Практическая работа №2 «Работа с Activity и верстка»

1 Верстка в Android. Язык XML

Как правило, для определения ресурсов, а в том числе и визуального интерфейса, в проектах под Android используются специальные файлы xml. Эти файлы являются ресурсами разметки и хранят определение визуального интерфейса в виде кода XML. Подобный подход напоминает создание веб-сайтов, когда интерфейс определяется в файлах html, а логика приложения - в коде javascript.

Объявление пользовательского интерфейса в файлах XML позволяет отделить интерфейс приложения от кода. Что означает, что мы можем изменять определение интерфейса без изменения кода java. Например, в приложении могут быть определены разметки в файлах XML для различных ориентаций монитора, различных размеров устройств, различных языков и т.д. Кроме того, объявление разметки в XML позволяет легче визуализировать структуру интерфейса и облегчает отладку.

Как уже говорилось ранее для описания ресурсов Android-приложений используется язык разметки XML (Extensible Markup Language) — расширяемый язык разметки, описывающий класс объектов, называемых XML-документами. Если открыть любой такой файл в текстовом редакторе, то мы увидим структуру текста, очень похожую на HTML.

Несмотря на внешнюю схожесть, не следует путать XML и HTML. Язык HTML (Hyper Text Markup Language) был предложен Тедом Нельсоном в 1963 году для представления нелинейных текстовых ресурсов. HTML был также языком разметки, то есть с его помощью можно было структурировать текст для его удобной визуализации. HTML и XML используют концепцию тега для разметки текста и поэтому внешне похожи. Однако предназначение их в корне разное. В отличие от HTML, который размечает текст для удобного представления на экране пользователя, XML размечает документ для структурирования представленной в нем информации.

Начнем с того, что XML — это язык свободного описания структур документов. То есть, если необходимо, чтобы в документе присутствовал какой-либо элемент, то мы для него определяем некоторый тег (маркер в тексте). Например, для описания элемента «текстовая строка» можно условиться использовать тег <string>, где первая метка указывает начало описания элемента, а вторая (со знаком /) — конец описания. Между парой тегов помещается текстовое представления содержимого элемента. Для каждого элемента применяется своя пара тегов, при этом однотипные элементы описываются одинаковой парой тегов. Таким образом, для описания двух строк нужны две пары тегов:

```
<string>это первая строка</string>
<string>это вторая строка</string>
```

В открывающем теге можно поместить атрибуты описываемого элемента, такие как цвет, размер, начертание, выравнивание и т. п., то есть описать особенности формируемого элемента. Атрибут — это свойство описываемого элемента. При этом у однотипных элементов полный набор атрибутов будет совпадать, но в описании можно использовать не все свойства. Каждому имени атрибута присваивается значение, записанное в виде текстовой строки, то есть заключенное в двойные кавычки. Разделяются свойства пробелом либо переносом строки. Вернемся к рассматриваемому примеру:

```
<string color = “red” align = “center”>это первая строка</string>
```

Данная разметка описывает текстовую строку, написанную красным шрифтом (начертание и размер установлены по умолчанию, поскольку эти свойства не указаны при описании) с выравниванием в центре страницы.

Каким бы свободным не был стиль XML-документа, все-таки существуют правила его формирования.

- В языке XML все теги парные. Это значит, что у каждого открывающего тега обязательно должен присутствовать закрывающий тег. Это правило позволяет описывать вложенные элементы, то есть помещать внутри одного элемента другие. Если тело тега пусто, то два тега записываются в один, который завершается косой чертой;
- Документ может содержать декларацию — строку заголовка, в которой указывается версия языка и используемая текстовая кодировка;
- Имена тегов могут содержать буквы, цифры и специальные знаки, такие как знак подчеркивания (_), но должны начинаться с буквы. Теги записываются с соблюдением регистра, поскольку XML регистрозависим;
- Если возникает необходимость использования одинаковых имен элементов для разного типа структур документа, применяют понятие пространства имен. Чтобы различать такие элементы, необходимо задать соответствие — специальный уникальный идентификатор ресурса или URI с конкретным именем элемента. В качестве идентификатора чаще всего используется адрес своего (необязательно реально существующий) ресурса. Пространство имен определяется благодаря атрибуту xmlns в начальном теге элемента;

- В XML-тексте комментарии выделяются тегами <!-- -->;

Элемент — это структурная единица XML-документа. Границы элементов маркируются одинаковыми начальным и конечным тегами. Внутри этой границы может быть текстовая строка значения элемента. Элемент может быть также представлен пустым тегом, то есть не включающим в себя другие элементы и/или символьные данные. Например, заключая строку 2017 в пару тегов <year> и </year>, мы определяем непустой элемент, называемый year, содержимым которого является значение 2017. В общем случае в качестве содержимого элементов могут выступать как просто какой-то текст, так и другие, вложенные, элементы документа, секции CDATA, комментарии — практически любые части XML- документа.

Помимо текстового значения элемент может включать другие элементы. Такие элементы называются дочерними (child) элементами. Дочерних элементов может быть несколько. Элемент, который окружает дочерний элемент, называется родительским (parent). По естественным причинам у дочернего элемента может быть только один родительский. Важно, чтобы любой дочерний элемент располагался целиком внутри родительского. То есть пары открывающих и закрывающих тегов всех дочерних элементов должны быть заключены (окружены) парой открывающего и закрывающего тегов родительского элемента. В случае нарушения этого правила любая программа не сможет прочитать ваш документ и выдаст сообщение об ошибочности. Автор документа, вкладывая одни элементы в другие, задает иерархическую структуру внутри документа.

2 Ресурсы в Android

Создавая приложение для Android, помимо написания программ на языке Java необходимо также работать с ресурсами. В экосистеме Android принято отделять такие файлы, как изображения, музыка, анимации, стили, макеты окон, строковые константы — в общем все части оформления GUI (Graphical User Interface — графический интерфейс пользователя) от программного кода. Большая часть ресурсов (за исключением мультимедийных) хранятся во внешних XML-файлах. При создании и развитии программного проекта внешние ресурсы легче поддерживать, обновлять и редактировать.

Как уже было показано, каждое приложение на Android содержит каталог для ресурсов res/. Доступ к информации в каталоге ресурсов из приложения осуществляется через класс R, который автоматически генерируется средой разработки.

В общем случае ресурсы представляют собой файл (например, изображение) или значение (например, заголовок программы), связанные с создаваемым приложением по имени ресурса. Удобство использования ресурсов заключается в том, что их можно заменять/изменять без изменения программного кода приложения или компиляции. Поскольку имена файлов для ресурсов фактически будут использованы как имена констант в R, то они должны удовлетворять правилам написания имен переменных в Java. Так как разработка ведется на различных ОС (Windows, Mac, Linux), то также есть еще ограничения. В итоге имена файлов должны состоять исключительно из букв в нижнем регистре, чисел и символов подчеркивания.

В Android используются два подхода к процессу создания ресурсов — первый подход заключается в том, что ресурсы задаются внутри файла и тогда его имя задается в месте его описания. Второй подход — ресурс задается в виде самого файла, и тогда имя файла уже и есть имя ресурса.

Для различных типов ресурсов, определенных в проекте, в каталоге res создаются подкаталоги. Поддерживаемые подкаталоги:

- animator/: xml-файлы, определяющие анимацию свойств
- anim/: xml-файлы, определяющие tween-анимацию
- color/: xml-файлы, определяющие список цветов
- drawable/: Графические файлы (.png, .jpg, .gif)
- dimensions/: xml-файлы, определяющие размерности элементов
- mipmap/: Графические файлы, используемые для иконок приложения под различные разрешения экранов

- layout/: xml-файлы, определяющие пользовательский интерфейс приложения
- menu/: xml-файлы, определяющие меню приложения
- raw/: различные файлы, которые сохраняются в исходном виде
- values/: xml-файлы, которые содержат различные используемые в приложении значения, например, ресурсы строк
- xml/: Произвольные xml-файлы
- font/: файлы с определениями шрифтом и расширениями .ttf, .otf или .ttc, либо файлы XML, который содержат элемент <font-family>

Чаще других используют следующие ресурсы: разметка (layout), строки (string), цвета (color) и графические рисунки (bitmap, drawable).

2.1 Строковые ресурсы в Android

Любой используемый в программе текст — это отдельный ресурс, такой же, как изображение или звук.

При создании нового приложения среда разработки (например, Android Studio) создает файл *strings.xml*, в котором хранятся строки для заголовка приложения и выводимого им (приложением) сообщения. Можно редактировать данный файл, добавляя новые строковые ресурсы. Также можно создавать новые файлы с любым именем, которые будут содержать строковые ресурсы. Все эти файлы должны находиться в подкаталоге */res/values*. Число файлов может быть любым. Строковые ресурсы обозначаются тегом. Типичный файл выглядит следующим образом:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name"> Hello world!</string>
    <string name="hello_world">Hello world!</string>
</resources>
```

Где name — имя строкового ресурса, с помощью которого можно отличить один ресурс от другого, а также обратиться к данному ресурсу в программе или в другом файле xml.

Подход, при котором строковые ресурсы хранятся в отдельном файле, удобен по двум причинам.

- Если одна и та же строка используется в нескольких частях программы, то благодаря использованию ссылки на текстовый ресурс в дальнейшем правку

(если, например, захотели изменить текст какого-то заголовка) нужно будет делать только в одном исходном файле ресурса.

- Создание отдельных файлов с текстовыми значениями удобно и для локализации приложения на несколько языков. Все, что нужно будет сделать, — это создать отдельный файл с переводом, например, на французский язык, и создать для него отдельную директорию res/values-fr (fr — сокращение для French).

При именовании строковых ресурсов необходимо придерживаться нескольких правил:

- название строкового ресурса должно состоять из строчных букв;
- в случае, если название состоит из двух или трех слов, то их рекомендуется разделять нижним подчеркиванием;
- в названиях рекомендуется использовать только латиницу.

3 Создание интерфейса пользователя при помощи ресурсов

Графический интерфейс создается с помощью представлений (View) и групп представлений (ViewGroup). Эти элементы размещаются на активности, их описания помещаются в файл манифеста, а действия с объектами прописываются программно в файле кода MainActivity.java в виде методов классов, наследуемых от классов View и ViewGroup или атрибутивно в файле разметки *layout/activity_main.xml*. У файла разметки также имеется графический вид Graphical layout — системная имитация мобильного устройства.

В окне Palette можно выбрать вид представления объектов: значки, названия, названия и значки. Выберите удобный для себя вид для работы с графическими представлениями.

В файле определяются все графические элементы и их атрибуты, которые составляют интерфейс. При создании разметки в XML следует соблюдать некоторые правила: каждый файл разметки должен содержать один корневой элемент, который должен представлять объект View или ViewGroup.

По умолчанию при создании проекта с пустой activity уже есть один файл ресурсов разметки activity_main.xml. В нем корневым элементом является элемент ConstraintLayout, который содержит элемент TextView. Как правило, корневой элемент содержит определение используемых пространств имен XML.

3.1 Контейнеры разметок

Layout — это разметка окна. Разметка, может быть, нескольких типов. Наиболее часто используемые:

- LinearLayout — линейная разметка, при которой элементы GUI (Views) размещаются вертикально или горизонтально. Расположение объектов в данной разметке осуществляется строго друг под другом в один столбец, если ориентация разметки вертикальная (android:orientation=«vertical»), или в одну строку рядом, если ориентация горизонтальная (android:orientation=«horizontal»). Линейная разметка позволяет объектам пропорционально заполнять пространство (android:layout_weight).
- FrameLayout — разметка в виде фреймов, когда все объекты выравниваются по левому верхнему углу экрана. Очень неудобна в качестве корневой, но можно применять как вложенную для размещения в ней одного объекта и резервирования свободного пространства рядом с ним.

- TableLayout — табулированная разметка, позволяющая располагать представления по строкам и столбцам. Каждый столбец таблицы в xml-файле ресурсов выделяется парным тегом <TableRow></TableRow>. В данной разметке во всех строках выделяется одинаковое количество столбцов. Если в строках было описано разное количество объектов, то длина строки измеряется по максимальному количеству элементов в TableRow. В строках меньшей длины крайние правые ячейки остаются свободными. Для того чтобы оставить пустой не крайнюю правую ячейку, в нее помещают пустой объект, например, TextView без текста и форматирования.
- RelativeLayout — относительная разметка, при которой позиции объектов можно определять относительно родительского элемента, а также относительно друг друга.
- ConstraintLayout — представляет контейнер, который позволяет создавать гибкие и масштабируемые визуальные интерфейсы. Для позиционирования элемента внутри ConstraintLayout необходимо указать ограничения (constraints). Есть несколько типов ограничений.

Для всех видов разметок обязательно имеются атрибуты высоты и ширины, при этом каждая разметка имеет свой набор атрибутов. Свойства описываются внутри парных тегов представления. Атрибут описывается пространством имен «android:» и названием свойства с присвоением ему значения: android:property = «property_value». Описание свойств можно прочитать, выделив соответствующую строку в ниспадающем списке.

3.2 Компоненты разметки

В отличии от контейнеров, компоненты разметки имеют конкретный внешний вид и конкретные задачи. В SDK Android находятся множество различных компонентов, однако будет рассмотрен основной перечень из них:

- TextView – компонент, предназначенный для простого вывода текста на экран. Он просто отображает текст без возможности его редактирования.
- EditText – является подклассом класса TextView. Он также представляет текстовое поле, но теперь уже с возможностью ввода и редактирования текста. Таким образом, в EditText мы можем использовать все те же возможности, что и в TextView.
- Button – один из часто используемых компонентов. Ключевой особенностью кнопок является возможность взаимодействия с пользователем через нажатия.

- ImageView – является базовым элементом-контейнером для использования графики. Можно загружать изображения из разных источников, например, из ресурсов программы, контент-провайдеров.

4 Взаимодействие программного кода с верстской пользовательского интерфейса из ресурсов

Для корректной работы с созданным пользовательским интерфейсом необходимо произвести подключение файла пользовательского интерфейса в файл программного кода java. Для этого необходимо вызвать команду:

```
setContentView(R.layout.activity_main);
```

Для получения ссылки на ресурс в коде java необходимо использовать выражение R.layout.[название_ресурса]. Название ресурса layout будет совпадать с именем файла, поэтому чтобы использовать файл activity_main.xml в качестве источника визуального интерфейса, необходимо использовать вышеприведенный пример.

Однако этого может не хватить для корректной и удобной работы с созданным пользовательским интерфейсом. К примеру, при необходимости поменять значение в текстовом компоненте из кода необходимо получить доступ к нему. В программном коде доступ к компоненту осуществляется посредством его id, установленного атрибутом android:id, поэтому данный атрибут необходимо устанавливать всем создаваемым представлениям, к которым планируется обращение. Далее в программном коде необходимо использовать специализированную команду для получения доступа к компоненту или контейнеру подключенной разметки.

В файле верстки:

```
android:id = “@+id/name”
```

В коде программы:

```
Type_object var_name =
(Type_object)findViewById(R.id.name);
```

Здесь инструкция findViewById() находит компонент по его id, считывая адрес из класса R. Этот метод возвращает объект View – объект базового класса для всех элементов, поэтому результат метода еще необходимо привести к типу TextView. Причем важно, что получение элемента происходит после того, как в методе setContentView была установлена разметка, в которой этот визуальный элемент был определен, так как иначе доступа к этому элементу не будет.

Одной из часто реализуемых задач в привязке графического интерфейса к программному коду – это реализация логики при нажатии на кнопку. Сам по себе объект Button ничего не делает, то есть при нажатии на кнопку ничего не происходит. Чтобы

действия выполнялись, необходим обработчик события нажатия кнопки. Задать метод обработчик события можно двумя способами:

- декларативно, при помощи атрибута кнопки onClick;
- программно, в коде вашего приложения используя метод, устанавливающий обработчик событий для кнопки setOnClickListener().

Программный метод является более традиционным для разработки. Для того, чтобы создать прослушивателя нажатия на кнопку, в первую очередь необходимо получить доступ к компоненту кнопки на пользовательском интерфейсе. После чего, необходимо создать нового слушателя действий компонентов и подключить к полученной кнопке. Для этого необходимо создать объект класса OnClickListener и переназначить в нем абстрактный метод onClick. После того, как объект класса получен, его необходимо передать в качестве параметра ранее изученному методу setOnClickListener. Пример такой реализации приведен ниже.

```
View.OnClickListener listener=new View.OnClickListener() {
        @Override
        public void onClick(View v){
            // Какая-то логика
        }
    };
```

5 ViewBinding

Для взаимодействия с компонентами в java-классе есть альтернативные способы. Одним из таких способов является ViewBinding. Для активации данного функционала в build.gradle файле проекта необходимо добавить следующие строки:

```
    android {
        ...
        buildFeatures {
            viewBinding true
        }
    }
```

После этого необходимо синхронизировать проект, что подключит заданный функционал. Теперь для каждого созданного xml-файла будет генерироваться аналогичный java-class с постфиксом Binding. Например, для файла разметки activity_main.xml

```
    <LinearLayout ... >
        <ImageView android:id="@+id/avatar" .../>
        <Button android:id="@+id/button" .../>
        <TextView android:text="Изучаем ViewBinding" />
    </LinearLayout>
```

Будет сгенерирован класс ActivityMainBinding.class, который будет содержать следующие неизменяемые поля виджетов:

- ImageView – avatar;
- Button – button;
- A TextView с текстом “Изучаем ViewBinding” не будет создано, поскольку в XML файле для этого виджета отсутствует id.

Помимо полей данный класс всегда содержит метод getRoot(), который позволяет получить корневой элемент XML-разметки (в данном примере это LinearLayout).

Чтобы создать экземпляр класса ActivityMainBinding у нас есть 3 статические функции. Рассмотрим каждую на различных примерах.

5.1 Использование в Activity

Для создания экземпляра используется статическая функция inflate(LayoutInflater inflater).

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ActivityMainBinding binding = ActivityMainBinding.inflate(
                                    getLayoutInflater()
                                );
    setContentView(binding.getRoot());

    binding.button.setOnClickListener(...);
}
```

Важно отметить то, что вместо R.layout.activity_main мы используем корневой компонент для инициализации контента у Activity.

5.2 Программная вставка компонента из XML

Существует необходимость динамически наполнить контейнер, например, если речь идёт о добавлении тегов. Пусть тег существует в XML файле item_tag.xml. Для динамического создания объекта и автоматического добавления её в контейнер мы можем использовать перегруженную функцию inflate(LayoutInflater inflater, ViewGroup container, boolean attachToParent), где:

- container – это группа, в которую будет добавлен созданный элемент;
- attachToParent – присоединить ли компонент к контейнеру. Если значение true – то компонент будет сразу отображён на экране, иначе необходимо дополнительно вызвать функцию у контейнера addView.

```
void addItem(LinearLayout container) {
    ItemTagBinding itemBinding = ItemTagBinding.inflate(getLayoutInflater(),
container, true);
    itemBinding.textView.setText(...);
}
```

5.3 Получение экземпляра *Binding из View

В дальнейшем будут ситуации, когда на вход будет приходить View, внутри которой необходимо проинициализировать различные элементы. Для создания экземпляра *Binding можно использовать метод bind(View view).

6 Создание второго Activity.

Для создания второй Activity необходимо нажать правой кнопкой мыши в папку, в которой находится класс MainActivity, и затем в контекстном меню выбрать New->Activity->Empty Activity.

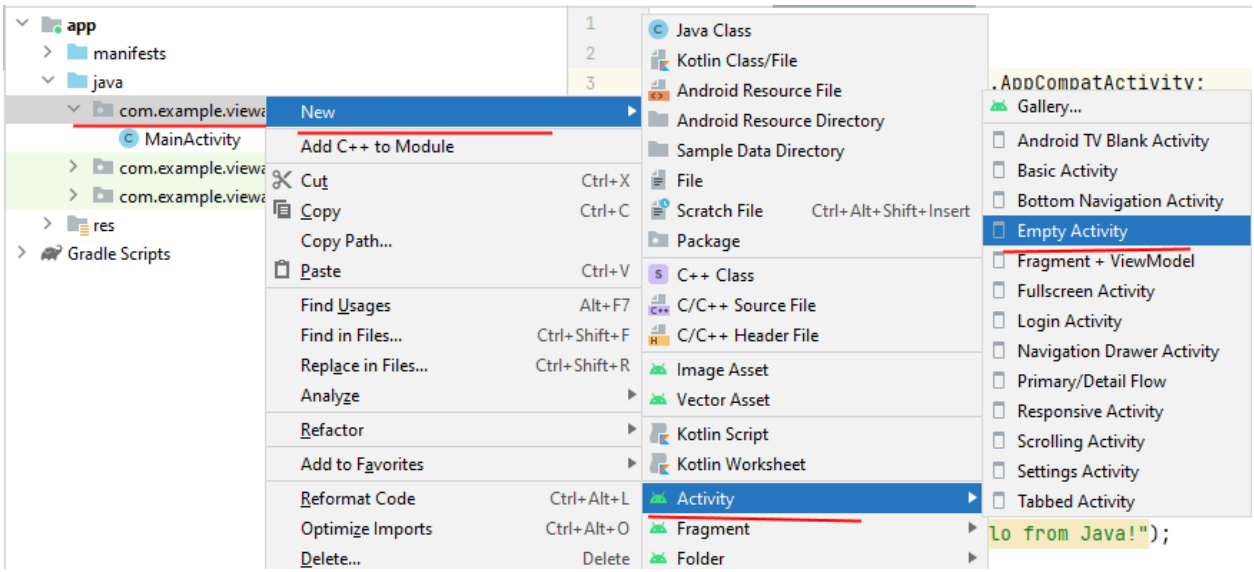

Рисунок 1 – создание дополнительной Activity

Новый класс Activity назовем SecondActivity, а все остальные настройки оставим по умолчанию:

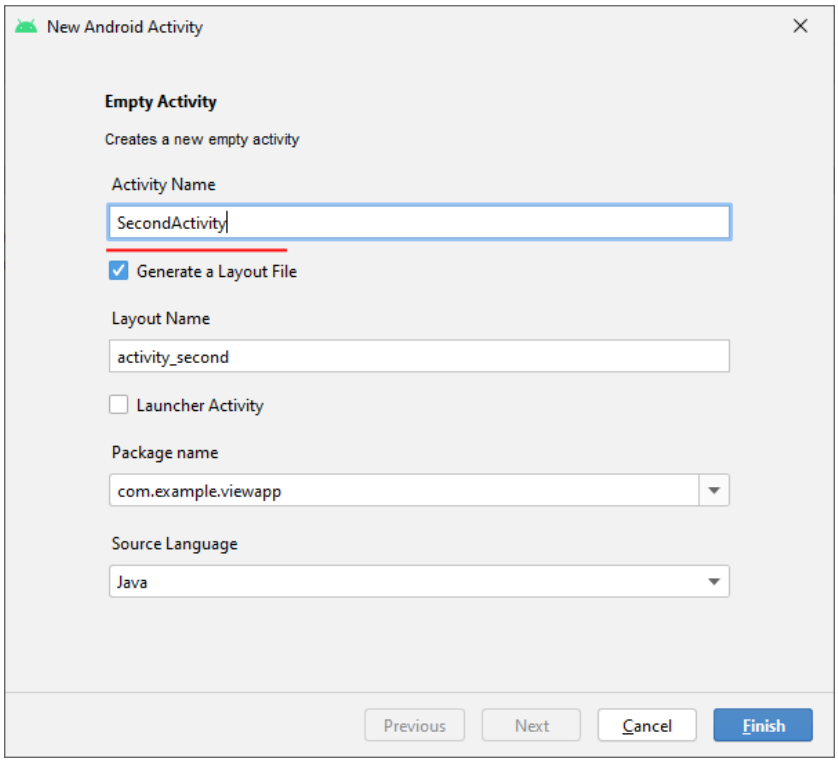

Рисунок 2 – название дополнительной Activity

И после этого в проект будет добавлена новая Activity - SecondActivity:

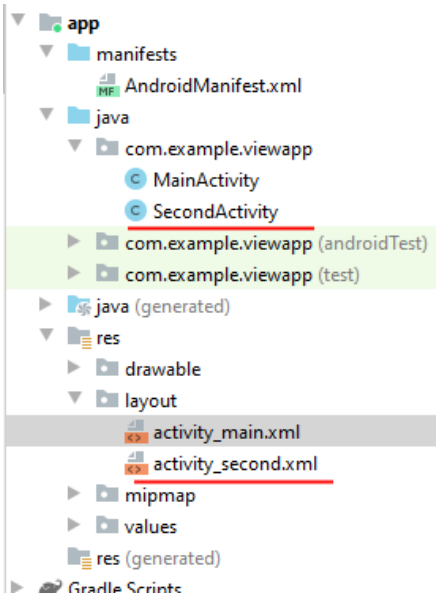

Рисунок 3 – дерево каталогов

После этого в файле манифеста AndroidManifest.xml появится возможность найти следующие строки:

```
    <activity android:name=".SecondActivity"></activity>
    <activity
        android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
```

Все используемые классы activity должны быть описаны в файле AndroidManifest.xml с помощью элемента <activity>. Каждый подобный элемент содержит как минимум один атрибут android:name, который устанавливает имя класса activity.

Однако, по сути, activity – это стандартный класс java, который наследуется от класса Activity или его наследников. Поэтому вместо встроенных шаблонов в Android Studio можно добавлять обычные классы, и затем их наследовать от класса Activity. Однако в этом случае нужно будет вручную добавлять в файл манифеста данные об activity.

Причем для MainActivity в элементе intent-filter определяется интент-фильтр. В нем элемент action значение "android.intent.action.MAIN" представляет главную точку входа в приложение. То есть MainActivity остается основной и запускается приложением по умолчанию.

Для SecondActivity просто указано, что она в проекте, и никаких intent-фильтров для нее не задано.

7 Переход между Activity. Передача данных между Activity.

Чтобы из MainActivity запустить SecondActivity, надо вызвать метод startActivity():

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

В качестве параметра в метод startActivity передается объект Intent. Для своего создания Intent в конструкторе принимает два параметра: контекст выполнения (в данном случае это текущий объект MainActivity) и класс, который используется объектом Intent и представляет передаваемые в задачу данные (фактически класс activity, которую мы будем запускать).

Теперь рассмотрим реализацию перехода от одной Activity к другой. Для этого в файле activity_main.xml (то есть в интерфейсе для MainActivity) определим кнопку:

```
<?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"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/navButton"
        android:textSize="20sp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Перейти к SecondActivity"
        android:onClick="onClick"

        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>

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

И определим для кнопки в классе MainActivity обработчик нажатия, по которому будет производиться переход к новой Activity:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void onClick(View view) {
        Intent intent = new Intent(this, SecondActivity.class);
        startActivity(intent);
    }
}
```

В обработчике нажатия будет запускаться SecondActivity.

Далее изменим код SecondActivity:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.TextView;

public class SecondActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.activity_second);
        TextView textView = new TextView(this);
        textView.setTextSize(20);
        textView.setPadding(16, 16, 16, 16);
        textView.setText("SecondActivity");
        setContentView(textView);
    }
}
```

Запустим приложение и перейдем от одной Activity к другой:

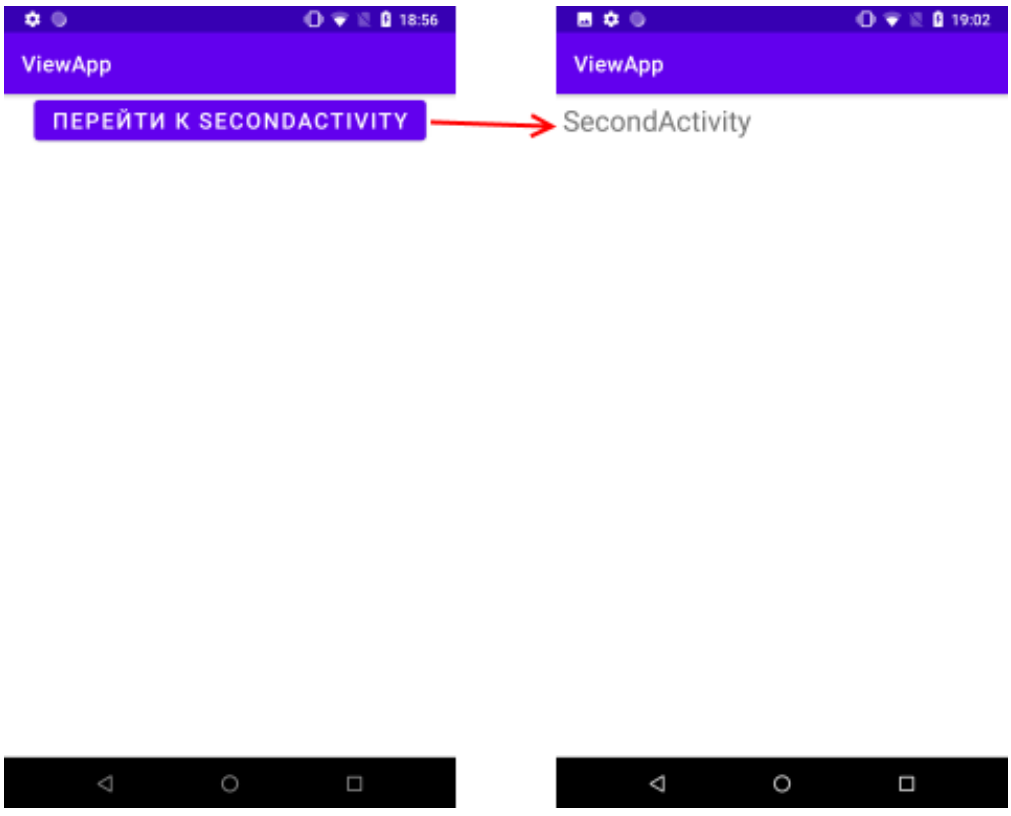

Рисунок 4 - результат

Однако простой запуск других activity может быть недостаточен для разработки целых приложений и в определенный момент может понадобиться передать некоторые данные как в открываемое activity, так и обратно.

7.1 Передача данных в открываемый activity

Для передачи данных между двумя Activity используется объект Intent. Через его метод putExtra() можно добавить ключ и связанное с ним значение.

Например, передача из текущей activity в SecondActivity строки "Hello World" с ключом "hello":

```
    // создание объекта Intent для запуска SecondActivity
    Intent intent = new Intent(this, SecondActivity.class);
    // передача объекта с ключом "hello" и значением "Hello World"
    intent.putExtra("hello", "Hello World");
    // запуск SecondActivity
    startActivity(intent);
```

Для передачи данных применяется метод putExtra(), который в качестве значения позволяет передать данные простейших типов - String, int, float, double, long, short, byte, char, массивы этих типов, либо объект интерфейса Serializable.

Чтобы получить отправленные данные при загрузке SecondActivity, можно воспользоваться методом get(), в который передается ключ объекта:

```
    Bundle arguments = getIntent().getExtras();
    String name = arguments.get("hello").toString();    // Hello World
```

В зависимости от типа отправляемых данных при их получении мы можем использовать ряд методов объекта Bundle. Все они в качестве параметра принимают ключ объекта.

7.2 Передача данных в вызвавший открытие activity

Для получения же результата работы запускаемой activity необходимо использовать Activity Result API. Activity Result API предоставляет компоненты для регистрации, запуска и обработки результата другой Activity. Одним из преимуществ применения Activity Result API является то, что он отвязывает результат Activity от самой Activity. Это позволяет получить и обработать результат, даже если Activity, которая возвращает результат, в силу ограничений памяти или в силу других причин завершила свою работу. Вкратце рассмотрим основные моменты применения Activity Result API.

Для регистрации функции, которая будет обрабатывать результат, Activity Result API предоставляет метод registerForActivityResult(). Этот метод в качестве параметров принимает объекты ActivityResultContract и ActivityResultCallback и возвращает объект ActivityResultLauncher, который применяется для запуска другой activity.

```
ActivityResultLauncher<I> registerForActivityResult (
                ActivityResultContract<I, O> contract,
                ActivityResultCallback<O> callback)
```

ActivityResultContract определяет контракт: данные какого типа будут подаваться на вход и какой тип будет представлять результат.

ActivityResultCallback представляет интерфейс с единственным методом onActivityResult(), который определяет обработку полученного результата. Когда вторая activity закончит работу и возвратит результат, то будет как раз вызываться этот метод. Результат передается в метод в качестве параметра. При этом тип параметра должен соответствовать типу результата, определенного в ActivityResultContract. Например:

```
ActivityResultLauncher<Intent> mStartForResult = registerForActivityResult(
    new ActivityResultContracts.StartActivityForResult(),
    new ActivityResultCallback<ActivityResult>() {
        @Override
        public void onActivityResult(ActivityResult result) {

            // обработка result
        }
});
```

Класс ActivityResultContracts предоставляет ряд встроенных типов контрактов. Например, в листинге кода выше применяется встроенный тип ActivityResultContracts.StartActivityForResult, который в качестве входного объекта устанавливает объект Intent, а в качестве типа результата - тип ActivityResult.

Метод registerForActivityResult() регистрирует функцию-колбек и возвращает объект ActivityResultLauncher. С помощью этого мы можем запустить activity. Для этого у объекта ActivityResultLauncher вызывается метод launch():

```
    mStartForResult.launch(intent);
```

В метод lauch() передается объект того типа, который определен объектом ActivityResultContracts в качестве входного.

Для возврата результата из второго activity необходимо вызвать метод setResult(), в который передается два параметра:

- числовой код результата
- отправляемые данные

После вызова метода setResult() нужно вызвать метод finish, который уничтожит текущую activity.

Задание:

1. Создать верстку приложения на основе предметной области.
    1.1 Реализовать несколько файлов разметки с применением следующих контейнеров: LinerLayout, RelativeLayout, Constraint Layout. (При желании можно расширить перечень собственными контейнерами);
    1.2 Добавить в созданные файлы разметки базовые компоненты: Текст, Кнопка, Поле ввода, Картинка. (При желании можно расширить перечень собственными компонентами);
2. Работа с ресурсами.
    2.1 Добавить в проект несколько строковых, размерных, цветовых и drawable ресурсов.
    2.2 Добавить в проект несколько изображений png и svg.
    2.3 Подключить созданные ресурсы в подготовленную на прошлом шаге верстку.
3. Произвести инициализацию нескольких компонентов в верстке данными из программного кода. Произвести инициализацию текстового компонента строковым ресурсом и компонента картинки – ресурсом картинки.
4. Произвести вывод сообщение в Log при нажатии на клавишу. Задать метод обработчик события двумя способами: декларативно и программно.
5. (Дополнительно) Подключить ViewBinding. Реализовать пункты 3 и 4 посредством работы с ViewBinding.
6. Создать второе Activity на основе предметной области. Произвести открытие второй Activity с передачей в нее данных, обоснованных предметной областью. При закрытии второй Activity, вернуть данные в изночальную, обоснованные предметной областью.