Дополнительный материал 2 к практической работе 8+.

Методические указания

Fragments. Динамическая работа

В этом материале- динамически работаем с фрагментами

Размещать статические фрагменты мы уже умеем. Но, ясно дело, что гораздо интереснее работать с ними динамически. Система позволяет нам добавлять, удалять и заменять фрагменты друг другом. При этом мы можем сохранять все эти манипуляции в BackStack и кнопкой Назад отменять. В общем, все удобно и красиво.

Создадим простое приложение с двумя фрагментами, которое будет уметь:

- добавлять первый фрагмент
- удалять первый фрагмент
- заменять первый фрагмент вторым фрагментом
- переключать режим сохранения в BackStack операций с фрагментами

Создадим проект с такими данными:

Project name: P1051_FragmentDynamic
Build Target: Android 4.1
Application name: FragmentDynamic
Package name: ru.startandroid.develop.p1051fragmentdynamic
Create Activity: MainActivity

В **strings.xml** добавим строки:

```
<string name="frag1_text">Fragment 1</string>
<string name="frag2_text">Fragment 2</string>
<string name="add">Add</string>
<string name="remove">Remove</string>
<string name="replace">Replace</string>
<string name="stack">add to Back Stack</string>
```

Создаем фрагменты. Как мы помним из прошлого урока, для этого нам нужны будут layout-файлы и классы, наследующие android.app.Fragment

fragment1.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="#77ff0000"
    android:orientation="vertical">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/frag1_text">
    </TextView>
</LinearLayout>
```

fragment2.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="#7700ff00"
    android:orientation="vertical">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/frag2_text">
    </TextView>
</LinearLayout>
```

Fragment1.java:

```
package ru.startandroid.develop.p1051fragmentdynamic;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class Fragment1 extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
      return inflater.inflate(R.layout.fragment1, null);
    }
}
```


Fragment2.java:

```
package ru.startandroid.develop.p1051fragmentdynamic;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
```

```
import android.view.ViewGroup;

public class Fragment2 extends Fragment {

  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
      Bundle savedInstanceState) {
    return inflater.inflate(R.layout.fragment2, null);
  }
}
```

Все почти аналогично прошлому уроку, только убрали вызовы кучи lifecycle методов с логами.

Рисуем основное 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:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical">
        <Button
            android:id="@+id/btnAdd"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onClick"
            android:text="@string/add">
        </Button>
        <Button
            android:id="@+id/btnRemove"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onClick"
            android:text="@string/remove">
        </Button>
        <Button
            android:id="@+id/btnReplace"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onClick"
            android:text="@string/replace">
        </Button>
        <CheckBox
            android:id="@+id/chbStack"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/stack">
        </CheckBox>
    </LinearLayout>
    <FrameLayout
```

```
        android:id="@+id/frgmCont"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    </FrameLayout>
</LinearLayout>
```

Три кнопки для добавления, удаления и замены фрагментов. Чекбокс для включения использования BackStack. И FrameLayout – это контейнер, в котором будет происходить вся работа с фрагментами. Он должен быть типа ViewGroup. А элементы Fragment, которые мы использовали на прошлом уроке для размещения фрагментов, нам не нужны для динамической работы.

MainActivity.java:

```
package ru.startandroid.develop.p1051fragmentdynamic;

import android.app.Activity;
import android.app.FragmentTransaction;
import android.os.Bundle;
import android.view.View;
import android.widget.CheckBox;

public class MainActivity extends Activity {

  Fragment1 frag1;
  Fragment2 frag2;
  FragmentTransaction fTrans;
  CheckBox chbStack;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    frag1 = new Fragment1();
    frag2 = new Fragment2();

    chbStack = (CheckBox)findViewById(R.id.chbStack);
  }

  public void onClick(View v) {
    fTrans = getFragmentManager().beginTransaction();
    switch (v.getId()) {
    case R.id.btnAdd:
      fTrans.add(R.id.frgmCont, frag1);
      break;
    case R.id.btnRemove:
      fTrans.remove(frag1);
      break;
    case R.id.btnReplace:
      fTrans.replace(R.id.frgmCont, frag2);
    default:
      break;
    }
    if (chbStack.isChecked()) fTrans.addToBackStack(null);
    fTrans.commit();
  }
}
```

В **onCreate** создаем пару фрагментов и находим чекбокс.

В **onClick** мы получаем менеджер фрагментов с помощью метода getFragmentManager. Этот объект является основным для работы с фрагментами. Далее, чтобы добавить/удалить/заменить фрагмент, нам необходимо использовать транзакции. Они аналогичны транзакциям в БД, где мы открываем транзакцию, производим операции с БД, выполняем commit. Здесь мы открываем транзакцию, производим операции с фрагментами (добавляем, удаляем, заменяем), выполняем commit.

Итак, мы получили FragmentManager и открыли транзакцию методом beginTransaction. Далее определяем, какая кнопка была нажата:

если **Add**, то вызываем метод add, в который передаем id контейнера (тот самый FrameLayout из main.xml) и объект фрагмента. В итоге, в контейнер будет помещен Fragment1

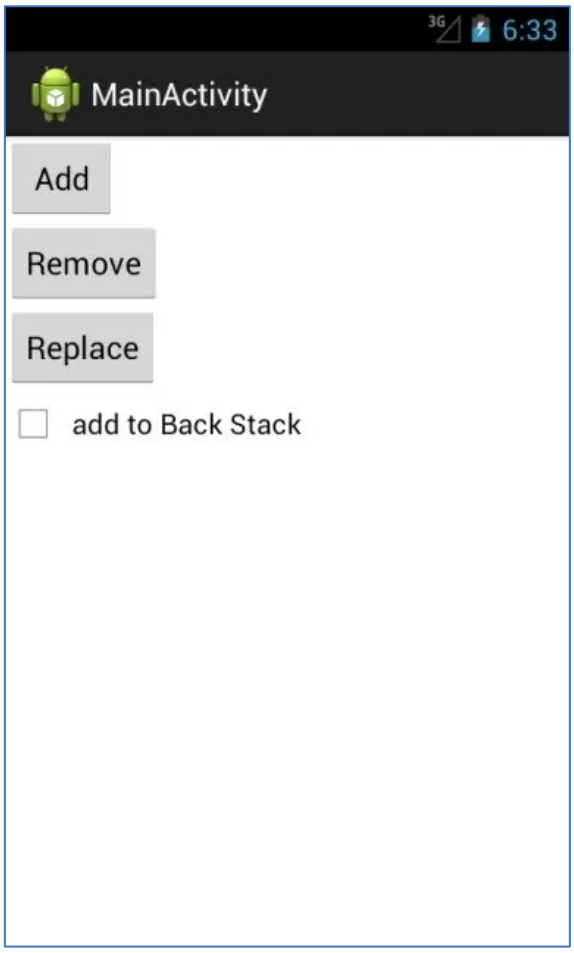
если **Remove**, то вызываем метод remove, в который передаем объект фрагмента, который хотим убрать. В итоге, фрагмент удалится с экрана.

если **Replace**, то вызываем метод replace, в который передаем id контейнера и объект фрагмента. В итоге, из контейнера удалится его текущий фрагмент (если он там есть) и добавится фрагмент, указанный нами.

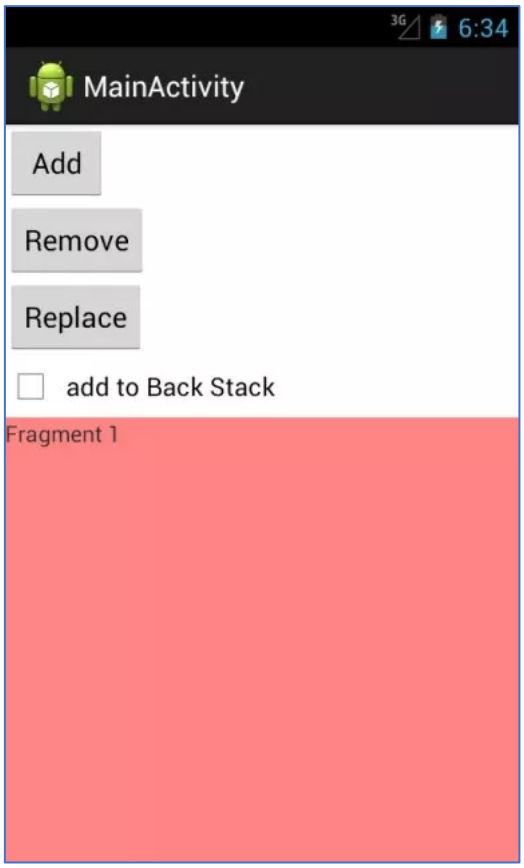
Далее проверяем чекбокс. Если он включен, то добавляем транзакцию в BackStack. Для этого используем метод addToBackStack. На вход можно подать строку-тэг. Я передаю null.

Ну и вызываем commit, транзакция завершена.

Давайте смотреть, что получилось. Все сохраняем, запускаем приложение.

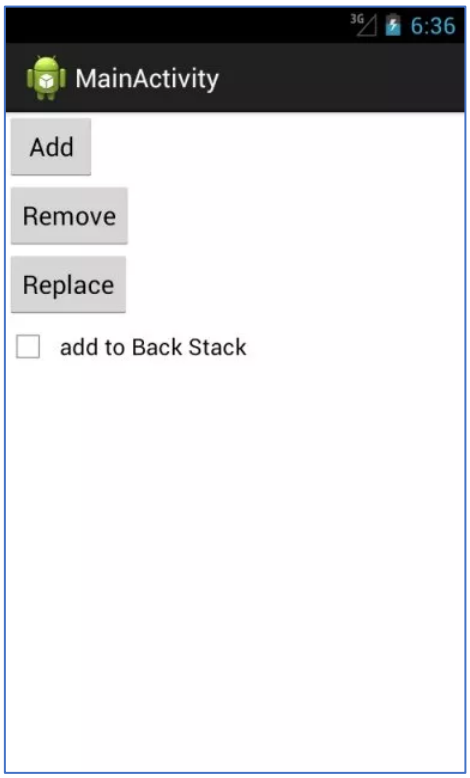

6:33
MainActivity
Add
Remove
Replace
add to Back Stack

Жмем **Add**


6:34
MainActivity
Add
Remove
Replace
add to Back Stack
Fragment 1

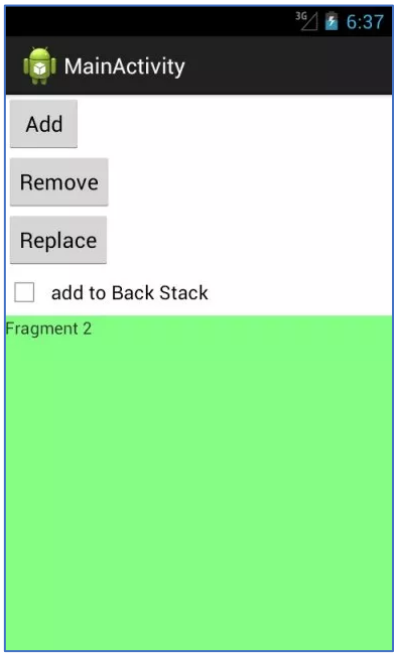
появился первый фрагмент.

Жмем **Remove**



фрагмент удалился.

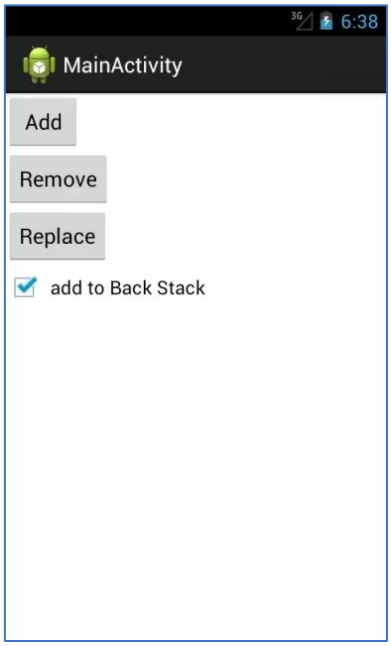
Еще раз добавим первый фрагмент – жмем **Add**. И жмем **Replace**



первый фрагмент заменился вторым.

Жмем кнопку **Назад**. Приложение закрылось, т.к. все эти операции с фрагментами не сохранялись в BackStack. Давайте используем эту возможность.

Снова запускаем приложение и включаем чекбокс **add to Back Stack**



Выполняем те же операции: **Add**, **Remove**, **Add**, **Replace**. У нас добавится первый фрагмент, удалится первый фрагмент, добавится первый фрагмент, заменится вторым. В итоге мы снова видим второй фрагмент. Теперь жмем несколько раз кнопку **Назад** и наблюдаем, как выполняются операции, обратные тем, что мы делали. Когда транзакции, сохраненные в стеке закончатся, кнопка Назад закроет приложение.

Т.е. все достаточно просто и понятно. Скажу еще про пару интересных моментов.

Я в этом примере выполнял всего одну операцию в каждой транзакции. Но, разумеется, их может быть больше.

Когда мы удаляем фрагмент и не добавляем транзакцию в BackStack, то фрагмент уничтожается. Если же транзакция добавляется в BackStack, то, при удалении, фрагмент не уничтожается (onDestroy не вызывается), а останавливается (onStop).

В качестве самостоятельной работы: попробуйте немного изменить приложение и добавлять в один контейнер сразу два фрагмента. Возможно, результат вас удивит)

Fragments. Взаимодействие с Activity

В этом материале:

- рассмотрим взаимодействие между Activity и ее фрагментами

После размещения фрагмента, хотелось бы начать с ним взаимодействовать. Т.е. размещать View-компоненты и работать с ними, обращаться к фрагментам из Activity и наоборот. Попробуем это реализовать.

Для чистоты эксперимента будем работать с двумя фрагментами: статическим и динамическим.

Создадим проект:

Project name: P1061_FragmentActivity

Build Target: Android 4.1

Application name: FragmentActivity

Package name: ru.startandroid.develop.p1061fragmentactivity

Create Activity: MainActivity

В strings.xml добавим строки:

<string name="frag1_text">Fragment 1</string>

<string name="frag2_text">Fragment 2</string>

<string name="log">Log</string>

<string name="find">Find</string>

Создаем layout и классы для двух фрагментов.

fragment1.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="#77ff0000"
    android:orientation="vertical">
    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/frag1_text">
    </TextView>
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/log">
    </Button>
</LinearLayout>
```

fragment2.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="#7700ff00"
    android:orientation="vertical">
    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/frag2_text">
    </TextView>
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/log">
    </Button>
</LinearLayout>
```

Fragment1.java:

```
package ru.startandroid.develop.p1061fragmentactivity;

import android.app.Fragment;
import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
```

```
import android.view.ViewGroup;
import android.widget.Button;

public class Fragment1 extends Fragment {

  final String LOG_TAG = "myLogs";

  public View onCreateView(LayoutInflater inflater, ViewGroup container,
      Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment1, null);

    Button button = (Button) v.findViewById(R.id.button);
    button.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        Log.d(LOG_TAG, "Button click in Fragment1");
      }
    });

    return v;
  }
}
```

У фрагмента нет привычного для нас метода findViewById для поиска компонентов с экрана. Поэтому вызываем этот метод для View, которое будет содержимым фрагмента. В методе onCreateView мы создаем View и сразу же находим в нем кнопку и ставим ей обработчик. Затем отдаем View системе.

Fragment2.java:

```
package ru.startandroid.develop.p1061fragmentactivity;

import android.app.Fragment;
import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;
import android.widget.Button;

public class Fragment2 extends Fragment {

  final String LOG_TAG = "myLogs";

  public View onCreateView(LayoutInflater inflater, ViewGroup container,
      Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment2, null);

    Button button = (Button) v.findViewById(R.id.button);
    button.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        Log.d(LOG_TAG, "Button click in Fragment2");
      }
    });

    return v;
  }
```

}

Все аналогично Fragment1.

Настраиваем основное Activity.

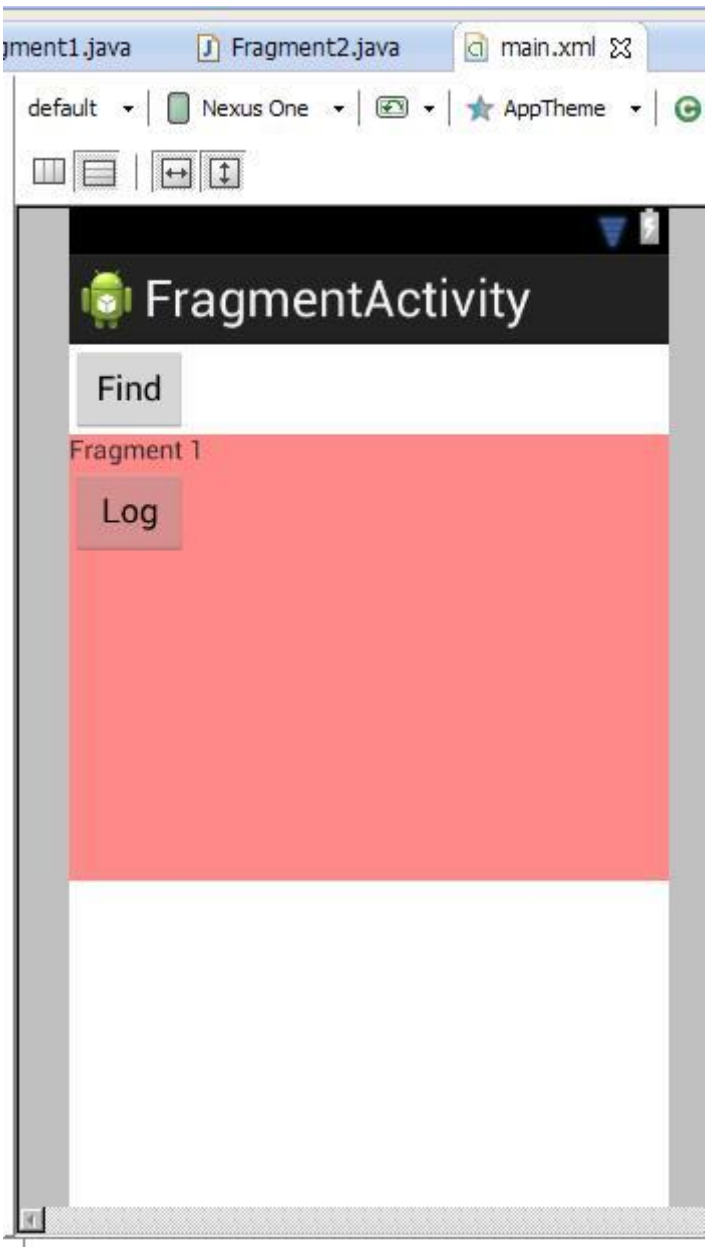
main.xml:

```
<?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:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <Button
        android:id="@+id/btnFind"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:text="@string/find">
    </Button>
    <fragment
        android:id="@+id/fragment1"
        android:name="ru.startandroid.develop.p1061fragmentactivity.Fragment1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
        android:layout_weight="1"
        tools:layout="@layout/fragment1">
    </fragment>
    <FrameLayout
        android:id="@+id/fragment2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1">
    </FrameLayout>
</LinearLayout>
```

Кнопка, компонент fragment, в который помещен Fragment1, и контейнер FrameLayout, в который потом поместим Fragment2.

Обратите внимание на атрибут tools:layout. В нем указан layout-файл, и мы можем на этапе разработки видеть, как будет выглядеть статический фрагмент, когда приложение будет запущено.



Для этого надо нажать правой кнопкой на компоненте fragment, и через пункт **Fragment Layout** указать нужный layout.

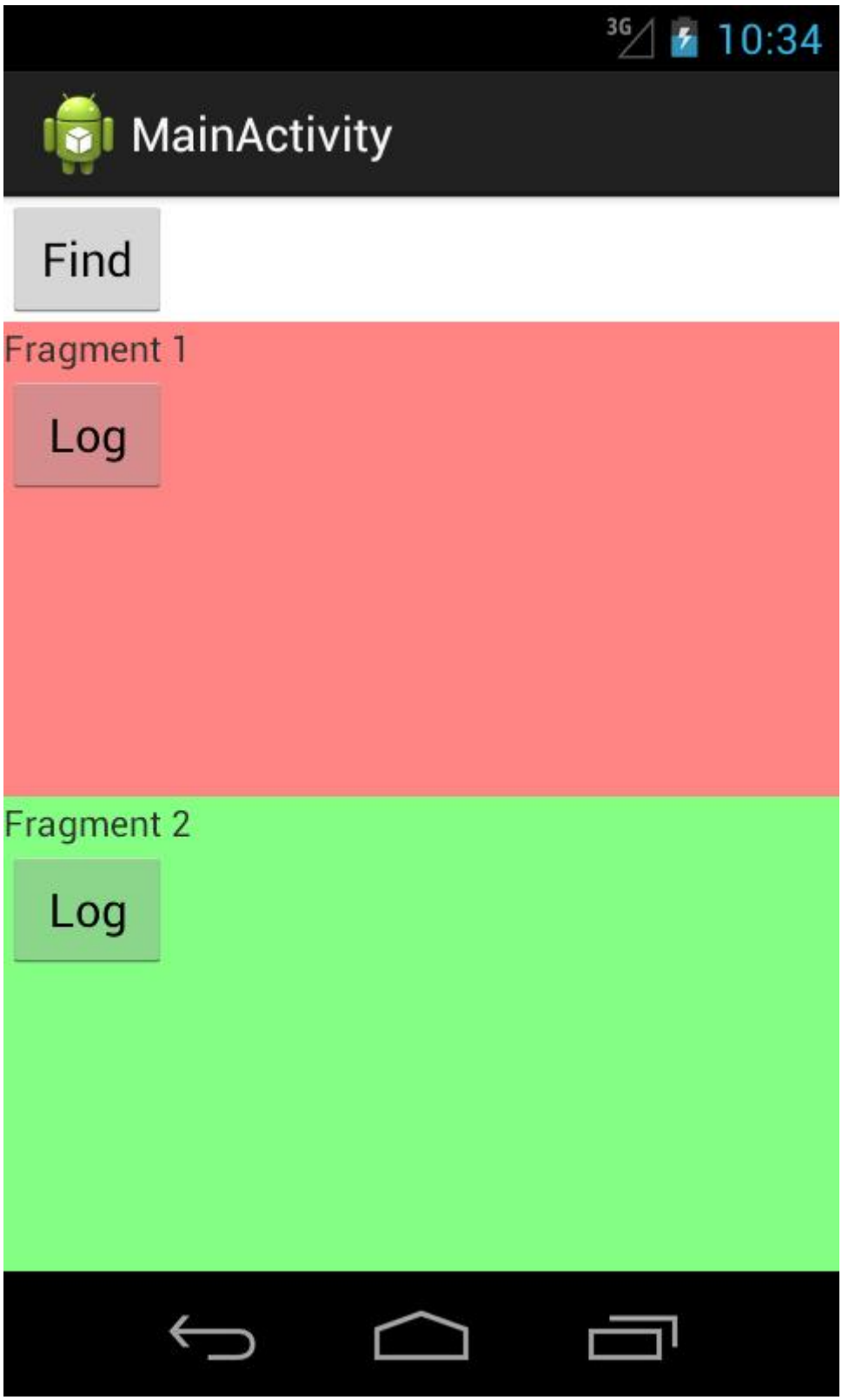
MainActivity.java:

```
package ru.startandroid.develop.p1061fragmentactivity;


import android.app.Activity;
import android.app.Fragment;
```

```
import android.app.FragmentTransaction;
import android.os.Bundle;

public class MainActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        Fragment frag2 = new Fragment2();
        FragmentTransaction ft = getFragmentManager().beginTransaction();
        ft.add(R.id.fragment2, frag2);
        ft.commit();
    }
}
```

Здесь мы просто добавляем Fragment2 в контейнер.

Все сохраняем, запускаем приложение.



Жмем кнопку Log в первом фрагменте и смотрим лог:

Button click in Fragment1

Жмем Log во втором фрагменте:

Button click in Fragment2

Все ок. Компоненты в фрагментах нашлись и обработчики среагировали на нажатия.

Атрибут onClick, который мы привыкли использовать для кнопки, здесь не прокатит. Указанный в этом атрибуте метод, будет вызван в Activity, а не в фрагменте.

Доступ к фрагменту из Activity

Разберемся, как получить доступ к фрагменту из Activity. Для этого у FragmentManager есть метод findFragmentById, который на вход принимает id компонента fragment (если фрагмент статический) или id контейнера (если динамический).

У нас в main.xml есть кнопка btnFind, вызывающая метод onClick при нажатии. Дорисуем в MainActivity.java метод onClick:

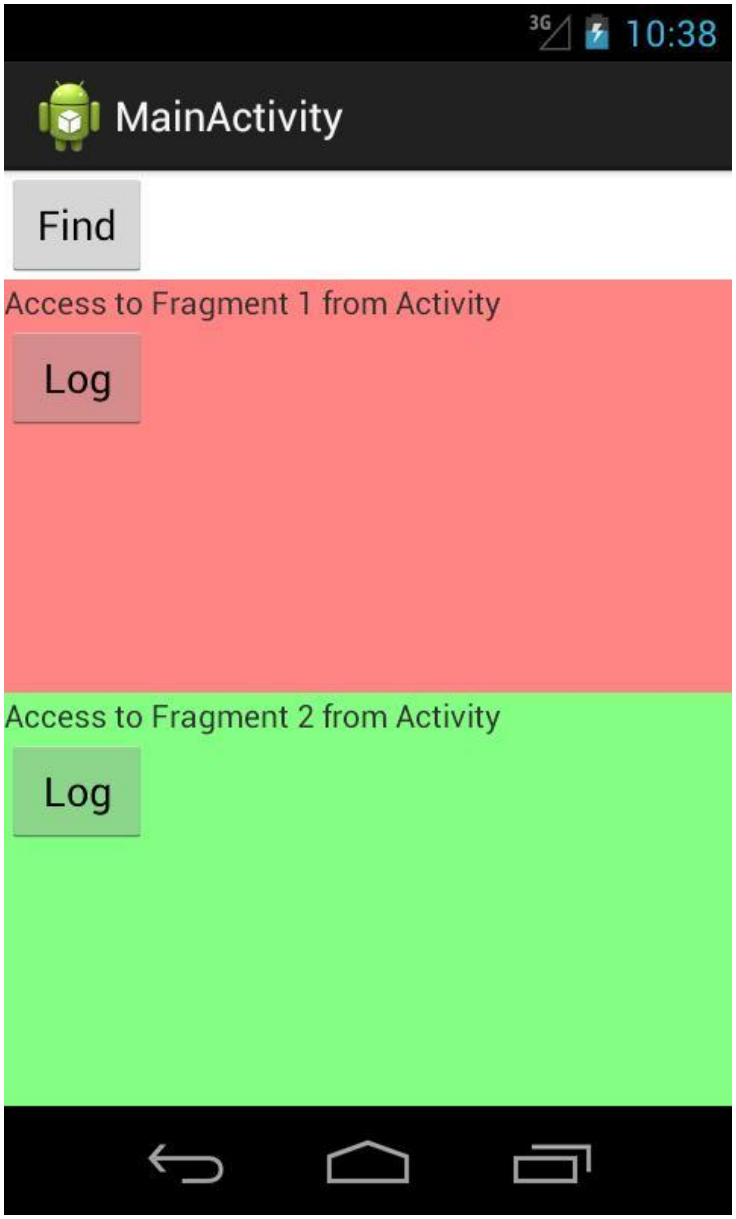
```
public void onClick(View v) {
  Fragment frag1 = getFragmentManager().findFragmentById(R.id.fragment1);
  ((TextView) frag1.getView().findViewById(R.id.textView))
      .setText("Access to Fragment 1 from Activity");

  Fragment frag2 = getFragmentManager().findFragmentById(R.id.fragment2);
  ((TextView) frag2.getView().findViewById(R.id.textView))
      .setText("Access to Fragment 2 from Activity");
```

}

Используем метод findFragmentById. В первом случае на вход передаем id компонента fragment, т.к. Fragment1 у нас размещен именно так. При поиске Fragment2 указываем id контейнера, в который этот фрагмент был помещен. В результате метод findFragmentById возвращает нам объект Fragment.

Далее мы получаем доступ к его View с помощью метода getView, находим в нем TextView и меняем текст.

Все сохраняем, запускаем. Жмем кнопку Find



Тексты в фрагментах обновились. Тем самым из Activity мы достучались до фрагментов и их компонентов.

На всякий случай проговорю одну вещь из разряда «Спасибо кэп!». Если посмотреть на код MainActivity, то можно заметить, что работая с frag2 в методе onCreate и с frag2 в методе onClick мы работаем с текущим фрагментом Fragment2. Это так и есть. Оба frag2 в итоге будут ссылаться на один объект. Так что, если вы динамически добавили фрагмент, то у вас уже есть ссылка на него, и искать его через findFragmentById вам уже не надо.

Доступ к Activity из фрагмента

Теперь попробуем из фрагмента поработать с Activity. Для этого фрагмент имеет метод getActivity.

Давайте перепишем обработчик кнопки в первом фрагменте. Будем менять текст кнопки btnFind.

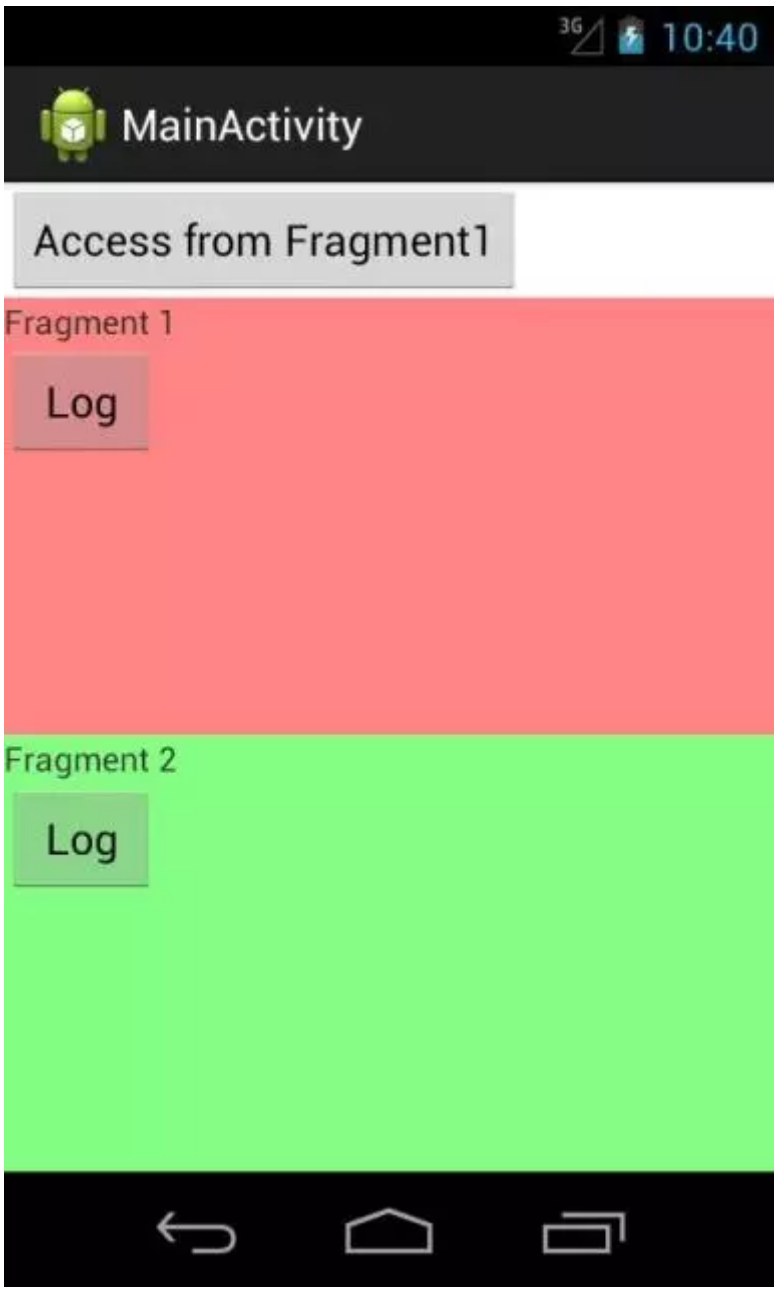
Fragment1.java:

```
package ru.startandroid.develop.p1061fragmentactivity;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;
import android.widget.Button;

public class Fragment1 extends Fragment {
```

```
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
      Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment1, null);


    Button button = (Button) v.findViewById(R.id.button);
    button.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        ((Button)getActivity().findViewById(R.id.btnFind)).setText("Access from
Fragment1");
      }
    });


    return v;
  }
}
```

Получаем Activity методом getActivity, ищем в нем кнопку и меняем текст.

Сохраняем, запускаем. Жмем кнопку в первом фрагменте:



Работает. Из фрагмента мы поменяли компонент Activity.

Обработка в Activity события из фрагмента

Рассмотрим механизм, который описан в хелпе: фрагмент генерирует некое событие и ставит Activity обработчиком.

Например, в Activity есть два фрагмента. Первый – список заголовков статей. Второй – отображает содержимое статьи, выбранной в первом. Мы

нажимаем на заголовок статьи в первом фрагменте и получаем содержимое во втором. В этом случае, цель первого фрагмента – передать в Activity информацию о том, что выбран заголовок. А Activity дальше уже сама решает, что делать с этой информацией. Если, например, приложение запущено на планшете в горизонтальной ориентации, то можно отобразить содержимое статьи во втором фрагменте. Если же приложение запущено на смартфоне, то экран маловат для двух фрагментов и надо запускать отдельное Activity со вторым фрагментом, чтобы отобразить статью.

Фишка тут в том, что первому фрагменту неинтересны все эти терзания Activity. Фрагмент – обособленный модуль. Его дело - проинформировать, что выбрана статья такая-то. Ему не надо искать второй фрагмент и работать с ним – это дело Activity.

Тут немного отвлекусь на небольшое лирическое отступление. Модульность, вообще, - очень важная и полезная штука. И ее надо использовать для универсальности, удобности и легкости в понимании работы своих приложений. Но уникальных рецептов, как правильно все организовать, конечно, нет. Каждый делает по-своему. Именно по этим причинам я в своих уроках даю чисто технические вещи про отдельные компоненты и не рассказываю, как организовывать и писать целое приложение. Иначе, форум бы уже ломился от сообщений, что я все делаю не так и надо по-другому, и каждый бы излагал свое видение. И была бы куча споров, где одна сторона говорит, что крокодил зеленый, а другая сторона говорит, что он нифига не зеленый, а длинный))

Вернемся к уроку. Фрагмент должен сообщить в Activity, что выбрана статья. Для этого он будет вызывать некий метод в Activity. И как нам сообщает хелп, лучший способ тут – это использовать интерфейс, который мы опишем в фрагменте и который затем будет реализован в Activity. Схема известная и распространенная. Давайте реализуем. В нашем приложении никаких статей нет, поэтому будем просто передавать произвольную строку из второго фрагмента в Activity. А Activity уже будет отображать эту строку в первом фрагменте.

Перепишем **Fragment2.java**:

```
package ru.startandroid.develop.p1061fragmentactivity;
```

```
import android.app.Activity;
import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;
import android.widget.Button;

public class Fragment2 extends Fragment {

  public interface onSomeEventListener {
    public void someEvent(String s);
  }

  onSomeEventListener someEventListener;

  @Override
  public void onAttach(Activity activity) {
    super.onAttach(activity);
        try {
          someEventListener = (onSomeEventListener) activity;
        } catch (ClassCastException e) {
            throw new ClassCastException(activity.toString() + " must implement
onSomeEventListener");
        }
  }
```

```
  final String LOG_TAG = "myLogs";

  public View onCreateView(LayoutInflater inflater, ViewGroup container,
      Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment2, null);

    Button button = (Button) v.findViewById(R.id.button);
    button.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        someEventListener.someEvent("Test text to Fragment1");
      }
    });

    return v;
  }
}
```

Описываем интерфейс **onSomeEventListener**. В нем метод **someEvent**, который на вход получает строку. Этот интерфейс будет реализовывать Activity.

В методе **onAttach** мы на вход получаем Activity, к которому присоединен фрагмент. Мы пытаемся привести это Activity к типу интерфейса onSomeEventListener, чтобы можно было вызывать метод someEvent и передать туда строку. Теперь someEventListener ссылается на Activity.

Далее, в **onCreateView**, в обработчике кнопки мы вызываем метод someEvent и передаем туда текст. Этот метод будет отработан в Activity.

Теперь меняем Activity.

MainActivity.java:

```
package ru.startandroid.develop.p1061fragmentactivity;

import
ru.startandroid.develop.p1061fragmentactivity.Fragment2.onSomeEventListener;
import android.app.Activity;
import android.app.Fragment;
import android.app.FragmentTransaction;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends Activity implements onSomeEventListener{

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    Fragment frag2 = new Fragment2();
    FragmentTransaction ft = getFragmentManager().beginTransaction();
    ft.add(R.id.fragment2, frag2);
    ft.commit();
  }

 @Override
 public void someEvent(String s) {
```

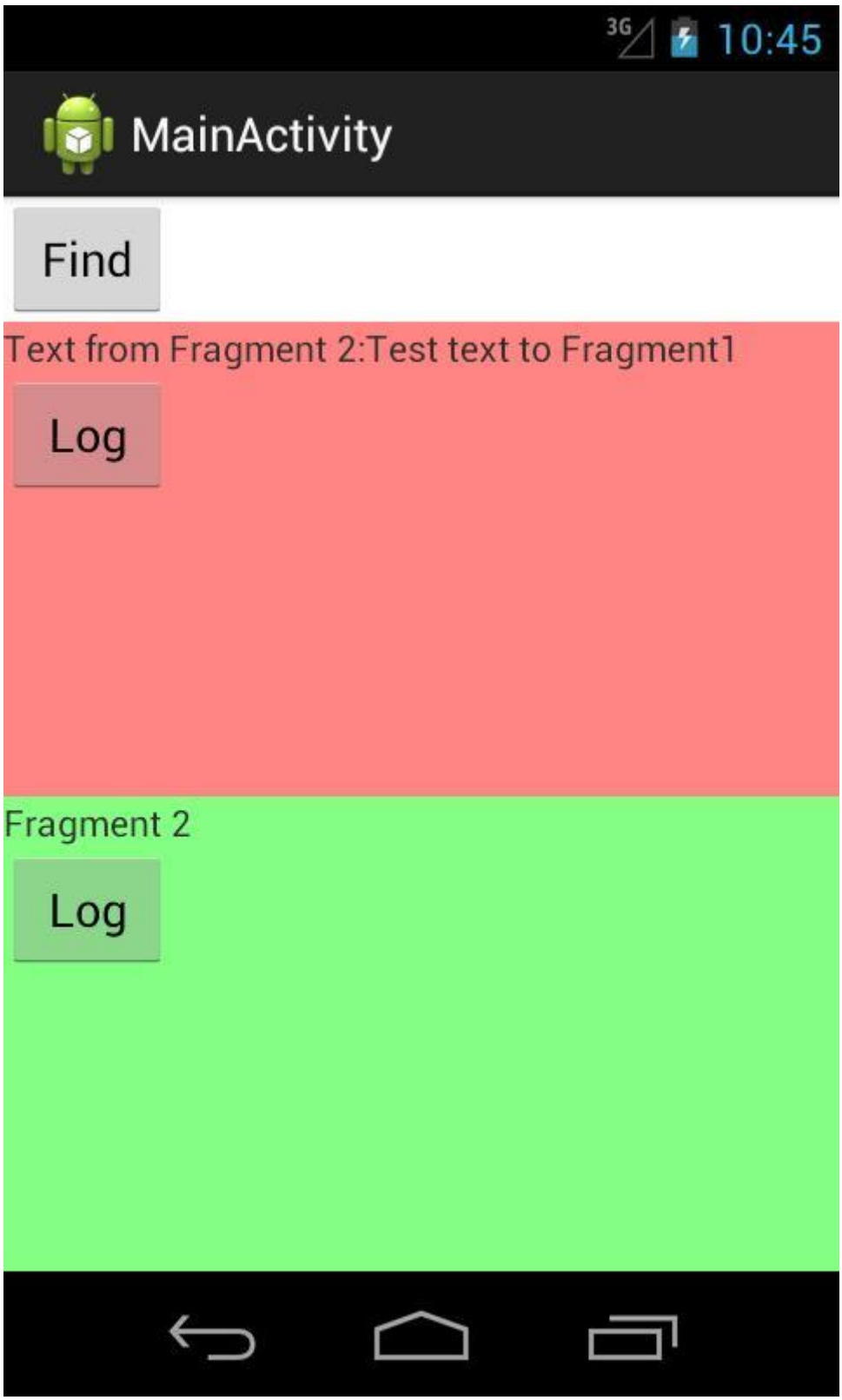
```
    Fragment frag1 = getFragmentManager().findFragmentById(R.id.fragment1);
    ((TextView)frag1.getView().findViewById(R.id.textView)).setText("Text from Fragment 2:" + s);
  }
}
```

Дописываем интерфейс **onSomeEventListener** к описанию класса.

onCreate без изменений.

Реализуем метод **someEvent**. Просто ищем первый фрагмент и вставляем туда текст.

Все сохраняем и запускаем. Жмем кнопку во втором фрагменте:



Второй фрагмент передал через интерфейс строку в Activity, а оно нашло первый фрагмент и отобразило там эту строку.

Fragments. ListFragment – список

В этом материале:

- работаем с ListFragment

Вернемся к фрагментам. В Android есть несколько полезных классов – наследников класса Fragment. Мы рассмотрим несколько из них. Начнем с ListFragment. В принципе, это просто Fragment, в котором есть методы, упрощающие доступ к ListView и некоторым его операциям.

Тут можно провести аналогию - для Activity есть класс наследник ListActivity. Когда мы только знакомились со списками, я не стал рассматривать этот класс, чтобы не вносить путаницы в непростую тему. Но в учебниках и хелпе о нем обычно говорят. Думаю, многие уже сталкивались с ним и примерно знают, что это такое. Если да, то ListFragment будет совсем прост для понимания.

Напишем приложение и рассмотрим основные возможности ListFragment.

Создадим проект:

Project name: P1091_ListFragment

Build Target: Android 4.1

Application name: ListFragment

Package name: ru.startandroid.develop.p1091listfragment

Create Activity: MainActivity

Создадим класс фрагмента, наследующий не android.app.Fragment как обычно, а android.app.ListFragment.

MainList.java:

package ru.startandroid.develop.p1091listfragment;

```
import android.app.ListFragment;


public class MainList extends ListFragment {


}
```

Класс так и оставляем пока пустым. И пока что не будем создавать Layout-файл для этого фрагмента. Дело в том, что ListFragment по умолчанию уже содержит ListView и мы вполне можем обойтись им. Адаптер мы пока также никакой не создаем, посмотрим чего получится.

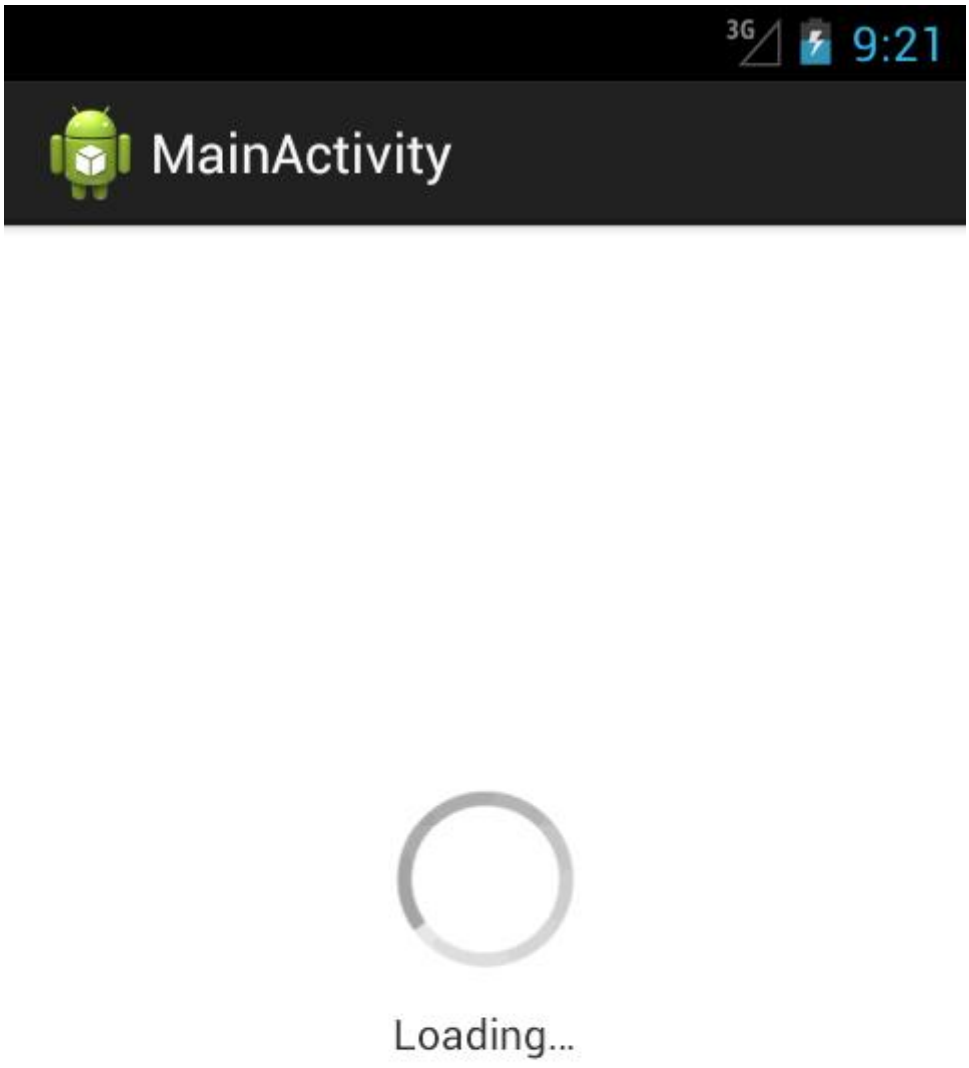
Редактируем layout для MainActivity

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:id="@+id/LinearLayout1"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical">
   <fragment
      android:name="ru.startandroid.develop.p1091listfragment.MainList"
      android:layout_width="match_parent"
      android:layout_height="match_parent">
   </fragment>
</LinearLayout>
```

Здесь только компонент fragment, использующий наш класс.

Все сохраняем, запускаем приложение.



ListFragment показывает нам, что он ждет данных.

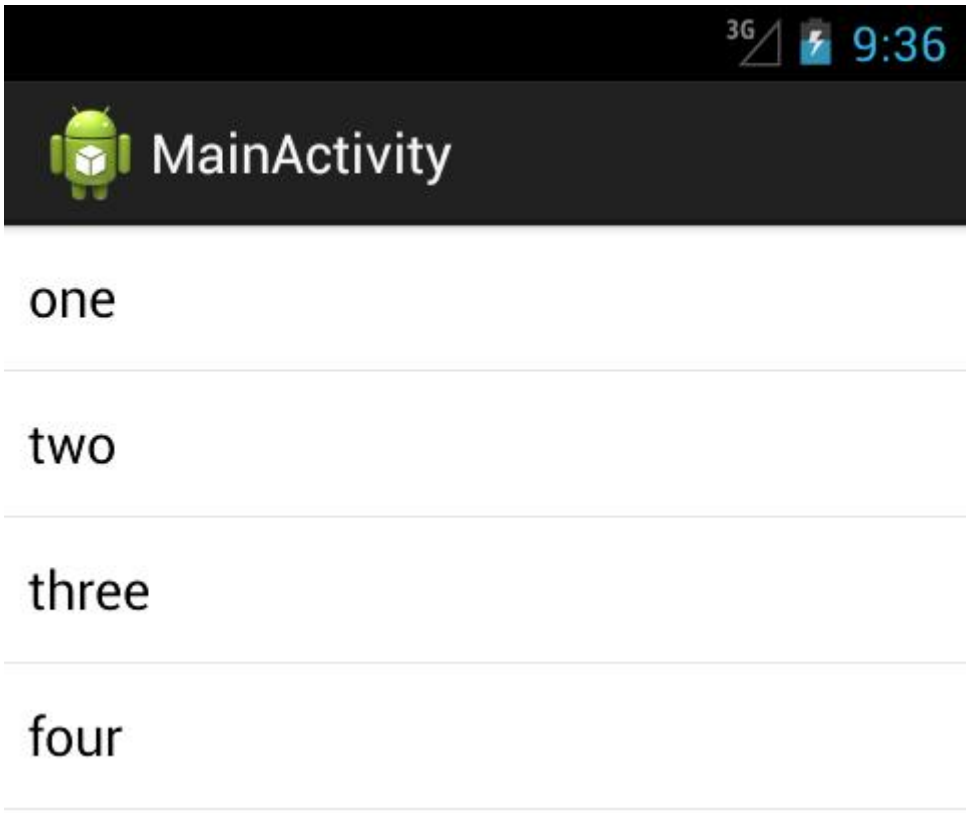
Ок, давайте дадим ему данные. Перепишем MainList.java:

```
package ru.startandroid.develop.p1091listfragment;

import android.app.ListFragment;
import android.os.Bundle;
import android.widget.ArrayAdapter;

public class MainList extends ListFragment {

  String data[] = new String[] { "one", "two", "three", "four" };

  @Override
  public void onActivityCreated(Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(getActivity(),
        android.R.layout.simple_list_item_1, data);
    setListAdapter(adapter);
  }

}
```

Мы создаем адаптер и используем метод setListAdapter, чтобы передать его списку. Обратите внимание - мы даже не создаем или не находим (findViewById) список (ListView), он уже есть где-то внутри фрагмента и

метод setListAdapter сам знает, как до него добраться. В принципе, это и есть основная фишка ListFragment - нам не надо работать с ListView.

Все сохраним, запустим приложение.



Данные появились.

Еще раз обращаю ваше внимание на то, что мы вообще не создавали никаких layout с ListView. ListFragment работает с каким-то своим, встроенным списком.

Если же вас чем-то не устраивает этот дефолтный список, можно использовать свой layout-файл для фрагмента.

В **strings.xml** добавим строки

```
<string name="number_list">Список чисел</string>
<string name="empty">Нет данных</string>
```

И создадим layout файл **fragment.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:orientation="vertical">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/number_list">
    </TextView>
    <ListView
        android:id="@id/android:list"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
```

```
    </ListView>
    <TextView
        android:id="@id/android:empty"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:text="@string/empty">
    </TextView>
</LinearLayout>
```

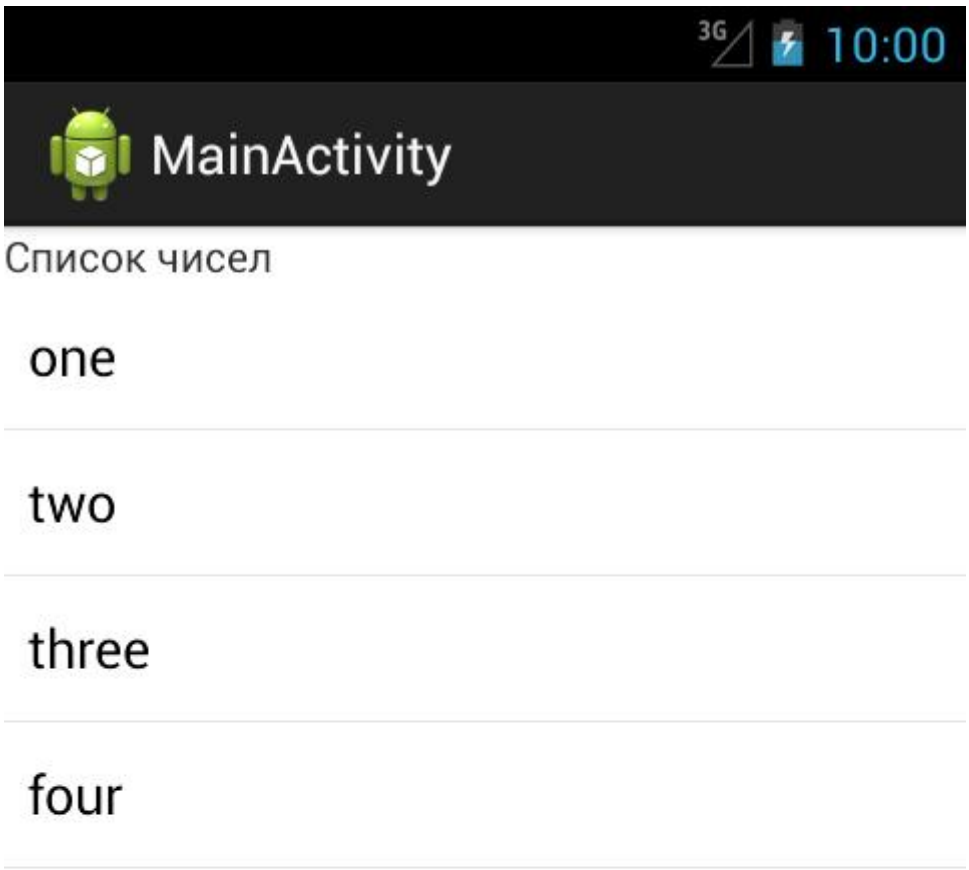
Первый TextView – просто заголовок списка. Далее идет ListView. Его ID обязательно должен быть равен @id/android:list. Чтобы ListFragment сам его нашел и мог с ним работать. Второй TextView – будет показан, если нет данных для списка. Его ID обязательно должен быть равен @id/android:empty.

Допишем в **MainList.java** метод создания View.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
  return inflater.inflate(R.layout.fragment, null);
}
```

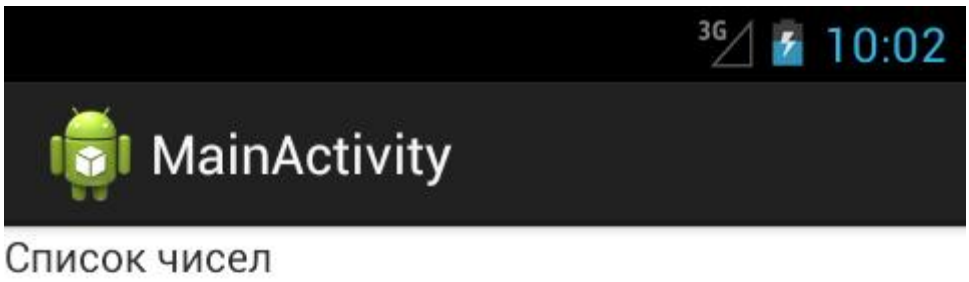
Фрагмент будет отображать компоненты из layout-файла, который мы только что создавали.

Все сохраняем и запускаем.



Видим заголовок и данные. Наш layout был использован.

Если не дать списку данные, то приложение будет выглядеть так

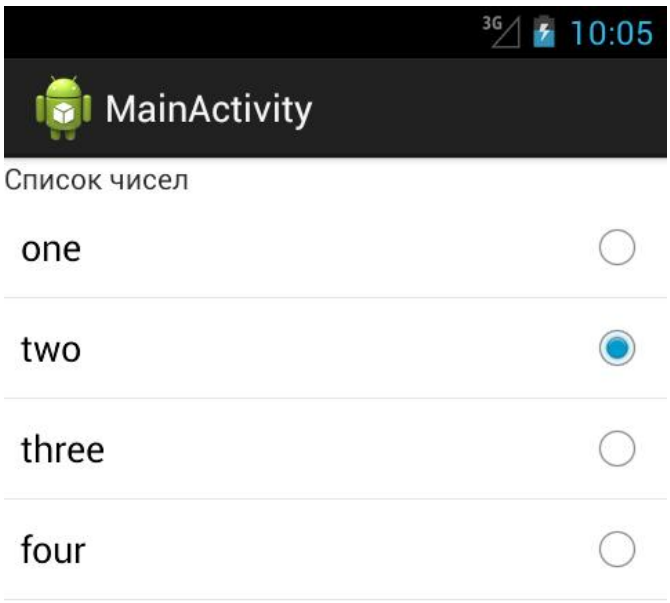


Отобразился наш второй TextView с ID = @id/android:empty. Причем, это вовсе не обязательно должен быть TextView. Главное тут именно этот ID. Компонент с этим ID будет показан вместо ListView, если нет данных.

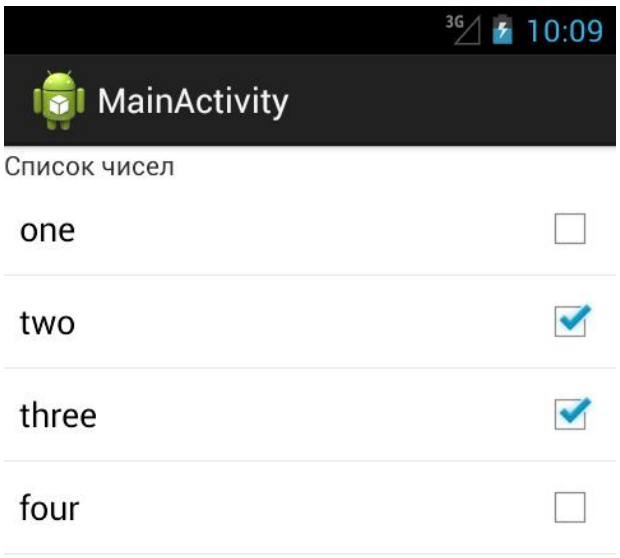
Чтобы включать одиночный и множественный выбор, необходимо провести все те же действия, которые мы рассматривали в Уроке 43: передать в адаптер соответствующий layout-файл и включить соответствующий режим выбора для списка. В этом же 43-м Уроке можно посмотреть, как определять, какие пункты списка выбраны. Чтобы в ListFragment получить доступ к списку – используйте метод **getListView**.

Результаты включения режима выбора будут такие:

одиночный выбор

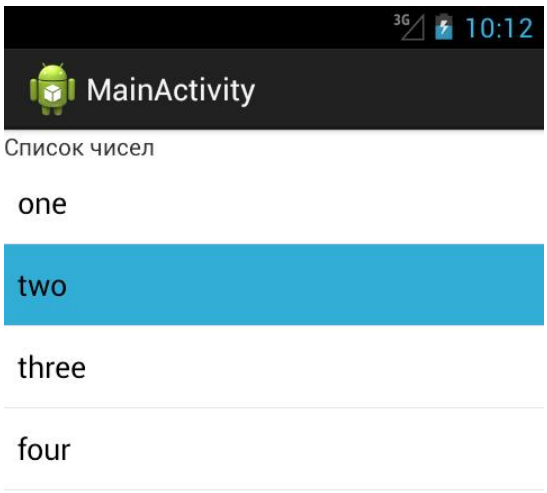


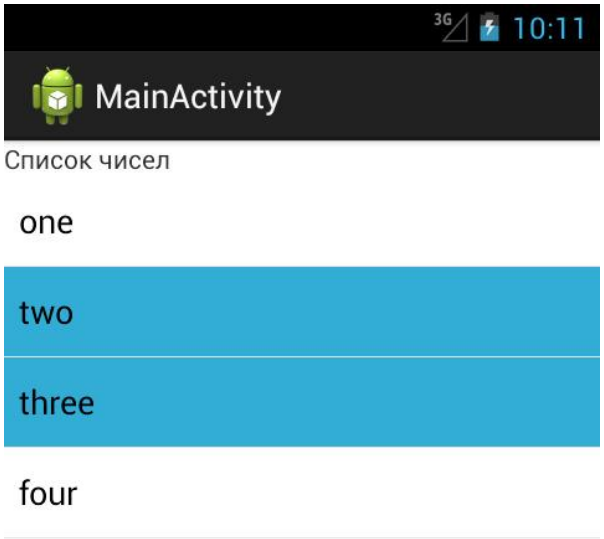
множественный выбор



Еще, как вариант, можно в адаптер передавать layout-файл android.R.layout.simple_list_item_activated_1.

В этом случае результаты будут такие:


10:12
MainActivity
Список чисел
one
two
three
four


10:11
MainActivity
Список чисел
one
two
three
four

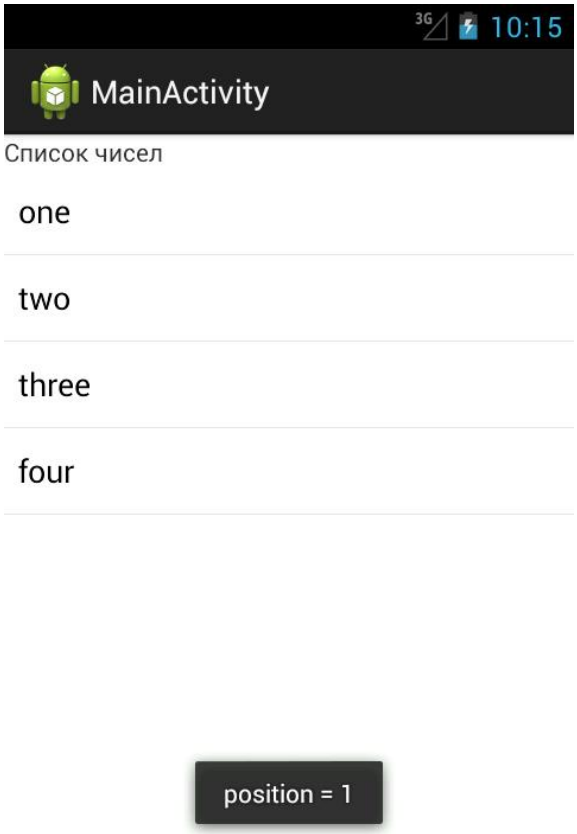
Это хорошо подходит для случая, когда у вас на экране слева фрагмент-список, а справа фрагмент-содержимое и вы всегда видите какой именно пункт из списка сейчас просматриваете.

Чуть не забыл про самое главное) Ловить нажатия можно в методе onListItemClick. Он очень похож на метод onItemClick из предыдущих материалов.

Если добавим в MainList.java его реализацию:

```
public void onListItemClick(ListView l, View v, int position, long id) {
  super.onListItemClick(l, v, position, id);
  Toast.makeText(getActivity(), "position = " + position,
Toast.LENGTH_SHORT).show();
}
```

то при нажатии на пункт списка, увидим результат:



Fragments. DialogFragment – диалог

В этом материале:

- работаем с DialogFragment

Продолжаем рассматривать наследников Fragment. DialogFragment – отличается от обычного фрагмента тем, что отображается как диалог и имеет соответствующие методы.

Построить диалог можно двумя способами: используя свой layout-файл и через AlertDialog.Builder. Нарисуем приложение, которое будет вызывать два диалога, построенных разными способами.

Создадим проект:

Project name: P1101_DialogFragment

Build Target: Android 4.1

Application name: DialogFragment

Package name: ru.startandroid.develop.p1101dialogfragment

Create Activity: MainActivity

Добавим строки в strings.xml:

<string name="dialog_1">Dialog 1</string>

<string name="dialog_2">Dialog 2</string>

<string name="message_text">Text of your message</string>

<string name="yes">Yes</string>

```
<string name="no">No</string>
<string name="maybe">Maybe</string>
```

Мы будем создавать два диалога, соответственно нам понадобятся два фрагмента.

Создадим layout-файл для первого фрагмента.

dialog1.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_margin="20dp"
        android:text="@string/message_text"
        android:textAppearance="?android:attr/textAppearanceLarge">
    </TextView>
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <Button
            android:id="@+id/btnYes"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
            android:layout_margin="10dp"
            android:text="@string/yes">
        </Button>
        <Button
            android:id="@+id/btnNo"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:text="@string/no">
        </Button>
        <Button
            android:id="@+id/btnMaybe"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:text="@string/maybe">
        </Button>
    </LinearLayout>
</LinearLayout>
```

Так будет выглядеть наш диалог – текст сообщения и три кнопки.

Создаем класс **Dialog1.java**:

```
package ru.startandroid.develop.p1101dialogfragment;

import android.app.DialogFragment;
import android.content.DialogInterface;
import android.os.Bundle;
import android.util.Log;
```

```
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;
import android.widget.Button;

public class Dialog1 extends DialogFragment implements OnClickListener {

  final String LOG_TAG = "myLogs";

  public View onCreateView(LayoutInflater inflater, ViewGroup container,
      Bundle savedInstanceState) {
    getDialog().setTitle("Title!");
    View v = inflater.inflate(R.layout.dialog1, null);
    v.findViewById(R.id.btnYes).setOnClickListener(this);
    v.findViewById(R.id.btnNo).setOnClickListener(this);
    v.findViewById(R.id.btnMaybe).setOnClickListener(this);
    return v;
  }

  public void onClick(View v) {
    Log.d(LOG_TAG, "Dialog 1: " + ((Button) v).getText());
    dismiss();
  }

  public void onDismiss(DialogInterface dialog) {
    super.onDismiss(dialog);
    Log.d(LOG_TAG, "Dialog 1: onDismiss");
  }
```

```
  public void onCancel(DialogInterface dialog) {
    super.onCancel(dialog);
    Log.d(LOG_TAG, "Dialog 1: onCancel");
  }
}
```

В onCreateView мы получаем объект Dialog с помощью метода getDialog и устанавливаем заголовок диалога. Далее мы создаем view из layout, находим в нем кнопки и ставим им текущий фрагмент в качестве обработчика.

В onClick выводим в лог текст нажатой кнопки и сами явно закрываем диалог методом dismiss.

Метод onDismiss срабатывает, когда диалог закрывается. Пишем об этом в лог.

Метод onCancel срабатывает, когда диалог отменяют кнопкой Назад. Пишем об этом в лог.

Создаем второй фрагмент. Здесь мы будем строить диалог с помощью билдера, поэтому layout-файл не понадобится. Создаем только класс **Dialog2.java**:

```
package ru.startandroid.develop.p1101dialogfragment;


import android.app.AlertDialog;
import android.app.Dialog;
import android.app.DialogFragment;
import android.content.DialogInterface;
import android.content.DialogInterface.OnClickListener;
import android.os.Bundle;
import android.util.Log;


public class Dialog2 extends DialogFragment implements OnClickListener {
```

```
final String LOG_TAG = "myLogs";

public Dialog onCreateDialog(Bundle savedInstanceState) {
  AlertDialog.Builder adb = new AlertDialog.Builder(getActivity())
      .setTitle("Title!").setPositiveButton(R.string.yes, this)
      .setNegativeButton(R.string.no, this)
      .setNeutralButton(R.string.maybe, this)
      .setMessage(R.string.message_text);
  return adb.create();
}

public void onClick(DialogInterface dialog, int which) {
  int i = 0;
  switch (which) {
  case Dialog.BUTTON_POSITIVE:
    i = R.string.yes;
    break;
  case Dialog.BUTTON_NEGATIVE:
    i = R.string.no;
    break;
  case Dialog.BUTTON_NEUTRAL:
    i = R.string.maybe;
    break;
  }
  if (i > 0)
    Log.d(LOG_TAG, "Dialog 2: " + getResources().getString(i));
}

public void onDismiss(DialogInterface dialog) {
```

```
    super.onDismiss(dialog);
    Log.d(LOG_TAG, "Dialog 2: onDismiss");
  }

  public void onCancel(DialogInterface dialog) {
    super.onCancel(dialog);
    Log.d(LOG_TAG, "Dialog 2: onCancel");
  }
}
```

Обычно для заполнения фрагмента содержимым мы использовали метод onCreateView. Для создания диалога с помощью билдера используется onCreateDialog. Создаем диалог с заголовком, сообщением и тремя кнопками. Обработчиком для кнопок назначаем текущий фрагмент.

В **onClick** определяем, какая кнопка была нажата и выводим соответствующий текст в лог. В случае создания диалога через билдер, диалог сам закроется по нажатию на кнопку, метод dismiss здесь не нужен.

Методы **onDismiss** и **onCancel** – это закрытие и отмена диалога, аналогично первому фрагменту.

Меняем layout-файл для MainActivity - **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:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <Button
        android:id="@+id/btnDlg1"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:text="@string/dialog_1">
    </Button>
    <Button
        android:id="@+id/btnDlg2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:text="@string/dialog_2">
    </Button>
</LinearLayout>
```

Здесь только две кнопки.

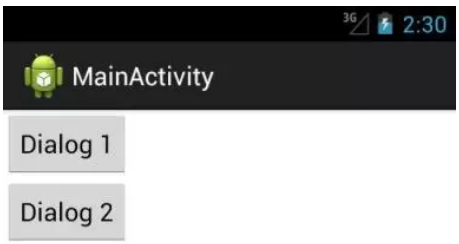
Кодим **MainActivity.java**:

```
package ru.startandroid.develop.p1101dialogfragment;

import android.app.Activity;
import android.app.DialogFragment;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends Activity {

  DialogFragment dlg1;
  DialogFragment dlg2;

  @Override
```

```
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    dlg1 = new Dialog1();
    dlg2 = new Dialog2();
  }

  public void onClick(View v) {
    switch (v.getId()) {
    case R.id.btnDlg1:
      dlg1.show(getFragmentManager(), "dlg1");
      break;
    case R.id.btnDlg2:
      dlg2.show(getFragmentManager(), "dlg2");
      break;
    default:
      break;
    }

  }
}
```

Создаем диалоги и запускаем их методом show, который на вход требует FragmentManager и строку-тэг. Транзакция и коммит происходят внутри этого метода, нам об этом думать не надо.

Все сохраняем и запускаем приложение.



Жмем Dialog1

Отобразился наш простенький диалог.

Жмем какую-нибудь кнопку, например, Yes - диалог закрылся. Смотрим логи:

Dialog 1: Yes

Dialog 1: onDismiss

Все верно.

Снова запустим первый диалог и нажмем клавишу **Назад (Back).** Смотрим лог:

Dialog 1: onCancel

Dialog 1: onDismiss

Сработал onCancel – диалог был отменен, и onDismiss – диалог закрылся.

Если мы будем поворачивать экран, то каждый раз будет отрабатывать onDismiss, но диалог снова будет отображен после поворота.

Запустим второй диалог – нажмем кнопку Dialog 2.

Отобразился стандартный сконструированный нами диалог. Жмем, например, No – диалог закрылся. В логах:

Dialog 2: No

Dialog 2: onDismiss

Снова запустим второй диалог и нажмем **Назад**. В логах:

Dialog 2: onCancel

Dialog 2: onDismiss

Все так же, как и в первом случае.

Еще несколько слов по теме.

Если вы не хотите, чтобы ваш диалог можно было закрыть кнопкой, используйте для вашего диалог-фрагмента метод setCancelable с параметром false.

Есть еще один вариант вызова диалога. Это метод show, но на вход он уже принимает не FragmentManager, а FragmentTransaction. В этом случае система также сама вызовет commit внутри show, но мы можем предварительно поместить в созданную нами транзакцию какие-либо еще операции или отправить ее в BackStack.

Вы можете использовать диалог-фрагменты, как обычные фрагменты и отображать их на Activity, а не в виде диалога. Но при этом будьте аккуратнее с использованием **getDialog**. Я так понял, что он возвращает null в этом случае.

Если **AlertDialog.Builder** вам незнаком, то посмотрите документацию. Там достаточно подробно описано, как создавать различные диалоги.

Fragments. PreferenceFragment - настройки. Headers

В этом материале:

- работаем с PreferenceFragment
- используем Headers

Есть особый вид Activity – PreferenceActivity. Оно позволяет нам удобно работать с Preferences. Для фрагментов есть аналог – это PreferenceFragment, имеющий тот же функционал.

У PreferenceActivity есть возможность – Headers (заголовки). Они позволяют на больших экранах без труда отображать настройки, разделенные на две панели по вертикали. Если же экран мал для такого разделения, то эти панели будут на разных экранах.

Создадим приложение, в котором используем PreferenceFragment с простыми настройками. А далее переделаем его под использование заголовков.

Создадим проект:

Project name: P1111_PreferenceFragment

Build Target: Android 4.1

Application name: PreferenceFragment

Package name: ru.startandroid.develop.p1111preferencefragment

Create Activity: MainActivity

Добавим строки в **strings.xml:**

```
<string name="checkbox1_title">CheckBox 1</string>
<string name="checkbox1_summary">Summary of CheckBox 1</string>
<string name="edittext1_title">EditText 1</string>
<string name="edittext1_summary">Summary of EditText 1</string>
<string name="list1_title">List 1</string>
<string name="list1_summary">Summary of List 1</string>
<string-array name="entries">
   <item>one</item>
   <item>two</item>
   <item>three</item>
</string-array>
<string-array name="entry_values">
   <item>1</item>
   <item>2</item>
   <item>3</item>
</string-array>
```

Тексты будут использоваться для описания настроек, а массивы – для списка.

Создадим файл, описывающий настройки. Для этого в папке **res** создаем папку **xml,** если ее там нет. И создаем файл: **res/xml/pref1.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
   xmlns:android="http://schemas.android.com/apk/res/android">
   <CheckBoxPreference
      android:key="chb1"
      android:summary="@string/checkbox1_summary"
```

```
        android:title="@string/checkbox1_title">
    </CheckBoxPreference>
    <EditTextPreference
        android:key="address1"
        android:summary="@string/edittext1_summary"
        android:title="@string/edittext1_title">
    </EditTextPreference>
    <ListPreference
        android:entries="@array/entries"
        android:entryValues="@array/entry_values"
        android:key="list1"
        android:summary="@string/list1_summary"
        android:title="@string/list1_title">
    </ListPreference>
</PreferenceScreen>
```

Три простейших настройки.

Итак, файл настроек создан. Нужен фрагмент, который эти настройки нам покажет. Создаем класс **Fragment1**, наследующий **android.preference.PreferenceFragment**.

Fragment1.java:

```
package ru.startandroid.develop.p1111preferencefragment;

import android.os.Bundle;
import android.preference.PreferenceFragment;

public class Fragment1 extends PreferenceFragment {
```

```
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    addPreferencesFromResource(R.xml.pref1);
  }

}
```

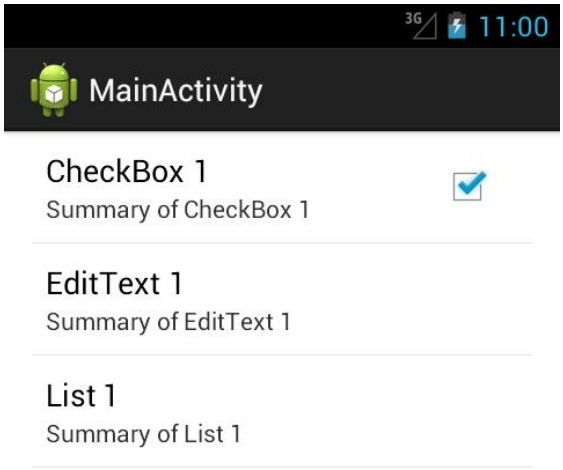
Метод addPreferencesFromResource прочитает файл с описанием настроек и выведет их на экран.

Осталось немного подправить **MainActivity.java**:

```
package ru.startandroid.develop.p1111preferencefragment;

import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity {

  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    getFragmentManager().beginTransaction()
        .replace(android.R.id.content, new Fragment1()).commit();

  }
}
```

Мы здесь не используем никакой layout, а сразу добавляем наш фрагмент в качестве контента, используя корневой контейнер с ID android.R.id.content.

Все сохраняем и запускаем приложение. Видим экран настроек



Можно поставить галку или нажать на два других пункта настроек и получить диалоговое окно для ввода значения. Все как обычно.

Заголовки

Теперь рассмотрим, что нам дают заголовки. Можно сказать, что заголовки - это корневые ветки дерева настроек. Мы сделаем три таких ветки: первые две будут открывать наши фрагменты с настройками, а третья откроет настройки звука.

Один фрагмент с настройками у нас уже есть, создадим второй.

Добавим строки в **strings.xml:**

<string name="checkbox2_title">CheckBox 2</string>

<string name="checkbox2_summary">Summary of CheckBox 2</string>

```
<string name="edittext2_title">EditText 2</string>
<string name="edittext2_summary">Summary of EditText 2</string>
<string name="list2_title">List 2</string>
<string name="list2_summary">Summary of List 2</string>
<string name="category1">Category 1</string>
<string name="category2">Category 2</string>
<string name="screen1_title">Screen 1</string>
<string name="screen1_summary">Summary of Screen 1</string>
<string name="screen2_title">Screen 2</string>
<string name="screen2_summary">Summary of Screen 2</string>
<string name="header1_title">Header 1</string>
<string name="header1_summary">Summary of Header 1</string>
<string name="header2_title">Header 2</string>
<string name="header2_summary">Summary of Header 2</string>
<string name="header3_title">Header 3</string>
<string name="header3_summary">Summary of Header 3</string>
```

Создадим файл res/xml/pref2.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
   xmlns:android="http://schemas.android.com/apk/res/android">
   <PreferenceCategory
      android:title="@string/category1">
      <CheckBoxPreference
         android:key="chb2"
         android:summary="@string/checkbox2_summary"
         android:title="@string/checkbox2_title">
      </CheckBoxPreference>
      <EditTextPreference
```

```
            android:key="address2"
            android:summary="@string/edittext2_summary"
            android:title="@string/edittext2_title">
        </EditTextPreference>
        <ListPreference
            android:entries="@array/entries"
            android:entryValues="@array/entry_values"
            android:key="list2"
            android:summary="@string/list2_summary"
            android:title="@string/list2_title">
        </ListPreference>
    </PreferenceCategory>
    <PreferenceCategory
        android:title="@string/category2">
        <PreferenceScreen

android:fragment="ru.startandroid.develop.p1111preferencefragment.Fragment1"
            android:summary="@string/screen1_summary"
            android:title="@string/screen1_title">
        </PreferenceScreen>
        <PreferenceScreen
            android:summary="@string/screen2_summary"
            android:title="@string/screen2_title">
            <intent
                android:action="android.intent.action.VIEW"
                android:data="http://www.developer.android.com">
            </intent>
        </PreferenceScreen>
    </PreferenceCategory>
```

```
</PreferenceScreen>
```

Здесь используем те же простые элементы, что и ранее, а также – категории и экраны Только в экранах мы указываем не дочерние элементы, а перенаправления. В первом экране с помощью атрибута android:fragment мы указываем Fragment1, который мы создали в начале урока. А во втором экране указываем intent, который откроет в браузере страницу.

Создаем фрагмент, который нам эти настройки покажет.

Fragment2.java:

```
package ru.startandroid.develop.p1111preferencefragment;


import android.os.Bundle;
import android.preference.PreferenceFragment;


public class Fragment2 extends PreferenceFragment {


  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    addPreferencesFromResource(R.xml.pref2);
  }


}
```

Переходим к созданию заголовков

Создаем файл с описанием заголовков **res/xml/pref_head.xml:**

```
<?xml version="1.0" encoding="utf-8"?>
<preference-headers
    xmlns:android="http://schemas.android.com/apk/res/android">
    <header
```

```
android:fragment="ru.startandroid.develop.p1111preferencefragment.Fragment1"
        android:icon="@android:drawable/ic_menu_call"
        android:summary="@string/header1_summary"
        android:title="@string/header1_title">
    </header>
    <header

android:fragment="ru.startandroid.develop.p1111preferencefragment.Fragment2"
        android:icon="@android:drawable/ic_menu_info_details"
        android:summary="@string/header2_summary"
        android:title="@string/header2_title">
    </header>
    <header
        android:icon="@android:drawable/ic_menu_view"
        android:summary="@string/header3_summary"
        android:title="@string/header3_title">
        <intent
            android:action="android.settings.DISPLAY_SETTINGS">
        </intent>
    </header>
</preference-headers>
```

Три заголовка. Первый откроет Fragment1, второй – Fragment2, третий, используя intent, – настройки экрана. Иконки взяты случайные.

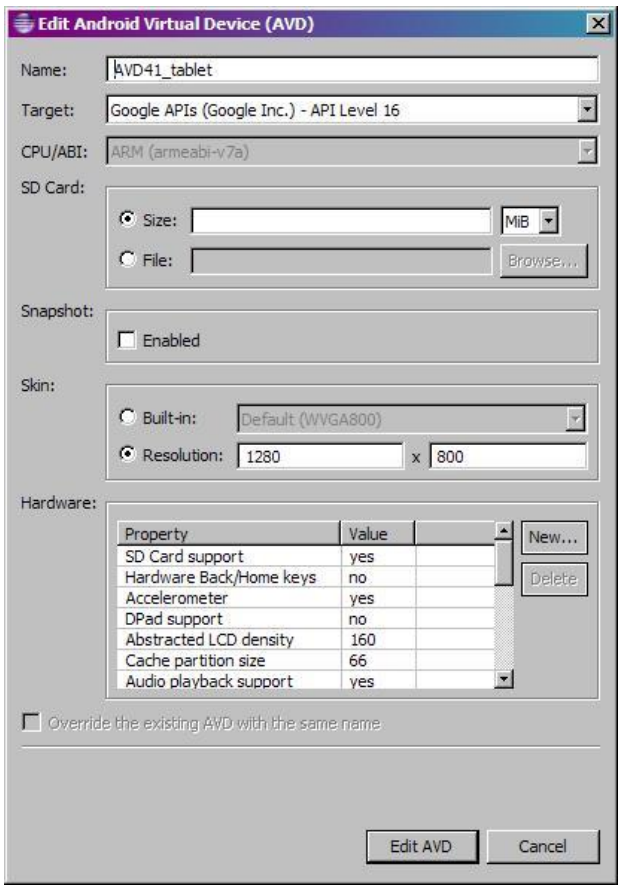
Ок. Заголовки есть, теперь надо их отобразить. Для этого используем MainActivity. Но, чтобы оно умело работать с заголовками, оно должно наследовать PreferenceActivity.

Перепишем **MainActivity.java**:

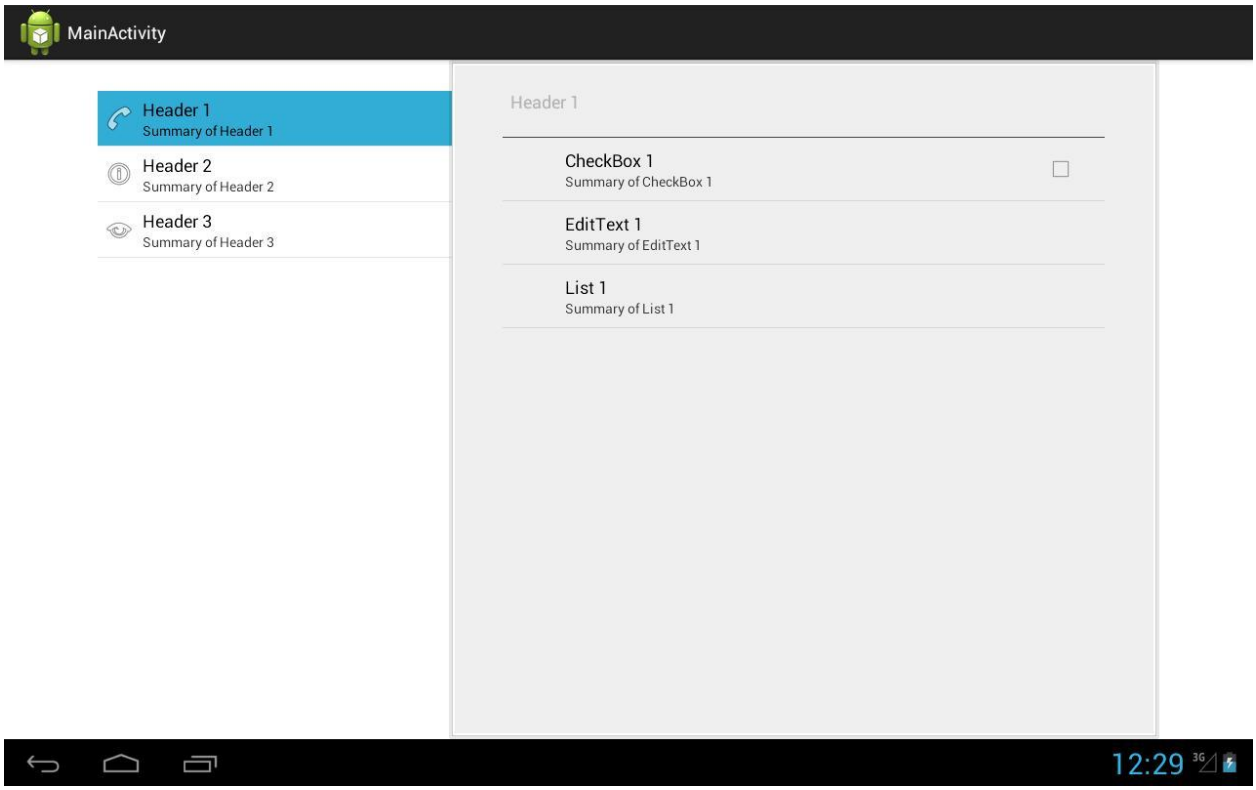
```
package ru.startandroid.develop.p1111preferencefragment;

import java.util.List;

import android.preference.PreferenceActivity;

public class MainActivity extends PreferenceActivity {

  public void onBuildHeaders(List<Header> target) {
    loadHeadersFromResource(R.xml.pref_head, target);
  }
}
```

Метод onBuildHeaders вызывается системой, когда надо строить заголовки. На вход он принимает список List<Header>, который нам надо наполнить. Для этого вызываем метод loadHeadersFromResource, и передаем ему наш файл с заголовками и наполняемый список.

Все сохраняем и запускаем. Я использую для запуска AVD, эмулирующий планшет на 4.1

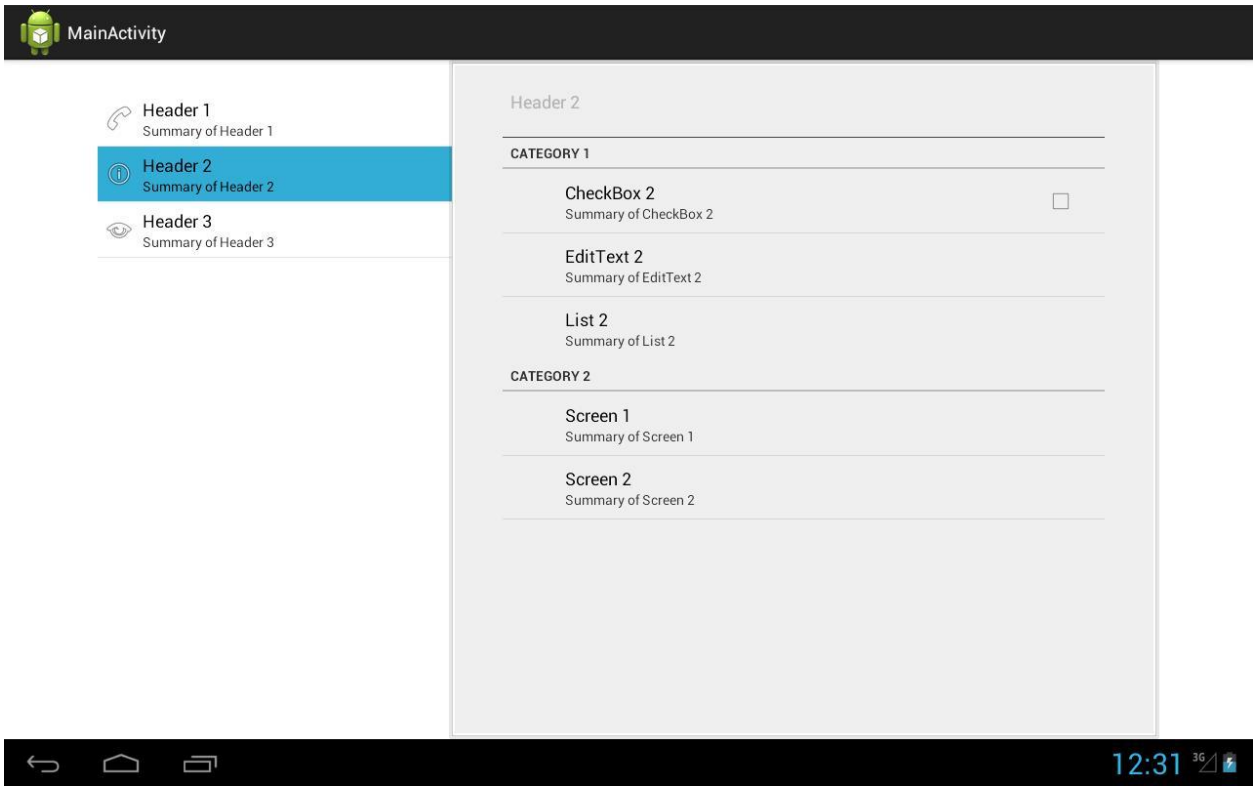


Приложение выглядит так:



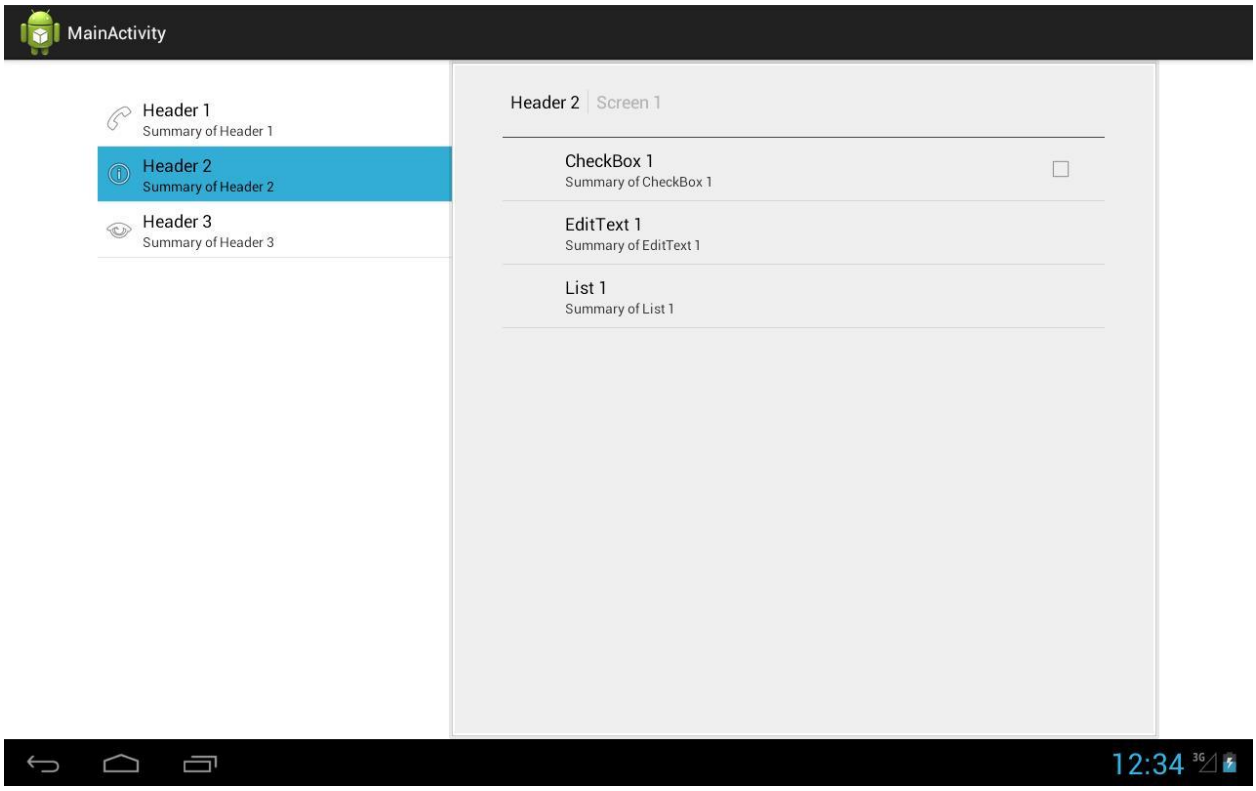
Мы видим три заголовка, которые мы прописывали в файле **pref_head.xml**. Открыт первый - **Header 1**, он отображает Fragment1. А **Fragment1** отображает настройки из **pref1.xml**.

Нажмем на второй заголовок - **Header 2.**



Видим **Fragment2** с настройками из **pref2.xml**. Видны категории и экраны, которые мы создавали. Напомню, что **Screen 1** должен открыть **Fragment1**, а **Screen 2** – сайт. Проверим.

Жмем Screen 1.

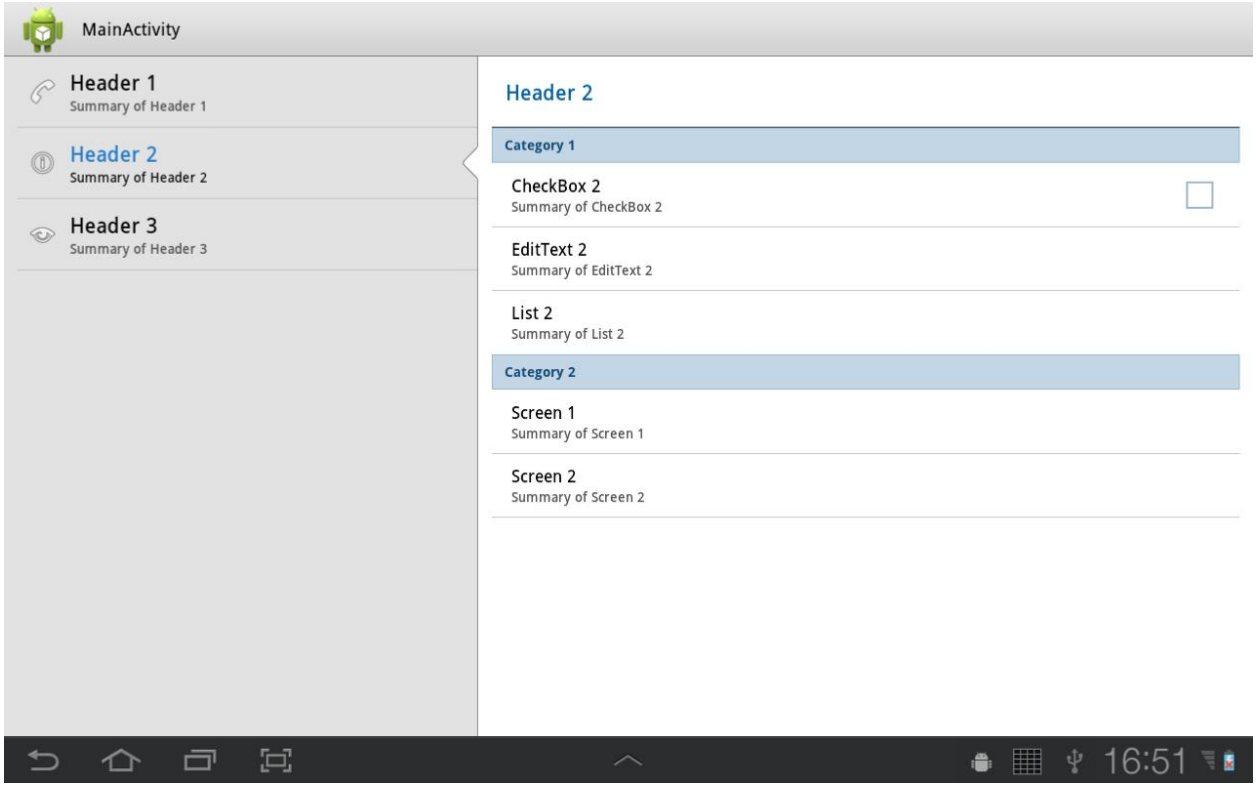


Открылся **Fragment1** как вложенный экран. Сверху отображается полный путь вложенности. Вернуться можно кнопкой Назад. Таким образом, вы можете создавать вложенные экраны с настройками, не уходя от заголовков.

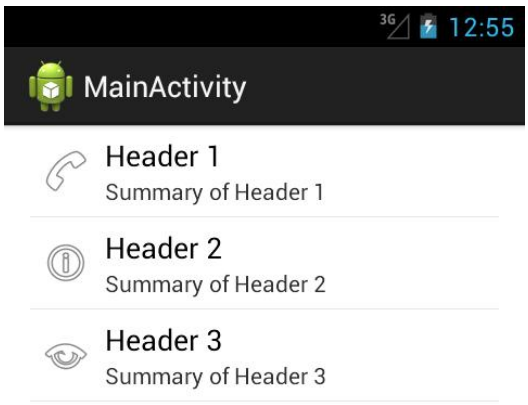
Нажатие на **Screen 2** запустит браузер и откроет ссылку.

Нажатие на **Header 3** откроет настройки экрана.

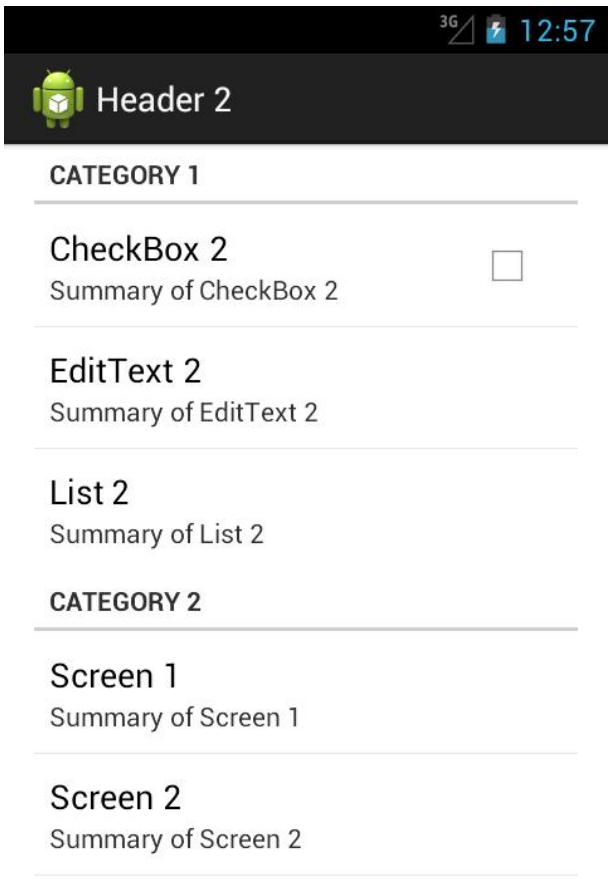
Если запустить это же приложение на планшете с Android 3.2, оно будет выглядеть так:



Если запустить, не на планшете, а на смартфоне с 4.1, то приложение само для себя каким-то образом решает, что разбить экран на две части здесь не получится и показывает заголовки на одном экране



а содержимое на другом



Чтобы программно определить, будет экран делиться на две части или нет, можно использовать метод isMultiPane.