

Лабораторная работа № 6 по теме:

«Работа с 2D-графикой»

Теоретическая часть

Двумерную графику в Android-приложении можно создавать двумя способами: либо нарисовать её в объекте View из компоновки, либо нарисовать её непосредственно на канве (в объекте Canvas).

Рисование графики в объекте View предпочтительнее, если требуется нарисовать простую графику, которая не будет динамически изменяться в процессе работы приложения. Недостаток рисования в объекте View заключается в том, что такое рисование выполняется в основном потоке приложения. Если же требуется динамически изменяющаяся графика, например, для реализации 2D-игры, то в этом случае лучше рисовать графику на канве.

Канва или холст (класс Canvas) содержит вызовы «рисования». Чтобы что-то нарисовать, вам требуются четыре основных компонента:

- растровое изображение (объект Bitmap) для хранения пикселей;
- холст (объект Canvas), содержащий вызовы отрисовки (метод draw), т.е. записи в растровое изображение;
- примитив для рисования (например, прямоугольник, контур, текст, растровое изображение);
- краска (объект Paint) для задания цвета и стиля для рисунка.

На платформе Android имеется двумерная графическая библиотека android.graphics.drawable. В классе Drawable определены разнообразные виды графики, включая классы ShapeDrawable, TransitionDrawable, VectorDrawable и т.д.

В таблице 1 перечислены некоторые классы, содержащиеся в библиотеке android.graphics.drawable.

Таблица 1. Классы из библиотеки `android.graphics.drawable`.

Класс	Описание
AnimationDrawable	Используется для создания покадровой анимации, определяемой серией Drawable объектов, которые могут использоваться в качестве фона объекта View.
BitmapDrawable	Drawable, который обертывает растровое изображение.
Drawable	Общая абстракция для «чего-то, что можно нарисовать». Чаще всего вы будете иметь дело с Drawable как с типом ресурса, извлекаемого для рисования объектов на экране.
GradientDrawable	Drawable с цветовым градиентом для кнопок, фона и т.д.
LayerDrawable	Drawable, который управляет массивом других Drawable.
PaintDrawable	Drawable, который рисует свои границы заданной краской с необязательными закругленными углами.
ShapeDrawable	Объект Drawable, который рисует графические примитивы.
TransitionDrawable	Расширение LayerDrawable, предназначенное для плавного перехода между первым и вторым слоями (изображениями).
VectorDrawable	Позволяет создавать объекты Drawable на основе векторной графики XML.

Класс Drawable

Когда требуется отобразить статические изображения, можно использовать класс Drawable и его подклассы для рисования фигур и изображений. Класс Drawable — это общая абстракция того, что можно нарисовать. Различные подклассы Drawable используются для различных типов изображений. Также можно определить собственный подкласс Drawable.

Есть два способа определить и инициализировать объект Drawable помимо использования конструкторов классов:

- использовать изображения (ресурсы), сохраненные в каталоге `res/drawable/`;
- использовать XML-файл, в котором будут определены свойства объекта Drawable.

Самый простой способ добавить графику в приложение — *через ссылку на ресурс drawable*. Загрузка через ссылку на ресурсы предпочтительна для значков, картинок или другой графики в приложении. Этот способ загрузки графических ресурсов мы использовали ранее.

Как отмечалось в практическом занятии № 2, ресурсы изображений, помещенные в каталог `res/drawable`, могут быть автоматически оптимизированы со сжатием утилитой `aapt`. Если же требуется загружать растровые изображения без сжатия, то их помещают в каталог `res/raw`. Однако в этом случае потребуется загрузка с использованием методов потокового ввода-вывода и последующая конвертация графики в растровый рисунок.

Каждый уникальный ресурс в проекте может поддерживать только одно состояние независимо от того, сколько различных объектов `Drawable` вы можете инициализировать в программном коде для этого ресурса. Например, если инициализировать два объекта `Drawable` от одного ресурса изображения, а затем изменить какое-либо свойство, например, прозрачность для одного из объектов `Drawable`, второй объект также изменит это свойство.

Для создания объекта `Drawable` из `xml`-ресурса необходимо определить объект `Drawable` в `xml`-файле, а затем сохранить его в каталог `res/drawable` проекта. В листинге 1 показан код `xml`-файла, который определяет ресурс `TransitionDrawable`, предназначенный для плавного перехода между двумя изображениями. В корневом элементе `<transition>` содержатся два элемента `<item>`, содержащие ссылки на графические ресурсы — файлы `photo1.jpg` и `photo2.jpg`.

Листинг 1. Ресурс `transition.xml`

```
<transition xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/photo1"/>
    <item android:drawable="@drawable/photo2"/>
</transition>
```

В коде активности необходимо создать объект `TransitionDrawable`, вызвав метод `Resources.getDrawable ()` и передав ему идентификатор `drawable`-ресурса (`xml`-файла с корневым элементом `<transition>`). Код в листинге 2 создает экземпляр `TransitionDrawable` и устанавливает его как содержимое

объекта `ImageView` с помощью метода `setImageDrawable()`. Затем в методе `onClick` запускается плавный переход от первой картинки ко второй с помощью метода `startTransition()`, принимающего продолжительность перехода в миллисекундах, или от второй к первой с помощью метода `reverseTransition()`.

Листинг 2. Работа с `TransitionDrawable`

```
package com.example.drawables;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.content.res.ResourcesCompat;
import android.content.res.Resources;
import android.graphics.drawable.TransitionDrawable;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ImageView;

public class MainActivity extends AppCompatActivity implements
OnClickListener {
    private TransitionDrawable mTransition;
    private boolean flag = true;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ImageView image = (ImageView)findViewById(R.id.transition_image);
        image.setOnClickListener(this);
        Resources res = this.getResources();
        mTransition =
(TransitionDrawable)ResourcesCompat.getDrawable(res,
R.drawable.transition, null);
        image.setImageDrawable(mTransition);
    }
    @Override
    public void onClick(View v) {
        if (flag) {
            mTransition.startTransition(2000);
            flag = false;
        }
        else
        {
            mTransition.reverseTransition(2000);
            flag = true;
        }
    }
}
```

Класс ShapeDrawable






Объекты ShapeDrawable могут использоваться, если необходимо динамически рисовать двухмерную графику. Можно программно отрисовывать графические примитивы на объекте ShapeDrawable и применять различные стили.

ShapeDrawable — это подкласс Drawable. Поэтому можно использовать ShapeDrawable везде, где ожидается Drawable. Например, можно использовать объект ShapeDrawable для установки фона представления, передав его в метод setBackgroundDrawable () представления.

Поскольку ShapeDrawable имеет собственный метод draw (), можно создать подкласс View, который будет рисовать объект ShapeDrawable во время события onDraw ().

Для создания графических примитивов используются классы, производных от базового класса Shape. Эти классы приведены в таблице 2.

Таблица 2. Классы потомки Shape.

Класс	Описание	Пример
RectShape	Определяет примитив — прямоугольник.	
OvalShape	Определяет примитив — овал.	
ArcShape	Определяет примитив — дугу. Дуга начинается с заданного угла и движется по часовой стрелке, рисуя «кусочки пирога» (похоже на пироговую диаграмму).	
PathShape	Определяет геометрические пути, используя класс Path.	
RoundRectShape	Определяет примитив — прямоугольник с закругленными углами. При желании можно добавить вставной (закругленный) прямоугольник.	

Приведённые в таблице 2 классы можно рисовать на канве с помощью собственного метода рисования, но, если вместо этого передать RectShape, OvalShape, ArcShape, PathShape или RoundRectShape в объект ShapeDrawable, становятся доступны дополнительные возможности по управлению этими графическими объектами.

Для создания, например, прямоугольника нужно определить в конструкторе ShapeDrawable рисуемый графический примитив как RectShape. Также для объекта ShapeDrawable необходимо установить цвет и границы фигуры. Если не установить границы, то графический примитив не будет рисоваться. Если вы не установить цвет, фигура по умолчанию будет черной.

Класс RectShape можно использовать не только для рисования прямоугольников, но и для рисования горизонтальных или вертикальных линий. Для этого надо задать высоту или ширину прямоугольника в 1-2 пикселя с помощью методов setIntrinsicHeight () и setIntrinsicWidth () соответственно (листинг 3). С помощью метода getPaint().setColor() устанавливается цвет примитива.

Листинг 3. Создание RectShape.

```
ShapeDrawable d = new ShapeDrawable(new RectShape());
d.setIntrinsicHeight(2);
d.setIntrinsicWidth(150);
d.getPaint().setColor(Color.MAGENTA);
```

Прорисовка прямоугольника с закругленными сторонами несколько сложнее (рис. 1). Конструктор класса RoundRect (листинг 4) для рисования прямоугольника с закругленными сторонами принимает три параметра:

- float[] outerRadii — массив из восьми значений радиуса закругленных сторон внешнего прямоугольника. Первые два значения для верхнего левого угла, остальные пары отсчитываются по часовой стрелке от первой пары. Если на внешнем прямоугольнике закруглений не будет, передается null;

- `RectF inset` — объект класса `RectF`, который определяет расстояние от внутреннего прямоугольника до каждой стороны внешнего прямоугольника. Конструктор `RectF` принимает четыре параметра: пары `X` и `Y` координат левого верхнего и правого нижнего углов внутреннего прямоугольника. Если внутреннего прямоугольника нет, передается `null`;
- `float[] innerRadii` — массив из восьми значений радиуса закруглений углов для внутреннего прямоугольника. Первые два значения — координаты верхнего левого угла, остальные пары отсчитываются по часовой стрелке от первой пары. Если на внутреннем прямоугольнике закруглений не будет, передается `null`.

Листинг 4. Создание `RoundRectShape`.

```
float[] outR = new float[] { 10, 10, 50, 50, 20, 20, 20, 20 };
RectF rectF = new RectF(10, 10, 10, 10);
float[] inR = new float[] { 40, 40, 40, 40, 40, 40, 40, 40 };
ShapeDrawable d = new ShapeDrawable(new RoundRectShape(outR, rectF,
inR));
d.setIntrinsicHeight(300);
d.setIntrinsicWidth(500);
d.getPaint().setColor(Color.BLUE);
```

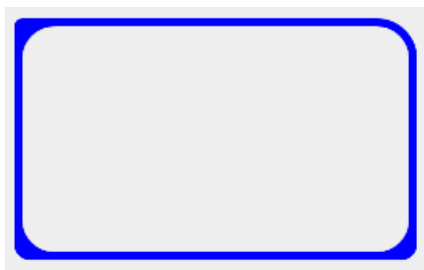


Рисунок 1. Примитив `RoundRectShape`.

Класс `Path` формирует множественный контур геометрических путей, состоящих из прямых линейных сегментов, квадратичных и кубических кривых. Для установки точек и перемещения линий (или кривых) используются открытые методы `moveTo()` и `lineTo()`. Например, так с помощью класса `Path` можно нарисовать пятиконечную звезду (листинг 5). Метод `moveTo(x, y)` устанавливает начало контура в точку (x, y) . Метод `lineTo(x, y)` рисует линию от последней точки к указанной точке.

Листинг 5. Создание Path.

```

Path p = new Path();
p.moveTo(50, 0);
p.lineTo(25,100);
p.lineTo(100,50);
p.lineTo(0,50);
p.lineTo(75,100);
p.lineTo(50,0);
ShapeDrawable d = new ShapeDrawable(new PathShape(p, 100, 100));
d.setIntrinsicHeight(200);
d.setIntrinsicWidth(200);
d.getPaint().setColor(Color.RED);
d.getPaint().setStyle(Paint.Style.STROKE);

```

Класс `ArcShape` создаёт графический примитив в форме дуги (листинг 6). Конструктор класса принимает два параметра: `ArcShape(float startAngle, float sweepAngle)`. Первый параметр в конструкторе — угол начала прорисовки дуги в градусах, второй — угловой размер дуги в градусах.

Листинг 6. Создание ArcShape.

```

ShapeDrawable d = new ShapeDrawable(new ArcShape(0, 255));
d.setIntrinsicHeight(200);
d.setIntrinsicWidth(200);
d.getPaint().setColor(Color.RED);

```

Класс `ShapeDrawable`, подобно многим другим типам `Drawable`, входящим в пакет `android.graphics.drawable`, позволяет определять различные свойства рисунка с помощью набора открытых методов, например `setAlpha()` — для установки прозрачности, `setColorFilter()` и т. д.

Создание ShapeDrawable в xml-файле

Для создания `ShapeDrawable` можно использовать xml-файл, который размещается в папке `res/drawable`. В качестве корневого элемента используется элемент `<shape>`, в котором в атрибуте `android:shape` указывается одна из четырёх форм: прямоугольник (`rectangle`), овал (`oval`), горизонтальная линия по ширине родительского контейнера (`line`) или кольцо (`ring`).

Если в атрибуте `android:shape` указано кольцо (листинг 7), то также могут использоваться дополнительные атрибуты для элемента `<shape>` (таблица 3).

Таблица 3. Атрибуты, задающиеся для кольца.

Атрибут	Описание
<code>android:innerRadius</code>	Радиус внутренней части кольца (отверстие посередине кольца).
<code>android:innerRadiusRatio</code>	Радиус внутренней части кольца, выраженный как отношение к ширине кольца. Например, если <code>android:innerRadiusRatio = "5"</code> , то внутренний радиус равен ширине кольца, деленной на 5. Это значение переопределяется <code>android:innerRadius</code> . Значение по умолчанию — 9.
<code>android:thickness</code>	Толщина кольца.
<code>android:thicknessRatio</code>	Толщина кольца, выраженная как отношение к ширине кольца. Например, если <code>android:thicknessRatio = "2"</code> , тогда толщина равна ширине кольца, деленной на 2. Это значение переопределяется <code>android:innerRadius</code> . Значение по умолчанию — 3.
<code>android:useLevel</code>	"true", если <code>shape</code> используется как <code>LevelListDrawable</code> .

Листинг 7. Фигура — кольцо.

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="ring"
    android:innerRadius="50dp"
    android:thickness="20dp"
    android:useLevel="false">
    <solid android:color="#0FE4D0" />
</shape>
```

Код приведенный в листинге 7 создаст кольцо, представленное на рисунке 2.



Рисунок 2. Фигура — кольцо.

В корневом элементе `<shape>` могут располагаться вложенные элементы со своими атрибутами, задающие внешний вид примитивов (таблица 4).

Таблица 4. Атрибуты, задающие внешний вид примитивов.

Элементы и их атрибуты	Описание
<code><corners></code>	<i>Создает закругленные углы фигуры. Применяется только тогда, когда фигура представляет собой прямоугольник.</i>
<code>android:radius</code>	Радиус закругления для всех углов. Для каждого угла это отменяется следующими далее атрибутами.
<code>android:topLeftRadius</code>	Радиус закругления для левого верхнего угла.
<code>android:topRightRadius</code>	Радиус закругления для правого верхнего угла.
<code>android:bottomLeftRadius</code>	Радиус закругления для левого нижнего угла.
<code>android:bottomRightRadius</code>	Радиус закругления для правого нижнего угла.
<code><gradient></code>	<i>Задаёт градиентную заливку для фигуры.</i>
<code>android:angle</code>	Угол наклона в градусах. Если равен 0, наклон слева направо, если 90, то снизу-вверх. Должно быть кратно 45. По умолчанию равно 0.
<code>android:centerX</code>	Относительное положение по оси X центра градиента. Принимает значение в диапазоне от 0 до 1.
<code>android:centerY</code>	Относительное положение по оси Y центра градиента. Принимает значение в диапазоне от 0 до 1.
<code>android:centerColor</code>	Необязательный цвет, который находится между начальным и конечным цветами (шестнадцатеричное значение или цветовой ресурс).
<code>android:endColor</code>	Конечный цвет в виде шестнадцатеричного значения или цветового ресурса.
<code>android:gradientRadius</code>	Радиус градиента. Применяется только при <code>android:type = "radial"</code> .
<code>android:startColor</code>	Начальный цвет в виде шестнадцатеричного значения или цветового ресурса.
<code>android:type</code>	Ключевое слово, определяющее тип применяемого градиентного узора (linear, radial, sweep).
<code>android:useLevel</code>	"true", если shape используется как LevelListDrawable.
<code><padding></code>	<i>Задаёт величину отступа внутри фигуры. Это актуально, например, для TextView, если shape будет использоваться в качестве фона. Отступ будет учтен при размещении текста.</i>
<code>android:left</code>	Отступ слева.
<code>android:top</code>	Отступ сверху.
<code>android:right</code>	Отступ справа.
<code>android:bottom</code>	Отступ снизу.
<code><size></code>	<i>Размер фигуры. По умолчанию фигура масштабируется до размера контейнера View пропорционально определенным здесь размерам. Когда вы используете shape в ImageView, вы можете</i>

	<i>ограничить масштабирование, установив android:scaleType="center".</i>
android:height	Высота фигуры.
android:width	Ширина фигуры.
<solid>	<i>Сплошной цвет для заливки фигуры.</i>
android:color	Цвет для заливки, как шестнадцатеричное значение или цветовой ресурс.
<stroke>	<i>Линия обводки для фигуры. Обязательно задаётся для фигуры line.</i>
android:width	Толщина линии.
android:color	Цвет линии в виде шестнадцатеричного значения или цветового ресурса.
android:dashGap	Расстояние между штрихами. Допустимо, только если установлен атрибут android:dashWidth.
android:dashWidth	Размер каждой пунктирной линии. Допустимо, только если установлен атрибут android:dashGap.

В листингах 8 и 9 приведены примеры различных фигур (рис. 3, 4).

Листинг 8. Фигура — овал.

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="oval">
    <gradient
        android:startColor="#00BCD4"
        android:endColor="#80FF00FF"
        android:centerX="0"
        android:type="radial"
        android:gradientRadius="150dp"/>
    <size
        android:height="300dp"
        android:width="300dp"/>
</shape>
```



Рисунок 3. Фигура — овал.

Листинг 9. Фигура — прямоугольник.

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <gradient
        android:startColor="#009688"
        android:endColor="#009688"
        android:centerColor="#9C27B0"
        android:type="sweep"/>
    <corners
        android:radius="20dp" />
    <size
        android:height="300dp"
        android:width="300dp"/>
    <stroke
        android:color="#9C27B0"
        android:width="3dp"
        android:dashGap="10dp"
        android:dashWidth="10dp"/>
</shape>
```



Рисунок 4. Фигура — прямоугольник.

Рисование графики на канве

Рисование на канве (холсте) лучше всего использовать, когда окно приложения должно регулярно перерисовываться во время работы приложения. Существует два способа реализации рисования на канве:

1. в основном потоке программы, в котором запускается Activity, создаётся собственный компонент View, затем вызывается метод `invalidate ()` и обрабатывается создание графики в методе обратного вызова `onDraw ()`;
2. в отдельном потоке — через объект `SurfaceView`. В этом случае не требуется вызывать метод `invalidate ()`.

Класс Canvas предоставляет методы для рисования, которые отображают графические примитивы на исходном растровом изображении. При этом надо сначала подготовить кисть (класс Paint), который позволяет указывать, как именно графические примитивы должны отображаться на растровом изображении (цвет, стиль и т.д.).

Canvas работает с пикселями, поэтому следует заботиться о конвертации единиц dp в px и наоборот при необходимости. Начало координат находится в левом верхнем углу.

Класс Canvas содержит набор методов для рисования для различных классов графики, например drawBitmap(), drawRect(), drawText() и другие. В таблице 5 приведены часто использующиеся методы.

Таблица 5. Методы класса Canvas для рисования

Метод	Описание
drawARGB(int a, int r, int g, int b)	Заполняет весь холст указанным цветом в формате ARGB.
drawRGB(int r, int g, int b)	Заполняет весь холст указанным цветом в формате RGB.
drawColor(long color, BlendMode mode) drawColor(int color, BlendMode mode)	Заполняет весь холст указанным цветом и с указанным режимом смешивания.
drawColor(long color) drawColor(int color)	Заполняет весь холст указанным цветом.
drawArc	Рисует дугу.
drawBitmap	Рисует растровое изображение, используя указанную матрицу.
drawLine	Рисует линию с указанными координатами начала и конца x, y, используя указанную краску.
drawCircle	Рисует окружность с определенным радиусом вокруг заданной точки.
drawOval	Рисует овал на основе прямоугольной области.
drawPaint	Закрашивает весь холст с помощью заданного объекта Paint.
drawPath	Рисует указанный контур (путь), используя указанную краску.
drawPoint	Рисует точку в заданном месте.
drawRect	Рисует прямоугольник.
drawRoundRect	Рисует прямоугольник с закругленными углами.

drawText	Рисует текст в заданной точке и с использованием стиля, определённого в объекте Paint.
drawTextOnPath	Рисует текст, который отображается вокруг определенного контура.
drawVertices	Рисует набор треугольников в виде совокупности вершинных (вертексных) точек.

Канва фактически является поверхностью, на которой ваша графика будет рисоваться. Когда прорисовка выполняется в пределах метода обратного вызова `View.onDraw()`, система передает в качестве параметра объект `Canvas`. Также можно получить объект `Canvas` вызовом метода `SurfaceHolder.lockCanvas()` при работе с объектом `SurfaceView`.

Система Android вызывает метод `onDraw()` по мере необходимости. Каждый раз, когда изображение на канве представления требует перерисовки, необходимо вызывать метод `invalidate()`. Он требует от системы обновления представления, поэтому система вызывает метод `onDraw()`.

Класс SurfaceView

Класс `SurfaceView` предоставляет специальную поверхность для рисования, встроенную в иерархию представлений. Можно управлять форматом этой поверхности и её размером.

По оси Z `SurfaceView` располагается так, чтобы находиться за окном, содержащим `SurfaceView`. Чтобы поверхность `SurfaceView` отображалась, она «пробивает» отверстие в этом окне.

Иерархия представлений заботится о правильном расположении объектов `View`, которые могут появляться поверх `SurfaceView`. Это может использоваться для размещения представлений, например, кнопок, поверх поверхности, однако это может повлиять на производительность, поскольку полное альфа-смешивание будет выполняться каждый раз, когда поверхность изменяется.

Доступ к поверхности предоставляется через интерфейс `SurfaceHolder`, который можно получить, вызвав `getHolder ()`.

Поверхность будет создана, пока отображается окно `SurfaceView`; следует реализовать методы `surfaceCreated` и `surfaceDestroyed`, чтобы обнаруживать, когда поверхность создается и уничтожается при отображении и скрывании окна.

Особенность класса `SurfaceView` в том, что он предоставляет отдельную область для рисования, действия с которой должны быть вынесены в отдельный поток приложения. Таким образом, приложению не нужно ждать, пока система будет готова к отрисовке всей иерархии представлений. Вспомогательный поток может использовать `Canvas` нашего `SurfaceView` для отрисовки с той скоростью, которая необходима.

Вся реализация сводится к двум основным моментам:

1. Создание класса, унаследованного от `SurfaceView` и реализующего интерфейс `SurfaceHolder.Callback`
2. Создание потока, который будет управлять отрисовкой.

Для начала, необходимо создать новый класс, расширяющий класс `SurfaceView` и реализующий интерфейс `SurfaceHolder.Callback`. В этом классе следует реализовать три метода: `surfaceCreated ()`, `surfaceChanged ()` и `surfaceDestroyed ()`, вызываемые соответственно при создании поверхности для рисования, её изменении и уничтожении.

Работа с поверхностью осуществляется не напрямую через созданный класс, а с помощью объекта `SurfaceHolder`. Получить его можно вызовом метода `getHolder ()`. Именно этот объект будет предоставлять нам объект `Canvas` для отрисовки. В конструкторе класса получаем объект `SurfaceHolder` и с помощью метода `addCallback ()` указываем, что хотим получать соответствующие обратные вызовы. Затем необходимо переопределить каждый из методов `SurfaceHolder.Callback` внутри созданного класса потомка `SurfaceView` (листинг 10).

Листинг 10. Класс, расширяющий класс SurfaceView.

```
package com.example.surfaceviewapp;
import android.content.Context;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class MySurfaceView extends SurfaceView implements
SurfaceHolder.Callback {
    public MySurfaceView(Context context) {
        super(context);
        getHolder().addCallback(this);
    }

    @Override
    public void surfaceChanged(SurfaceHolder holder, int format, int
width, int height) {
    }

    @Override
    public void surfaceCreated(SurfaceHolder holder) {
    }

    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {
    }
}
```

Создание потока следует выполнять в методе `surfaceCreated ()`.

Для реализации потока, отвечающего за отрисовку, создаётся класс, унаследованный от `Thread`. Его конструктор принимает два параметра: `SurfaceHolder` и `Resources` для загрузки картинки, которая будет рисовать на экране. В классе, унаследованном от `Thread`, потребуется переменная-флаг, указывающая на то, что производится отрисовка и метод для установки этой переменной. Также потребуется переопределить метод `run ()`.

Чтобы рисовать на `Surface Canvas` из второго потока, нужно передать потоку `SurfaceHolder` и получить `Canvas` вызовом `lockCanvas ()`. После этого можно взять `Canvas`, предоставленный `SurfaceHolder`, и сделать на нем все нужные отрисовки. Как только отрисовка завершена вместе с `Canvas`, вызывается метод `unlockCanvasAndPost ()`, в который передаётся объект `Canvas`. Каждый раз, когда нужно перерисовать, выполняется последовательность захвата (locking) и освобождения (unlocking) `Canvas`.

На каждом проходе получения Canvas от SurfaceHolder будет возвращено предыдущее состояние Canvas. Чтобы правильно анимировать графику, нужно перерисовать всю поверхность. Например, можно очистить предыдущее состояние Canvas путем заливки цветом с `drawColor()` или установкой фоновой картинке с `drawBitmap()`. Иначе пользователь увидит выполненные ранее отрисовки.

Создание анимации представлений

Создание анимации представлений в Android-приложениях осуществляется с использованием двумерной графической библиотеки анимации на канве и объектах View android.view.animation.packages.

Анимация в Android представлена двумя видами:

- Tween Animation — анимация, выполняемая в виде простых преобразований объектов;
- Frame Animation — кадровая анимация.

Основные классы анимации преобразованием:

- AnimationSet — класс, представляющий группу анимаций, которые должны запускаться вместе. Если класс AnimationSet устанавливает какие-либо свойства, эти свойства наследуют и объекты, входящие в группу.
- AlphaAnimation — класс анимации, который управляет прозрачностью объекта.
- RotateAnimation — класс анимации, который управляет вращением объекта.
- ScaleAnimation — класс анимации, который управляет масштабированием объекта.
- TranslateAnimation — класс анимации, который управляет позиционированием объекта.

XML-файлы анимации располагают в каталоге `res/anim/` Android-проекта. Файл с анимацией должен иметь единственный корневой элемент,

определяющий какой тип анимации будет создан: <set>, <alpha>, <scale>, <translate> или <rotate>.

В листинге 11 приведён пример всех типов анимации.

Листинг 11. Анимация преобразованием.

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false">
    <alpha
        android:fromAlpha="0.0"
        android:toAlpha="1.0"
        android:startOffset="0"
        android:duration="5000">
    </alpha>
    <scale
        android:duration="2500"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fromXScale="1.0"
        android:fromYScale="1.0"
        android:toXScale="2.0"
        android:toYScale="2.0" />
    <rotate
        android:fromDegrees="0"
        android:toDegrees="360"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="5000" />
    <translate
        android:toYDelta="-100"
        android:fillAfter="false"
        android:duration="2500" />
    <scale
        android:duration="2500"
        android:startOffset="2500"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fromXScale="1.0"
        android:fromYScale="1.0"
        android:toXScale="0.5"
        android:toYScale="0.5" />
    <translate
        android:toYDelta="100"
        android:fillAfter="false"
        android:duration="2500"
        android:startOffset="2500" />
</set>
```

Для загрузки и запуска анимации используются методы AnimationUtils.loadAnimation () и startAnimation () соответственно. В

листинге 12 приведён пример запуска анимации, описанной в ресурсе `total.xml`, для объекта `ImageView`.

Листинг 12. Запуск анимации.

```
mImage = (ImageView)findViewById(R.id.image);  
animation = AnimationUtils.loadAnimation(this, R.anim.total);  
animation.setAnimationListener(this);  
mImage.startAnimation(animation);
```

В классе `Animation` есть вложенный интерфейс `AnimationListener`, в котором объявлены три метода обратного вызова:

- `onAnimationEnd()`;
- `onAnimationRepeat()`;
- `onAnimationStart()`.

В этих методах можно реализовать код обработки события запуска, окончания и перезапуска анимации.

Кадровая (фреймовая) анимация — традиционная анимация, которая создается последовательностью различных изображений. Основой для кадровой анимации является класс `AnimationDrawable`.

Для кадровой анимации XML-файл состоит из корневого элемента `<animation-list>` и дочерних узлов `<item>`, каждый из которых определяет кадр, который имеет две составляющие:

- графический ресурс для кадра (атрибут `android:drawable`);
- продолжительность кадра (атрибут `android:duration`).

В элементе `<animation-list>` в атрибуте `android:oneshot` указывается, будет ли анимация циклична. Если в этом атрибуте указано значение `true`, то анимация повторится только один раз и после остановки будет содержать последний кадр. Если `false`, то анимация будет циклической.

Задание

Разработайте приложение, содержащие различные объекты ShapeDrawable, анимированные с использованием анимации преобразованием и кадровой анимации.

Контрольные вопросы:

1. Для чего используются объекты Drawable?
2. Для чего используется класс TransitionDrawable?
3. Для чего используется класс ShapeDrawable?
4. Как осуществляется объявление ShapeDrawable в xml-файле?
5. Какие свойства графических фигур задаются с помощью атрибутов в xml-файле?
6. Чем рисование на канве отличается от использования объектов Drawable для рисования?
7. Для чего используется класс SurfaceView?
8. Что представляет собой объект Canvas?
9. Для чего используется класс SurfaceHolder?
10. Для чего используется класс Tween Animation?
11. Какие типы анимации можно реализовать с помощью класса Tween Animation?
12. Для чего используется класс Frame Animation?
13. Как задать кадровую анимацию?