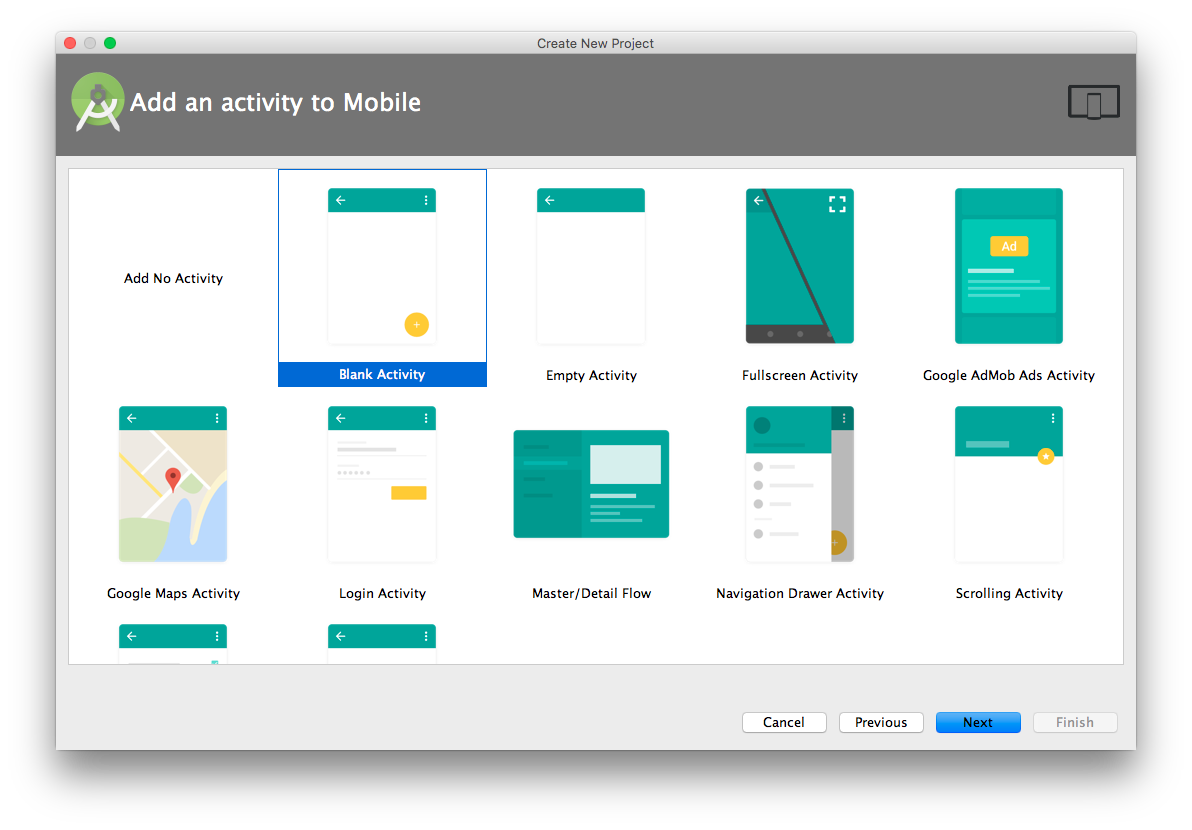
Анимация Floating Action Button в Android

С момента возникновения концепции Material design одним из самых простых в реализации элементов стала плавающая кнопка действия — FAB, Floating Action Button. Этот элемент быстро обрёл широчайшую популярность среди разработчиков и дизайнеров. В этой публикации мы рассмотрим, как можно анимировать FAB и сделать её интерактивной. Но сначала разберём, как вообще добавить этот элемент в ваш проект.   
  
FAB выглядит как цветной круг в правом нижнем углу экрана. Если в Android Studio создать новый проект Blank Activity, то в нём автоматически будет сгенерирована плавающая кнопка действия.  
  


<android.support.design.widget.FloatingActionButton

android:id="@+id/fab"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

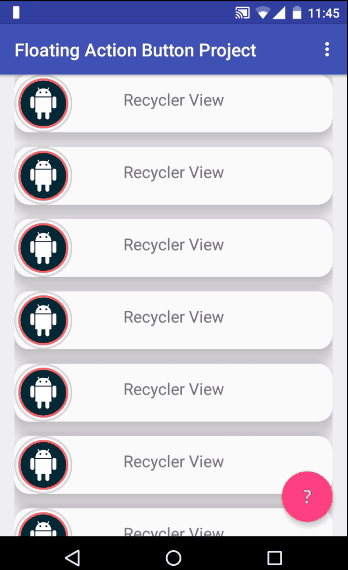
android:layout\_gravity="bottom|end"

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

android:src="@android:drawable/ic\_menu\_help"

/>

Floating Action Button

FAB может быть одного из двух размеров: 56 dp (по умолчанию) или 40 dp. Если вы хотите подробнее изучить принципы использования FAB в дизайне приложения, то обратите внимание на [официальные гайдлайны Google](https://www.google.com/design/spec/components/buttons-floating-action-button.html).  
  
В самых свежих Android-приложениях FAB реагирует на прокручивание списка элементов. Было бы логичнее скрывать её во время прокручивания. Вот что имеется в виду:  
  
  
  
Для отображения этой анимации создадим recyclerView, благодаря которому FAB реагирует на прокручивание. Сегодня доступно немало библиотек, позволяющих добиться этого с помощью пары строк кода. Например:

**public** **class** **FAB\_Hide\_on\_Scroll** **extends** **FloatingActionButton**.**Behavior** {

**public** **FAB\_Hide\_on\_Scroll**(Context context, AttributeSet attrs) {

**super**();

}

**@Override**

**public** **void** **onNestedScroll**(CoordinatorLayout coordinatorLayout, FloatingActionButton child, View target, **int** dxConsumed, **int** dyConsumed, **int** dxUnconsumed, **int** dyUnconsumed) {

**super**.onNestedScroll(coordinatorLayout, child, target, dxConsumed, dyConsumed, dxUnconsumed, dyUnconsumed);

*//child -> Floating Action Button*

**if** (child.getVisibility() == View.VISIBLE && dyConsumed > 0) {

child.hide();

} **else** **if** (child.getVisibility() == View.GONE && dyConsumed < 0) {

child.show();

}

}

**@Override**

**public** **boolean** **onStartNestedScroll**(CoordinatorLayout coordinatorLayout, FloatingActionButton child, View directTargetChild, View target, **int** nestedScrollAxes) {

**return** nestedScrollAxes == ViewCompat.SCROLL\_AXIS\_VERTICAL;

}

}

Здесь использован класс FloatingActionButton.Behavior(), чья основная задача, согласно официальной документации, заключается в перемещении видов FloatingActionButton, чтобы ни один из Snackbar их не перекрывал. Но в нашем случае этот класс является расширенным, так что мы можем его использовать для реализации нужного поведения кнопки.  
  
Что же делает данный класс? При каждой инициализации прокрутки вниз метод onStartNestedScroll()возвращает значение true. После этого метод onNestedScroll() отображает или прячет кнопку, в зависимости от её текущей видимости. Конструктор класса FloatingActionButton.Behavior() является важной частью описанного поведения вида (view) и извлекается из XML-файла.

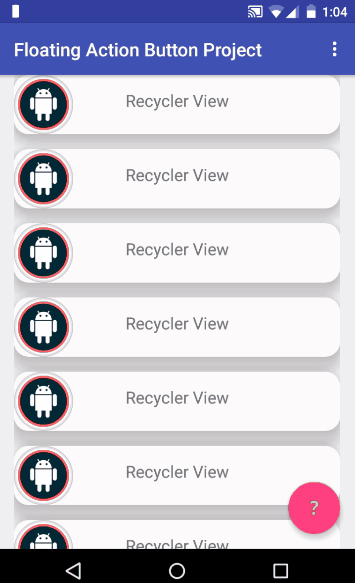
**public** **FAB\_Hide\_on\_Scroll**(Context context, AttributeSet attrs) {

**super**();

}

Добавим в FAB атрибут layout\_behavior, содержащий название пакета, а в конце — имя класса. Иначе говоря, в атрибуте должно быть указано точное размещение класса в проекте. Например:

app:layout\_behavior="com.valdio.valdioveliu.floatingactionbuttonproject.Scrolling\_Floating\_Action\_Button.FAB\_Hide\_on\_Scroll"

Анимация выглядит хорошо, но можно сделать ещё лучше. Например, чтобы кнопка уходила за пределы экрана во время прокрутки — это более реалистичное поведение:  
  
  
  
Здесь используется та же логика, что и в предыдущем варианте, за исключением способа исчезновения FAB. Анимация довольно проста. Кнопка уходит вниз с помощью [LinearInterpolator](http://developer.android.com/reference/android/view/animation/LinearInterpolator.html). Расстояние, которое ей нужно пройти, равно высоте кнопки плюс ширина нижнего поля.  
  
Обратите внимание, что в выражениях if отсутствуют проверки View.VISIBLE и View.GONE, поскольку в данном случае вид не скрывается, а лишь уплывает за пределы экрана.

**public** **class** **FAB\_Float\_on\_Scroll** **extends** **FloatingActionButton**.**Behavior** {

**public** **FAB\_Float\_on\_Scroll**(Context context, AttributeSet attrs) {

**super**();

}

**@Override**

**public** **void** **onNestedScroll**(CoordinatorLayout coordinatorLayout, FloatingActionButton child, View target, **int** dxConsumed, **int** dyConsumed, **int** dxUnconsumed, **int** dyUnconsumed) {

**super**.onNestedScroll(coordinatorLayout, child, target, dxConsumed, dyConsumed, dxUnconsumed, dyUnconsumed);

*//child -> Floating Action Button*

**if** (dyConsumed > 0) {

CoordinatorLayout.LayoutParams layoutParams = (CoordinatorLayout.LayoutParams) child.getLayoutParams();

**int** fab\_bottomMargin = layoutParams.bottomMargin;

child.animate().translationY(child.getHeight() + fab\_bottomMargin).setInterpolator(**new** LinearInterpolator()).start();

} **else** **if** (dyConsumed < 0) {

child.animate().translationY(0).setInterpolator(**new** LinearInterpolator()).start();

}

}

**@Override**

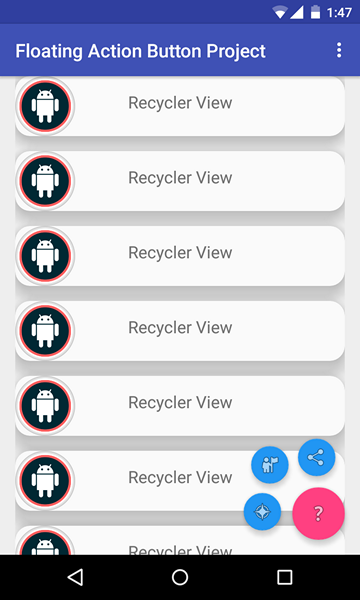
**public** **boolean** **onStartNestedScroll**(CoordinatorLayout coordinatorLayout, FloatingActionButton child, View directTargetChild, View target, **int** nestedScrollAxes) {

**return** nestedScrollAxes == ViewCompat.SCROLL\_AXIS\_VERTICAL;

}

}

Меню из FAB'ов

Существует немало приложений, авторы которых создали красивые и хорошо работающие меню, состоящие из плавающих кнопок действия.  
  
  
  
Давайте сделаем нечто подобное. Для начала создадим макет, содержащий три маленькие кнопки. Они невидимы и расположены в самом низу макета, под главной FAB. Содержимое *fab\_layout.xml*:

<FrameLayout 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">

<android.support.design.widget.FloatingActionButton

android:id="@+id/fab\_1"

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\_menu\_compass"

android:visibility="invisible"

app:backgroundTint="@color/colorFAB"

app:fabSize="mini" />

<android.support.design.widget.FloatingActionButton

android:id="@+id/fab\_2"

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\_menu\_myplaces"

android:visibility="invisible"

app:backgroundTint="@color/colorFAB"

app:fabSize="mini" />

<android.support.design.widget.FloatingActionButton

android:id="@+id/fab\_3"

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\_menu\_share"

android:visibility="invisible"

app:backgroundTint="@color/colorFAB"

app:fabSize="mini" />

</FrameLayout>

Этот макет нужно включить в макет activity под главной FAB.

<include layout="@layout/fab\_layout" />

Теперь нужно добавить анимацию исчезновения и появления каждой из малых кнопок.  
  
**Примечание**: здесь вы можете столкнуться с проблемой, связанной с отработкой нажатия на малые кнопки. Когда анимация завершается, реальное положение кнопки не меняется, перемещается только вид. Поэтому вы не сможете правильно обработать касание кнопки. Для решения этой проблемы можно настроить параметры макетов каждой кнопки с учётом их нового положения, и только потом выполнять анимацию перемещения вида.  
  
Саму анимацию вы можете посмотреть в конце этой публикации. Порядок действий для всех кнопок один и тот же, различаются лишь координаты перемещения.  
  
Отображение меню:

FrameLayout.LayoutParams layoutParams = (FrameLayout.LayoutParams) fab1.getLayoutParams();

layoutParams.rightMargin += (**int**) (fab1.getWidth() \* 1.7);

layoutParams.bottomMargin += (**int**) (fab1.getHeight() \* 0.25);

fab1.setLayoutParams(layoutParams);

fab1.startAnimation(show\_fab\_1);

fab1.setClickable(**true**);

fab1 перемещается с помощью добавления в layoutParams полей справа и снизу, после чего инициируется анимация.  
  
Скрывание меню:  
  
FrameLayout.LayoutParams layoutParams = (FrameLayout.LayoutParams) fab1.getLayoutParams();  
layoutParams.rightMargin -= (int) (fab1.getWidth() \* 1.7);  
layoutParams.bottomMargin -= (int) (fab1.getHeight() \* 0.25);  
fab1.setLayoutParams(layoutParams);  
fab1.startAnimation(hide\_fab\_1);  
fab1.setClickable(false);  
  
Процесс скрывания представляет собой обратное воспроизведение предыдущей анимации.   
  
//Анимации одной из малых кнопок  
Animation show\_fab\_1 = AnimationUtils.loadAnimation(getApplication(), R.anim.fab1\_show);  
Animation hide\_fab\_1 = AnimationUtils.loadAnimation(getApplication(), R.anim.fab1\_hide);  
  
Теперь создадим в папке *res/anim/* файлы для каждой из анимаций. Делается это просто, но если у вас возникнут затруднения, то можете обратиться к [документации](http://developer.android.com/guide/topics/resources/animation-resource.html).   
  
Содержимое *fab1\_show.xml*:

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

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

android:fillAfter="true">

<!-- Rotate -->

<rotate

android:duration="500"

android:fromDegrees="30"

android:interpolator="@android:anim/linear\_interpolator"

android:pivotX="50%"

android:pivotY="50%"

android:repeatCount="4"

android:repeatMode="reverse"

android:toDegrees="0"></rotate>

<!--Move-->

<translate

android:duration="1000"

android:fromXDelta="170%"

android:fromYDelta="25%"

android:interpolator="@android:anim/linear\_interpolator"

android:toXDelta="0%"

android:toYDelta="0%"></translate>

<!--Fade In-->

<alpha

android:duration="2000"

android:fromAlpha="0.0"

android:interpolator="@android:anim/decelerate\_interpolator"

android:toAlpha="1.0"></alpha>

</set>

Содержимое *fab1\_hide.xml*:

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

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

android:fillAfter="true">

<!--Move-->

<translate

android:duration="1000"

android:fromXDelta="-170%"

android:fromYDelta="-25%"

android:interpolator="@android:anim/linear\_interpolator"

android:toXDelta="0%"

android:toYDelta="0%"></translate>

<!--Fade Out-->

<alpha

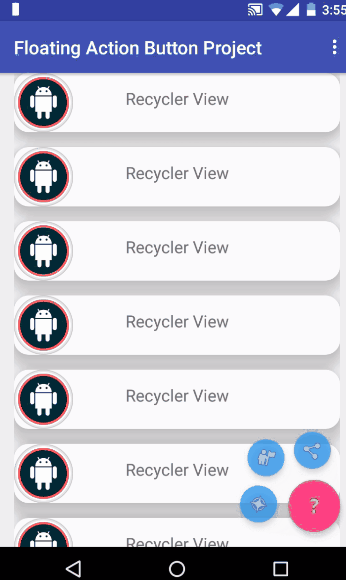
android:duration="2000"

android:fromAlpha="1.0"

android:interpolator="@android:anim/accelerate\_interpolator"

android:toAlpha="0.0"></alpha>

</set>

Если вы посмотрите на тэг перевода (translate tag), отвечающий за движение вида, то увидите, что коэффициент перемещения (170% и 25%) соответствует коэффициентам, использованным при добавлении полей и извлечённым в Java-код.  
  
Все вышеописанные шаги мы повторяем и для остальных малых кнопок. Различаются только коэффициенты перемещения: fab2 — 150% и 150%, fab3 — 25% и 170%.  
  
Результат наших усилий:  
  


Это не конец

Надеюсь, эта статья помогла вам понять, как можно анимировать Floating Action Button в вашем проекте. Дальше вы можете изучить [ресурсы, посвящённые анимации в Android](http://developer.android.com/guide/topics/resources/animation-resource.html), и начать создавать собственные виды анимаций. Весь код описанного в этой публикации проекта доступен на [GitHub](https://github.com/sitepoint-editors/FloatingActionButton_Animation_Project).