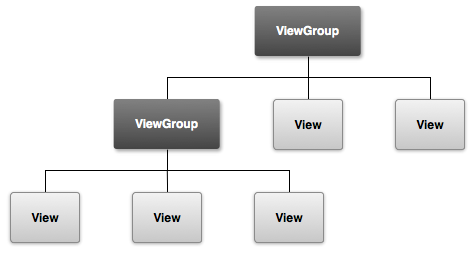
Обзор пользовательского интерфейса

Все элементы интерфейса пользователя в приложении Android создаются с помощью объектов [View](https://developer.android.com/reference/android/view/View.html?hl=ru) и[ViewGroup](https://developer.android.com/reference/android/view/ViewGroup.html?hl=ru). Объект [View](https://developer.android.com/reference/android/view/View.html?hl=ru) формирует на экране элемент, с которым пользователь может взаимодействовать. Объект [ViewGroup](https://developer.android.com/reference/android/view/ViewGroup.html?hl=ru) содержит другие объекты [View](https://developer.android.com/reference/android/view/View.html?hl=ru) (и [ViewGroup](https://developer.android.com/reference/android/view/ViewGroup.html?hl=ru)) для определения макета интерфейса.

Android предоставляет коллекцию подклассов [View](https://developer.android.com/reference/android/view/View.html?hl=ru) и [ViewGroup](https://developer.android.com/reference/android/view/ViewGroup.html?hl=ru), которая включает в себя обычные элементы ввода (такие как кнопки и текстовые поля) и различные модели макет (такие как линейный или относительный макет).

Макеты пользовательского интерфейса

Пользовательский интерфейс для каждого компонента вашего приложения определяется с помощью иерархии объектов [View](https://developer.android.com/reference/android/view/View.html?hl=ru) и [ViewGroup](https://developer.android.com/reference/android/view/ViewGroup.html?hl=ru), как показано на рисунке 1. Каждая группа просмотра представляет собой невидимый контейнер, в котором объединены дочерние виды, причем дочерние виды могут представлять собой элементы ввода или другие виджеты, которые составляют часть пользовательского интерфейса. Эта древовидная иерархия может быть настолько простой или сложной, насколько требуется (чем проще, тем лучше для производительности).



**Рисунок 1**. Иллюстрация иерархии, которая определяет макет интерфейса.

Чтобы объявить свой макет, можно создать экземпляры объектов [View](https://developer.android.com/reference/android/view/View.html?hl=ru) в коде и запустить построение дерева, но самый простой и наиболее эффективный способ — определение макета с помощью файла XML. XML позволяет создавать удобочитаемую структуру макета, подобно HTML.

Имя элемента XML для вида соответствует классу Android, к которому от относится. Так, элемент<TextView> создает виджет [TextView](https://developer.android.com/reference/android/widget/TextView.html?hl=ru) в пользовательском интерфейсе, а элемент <LinearLayout>создает группу просмотра [LinearLayout](https://developer.android.com/reference/android/widget/LinearLayout.html?hl=ru) .

Например, простой вертикальный макет с текстом и кнопкой выглядит следующим образом:

<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
              android:layout\_width="fill\_parent"  
              android:layout\_height="fill\_parent"  
              android:orientation="vertical" >  
    <TextView android:id="@+id/text"  
              android:layout\_width="wrap\_content"  
              android:layout\_height="wrap\_content"  
              android:text="I am a TextView" />  
    <Button android:id="@+id/button"  
            android:layout\_width="wrap\_content"  
            android:layout\_height="wrap\_content"  
            android:text="I am a Button" />  
</LinearLayout>

При загрузке ресурсов макетов в приложение Android инициализирует каждый узел макета в объект режима выполнения, который можно использовать для определения дополнительного поведения, запроса состояния объекта или изменения макета.

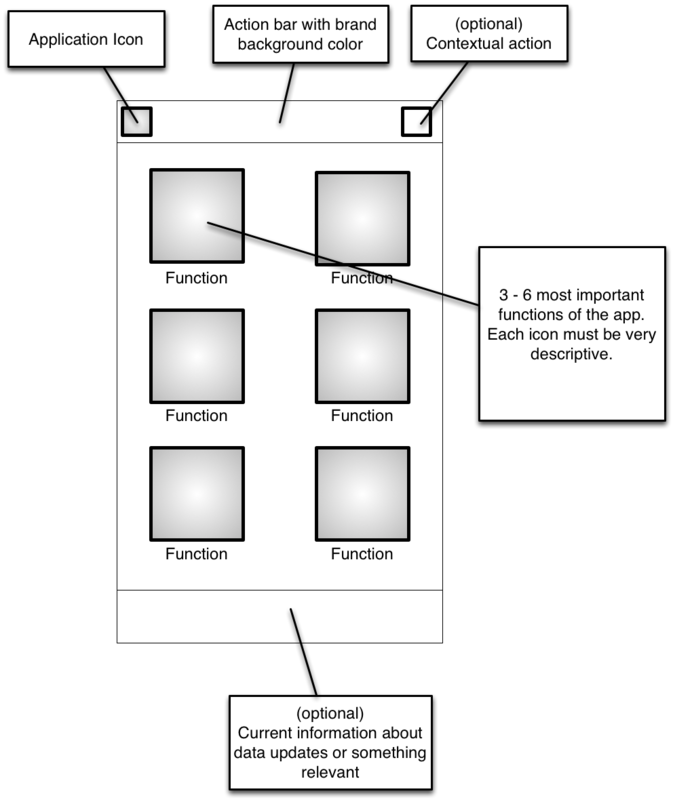
Полное руководство по созданию макета пользовательского интерфейса см. в документе [Макеты XML](https://developer.android.com/guide/topics/ui/declaring-layout.html?hl=ru).

Компоненты пользовательского интерфейса

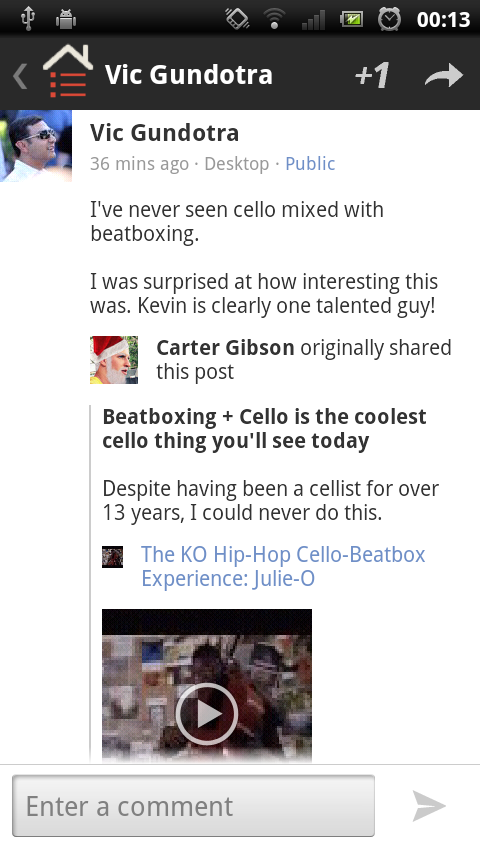
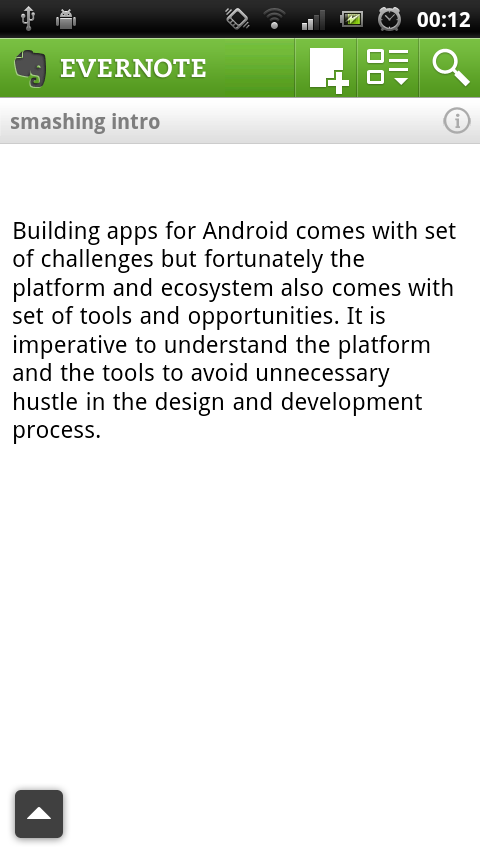
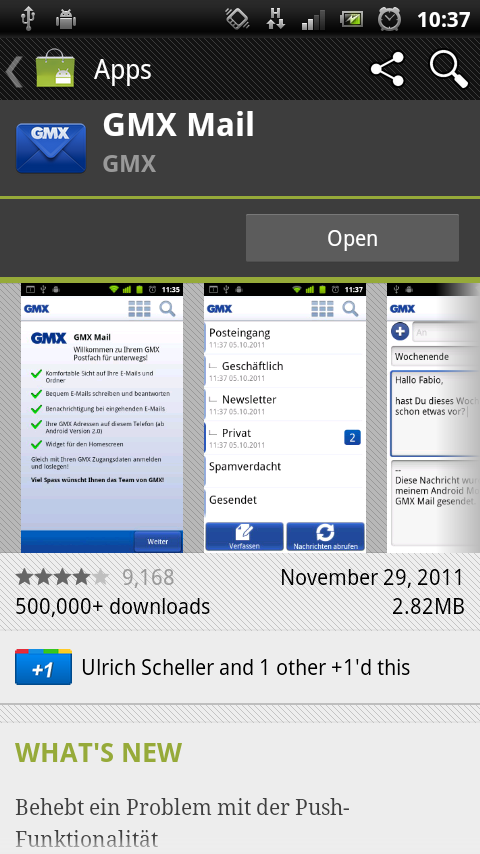
Не обязательно создавать все элементы пользовательского интерфейса с помощью объектов [View](https://developer.android.com/reference/android/view/View.html?hl=ru) и [ViewGroup](https://developer.android.com/reference/android/view/ViewGroup.html?hl=ru). Android предоставляет несколько компонентов приложений, которые содержат стандартный макет пользовательского интерфейса, где остается лишь определить содержимое. Каждый из этих компонентов пользовательского интерфейса содержит уникальный набор API, который описан в соответствующих документах, таких как [Строка действий](https://developer.android.com/guide/topics/ui/actionbar.html?hl=ru), [Диалоги](https://developer.android.com/guide/topics/ui/dialogs.html?hl=ru) и [Уведомления о состоянии](https://developer.android.com/guide/topics/ui/notifiers/notifications.html?hl=ru).

# Как должны выглядеть Android приложения?

Платформа Android не имеет жестких рекомендаций, как должны выглядеть и работать приложения. Google с самого начала дал ясно понять, что у них нет планов начать диктовать, что приемлемо, а что нет. Существует набор [UI рекомендаций](http://developer.android.com/guide/practices/ui_guidelines/index.html), но они в основном концентрируются на мелочах вроде значков, виджетов и меню.  
  
С самого запуска платформы были сотни различных идей интерфейсов, и внешний вид приложений был очень разнообразным. Теперь, когда платформа достигла зрелости и количество приложений резко возросло, происходит формирование пользовательского интерфейса Android. Некоторые функции интерфейса стали общими, а некоторые из них даже нашли свой путь в библиотеках Android SDK. Вскоре пользователи будут ожидать от приложений более единообразной работы. Некоторые элементы управления и модели взаимодействия будут интегрировать в платформу Android.  
  
В этой статье я хочу обобщить, как обычно работает Android UIs на более высоком уровне. Раньше я писал о многих принципах пользовательского интерфейса, но они были изолированы от общей схемы. Теперь я хочу свести их вместе, чтобы показать, что я думаю о том, как должны выглядеть Android приложения.  
  
**Главный экран приложения**

Принцип интерфейса “панель управления” используется во многих приложениях. Если ваше приложение имеет более одной основной функции, то это может быть очень хорошей отправной точкой. Приборная панель показывает самые важные функции приложения и обеспечивает легкий доступ к ним.  
  
  
Панель управления хорошо знакома пользователям Android приложений. Это верный способ заставить пользователя почувствовать себя как дома на первом экране вашего приложения, если оно используется правильно.

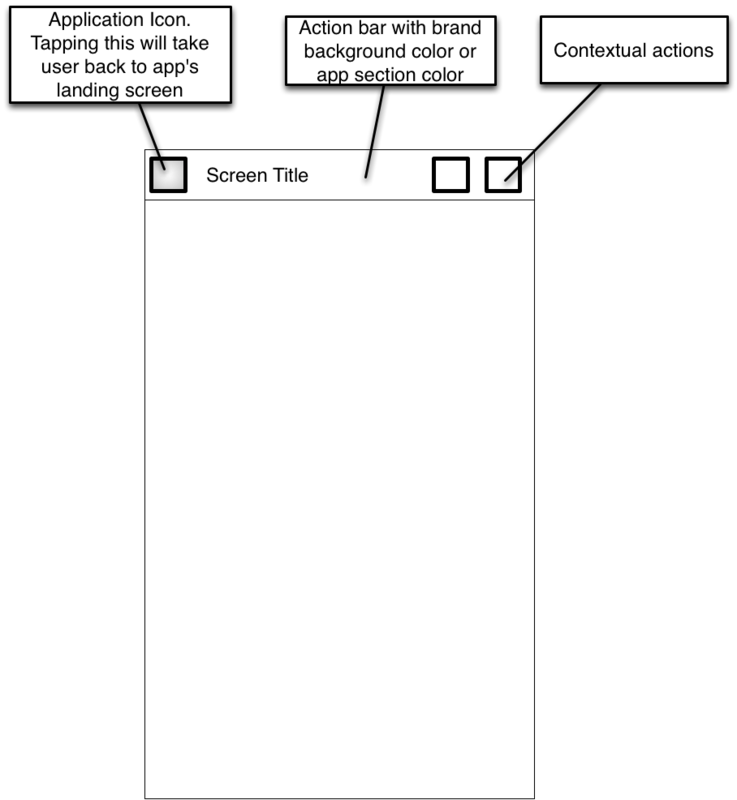
Примеры

#### Общий экран приложения

Фактически Activity проявляются во многих формах, но некоторые особенности стали очень распространенными и пользователи научились понимать и ожидать их. Панель действий в верхней части экрана является очень распространенной и легкой для восприятия концепцией.

* В верхнем левом углу расположен значок приложения или возврата на главный экран. При нажатии пользователь должен вернуться на главную страницу приложения. Стоит отметить, что новые панели действий не возвращают пользователей на главную страницу, вместо этого совершается переход на один уровень вверх в иерархии меню.
* В средней части панели действий будет расположено название экрана и цвета торговой марки или цветах текущего раздела приложения.
* В правом верхнем углу экрана расположены иконки для наиболее важных действий, которые могут быть выполнены на этом экране. При этом данная часть экрана должна содержать только действия, связанные с содержимым текущего экрана. Функция поиска стала исключением этого правила.

  
  
Проект [ActionBarSherlock](http://actionbarsherlock.com/) (Jake Wharton) позволяет облегчить внедрение панелей действий.

#### Списки

Списки являются одним из наиболее распространенных компонентов пользовательского интерфейса Android. Списки очень полезны при отображении данных, особенно, если не известно, какой будет их объем.  
  
У списков есть недостаток. Каждый элемент списка должен быть относительно небольшим, позволяющим удобно просматривать содержимое списка. С другой стороны, очень много информации на небольшом участке может сделать очень затруднительным для пользователей использование списка и поиска элементов, с которыми они хотят взаимодействовать.  
  
Хорошо, что есть некоторые руководства работы списков Android в целом. Пользователи привыкли к определенным элементам и функциям, и если ваш список работает подобным образом, то вашим пользователям гораздо проще освоиться.

##### **Панель действий на экранах списков.**

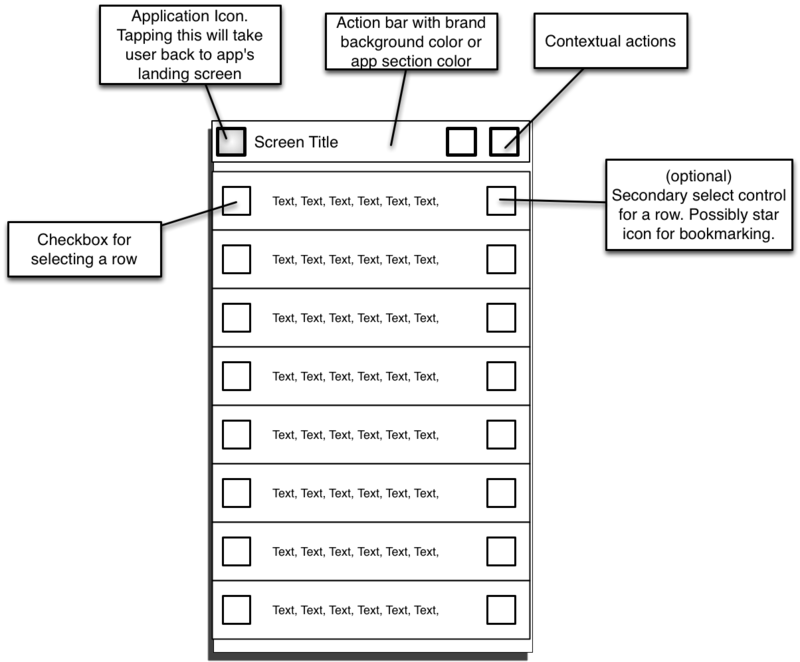
Экраны списков могут использовать панель действий, чтобы отображать действия, которые направлены на весь список. Обратите внимание, что панель действий на экране для операции над несколькими элементами списка должна отличаться, нежели при выполнении действия на одном элементе списка.

##### **Элементы списка и чекбоксы**

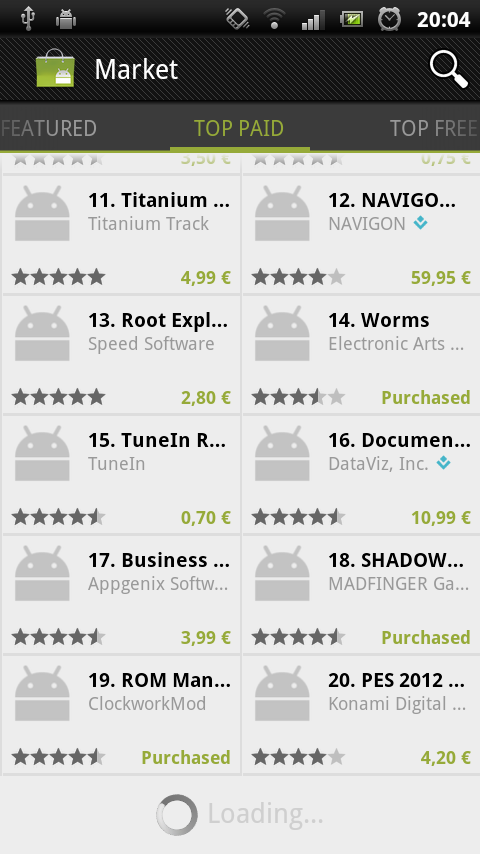
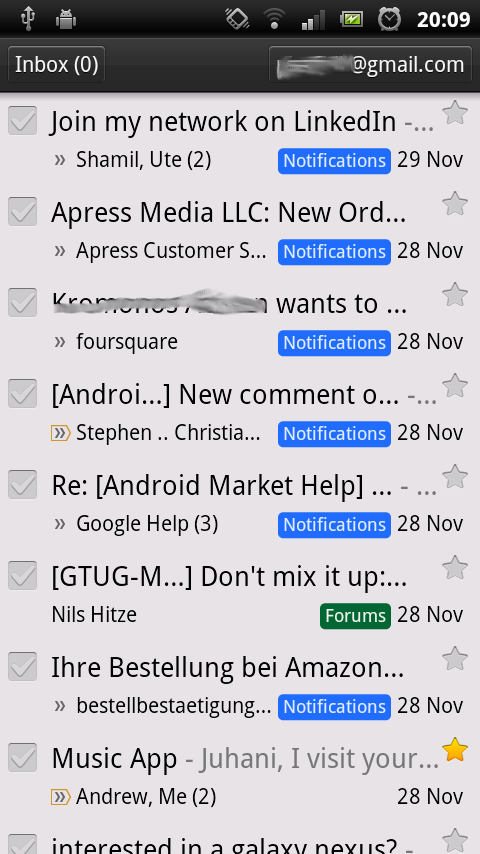
Список элементов, как правило, содержит текст и несколько графических элементов. Очень часто встречаются чекбоксы на каждом элементе, которые могут быть использованы для выбора одного или нескольких элементов списка для выполнения операций над ними.  
  
Размещение чекбокса слева в элементе списка имеет следующие преимущества:

1. Мы привыкли видеть чекбоксы с левой стороны элементов, которые мы выбираем. Это верно для веб, рабочего стола и других мобильных приложений.
2. Чекбокс на краю пункта позволяет нам создавать большие области для нажатия, что позволяет пользователям легче различать нажатия на элементы списка и выбор одного элемента.
3. Графический компонент с левой стороны элемента создает легкую визуальную подсказку, где заканчивается один элемент и начинается другой, что делает его намного проще для пользователей, чтобы быстро просмотреть список.

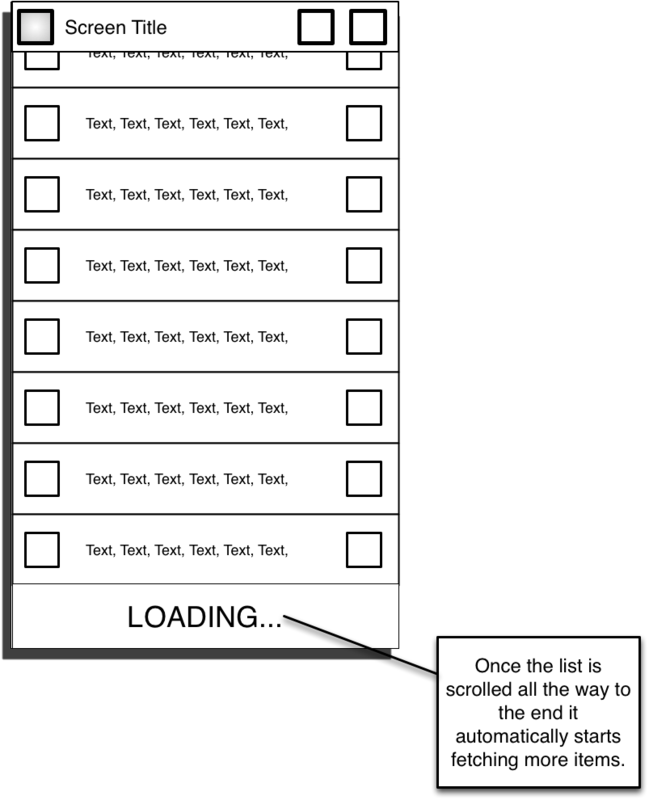
##### **Вторичные элементы управления**

Некоторым элементам необходимо больше возможностей взаимодействия, чем простой выбор (чекбокс) или навигация (нажатие). Наиболее распространенный случай использования этого элемента управление рейтингом или добавления элемента в закладки. Единственное естественное место для вторичного элемента управления — это правый край элемента. Любое другое место может привести к проблемам с расположением.  
  
  
  
Aldiko и Google Mail являются хорошими примерами приложений, которые используют приятные списки. Aldiko решил поставить флажки справа, что визуально делает пользовательский интерфейс несбалансированным.

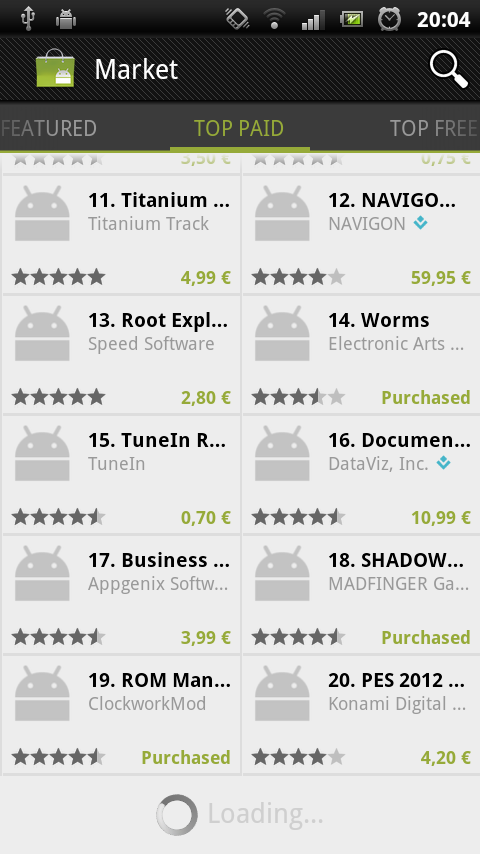
Примеры

##### **Бесконечные списки**

Многие списки содержат элементы, которые загружаются по сети. В этой ситуации процесс загрузки может занять много времени, и список не может быть заполнен так быстро, как пользователь прокручивает его. В случае, когда пользователь достигает конца списка, приложение должно автоматически начать загрузку следующих элементов. Индикатор говорит пользователю, что следующие загруженные элементы будут находиться в конце списка. В том числе некоторые виды загрузки анимации, например, хорошей идеей является индикатор прогресса. Анимация позволяет пользователям понять что выполняется подгрузка данных  
  
  
  
Android Market и Twitter автоматически загружают несколько элементов при достижении конца списка.

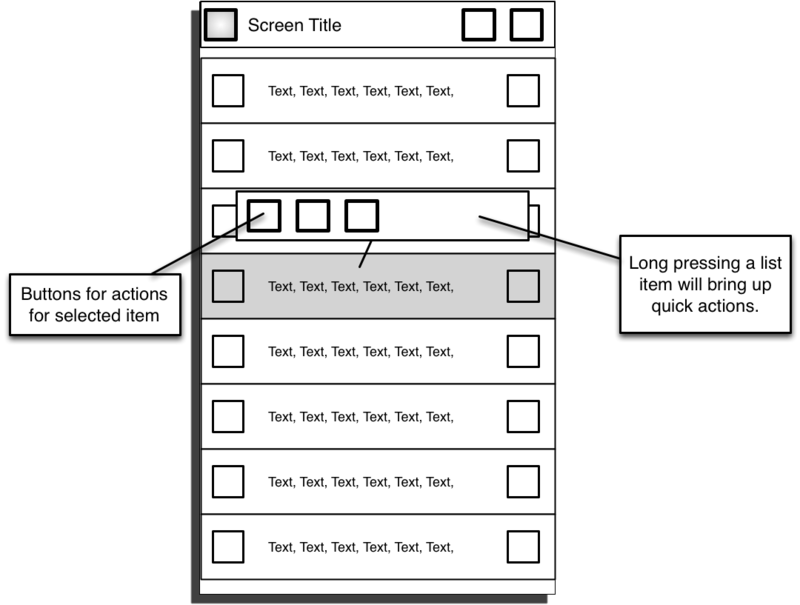
Примеры

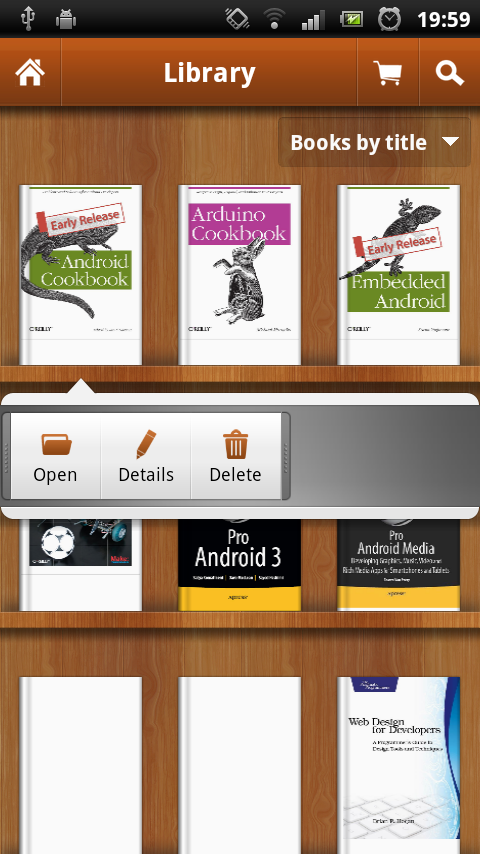
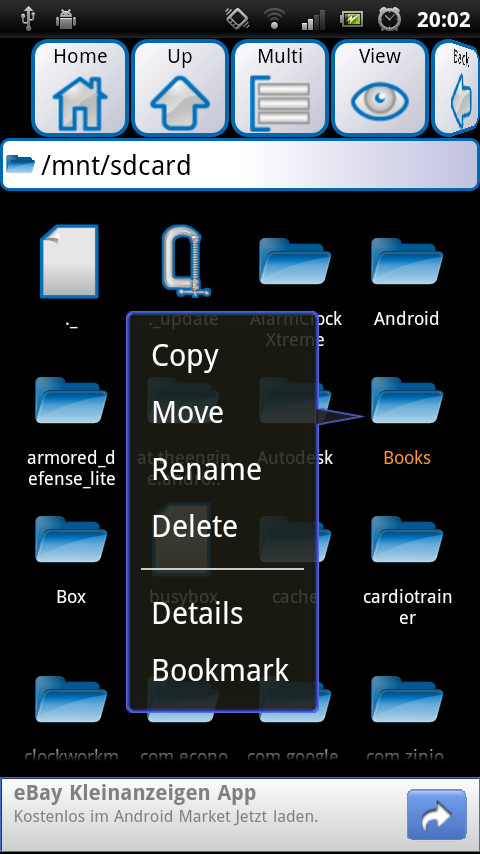

##### **Действия над элементами — Долгое нажатие — Быстрые действия**

Нужно дать пользователям возможность выполнять операции над одним элементом списка без необходимости перемещаться в начало экрана.  
  
Так как телефоны и планшетные компьютеры не имеют функции нажатия правой кнопки мыши, специфика «правого клика» у сенсорных экранов эволюционировала. При длительном нажатии на элементе пользователю дают понять, что он хочет выполнить операцию для текущего элемента.  
  
Существует модель пользовательского интерфейса, которая называется “быстрые действия” для отображения действий над элементами списка. Использование оригинального графического подхода в значительной степени вымерло, но суть остается все та же. Она является одной из форм наложения меню, которое отображает очень простой перечень действий. Обычно от трех до пяти. Независимо от того, как быстро визуально реализуется действие, имейте в виду:

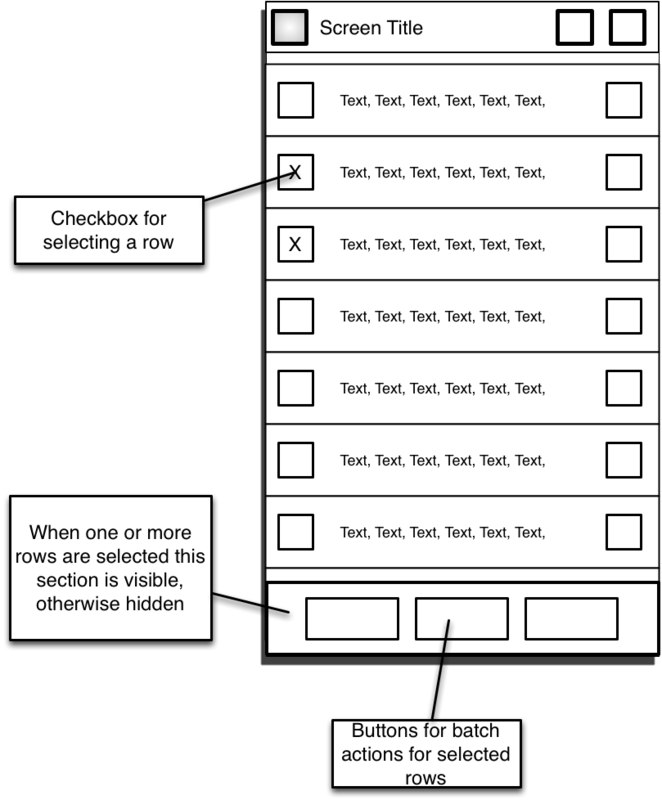
* Не закрывайте выбранный элемент! Особенно, когда пользователи выполняют операцию удаления. Пользователи более уверены в своих действиях, если видят элемент все время.
* Показывайте только простые действия. Все, что требуется сложного взаимодействия, лучше обрабатывать на отдельном экране, нежели в быстрых действиях.

  
  
Aldiko, Asto File Manager и Google+ используют различные визуальные стили. Во всех случаях эти действия появляются при длительном нажатии на элемента экрана.  
  
Aldiko и Astro примеры хорошего дизайна, но Google + нарушает правило о сокрытии целевого элемента, поскольку они используют простые всплывающие окна. Я надеюсь, что они исправят этот недостаток в будущих версиях.

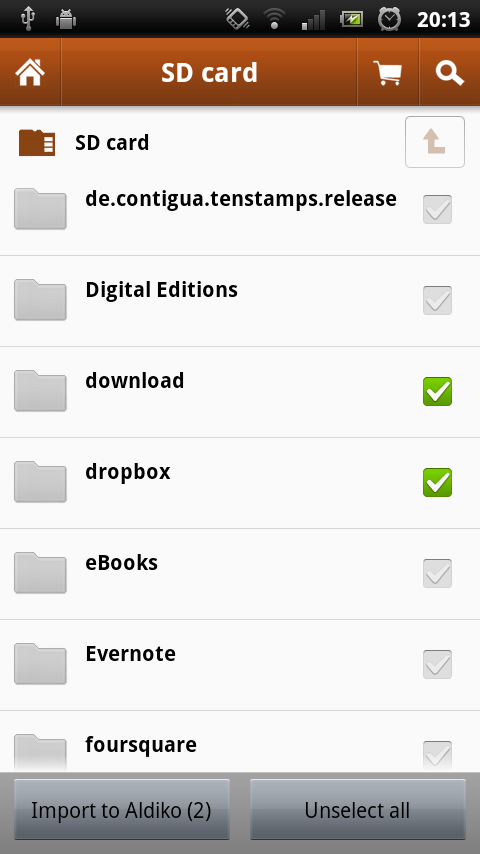
Примеры


##### **Действия над несколькими элементами**

Если список содержит элемент управления чекбокс, то это позволяет пользователю выбирать несколько элементов. При выборе нескольких элементов пользователь может выполнять действия сразу для всех выбранных элементов.  
  
Распространенный способ обработки действий нескольких элементов — это добавление панели в нижней части экрана, где расположены кнопки для возможных действий сразу над всеми выделенными элементами. Анимация с хорошим скольжением добавляет пользовательскому интерфейсу плавность и изысканность. Панель должна автоматически скрываться, когда снято последнее выделение или действие выполнено.  
  
  
  
Aldiko и GMail являются хорошим примером, как должна быть организована работа с выделенными элементами. Оба приложения имеют хорошую скользящую анимацию при появлении нижней панели. Aldiko также добавляет число на кнопке импорта, показывая пользователю, как много элементов он выбрал. Это очень приятное дополнение, но не во всех случаях.

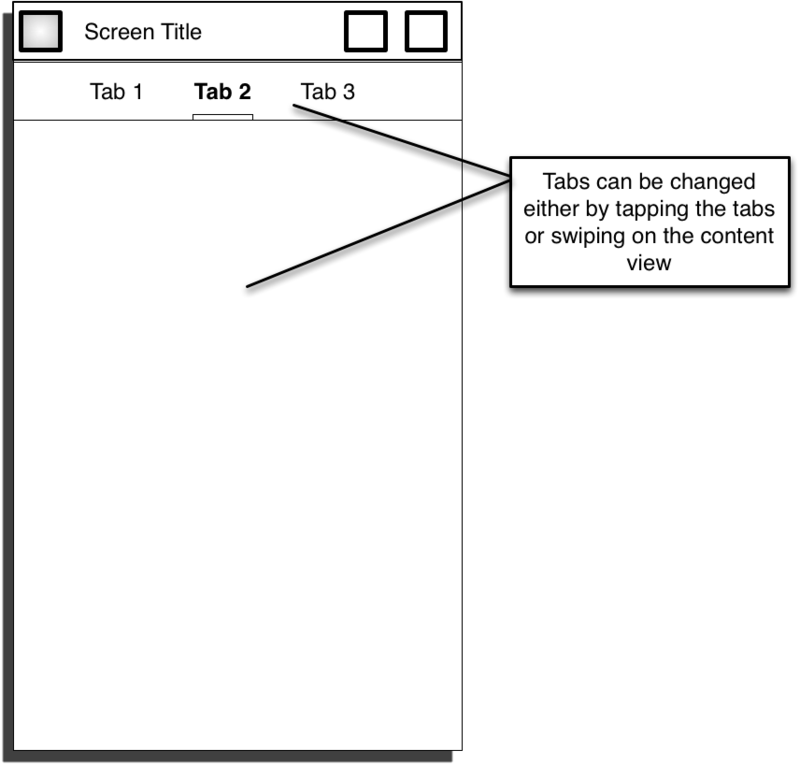
Примеры

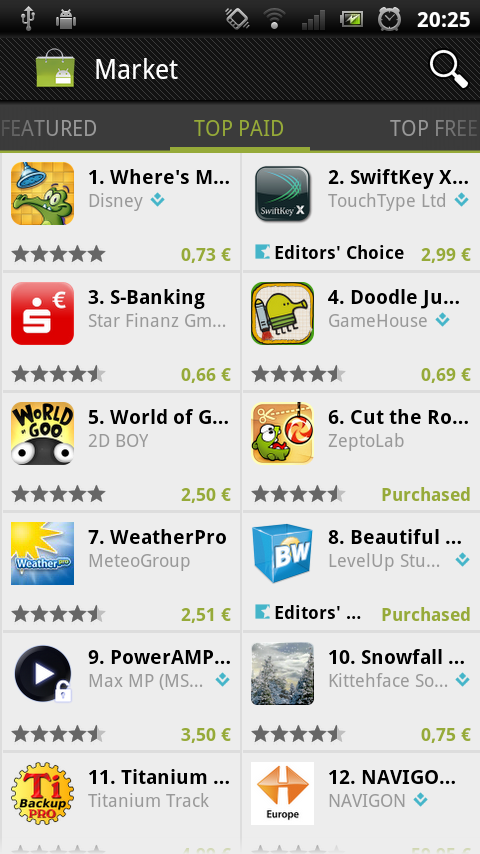

##### **Больше информации о списках**

Для получения большей информации о технических деталях работы со списками можно посмотреть следующие две превосходные серии статей:  
  
Styling Android's Mark Allison:  
[ListView – Part 1](http://blog.stylingandroid.com/archives/605)  
[ListView – Part 2](http://blog.stylingandroid.com/archives/617)  
[ListView – Part 3](http://blog.stylingandroid.com/archives/623)  
[ListView – Part 4](http://blog.stylingandroid.com/archives/632)  
  
AndroidDevBlog's Cyril Mottier:  
[ListView Tips & Tricks #1: Handle emptiness](http://android.cyrilmottier.com/?p=422)  
[ListView Tips & Tricks #2: Section your ListView](http://android.cyrilmottier.com/?p=440)  
[ListView Tips & Tricks #3: Create fancy ListViews](http://android.cyrilmottier.com/?p=454)  
[ListView Tips & Tricks #4: Add several clickable areas](http://android.cyrilmottier.com/?p=525)

#### Вкладки

Многие приложения в той или иной форме используют вкладки, чтобы помочь пользователям перемещаться между страницами. Android версии Honeycomb (3,0) и Ice Cream Sandwich (4,0) немного изменил способ работы и внешний вид вкладок. Мое мнение, что мы должны попытаться использовать это нововведение во всех наших приложениях, независимо от версии, на которой они работают.  
  
Если ваше приложение использует вкладки, то пользователи ожидают, что можно перемещаться между ними просто проведя пальцем.  
  
  
  
Хорошими примерами приложений, которые используют слайдинг для навигации между вкладками являются Android Market и Google+.

Примеры


Mark Allison написал несколько превосходных статей о технической реализации данной темы:  
[ViewPager – Part 1](http://blog.stylingandroid.com/archives/537)  
[ViewPager – Part 2](http://blog.stylingandroid.com/archives/542)  
[ViewPager – Part 3](http://blog.stylingandroid.com/archives/577)  
  
Проект Jake Wharton's показывает как можно работать с вкладками:  
[ViewPagerIndicator](https://github.com/JakeWharton/Android-ViewPagerIndicator) на GitHub

#### Заключение

Android быстро становится зрелой и консистентной платформой. Внешний вид и поведение приложения начинают походить друг на друга, и пользователи начинают ожидать определенного взаимодействия с пользовательским интерфейсом. Пока нет официальных рекомендаций, более глубокий взгляд на наиболее известные приложения дает нам хорошее понимание того, что мы должны делать.  
  
P.s. статья немного запылилась, но думаю будет полезна.

Единицы измерения

Исторически так сложилось, что разработчики всегда использовали пиксели при создании компьютерных интерфейсов. Но с развитием технологий данный подход стал источником проблем - на многих новых дисплеях элементы стали очень мелкими при установленных высоких разрешениях. Поэтому стали появляться новые единицы измерения, не зависящие от разрешения экрана.

Android поддерживает несколько стандартных единиц измерения. Вкратце перечислим их.

* px (pixels) — пиксели. Точки на экране - минимальные единицы измерения;
* dp (density-independent pixels) — независимые от плотности пиксели. Абстрактная единица измерения, основанная на физической плотности экрана с разрешением 160 dpi. В этом случае 1dp = 1px;
* dip - синоним для dp. Иногда используется в примерах Google;
* sp (scale-independent pixels) — независимые от масштабирования пиксели. Допускают настройку размеров, производимую пользователем. Полезны при работе с шрифтами;
* in (inches) — дюймы, базируются на физических размерах экрана. Можно измерить обычной линейкой;
* mm (millimeters) — миллиметры, базируются на физических размерах экрана. Можно измерить обычной линейкой;
* pt (points) — 1/72 дюйма, базируются на физических размерах экрана;

Как правило, при установке размера текста используются единицы измерения **sp**, которые наиболее корректно отображают шрифты:

android:textSize="48sp"

В остальных случаях рекомендуется использовать **dp**.

Переводим dp в пиксели

Так как на разных устройствах **dp** может различаться, то для получения величин в пикселях и наоборот используйте методы:

private float dpFromPx(float px) {

return px

/ getApplicationContext().getResources().getDisplayMetrics().density;

}

private float pxFromDp(float dp) {

return dp

\* getApplicationContext().getResources().getDisplayMetrics().density;

}

Если вы не определили размеры в XML, то их можно задать программно с помощью следующего кода (устанавливаем отступы для компонента):

final float scale = getResources().getDisplayMetrics().density;

int padding\_5dp = (int) (5 \* scale + 0.5f);

int padding\_20dp = (int) (20 \* scale + 0.5f);

int padding\_50dp = (int) (50 \* scale + 0.5f);

RadioButton rb = new RadioButton(this);

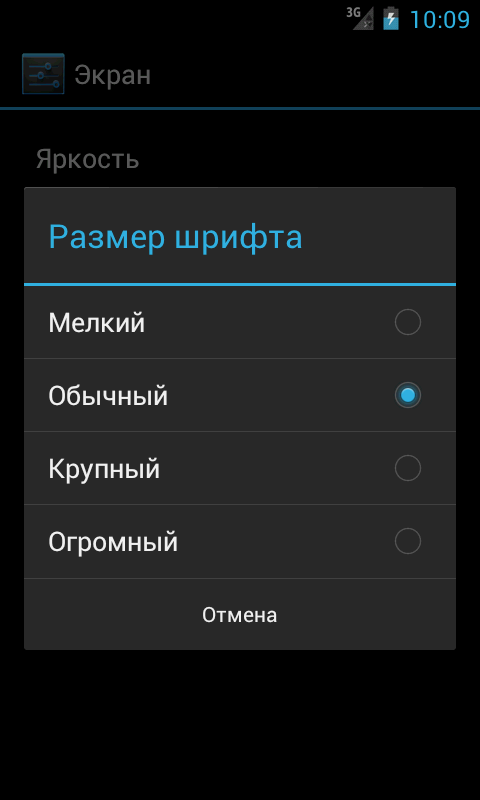
rb.setText("My Radio Button");

rb.setLayoutParams(new LayoutParams(padding\_50dp,padding\_50dp));

rb.setPadding(padding\_20dp,padding\_5dp,padding\_5dp,padding\_5dp);

Настройка шрифтов

Давайте чуть подробнее поговорим о работе со шрифтами, чтобы лучше понять специфику работы с текстами. Все люди разные - у кого-то зрение хорошое, у кого-то плохое. Android позволяет в настройках задать размеры шрифта в четырёх вариантах: Мелкий, Обычный, Крупный, Огромный. Для этого нужно зайти в Настройки | Экран | Размер шрифта.



Можно узнать программно выбранный вариант через свойство **fontScale**:

float fontScale = getResources().getConfiguration().fontScale;

mInfoTextView.setText("fontScale:: " + fontScale);

Обычному шрифту соответствует значение 1, мелкому - 0.9, крупному - 1.1, огромному - 1.15.

Если вы хотите, чтобы ваш текст мог меняться в зависимости от выбора пользователя, то используйте единицы измерения SP:

<TextView

android:id="@+id/textView1"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Текст с размером 26sp"

android:textSize="26sp" />

В тех случаях, когда изменять текст по желанию пользователя не следует, но при этом текст должен отображаться корректно в зависимости от разрешения экрана устройства, то используйте единицы измерения DP.

Третий вариант - если вы ни при каких обстоятельствах (какой же вы упрямый) не хотите зависеть от предпочтений пользователя и разрешения экрана, то пользуйтесь PX (пиксели). Среда разработки будет сопротивляться вашему желанию и выводить предупреждающие сообщения. Подумайте ещё раз о своём решении.

Используем стандартные системные размеры шрифтов

В Android зашиты три системный размера шрифтов, основанных на SP: Small, Medium и Large. Вы можете использовать их в стандартных случаях, когда вам не нужно задавать конкретные значения (атрибут style):

<TextView

android:id="@+id/textView1"

style="@android:style/TextAppearance.Small"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Пример текста - Small" />

<TextView

android:id="@+id/textView2"

style="@android:style/TextAppearance.Medium"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Пример текста - Medium" />

<TextView

android:id="@+id/textView3"

style="@android:style/TextAppearance.Large"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Пример текста - Large" />

На самом деле стилей **@android:style/TextAppearance.\*** гораздо больше. Если вы вдруг забыли про названия стилей, то можете использовать встроенные возможности среды разработки. На панели инструментов виджет TextView представлен в четырёх вариантах: TextView, Large, Medium, Small, и в них используется атрибут **android:textAppearance**.

Создадим проект со всеми возможными вариантами и посмотрим на результат. В первом случае будем использовать стандартные настройки шрифта, во втором - увеличим его.

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

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

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:orientation="vertical" >

<TextView

android:id="@+id/textView1"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Пример текста. Размер 20px"

android:textSize="20px" />

<TextView

android:id="@+id/textView2"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Пример текста. Размер не указан" />

<TextView

android:id="@+id/textView3"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Пример текста. textAppearanceSmall"

android:textAppearance="?android:attr/textAppearanceSmall" />

<TextView

android:id="@+id/textView4"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Пример текста. textAppearanceMedium"

android:textAppearance="?android:attr/textAppearanceMedium" />

<TextView

android:id="@+id/textView5"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Пример текста. textAppearanceLarge"

android:textAppearance="?android:attr/textAppearanceLarge" />

<TextView

android:id="@+id/textView6"

style="@android:style/TextAppearance.Small"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Пример текста. TextAppearance.Small" />

<TextView

android:id="@+id/textView7"

style="@android:style/TextAppearance.Medium"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Пример текста. TextAppearance.Medium" />

<TextView

android:id="@+id/textView8"

style="@android:style/TextAppearance.Large"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Пример текста. TextAppearance.Large" />

<TextView

android:id="@+id/textView9"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Пример текста. Размер 30sp"

android:textSize="30sp" />

</LinearLayout>

