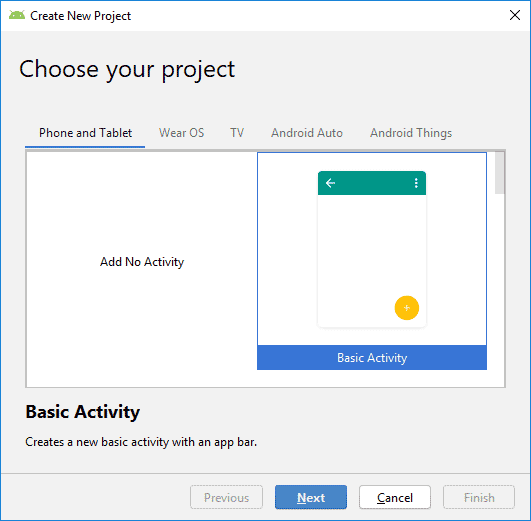
Шаблон "Basic Activity"

Библиотека **Android Support Design** (старое название) 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. Мы можем использовать элементы библиотеки вручную, но для быстрого знакомства с некоторыми из них удобнее воспользоваться готовым шаблоном Android Studio.

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



На сегодняшний день пора использовать AndroidX, поэтому в следующем окне мастера установите флажок у опции **Use androidx.\* artifacts**.

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**.

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

<androidx.coordinatorlayout.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"

tools:context=".MainActivity">

<com.google.android.material.appbar.AppBarLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

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

<androidx.appcompat.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" />

</com.google.android.material.appbar.AppBarLayout>

<include layout="@layout/content\_main" />

<com.google.android.material.floatingactionbutton.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"

app:srcCompat="@android:drawable/ic\_dialog\_email" />

</androidx.coordinatorlayout.widget.CoordinatorLayout>

Корневым элементом разметки служит компонент **CoordinatorLayout**, который является наследником стандартного [FrameLayout](http://developer.alexanderklimov.ru/android/layout/framelayout.php). 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.

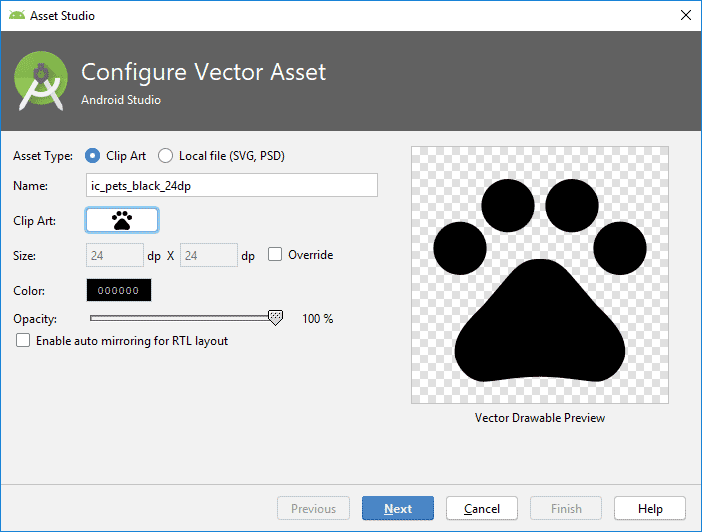
Далее идёт **AppBarLayout** с вложенным **Toolbar**. Связка компонентов образуют внешний вид и поведение продвинутого заголовка экрана активности, который пришёл на смену **ActionBar** из Android 4.x, который в свою очередь заменил стандартный заголовок (Title) в Android 2.x. Опять оставляем их пока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

Потом следует интересный приём, который вам может впоследствии пригодиться. В существующую разметку вставляется ещё одна разметка при помощи тега **include** и с указанием вставляемой разметки - **layout/content\_main.xml**.

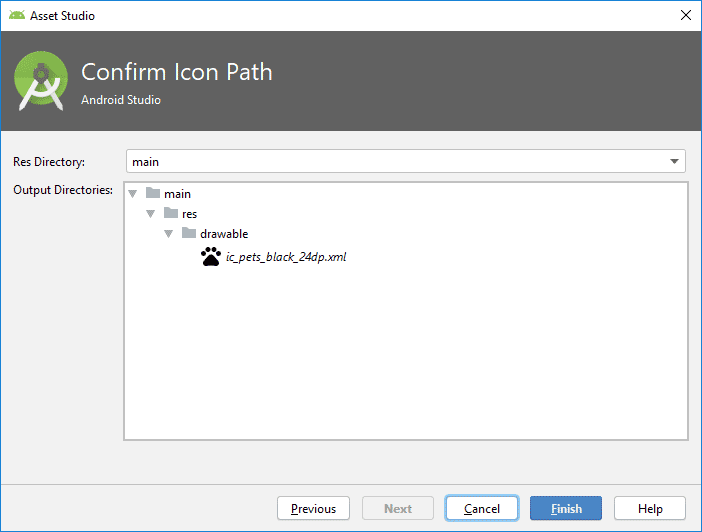
Завершает основную разметку красивая кнопка **FloatingActionButton**, которая на самом деле является продвинутым вариантом **ImageView**. Но в спецификации **Material Design** кнопке отводится большая роль и практически является визитной карточкой нового дизайна. Наверняка, вы уже видели её на различных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

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.

FloatingActionButton fab = findViewById(R.id.fab);

fab.setOnClickListener(new View.OnClickListener() {

@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()

Метод **dismiss()** закрывает сообщение. Может пригодиться при использовании **LENGTH\_INDEFINITE**, когда сообщение будет висеть на экране вечно. Добавим кнопку для закрытия сообщения и напишем код.

private Snackbar mSnackbar;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

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

setSupportActionBar(toolbar);

FloatingActionButton fab = 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 = findViewById(R.id.button\_dismiss);

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**.

mSnackbar = Snackbar.make(view, "Пора кормить кота!",

Snackbar.LENGTH\_LONG)

.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**:

mSnackbar = Snackbar.make(view, "Пора кормить кота!", Snackbar.LENGTH\_INDEFINITE)

.setAction("Action", null);

View snackbarView = mSnackbar.getView();

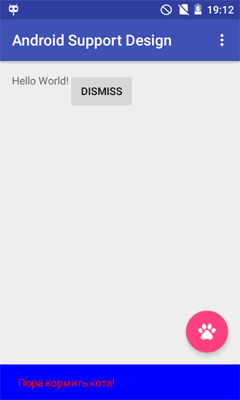
snackbarView.setBackgroundColor(Color.BLUE);

//**TextView snackTextView = snackbarView.findViewById(android.support.design.R.id.snackbar\_text); // старый вариант**

**TextView snackTextView = snackbarView.findViewById(com.google.android.material.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.

@Override

protected void onCreate(Bundle savedInstanceState) {

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**

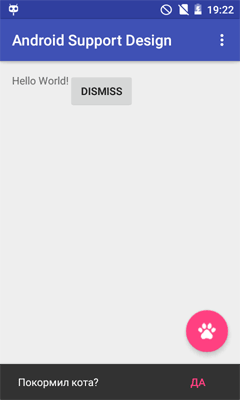
**public void onClick(View view) {**

**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()**.

С помощью метода **setActionTextColor()** можно поменять цвет текста у кнопки действия.

Snackbar

.make(parent, "Покормил кота?", Snackbar.LENGTH\_LONG)

.setAction("Да", snackOnClickListener)

**.setActionTextColor(Color.MAGENTA)** // цвет текста у кнопки действия

.show();

Если цвет не задавать явно, то будет использован стиль **colorAccent**. Для изменения цвета обычного текста в **Snackbar** можно установить стиль **android:textColor** (он подействует на все компоненты на экране).

<style name="SnackbarTheme" parent="Theme.AppCompat.Light">

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

<item name="android:textColor">@color/blue</item>

</style>

...

<activity

...

android:theme="@style/AppTheme">

</activity>

Приподнимаем компонент

Вы замечали, что FAB приподнимается при появлении **SnackBar**. Такую же возможность можно добавить для компонента, который находится в нижней части экрана. Чтобы **SnackBar** не загораживал обзор, нужно для компонента добавить новый атрибут **app:layout\_dodgeInsetEdges="bottom"**. Это сработает, если компонент является дочерним элементом разметки **CoordinatorLayout**. Для других разметок и для других степеней вложенности приём не сработает.

Настраиваем SnackBar

Стандартный компонент имеет ширину экрана в телефонах. Но мы можем изменить его поведение. Создадим уменьшенный прямоугольник с закруглёнными углами. Для начала создадим ресурс в в папке **res/layout** под именем **bg\_snackbar.xml**, который послужит подложкой.

<shape xmlns:android="http://schemas.android.com/apk/res/android"

android:shape="rectangle">

<solid android:color="#323232" />

<corners android:radius="4dp" />

</shape>

Добавим новый класс **SnackbarHelper**.

package ru.alexanderklimov.basicactivity;

import android.content.Context;

import android.support.design.widget.Snackbar;

import android.support.v4.view.ViewCompat;

import android.view.ViewGroup;

class SnackbarHelper {

static void configSnackbar(Context context, Snackbar snackbar) {

addMargins(snackbar);

setRoundBordersBg(context, snackbar);

ViewCompat.setElevation(snackbar.getView(), 6f);

}

private static void addMargins(Snackbar snack) {

ViewGroup.MarginLayoutParams params = (ViewGroup.MarginLayoutParams) snack.getView().getLayoutParams();

params.setMargins(12, 12, 12, 12);

snack.getView().setLayoutParams(params);

}

private static void setRoundBordersBg(Context context, Snackbar snackbar) {

// API 21

snackbar.getView().setBackground(context.getDrawable(R.drawable.bg\_snackbar));

}

}

Заменим вызов **SnackBar**

Snackbar snackbar = Snackbar.make(view,

"Пора кормить кота!", Toast.LENGTH\_SHORT);

SnackbarHelper.configSnackbar(MainActivity.this, snackbar);

//snackbar.config(this) if you're using Kotlin

snackbar.show();

В комментариях есть пример вызова метода-расширения на Kotlin вместо класса **SnackbarHelper**.

fun Snackbar.config(context: Context){

val params = this.view.layoutParams as ViewGroup.MarginLayoutParams

params.setMargins(12, 12, 12, 12)

this.view.layoutParams = params

this.view.background = context.getDrawable(R.drawable.bg\_snackbar)

ViewCompat.setElevation(this.view, 6f)

}

Мы рассмотрели часть возможностей библиотеки **Android Support Design**, которые используются в новом шаблоне **Blank Activity**. Остальные части библиотеки будут рассмотрены в других статьях.