Material Design

**Material Design** был представлен вместе с Android 5 в 2014 году.

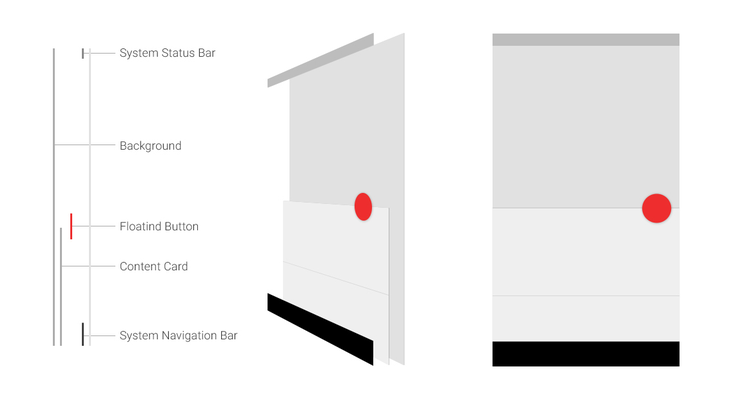
С релизом **Material Design** Android наконец-то получил не только гайдлайны по дизайну приложений, но и философию дизайна. До его релиза в Android использовался **Holo**, и не было полноценных гайдов по дизайну. Кроме того, Holo был достаточно некрасив, и разработчикам, которые хотели выпустить симпатичное приложение приходилось нанимать дизайнера.

С выходом же Material Design любой разработчик может создать приложение, которое будет прекрасно выглядеть, не привлекая никого "со стороны".

Так что же такое Material Design? Google называет его "визуальным языком", с помощью которого можно представить на экране физические объекты, при этом не используется так называемый скевоморфизм, типичный для iOS до версии 7.

Вместо этого используется **Flat**, при этом объекты расположены по трём осям, и "высота" (по оси Z) выделяется при помощи теней.

Взгляните на эту картинку:

[](https://skillberg.com/media/uploads/2017/11/17/0a57bfbb81b54ee39d3b9f0ac1bdd4f8.png)Z-index в Material Design

На ней вы видите, что элементы расположены "слоями" друг над другом, и это подчёркивается тенями. Такой подход даёт пользователю ощущение, что он взаимодействует с реальными объектами, но при этом не отходит от общего тренда на флэт-дизайн.

Основные принципы Material Design

1. **Тактильные поверхности.** Как вы видели выше, интерфейс в Material Design строится из слоёв, находящихся на разной высоте друг над другом, и помогают пользователю лучше понимать структуру интерфейса. При этом эти слои выглядят, как листы бумаги, поэтому пользователь как бы взаимодействует с реальной "бумагой".
2. **Полиграфический дизайн.** Поскольку слои интерфейса — это "бумага", типографика невероятно важна. Современный дизайн строится на принципе "Content First", то есть самое главное в интерфейсе — это контент. А поскольку основным контентом в большинстве приложений является текст, очень важно, чтобы текст читался максимально хорошо.
3. **Осмысленная анимация.** В настоящем мире предметы не возникают из ниоткуда (и не исчезают в никуда), поэтому важно, чтобы все анимации были похожи на поведение объектов в реальности.
4. **Адаптивный дизайн.** Дизайн должен адаптироваться под различные устройства, соблюдая при этом принципы, изложенные выше.

Изучаем Material Design

Самое крутое, что есть в **Material Design** — это документация. На [официальном сайте](https://material.io/guidelines/?hl=ru) вы найдёте подробнейшие гайды, используя которые можно построить идеальный интерфейс, в том числе примеры, как нужно поступать, а как не стоит.

**Обратите внимание:** обязательно полностью прочитайте гайды на сайте, и регулярно обращайтесь к ним при разработке.

Я видел очень много хороших Android-разработчиков и отличных приложений, и заметил один факт: большинство разработчиков — очень плохие дизайнеры.

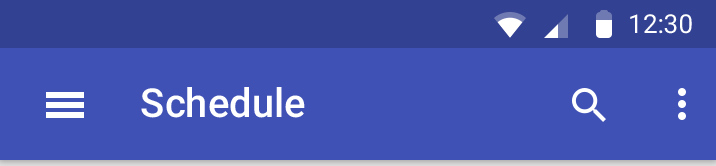
Гайдов по Material Design вполне достаточно, чтобы делать очень неплохие интерфейсы.

Поскольку наш ресурс — не документация, а проводник в мир Android-разработки, я не буду разбирать дизайн во всех подробностях, а лишь объясню основные принципы и паттерны в достаточном объёме.

Давайте рассмотрим на конкретных примерах, как использовать **Material Design** в своих проектах.

Toolbar

**Toolbar** — специальный View, который обычно отображается в верхней части экрана.

[](https://skillberg.com/media/uploads/2017/11/17/12b5b86da7134481a7d531483415b52b.png)Toolbar в Android

Обычно в нём находятся заголовок экрана и различные иконки действий (меню).

До Android 5.0 те же функции выполнял **ActionBar**, и хотя визуально в общем случае Toolbar не отличается от ActionBar, технически это совершенно разные вещи.

Тогда как ActionBar — часть окна, в котором расположена Activity, Toolbar — это View внутри иерархии View в Activity, и такой подход позволяет достичь невероятной гибкости при создании интерфейсов.

Toolbar поддерживается начиная с Android 5. К счастью, в отличие от ActionBar, Toolbar был портирован Google для версий Android ниже Lollipop, и для его использования достаточно лишь подключить библиотеку appcompat-v7, которая, кстати, подключена по умолчанию в нашем проекте:

dependencies {

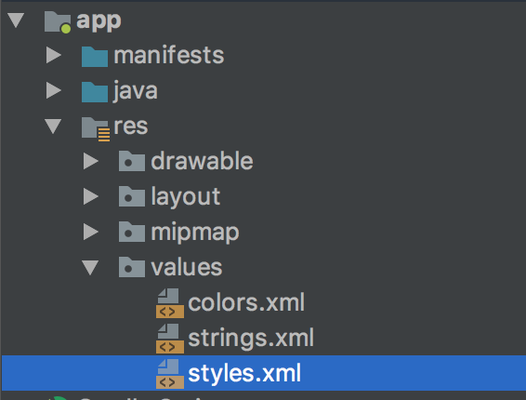
// ...

implementation 'com.android.support:appcompat-v7:26.1.0'

// ...

Поскольку мы работаем с эмулятором, версия которого выше Lollipop, Toolbar отображается без каких-либо телодвижений. Тем не менее, в реалиях Android нужно поддерживать и более ранние версии Android.

В первую очередь нам нужно изменить тему приложения. Android позволяет кастомизировать интерфейс приложений, и мы можем сделать это, в том числе, используя темы. Откройте файл res/values/styles.xml:

[](https://skillberg.com/media/uploads/2017/11/17/2e18e3c8a1314ef5976f5f9d1152030a.png)Файл styles.xml

В нём вы увидите такой код:

<resources>

<!-- Base application theme. -->

<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">

<!-- Customize your theme here. -->

<item name="colorPrimary">@color/colorPrimary</item>

<item name="colorPrimaryDark">@color/colorPrimaryDark</item>

<item name="colorAccent">@color/colorAccent</item>

</style>

</resources>

Как вы видите, в нём задана дефолтная тема оформления, подразумевающая использование тёмного ActionBar.

Эта тема наследуется от системной Theme.AppCompat.Light.DarkActionBar. Чтобы использовать Toolbar, нужно добавить тему, убирающую ActionBar и создать тему для Toolbar. Давайте сделаем это:

<resources>

<!-- Base application theme. -->

<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">

<!-- Customize your theme here. -->

<item name="colorPrimary">@color/colorPrimary</item>

<item name="colorPrimaryDark">@color/colorPrimaryDark</item>

<item name="colorAccent">@color/colorAccent</item>

</style>

<style name="AppTheme.NoActionBar">

<item name="windowActionBar">false</item>

<item name="windowNoTitle">true</item>

</style>

<style name="Toolbar" parent="ThemeOverlay.AppCompat.Dark.ActionBar">

<item name="android:textColorPrimary">@android:color/white</item>

<item name="android:textColorSecondary">@android:color/white</item>

</style>

</resources>

Так же нужно установить новую тему в AndroidManifest.xml:

<application

android:allowBackup="true"

android:icon="@mipmap/ic\_launcher"

android:label="@string/app\_name"

android:roundIcon="@mipmap/ic\_launcher\_round"

android:supportsRtl="true"

android:theme="@style/AppTheme.NoActionBar">

Теперь у нас установлена светлая тема (т.е. темный контент на светлом фоне) без ActionBar.

Поскольку Toolbar является обычным View, давайте создадим лэйаут для него. Создайте новый файл toolbar.xml в директории res/layout со следующим содержимым:

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

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

android:id="@+id/toolbar"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:theme="@style/Toolbar"

android:minHeight="?android:attr/actionBarSize" />

Нам понадобится лэйаут под названием CoordinatorLayout, находящийся в библиотеке AppCompat Design, поэтому подключите эту библиотеку в build.gradle:

implementation 'com.android.support:design:26.1.0'

Всегда старайтесь использовать актуальную версию Support-библиотек. На момент создания этого урока актуальной является 26.1.0, но к тому моменту, когда вы будете читать этот урок, скорее всего уже выйдет несколько обновлений для неё. Android Studio сообщит вам об этом — главное, чтобы у всех Support-библиотек была одна версия.

Теперь мы можем включить этот файл в разметку activity\_main.xml, и она станет выглядеть так:

<?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"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context="com.skillberg.notes.MainActivity">

<android.support.design.widget.AppBarLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content">

<include layout="@layout/toolbar" />

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

<android.support.v7.widget.RecyclerView

android:id="@+id/notes\_rv"

android:layout\_width="match\_parent"

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

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

Что изменилось?

1. Мы заменили FrameLayout на CoordinatorLayout. CoordinatorLayout — это серьёзно изменённый FrameLayout, наделённый дополнительными возможностями. В современных приложениях, использующих Material Design он почти всегда является дефолтным лэйаутом.
2. Мы добавили AppBarLayout, и включили в него Toolbar.

AppBarLayout наследуется от LinearLayout и предназначен для размещения в нём Toolbar.

Тег <include/> позволяет включать уже имеющийся лэйаут в другой лэйаут. По сути, тот код, что мы написали, аналогичен вот такому:

<?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"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context="com.skillberg.notes.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="wrap\_content"

android:theme="@style/Toolbar"

android:minHeight="?android:attr/actionBarSize" />

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

<android.support.v7.widget.RecyclerView

android:id="@+id/notes\_rv"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

app:layout\_behavior="@string/appbar\_scrolling\_view\_behavior" />

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

Преимущество такого подхода в том, что мы можем задать вёрстку элемента один раз и использовать её неограниченное количество раз в других частях проекта.

Обратите внимание на атрибут app:layout\_behaviour. Понятие Behaviour — достаточно обширная тема. Как можно догадаться по названию, атрибут отвечает за поведение элемента в лэйауте.

Конкретно в данном случае мы говорим, что наш RecyclerView должен скроллиться. Задав дополнительные атрибуты, мы можем, например, заставить Toolbar автоматически скрываться при скролле (пока что мы не будем этого делать).

Помимо этого, благодаря этому атрибуту RecyclerView не будет уезжать под Toolbar (попробуйте удалить его, и увидите, что первый элемент списка скрыт под тулбаром).

Осталось лишь установить Toolbar в MainActivity:

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

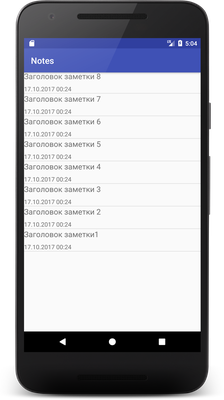
setContentView(R.layout.activity\_main);

Toolbar toolbar = findViewById(R.id.toolbar);

setSupportActionBar(toolbar);

// ...

Запустите приложение, и вы увидите…

[](https://skillberg.com/media/uploads/2017/11/17/7cb104a577d94174a9abde0bc612cac0.png)Toolbar в Android

… что ничего не поменялось. Но это лишь внешне. На самом деле, приложение теперь использует Toolbar, в том числе на Android ниже пятой версии!

Если вы откроете файл res/values/colors.xml, то увидите там такой код:

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

<resources>

<color name="colorPrimary">#3F51B5</color>

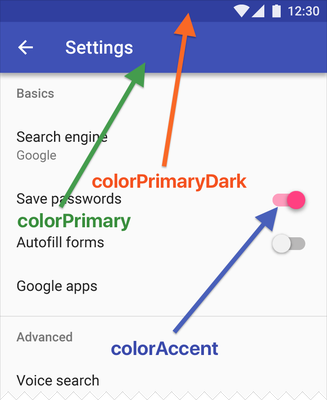
<color name="colorPrimaryDark">#303F9F</color>

<color name="colorAccent">#FF4081</color>

</resources>

Он, как не трудно догадаться, отвечает за цвета, используемые приложением.

Что это за цвета такие?

[](https://skillberg.com/media/uploads/2017/11/17/2fc5fb1ac8574a95860a61944b9f83f9.png)Цвета в Material Design теме

* colorPrimary — основной цвет приложения. В него окрашивается, например, Toolbar.
* colorPrimaryDark — затемнённый основной цвет приложения. В него окрашивается StatusBar, расположенный чуть выше Toolbar.
* colorAccent — в него окрашиваются такие элементы интерфейса, как кнопки, переключатели и т.д. Как правило, используется цвет, отличный от colorPrimary.

Для цвета текста используются следующие правила (при условии использования светлого фона):

* **Основной текст** — чёрный с прозрачностью 87% (ARGB: #DD000000). Используется, как ни странно, для основного (самого важного) текста на экране.
* **Второстепенный текст** — чёрный с прозрачностью 54% (ARGB: #89000000). Используется для текста, который расположен в иерархии ниже, чем основной текст.
* **Выключенный текст** — чёрный с прозрачностью 38% (ARGB: #61000000). Используется для текста на "выключенных" (недоступных) элементах и, например, в подсказках.

Можно заранее добавить эти цвета в colors.xml:

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

<resources>

<color name="colorPrimary">#3F51B5</color>

<color name="colorPrimaryDark">#303F9F</color>

<color name="colorAccent">#FF4081</color>

<color name="colorTextPrimary">#DD000000</color>

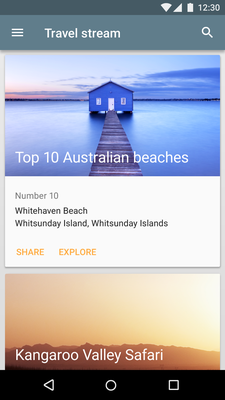
<color name="colorTextSecondary">#89000000</color>

<color name="colorTextDisabled">#61000000</color>

</resources>

CardView

Карточка — один из самых часто используемых компонентов в Material Design. Мы будем использовать их в списке заметок.

[](https://skillberg.com/media/uploads/2017/11/17/be45e427cb92463bbcb05f1aed697709.png)Пример CardView в Android

Вообще, Google не рекомендует использовать карточки в списках, так как они могут отвлекать пользователя от контента. Однако, вполне можно использовать плоские карточки без тени, которые будут выглядеть симпатично и не мешать просмотру контента.

Для начала добавьте AppCompat-библиотеку для CardView:

implementation 'com.android.support:cardview-v7:26.1.0'

Теперь откройте view\_item\_note.xml и замените код на следующий:

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

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

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_marginEnd="4dp"

android:layout\_marginStart="4dp"

android:layout\_marginTop="8dp">

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:orientation="vertical">

<TextView

android:id="@+id/title\_tv"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textSize="18sp" />

<TextView

android:id="@+id/date\_tv"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_marginTop="8dp"

android:textSize="14sp" />

</LinearLayout>

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

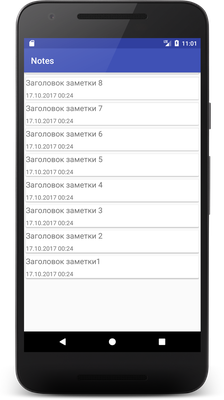
**Обратите внимание**: CardView должен содержать только **один** дочерний View! Если вам нужно разместить в карточке больше одного View, то "оберните" их в ViewGroup, к примеру, как в нашем случае.

В MainActivity удалите код, отвечающий за разделители, то есть вот этот:

layoutManager.getOrientation());

recyclerView.addItemDecoration(dividerItemDecoration);

Запустив приложение вы увидите следующее:

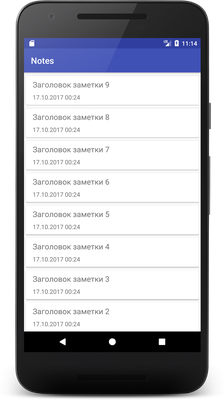
[](https://skillberg.com/media/uploads/2017/11/17/288c615d007647e89a4b58ae77350bee.png)Стандартный CardView без отступов

Выглядит страшновато, но карточки уже показываются. Давайте улучшать их внешний вид — начнём с отступов внутри карточки.

Задайте у LinearLayout следующие атрибуты:

* android:paddingBottom равный 8dp.
* android:paddingTop равный 8dp.
* android:paddingEnd равный 16dp.
* android:paddingStart равный 16dp.

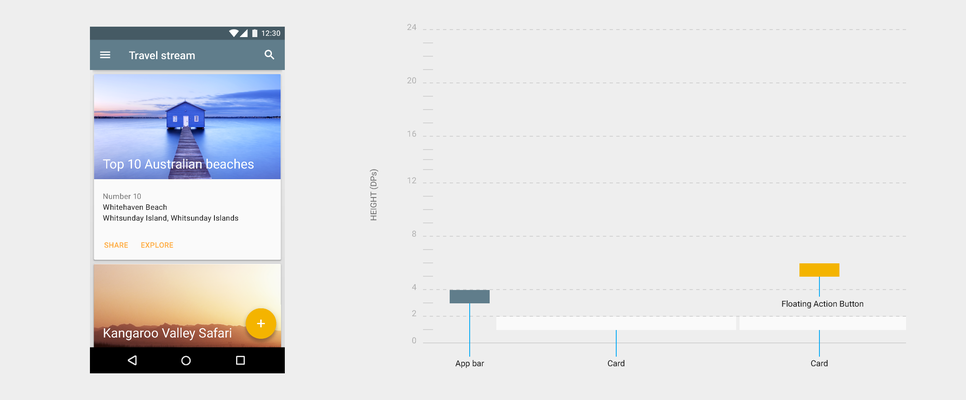
Запустите приложение:

[](https://skillberg.com/media/uploads/2017/11/17/eebd085f188648348ffb4039ba8a07e5.png)CardView с внутренними отступами

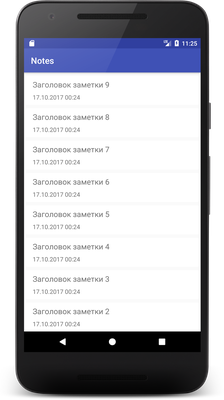
Уже лучше!

Теперь нужно убрать тени. Сейчас карточка как бы выпирает вверх, так как у неё есть определённая "высота" (по оси Z), и потому отбрасывает тень.

Если вам ещё непонятно, как это работает, то взгляните на это изображение:

[](https://skillberg.com/media/uploads/2017/11/17/f52162d0486d4065bb7a264f3e665fc3.png)

Большая высота подходит для больших карточек, но на маленьких выглядит очень уродливо. Управлять этим параметром мы можем используя атрибут elevation, а у карточек — app:cardElevation. Установите его равным 0dp, и увидите, что тени исчезли.

[](https://skillberg.com/media/uploads/2017/11/17/c3c7643b8ed244138c9fc46b2b614e06.png)CardView без тени (app:cardElevation="0dp")

Стало существенно лучше.

Типографика

Теперь поработаем над текстом. Достаточно изменить цвет, немного подправить размер и установить подходящее семейство шрифтов:

<TextView

android:id="@+id/title\_tv"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:fontFamily="sans-serif-light"

android:textColor="@color/colorTextPrimary"

android:textSize="20sp" />

<TextView

android:id="@+id/date\_tv"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_marginTop="8dp"

android:fontFamily="sans-serif"

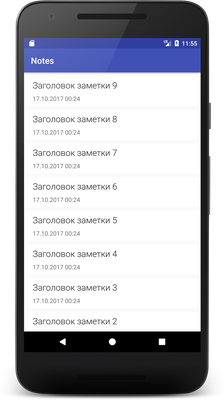
android:textColor="@color/colorTextSecondary"

android:textSize="14sp" />

Что мы сделали?

* Изменили семейство шрифтов, используя атрибут android:fontFamily.
* Увеличили текст заголовка.
* Для заголовка задали основной цвет текста, а для даты — второстепенный. Эти цвета мы ранее объявляли в colors.xml.

А вот что получилось:

[](https://skillberg.com/media/uploads/2017/11/17/38df8550224c4f89a16c5af6772f63e8.png)CardView с улучшенной типографикой

Выглядит не идеально, но хорошо.

На этом, пожалуй, остановимся. В следующем уроке мы будем учиться создавать новые заметки.

ПРИМЕРЫ КОДА

[ИСХОДНЫЙ КОД УРОКА](https://github.com/Skillberg/NotesAndroid/tree/lesson-23)