Лекция #25.1. Фрагменты в котиках

Фрагменты немного пугают новичков. Постараемся изложить это как можно проще, чтобы отдельные фрагменты (~~каламбур~~ 😊) пазла сложились в единую картину.

Зачем они вообще нужны?

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

Старые программы прекрасно на них запускались, но обнаружилось несколько недостатков. На больших экранах интерфейс выглядел не слишком элегантно, появились большие пустые пространства. И тогда возникла идея объединить два отдельных экрана из смартфона в один экран на планшете. Это самый

классический пример применения фрагмента. По сути, это **костыль**. Возможно, если бы сразу подумали головой, то придумали бы более элегантное решение. Но нам выбирать не приходится – будем использовать предложенную концепцию.

Фрагменты были представлены в **API 11 (Android 3.0)**, но в целях совместимости была написана специальная библиотека **Android Support library** для старых устройств. Долгое время существовало два класса **Fragment**: для новых устройств и для старых устройств. Названия методов и классов были очень похожи, и разработчики часто путались, смешивая в одном проекте два разных несовместимых класса. Спустя некоторое время решили отказаться от странного разделения, класс для новых устройств признали устаревшим, а класс из библиотеки поддержки старых устройств сменил своё полное имя и вошёл в состав **AndroidX**.

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

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

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

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

Фрагменты являются строительным материалом для приложения. Вы можете в нужное время добавить новый фрагмент, удалить ненужный фрагмент или

заменить один фрагмент на другой. Точно так же мы собираем пазл - подносим фрагмент кота в общую картину, иногда ошибаемся и тогда заменяем кусочек пазла на другой и т.д.

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

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

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

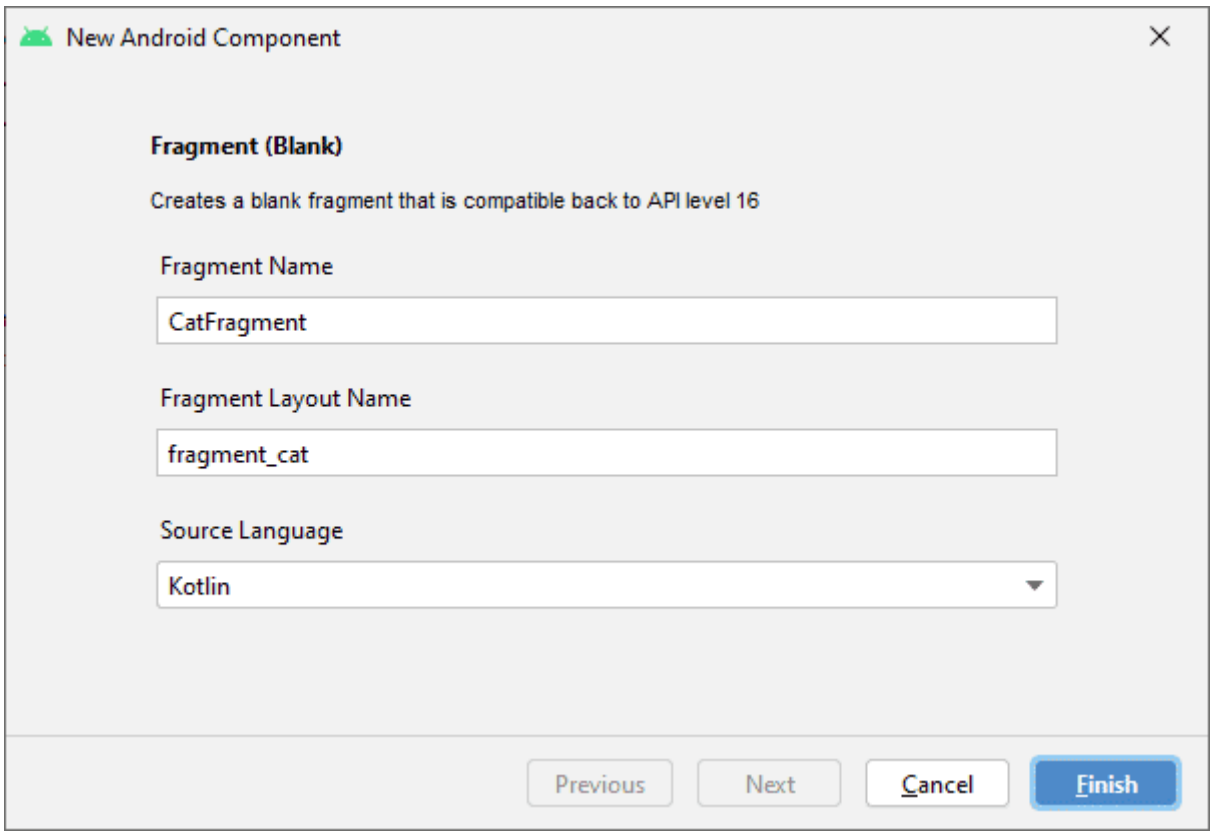
Второй вариант использует динамическое подключение фрагмента. Принцип следующий – в разметку помещается макет из группы **ViewGroup**, который становится контейнером для фрагмента. Обычно, для этой цели используют **FrameLayout**, но это не обязательное условие. И в нужный момент фрагмент замещает контейнер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

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

Реакция разработчиков на появление фрагментов противоречива. Кто-то активно использует их в своих проектах, а кто-то их не переносит и использует альтернативные варианты. Похоже, в стане **Гугла** также идёт борьба между двумя группами программистов. Фрагменты постоянно развиваются, меняются меняются и дорабатываются.

Первое знакомство

Для первого знакомства создадим стандартный проект на базе **Empty Activity** и вручную добавим фрагмент. Итак, после создания проекта выбираем из контекстного меню пакета **New | Fragment | Fragment (Blank)**. В диалоговом окне мастера назначаем имя для фрагмента.



На этом этапе создание нового фрагмента напоминает создание новой активности. Мы создаём новый класс и автоматически генерируется макет для него **fragment_cat.xml.** Единственное отличие, в манифест ничего не добавляется.

Откроем макет фрагмента и немного отредактируем содержимое, добавив центрирование по центру. Также можно было добавить **ImageView** с изображением кота. Но не будем пока ничего усложнять.

```
<?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=".CatFragment">

    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:textAppearance="@style/TextAppearance.AppCompat.Display2"
        android:text="@string/hello_blank_fragment" />

</FrameLayout>
```

В **res/values/strings.xml** содержится ресурс **hello_blank_fragment**. Изменим текст на что-то понятное:

```
<string name="hello_blank_fragment">Это фрагмент для отображения
кота</string>
```

Приготовления закончены. Осталось добавить фрагмент в активность. Открываем **activity_main.xml** и добавляем новый элемент:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    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=".MainActivity">

    <fragment
        android:id="@+id/cat_fragment"
        android:name="ru.alexanderklimov.fragment.CatFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

В элементе **fragment** в атрибуте **android:name** указываем полное имя класса фрагмента.

Можно запустить проект и увидеть фрагмент внутри активности.

Жизненный цикл фрагмента

У фрагмента есть жизненный цикл, как у активности. Но число методов цикла гораздо больше. Откроем теперь файла класса фрагмента **CatFragment** и добавим новый код (выделено синим).

```
package ru.mdk0103.fragment

import android.os.Bundle
import android.util.Log
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup

// TODO: Rename parameter arguments, choose names that match
// the fragment initialization parameters, e.g. ARG_ITEM_NUMBER
private const val ARG_PARAM1 = "param1"
private const val ARG_PARAM2 = "param2"

/**
 * A simple [Fragment] subclass.
 * Use the [CatFragment.newInstance] factory method to
 * create an instance of this fragment.
 */
class CatFragment : Fragment() {
    // TODO: Rename and change types of parameters
    private var param1: String? = null
    private var param2: String? = null

    override fun onAttach(context: Context) {
        super.onAttach(context)
        Log.d(TAG, "onAttach")
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        Log.d(TAG, "onCreate")
        arguments?.let {
            param1 = it.getString(ARG_PARAM1)
            param2 = it.getString(ARG_PARAM2)
        }
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        Log.d(TAG, "onCreateView")
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_cat, container, false)
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        Log.d(TAG, "onViewCreated")
    }

    override fun onActivityCreated(savedInstanceState: Bundle?) {
        super.onActivityCreated(savedInstanceState)
        Log.d(TAG, "onActivityCreated")
```

```
    }

    override fun onStart() {
        super.onStart()
        Log.d(TAG, "onStart")
    }

    override fun onResume() {
        super.onResume()
        Log.d(TAG, "onResume")
    }

    override fun onPause() {
        super.onPause()
        Log.d(TAG, "onPause")
    }

    override fun onStop() {
        super.onStop()
        Log.d(TAG, "onStop")
    }

    override fun onDestroyView() {
        super.onDestroyView()
        Log.d(TAG, "onDestroyView")
    }

    override fun onDestroy() {
        super.onDestroy()
        Log.d(TAG, "onDestroy")
    }

    override fun onDetach() {
        super.onDetach()
        Log.d(TAG, "onDetach")
    }

    companion object {

        private const val TAG = "CatFragment"

        /**
         * Use this factory method to create a new instance of
         * this fragment using the provided parameters.
         *
         * @param param1 Parameter 1.
         * @param param2 Parameter 2.
         * @return A new instance of fragment CatFragment.
         */
        // TODO: Rename and change types and number of parameters
        @JvmStatic
        fun newInstance(param1: String, param2: String) =
            CatFragment().apply {
                arguments = Bundle().apply {
                    putString(ARG_PARAM1, param1)
                    putString(ARG_PARAM2, param2)
                }
            }
    }
}
```

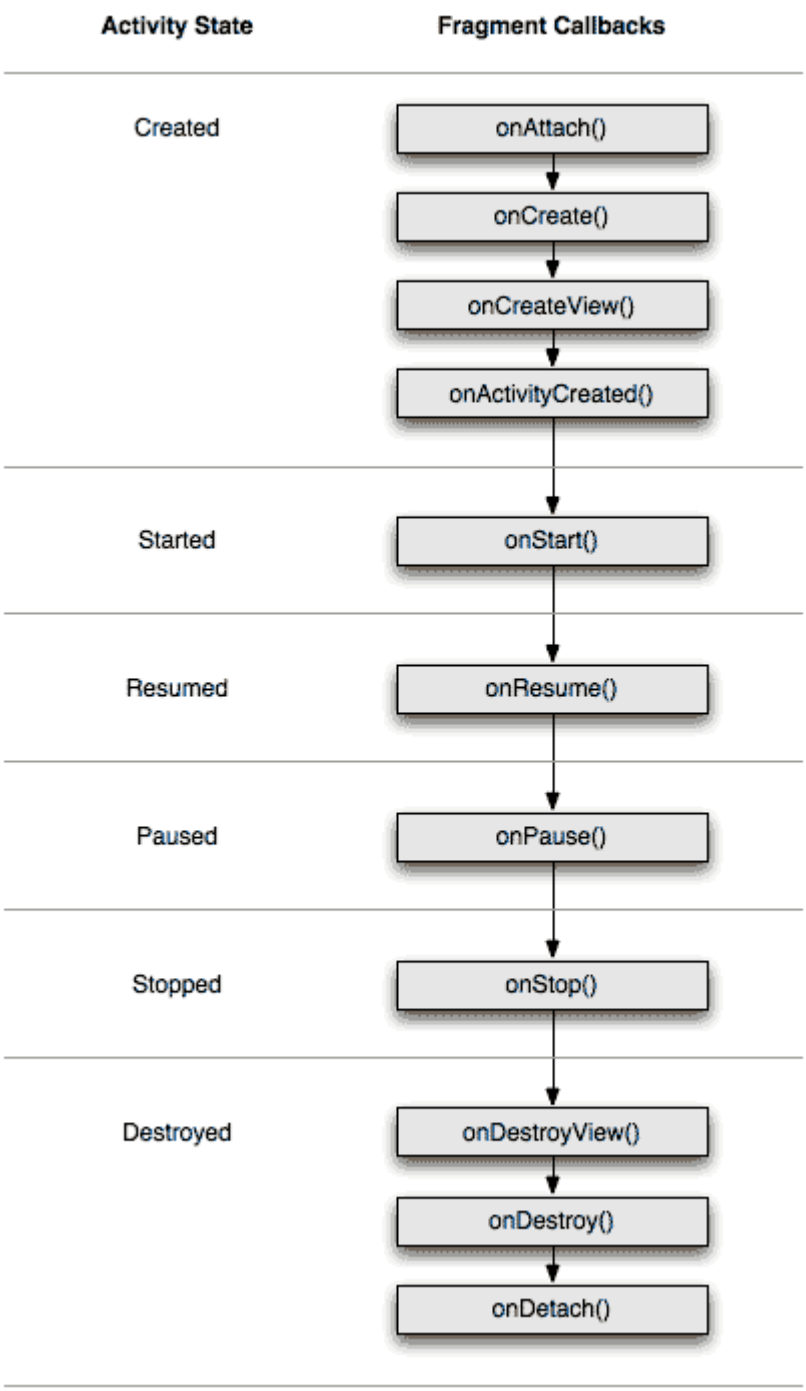
Чтобы увидеть связь между жизненными циклами фрагмента и активности, добавим логи и в активность.

```
package ru.mdk0103.fragment

import android.os.Bundle
import android.util.Log
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {

    companion object {
        private const val TAG = "MainActivity"
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        Log.d(TAG, "onCreate")
    }

    override fun onStart() {
        super.onStart()

        Log.d(TAG, "onStart")
    }

    override fun onResume() {
        super.onResume()

        Log.d(TAG, "onResume")
    }
}
```

Снова запускаем проект и смотрим логи.

```
D/CatFragment: onAttach
D/CatFragment: onCreate
D/CatFragment: onCreateView
D/CatFragment: onViewCreated
D/MainActivity: onCreate
D/CatFragment: onActivityCreated
D/CatFragment: onStart
D/MainActivity: onStart
D/MainActivity: onResume
D/CatFragment: onResume
```

У фрагментов есть жизненный цикл, который во многом совпадает с жизненным циклом активности, внутри которой они находятся.



Два отдельных фрагмента

Мы научились размещать один фрагмент на экране активности. Ничто не мешает разместить на экране несколько фрагментов, которые являются контейнерами для компонентов. Давайте совместим на одном экране два старых примера - **"Hello Kitty"** и **"Счётчик ворон/котов"**.

Повторим шаги создания нового фрагмента через готовый шаблон **Fragment** (**Blank**) и создадим два новых фрагмента **KittyFragment** и **CounterFragment**.

Разметку для **KittyFragment** копируем из старого примера.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#ffffc0cb"
    tools:context=".KittyFragment">

    <TextView
        android:id="@+id/hello_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="@style/TextAppearance.AppCompat.Display2"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/imageButton"
        tools:text="Имя кота" />

    <ImageButton
        android:id="@+id/imageButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="32dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:srcCompat="@drawable/pinkhellokitty" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Код также копируем с небольшим отличием. Если в активности мы размещали основной код в методе **onCreate()**, то для фрагмента используем **onViewCreated()**. Код шаблона для экономии места опустим.

```
package ru.mdk0103.fragment

...

class KittyFragment : Fragment() {
    ...


    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        val helloTextView: TextView = view.findViewById(R.id.hello_text)
        val imageButton: ImageButton = view.findViewById(R.id.imageButton)
        imageButton.setOnClickListener {
            helloTextView.text = "Hello Kitty"
        }
    }

}
```

Для счётчика котов (ворон больше считать не будем) напишем продвинутую версию, которая позволить не только увеличивать показания счётчика, но и уменьшать. В конце концов коты имеют привычку гулять сами по себе и пропадают из поля нашего зрения.

Разметка для фрагмента **counter_fragment.xml**.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".CounterFragment">

    <TextView
        android:id="@+id/counter_label"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Счётчик"
        android:paddingTop="10dp"
        android:textSize="40sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>

    <TextView
        android:id="@+id/counter_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/zero"
        android:textSize="50sp"
        android:textStyle="bold"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/counter_label"
        app:layout_constraintBottom_toTopOf="@id/plus_button"/>

    <Button
        android:id="@+id/plus_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/plus"
        android:paddingStart="30dp"
        android:paddingEnd="30dp"
        android:textSize="40sp"
        android:textStyle="bold"
        app:layout_constraintEnd_toStartOf="@id/minus_button"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/counter_text"
        app:layout_constraintBottom_toBottomOf="parent"/>

    <Button
        android:id="@+id/minus_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/minus"
        android:paddingStart="30dp"
        android:paddingEnd="30dp"
        android:textSize="40sp"
```

```
        android:textStyle="bold"
        app:layout_constraintStart_toEndOf="@id/plus_button"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toBottomOf="@id/counter_text"
        app:layout_constraintBottom_toBottomOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Код для класса **CounterFragment**. Здесь также мы добавляем весь код в **onViewCreated()**.

```
package ru.mdk0103.fragment

class CounterFragment : Fragment() {
    ....

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        val counterText: TextView = view.findViewById(R.id.counter_text)

        view.findViewById<Button>(R.id.plus_button).setOnClickListener {
            var counterValue = counterText.text.toString().toInt()
            counterText.text = (++counterValue).toString()
        }

        view.findViewById<Button>(R.id.minus_button).setOnClickListener {
            var counterValue = counterText.text.toString().toInt()
            if (counterValue > 0) counterText.text = (--
counterValue).toString()
        }
    }
}
```

Осталось разместить два фрагмента в **activity_main.xml**.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <fragment
        android:id="@+id/kitty_fragment"
        android:name="ru.alexanderklimov.as42k.KittyFragment"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toTopOf="@+id/counter_fragment"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <fragment
        android:id="@+id/counter_fragment"
```

```
        android:name="ru.alexanderklimov.as42k.CounterFragment"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/kitty_fragment" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Запускаем проект и видим два приложения в одном. В верхней части мы здороваемся с котёнком, а в нижней считаем котов. Обратите внимание, что весь код сосредоточился в классах фрагментов, а в **MainActivity** нам вообще ничего не пришлось писать. Фрагменты работают независимо и мы можем писать код в разных модулях.