Fragment (Фрагменты)



Фрагменты немного пугают новичков. Постараюсь объяснить как можно проще, чтобы отдельные фрагменты пазла сложились в единую картину.

Зачем?

Создатели операционной системы Android оказались недальновидными разработчиками. Не посоветовавшись с котами, они разработали систему под маленькие экраны телефонов. Но котам было неудобно пользоваться такими экранами, и тогда придумали планшеты.



Старые программы прекрасно на них запускались, но обнаружилось несколько недостатков. На больших экранах интерфейс выглядел не слишком элегантно, появились большие пустые пространства. И тогда возникла идея объединить два отдельных экрана из смартфона в один экран на планшете. Это самый классический пример применения фрагмента. По сути, это костыль. Возможно, если бы сразу подумали головой, то придумали бы более элегантное решение. Но теперь поздно пить Боржоми, будем использовать предложенную концепцию.

Фрагменты были представлены в API 11 (Android 3.0), но в целях совместимости была написана специальная библиотека **Android Support library** для старых устройств. Таким образом сейчас существуют два класса **Fragment**: для новых устройств и для старых устройств. Названия методов и классов очень похожи, но не стоит смешивать в одном проекте два разных класса.

При желании можно было продолжить писать приложения в старом стиле, отслеживая размеры экрана. Но такой код получится слишком сложным. Пришлось бы писать один код для переключения от одной активности к другой при использовании смартфона и другой код, когда взаимодействие между объектами происходит на одном экране в планшете. Чтобы устранить это противоречие, были придуманы фрагменты. Хотя там тоже придётся писать много кода.

Несколько слов о том, как проще воспринимать фрагмент. Считайте, что фрагмент - это тот же компонент как **Button**, **TextView** или **LinearLayout** с дополнительными возможностями. Фрагмент, как и кнопку, нужно поместить на экран активности. Но фрагмент является модульным компонентом и один и тот же фрагмент можно встроить в две разные активности. С кнопкой такой номер не пройдёт. Для каждой активности вы должны создать свою отдельную кнопку, даже если их нельзя будет отличить друг от друга.

Но фрагменты - это не замена активности, они не существуют сами по себе, а только в составе активностей. Поэтому в манифесте прописывать их не нужно. Но в отличие от стандартной кнопки, для каждого фрагмента вам придётся создавать отдельный класс, как для активности.

В составе активности есть специальный менеджер фрагментов, который может контролировать все классы фрагментов и управлять ими. О нём позже.

Фрагменты являются строительным материалом для приложения. Вы можете в нужное время добавить новый фрагмент, удалить ненужный фрагмент или заменить один фрагмент на другой. Точно так же мы собираем пазл - подносим фрагмент кота в общую картину, иногда ошибаемся и тогда заменяем кусочек пазла на другой и т.д.

Фрагмент может иметь свою разметку, а может обойтись без неё. Также у фрагмента есть свой жизненный цикл, во многом совпадающий с жизненным циклом активности. Пожалуй, это единственное сходство с активностью.

Имеются специальные виды фрагментов, заточенные под определённые задачи - **ListFragment**, **DialogFragment** и другие, которые изучим в других уроках.

Есть два варианта использования фрагментов в приложении (при желании можно использовать сразу оба варианта). Первый вариант заключается в том, что вы в разметке сразу указываете фрагмент с помощью тега **fragment**, так же как и с другими компонентами.

Второй вариант использует динамическое подключение фрагмента. Принцип следующий - в разметку помещается макет из группы **ViewGroup**, который становится контейнером для фрагмента. Обычно, для этой цели используют **FrameLayout**, но это не обязательное условие. И в нужный момент фрагмент замещает контейнер и становится частью разметки.

Поначалу фрагменты кажутся неудобными, так как количество кода увеличивается. Но если с ними работать постоянно, то станет понятнее их принцип.

Работаем по старинке

Чтобы было легче перестроиться на новую технологию, начнём издалека и создадим сначала стандартную программу. Набросаем на экран несколько кнопок и других компонентов.

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

xmlns:tools="http://schemas.android.com/tools"

android:id="@+id/LinearLayout1"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:gravity="center\_horizontal"

android:orientation="vertical"

android:paddingBottom="@dimen/activity\_vertical\_margin"

android:paddingLeft="@dimen/activity\_horizontal\_margin"

android:paddingRight="@dimen/activity\_horizontal\_margin"

android:paddingTop="@dimen/activity\_vertical\_margin"

tools:context=".MainActivity" >

<Button

android:id="@+id/button1"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="Рыжик" />

<Button

android:id="@+id/button2"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="Барсик" />

<Button

android:id="@+id/button3"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="Мурзик" />

<TextView

android:id="@+id/textView"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Описание кота"

android:textAppearance="?android:attr/textAppearanceLarge" />

<ImageView

android:id="@+id/imageView"

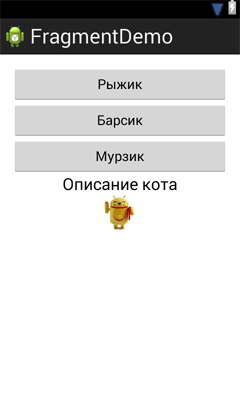
android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:scaleType="fitCenter"

android:src="@drawable/cat\_yellow" />

</LinearLayout>



Думаю, вам уже не составит труда написать код для кнопок, чтобы в нижней части экрана менялась картинка и текстовое содержание про каждого кота. Но я этого делать пока не буду.

Заворачиваем в фрагменты

Фрагмент, как и активность, состоит из разметки и класса. Сначала займёмся разметкой.

Логически экран можно разделить на две части - верхняя неизменяемая часть с кнопками и нижняя часть с текстовым блоком и контейнером для картинки, которая изменяет свой вид в зависимости от нажатой кнопки.

Создадим две отдельные разметки и скопируем нужные части из общей разметки в разметки для фрагментов. Делаем щелчок правой кнопкой мыши на папке **res/layout** и выбираем **New | Layout resource file**.

Создаём новый файл **fragment1.xml** и размещаем верхнюю часть кода:

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

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

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical" >

<Button

android:id="@+id/button1"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="Рыжик" />

<Button

android:id="@+id/button2"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="Барсик" />

<Button

android:id="@+id/button3"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="Мурзик" />

</LinearLayout>

Также поступаем со вторым фрагментом - создаём новый файл **fragment2.xml** и в него копируем код из нижней части кода.

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

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

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical" >

<TextView

android:id="@+id/textView"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Описание кота"

android:textAppearance="?android:attr/textAppearanceLarge" />

<ImageView

android:id="@+id/imageView"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:scaleType="fitCenter"

android:src="@drawable/cat\_yellow" />

</LinearLayout>

В обоих случаях мы вставляли код в корневой контейнер **LinearLayout**. Но если у вас была бы более сложная разметка с использованием контейнерных элементов, то вы могли бы копировать сразу готовый кусок кода без необходимости оборачивать его корневым элементом, как в нашем случае.

Пока мы создали разметки для будущих фрагментов. Теперь нужно создать отдельные классы для двух фрагментов. Делаем щелчок правой кнопкой мыши на имени пакета и выбираем **New | Java Class**. Присваиваем имя **Fragment1**.

На данный момент студия создаёт проекты с использованием библиотеки совместимости, поэтому будем использовать фрагменты из этой библиотеки. В будущем переделать код под обычные фрагменты будет не сложно. Если ваш проект создан сразу под новые устройства (в главной активности строчка **MainActivity extends Activity**), то можете сразу использовать новые фрагменты.

Для начала укажем, что наш класс должен наследоваться от класса **Fragment**. Не копируйте, а пишите код самостоятельно.

public class Fragment1 **extends Fragment** {

}

Следите, чтобы импортировался класс **android.support.v4.app.Fragment**, а не **android.app.Fragment**, который предназначен для новых устройств.

Самостоятельно создайте класс для второго фрагмента **Fragment2** по такому же принципу.

Настало время подключить разметки к фрагментам. В активностях мы подключали разметку в методе **onCreate()**через метод **setContentView()**. В фрагментах метод **onCreate()** служит для других задач. А для подключения разметки используется отдельный метод **onCreateView()**.

Если у вас открыт код для **Fragment1**, то нажимаем комбинацию клавиш **Ctrl+O** (или меню **Code | Override Methods...**). Чтобы долго не искать нужный нам метод, просто вводите на клавиатуре первую и заглавные буквы метода - ocv. Такой комбинации соответствует только один метод, который нам и нужен. Нажимаем кнопку **OK** и в код фрагмента будет вставлен следующий шаблон:

@Nullable

@Override

public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container,

@Nullable Bundle savedInstanceState) {

return super.onCreateView(inflater, container, savedInstanceState);

}

У метода используются три параметра. В первом параметре используется объект класса **LayoutInflater**, который позволяет построить нужный макет, считывая информацию из указанного XML-файла. Удалим строчку, которая возвращает результат и напишем свой вариант.

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

//return super.onCreateView(inflater, container, savedInstanceState);

View rootView =

inflater.inflate(R.layout.fragment1, container, false);

return rootView;

}

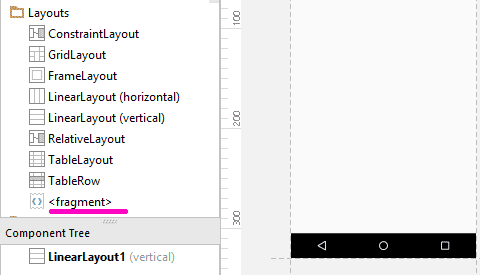
Для удобства код разбит на две части. Сначала мы получаем объект **View**, а затем уже его возвращаем в методе.

Скопируйте код метода **onCreateView()** и вставьте его в код класса **Fragment2**, не забыв указать разметку **R.layout.fragment2**.

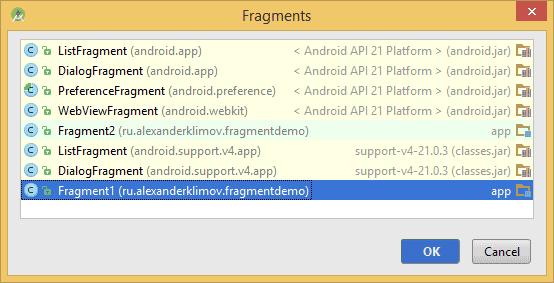
Остальные два параметра **container**, **false** используются в связке и указывают на возможнось подключения фрагментов в активность через контейнер. Мы обойдёмся без контейнеров, а создадим собственные блоки для фрагментов, поэтому у нас используется значение **false**.

Возвращаемся к главной разметке активности и переключаемся в режим **Design**. Смело удаляем все элементы с экрана, чтобы остался только корневой элемент **LinearLayout**, который удалить не получится.

В панели **Palette** находим в разделе **Layout** элемент **<fragment>**.



Щёлкнув на нём, мы вызовем диалоговое окно, в котором нужно найти свой созданный фрагмент.



Разместите его на экране активности и повторите эти действия для размещения второго фрагмента.

В студии версии 2.2.3 пытался повторить эти шаги, но ничего не получилось. Возможно, это баг. Поэтому, если у вас тоже не получилось, то пишем код вручную (см. ниже).

Переключитесь в режим **Text** и посмотрите, как фрагменты описываются в XML-коде. Для фрагментов используется тег **fragment** с стандартными атрибутами. В атрибуте **name** указываются имена классов созданных фрагментов.

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

xmlns:tools="http://schemas.android.com/tools"

android:id="@+id/LinearLayout"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:gravity="center\_horizontal"

android:orientation="vertical"

android:paddingBottom="@dimen/activity\_vertical\_margin"

android:paddingLeft="@dimen/activity\_horizontal\_margin"

android:paddingRight="@dimen/activity\_horizontal\_margin"

android:paddingTop="@dimen/activity\_vertical\_margin"

tools:context=".MainActivity">

<fragment

android:id="@+id/fragment"

android:name="ru.alexanderklimov.fragmentdemo.Fragment1"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_gravity="center\_horizontal" />

<fragment

android:id="@+id/fragment2"

android:name="ru.alexanderklimov.fragmentdemo.Fragment2"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_gravity="center\_horizontal" />

</LinearLayout>

Если вас не пугает XML-код, то можете создавать сразу в редакторе кода.

Стандартные настройки могут нас не устраивать. Укажем другие параметры, влияющие на вес элементов:

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

xmlns:tools="http://schemas.android.com/tools"

android:id="@+id/LinearLayout"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:gravity="center\_horizontal"

android:orientation="vertical"

android:paddingBottom="@dimen/activity\_vertical\_margin"

android:paddingLeft="@dimen/activity\_horizontal\_margin"

android:paddingRight="@dimen/activity\_horizontal\_margin"

android:paddingTop="@dimen/activity\_vertical\_margin"

tools:context=".MainActivity" >

<fragment

android:id="@+id/fragment1"

android:name="ru.alexanderklimov.fragmentdemo.Fragment1"

android:layout\_width="match\_parent"

android:layout\_height="0dp"

android:layout\_weight="1" />

<fragment

android:id="@+id/fragment2"

android:name="ru.alexanderklimov.fragmentdemo.Fragment2"

android:layout\_width="match\_parent"

android:layout\_height="0dp"

android:layout\_weight="1" />

</LinearLayout>

Студия в окне **Preview** сообщает нам о проблемах. Она не может отобразить правильно разметку, так как немного туповата.

Вы можете помочь ей. Щёлкните по ссылке **Use @layout/fragment1**, и нужная разметка фрагмента подключится к разметке активности. Аналогично проделайте со вторым фрагментом. Ссылка **Pick Layout...** выводит диалоговое окно, если надо выбрать разметку из списка.

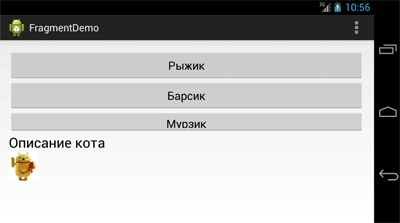
Внешне мы получим такой же результат, как и без фрагментов.

Если посмотрите на XML-код, то увидите, что у тегов **fragment** появились новые строчки **tools:layout="@layout/fragmentX"**.

Если сейчас запустим приложение, то тоже никаких изменений не увидим. Зачем тогда потратили столько времени на создание фрагментов? Непонятно.

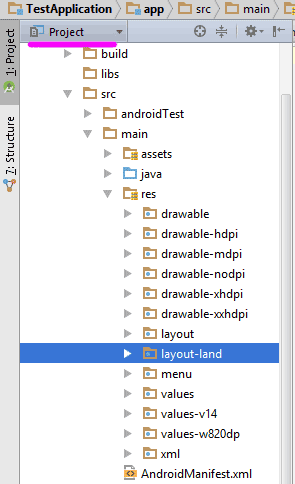
А, я понял. Можно теперь писать в резюме про свои умения: использую фрагменты.

Однако, продолжим. Если повернуть устройство в альбомную ориентацию, то программа будет выглядеть не слишком красиво.



Мы знаем, что можно создать отдельную папку **res/layout-land** (перечитайте урок [Ориентация](http://developer.alexanderklimov.ru/android/orientation.php)) и разместить там разметку для такого случая. Скопируем файл **activity\_main.xml** и вставим его в новую папку.

Скорее всего вы не видите созданную папку, но она есть! Переключитесь в режим **Project** и заново откройте структуру проекта, найдя нужную папку. В дальнейшем оставайтесь в этом режиме для данного урока.



Изменим разметку фрагмента для альбомной ориентации.

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

xmlns:tools="http://schemas.android.com/tools"

android:id="@+id/LinearLayout"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="horizontal"

android:paddingBottom="@dimen/activity\_vertical\_margin"

android:paddingLeft="@dimen/activity\_horizontal\_margin"

android:paddingRight="@dimen/activity\_horizontal\_margin"

android:paddingTop="@dimen/activity\_vertical\_margin"

android:baselineAligned="false"

tools:context=".MainActivity" >

<fragment

android:id="@+id/fragment1"

android:name="ru.alexanderklimov.fragmentdemo.Fragment1"

android:layout\_width="0dp"

android:layout\_height="match\_parent"

android:layout\_weight="1"

tools:layout="@layout/fragment1" />

<fragment

android:id="@+id/fragment2"

android:name="ru.alexanderklimov.fragmentdemo.Fragment2"

android:layout\_width="0dp"

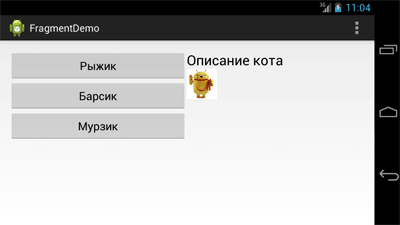
android:layout\_height="match\_parent"

android:layout\_weight="1"

tools:layout="@layout/fragment2" />

</LinearLayout>

Не забывайте в имени фрагментов использовать свои названия пакетов. Совсем другое дело. Теперь в альбомной ориентации приложение выглядит намного лучше.



Но это мы могли сделать и без фрагментов. Зачем же они нужны? Пока версия с лишней строчкой в резюме остаётся основной - чтобы работодатель уважал за прогрессивный стиль.

Хотя небольшое удобство есть. Благодаря модульности, мы поменяли разметку только у фрагментов, а то, что было внутри фрагментов (кнопки, текстовые блоки и т.д.), мы не трогали.

Как уже говорилось, фрагменты были придуманы для того, чтобы обеспечить быстрое написание приложения под разные типы экранов - для смартфонов и планшетов. Часто бывает так, что на смартфоне на первом экране находится список, а когда пользователь нажимает на отдельный элемент списка, то запускается отдельная активность. А на планшете можно уместить список и дополнительные данные на одном экране, как можно увидеть на нашем последнем примере с альбомной ориентацией.

Давайте подключим поддержку планшетов. Создадим новую папку **layout-sw600dp** и скопируем в него файл из папки **layout-land**. Идентификатор **sw600** говорит о минимальной ширине 600dp, что соответствует 7-дюймовым планшетам в альбомной ориентации. Существуют и другие варианты для планшетов с большими размерами.

Тут возникает небольшая проблема - если нам понадобится что-то изменить в разметке для альбомной ориентации, то придётся редактировать файлы во всех папках. Но есть выход из этой ситуации - использовать псевдонимы.

Мы можем создать одну копию разметки и указать, чтобы её использовали все нужные размеры устройств.

Делаем следующее. В папке **layout-land** переименовываем файл **activity\_main.xml** в **activity\_main\_wide.xml**(**Refactor | Rename**) и перемещаем файл в папку **layout**. Пустую папку **layout-land** можно удалить.

Теперь создайте новую папку **res/values-land**. В созданной папке создаём новый файл **refs.xml** (имя не имеет значения, но так принято).

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

<resources>

<item name="activity\_main" type="layout">@layout/activity\_main\_wide

</item>

</resources>

Этот файл говорит, что в альбомной ориентации вместо ресурса **activity\_main** следует подключать ресурс **layout/activity\_main\_wide**. Можете запустить приложение и убедиться, что ничего не изменилось.

Если у вас будет поддержка альбомных ориентаций для разных размеров планшетов, то просто копируйте файл **refs.xml** в папки типа **values-720dp\_land** и др.

Теперь вы можете вносить изменения в одном файле **activity\_main\_wide.xml**, а не по отдельности в каждом файле.

В первой части мы узнали, что для создания фрагмента необходимо создать разметку, затем новый класс и в методе **onCreateView()** указать разметку. Затем в разметке активности указать тег **fragment** и присвоить ему имя класса фрагмента.

# Fragment (Фрагменты). Часть вторая



Поняв, как создавать фрагмент, можно переходить к следующей части - как взаимодействовать с фрагментами.

Поговорим о важном моменте. Вы можете установить связь между двумя фрагментами напрямую, чтобы при нажатии кнопки в первом фрагменте менялось содержимое во втором фрагменте. Но это неправильный подход, так как теряется смысл модульности фрагментов. Фрагменты ничего не должны знать о существовании друг друга. Любой фрагмент существует только в активности и только активность через свой специальный менеджер фрагментов должна управлять ими. А сами фрагменты должны реализовать необходимые интерфейсы, которые активность будет использовать в своих целях.

Начнём издалека. В первом фрагменте имеются кнопки. Добавим обработчик нажатий кнопок (такой же код вы могли использовать в активности, всё знакомо):

public class Fragment1 extends Fragment implements View.OnClickListener {

...

@Override

public void onClick(View view) {

}

}

Подключаем кнопки в методе **onCreateView()**. Код будет похож на код, который мы обычно используем в методе **onCreate()** у активности, только метод **findViewById()** будет относиться уже не к классу **Activity** (обычно, мы опускали это), а к корневому элементу разметки фрагмента, в нашем случае **rootView**.

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

View rootView =

inflater.inflate(R.layout.fragment1, container, false);

**Button button1 = (Button) rootView.findViewById(R.id.button1);**

**Button button2 = (Button) rootView.findViewById(R.id.button2);**

**Button button3 = (Button) rootView.findViewById(R.id.button3);**

**button1.setOnClickListener(this);**

**button2.setOnClickListener(this);**

**button3.setOnClickListener(this);**

return rootView;

}

Для начала просто выведем сообщение, что кнопка нажата.

@Override

public void onClick(View view) {

Toast.makeText(getActivity(), "Вы нажали на кнопку",

Toast.LENGTH\_SHORT).show();

}

Запустите пример и проверьте. Но у нас три кнопки. Надо написать код, который бы получал информацию о нажатой кнопке, чтобы активность могла использовать эту информацию и использовать её для управления вторым фрагментом. Для удобства создадим отдельный метод, который на основании идентификатора кнопки создаст нужный индекс:

int translateIdToIndex(int id) {

int index = -1;

switch (id) {

case R.id.button1:

index = 1;

break;

case R.id.button2:

index = 2;

break;

case R.id.button3:

index = 3;

break;

}

return index;

}

Каждой кнопке соответствует свой индекс от 1 до 3.

Фрагмент всегда может узнать, в какой активности он находится, через метод **getActivity()**. В методе **makeText()** мы уже воспользовались данным методом, так как в фрагментах нет метода **getApplicationContext()**.

Перепишем код для щелчка кнопки, чтобы узнать индекс нажатой кнопки.

@Override

public void onClick(View view) {

int buttonIndex = translateIdToIndex(v.getId());

// Временный код для получения индекса нажатой кнопки

Toast.makeText(getActivity(), String.valueOf(buttonIndex),

Toast.LENGTH\_SHORT).show();

}

Теперь мы умеем определять индекс нажатой кнопки. Но пока эта информация доступна только самому фрагменту. Наша задача - передать эту информацию активности, которая затем передаст её другой активности.

Для этой цели используются интерфейсы.

Открываем код первого фрагмента **Fragment1** и объявляем интерфейс с единственным методом:

public interface OnSelectedButtonListener {

void onButtonSelected(int buttonIndex);

}

Интерфейс не определяет работу метода, а только даёт ему имя. Класс, который будет использовать данный интерфейс, должен придумать, что делать в методе с данным именем.

У нас интерфейс будет использовать класс активности.

Переходим в класс активности и добавляем интерфейс **OnSelectedButtonListener**, который следует реализовать.

public class MainActivity extends AppCompatActivity implements Fragment1.OnSelectedButtonListener { ...

}

Среда разработки поможет создать заготовку для необходимого метода:

@Override

public void onButtonSelected(int buttonIndex) {

}

В этом методе надо написать такой код, чтобы активность получила индекс нажатой кнопки и передала информацию другому фрагменту, которая должна выполнить свою работу.

Но сначала подготовим второй фрагмент к работе.

Объявим ссылки на компоненты, которые есть в разметке второго фрагмента. А также загрузим массив строк из ресурсов, который будем использовать для описания котов.

public class Fragment2 extends Fragment {

private TextView mInfoTextView;

private ImageView mCatImageView;

private String[] mCatDescriptionArray;

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

View rootView =

inflater.inflate(R.layout.fragment2, container, false);

mInfoTextView = (TextView) rootView.findViewById(R.id.textView);

mCatImageView = (ImageView) rootView.findViewById(R.id.imageView);

// загружаем массив из ресурсов

mCatDescriptionArray = getResources().getStringArray(R.array.cats);

return rootView;

}

}

Массив задаём в ресурсах (файл **res/values/strings.xml**). Так как первый элемент массива идёт под индексом 0, то добавим нейтральный текст:

<string-array name="cats">

<item>Просто кот</item>

<item>Рыжик - рыжий кот</item>

<item>Барсик болеет за Барселону</item>

<item>Мурзик выписывает Мурзилку</item>

</string-array>

Подготовим метод, который будет менять содержимое фрагмента в зависимости от индекса нажатой кнопки:

public void setDescription(int buttonIndex) {

String catDescription = mCatDescriptionArray[buttonIndex];

mInfoTextView.setText(catDescription);

switch (buttonIndex) {

case 1:

mCatImageView.setImageResource(R.drawable.cat\_yellow);

break;

case 2:

mCatImageView.setImageResource(R.drawable.cat\_white);

break;

case 3:

mCatImageView.setImageResource(R.drawable.cat\_green);

break;

default:

break;

}

}

Осталось только получить информацию от активности (не от фрагмента) об индексе.

Опять возвращаемся в активность и напишем код для пустого метода **onButtonSelected()**, который будет получать от первого фрагмента индекс нажатой кнопки и передавать его второму фрагменту:

@Override

public void onButtonSelected(int buttonIndex) {

// подключаем FragmentManager

FragmentManager fragmentManager = getSupportFragmentManager();

// Получаем ссылку на второй фрагмент по ID

Fragment2 fragment2 = (Fragment2) fragmentManager

.findFragmentById(R.id.fragment2);

// Выводим нужную информацию

if (fragment2 != null)

fragment2.setDescription(buttonIndex);

}

Активность получает доступ к своим фрагментам через специальный менеджер фрагментов (коты называют его манагером). Менеджер есть у любой активности, поэтому мы его не создаём через конструкцию **new FragmentManager**, а получаем через метод **getSupportFragmentManager()** или для новых версий **getFragmentManager()**.

Так было не всегда, менеджеры появились у активностей только в Android 3.0. А старые активности понятия не имели о них. По этой причине и добавили библиотеку совместимости, в которой переписали код, добавив новые классы в пакетах **android.support.xxx**, чтобы новый тип активностей работал на старых устройствах. По сути, библиотека является патчем к операционной системе.

Менеджер фрагментов держит в руках все нити управления над своими фрагментами. Найти нужный фрагмент можно по идентификатору через метод **FragmentManager.findFragmentById()**, который похож на метод **findViewById()** для поиска кнопки, метки и т.д. У менеджера есть ещё один метод для поиска фрагмента по тегу **findFragmentByTag()**.

В созданной заготовке вызываем менеджер фрагментов, получаем ссылку на второй фрагмент через его идентификатор и вызываем его метод **setDescription()**.

Во втором фрагменте у нас нет надобности создавать интерфейс, так как фрагменту не нужно ничего сообщать активности. Он исполняет пассивную роль и ему нужно только получить данные для работы.

Опять возвращаемся к классу первого фрагмента. Сейчас нажатие кнопки приводит к появлению всплывающего сообщения об индексе кнопки.

Но теперь в методе **onClick()** мы можем получить доступ к слушателю активности.

@Override

public void onClick(View view) {

int buttonIndex = translateIdToIndex(view.getId());

OnSelectedButtonListener listener = (OnSelectedButtonListener) getActivity();

listener.onButtonSelected(buttonIndex);

// Можно закомментировать

Toast.makeText(getActivity(), String.valueOf(buttonIndex),

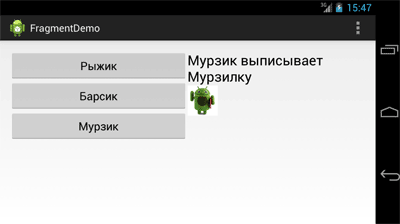
Toast.LENGTH\_SHORT).show();

}

По цепочке мы передаём информацию от первого фрагмента в активность, а затем активность передаёт информацию во второй фрагмент.

Если посмотреть на код двух фрагментов, то увидим, что они полностью независимы и не обращаются ни конкретно к друг другу, ни к определённой активности. Принцип модульности соблюдён. Вы можете добавить любой из этих фрагментов в любую новую активность и при этом вам не придётся менять код в самих фрагментах. Весь необходимый функционал в фрагментах уже прописан.

Запустите проект и проверьте на работоспособность. Для данного случая мы пока не получили никаких преимуществ в использовании фрагментов. Но сейчас главное для вас - понять основные принципы создания и взаимодействия фрагментов.



# Fragment (Фрагменты). Часть третья



А теперь представим, что котов много (котов много не бывает), и нам придётся добавлять новые кнопки на экран к трём имеющимся. И тогда совсем не останется места картинке и сопроводительному тексту в портретной ориентации, которые уползут вниз и будут не видны. Для решения этой задачи можно кнопки оставить на первой активности, а картинку с текстом перенести на вторую активность. Для альбомной ориентации можно оставить как было.

Создадим вторую активность **SecondActivity** через мастер создания активностей. Скопируем в его разметку вторую нижнюю часть из разметки от первой активности (**activity\_second.xml**):

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

xmlns:tools="http://schemas.android.com/tools"

android:id="@+id/linearLayout"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:gravity="center\_horizontal"

android:orientation="vertical"

android:paddingBottom="@dimen/activity\_vertical\_margin"

android:paddingLeft="@dimen/activity\_horizontal\_margin"

android:paddingRight="@dimen/activity\_horizontal\_margin"

android:paddingTop="@dimen/activity\_vertical\_margin"

tools:context=".MainActivity" >

<fragment

android:id="@+id/fragment2"

android:name="ru.alexanderklimov.fragmentdemo.Fragment2"

android:layout\_width="match\_parent"

android:layout\_height="0dp"

android:layout\_weight="1"

tools:layout="@layout/fragment2" />

</LinearLayout>

А в файле **activity\_main.xml**, наоборот, удалим второй фрагмент, чтобы остался только первый фрагмент с кнопками.

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

xmlns:tools="http://schemas.android.com/tools"

android:id="@+id/linearLayout"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:gravity="center\_horizontal"

android:orientation="vertical"

android:paddingBottom="@dimen/activity\_vertical\_margin"

android:paddingLeft="@dimen/activity\_horizontal\_margin"

android:paddingRight="@dimen/activity\_horizontal\_margin"

android:paddingTop="@dimen/activity\_vertical\_margin"

tools:context=".MainActivity" >

<fragment

android:id="@+id/fragment1"

android:name="ru.alexanderklimov.fragmentdemo.Fragment1"

android:layout\_width="match\_parent"

android:layout\_height="0dp"

android:layout\_weight="1"

tools:layout="@layout/fragment1" />

</LinearLayout>

Если запустить приложение, то столкнёмся с такой ситуацией. У нас запустится активность с кнопками. Повернём устройство в альбомную ориентацию и увидим знакомый интерфейс, когда слева располагаются кнопки, а справа - картинка и текст. Мы знаем, что на экране сейчас два фрагмента и нажатия на кнопки по-прежнему работают.

Повернём устройство обратно в портретную ориентацию и попробуем нажать на любую кнопку. Программа аварийно закроется. Что произошло? А произошло следующее. На экране первой активности в портретном режиме у нас теперь один фрагмент, а вот код об этом не знает и пытается обратиться к фрагменту, которого не существует для него. Ведь разметка поменялась.

Иными словами, нам нужно отслеживать ориентацию или размеры устройства и выбирать, какой код использовать - с участием фрагментов или запускать вторую активность.

Прежде чем решать эту задачу, добавим код во вторую активность.

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_second);

// Получим индекс из намерения активности

Intent intent = getIntent();

int buttonIndex = intent.getIntExtra("buttonIndex", -1);

if (buttonIndex != -1) {

FragmentManager fragmentManager = getSupportFragmentManager();

Fragment2 fragment2 = (Fragment2) fragmentManager

.findFragmentById(R.id.fragment2);

fragment2.setDescription(buttonIndex);

}

}

Мы знаем, что на другую активность можно перейти через намерение и передавать данные через метод **putExtra()**. А вторая активность, соответственно, может запуститься и получить данные от первой активности через принимающий метод **getIntExtra()**.

Вторую активность интересует только индекс нажатой кнопки. Получив его, мы вызываем менеджер фрагментов, находим по идентификатору сам фрагмент **Fragment2** и вызываем его метод **setDescription()**, передав ему индекс. Как видите, мы по-прежнему, не вносим изменений в код для фрагмента, всё делается на уровне активностей.

Перейдём к первой активности. Тут придётся попотеть.

Когда устройство находится в альбомной ориентации, то всё нормально. В активности присутствуют два фрагмента из разметки **res/layout/activity\_main\_wide.xml** и код соответствует этому состоянию.

Таким образом нам надо отслеживать ситуацию и поправить код в методе **onButtonSelected()**.

@Override

public void onButtonSelected(int buttonIndex) {

// подключаем FragmentManager

FragmentManager fragmentManager = getSupportFragmentManager();

// Получаем ссылку на второй фрагмент по ID

Fragment2 fragment2 = (Fragment2) fragmentManager

.findFragmentById(R.id.fragment2);

// если фрагмента не существует или он невидим

if (fragment2 == null || !fragment2.isVisible()) {

// запускаем активность

Intent intent = new Intent(this, SecondActivity.class);

intent.putExtra("buttonIndex", buttonIndex);

startActivity(intent);

} else {

// Выводим нужную информацию

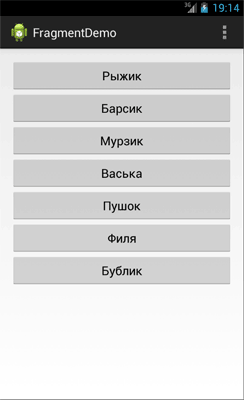
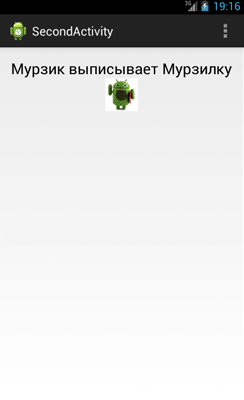
fragment2.setDescription(buttonIndex);

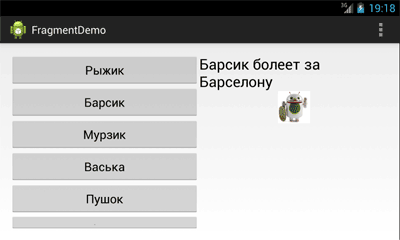
}

}

А отслеживать мы будем через условие **if**. Если фрагмента не существует или он невидим, то запускаем вторую активность. В противном случае фрагмент присутствует в активности, и тогда работаем по старому сценарию.

Запустив код, вы теперь можете нажимать на кнопки в любой ориентации. Программа в состоянии определить, какой код следует выполнять - запускать новую активность или работать в одной активности с двумя фрагментами.



Остался один штрих. Если в портретной ориентации перейти на вторую активность и повернуть устройство в альбомную ориентацию, то увидим не тот результат, на который рассчитывали. Мы ожидаем увидеть двухпанельный режим, а видим альбомную ориентации второй активности. Добавим в метод **onCreate()** второй активности проверку текущего состояния ориентации. Если обнаружим, что находимся в альбомной ориентации, то просто выходим из активности. Конечно, можно дополнительно запоминать, какая кнопка была нажата, чтобы при возврате на первую активность вывести соответствующие картинки, но это уже детали.

...

setContentView(R.layout.activity\_second);

if (getResources().getConfiguration().orientation == Configuration.ORIENTATION\_LANDSCAPE) {

finish();

return;

}

...

Для новичков тема с фрагментами может показаться сложной из-за обилия кода с применением интерфейсов и всякой ерунды. Тем более, что мы рассмотрели только часть возможностей фрагментов.

На последнем рисунке видно, что не все кнопки поместились на экран. Можно было добавить контейнер **ScrollView** для решения проблемы. Но в большинстве случаев используют специальный фрагмент **ListFragment**. Если помните, когда на экране имеется только один компонент **ListView**, то можно использовать готовую активность **ListActivity**. Аналогично, если фрагмент состоит только из **ListView**, то нужно использовать **ListFragment**.

# Fragment (Фрагменты). Часть четвёртая



## Динамическое управление фрагментами

Мы использовали теги **fragment** в разметке для размещения фрагментов. Но существует ещё альтернативный вариант, когда фрагмент вставляется в какой-нибудь контейнер динамически. В качестве контейнера обычно используют **FrameLayout**, но можно и другие элементы из **ViewGroup**. Давайте попробуем изменить свою программу под новый способ.

В разметке **activity\_main.xml** удалим тег **fragment**, а корневому компоненту присвоим идентификатор (у нас уже был идентификатор **@+id/linearLayout**, но я решил его заменить на более говорящий).

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

xmlns:tools="http://schemas.android.com/tools"

android:id="@+id/container"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:gravity="center\_horizontal"

android:orientation="vertical"

android:paddingBottom="@dimen/activity\_vertical\_margin"

android:paddingLeft="@dimen/activity\_horizontal\_margin"

android:paddingRight="@dimen/activity\_horizontal\_margin"

android:paddingTop="@dimen/activity\_vertical\_margin"

tools:context=".MainActivity" >

<!-- <fragment -->

<!-- android:id="@+id/fragment1" -->

<!-- android:name="ru.alexanderklimov.fragmentdemo.Fragment1" -->

<!-- android:layout\_width="match\_parent" -->

<!-- android:layout\_height="0dp" -->

<!-- android:layout\_weight="1" -->

<!-- tools:layout="@layout/fragment1" /> -->

</LinearLayout>

Ещё раз обращаю внимание, что заменить статический фрагмент программно нельзя. Если вы используете в разметке тег **fragment**, то это уже навсегда.

А у нас сложилась странная ситуация, мы удалили фрагмент из разметки и на что-то рассчитываем? Наивные.

На самом деле, всё не так безнадёжно. Сам класс фрагмента **Fragment1** у нас остался и его по-прежнему можно использовать.

Добавим в класс **MainActivity** новую переменную:

private boolean mIsDynamic;

Далее в методе **onCreate()** устроим небольшую проверку для второго фрагмента. Если он не существует (**null**) или не является частью разметки (**isInLayout**), то переменная **mIsDynamic** будет иметь значение **true**.

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

FragmentManager fragmentManager = getSupportFragmentManager();

Fragment2 fragment2 = (Fragment2) fragmentManager

.findFragmentById(R.id.fragment2);

mIsDynamic = fragment2 == null || !fragment2.isInLayout();

Toast.makeText(getApplicationContext(), mIsDynamic + "", Toast.LENGTH\_SHORT).show();

}

Я специально добавил в код вызов всплывающего сообщения, чтобы вы увидели, как меняется значение переменной **mIsDynamic**. При запуске в портретном режиме вы увидите пустой экран, так как удалили из разметки фрагмент, а переменная будет равна **true**. При повороте в альбомный режим загрузится альтернативная разметка с двумя фрагментами, которую мы не трогали. А булева переменная примет значение **false**.

Теперь, когда мы знаем, что при старте второй фрагмент не используется, загрузим программно первый фрагмент в том же методе **onCreate()**.

// Зная, что второго фрагмента нет, загружаем первый

if (mIsDynamic) {

// начинаем транзакцию

FragmentTransaction ft = fragmentManager.beginTransaction();

// Создаем и добавляем первый фрагмент

Fragment1 fragment1 = new Fragment1();

ft.add(R.id.container, fragment1, "fragment1");

// Подтверждаем операцию

ft.commit();

}

Активность может иметь несколько фрагментов, которые должны выполнить какую-то операцию. Транзакция позволяет выполнить все операции скопом. Вы сообщаете менеджеру про все операции, запускаете их в методе **beginTransaction()** и подтверждаете своё намерение через метод **commit()**.

В методе **add()** мы указываем идентификатор контейнера **R.id.container**, в который нужно загрузить наш фрагмент. Запустите пример, чтобы увидеть, что приложение работает как и раньше.

Для несложного примера этого вполне достаточно, но иногда требуется более сложное взаимодействие между фрагментами. Поэтому продолжим эксперименты.

Перепишем у первой активности метод **onButtonSelected()** с использованием переменной **mIsDynamic**:

@Override

public void onButtonSelected(int buttonIndex) {

// подключаем FragmentManager

FragmentManager fragmentManager = getSupportFragmentManager();

Fragment2 fragment2;

// Если фрагмент недоступен

if (mIsDynamic) {

// Динамическое переключение на другой фрагмент (позже)

} else {

// Если фрагмент доступен

fragment2 = (Fragment2) fragmentManager

.findFragmentById(R.id.fragment2);

fragment2.setDescription(buttonIndex);

}

}

Для альбомного режима всё осталось без изменений. Если второй фрагмент доступен, то выводим данные в соответствии с нажатой кнопкой.

С портретной ориентацией ситуация интереснее. Мы можем не запускать новую активность, которая содержит второй фрагмент, а динамически заменить первый фрагмент на второй. Удобно! Нам не нужна лишняя активность. Напишем код для условия **if** из предыдущего примера.

if (mIsDynamic) {

// Динамическое переключение на другой фрагмент

FragmentTransaction ft = fragmentManager.beginTransaction();

fragment2 = new Fragment2();

ft.replace(R.id.container, fragment2, "fragment2");

ft.addToBackStack(null);

ft.setCustomAnimations(

android.R.animator.fade\_in, android.R.animator.fade\_out);

ft.commit();

}

Запускаем проект и проверяем. Действительно, вместо первого фрагмента появляется второй. Правда при этом никак не учитывается нажатая кнопка. Непорядок.

А происходит это потому, что транзакция вызывается чуть раньше, чем методы фрагмента **onCreate()** и **onCreateView()**. Мы пойдём другим путём. У фрагментов есть методы **getArguments/setArguments()**, способные передавать и принимать данные.

Объявим переменные в классе **Fragment2**:

// Имя для аргумента

public static final String BUTTON\_INDEX = "button\_index";

// Значение по умолчанию

private static final int BUTTON\_INDEX\_DEFAULT = -1;

Переделаем в этом же классе метод **onCreateView()** с применением аргументов.

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

View rootView =

inflater.inflate(R.layout.fragment2, container, false);

mInfoTextView = (TextView) rootView.findViewById(R.id.textView1);

mCatImageView = (ImageView) rootView.findViewById(R.id.imageView1);

// загружаем массив из ресурсов

mCatDescriptionArray = getResources().getStringArray(R.array.cats);

**// Получим индекс, если имеется**

**Bundle args = getArguments();**

**int buttonIndex = args != null ? args.getInt(BUTTON\_INDEX,**

**BUTTON\_INDEX\_DEFAULT) : BUTTON\_INDEX\_DEFAULT;**

**// Если индекс обнаружен, то используем его**

**if (buttonIndex != BUTTON\_INDEX\_DEFAULT)**

**setDescription(buttonIndex);**

return rootView;

}

Если в фрагмент поступят данные, то обрабатываем их и выводим нужную информацию. Осталось только подготовить нужные данные. Сделаем это в методе первой активности **onButtonSelected()**

@Override

public void onButtonSelected(int buttonIndex) {

// подключаем FragmentManager

FragmentManager fragmentManager = getSupportFragmentManager();

Fragment2 fragment2;

// Если фрагмент недоступен

if (mIsDynamic) {

// Динамическое переключение на другой фрагмент

FragmentTransaction ft = fragmentManager.beginTransaction();

fragment2 = new Fragment2();

// Подготавливаем аргументы

Bundle args = new Bundle();

args.putInt(Fragment2.BUTTON\_INDEX, buttonIndex);

fragment2.setArguments(args);

ft.replace(R.id.container, fragment2, "fragment2");

ft.addToBackStack(null);

ft.setCustomAnimations(

android.R.animator.fade\_in, android.R.animator.fade\_out);

ft.commit();

} else {

// Если фрагмент доступен

fragment2 = (Fragment2) fragmentManager

.findFragmentById(R.id.fragment2);

fragment2.setDescription(buttonIndex);

}

}

Это был заключительный аккорд. Мы переделали приложение, полностью отказавшись от второй активности, так как все фрагменты загружаются в один и тот же контейнер.

Дополнительные материалы по фрагментам есть в [отдельном цикле статей](http://developer.alexanderklimov.ru/android/theory/fragments.php).

Важно понять, в каких случаях удобнее использовать статичные фрагменты, а в каких - динамические. В статье мы многое делали вручную. Но в студии есть готовые шаблоны, и поэтому часть работы будут автоматизирована. Кроме того есть ещё различные нюансы, о которых вы узнаете с практикой.

# Fragment (Фрагменты). Часть пятая



## Сохранение данных

Теперь рассмотрим важный вопрос, как сохранять данные при изменении конфигурации и других операциях.

Для примера возьмём старый урок по подсчёту ворон, но на этот раз будем считать котов. Вороны вечно порхают с ветки на ветку, их сложно считать. Коты - совсем другое дело, они лежат себе на одном месте и спят. Считать их одно удовольствие.

Кажется, я поторопился со своим утверждением.

Новый проект создавать не будем, а модифицируем старый. Заодно закрепим материал.

Создадим в проекте два новых фрагмента: **WithButtonFragment** и **WithTextViewFragment**. Из названий понятно, что в первом фрагменте будет кнопка, а во втором - **TextView**, в котором будет отражаться информация о количестве котов.

Повторяем прошлые шаги. Создаём новый класс, наследуемся от **Fragment**.

package ru.alexanderklimov.fragmentdemo;

import android.support.v4.app.Fragment;

public class WithButtonFragment extends Fragment {

}

Второй фрагмент.

package ru.alexanderklimov.fragmentdemo;

import android.support.v4.app.Fragment;

public class WithTextViewFragment extends Fragment {

}

Создадим две разметки для них. Для удобства будем использовать фон, чтобы различать фрагменты на экране.

**fragment\_with\_button.xml**

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

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

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:background="@android:color/holo\_green\_light"

android:orientation="vertical">

<Button

android:id="@+id/button"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center\_horizontal"

android:layout\_margin="20dp"

android:text="Считаем котов" />

</LinearLayout>

**fragment\_with\_textview.xml**

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

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

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:background="@android:color/holo\_blue\_light"

android:orientation="vertical">

<TextView

android:id="@+id/textView"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center\_horizontal"

android:layout\_margin="20dp"

android:text="Всего:"

android:textAppearance="?android:attr/textAppearanceLarge" />

</LinearLayout>

Подключаем разметки к соответствующим фрагментам в методе **onCreateView()**.

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {

return inflater.inflate(R.layout.fragment\_with\_button, container, false);

}

Для второго фрагмента напишите код самостоятельно.

Подключаем фрагменты в разметке основной активности **activity\_main.xml**:

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

xmlns:tools="http://schemas.android.com/tools"

android:id="@+id/container"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:gravity="center\_horizontal"

android:orientation="vertical"

android:paddingBottom="@dimen/activity\_vertical\_margin"

android:paddingLeft="@dimen/activity\_horizontal\_margin"

android:paddingRight="@dimen/activity\_horizontal\_margin"

android:paddingTop="@dimen/activity\_vertical\_margin"

tools:context=".MainActivity">

<fragment

android:id="@+id/fragment\_withButton"

android:name="ru.alexanderklimov.fragmentdemo.WithButtonFragment"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_gravity="center\_horizontal"

tools:layout="@layout/fragment\_with\_button" />

<fragment

android:id="@+id/fragment\_withTextView"

android:name="ru.alexanderklimov.fragmentdemo.WithTextViewFragment"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_gravity="center\_horizontal"

tools:layout="@layout/fragment\_with\_textview" />

</LinearLayout>

Не копируйте код с сайта, проделывайте операции самостоятельно через редактор в режиме **Design** и **Text**, как это объяснялось в предыдущих примерах.

Подключим кнопку в первом фрагменте.

package ru.alexanderklimov.fragmentdemo;

import android.os.Bundle;

import android.support.v4.app.Fragment;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.Button;

public class WithButtonFragment extends Fragment implements View.OnClickListener {

private int mCounter = 0; // счётчик котов

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {

View rootView =

inflater.inflate(R.layout.fragment\_with\_button, container, false);

Button button = (Button)rootView.findViewById(R.id.button);

button.setOnClickListener(this);

return rootView;

}

@Override

public void onClick(View v) {

mCounter++;

}

}

Во втором фрагменте подключим компонент **TextView** и добавим метод для вывода текста.

package ru.alexanderklimov.fragmentdemo;

import android.os.Bundle;

import android.support.v4.app.Fragment;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.TextView;

public class WithTextViewFragment extends Fragment {

private TextView mTextView;

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {

View rootView =

inflater.inflate(R.layout.fragment\_with\_textview, container, false);

mTextView = (TextView) rootView.findViewById(R.id.textView);

return rootView;

}

public void changeText(String data) {

mTextView.setText(data);

}

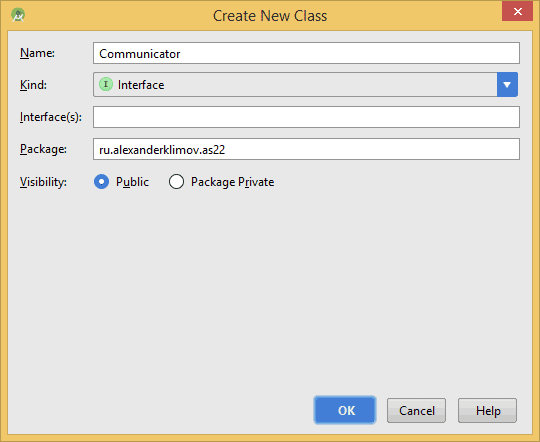
}

Если мы вызовем метод **changeText()** с каким-нибудь текстом, то текст отобразится в **TextView**. Наша задача - научить фрагменты обмениваться данными через активность.

Если сейчас запустить пример, то фрагменты будут работать независимо друг от друга. Счётчик будет увеличиваться, но его значения пока не попадают во второй фрагмент и мы не можем увидеть число котов.

Напоминаю, мы не связываем два фрагмента между собой напрямую. Создаём интерфейс для этих целей.

В предыдущем примере мы создавали интерфейс внутри первого фрагмента. Для разнообразия изменим подход. Создадим новый класс через **New | Java Class** и в диалоговом окне для поля **Kind** выберем **Interface**. Также зададим ему имя **Communicator**.



Интерфейс будет состоять из одного метода.

package ru.alexanderklimov.fragmentdemo;

public interface Communicator {

public void count(String data);

}

Активность должна реализовать интерфейс.

public class MainActivity extends ActionBarActivity implements Communicator {

@Override

public void count(String data) {

}

}

Фрагмент с кнопкой может использовать объект интерфейса для отправки данных. Инициализируем его в новом методе **onActivityCreated()**, который ранее не использовали. Метод сработает, когда активность будет создана и готова к работе.

package ru.alexanderklimov.fragmentdemo;

import android.os.Bundle;

import android.support.v4.app.Fragment;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.Button;

public class WithButtonFragment extends Fragment implements View.OnClickListener {

private int mCounter = 0; // счётчик котов

private Communicator mCommunicator;

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

View rootView =

inflater.inflate(R.layout.fragment\_with\_button, container, false);

Button button = (Button)rootView.findViewById(R.id.button);

button.setOnClickListener(this);

return rootView;

}

@Override

public void onActivityCreated(Bundle savedInstanceState) {

super.onActivityCreated(savedInstanceState);

mCommunicator = (Communicator)getActivity();

}

@Override

public void onClick(View v) {

mCounter++;

mCommunicator.count("Я насчитал " + mCounter + " котов");

}

}

При щелчках на кнопках мы вызываем метод **count()**, которому передаём нужную информацию.

А сам метод в активности будет выглядеть следующим образом:

@Override

public void count(String data) {

FragmentManager manager = getSupportFragmentManager();

WithTextViewFragment withTextViewFragment =

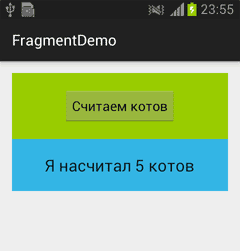
(WithTextViewFragment) manager.findFragmentById(R.id.fragment\_withTextView);

withTextViewFragment.changeText(data);

}

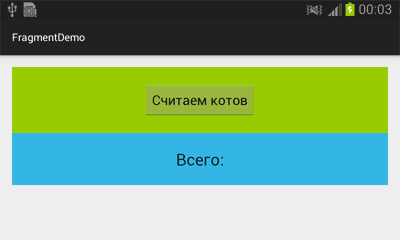
Фрагмент передаёт через метод **count()** данные **data**, а активность их принимает и передаёт их в метод второго фрагмента **changeText()**.

Подготовительные работы завершены и можно запустить пример для проверки.



Если вы создавали пример на основе предыдущих уроков, то не поворачивайте экран. Сначала удалите (или переименуйте) файлы для альбомной ориентации, чтобы не получить ошибку и крах приложения.

Теперь переходим непосредственно к теме нашего урока. Повернув экран, мы обнаружим, что данные из текстового блока пропали. Наши подсчёты - коту под хвост!



Мы можем в новой ориентации начать новый подсчёт, но повернув устройство обратно в портретный режим, снова потеряем данные.

Мы знаем, что при поворотах активность создаётся заново. Поэтому все данные сбрасываются. Чтобы сохранить данные, у фрагментов есть соответствующие методы, схожие с подобными методами у активностей. Задействуем их.

Метод **onSaveInstanceState()** поможет нам. Добавим метод в первый фрагмент с кнопкой.

@Override

public void onSaveInstanceState(Bundle outState) {

super.onSaveInstanceState(outState);

outState.putInt("counter", mCounter);

}

Параметр метода **outState** является объектом класса **Bundle** и позволяет хранить различные типы в формате "ключ-значение".

У фрагментов также есть метод **onCreate(Bundle savedInstanceState)**, где используется объект этого же класса **Bundle** только под другим именем **savedInstanceState**. Несмотря на разные имена, речь идёт об одном и том же объекте. И сохраняя данные в методе **onSaveInstanceState()**, мы можем их получить в методе **onCreate()**:

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

if (savedInstanceState == null) {

mCounter = 0;

} else {

mCounter = savedInstanceState.getInt("counter", 0);

}

}

При повороте фрагмент сохранит значение счётчика, перезапустится и восстановит значение счётчика. Если запустить пример, то увидим следующее. Щёлкнем несколько раз по кнопке и повернём экран. Данные сбросятся и мы снова увидим пустой текст. Но стоит нам нажать на кнопку, то увидим, что отсчёт пошёл не сначала, а продолжил со своего последнего значения. Мы видим, что первый фрагмент запоминает свои данные. А второй фрагмент пока тупит. Поможем ему.

package ru.alexanderklimov.fragmentdemo;

import android.os.Bundle;

import android.support.v4.app.Fragment;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.TextView;

public class WithTextViewFragment extends Fragment {

private TextView mTextView;

**private String mData;**

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {

View rootView =

inflater.inflate(R.layout.fragment\_with\_textview, container, false);

mTextView = (TextView) rootView.findViewById(R.id.textView);

**if(savedInstanceState == null){**

**}else {**

**mData = savedInstanceState.getString("text");**

**mTextView.setText(mData);**

**}**

return rootView;

}

**@Override**

**public void onSaveInstanceState(Bundle outState) {**

**super.onSaveInstanceState(outState);**

**outState.putString("text", mData);**

**}**

public void changeText(String data) {

**mData = data;**

mTextView.setText(mData);

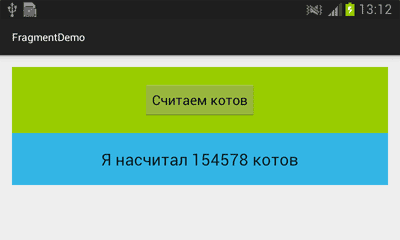
}

}

Добавим новую переменную **mData**, которая будет хранить текст сообщения. У метода **onCreateView()** также есть параметр **savedInstanceState** класса **Bundle**, позволяющий извлечь сохранённые данные.

А текст мы сохраним в методе **onSaveInstanceState()**. В методе **changeText()** добавим строку кода, чтобы текст брался из новой переменной. Теперь при первом запуске всё работает как прежде. При повороте текст из **mData** сохраняется в методе **putString()** и восстанавливается через **getString()**.

После этих изменений программа больше не теряет своих данных и можно спокойно считать котов. Я стал смотреть на гифку и считать. Через 8 минут 16 секунд последний кот выпрыгнул в окно и в итоге получилось 154578 котов. Перепроверьте.



Fragment (Фрагменты). Часть шестая



Удержание состояния фрагментов

В предыдущей статье мы рассмотрели, как сохранять данные при изменении конфигурации и других операциях.

Но есть другая проблема - нам нужно сохранить не просто данные, а работающий объект. К примеру, наш объект воспроизводит музыку или видео. Мы можем сохранить название песни, время проигрывания и т.д, но при повороте музыка прервётся, так как объект будет уничтожен вместе с фрагментом. Нужен другой подход.

У фрагмента есть свойство **retainInstance**, которое по умолчанию содержит значение **false**. Это означает, что при поворотах фрагмент не сохраняется, а уничтожается и создается заново вместе с активностью-хостом. Вызов **setRetainInstance(true)** сохраняет фрагмент, который не уничтожается вместе с активностью, а передается новой активности в неизменном виде.

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setRetainInstance(true);

}

Представление фрагмента может уничтожаться и создаваться заново без необходимости уничтожать сам фрагмент. При изменении конфигурации (повороте) **FragmentManager** сначала уничтожает макет фрагмента в своем списке. Макеты фрагментов всегда уничтожаются и создаются заново по тем же причинам, по которым уничтожаются и создаются заново разметки активности: в новой конфигурации могут потребоваться новые ресурсы. Затем **FragmentManager** проверяет свойство **retainInstance** каждого фрагмента. Если оно равно **false** (по умолчанию), **FragmentManager** уничтожает экземпляр фрагмента. Фрагмент и его макет будут созданы заново новым экземпляром **FragmentManager** новой активности.

Если значение **retainInstance** равно *true*, макет фрагмента уничтожается, но сам фрагмент остаётся. При создании новой активности новый экземпляр **FragmentManager** находит сохранённый фрагмент и воссоздаёт его макет.

Сохранённый фрагмент не уничтожается, а отсоединяется (detached) от «умирающей» активности. В сохранённом состоянии фрагмент всё ещё существует, но не имеет активности-хоста.

Переход в сохранённое состояние происходит только при выполнении двух условий:

* для фрагмента был вызван метод **setRetainInstance(true)**
* активность-хост уничтожается для изменения конфигурации (обычно поворот). Фрагмент находится в сохранённом состоянии очень недолго — от момента отсоединения от старой активности до повторного присоединения к новой, немедленно создаваемой активности.

Сохранённые фрагменты продолжают существовать только при уничтожении активности при изменения конфигурации. Если активность уничтожается из-за того, что системе потребовалось освободить память, то все сохранённые фрагменты также будут уничтожены.