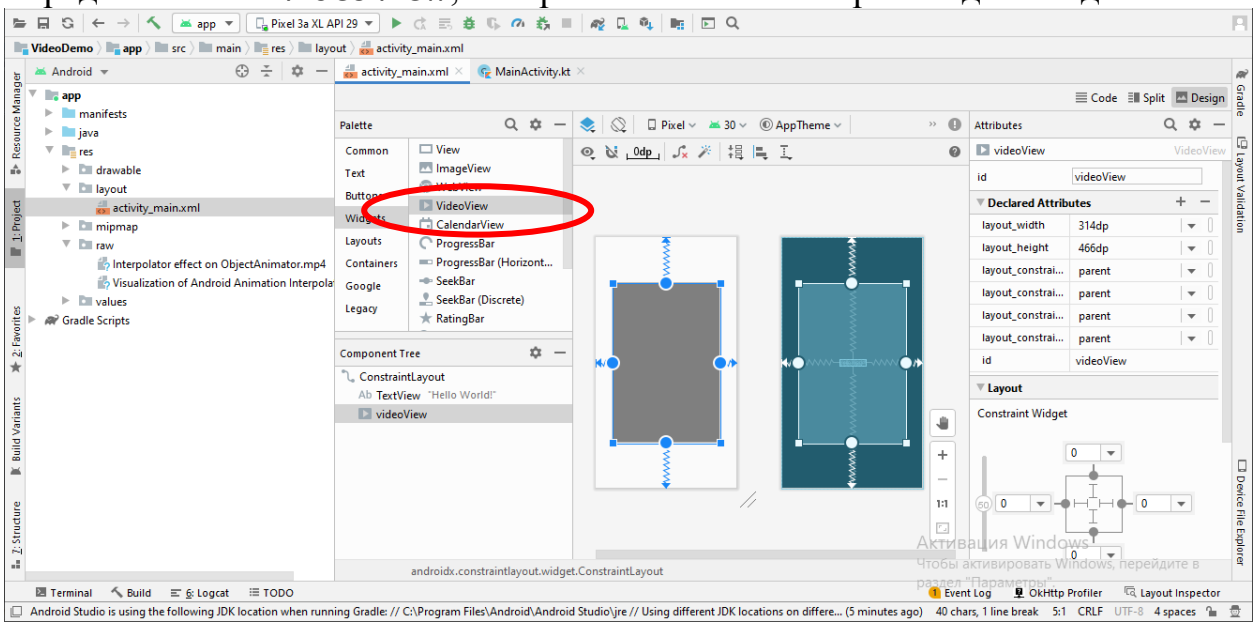
№17 Audio. Video. Animation

Работа с видео

Для работы с видеоматериалами в стандартном наборе виджетов Android определен класс **VideoView**, который позволяет воспроизводить видео.



Android поддерживает все те же форматы видео, что и HTML 5, в частности, формат MPEG-4 H.264(еще известный как WebM), который является оптимальным для Android. Но также можно использовать распространенные форматы MPEG4 H.264 AVC (.mp4) и MPEG4 SP (.3gp)

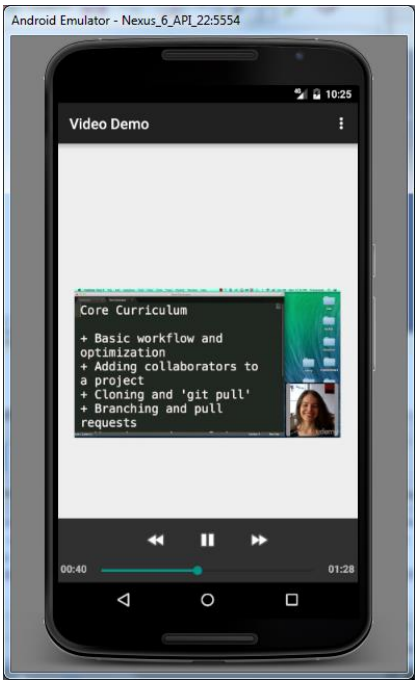
Обычно видеоматериалы помещают в проекте в папку **res/raw**. По умолчанию проект не содержит подобной папки.

Чтобы управлять потоком воспроизведения, надо получить объект VideoView. Чтобы указать источник воспроизведения, необходим объект Uri.

Строка URI имеет ряд частей: сначала идет Uri-схема (http:// или как здесь android.resource://), затем название пакета, получаемое через метод getPackageName(), и далее непосредственно название ресурса видео из папки res/raw, которое совпадает с названием файла.

Затем этот Uri устанавливается у videoPlayera

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val videoView = findViewById(R.id.videoView) as VideoView
        videoView.setVideoPath(
            "android.resource://" + packageName +
                    "/" + R.raw.demovideo2
        )
        val mediaController = MediaController(this)
        mediaController.setAnchorView(videoView)

        videoView.setMediaController(mediaController)
        videoView.setBackgroundColor(Color.TRANSPARENT)

        videoView.start()
      //  videoView.setZOrderOnTop(true)

    }
}
```



Метод videoPlayer.start() начинает или продолжает воспроизведение.

Метод videoPlayer.pause() приостанавливает видео.

Метод videoPlayer.stopPlayback() полностью останавливает видео.

Метод videoPlayer.resume() позволяет снова начать воспроизведение видео с начала после его полной остановки.

С помощью класса **MediaController** мы можем добавить к VideoView дополнительно элементы управления.

Управление аудио

Для воспроизведения музыки и других аудиоматериалов Android предоставляет класс **MediaPlayer**.

Установить нужный ресурс для воспроизведения можно тремя способами:

- в метод *create*() объекта MediaPlayer передается id ресурса, представляющего аудиофайл
- в метод *create*() объекта MediaPlayer передается объект Uri, представляющего аудиофайл
- в метод *setDataSource*() объекта MediaPlayer передается полный путь к аудиофайлу

После установки ресурса вызывается метод *prepare*() или *prepareAsync*()(асинхронный вариант prepare()).

```
val mediaPlayer: MediaPlayer = MediaPlayer()
    mediaPlayer.setDataSource(this, R.raw.somg)

mediaPlayer.start()

mediaPlayer.pause()

mediaPlayer.stop()
```

Управление звуком

Для управления громкостью звука применяется класс **AudioManager**. А в с помощью вызова
audioManager.setStreamVolume(AudioManager.STREAM_MUSIC, progress, 0);
в качестве второго параметра можно передать нужное значение громкости.

```
val audio = getSystemService(Context.AUDIO_SERVICE) as AudioManager

val maxVol = audio.getStreamMaxVolume(AudioManager.STREAM_MUSIC)
val curVol = audio.getStreamVolume(AudioManager.STREAM_MUSIC)

audio.setStreamVolume(AudioManager.STREAM_MUSIC, curVol, 0)
```

Запись

Для записи можно использовать **MediaRecorder**. Этот же класс используется и для записи **видео**.

Чтобы *MediaRecorder* записал для вас звук, он должен знать:
- источник звука
- формат записи
- аудио-кодек
- имя файла

AudioRecorder не пишет данные, а позволяет нам их получать в приложении. т.е. является посредником между приложением и микрофоном. *AudioRecorder* начинает получать данные с микрофона и хранит их у себя во внутреннем буфере. Мы можем при создании *AudioRecorder* указать желаемый размер этого буфера и далее запрашивать из него данные методом *read*.

Анимация

По объекту применения анимации можно классифицировать на следующие:

Анимированные bitmaps

Обычно эти анимации определяются статически с помощью ресурса, но можно определить поведение анимации во время выполнения.

Анимация UI видимости и движения

Для перемещения, раскрытия или скрытия представлений в текущем макете вы можете использовать *property animation* , предоставляемую пакетом android.animation, доступным в Android 3.0 (уровень API 11) и выше.

Physics-based motion

- ***Spring Animation***
- ***Fling Animation***

Анимации, не основанные на физике, например, созданные с помощью ***API ObjectAnimator***, довольно статичны и имеют фиксированную продолжительность

Анимация изменений layout

На Android 4.4 (уровень API 19) и выше вы можете использовать инфраструктуру перехода для создания анимации при смене макета внутри активности или фрагмента.

Начальный и конечный макеты хранятся в *Scene*, хотя начальная сцена обычно определяется автоматически из текущего макета. Затем вы создаете переход, чтобы сообщить системе, какой тип анимации вы хотите, и затем вызываете *TransitionManager.go*(), и система запускает анимацию, чтобы поменять местами макеты.

Анимация между Activity

На Android 5.0 (уровень API 21) и выше вы также можете создавать анимации, которые выполняются между активностями.

Другая классификация может быть выплнена так
Устаревшую -- *Новую*
В XML -- *в Java*

View animation

Изначальная анимация (поддерживается всеми версиями Android SDK)
Для любых объектов расширяющих View

- position
- size
- rotation
- transparency

Преобразования во времени вычисляются автоматически

Frame (drawable) animation

- Поддерживается всеми версиями Android
- основана на традиционной технике
- требует изображений в ресурсах
- Определяется в XML → запускается в Java

Property animation

- Более новая техника (с Android 3.0 API 11)
- Может быть применен к любому объекту и свойству
  - ValueAnimator – расчет анимации движения
  - ObjectAnimator – применяет преобразования к объектам

Transitions framework

- (с Android 4.4 API 19)
- Определяет анимации между XML layout
- XML и Java
- Свойства:
  - Групповая анимация
    - ► Применяет один или несколько эффектов анимации ко всем представлениям в иерархии представлений.
  - Трансляционная анимация
    - ► Выполняет анимацию на основе изменений между значениями свойств запуска и завершения.
  - Встроенная анимация
    - Включает предопределенные анимации для обычных эффектов, таких как затухание или движение.
  - и др.

Material Design

- Android 5.0 (уровень API 21)

- содержит ряд анимации по умолчанию для кнопок и переходов
- Анимация изменений состояния
- Анимация векторных элементов

View animation

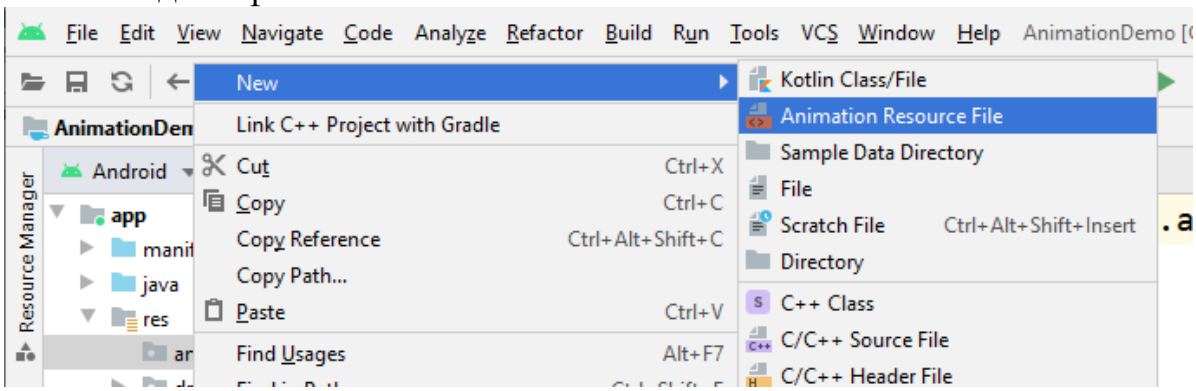
Создание View анимации на основе XML (Tween)

Трансформации конфигурируются в XML файлах, затем в коде программы считываются и присваиваются View-элементам.

В нашем проекте есть папка res. Надо в ней создать папку anim

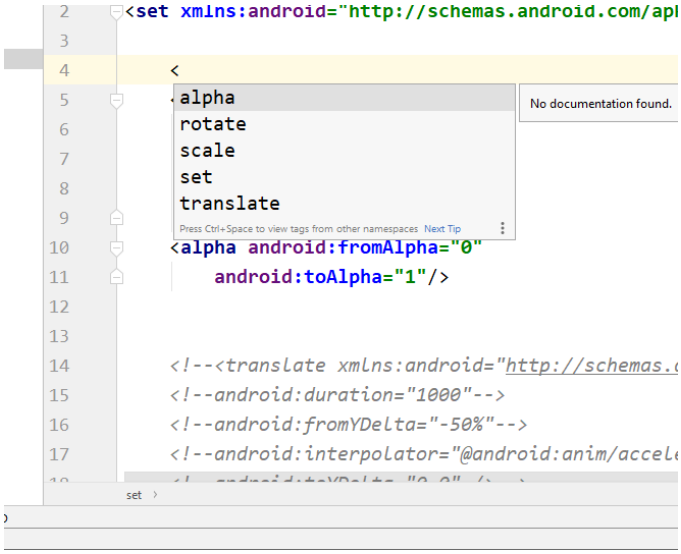
res→anim

Создаем файл someanimation.xml



Задет анимацию для прозрачности, масштабировния, поворота, перемщения

```
<?xml version="1.0" encoding="utf-8"?>
<set
xmlns:android="http://schemas.android.com/apk/res/android">

    <scale android:fromXScale="1.0"
        android:fromYScale="1.0"
        android:toXScale="0.5"
        android:toYScale="0.5"
        />
    <alpha android:fromAlpha="0"
        android:toAlpha="1"/>
</set>
```



<alpha>, <scale>, <translate>, <rotate> и <set> унаследованны от базового класса Animation:

- **duration** — продолжительность эффекта;
- **startOffset** — начальное время смещения;
- **fillBefore** — когда установлен в *true*, то преобразование анимации применяется перед началом анимации;
- **fillAfter** — когда установлен в *true*, то преобразование применяется после конца анимации;
- **repeatCount** — определяет число повторений анимации
- **repeatMode** — определяет поведение анимации при ее окончании : restart (перезапустить без изменений) или reverse (изменить анимацию в обратном направлении);
- **zAdjustment** — определяет режим упорядочения оси Z
- **interpolator** — определяет скорость изменения анимации
  R.styleable:
  android:interpolator="@android:anim/decelerate_interpolator"

Следующий пример управляет анимацией изменения размеров объекта

```
<?xml version="1.0" encoding="utf-8"?>
<set
xmlns:android="http://schemas.android.com/apk/res/android">

    <scale android:fromXScale="1.0"
        android:fromYScale="1.0"
        android:toXScale="0.5"
        android:toYScale="0.5"
        />
</set>
```

создает вертикальную или горизонтальную анимацию движения

```
<translate
xmlns:android="http://schemas.android.com/apk/res/android"

android:duration="1000"

android:fromYDelta="-50%"

android:interpolator="@android:anim/accelerate_interpolator"
android:toYDelta="0.0" />
```

начальное положение - абсолютное или проценты

<rotate> предназначен для анимации вращения и представляет класс **RotateAnimation**

- **fromDegrees** — начальный угол вращения в градусах;
- **toDegrees** — конечный угол вращения в градусах;
- **pivotX** — координата X центра вращения в пикселах;
- **pivotY** — координата Y центра вращения в пикселах;

Классы

- **AnimationSet** (элемент <set>) — класс, представляющий группу анимаций, которые должны запускаться вместе;
- **AlphaAnimation** (элемент <alpha>) — управляет прозрачностью объекта;
- **RotateAnimation** (элемент <rotate>) — управляет вращением объекта;
- **ScaleAnimation** (элемент <scale>) — управляет масштабированием объекта, т.е. изменениями размеров;
- **TranslateAnimation** (элемент <translate>) — управляет позиционированием объекта (перемещение)

```
class MainActivity : AppCompatActivity() {

    private var star: ImageView? = null
    fun onImageClick(view: View?) {
        val animation = AnimationUtils
                .loadAnimation(this, R.anim.someanimation)
        star?.startAnimation(animation)
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        star = findViewById(R.id.star);
    }
}
```

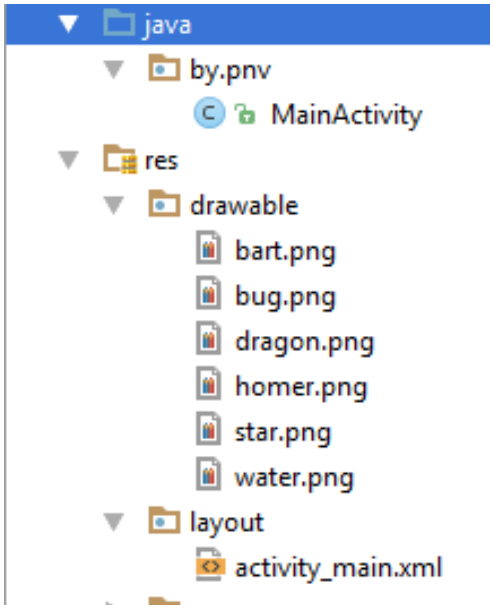
```
animation.setAnimationListener(object : AnimationListener {
    override fun onAnimationStart(animation: Animation) {}
    override fun onAnimationEnd(animation: Animation) {}
    fun onAnimationCancel(animation: Animator?) {}
    override fun onAnimationRepeat(animation: Animation) {}
})
```

Создание View анимации из Java

Для любого view объекта с API 12 или выше можно применить метод animate()

```
star?.animate()
        ?.scaleY(2F)
        ?.scaleY(2F)
        ?.rotationX(180F)
        ?.rotationY(180F)
        ?.translationX(200F)
        ?.translationY(200F)
        ?.setDuration(2000);
```

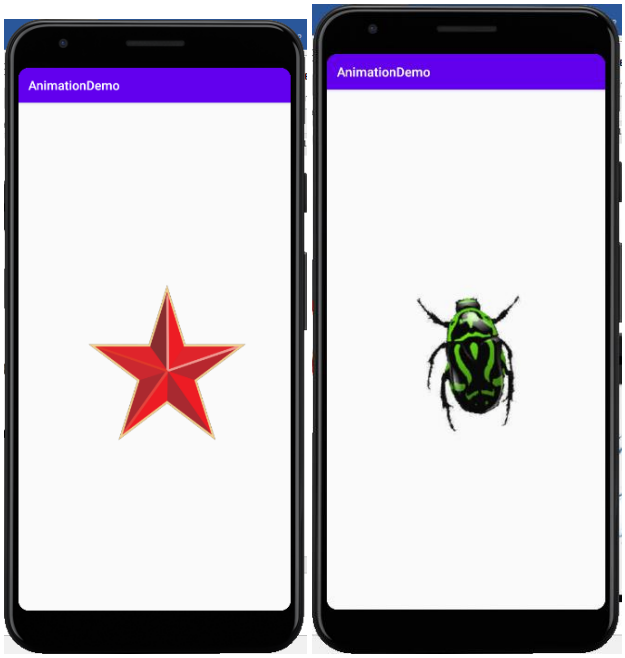
Расмотрим пример. Пусть необхоимо сделать замну одного *imagview* на другой



Создадим два элмента на одном и том же месте. И управляя прозрачностью сделаем замену.

```
<ImageView
    android:id="@+id/star"
    android:layout_width="249dp"
    android:layout_height="556dp"
    android:layout_marginStart="101dp"
    android:layout_marginTop="97dp"
    android:layout_marginEnd="61dp"
    android:layout_marginBottom="78dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:srcCompat="@drawable/star"
    android:onClick="onImageClick"/>

<ImageView
    android:id="@+id/bug"
    android:layout_width="249dp"
    android:layout_height="556dp"
    android:layout_marginStart="101dp"
    android:layout_marginTop="97dp"
    android:layout_marginEnd="61dp"
    android:layout_marginBottom="78dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:srcCompat="@drawable/bug"
    android:alpha="0"/>
```

```
fun onImageClick(view: View?) {
val star = findViewById(R.id.star) as ImageView
val bug = findViewById(R.id.bug) as ImageView
star.animate().alpha(0f).duration = 4000
bug.animate().alpha(1f).duration = 4000
}
```

b 

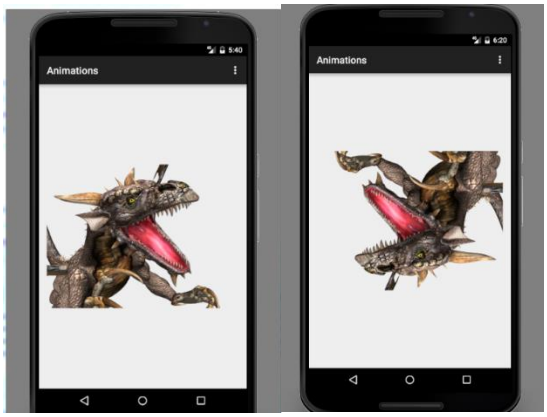
А так будет выглядеть пермещение

```
val star = findViewById(R.id.star) as ImageView
star.animate().translationYBy(200f).duration = 2000
```

```
val star = findViewById(R.id.star) as ImageView
star.animate().translationXBy(-1000f).duration = 2000
```

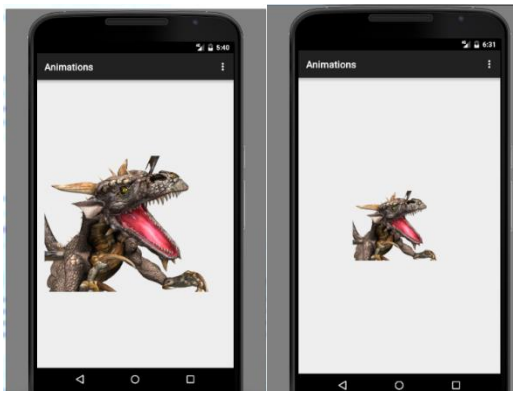
А это будет вращение:

```
val drag = findViewById(R.id.dragon) as ImageView
drag.animate().rotation(180f).duration = 2000
```



И наконец машстабирование

```
val drag = findViewById(R.id.dragon) as ImageView
drag.animate().scaleX(0.5f).scaleY(0.5f).duration = 2000
```



Frame (Drawable) animation

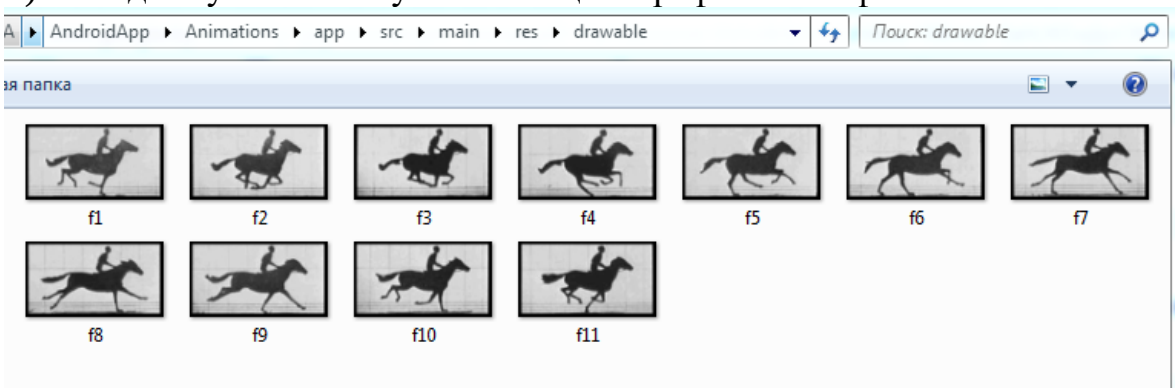
Drawable Animation позволяет Вам загрузить последовательность Drawable-ресурсов один за другим, чтобы в результате получилась анимация. Это традиционный способ анимации, созданной с последовательностью разных картинок, проигрываемых в определенном порядке, наподобие

фильма. Класс **AnimationDrawable** является базовым для анимации такого типа.

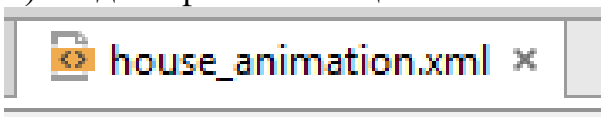
Когда Вы задаете в коде фреймы анимации с использованием AnimationDrawable класса API, более просто это можно реализовать в одном файле XML, в котором перечислены фреймы, составляющие анимацию. Файл XML для такой анимации должен принадлежать директории res/drawable/ Вашего проекта Android. В этом случае в файле XML могут быть заданы инструкции по порядку воспроизведения и длительность для каждого фрейма в анимации.

Для этого надо:

1) слайд-шоу - использует коллекцию графических файлов



2) создать файл анимации



Файл XML состоит из элемента < animation-list > в качестве корневого и последовательных дочерних узлов < item >, которые задают каждый фрейм: drawable-ресурс для фрейма и длительность фрейма. Вот пример файла XML для Drawable-анимации:

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list
xmlns:android="http://schemas.android.com/apk/res/android">
<item android:drawable="@drawable/f1" android:duration="50"/>
    <item android:drawable="@drawable/f1" android:duration="50"/>
    <item android:drawable="@drawable/f2" android:duration="50"/>
    <item android:drawable="@drawable/f3" android:duration="50"/>
    <item android:drawable="@drawable/f4" android:duration="50"/>
    <item android:drawable="@drawable/f5" android:duration="50"/>
    <item android:drawable="@drawable/f6" android:duration="50"/>
    <item android:drawable="@drawable/f7" android:duration="50"/>
    <item android:drawable="@drawable/f8" android:duration="50"/>
    <item android:drawable="@drawable/f9" android:duration="50"/>
</animation-list>
```

3) Подготовка layout

```
        <ImageView
            android:id="@+id/house"
            android:layout_width="408dp"
            android:layout_height="340dp"
            android:layout_marginTop="58dp"
            android:onClick="onHoseClick"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/star" />
```

4) code

```
class MainActivity : AppCompatActivity() {

    private var house: ImageView = findViewById(R.id.house) as ImageView
    private var houseAnimation: AnimationDrawable = house.getBackground() as
AnimationDrawable

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        if (house == null) throw AssertionError()
        houseAnimation.start()

    }
    fun onHoseClick(view: View?) {
        if (houseAnimation.isRunning) houseAnimation.stop()
           else houseAnimation.start()
    }

}
```

Метод start() экземпляра **AnimationDrawable** не может быть вызван во время метода onCreate() Вашей Activity, потому что AnimationDrawable еще не полностью подключен к окну приложения. Если Вы хотите проиграть анимацию немедленно, без необходимости взаимодействия с пользователем, то Вы можете вызвать её из метода ЖЦ, когда окно приложения Android получает фокус.

```
houseAnimation.setOneShot(true);
```

Анимация проиграется один раз и остановится на показе последнего кадр. Если этот атрибут установить в false, то анимация зациклится.

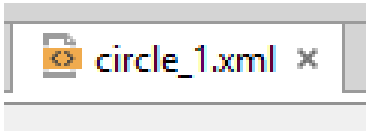
Покадровая в коде

```
val frame1 = resources.getDrawable(R.drawable.f1) as BitmapDrawable
val frame2 = resources.getDrawable(R.drawable.f2) as BitmapDrawable
val frame3 = resources.getDrawable(R.drawable.f3) as BitmapDrawable

val mAnimation = AnimationDrawable()

mAnimation.isOneShot = false

mAnimation.addFrame(frame1, 50)
mAnimation.addFrame(frame2, 50)
mAnimation.addFrame(frame3, 50)
```

типы Drawable

- <level-list> позволяет отображать Drawable в зависимости от значения level

```
<?xml version="1.0" encoding="utf-8"?>
<shape
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="oval">
    <solid
        android:color="#123000">
    </solid>
</shape>
```



```
<?xml version="1.0" encoding="utf-8"?>
<level-list
xmlns:android="http://schemas.android.com/apk/res/android">
<item
    android:drawable="@drawable/circle_1"
    android:maxLevel="0">
</item>
<item
    android:drawable="@drawable/circle_2"
    android:maxLevel="1">
</item>
</level-list>
```

Тег <transition> позволяет указать два Drawable и программно переключаться между ними с fade-эффектом

класс TransitionDrawable

тег <clip> позволяет обрезать Drawable по горизонтальной и (или) вертикальной оси

класс ClipDrawable

Тег <scale> позволяет сжать картинку по горизонтальной (scaleWidth) и (или) вертикальной (scaleHeight) оси и сместить полученное изображение в указанную часть (scaleGravity) доступного пространства.

класс ScaleDrawable

Animation уже является Legacy code, то есть устарел и не поддерживается, хотя и используется.

Property animation

- Изменяет свойство любого объекта
- Поддерживает пользовательские интерполяции
- Свойства
- Продолжительность
- Время интерполяции
- Повторение и поведение
- Множество анимаций
- Задержки обновления

View	***Property animation***
Только View	Любые объекты
Ограниченные изменения: Позиция, поворот, масштаб, прозрачность	+ Фон, цвет и т.п.
	Определяет аспекты анимации – синхронизация и интерполяция
Отображаются моификации view, но не сам view	Объект изменяется фактически
Меньше кода, времени для настроки	Гибкий

Animator — это тип классов, предназначенных для изменения значений выбранного объекта относительно времени.

Идеологическое отличие классов *Animator* от *View Animation* в том, что View Animation изменяет «представление» объекта, не изменяя сам объект.

ValueAnimator (наследуется от *Animator*)

В самом простом варианте мы задаём этому классу тип изменяемого значения, начальное значение и конечное значение, и запускаем. В ответ нам будут приходить события на начало, конец, повторение и отмену анимации и ещё на два события, которые задаются отдельно для паузы и изменения значения. Расчитаем и выведем значения для value. Используя слушателя выведем промежуточные значения на экран в лог:

```
class ValurAnimatorActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_valur_animator)

        val valueAnimator = ValueAnimator.ofFloat(1f, 10f).setDuration(100)
        valueAnimator.addUpdateListener(object : ValueAnimator.AnimatorUpdateListener
{
            override fun onAnimationUpdate(animation: ValueAnimator?) {
                   Log.d(
                        "time",
                       animation?.animatedFraction.toString()
                    );
                    Log.d(
                        "value",
                        animation?.animatedValue.toString()
                    );
                }
            });
        valueAnimator.start();
    }

}
```

```
ck: AUDIO_OUTPUT_FLAG_FAST deni
s D/time: 2
s D/value: 1.0
s D/time: 18
s D/value: 1.6266606
s D/time: 35
s D/value: 3.332109
s D/time: 52
s D/value: 5.5
s D/time: 68
s D/value: 7.790687
s D/time: 84
s D/value: 9.443379
s D/time: 101
s D/value: 10.0
```

```
valueAnimator.setRepeatCount(5);
```

Это задает число повторений.

Инкапсулирует ***TimeInterpolator***, который определяет интерполяцию анимации, и ***TypeEvaluator***, который определяет, как вычислять значения для анимированного свойства.

```
valueAnimator.setRepeatCount(3);
valueAnimator.setRepeatMode(ValueAnimator.REVERSE);
```

```
D/time: 0
D/value: 1.0
D/time: 5
D/value: 1.0
D/time: 52
D/value: 5.5
D/time: 69
D/value: 7.790687
D/time: 86
D/value: 9.443379
D/time: 3
D/value: 10.0
D/time: 21
D/value: 9.373339
D/time: 36
D/value: 7.667891
D/time: 52
D/value: 5.5
D/time: 69
D/value: 3.2093127
D/time: 92
D/value: 1.5566206
D/time: 3
D/value: 1.0
D/time: 27
D/value: 1.6266606
D/time: 39
D/value: 3.332109
D/time: 57
```

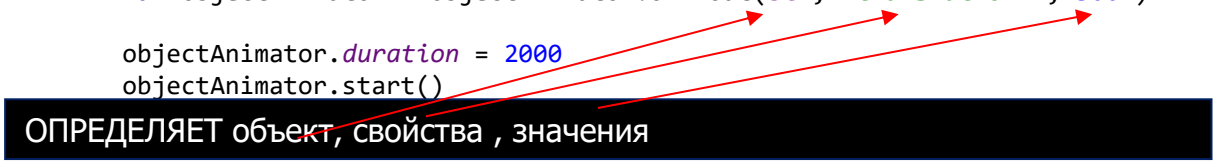
Задает циклическое повторение анимации

```
valueAnimator.setRepeatCount(ValueAnimator.INFINITE);
valueAnimator.start();
```

ObjectAnimator, наследуется от ValueAnimator

С ним вам не нужно вручную изменять какое-либо значение по событию изменения — вы просто даёте Animator'у объект и указываете поле, которое вы хотите изменить, например scaleX. Ищется setter для этого поля (в данном случае — setScaleX. Далее Animator самостоятельно будет менять значение этого поля.

```
class ValurAnimatorActivity : AppCompatActivity() {

    private lateinit var btn: Button

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_valur_animator)
        btn = findViewById(R.id.button)

    }
    fun onBtnClick(view: View?) {
        val objectAnimator = ObjectAnimator.ofFloat(btn, "translationX", 300f)

        objectAnimator.duration = 2000
        objectAnimator.start()
```



Рассмотрим пример

```
val objectAnimator: ObjectAnimator = ObjectAnimator
    .ofFloat(btn, "scaleX", 1f, 2.5f).setDuration(2000)
objectAnimator.start()

val objectAnimator2: ObjectAnimator = ObjectAnimator
    .ofFloat(btn, "scaleY", 1f, 2.5f).setDuration(2000)
objectAnimator2.start()
```

В данном примере анимации выполняются одновременно

AnimatorSet - *(наследуется от Animator)* механизм объединения анимаций вместе, хореография по отношению друг к другу.

```
   val objectAnimator: ObjectAnimator = ObjectAnimator
        .ofFloat(btn, "scaleX", 1f, 2.5f).setDuration(2000)
 //   objectAnimator.start()

    val objectAnimator2: ObjectAnimator = ObjectAnimator
        .ofFloat(btn, "scaleY", 1f, 2.5f).setDuration(2000)
//    objectAnimator2.start()

    val animationSet = AnimatorSet()
   // animationSet.playTogether(objectAnimator, objectAnimator2)
    animationSet.playSequentially(objectAnimator, objectAnimator2)
    animationSet.duration = 4000
    animationSet.start()
```

Для синхронизвции можно использовать :

```
//animationSet.play(objectAnimator).before(objectAnimator2);
//animationSet.play(objectAnimator).after(objectAnimator2);
```

ViewPropertyAnimator - Это отдельный класс. Он не наследуется от Animator, но обладает той же логикой, что и ObjectAnimator для View, и предназначен для лёгкого анимирования какой-либо View.

```
ViewCompat.animate(textView)
        .translationX(50f)
        .translationY(100f)
        .setDuration(1000)
        .setInterpolator(AccelerateDecelerateInterpolator())
        .setStartDelay(50)
        .setListener(object : Animator.AnimatorListener {
            override fun onAnimationRepeat(animation: Animator) {}
            override fun onAnimationEnd(animation: Animator) {}
            override fun onAnimationCancel(animation: Animator) {}
            override fun onAnimationStart(animation: Animator) {}
        })
```

Метод *ViewCompat.animate* возвращает *ViewPropertyAnimator.* Есть **AnimatorListener с** его помощью можно подписываться на определённые события, которые возникают во время выполнения анимации.

https://habr.com/ru/company/e-Legion/blog/418383/

Evaluator (оценки) - сообщают системе анимации свойств, как вычислять значения для данного свойства

IntEvaluator	Вычисляет int свойства (default)
FloatEvaluator	Вычисляет float свойства
ArgbEvaluator	Цвет (hex)
TypeEvaluator	Пользовательский (для выше перечисленных тоже можно опреде

Interpolator - класс, который модифицирует анимацию с использованием математических вычислений (включенные в SDK и собственные) и определяет, как значения вычисляются (функция времени)

Class/Interface	
AccelerateDecelerateInterpolator	скорость изменения начинается и заканчивается медленно, но ускоряется в середине.
AccelerateInterpolator	Интерполятор, скорость изменения которого начинается медленно, а затем ускоряется.
AnticipateInterpolator	Интерполятор, чье изменение начинается с конца, затем движется вперед.
AnticipateOvershootInterpolator	Интерполятор, чье изменение начинается с конца, движется вперед и перевыполняет целевое значение, затем, возвращается к значению.
BounceInterpolator	Интерполятор, чье изменение отскакивает в конце.

CycleInterpolator	Интерполятор, анимация которого повторяется для определенного количества циклов.
DecelerateInterpolator	Интерполятор, скорость изменения которого начинается быстро и затем замедляется.
LinearInterpolator	Интерполятор, скорость изменения которого постоянна.
OvershootInterpolator	Интерполятор, чье изменение движется вперед, перевыполняет кочное значение, затем возвращается.
TimeInterpolator	Интерфейс для собственного интерполятора

```
objectAnimator.interpolator = BounceInterpolator()
```

Определение пути аниматора

Класс *ObjectAnimator* имеет новые конструкторы, которые позволяют анимировать координаты вдоль пути. Например, следующий аниматор использует объект **Path** для анимации свойств X и Y представления

```
val path = Path()
path.arcTo(0f, 0f, 1000f, 1000f, 270f, -180f, true)
val animator = ObjectAnimator.ofFloat(view, View.X, View.Y, path)
animator.duration = 2000
animator.start()
```

Keyframes - состоит из пары времени / значения, которая позволяет вам определить определенное состояние в определенное время анимации. Каждый ключевой кадр может также иметь свой собственный интерполятор

```
val  kf0 = Keyframe.ofFloat(0f, 0f)
val kf1 = Keyframe.ofFloat(.5f, 360f)
val  kf2 = Keyframe.ofFloat(1f, 0f)
val pvhRotation = PropertyValuesHolder.ofKeyframe("rotation", kf0,
kf1, kf2);
val rotationAnim =  ObjectAnimator.ofPropertyValuesHolder(target,
pvhRotation)
rotationAnim.setDuration(5000);
```

ofInt (), ofFloat () или ofObject ()

Слушатели и добавление реакции на анимации

```
val objectAnimator: ObjectAnimator = ObjectAnimator
    .ofFloat(view, "y", 0, targetY).setDuration(10000)
```

```
objectAnimator.repeatCount = 2
objectAnimator.interpolator = BounceInterpolator()
objectAnimator.addListener(object : AnimatorListener {
    override fun onAnimationStart(animation: Animator) {}
    override fun onAnimationEnd(animation: Animator) {}
    override fun onAnimationCancel(animation: Animator) {}
    override fun onAnimationRepeat(animation: Animator) {}
})
objectAnimator.start()
```

Вы можете слушать важные события во время анимации.
Прослушайте это событие, чтобы использовать рассчитанные значения ValueAnimator во время анимации

```
...
objectAnimator.addUpdateListener(new
ValueAnimator.AnimatorUpdateListener() {

  @Override
    public void onAnimationUpdate(ValueAnimator animation) {

     animation.getAnimatedValue();
    }
});
objectAnimator.addListener(new Animator.AnimatorListener() {
    @Override
...
```

Объявление анимаций в XML

ValueAnimator - <animator>
ObjectAnimator - <objectAnimator>
AnimatorSet - <set>

```
<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>
val set = AnimatorInflater.loadAnimator(myContext,
R.anim.property_animator) as AnimatorSet
set.setTarget(myObject)
set.start()
```

Использование crossfade анимации между Activity (Java)

Crossfade анимация (также известная как затухание), это плавное исчезновение одного компонента с одновременным плавным появлением другого.

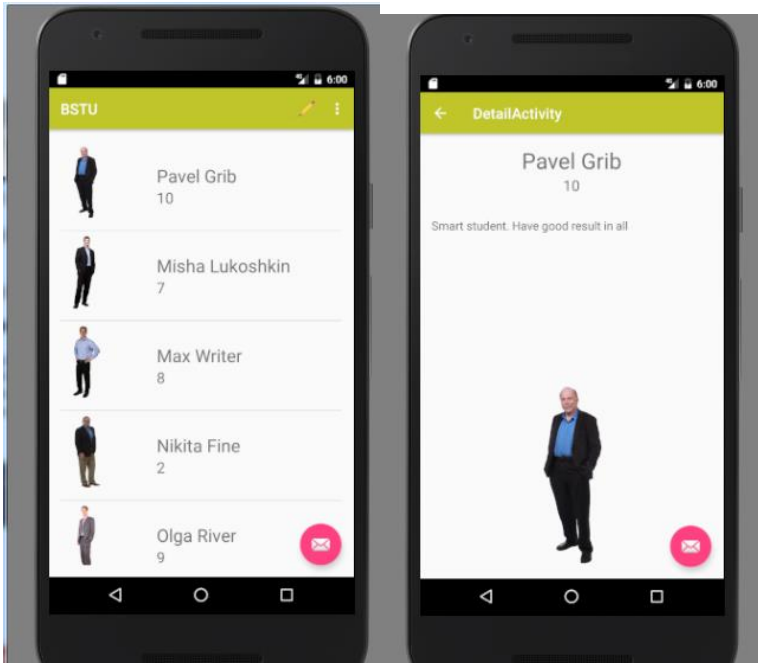
```
startActivity(intent);
overridePendingTransition(android.R.anim.fade_in,
        android.R.anim.fade_out);
```

У класса Activity есть метод *overridePendingTransition*() и он должен идти сразу после вызова метода *startActivity*() или *finish*(). Метод принимает два параметра: ресурс анимации для запускающей активности и ресурс анимации для уходящей активности.

Таким образом вам нужно создать в папке res/anim два файла анимации и передать их идентификаторы в метод.

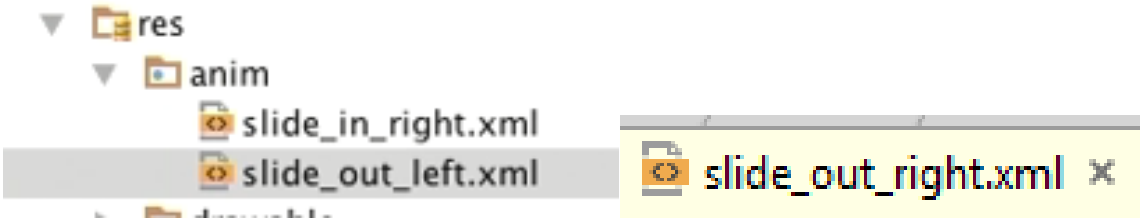
Вы можете воспользоваться готовыми системными ресурсами *android.R.anim.fade_out, android.R.anim.fade_in, android.R.anim.slide_out_right и android.R.anim.slide_in_left.*

```
@Override
protected void onPause() {
    super.onPause();
    overridePendingTransition(android.R.anim.slide_in_left,
            android.R.anim.slide_out_right);
}
```



Определяем свою анимацию для переходов

```
startActivity(Intent);
overridePendingTransition(android.R.anim.fade_in,
                          android.R.anim.fade_out);

overridePendingTransition(R.anim.slide_in_left,
                        R.anim.slide_out_right);
```




```
<set xmlns:android="http://schemas.android.com/apk/res/android">
   <translate android:fromXDelta="0" android:toXDelta="50%p"
            android:duration="@android:integer/config_mediumAnimTime"/>
   <alpha android:fromAlpha="1.0" android:toAlpha="0.0"
            android:duration="@android:integer/config_mediumAnimTime" />
</set>
```

Использование ActivityOptions

Android SDK включает API ActivityOptions.makeScaleUpAnimation() для запуска активности с использованием масштабируемой анимации из представления.

https://developer.android.com/reference/android/app/ActivityOptions

```
public void onClick(View view) {
    Intent intent = new Intent(this, SecondActivity.class);

    ActivityOptions options =
      ActivityOptions.makeScaleUpAnimation(view, 0,
                                0, view.getWidth(), view.getHeight());
    startActivity(intent, options.toBundle());
}
```

Анимация Activity при помощи XML

1) res/anim – создаем анимацию .xml
2) res/values/styles.xml - создаем стиль

```
<style name="DownUpActivity" parent="@android:style/Animation.Activity">
<item name="android:activityOpenEnterAnimation">@anim/activity_down_up_enter</item>
<item name="android:activityOpenExitAnimation">@anim/activity_down_up_exit</item>
<item name="android:activityCloseExitAnimation">@anim/activity_down_up_close_exit</item>
<item name="android:activityCloseEnterAnimation">@anim/activity_down_up_close_enter</item>
    </style>
```

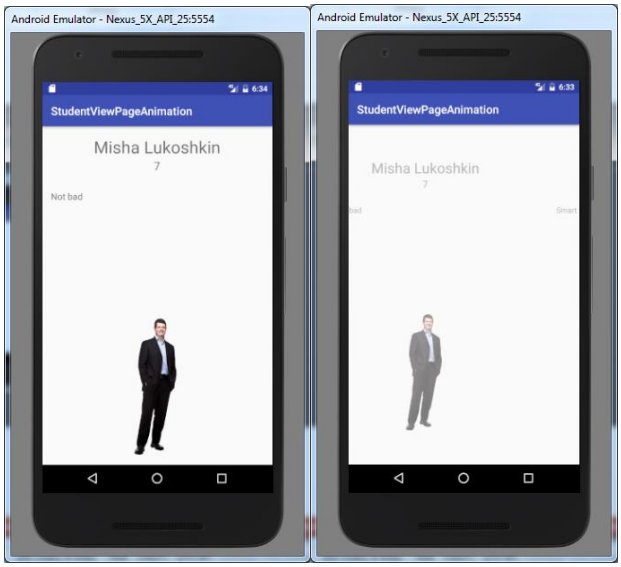
3) res/values/themes.xml - файл для темы

```
<style
name="AnimationActivity"
parent="@android:style/Theme.Light">
<item name="android:windowAnimationStyle">@style/DownUpActivity</item>
</style>
```

4) Прописываем темы в манифесте для двух активностей

```
<activity
android:name=“.SecondActivity"
android:label="@string/title_activity_second"
android:theme="@style/AnimationActivity" >
```

Анимации между фрагментами в ViewPager при помощи transformers



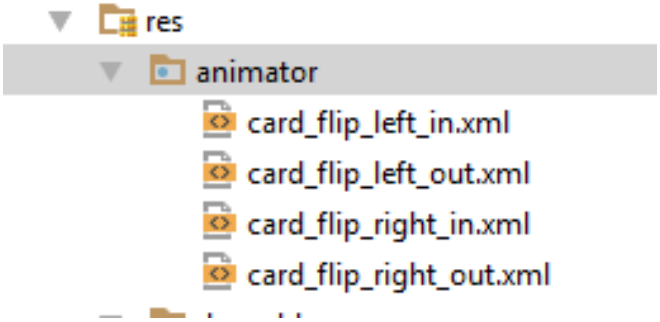
```
 @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Instantiate a ViewPager and a PagerAdapter.
        mPager = (ViewPager) findViewById(R.id.pager);
//        mPager.setPageTransformer(true,new StudPageTransformer());

      mPager.setPageTransformer(true,new ZoomPageTransformer());

       PagerAdapter mPagerAdapter = new
                  ViewPagerAdapter(getSupportFragmentManager());

        mPager.setAdapter(mPagerAdapter);
    }



public class ZoomPageTransformer implements ViewPager.PageTransformer {
        private static final float MIN_SCALE = 0.85f;
        private static final float MIN_ALPHA = 0.5f;

        @Override
        public void transformPage(View view, float position) {
            int pageWidth = view.getWidth();
            int pageHeight = view.getHeight();

            if (position < -1) {
                view.setAlpha(0);

            } else if (position <= 1) {
                float scaleFactor = Math.max(MIN_SCALE, 1 - Math.abs(position));
                float vertMargin = pageHeight * (1 - scaleFactor) / 2;
                float horzMargin = pageWidth * (1 - scaleFactor) / 2;
                if (position < 0) {
                    view.setTranslationX(horzMargin - vertMargin / 2);
                } else {
                    view.setTranslationX(-horzMargin + vertMargin / 2);
                }

                view.setScaleX(scaleFactor);
                view.setScaleY(scaleFactor);

                view.setAlpha(MIN_ALPHA +
                        (scaleFactor - MIN_SCALE) /
                                (1 - MIN_SCALE) * (1 - MIN_ALPHA));

            } else {
                view.setAlpha(0);
            }
        }
}
```

Card Flip анимация с фрагментами

Card Flip позволяет анимировать просмотр содержимого, показывая анимацию, имитирующую переворачивание карточки. Показанная здесь

анимация использует FragmentTransaction, который доступен для Android 3.0 (уровень API 11) и выше.

1) создать файлы анимации




```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Перед поворотом установить alpha to 0. -->
    <objectAnimator
        android:valueFrom="1.0"
        android:valueTo="0.0"
        android:propertyName="alpha"
        android:duration="0" />

    <!-- Поворот. -->
    <objectAnimator
        android:valueFrom="-180"
        android:valueTo="0"
        android:propertyName="rotationY"
        android:interpolator="@android:interpolator/accelerate_decelerate"
        android:duration="@integer/card_flip_time_full" />

    <!-- на половине поворота установить alpha to 1. -->
    <objectAnimator
        android:valueFrom="0.0"
        android:valueTo="1.0"
        android:propertyName="alpha"
        android:startOffset="@integer/card_flip_time_half"
        android:duration="1" />
</set>
```

Приложение содержит фрагменты

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {

 int id = item.getItemId();
    switch (id) {
        case R.id.action_about:
            viewAbout();
            return true;
    }
    return super.onOptionsItemSelected(item);
}
```

```
private void viewAbout() {

    if (mShowingAbout) {
        getFragmentManager().popBackStack();
        mShowingAbout = false;
        return;
    }

    aboutFragment = new AboutFragment();
    getFragmentManager()
            .beginTransaction()
            .setCustomAnimations(R.animator.card_flip_left_in,
                    R.animator.card_flip_left_out,
                    R.animator.card_flip_right_in,
                    R.animator.card_flip_right_out)
            .replace(R.id.fragment_container, aboutFragment)
            .addToBackStack(null)
            .commit();
    mShowingAbout = true;
}
```

Transitions framework (инфраструктура переходов)

Transition frameworвключает в себя следующие функции:

Group-level animations: примените один или несколько эффектов анимации ко всем представлениям в иерархии представлений.

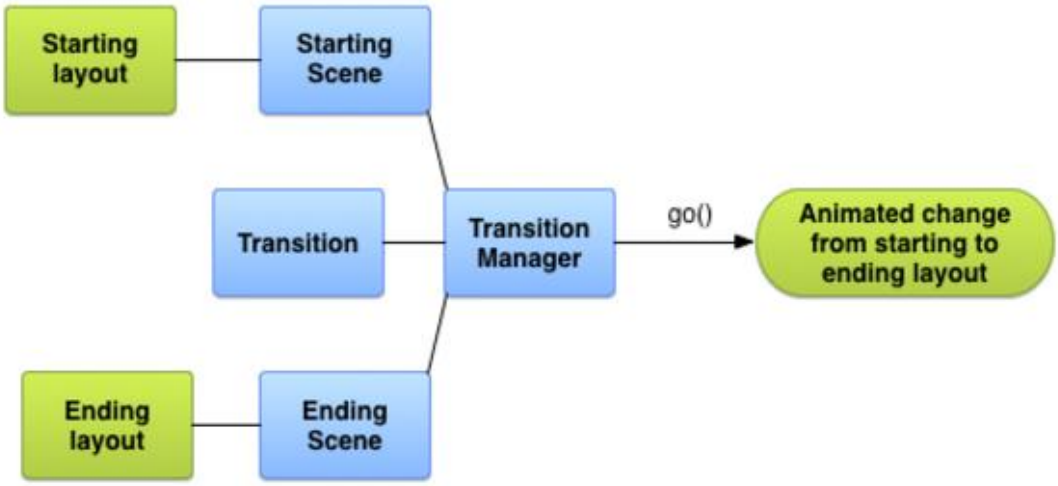
Built-in animations: используйте предопределенные анимации для общих эффектов, таких как затухание или движение.

Поддержка файлов ресурсов: загрузка иерархий представления и встроенных анимаций из файлов ресурсов макета.

Обратные вызовы жизненного цикла: получение обратных вызовов, которые обеспечивают контроль над процессом анимации и изменения иерархии.

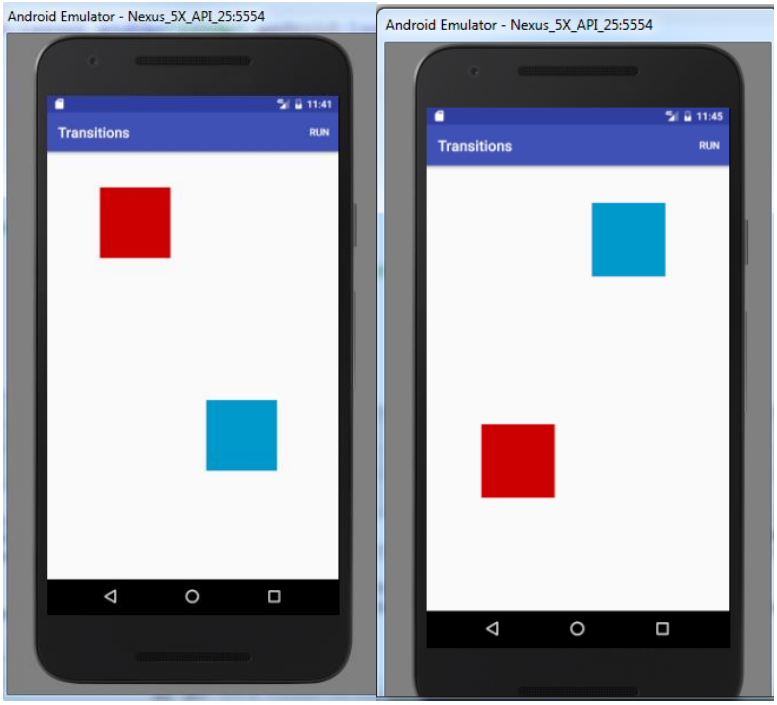
Основной процесс анимации между двумя макетами выглядит следующим образом:

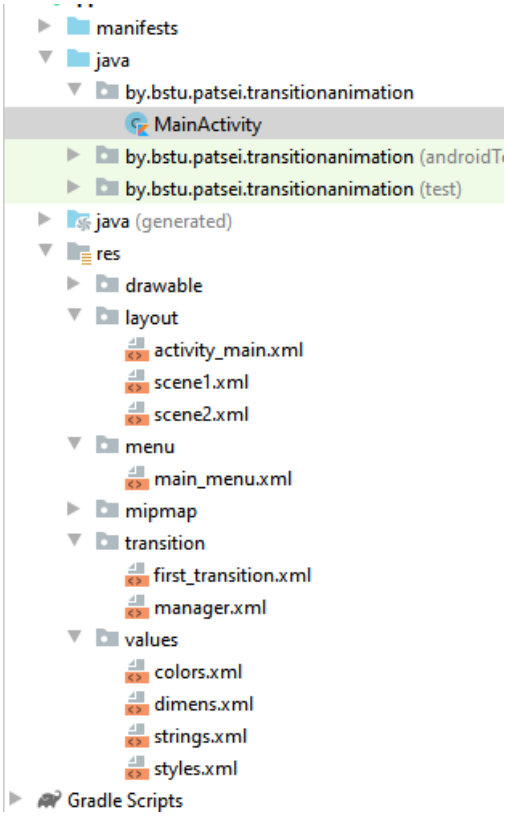
1) Создайте объект ***Scene*** для начального макета и конечного макета. Сцена начального макета часто определяется автоматически из текущего макета.
2) Создайте объект ***Transition***, чтобы определить, какой тип анимации вы хотите.
3) Вызовите ***TransitionManager.go***(), и система запустит анимацию, чтобы поменять местами макеты.



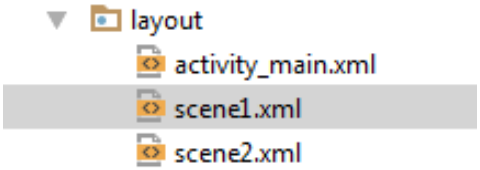
Структура анимирует каждое *View*, изменяя одно или несколько значений его свойств между начальной или начальной иерархией View и конечной или конечной иерархией View.

- ***Scene*** – последовательность View организованных иерархически (XML layout) - ***ViewGroup***
- ***Transition*** – определяет аниматор переходов, который показывает как сцены должны сменяться (XML)
- ***TransitionInflater*** – класс читает изменения XML
- ***TransitionManager*** – управляет изменениями во время выполнения





1) создаем макет



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

    <FrameLayout
        android:id="@+id/scene_root"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <include layout="@layout/scene1" />
    </FrameLayout>
</LinearLayout>
```

2)Создаем scene

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <ImageView
        android:id="@+id/imageViewRed"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_marginStart="75dp"
        android:layout_marginTop="50dp"
        android:background="@android:color/holo_red_dark" />
   <ImageView
        android:id="@+id/imageViewBlue"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_marginStart="225dp"
        android:layout_marginTop="350dp"
        android:background="@android:color/holo_blue_dark" />

</RelativeLayout>
```

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <ImageView
        android:id="@+id/imageViewRed"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_marginStart="75dp"
        android:layout_marginTop="350dp"
        android:background="@android:color/holo_red_dark" />

    <ImageView
        android:id="@+id/imageViewBlue"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_marginStart="225dp"
        android:layout_marginTop="50dp"
        android:background="@android:color/holo_blue_dark" />

</RelativeLayout>
```

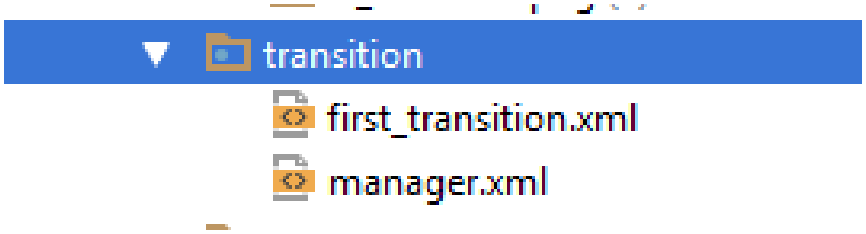
3) *Определяем Transition (Java или XML)*

После того, как вы определили начальную сцену и конечную сцену, между которыми вы хотите переключиться, вам нужно создать объект Transition, который определяет анимацию. Каркас позволяет вам указать встроенный переход в файле ресурсов или создать экземпляр встроенного перехода непосредственно в вашем коде.

Простые типы Transition

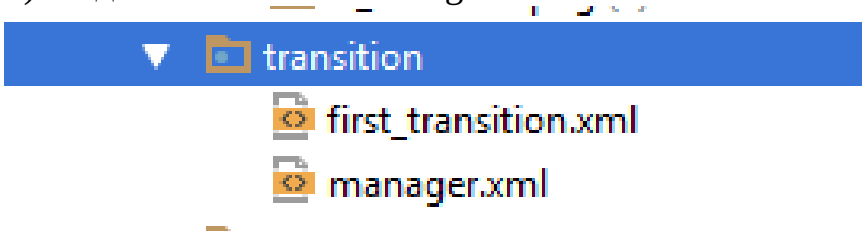
- **ChangeBounds. <changeBounds/>**
  - изменение координат View внутри layout и его размеров
- **Fade**
  - fade in и fade out видна в первой сцене, но должна исчезнуть при переходе во вторую, или же, наоборот, появиться.
    fade_in исчезает во view
    fade_out
    fade_in_out (по умолчанию) выполняет fade_out, за которым следует fade_in.
- **TransitionSet**
- **AutoTransition** - Default transition. исчезает, перемещает и изменяет размеры, исчезает во view. Именно в таком порядке.

Добавьте каталог **res / transition /** в ваш проект.
Создайте новый файл ресурсов XML внутри этого каталога.
Добавьте XML для одного из встроенных переходов.




```
<?xml version="1.0" encoding="utf-8"?>
    <transitionSet xmlns:android="http://schemas.android.com/apk/res/android">
    <fade android:fadingMode="fade_in"/>
    <changeBounds android:duration="1000"/>
    <explode/>
</transitionSet>
```

4) создание Transition Manager




```
<?xml version="1.0" encoding="utf-8"?>

    <transitionManager xmlns:android="http://schemas.android.com/apk/res/android">
    <transition
        android:fromScene="@layout/scene2"
        android:toScene="@layout/scene1"
        android:transition="@transition/first_transition" />
</transitionManager>
```

5) запуск анимации

```
class MainActivity : AppCompatActivity() {

    var mCurrentScene = 2
    private lateinit var mScene1: Scene
    private lateinit var mScene2: Scene
    private lateinit var mSceneRoot: ViewGroup
    private lateinit var transitionManager: TransitionManager

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        mSceneRoot = findViewById(R.id.scene_root) as ViewGroup
        mScene1 = Scene.getSceneForLayout(mSceneRoot, R.layout.scene1, this)
        mScene2 = Scene.getSceneForLayout(mSceneRoot, R.layout.scene2, this)

        transitionManager = TransitionInflater.from(this)
                .inflateTransitionManager(R.transition.manager, mSceneRoot)
    }

    override fun onCreateOptionsMenu(menu: Menu?): Boolean {
        menuInflater.inflate(R.menu.main_menu, menu)
        return true
```

```
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        if (item.itemId == R.id.action_transition) {
           runTransition()
            return true
        }
        return super.onOptionsItemSelected(item)
    }

    private fun runTransition() {
        mCurrentScene = if (mCurrentScene == 1) {
            TransitionManager.go(mScene2)
            2
        } else {
            TransitionManager.go(mScene1)
            1
        }
    }
}
```

► Изменение свойств трансформации

```
val set = TransitionSet()
set.addTransition(Fade())
set.addTransition(ChangeBounds())

set.ordering = TransitionSet.ORDERING_TOGETHER

    set.interpolator = AccelerateInterpolator()
TransitionManager.go(scene2, set)
```

Physics-based Animations

Fling Animation

Анимация на основе Fling использует силу трения, которая пропорциональна скорости объекта. У нее есть начальный импульс, который главным образом определяется скоростью жеста, и постепенно замедляется. Анимация заканчивается, когда скорость анимации низкая и нет видимых изменений на экране устройства.

Чтобы использовать анимацию на основе физики, вы должны добавить библиотеку поддержки в файл build.gradle для вашего модуля приложения:

```
// Java language implementation
    implementation "androidx.dynamicanimation:dynamicanimation:1.0.0"
    // Kotlin
    implementation "androidx.dynamicanimation:dynamicanimation-ktx:1.0.0-alpha03"
```

Класс FlingAnimation позволяет вам создавать анимацию fling для объекта. Создайте экземпляр класса **FlingAnimation** и предоставьте объект и свойство объекта, которое вы хотите анимировать.

```
val fling = FlingAnimation(view, DynamicAnimation.SCROLL_X)
```

Установите *скорость*. Начальная скорость определяет скорость, с которой свойство анимации изменяется в начале.

Чтобы установить скорость, вызовите метод **setStartVelocity** () и передайте скорость в пикселях в секунду. Метод возвращает объект, на котором установлена скорость.

Установите *диапазон значений анимации*. Чтобы установить минимальное и максимальное значения, вызовите методы **setMinValue**() и **setMaxValue**() соответственно. Оба метода возвращают объект анимации, для которого вы установили значение.

Метод **setFriction**() позволяет изменить трение анимации. Он определяет, насколько быстро скорость анимации уменьшается

Пример ниже иллюстрирует горизонтальное перемещение. Скорость, полученная от трекера скорости, равна velocityX, а границы прокрутки установлены в 0 и maxScroll. Трение установлено на 1,1.

```
val fling = FlingAnimation(view, DynamicAnimation.SCROLL_X)
fling.setStartVelocity(100f)
    .setFriction(1.1f)
    .start()
```

Spring Animation (пружина)

Усилие пружины обладает следующими свойствами: демпфирование и жесткость. В анимации на основе пружины значение и скорость рассчитываются на основе силы пружины, приложенной к каждому кадру.

Класс **SpringForce** позволяет настраивать жесткость пружины, ее коэффициент деформирования и конечное положение. Как только анимация начинается, сила пружины обновляет значение и скорость в каждом кадре. Анимация продолжается, пока сила пружины не достигнет равновесия.

Основные шаги по созданию анимации:

1) Добавление библиотеки поддержки

```
// Java language implementation
    implementation "androidx.dynamicanimation:dynamicanimation:1.0.0"
    // Kotlin
    implementation "androidx.dynamicanimation:dynamicanimation-ktx:1.0.0-alpha03"
```

2) Создание анимации пружины. Первым шагом является создание экземпляра класса SpringAnimation и установка параметров движения.

```
val springAnim = SpringAnimation(view, DynamicAnimation.TRANSLATION_Y, 400f)
```

Доступны следующее парамтеры:

ALPHA: представляет альфа-прозрачность в представлении. По умолчанию это значение равно 1 (непрозрачное), а значение 0 соответствует полной прозрачности (не отображается).

TRANSLATION_X, TRANSLATION_Y и TRANSLATION_Z: эти свойства управляют тем, где представление находится в виде дельты от его координаты.

TRANSLATION_X описывает левую координату.

TRANSLATION_Y описывает верхнюю координату.

TRANSLATION_Z описывает глубину обзора

ROTATION, ROTATION_X и ROTATION_Y: эти свойства управляют вращением в 2D (свойство поворота) и 3D вокруг точки поворота.

SCROLL_X и SCROLL_Y: свойства указывают смещение прокрутки левого и верхнего края источника в пикселях.

SCALE_X и SCALE_Y: эти свойства управляют 2D-масштабированием вида вокруг его точки поворота.

X, Y и Z: это основные служебные свойства, которые описывают конечное местоположение представления в его контейнере.

X - сумма левого значения и TRANSLATION_X.

Y является суммой верхнего значения и TRANSLATION_Y.

Z является суммой значения высоты и TRANSLATION_Z.

3) (Необязательно) Зарегистрируйте слушателей, чтобы следить за изменениями жизненного цикла анимации и обновлениями значений анимации.

```
val springAnim = SpringAnimation(view, DynamicAnimation.TRANSLATION_Y, 400f)
 val springForce: SpringForce = SpringForce()
 springForce.setFinalPosition(400f)
 springForce.setDampingRatio(SpringForce.DAMPING_RATIO_HIGH_BOUNCY)
 springForce.setStiffness(SpringForce.STIFFNESS_LOW)
 springAnim.setSpring(springForce);
springAnim.animateToFinalPosition(500F);
```

Класс **DynamicAnimation** предоставляет двух слушателей: **OnAnimationUpdateListener** и **OnAnimationEndListener**. Эти слушатели слушают обновления анимации, например, когда изменяется значение анимации и когда заканчивается анимация.

```
        // Setting up a spring animation to animate the view1 and view2 translationX
and translationY properties
        val anim1X = SpringAnimation(
            view1,
            DynamicAnimation.TRANSLATION_X
        )
        val anim1Y = SpringAnimation(
            view1,
            DynamicAnimation.TRANSLATION_Y
        )
        val anim2X = SpringAnimation(
            view2,
            DynamicAnimation.TRANSLATION_X
        )
        val anim2Y = SpringAnimation(
            view2,
            DynamicAnimation.TRANSLATION_Y
        )
```

```
// Registering the update listener
        anim1X.addUpdateListener { dynamicAnimation, value, velocity ->
// Overriding the method to notify view2 about the change in the view1's property.
            anim2X.animateToFinalPosition(value)
        }


        anim1Y.addUpdateListener { dynamicAnimation, value, velocity ->
            anim2Y.animateToFinalPosition(
                value
            )
        }
```

4)(Необязательно) Удалить слушателей: удалить слушателей, которые больше не используются.

5) (Необязательно) Установите начальное значение

Чтобы установить начальное значение анимации, вызовите метод *setStartValue* () и передайте начальное значение анимации. Если вы не установите начальное значение, анимация использует текущее значение свойства объекта в качестве начального значения.

6) (Необязательно) Установите диапазон значений анимации, чтобы ограничить значения в пределах минимального и максимального.

Чтобы установить минимальное значение, вызовите метод *setMinValue* () и передайте минимальное значение свойства. Чтобы установить максимальное значение - *setMaxValue*(). Оба метода возвращают анимацию, для которой устанавливается значение.

7) (Необязательно) Установите начальную скорость.

```
vt.computeCurrentVelocity(1000);
float velocity = vt.getYVelocity();
anim.setStartVelocity(velocity);

float pixelPerSecond = TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_DIP,
dpPerSecond, getResources().getDisplayMetrics());
```

8) (Необязательно) Установите свойства пружины.

9) (Необязательно) Создание настраиваемой пружины.

10) Запустить анимацию

Есть два способа запустить анимацию: вызывая ***start*** () или вызывая метод ***animateToFinalPosition***(). Оба метода должны быть вызваны в главном потоке.

Метод ***animateToFinalPosition*** () выполняет две задачи. Устанавливает конечную позицию пружины. Запускает анимацию, если она еще не началась.

Поскольку метод обновляет конечную позицию пружины и запускает анимацию, если необходимо, вы можете вызвать этот метод в любое время, чтобы изменить ход анимации. Например, в анимации цепочки пружин анимация одного вида зависит от другого. Для такой анимации удобнее использовать метод ***animateToFinalPosition*** (). Используя этот метод в анимации цепочки пружин, вам не нужно беспокоиться о том, запущена ли анимация, которую вы хотите обновить следующей.

Метод ***start*** () не устанавливает значение свойства в начальное значение немедленно. Значение свойства изменяется при каждом импульсе анимации,

что происходит перед проходом прорисовки. В результате изменения отражаются в следующем кадре, как будто значения устанавливаются немедленно.

```
   val img = findViewById(R.id.imageView);
  val anim = SpringAnimation(img, DynamicAnimation.TRANSLATION_Y);
    …
//Starting the animation
    anim.start();
```

11) (Необязательно) Отменить анимацию.

Вы можете отменить или перейти к концу анимации. В основном это происходит, когда пользователь внезапно выходит из приложения или представление становится невидимым.

Есть два метода, которые вы можете использовать для завершения анимации. Метод ***cancel*** () завершает анимацию со значением, в котором она находится. Метод ***skipToEnd*** () пропускает анимацию до конечного значения и затем завершает ее.

Прежде чем вы сможете завершить анимацию, важно сначала проверить состояние пружины. Если состояние незатухающее, анимация никогда не сможет достичь исходной позиции. Чтобы проверить состояние пружины, вызовите метод ***canSkipToEnd*** (). Если пружина затухает, метод возвращает true, иначе false.

Как только вы узнаете состояние пружины, вы можете прервать анимацию, используя метод ***skipToEnd*** () или метод ***cancel*** (). Метод ***cancel*** () должен вызываться только в основном потоке.

2D рисование (Java)

Canvas - 1) получение доступа (один поток) **View**

```
class DrawView extends View {

    public DrawView(Context context) {
        super(context);
    }

    @Override
    protected void onDraw(Canvas canvas) {
        canvas.drawColor(Color.BLUE);
    }
}
```

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(new DrawView(this));
    }
```

2) **SurfaceView**

```
class DrawView extends SurfaceView implements SurfaceHolder.Callback
{

    private DrawThread drawThread;
...
@    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        drawThread = new DrawThread(getHolder());
        drawThread.setRunning(true);
        drawThread.start();
    }

    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {
}
    class DrawThread extends Thread {
        private boolean running = false;
        private SurfaceHolder surfaceHolder;
...
        }
        @Override
        public void run() {
            Canvas canvas;
            ...}
}
```

Фигуры

- ► Region
- ► clip
- ► Path

```
Paint p;
p = new Paint();
p.setColor(Color.RED);
p.setStrokeWidth(10);

rect = new Rect();
canvas.drawARGB(80, 102, 204, 255);
canvas.drawPoint(50, 50, p);
canvas.drawLine(100,100,500,50,p);
canvas.drawCircle(100, 200, 50, p);
canvas.drawRect(200, 150, 400, 200, p);
Rect rect;
rect.set(250, 300, 350, 500);
canvas.drawRect(rect, p);
```

```
Path path;
 path = new Path();

path.reset();
path.moveTo(100, 100);
path.lineTo(150, 200);
path.lineTo(50, 200);

path.close();
```

Matrix

- **translate** – перемещение
- **scale** – изменение размера
- **rotate** – поворот
- **skew** - наклон
- **перспектива**

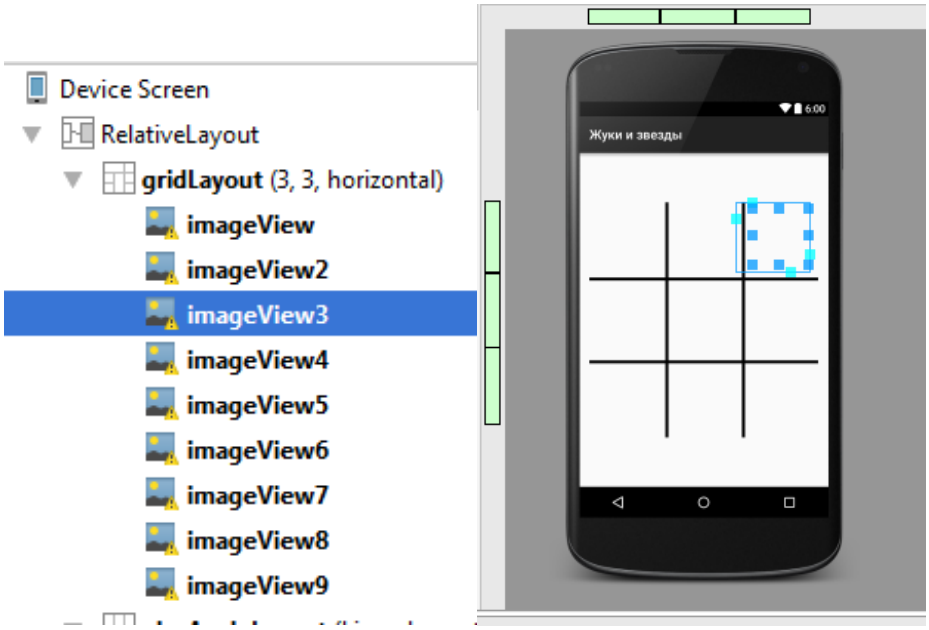
```
path = new Path();
Matrix matrix = new Matrix();

path.addRect(300, 150, 450, 200, Path.Direction.CW);
path.addRect(350, 100, 400, 250, Path.Direction.CW);
canvas.drawPath(path, p);
matrix.reset();
matrix.setTranslate(300, 200);

path.transform(matrix);
```

Bitmap и BitmapFactory

```
Bitmap bitmap = BitmapFactory.decodeResource(getResources(),
                                R.drawable.misha);

                    bitmap.getWidth();
                    bitmap.getHeight();
                    bitmap.getByteCount();
                    bitmap.getRowBytes();
                    bitmap.getConfig();

  matrix = new Matrix();
  matrix.postRotate(45);
  matrix.postScale(2, 3);
  matrix.postTranslate(200, 50);
  Paint paint =  new Paint(Paint.ANTI_ALIAS_FLAG);

  canvas.drawARGB(80, 102, 204, 255);
  canvas.drawBitmap(bitmap, 50, 50, paint);
  canvas.drawBitmap(bitmap, matrix, paint);
```

Игра жуки и звезды

1) Проектирование макета



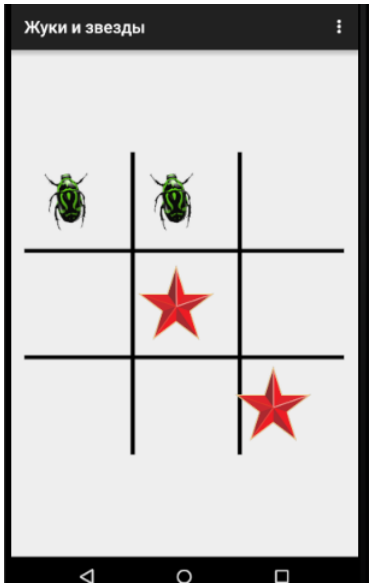
Фрагмент разметки

```
<GridLayout
    android:layout_width="match_parent"
    android:layout_height="360dp"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true"
    android:columnCount="3"
    android:rowCount="3"
    android:background="@drawable/board"
    android:layout_alignParentEnd="true"
    android:id="@+id/gridLayout">

    <ImageView
        android:layout_width="90dp"
        android:layout_height="90dp"
        android:id="@+id/imageView"
        android:layout_row="0"
        android:layout_column="0"
        android:layout_marginLeft="10dp"
        android:layout_marginTop="10dp"
        android:layout_marginRight="10dp"
        android:layout_marginBottom="10dp"
        android:onClick="dropIn"
        android:tag="0" />

    <ImageView
        android:layout_width="90dp"
        android:layout_height="90dp"
        android:id="@+id/imageView2"
        android:layout_row="0"
        android:layout_column="1"
        android:layout_marginTop="10dp"
        android:layout_marginLeft="25dp"
        android:onClick="dropIn"
        android:tag="1" />
```

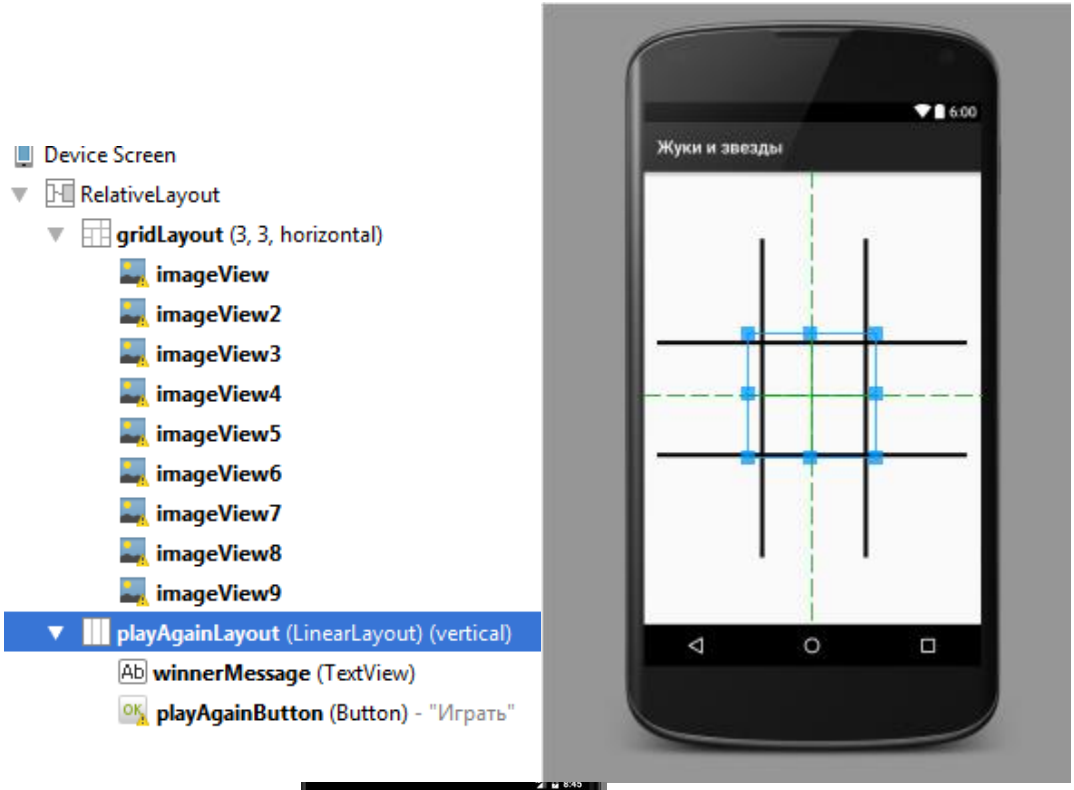
2) Обработка клика и проверка выйгрышной комбинации

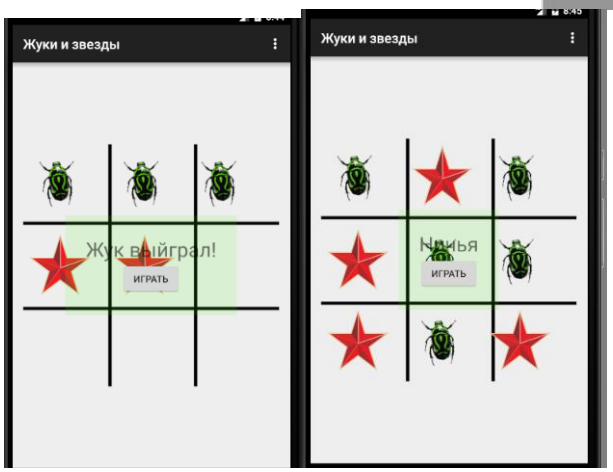
```
// 0 = bug, 1 = star
var activePlayer = 0
var gameIsActive = true
// 2 means unplayed
var fieldState = intArrayOf(2, 2, 2, 2, 2, 2, 2, 2, 2)
var winPositions = arrayOf(intArrayOf(0, 1, 2), intArrayOf(3, 4, 5), intArrayOf(6, 7,
8), intArrayOf(0, 3, 6), intArrayOf(1, 4, 7), intArrayOf(2, 5, 8), intArrayOf(0, 4,
8), intArrayOf(2, 4, 6))

fun dropIn(view: View) {
    val counter = view as ImageView
    val tappedCounter = counter.tag.toString().toInt()

    if (fieldState[tappedCounter] == 2 && gameIsActive) {
        fieldState[tappedCounter] = activePlayer
        counter.translationY = -1000f
        activePlayer = if (activePlayer == 0) {
            counter.setImageResource(R.drawable.bug)
            1
        } else {
            counter.setImageResource(R.drawable.star)
            0
        }

        counter.animate().translationYBy(1000f).rotation(360f).duration = 300
        for (winningPosition in winPositions) {
            if (fieldState[winningPosition[0]] == fieldState[winningPosition[1]] &&
fieldState[winningPosition[1]] == fieldState[winningPosition[2]] &&
fieldState[winningPosition[0]] != 2) {

                // Someone has won!
                gameIsActive = false
                var winner = "Звезда"

                if (fieldState[winningPosition[0]] == 0) {
                    winner = "Жук"
                }

                val winnerMessage = findViewById(R.id.winnerMessage) as TextView
                winnerMessage.text = "$winner выйграл!"
                val layout = findViewById(R.id.playAgainLayout) as LinearLayout
                layout.visibility = View.VISIBLE
            } else {

                var gameIsOver = true
               for (counterState in fieldState) {
                    if (counterState == 2) gameIsOver = false
                }

                if (gameIsOver) {
                    val winnerMessage = findViewById(R.id.winnerMessage) as TextView
                    winnerMessage.text = "Ничья"
                    val layout = findViewById(R.id.playAgainLayout) as LinearLayout
                    layout.visibility = View.VISIBLE
                }
            }
        }
    }
}
```

Игроки делают ход по очереди
Получить imageview на котором сделан click
Из ресурсов загружаем изображения




```
<GridLayout …
</GridLayout>

<LinearLayout
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#1f4aff0b"
    android:padding="30dp"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true"
    android:id="@+id/playAgainLayout"
    android:visibility="invisible">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=""
        android:id="@+id/winnerMessage"
        android:layout_gravity="center_horizontal"
        android:textSize="30sp" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Играть"
        android:id="@+id/playAgainButton"
        android:layout_gravity="center_horizontal"
        android:onClick="playAgain" />
</LinearLayout>
```





3) Продолжение игры

```
fun playAgain(view: View?) {

    gameIsActive = true
    val layout = findViewById(R.id.playAgainLayout) as LinearLayout
    layout.visibility = View.INVISIBLE
    activePlayer = 0
    for (i in fieldState.indices) {
        fieldState[i] = 2
    }

    val gridLayout = findViewById(R.id.gridLayout) as GridLayout
    for (i in 0 until gridLayout.childCount) {
        (gridLayout.getChildAt(i) as ImageView).setImageResource(0)
    }
}
```