**Лекция 5. Материальный дизайн**

В этой лекции обсуждается [Material design](https://material.io/guidelines/material-design/introduction.html) : *язык дизайна* , созданный Google для поддержки мобильных пользовательских интерфейсов. Язык дизайна фокусируется на мобильных дизайнах (например, для Android), но может также использоваться на разных платформах (например, в Интернете через [фреймворки css](https://material.io/components/web/) ).

В этой лекции используется код, найденный по адресу <https://github.com/info448/lecture05-material> .

Поддержка Material Design была введена в API 21 Lollipop, поэтому для доступа ко всем функциям Material, поддерживаемым Android, требуется устройство под управлением этой версии Android или более поздней. Однако большинство функций также доступны через [библиотеки совместимости](https://developer.android.com/training/material/compatibility.html) (например, AppCompat), и поэтому могут использоваться и на старых устройствах.

Google находится в процессе обновления своей реализации Material Design для использования [Material Design Components (MDC)](https://material.io/develop/android/) . Поскольку эта версия все еще находится в стадии бета-тестирования и доступна только на Android Pie, в этой лекции описывается, как использовать «устаревшую» версию библиотеки поддержки Material.

**5.1 Язык материального дизайна**

**Material design** — это *язык дизайна* : «словарь» визуальных и интерактивных шаблонов, которые являются общими для всех систем. Material формирует как «внешний вид и ощущение» для приложений Google, так и для современных (API 21+) приложений Android в целом, а также предоставляет

Видеообъяснение языка Material (по ссылке <https://developer.android.com/design/material/index.html> )

Подводя итог, можно сказать, что язык Material Design основан на [трех основных принципах](https://material.io/guidelines/material-design/introduction.html#introduction-principles) :

* ***Материал — это метафора*** . Язык дизайна Material строится вокруг представления пользовательских интерфейсов как созданных из виртуальных[**материалов**](https://material.io/guidelines/material-design/material-properties.html) , которые представляют собой[поверхности, подобные бумаге,](https://material.io/guidelines/layout/principles.html#principles-how-paper-works) плавающие в пространстве. Каждая поверхность имеет одинаковую толщину (1dp), но разную*высоту* (передаваемую в основном через тени и*перспективу* ). Эта физическая метафора помогает указать различные возможности и взаимодействия с пользователем: например, кнопка «поднята» и может быть «нажата».
* ***Движение придает смысл*** . Material также уделяет большое внимание использованию[**движения**](https://material.io/guidelines/motion/material-motion.html) этих материалов, чтобы помочь описать отношения между компонентами, а также сделать дизайн в целом более «восхитительным». Приложения Material включают множество анимаций и завитушек, делая использование приложения непрерывным и связанным. Поверхности могут менять форму, размер и положение (пока они остаются в своей плоскости — они не могут складываться, но могут разделяться и соединяться) в ответ на действия пользователя.
* ***Смелый, графический, намеренный*** . Материальные приложения следуют определенной эстетике с точки зрения таких вещей, как[цвет](https://material.io/guidelines/style/color.html) (не приглушенный),[изображения](https://material.io/guidelines/style/imagery.html) (обычно их много). Материальные приложения*выглядят* как материальные приложения, хотя их все еще можно настраивать в соответствии с вашими конкретными потребностями.

Для получения дополнительной информации о языке дизайна Material см. [**официальные рекомендации**](https://material.io/guidelines/) (нажмите кнопку-гамбургер слева для навигации). Эта документация содержит обширные примеры и инструкции, начиная от рекомендуемых размеров шрифтов и заканчивая советами по использованию виджетов.

В этой лекции основное внимание уделяется способам реализации отдельных аспектов языка Material Design в приложениях Android, а не особенностям соблюдения рекомендаций языка.

**5.2 Материальные стили и значки**

Первый и самый простой шаг к созданию приложения на основе Material — использовать предоставленные [Material Themes](https://developer.android.com/training/material/theme.html) для «оформления» вашего приложения. Эти темы доступны в API 21 Lollipop и более поздних версиях; для более ранних операционных систем вы можете вместо этого использовать эквивалентные темы в AppCompat(которые на самом деле являются темами по умолчанию для новых приложений Android!)

Применение тем (включая темы Material) более подробно обсуждается в лабораторной работе [«Стили и темы»](https://info448.github.io/styles-themes.html#styles-themes) .

* Вы можете увидеть, какие конкретные свойства применяются этими стилями и темами, просмотрев [исходный код](https://android.googlesource.com/platform/frameworks/base/+/refs/heads/master/core/res/res/values) для фреймворка Android — проверьте styles\_material.xmlи themes\_material.xml(они также будут ссылаться на значения, определенные в переменной colorи dimensресурсах). AppCompatСтили и темы могут быть [исходным кодом](https://android.googlesource.com/platform/frameworks/support.git/+/refs/heads/master/v7/appcompat/res/values/) для v7библиотеки поддержки.

Давайте начнем с указания на один из самых заметных визуальных компонентов в приложении по умолчанию: [***App Bar***](https://developer.android.com/training/appbar/index.html) или ***Action Bar*** . Он действует как своего рода «заголовок» для вашего приложения, предоставляя выделенное пространство для навигации и взаимодействия (например, через меню). [14](https://info448.github.io/material-design.html#fn14) — это особый тип, который чаще всего используется в качестве App Bar, предлагая определенный «внешний вид», общий для приложений Android.[ActionBar](https://developer.android.com/reference/android/support/v7/app/ActionBar.html)[Toolbar](https://developer.android.com/reference/android/support/v7/widget/Toolbar.html)

Хотя AppCompatActivityиспользуемый в этом курсе автоматически предоставляет панель действий для приложения, его также можно добавить напрямую (например, если вы используете другой подкласс Activity). Чтобы добавить собственную панель действий, вы указываете **тему** , которая *не* включает ActionBar, а затем включаете <android.support.v7.window.Toolbar>элемент в свой макет, где бы вы хотели разместить панель инструментов. Подробнее см. в разделе [Настройка панели приложений](http://developer.android.com/training/appbar/setting-up.html) . Это также позволит вам разместить панель инструментов в любом месте макета приложения (например, если вы хотите, чтобы она была прикреплена к нижней части).

На практике, самая большая часть использования Material Theme — это определение [цветовой палитры](https://developer.android.com/training/material/theme.html#ColorPalette) для вашего приложения. Спецификация Material design описывает широкую цветовую палитру (с доступными [образцами](https://material.io/guidelines/style/color.html#color-color-palette) ); однако, часто бывает полезнее выбирать цвета с помощью предоставленного [**инструмента выбора цвета**](https://material.io/color/#!/)[15](https://info448.github.io/material-design.html#fn15) , который позволяет вам легко экспериментировать с различными цветовыми сочетаниями.

* Выберите *основной* и *дополнительный* цвета, а затем назначьте значения ресурсов colorPrimary, colorPrimaryDark, и colorAccentв res/values/colors.xml. Это позволит вам легко «брендировать» свое приложение.

Атрибуты, специфичные для материалов, такие как , android:elevationтакже доступны в API 21+, хотя для того, чтобы сделать тени видимыми на произвольных элементах, требуется некоторая дополнительная работа.

Помимо стилей, Material также включает большой набор [**иконок**](https://material.io/icons/) для использования в приложениях. Эти иконки дополняют встроенные в платформу ic\_\* [элементы рисования](http://androiddrawables.com/) , доступные для использования (и отображаются как опции автозаполнения в IDE).

Вместо этого эти значки доступны в виде [векторных рисунков](https://developer.android.com/training/material/drawables.html#VectorDrawables) — вместо того, чтобы быть .pngфайлом, изображения определяются с использованием схемы XML, аналогичной той, которая используется в [масштабируемой векторной графике (SVG)](https://en.wikipedia.org/wiki/Scalable_Vector_Graphics) .

SVG используется для представления [векторной графики](https://en.wikipedia.org/wiki/Vector_graphics) — то есть изображений, определенных в терминах соединения между точками (линиями и другими фигурами), а не в терминах отображаемых пикселей (называемых [растровым изображением](https://en.wikipedia.org/wiki/Raster_graphics) ). Для отображения на экране эти линии преобразуются (растрируются) в сетки пикселей на основе размера и разрешения дисплея. Это позволяет изображениям SVG «масштабироваться» или «увеличиваться» независимо от размера дисплея или изображения — вам никогда не придется беспокоиться о том, что что-то станет размытым, когда вы увеличите их!

Чтобы включить иконку Material, вам нужно будет сгенерировать XML-код для этого конкретного изображения. К счастью, Android Studio включает определения XML для всех иконок Material (хотя вы также можете определить свои собственные векторные чертежи).

* Чтобы создать векторный рисунок, выберите File > New > Vector Assetв меню Android Studio. Затем вы можете нажать на значок, чтобы выбрать значок Material, который вы хотите создать.
* По умолчанию значок будет окрашен в *черный* цвет . Если вы хотите изменить цвет значка при включении его в макет, укажите цвет через [android:tint](https://developer.android.com/training/material/drawables.html#DrawableTint)атрибут View в вашем XML (например, в ImageButton).

Также возможно определять произвольные векторные формы в API 21+. Например, стартовый код включает ресурс для элемента <shape>, имеющего форму овала.

**5.3 Библиотеки поддержки проектирования**

Помимо свойств стилей и ресурсов, доступных как часть фреймворка Android, существуют также дополнительные компоненты, доступные через дополнительные [**библиотеки поддержки**](https://developer.android.com/training/material/design-library.html) .

Подобно Volley, эти библиотеки не встроены в Android и поэтому должны быть явно указаны dependenciesв файле вашего приложения build.gradle. Например:

//note the version number needs to match your SDK version

implementation 'com.android.support:design:28.0.0'

будет включать последнюю версию (на момент написания статьи) **библиотеки поддержки дизайна** , которая включает такие элементы, как плавающая кнопка действия и макеты координатора (описанные ниже).

Обратите внимание, что если у вас возникли проблемы с установкой зависимости, [настройка для включения библиотек поддержки](https://developer.android.com/topic/libraries/support-library/setup.html) была изменена в июле 2017 года для поддержки загрузки библиотек зависимостей через [maven](https://maven.apache.org/) , а не напрямую из Android Studio. Для поддержки этого вам нужно будет изменить объявление репозитория *проекта* build.gradle , чтобы оно выглядело так:

allprojects {

repositories {

jcenter()

maven {

url "https://maven.google.com"

}

}

}

**Виджеты**

Вспомогательные библиотеки поддерживают ряд полезных виджетов (специализированных представлений) для создания приложений в стиле Material.

**RecyclerПросмотр**

Это [RecyclerView](https://developer.android.com/training/material/lists-cards.html#RecyclerView)более продвинутая версия ListView, предоставляющая более надежную систему для поддержки интерактивных представлений в списке, а также анимацию для таких действий, как добавление и удаление элементов списка.

Этот класс является частью v7библиотеки поддержки (а не библиотеки Material Design конкретно), поэтому вам нужно будет включить его специально в файл вашего *приложения* build.gradle (так же, как вы включили Volley):

implementation 'com.android.support:cardview-v7:28.0.0'

Реализация a RecyclerViewочень похожа на реализацию a ListView(с добавлением [шаблона ViewHolder](https://developer.android.com/training/improving-layouts/smooth-scrolling.html#ViewHolder) ), хотя вам также необходимо объявить a, LayoutManagerчтобы указать, должен ли ваш RecyclerView использовать список или сетку.

Лучший способ понять RecyclerView — посмотреть на пример и изменить его в соответствии с вашим конкретным представлением элемента. Например, вы можете изменить ListView на RecyclerView, адаптировав пример кода, предоставленный [официальной документацией Google](https://developer.android.com/guide/topics/ui/layout/recyclerview)[16](https://info448.github.io/material-design.html#fn16) :

* Во-первых, вам нужно будет заменить декларацию XML View на <android.support.v7.widget.RecyclerView>элемент. По сути, мы просто изменяем **контроллер** для списка.
* При настройке RecyclerViewв коде Java есть несколько дополнительных шагов. В частности, вам также понадобится связать LayoutManagerобъект с View — например, LinearLayoutManagerдля отображения элементов в строке (a la ListView) или GridLayoutManagerдля отображения элементов в сетке (a la GridView).
  + В Kotlin их можно назначить с помощью .apply()метода. Этот метод принимает функцию обратного вызова и выполняет каждую строку в этой функции обратного вызова с объектом (например, recyclerView), областью действия которого является this. Это сокращение для вызова множества методов для одного и того же объекта подряд.

Аналогичные сочетания клавиш предусмотрены функциями run(), with()и let(). Для ясного объяснения см. [это руководство .](https://medium.com/@elye.project/mastering-kotlin-standard-functions-run-with-let-also-and-apply-9cd334b0ef84)

* Все RecyclerViewsтребуют *специальных адаптеров* : мы не можем просто использовать встроенный адаптер, такой как ArrayAdapter. Для этого создайте класс, который extends RecyclerView.Adapter<VH>.
  + Generic в этом классе — это класс, представляющий [View Holder](https://developer.android.com/training/improving-layouts/smooth-scrolling.html#ViewHolder) . Это шаблон, по которому каждое отдельное View, которое будет расширено и на которое будет сделана ссылка, хранится как отдельный *объект* с конкретными View, которые будут изменены (например, TextView), сохраненными как переменные экземпляра. Это позволяет избежать необходимости для адаптера повторно использовать findViewById(), что является дорогостоящей операцией (поскольку она включает обход иерархического дерева View!)

Будет ViewHolderдругим классом — обычно внутренним классом адаптера. Мы можем рассматривать его как *класс*data (простой объект «контейнер», похожий на a structв C). но не нужно объявлять его таковым напрямую. При необходимости вы можете назначить этим переменным экземпляра результаты вызовов findViewById()при инициализации объекта ViewHolder.

* + RecyclerView(а не ) ViewHolderтребует переопределения onCreateViewHolder()метода, который будет inflate()представлять и создавать экземпляр ViewHolderдля каждого элемента, когда этот элемент необходимо отобразить в первый раз.
  + В переопределенном onBindViewHolder()вы можете выполнить фактическую работу по назначению содержимого модели конкретному View (например, называемому setText()). Вам не нужно использовать findViewById()здесь, потому что вы уже сохранили ссылку на этот View в ViewHolder, поэтому вы можете назначить ему напрямую!
  + Используется getItemCount()внутри RecyclerView для определения того, когда вы достигли «конца» списка.
* Наконец, вы можете назначить пользовательский адаптер для RecyclerViewтого, чтобы связать модель и представление.

Вот и все! Хотя это больше кода и сложнее, чем базовый ListView, как только вы добавите шаблон ViewHolder или сделаете какие-либо другие настройки, вы в основном на том же уровне. Использование RecyclerView вместо ListView также включает встроенные анимации, а также общие действия пользователя, такие как «перетащить для заказа» или «провести для отклонения». Смотрите [это руководство](https://medium.com/@ipaulpro/drag-and-swipe-with-recyclerview-b9456d2b1aaf) для пошагового руководства по добавлению этих возможностей.

Обратите внимание, что *в отличие* от ListView, мы обычно изменяем элементы, показанные в , RecyclerViewнапрямую изменяя модель данных (например, массив или ArrayList). Но нам нужно будет [уведомить](https://guides.codepath.com/android/using-the-recyclerview#notifying-the-adapter) адаптер RecyclerViewоб этих изменениях, вызвав один из различных .notify()методов адаптера (например, adapter.notifyItemInserted(position)). Это заставит адаптер «обновить» отображение, и он фактически анимирует изменения в списке по умолчанию!

**Карты**

Библиотека v7поддержки также предоставляет View для простого оформления содержимого в виде [**карточек**](https://developer.android.com/training/material/lists-cards.html#CardView) . A CardViewпо сути представляет собой FrameLayout(ViewGroup, содержащую одно дочернее View), но включает границы и тени, благодаря которым группа выглядит как карточка.

* Вам нужно будет загрузить CardViewкласс как зависимость Gradle с помощью compile 'com.android.support:cardview-v7:26.1.0'.

Чтобы использовать CardView, просто включите его в свой макет, как и любую другую ViewGroup:

**<android.support.v7.widget.CardView**

xmlns:card\_view="http://schemas.android.com/apk/res-auto"

android:layout\_width="@dimen/card\_width"

android:layout\_height="@dimen/card\_height"

android:layout\_gravity="center"

card\_view:cardCornerRadius="4dp"**>**

*<!-- A single TextView in the card -->*

**<TextView**

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="@string/card\_text" **/>**

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

* Обратите внимание на card\_view:cardCornerRadiusатрибут; это пример того, как определенные представления могут иметь свои собственные пользовательские свойства (иногда доступные через другую схему).

Поскольку карты являются FrameLayouts, они должны содержать только один дочерний элемент. Если вы хотите включить несколько элементов в карту (например, изображение, текст и кнопку), вам нужно будет вложить еще одну ViewGroup (например, LinearLayout) внутрь карты.

Рекомендации по дизайну карточек, включая информацию о интервалах, см. [в руководстве по материальному дизайну](https://material.io/guidelines/components/cards.html) .

Если вы хотите включить круглое изображение в свою карточку (или в любое другое место вашего приложения), самым простым решением будет включение внешней библиотеки, которая предоставляет такое представление, самой популярной из которых является [de.hdodenhof.circleimageview](https://github.com/hdodenhof/CircleImageView).

**Плавающие кнопки действий (FAB)**

Хотя RecyclerViews и Cards находятся в v7библиотеке поддержки, наиболее заметные и интересные компоненты находятся в [библиотеке поддержки дизайна](https://developer.android.com/training/material/design-library.html) , которая специально включает компоненты для поддержки Material Design.

* Эту библиотеку следует включить в gradle как com.android.support:design:27.1.1в примере выше.

Наиболее распространенным элементом этой библиотеки является [**Floating Action Button (FAB)**](https://developer.android.com/training/material/design-library.html#CreateFAB) . Это круглая кнопка, которая «плавает» над содержимым экрана (на большей высоте) и представляет собой *основное действие* пользовательского интерфейса.

* Экран должен иметь только один FAB и только если есть одно основное действие, которое должно быть выполнено. См. [руководство по проектированию](https://material.io/guidelines/components/buttons-floating-action-button.html) для получения дополнительных примеров того, как использовать (и не использовать) FABS.

Как и карту, вы можете включить FAB в свое приложение, указав ее как элемент в XML:

**<android.support.design.widget.FloatingActionButton**

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="end|bottom"

android:layout\_margin="@dimen/fab\_margin"

android:src="@drawable/ic\_my\_icon" **/>**

* Поскольку FAB являются подклассом ImageButton, мы можем просто заменить существующую кнопку на FAB, ничего не сломав.

Fabs поддерживают ряд дополнительных эффектов. Например, если вы сделаете FAB кликабельным (через android:clickable), вы можете указать , app:rippleColorчтобы придать ему эффект ряби при нажатии. Более подробная информация будет представлена ​​ниже.

**Закусочные**

Библиотека поддержки дизайна также включает альтернативу сообщениям Toast, называемую [**Snackbars**](https://developer.android.com/training/snackbar/showing.html) . Это всплывающее сообщение, которое отображается в нижней части экрана (похоже на Toast).

Закусочные показаны с использованием структуры, похожей на тосты:

**val** snack = Snackbar.make(view, "Let's go out to the lobby!", Snackbar.LENGTH\_LONG).show();

* Вместо вызова Toast.makeText()метода фабрики мы вызываем Snackbar.make()метод фабрики. Первым параметром в этом случае должен быть View, к которому будет «прикреплен» Snackbar (так показано) — однако, на самом деле не имеет значения, какой View задан, поскольку метод будет искать вверх по иерархии View, пока не доберется до представления корневого содержимого или специального макета, называемого CoordinatorLayout.
* Вы заметите, что Snackbar накладывается на содержимое (включая FAB). Это будет решено ниже путем введения CoordinatorLayout.

Кроме того, можно дать Snackbar-ам их собственное *действие* , которое пользователь может активировать, нажав на Snackbar. Это позволяет панели, например, показывать подтверждение удаления, но при этом предоставлять действие «отменить». Действие указывается путем вызова метода .setAction()на Snackbar и передачи заголовка для действия, а также OnClickListener:

mySnackbar.setAction("Click", **new** View.OnClickListener() {

@Override

**public** void onClick(View view) {

*//...*

}

});

snack.setAction("Click") {

*//...*

};

* Для практики сделайте Snackbar .hide()FAB, но предусмотрите действие «отменить», которое .show()его отменит!

Опять же, ознакомьтесь [с рекомендациями по проектированию](https://material.io/guidelines/components/snackbars-toasts.html) для получения дополнительных примеров того, как использовать (и не использовать) закусочные.

**Координатор макета**

Чтобы исправить перекрытие Snackbar и Fab, нам нужно будет использовать один из самых мощных, но сложных классов в библиотеке поддержки Material: [**CoordinatorLayout**](https://developer.android.com/reference/android/support/design/widget/CoordinatorLayout.html) . Этот макет описывается как «супермощный Framelayout» и обеспечивает поддержку ряда интерактивных и анимированных поведений, включающих другие классы. Многие из «крутых» эффектов в Material построены поверх CoordinatorLayout (или других классов, которые полагаются на него).

Чтобы начать наше исследование CoordinatorLayout, давайте начнем с исправления перекрытия Snackbar. Для этого мы возьмем существующий макет для активности и «обернем» его в <android.support.design.widget.CoordinatorLayout>элемент:

**<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"**>**

*<!-- Previous layout elements -->*

**<RelativeLayout**

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"**>**

*<!-- etc -->*

**</RelativeLayout>**

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

Нам также нужно будет переместить определение FAB так, чтобы оно стало *прямым потомком* CoordinatorLayout. Как только мы это сделаем, мы сможем нажать кнопку и наблюдать, как она перемещается вверх, освобождая место для Snackbar!

* Это работает так: CoordinatorLayout позволяет вам задавать [**Behaviors**](https://developer.android.com/reference/android/support/design/widget/CoordinatorLayout.Behavior.html) для своих дочерних представлений; эти Behaviors затем будут выполняться при изменении состояния CoordinatorLayout (или его дочерних представлений). Например, встроенный FloatingActionButton.Behaviorопределяет, как кнопка должна двигаться в ответ на изменение размера ее родителя, но Behaviors также можно определить в ответ на использование взаимодействий, таких как смахивания или другие жесты.

**Прокручиваемые макеты**

Действительно, встроенные поведения могут быть довольно сложными (и многословными для реализации), и лучший способ понять их — увидеть их в действии. Чтобы увидеть пример этого, создайте **новую** Activity для вашего приложения (например, File > New > Activity). Но вместо создания *Empty* Activity, как вы делали раньше, вам следует создать новую **ScrollingActivity** .

* Измените действие FAB так, чтобы при нажатии на него отправлялся запрос Intentна открытие этого нового действия:

startActivity(**new** Intent(MainActivity.this, ScrollingActivity.class));

* И как только вы откроете Activity... попробуйте прокрутить! Вы должны увидеть, как ActionBar схлопнется, а FAB переместится вверх и исчезнет.

Это пример коллекции поведений, встроенных в CoordinatorLayoutи другие классы в библиотеке Design Support. Чтобы получить представление о том, как они работают, откройте недавно созданный activity\_scrolling.xmlресурс макета и посмотрите, как был построен этот макет!

* В корне макета мы находим CoordinatorLayout, готовый «координировать» все свои дочерние элементы и позволять им взаимодействовать.
* Первый дочерний элемент — [AppBarLayout](https://developer.android.com/reference/android/support/design/widget/AppBarLayout.html). Этот макет специально поддерживает реагирование на события прокрутки, созданные в CoordinatorLayout (например, когда пользователь прокручивает текстовое содержимое). Вы можете управлять видимостью этого элемента на основе прокрутки с помощью app:layout\_scrollFlagsатрибута.

AppBarLayout работает вместе со своим дочерним элементом [CollapsingToolbarLayout](http://developer.android.com/reference/android/support/design/widget/CollapsingToolbarLayout.html), который делает именно то, что и заявлено. Он показывает больший заголовок, но затем сжимается в ответ на прокрутку. Здесь scrollFlagsобъявлены: scroll|exitUntilCollapseуказывает два флага (объединенных с побитовым ИЛИ |): что содержимое должно прокручиваться, и что оно должно сжиматься до минимальной высоты, пока не схлопнется.

Сама Toolbar в конечном итоге определяется как дочерний элемент CollapsingToolbarLayout, хотя сюда можно добавить и другие дочерние элементы. Например, ImageViewможно включить в качестве дочернего элемента CollapsingToolbarLayout для создания сворачивающегося изображения!

(Подробную информацию обо всех конкретных атрибутах смотрите в документации).

* После сворачивания AppBar CoordinatorLayoutвключает [NestedScrollView](https://developer.android.com/reference/android/support/v4/widget/NestedScrollView.html)(объявлено в отдельном файле для организации). Это прокручиваемое представление (похожее на то, что используется в ListView), но может как включать, так и быть включенным в прокручиваемые макеты.
  + Обратите внимание, что этот элемент включает app:layout\_behaviorатрибут, который ссылается на определенный класс: AppBarLayout$ScrollingViewBehavior( $используется для ссылки на скомпилированный вложенный класс). Это поведение будет «автоматически прокручивать любые родственные элементы AppBarLayout», позволяя прокрутке содержимого страницы *также* вызывать прокрутку AppBarLayout!
* Наконец, у нас есть FAB для этого экрана. Самое важное, что следует здесь отметить, это то, как FAB включает атрибут, app:layout\_anchorназначенный ссылке на AppBarLayout. Это указывает, что FAB должен следовать (прокручиваться вместе) за AppBarLayout; свойство app:anchorGravityуказывает, где он должен находиться относительно своей привязки. Более того, поведение FAB по умолчанию заставит его исчезнуть, когда не будет места… и поскольку AppBarLayout завершается при сворачивании, FAB также исчезнет!

Вкратце: NestedScrollingViewимеет Behavior, который заставит AppBarLayout прокручиваться вместе с ним. AppBarLayout имеет Behavior, который позволяет ему сворачиваться, и FAB подключен к этому макету, чтобы перемещаться вместе с ним и в конечном итоге исчезать.

**Индивидуальное поведение**

Мы также можем создать наши собственные пользовательские поведения, если мы хотим изменить способ взаимодействия элементов с CoordinatorLayout. Например, мы можем создать Поведение, чтобы FAB на сжимался MainActivityи исчезал, когда отображается Snackbar, а не убирался с дороги!

Сначала мы создадим новый класс Java для представления нашего ShrinkBehavior. Этот класс понадобится extend CoordinatorLayout.Behavior<FloatingActionButton>(потому что он является , CoordinatorLayout.Behaviorи он будет применен к FloatingActionButton).

* Нам также потребуется переопределить конструктор, чтобы мы могли объявить/создать этот класс из XML:
* **public** ShrinkBehavior(Context context, AttributeSet attrs) {
* **super**(context, attrs);

}

Следующий шаг — убедиться, что Behavior способен реагировать на изменения в Snackbar. Для этого нам нужно сделать отчет Behavior, который имеет Snackbar в качестве *зависимости* . Таким образом, когда CoordinatorLayout распространяет события и изменения на все свои дочерние элементы, он будет знать, что он также должен информировать FAB об изменениях в Snackbar. Мы делаем это, переопределяя layoutDependsOn()метод:

**public** boolean layoutDependsOn(CoordinatorLayout parent,

FloatingActionButton child, View dependency) {

*//add SnackbarLayout to the dependency list (if any)*

**return** dependency **instanceof** Snackbar.SnackbarLayout ||

**super**.layoutDependsOn(parent, child, dependency);

}

* (Технически superкласс не имеет никаких других зависимостей, но все равно хорошей практикой является вызов дерева).

Наконец, мы можем указать, что должно произойти при изменении одного из представлений зависимости, переопределив onDependentViewChange()обратный вызов:

**public** boolean onDependentViewChanged(CoordinatorLayout parent, FloatingActionButton child, View dependency) {

**if**(dependency **instanceof** Snackbar.SnackbarLayout){

*//calculate how much Snackbar we see*

float snackbarOffset = 0;

**if**(parent.doViewsOverlap(child, dependency)){

snackbarOffset = Math.min(snackbarOffset, dependency.getTranslationY() - dependency.getHeight());

}

float scaleFactor = 1 - (-snackbarOffset/dependency.getHeight());

child.setScaleX(scaleFactor);

child.setScaleY(scaleFactor);

**return** **true**;

}**else** {

**return** **super**.onDependentViewChanged(parent, child, dependency);

}

}

* Этот метод будет передан ссылке на , CoordinatorLayoutкоторый управляет изменениями, FloatingActionButtonкто получает изменение и *какая зависимость* изменила свое View. Мы проверяем, что зависимость на самом деле является Snackbar(так как у нас может быть несколько зависимостей и мы хотим реагировать по-разному на каждую из них), а затем вызываем некоторые геттеры для этой зависимости, чтобы выяснить, насколько она высока (и, следовательно, насколько мы должны уменьшиться). Наконец, мы используем сеттеры для изменения масштабирования child(FAB), тем самым заставляя его масштабироваться!
* А поскольку этот масштаб зависит от высоты Snackbar, FAB также «вырастет снова», когда Snackbar исчезнет!

Это самая простая форма поведения, которую мы можем иметь: более сложное поведение может быть основано на событиях прокрутки или перетаскивания, использующих различные атрибуты состояния для различных зависимостей!

Пользовательские поведения очень сложны в написании и проектировании; переопределенные функции не очень хорошо документированы, и по определению эти поведения подразумевают координацию множества различных классов! Большинство пользовательских поведений разрабатываются путем чтения исходного кода Android (например, для FloatingActionButton.Behavior) и его модификации в качестве примера. Я предлагаю поискать в Интернете поведения, похожие на то, которое вы пытаетесь достичь, и работать с этим.

**5.4 Анимации**

Одним из ключевых принципов Material Design было использование *движения* : многие из предыдущих примеров включали добавление анимированных изменений к элементам (например, перемещение или прокрутку). Тема Material, доступная в API 21+, предоставляет ряд различных методов для включения [**анимации**](https://developer.android.com/training/material/animations.html) в ваше приложение, в частности, использование анимации для предоставления обратной связи пользователю и обеспечения связей между элементами при изменении отображения. В качестве последнего примера в этом разделе будет рассмотрено, как добавлять простые [переходы Activity](https://developer.android.com/training/material/animations.html#Transitions) , чтобы Views «трансформировались» из одного Activity в другой.

Чтобы использовать переходы между действиями, вам необходимо включить их в *тему* вашего приложения , добавив дополнение <item>в объявление темы (в res/values/styles.xml):

*<!-- in style: enable window content transitions -->*

**<item** name="android:windowActivityTransitions"**>**true**</item>**

Мы можем указать три различных типа переходов активности:

1. *входные* переходы, или как представления в действии попадают на экран
2. *переходы выхода* , или как представления в действии покидают экран
3. *переходы общих элементов* или как изменяются представления, которые являются общими для разных видов деятельности

Мы поговорим о последнем, хотя предыдущие два следуют схожему процессу.

Чтобы анимировать **общий элемент** между двумя Activity, нам нужно дать им соответствующие идентификаторы, чтобы фреймворк перехода знал, что их нужно анимировать. Мы делаем это, давая каждому из общих элементов атрибут android:transitionName, гарантируя, что они имеют *одинаковое* значение.

* Например, мы можем присвоить FAB в каждом действии android:transitionName="fab".
* Поскольку FAB *закреплен* , нам на самом деле нужно проделать некоторую дополнительную работу (потому что FAB трансформируется в «незакрепленную» точку, а затем будет перемещен после расчета закрепления). Самый простой способ обойти это — обернуть закрепленный FAB внутрь закрепленного FrameLayout— FAB просто тогда станет обычным элементом с FrameLayout, обрабатывающим поведение прокрутки.
* **<FrameLayout**
* android:layout\_width="wrap\_content"
* android:layout\_height="wrap\_content"
* app:layout\_anchor="@id/app\_bar"
* app:layout\_anchorGravity="bottom|end"
* android:elevation="12dp"
* **>**
* **<android.support.design.widget.FloatingActionButton**
* android:id="@+id/fab"
* android:transitionName="same\_fab"
* android:layout\_width="wrap\_content"
* android:layout\_height="wrap\_content"
* android:layout\_margin="@dimen/fab\_margin"
* app:srcCompat="@android:drawable/ic\_dialog\_email" **/>**

**</FrameLayout>**

Наконец, нам нужно убедиться, что Intent, используемый для запуска Activity, также запускает анимацию перехода. Мы делаем это, включая дополнительный аргумент в startActivity()метод: параметры, Bundleсодержащие сведения об анимации:

*//transition this single item*

ActivityOptions options = ActivityOptions.makeSceneTransitionAnimation(MainActivity.this, button, "fab");

*// start the new activity*

startActivity(**new** Intent(MainActivity.this, ScrollingActivity.class), options.toBundle());

Это должно привести к тому, что FAB будет «трансформироваться» между действиями (и даже трансформироваться обратно, когда вы нажмете кнопку «назад»)!

Дополнительную информацию о том, что делает анимацию эффективной, см. [в руководстве по дизайну материалов](https://material.io/guidelines/motion/material-motion.html) .

Я считаю Activity Transitions гладкими, но капризными: чтобы заставить их работать правильно, требуется много усилий и практики. Большинство профессиональных приложений, использующих Material Design, используют обширные пользовательские анимации для создания этих переходов. Для примеров более сложных анимаций и шаблонов Material Design ознакомьтесь с образцами приложений, такими как [cheesesquare](https://github.com/chrisbanes/cheesesquare) или [plaid](https://github.com/nickbutcher/plaid) .

**Ресурсы**

* [Руководство по материальному дизайну](https://material.io/guidelines/) проверяет весь документ (через меню-гамбургер слева).
* [Material Design for Developers (Google)](https://developer.android.com/training/material/index.html) официальная документация по внедрению Material Design
* [Material Design Primer (CodePath) —](https://guides.codepath.com/android/Material-Design-Primer) превосходно скомпилированная документация и примеры для реализации шаблонов Material (CodePath в целом — отличный ресурс).
* [Библиотека поддержки дизайна Android (Блог Google)](https://android-developers.googleblog.com/2015/05/android-design-support-library.html) — введение в функции библиотеки поддержки
* [Освоение макета Coordinator (блог) —](http://saulmm.github.io/mastering-coordinator) отличный набор примеров использования CoordinatorLayout.
* [Использование CoordinatorLayout в приложениях Android (Блог)](http://www.androidauthority.com/using-coordinatorlayout-android-apps-703720/) еще одно хорошее объяснение CoordinatorLayout
* <https://lab.getbase.com/introduction-to-coordinator-layout-on-android/>
* <https://medium.com/@andkulikov/animate-all-the-things-transitions-in-android-914af5477d50>

1. <http://developer.android.com/reference/android/support/v7/app/ActionBar.html>[↩](https://info448.github.io/material-design.html#fnref14)
2. <https://material.io/color/#!/>[↩](https://info448.github.io/material-design.html#fnref15)
3. <https://developer.android.com/guide/topics/ui/layout/recyclerview>[↩](https://info448.github.io/material-design.html#fnref16)