

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

Брянский государственный технический университет

**Утверждаю**

**Ректор университета**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_О.Н.Федонин**

**«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2017 г.**

**ПРОГРАММИРОВАНИЕ МОБИЛЬНЫХ СИСТЕМ**

**Многоэкранные мобильные приложения с использованием XML ресурсов**

**Методические указания**

**к выполнению лабораторной работы №2**

**для студентов очной формы обучения по направлениям подготовки 02.03.03 – «Математическое обеспечение и администрирование информационных систем», 09.03.01 – «Информатика и вычислительная техника» и 09.03.04 – «Программная инженерия»**

**Брянск 2017**

УКД 004.43

Программирование мобильных систем. Многоэкранные мобильные приложения с использованием XML ресурсов. [Текст] + [Электронный ресурс]: Методические указания к выполнению лабораторной работы №2 для студентов очной формы обучения по направлениям подготовки 02.03.03 – «Математическое обеспечение и администрирование информационных систем», 09.03.01 – «Информатика и вычислительная техника» и 09.03.04 – «Программная инженерия». – Брянск: БГТУ, 2017. – 33с.

Разработал:

Д.Н.Панус

ст.преп.

Рекомендовано кафедрой «Информатика и программное обеспечение» БГТУ (протокол №2 от 16.09.2016)

Научный редактор Д.А.Коростелев

Редактор издательства Л.И.Афонина

Компьютерный набор Д.Н. Панус

Темплан 2017 г., п.273

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписано в печать 23.09.17. Формат 60х84 1/16 Бумага офсетная. Офсетная печать. Усл.печ.л. 1,8. Уч.-изд.л. 1,8 Тираж 1 экз. Заказ Бесплатно.

Издательство Брянского государственного технического университета

241035, Брянск, бульвар 50-летия Октября, 7, БГТУ. 58-82-49.

Лаборатория оперативной полиграфии БГТУ, ул. Институтская, 16.

# Цель работы

Целью работы является приобретение навыка разработки интерфейсов мобильных приложений, при помощи описания в виде XML ресурсов.

Продолжительность работы – 2 часа.

# Порядок выполнения работы

1. Ознакомится с возможностями IDE по проектированию интерфейсов.
2. Выяснить программные возможности, предоставляемые тестовой средой (телефон или эмулятор ОС Android).
3. Разобраться, какие компоненты, используемые в ОС Android, необходимо применить в данной лабораторной работе.
4. Ознакомиться со структурой файла манифеста приложения и описать в нем требуемый от приложений функционал, а также требования, предъявляемые к аппаратуре.
5. Ознакомиться с правилами хранения ресурсов приложения. Реализовать в дереве проекта требуемую иерархию файловой системы.
6. Описать в виде XML требуемые ресурсы приложения: анимации, меню, рисуемые объекты.
7. Произвести сборку готового приложения в виде \*.apk файла приложения средствами IDE.
8. Продемонстрировать полученный результат преподавателю.

# Отделение ресурсов от кода программы

Неважно, для какой системы ведется разработка, такие ресурсы, как изображения и строковые константы, всегда лучше держать за пределами исходного кода. Android поддерживает выделение ресурсов во внешние файлы, начиная с простых значений: строк и цветов, заканчивая более сложными данными вроде изображений (объектов *Drawable*), анимации и визуальных стилей. Внешние ресурсы легче поддерживать, обновлять и контролировать. Также можно описывать альтернативные ресурсы для поддержки различного аппаратного обеспечения и локализации.

Android динамически выбирает данные из дерева ресурсов, содержащего разные значения для разных аппаратных конфигураций, языков и регионов. Это позволяет описывать уникальные значения для конкретных языков, стран, экранов и клавиатур. При запуске приложения Android автоматически выберет ресурс с соответствующими данными. Это дает возможность изменять разметку, учитывая размер экрана и его ориентацию, выводить разные текстовые подсказки в зависимости от языка и страны.

# Создание ресурсов

Ресурсы приложения хранятся в каталоге res/ внутри дерева проекта. Каждый тип ресурсов представлен в виде подкаталога, содержащего соответствующие данные. При создании нового проекта дополнение ADT автоматически добавит в него каталог *res* с подкаталогами *values*, *drawable-ldpi*, *drawable-mdpi*, *drawable-hdpi* и *layout*. В них будут храниться следующие ресурсы: разметка по умолчанию, значок приложения и определения строковых констант. Каждый из каталогов drawable-\* содержит разные значки ‒ для экранов с низким, средним и высоким значением DPI.

Разные каталоги предусмотрены для девяти главных типов ресурсов: простых значений, ресурсов Drawable, менеджеров компоновки, анимации, стилей, меню, настроек поиска, XML и «сырых» (необработанных) данных. При сборке приложения эти ресурсы скомпилируются самым эффективным образом и включатся в программный пакет. При этом генерируется файл для класса R, содержащий ссылки на все ресурсы проекта, что позволяет ссылаться на ресурсы внутри кода программы. Это дает одно преимущество ‒ проверку синтаксиса во время разработки. Имена файлов для ресурсов должны состоять исключительно из букв в нижнем регистре, чисел, а также символов . (точка) и \_ (нижнее подчеркивание).

### Создание простых значений

Поддерживаются простые значения ‒ строки, цвета, размеры и массивы (строковые и целочисленные), эти данные хранятся в формате XML внутри каталога res/values. Используя теги, указывайте типы хранимых значений, как показано на примере простого XML-файла.

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

<resources>

<string name="app\_name">To Do List</string>

<color name="app\_background">#FF0000FF</color>

<dimen name="default\_border">5px</dimen>

<array name="string\_array">

<item>Item 1</item>

<item>Item 2</item>

<item>Item 3</item>

</array>

<array name="integer\_array">

<item>3</item>

<item>2</item>

<item>1</item>

</array>

</resources>

В этом примере содержатся все доступные типы простых значений. Каждый тип ресурсов принято хранить в отдельном файле, например, файл *res/values/strings.xml* включает только строковые константы.

### Строки

Внешние строковые ресурсы помогают поддерживать совместимость внутри приложения, упрощая процесс создания локализованных версий.

Строковые ресурсы обозначаются тегом *<string>,* как показано в следующем фрагменте:

*<string name="stop\_message">Stop.</string>*

Android поддерживает простую текстовую разметку, поэтому можно использовать теги <b>, <i> и <u> из языка HTML, чтобы выделять части текста полужирным, наклонным или подчеркнутым стилем соответственно:

*<string name="stop\_message"><b>Stop.</b></string>*

Можно использовать строковые ресурсы в качестве входящих параметров для метода *String.format*. Однако String.format не поддерживает стилизацию текста, описанную выше. Чтобы форматировать строку, при создании ресурса экранируйте HTML-теги:

*<string name="stop\_message">&lt;b>Stop&lt;/b>. %1$s</string>*

Внутри кода используйте метод *Html.fromHtml*, чтобы преобразовать эту строку обратно в форматированную последовательность символов:

String rString = getString(R.string.stop\_message);

String fString = String.format(rString, "Collaborate and listen.");

CharSequence styledString = Html.fromHtml(fString);

### Цвета

Для описания цветовых ресурсов применяется тег *<color>*. Цвет указывается помощью символа #, за которым следуют альфа-канал (необязательно) и значения для красного, зеленого и синего цветов в виде одного или двух шестнадцатеричных чисел. Поддерживаются следующие форматы записи: #RGB, #RRGGBB, #ARGB, #AARRGGBB.

В примере описываются два цвета: полностью непрозрачный синий и полупрозрачный зеленый:

<color name="opaque\_blue">#00F</color>

<color name="transparent\_green">#7700FF00</color>

### Размеры

Ссылки на размеры чаще всего встречаются внутри ресурсов со стилями и разметкой. Они пригодятся при создании констант — толщины рамки или высоты шрифта.

Чтобы описать ресурс, используйте тег *<dimen>*, указывая масштаб и один из видов размерности: *px* (экранные пикселы), *in* (физические дюймы), *pt* (физические точки), *mm* (физические миллиметры), *dp* (аппаратно-независимые пикселы, которые вычисляются относительно экрана с плотностью 160 dpi), *sp* (пикселы, не зависящие от масштаба).

В итоге можно описывать размеры не только в абсолютных, но и в относительных значениях, которые зависят от разрешения и плотности экрана, упрощая тем самым масштабирование интерфейса на разных устройствах. В следующем примере указаны размеры для большого шрифта и стандартной рамки:

<dimen name="standard\_border">5dp</dimen>

<dimen name="large\_font\_size">16sp</dimen>

### Визуальные стили и темы

Ресурсы со стилями позволяют поддерживать единство внешнего вида приложения с помощью атрибутов, используемых Представлениями. Чаще всего визуальные стили и темы используются для хранения цветовых значений и шрифтов для программы.

Вы можете легко менять внешний вид приложения, указывая различные стили в качестве темы в манифесте своего проекта.

Чтобы создать стиль, используйте тег *<style>*, включающий атрибут *name*, а также один или несколько вложенных узлов *item*. Каждый тег *item*, в свою очередь, также должен иметь атрибут *name*, содержащий тип описываемого значения (например, размер шрифта или цвет). Внутри тега должно храниться само значение. Пример описания визуального стиля показан в следующем фрагменте:

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

<resources>

<style name="StyleName">

<item name="attributeName">value</item>

</style>

</resources>

Тег *<style>* поддерживает наследование с помощью атрибута *parent*, благодаря чему стили можно свободно варьировать.

В следующем примере демонстрируются два стиля (могут быть использованы в качестве визуальной темы): базовый, описывающий несколько свойств для текста, и дочерний, изменяющий ранее заданный шрифт на более мелкий:

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

<resources>

<style name="BaseText">

<item name="android:textSize">14sp</item>

<item name="android:textColor">#111</item>

</style>

<style name="SmallText" parent="BaseText">

<item name="android:textSize">8sp</item>

</style>

</resources>

### Ресурсы Drawable

Ресурсы *Drawable* содержат растровые и растягиваемые (*NinePatch*) изображения. В эту категорию также входят сложные композитные ресурсы, такие как *LevelListDrawables* и *StateListDrawables*, которые могут быть описаны в формате XML.

Все ресурсы *Drawable* хранятся в виде отдельных файлов в каталоге *res/drawable*. Идентификаторами для них служат имена файлов в нижнем регистре без расширения. Рассмотрим примеры ресурсы *Drawable* более подробно.

Android включает простые ресурсы для рисования, которые можно полностью описать в формате XML. Это касается классов *ColorDrawable*, *ShapeDrawable* и *GradientDrawable*. Данные ресурсы хранятся в каталоге *res/drawable* и могут быть идентифицированы в коде приложения по именам файлов, записанным в нижнем регистре.

Если описывать эти ресурсы в формате XML и указывать атрибуты для них с помощью аппаратно-независимых пикселов (*density-independent pixels*), система сможет их плавно масштабировать. Как и в случае с векторной графикой, эти ресурсы могут динамически масштабироваться, отображаясь корректно и без артефактов при любых размерах и разрешениях экрана, независимо от плотности пикселов. Исключение ‒ ресурс *GradientDrawable*, радиус для которого должен быть указан в пикселах.

**ColorDrawable** ‒ простейший ресурс для рисования, он позволяет указывать свойство изображения, основанное на единственном сплошном цвете. ColorDrawable описывается в виде XML-файлов (хранящихся в каталоге с ресурсами) с помощью тега *<color>*. Ниже показан код для ресурса, описывающего сплошной красный цвет.

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

android:color="#FF0000"

/>

Данный вид ресурсов позволяет описывать простые геометрические фигуры, указывая их размеры, фон и контур с помощью тега *<shape>*.

Каждый такой тег состоит из типа (указывается с помощью атрибута *android:shape*), атрибутов, определяющих размер фигуры, и дочерних узлов, в которых задаются значения для отступов, контура (или очертания) и фона.

На сегодняшний день Android поддерживает несколько типов фигур, задать которые можно в атрибуте *android:shape*:

* **oval**. Простой овал.
* **rectangle**. Поддерживает также вложенный тег *<corners>*, с помощью которого можно создать прямоугольник с закругленными углами (используя атрибут *radius*).
* **ring** (кольцо). Поддерживает атрибуты *innerRadius* и thickness, с помощью которых задаются внутренний радиус кольца и его толщина соответственно. В качестве альтернативы вы можете использовать атрибуты *innerRadiusRatio* и/или *thicknessRatio*, чтобы указать те же параметры в виде значений, пропорциональных ширине (где внутренний радиус, равный четверти ширины, будет использовать значение 4).

Вложенный тег *<stroke>* используется, чтобы с помощью атрибутов *width* и *color* задать контур для своих фигур. Также можно добавить узел *<padding>*, чтобы задать отступ при позиционировании фигуры на Холсте. Кроме того, можно включить в описание дочерний тег для определения фонового цвета. В простейшем случае это использование узла *<solid>* в сочетании с атрибутом *color*, который описывает сплошной цвет заливки.

Ниже приводится пример ресурса с фигурой (прямоугольником), которая имеет сплошной фон, закругленные углы, отступ от каждой грани размером 10 dp и такой же толщины контур.

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

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

android:shape="rectangle">

<solid

android:color="#f0600000"/>

<stroke

android:width="10dp"

android:color="#00FF00"/>

<corners

android:radius="15dp" />

<padding

android:left="10dp"

android:top="10dp"

android:right="10dp"

android:bottom="10dp"

/>

</shape>

**GradientDrawable** позволяет создавать сложные градиентные заливки. Каждый градиент описывает плавный переход между двумя или тремя цветами с помощью линейного/радиального алгоритма или же используя метод развертки. *GradientDrawable* описывается в виде тега *<gradient>*, находясь внутри определения ресурса *ShapeDrawable*. Каждый ресурс с градиентом должен содержать как минимум по одному атрибуту *startColor* и *endColor*. Атрибут *middleColor* необязателен. Используя атрибут *type*, можно описать свой градиент.

*Композитные ресурсы для рисования* используются для объединения и манипулирования другими ресурсами. Внутри описания композитных ресурсов могут быть использованы растровые изображения, геометрические фигуры и цвета. Кроме того, сами композитные ресурсы можно использовать один внутри другого, назначая их для Представлений точно так же, как и любые другие виды ресурсов.

*Ресурсы, описывающие преобразования*

Вы можете масштабировать и поворачивать ресурсы Drawable, используя классы ScaleDrawable и RotateDrawable. Такие преобразования чрезвычайно полезны при создании индикаторов выполнения задач или для добавления анимации к Представлениям.

* **ScaleDrawable**. Используйте атрибуты *scaleHeight* и *scaleWidth* внутри тега *<scale>*, чтобы описать высоту и ширину относительно границ оригинального объекта *Drawable*. Добавьте атрибут *scaleGravity* для изменения опорной точки масштабированного изображения.
* **RotateDrawable**. Используйте атрибуты *fromDegrees* и *toDegrees* внутри тега *<rotate>*, чтобы задать начальный и конечный углы поворота вокруг опорной точки. Опорная точка указывается с помощью атрибутов *pivotX* и *pivotY*, которые содержат процентные значения относительно ширины и высоты объекта *Drawable* соответственно. Запись делается в виде nn%.

Чтобы применить масштабирование и поворот в процессе выполнения программы, используйте метод *setLevel* из Представления, содержащего объект *Drawable*, переходя при этом от начального к конечному значению (от 0 до 10 000).

В процессе перехода уровень 0 представляет начальный угол (или наименьший масштаб). Уровень 10 000 указывает на завершение преобразования (конечный угол или самый большой масштаб).

Ниже описаны ресурсы *ScaleDrawable* и *RotateDrawable*

<!-- Ресурс RotationDrawable -->

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

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

android:drawable="@drawable/icon"

android:fromDegrees="0"

android:toDegrees="90"

android:pivotX="50%"

android:pivotY="50%"

/>

<!-- Ресурс ScaleDrawable -->

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

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

android:drawable="@drawable/icon"

android:scaleHeight="100%"

android:scaleWidth="100%"/>

В примере ниже показано, как управлять этими ресурсами в коде программы после того, как они были назначены для объекта *ImageView*.

ImageView rotatingImage = (ImageView)findViewById(R.

id.RotatingImageView);

ImageView scalingImage = (ImageView)findViewById(R.id.ScalingImageView);

// Поверните изображение на 50% от итогового угла.

rotatingImage.setImageLevel(5000);

// Масштабируйте изображение до 50% от его финального размера.

scalingImage.setImageLevel(5000);

**LayerDrawable** позволяет накладывать несколько объектов *Drawable* один поверх другого. Описав массив полупрозрачных объектов *Drawable*, вы можете создать сложную комбинацию динамических фигур и преобразований.

Кроме того, вы можете применять к *LayerDrawable* те же преобразования, которые были описаны в предыдущем разделе, а также сочетать их с ресурсами *StateListDrawable* и *LevelListDrawable*, речь о которых пойдет дальше.

Ниже описывается с помощью тега *<layerlist>,* внутри которого для каждого дочернего узла *<item>* используется атрибут *drawable*, указывающий на ресурс для наложения.

Каждый объект *Drawable* будет накладываться в соответствии со своим индексом ‒ первый элемент массива размещается в самом низу.

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

<layer-list xmlns:android="http://schemas.android.com/apk/res/android">

<item android:drawable="@drawable/bottomimage"/>

<item android:drawable="@drawable/image2"/>

<item android:drawable="@drawable/image3"/>

<item android:drawable="@drawable/topimage"/>

</layer-list>

**StateListDrawable** ‒ это композитный ресурс, позволяющий отображать разные объекты Drawable в зависимости от состояния Представления, для которого они были назначены.

Ресурс *StateListDrawable* используется в большинстве стандартных Представлений в Android, например в виде изображений кнопок или фона для обычных элементов списка *ListView*.

Чтобы описать *StateListDrawable*, создайте файл в формате XML, в котором указываются разные ресурсы Drawable для каждого состояния.

Используя **LevelListDrawable**, можно эффективно размещать ресурсы Drawable один поверх другого, указывая целочисленный индекс для каждого слоя.

Изображения формата **NinePatch** (растягивающиеся)‒ это PNG-файлы, где некоторые помеченные части могут быть растянуты. Они должны описываться в формате PNG и иметь расширение .9.png. Идентификатор ресурсов для *NinePatch* ‒ это имя файла без окончания .9.png.

### Разметка

Ресурсы с разметкой (или менеджеры компоновки) позволяют отделять уровень представления от бизнес-логики. С помощью разметки вы можете проектировать пользовательские интерфейсы в формате XML, вместо того чтобы описывать их в коде программы.

Чаще всего разметка применяется при описании пользовательского интерфейса для Активности. Создав разметку в формате XML, можно загрузить ее в Активность с помощью метода *setContentView* (как правило, внутри обработчика *onCreate*). Вы также можете получать ссылки на экземпляры разметки, содержащиеся в других ресурсах (например, разметка для каждой строки в элементе *ListView*).

Использование менеджеров компоновки ‒ рекомендуемый подход при проектировании пользовательских интерфейсов в Android. Отделяя разметку от кода программы, вы получаете возможность оптимизировать пользовательский интерфейс для различных аппаратных конфигураций, учитывая размеры экрана, ориентацию, наличие клавиатуры и сенсорного экрана. Каждый ресурс, описывающий разметку, хранится в отдельном файле в каталоге *res/layout*. Имя файла выступает как идентификатор ресурса.

В качестве примера менеджеров компоновки представлен ресурс, добавленный мастером создания проектов. Он содержит разметку *LinearLayout*, которая выступает в качестве контейнера для элемента *TextView*, отображающего приветствие Hello World!

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

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

android:orientation="vertical"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent">

<TextView

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:text="Hello World!"

/>

</LinearLayout>

### Анимация

Android поддерживает два вида анимации. Анимация, основанная на расчете промежуточных кадров, используется для поворачивания, перемещения, растягивания и затемнения Представлений. Также можно создавать пошаговую анимацию, чтобы последовательно выводить на экран изображения.

Описывая анимацию в виде внешнего ресурса, можно использовать одну и ту же последовательность в разных местах программы. Вы также получаете возможность выбирать анимацию в зависимости от аппаратных особенностей устройства или ориентации в пространстве.

**Анимация, основанная на расчете промежуточных кадров.** Каждый экземпляр анимации данного типа хранится в отдельном XML-файле внутри каталога *res/anim*. Аналогично разметке и объектам *Drawable* имена файлов с описанием анимации служат идентификаторами для ресурсов.

Анимацию можно задать в виде изменений параметров **alpha** (затемнение), **scale** (масштабирование), **translate** (перемещение) или **rotate** (поворот).

Ниже приведены допустимые атрибуты и значения, которые поддерживаются при описании анимации.

|  |  |  |
| --- | --- | --- |
| **Тип**  **анимации** | **Атрибуты** | **Допустимые значения** |
| Alpha | fromAlpha/toAlpha | Значения типа Float от 0 до 1 |
| Scale | fromXScale/toXScale | Значения типа Float от 0 до 1 |
| fromYScale/toYScale | Значения типа Float от 0 до 1 |
| pivotX/pivotY | Описывает центральную точку масштабирования в процентах от ширины/высоты —  от 0 % до 100 % |
| Translate | fromX/toY | Значения типа Float от 0 до 1 |
| fromY/toY | Значения типа Float от 0 до 1 |
| Rotate | fromDegrees/toDegrees | Значения типа Float от 0 до 360о |
| pivotX/pivotY | Описывает опорную точку для поворота в процентах от ширины/высоты — от 0 % до 100 % |

Можно комбинировать разные экземпляры анимации, используя тег *set*. Такой набор содержит одно или несколько анимационных преобразований и поддерживает различные дополнительные теги и атрибуты, с помощью которых можно указать, как и когда должен запускаться каждый экземпляр.

Ниже перечислены некоторые атрибуты, доступные для тега *set*:

* **duration** ‒ продолжительность анимации в миллисекундах;
* **startOffset** ‒ миллисекундная задержка перед началом анимации;
* **fillBefore** ‒ установите значение true, чтобы применить преобразование перед началом анимации;
* **fillAfter** ‒ установите значение true, чтобы применить преобразование после завершения анимации;
* **interpolator** ‒ описывает изменения в скорости эффекта.

Чтобы использовать один из них, укажите ссылку на системный ресурс с анимацией вида *android:anim/interpolatorName*.

В следующем примере показан анимационный набор, с помощью которого целевой элемент одновременно поворачивается на 360о, сжимается и исчезает:

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

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

android:interpolator="@android:anim/accelerate\_interpolator">

<rotate

android:fromDegrees="0"

android:toDegrees="360"

android:pivotX="50%"

android:pivotY="50%"

android:startOffset="500"

android:duration="1000" />

<scale

android:fromXScale="1.0"

android:toXScale="0.0"

android:fromYScale="1.0"

android:toYScale="0.0"

android:pivotX="50%"

android:pivotY="50%"

android:startOffset="500"

android:duration="500" />

<alpha

android:fromAlpha="1.0"

android:toAlpha="0.0"

android:startOffset="500"

android:duration="500" />

</set>

**Пошаговая анимация.** Она подразумевает создание последовательности объектов *Drawable*, каждый из которых будет отображаться в качестве фона для Представления на протяжении указанного промежутка времени.

В отличие от анимации, основанной на расчете промежуточных кадров, пошаговая хранится в виде ресурсов *Drawable* в каталоге *res/drawable*. Имена файлов (без расширения .xml) используются в качестве идентификаторов.

В следующем фрагменте представлена простая анимация, в ходе которой происходит перебор последовательности растровых ресурсов и вывод каждого из них на экран (отображаются полсекунды). Чтобы использовать этот пример, необходимо создать три новых ресурса с изображениями rocket (1, 2, 3).

<animation-list xmlns:android="http://schemas.android.com/apk/res/android"

android:oneshot="false">

<item android:drawable="@drawable/rocket1" android:duration="500" />

<item android:drawable="@drawable/rocket2" android:duration="500" />

<item android:drawable="@drawable/rocket3" android:duration="500" />

</animation-list>

### Меню

Меню можно описать в виде отдельных ресурсов в формате XML, вместо того чтобы создавать их прямо в коде приложения. Это поспособствует дальнейшему повышению автономности уровня представления вашей программы.

Данный вид ресурсов может использоваться для описания как главного (принадлежащего Активности), так и контекстного меню, и предоставляет те же свойства, какие доступны при создании меню программным способом. Меню, описанное в формате XML, загружается в виде программного объекта с помощью метода *inflate*, принадлежащего Сервису *MenuInflator*. Как правило, это происходит внутри обработчика *onCreateOptionsMenu*.

Описание каждого экземпляра меню хранится в отдельном файле в каталоге *res/menu*. Имена файлов автоматически становятся идентификаторами ресурсов. Именно такой способ определения меню наиболее предпочтителен в Android.

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

<menu xmlns:android="http://schemas.android.com/apk/res/android">

<item android:id="@+id/menu\_refresh"

android:title="Refresh" />

<item android:id="@+id/menu\_settings"

android:title="Settings" />

</menu>

### Спецификаторы ресурсов для указания размеров экрана и плотности пикселов

Спецификаторы для имен каталогов, которые можно использовать при добавлении альтернативных ресурсов, чтобы поддерживать разные характеристики экрана, такие как размер, плотность пикселов и соотношение сторон, приведены ниже:

* **Размер экрана.** Размер экрана относительно «стандартного» смартфона (например, G1 или Droid):small ‒ экран меньший, чем стандартные 3,2”, medium ‒ типичный размер экрана смартфона, large ‒ экран значительно больший, чем у типичного смартфона (например, экран планшетного компьютера или нетбука).
* **Плотность пикселов.** Описывает плотность пикселов на экране. Как правило, измеряется в точках на квадратный дюйм (dpi) и зависит от размеров экрана и его разрешения:
  + *ldpi* ‒ предназначен для хранения ресурсов, рассчитанных на экраны с низкой плотностью пикселов (100–140 dpi);
  + *mdpi* ‒ для экранов со средней плотностью пикселов (140–180 dpi);
  + *hdpi* ‒ для экранов с высокой плотностью пикселов (190–250 dpi);
  + *nodpi* ‒ для ресурсов, которые не должны масштабироваться, в зависимости от плотности пикселов на экране устройства.
* **Соотношение сторон.** Описывает отношение высоты экрана к его ширине: *long* ‒ для экранов, которые в альбомном режиме значительно шире, чем на стандартных смартфонах (таких как G1), *notlong* ‒ для экранов с обычным соотношением сторон.

Каждый из этих спецификаторов может быть использован в комбинации с любым другим или независимо от остальных.

### Определение поддерживаемых экранных размеров.

Пользовательский интерфейс некоторых приложений просто не получится оптимизировать для всех возможных типов экранов. Можно использовать тег *<supports-screens>* в манифесте, чтобы указать, на устройствах с какими экранами может работать ваша программа.

Ниже приведет код элемента манифеста, описывающий поддержку нормальных и больших экранов.

<supports-screens

android:smallScreens="false"

android:normalScreens="true"

android:largeScreens="true"

android:anyDensity="true"

/>

В данном случае маленьким экраном можно назвать любой дисплей с разрешением меньше, чем HVGA. Под большим экраном подразумевается такой, который значительно больше, чем у смартфона (например, у планшетных компьютеров). Экран нормальных размеров имеет большинство смартфонов.

Атрибут *anyDensity* говорит о том, каким образом ваше приложение будет масштабироваться при отображении на устройствах с разной плотностью пикселов. Если вы учитываете это свойство экрана в своем интерфейсе (а делать это необходимо), установите этому атрибуту значение **true**.

При значении **false** Android будет использовать режим совместимости, пытаясь корректно масштабировать пользовательский интерфейс приложения. Как правило, это снижает качество изображения и приводит к артефактам при масштабировании.

Для приложений, собранных с помощью SDK с API level 4 и выше, этот атрибут по умолчанию имеет значение **true**.

# Рекомендации по разработке интерфейсов, не зависящих от разрешения

Помимо возможностей, предоставляемых широким разнообразием устройств под управлением Android, разработчик приложений не застрахован и от потенциальных опасностей.

Здесь собраны некоторые из наиболее популярных методик по созданию приложений, работающих одинаково эффективно на устройствах с любым экраном.

Главное, что нужно помнить: никогда не пытайтесь угадать, какой экран будет у устройства, на котором должно работать ваше приложение.

Создавайте ресурсы и разметку для разных типов экранов (маленький, нормальный и большой размер в сочетании с низкой, средней и высокой плотностью пикселов), а не для конкретных разрешений или показателей dpi. Изначально ориентируясь на то, что приложение будет слегка масштабироваться на каждом устройстве, вы можете быть уверены, что пользовательский интерфейс не пострадает.

### Менеджер компоновки RelativeLayout и аппаратно-независимые пикселы

По возможности следует избегать использования пиксельных значений, прописанных в коде программы. Это касается разметки, объектов *Drawable* и размеров шрифтов.

В частности, нужно как можно меньше (насколько это возможно) использовать класс *AbsoluteLayout*, так как он предусматривает указывание пиксельных координат для каждого дочернего Представления. Вместо этого применяйте другие менеджеры компоновки, размещайте дочерние элементы друг относительно друга или основываясь на границах экрана. Для большинства сложных пользовательских интерфейсов лучше всего подойдет разметка *RelativeLayout*. Внутри экземпляров своей разметки также стоит избегать использования пиксельных значений при указании размеров для Представлений, объектов *Drawable* и шрифтов. Вместо этого описывайте высоту и ширину элементов, применяя атрибуты *wrap\_content* или *fill\_parent* везде, где это уместно. При необходимости пользуйтесь аппаратно-независимыми пикселами (*density-independent pixels* или *dp*), чтобы указывать размеры Представлений, и значениями *sp* (*scale-independent pixels*) в случае со шрифтами.

### Использование возможностей масштабируемой графики

По возможности вместо фиксированных растровых изображений используйте следующие ресурсы *Drawable*: *NinePatch*, *ShapeDrawable*, *GradientDrawable*. Из композитных ресурсов и ресурсов для описания преобразований предпочтительнее использовать следующие: *RotateDrawable* и *ScaleDrawable*, *LevelListDrawable*, *StateListDrawable*. При описании этих ресурсов необходимо использовать аппаратно-независимые пикселы (dp).

Преимущество от использования масштабируемых ресурсов ‒ универсальная поддержка экранов с разными размерами и разрешениями, фреймворк автоматически масштабирует ваши элементы, обеспечивая самое высокое качество изображения.

### Описание ресурсов, оптимизированных для разных экранов

При использовании ресурсов Drawable, которые не могут достаточно хорошо масштабироваться динамически, вам необходимо создавать наборы изображений, оптимизированные для каждой категории пиксельной плотности (низкой, средней и высокой). Отличный пример ресурса, который должен быть оптимизирован для экранов с разной плотностью пикселов, ‒ значки приложений.

Используя фреймворк для управления ресурсами можно создавать директории со спецификаторами, чтобы хранить наборы ресурсов для всех экранов с поддерживаемой плотностью пикселов, например: *res/drawable-ldpi*, *res/drawable-mdpi*, *res/drawable-hdpi*.

Разметка, оптимизированная для экрана обычного смартфона, может не вместить всю важную информацию, если ее использовать на маленьком телефоне, если же ее отобразить на большом устройстве, таком как планшетный компьютер, она может выглядеть слишком разреженной. С помощью системы для управления ресурсами добавляйте спецификаторы к каталогам, создавая разметку, оптимизированную под маленькие, обычные и большие экраны: *res/layout-small*, *res/layout-normal*, *res/layout-large*.

# Создание ресурсов для разных языковых настроек и аппаратных конфигураций

Одна из наиболее очевидных выгод от применения внешних ресурсов ‒ возможность использовать механизм динамического выбора ресурсов, доступный в Android.

В следующем примере показано дерево каталогов, которое наряду со стандартными строковыми значениями хранит варианты для французского языка и франкоканадского региона:

Project/

res/

values/

strings.xml

values-fr/

strings.xml

values-fr-rCA/

strings.xml

Ниже перечислены спецификаторы, которые можно применить для предоставления альтернативных значений в ресурсах:

* Мобильный код страны и код мобильного оператора (MCC/MNC). Информация о стране и опционально о мобильной сети привязана к SIM-карте устройства. MCC состоит из символов mcc, за которыми следует трехзначный код страны. При желании можете добавить MNC, используя символы mnc и двузначный код мобильной сети (например, mcc234-mnc20 или mcc310).
* Язык и регион. Язык указывается с помощью языкового кода в формате ISO 639-1, состоящего из двух символов в нижнем регистре. В случае необходимости за ним может следовать обозначение региона в виде буквы r и двухсимвольного кода в формате ISO 3166-1-alpha-2, записанного в верхнем регистре (например, en, en-rUS или en-rGB).
* Размер экрана. Может иметь одно из следующих значений: small (меньше, чем HVGA), medium (HVGA или меньше, чем VGA) или large (VGA или больше).
* Высота и ширина экрана. Указывайте значения long или notlong, если хотите создать ресурсы, предназначенные специально для широкоформатных дисплеев (например, long для WVGA, notlong для QVGA).
* Ориентация экрана в пространстве. Поддерживаются режимы port (портретный), land (альбомный) и square (если высота экрана равна ширине).
* Плотность пикселов на экране. Плотность пикселов, измеряемая в точках на дюйм (dpi). Для экранов с низкой (120 dpi), средней (160 dpi) и высокой (240 dpi) плотностью рекомендуется использовать значения ldpi, mdpi и hdpi соответственно. Вы можете указывать значение nodpi для растровых ресурсов, которые не должны масштабироваться, чтобы поддерживать конкретную плотность пикселов.

При выборе именно этого типа ресурсов (в отличие от остальных) система не требует точного совпадения. Подбирая подходящий каталог, Android остановится на спецификаторе, который точнее всего описывает плотность пикселов экрана устройства, и откорректирует масштаб результирующего объекта Drawable соответствующим образом:

1. Тип сенсорного экрана: notouch, stylus или finger.
2. Наличие клавиатуры: keysexposed, keyshidden или keyssoft.
3. Тип ввода, поддерживаемый клавиатурой: nokeys, qwerty или 12key.
4. Способ навигации по пользовательскому интерфейсу: nonav, dpad, trackball или wheel.

Можно задать несколько спецификаторов для любого типа ресурсов, разделив их дефисами. Поддерживаются любые сочетания, однако они должны использоваться в порядке из вышеприведенного списка. В одном спецификаторе может применяться не более одного значения.

При извлечении ресурсов во время выполнения программы Android найдет самый подходящий вариант из всех доступных. Пройдясь по списку каталогов, где хранятся нужные значения, он выберет тот, у которого больше всего совпавших спецификаторов. Если требованиям соответствуют сразу два каталога, учитывается порядок совпавших спецификаторов из предыдущего списка.

# Изменение конфигурации во время выполнения программы

При изменении языка, региона или аппаратной конфигурации Android прерывает работу всех приложений и затем запускает их повторно, перезагружая значения из ресурсов.

Чтобы заставить Активность отслеживать изменения конфигурации при выполнении программы, добавьте в ее узел в манифесте атрибут android:configChanges, указав, какие именно события хотите обрабатывать.

Значения, с помощью которых можно описать изменения конфигурации:

* **оrientation** ‒ положение экрана изменено с портретного на альбомное (или наоборот);
* **keyboardHidden** ‒ клавиатура выдвинута или спрятана;
* **fontScale** ‒ пользователь изменил предпочтительный размер шрифта;
* **locale** ‒ пользователь выбрал новые языковые настройки;
* **keyboard** ‒ изменился тип клавиатуры; например, телефон может иметь 12-клавишную панель, при повороте которой появляется полноценная клавиатура;
* **touchscreen** или **navigation** ‒ изменился тип клавиатуры или способ навигации. Как правило, такие события не встречаются.

В некоторых случаях одновременно будут срабатывать несколько событий. Например, когда пользователь выдвигает клавиатуру, большинство устройств генерируют события *keyboardHidden* и *orientation*.

# Основы проектирования пользовательского интерфейса в Android

Пользовательский интерфейс (user interface, UI), впечатления от использования (user experience, UX), взаимодействие человека с компьютером (human computer interaction, HCI), юзабилити (usability) ‒ весьма обширные темы. Тем не менее при создании собственных пользовательских интерфейсов вы должны в них ориентироваться.

С появлением Android введено несколько новых терминов для обозначения уже известных программных абстракций:

* **Представления.** Это базовый класс для всех визуальных элементов интерфейса (более известных как элементы управления или виджеты). Все элементы UI, включая разметку, ‒ производные от класса View;
* **Группы представлений.** Это потомок класса View, который может содержать внутри себя несколько дочерних Представлений. Наследуйте класс ViewGroup, чтобы создавать сложные Представления, состоящие из взаимосвязанных элементов. Класс ViewGroup также стал основой для менеджеров компоновки, которые помогают размещать элементы управления внутри Активностей.
* **Активности.** Активности ‒ это отображаемые окна (или экраны). Активность ‒ эквивалент Формы для Android. Чтобы вывести на экран пользовательский интерфейс, необходимо назначить для Активности хотя бы одно Представление (как правило, разметку).

Android предоставляет несколько универсальных элементов пользовательского интерфейса, виджеты и менеджеры компоновки (разметку).

Чтобы создавать собственные уникальные графические приложения нужно расширять и модифицировать стандартные Представления (комбинировать имеющиеся или воплощать совершенно новые).

# Создание пользовательского интерфейса для Активностей с помощью Представлений

Все визуальные компоненты в Android ‒ потомки класса View и, как правило, называются Представлениями. Новая Активность изначально ‒ пустой экран, где разработчик должен разместить пользовательский интерфейс. Для этого необходимо вызвать метод setContentView, передав ему экземпляр Представления или ресурс с разметкой, который вы хотите отобразить. Так как пустой дисплей выглядит не очень вдохновляюще, при переопределении обработчика onCreate почти всегда используется метод setContentView, чтобы назначать пользовательский интерфейс для Активности.

Этот метод принимает либо идентификатор ресурса с разметкой, либо единственный экземпляр Представления. Благодаря этому можно описывать пользовательский интерфейс как внутри кода программы, так и с помощью внешних ресурсов с разметкой, что более предпочтительно.

Используя ресурсы с разметкой, вы отделяете уровень представления от бизнес-логики. Благодаря такому гибкому подходу можно изменять внешний вид, не затрагивая код программы. Это позволяет указывать различные экземпляры разметки, оптимизированные под разные аппаратные конфигурации, вы даже можете менять их во время выполнения программы, основываясь на аппаратных изменениях (например, при повороте экрана).

Ниже показано как с помощью внешнего ресурса с разметкой задать пользовательский интерфейс для Активности. Можно получить ссылки на Представления, размещенные внутри разметки, используя метод *findViewById*. В этом примере предполагается, что файл *main.xml* хранится в каталоге проекта *res/layout*.

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

TextView myTextView = (TextView)findViewById(R.id.myTextView);

}

Ниже приводится пример, как назначить новый объект *TextView* в качестве пользовательского интерфейса.

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

TextView myTextView = new TextView(this);

setContentView(myTextView);

myTextView.setText("Hello, Android");

}

Метод *setContentView* в качестве параметра принимает единственный экземпляр Представления, поэтому чтобы добавить в Активность сразу несколько элементов, необходимо использовать разметку.

# Виджеты, доступные в Android

Android ‒ это набор стандартных элементов управления, с помощью которых можно создавать простые интерфейсы. Используя эти Представления (а также изменяя и расширяя их при необходимости), можно упростить процесс разработки и обеспечить преемственность между разными приложениями.

Некоторые из наиболее используемых элементов:

* **TextView.** Стандартная метка, предназначенная для вывода текста. Она поддерживает многострочное отображение, форматирование и автоматический перенос слов и символов.
* **EditText.** Редактируемое поле для ввода текста. Поддерживает многострочный ввод, перенос слов на новую строку и текст подсказки.
* **ListView.** Группа представлений, которая формирует вертикальный список элементов, отображая их в виде строк внутри списка. Простейший объект ListView использует TextView для вывода на экран значений toString, принадлежащих элементам массива.
* **Spinner.** Составной элемент, отображающий TextView в сочетании с соответствующим Представлением ListView, которое позволяет выбрать элемент списка для отображения в текстовой строке. Сама строка состоит из объекта TextView, показывающего текущий выбор, и кнопки, при нажатии которой всплывает диалог выбора.
* **Button.** Стандартная кнопка, которую можно нажимать.
* **CheckBox.** Кнопка, поддерживающая два состояния. Представлена в виде отмеченного или неотмеченного флажка.
* **RadioButton.** Переключатель, который поддерживает два состояния и группировку. Группы таких переключателей пользователь видит как набор двоичных вариантов, из которых в определенный момент времени может быть выбран только один.
* **ViewFlipper.** Группа представлений, позволяющая определить набор элементов и горизонтальную строку, в которой одновременно может выводиться только одно Представление. При этом переходы между отображаемыми элементами осуществляются с помощью анимации.
* **QuickContactBadge.** Выводит на экран эмблему со значком, присвоенным указанному контакту, используя такие данные, как номер телефона, имя, адрес электронной почты или сайта. Нажатие значка приведет к отображению панели, которая предоставляет различные варианты для связи с выбранным контактом, включая звонок, SMS, электронную почту и системы обмена мгновенными сообщениями (IM).

Это только некоторые из доступных виджетов. Android также поддерживает несколько более продвинутых реализаций Представлений, включая элементы для выбора даты и времени, поля ввода с автодополнением, карты, галереи и вкладки.

# Менеджеры компоновки

Менеджер компоновки (более известный как разметка) — это расширение класса *ViewGroup*, которое используется для позиционирования дочерних элементов внутри пользовательского интерфейса. Экземпляры разметки могут быть вложенными. Комбинируя их, вы можете создавать сколь угодно сложные интерфейсы.

Android SDK включает некоторые простые виды разметки, которые могут помочь конструировать пользовательские интерфейсы. Только от вас самих зависит выбор правильного сочетания менеджеров компоновки, которые помогут сделать интерфейсы понятными и простыми в использовании.

Наиболее универсальные доступные классы разметки:

* **FrameLayou**t. Самый простой из менеджеров компоновки, прикрепляет каждое дочернее Представление к верхнему левому углу экрана. Каждый новый элемент накладывается на предыдущий, заслоняя его.
* **LinearLayout.** Помещает дочерние Представления в ряд (горизонтальный или вертикальный). Вертикальная разметка представляет собой колонку с элементами, горизонтальная вытягивает их в строку. LinearLayout позволяет задавать «ширину» каждого дочернего Представления, благодаря чему можно контролировать их размеры в пределах доступного пространства.
* **RelativeLayout.** Наиболее гибкий среди стандартных видов разметки, позволяет задавать позицию каждого дочернего Представления относительного других элементов и границ экрана.
* **TableLayout.** Позволяет размещать Представления с помощью сетки, состоящей из строк и столбцов. При этом столбцы могут либо автоматически растягиваться, либо оставаться постоянной ширины.
* **Gallery.** Отображает элементы в виде однострочного горизонтального списка, который можно прокручивать.

В документации к Android подробно описаны возможности и свойства каждого класса разметки: http://developer.android.com/guide/topics/ui/layout-objects.html.

### Использование разметки

Предпочтительный способ реализации разметки ‒ использование XML-файлов в качестве внешних ресурсов. Любой такой файл должен содержать корневой элемент (узел), который, в свою очередь, может включать столько вложенных экземпляров разметки и Представлений, сколько необходимо для построения любых сложных интерфейсов.

Ниже показана простая вертикальная разметка *LinearLayout*, которая помещает *TextView* над элементом *EditText*.

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

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

android:orientation="vertical"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent">

<TextView

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:text="Enter Text Below"

/>

<EditText

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:text="Text Goes Here!"

/>

</LinearLayout>

Обратите внимание, что для каждого элемента разметки вместо абсолютных значений ширины и высоты в пикселах используются константы *wrap\_content* и *fill\_parent*. Это обеспечивает мощный механизм, благодаря которому экземпляры разметки не зависят от размера и разрешения экрана. Константа *wrap\_content* задает Представлению минимальный размер, необходимый для отображения содержимого (например, высоту, которая нужна для вывода многострочного текста). Константа *fill\_parent* расширяет Представление, чтобы оно заполнило все доступное пространство внутри родительского элемента (или экрана). В примере разметка заполняет весь экран, тогда как оба текстовых представления пытаются вытянуться во всю доступную ширину. Их высота ограничена размерами, необходимыми для отображения текста.

Реализация разметки в виде XML-файлов ведет к отделению уровня представления от логики, содержащейся в элементах управления и Активностях. Это также позволяет создавать разновидности разметки, предназначенные для конкретной аппаратной платформы, и загружать их динамически, без изменений в коде программы.

# Обработка событий, основанных на взаимодействии с пользователем

Чтобы новое Представление было интерактивным, оно должно реагировать на нажатие клавиш и касание экрана. В Android есть несколько обработчиков событий, которые позволяют реагировать на пользовательский ввод.

* **onKeyDown.** Вызывается при нажатии любой аппаратной клавиши, включая манипулятор D-pad, клавиатуру, а также кнопки для телефонного вызова, отмены звонка, возврата и управления камерой.
* **onKeyUp.** Вызывается, когда пользователь отпускает нажатую клавишу.
* **onTrackballEvent.** Вызывается при перемещении трекбола.
* **onTouchEvent.** Вызывается при нажатии / отпускании сенсорного экрана или же при обнаружении движения.

Ниже показан каркас класса, в котором переопределяются все обработчики событий, отвечающие за взаимодействие с пользователем.

@Override

public boolean onKeyDown(int keyCode, KeyEvent keyEvent) {

// Верните значение true, если событие было обработано.

return true;

}

@Override

public boolean onKeyUp(int keyCode, KeyEvent keyEvent) {

// Верните значение true, если событие было обработано.

return true;

}

@Override

public boolean onTrackballEvent(MotionEvent event ) {

// Получите тип действия, которое представлено данным событием.

int actionPerformed = event.getAction();

// Верните значение true, если событие было обработано.

return true;

}

@Override

public boolean onTouchEvent(MotionEvent event) {

// Получите тип действия, которое представлено данным событием.

int actionPerformed = event.getAction();

// Верните значение true, если событие было обработано.

return true;

}

# Использование нестандартных элементов управления

В Android можно создавать свои Представления, наследуя класс *View*. Например, можно создать элемент компас, который будет выводить на экран традиционное изображение компаса (в виде розы ветров), указывая курс (направление).

Создавая собственные Представления, их можно использовать внутри кода программы и в разметке, как и любые другие элементы. Ниже показано, как переопределить метод *onCreate*, чтобы добавить в Активность кастомный элемент *CompassView*.

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

CompassView cv = new CompassView(this);

setContentView(cv);

cv.setBearing(45);

}

Чтобы использовать этот же элемент внутри ресурса, укажите полное имя класса при создании нового узла в описании разметки, как показано в следующем фрагменте XML-кода:

<com.paad.compass.CompassView

android:id="@+id/compassView"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

/>

Вы можете наполнить разметку и получить ссылку на CompassView, используя следующий код:

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

CompassView cv = (CompassView)this.findViewById(R.id.compassView);

cv.setBearing(45);

}

# Введение в Linkify

Linkify ‒ вспомогательный класс, который автоматически создает гиперссылки внутри элементов *TextView* (и их производных) на основе регулярных выражений RegEx.

Текст, соответствующий заданному регулярному выражению, будет преобразован в гиперссылку, при нажатии которой срабатывает код вида *startActivity* (new Intent(Intent.ACTION\_VIEW, uri)), где uri ‒ совпавший текст.

С помощью *Linkify* можно задать шаблон строк, которые нужно превратить в ссылки. Для удобства класс *Linkify* предлагает предустановленные шаблоны для часто встречающихся типов содержимого (таких как телефонные номера или адреса электронной почты).

### Стандартные типы ссылок в Linkify

Статический метод *Linkify.addLinks* в качестве параметров принимает Представление, к которому будут применены шаблоны, а также битовые маски для одного или нескольких стандартных типов содержимого, предоставляемые классом *Linkify*: WEB\_URLS, EMAIL\_ADDRESSES, PHONE\_NUMBERS и ALL.

Ниже показано, как с помощью класса *Linkify* отобразить адреса веб-сайтов и электронной почты, содержащихся в элементе *TextView*, в виде гиперссылок. При нажатии ссылки откроются браузер и почтовый клиент соответственно.

TextView textView = (TextView)findViewById(R.id.myTextView);

Linkify.addLinks(textView, Linkify.WEB\_URLS|Linkify.EMAIL\_ADDRESSES);

Также можно применить класс *Linkify* к Представлению прямо внутри ресурса с разметкой, используя атрибут *android:autoLink*. Он поддерживает одно или несколько значений (разделенных символом |), названия которых говорят сами за себя: none, web, email, phone и all.

Ниже показано, как добавить гиперссылку для телефонных номеров и адресов электронной почты.

<TextView

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:text="@string/linkify\_me"

android:autoLink="phone|email"

/>

### Создание собственных шаблонов для Linkify

Чтобы описать собственный шаблон для *Linkify*, необходимо создать новое регулярное выражение *RegEx*, соответствующее тексту, который вы хотите отобразить в виде ссылки.

Как и в случае со стандартными типами, применять *Linkify* к Представлению можно с помощью метода *Linkify.addLinks*, но вместо константы необходимо передать новое регулярное выражение *RegEx*. Вы также можете передать префикс, который будет добавлен к целевому пути URI после нажатия ссылки.

В листинге 5.14 показано, как применить *Linkify* к Представлению для добавления поддержки данных о землетрясениях, предоставляемых с помощью Источника данных. Чтобы не добавлять всю схему, шаблон для *Linkify* считает за совпадение любой текст, начинающийся со строки quake, за которым следует номер. Потом схема добавляется в начало пути URI, прежде чем сработает Намерение.

int flags = Pattern.CASE\_INSENSITIVE;

Pattern p = Pattern.compile("\\bquake[0-9]\*\\b", flags);

Linkify.addLinks(myTextView, p,

"content://com.paad.earthquake/earthquakes/");

### Использование интерфейса MatchFilter

*Linkify* также поддерживает интерфейсы *TransformFilter* и *MatchFilter*, что позволяет установить дополнительный контроль над структурой целевого пути URI и описание регулярного выражения.

Ниже показана реализация интерфейса *MatchFilter*, которая отменяет любое совпадение, если перед найденной строкой находится знак восклицания.

class MyMatchFilter implements MatchFilter {

public boolean acceptMatch(CharSequence s, int start, int end) {

return (start == 0 || s.charAt(start-1) != ‘!’);

}

### Использование интерфейса TransformFilter

*TransformFilter* дает больше свободы для форматирования текстовых строк, изменения внешнего вида генерируемых ссылок. Отделение текста ссылки от целевого пути URI позволяет менять способ вывода текстовых данных на экраны.

Чтобы задействовать интерфейс *TransformFilter*, необходимо переопределить метод *transformUrl*. Он начнет вызываться при нахождении совпадения. В качестве параметров он принимает регулярное выражение и строку URI, которую должен создать. Вы можете изменить совпавшую строку и вернуть путь URI в качестве цели, подходящей для отображения с помощью другой программы.

Реализация интерфейса *TransformFilter*, приведенная ниже, преобразует совпавшую строку в путь URI, возвращаемый в нижнем регистре.

class MyTransformFilter implements TransformFilter {

public String transformUrl(Matcher match, String url) {

return url.toLowerCase();

}

}

# Задание на лабораторную работу

Необходимо повторить интерфейс Android приложения, выбранного студентом, содержащее определенные особенности пользовательского интерфейса.

**Общие требования** при выполнении лабораторной работы:

1. Экраны могут быть реализованы либо как отдельные Activity, либо как фрагменты одного Activity. Можно строить экраны пользовательского интерфейса с помощью Фрагментов.
2. Реализовать системное меню в приложении, или боковую навигацию, для быстрого перехода к любому из демонстрационных экранов, если такое реализовано в оригинальном приложении.
3. Для View элементов, располагаемых на экране, необходимо применить стилистику.
4. View объекты не должны иметь каких либо XML атрибутов отвечающих за стилизацию их отображения, кроме имени стиля.

* Все что отвечает за стилизацию элементов на экране должно быть описано в XML файле с описанием стилей;
* Должно обязательно быть применено наследование стилей;
* Наследование стилей подразумевает, что в новом стиле либо добавляется новое свойство, либо меняется значение родительского свойства.

1. На одном из экранов должна присутствовать демонстрацию анимации. В сложных случаях данных пункт будет обязательно реализовываться в следующей лабораторной работе.
2. К экранам приложения предъявляются следующие требования по содержанию изображений.

* Фон экрана или значимый элемент интерфейс должны иметь градиент при его наличии в интерфейсе;
* При оформлении элементов экрана необходимо применить в стилях не менее 2 описанных через XML векторных Drawable объектов;
* В приложении должно быть применено не менее двух растровых Drawable изображенией, для следующих плотностей экрана: mdpi, hdpi, xhdpi, xxhdpi.

1. Экраны в рамках данной лабораторной работы создаются при помощи XML разметки и только.
2. В приложении должна быть реализована собственная иконка, для всех типов плотностей экранов, без использования генератора иконок IDE.
3. Приложение должно выводить через объект Log всю информацию о своей работе, с применением разных уровней информирования разработчика.
4. Все текстовые константы и константные строки, в обязательном порядке должны быть вынесены в ресурсы приложения.
5. Допускается проектировать интерфейс приложения с расчетом на использование только на планшете или телефоне, что должно быть отражено соответствующим образом в манифесте приложения, в ином случае интерфейс должен быть рабочим в любом из вариантов.

**Контрольные вопросы**

1. Как организовано хранение ресурсов приложения. Опишите структуру директории с ресурсами.
2. Хранение и использование строк как ресурсов. Правила описания строковых констант. Локализация приложения.
3. Правила описания цветовых констант.
4. Единицы размеров применяемые в Android.
5. Стили в Android. Наследование стилей.
6. Drawable объекты. Виды, особенности и назначение.
7. Ресурсы типа NinePatch. Особенности. Назначение. Применение.
8. Разметка экрана в Android. Правила описания и расположения ресурсов разметки.
9. Анимация в Android. XML описание в качестве ресурсов. Использование в Java коде.
10. Меню в Android. Описание в виде XML ресурсов.
11. Создание ресурсов для разных языковых и аппаратных конфигураций.
12. Правила ссылки на ресурсы Android. Применение системных и собственных ресурсов.
13. Изменение аппаратной конфигурации во время работы приложения.

# Список рекомендуемой литературы

1. Майер, Р. Android 2: Программирование приложений для планшетных компьютеров и смартфонов:[пер. с англ] / Р. Майер. - М.: Эксмо, 2011. - 672с.
2. Коматинени, С. Android 4 для профессионалов. Создание приложений для планшетных компьютеров и смартфонов:[пер. с англ]/ С. Коматинени, Д. Маклин. - М.:2012. 880с.
3. Медникс, З. Программирование под Android:[пер. с англ]/ З. Медникс, Л. Дорнин, Б. Мик, М. Накамура. - Спб.:Питер, 2012. - 496с.
4. Аndroid Developers. режим доступа: http://developer.android.com