layout;
- Заполняем constraints конечного экрана через `clone`;
- Создаём `Transition` типа `ChangeBounds` (изменение границ view) и в качестве Interpolator устанавливаем `AnticipateOvershootInterpolator` (позволяет добиться анимации отскока);
- Благодаря интерполятору мы можем произвести анимации сразу над всеми элементами без лишнего кода. Далее указываем длительность анимации и вызываем у `TransitionManager` метод `beginDelayedTransition` Не забываем, что у `ConstraintSet` тоже нужно вызвать метод `applyTo` чтобы анимация стартовала.

Внимание! *Приложение упадёт, если в нашем контейнере constraint_container будут view без id.* ***Id нужен даже там, где вы не используете какой-то view напрямую****. То же самое касается view, которые вы создаете динамически. Им тоже нужно присвоить id программно, чтобы анимации запустились.*

Вопросы?

Рефлексия

Маршрут вебинара



Вопросы для проверки

По пройденному материалу всего вебинара

1. Как используются Transitions?
2. Чем ObjectAnimator отличается от ValueAnimator?
3. Где лучше всего использовать ConstraintSet?

Факультатив

Домашнее задание

Реализуйте анимацию, используя Animator фреймворк:

1. Загрузка в стиле Тиктока;
2. Расходящиеся круги.

Вы можете использовать любой из пройденных компонентов Animator фреймворка: `ValueAnimator`, `ObjectAnimator`, `AnimatorSet` и другие. Нарисуйте фигуры с использованием Canvas.

Заполните, пожалуйста, опрос о занятии

Тема вебинара

Android
Анимации

Александр Аникин

Руководитель команды Андроид-разработки

- Пришёл в IT в 34 года из маркетинга;
- Опыт разработки под Android — 9 лет;
- Опыт преподавания — 8 лет.