IT ШКОЛА SAMSUNG

Курс обучения преподавателей

Элементы Material Design

Методические материалы

Михаил Варакин

Август 2015

Оглавление

Material Design: цели и принципы

Цитата с официального сайта (http://www.google.com/design/spec/material-design/introduction.html):

Goals

> «Create a visual language that synthesizes classic principles of good design with the innovation and possibility of technology and science.
> Develop a single underlying system that allows for a unified experience across platforms and device sizes. Mobile precepts are fundamental, but touch, voice, mouse, and keyboard are all first-class input methods.»

Principles

> «**Material is the metaphor**
> A material metaphor is the unifying theory of a rationalized space and a system of motion. The material is grounded in tactile reality, inspired by the study of paper and ink, yet technologically advanced and open to imagination and magic.
> Surfaces and edges of the material provide visual cues that are grounded in reality. The use of familiar tactile attributes helps users quickly understand affordances. Yet the flexibility of the material creates new affordances that supercede those in the physical world, without breaking the rules of physics.
> The fundamentals of light, surface, and movement are key to conveying how objects move, interact, and exist in space and in relation to each other. Realistic lighting shows seams, divides space, and indicates moving parts.»

> «**Bold, graphic, intentional**
> The foundational elements of print-based design—typography, grids, space, scale, color, and use of imagery—guide visual treatments. These elements do far more than please the eye. They create hierarchy, meaning, and focus. Deliberate color choices, edge-to-edge imagery, large-scale typography, and intentional white space create a bold and graphic interface that immerse the user in the experience.
> An emphasis on user actions makes core functionality immediately apparent and provides waypoints for the user.»

> «**Motion provides meaning**
> Motion respects and reinforces the user as the prime mover. Primary user actions are inflection points that initiate motion, transforming the whole design.
> All action takes place in a single environment. Objects are presented to the user without breaking the continuity of experience even as they transform and reorganize.
> Motion is meaningful and appropriate, serving to focus attention and maintain continuity. Feedback is subtle yet clear. Transitions are efficient yet coherent.»

Библиотека поддержки AppCompat

Для использования возможностей, появившихся в API21+, на устройствах с более старыми версиями *Android*, *Google* предоставляет библиотеку поддержки *AppCompat*. В *Android Studio* при создании нового проекта требуется добавить нужную строку (с правильной версией библиотеки!) в секцию *dependencies* для соответствующего модуля (по умолчанию это модуль «*app*»):

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:design:23.0.+'
}
```

В любом случае, использование этой библиотеки подразумевает выполнение разработчиком нескольких условий:

- Все *Activity* являются наследниками класса *AppCompatActivity*
- Все темы являются наследниками тем *Theme.AppCompat**

Следует учесть, что библиотека *AppCompat* в настоящий момент поддерживает не все возможности, доступные разработчику в API21+, поэтому при работе с *AppCompat* требуется приложить несколько больше усилий и стоит учитывать имеющиеся ограничения:

Несколько вариантов стилей

Атрибуты визуальных тем из *Material Design* полноценно поддерживаются только в современных (*Lollipop* и новее) версиях *Android*, а для доледенцовых устройств приходится «договариваться» с *AppCompat*. На практике это означает, что разработчик должен поддерживать два набора тем: «родные темы» для API21+ (обфчно в **values-v21/styles.xml**) и «костыли» для остальных версий (обфчно в **values/styles.xml**). Чисто текстуально разница между темами обычно заключается в наличии префикса «*android:*» в «родных» темах:

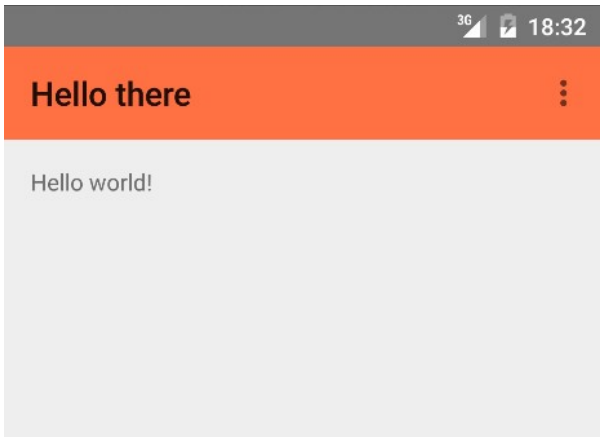
values-v21/styles.xml

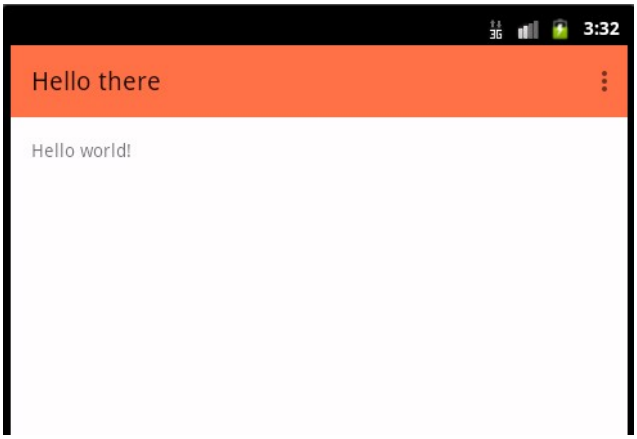
```
<!-- Base application theme. -->
<style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
    <item name="android:windowNoTitle">true</item>
    <!-- colorPrimary - цвет фона Toolbar -->
    <item name="android:colorPrimary">@color/colorPrimary</item>
</style>
```

values/styles.xml

```
<!-- Base application theme. -->
<style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
    <!-- colorPrimary - цвет фона Toolbar -->
    <item name="colorPrimary">@color/colorPrimary</item>
</style>
```

Результат на разных версиях будет, конечно же, отличаться, но общие черты будут явно видны (слева – API22, справа – API10):





AppCompat Widgets & Themes

Библиотека добавляет несколько новых виджетов:

- RecyclerView http://developer.android.com/reference/android/support/v7/widget/RecyclerView.html
- CardView http://developer.android.com/reference/android/support/v7/widget/CardView.html
- Palette http://developer.android.com/reference/android/support/v7/graphics/Palette.html

а также обеспечивает поддержку семейства тем *Theme.AppCompat** для системных виджетов:

- EditText http://developer.android.com/reference/android/widget/EditText.html
- Spinner http://developer.android.com/reference/android/widget/Spinner.html
- CheckBox http://developer.android.com/reference/android/widget/CheckBox.html
- RadioButton http://developer.android.com/reference/android/widget/RadioButton.html
- SwitchCompat http://developer.android.com/reference/android/support/v7/widget/SwitchCompat.html
- CheckedTextView http://developer.android.com/reference/android/widget/CheckedTextView.html

Для настройки атрибутов тем, унаследованных от *Theme.AppCompat**, обычно используются атрибуты *colorPrimary, colorPrimaryDark, colorAccent*.

Ограниченная поддержка некоторых операций

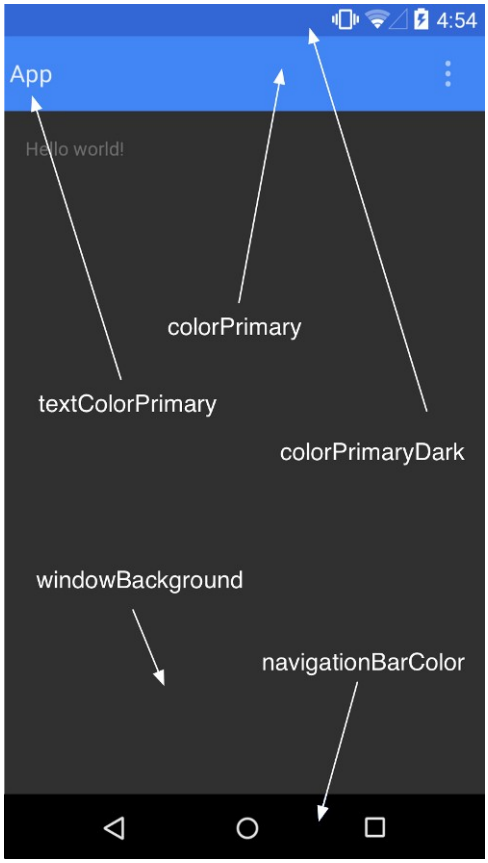
Material Design – это не только ~~ценный мех~~ стилевое оформление, но и визуальные эффекты

и новые возможности (например, векторная графика), которые (в текущей версии *AppCompat*) не полностью поддерживаются на старых версиях Android. Следовательно, перед вызовом методов, которые могут не поддерживаться в доледенцовом *Android*, имеет смысл проверить версию операционной системы:

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
      // вызываем свежие методы
} else {
      // довольствуемся тем, что есть
}
```

Toolbar – замена ActionBar

Одновременно с появлением на сцене *Material Design*, уже полюбившийся пользователям и разработчикам *ActionBar* стал *deprecated*, и теперь вместо него в UI приложений предлагается использовать его собрата – *Toolbar*. Одним из важных различий между ними является положение в иерархии UI – если ActionBar был частью оформления окна *Activity* и управлялся фреймворком UI, то Toolbar может находиться на практически любом уровне иерархии (и в любом месте на экране), и если, например, *Activity* захочет использовать Toolbar в качестве ActionBar, для это достаточно вызвать метод *setActionBar()*. В общем, возможности *Toolbar* богаче, чем у ActionBar. Так, например, на *ToolBar* можно добавлять кнопки, иконки/логотипы, заголовки и подзаголовки, пункты *action menu*,и даже собственные элементы разметки (*View*).



Использование при создании UI инновационнных и прогрессивных инструментов, доступных в *Material Design Library*, подразумевает разумное и грамотное использование цвета для различных элементов. На иллюстрации (источник http://developer.android.com) показаны имена атрибутов тем, использующиеся для отображения UI. Предлагаемые палитры цветов и рекомендации по их использованию можно найти в спецификации *Material Design*: http://www.google.com/design/spec/style/color.html

Для использования *Toolbar* в качестве *ActionBar* для Activity требуется совсем немного:

1. Выбрать тему *без ActionBar* и настроить цвета:

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <item name="android:windowNoTitle">true</item>
        <!-- colorPrimary - цвет фона Toolbar -->
        <item name="colorPrimary">@color/colorPrimary</item>
    </style>
```

```
</resources>
```

2. В разметке *для Activity* расположить *Toolbar* в нужном месте:

```
<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"
    android:orientation="vertical">

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

    <fragment . . . />

</LinearLayout>
```

3. В методе *onCreate()* найти *Toolbar*, настроить его и назначить *ActionBar*'ом:

```
public class MainActivity extends AppCompatActivity implements
SearchView.OnQueryTextListener {

    private MenuItem mSearchItem;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        toolbar.setTitle("Hello there");
        setSupportActionBar(toolbar);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_main, menu);
        mSearchItem = menu.findItem(R.id.action_search);
        SearchView mSearchview = (SearchView)
            MenuItemCompat.getActionView(mSearchItem);
        mSearchview.setOnQueryTextListener(this);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.action_say_hi:
                Toast.makeText(this, "Hi!", Toast.LENGTH_LONG).show();
                return true;
            case R.id.action_copy:
                Toast.makeText(this,
                        "Here we could copy selection to clipboard",
                        Toast.LENGTH_LONG).show();
                return true;
            case R.id.action_exit:
                finish();
```

```
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }

    @Override
    public boolean onQueryTextSubmit(String query) {
        //mSearchview.setQuery(null, false);
        Toast.makeText(this, "Here we could search '" + query + "'",
            Toast.LENGTH_LONG).show();
        MenuItemCompat.collapseActionView(mSearchItem);
        return true;
    }

    @Override
    public boolean onQueryTextChange(String newText) {
        return false;
    }
}
```

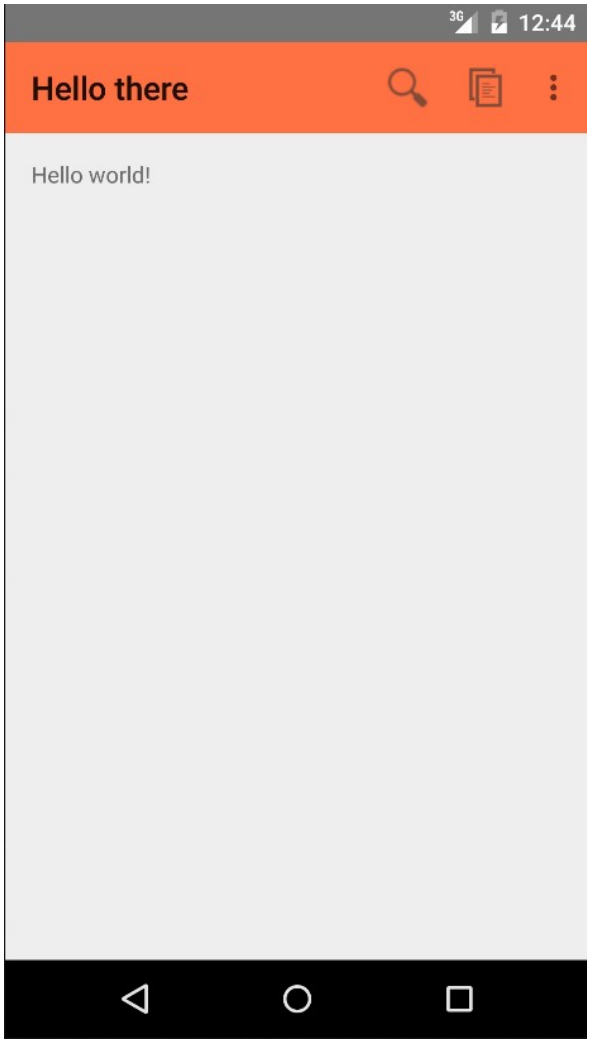
Как и в случае с обычным ActionBar, на устройствах с аппаратной кнопкой меню основное меню будет вызываться и этой кнопкой, и через область *overflow*, а на устройствах без такой кнопки – только через *overflow*. Меню обычно удобнее всего описывать в соответствующем ресурсе, как показано на примере далее. Обратите внимание на добавление дополнительного адресного пространства *XML* с именем *app:* `xmlns:app="http://schemas.android.com/apk/res-auto"` .

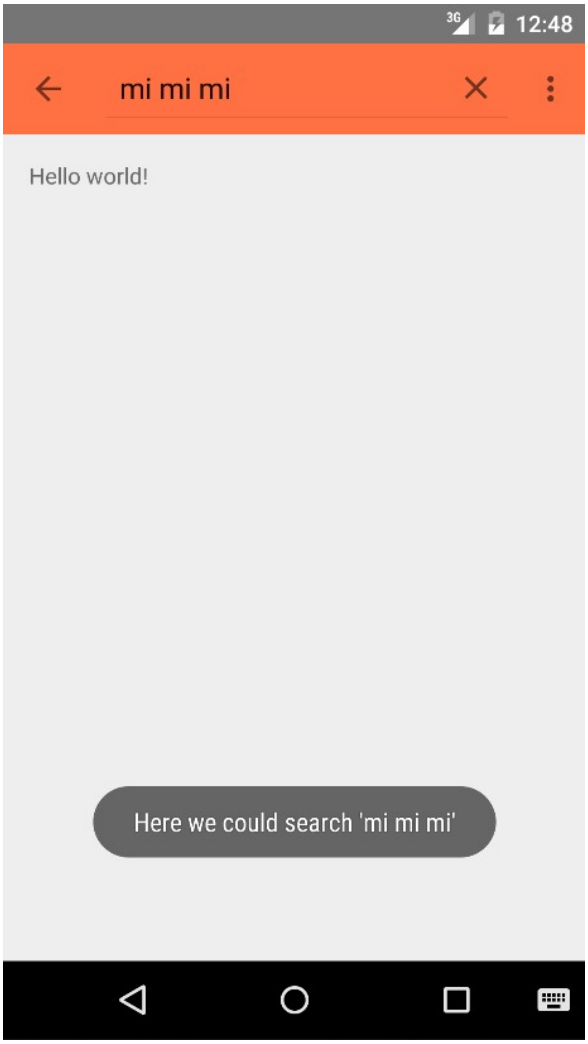
Это необходимо для корректного распознавания атрибутов, «не родных» для текущей версии API – одной стороны, в *xmlns* ***android*** и ***tools*** они не распознаются, а с другой стороны – без *xmlns* никак нельзя. Поэтому, как это часто бывает при использовании *support library*, приходится изобретать очередной «костыль»:

```
<menu 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"
    tools:context=".MainActivity">
    <item
        android:id="@+id/action_search"
        android:icon="@drawable/ic_menu_search_holo_light"
        android:title="@string/action_search"
        app:actionViewClass="android.support.v7.widget.SearchView"
        app:showAsAction="ifRoom|collapseActionView" />
    <item
        android:id="@+id/action_copy"
        android:icon="@drawable/ic_menu_copy_holo_light"
        android:title="@string/action_copy"
        app:showAsAction="ifRoom" />
    <item
        android:id="@+id/action_say_hi"
        android:title="@string/action_say_hi"
        app:showAsAction="never" />
    <item
        android:id="@+id/action_exit"
        android:title="@string/action_exit"
        app:showAsAction="never" />
</menu>
```

Как видно из примера, обычные возможности *ActionBar*, доступны и в *Toolbar*. Подробнее с этими возможностями можно познакомиться в руководстве, посвященном *ActionBar*: http://developer.android.com/guide/topics/ui/actionbar.html

Ниже показан результат описанных выше действий.





При необходимости добавить еще один *Toolbar*, например, внизу экрана, это можно сделать, просто разместив его внутри нужного *Layout*'а в соответствующем месте. Единственное отличие этого (нижнего) Toolbar'а заключается в том, это не ActionBar, поэтому «меню» для него должно включаться по-другому:

```
Toolbar bottomToolbar = (Toolbar) findViewById(R.id.toolbar_bottom);
bottomToolbar.setTitle("Hi from bottom");
bottomToolbar.inflateMenu(R.menu.menu_main_bottom);
bottomToolbar.setOnMenuItemClickListener(new Toolbar.OnMenuItemClickListener() {
    @Override
    public boolean onMenuItemClick(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.action_say_hello:
                Toast.makeText(MainActivity.this, "Hello!",
                        Toast.LENGTH_LONG).show();
                return true;
            case R.id.action_share:
                Toast.makeText(MainActivity.this,
                        "Here we could share something",
```

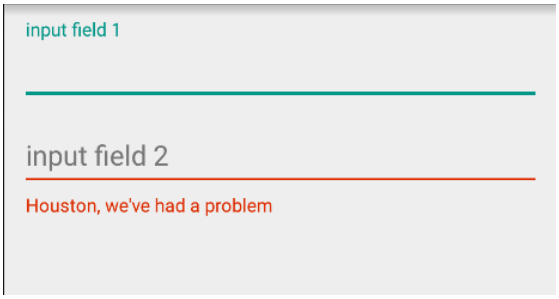
```
                        Toast.LENGTH_LONG).show();
                return true;
            case R.id.action_quit:
                finish();
                return true;
            default:
                return false;
        }
    }
});
```

А разметка *MainActivity* могла бы выглядеть так:

```
<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"
    android:orientation="vertical">
    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/colorPrimary"/>
    <fragment
        android:id="@+id/fragment"
        android:name=
            "com.example.designsupportlibrarysample1.MainActivityFragment"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        tools:layout="@layout/fragment_main"
        android:layout_weight="1"/>

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar_bottom"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/colorPrimary"/>
</LinearLayout>
```

Floating labels

Одно из не очень больших, но весьма полезных новшеств – так называемые «плавающие метки». Они решают важную задачу при вводе информации пользователем в поле *EditText:* подсказка (*hint*) для этого поля не исчезает при получении фокуса, а «уплывает» вверх, оставаясь все время видимой. Кроме того, при валидации ввода можно выводить сообщения об ошибках (методом *setError()*), которые будут органично выглядеть и на которые пользователь обязательно обратит внимание:



Для этих целей предлагается использовать «оболочку» вокруг EditText'a – новый элемент *android.support.design.widget.TextInputLayout:*

```
<android.support.design.widget.TextInputLayout
    android:id="@+id/field1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="input field 1" />

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

Получить доступ к «обернутому» полю *EditText* можно с помощью метода *getEditText().*

Подробности доступны здесь:
http://developer.android.com/reference/android/support/design/widget/TextInputLayout.html

Floating Action Button, Snakebar и CoordinatorLayout

Floating Action Button

«Фирменным» знаком *Material Design* является Плавающая Кнопка, но свои чудесные свойства они получила благодаря новому элементу *CoordinatorLayout*, внутри которого она (и не только она) размещается для получения . В самом простом случае для использования *Floating Action Button* достаточно следующей разметки:

```
<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">

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

        <LinearLayout
            android:id="@+id/viewA"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_weight="0.1"
            android:background="#DDDDDD"
            android:orientation="horizontal"/>

        <LinearLayout
            android:id="@+id/viewB"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_weight="0.9"
            android:background="#DDDDDD"
            android:orientation="horizontal"/>
    </LinearLayout>
```

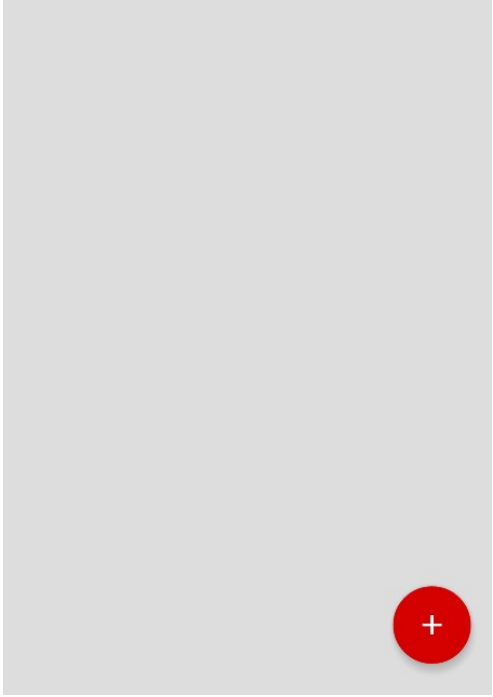
```
    <android.support.design.widget.FloatingActionButton
        android:id="@+id/fab"
        android:onClick="onButtonClick"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="16dp"
        android:clickable="true"
        android:src="@drawable/ic_add_white"
        app:layout_anchor="@id/viewA"
        app:layout_anchorGravity="bottom|end"/>

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

При таком варианте разметки FAB будет расположена в правой нижней части viewA, к которому она «привязана» тэгом *app:layout_anchor*.

Для FAB требуется указать изображение, которое она будет показывать, а цвет для кнопки либо берется из темы (атрибут *colorAccent*):

```
<style name="AppTheme"
parent="Theme.AppCompat.Light.DarkActionBar">
     <item name="colorAccent">#D50000</item>
</style>
```

либо из атрибута *app:backgroundTint* самой кнопки. Из кода цвет кнопки можно менять, вызвав её метод *setBackgroundTintList*().

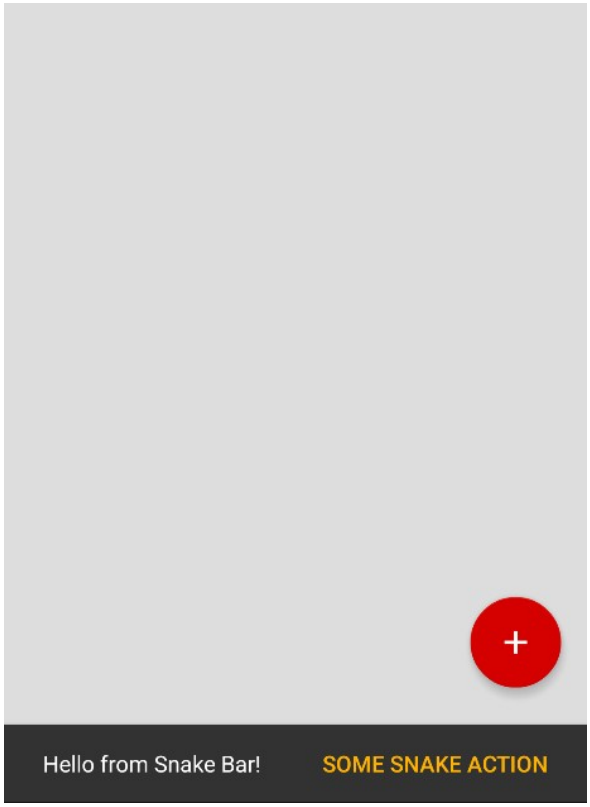


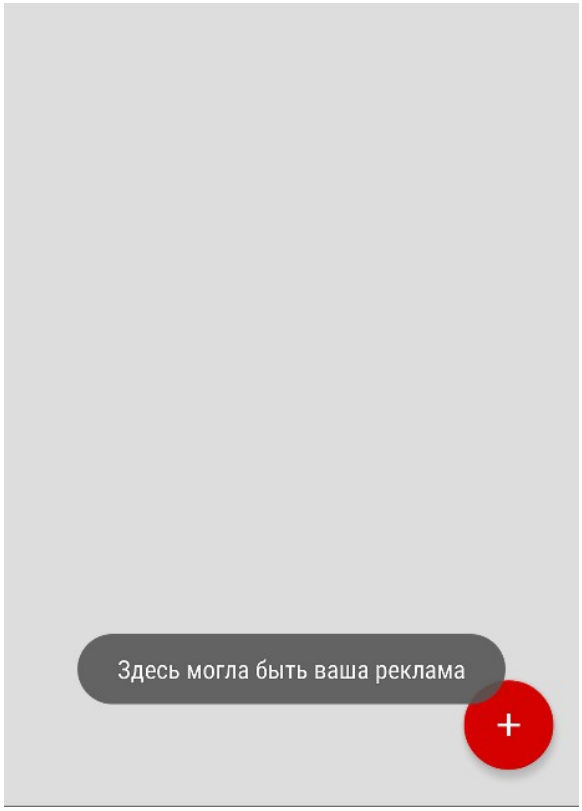
Snakebar

Snakebar представляет собой «модернизированный» Toast – уведомление, появляющееся обычно в нижней части экрана и дающее возможность пользователю произвести какое-то действие в ответ на сообщение. Использование Snakebar очень похоже на использование Toast (ниже приведен пример обработчика нажатия на FAB, описанную выше):

```
public void onButtonClick(View v) {
    View coordinator = findViewById(R.id.coordinator);
    Snackbar.make(coordinator, "Hello from Snake Bar!", Snackbar.LENGTH_LONG)
            .setAction("Some snake action", new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    Toast.makeText(MainActivity.this,
                        "Здесь могла быть ваша реклама",
                        Toast.LENGTH_SHORT).show();
                }
            }).setActionTextColor(0xFFFFB300).show();
}
```

Благодаря наличию корневого элемента разметки – *CooordinatorLayout* – при появлении на экране Snakebar он показывается не «поверх» кнопки, а под ней – кнопка упалывает вверх. Всю эту магию и обеспечивает CoordinatorLayout:





Стоит упомянуть полезный бонус, отсутствующий в *Toast* – *Snakebar* можно «смахнуть» с экрана, если он не нужен или мешает.

RecyclerView

RecyclerView – еще один элемент UI, призванный придти на замену старому доброму *ListView*. *RecyclerView* более гибок и обеспечивает более эффективное управление отображением элементнов списка и используемыми ресурсами. Его применение особенно полезно в случаях, когда отображаемые списки могут меняться на лету из-за действий пользователя или других изменений в данных. Для использования *RecyclerView* в самом простом случае требуется совсем немного – сделать адартер и прицепить его к *RecyclerView*:

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

Код *MainActivity*

```
public class MainActivity extends AppCompatActivity {
    static final String NOTE_PREFIX = "Note #";
    static final int NOTES_QTY = 30;
    String[] mNotes = new String[NOTES_QTY];

    private RecyclerView mRecyclerView;
    private RecyclerView.Adapter mAdapter;
    private RecyclerView.LayoutManager mLayoutManager;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        for(int i = 0; i < NOTES_QTY; i++) {
            mNotes[i] = NOTE_PREFIX + i+1;
        }
        mRecyclerView = (RecyclerView) findViewById(R.id.recycler_view);
        mLayoutManager = new LinearLayoutManager(this);
        mRecyclerView.setLayoutManager(mLayoutManager);
        mAdapter = new NotesAdapter(mNotes);
        mRecyclerView.setAdapter(mAdapter);
    }
}
```

Код класса адаптера:

```
public class NotesAdapter extends RecyclerView.Adapter<NotesAdapter.ViewHolder>
{
    private String[] mDataset;

    public static class ViewHolder extends RecyclerView.ViewHolder {
        public TextView mTextView;
        public ViewHolder(TextView v) {
            super(v);
            mTextView = v;
        }
    }

    public NotesAdapter(String[] myDataset) {
        mDataset = myDataset;
    }

    @Override
    public NotesAdapter.ViewHolder onCreateViewHolder(ViewGroup parent,
                                                      int viewType) {

        TextView v = (TextView) LayoutInflater.from(parent.getContext())
                .inflate(R.layout.note_view, parent, false);
        return new ViewHolder(v);
    }

    @Override
    public void onBindViewHolder(ViewHolder holder, int position) {
         holder.mTextView.setText(mDataset[position]);
    }

    @Override
    public int getItemCount() {
        return mDataset.length;
    }
}
```