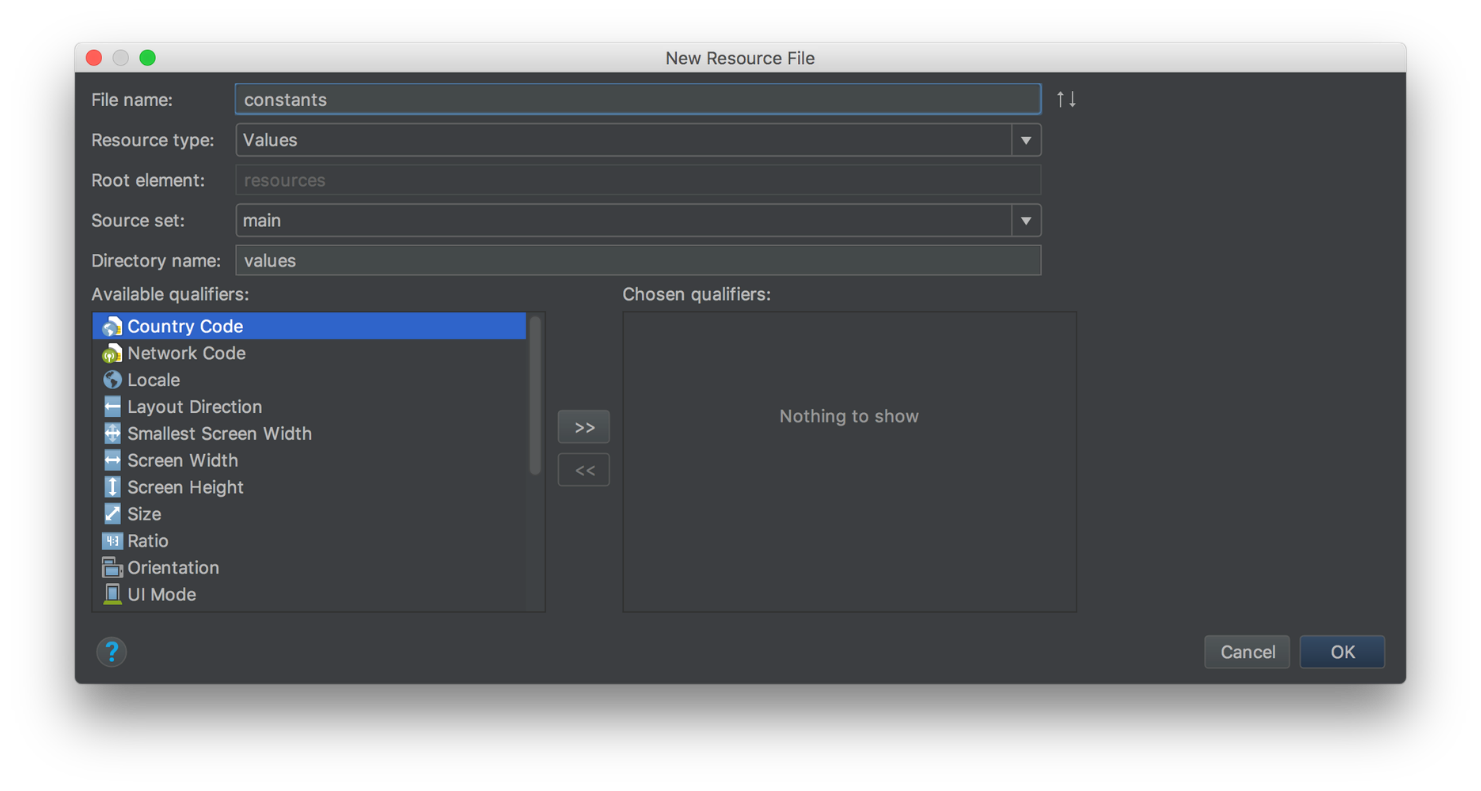
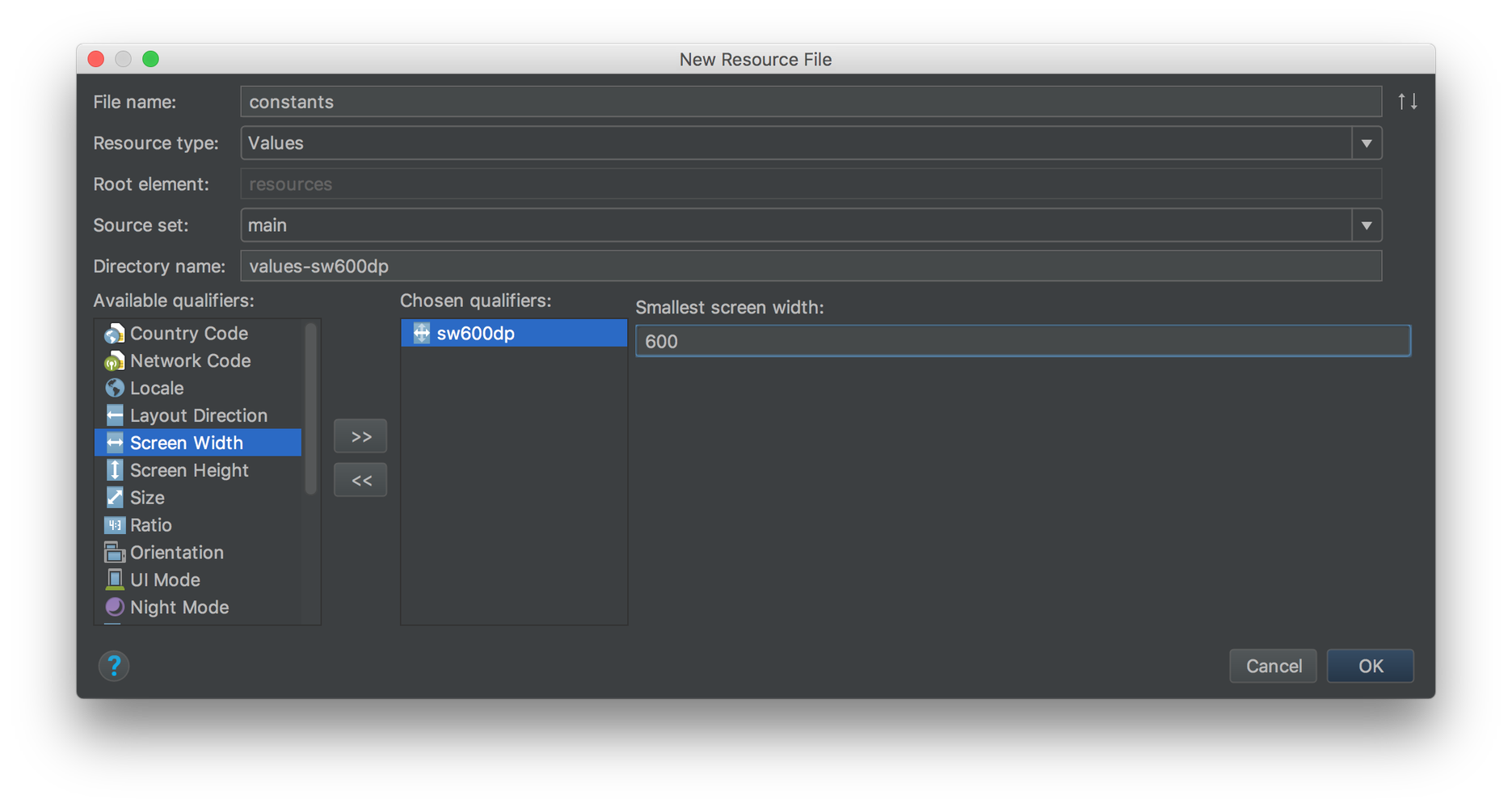
В этом уроке мы продолжим работу над нашим адаптивным приложением.

Идентификация типа устройства

Как вы помните, на разных устройствах мы выполняем разные действия — на телефонах запускаем новую Activity, а на планшетах добавляем фрагмент в MainActivity.

Как нам понять, на каком устройстве сейчас запущено приложение?

Самый простой способ — добавить boolean ресурс для двух конфигураций, как в случае с лэйаутами:

[](https://skillberg.com/media/uploads/2017/11/17/22072ebe9dbf4ccc8648bae50f48c5d3.png)[](https://skillberg.com/media/uploads/2017/11/17/f7046791db4f4c3fbfc633ed35b68029.png)

В первый мы добавляем такой контент:

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

<resources>

<bool name="isTablet">false</bool>

</resources>

А во второй — вот такой:

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

<resources>

<bool name="isTablet">true</bool>

</resources>

Теперь, если нам нужно узнать тип устройства — просто получаем это значение из ресурсов, и всё!

Фрагмент, отображающий цвет

Создадим второй фрагмент, который будет отображать выбранный цвет:

public class DetailFragment extends Fragment {

public DetailFragment() {

}

@Nullable

@Override

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

Bundle savedInstanceState) {

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

}

}

И лэйаут:

<?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" />

В onCreateView() просто заинфлейтим лэйаут:

@Nullable

@Override

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

Bundle savedInstanceState) {

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

}

Передача данных во фрагмент

Механизм передачи данных в Fragment похож на аналогичных механизм в Activity, разве что называется оно **Arguments**, а не **Extras**.

Есть и ещё одно отличие: поскольку мы создаём Fragment напрямую, а не используя механизм Intent, как в случае с Activity, непосредственная установка аргументов выглядит несколько иначе.

В нашем случае нам нужно передать цвет (int). Создадим константу для ключа в DetailFragment, поскольку **аргументы**, как и **Extras**, выглядят как пары ключ-значение:

private static final String ARG\_COLOR = "color";

Хорошим тоном является создание и использование специального статического метода newInstance(), который будет создавать фрагмент, а не создание напрямую:

public static DetailFragment newInstance(int color) {

DetailFragment fragment = new DetailFragment();

Bundle args = new Bundle();

args.putInt(ARG\_COLOR, color);

fragment.setArguments(args);

return fragment;

}

В случае, если передача аргументов не требуется, этот метод будет состоять лишь из одной строки — возвращать новый инстанс фрагмента.

После того, как иерархия View в фрагменте будет создана и готова к использованию, вызовется метод onViewCreated():

@Override

public void onViewCreated(View view, @Nullable Bundle savedInstanceState) {

super.onViewCreated(view, savedInstanceState);

}

Параметр view — корневой View в фрагменте, то есть тот, что мы вернули из метода onCreateView().

В этот момент мы уже можем получить переданные во фрагмент аргументы и как-то использовать их:

@Override

public void onViewCreated(View view, @Nullable Bundle savedInstanceState) {

super.onViewCreated(view, savedInstanceState);

Bundle args = getArguments();

if (args != null && args.containsKey(ARG\_COLOR)) {

int color = args.getInt(ARG\_COLOR);

view.setBackgroundColor(color);

}

}

Как видите, всё просто.

Создание интерфейса для телефона

В случае использования на телефоне, нам также понадобится дополнительная **Activity** — DetailActivity:

public class DetailActivity extends AppCompatActivity {

@Override

protected void onCreate(@Nullable Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_detail);

}

}

lang=xml

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

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

android:id="@+id/container"

android:layout\_width="match\_parent"

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

</FrameLayout>

**Не забудьте** добавить Activity в манифест!

Поскольку в этом случае мы не сможем передать цвет в DetailFragment напрямую из MainActivity, придётся передавать их в DetailActivity и оттуда уже в DetailFragment:

public class DetailActivity extends AppCompatActivity {

public static final String EXTRA\_COLOR = "color";

@Override

protected void onCreate(@Nullable Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_detail);

Bundle extras = getIntent().getExtras();

if (extras == null || !extras.containsKey(EXTRA\_COLOR)) {

finish();

return;

}

int color = extras.getInt(EXTRA\_COLOR);

DetailFragment detailFragment = DetailFragment.newInstance(color);

getFragmentManager()

.beginTransaction()

.add(R.id.container, detailFragment)

.commit();

}

}

Вернёмся к MainActivity. Добавим поле для определения типа устройства, и будем получать тип в onCreate().

После выбора цвета просто выполняем нужную нам операцию в зависимости от типа устройства (пока что поддерживаем только телефон):

public class MainActivity extends AppCompatActivity

implements SelectionFragment.OnColorSelectedListener {

private boolean isTablet;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

isTablet = getResources().getBoolean(R.bool.isTablet);

}

@Override

public void onColorSelected(int color) {

if (!isTablet) {

// Телефон

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

intent.putExtra(DetailActivity.EXTRA\_COLOR, color);

startActivity(intent);

} else {

// Планшет

}

}

}

Запустите приложение и попробуйте выбрать цвет:

[](https://skillberg.com/media/uploads/2017/11/17/b95b927fed9047ec81dff24df95d00b1.png)DetailActivity на телефоне

Создание интерфейса для планшета

Осталось добавить поддержку планшетов:

@Override

public void onColorSelected(int color) {

if (!isTablet) {

// Телефон

// ...

} else {

// Планшет

DetailFragment detailFragment = DetailFragment.newInstance(color);

FragmentManager fragmentManager = getFragmentManager();

FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();

if (fragmentManager.findFragmentByTag("detail") != null) {

// Если фрагмент уже добавлен — заменяем

fragmentTransaction.replace(R.id.container, detailFragment, "detail");

fragmentTransaction.addToBackStack(null);

} else {

// Иначе добавляем

fragmentTransaction.add(R.id.container, detailFragment, "detail");

}

fragmentTransaction.commit();

}

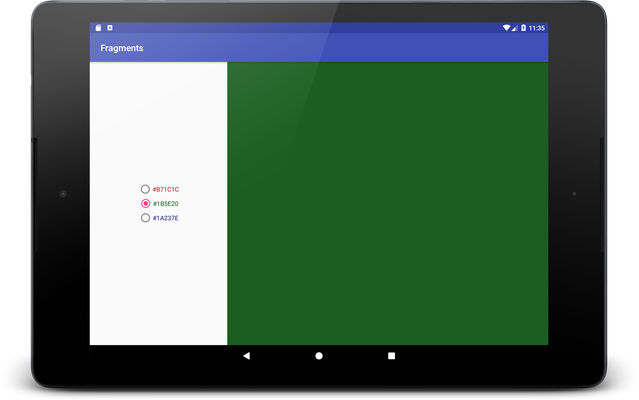
}

Как видите, мы добавляем **фрагмент**, если его ещё нет, и заменяем на новый, если он есть.

Понять, есть ли **фрагмент** можно используя поиск по тэгу, который мы задаём при добавлении или замене.

Вы также могли обратить внимание на метод addToBackStack(). Если мы вызываем этот метод для транзакции, то она добавляется в так называемый **Back Stack**, историю переходов. Если пользователь после этого нажмёт на кнопку **назад**, он вернётся к предыдущей транзакции.

Запустите приложение:

[](https://skillberg.com/media/uploads/2017/11/17/f8e8962113114604bde55abab04d1f37.png)Планшетный двухпанельный интерфейс

Попробуйте повыбирать различные цвета, а потом понажимать на кнопку **назад**. Работает :)

Есть, правда, один нюанс: при возврате назад не меняются радиокнопки. Давайте это исправим.

OnBackStackChangedListener

Самым простым способом будет добавление OnBackStackChangedListener, который позволяет отслеживать изменение **Back Stack** — он сработает при добавлении/замене фрагментов и при переходе назад.

Стратегия такая:

1. В SelectionFragment создаём поле, которое будет хранить набор пар **"цвет"** - **"id RadioButton"**. С помощью этого поля мы сможем легко найти нужный RadioButton по цвету.
2. В DetailFragment создаём поле, в котором храним текущий цвет.
3. В MainActivity создаём OnBackStackChangedListener, в котором обрабатываем изменение **Back Stack**.
4. При срабатывании OnBackStackChangedListener получаем текущий цвет из DetailFragment и передаём его в SelectionFragment, который, исходя из цвета отметит необходимую кнопку.

Итак, начнём с SelectionFragment:

public class SelectionFragment extends Fragment {

private RadioGroup radioGroup;

private final SparseIntArray colorToButtonIdMap = new SparseIntArray();

**SparseArray** — альтернатива Map, созданная разработчиками Android. Отличается тем, что в качестве ключей выступают примитивные типы (например, int, как в нашем случае), а не объекты. Такая реализация требует меньше памяти, но чуть медленнее работает (что несущественнее).

Если в качестве ключа используется примитивный тип, рекомендуется использовать именно SparseArray.

Также вынесем RadioGroup в отдельное поле, поскольку нам потребуется доступ к нему.

@Nullable

@Override

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

Bundle savedInstanceState) {

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

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

radioGroup.setOnCheckedChangeListener(onCheckedChangeListener);

for (int i = 0; i < radioGroup.getChildCount(); i++) {

RadioButton radioButton = (RadioButton) radioGroup.getChildAt(i);

int color = radioButton.getCurrentTextColor();

int id = radioButton.getId();

colorToButtonIdMap.append(color, id);

}

return view;

}

Здесь мы перенесли RadioGroup в созданное ранее поле, а также прошлись по каждой RadioButton и заполнили таким образом colorToButtonIdMap.

Поскольку RadioGroup — это ViewGroup, мы можем получить количество дочерних View, используя метод getChildCount(), а потом получить каждый дочерний View в цикле. То, что происходит далее, я думаю, очевидно :)

Теперь добавим метод, который будет выбирать нужную радиокнопку, основываясь на цвете.

public synchronized void selectButtonWithColor(int color) {

// Находим нужный id

int id = colorToButtonIdMap.get(color, -1);

if (id == -1) {

return;

}

// Смотрим, какая кнопка выбрана сейчас

int currentlyCheckedId = radioGroup.getCheckedRadioButtonId();

// Если они равны — второй раз выбирать её не нужно

if (currentlyCheckedId == id) {

return;

}

// Снимаем listener, потому что иначе он сработает

radioGroup.setOnCheckedChangeListener(null);

// Отмечаем нужную кнопку

radioGroup.check(id);

// Возвращаем listener

radioGroup.setOnCheckedChangeListener(onCheckedChangeListener);

}

Поясню манипуляции с onCheckedChangeListener: проблема в том, что в Android подобные слушатели срабатывают независимо от того, кто "выбрал" радиокнопку (и любого наследника CompoundButton в принципе).

Нам же совершенно не нужно, чтобы это происходило, поэтому мы обнуляем **listener**, прежде чем "отметить" радиокнопку, после чего возвращаем его назад.

Теперь DetailFragment:

public class DetailFragment extends Fragment {

private static final String ARG\_COLOR = "color";

private int color = 0;

Добавим **getter**:

public int getColor() {

return color;

}

И заменим строчку

int color = args.getInt(ARG\_COLOR);

на

color = args.getInt(ARG\_COLOR);

И, наконец, MainActivity. Добавьте такой код в конец onCreate():

final FragmentManager fragmentManager = getFragmentManager();

final SelectionFragment selectionFragment =

(SelectionFragment) fragmentManager.findFragmentById(R.id.selection\_fragment);

if (isTablet) {

fragmentManager.addOnBackStackChangedListener(new FragmentManager

.OnBackStackChangedListener() {

@Override

public void onBackStackChanged() {

DetailFragment detailFragment = (DetailFragment)

fragmentManager.findFragmentByTag("detail");

selectionFragment.selectButtonWithColor(detailFragment.getColor());

}

});

}

Запустите приложение, выберите цвет несколько раз, а затем вернитесь назад и увидите, что всё работает!

ПРИМЕРЫ КОДА

[ИСХОДНЫЙ КОД УРОКА](https://github.com/Skillberg/FragmentsAndroid/tree/lesson-30)