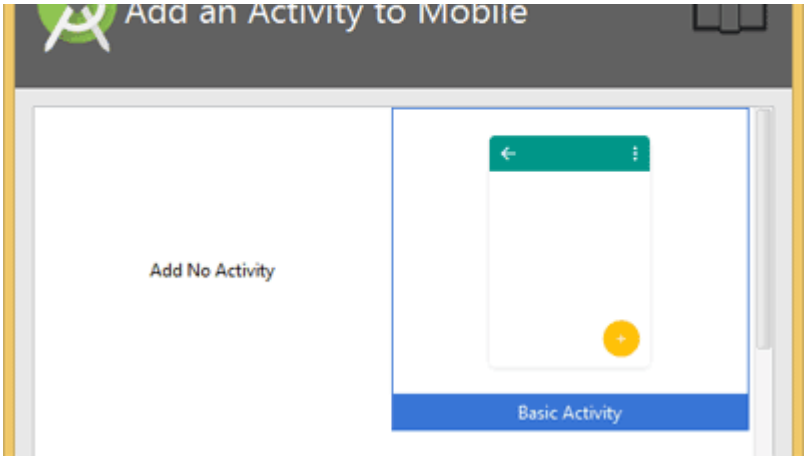
Библиотека Android Support Design

Метки: CoordinatorLayout, FloatingActionButton, Snackbar, **Material Design**, setAction()

Библиотека **Android Support Design** была представлена одновременно с новой версией Android Marshmallow, но её можно использовать на устройствах, начиная с версии Android 2.1. Библиотека была специально спроектирована под новый дизайн Material Design и позволяет использовать готовые компоненты в новом стиле.

Используем шаблон "Basic Activity"

Создадим проект при помощи шаблона **Basic Activity**.



Если подключать вручную, то следует прописать зависимость.

```
implementation 'com.android.support:design:27.1.1'
```

При использовании нового шаблона будут задействованы четыре компонента библиотеки: **CoordinatorLayout**, **AppBarLayout**, **FloatingActionButton**, **Snackbar**.

Первые три компонента доступны через XML-разметку, а **Snackbar** вызывается программно.

Изучим разметку активности **activity_main.xml**.

```
<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"
    tools:context=".MainActivity">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/AppTheme.AppBarOverlay">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            app:popupTheme="@style/AppTheme.PopupOverlay"/>

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

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

    <android.support.design.widget.FloatingActionButton
        android:id="@+id/fab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|end"
        android:layout_margin="@dimen/fab_margin"
        android:src="@android:drawable/ic_dialog_email"/>

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

Корневым элементом разметки служит компонент **CoordinatorLayout**, который является наследником стандартного FrameLayout. Но если обычный **FrameLayout** позволяет просто накладывать один компонент поверх другого, то **CoordinatorLayout** позволяет координировать определённые зависимости между дочерними компонентами. Сейчас мы не будем вдаваться в тонкости, этой теме можно посвятить отдельную статью.

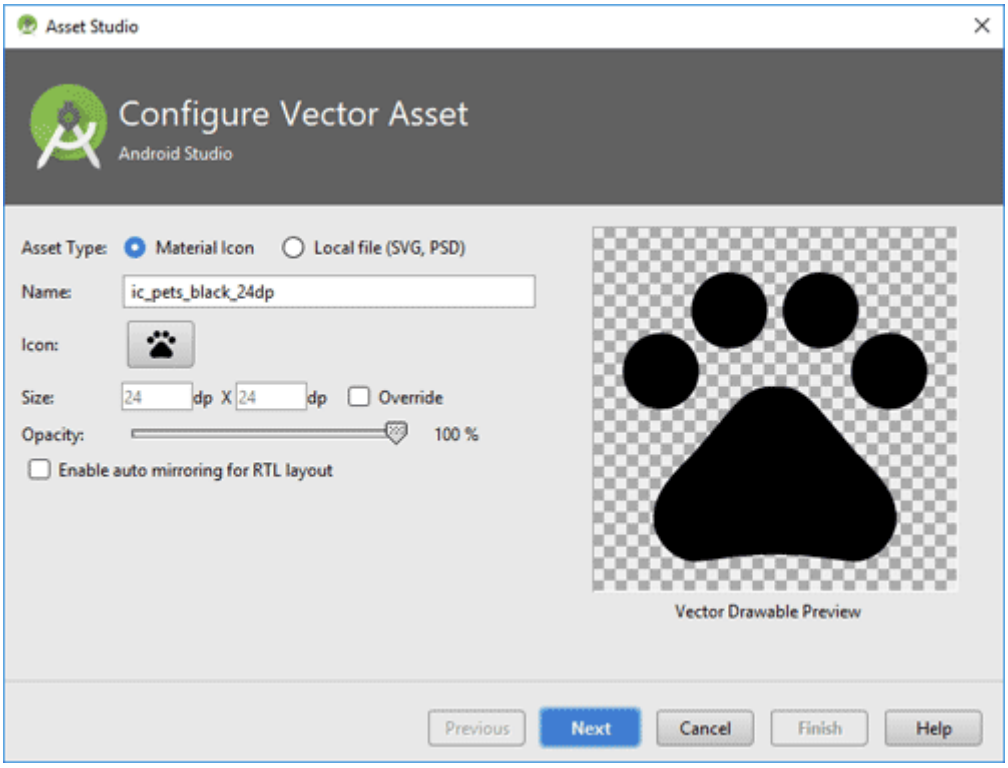
Далее идёт **AppBarLayout** с вложенным **Toolbar**. Связка компонентов образуют внешний вид и поведение продвинутого заголовка экрана активности, который пришёл на смену **ActionBar** из Android 4.x, который в свою очередь заменил стандартный заголовок (Title) в Android 2.x. Опять оставляем их пока без внимания.

указанием вставляемои разметки - **layout/content_main.xml**.

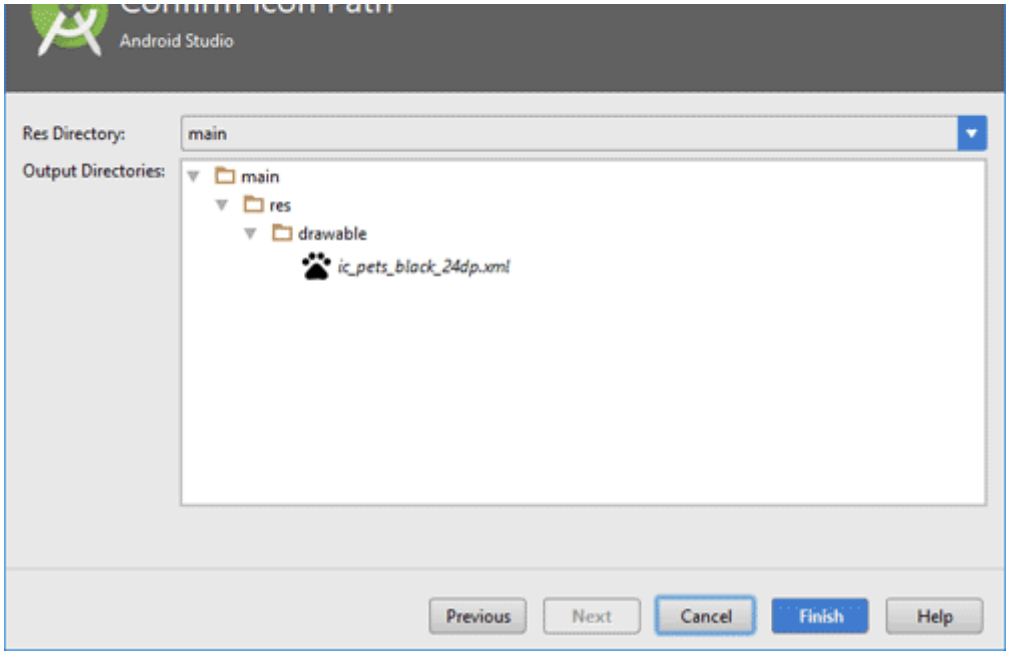
Завершает основную разметку красивая кнопка **FloatingActionButton**, которая на самом деле является продвинутым вариантом **ImageView**. Но в спецификации **Material Design** кнопке отводится большая роль и практически является визитной карточкой нового дизайна. Наверняка, вы уже видели её на различных картинках.

Подробнее о возможностях кнопки также в других материалах. Обратите внимание, что кнопка "пришпилена" к нижнему правому углу экрана при помощи **layout_gravity**. В качестве значка используется изображение из системных ресурсов **@android:drawable/ic_dialog_email** в свойстве **srcCompat**. Вы можете установить собственное изображение, подходящее по контексту. На данный момент Гугл рекомендует отказываться от растровых изображений и активно использовать векторные изображения.

Заменим значок электронной почты на изображение лапы кота. Щёлкаем правой кнопкой мыши по папке **app** и вызываем контекстное меню **New | Vector Asset**. В диалоговом окне щёлкаем по значку **Icon**, чтобы открыть другое окно для выбора значка. В строке поиска набираем **pets** и находим нужный значок. Выделяем его и нажимаем **OK**.



Далее нажимаем кнопку **Next** и в следующем окне запоминаем название файла в ресурсах.



Теперь вы можете заменить значок для кнопки.

Переходим к программной части.

Закусывать надо! Базовый пример с Snackbar

Если раньше для всплывающих сообщений использовались хлебные тосты **Toast**, то теперь можно использовать новый класс **Snackbar**, который переводится как "Закусочная".

Новый класс **Snackbar** имеет много общего с классом **Toast** и имеет практически тот же синтаксис.

Пример запуска сообщения находится в строчках.

```
    @Override
    public void onClick(View view) {
        Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
                .setAction("Action", null).show();
    }
});
```

Как видите, код очень похож. Но есть и различия. Если **Toast** является частью активности и выводится поверх неё в нижней части по умолчанию, если не заданы другие параметры, то **Snackbar** выводится в "подвале" родительского элемента. В первом параметре указывается подходящий компонент, по которому система попытается найти родителя, обычно им является **CoordinatorLayout**. В некоторых примерах я видел код, когда родитель указывается явно.

Код можно немного упростить, если убрать вызов метода **setAction()**, который сейчас не используется.

```
Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG).show();
```

Сообщение появляется на несколько секунд с указанным текстом и исчезает через несколько секунд. Для константы **LENGTH_SHORT** это будет 1.5 секунд, для **LENGTH_LONG** - 2.7 секунд. Практически аналогичное поведение у старого **Toast**.

Но есть и небольшие отличия. Например, когда выводится сообщение, его можно смахнуть с экрана слева направо, не дожидаясь, когда оно само исчезнет.

Также можно задать свою продолжительность. В первых версиях это не работало, я даже задавал этот вопрос на StackOverFlow, где проблему подтвердили и обещали исправить. Недавно проверял - действительно, теперь работает.

```
Snackbar snackbar = Snackbar.make(view, "Replace with your own action",
Snackbar.LENGTH_LONG)
        .setAction("Action", null);
snackbar.setDuration(8000); // 8 секунд
snackbar.show();
```

В версии 22.2.1 появилась новая константа **Snackbar.LENGTH_INDEFINITE** (неопределённая длительность). В этом случае компонент не исчезает самостоятельно.

Метод dismiss()

закрытия сообщения и напишем код.

```
private Snackbar mSnackbar;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    final Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
    fab.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            mSnackbar = Snackbar.make(view, "Пора кормить кота!",
Snackbar.LENGTH_INDEFINITE)
                    .setAction("Action", null);
            mSnackbar.show();
        }
    });

    Button dismissButton = (Button) findViewById(R.id.buttonDismiss);
    dismissButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            mSnackbar.dismiss();
        }
    });
}
```

Чтобы **Snackbar** можно было убирать движением пальца, компонент следует поместить в **CoordinatorLayout**. В стандартных разметках **LinearLayout** и ему подобных операция не сработает.

Snackbar.События

Отслеживать события появления и исчезновения **Snackbar** можно с помощью методов обратного вызова через **addCallback()**.

Параметр **event** у метода **onDismissed()** позволяет узнать конкретное событие, повлекшее исчезновение, при помощи констант **DISMISS_EVENT_SWIPE, DISMISS_EVENT_ACTION, DISMISS_EVENT_TIMEOUT, DISMISS_EVENT_MANUAL, DISMISS_EVENT_CONSECUTIVE**.

```
        .setAction("Action", null);

mSnackbar.show();

mSnackbar.addCallback(new Snackbar.Callback() {

    @Override
    public void onDismissed(Snackbar snackbar, int event) {
        if (event == Snackbar.Callback.DISMISS_EVENT_TIMEOUT) {
            Log.i("SnackBar", "Закрыт по истечении таймаута");
        }
        if(event == Snackbar.Callback.DISMISS_EVENT_SWIPE){
            Log.i("SnackBar", "Swipe");
        }
    }

    @Override
    public void onShown(Snackbar snackbar) {
        Log.i("SnackBar", "onShown");
    }
});
```

Меняем цвет текста и фона

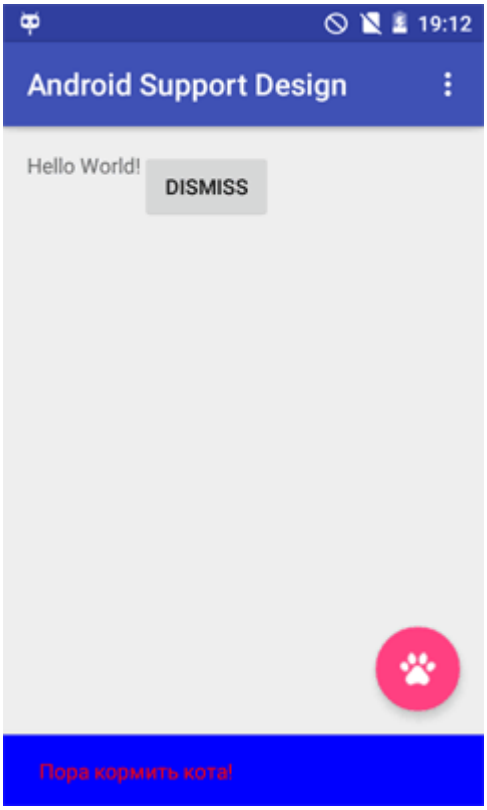
Класс **Snackbar** является закрытым ящиком и мы не знаем, как он устроен. Но у него есть метод **getView()**, который возвращает некий компонент **View**. Получив его, мы можем поменять у него фоновый цвет.

```
fab.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        mSnackbar = Snackbar.make(view, "Пора кормить кота!",
Snackbar.LENGTH_INDEFINITE)
                .setAction("Action", null);

        View snackbarView = mSnackbar.getView();
        snackbarView.setBackgroundColor(Color.BLUE);
        mSnackbar.show();
    }
});
```

Если вам захочется поменять также и цвет текста в сообщении, то нужно получить доступ к **TextView**:

```
View snackbarView = mSnackbar.getView();
snackbarView.setBackgroundColor(Color.BLUE);
TextView snackTextView = (TextView)
snackbarView.findViewById(android.support.design.R.id.snackbar_text);
snackTextView.setTextColor(Color.RED);
snackbar.show();
```



На практике, вы вряд ли будете часто менять эти цвета, но вдруг пригодится.

Добавляем кнопку действия

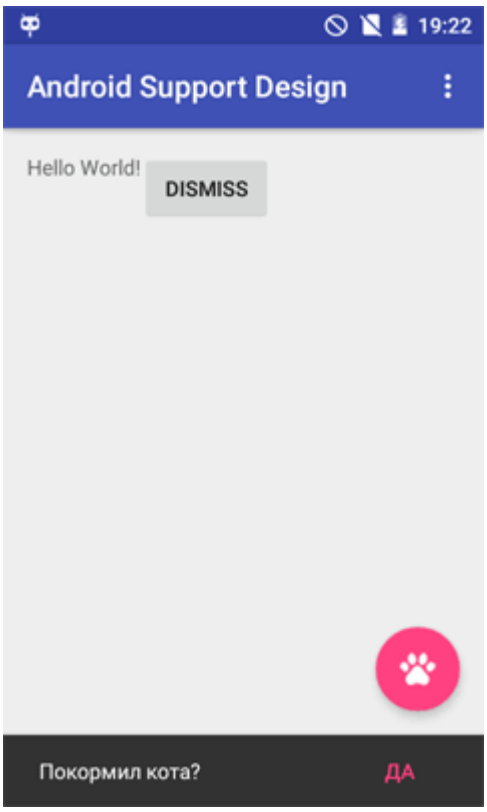
Ради вывода сообщения в подвале экрана не стоило создавать новый класс, **Toast** вполне справлялся с этой задачей. Теперь рассмотрим существенное различие. На панели **Snackbar** можно разместить кнопку действия с помощью метода **setAction()**. В методе нужно указать текст для кнопки и обработчик для щелчка.

```
    super.onCreate(savedInstanceState);

    ...

    FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
    fab.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            mSnackbar = Snackbar.make(view, "Покормил кота?",
Snackbar.LENGTH_INDEFINITE)
                    .setAction("Да", snackbarOnClickListener);
            mSnackbar.show();
        }
    });
}

View.OnClickListener snackbarOnClickListener = new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Toast.makeText(getApplicationContext(), "Молодец!", Toast.LENGTH_LONG).show();
    }
};
```



В моём случае при нажатии на кнопку действия панель исчезала. Если этого не происходит, всегда можно добавить дополнительную строчку с вызовом метода **dismiss()**.