

ПРОГРАММИРОВАНИЕ МОБИЛЬНЫХ
ПРИЛОЖЕНИЙ ПОД ПЛАТФОРМУ

ANDROID



Урок № 6

Material Design

Содержание

1. Material Design — концепция дизайна для приложений Android	4
1.1. Тактильные поверхности	9
1.2. Полиграфический дизайн	10
1.3. Осмысленная анимация	16
1.4. Адаптивный дизайн	17
2. Библиотеки совместимости AppCompat v.7 и Design Support Library для Material Design	19
2.1. Базовые виджеты в библиотеке AppCompat v.7	21
2.2. Навигационное меню.	
Виджеты DrawerLayout и NavigationView.....	28
2.3. Плавающая кнопка.	
Виджет FloatingActionButton	49
2.4. Тост с обратной связью. Виджет Snackbar	55
2.5. Менеджер раскладки CoordinatorLayout	64

2.6. Контейнер NestedScrollView	106
2.7. Контейнер AppBarLayout.....	113
2.8. Контейнер CollapsingToolbarLayout.....	121
2.9. Создание карточек. Виджет CardView.....	140
2.10. Список RecyclerView.....	146
2.11. Контейнер ViewPager.....	167
2.12. Менеджер TabLayout.....	185
3. Домашнее задание	198

1. Material Design — концепция дизайна для приложений Android

Material Design — концепция дизайна программного обеспечения и приложений операционной системы Android от компании Google. Material Design впервые был представлен в 2014 году. Идея дизайна, которую представляет Material Design, заключается в создании приложений, которые открываются и сворачиваются как физические (материальные) карточки. Как и все физические объекты, они должны отбрасывать тень и иметь некоторую инерционность, у приложений не должно быть острых углов, карточки должны переключаться между собой плавно и практически незаметно. Примеры внешнего вида дизайна концепции Material Design изображены на Рис. 1.1.

Когда-то все программные продукты компании Google выглядели по-разному. Даже один и тот же программный продукт на разных платформах выглядел по-разному. В 2011 году специалисты компании Google начали работать над унификацией визуальной части своих программных продуктов. Результат в первую очередь коснулся веб-приложений, но затронул и некоторые мобильные продукты. Однако, пользователям приложений приходилось подстраиваться под новый интерфейс при

переключении между платформами, привыкать к внешнему виду и расположению элементов управления и так далее. Поэтому в какой-то момент группа дизайнеров компании Google решила справиться с проблемой раз и навсегда.

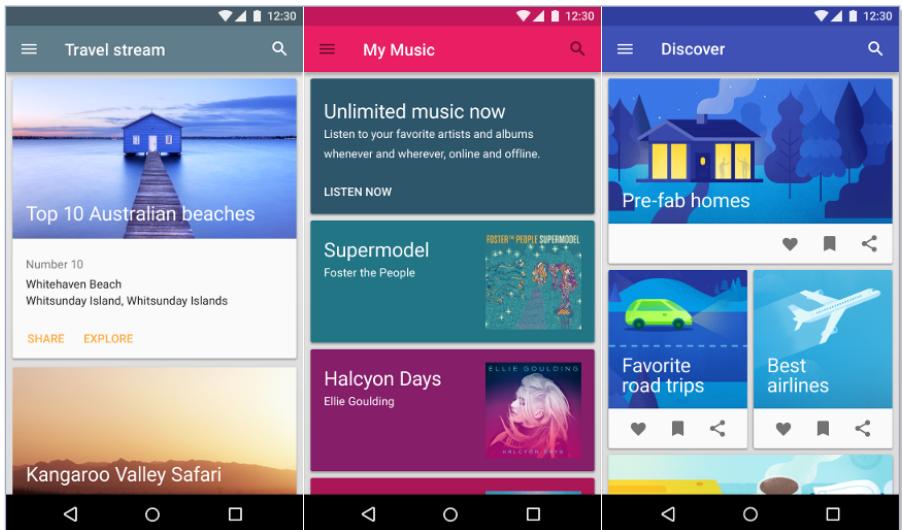


Рис. 1.1. Примеры дизайна концепции Material Design

В 2014 году на конференции Google I/O была представлена новая концепция, которая получила название Material Design. Эта концепция позволяет создавать одинаковый пользовательский дизайн на всех платформах: веб, десктопные приложения, смартфоны, планшеты, часы, телевизоры.

Концепция Material Design позволяет более объективно подходить к созданию дизайна. Концепция описывает, что и как должно выглядеть, как это должно работать и как при этом должна осуществляться анимация.

Концепция Material Design задает разумные рамки, но не излишние ограничения. Настоятельно рекомендуем изучать официальную документацию по применению Material Design, которую можно найти по адресу <https://material.io/guidelines>.

Для разработки Android приложений в соответствии с концепцией Material Design, разработчикам предоставляются следующие элементы:

- **Новая тема.**

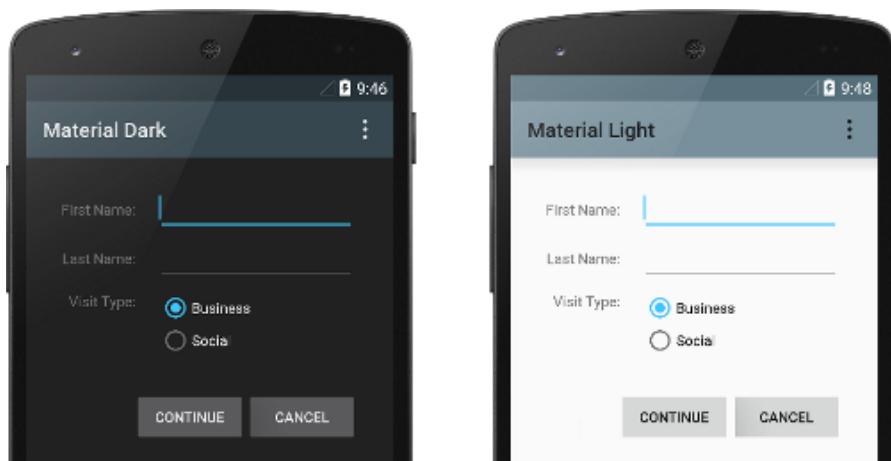


Рис. 1.2. Тема Material Design в темных и светлых тонах

Тема Material Design предоставляет новый стиль для приложений. Тема позволяет подбирать подходящие для приложения цвета (см. Рис 1.2). В теме имеются и системные виджеты, для которых можно также настраивать цветовую палитру. А еще в теме есть анимации, выполняемые по умолчанию в качестве реакции на касание и при переходах между действиями.

Настройка цветов производится через атрибуты темы, которые автоматически используются всеми компонентами приложения, например **colorPrimaryDark** для StatusBar и **colorPrimary** для ToolBar или ActionBar (см. Рис. 1.3).

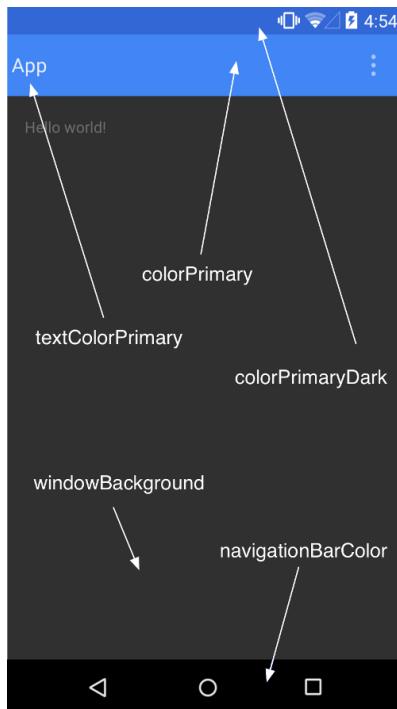


Рис. 1.3. Атрибуты темы Material Design для настройки цветовой палитры

- **Новые виджеты для сложных представлений.**

Разработчикам Android приложений предоставляются новые виджеты. Например, виджеты [android.support.v7.widget.RecyclerView](#) и [android.support.v7.widget.CardView](#), внешний вид которых изображен на Рис. 1.4.

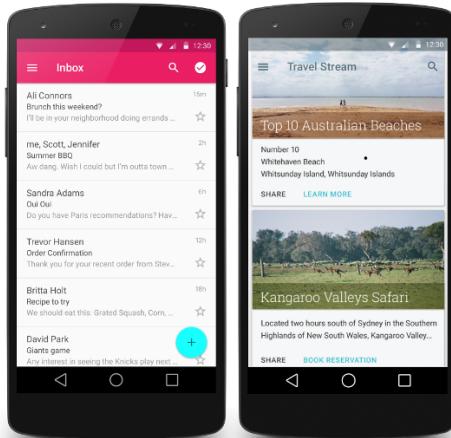


Рис. 1.4. Внешний вид новых виджетов android.support.v7.widget.RecyclerView и android.support.v7.widget.CardView

Виджет **android.support.v7.widget.RecyclerView** представляет собой более гибкую версию виджета **android.widget.ListView**, которая поддерживает различные типы макетов и является более производительной. Виджет **android.support.v7.widget.CardView** позволяет отображать важные элементы имеющие согласованный внешний вид и поведение.

- **Тени и новые API-интерфейсы анимации.**

Помимо свойств X и Y, виджеты в Android теперь еще имеют свойство Z. Это новое свойство показывает, насколько виджет «приподнят» над поверхностью родительского контейнера. Чем больше виджет приподнят, тем больше размер тени, которую он отбрасывает. Также, виджеты с более высоким значением свойства Z отображаются поверх других виджетов у которых свойство Z меньше.

Новые API-интерфейсы анимации позволяют создавать нестандартную анимацию для реакции на касание в элементах пользовательского интерфейса, изменения состояния виджетов и переходов между Активностями.

Концепция Material Design базируется на следующих принципах:

- Тактильные поверхности.
- Полиграфический дизайн.
- Осмысленная анимация.
- Адаптивный дизайн.

Давайте вкратце познакомимся с этими базовыми принципами концепции Material Design. Более подробно об этих базовых принципах рекомендуем почитать в статье по ссылке <https://habrahabr.ru/company/redmadrobot/blog/252773/>.

1.1. Тактильные поверхности

Поверхность, это контейнер, содержащий в себе информацию. Пользователь может прикасаться к поверхности и его прикосновения могут привести к какой-либо реакции. Сам термин «тактильность» означает прикосновение, возможность к чему-то прикасаться.

Поверхность также можно сравнить с кусочком бумаги. Только в данном случае речь идет о «цифровой бумаге», которая отображается на экране устройства и которая может изменять свой размер (адаптироваться). Также как и листы реальной бумаги, листы «цифровой бумаги» имеют толщину и могут располагаться друг над другом. Толщина «цифровой бумаги» подчеркивается при по-

моши теней. Чем выше листочек «цифровой бумаги» (то есть чем больше значение свойства Z) тем длиннее он отбрасывает тень. В концепции Material Design, интерфейс приложения должен складываться из листочков «цифровой бумаги» (или, как уже упоминалось выше — карточек). С этими карточками и листочками пользователь осуществляет взаимодействие — выполняет скроллинг, прикасается, перемещает и т.д. Идея интерфейса приложения, состоящего из тактильных поверхностей изображена на Рис. 1.5.

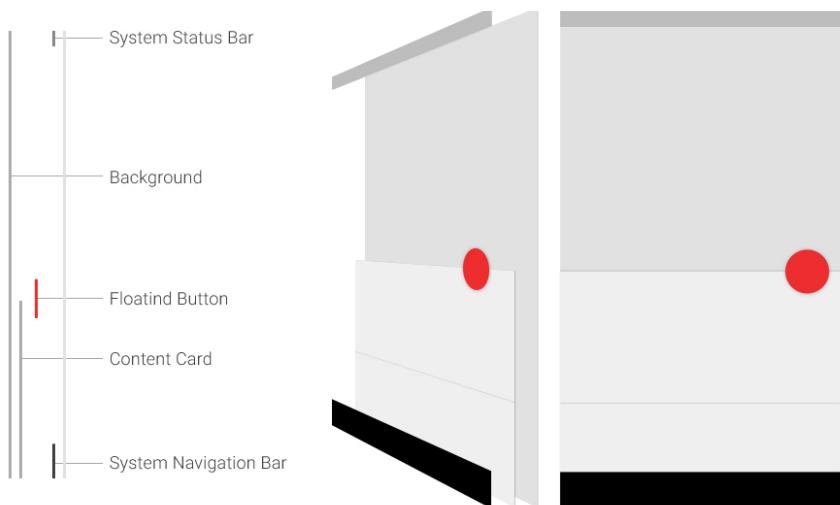


Рис. 1.5. Идея построения интерфейса приложения в Material Design

1.2. Полиграфический дизайн

Material Design использует классические принципы полиграфического дизайна при создании интерфейсов приложений. Если быть более конкретными, то Material

Design использует журнальный или плакатный дизайн. Возьмем к примеру, любое брендовое журнальное издание. Что самое первое бросается в глаза? — шрифт. Выбор шрифта является ключевым, так как он образует стиль журнального издания. Аналогично и в Material Design — выбор шрифта является стилеобразующим элементом дизайна приложения (см. Рис. 1.6).

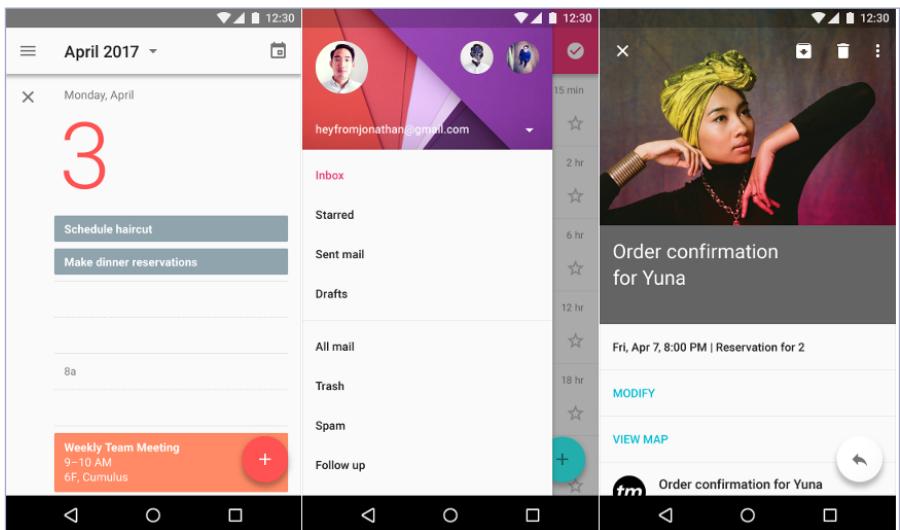


Рис. 1.6. Примеры дизайна приложений с полиграфическим дизайном

На текущий момент, стандартными шрифтами Material Design являются шрифты Roboto и Noto. Выбору размера и стиля шрифта также уделяется большое значение (см. Рис. 1.7). С рекомендациями по подбору размеров шрифтов можно ознакомиться на странице <https://material.io/guidelines/style/typography.html#typography-styles>.



Рис. 1.7. Примеры размеров и стилей стандартного шрифта Roboto с сайта <https://material.io/guidelines/style/typography.html#typography-typeface>

Как и в полиграфическом дизайне, в дизайне интерфейсов цвет является важным средством выразительности. В Material Design стандартная цветовая палитра приложения состоит из основного и акцентного цветов. Примеры применения основного и акцентного (вторичного) цветов изображены на Рис. 1.8 и Рис. 1.9.

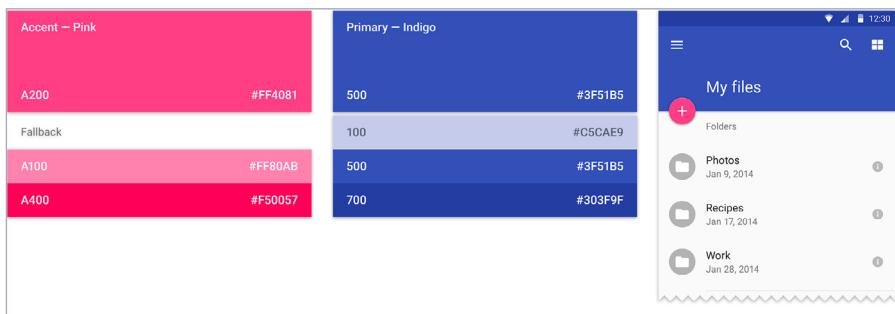


Рис. 1.8. Пример применения основного (первичного) и акцентного (вторичного) цветов в Material Design

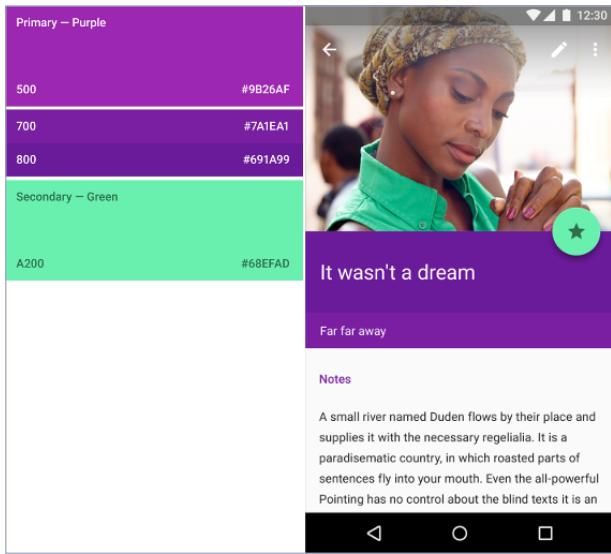


Рис. 1.9. Пример применения основного (первичного) и акцентного (вторичного) цветов в Material Design

В Material Design основной цвет относится к цвету, который чаще всего появляется в вашем приложении. Основной цвет применяется для больших областей, таких как ToolBar (значение палитры 500), в более темную вариацию основного цвета окрашивается StatusBar (значение палитры 700). Вторичный, более яркий цвет, относится к цвету, используемому для привлечения внимания пользователя на ключевые элементы вашего пользовательского интерфейса, такие как плавающая кнопка. Вторичный цвет используется также в кнопках, индикаторах и т. д. Вторичный цвет в Material Design называется акцентным — с его помощью делается акцент на ключевые элементы интерфейса. Рекомендации по использованию цветов, а также примеры цветовых палитр смотрите на

официальном сайте Material Design: <https://material.io/guidelines/style/color.html>.

Кроме цвета и шрифта, в полиграфическом дизайне большое внимание отводится размещению информации (размещению контента) на экране. В полиграфическом дизайне применяют так называемые модульные (или базовые) сетки. Шаг в базовой сетке Material Design составляет 8dp. Также базовая сетка Material Design задает отступы от краев экрана, структурирует информацию, делает представление этой информации более наглядной для пользователя. На Рис. 1.10 изображен пример базовой сетки Material Design для панели навигации, а готовую панель навигации из макета Рис. 1.10 можно увидеть на среднем скриншоте на Рис. 1.6.

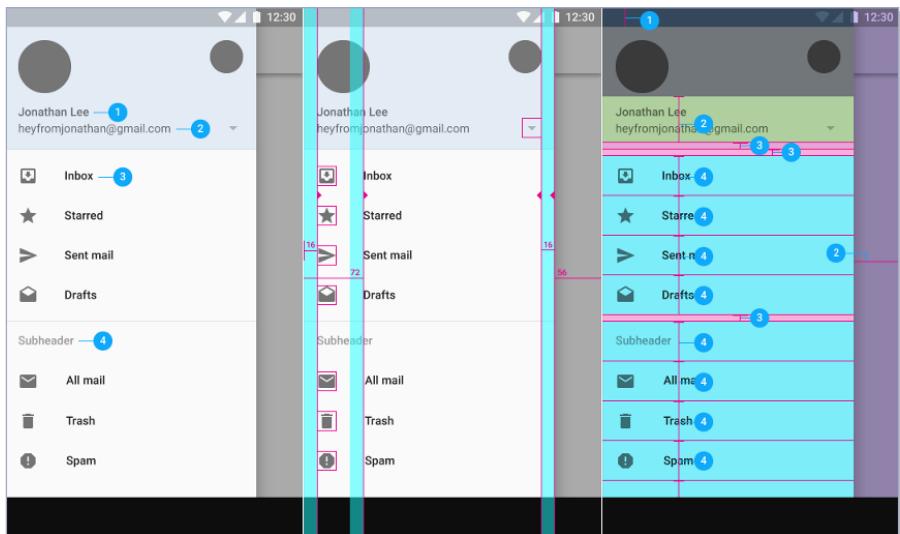


Рис. 1.10. Пример базовой сетки Material Design
и рекомендации по размещению контента

Более подробно ознакомиться с рекомендациями, продемонстрированными на Рис. 1.10 можно по адресу <https://material.io/guidelines/patterns>.

И последнее, в Material Design, так же как и в полиграфическом дизайне, приветствуется использование фотографий и всевозможных иллюстраций. Изображения чаще всего размещаются без рамок, часто «навылет». Применение фотографий и иллюстраций в Material Design предназначены опять же, для выразительности дизайна вашего приложения. На Рис. 1.11 показаны примеры использования фотографий и иллюстраций.

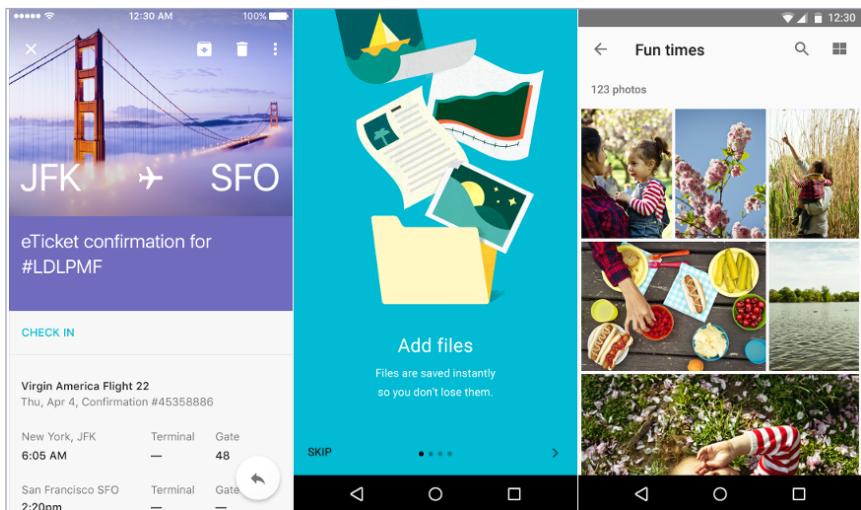


Рис. 1.11. Примеры использования фотографий и иллюстраций в Material Design

Ознакомиться с рекомендациями Material Design по применению фотографий и изображений в дизайне приложений можно на официальном сайте по адресу <https://material.io/guidelines/style/imagery.html>.

1.3. Осмысленная анимация

В реальном мире объекты не могут моментально появляться из ниоткуда или исчезать в никуда. Поэтому и в Material Design осмысленная анимация используется, чтобы показать, что именно только что произошло. Активное движение в дизайне привлекает взгляд пользователя — это естественно для человеческого зрения. С помощью анимации происходит управление вниманием пользователя.

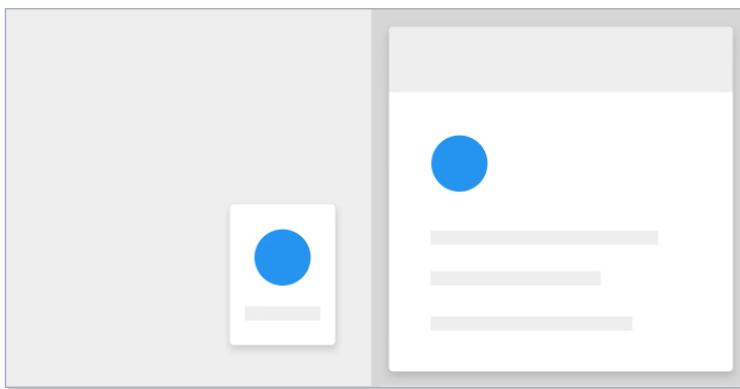


Рис. 1.12. Скриншоты видеороликов демонстрирующих асимметричную анимацию в Material Design с сайта <https://material.io/guidelines/motion/material-motion.html>

При использовании анимации необходимо помнить о том, что глубина интерфейса приложения ограничена глубиной устройства, поэтому перемещения элементов интерфейса необходимо производить в плоскости. При этом анимация должна быть асимметричной, то есть изменение ширины и высоты элементов интерфейса должны быть независимыми друг от друга, в противном слу-

чае, зависимость ширины и высоты элементов при анимации приведет к эффекту удаления или приближения на величину расстояний, больших чем глубина экрана, что не рекомендуется концепцией Material Design.

Также анимация перемещения элементов пользовательского интерфейса должна быть быстрой и четкой, чтобы пользователю приходилось меньше ждать, так как ожидание, как правило, раздражает пользователей.

Познакомиться с рекомендациями Material Design по анимации перемещений элементов пользовательского интерфейса можно и нужно по адресу <https://material.io/guidelines/motion/material-motion.html>. Также еще раз напомним, что есть очень хорошая статья с иллюстрациями и видеороликами о концепции Material Design по адресу <https://habrahabr.ru/company/redmadrobot/blog/252773/>.

1.4. Адаптивный дизайн

Еще один базовый принцип концепции Material Design — это Адаптивный дизайн. Адаптивный дизайн это подход к созданию дизайна приложений, который позволяет отображать информацию на экранах разных устройств при этом сохраняя главный стиль и концепцию дизайна (см. Рис. 1.13).

Основными рекомендациями Material Design для адаптивного дизайна являются рекомендации по отображению информации блоками (так как это упрощает работу со свободным пространством на больших экранах), рекомендации по отступам (отступы слева в 72dp и выше упоминавшаяся базовая сетка в 8dp и так далее), рекомендации по размерам и рекомендации по использо-

зованию тулбара. В прошлом уроке мы уже рассказывали о том, как использовать тулбар (виджет `android.support.v7.widget.Toolbar`), а также упоминали о том, что `ActionBar` уже считается устаревшим. Так вот, тулбар в Material Design является функциональным и выразительным блоком управления приложения, так как теперь в тулбар можно поместить такие функциональные элементы как: поля ввода, плавающая кнопка, формы. Тулбар может скрываться и появляться, или изменять свой размер в зависимости от функциональности приложения. Тулбар является виджетом, а значит им можно управлять также, как и другими виджетами.



Рис. 1.13. Концепция адаптивного дизайна для разных экранов и устройств

В этой главе мы вкратце познакомили вас с принципами Material Design. Авторы урока не ставили перед собой цель рассказать более подробно и глубоко о концепции Material Design. Однако, мы настоятельно рекомендуем изучать и применять эту концепцию для создания современных приложений с красивым и унифицированным пользовательским интерфейсом. Руководство по Material Design доступно на сайте <https://material.io/guidelines>.

2. Библиотеки совместимости AppCompat v.7 и Design Support Library для Material Design

Виджеты, менеджеры раскладки (контейнеры), стили, темы, тени и анимации Material Design доступны разработчикам Android приложений начиная с версии API 21. Однако, если создавать приложения которые могут использоваться только на операционных системах Android начиная с версии API 21 и выше, то такие приложения не смогут охватить все устройства, которые в настоящий момент находятся в использовании у людей по всему миру. Компания Google предлагает решение данной проблемы в виде специальной библиотеки совместимости AppCompat (которая раньше называлась *ActionBarCompat*) версии v7. При помощи библиотеки совместимости AppCompat v.7 можно использовать элементы Material Design в ранних версиях операционных систем Android (начиная с версии API 7). Также, в библиотеке AppCompat получили новое визуальное оформление в стилистике Material Design такие базовые виджеты как **TextView**, **EditText**, **Button**, **Spinner**, **CheckBox**, **RadioButton**.

Также, кроме расширения библиотеки AppCompat v.7, компания Google создала еще одну библиотеку совместимости Design Support Library. В этой библиотеке находятся классы для таких виджетов как: удобное боковое меню (*NavigationView*), плавающая кнопка (*Floating*

Action Button), всплывающее сообщение (*Snackbar*), анимационный Toolbar и многое другое.

Библиотеки AppCompat v.7 и Design Support Library можно подключать к модулям проекта как целиком, так и по отдельности для отдельных классов. Для подключения к модулю проекта необходимо внести соответствующие записи (см. Листинг 2.1) в файл build.gradle для модуля в секцию dependencies.

Листинг 2.1. Подключение библиотек совместимости или отдельных классов из этих библиотек к модулю проекта Android Studio

```
dependencies {  
    compile fileTree(dir: 'libs', include: ['*.jar'])  
    testCompile 'junit:junit:4.12'  
    compile 'com.android.support:appcompat-v7:23.1.1'  
    compile 'com.android.support:design:23.1.1'  
    compile 'com.android.support:recyclerview-v7:23.1.1'  
    compile 'com.android.support:palette-v7:23.1.1'  
    compile 'com.android.support:cardview-v7:23.1.1'  
}
```

В Листинге 2.1 указана версия библиотек под номером 23.1.1, но на самом деле это не принципиально. Android Studio сама предложит вам номер последней версии и даже загрузит библиотеки этой версии из Интернет в случае необходимости. Авторы урока намеренно используют версию 23.1.1 (API 23) в этом уроке с одной лишь целью — далеко не у всех людей, изучающих наши уроки, есть мощные компьютеры, способные предоставить для Android Studio ресурсы для комфортной работы. Версия библиотек и SDK API 23 потребляет меньше ре-

урсов чем более поздние версии API, но при этом полноценно позволяет создавать приложения в стиле Material Design, а значит, можно изучать учебный материал наших уроков не делая существенных затрат на приобретение новой компьютерной техники.

Давайте перейдем к изучению интересных виджетов, которые появились в рассматриваемых библиотеках со-вместимости для концепции Material Design.

2.1. Базовые виджеты в библиотеке AppCompat v.7

Как уже упоминалось выше, в концепции Material Design базовые виджеты (текстовые поля, кнопки, выпадающие списки, чекбоксы и т.д) получили новое визуальное оформление. Внешний вид этих базовых виджетов в Material Design теперь выглядит так, как показано на Рис. 2.1.

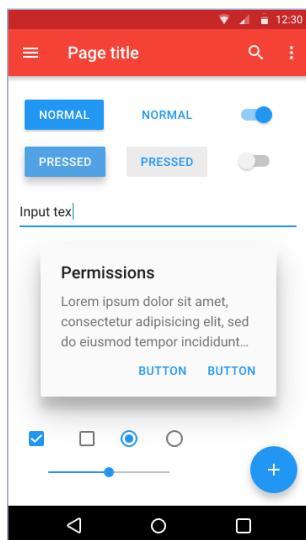


Рис. 2.1. Внешний вид базовых виджетов в Material Design

Дизайнеры компании Google очень досконально подошли к требованиям по внешнему виду для базовых виджетов. К каждому виджету в стилистике Material Design прилагается четкий набор правил по размерам, цветам, отступам. В качестве иллюстрации этих правил на Рис. 2.2 показано изображение с сайта рекомендаций Material Design (<https://material.io/guidelines/components/buttons.html#buttons-style>). Мы отдаём должное подобной скрупулезности, с которой дизайнеры компании Google подошли к концепции Material Design. На текущий момент (см. материалы прошлых уроков этого курса) мы с вами владеем знаниями, при помощи которых можно реализовать подобные требования. Давайте представим, как бы мы поступили для реализации внешнего вида кнопок **android.widget.Button** в стиле Material Design.

Во-первых, в файле /res/values/styles.xml необходимо было бы создать стиль, который показан в Листинге 2.2.

Листинг 2.2. Пример стиля для кнопки android.widget.Button в стилистике Material Design

```
<style name="Button">
    <item name="android:textColor">@color/white</item>
    <item name="android:paddingLeft">16dp</item>
    <item name="android:paddingRight">16dp</item>
    <item name="android:minWidth">88dp</item>
    <item name="android:minHeight">36dp</item>
    <item name="android:layout_margin">8dp</item>
    <item name="android:background">
        @drawable/primary_round</item>
</style>
```

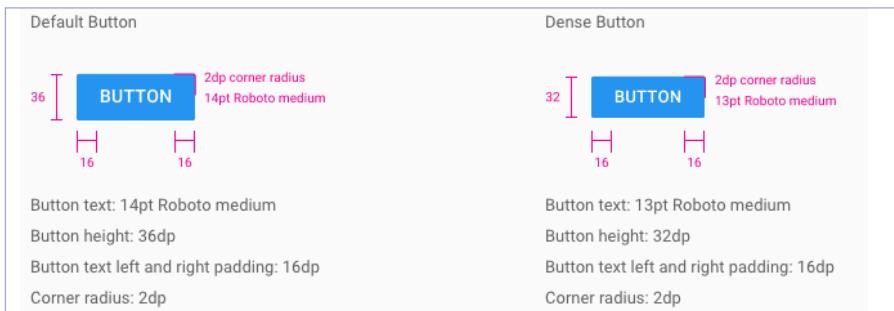


Рис. 2.2. Рекомендации Material Design
по внешнему виду кнопок android.widget.Button

Во-вторых, в каталоге ресурсов /res/drawable необходимо было бы создать файл ресурсов /res/drawable/primary_round.xml (идентификатор файла ресурсов в Листинге 2.2 выделен жирным шрифтом). Этот файл ресурсов описывает анимацию реакции на нажатие кнопки (*ripple*) и фон кнопки в виде фигуры прямоугольника с закругленными углами. Пример xml кода этого файла ресурсов приведен в Листинге 2.3.

Листинг 2.3. Файл ресурсов с анимацией реакции на касание и фоном для стиля кнопки android.widget.Button из Листинга 2.2

```
<ripple xmlns:android=
    "http://schemas.android.com/apk/res/android"
    android:color="@color/primary_600">
    <item>
        <shape>
            <corners android:radius="2dp" />
            <solid android:color="@color/primary_500" />
        </shape>
    </item>
</ripple>
```

И последнее, в файле макета Активности /res/layout/activity_main.xml необходимо было бы создать кнопку **android.widget.Button** которой назначается стиль из Листинга 2.2 (см. Листинг 2.4).

Листинг 2.4. Пример виджета android.widget.Button с использованием стиля из Листинга 2.2

```
<Button  
    style="@style/Button"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    ...  
/>
```

Как видно, ничего сложного в создании виджетов в стиле концепции Material Design нет, за исключением того, что на эту работу необходимо тратить дополнительное время. Чтобы этого не делать, можно использовать виджеты из библиотеки AppCompat v.7. Названия классов виджетов из этой библиотеки начинаются с префикса **AppCompat**, другими словами класс кнопки **Button** будет иметь название **AppCompatButton**, класс текстового поля **TextView** будет иметь название **AppCompatTextView** и так далее. Поэтому можно легко перейти на использование этих виджетов. В Листинге 2.5 показано применение виджетов в макете Активности /res/layout/activity_main.xml.

Листинг 2.5. Использование базовых виджетов из библиотеки AppCompat v.7

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

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:orientation="vertical"
    tools:context="itstep.com.app.MainActivity">

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:background="#F44336"      />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:paddingLeft="16dp">

        <android.support.v7.widget.AppCompatTextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="21sp"
            android:text="Widgets in AppCompat v.7"
            android:textColor="#000000"
            />

        <View
            android:layout_width="match_parent"
            android:layout_height="1dp"
            android:layout_marginTop="8dp"
            android:layout_marginRight="16dp"
            android:background="#B0B0B0"  />

        <android.support.v7.widget.AppCompatTextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="21sp"
            android:text="AppCompatTextView"
            />
```

```
<android.support.v7.widget.AppCompatButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="AppCompatButton"
    android:textColor="#FFFFFF"
    app:backgroundTint="#2196f3"
    android:textAllCaps="false"
/>

<android.support.v7.widget.AppCompatCheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="AppCompatCheckBox"
/>

<android.support.v7.widget.AppCompatEditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginRight="16dp"
    android:text="AppCompatEditText"
/>

<RadioGroup
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

    <android.support.v7.widget.
        AppCompatRadioButton
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Black" />

    <android.support.v7.widget.
        AppCompatRadioButton
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="White" />

```

```
</RadioGroup>  
  
</LinearLayout>  
</LinearLayout>
```

Внешний вид примера из Листинга 2.5 изображен на Рис. 2.3.

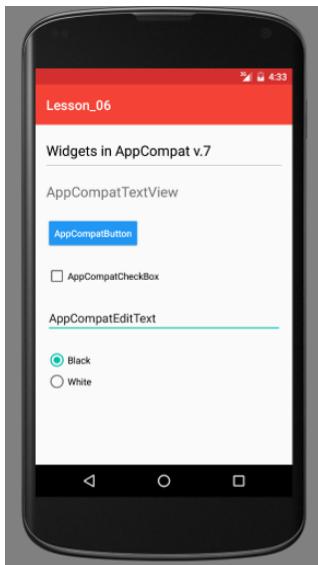


Рис. 2.3. Внешний вид базовых виджетов из библиотеки AppCompat v.7

Если вы внимательно посмотрите на Рис. 2.3, то увидите, что внешней разницы между обычными виджетами и виджетами библиотеки AppCompat v.7 нет. Это действительно так, потому что, исходя из официальной документации компании Google для разработчиков, при использовании обычных виджетов, происходит их авто-

матическая замена на виджеты из библиотеки совместимости AppCompat v.7. Разработчикам рекомендуют явно использовать названия классов виджетов из библиотеки AppCompat только в случае создания своих кастомных виджетов. Другими словами, оказывается, что мы уже давно используем виджеты из библиотеки AppCompat v.7 в своих учебных приложениях.

2.2. Навигационное меню. Виджеты **DrawerLayout** и **NavigationView**

В этом разделе мы рассмотрим применение бокового меню в приложениях. Боковое меню еще часто называют навигационным меню или панелью навигации (кому как удобно). Пример такого меню можно увидеть на Рис. 2.4.

Класс [android.support.design.widget.NavigationView](#) представляет стандартное меню навигации для приложения. Содержимое этого меню может быть заполнено файлом ресурсов меню.

Иерархия классов для класса [android.support.design.widget.NavigationView](#) имеет следующий вид:

```
java.lang.Object
|
+--- android.view.View
|
+--- android.view.ViewGroup
|
+--- android.widget.FrameLayout
|
+--- android.support.design.widget.NavigationView
```

Как видно из иерархии классов, виджет **android.support.design.widget.NavigationView** является наследником хорошо знакомого нам по прошлым урокам контейнера **android.widget.FrameLayout**.

Объект **android.support.design.widget.NavigationView** обычно размещается внутри **android.support.v4.widget.DrawerLayout**. Объект **android.support.v4.widget.DrawerLayout** обычно выступает в качестве контейнера верхнего уровня для содержимого окна, что позволяет выводить интерактивные «выдвижные» виды из одного или обоих вертикальных краев окна. Применительно к навигационному меню **NavigationView**, **DrawerLayout** позволяет плавно «выдвигать» (и конечно же «задвигать» обратно) навигационное меню **NavigationView**.

Иерархия классов для класса **android.support.v4.widget.DrawerLayout** выглядит следующим образом:

```
java.lang.Object
  |
  +--- android.view.View
    |
    +--- android.view.ViewGroup
      |
      +--- android.support.v4.widget.
          DrawerLayout
```

Как видно из названия пакетов, в которых объявлен класс **android.support.v4.widget.DrawerLayout**, класс добавлен в библиотеку AppCompat еще в версии v4.

Размещение объекта **NavigationView** внутри контейнера **DrawerLayout** как правило выглядит так, как показано в Листинге 2.6.

Листинг 2.6. Размещение объекта NavigationView внутри контейнера DrawerLayout

```
<android.support.v4.widget.DrawerLayout

    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:app=
        "http://schemas.android.com/apk/res-auto"

    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true">

    <!-- Your contents -->

    <android.support.design.widget.NavigationView
        android:id="@+id/navigation"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        app:menu="@menu/drawer_menu"
        app:headerLayout="@layout/drawer_header"
    />

</android.support.v4.widget.DrawerLayout>
```

Обратите внимание (см. Листинг 2.6), что кроме виджета **NavigationView**, в **DrawerLayout** можно вставить любой контейнер с контентом (в Листинге 2.6 место для вставки этого контейнера обозначено комментарием `<!-- Your Content -->`). Пункты меню для навигационного меню **NavigationView** задаются при помощи файла ресурсов, ссылка на который указывается при помощи атрибута

app:menu. Также, при помощи атрибута **app:headerLayout** задается файл ресурсов с макетом раскладки виджетов для внешнего вида заголовка навигационного меню **NavigationView**.

Давайте рассмотрим пример создания и использования навигационного меню. Для этого создадим модуль «app» в проекте с примерами, которые прилагаются к данному уроку.

В Листинге 2.7 показан файл ресурсов */res/layout/activity_main.xml* с макетом внешнего вида Активности.

Листинг 2.7. Файл */res/layout/activity_main.xml* с макетом внешнего вида Активности

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

<android.support.v4.widget.DrawerLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:app=
        "http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"

    android:id="@+id/drawerMain">

    <!-- Content layout -->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
```

```
    android:layout_width="match_parent"
    android:layout_height="60dp"
    android:background="#E91E63"
    android:subtitleTextColor="#FFFFFF"
    android:titleTextColor="#FFFFFF"
    />

<FrameLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="72dp"
    android:id="@+id/flContent">
    ...
</FrameLayout>
</LinearLayout>

<!-- NavigationView -->
<android.support.design.widget.NavigationView
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:layout_gravity="start"
    app:headerLayout="@layout/drawer_header"
    app:menu="@menu/drawer_menu"
    android:id="@+id/naviView"
    />

</android.support.v4.widget.DrawerLayout>
```

Как видно из Листинга 2.7, корневым элементом (и соответственно главным контейнером Активности) является контейнер **android.support.v4.widget.DrawerLayout** (идентификатор **R.id.drawerMain**). Контейнером для контента является контейнер **android.widget.LinearLayout**, в который вложен тулбар **android.support.v7.widget.Toolbar** (идентификатор **R.id.toolbar**) и контей-

нер `android.widget.FrameLayout` (идентификатор `R.id.flContent`). Контейнер `R.id.flContent` и предназначен в нашем примере для размещения информации (контента) нашего приложения. В Листинге 2.7 не показано содержимое контейнера `R.id.flContent`, так как этому содержимому посвящен отдельный листинг ниже. И наконец, в главном контейнере `DrawerLayout` находится панель навигации `android.support.design.widget.NavigationView` (идентификатор `R.id.navView`).

Для панели навигации (которая размещается в левой части `DrawerLayout`) созданы два файла ресурсов:

- Файл `/res/layout/drawer_header.xml` который содержит макет внешнего вида заголовка панели навигации (см. Листинг 2.8) и назначается панели навигации при помощи атрибута `app:headerLayout="@layout/drawer_header"`;
- Файл `/res/menu/drawer_menu.xml` который содержит описание пунктов меню для панели навигации (см. Листинг 2.9) и назначается панели навигации при помощи атрибута `app:menu="@menu/drawer_menu"`;

Листинг 2.8. Содержимое файла ресурсов `/res/layout/drawer_header.xml` рассматриваемого примера

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="120dp"
```

```
    android:background="#003366"
    android:gravity="bottom"
    android:paddingLeft="16dp">
<android.support.v7.widget.AppCompatTextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="17sp"
    android:textColor="#FFFFFF"
    android:text="Rest Places" />
</LinearLayout>
```

Листинг 2.9. Содержимое файла ресурсов /res/menu/drawer_menu.xml с пунктами меню для панели навигации рассматриваемого примера

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android=
        "http://schemas.android.com/apk/res/android">
<group android:checkableBehavior="single">
    <item
        android:id=
            "@+id/navigation_item_fintess_center"
        android:icon=
            "@drawable/ic_fitness_center_black_24dp"
        android:checked="true"
        android:title="Fitness Center" />
    <item
        android:id="@+id/navigation_item_casino"
        android:icon=
            "@drawable/ic_casino_black_24dp"
        android:title="Casino" />
    <item
        android:id="@+id/navigation_item_pool"
        android:icon="@drawable/ic_pool_black_24dp"
```

```
        android:title="Pool" />
    </group>
</menu>
```

Чтобы панель навигации **NavigationView** появилась, необходимо нажать пальцем на экране устройства в самой левой его части и не отпуская пальца потянуть вправо. Если вы используете эмулятор, то вместо пальца необходимо использовать курсор мыши. Внешний вид появления панели навигации из Листингов 2.7, 2.8, 2.9 изображен на Рис. 2.4 в виде последовательности скриншотов.

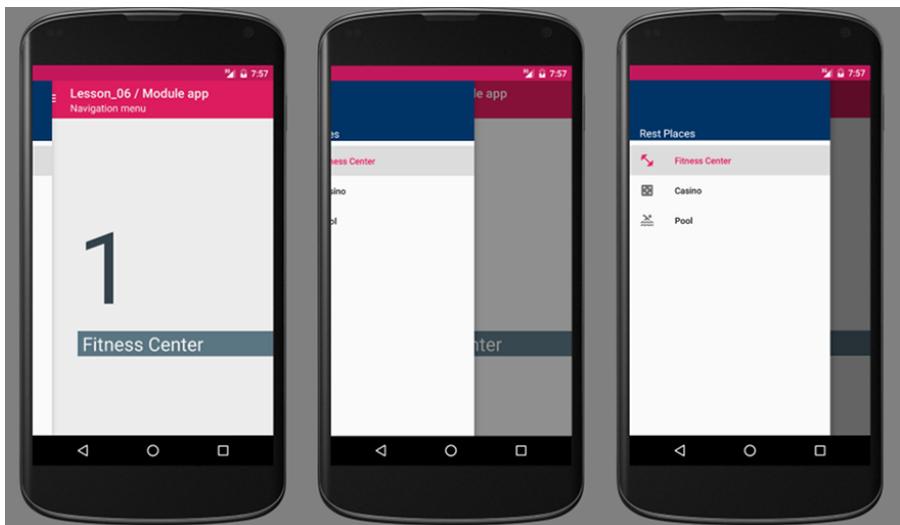


Рис. 2.4. Последовательность скриншотов появления панели навигации **NavigationView** из Листингов 2.7, 2.8, 2.9

Чтобы убрать навигационное меню **NavigationView** необходимо проделать обратные действия или нажать на области экрана, которая не содержит **NavigationView**.

Показывать и скрывать навигационное меню `android.support.design.widget.NavigationView` можно и программно. Для этого предназначены методы класса `android.support.v4.widget.DrawerLayout`:

- `void openDrawer(View drawerView)` — Показывает («выдвигает») виджет `drawerView`. В нашем примере в качестве значения для параметра `drawerView` будем передавать ссылку на объект `NavigationView`.
- `void openDrawer(View drawerView, boolean animate)` — То же самое, что и предыдущий метод, только в параметре `animate` передается значение, указывающее нужно ли «выдвигать» виджет `drawerView` с эффектом анимации или без него.
- `void openDrawer(int gravity)` — Показывает («выдвигает») виджет, который расположен в `DrawerLayout` с того края, который задается параметром `gravity`. Например, передав в качестве значения для параметра `gravity` значение `Gravity.LEFT` будет выдвинут виджет, находящийся с левого края от `DrawerLayout`. Если с левого края не будет расположено ни одного виджета, то произойдет исключительная ситуация.
- `void closeDrawer(View drawerView)` — Убирает («задвигает») ранее выдвинутый виджет, ссылка на который задается параметром `drawerView`.
- `void closeDrawer(View drawerView, boolean animate)` — То же, что и предыдущий метод, только в параметре `animate` передается значение, указывающее нужно ли «задвигать» виджет `drawerView` с эффектом анимации или без него.

- **void closeDrawer(int gravity)** — «Задвигает» виджет, который находится с того края **DrawerLayout**, который указан в параметре **gravity**.
- **void closeDrawers()** — «Задвигает» все ранее выдвинутые виджеты.

В нашем примере добавлена возможность выдвигать панель навигации **android.support.design.widgetNavigationView** при помощи нажатия на иконку навигации (*Navigation Icon*) в тулбаре, которую можно увидеть на скриншотах Рис. 2.5 в левой части тулбара. Программный код Java назначения иконки навигации тулбару **android.support.v7.widget.Toolbar** и назначение обработчика события нажатия на иконку навигации приведен в Листинге 2.10.

Листинг 2.10. Назначение иконки и обработчика события нажатия на иконку тулбара

```
toolBar.setNavigationIcon(
    R.drawable.ic_menu_white_24dp);
toolBar.setNavigationOnClickListener(
    new View.OnClickListener()
    {
        @Override
        public void onClick(View v)
        {
            MainActivity.this.drawerMain.
                openDrawer(MainActivity.this.naviView);
        }
    });
}
```

Как видно из Листинга 2.10, при нажатии на иконку навигации тулбара **android.support.v7.widget.Toolbar**

будет происходить выдвижение навигационного меню **NavigationView** (см. Рис. 2.4).

Далее, когда нажать на пункт меню при открытой (выдвинутой) панели навигации, то панель навигации не исчезнет. Поэтому необходимо самостоятельно писать программный код обработчика события выбора пункта меню в навигационной панели в котором выполнять закрытие панели навигации. Обработчик события выбора пункта меню в панели навигации **NavigationView** должен реализовывать интерфейс **NavigationView.OnNavigationItemSelectedListener**. В этом интерфейсе объявлен всего один метод:

```
boolean onNavigationItemSelected(MenuItem item);
```

Метод принимает ссылку на выбранный пункт меню **android.view.MenuItem** в навигационной панели и возвращает **true** если событие выбора пункта меню обработано или **false** в противном случае. Посмотреть на программный код обработки события выбора пункта меню в панели навигации **NavigationView** в частности и на всю функциональность рассматриваемого примера можно в Листинге 2.11.

Листинг 2.11. Содержимое класса **MainActivity** рассматриваемого в данном разделе примера

```
public class MainActivity extends AppCompatActivity
{
    // ----- Class members -----
    /**

```

```
* Current visible container
*
* Текущий видимый контейнер
*/
private View curView;

/**
 * Reference to DrawerLayout object
 *
 * Ссылка на объект DrawerLayout
 */
private DrawerLayout drawerMain;

/**
 * Reference to NavigationView object
 *
 * Ссылка на объект NavigationView
 */
private NavigationView naviView;

// ----- Class methods -----
@Override
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

// ----- Initialize the object fields -----
// ----- инициализация полей объекта -----
    this.drawerMain = (DrawerLayout) this.
        findViewById(R.id.drawerMain);

    this.naviView = (NavigationView) this.
        findViewById(R.id.naviView);

    this.curView = this.findViewById(
        R.id.llFitness);
}
```

```
// ----- ToolBar -----
Toolbar toolBar = (Toolbar) this.findViewById(
    R.id.toolbar);
this.setSupportActionBar(toolBar);
toolBar.setTitle("Lesson_06 / Module app");
toolBar.setSubtitle("Navigation menu");
toolBar.setTitleTextColor(Color.WHITE);
toolBar.setSubtitleTextColor(Color.WHITE);
toolBar.setNavigationIcon(R.drawable.
    ic_menu_white_24dp);
toolBar.setNavigationOnClickListener(
    new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        MainActivity.this.drawerMain.
            openDrawer(MainActivity.this.
                naviView);
    }
});
// ----- Set the NavigationItemSelectedListener
// ----- observer -----
// ----- Назначение обработчика выбранного
// ----- элемента навигационного меню -----
this.naviView.setNavigationItemSelectedListener(
    new NavigationView.
        OnNavigationItemSelectedListener()
{
    @Override
    public boolean
        onNavigationItemSelected
        (MenuItem item)
    {
        int id = -1;
```

```
        switch (item.getItemId())
        {
            case R.id.
                navigation_item_fintess_center:
                    id = R.id.llFitness;
                    break;

            case R.id.navigation_item_casino:
                id = R.id.llCasino;
                break;

            case R.id.navigation_item_pool:
                id = R.id.llPool;
                break;
        }

        if (id == -1) return false;

// ----- Hide the DrawerLayout -----
// ----- Прячем DrawerLayout -----
        item.setChecked(true);
        MainActivity.this.drawerMain.
        closeDrawers();
// ----- Запуск анимации на исчезновение текущего
// ----- видимого контейнера -----
// ----- Run animation to disappear the current
// ----- visible container -----
        ObjectAnimator animHide =
            (ObjectAnimator) AnimatorInflater.
            loadAnimator(
                MainActivity.this,
                R.animator.hide_animator);
        animHide.setTarget(MainActivity.
            this.curView);
        animHide.start();
// ----- Remember the container that becomes
```

```
// ----- visible -----
// ----- Запоминаем контейнер, который становится
// ----- видимым -----
        MainActivity.this.curView =
            MainActivity.this.findViewById(id);
// ----- Running the animation on the appearance
// ----- of the next container -----
// ----- Запуск анимации на появление следующего
// ----- контейнера -----
        ObjectAnimator animShow =
            (ObjectAnimator) AnimatorInflater.
                loadAnimator(
                    MainActivity.this,
                    R.animator.show_animator);
        animShow.setTarget(MainActivity.
            this.curView);
        animShow.start();
        return true;
    }
}
}
```

Что происходит в нашем примере при выборе пункта навигационного меню? В навигационном меню перечислены категории мест отдыха (*Rest places*), информацию о которых показывает наше приложение пользователю. В нашем приложении есть информация о трех категориях мест отдыха: Фитнес центр (*Fitness Center*), Казино (*Casino*), Бассейн (*Pool*). Поскольку наше приложение является всего лишь примером, то информация об этих местах отдыха представлена в виде так называемых виджетов «заглушек» (см. Рис. 2.5), то есть в виде виджетов с условной информацией.

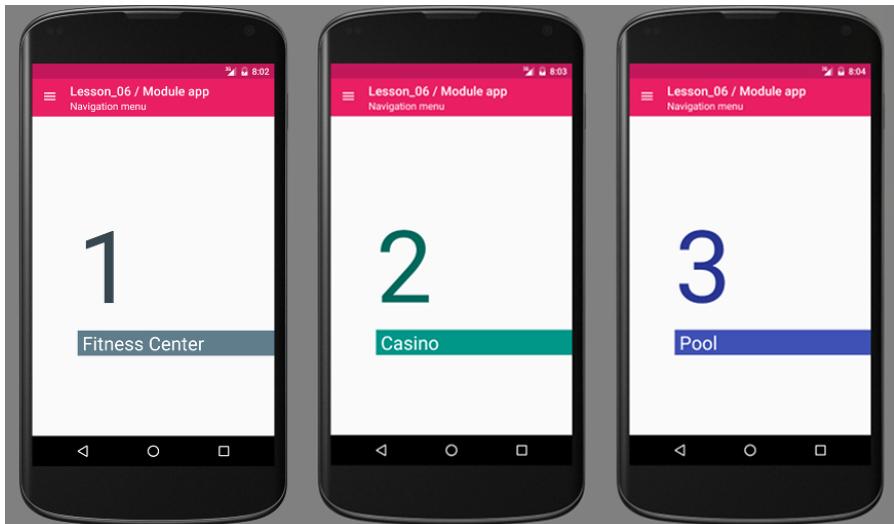


Рис. 2.5. Виджеты с условной информацией между которыми будет переключаться рассматриваемый в данном разделе пример при помощи меню навигации

Причем переключение между виджетами с условной информацией происходит с применением анимации. Для этого в примере созданы два файла ресурсов с анимацией «Property Animation»: `/res/animator/hide_animator.xml` для анимации исчезновения виджета и `/res/animator/show_animator.xml` для анимации появления виджета. Применение анимации при переключении между виджетами с информацией обусловлено следованием концепции Material Design, в которой особое внимание уделяется изменениям, происходящим на экране устройства в нашем приложении (см. предыдущие разделы этого урока которые посвящены Material Design).

Содержимое файлов ресурсов с анимацией «Property Animation» `/res/animator/hide_animator.xml` и `/res/`

animator/show_animator.xml приведены в Листингах 2.12 и 2.13 соответственно.

Листинг 2.12. Содержимое файла ресурса /res/Animator/hide_animator.xml

```
<?xml version="1.0" encoding="utf-8"?>
<objectAnimator
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:duration="300"
    android:valueTo="0"
    android:valueFrom="1"
    android:valueType="floatType"
    android:propertyName="alpha"
    android:repeatCount="0" />
```

Листинг 2.13. Содержимое файла ресурса /res/Animator/show_animator.xml

```
<objectAnimator
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:duration="300"
    android:valueTo="1"
    android:valueFrom="0"
    android:valueType="floatType"
    android:propertyName="alpha"
    android:repeatCount="0"
    android:startOffset="300" />
```

Как видно из Листингов 2.12 и 2.13, исчезновение и появление виджетов осуществляется при помощи изменения свойства прозрачности (**alpha**) виджетов. И теперь пришло время посмотреть, какое содержимое находится

в виджете **android.widget.FrameLayout** (с идентификатором **R.id.flContent**) из Листинга 2.7. В этом виджете находятся три дочерних контейнера **android.widget.LinearLayout** с идентификаторами **R.id.llFitness**, **R.id.llCasino** и **R.id.llPool** соответственно (см. Листинг 2.14).

Листинг 2.14. Дочерние контейнера контейнера R.id.flContent рассматриваемого примера

```
<FrameLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="72dp"
    android:id="@+id/flContent">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:orientation="vertical"
        android:alpha="1"
        android:id="@+id,llFitness">
        ...
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:orientation="vertical"
        android:alpha="0"
        android:id="@+id,llCasino">
        ...
    </LinearLayout>

    <LinearLayout
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:orientation="vertical"
        android:alpha="0"
        android:id="@+id/llPool">
    ...
</LinearLayout>

</FrameLayout>
```

Контейнер с идентификатором **R.id.llFitness** предназначен для отображения информации о фитнесс центрах, контейнер **R.id.llCasino** — о казино, и контейнер **R.id.llPool** — о бассейнах. В Листинге 2.14 не показано содержимое этих контейнеров, так как оно однотипно и поэтому вынесено в отдельный Листинг 2.15.

Листинг 2.15. Содержимое контейнеров R.id.llFitness, R.id.llCasino и R.id.llPool на примере содержимого контейнера R.id.llFitness

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:orientation="vertical"
    android:id="@+id/llFitness">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="1"
        android:textSize="160sp"
```

```
    android:textColor="#37474F" />

<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text=" Fitness Center"
    android:textSize="30sp"
    android:textColor="#FFFFFF"
    android:background="#607D8B" />

</LinearLayout>
```

Так вот (см. Листинг 2.14), два из трех контейнеров имеют значение свойства прозрачности **alpha** равным нулю — то есть не видны на экране. Когда пользователь выбирает пункт меню навигации, текущий видимый виджет при помощи анимации из файла ресурсов */res/animator/hide_animator.xml* (см. Листинг 2.12) плавно исчезает с экрана устройства (см. Рис. 2.6), а виджет, который соответствует выбранному пункту навигационного меню при помощи анимации из файла ресурсов */res/animator/show_animator.xml* (см. Листинг 2.13) плавно появляется на экране устройства после исчезновения предыдущего виджета (см. Рис. 2.7).

Запуск анимаций из Листингов 2.12 и 2.13, а также закрытие навигационного меню можно увидеть в Листинге 2.11 в методе обработчике события выбора пункта навигационного меню **onNavigationItemSelected()**.

Весь исходный код примера из данного раздела можно найти в модуле «app» среди файлов с исходными кодами, которые прилагаются к данному уроку.

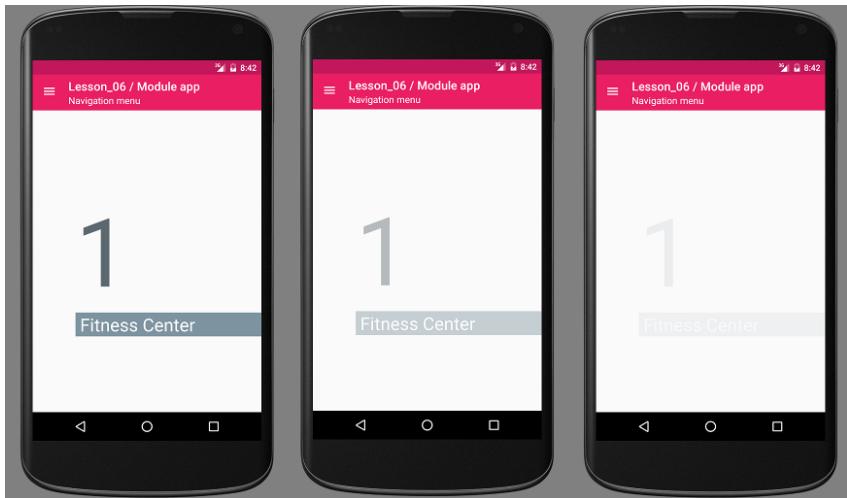


Рис. 2.6. Последовательность скриншотов показывающая исчезновение виджета с условной информацией при помощи анимации «Property Animation» из Листинга 2.12

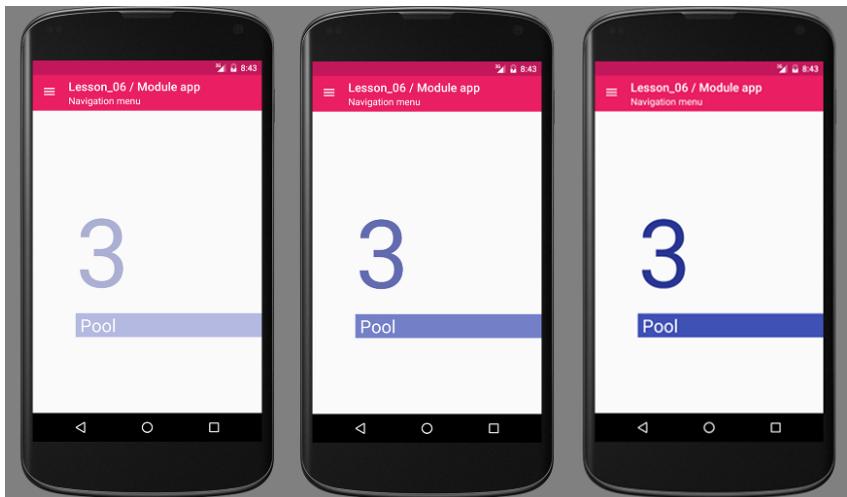


Рис. 2.7. Последовательность скриншотов показывающая появление виджета с условной информацией при помощи анимации «Property Animation» из Листинга 2.13

2.3. Плавающая кнопка. Виджет **FloatingActionButton**

Виджет [**android.support.design.widget.FloatingActionButton**](#) это круглая кнопка, отражающая главное действие в интерфейсе вашего приложения. Плавающей эта кнопка названа потому, что ее положение не фиксировано и может меняться. Причем, это изменение должно быть плавно и осмысленно анимировано. То есть, например, при скроллинге виджета [**android.widget.ListView**](#) или переключении фрагмента кнопка [**android.support.design.widget.FloatingActionButton**](#) может «уезжать» за экран или «растворяться».

Иерархия классов для класса [**android.support.design.widget.FloatingActionButton**](#) выглядит следующим образом:

```
java.lang.Object
|
+-- android.view.View
|
+-- android.widget.ImageView
|
+-- android.widget.ImageButton
|
+-- android.support.design.widget.FloatingActionButton
```

Как видно из иерархии классов, кнопка [**android.support.design.widget.FloatingActionButton**](#) находится в библиотеке Design Support Library, поэтому для ее использования необходимо импортировать эту библиотеку в модуль вашего проекта.

Также отметим, что кнопка `android.support.design.widget.FloatingActionButton` по умолчанию использует цвет `colorAccent` (акцентный цвет) из темы вашего приложения. Кнопка `android.support.design.widget.FloatingActionButton` может иметь два размера: по умолчанию и мини. Размер можно задавать с помощью атрибута `fabSize="mini"`. Внешний вид кнопки `android.support.design.widget.FloatingActionButton` изображен на Рис. 2.8.

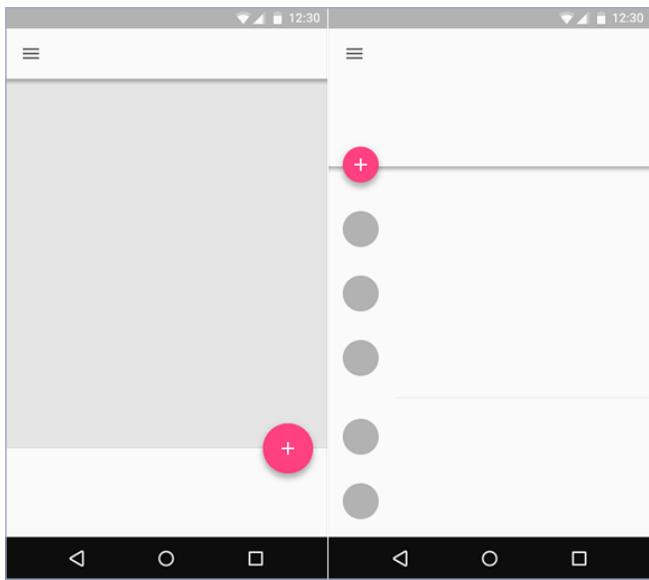


Рис. 2.8. Внешний вид кнопки `android.support.design.widget.FloatingActionButton`. Слева — размер по умолчанию, справа — размер мини

Как уже упоминалось выше, кнопка `android.support.design.widget.FloatingActionButton` должна плавать (то есть находится поверх всех виджетов). Однако, сама по себе эта кнопка плавать не будет — необходимо самосто-

ятельно позаботиться об этом. В Листинге 2.16 показан файл ресурсов /res/layout/activity_main.xml с макетом внешнего вида Активности, в котором находится кнопка **android.support.design.widget.FloatingActionButton**.

Листинг 2.16. Файл макета внешнего вида Активности /res/layout/activity_main.xml с плавающей кнопкой android.support.design.widget.FloatingActionButton

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

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <ListView
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:id="@+id/lvList">
        </ListView>

        <android.support.design.widget.
            FloatingActionButton
            android:id="@+id/fab"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="end|bottom"
            android:layout_margin="16dp"
            android:src="@drawable/ic_grade_white_24dp" />
    </FrameLayout>
</LinearLayout>
```

Как видно из Листинга 2.16, чтобы кнопка была поверх остальных виджетов, был применен контейнер **android.widget.FrameLayout**, в котором если расположить несколько виджетов, то они будут располагаться друг над другом.

В виджет **android.widget.FrameLayout** первым помещен виджет **android.widget.ListView** и последней помещена кнопка **android.support.design.widget.FloatingActionButton**. Также, кнопке при помощи атрибута **android:layout_gravity="end|bottom"** задано расположение внутри контейнера **FrameLayout**. Таким образом кнопка **android.support.design.widget.FloatingActionButton** находится над списком **android.widget.ListView** в правом нижнем углу контейнера **android.widget.FrameLayout**.

Для демонстрации работы текущего примера, заполним список **android.widget.ListView** (идентификатор **R.id.lvList**) некоторыми значениями и назначим кнопке **android.support.design.widget.FloatingActionButton** обработчик события нажатия (см. Листинг 2.17) в методе **onCreate()** класса Активности.

Листинг 2.17. Заполнение списка **android.widget.ListView** значениями и назначение обработчика события нажатия на кнопку **android.support.design.widget.FloatingActionButton**

```
public class MainActivity extends AppCompatActivity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
```

```
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

// ----- ArrayAdapter fot ListView -----
ArrayList<String> arr = new ArrayList<>();
for (int i = 0; i < 150; i++)
{
    arr.add("Hello World " + (i + 1));
}

ArrayAdapter<String> adapter =
    new ArrayAdapter<>(this,
        android.R.layout.
        simple_list_item_1,
        arr);
ListView lvList = (ListView) this.
    findViewById(R.id.lvList);
lvList.setAdapter(adapter);

// -- Handling click event for Floating
// -- Action Button -----
// -- Обработка события нажатия на плавающую
// -- кнопку -----
    findViewById(R.id.fab).setOnClickListener(
        new View.OnClickListener()
    {
        @Override
        public void onClick(View view)
        {
            Toast.makeText(MainActivity.this,
                "Floating Action Button Pressed!",
                Toast.LENGTH_LONG).show();
        }
    });
}
```

На Рис. 2.9 показана работа примера из Листингов 2.16 и 2.17.

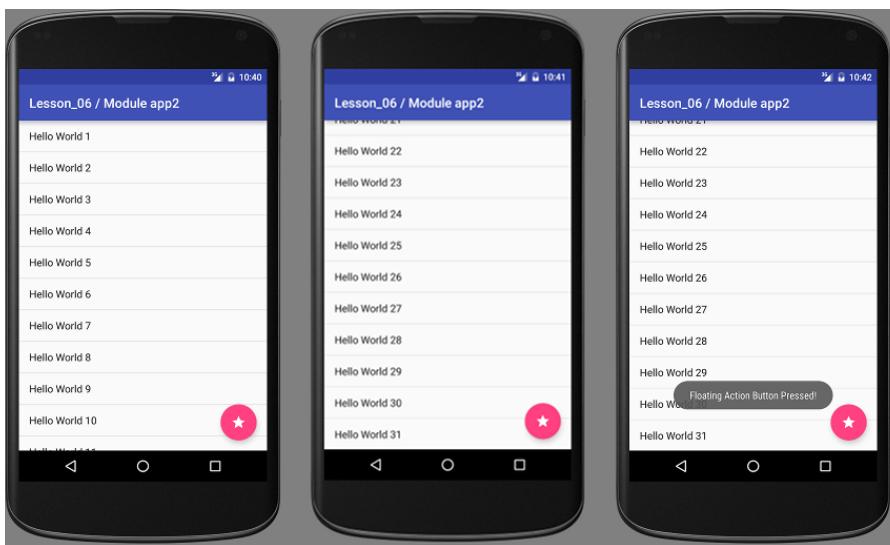


Рис. 2.9. Внешний вид работы Листингов 2.16 и 2.17

Как видно из Рис. 2.9, плавающая кнопка `android.support.design.widget.FloatingActionButton` остается на своем месте при скроллинге списка `android.widget.ListView` (первый и второй скриншоты) и успешно обрабатывает событие нажатия (третий скриншот).

В данном разделе показано самое простое и не самое красивое применение плавающей кнопки `android.support.design.widget.FloatingActionButton`. В следующих разделах данного урока мы еще вернемся к плавающей кнопке. Исходные коды примера из текущего раздела можно найти в модуле «app2» среди файлов с исходными кодами, которые прилагаются к текущему уроку.

2.4. Тост с обратной связью. Виджет Snackbar

Виджет [android.support.design.widget.Snackbar](#) выводит короткое сообщение и позволяет получить обратную связь. Этот виджет можно использовать как виджет [android.widget.Toast](#), однако, существенным отличием виджета [android.support.design.widget.Snackbar](#) от [android.widget.Toast](#) является, как уже упоминалось выше, возможность получать обратную связь от пользователя (например в виде подтверждения или отмены операции). Внешний вид сообщений [android.support.design.widget.Snackbar](#) можно увидеть на Рис. 2.10.

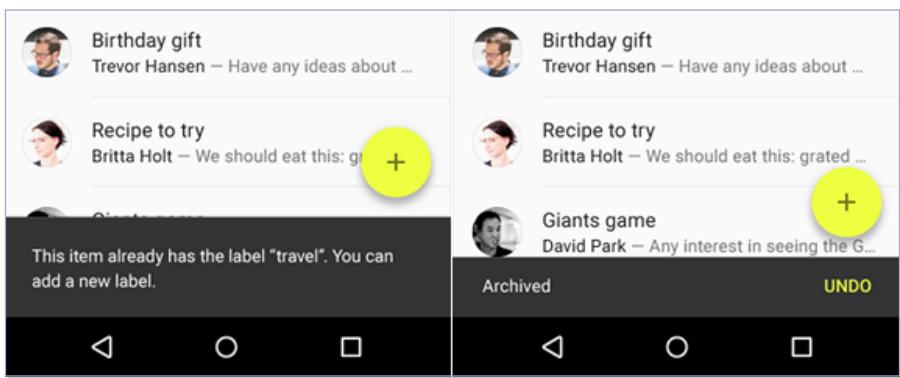


Рис. 2.10. Внешний вид виджета `android.support.design.widget.Snackbar`. Слева — без обратной связи, справа — с обратной связью

Использовать [android.support.design.widget.Snackbar](#) очень легко — практически также, как и класс [android.widget.Toast](#). В Листинге 2.18 показано простейшее использование виджета [android.support.design.widget.Snackbar](#).

Листинг 2.18. Пример применения виджета android.support.design.widget.Snackbar для вывода сообщения «Hello World»

```
Snackbar.make(llMain, "Hello World!",  
    Snackbar.LENGTH_LONG).show();
```

Как видно из Листинга 2.18, показ сообщения в виджете **android.support.design.widget.Snackbar** отличается от показа сообщения в виджете **android.widget.Toast** только названием класса. Однако, в отличии от класса **android.widget.Toast**, метод, создающий объект **android.support.design.widget.Snackbar**:

```
static Snackbar make (View view,  
    CharSequence text, int duration);
```

первым параметром принимает не ссылку на объект Контекста приложения, а ссылку на объект **android.view.View**, который должен ссылаться на любой виджет, который будет использован для поиска родительского (корневого) виджета. Можно в этот параметр сразу передать ссылку на главный (корневой) контейнер Активности. Если главным контейнером Активности является **android.support.design.widget.CoordinatorLayout**, то это позволяет **android.support.design.widget.Snackbar** включать определенные функции, такие как «прокрутка-отмена» («swipe-to-dismiss»), а также автоматическое перемещение виджета **android.support.design.widget.FloatingActionButton**. Менеджер раскладки **android.**

support.design.widget.CoordinatorLayout и его возможности будут рассмотрены ниже в данном уроке.

Если есть необходимость получить ответ пользователя на сообщение, которое выводится объектом **android.support.design.widget.Snackbar**, то программный код будет выглядеть так, как показано в Листинге 2.19.

Листинг 2.19. Виджет **android.support.design.widget.Snackbar** с обратной связью — кнопкой «OK»

```
Snackbar.make(llMain, "Hello World!",
    Snackbar.LENGTH_INDEFINITE).
    setAction("OK", new View.
        OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        ... // Handling event
    }
}).
show();
```

Хотим обратить внимание, что в качестве значений (параметр **duration** метода **make()**), задающих длительность показа сообщения, в классе **android.support.design.widget.Snackbar** используются следующие константы: **LENGTH_SHORT**, **LENGTH_LONG**, **LENGTH_INDEFINITE** (бесконечно). Если задать значение времени показа равным **LENGTH_INDEFINITE**, то сообщение будет отображаться на экране до тех пор пока пользователь не нажмет на кнопку обратной связи или пока не появится следующее сообщение.

Также хотим отметить, что на виджете **android.support.design.widget.Snackbar** можно разместить только одну кнопку обратной связи.

Давайте на примере рассмотрим использование виджета **android.support.design.widget.Snackbar**. Для этого создадим модуль «app3». Файл ресурсов с макетом внешнего вида Активности (/res/layout/activity_main.xml) приведен в Листинге 2.20.

Листинг 2.20. Файл ресурсов с макетом внешнего вида Активности /res/layout/activity_main.xml рассматриваемого примера

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app=
        "http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/l1Main"
    android:orientation="vertical"
    tools:context="itstep.com.myapp3.MainActivity">
    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:background="@color/colorPrimary"
        android:subtitleTextColor="#FFFFFF"
        android:titleTextColor="#FFFFFF" />

    <FrameLayout
        android:layout_width="match_parent"
```

```
    android:layout_height="match_parent">

<TableLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="16dp"
    android:layout_marginRight="16dp"
    android:layout_gravity="center"
    android:stretchColumns="*">

    <TableRow>
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src=
                "@drawable/ic_message_black_24dp" />

        <android.support.v7.widget.AppCompatTextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="17sp"
            android:text=
                "No Feedback Snackbar Example" />
    </TableRow>

    <TableRow>
        <Space />
        <android.support.v7.widget.AppCompatButton
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Show Snackbar"
            android:textColor="#FFFFFF"
            app:backgroundTint="#2196f3"
            android:id="@+id/btnShowOne"
            android:onClick="btnShowSnackbar" />
    </TableRow>
```

```
<TableRow>
    <View
        android:layout_width="match_parent"
        android:layout_span="2"
        android:layout_height="1dp"
        android:layout_marginTop="16dp"
        android:layout_marginBottom="16dp"
        android:background="#B0B0B0" />
    </TableRow>

    <TableRow>
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src=
                "@drawable/ic_message_black_24dp" />

        <android.support.v7.widget.AppCompatTextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="17sp"
            android:text="Feedback Snackbar Example" />
    </TableRow>

    <TableRow>
        <Space />
        <android.support.v7.widget.AppCompatButton
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Show Snackbar"
            android:textColor="#FFFFFF"
            app:backgroundTint="#2196f3"
            android:id="@+id/btnShowTwo"
            android:onClick="btnShowSnackbar" />
    </TableRow>
```

```
<TableRow>
    <View
        android:layout_width="match_parent"
        android:layout_span="2"
        android:layout_height="1dp"
        android:layout_marginTop="16dp"
        android:layout_marginBottom="16dp"
        android:background="#B0B0B0" />
</TableRow>
</TableLayout>

</FrameLayout>
</LinearLayout>
```

Внешний вид Активности из Листинга 2.20 показан на Рис. 2.11 (слева).

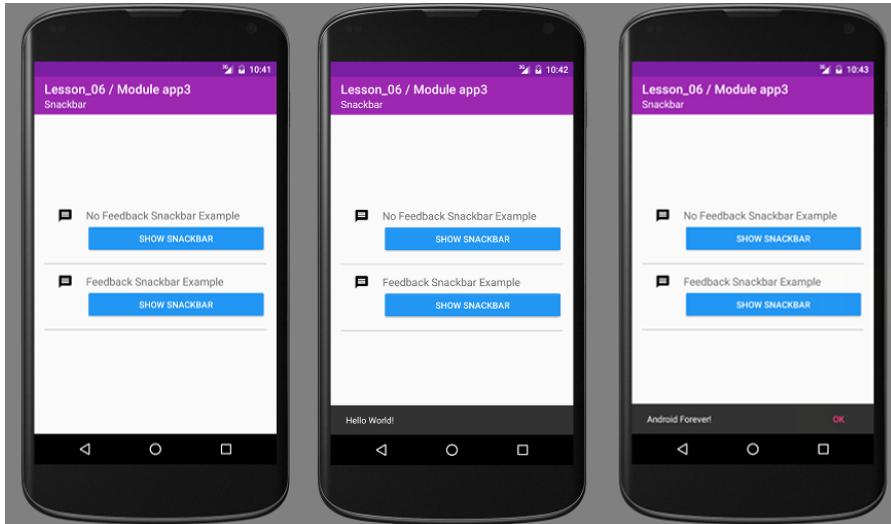


Рис. 2.11. Внешний вид Активности рассматриваемого в данном разделе примера

Как видно из Листинга 2.20 и Рис. 2.11, на Активности размещены две кнопки `android.support.v7.widget.AppCompatButton` с идентификаторами `R.id.btnShowOne` и `R.id.btnShowTwo`. Обоим кнопкам назначен метод обработчик события нажатия `btnShowSnackbar()`. Кнопка с идентификатором `R.id.btnShowOne` предназначена для показа сообщения `android.support.design.widget.Snackbar` без обратной связи, а кнопка с идентификатором `R.id.btnShowTwo` предназначена для показа сообщения с обратной связью. Программный код метода обработчика события нажатия на кнопки `R.id.btnOne` и `R.id.btnTwo` (метод `btnShowSnackbar()`) приведен в Листинге 2.21.

На Рис. 2.11 на скриншоте слева изображена верстка макета Активности из Листинга 2.20. На скриншоте в центре показано сообщение без обратной связи (была нажата кнопка `R.id.btnOne`). На скриншоте справа показано сообщение `android.support.design.widget.Snackbar` с обратной связью, которое появилось в результате нажатия на кнопку с идентификатором `R.id.btnTwo`.

Листинг 2.21. Метод `btnShowSnackbar()` класса `MainActivity`, который обрабатывает события нажатия на кнопки с идентификаторами `R.id.btnOne` и `R.id.btnTwo`

```
public void btnShowSnackbar(View v)
{
    LinearLayout llMain = (LinearLayout) this.
        findViewById(R.id.llMain);

    switch (v.getId())
    {
        case R.id.btnShowOne:
```

```
        {
            Snackbar.make(llMain,
                "Hello World!",
                Snackbar.LENGTH_LONG).show();
        }
        break;

    case R.id.btnShowTwo:
        {
            Snackbar.make(llMain,
                "Android Forever!",
                Snackbar.LENGTH_INDEFINITE).
                setAction("OK", new View.
                    OnClickListerner())
            {
                @Override
                public void onClick(View v)
                {
                    Toast.makeText(
                        MainActivity.this,
                        "OK Pressed",
                        Toast.LENGTH_SHORT).show();
                }
            } .
            show();
        }
        break;
    }
}
```

Как видно из Листинга 2.21, для сообщения с обратной связью, при нажатии на кнопку «OK» будет показан виджет **android.widget.Toast** с сообщением «OK Pressed».

Исходный код примера, рассматриваемого в данном разделе можно найти в модуле «app3» среди файлов с исходными кодами, которые прилагаются к данному уроку.

2.5. Менеджер раскладки CoordinatorLayout

Перед тем, как рассматривать менеджер раскладки `android.support.design.widget.CoordinatorLayout`, давайте вернемся к примеру из модуля «app2» (см. Листинги 2.16 и 2.17). Напомним, что в этом примере мы знакомились с использованием плавающей кнопки `android.support.design.widget.FloatingActionButton`. В методе обработчике события нажатия на плавающую кнопку `onClick()` заметим отображение сообщения при помощи `android.widget.Toast` на отображение сообщения при помощи `android.support.design.widget.Snackbar` (см. Листинг 2.22).

Листинг 2.22. Вывод сообщения при помощи `android.support.design.widget.Snackbar` в примере модуля «app2»

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

    ...

    // -- Handling click event for Floating Action Button
    // -- Обработка события нажатия на плавающую кнопку
    findViewById(R.id.fab).setOnClickListener(new View.
        OnClickListerner())

    {

        @Override
        public void onClick(View view)
        {
            ...
            // Toast.makeText(MainActivity.this,
            // "Floating Action Button Pressed!",
            // Toast.LENGTH_LONG).show();
        }
    }
}
```

```
    LinearLayout llMain = (LinearLayout)
        MainActivity.this.
        findViewById(R.id.llMain);
    Snackbar.make(llMain,
        "Floating Action Button Pressed!",
        Snackbar.LENGTH_LONG).show();
    }
}
}
```

Результат работы примера из Листинга 2.22 изображен на Рис. 2.12.

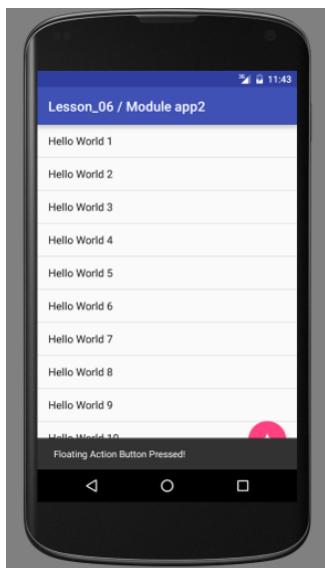


Рис. 2.12. Внешний вид работы примера из Листинга 2.22

Как видим (см. Рис 2.12), виджет **Snackbar** перекрывает плавающую кнопку **FloatingActionButton**. Однако, если мы откроем рекомендации Material Design о [android](#).

[support.design.widget.Snackbar](#), то увидим, что не рекомендуется чтобы **Snackbar** перекрывал плавающую кнопку **FloatingActionButton** (см. Рис. 2.13).

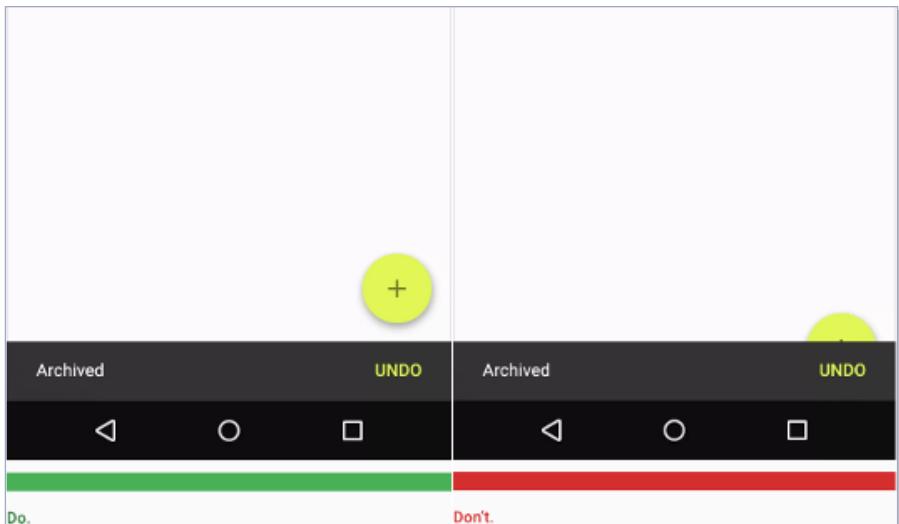


Рис. 2.13. Рекомендации Material Design по использованию Snackbar и FloatingActionButton с сайта <https://material.io/guidelines/components/snackbars-toasts.html#snackbars-toasts-usage>

На Рис. 2.13 слева показано рекомендованное взаимодействие **Snackbar** и **FloatingActionButton**, справа — не рекомендованное.

Можно решить проблему из нашего примера (см. Листинг 2.22 и Рис. 2.12) при помощи самостоятельного решения (что не плохо, но потребует дополнительных затрат времени), а можно решить эту проблему при помощи менеджера [android.support.design.widget.CoordinatorLayout](#), который как раз и предназначен для координации взаимодействия между своими дочерними виджетами.

Менеджер раскладки `android.support.design.widget.CoordinatorLayout` — это супермощный `android.widget.FrameLayout`.

Иерархия классов для класса `android.support.design.widget.CoordinatorLayout` выглядит так:

```
java.lang.Object
  |
  +-- android.view.View
    |
    +-- android.view.ViewGroup
      |
      +-- android.support.design.widget.
          CoordinatorLayout
```

☞ **Внимание!** Для использования менеджера раскладки `android.support.design.widget.CoordinatorLayout` не забудьте добавить библиотеку *Design Support Library* в модуль вашего проекта.

Главное предназначение менеджера раскладки `android.support.design.widget.CoordinatorLayout` заключается в том, чтобы координировать зависимости между включенными в него виджетами. Использование менеджера раскладки `android.support.design.widget.CoordinatorLayout` является и простым и сложным одновременно. Если использовать классы с готовыми решениями, которые предоставляет компания Google, то все просто, но если захочется сделать что-то необычное, то придется приложить определенные усилия. В любом случае, не возможно использовать библиотеку *Design Support Library* не столкнувшись с менеджером

`android.support.design.widget.CoordinatorLayout`. Множество виджетов из библиотеки Design Support Library требуют использования `android.support.design.widget.CoordinatorLayout`.

Если использовать `CoordinatorLayout` с обычными виджетами, входящими в пакет `android.widget`, то он будет работать как обычный `android.widget.FrameLayout`. Вся мощь менеджера `CoordinatorLayout` проявляется при использовании объектов класса `android.support.design.widget.CoordinatorLayout.Behavior`. Подключив `Behavior` (`behavior` — поведение) к виджету, размещенному в `CoordinatorLayout`, можно перехватывать касания, оконные вставки (*window insets*), изменения размеров и раскладки (*measurement* и *layout*), а также вложенную прокрутку (*nested scroll*). Перехватывая эти события можно заставить дочерние виджеты взаимодействовать друг с другом. Классы библиотеки Design Support Library широко использует `Behavior` чтобы добавлять мощь и выразительность большинству функциональности, которую вы можете наблюдать в современных приложениях. И в этом разделе мы сделаем с вами первые и главные шаги к созданию подобных приложений.

Для начала мы справимся с проблемой из Листинга 2.22 (Рис. 2.12). Для этого создадим модуль «app4». В файле ресурсов с макетом раскладки Активности (`/res/layout/activity_main.xml`) заменим главный контейнер на менеджер раскладки `android.support.design.widget.CoordinatorLayout` (см. Листинг 2.23).

Листинг 2.23. Файл /res/layout/activity_main.xml с макетом внешнего вида Активности и главным контейнером android.support.design.widget.CoordinatorLayout

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    android:id="@+id/clMain">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="60dp"
            android:background="@color/colorPrimary" />

        <ListView
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:id="@+id/lvList">
        </ListView>
    </LinearLayout>

    <android.support.design.widget.FloatingActionButton
        android:id="@+id/fab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:layout_gravity="end|bottom"
        android:layout_margin="16dp"
        android:src="@drawable/ic_grade_white_24dp" />
</android.support.design.widget.CoordinatorLayout>
```

Внешний вид Активности из Листинга 2.23 изображен на Рис. 2.14.

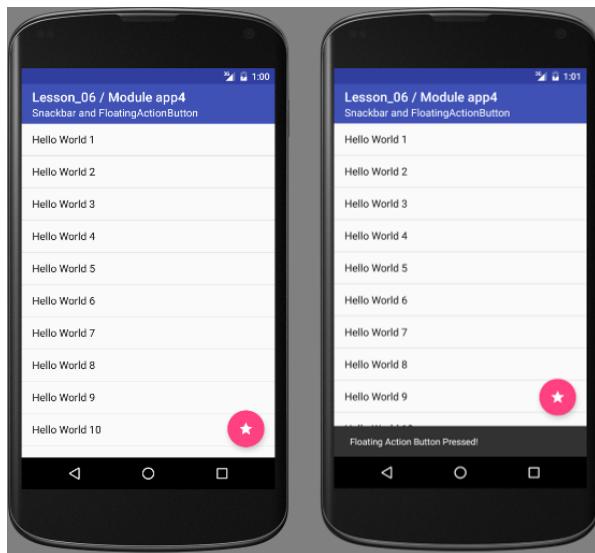


Рис. 2.14. Внешний вид работы примера из модуля «app4»

Программный код Java для класса **MainActivity** совпадает с Листингом 2.17 (заполнение Адаптера Данных для **android.widget.listView**) и Листингом 2.22 (обработка события нажатия на плавающую кнопку) и поэтому в листинги данного урока не включается. Посмотреть исходный код класса **MainActivity** можно в модуле «app4» среди файлов с исходными кодами, прилагаемыми к уроку.

Как видно из Рис. 2.14, проблема перекрытия виджетом **Snackbar** плавающей кнопки **FloatingActionButton** решена путем использования контейнера **android.support.design.widget.CoordinatorLayout**. Другими словами, использование контейнера **CoordinatorLayout** приводит к правильно-му взаимодействию **Snackbar** и **FloatingActionButton**. Это происходит потому, что классы **android.support.design.widget.Snackbar** и **android.support.design.widget.FloatingActionButton** (как и другие классы библиотеки Design Support Library) уже «научены» вести себя правильно (то есть обладают поведением в виде объекта **CoordinatorLayout.Behavior**) в менеджере раскладки **android.support.design.widget.CoordinatorLayout**. У нас также есть возможность быстро «научить правильному поведению» свои виджеты если использовать поведение по умолчанию **android.support.design.widget.DefaultBehavior**. Это поведение можно назначить своим (кастомным) виджетам при помощи аннотации, как показано в Листинге 2.24 на примере объявления класса **android.support.design.widget FloatingActionButton** из библиотеки Design Support Library.

Листинг 2.24. Пример назначения поведения по умолчанию **android.support.design.widget.DefaultBehavior** классу виджета в виде аннотации

```
@CoordinatorLayout.DefaultBehavior  
        (FloatingActionButton.Behavior.class)  
  
public class FloatingActionButton extends ImageButton  
{  
    ...  
}
```

Однако, использование классов из библиотеки Design Support Library или использование поведения по умолчанию или использование других поведений (классы которых находятся в библиотеке Design Support Library) не раскрывает нам сути того, как происходит координация зависимостей между виджетами и, непосредственно, взаимодействие между виджетами. Чтобы увидеть эту суть, давайте вначале познакомимся с методами, которые объявлены в абстрактном классе [android.support.design.widget.CoordinatorLayout.Behavior](#), и с методами, объявленными в классе [android.support.design.widget.CoordinatorLayout](#). А потом перейдем к рассмотрению примера.

Иерархия классов для класса [android.support.design.widget.CoordinatorLayout.Behavior](#) достаточно проста:

```
java.lang.Object
  |
  +-- android.support.design.widget.
      CoordinatorLayout.Behavior<
          V extends android.view.View>
```

Как видно из иерархии классов, класс [android.support.design.widget.CoordinatorLayout.Behavior](#) является обобщенным классом (*generic* классом). Обобщенным типом является тип **V** который должен быть производным от класса [android.view.View](#). Создавая производный от класса [CoordinatorLayout.Behavior](#) класс (то есть создавая новое поведение), разработчик указывает для какого типа виджетов создается это новое поведение. Например, как показано в Листинге 2.25, но-

вый класс поведения создается для применения только к виджетам **android.widget.Button**.

Листинг 2.25. Пример создания нового класса поведения, производного от класса CoordinatorLayout.Behavior

```
class SomeBehavior extends CoordinatorLayout.  
    Behavior<Button>  
{  
    ...  
}
```

Можно еще поступить и другим способом, как показано в Листинге 2.26.

Листинг 2.26. Пример создания нового класса поведения, производного от класса CoordinatorLayout.Behavior

```
class SomeBehavior<V extends View> extends  
    CoordinatorLayout.Behavior<V>  
{  
    ...  
}
```

Как видно из Листинга 2.26, производный класс **SomeBehavior** не уточняет обобщенный тип V, и сам, автоматически, становится обобщенным классом. Класс из Листинга 2.26 можно назначать любым виджетам, производным от класса **android.view.View**.

Итак, интересующие нас методы класса **android.support.design.widget.CoordinatorLayout.Behavior** (все эти методы автоматически вызываются менеджером

раскладки **CoordinatorLayout** для координации между дочерними виджетами):

- **boolean layoutDependsOn(CoordinatorLayout parent, V child, View dependency)** — метод вызывается для того, чтобы сообщить менеджеру **CoordinatorLayout** зависит ли раскладка виджета **child** от раскладки виджета **dependency**. Оба виджета **child** и **dependency** являются дочерними виджетами для менеджера **parent**. Если метод возвращает **true**, то это означает, что виджет **child** зависит (а значит ему требуется корректировать свои параметры раскладки) от виджета **dependency**. В этом случае родительский **CoordinatorLayout (parent)** будет вызывать метод **onDependentViewChanged()** для виджета **child** при изменении раскладки виджета **dependency**. Если метод вернет **false**, то это означает, что виджет **child** не зависит от виджета **dependency**, а значит метод **onDependentViewChanged()** для корректировки параметров раскладки виджета **child** вызывать не нужно.

⌚ **Внимание!** Метод **layoutDependsOn()** вызывается всего один раз перед началом изменений виджета **dependency**. Каждое изменение виджета **dependency** влечет за собой последовательность вызовов метода **onDependentViewChanged()**. И так будет происходить всякий раз при изменениях в **dependency**.

- **boolean onDependentViewChanged(CoordinatorLayout parent, V child, View dependency)** — метод вызыва-

ется менеджером **CoordinatorLayout** при изменении параметров раскладки его дочернего виджета **dependency**. Параметр **parent** ссылается на менеджер **CoordinatorLayout**, который вызвал данный метод. Параметр **child** ссылается на виджет, которому назначен текущий объект Поведения (**CoordinatorLayout.Behavior**). В этом методе необходимо изменить (скоординировать) параметры виджета **child** в зависимости от параметров виджета **dependency**. Метод должен вернуть **true**, если текущий объект **CoordinatorLayout.Behavior** изменил параметры виджета **child**. В противном случае необходимо вернуть **false**.

- **boolean onStartNestedScroll(CoordinatorLayout coordinatorLayout, V child, View directTargetChild, View target, int axes, int type)** — метод вызывается менеджером **CoordinatorLayout** при старте вложенного скроллинга в одном из дочерних виджетов. Параметр **coordinatorLayout** ссылается на родительский контейнер **CoordinatorLayout**, который и вызвал текущий метод. Параметр **child** ссылается на виджет, с которым ассоциирован текущий объект поведения **CoordinatorLayout.Behavior**. Параметр **directTargetChild** ссылается на виджет, который инициирует прокрутку. Параметр **target** ссылается на виджет, который содержит виджет **directTargetChild**. Параметр **axes** содержит информацию об осях, в направлении которых осуществляется прокрутка. Значения для этого параметра задаются в виде констант **android.support.**

v4.view.ViewCompat.SCROLL_AXIS_HORIZONTAL и android.support.v4.view.ViewCompat.SCROLL_AXIS_VERTICAL. Параметр **type** содержит тип ввода, который вызвал это событие прокрутки. К сожалению, на текущий момент, компания Google не дает расширенной информации по значениям параметра **type**. Метод должен вернуть значение **true**, если виджет **child** заинтересован в получении информации о прокрутке в виджете **target**. В этом случае, при событиях прокрутки, будет вызываться метод **onNestedScroll()**. Если виджет **child** не заинтересован в получении событий о прокрутке в виджете **target**, то метод должен вернуть значение **false**.

- **void onNestedScroll(CoordinatorLayout coordinatorLayout, V child, View target, int dxConsumed, int dyConsumed, int dxUnconsumed, int dyUnconsumed, int type)** — Метод вызывается контейнером **coordinatorLayout** для объекта **Behavior** с которым ассоциирован виджет **child** при событиях прокрутки в виджете **target**. Параметры **dxConsumed** и **dyConsumed** содержат величину выполняемой прокрутки (в пикселях) по горизонтали и по вертикали которую инициировал пользователь в виджете **target**. Параметры **dxUnconsumed** и **dyUnconsumed** содержат величину прокрутки (в пикселях) которую инициирует пользователь, но которая не может быть выполнена из-за того, что достигнута или начальная или конечная граница скроллинга и дальше скроллинговать некуда. Параметры **dxConsumed** и **dyConsumed** могут быть

как больше, так и меньше нуля, в зависимости от направления прокрутки.

- `void onStopNestedScroll(CoordinatorLayout coordinatorLayout, V child, View target, int type)` — Метод вызывается контейнером `coordinatorLayout` для объекта `Behavior` с которым ассоциирован виджет `child` тогда, когда вложенная прокрутка останавливается. Параметр `target` ссылается на виджет в котором происходила прокрутка. Описание параметра `type` см. в описании метода `boolean onStartNestedScroll()`.

Мы перечислили далеко не все методы, которые объявлены в классе [android.support.design.widget.CoordinatorLayout.Behavior](#). В полном описании класса можно еще найти методы, которые будут вызываться для объекта `Behavior` в случае событий касания (*touch event*), оконных вставок (*window insets*), изменения размеров (*measurement*) и событий быстрого касания (*fling*). Мы рассмотрели только те методы, которые пригодятся нам при рассмотрении примеров, раскрывающих концепцию работы менеджера `CoordinatorLayout`. В этом контексте знакомства с `CoordinatorLayout` нам необходимо также познакомиться с некоторыми его методами:

- `boolean onStartNestedScroll(View child, View target, int nestedScrollAxes)` — Метод вызывается дочерним виджетом для того, чтобы сообщить родительскому контейнеру `CoordinatorLayout` о том, что в дочернем виджете начинается прокрутка содержимого. Параметр `child` ссылается на виджет (являющийся дочерним для `CoordinatorLayout`) в котором содер-

жится виджет **target**. Параметр **target** ссылается на виджет, который инициирует события вложенной прокрутки. Параметр **nestedScrollAxes** содержит информацию о направлении прокрутки по осям X и Y в виде значений **SCROLL_AXIS_HORIZONTAL**, **SCROLL_AXIS_VERTICAL**. Метод возвращает true, если родительский контейнер **CoordinatorLayout** принял операцию вложенной прокрутки.

- **void onNestedScroll(View target, int dxConsumed, int dyConsumed, int dxUnconsumed, int dyUnconsumed)** — Метод вызывается дочерним виджетом для того, чтобы сообщить родительскому контейнеру **CoordinatorLayout** о том, что в дочернем виджете **target** происходит прокрутка содержимого. Параметр **target** ссылается на виджет, в котором происходит прокрутка. Параметры **dxConsumed** и **dyConsumed** содержат величину прокрутки (в пикселях) по горизонтали и по вертикали на которую осуществляется прокрутка в виджете **target**. Параметры **dxUnconsumed** и **dyUnconsumed** содержат величину прокрутки (в пикселях) которую пользователь инициирует в виджете **target** но которая не может быть выполнена потому что достигнута или начальная или конечная граница скроллинга. Параметры **dxConsumed** и **dyConsumed** могут быть как больше, так и меньше нуля, в зависимости от направления прокрутки.
- **void onStopNestedScroll(View target)** — Метод вызывается дочерним виджетом для того, чтобы сообщить родительскому контейнеру **CoordinatorLayout**

о том, что в дочернем виджете `target` завершилась прокрутка содержимого.

Мы перечислили далеко не все методы, которые объявлены в классе `android.support.design.widget.CoordinatorLayout`. Со всеми методами класса [android.support.design.widget.CoordinatorLayout](#) можно познакомиться на сайте для разработчиков по адресу.

Как уже говорилось, контейнер `CoordinatorLayout` никогда сам не узнает о том, что в его дочернем виджете что-то изменяется, до тех пор, пока этот дочерний виджет сам ему об этом не сообщит. Получив сообщение (например при помощи вызова метода `onStartNestedScroll()`), менеджер `CoordinatorLayout` перебирает все свои дочерние виджеты и при наличии у дочернего виджета поведения (объекта `CoordinatorLayout.Behavior`) вызывает соответствующий событию метод в объекте поведения (в нашем примере — `onStartNestedScroll()`). Реализация метода `onStartNestedScroll()` в классе `android.support.design.widget.CoordinatorLayout` показана в Листинге 2.27.

Листинг 2.27. Реализация метода `onStartNestedScroll()` в классе `android.support.design.widget.CoordinatorLayout`

```
public boolean onStartNestedScroll(View child,
    View target, int nestedScrollAxes)
{
    boolean handled = false;
    final int childCount = getChildCount();

    for (int i = 0; i < childCount; i++)
    {
```

```
final View view = getChildAt(i);
final LayoutParams lp = (LayoutParams)
    view.getLayoutParams();
final Behavior viewBehavior =
    lp.getBehavior();

if (viewBehavior != null)
{
    final boolean accepted =
        viewBehavior.onStartNestedScroll(
            this, view, child, target,
            nestedScrollAxes);
    handled |= accepted;
    lp.acceptNestedScroll(accepted);
}
else
{
    lp.acceptNestedScroll(false);
}

}

return handled;
}
```

Как видно из Листинга 2.27, объект **android.support.design.widget.CoordinatorLayout** получив сообщение от дочернего виджета, распространяет это сообщение всем своим дочерним виджетам которым назначен объект **CoordinatorLayout.Behavior**. Этот факт мы и будем использовать в нашем следующем примере.

Суть примера, который мы рассмотрим, заключается в том, что мы будем координировать величину прокрутки содержимого виджета **android.widget.ListView**.

и ширину виджета `android.view.View`. Причем, чем на большее количество элементов будет прокручен список `android.widget.ListView`, тем шире будет виджет `android.view.View`, и наоборот. Получится, что виджет `android.view.View` в рассматриваемом примере будет играть роль прогресс бара.

Для рассматриваемого примера создадим модуль «app5». Файл ресурсов (/res/layout/activity_main.xml) с макетом внешнего вида Активности примера из модуля «app5» приведен в Листинге 2.28.

Листинг 2.28. Файл /res/layout/activity_main.xml с макетом Активности примера из модуля «app5»

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

<android.support.design.widget.CoordinatorLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"

    android:layout_width="match_parent"
    android:layout_height="match_parent"

    android:fitsSystemWindows="true"
    android:id="@+id/clMain"
    tools:context="itstep.com.myapp5.MainActivity">

    <ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/lvList">
    </ListView>
```

```
<View  
    android:layout_width="100dp"  
    android:layout_height="30dp"  
    android:background="#80FF0000"  
    android:layout_margin="16dp"  
    android:layout_gravity="bottom|left"  
    app:layout_behavior=".ProgressBehavior"  
/>  
  
</android.support.design.widget.CoordinatorLayout>
```

Как видно из Листинга 2.28, корневым элементом макета Активности является контейнер **android.support.design.widget.CoordinatorLayout**. Внутри него находятся два вида: виджет **android.widget.ListView** (с идентификатором **R.id.lvList**) который будет использоваться для прокрутки содержимого и виджет **android.view.View** который будет использоваться в качестве прогресс бара, показывающего процесс прокрутки списка **android.widget.ListView**. В Листинге 2.28, жирным шрифтом выделен код, в котором происходит назначение поведения (**CoordinatorLayout.Behavior**) виджету **android.view.View**: **app:layout_behavior=".ProgressBehavior"**, где **ProgressBehavior** является классом, производным от класса **android.support.design.widget.CoordinatorLayout.Behavior**. Содержимое класса **ProgressBehavior** приведено в Листинге 2.29. В Листинге 2.28 имя класса **ProgressBehavior** начинается с точки потому что этот класс находится в том же пакете, что и класс Активности **MainActivity**.

Внешний вид Активности из Листинга 2.28 показан на Рис. 2.15.

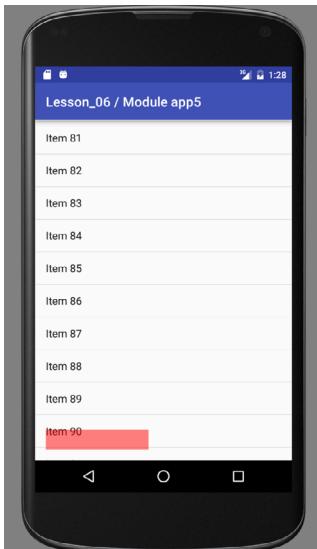


Рис. 2.15. Внешний вид макета Активности из Листинга 2.28

Как видно из Рис. 2.15, виджет `android.view.View` находится над списком `android.widget.ListView` в левом нижнем углу (что соответствует значению атрибута `android:layout_gravity="bottom|left"`). Поскольку контейнер `CoordinatorLayout` является аналогом `android.widget.FrameLayout`, то он располагает свои дочерние виджеты друг над другом, что мы и наблюдаем на Рис. 2.15. Виджет `android.view.View` имеет красный, полупрозрачный, цвет фона. Напомним, что при увеличении количества прокрученных элементов в списке `android.widget.ListView`, ширина виджета `android.view.View` будет увеличиваться. Итак, давайте теперь рассмотрим Листинг 2.29, в котором приведен программный код класса `ProgressBehavior`.

Листинг 2.29. Программный код класса ProgressBehavior рассматриваемого примера

```
/*
 * Class ProgressBehavior implements progress bar
 * behavior for scrolling items in android.widget.
 * ListView
 *
 * Class ProgressBehavior реализует поведение
 * Прогрессбара для прокрутки элементов в списке
 * android.widget.ListView
 */
class ProgressBehavior<V extends View> extends
    CoordinatorLayout.Behavior<V>
{
    // ----- Class constants -----
    private final static String TAG =
        "===== ProgressBehavior";
    // ----- Class members -----
    /**
     * The index of the first visible item in the
     * android.widget.ListView
     *
     * Индекс первого видимого элемента в списке
     * android.widget.ListView
     */
    private int firstVisibleItem = 0;

    /**
     * Total items in the android.widget.ListView
     *
     * Всего элементов в списке android.widget.ListView
     */
    private int totalItems = 0;
```

```
// ----- Class methods -----  
  
public ProgressBehavior()  
{  
}  
  
/**  
 * Constructor for creating a ProgressBehavior  
 * instance through xml.  
 * Конструктор для создания экземпляра  
 * ProgressBehavior через разметку.  
 */  
  
public ProgressBehavior(Context context,  
                      AttributeSet attrs)  
{  
    super(context, attrs);  
}  
  
@Override  
public boolean layoutDependsOn(  
        CoordinatorLayout parent,  
        V child, View dependency)  
{  
    Log.d(TAG, "layoutDependsOn." + " child: " +  
        child.getClass().getName() + ", dependency: " +  
        dependency.getClass().getName() +  
        ", returns: " +  
        (dependency instanceof ListView));  
  
    return dependency instanceof ListView;  
}  
  
@Override  
public boolean onDependentViewChanged(  
        CoordinatorLayout parent,  
        V child, View dependency)
```

```
{  
    Log.d(TAG, "onDependentViewChanged." +  
        " child: " + child.getClass().getName() +  
        ", dependency: " +  
        dependency.getClass().getName());  
  
    ListView lv = (ListView) dependency;  
    this.totalItems = lv.getAdapter().getCount();  
    this.firstVisibleItem =  
        lv.getFirstVisiblePosition();  
  
    CoordinatorLayout.LayoutParams LP =  
        (CoordinatorLayout.LayoutParams) child.  
        getLayoutParams();  
  
    LP.width = this.calculateWidth(  
        this.firstVisibleItem, this.totalItems,  
        parent.getWidth());  
    child.setLayoutParams(LP);  
  
    return true;  
}  
  
@Override  
public boolean onStartNestedScroll(  
    CoordinatorLayout coordinatorLayout,  
    V child,  
    View directTargetChild,  
    View target,  
    int axes/*, int type*/) {  
    Log.d(TAG, "onStartNestedScroll." +  
        " child: " + child.getClass().getName() +  
        ", target: " +  
        directTargetChild.getClass().getName() +  
        ", returns: " +  
        (directTargetChild instanceof ListView));  
}
```

```
        return directTargetChild instanceof ListView;
    }

@Override
public void onNestedScroll(
        CoordinatorLayout coordinatorLayout,
        V child,
        View target,
        int dxConsumed,
        int dyConsumed,
        int dxUnconsumed,
        int dyUnconsumed/*, int type*/)
{
    Log.d(TAG, "onNestedScroll." + " target: " +
            target.getClass().getName() +
            ", dyConsumed: " + dyConsumed);

    CoordinatorLayout.LayoutParams LP =
            (CoordinatorLayout.LayoutParams) child.
            getLayoutParams();

    this.firstVisibleItem += dyConsumed;
    LP.width = this.calculateWidth(this.
            firstVisibleItem,
            this.totalItems, coordinatorLayout.
            getWidth());
    child.setLayoutParams(LP);
}

/**
 * An auxiliary function for calculating
 * the width of the progress bar widget.
 * @param curValue - the current number of scrolled
 * items in the ListView
 * @param maxValue - the total number of items
 * in the ListView
 * @param widgetWidth - width of the parent
 * container
*/
```

```
* @return - width of the progress bar relative  
* to the width of the parent container  
*  
* Вспомогательная функция для расчета ширина  
* виджета прогресс бара.  
*  
* @param curValue - текущее количество прокрученных  
* элементов в списке ListView  
* @param maxValue - общее количество элементов  
* в списке ListView  
* @param widgetWidth - ширина родительского  
* контейнера  
* @return - ширина прогресс бара относительно  
* ширины родительского контейнера  
*/  
  
private int calculateWidth(int curValue,  
                           int maxValue, int widgetWidth)  
{  
    int width = (curValue * widgetWidth) / maxValue;  
    if (width < 4) width = 4;  
    return width;  
}  
}
```

В Листинге 2.29 показан код класса **ProgressBehavior**. Давайте рассмотрим его подробнее:

- Метод **layoutDependsOn()** переопределен в классе **ProgressBehavior** для того, чтобы сообщить менеджеру **CoordinatorLayout** что раскладка виджета, которому назначено данное поведение (**CoordinatorLayout.Behavior**), зависит от раскладки виджета **android.widget.ListView**, потому что, ког-

да будет происходить раскладка виджета `android.widget.ListView`, нашему прогрессбару необходимо изменить свою ширину таким образом, чтобы соответствовать количеству прокрученных элементов в списке `android.widget.ListView`. Поэтому, если параметр `dependency` ссылается на `android.widget.ListView`, то метод `layoutDependsOn()` возвращает `true`, и следовательно, последуют вызовы метода `onDependentViewChanged()`.

- Метод `onDependentViewChanged()` вызывается чтобы рассчитать ширину прогрессбара во время раскладки виджета `android.widget.ListView`. Ширина назначается прогрессбару (виджету `android.view.View`) через объект `LayoutParams`. Для расчета величины ширины используется вспомогательный метод `calculateWidth()`, который объявлен в этом же классе (см. Листинг 2.29).
- Метод `onStartNestedScroll()` в классе `ProgressBehavior` переопределен для того, чтобы сообщать менеджеру `CoordinatorLayout` о том, что виджет, которому назначен данный объект поведения (`ProgressBehavior`) зависит от состояния прокрутки в виджете `directTargetChild`. В нашем примере `directTargetChild` должен быть объектом `android.widget.ListView`. И если это так, то метод `onStartNestedScroll()` возвращает `true`. Как следствие, для данного объекта поведения `ProgressBehavior` последует череда вызовов метода `onNestedScroll()`, так как метод `onStartNestedScroll()` вызывается при событии начала прокрутки.

- В методе `onNestedScroll()` происходит перерасчет ширины прогрессбара (виджета `android.view.View`) в зависимости от текущего количества прокрученных элементов в списке `android.widget.ListView`.

☞ **Внимание!** Для данного примера авторы урока, используют в качестве величины значения прокрутки списка `android.widget.ListView` не пиксели, а количество элементов. Это сделано с целью упрощения рассматриваемого примера. Поэтому в нашем примере, метод `onNestedScroll()` в параметре `dyConsumed` принимает количество элементов в списке `android.widget.ListView`, на которое осуществляется прокрутка.

☞ **Примечание:** В Листинге 2.29 в методах `onStartNestedScroll()` и `onNestedScroll()` последний параметр (`type`) закомментирован. Дело в том, что этот параметр добавлен в эти методы начиная с версия API 26.0.1. В более ранних версиях API этого параметра не было. Поэтому, если у вас версия API 26.0.1 и старше, то вам придется этот параметр раскомментировать. На работу примера наличие или отсутствие этого параметра никак не влияет.

Также обратите внимание, что в Листинге 2.29 во все методы класса `ProgressBehavior` добавлен вывод в логи приложения при помощи метода `Log.d()`. Когда будете запускать рассматриваемый пример, вывод в логи при-

ложения поможет вам более детально увидеть последовательность событий, происходящих при координации зависимостей между виджетами в контейнере `android.support.design.widget.CoordinatorLayout`.

Теперь рассмотрим, как необходимо работать списку `android.widget.ListView`, чтобы менеджер `CoordinatorLayout` мог выполнять свои задачи по координации между своими дочерними виджетами.

Как уже говорилось выше, виджет, который фиксирует свои изменения, должен сообщить об этом менеджеру `CoordinatorLayout`, который, в свою очередь, разошлет всем своим дочерним виджетам информацию о произошедшем событии (как показано в Листинге 2.27). В нашем рассматриваемом примере, список `android.widget.ListView` будет фиксировать события прокрутки своих элементов. Следовательно, ему необходимо назначить обработчик события скроллинга `AbsListView.OnScrollListener` при помощи метода `setOnScrollListener()`.

В этом обработчике события, обо всех изменениях в состоянии скроллинга (начало скроллинга, сам скроллинг и завершение скроллинга) списка `android.widget.ListView` необходимо сообщать менеджеру `CoordinatorLayout`. Описанная функциональность реализована в обработчике событий прокрутки (`AbsListView.OnScrollListener`) который назначается виджету `android.widget.ListView` в методе `onCreate()` класса `MainActivity`. Программный код класса `MainActivity` рассматриваемого примера приведен в Листинге 2.30.

Листинг 2.30. Программный код класса MainActivity

```
public class MainActivity extends AppCompatActivity
{
    // ----- Class constants -----
    private final static String TAG =
        "===== MainActivity";

    // ----- Class members -----
    private ListView lvList;

    // ----- Class methods -----
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // ----- ListView lvList -----
        this.lvList = (ListView)
            this.findViewById(R.id.lvList);

        // ----- Creating ArrayAdapter for ListView -----
        // ----- Создание Адаптера Данных ArrayAdapter<> -
        ArrayList<String> arr = new ArrayList<>();
        for (int i = 0; i < 200; i++)
        {
            arr.add("Item " + i);
        }

        ArrayAdapter<String> adapter = new ArrayAdapter<>(
            this, android.R.layout.simple_list_item_1, arr);
        this.lvList.setAdapter(adapter);

        // ----- Set the scrolling event handler for ListView
        // ----- Назначение обработчика событий скроллинга
        // ----- для ListView -----
    }
}
```

```
if (this.lvList.getParent() instanceof
    android.support.design.widget.
    CoordinatorLayout)
{
    final CoordinatorLayout CL =
        (CoordinatorLayout) this.lvList.
        getParent();
    this.lvList.setOnScrollListener(
        new AbsListView.OnScrollListener()
    {
        /**
         * The index of the item in
         * the android.widget.ListView
         * which was the first visible element before
         * the current scrolling event.
         *
         * Индекс элемента в списке
         * android.widget.ListView
         * который был первым видимым элементом
         * до текущего события прокрутки.
        */
        private int prevFirstVisible = 0;

        @Override
        public void onScrollStateChanged(AbsListView
            absListView, int i)
        {
            switch (i)
            {

// ----- Start scrolling -----
                case 1:
                    Log.d(TAG,
                        "onScrollStateChanged: " + "Start");
                    CL.onStartNestedScroll(
                        MainActivity.this.lvList, CL, 0);
                    break;
            }
        }
    });
}
```

```
// ----- Stop scrolling -----
        case 0:
            Log.d(TAG, "onScrollStateChanged:" +
                    + "Stop");
            CL.onStopNestedScroll(CL);
            break;
        }
    }

@Override
public void onScroll(AbsListView absListView,
                     int i, int i1, int i2)
{
    Log.d(TAG, "onScroll: scroll items: " +
           i + ", all items: " + i2);
    CL.onNestedScroll(MainActivity.this.
                      lvList, 0, (i - this.
                      prevFirstVisible), 0, 0);
    this.prevFirstVisible = i;
}
});
```

Как видно из Листинга 2.30, обработчик событий прокрутки `AbsListView.OnScrollListener` назначается списку `android.widget.ListView` (который объявлен в виде поля объекта с именем `lvList`) в методе `onCreate()`. В объекте `AbsListView.OnScrollListener` находятся два метода обратного вызова: метод `onScrollStateChanged()`, который вызывается при изменении состояния прокрутки, и метод `onScroll()`, который вызывается во время прокрутки.

В методе **onScrollStateChanged()** при событии начала прокрутки списка **lvList** происходит оповещение менеджера **CoordinatorLayout** о событии начала прокрутки при помощи вызова его метода **onStartNestedScroll()**. Этот фрагмент кода выделен жирным шрифтом в Листинге 2.30.

В процессе самой прокрутки списка **lvList** происходит последовательность вызовов метода **onScroll()**. Метод **onScroll()** принимает кроме ссылки на источник события **android.widget.ListView** (параметр **absListView**) следующие параметры: **i** — количество прокрученных элементов, **i1** — количество видимых элементов на экране, **i2** — общее количество элементов в списке. В методе **onScroll()** менеджеру **CoordinatorLayout** сообщается о событии прокрутки при помощи вызова метода **onNestedScroll()**. В Листинге 2.30 этот фрагмент кода выделен жирным шрифтом. Обращаем ваше внимание, что методу **onNestedScroll()** в качестве параметра **dyConsumed** передается не величина прокрутки в пикселях, а текущее количество прокрученных элементов **lvList**. Это сделано авторами урока умышленно в целях упрощения рассматриваемого примера, тем более, что такое отклонение не влияет на результат рассматриваемого примера, цель которого — показать, каким образом менеджер раскладки **android.support.design.widget.CoordinatorLayout** координирует зависимости своих дочерних виджетов друг от друга.

Ну и наконец, когда происходит событие остановки прокрутки списка **lvList**, вызывается метод **onScrollStateChanged()**, в котором менеджеру **CoordinatorLayout** сообщается о завершении прокрутки при помощи вызова

его метода `onStopNestedScroll()`). Этот фрагмент кода выделен жирным шрифтом в Листинге 2.30.

Таким образом, при помощи методов `onStartNestedScroll()`, `onNestedScroll()` и `onStopNestedScroll()`, менеджер раскладки `CoordinatorLayout` узнает об изменениях в одном из своих дочерних виджетах. Ну а затем, менеджер `CoordinatorLayout` передает эту информацию всем остальным своим дочерним виджетам (пример, как это делается, показан в Листинге 2.27).

Внешний вид работы примера из Листингов 2.28, 2.29, 2.30, показан на Рис. 2.16.

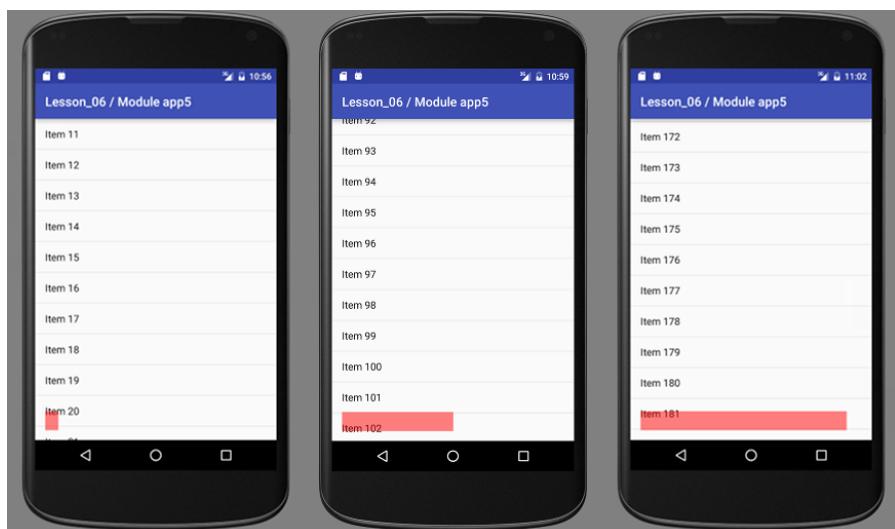


Рис. 2.16. Внешний вид работы примера из Листингов 2.28, 2.29, 2.30

Как видно из Рис. 2.16, при увеличении количества прокрученных элементов в списке `android.widget.ListView` (`lvList`) ширина прогрессбара увеличивается,

что является результатом координации зависимостей которую осуществляет менеджер `android.support.design.widget.CoordinatorLayout`.

Рассмотрим еще несколько штрихов для текущего примера. Созданный класс поведения `ProgressBehavior` можно назначать нескольким виджетам одновременно. Давайте добавим в контейнер `CoordinatorLayout` (идентификатор `clMain`) еще один виджет `android.view.View`, которому назначим это же поведение `ProgressBehavior`. Программный код файла макета Активности `/res/layout/activity_main.xml` в который добавлен виджет, приведен в Листинге 2.31.

Листинг 2.31. Файл макета Активности `/res/layout/activity_main.xml` с еще одним виджетом, которому назначено поведение `ProgressBehavior`

```
<android.support.design.widget.CoordinatorLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    android:id="@+id/clMain"
    tools:context="itstep.com.myapp5.MainActivity">

    ...

    <View
        android:layout_width="100dp"
        android:layout_height="30dp"
        android:background="#8000FF00"
        android:layout_marginLeft="16dp"
        android:layout_marginBottom="64dp"
```

```
        android:layout_gravity="bottom|left"
        app:layout_behavior=".ProgressBehavior" />
</android.support.design.widget.CoordinatorLayout>
```

К добавленному виджету `android.view.View` (см. Листинг 2.31 фрагмент кода, выделенный жирным шрифтом) начнет применяться поведение объекта `ProgressBehavior` и этот виджет будет вести себя как прогрессбар (см. Рис. 2.17).

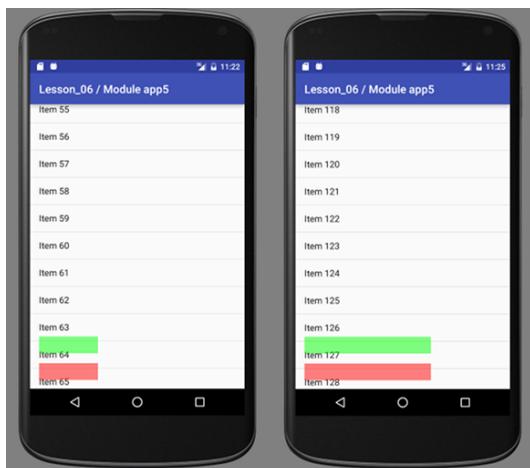


Рис. 2.17. Два разных виджета `android.view.View` с одинаковым поведением `ProgressBehavior`

Как видно из Рис. 2.17, однажды написанное поведение (производный от `CoordinatorLayout.Behavior` класс) можно многократно применять для разных виджетов внутри менеджера `CoordinatorLayout`.

В рассмотренном выше примере из Листингов 2.28, 2.29, 2.30, класс поведения `ProgressBehavior` назначал-

ся виджету `android.view.View` (см. Листинг 2.28) при помощи xml атрибута `app:layout_behavior=".CoordinatorLayout.Behavior"`. Однако, это не единственный способ назначения поведения `CoordinatorLayout.Behavior` какому-либо виджету, находящемуся внутри `CoordinatorLayout`. Кроме назначения при помощи xml есть еще два способа: программно и автоматически при помощи аннотации `@CoordinatorLayout.DefaultBehavior()`.

Давайте рассмотрим программное назначение объекта `CoordinatorLayout.Behavior`. Объект `CoordinatorLayout.Behavior` хранится в `LayoutParams` каждого виджета, который размещен в контейнере `CoordinatorLayout`. Это объясняет, почему `CoordinatorLayout.Behavior` должен быть объявлен у виджетов которые находятся внутри `CoordinatorLayout`, так как только у этих виджетов есть конкретный подтип `CoordinatorLayout.LayoutParams`, который может хранить `CoordinatorLayout.Behavior`. Поэтому, для программного назначения виджету поведения `CoordinatorLayout.Behavior` необходимо выполнить код, пример которого приведен в Листинге 2.32.

Листинг 2.32. Пример программного назначения виджету объекта поведения, производного от `CoordinatorLayout.Behavior`

```
ProgressBehavior progressBehavior =
        new ProgressBehavior();
CoordinatorLayout.LayoutParams params =
        (CoordinatorLayout.LayoutParams)
        progressView.getLayoutParams();
params.setBehavior(progressBehavior);
```

Что касается автоматического назначения поведения при помощи аннотации `@CoordinatorLayout.DefaultBehavior()`, то этот способ используется в случае, когда создается свой (кастомный) виджет. И при этом необходимо назначать поведение этому виджету, которое будет поведением по умолчанию, без необходимости назначать это поведение вручную при помощи xml или программно. Пример назначения поведения при помощи аннотации `@CoordinatorLayout.DefaultBehavior()` приведен в Листинге 2.33.

Листинг 2.33. Пример автоматического назначения поведения виджету при помощи аннотации `@CoordinatorLayout.DefaultBehavior()`

```
@CoordinatorLayout.DefaultBehavior(
    ProgressBehavior.class)
public class MyCustomFrameLayout extends FrameLayout
{}
```

Также, мы можем создавать столько классов поведений, сколько нам необходимо. Давайте создадим еще одно поведение в нашем примере — класс `InfoBehavior`. Этот класс будет проще, чем класс `ProgressBehavior`. Класс `InfoBehavior` нам очень пригодится в следующем разделе, так как будет предназначен для координации зависимостей не только от `android.widget.ListView`, но и от других классов, которые могут быть источниками событий прокрутки. Функциональность класса поведения `InfoBehavior` будет состоять в том, он будет во-первых, назначаться только виджетам `android.widget.TextView`.

(или `android.support.v7.widget.AppCompatTextView`), а во-вторых, он будет всего-лишь выводить в виджет `TextView` величину, на которую осуществляется прокрутка. Программный код класса `InfoBehavior` приведен в Листинге 2.34.

Листинг 2.34. Программный код класса поведения `InfoBehavior`

```
/***
 *
 * Class InfoBehavior implements information bar
 * behavior for scrolling in android.widget.
 * ListView
 *
 * Class ProgressBehavior реализует поведение
 * информбара для прокрутки android.widget.ListView
 */
class InfoBehavior extends CoordinatorLayout.
    Behavior<TextView>
{
    // ----- Class constants -----
    private final static String TAG =
        "===== InfoBehavior";

    // ----- Class members -----
    private int curScrollY = 0;

    // ----- Class methods -----
    public InfoBehavior()
    {
    }

    /**
     * Constructor for creating a InfoBehavior
     * instance through xml.
    }
```

```
/*
 * Конструктор для создания экземпляра
 * InfoBehavior через разметку.
 */
public InfoBehavior(Context context, AttributeSet attrs)
{
    super(context, attrs);
}

@Override
public boolean layoutDependsOn(
        CoordinatorLayout parent,
        TextView child, View dependency)
{
    return dependency instanceof ListView;
}

@Override
public boolean onDependentViewChanged(
        CoordinatorLayout parent,
        TextView child, View dependency)
{
    String str = "Scrolled: " + this.curScrollY;
    child.setText(str);
    return true;
}

@Override
public void onNestedScroll(
        CoordinatorLayout coordinatorLayout,
        TextView child,
        View target,
        int dxConsumed,
        int dyConsumed,
        int dxUnconsumed,
        int dyUnconsumed /*, int type*/)
{
```

```

Log.d(TAG, "onNestedScroll." +
        " target: " +
        target.getClass().getSimpleName() +
        ", dyConsumed: " + dyConsumed +
        ", dyUnconsumed: " + dyUnconsumed);

this.curScrollY += dyConsumed;
String str = "Scrolled: " + this.curScrollY;
child.setText(str);
}

@Override
public boolean onStartNestedScroll(
        CoordinatorLayout coordinatorLayout,
        TextView child,
        View directTargetChild,
        View target,
        int axes/*, int type*/)
{
    return directTargetChild instanceof ListView;
}
}

```

В макет Активности /res/layout/activity_main.xml добавим виджет **android.widget.TextView** которому назначим поведение **InfoBehavior** (см. Листинг 2.35).

Листинг 2.35. Добавление виджета **android.widget.TextView** в макет Активности и назначение этому виджету поведения **InfoBehavior**

```

<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/ android"
    xmlns:tools="http://schemas.android.com/tools"

```

```
xmlns:app="http://schemas.android.com/apk/res-auto"

    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    android:id="@+id/clMain"
    tools:context="itstep.com.myapp5.MainActivity">
    ...

<TextView
    android:layout_width="160dp"
    android:layout_height="wrap_content"
    android:layout_gravity="top|right"
    android:layout_marginRight="16dp"
    android:layout_marginTop="16dp"
    android:background="#80FFFF00"
    android:textSize="21sp"
    android:textColor="#003366"
    android:text=""
    app:layout_behavior=".InfoBehavior"
/>

</android.support.design.widget.CoordinatorLayout>
```

Как видно из Листинга 2.35, добавленный виджет **android.widget.TextView** располагается в верхнем правом углу контейнера **CoordinatorLayout** и имеет полупрозрачный желтый цвет фона. Внешний вид работы примера (модуль «app5») с Листингами 2.34, 2.35 показан на Рис. 2.18.

➲ **Подведем итоги.** Менеджер раскладки **android.support.design.widget.CoordinatorLayout** является мощным инструментом для создания сложного набора взаимосвязанных между собой виджетов.

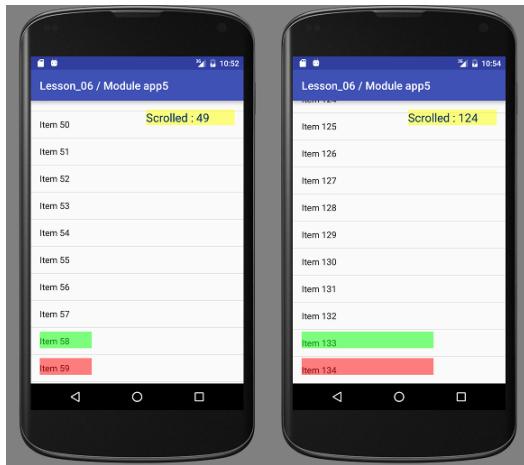


Рис. 2.18. Внешний вид работы примера из Листингов 2.34 и 2.35

Управление взаимосвязями между виджетами осуществляется при помощи объектов **CoordinatorLayout.Behavior**, которые назначаются размещенным в **CoordinatorLayout** виджетам. Можно создавать свои классы поведений, производных от класса **CoordinatorLayout.Behavior**. При этом, самостоятельное написание программного кода по оповещению менеджера **CoordinatorLayout** о необходимости выполнить координацию зависимостей между дочерними виджетами, является возможной но не очень интересной задачей (пример см. в Листинге 2.30). И здесь для нас есть хорошая новость! Компания Google представляет разработчикам целый набор классов, специально предназначенных для размещения внутри менеджера раскладки **CoordinatorLayout**, с целью предоставления разработчикам удобного механизма для управления взаимосвязями между сложными наборами размещенных

в **CoordinatorLayout** виджетами. С самыми значимыми из этих классов классами мы с вами познакомимся в следующих разделах.

Также добавим, что существует и целый набор классов поведений: **AppBarLayout.Behavior**, **AppBarLayout.ScrollingViewBehavior**, **BottomSheetBehavior<V extends View>**, **FloatingActionButton.Behavior**, **SwipeDismissBehavior<V extends View>**. Каждое из этих поведений реализует свою, очень интересную функциональность, и мы рекомендуем вам ознакомится с этими классами на сайте для разработчиков Android приложений.

Исходный код примера из Листингов 2.28, 2.29, 2.30, 2.31, 2.34, 2.35, находится в модуле «app5» среди файлов с исходными кодами, которые прилагаются к данному уроку.

2.6. Контейнер NestedScrollView

Менеджер раскладки [android.support.v4.widget.NestedScrollView](#) аналогичен ранее изученному нами менеджеру [android.widget ScrollView](#). Главное предназначение менеджера [android.support.v4.widget.NestedScrollView](#) заключается в том, то он предназначен для использования внутри менеджера раскладки [android.support.design.widget.CoordinatorLayout](#). Другими словами, этот менеджер самостоятельно фиксирует свои события прокрутки и самостоятельно оповещает о них родительскому **CoordinatorLayout**. Такая функциональность упрощает разработчику работу над координацией зависимостей между виджетами. Напомним, что в предыдущем разделе

в Листинге 2.30 подобную функциональность для списка `android.widget.ListView` мы реализовывали самостоятельно и обратили внимание на то, что такая самостоятельная реализация возможна, но не является интересной. Так вот, менеджер `android.support.v4.widget.NestedScrollView` избавляет разработчиков от такой вот не интересной работы. Сразу отметим, что `android.support.v4.widget.NestedScrollView` не является аналогом списка `android.widget.ListView`, и если речь зашла об аналоге списка `android.widget.ListView` из библиотек совместимости, то таким аналогом можно назвать виджет `android.support.v7.widget.RecyclerView` который будет нами рассмотрен ниже в текущем уроке.

Иерархия классов для класса `android.support.v4.widget.NestedScrollView` выглядит следующим образом:

```
java.lang.Object
  |
  +-- android.view.View
  |
  +-- android.view.ViewGroup
  |
  +-- android.widget.FrameLayout
  |
  +-- android.support.v4.widget.NestedScrollView
```

Как видим, контейнер `android.support.v4.widget.NestedScrollView` является прямым наследником хорошо известного нам контейнера `android.widget.FrameLayout`. Поэтому сразу же перейдем к рассмотрению примера

использования контейнера `android.support.v4.widget.NestedScrollView` на практике. Для этого примера создан модуль «app6» в нашем проекте.

Файл макета внешнего вида Активности `/res/layout/activity_main.xml` с использованием `android.support.v4.widget.NestedScrollView` приведен в Листинге 2.36.

Листинг 2.36. Файл макета внешнего вида Активности с применением контейнера `android.support.v4.widget.NestedScrollView`

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"

    android:layout_width="match_parent"
    android:layout_height="match_parent"

    android:fitsSystemWindows="true"
    android:id="@+id/clMain"
    tools:context="itstep.com.myapp6.MainActivity">

    <android.support.v4.widget.NestedScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView
            android:layout_width="match_parent"

            android:layout_height="wrap_content"
            android:id="@+id/tvOne"
            />
    </android.support.v4.widget.NestedScrollView>
```

```
<TextView  
    android:layout_width="150dp"  
    android:layout_height="30dp"  
    android:background="#8000FF00"  
    android:layout_margin="16dp"  
    android:layout_gravity="bottom|left"  
    android:textSize="21sp"  
    android:textColor="#000"  
    app:layout_behavior="itstep.com.myapp6.  
        InfoBehavior"  
/>  
  
</android.support.design.widget.CoordinatorLayout>
```

Как видно из Листинга 2.36, главным контейнером Активности является контейнер `android.support.design.widget.CoordinatorLayout`. Контейнер `android.support.v4.widget.NestedScrollView` является дочерним виджетом для контейнера `CoordinatorLayout`. Внутрь контейнера `android.support.v4.widget.NestedScrollView` вложен виджет `android.widget.TextView`, который имеет идентификатор `tvOne`. Содержимое этого виджета будет заполняться программно из метода `onCreate()` Активности (см. Листинг 2.37). Также, в контейнер `CoordinatorLayout` добавлен еще один виджет `android.widget.TextView`, которому назначено поведение в виде класса `InfoBehavior`. Этот виджет находится в левом нижнем углу контейнера `CoordinatorLayout` и имеет полупрозрачный зеленый цвет фона (см. Рис. 2.19). Класс `InfoBehavior` уже рассматривался нами в примере из предыдущего раздела (см. Листинг 2.34) и в примере те-

кущего раздела этот класс попал с незначительными изменениями (см. Листинг 2.38).

Листинг 2.37. Заполнение содержимым виджета android.widget.TextView с идентификатором tvOne для демонстрации работы скроллинга в контейнере android.support.v4.widget.NestedScrollView

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

    // ----- Filling the TextView content -----
    // ----- Заполнение TextView содержимым -----

    TextView tvOne = (TextView)
        this.findViewById(R.id.tvOne);
    String[] arr =
    {
        "Lemon ", "Orange ", "Peach ", "Banana ",
        "Winter ", "Summer ", "Spring ",
        "Autumn ", "Monday ", "Tuesday ",
        "Wednesday ", "Thursday ", "Friday ",
        "Saturday ", "Sunday "
    };

    String content = "";
    for (int i = 0; i < 1000; i++)
    {
        content += arr[(int)(Math.random() *
            arr.length)] + (i + 1) + " ";
    }

    tvOne.setText(content);
}
```

Обращаем ваше внимание, что цель рассматриваемого примера не только знакомство с контейнером `android.support.v4.widget.NestedScrollView` как с контейнером, который обеспечивает прокрутку своих дочерних виджетов. Акцент в рассматриваемом примере делается на уже готовую совместимость («заточенность») контейнера `android.support.v4.widget.NestedScrollView` для использования его внутри `CoordinatorLayout` для прямого участия в обеспечении координации зависимостей между дочерними виджетами этого `CoordinatorLayout`. Именно с этой целью в данном примере используется класс поведения `InfoBehavior` из предыдущего раздела (см. Листинг 2.34). Как уже упоминалось, класс `InfoBehavior` пришлось немного модифицировать — чтобы он мог использоваться как объект поведения для виджетов, находящихся в зависимости от `android.support.v4.widget.NestedScrollView`. Изменения в классе `InfoBehavior` приведены в Листинге 2.38 (выделены жирным шрифтом).

Листинг 2.38. Изменения в классе `InfoBehavior` для использования его при координации зависимостей от контейнера `android.support.v4.widget.NestedScrollView`

```
class InfoBehavior extends CoordinatorLayout.  
    Behavior<TextView>  
{  
    ...  
    @Override  
    public boolean layoutDependsOn(  
        CoordinatorLayout parent,  
        TextView child, View dependency)  
    {
```

```
        return dependency instanceof
            android.support.v4.widget.NestedScrollView;
    }
    ...
@Override
public boolean onStartNestedScroll(
        CoordinatorLayout coordinatorLayout,
        TextView child,
        View directTargetChild, View target,
        int axes/*, int type*/)
{
    return directTargetChild instanceof
        android.support.v4.widget.
        NestedScrollView;
}
}
```

Как видно из Листинга 2.38, класс **InfoBehavior** предназначен для использования в случае зависимости виджета, которому **InfoBehavior** назначен в качестве объекта поведения, от событий прокрутки в контейнере **android.support.v4.widget.NestedScrollView**. Внешний вид работы примера из Листингов 2.36, 2.37, 2.38, изображен на Рис. 2.19.

Как видно из Рис. 2.19, объект поведения **InfoBehavior** отображает в виджете, которому он назначен, количество прокрученных пикселей в **android.widget.TextView** (с идентификатором **tvOne**). Данный пример подтверждает, что контейнер **android.support.v4.widget.NestedScrollView** является полностью приспособленным для использования при координации зависимостей между виджетами внутри **CoordinatorLayout**.

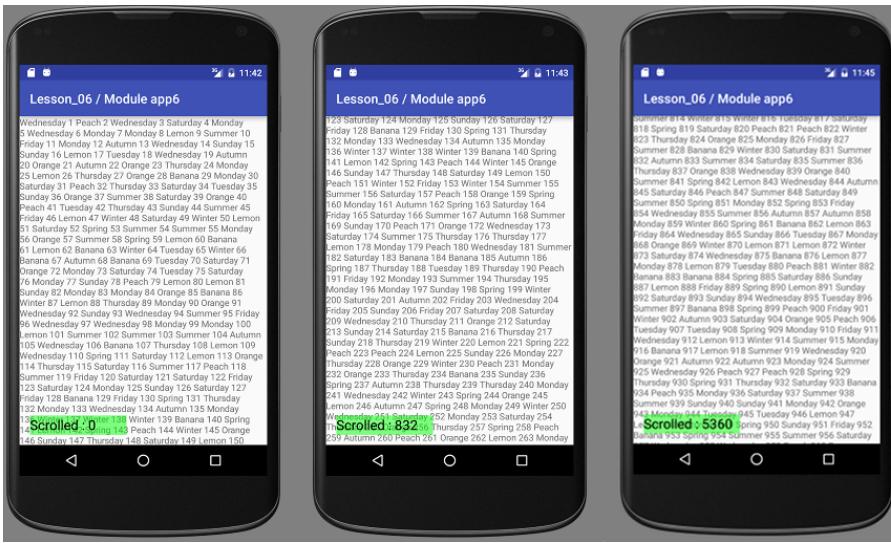


Рис. 2.19. Внешний вид работы примера из Листингов 2.36, 2.37, 2.38

Исходный код примера данного раздела находится в модуле «app6» среди файлов с исходными кодами, которые прилагаются к данному уроку.

2.7. Контейнер AppBarLayout

Наверное вы уже обратили внимание, что при изучении менеджера `android.support.design.widget.CoordinatorLayout` мы не использовали `Toolbar` в рассматриваемых примерах. Это было сделано не случайно. Дело в том, что использование `Toolbar` внутри `CoordinatorLayout` оказывается немного более сложным, чем может показаться вначале. Вариант использования `Toolbar` вне контейнера `CoordinatorLayout` мы рассматривать не будем — так как именно использова-

ние **Toolbar** внутри **CoordinatorLayout** является желательным и рекомендованным, в чем вы сможете убедиться ниже в данном уроке.

Итак, для примера текущего раздела создан модуль «app7». В этот модуль полностью скопировано содержимое модуля «app6» — а именно содержимое файла /res/layout/activity_main.xml и MainActivity.java. Далее, в файл макета внешнего вида Активности /res/layout/activity_main.xml добавим виджет **android.support.v7.widget.Toolbar**. Содержимое файла макета /res/layout/activity_main.xml приведено в Листинге 2.39.

Листинг 2.39. Содержимое файла макета Активности /res/layout/activity_main.xml рассматриваемого примера

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    ...
    tools:context="itstep.com.lesson_myapp7.MainActivity">

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:background="@color/colorPrimary"
        android:subtitleTextColor=
            "@color/primary_text_material_light"
        android:titleTextColor=
            "@color/primary_text_material_light"
    />

    <android.support.v4.widget.NestedScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent">
```

```
<TextView  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:id="@+id/tvOne" />  
  
</android.support.v4.widget.NestedScrollView>  
  
<TextView  
    android:layout_width="150dp"  
    android:layout_height="30dp"  
    android:background="#8000FF00"  
    android:layout_margin="16dp"  
    android:layout_gravity="bottom|left"  
    android:textSize="21sp"  
    android:textColor="#000"  
    app:layout_behavior="itstep.com.  
        lesson_myapp7.InfoBehavior"  
/>  
  
</android.support.design.widget.CoordinatorLayout>
```

Как видно из Листинга 2.39, корневым контейнером Активности является контейнер **CoordinatorLayout**. Внутрь этого контейнера помещены: виджет **Toolbar**, контейнер **NestedScrollView** и виджет **android.widget.TextView** с назначенным поведением **InfoBehavior**. Поведение **InfoBehavior** в текущем примере необходимо для демонстрации величины прокрутки в контейнере **NestedScrollView**, так как на скриншотах этого примера визуально сложно увидеть величину прокрутки контейнера **NestedScrollView**. Давайте теперь запустим приложение с макетом Активности из Листинга 2.39 и посмотрим на результат, который изображен на Рис. 2.20.

Урок № 6

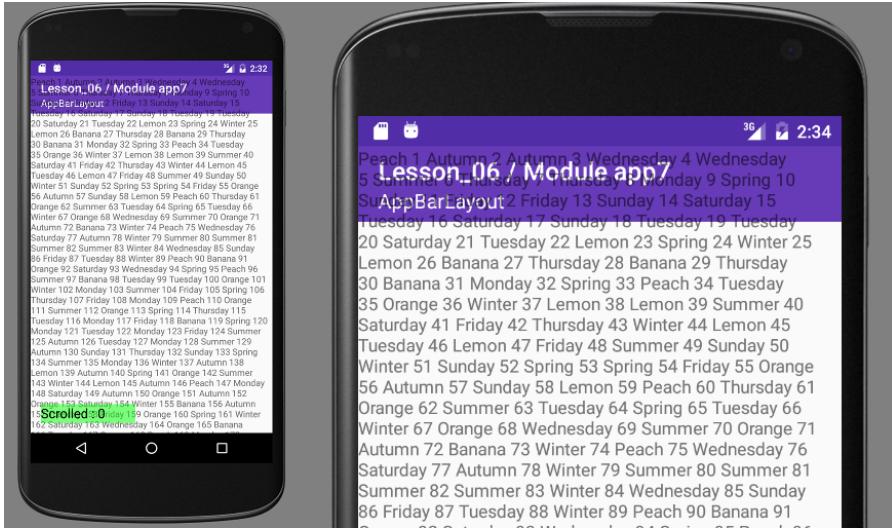


Рис. 2.20. Результат работы примера из Листинга 2.39.
Слева — скриншот всего экрана, справа — увеличенная
верхняя часть экрана с тулбаром

Как видно из Рис. 2.20, результат верстки из Листинга 2.39 не дал ожидаемого результата. И это логично, так как менеджер раскладки `android.support.design.widget.CoordinatorLayout` это прямой наследник `android.widget.FrameLayout`, и следовательно, размещает свои дочерние виджеты в Z-порядке, то есть друг над другом, что мы и наблюдаем на Рис. 2.20.

Решением данной проблемы является использование специального менеджера раскладки `android.support.design.widgetAppBarLayout`. Менеджер раскладки `AppBarLayout` — это вертикальный `android.widget.LinearLayout`, который реализует многие функции концепции Material Design, в том числе и функции, связанные с жестами прокрутки. И конечно же, `AppBarLayout` пред-

назначен для размещения внутри себя таких виджетов как `android.support.v7.widget.Toolbar` и `android.support.design.widget.TabLayout`.

Иерархия классов для класса `android.support.design.widget.AppBarLayout` имеет следующий вид:

```
java.lang.Object
|
+--- android.view.View
|
+--- android.view.ViewGroup
|
+--- android.widget.LinearLayout
|
+--- android.support.design.widget.AppBarLayout
```

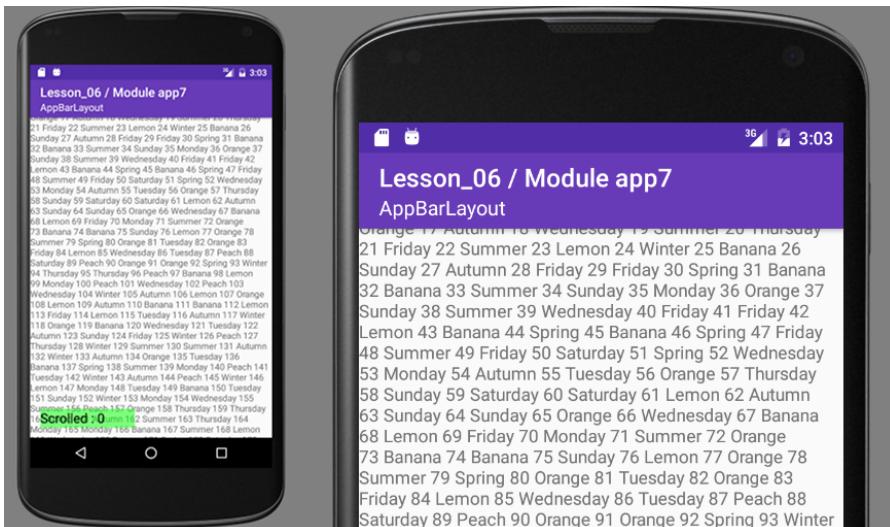


Рис. 2.21. Результат работы примера из Листинга 2.40. Слева — скриншот всего экрана, справа — увеличенная верхняя часть экрана с `AppBarLayout` и размещенным внутри него тулбаром

Давайте разместим наш **Toolbar** из Листинга 2.39 внутрь контейнера **android.support.design.widget.AppBarLayout**, как это показано в Листинге 2.40.

Листинг 2.40. Размещение виджета **android.support.v7.widget.Toolbar** внутрь контейнера **android.support.design.widget.AppBarLayout**

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

<android.support.design.widget.CoordinatorLayout
    ...
    tools:context="itstep.com.lesson_myapp7.MainActivity">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="60dp"
            android:background="@color/colorPrimary"
            android:subtitleTextColor=
                "@color/primary_text_material_light"
            android:titleTextColor=
                "@color/primary_text_material_light"
        />

    </android.support.design.widget.AppBarLayout>
    <android.support.v4.widget.NestedScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content">

    
```

```
        android:id="@+id/tvOne"  
    />  
  
    </android.support.v4.widget.NestedScrollView>  
    ...  
</android.support.design.widget.CoordinatorLayout>
```

Результат применения контейнера **AppBarLayout** из Листинга 2.40 изображен на Рис. 2.21. Как видно из Рис.2.21, результат применения **AppBarLayout** частично улучшил ситуацию. Однако, если присмотреться, то при величине скроллинга 0 пикселей содержимое текстового поля **android.widget.TextView** с идентификатором **tvOne**, оказалось под виджетом **android.support.v7.widget.Toolbar** (см. Рис. 2.21). Причиной этому является то, что контейнер **NestedScrollView** не знает о наличии рядом с ним контейнера **AppBarLayout**. Однако, если контейнеру **NestedScrollView** назначить поведение **AppBarLayout.ScrollingViewBehavior** то менеджер **CoordinatorLayout** сможет скоординировать раскладку **NestedScrollView** и **AppBarLayout** таким образом, чтобы эти контейнеры не пересекались.

Поведение **AppBarLayout.ScrollingViewBehavior** как рак и предназначено для координации расположения контейнера **AppBarLayout** и контейнера который обеспечивает функциональность скроллинга. Для xml верстки для класса **AppBarLayout.ScrollingViewBehavior** есть специальная строковая константа: **@string/appbar_scrolling_view_behavior**. Давайте назначим контейнеру **NestedScrollView** из Листинга 2.40 поведение **AppBarLayout.ScrollingViewBehavior** (см. Листинг 2.41).

Листинг 2.41. Назначение контейнеру android.support.v4.widget.NestedScrollView поведения AppBarLayout.ScrollingViewBehavior

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    ...
    tools:context=
        "itstep.com.lesson_myapp7.MainActivity">
    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        ...
    </android.support.design.widget.AppBarLayout>

    <android.support.v4.widget.NestedScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior=
            "@string/appbar_scrolling_view_behavior">
        ...
    </android.support.v4.widget.NestedScrollView>
    ...
</android.support.design.widget.CoordinatorLayout>
```

Результат работы примера из Листинга 2.41 изображен на Рис. 2.22.

Как видно из Рис. 2.22, контейнеры **NestedScrollView** и **AppBarLayout** не пересекаются и мы получили желанный результат. Однако, это еще далеко не все, что предлагают нам библиотеки совместимости. Еще более интересный контейнер **android.support.design.widget.CollapsingToolbarLayout** рассматривается в следующем разделе данного урока.

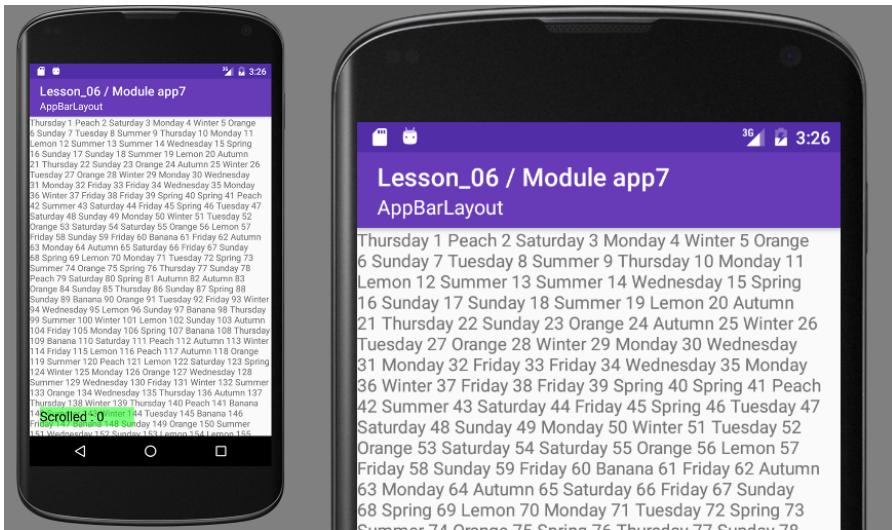


Рис. 2.22. Результат работы примера из Листинга 2.41

Исходный код примера данного раздела находится в модуле «app7» среди файлов с исходными кодами, которые прилагаются к данному уроку.

2.8. Контейнер CollapsingToolbarLayout

Контейнер [android.support.design.widget.CollapsingToolbarLayout](#) добавляет дополнительную функциональность для панели инструментов (так есть для Toolbar или для Actionbar) позволяя ей сворачиваться или разворачиваться в зависимости от прокрутки содержимого в контейнере [android.support.design.widget.CoordinatorLayout](#). Как выглядит сворачивание панели инструментов при использовании контейнера [CollapsingToolbarLayout](#) изображено на Рис. 2.23. Фактически, класс [android.support.design.widget.CollapsingToolbarLayout](#)

реализует паттерн «Decorator» (он же «Wrapper») для панели инструментов.

Контейнер `android.support.design.widget.CollapsingToolbarLayout` предназначен для использования в качестве прямого дочернего элемента для контейнера `android.support.design.widget.AppBarLayout` и предоставляет следующие возможности:

- Свертывание заголовка панели инструментов. Размер шрифта для текста заголовка панели инструментов увеличен, когда прокрутка содержимого не осуществлялась, и уменьшается по мере того как увеличивается прокрутка содержимого (см. Рис. 2.23). Разворачивание и сворачивание заголовка может быть изменено при помощью атрибутов `collapsedTextAppearance` и `extendedTextAppearance`.
- Подстановка вместо панели инструментов некоторого изображения — холста (*scrim*). Этот холст показывается или скрывается когда прокрутка достигает определенного порога. На Рис. 2.23 в качестве холста выступает сплошной желтый (#FFC107, «Material Amber 500») цвет. Растровое изображение (фотография) цветка не является холстом, а является виджетом `android.widget.ImageView`, который плавно «уступает место» панели инструментов когда прокрутка достигает определенного порога.
- Возможность смещения дочерних виджетов во время прокрутки или возможность закрепления дочерних виджетов на месте во время прокрутки.

2. Библиотеки AppCompat v.7 и Design Support Library

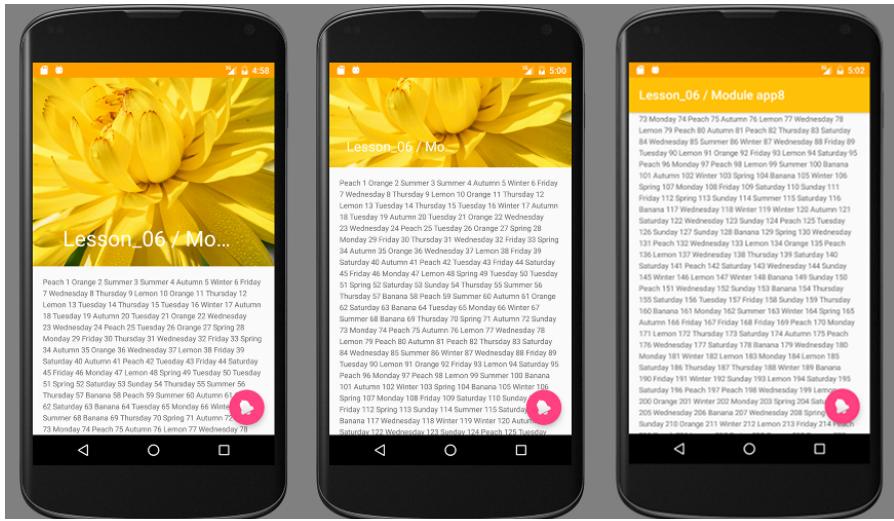


Рис. 2.23. Демонстрация функциональности, которую предоставляет контейнер CollapsingToolbarLayout по сворачиванию и разворачиванию панели инструментов

Иерархия классов для `android.support.design.widget.CollapsingToolbarLayout` выглядит следующим образом:

```
java.lang.Object
  |
  +--- android.view.View
    |
    +--- android.view.ViewGroup
      |
      +--- android.widget.FrameLayout
        |
        +--- android.support.design.widget.CollapsingToolbarLayout
```

Как видно из иерархии классов, контейнер `CollapsingToolbarLayout` является прямым наследником контейнера `android.widget.FrameLayout`.

Давайте рассмотрим применение **CollapsingToolbarLayout** на примере. Для этого создадим модуль «app8» в проекте Android Studio, который прилагается к данному уроку. В примере в контейнер **CollapsingToolbarLayout** вставляется два дочерних виджета: **android.support.v7.widget.Toolbar** который является панелью инструментов и **android.widget.ImageView** который будет замещать панель инструментов когда она развернута, как показано на Рис. 2.23. Ну а контейнер **CollapsingToolbarLayout**, как говорилось выше, добавляется в качестве прямого дочернего виджета в контейнер **AppBarLayout**. Содержимое файла ресурсов (/res/layout/activity_main.xml) с макетом внешнего вида Активности рассматриваемого примера приведено в Листинге 2.42.

Листинг 2.42. Содержимое файла ресурсов (/res/layout/activity_main.xml) с макетом внешнего вида Активности рассматриваемого примера

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

<android.support.design.widget.CoordinatorLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    android:id="@+id/clMain"
    tools:context="itstep.com.myapp8.MainActivity">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content">

    <android.support.design.widget.
        CollapsingToolbarLayout
        android:id="@+id/main_collapsing"
        android:layout_width="match_parent"
        android:layout_height="300dp"
        app:layout_scrollFlags=
            "scroll|exitUntilCollapsed"
        android:fitsSystemWindows="true"
        app:contentScrim="?attr/colorPrimary"
        app:expandedTitleMarginStart="48dp"
        app:expandedTitleMarginEnd="64dp">

        <ImageView
            android:id="@+id/main.backdrop"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:scaleType="centerCrop"
            android:fitsSystemWindows="true"
            android:src=
                "@drawable/material_background_1"
            app:layout_collapseMode="parallax"
        />

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:subtitleTextColor=
                "@color/primary_text_material_light"
            android:titleTextColor=
                "@color/primary_text_material_light"
            app:popupTheme=
                "@style/ThemeOverlay.AppCompat.Light"
            app:layout_collapseMode="pin"
        />
```

```
</android.support.design.widget.  
    CollapsingToolbarLayout>  
  
</android.support.design.widget.AppBarLayout>  
  
<android.support.v4.widget.NestedScrollView  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    app:layout_behavior=  
        "@string/appbar_scrolling_view_behavior">  
  
    <TextView  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:textSize="12sp"  
        android:lineSpacingExtra="4dp"  
        android:padding=  
            "@dimen/activity_horizontal_margin"  
        android:id="@+id/tvOne"  
    />  
  
</android.support.v4.widget.NestedScrollView>  
  
    <TextView  
        android:layout_width="150dp"  
        android:layout_height="30dp"  
        android:background="#80FFFF00"  
        android:layout_margin="16dp"  
        android:layout_gravity="bottom|left"  
        android:textSize="21sp"  
        android:textColor="#000"  
        app:layout_behavior=  
            "itstep.com.myapp8.InfoBehavior"  
    />
```

```
<android.support.design.widget.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|right"
    android:layout_marginBottom=
        "@dimen/activity_vertical_margin"
    android:layout_marginRight=
        "@dimen/activity_horizontal_margin"
    android:src=
        "@android:drawable/ic_popup_reminder"
    />

</android.support.design.widget.CoordinatorLayout>
```

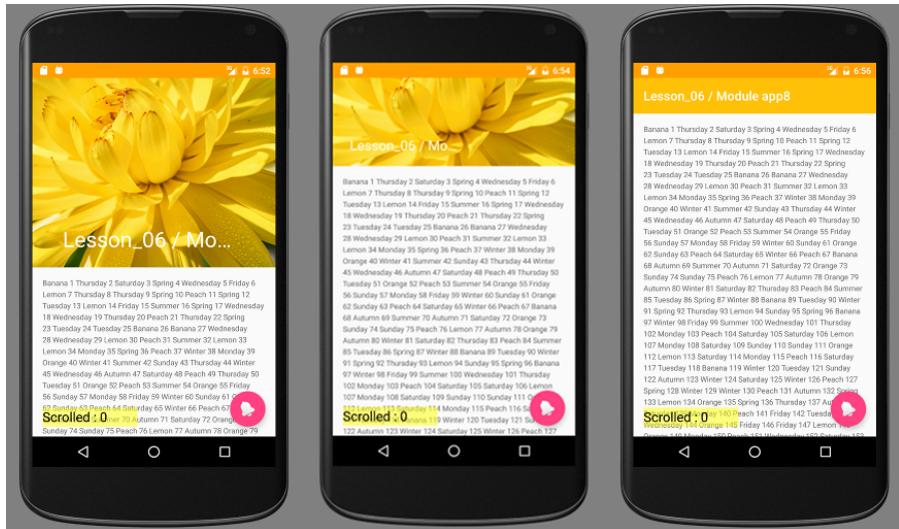


Рис. 2.24. Внешний вид работы примера из Листинга 2.42. Скриншоты слева направо показывают начало прокрутки содержимого и происходящее при этом сворачивание панели инструментов с исчезновением холста и появлением тулбара

Как видно из Листинга 2.42, в родительский контейнер **CoordinatorLayout** добавлены следующие виджеты: контейнер **AppBarLayout**, контейнер **NestedScrollView**, виджет «плавающая кнопка» **FloatingActionButton** и виджет **TextView** которому назначено поведение **InfoBehavior** из Листинга 2.34. Этот виджет **TextView** с поведением **InfoBehavior** необходим в рассматриваемом примере для более детальной демонстрации функциональности контейнера **CollapsingToolbarLayout**.

В контейнер **AppBarLayout** добавлен в качестве дочернего виджета контейнер **CollapsingToolbarLayout**, в котором находятся панель инструментов **Toolbar** и замещающий панель инструментов виджет **android.widget.ImageView**. Контент (содержимое) в рассматриваемом примере располагается в текстовом поле **android.widget.TextView** с идентификатором **tvOne**. Заполнение этого виджета осуществляется случайными строками в методе **onCreate()** класса Активности. Также в методе **onCreate()** происходит назначение объекта **android.support.v7.widget.Toolbar** (идентификатор **toolbar**) в качестве панели инструментов для приложения. Код метода **onCreate()** не приведен в данном уроке, так как не содержит никакой новой для нас информации. Вы можете ознакомиться с методом **onCreate()** в модуле «app8» среди файлов с исходными кодами, которые прилагаются к текущему уроку.

Внешний вид работы примера из Листинга 2.42 изображен на Рис. 2.24.

Как видно из Рис. 2.24, когда начинается прокрутка содержимого Активности происходит сворачивание па-

нели инструментов. При этом, текстовое поле с поведением **InfoBehavior** показывает величину прокрутки в контейнере **NestedScrollView** которая равна нулю. Другими словами, прокрутка в **NestedScrollView** начнется тогда, когда панель инструментов достигнет своего минимального размера.

В Листинге 2.42 мы видим использование нескольких новых для нас xml атрибутов. Эти атрибуты имеют прямое отношение к использованию контейнера **android.support.design.widget.CollapsingToolbarLayout**. Давайте познакомимся с этими атрибутами.

Для контейнера **android.support.design.widget.CollapsingToolbarLayout** используются следующие атрибуты (и их значения):

- **android: fitsSystemWindows = "true"**. Атрибут **android: fitsSystemWindows** принадлежит классу **android.view.View**. Предназначен для настройки раскладки виджетов с учетом системных окон, таких как строка состояния. Если атрибут равен true, то виджет получит отступы чтобы оставить пространство для системных окон. Этот атрибут играет роль только в случае, если виджет принадлежит Активности которая не является встроенной.
- **app:layout_scrollFlags="scroll|exitUntilCollapsed"**. Атрибут **app:layout_scrollFlags** принадлежит классу **AppBarLayout.LayoutParams**. Предназначен для установки флагов прокрутки. Флаги могут комбинироваться путем побитового сложения. Значения флагов следующие:

- ▷ **scroll** — прокрутка этого виджета напрямую зависит от событий прокрутки о которых оповещает **CoordinatorLayout**. Этот флаг должен быть установлен для вступления в силу любого из других флагов. Если у соседних виджетов, находящихся перед этим виджетом, нет этого флага, то назначение этого флага данному виджету не даст никакого эффекта;
- ▷ **snap** — после окончания прокрутки, если виджет будет только частично видимым, то он будет прокручен до ближайшего края. Например, если в родительском контейнере отображается только нижние 25% виджета, то виджет будет полностью прокручен до своего полного исчезновения. И наоборот, если в родительском контейнере отображается верхние 75% виджета, то виджет будет прокручен до своего полного отображения;
- ▷ **enterAlways** — при прокрутке, виджет будет прокручиваться по любому событию прокрутки вниз до своего полного исчезновения. Если контейнеру **CollapsingToolbarLayout** из Листинга 2.42 поставить значение атрибута `app:layout_scrollFlags="scroll|enterAlways"`, то при прокрутке вниз **Toolbar** также будет прокручен до своего полного исчезновения, как это изображено на Рис. 2.25;
- ▷ **enterAlwaysCollapsed** — дополнительный флаг для флага **enterAlways**. Если виджет прокручен при прокрутке вниз до своего полного исчезновения и пользователь сделает прокрутку вверх, то исчезнувший виджет сразу же появится на величину своей ми-

нимальной высоты. Если контейнеру **CollapsingToolbarLayout** из Листинга 2.42 поставить значение атрибута `app:layout_scrollFlags="scroll|enterAlways|enterAlwaysCollapsed"`, то ранее исчезнувший при прокрутке вниз **Toolbar** будет показан на свою минимальную высоту при небольшой прокрутке вверх, как это изображено на Рис. 2.26. Ну а когда все содержимое будет прокрученено вверх до своего начального положения, то тулбар примет свою максимальную высоту;

- ▷ **exitUntilCollapsed** — при прокрутке вниз виджет будет прокручиваться до тех пор, пока не достигнет своей минимальной высоты. Внешне это будет выглядеть как сворачивание виджета. Как работает этот флаг вы уже могли наблюдать на Рис. 2.24;
- **app:contentScrim = "?attr/colorPrimary"**. Атрибут **app:contentScrim** принадлежит классу **android.support.design.CollapsingToolbarLayout**. Предназначен для назначения панели инструментов (**Toolbar** или **ActionBar**) холста (фонового изображения или цвета), который применится к панели инструментов тогда, когда она выйдет на передний план. В нашем рассматриваемом примере панель инструментов выходит на передний план, когда виджет **android.widget.ImageView** исчезает. Значение для этого атрибута должно быть **Drawable** ресурсом. В нашем рассматриваемом примере этому атрибуту присваивается первичный цвет (`@color/colorPrimary`), который равен значению `#FFC107` (**Material Amber 500**).

Урок № 6

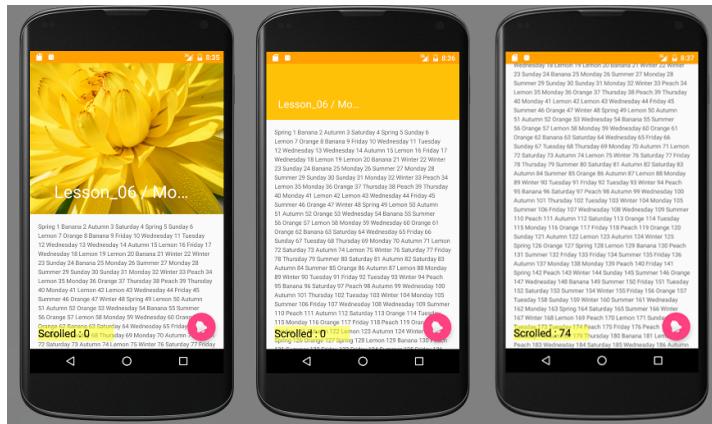


Рис. 2.25. Прокрутка панели инструментов для случая, когда контейнеру CollapsingToolbarLayout установлены значения атрибута app:layout_scrollFlags=>scroll|enterAlways». На всех скриншотах скроллинг осуществляется вниз (то есть пользователь нажимает внизу экрана и ведет пальцем вверх)

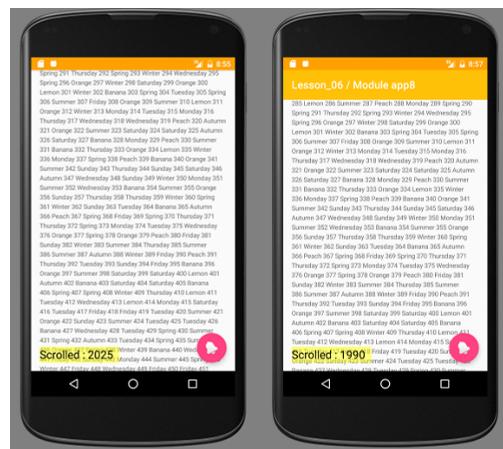


Рис. 2.26. Прокрутка панели инструментов для случая, когда контейнеру CollapsingToolbarLayout установлены значения атрибута app:layout_scrollFlags=>scroll|enterAlways|enterAlwaysCollapsed». На левом скриншоте скроллинг осуществляется вниз, а на правом скриншоте скроллинг осуществляется вверх на небольшую величину

- `app:expandedTitleMarginStart="48dp"` — атрибут `app:expandedTitleMarginStart` принадлежит классу `android.support.design.widget.CollapsingToolbarLayout`. Устанавливает начальный отступ расширенного заголовка в пикселях. Добавлено в версии API 24.0.1.
- `app:expandedTitleMarginEnd="64dp"` — атрибут `app:expandedTitleMarginEnd` принадлежит классу `android.support.design.widget.CollapsingToolbarLayout`. Устанавливает конечный отступ расширенного заголовка в пикселях. Добавлено в версии API 24.0.1.

Для контейнера `android.widget.ImageView` используются следующие атрибуты (и их значения):

- `app:layout_collapseMode="parallax"` — Атрибут `app:layout_collapseMode` принадлежит классу `android.support.design.widget.CollapsingToolbarLayout.LayoutParams`. Этот атрибут устанавливает режим сворачивания для виджета. Значения для этого атрибута следующие: `off` (виджет не имеет режима сворачивания), `parallax` (виджет будет смещаться при прокрутке), `pin` (виджет будет поставлен на место когда при прокрутке он достигнет конца контейнера `CollapsingToolbarLayout`). Как видно из Листинга 2.42, для виджета `android.widget.ImageView` значение этого атрибута равно `parallax`. И действительно, во время прокрутки содержимого Активности виджет `android.widget.ImageView` смещается при этом продолжая показывать свое изображение по центру как указано в его атрибуте `android:scaleType = "centerCrop"`.

Для контейнера `android.support.v7.widget.ToolBar` используются следующие атрибуты (и их значения):

- `app:layout_collapseMode="pin"` — См. описание этого атрибута выше. Как видно из Листинга 2.42, для виджета `android.support.v7.widget.ToolBar` значение этого атрибута равно `pin`. Во время прокрутки содержимого Активности этот виджет занимает свое место как только он достигает края контейнера `CollapsingToolbarLayout`.
- `app:popupTheme="@style/ThemeOverlay.AppCompat.Light"` — Атрибут `app:popupTheme` принадлежит классу `android.support.v7.widget.ToolBar`. С помощью этого атрибута можно указать тему, используемую для создания выпадающих меню. По умолчанию используется та же тема, что и для панели инструментов.

Как видим, перечисленные выше атрибуты не являются простыми для понимания. Однако, можно смело использовать содержимое Листинга 2.42 в качестве готового решения для создания эффектных приложений.

Исходный код рассмотренного примера можно найти в модуле «app8» среди файлов с исходными кодами, которые прилагаются к данному уроку.

Далее, а что если нам необходимо создать приложение в котором будет присутствовать панель навигации `android.support.design.widget.NestedScrollView` (совместно с контейнером `android.support.v4.widget.DrawerLayout`) и сворачивающаяся панель инструментов (то есть контейнеры `android.support.design.CoordinatorLayout`, `android.`

`support.design.AppBarLayout`, `android.support.design.CollapsingToolbarLayout`)? Такая реализация приложения вполне возможна и сейчас мы создадим такое приложение. Для этой цели создан модуль «app9» в проекте приложения который прилагается к данному уроку.

Интерес в рассматриваемом примере представляет файл с макетом внешнего вида Активности `/res/layout/activity_main.xml`. За основу возьмем содержимое Листинга 2.6 и вместо комментария `<!-- Your contents -->` этого Листинга вставим содержимое Листинга 2.42. Полученный файл макета Активности приведен в Листинге 2.43.

Листинг 2.43. Совместное использование контейнеров `DrawerLayout`, `CoordinatorLayout`, `AppBarLayout`, `CollapsingToolbarLayout`

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

<android.support.v4.widget.DrawerLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:app=
        "http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    tools:context="itstep.com.myapp9.MainActivity">

    <android.support.design.widget.CoordinatorLayout
        xmlns:android=
            "http://schemas.android.com/apk/res/android"
```

```
xmlns:app=
    "http://schemas.android.com/apk/res-auto"

    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    android:id="@+id/clMain">

<android.support.design.widget.AppBarLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <android.support.design.widget.
        CollapsingToolbarLayout
        android:id="@+id/main_collapsing"
        android:layout_width="match_parent"
        android:layout_height="300dp"
        app:layout_scrollFlags =
            "scroll|exitUntilCollapsed"
        android:fitsSystemWindows="true"
        app:contentScrim="?attr/colorPrimary"
        app:expandedTitleMarginStart="48dp"
        app:expandedTitleMarginEnd="64dp">

        <ImageView
            android:id="@+id/main.backdrop"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:scaleType="centerCrop"
            android:fitsSystemWindows="true"
            android:src =
                "@drawable/material_background_1"
            app:layout_collapseMode="parallax"
        />

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
```

```
        android:layout_width="match_parent"
        android:layout_height="60dp"
        app:popupTheme=
            "@style/ThemeOverlay.AppCompat.Light"
        app:layout_collapseMode="pin"
    />
</android.support.design.widget.
    CollapsingToolbarLayout>

</android.support.design.widget.AppBarLayout>
<android.support.v4.widget.NestedScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior=
        "@string/appbar_scrolling_view_behavior">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="12sp"
        android:lineSpacingExtra="4dp"
        android:padding=
            "@dimen/activity_horizontal_margin"
        android:id="@+id/tvOne"
    />

</android.support.v4.widget.NestedScrollView>
<android.support.design.widget.
    FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|right"
    android:layout_marginBottom=
        "@dimen/activity_vertical_margin"
    android:layout_marginRight=
        "@dimen/activity_horizontal_margin"
```

```
    android:src=
        "@android:drawable/ic_popup_reminder"
    />

</android.support.design.widget.CoordinatorLayout>
<android.support.design.widget.NavigationView
    android:id="@+id/navigation"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:layout_gravity="start"
    app:menu="@menu/drawer_menu"
    app:headerLayout="@layout/drawer_header"
    />

</android.support.v4.widget.DrawerLayout>
```

Результат работы примера из Листинга 2.43 изображен на Рис. 2.27.

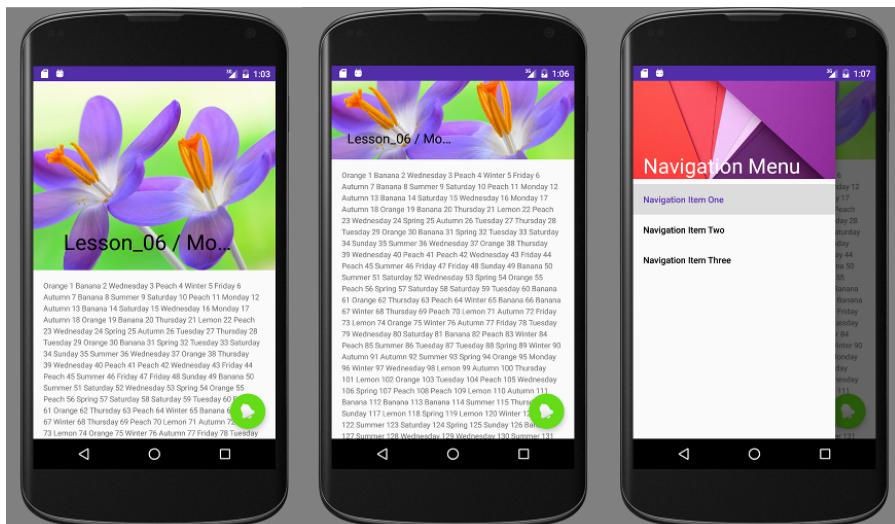


Рис. 2.27. Внешний вид работы примера из Листинга 2.43

Как видно из Рис. 2.27, совместить контейнеры **DrawerLayout**, **CoordinatorLayout**, **AppBarLayout**, **CollapsingToolbarLayout** совместно с **ToolBar** несложно. При этом сохраняются эффекты сворачивания и разворачивания панели инструментов (первые два скриншота на Рис. 2.27) и навигационное меню (последний скриншот на Рис. 2.27).

Обратите внимание, что на Рис. 2.27 цвет заголовка панели инструмента черный, хотя в методе **onCreate()** класса Активности назначается белый цвет тексту заголовка тулбара **android.support.v7.widget.Toolbar** как это показано в Листинге 2.11. Дело в том, что у контейнера **android.support.design.widget.CollapsingToolbarLayout** есть свои значения для цвета текста заголовка как для свернутого так и для развернутого состояния. Задать цвет текста заголовка панели инструментов для свернутого (*collapsed*) и развернутого (*expanded*) состояний можно при помощи методов класса **android.support.design.widget.CollapsingToolbarLayout**:

```
void setExpandedTitleColor(int color);
void setCollapsedTitleTextColor(int color);
```

Программная реализация, показана в Листинге 2.44.

Листинг 2.44. Назначение цвета тексту заголовка панели инструментов для свернутого и развернутого состояний

```
@Override
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
```

```
setContentView(R.layout.activity_main);

// ----- Toolbar -----
Toolbar toolBar = (Toolbar) this.
    findViewById(R.id.toolbar);
this.setSupportActionBar(toolBar);
toolBar.setTitle(R.string.app_name);
toolBar.setSubtitle("CardView");
toolBar.setTitleTextColor(Color.WHITE);
toolBar.setSubtitleTextColor(Color.WHITE);

// ----- CollapsingToolbarLayout text title colors
CollapsingToolbarLayout collapsingToolbarLayout =
    (CollapsingToolbarLayout) this.
        findViewById(R.id.main_collapsing);
collapsingToolbarLayout.
    setExpandedTitleColor(Color.WHITE);
collapsingToolbarLayout.
    setCollapsedTitleTextColor(Color.WHITE);
}
```

Исходный код рассмотренного примера, который изображен на Рис. 2.27 находится в модуле «app9» среди файлов с исходными кодами, которые прилагаются к данному уроку.

2.9. Создание карточек. Виджет CardView

Виджет [android.support.v7.widget.CardView](#) является производным классом от класса [android.widget.FrameLayout](#) и предназначен для отображения информации внутри карточек. Виджеты [android.support.v7.widget.CardView](#) могут отбрасывать тени и иметь закруглен-

ные углы. Иерархия классов для класса `android.support.v7.widget.CardView` выглядит следующим образом:

```
java.lang.Object
|
+--- android.view.View
|
+--- android.view.ViewGroup
|
+--- android.widget.FrameLayout
|
+--- android.support.v7.widget.CardView
```

Карточки `android.support.v7.widget.CardView` предназначены в первую очередь для использования внутри списков `android.support.v7.widget.RecyclerView` (рассматривается ниже в данном уроке). Однако, карточки можно использовать и по отдельности, аналогично использованию контейнера `android.widget.FrameLayout`. Конечно же, карточки применимы не во всех ситуациях, но отлично подходят при отображении какого-то элемента, состоящего из картинки, заголовка и небольшой вводной информации. Многие популярные приложения активно используют `android.support.v7.widget.CardView`.

Виджет `android.support.v7.widget.CardView` входит во вспомогательные библиотеки v7. Чтобы использовать этот виджет в своем проекте, добавьте в модуль приложения необходимые зависимости Gradle (файл `build.gradle` для модуля) как это показано в Листинге 2.45.

Листинг 2.45. Добавление зависимостей в файл build.gradle модуля для возможности использования виджетов android.support.v7.widget.CardView и android.support.v7.widget.RecyclerView

```
dependencies
{
    ...
    compile 'com.android.support:cardview-v7: 23.0.+'
    compile 'com.android.support:recyclerview-v7: 23.0.+'
}
```

Синтаксис создания виджета android.support.v7.widget.CardView показан в Листинге 2.46.

Листинг 2.46. Синтаксис создания виджета android.support.v7.widget.CardView в xml файлах разметки

```
<android.support.v7.widget.CardView
    xmlns:card_view=
        "http://schemas.android.com/apk/res-auto"
    android:id="@+id/card_view"
    android:layout_gravity="center"
    android:layout_width="200dp"
    android:layout_height="200dp"
    card_view:cardCornerRadius="4dp">

    <!-- Contents here -->

</android.support.v7.widget.CardView>
```

У виджета **android.support.v7.widget.CardView** есть несколько атрибутов:

- **card_view:cardCornerRadius** — Задает величину за кругления углов карточки.

- **card_view:contentPadding** — Задает величину внутреннего отступа от границ карточки.
- **card_view:cardElevation** — Задает величину тени.

Для знакомства с виджетом **android.support.v7.widget.CardView** создадим модуль «app10» в проекте, который прилагается к данному уроку.

Структура файла макета Активности для модуля «app10» аналогична структуре макета из Листинга 2.42. В макете Активности модуля «app10» размещены следующие контейнеры: **CoordinatorLayout** (в качестве корневого элемента), **AppBarLayout** и **CollapsingToolbarLayout** для размещения в них тулбара **android.support.v7.widget.Toolbar** с эффектами сворачивания и разворачивания при прокрутке, а также контейнер **NestedScrollView** в котором и будет находиться виджет **android.supprt.v7.widget.CardView**, как показано в Листинге 2.47.

Листинг 2.47. Файл макета внешнего вида Активности рассматриваемого в данном разделе примера

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:app=
        "http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"

    android:layout_width="match_parent"
    android:layout_height="match_parent"

    android:fitsSystemWindows="true"
    android:id="@+id/clMain"
```

```
    android:background="#E0E0E0"
    tools:context="itstep.com.myapp10.MainActivity">
    ...
<android.support.v4.widget.NestedScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior=
        "@string/appbar_scrolling_view_behavior">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <android.support.v7.widget.CardView
            xmlns:card_view=
                "http://schemas.android.com/apk/res-auto"
            android:id="@+id/card_view"
            android:layout_width="match_parent"
            android:layout_height="150dp"
            android:layout_margin="16dp"
            card_view:contentPadding="8dp"
            card_view:cardCornerRadius="4dp">

            <ImageView
                android:layout_width="wrap_content"
                android:layout_height=
                    "wrap_content"
                android:layout_gravity=
                    "center_vertical"
                android:src="@drawable/user_man" />

            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:orientation="vertical">
```

```
        android:layout_gravity=
                "center_vertical">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="27sp"
            android:textColor="#003366"
            android:text="William Blake"
            />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="19sp"
            android:text="Android programmer"
            />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="15sp"
            android:text="25 years old"
            />

        </LinearLayout>
    </LinearLayout>
</android.support.v7.widget.CardView>

</android.support.v4.widget.NestedScrollView>

...
</android.support.design.widget.CoordinatorLayout>
```

Внешний вид виджета **android.support.v7.widget.CardView** из Листинга 2.47 изображен на Рис. 2.28.

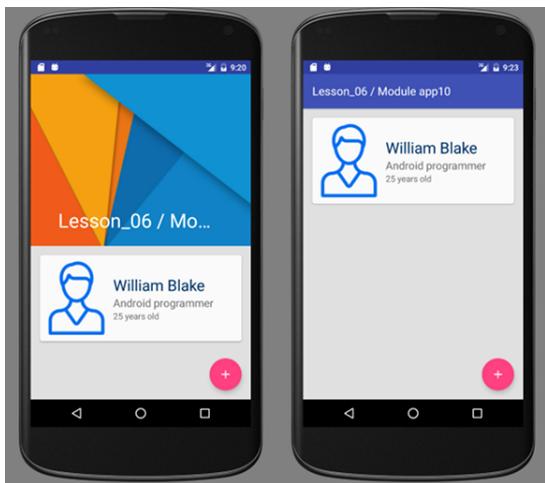


Рис. 2.28. Внешний вид работы примера из Листинга 2.47

Исходный код рассмотренного в данном разделе примера находится в модуле «app10» среди файлов с исходными кодами, которые прилагаются к данному уроку.

2.10. Список RecyclerView

Виджет [`android.support.v7.widget.RecyclerView`](#) представляет собой расширенную и более мощную версию виджета [`android.widget.ListView`](#), однако он не является каким-либо наследником [`android.widget.ListView`](#). Он предназначен для отображения элементов из набора данных, каждый элемент которых содержит несколько значений. Часто, в качестве контейнера для каждого элемента списка `RecyclerView` используется виджет [`android.support.v7.widget.CardView`](#). Виджет [`android.support.v7.widget.RecyclerView`](#) предназначен для использования внутри контейнера [`CoordinatorLayout`](#), а значит, предоставляет последнему функциональные возможности по

координации зависимостей между своими дочерними виджетами. Также, виджет **android.support.v7.widget.RecyclerView** рекомендуется использовать в случаях, когда имеются коллекции данных, элементы которых изменяются во время выполнения в зависимости от действий пользователя или сетевых событий. Виджет **android.support.v7.widget.RecyclerView** входит во вспомогательные библиотеки v7. Чтобы использовать этот виджет в своем проекте, добавьте в модуль приложения необходимые зависимости Gradle (файл build.gradle для модуля) как это показано в Листинге 2.45. Иерархия классов для класса **android.support.v7.widget.RecyclerView** выглядит следующим образом:

```
java.lang.Object
  |
  +--- android.view.View
    |
    +--- android.view.ViewGroup
      |
      +--- android.support.v7.widget.
          RecyclerView
```

Виджет **android.support.v7.widget.RecyclerView**, также как и виджет **android.widget.ListView**, получает данные из Адаптера Данных. Для создания Адаптера Данных необходимо создать класс, производный от класса **android.support.v7.widget.RecyclerView.Adapter**.

Еще, в отличии от виджета **android.widget.ListView**, виджет **android.support.v7.widget.RecyclerView** позволяет настраивать внешний вид расположения элементов

списка. Это достигается за счет внутреннего Менеджера Раскладки (*Layout Manager*), для которого существуют готовые специальные классы менеджеров раскладок:

- [android.support.v7.widget.LinearLayoutManager](#) для отображения элементов в виде списка с вертикальной или горизонтальной прокруткой;
- [android.support.v7.widget.GridLayoutManager](#) для отображения элементов в виде сетки;
- [android.support.v7.widget.StaggeredGridLayoutManager](#) для отображения элементов в виде шахматной сетки;

Также существует возможность создания своих собственных менеджеров раскладок — для этого необходимо создать производный класс от класса [android.support.v7.widget.RecyclerView.LayoutManager](#).

Схема взаимодействия виджета [android.support.v7.widget.RecyclerView](#) с Адаптером Данных и Менеджером Раскладки изображена на Рис. 2.29.



Рис. 2.29. Схема взаимодействия виджета [android.support.v7.widget.RecyclerView](#) с Адаптером Данных и встроенным Менеджером Раскладки

Давайте познакомимся с классом [android.support.v7.widget.RecyclerView.Adapter](#). Иерархия классов для него выглядит следующим образом:

```
java.lang.Object
|
+-- android.support.v7.widget.
    RecyclerView.Adapter<VH
        extends android.support.v7.
        widget.RecyclerView.ViewHolder>
```

Как видно из иерархии, при переопределении класса **android.support.v7.widget.RecyclerView.Adapter** необходимо еще создать класс **android.support.v7.RecyclerView.ViewHolder**. Класс **android.support.v7.RecyclerView.ViewHolder** предназначен для визуального представления каждого элемента списка **android.support.v7.widget.RecyclerView**. Другими словами, класс **android.support.v7.RecyclerView.ViewHolder** представляет собой контейнер для размещения элементов списка **RecyclerView**.

Рассмотрим создание производного класса от класса **android.support.v7.RecyclerView.ViewHolder** на примере. Для этого создан модуль «app11» в проекте, который прилагается к данному уроку.

Предположим, мы создаем список сотрудников для размещения внутри виджета **android.support.v7.widget.RecyclerView**. У каждого сотрудника есть имя, должность, возраст и пол. Класс сотрудника (**Person**) показан в Листинге 2.48.

Листинг 2.48. Класс Person рассматриваемого примера

```
class Person
{
    /**
     * Person name
     *
```

```
* Имя человека
*/
String name;

/**
 * Person speciality
 *
 * Специальность
 */
String speciality;

/**
 * Person age
 * Возраст
 */
int age;

/**
 * Person's gender
 *
 * Пол человека
 */
boolean gender;
public Person(String name, String speciality,
              int age, boolean gender)
{
    this.name = name;
    this.speciality = speciality;
    this.age = age;
    this.gender = gender;
}
}
```

Далее, каждый элемента списка **android.support.v7.widget.RecyclerView** будет отображаться в наборе виджетов, который описывается в файле ресурсов `/res/layout/`

my_recycler_item.xml, содержимое которого приведено в Листинге 2.49.

Листинг 2.49. Содержимое файла /res/layout/my_recycler_item.xml который содержит набор виджетов для отображения каждого элемента списка сотрудников, каждый из которых представлен объектом Person

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

<android.support.v7.widget.CardView
    xmlns:android=
        "http://schemas.android.com/apk/res/android"

    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/card_view"
    android:layout_width="match_parent"
    android:layout_height="140dp"
    android:layout_marginLeft="16dp"
    android:layout_marginRight="16dp"
    android:layout_marginTop="8dp"

    app:cardCornerRadius="4dp">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal">

        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_vertical"
            android:id="@+id/ivAvatar"
            />

        <LinearLayout
            android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:layout_gravity="center_vertical">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="27sp"
        android:textColor="#003366"
        android:id="@+id/tvName"
    />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="19sp"
        android:id="@+id/tvSpeciality"
    />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="15sp"
        android:id="@+id/tvAge"
    />

</LinearLayout>

</LinearLayout>

</android.support.v7.widget.CardView>
```

Внешний вид макета из Листинга 2.49 изображен на Рис. 2.30. Обратите внимание, что элементы списка **RecyclerView** будут отображаться в карточках — в видах **android.support.v7.widget.CardView**.



Рис. 2.30. Внешний вид макета из Листинга 2.49

Так вот, производный класс `android.support.v7.widget.RecyclerView.ViewHolder` для модели данных представленной классом `Person` из Листинг 2.48 и макета внешнего вида из Листинга 2.49 будет выглядеть как показано в Листинге 2.50.

Листинг 2.50. Производный класс от класса `android.support.v7.widget.RecyclerView.ViewHolder` для макета из Листинга 2.49 для списка объектов `Person`

```
class PersonViewHolder extends RecyclerView.ViewHolder
{
    /**
     * Widget in which the name of the Person is
     * displayed
     * Виджет в котором отображается имя сотрудника
     */
    public TextView tvName;

    /**
     * Widget in which the specialty of the Person
     * is displayed
     * Виджет в котором отображается специальность
     * сотрудника
     */
    public TextView tvSpeciality;

    /**

```

```
* Widget in which the age of the employee is
* displayed
* Виджет в котором отображается возраст
* сотрудника
*/
public TextView tvAge;

/**
* Widget in which the avatar for the Person's
* gender is displayed
* Виджет в котором отображается аватар для
* пола сотрудника
*/
public ImageView ivAvatar;

public PersonViewHolder(View v)
{
    super(v);
    this.tvName = (TextView)
        v.findViewById(R.id.tvName);
    this.tvSpeciality =
        (TextView)
        v.findViewById(R.id.tvSpeciality);
    this.tvAge = (TextView)
        v.findViewById(R.id.tvAge);
    this.ivAvatar = (ImageView)
        v.findViewById(R.id.ivAvatar);
}
```

Как видно из Листинга 2.50, виджеты, в которых будут отображаться данные сотрудника, объявлены public полями. Это сделано для удобства доступа к ним Адаптера Данных при размещении информации о сотруднике.

Теперь вернемся к Адаптеру Данных (классу `android.support.v7.widget.RecyclerView.Adapter`). При создании производного класса от класса `android.support.v7.widget.RecyclerView.Adapter` необходимо переопределить следующие методы:

- `VH onCreateViewHolder(ViewGroup parent, int viewType)` — Метод вызывается, когда виджету `android.support.v7.widget.RecyclerView` необходим новый контейнер `RecyclerView.ViewHolder` для представления элемента списка. Параметр `parent` ссылается на список `RecyclerView`, а параметр `viewType` содержит информацию о типе создаваемого контейнера `RecyclerView.ViewHolder`. Наличие параметра `viewType` позволяет использовать виджет `RecyclerView` с разными контейнерами для разных элементов списка, что является довольно гибкой возможностью виджета `RecyclerView`. Параметр `viewType` задается при помощи переопределения метода `int getItemViewType (int position)`, где `position` это индекс элемента в наборе данных Адаптера Данных.
- `void onBindViewHolder(ViewHolder holder, int position)` — Метод вызывается виджетом `RecyclerView` для отображения элемента с индексом `position` из набора данных Адаптера Данных. Параметр `holder` ссылается на контейнер производного класса от класса `RecyclerView.ViewHolder` (см. Листинг 2.50). Этот метод должен обновить содержимое контейнера `holder`, чтобы отобразить данные элемента с индексом `position` из набора данных.

- **public int getItemCount()** — Метод вызывается, когда виджету **RecyclerView** необходимо узнать количество элементов в наборе данных Адаптера Данных.

В рассматриваемом примере (модуль «app11») создадим класс, производный от класса **android.support.v7.widget.RecyclerView.Adapter** который будет хранить в себе список сотрудников (коллекцию объектов **Person**) и заполнять данными контейнеры из Листинга 2.50 информацией о этих сотрудниках. Класс Адаптера Данных назовем **MyRecyclerAdapter**. Содержимое этого класса приведено в Листинге 2.51.

Листинг 2.51. Класс Адаптера Данных для списка android.support.v7.widget.RecyclerView рассматриваемого примера

```
class MyRecyclerAdapter extends RecyclerView.  
Adapter<RecyclerView.ViewHolder>  
{  
    /**  
     * Data Adapter Data Set - Person List  
     *  
     * Набор данных Адаптера Данных - список сотрудников  
     */  
    private ArrayList<Person> dataSet;  
  
    /**  
     * Reference to application context  
     *  
     * Ссылка на контекст приложения  
     */  
    private Context context;  
  
    // ----- Inner classes -----
```

```
public class ViewHolder extends RecyclerView.  
    ViewHolder  
{  
    /**  
     * Widget in which the name of the Person is  
     * displayed  
     *  
     * Виджет в котором отображается имя сотрудника  
     */  
    public TextView tvName;  
  
    /**  
     * Widget in which the specialty of the Person  
     * is displayed  
     *  
     * Виджет в котором отображается специальность  
     * сотрудника  
     */  
  
    public TextView tvSpeciality;  
  
    /**  
     * Widget in which the age of the employee is  
     * displayed  
     *  
     * Виджет в котором отображается возраст сотрудника  
     */  
    public TextView tvAge;  
  
    /**  
     * Widget in which the avatar for the Person's  
     * gender is displayed  
     *  
     * Виджет в котором отображается аватар для  
     * пола сотрудника  
     */  
}
```

```
public ImageView ivAvatar;
public ViewHolder(View v)
{
    super(v);
    this.tvName = (TextView)
        v.findViewById(R.id.tvName);
    this.tvSpeciality = (TextView)
        v.findViewById(R.id.tvSpeciality);
    this.tvAge = (TextView)
        v.findViewById(R.id.tvAge);
    this.ivAvatar = (ImageView)
        v.findViewById(R.id.ivAvatar);
}
}

// ----- Class methods -----
public MyRecyclerAdapter(Context context,
    ArrayList<Person> dataSet)
{
    this.context = context;
    this.dataSet = dataSet;
}

@Override
public MyRecyclerAdapter.ViewHolder onCreateViewHolder(
    ViewGroup parent, int viewType)
{
    View v = LayoutInflater.from(parent.getContext())
        .inflate(R.layout.my_recycler_item,
            parent, false);
    ViewHolder vh = new ViewHolder(v);
    return vh;
}

@Override
public void onBindViewHolder(ViewHolder holder,
    int position)
```

```
{  
// ----- Get a reference to the current element  
// ----- from the data set -----  
// ----- Получаем ссылку на текущий элемент из  
// ----- набора данных -----  
    Person person = this.dataSet.get(position);  
  
// ----- Fill container widgets with data -----  
// ----- Заполняем виджеты контейнера данными -----  
    holder.tvName.setText(person.name);  
    holder.tvSpeciality.setText(person.speciality);  
    String age = String.valueOf(person.age) +  
        " years old";  
    holder.tvAge.setText(age);  
    holder.ivAvatar.setImageDrawable(  
        this.context.getResources().  
        getDrawable(((person.gender) ?  
            R.drawable.user_man: R.drawable.  
            user_woman), this.context.getTheme()));  
}  
  
@Override  
  
public int getItemCount()  
{  
    return this.dataSet.size();  
}  
}
```

Как видно из Листинга 2.51, производный от класса **RecyclerView.ViewHolder** класс для контейнера каждого элемента списка сделан в виде вложенного класса в класс Адаптера Данных **MyRecyclerAdapter**. Это сделано для удобства программирования.

Разместим в макете Активности (файл /res/layout/activity_main.xml) виджет **android.support.v7.widget.RecyclerView** (см. Листинг 2.52).

Листинг 2.52. Xml верстка виджета android.support.v7.widget.RecyclerView в файле макета Активности /res/layout/activity_main.xml

```
<android.support.v7.widget.RecyclerView  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:id="@+id/rvPersons"  
/>
```

Создание Адаптера Данных (объекта **MyRecyclerView** из Листинга 2.51) и заполнение его данными происходит в методе **onCreate()** класса Активности (класс **MainActivity** модуля «app11»). Содержимое этого класса **MainActivity** показано в Листинге 2.53.

Листинг 2.53. Содержимое класса MainActivity рассматриваемого в данном разделе примера

```
public class MainActivity extends AppCompatActivity  
{  
    // ----- Class members -----  
    private RecyclerView recyclerView;  
    private RecyclerView.Adapter adapter;  
    private RecyclerView.LayoutManager layoutManager;  
  
    // ----- Class methods -----  
    @Override  
    protected void onCreate(Bundle savedInstanceState)  
    {  
        super.onCreate(savedInstanceState);  
    }
```

```
setContentView(R.layout.activity_main);
// ----- Toolbar -----
Toolbar toolBar = (Toolbar)
    this.findViewById(R.id.toolbar);
this.setSupportActionBar(toolBar);
toolBar.setTitle(R.string.app_name);
toolBar.setSubtitle("RecyclerView");
toolBar.setTitleTextColor(Color.WHITE);
toolBar.setSubtitleTextColor(Color.WHITE);

// ----- CollapsingToolbarLayout text title colors
CollapsingToolbarLayout collapsingToolbarLayout =
    (CollapsingToolbarLayout)
        this.findViewById(R.id.main_collapsing);
collapsingToolbarLayout.setExpandedTitleColor(
    Color.rgb(0x00, 0x33, 0x66));
collapsingToolbarLayout.
    setCollapsedTitleTextColor(Color.WHITE);

// ----- RecyclerView -----
this.recyclerView = (RecyclerView)
    findViewById(R.id.rvPersons);

// ----- Use this setting to improve performance
// if you know that changes in content do not
// change the layout size of the RecyclerView -----
//
// ----- Используйте параметр true для повышения
// производительности, если вы знаете, что
// изменения в содержании элемента данных
// не меняют размер макета RecyclerView -----
    this.recyclerView.setHasFixedSize(true);

// ----- Use a linear layout manager for RecyclerView
// ----- Используем LinearLayoutManager для списка
// RecyclerView -----
```

```
    this.layoutManager =
        new LinearLayoutManager(this);
    this.recyclerView.setLayoutManager(this.
        layoutManager);

// ----- Creating a data set for a Data Adapter
// ----- Создание набора данных для Адаптера Данных

ArrayList<Person> list = new ArrayList<>();
list.add(new Person("William Blake",
    "Android programmer", 25, true));

list.add(new Person("Sarah White",
    "Selling manager", 29, false));

list.add(new Person("Steven Brown",
    "PHP programmer", 21, true));

list.add(new Person("John Orange",
    "Web designer", 32, true));

list.add(new Person("Emma Lynch",
    "Advertising manager", 27, false));

list.add(new Person("Susanna Peach",
    "Insurance agent", 24, false));

// ----- Assigning the Data Adapter to the
// RecyclerView List -----
// -----
// ----- Назначение Адаптера Данных списку
// RecyclerView -----
this.adapter = new MyRecyclerAdapter(this, list);
this.recyclerView.setAdapter(this.adapter);
}

}
```

Внешний вид работы примера из Листингов 2.48, 2.49, 2.51 и 2.53 изображен на Рис. 2.31.

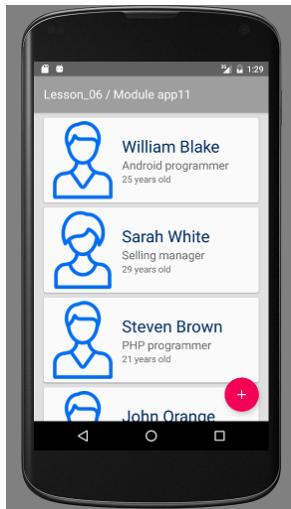


Рис. 2.31. Внешний вид работы примера из Листингов 2.48, 2.49, 2.51 и 2.53

Конечно же, виджет `android.support.v7.RecyclerView`.
`Adapter` интересен как виджет, предназначенный для использования внутри контейнера `android.support.design.widget.CoordinatorLayout`. Давайте сверстаем внешний вид макета Активности таким образом, чтобы в макете присутствовали контейнеры и виджеты из библиотек совместимости: `CoordinatorLayout`, `AppBarLayout`, `CollapsingToolbarLayout`, `Toolbar`, `FloatingActionButton`. Как вы уже наверное догадались, мы хотим увидеть (то есть продемонстрировать вам) поведение сворачивания панели инструментов при прокрутке как в списке `RecyclerView` так и при прокрутке самого списка `RecyclerView` внутри `CoordinatorLayout`. Содержимое файла `/res/layout/activity_main.xml` с макетом внешнего вида Активности показано в Листинге 2.54.

Листинг 2.54. Содержимое файла /res/layout/activity_main.xml с версткой макета внешнего вида Активности рассматриваемого в данном разделе примера

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"

    android:layout_width="match_parent"
    android:layout_height="match_parent"

    android:fitsSystemWindows="true"
    android:id="@+id/clMain"
    android:background="#E0E0E0"
    tools:context="itstep.com.myapp11.MainActivity">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <android.support.design.widget.
            CollapsingToolbarLayout
            android:id="@+id/main_collapsing"
            android:layout_width="match_parent"

            android:layout_height="300dp"
            app:layout_scrollFlags=
                "scroll|exitUntilCollapsed"
            android:fitsSystemWindows="true"
            app:contentScrim="?attr/colorPrimary"
            app:expandedTitleMarginStart="48dp"
            app:expandedTitleMarginEnd="64dp">

            <ImageView
                android:id="@+id/main.backdrop"
```

```
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="centerCrop"
        android:fitsSystemWindows="true"
        android:src="@drawable/material_background"
        app:layout_collapseMode="parallax"
    />

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="60dp"
        app:popupTheme=
            "@style/ThemeOverlay.AppCompat.Light"
        app:layout_collapseMode="pin"
    />
</android.support.design.widget.
    CollapsingToolbarLayout>

</android.support.design.widget.AppBarLayout>
<android.support.v7.widget.RecyclerView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/rvPersons"
    app:layout_behavior=
        "@string/appbar_scrolling_view_behavior"
/>

<android.support.design.widget.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|right"
    android:layout_marginBottom=
        "@dimen/activity_vertical_margin"
```

```
        android:layout_marginRight=
            "@dimen/activity_horizontal_margin"
        android:src="@drawable/plus"
    />

</android.support.design.widget.CoordinatorLayout>
```

Как видно из Листинга 2.54, виджет **android.support.v7.widget.RecyclerView** помещен в качестве дочернего элемента в контейнер **android.support.design.widget.CoordinatorLayout**. В Листинге 2.54 этот фрагмент кода выделен жирным шрифтом. Виджету **android.support.v7.widget.RecyclerView** назначено поведение **AppBarLayout.ScrollingViewBehavior**. Такое же поведение мы назначали в предыдущих примерах контейнеру **android.support.v4.widget.NestedScrollView**. Это поведение сообщает родительскому **CoordinatorLayout** о событиях прокрутки виджета, что побуждает контейнер **CoordinatorLayout** приступить к координации зависимостей между своими дочерними виджетами.

Внешний вид работы примера из Листингов 2.48, 2.49, 2.51 и 2.53 с учетом макета Активности из Листинга 2.54 изображен на Рис. 2.32.

Как видно из Рис. 2.32, виджет **android.support.v7.widget.RecyclerView** очень гармонично сочетается с использованием внутри менеджера **android.support.design.widget.CoodinatorLayout**.

Исходный код рассмотренного в данном разделе примера находится в модуле «app11» среди файлов с исходными кодами, которые прилагаются к данному уроку.

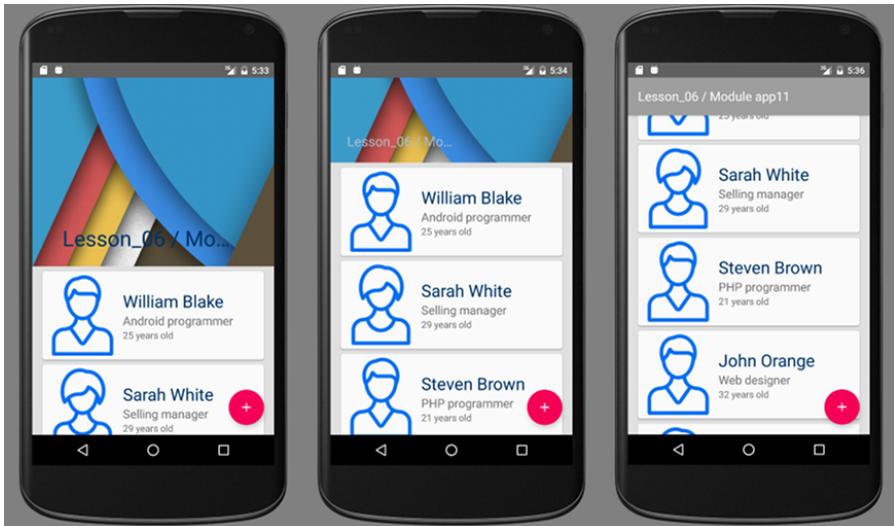


Рис. 2.32. Внешний вид работы примера из Листингов 2.48, 2.49, 2.51, 2.53 и 2.54. Осуществляется прокрутка содержимого виджета RecyclerView

2.11. Контейнер ViewPager

Менеджер раскладки [android.support.v4.view.ViewPager](#) позволяет пользователю на Активности перелистывать влево и вправо страницы с данными (см. Рис. 2.33). В качестве страниц с данными выступают Фрагменты (объекты [android.support.v4.app.Fragment](#)), а поставщиком этих страниц выступает объект класса [android.support.v4.view.PagerAdapter](#). Использование Фрагментов в качестве страниц позволяет удобно управлять жизненным циклом этих страниц. А также, с точки зрения программирования, позволяет распределить функциональность одного приложения между разными классами, поскольку одна страница представляется одним Фрагментом.

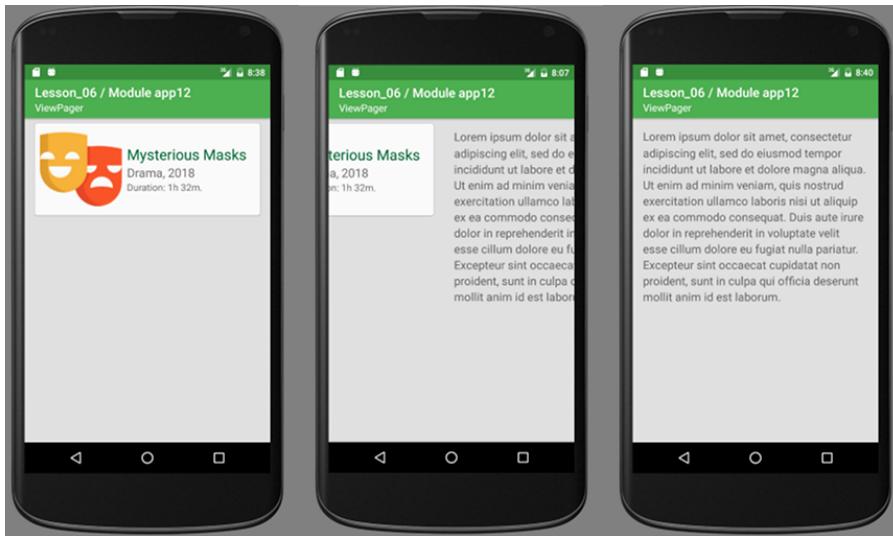


Рис. 2.33. Перелистывание страниц при помощи контейнера `android.support.v4.view.ViewPager`

Иерархия классов для класса `android.support.v4.view.ViewPager` имеет следующий вид:

```
java.lang.Object
|
+--- android.view.View
|
+--- android.view.ViewGroup
|
+--- android.support.v4.
    view.ViewPager
```

Как видно из иерархии классов, класс `android.support.v4.view.ViewPager` наследуется только от абстрактного класса `android.view.ViewGroup`, то есть, не похож ни на один из известных нам менеджеров раскладки. Что касается класса для поставщика страниц `android.support.v4.view.PagerAdapter`, то этот класс является абстрактным

классом, у которого есть два производных класса, которые являются его стандартными реализациями: абстрактный класс [android.support.v4.app.FragmentPagerAdapter](#) и абстрактный класс [android.support.v4.app.FragmentStatePagerAdapter](#). Эти классы охватывают наиболее распространенные варианты использования Фрагментов для [android.support.v4.view.ViewPager](#). Однако, поскольку эти оба класса также являются абстрактными, то для создания объекта поставщика страниц необходимо будет вначале создать производный класс.

Класс [android.support.v4.app.FragmentPagerAdapter](#) представляет каждую страницу как Фрагмент, который постоянно хранится в менеджере фрагментов, до тех пор, пока пользователь может вернуться на эту страницу. Этот вариант поставщика страниц лучше всего использовать, когда есть несколько типичных статических Фрагментов, которые нужно перелистывать. Примером таких страниц является набор вкладок. При использовании [android.support.v4.app.FragmentPagerAdapter](#), Фрагмент каждой страницы, которую посещает пользователь, будет храниться в памяти, хотя его иерархия виджетов может быть уничтожена, если она не отображается на экране. Это может привести к использованию значительного объема памяти, что не всегда является допустимым. Для более крупных наборов страниц рекомендуется использовать поставщик страниц [android.support.v4.app.FragmentStatePagerAdapter](#). При создании класса, производного от класса [android.support.v4.app.FragmentPagerAdapter](#), необходимо переопределить два метода:

- **android.support.v4.app.Fragment getItem (int position)** — Метод возвращает Фрагмент, который соответствует странице с индексом **position**.
- **int getCount ()** — Метод возвращает количество доступных страниц.

Класс **android.support.v4.app.FragmentStatePagerAdapter** также представляет каждую страницу как Фрагмент. Этот класс также поддерживает сохранение и восстановление состояния каждого Фрагмента. Этот вариант версия поставщика страниц более полезен, когда имеется большое количество страниц, похожих на элементы списка. Когда страницы не видны пользователю, Фрагменты, которые их представляют, могут быть уничтожены (удалены из памяти), при этом в памяти остаются сохраненные состояния этих Фрагментов. Это позволяет поставщику страниц **android.support.v4.app.FragmentStatePagerAdapter** занимать гораздо меньшую память для своих страниц, чем в случае использования поставщика **android.support.v4.app.FragmentPagerAdapter**. При создании класса, производного от класса **android.support.v4.app.FragmentStatePagerAdapter**, необходимо переопределить два метода:

- **android.support.v4.app.Fragment getItem (int position)** — Метод возвращает Фрагмент, который соответствует странице с индексом **position**.
- **int getCount ()** — Метод возвращает количество доступных страниц.

После создания класса поставщика страниц, объект этого класса назначается контейнеру **android.support.**

v4.view.ViewPager при помощи метода `setAdapter()`, как показано в Листинге 2.55.

Листинг 2.55. Создание объекта поставщика страниц и назначение его контейнеру `android.support.v4.view.ViewPager`

```
ViewPager viewPager = (ViewPager)
    this.findViewById(R.id.viewpager);
MyFragmentPagerAdapter adapter =
    new MyFragmentPagerAdapter(this.
        getSupportFragmentManager());
viewPager.setAdapter(adapter);
```

Давайте рассмотрим на примере использование контейнера **android.support.v4.view.ViewPager**. Для этого создадим модуль «app12» в проекте Android Studio, который прилагается к данному уроку.

Содержимое файла с макетом внешнего вида Активности (`/res/layout/activity_main.xml`), который добавлен контейнер **android.support.v4.view.ViewPager** приведен в Листинге 2.56.

Листинг 2.56. Файл макета внешнего вида Активности (`/res/layout/activity_main.xml`) рассматриваемого в данном разделе примера

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    android:id="@+id/clMain"
    android:background="#E0E0E0"
    tools:context="itstep.com.myapp12.MainActivity">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="60dp"
            app:popupTheme=
                "@style/ThemeOverlay.AppCompat.Light"
            app:layout_collapseMode="pin"
        />

    </android.support.design.widget.AppBarLayout>

    <android.support.v4.view.ViewPager
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior=
            "@string/appbar_scrolling_view_behavior"
        android:id="@+id/vpContent">
    </android.support.v4.view.ViewPager>
</android.support.design.widget.CoordinatorLayout>
```

Как видно из Листинга 2.56, контейнер **android.support.v4.view.ViewPager** размещен внутри контейнера **android.support.design.widget.CoordinatorLayout** в качестве непосредственного дочернего виджета. Чтобы контейнер **android.support.v4.view.ViewPager** не «залез» под контейнер **android.support.design.widget.AppBarLayout**,

контейнеру `android.support.v4.view.ViewPager` назначено поведение в виде объекта класса `AppBarLayout.ScrollingViewBehavior`.

Далее, в рассматриваемом примере при помощи контейнера `android.support.v4.view.ViewPager` будут отображаться и листаться три страницы. Для каждой страницы создадим по Фрагменту (`android.support.v4.app.Fragment`). Класс этих Фрагментов будут называться `FragmentPageOne`, `FragmentPageTwo` и `FragmentPageThree` соответственно. Для каждого из этих Фрагментов также создадим файлы ресурсов с макетами внешнего вида. Названия этих файлов ресурсов следующие: `/res/layout/fragment_page_one.xml`, `/res/layout/fragment_page_two.xml`, `/res/layout/fragment_page_three.xml`.

Содержимое файлов ресурсов с макетами внешнего вида для страниц показано в Листинге 2.57.

Листинг 2.57. Содержимое файлов ресурсов с макетами внешнего вида для страниц контейнера `android.support.v4.view.ViewPager`

```
/res/layout/fragment_page_one.xml
-----
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.widget.NestedScrollView
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:app=
        "http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#E0E0E0">
```

```
<android.support.v7.widget.CardView
    android:id="@+id/card_view"
    android:layout_width="match_parent"
    android:layout_height="140dp"
    android:layout_marginLeft="16dp"
    android:layout_marginRight="16dp"
    android:layout_marginTop="8dp"
    app:contentPadding="8dp"
    app:cardCornerRadius="4dp">
    ...
</android.support.v7.widget.CardView>
</android.support.v4.widget.NestedScrollView>

/res/layout/fragment_page_two.xml
-----
<?xml version="1.0" encoding="utf-8"?>
<ScrollView
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:id="@+id/content"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView style="?android:textAppearanceMedium"
        android:padding="16dp"
        android:lineSpacingMultiplier="1.2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/lorem_ipsum" />
</ScrollView>

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

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/listView">

```

```
</ListView>
```

Как видно из Листинг 2.57, на первой странице будет отображена карточка **android.support.v7.widget.CardView**. На второй странице будет отображено текстовое поле **android.widget.TextView** с текстом «*lorem ipsum...*» который объявлен как строковая константа в файле ресурсов */res/values/strings.xml*. И на третьей странице будет отображаться список **android.widget.ListView** с какими-то данными. Внешний вид страниц изображен на Рис. 2.34.

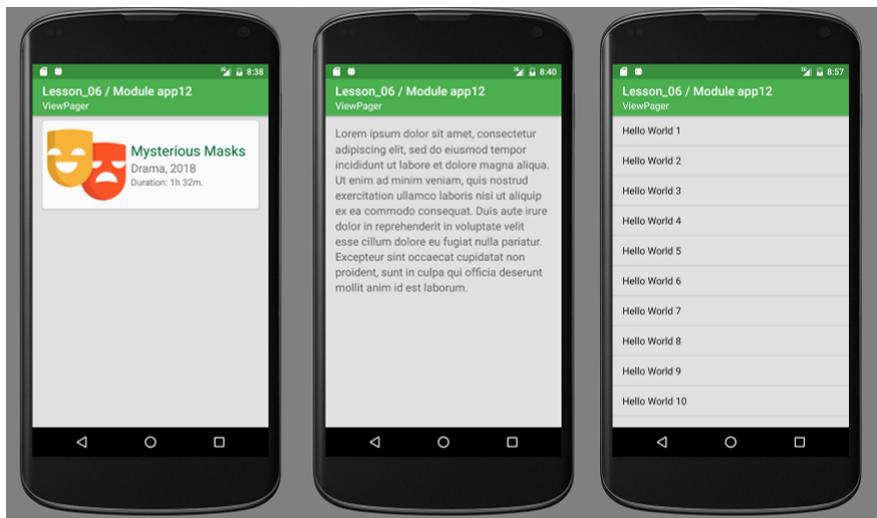


Рис. 2.34. Внешний вид страниц (слева направо) из файлов макетов: /res/layout/fragment_page_one.xml, /res/layout/fragment_page_two.xml, /res/layout/fragment_page_three.xml

Содержимое классов Фрагментов (**FragmentPageOne**, **FragmentPageTwo**, **FragmentPageThree**) показано в Листинге 2.58.

Листинг 2.58. Содержимое классов Фрагментов FragmentPageOne, FragmentPageTwo, FragmentPageThree для страниц контейнера android.support.v4.view.ViewPager

FragmentPageOne.java

```
-----  
public class FragmentPageOne    extends Fragment  
{  
    @Override  
    public View onCreateView(LayoutInflater inflater,  
                             ViewGroup container,  
                             Bundle savedInstanceState)  
    {  
        ViewGroup rootView = (ViewGroup)  
            inflater.inflate(  
                R.layout.fragment_page_one, container, false);  
        return rootView;  
    }  
}
```

FragmentPageTwo.java

```
-----  
public class FragmentPageTwo extends Fragment  
{  
    @Override  
    public View onCreateView(LayoutInflater inflater,  
                             ViewGroup container, Bundle savedInstanceState)  
    {  
        ViewGroup rootView = (ViewGroup) inflater.inflate(  
            R.layout.fragment_page_two, container, false);  
        return rootView;  
    }  
}
```

FragmentPageThree.java

```
public class FragmentPageThree extends Fragment
{
    @Override
    public View onCreateView(LayoutInflater inflater,
        ViewGroup container,
        Bundle savedInstanceState)
    {
        ViewGroup rootView = (ViewGroup) inflater.inflate(
            R.layout.fragment_page_three,
            container, false);
        ListView listView = (ListView) rootView.
            findViewById(R.id.listView);
        ArrayList<String> arrayList = new ArrayList<>();
        for (int i = 0; i < 50; i++)
        {
            arrayList.add("Hello World " + (i + 1));
        }
        ArrayAdapter<String> adapter =
            new ArrayAdapter<>(this.getActivity(),
                android.R.layout.simple_list_item_1,
                arrayList);
        listView.setAdapter(adapter);
        return rootView;
    }
}
```

Теперь, когда готовы классы Фрагментов для страниц, настало время создать класс для поставщика страниц. В рассматриваемом примере класс для поставщика страниц будет классом, производным от класса **android.support.v4.app.FragmentManagerAdapter**. Назовем этот производный класс **MyFragmentManagerAdapter**. Содержимое этого класса показано в Листинге 2.59.

Листинг 2.59. Класс поставщика страниц MyFragmentStatePagerAdapter для контейнера android.support.v4.view.ViewPager

```
class MyFragmentStatePagerAdapter extends
        FragmentStatePagerAdapter
{

    private Fragment[] pages =
    {
        new FragmentPageOne(),
        new FragmentPageTwo(),
        new FragmentPageThree()
    };

    public MyFragmentStatePagerAdapter(FragmentManager fm)
    {
        super(fm);
    }

    @Override
    public Fragment getItem(int position)
    {
        return this.pages[position];
    }

    @Override
    public int getCount()
    {
        return this.pages.length;
    }
}
```

Содержимое класса **MyFragmentStatePagerAdapter** из Листинга 2.59 достаточно простое и понятное — это класс который является коллекцией Фрагментов для страниц контейнера **android.support.v4.view.ViewPager**.

Теперь назначим объект поставщика страниц контейнеру `android.support.v4.view.ViewPager`. Сделаем это в методе `onCreate()` класса Активности `MainActivity`. Содержимое класса `MainActivity` показано в Листинге 2.60. Назначение контейнеру `android.support.v4.view.ViewPager` объекта поставщика страниц выделено в Листинге 2.60 жирным шрифтом.

Листинг 2.60. Содержимое класса Активности `MainActivity`, в методе `onCreate()` которого происходит назначение объекта поставщика страниц для контейнера `android.support.v4.view.ViewPager`

```
public class MainActivity extends AppCompatActivity
{
    // ----- Class members -----
    /**
     * The pager widget, which handles animation and
     * allows swiping horizontally to access previous
     * and next wizard steps.
     *
     * Контейнер ViewPager который отображает страницы
     * с возможностью перелистывания этих страниц
     * вперед и назад.
     */
    private ViewPager viewPager;

    /**
     * The pager adapter, which provides the pages
     * to the view pager widget.
     *
     * Поставщик страниц для контейнера ViewPager
     * (поле viewPager).
     */
    private PagerAdapter adapter;
```

```

// ----- Class methods -----
@Override

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

// ----- Toolbar -----
Toolbar toolBar = (Toolbar)
    this.findViewById(R.id.toolbar);
    this.setSupportActionBar(toolBar);
toolBar.setTitle(R.string.app_name);
toolBar.setSubtitle("ViewPager");
toolBar.setTitleTextColor(Color.WHITE);
toolBar.setSubtitleTextColor(Color.WHITE);

// ----- PagerAdapter -----
this.viewPager = (ViewPager)
    this.findViewById(R.id.vpContent);
this.adapter = new MyFragmentStatePagerAdapter(
    this.getSupportFragmentManager());
this.viewPager.setAdapter(this.adapter);
}

}

```

Внешний вид работы примера из Листингов 2.56, 2.57, 2.58, 2.59, 2.60 изображен на Рис. 2.33 и 2.34.

Добавим, что для контейнера `android.support.v4.view.ViewPager` есть специальные классы индикаторы страниц:

- [`android.support.v4.view.PagerTitleStrip`](#) () — не интерактивный индикатор текущей, следующей и предыдущей страниц контейнера `android.support.v4.view.ViewPager`.

ViewPager. Этот индикатор только отображает информацию не позволяя при этом пользователю взаимодействовать с ним.

- **android.support.v4.view.PagerTabStrip** — интерактивный индикатор текущей, следующей и предыдущей страниц контейнера **android.support.v4.view.ViewPager.** Этот индикатор позволяет пользователю при помощи нажатий осуществлять переход к следующей или предыдущей странице.

Любой из этих индикаторов добавляются в качестве дочернего элемента в контейнер **android.support.v4.view.ViewPager.** Как это делается, показано в Листингах 2.61 и 2.62.

Листинг 2.61. Пример добавления индикатора `android.support.v4.view.PagerTitleStrip` в контейнер `android.support.v4.view.ViewPager` для индикации страниц

```
<android.support.v4.view.ViewPager
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.v4.view.PagerTitleStrip
        android:id="@+id/pager_title_strip"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="top"
        android:background="#D0D0D0"
        android:textColor="#101010"
        android:paddingTop="5dp"
        android:paddingBottom="5dp" />
</android.support.v4.view.ViewPager>
```

Листинг 2.62. Пример добавления индикатора android.support.v4.view.PagerTabStrip в контейнер android.support.v4.view.ViewPager для индикации страниц

```
<android.support.v4.view.ViewPager  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    app:layout_behavior=  
        "@string/appbar_scrolling_view_behavior"  
    android:id="@+id/vpContent">  
  
<android.support.v4.view.PagerTabStrip  
  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:background="#D0D0D0"  
    android:id="@+id/pager_tab_strip"  
    android:layout_gravity="top" />  
  
</android.support.v4.view.ViewPager>
```

После добавления любого из индикаторов в качестве дочернего элемента в контейнер **android.support.v4.view.ViewPager**, необходимо в классе поставщика страниц переопределить метод

```
public CharSequence getPageTitle(int position)
```

который возвращает название страницы по индексу этой страницы, который передается в параметре **position**.

Добавим в наш класс поставщика страниц **MyFragmentStatePagerAdapter** из Листинга 2.59 метод **getPageTitle()**, как это показано в Листинге 2.63.

Листинг 2.63. Переопределение в классе поставщика страниц метода getPageTitle() для получения названия страниц, которые будут отображаться индикатором страниц

```
class MyFragmentStatePagerAdapter extends
    FragmentStatePagerAdapter
{
    ...
    @Override
    public CharSequence getPageTitle(int position)
    {
        switch (position)
        {
            case 0:
                return "One";
            case 1:
                return "Two";
            case 2:
                return "Three";
            default:
                return "";
        }
    }
}
```

В файл макета внешнего вида Активности добавим в контейнер **android.support.v4.view.ViewPager** в качестве дочернего элемента индикатор **android.support.v4.view.PagerTabStrip**, как это показано в Листинге 2.62. Результат работы примера из модуля «app12» с индикатором **android.support.v4.view.PagerTabStrip** (Листинг 2.62 и 2.63) изображен на Рис. 2.35.

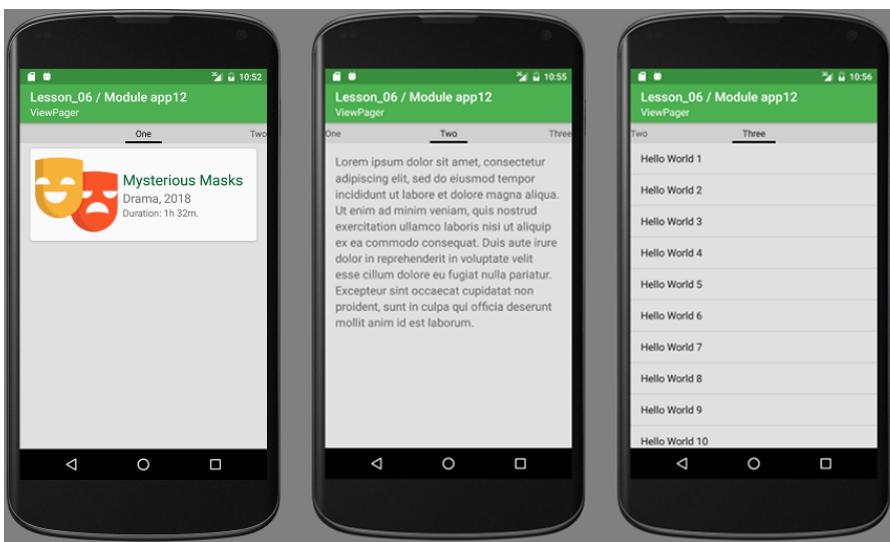


Рис. 2.35. Работа примера с индикатором страниц `android.support.v4.view.PagerTabStrip`

Интерактивный индикатор страниц `android.support.v4.view.PagerTabStrip` на Рис. 2.35 находится под тулбаром.

Однако, чаще всего контейнер `android.support.v4.view.ViewPager` используется совместно с менеджером `android.support.design.widget.TabLayout`, который является более функциональным индикатором страниц. С менеджером раскладки `android.support.design.widget.TabLayout` мы познакомимся в следующем разделе данного урока.

Исходные коды примера, который рассматривался в данном разделе, находятся в модуле «app12» среди файлов с исходными кодами, которые прилагаются к данному уроку.

2.12. Менеджер TabLayout

Менеджер раскладки [android.support.design.widget.TabLayout](#) предоставляет горизонтальную компоновку для отображения вкладок (см. Рис. 2.36). Иерархия классов для класса [android.support.design.widget.TabLayout](#) выглядит следующим образом:

```
java.lang.Object
  |
  +--- android.view.View
    |
    +--- android.view.ViewGroup
      |
      +--- android.widget.FrameLayout
        |
        +--- android.widget.HorizontalScrollView
          |
          +--- android.support.design.widget.TabLayout
```

Как видно из иерархии классов, класс [android.support.design.widget.TabLayout](#) является прямым наследником менеджера горизонтальной прокрутки [android.widget.HorizontalScrollView](#). По сути, менеджер [android.support.design.widget.TabLayout](#) является коллекцией вкладок. Каждая вкладка представлена классом [android.support.design.widget.TabLayout.Tab](#). У вкладки есть текстовая надпись или иконка, которые идентифицируют отображаемое содержимое вкладки. Но никакого содержимого (контента) у вкладок нет. Как уже упоминалось в предыдущем разделе, менеджер [android.support.design.widget.TabLayout](#) чаще всего применяется для контейнера

страниц **android.support.v4.view.ViewPager**. Это значит, что содержимое вкладок предоставляется контейнером **android.support.v4.view.ViewPager**, а заголовок вкладки (иконка, надпись) предоставляет **android.support.design.widget.TabLayout**. Текст заголовка назначается при помощи метода класса **android.support.design.widget.TabLayout.Tab**:

```
TabLayout.Tab setText (int resId);
TabLayout.Tab setText (CharSequence text)
```

Иконка назначается при помощи метода класса **android.support.design.widget.TabLayout.Tab**:

```
TabLayout.Tab setIcon (int resId);
```

Вкладка **android.support.design.widget.TabLayout.Tab** создается при помощи метода класса **android.support.design.widget.TabLayout**:

```
TabLayout.Tab newTab ();
```

Пример, как создается набор вкладок **android.support.design.widget.TabLayout.Tab** для менеджера **android.support.design.widget.TabLayout** показан в Листинге 2.64.

Листинг 2.64. Пример программного создания набора вкладок **android.support.design.widget.TabLayout.Tab** для менеджера **android.support.design.widget.TabLayout**

```
TabLayout tabLayout = (TabLayout)
    this.findViewById(R.id.tabLayout);
tabLayout.addTab(tabLayout.newTab()
    .setText("One"));
```

```
tabLayout.addTab(tabLayout.newTab() .
    setText("Two"));
tabLayout.addTab(tabLayout.newTab() .
    setText("Three"));
```

Также, набор вкладок можно создать и при помощи xml — в файле ресурсов с макетом внешнего вида. Как это делается показано в Листинге 2.65.

Листинг 2.65. Пример создания набора вкладок android.support.design.widget.TabLayout.Tab для менеджера android.support.design.widget.TabLayout при помощи xml в файле ресурсов макета внешнего вида

```
<android.support.design.widget.TabLayout
    android:layout_height="wrap_content"
    android:layout_width="match_parent">
    <android.support.design.widget.TabItem
        android:text="@string/tab_text"/>
    <android.support.design.widget.TabItem
        android:icon="@drawable/ic_android"/>
</android.support.design.widget.TabLayout>
```

После создания набора вкладок, менеджеру **android.support.design.widget.TabLayout** назначается обработчик событий смены выбранных вкладок при помощи метода:

```
void addOnTabSelectedListener (TabLayout.
    OnTabSelectedListener listener);
```

Метод принимает ссылку на объект, который будет обрабатывать события смены выбранной вкладки. Объект этого класса должен реализовать интерфейс **android.**

[support.design.widget.TabLayout.OnTabSelectedListener](#), в котором объявлены три метода:

- **void onTabReselected(TabLayout.Tab tab)** — Метод вызывается, когда выделенная в данный момент вкладка выбирается пользователем повторно.
- **void onTabSelected(TabLayout.Tab tab)** — Метод вызывается когда вкладка становится выбранной.
- **void onTabUnselected(TabLayout.Tab tab)** — Метод вызывается когда вкладка перестает быть выбранной по причине выбора другой вкладки.

Перечисленные выше методы в качестве параметра принимают ссылку на вкладку [android.support.design.widget.TabLayout.Tab](#) которая является источником события.

Рассмотрим на примере использование менеджера [android.support.design.widget.TabLayout](#). С этой целью создан модуль «app13» в проекте Android Studio, который прилагается к данному уроку. В этом примере мы будем использовать те же классы Фрагментов ([android.support.v4.app.Fragment](#)) что и в предыдущем примере из модуля «app12»: [FragmentPageOne](#), [FragmentPageTwo](#), [FragmentPageThree](#) (см. Листинг 2.58). И, соответственно, файлы ресурсов с макетами внешнего вида Фрагментов: /res/layout/fragment_page_one.xml, /res/layout/fragment_page_two.xml и /res/layout/fragment_page_three.xml (см. Листинг 2.57).

Файлы Java с классами фрагментов и файлы ресурсов с макетами внешнего вида Фрагментов скопированы их модуля «app12» в модуль «app13». Также, из модуля «app12» взят класс [MyFragmentStatePagerAdapter](#) который

является поставщиком страниц для контейнера **android.support.v4.view.ViewPager** (см. Листинг 2.59). Файл ресурсов с макетом внешнего вида Активности (/res/layout/activity_main.xml) для текущего примера также взят из модуля «app12» (см. Листинг 2.56). Только в макет Активности добавлен менеджер **android.support.design.widget.TabLayout**, и это отображено в Листинге 2.66 (выделено жирным шрифтом).

Листинг 2.66. Файл макета внешнего вида Активности /res/layout/activity_main.xml с добавленным менеджером android.support.design.widget.TabLayout

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    android:id="@+id/clMain"
    tools:context="itstep.com.myapp13.MainActivity">
    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="60dp"
            app:popupTheme=
                "@style/ThemeOverlay.AppCompat.Light"
            app:layout_collapseMode="pin"
        />
```

```
<android.support.design.widget.TabLayout  
    android:id="@+id/tabLayout"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    app:tabGravity="fill"  
    app:tabMode="fixed" >  
</android.support.design.widget.TabLayout>  
  
</android.support.design.widget.AppBarLayout>  
  
<android.support.v4.view.ViewPager  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    app:layout_behavior=  
        "@string/appbar_scrolling_view_behavior"  
    android:id="@+id/vpContent">  
</android.support.v4.view.ViewPager>  
  
</android.support.design.widget.CoordinatorLayout>
```

Как видно из Листинга 2.66, менеджер **android.support.design.widget.TabLayout** (идентификатор **tabLayout**) добавлен рядом с тулбаром **android.support.v7.widget.Toolbar** в качестве прямого дочернего элемента для контейнера **android.support.design.widget.AppBarLayout**.

Далее, в методе **onCreate()** класса Активности (**MainActivity**) добавим для менеджера **android.support.design.widget.TabLayout** (идентификатор **tabLayout**) обработчик событий смены выбранных вкладок **TabLayout.OnTabSelectedListener**. Назначение обработчика событий смены выбранных вкладок и сам код обработки этих событий показаны в Листинге 2.67.

Листинг 2.67. Класс MainActivity с программным кодом создания вкладок для менеджера android.support.design.widget.TabLayout и назначением этому менеджеру обработчика события смены выбранной вкладки

```
public class MainActivity extends AppCompatActivity
{
    // -----
    private final static String TAG = "===== MainActivity";

    // -----
    /**
     * The pager widget, which handles animation and
     * allows swiping horizontally to access previous
     * and next wizard steps.
     *
     * Контейнер ViewPager который отображает страницы
     * с возможностью перелистывания этих страниц
     * вперед и назад.
     */
    private ViewPager viewPager;

    /**
     * The pager adapter, which provides the pages
     * to the view pager widget.
     *
     * Поставщик страниц для контейнера ViewPager
     * (поле viewPager).
     */
    private PagerAdapter adapter;

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

```
// ----- Toolbar -----
Toolbar toolBar = (Toolbar) this.findViewById(
    R.id.toolbar);
this.setSupportActionBar(toolBar);
toolBar.setTitle(R.string.app_name);
toolBar.setSubtitle("TabLayout");
toolBar.setTitleTextColor(Color.WHITE);
toolBar.setSubtitleTextColor(Color.WHITE);

// ----- PagerAdapter -----
this.viewPager = (ViewPager) this.
    findViewById(R.id.vpContent);
this.adapter = new MyFragmentStatePagerAdapter(
    this.getSupportFragmentManager());
this.viewPager.setAdapter(this.adapter);

// ----- TabLayout -----
final TabLayout tabLayout = (TabLayout) this.
    findViewById(R.id.tabLayout);
tabLayout.addTab(tabLayout.newTab().
    setText("One"));
tabLayout.addTab(tabLayout.newTab().
    setText("Two"));

tabLayout.addTab(tabLayout.newTab().
    setText("Three"));
tabLayout.setOnTabSelectedListener(
    new TabLayout.OnTabSelectedListener()
    {
        @Override
        public void onTabSelected(TabLayout.
            Tab tab)
        {
            Log.d(TAG, "onTabSelected: " +
                tab.getText().toString());
            MainActivity.this.viewPager.
                setCurrentItem(tab.
                    getPosition(), true);
        }
    });
}

// ----- Fragment -----
public class MyFragmentStatePagerAdapter extends FragmentStatePagerAdapter
{
    public MyFragmentStatePagerAdapter(FragmentManager fm)
    {
        super(fm);
    }

    @Override
    public Fragment getItem(int position)
    {
        switch (position)
        {
            case 0:
                return new OneFragment();
            case 1:
                return new TwoFragment();
            case 2:
                return new ThreeFragment();
            default:
                return null;
        }
    }

    @Override
    public int getCount()
    {
        return 3;
    }
}
```

```
        }

    @Override
    public void onTabUnselected(TabLayout.
                                Tab tab)
    {
        Log.d(TAG, "onTabUnselected: " +
               tab.getText().toString());
    }

    @Override
    public void onTabReselected(TabLayout.
                               Tab tab)
    {
        Log.d(TAG, "onTabReselected: " +
               tab.getText().toString());
    }
}

// ***
}
```

Как видно из Листинга 2.67, при смене выбранной вкладки (в методе **onTabSelected()** обработчика события) происходит вызов метода **setCurrentItem()** контейнера **android.support.v4.view.ViewPager**, с помощью которого отображается страница, которая соответствует выбранной вкладке.

Внешний вид примера из Листингов 2.66 и 2.67 изображен на Рис. 2.36.

Как видно из Рис. 2.36, цвет фона менеджера **android.support.design.widget.TabLayout** является первичным цветом (**colorPrimary**), а цвет подсветки выбранной вкладки является акцентным цветом (**colorAccent**).

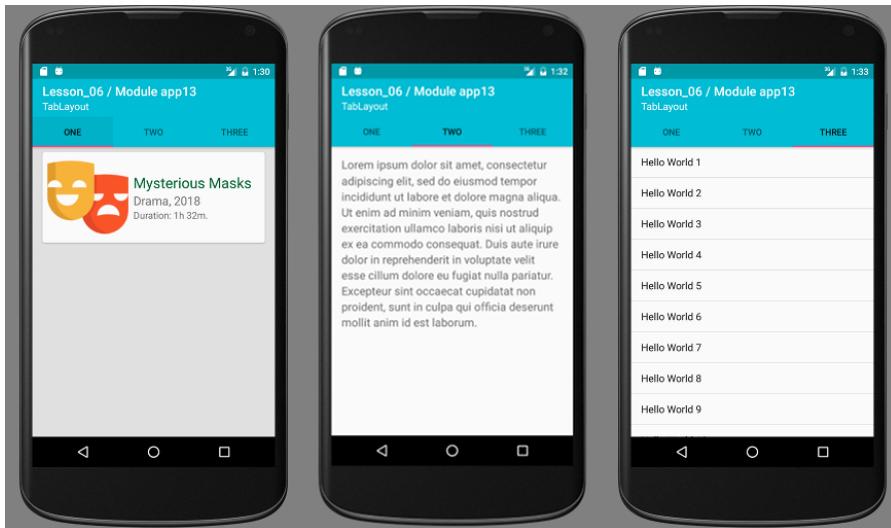


Рис. 2.36. Внешний вид примера из Листингов 2.66 и 2.67

Теперь, при выборе пользователем вкладки, происходит отображение соответствующей страницы из контейнера `android.support.v4.view.ViewPager`. Однако, если пользователь будет перелистывать страницы не при помощи менеджера `android.support.design.widget.TabLayout` а при помощи контейнера страниц `android.support.v4.view.ViewPager`, то смена выбранной вкладки в менеджере `TabLayout` происходит не будет. Это нужно самостоятельно реализовывать программно. Для этого необходимо добавить для контейнера `android.support.v4.view.ViewPager` обработчик событий [`ViewPager.OnPageChangeListener`](#), который добавляется при помощи метода:

```
void addOnPageChangeListener(ViewPager.  
    OnPageChangeListener listener);
```

В интерфейсе `android.support.v4.view.ViewPager.OnPageChangeListener` объявлено три метода:

- `void onPageScrollStateChanged(int state)` — Метод вызывается при изменении состояния прокрутки в контейнере `ViewPager`: когда пользователь начинает движение перелистывания страницы (`state` равен `SCROLL_STATE_DRAGGING`), когда контейнер `ViewPager` автоматически устанавливается на текущую страницу (`state` равен `SCROLL_STATE_SETTLING`) или когда контейнер `ViewPager` полностью остановлен (`state` равен `SCROLL_STATE_IDLE`).
- `void onPageScrolled(int position, float positionOffset, int positionOffsetPixels)` — Метод будет вызываться, когда прокручивается текущая страница в контейнере `ViewPager`. Параметр `position` — индекс первой отображаемой в данный момент страницы, если параметр `positionOffsetPixels` отличен от нуля, то страница с индексом `position + 1` видна. Параметр `positionOffset` содержит значение в диапазоне от 0 до 1 не включительно, указывающее смещение страницы от своего нормального положения. Параметр `positionOffsetPixels` содержит значение в пикселях, указывающих смещение страницы от своего нормального положения.
- `void onPageSelected(int position)` — Метод вызывается когда выбрана страница с индексом `position`.

Из перечисленных выше методов нам больше всего подходит метод `onPageSelected()`. В этом методе мы и бу-

дем сообщать менеджеру `android.support.design.widget.TabLayout` о необходимости смены выбранной вкладки. Программный код обработчика события `android.support.v4.view.ViewPager.OnPageChangeListener` показан в Листинге 2.68. Этот код добавляется в метод `onCreate()` класса `MainActivity` из Листинга 2.67 в место, которое помечено тремя звездочками (***)�.

Листинг 2.68. Обработчик событий перелистывания страниц в контейнере `android.support.v4.view.ViewPager`

```
@Override  
protected void onCreate(Bundle savedInstanceState)  
{  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    ...  
  
    // ***  
    this.viewPager.addOnPageChangeListener(  
        new ViewPager.OnPageChangeListener()  
    {  
        @Override  
        public void onPageScrolled(int position,  
            float positionOffset,  
            int positionOffsetPixels)  
        {  
            Log.d(TAG, "onPageScrolled.  
                position: " + position +  
                ", positionOffset: " +  
                positionOffset +  
                ", positionOffsetPixels: " +  
                positionOffsetPixels);  
        }  
  
        @Override
```

```
public void onPageSelected(int position)
{
    Log.d(TAG, "onPageSelected");
    tabLayout.getTabAt(position).select();
}

@Override
public void onPageScrollStateChanged(
        int state)
{
    Log.d(TAG, "onPageScrollStateChanged");
}
});
```

В Листинге 2.68 в методе **onPageSelected()** программно сообщается менеджеру **TabLayout** о необходимости смены выбранной вкладки. Этот фрагмент кода в Листинге 2.68 выделен жирным шрифтом.

Исходные коды примера, который рассматривался в данном разделе, находятся в модуле «app13» среди файлов с исходными кодами, которые прилагаются к данному уроку.

3. Домашнее задание

1. Создайте приложение со списком Музыкальных Альбомов, которое будет соответствовать концепции Material Design. Примерный вид приложения изображен на Рис. 3.1.

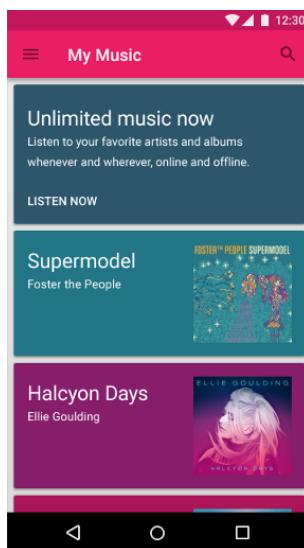


Рис. 3.1. Примерный вид приложения из домашнего задания №5

Для списка Музыкальных Альбомов используйте виджет **android.support.v7.widget.RecyclerView**. Информацию о каждом Альбоме (название альбома, исполнитель, обложка альбома, год выпуска) отображать в виджетах **android.support.v7.widget.CardView**. Сам список Музыкальных Альбомов хранить в Базе Данных. В приложе-

нии должно быть навигационное меню `android.support.design.widget.NavigationView` со следующими пунктами: «Отобразить по названию», «Отобразить по исполнителю», «Отобразить по году выпуска». С помощью этих пунктов меню список Музыкальных Альбомов упорядочивается (сортируется) в соответствии со своим пунктом меню. Не забудьте о контейнере `android.support.design.widget.CoordinatorLayout` и сворачивающемся и разворачивающемся тулбаре.



Урок № 6. Material Design

© Бояршинов Юрий
© Компьютерная Академия «Шаг»
www.itstep.org

Все права на охраняемые авторским правом фото-, аудио- и видеопроизведения, фрагменты которых использованы в материале, принадлежат их законным владельцам. Фрагменты произведений используются в иллюстративных целях в объеме, оправданном поставленной задачей, в рамках учебного процесса и в учебных целях, в соответствии со ст. 1274 ч. 4 ГК РФ и ст. 21 и 23 Закона Украины «Про авторське право і суміжні права». Объем и способ цитируемых произведений соответствует принятым нормам, не наносит ущерба нормальному использованию объектов авторского права и не ущемляет законные интересы автора и правообладателей. Цитируемые фрагменты произведений на момент использования не могут быть заменены альтернативными, не охраняемыми авторским правом аналогами, и как таковые соответствуют критериям добросовестного использования и честного использования.

Все права защищены. Полное или частичное копирование материалов запрещено. Согласование использования произведений или их фрагментов производится с авторами и правообладателями. Согласованное использование материалов возможно только при указании источника.

Ответственность за несанкционированное копирование и коммерческое использование материалов определяется действующим законодательством Украины.