Рассказывая о разработке под Android невозможно не упомянуть **фрагменты**.

Что такое Fragment?

Fragment — это специальный элемент интерфейса, предназначенный для того, чтобы облегчить создание адаптивных приложений, которые работают и на смартфонах, и на планшетах.

Фрагмент содержит внутри себя элементы интерфейса, точно так же, как и Activity, однако, между этими двумя понятиями есть несколько ключевых отличий:

* **Фрагмент** содержится внутри **Activity**.
* Внутри **Activity** может находиться несколько **фрагментов**, то есть на экране может быть несколько фрагментов сразу, тогда как **Activity** всегда одна в каждый момент времени.

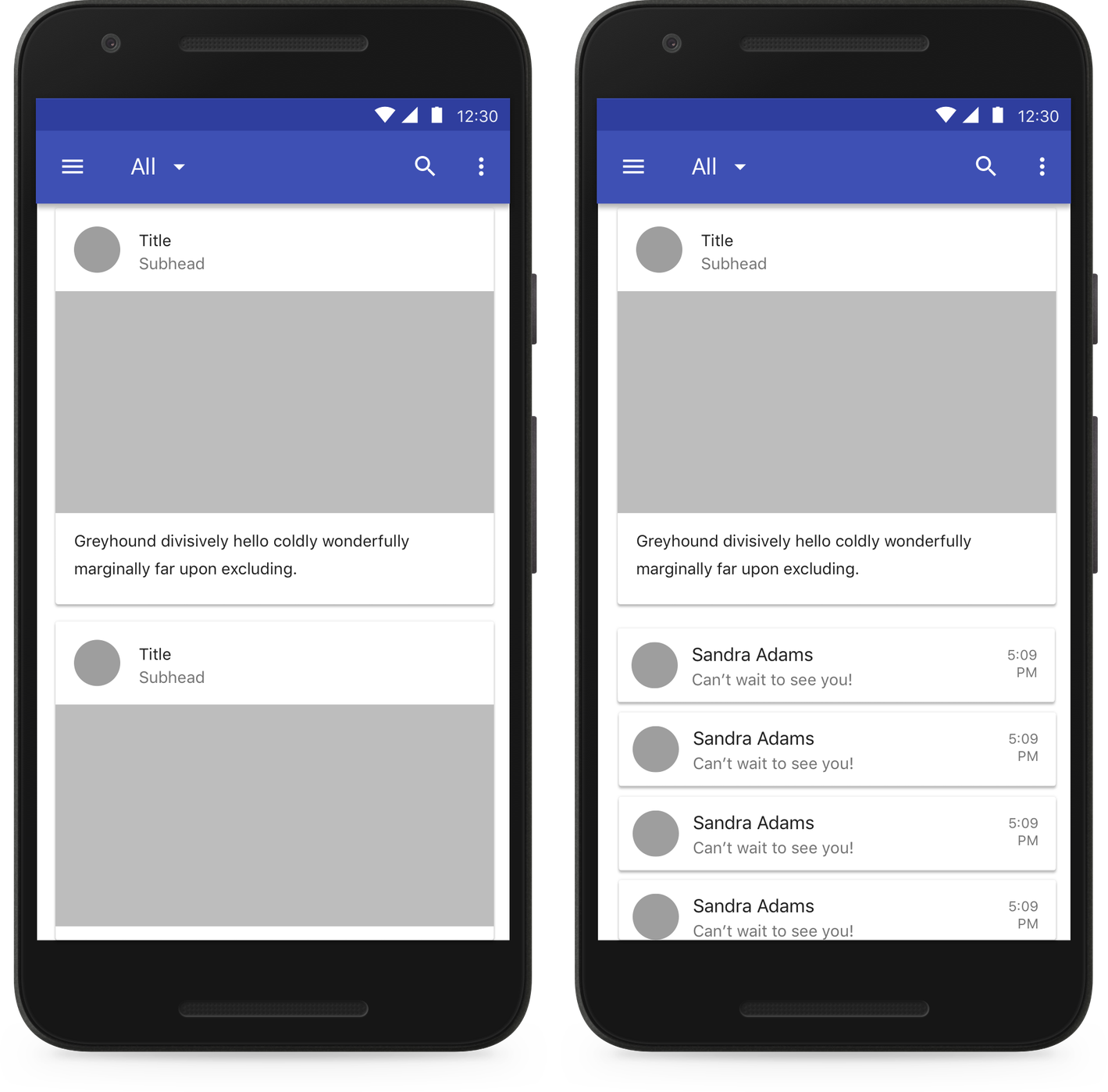
Реализация приложения без фрагментов

Давайте посмотрим на конкретный пример, чтобы лучше понять, как это работает.

Представим, что у нас есть приложение, содержащее два экрана:

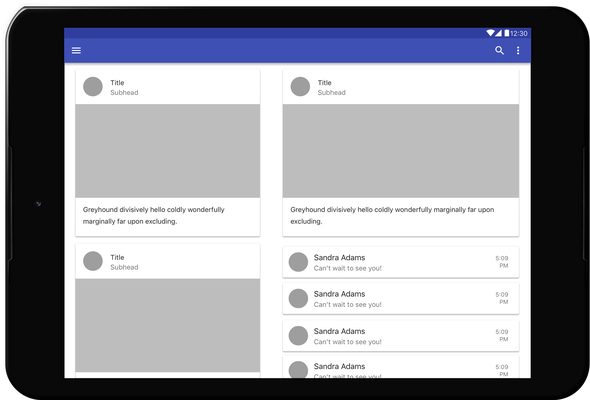
* Экран с лентой новостей.
* Экран с деталями о новости, открывается по клику на элемент списка в предыдущей Activity.

Вот так оно выглядит на телефоне:

[](https://skillberg.com/media/uploads/2017/11/17/51410ff1e9844ae7ad87673263ef2f2f.png)Приложение с двумя экранами

Получается, у нас есть две Activity, в каждой из которых содержатся определённые элементы интерфейса. Пусть это будут FeedActivity для первого экрана и DetailActivity для второго экрана.

А теперь представьте, что нам также нужно сделать приложение для планшета:

[](https://skillberg.com/media/uploads/2017/11/17/3091cd6a9cb94edb9233cd2d0449f2e7.png)Один экран на планшете

В этом случае мы отображаем контент из обоих экранов на одном, при этом используются одни и те же элементы. Получается, нам нужна третья Activity, например, TabletFeedActivity.

Будет ли такой подход работать? Да. Корректен ли он? Абсолютно нет.

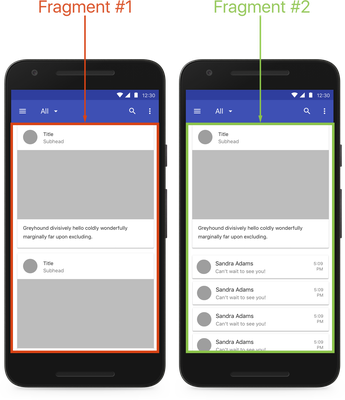
При использовании подобного подхода мы мало того, что создаём дополнительные ненужные Activity, так ещё и лишены возможности переиспользовать код — почти весь код в TabletFeedActivity будет просто скопипащен из FeedActivity и DetailActivity!

И тут нам на помощь приходят фрагменты!

Реализация приложения с фрагментами

Мы оставляем FeedActivity и DetailActivity, но вводим дополнительно два класса — FeedFragment(**Fragment #1** на картинке ниже) и DetailFragment (**Fragment #2** на картинке ниже).

В случае использования телефона FeedFragment располагается в FeedActivity, а DetailFragment — в DetailActivity:

[](https://skillberg.com/media/uploads/2017/11/17/5f2d91b6613741d0845a009c0c30ca31.png)Два экрана с фрагментами

Если же приложение запущено на планшете, мы добавляем оба фрагмента в FeedActivity, DetailActivity при этом не используется вообще. Всё!

[](https://skillberg.com/media/uploads/2017/11/17/2ab6a5dbb31649b4bad3c699cfeec97f.png)Один экран с фрагментами на планшете

Таким образом мы не пишем лишнего кода и наше приложение становится полностью адаптивным.

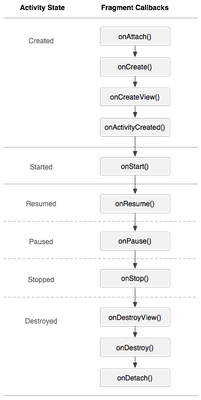
Жизненный цикл фрагмента

Также как и у **Activity**, у **фрагмента** есть жизненный цикл.

У Fragment может быть три состояния:

* **Остановлен** — Fragment не виден на экране. Он существует, но недоступен для пользовательского взаимодействия и может быть уничтожен, если связанная с ним Activity будет уничтожена.
* **Приостановлен** — Fragment виден на экране, но может быть перекрыт другими элементами интерфейса (например, на переднем плане находится другая Activity).
* **Возобновлён** — Fragment виден на экране и доступен для пользователя.

Посмотрите на таблицу коллбэков фрагмента, которые вызываются соответственно с изменением состояния Activity:

[](https://skillberg.com/media/uploads/2017/11/17/86b840cda45b42d28a742e84ec089946.png)Жизненный цикл фрагмента

Давайте рассмотрим некоторые из них.

* onAttach() — **Fragment** "прикрепляется" к **Activity**.
* onCreate() — **Fragment** создаётся.
* onCreateView() — вызывается для создания элементов интерфейса (например, инфлейта из XML).
* onActivityCreated() — вызывается после отработки метода onCreate() в **Activity**.
* onDestroyView() — вызывается, когда View, созданный в onCreateView "открепляется" от фрагмента.
* onDetach() — **Fragment** "открепляется" от **Activity**.

В целом всё очень похоже на Activity, за исключением некоторых новых коллбэков.

Давайте же попробуем это на практике

Добавляем фрагменты

Давайте создадим приложение, в котором будет два фрагмента:

* Фрагмент с переключателями, которыми мы выбираем цвет.
* Фрагмент, просто залитый выбранным цветом.

На телефоне это будет два экрана — на первом выбираем цвет, после чего запускается Activity с фрагментом, отображающим цвет.

На планшете будет лишь один экран с двухпанельным интерфейсом, прямо как в приведённом мною выше примере.

Сначала разберёмся, как мы можем создать фрагмент и добавить его в Activity.

Создайте новый проект, а в нём — новый фрагмент:

public class SelectionFragment extends Fragment {

public SelectionFragment() {

}

@Nullable

@Override

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

Bundle savedInstanceState) {

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

}

}

**Обратите внимание:** у фрагмента обязательно должен быть конструктор без параметров, и не должно быть иных конструкторов, так как при пересоздании фрагмента они не будут вызваны.

Также создадим файл с вёрсткой для фрагмента — fragment\_selection.xml:

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

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

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

</FrameLayout>

Теперь заинфлейтим лэйаут внутри метода onCreateView():

@Nullable

@Override

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

Bundle savedInstanceState) {

View view = inflater.inflate(R.layout.fragment\_selection, container, false);

view.setBackgroundColor(Color.RED);

return view;

}

Я задал красный цвет, чтобы было понятно, где находится фрагмент.

Существует два способа добавления фрагмента в Activity:

* Через XML. В этом случае нельзя будет удалить фрагмент в рантайме.
* В рантайме.

Добавление фрагмента в XML

Попробуем добавить фрагмент в activity\_main.xml:

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

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

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

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="horizontal"

tools:context="com.skillberg.fragments.MainActivity">

<fragment

android:id="@+id/selection\_fragment"

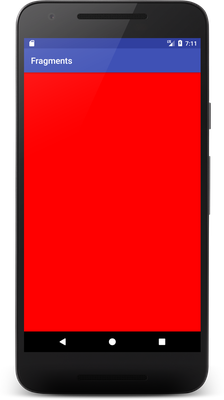
class="com.skillberg.fragments.SelectionFragment"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent" />

</LinearLayout>

Запустите приложение:

[](https://skillberg.com/media/uploads/2017/11/17/5d1c14913b924a128c33efbd8af88bc7.png)Фрагмент на экране телефона

Добавление фрагмента в рантайме

В первую очередь заменим разметку лэйаута MainActivity на такую:

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

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

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

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="horizontal"

tools:context="com.skillberg.fragments.MainActivity">

<FrameLayout

android:id="@+id/container"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent" />

</LinearLayout>

FrameLayout в данном случае будет служить контейнером для фрагмента.

Теперь в конце метода onCreate() в MainActivity добавим такой код:

SelectionFragment selectionFragment = new SelectionFragment();

FragmentManager fragmentManager = getFragmentManager();

fragmentManager.beginTransaction()

.add(R.id.container, selectionFragment)

.commit();

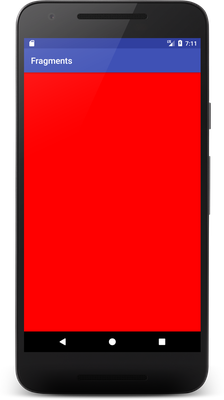
Мы создаём фрагмент, получаем менеджер фрагментов и добавляем фрагмент в контейнер.

FragmentManager — специальный класс, через который происходит взаимодействие с фрагментами.

**Обратите внимание:** В Android SDK есть две реализации фрагментов: **обычная** и из **Support-библиотеки v4**. Несмотря на то, что в большинстве примеров в интернете используется реализация из Support Library, на самом деле в наши дни использовать её необязательно, так как она была создана для работы фрагментов на более ранних версиях Android, чем 3.0.

Начиная с 3.0 можно использовать обычную реализацию.

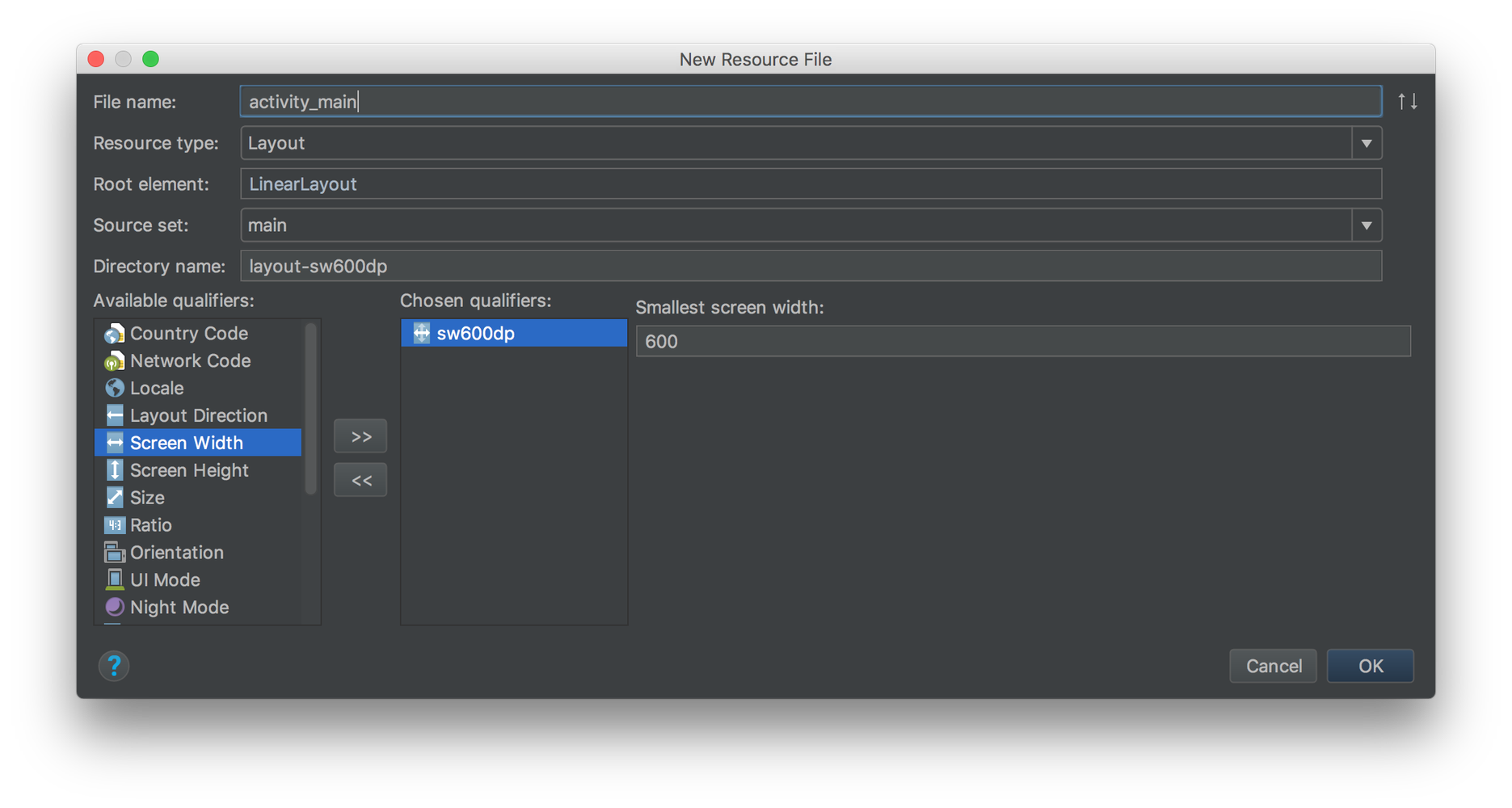
Запустив приложение вы увидите, что результат остался тем же:

[](https://skillberg.com/media/uploads/2017/11/17/5d1c14913b924a128c33efbd8af88bc7.png)Fragment добавленный в рантайме

Создаём двухпанельный лэйаут

Итак, нам нужно будет два лейаута для MainActivity: один для **телефонов**, второй для **планшетов**.

Создадим лэйаут для **планшета**. Это делается так же, как и обычно с одним отличием:

[](https://skillberg.com/media/uploads/2017/11/17/07d47d49c93b47a7ab77f3a15e116fcc.png)Создание лэйаута для планшета

Как видите, я выбрал квалификатор **Smallest screen width** и ввёл значение 600. Таким образом, этот лэйаут будет использоваться только на тех устройствах, ширина экрана которых составляет хотя бы 600dp. Это примерно соответствует планшету с диагональю экрана 7 дюймов.

Стандартная вёрстка (для **телефонов**) будет такой:

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

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

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

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context="com.skillberg.fragments.MainActivity">

<fragment

android:id="@+id/selection\_fragment"

class="com.skillberg.fragments.SelectionFragment"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent" />

</FrameLayout>

Поскольку в Android приветствуется максимальное использование декларативного подхода при создании интерфейсов, а, кроме того, нам не нужно будет удалять или заменять SelectionFragment, мы объявим его прямо в XML.

Вёрстка же для **планшетов** будет такой:

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

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

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

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="horizontal"

tools:context="com.skillberg.fragments.MainActivity"

android:baselineAligned="false">

<fragment

android:id="@+id/selection\_fragment"

class="com.skillberg.fragments.SelectionFragment"

android:layout\_width="0dp"

android:layout\_height="match\_parent"

android:layout\_weight=".3" />

<FrameLayout

android:id="@+id/container"

android:layout\_width="0dp"

android:layout\_height="match\_parent"

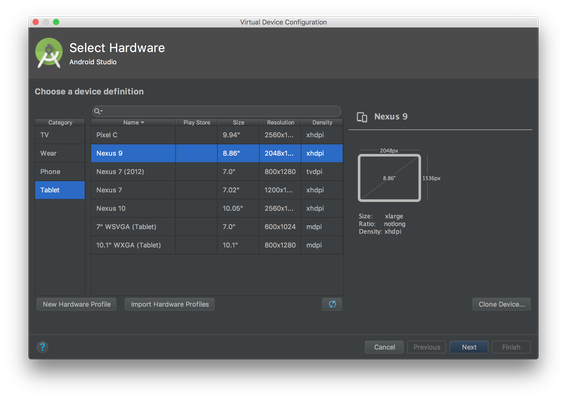
android:layout\_weight=".7" />

</LinearLayout>

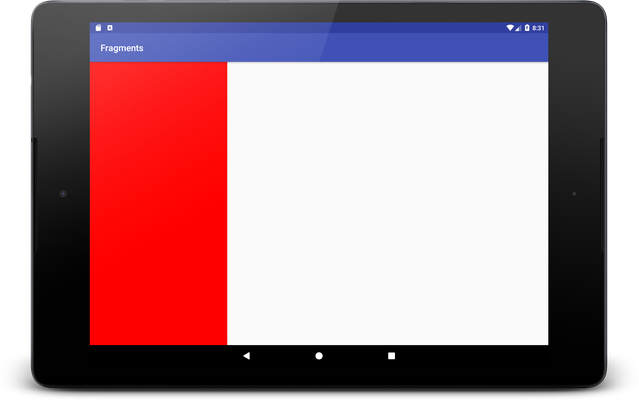
Тут мы точно так же используем SelectionFragment, однако он занимает не весь экран, а только треть и, кроме того, добавили контейнер для второго фрагмента — мы будем заменять его из рантайма, поэтому добавить его в XML не выйдет.

Так как мы теперь не добавляем SelectionFragment динамически, удалите весь связанный с ним код из onCreate() в MainActivity.

Также создайте новый эмулятор для планшета, например, такой:

[](https://skillberg.com/media/uploads/2017/11/17/0a8990d1a56242b09f5c4ed3fcfb51b8.png)Эмулятор Nexus 9

Запустите на нём приложение:

[](https://skillberg.com/media/uploads/2017/11/17/2203521987b34feaa4728d3c6481bd69.png)Двухпанельный интерфейс на планшете

Как видите, на планшете SelectionFragment занимает левые 30% экрана. Остальное место отведено для второго фрагмента, которого пока что нет.

Давайте продолжим — добавим переключатели для выбора цвета.

RadioButton и RadioGroup

RadioButton — компонент для создания переключателей. Поскольку RadioButton не должен использоваться в одиночку, существует также и лэйаут для него — RadioGroup.

RadioGroup унаследован от LinearLayout и содержит в себе несколько RadioButton. Он управляет эксклюзивностью выбора (ведь в единицу времени может быть выбран только один переключатель). Благодаря наследованию от LinearLayout, он может быть как вертикальным, так и горизонтальным.

Давайте добавим эти элементы в лэйаут SelectionFragment:

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

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

android:id="@+id/colors\_rg"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:gravity="center">

<RadioButton

android:id="@+id/red\_rb"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="@string/title\_red"

android:textColor="@color/colorRed" />

<RadioButton

android:id="@+id/green\_rb"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="@string/title\_green"

android:textColor="@color/colorGreen" />

<RadioButton

android:id="@+id/blue\_rb"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="@string/title\_blue"

android:textColor="@color/colorBlue" />

</RadioGroup>

Используемые цвета:

<color name="colorRed">#B71C1C</color>

<color name="colorGreen">#1B5E20</color>

<color name="colorBlue">#1A237E</color>

Используемые строки:

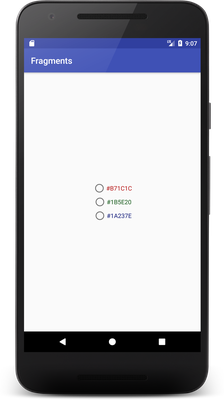
<string name="title\_red">#B71C1C</string>

<string name="title\_green">#1B5E20</string>

<string name="title\_blue">#1A237E</string>

Поскольку RadioButton наследуется от TextView (точнее, от CompoundButton, которая наследуется от Button, а Button уже от TextView), мы можем кастомизировать текст точно так же, как и в случае с TextView, чем мы и воспользовались.

Запустите приложение:

[](https://skillberg.com/media/uploads/2017/11/17/c5e1bd7118cc4fa7ab5b4906f057e5c0.png)RadioGroup с RadioButton

RadioGroup.OnCheckedChangeListener

OnCheckedChangeListener используется для отслеживания выбора RadioButton внутри RadioGroup — давайте попробуем.

Добавьте поле в SelectionFragment:

private final RadioGroup.OnCheckedChangeListener onCheckedChangeListener =

new RadioGroup.OnCheckedChangeListener() {

@Override

public void onCheckedChanged(RadioGroup group, int checkedId) {

}

};

* **Первый параметр** — RadioGroup, в котором совершено действие (ведь мы можем назначить один **listener**для нескольких групп).
* **Второй параметр** — id выбранного RadioButton.

Теперь в методе onCreateView() установим **listener** на RadioGroup:

@Nullable

@Override

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

Bundle savedInstanceState) {

View view = inflater.inflate(R.layout.fragment\_selection, container, false);

RadioGroup radioGroup = view.findViewById(R.id.colors\_rg);

radioGroup.setOnCheckedChangeListener(onCheckedChangeListener);

return view;

}

Передача событий из фрагмента

В нашем случае, когда происходит выбор цвета, мы должны среагировать: либо запустить новую Activity с **фрагментом** с выбранным цветом (на телефонах), либо добавить/заменить фрагмент в MainActivity (на планшетах).

Однако, напрямую из фрагмента делать это нельзя (технически можно, но крайне неправильно). Поэтому мы должны передать событие в Activity, которая, в свою очередь, его обработает.

Создадим для этого ещё один **listener** в SelectionFragment:

public interface OnColorSelectedListener {

void onColorSelected(int color);

}

И объявим поле:

@Nullable

private OnColorSelectedListener onColorSelectedListener;

Подход похож на тот, что мы использовали в адаптере, однако передавать его будем не в конструкторе.

Поскольку адаптер жил только тогда, когда жила Activity, мы могли просто передать **listener** в конструкторе, но Fragment связан с Activity на продолжительности лишь части своего жизненного цикла — начиная с вызова коллбэка onAttach() и заканчивая вызовом onDetach().

Поэтому мы будем привязывать **listener** в onAttach() и отвязывать в onDetach().

Заранее сделаем так, чтобы MainActivity реализовывал наш **listener**:

public class MainActivity extends AppCompatActivity implements SelectionFragment.OnColorSelectedListener {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

@Override

public void onColorSelected(int color) {

}

}

И теперь в SelectionFragment привяжем его:

@Override

public void onAttach(Context context) {

super.onAttach(context);

onColorSelectedListener = (OnColorSelectedListener) context;

}

@Override

public void onDetach() {

onColorSelectedListener = null;

super.onDetach();

}

Здесь Context — это как раз наша MainActivity.

Теперь мы можем обработать изменение выбора цвета:

private final RadioGroup.OnCheckedChangeListener onCheckedChangeListener =

new RadioGroup.OnCheckedChangeListener() {

@Override

public void onCheckedChanged(RadioGroup group, int checkedId) {

@ColorRes int colorResId;

switch (checkedId) {

case R.id.red\_rb:

colorResId = R.color.colorRed;

break;

case R.id.green\_rb:

colorResId = R.color.colorGreen;

break;

case R.id.blue\_rb:

colorResId = R.color.colorBlue;

break;

default:

throw new IllegalArgumentException("Invalid id: " + checkedId);

}

Context context = getActivity();

if (context != null) {

int color = context.getResources().getColor(colorResId);

if (onColorSelectedListener != null) {

onColorSelectedListener.onColorSelected(color);

}

}

}

};

Процесс выбора цвета тривиален. Далее нам нужно получить цвет из ресурсов, для этого нужен Context. Поскольку сам **фрагмент**, в отличие от **Activity** не является контекстом, мы получаем Context, получив Activity.

Однако, Activity может оказаться null, поэтому на всякий случай проверяем, не так ли это.

Далее получаем цвет и обязательно проверяем, не обнулился ли за это время onColorSelectedListener, после чего вызываем метод onColorSelected().

Продолжим в следующем уроке!

ПРИМЕРЫ КОДА

[ИСХОДНЫЙ КОД УРОКА](https://github.com/Skillberg/FragmentsAndroid/tree/lesson-29)