Часть 1. Передача сложных объектов между активностями

В предыдущей практике передавались простые данные – числа, строки. Но также можно передавать более сложные данные.

Передача сложных объектов между активностями в Android осуществляется с помощью механизма сериализации и передачи данных через Intent.

Сериализация — это процесс преобразования объекта в последовательность байтов, которая может быть сохранена в памяти устройства или передана по сети, а затем восстановлена обратно в объект.

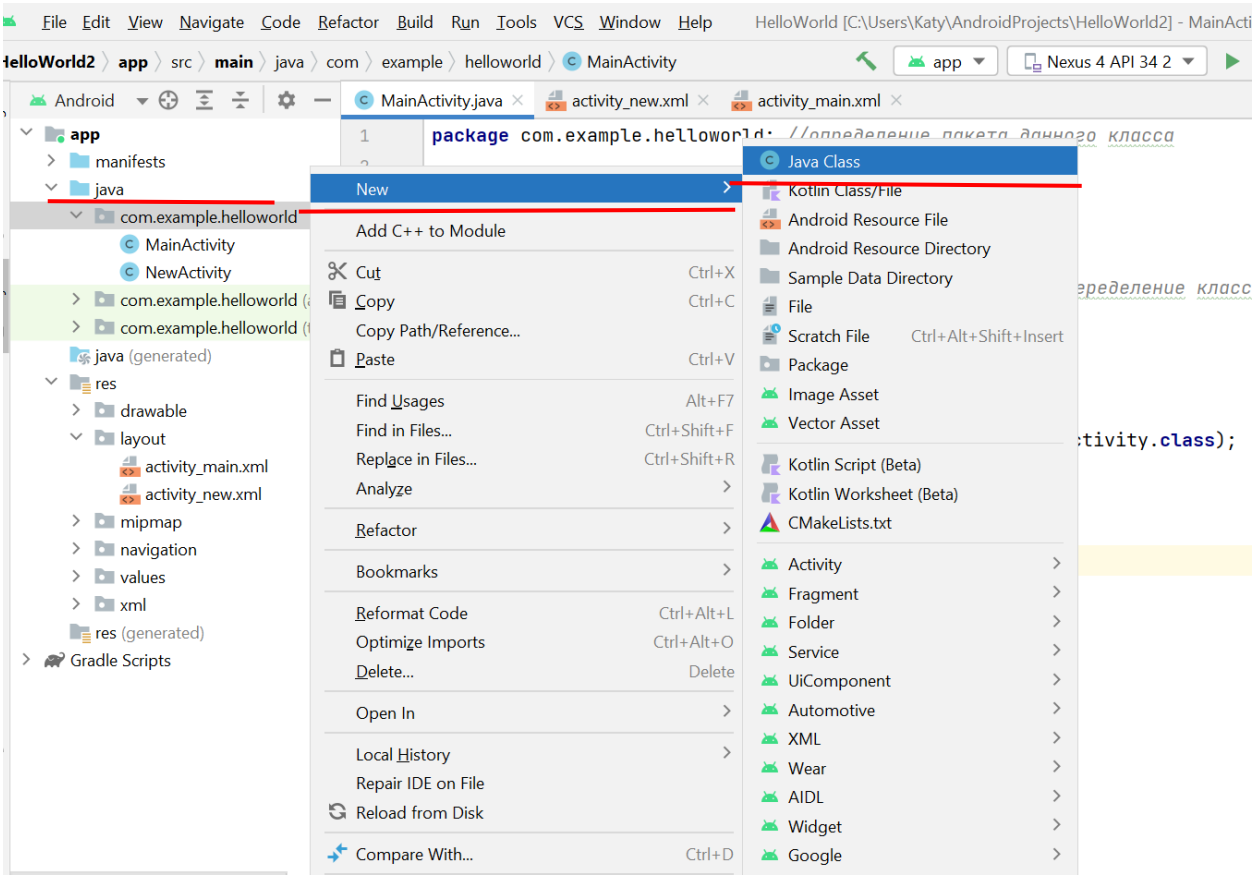
Для передачи сложных объектов между активностями необходимо, чтобы класс объекта имплементировал интерфейс **Serializable** или **Parcelable**. Оба интерфейса предоставляют способы сериализации объекта, но Parcelable часто используется в Android, так как он более эффективен в производительности.

Процесс передачи объекта между активностями включает следующие шаги:

1. Создание объекта для передачи: Сначала необходимо создать экземпляр класса, который вы хотите передать между активностями.
2. Имплементация интерфейса Parcelable (или Serializable): Если класс объекта не имплементирует Parcelable, то необходимо добавить соответствующие методы чтения и записи для сериализации и десериализации данных объекта.
3. Упаковка объекта в Intent: Для передачи объекта между активностями необходимо создать новый Intent и использовать его методы putExtra() или putParcelableExtra(), чтобы добавить объект в Intent.

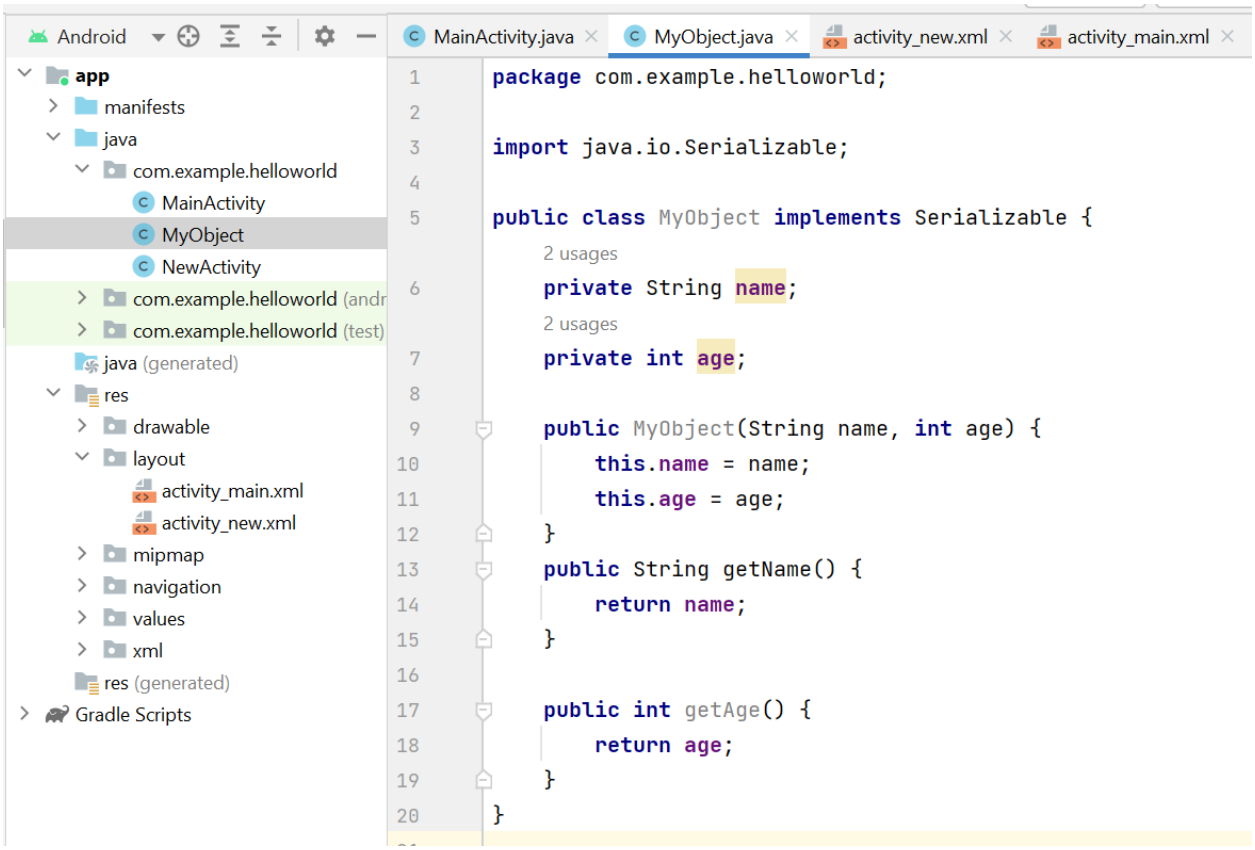
Сначала создадим класс, который будем передавать. Например, MyObject.

Для создания класса нажимает правой кнопкой мыши на **"java"→ "New" → "Java Class"**. После выбора вводим название класса.



После создания класса необходимо описать его. Для этого заходит в класс MyObject, объявляем внутри него переменные и необходимые методы.

```
import java.io.Serializable;

public class MyObject implements Serializable {
    private String name;
    private int age;

    public MyObject(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
}
```



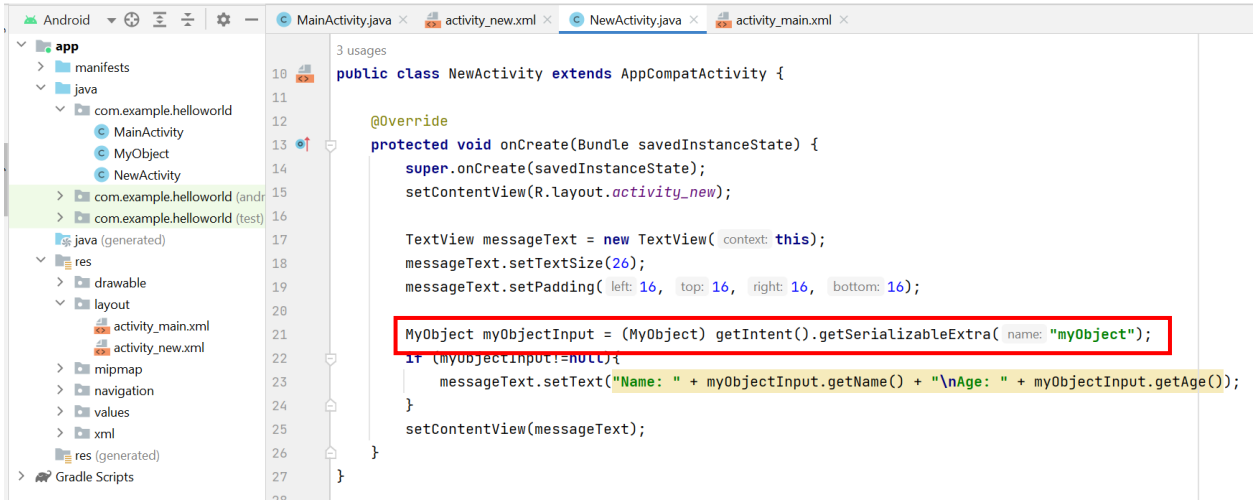
Затем создадим объект этого класса в активности.

```
MyObject myObject = new MyObject("John", 30);
Intent intent = new Intent(this, TargetActivity.class);
intent.putExtra("myObject", myObject);
startActivity(intent);
```

Для наглядности используем пример из прошлой практической работы и будем передавать те же данные, которые вводили на первой Activity. В таком случае передаем конкретные переменные.

После чего необходимо извлечь объект из целевой активности.

```
MyObject myObject = (MyObject) getIntent().getSerializableExtra("myObject");
```



Таким образом, осуществляется передача сложных объектов между разными активностями.

Часть 2. BackStack

BackStack в Android — это концепция, используемая для управления навигацией между различными экранами в приложении. Этот механизм можно представить как стек, где каждый элемент представляет собой активность или фрагмент, к которому пользователь может вернуться, нажав кнопку «Назад». Например, когда пользователь открывает новую активность, она помещается в вершину стека. При нажатии кнопки «Назад» текущая активность удаляется из стека,

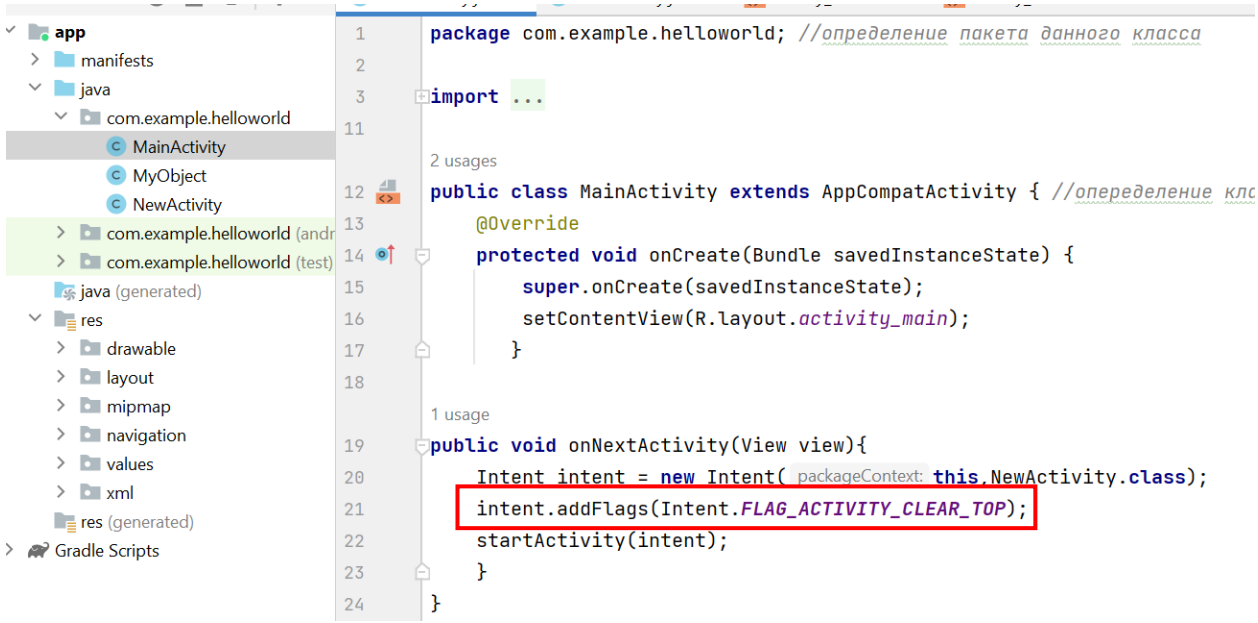
и пользователь возвращается к предыдущей активности, которая теперь находится на вершине стека.

Android предоставляется возможность программно манипулировать BackStack, что позволяет настраивать поведение навигации в своих приложениях. Так можно программно удалить определённые активности из стека или очистить весь стек, чтобы предотвратить возвращение пользователя к предыдущим экранам.

Предположим, что в NewActivity пользователь выполняет некоторые действия, и вы хотите, чтобы при возвращении в MainActivity, NewActivity была удалена из BackStack. В этом случае вы можете использовать флаг Intent.FLAG_ACTIVITY_CLEAR_TOP. Вот как это выглядит:

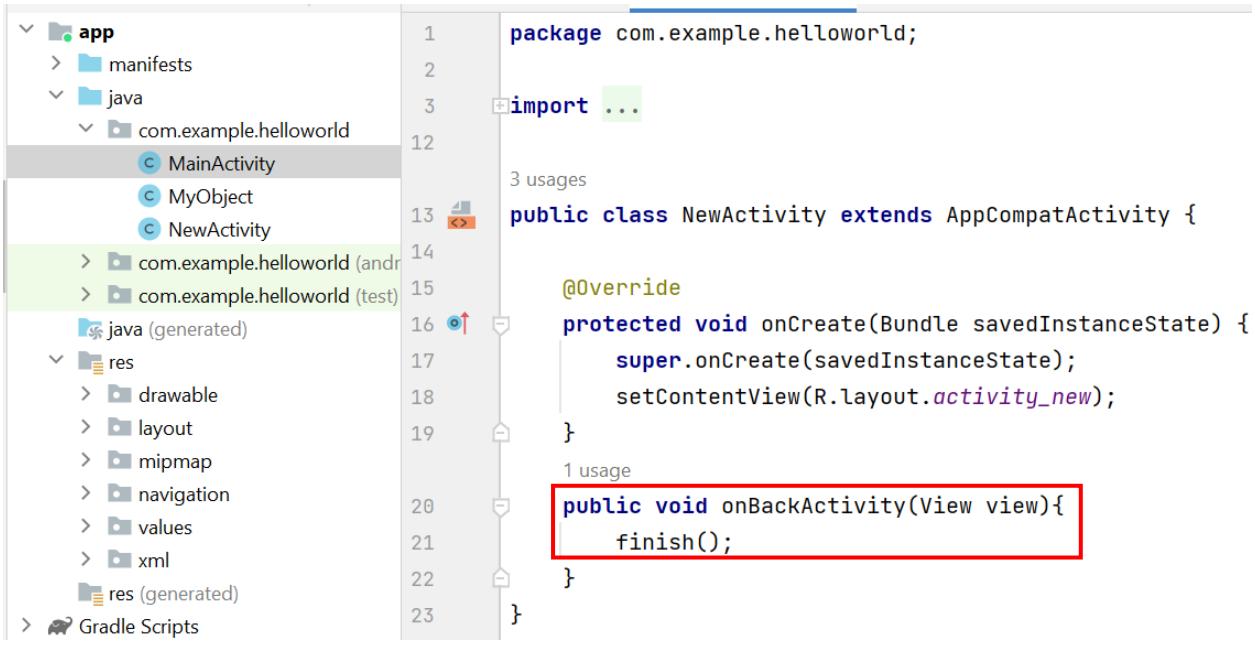
```
Intent intent = new Intent(NewActivity.this, MainActivity.class);
intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
startActivity(intent);
```

Добавление флага FLAG_ACTIVITY_CLEAR_TOP будет указывать системе, что при переходе к MainActivity, все активности, находящиеся над MainActivity в стеке, должны быть удалены. Таким образом, NewActivity будет удалена из стека, и при нажатии кнопки «Назад» в MainActivity, пользователь выйдет из приложения, а не вернется к NewActivity.



С помощью метода **finish()** можно завершить работу активности. Если приложение состоит из одной активности, то этого делать не следует, так как система сама завершит работу приложения. Если же приложение содержит несколько

активностей, между которыми нужно переключаться, то данный метод позволяет экономить ресурсы.



Часть 3. Контейнеры

Контейнер компоновки в Android — это специализированный объект, который используется для управления расположением элементов пользовательского интерфейса (таких как кнопки, текстовые поля, изображения и другие элементы) на экране устройства. Эти контейнеры обеспечивают структуру и организацию элементов пользовательского интерфейса в приложении Android.

В основе работы контейнера компоновки лежит концепция управления размещением и размером дочерних элементов в соответствии с определенными правилами. Контейнеры компоновки определяют, как элементы интерфейса должны быть расположены относительно друг друга и как они должны вести себя при изменении размеров экрана или ориентации устройства. Это особенно важно в Android, где приложения используются на широком спектре устройств с различными размерами и разрешениями экранов.

Существует несколько типов контейнеров компоновки в Android, каждый из которых предлагает различные стратегии для расположения элементов:

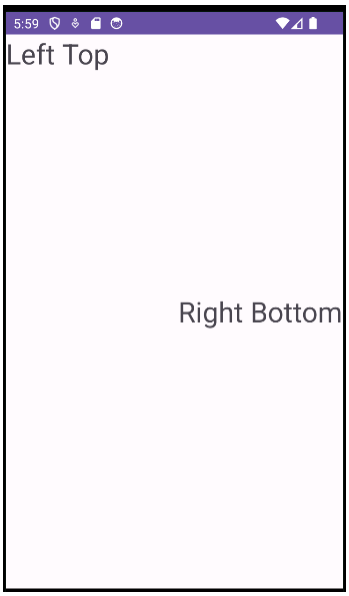
1. **LinearLayout:** Упорядочивает элементы в линейном порядке, вертикально или горизонтально. Это простой и интуитивно понятный способ компоновки, где элементы размещаются один за другим. LinearLayout поддерживает такое свойство, как вес элемента, которое передается

атрибутом **android:layout_weight**. Это свойство принимает значение, указывающее, какую часть оставшегося свободного места контейнера по отношению к другим объектам займет данный элемент. Например, если один элемент у нас будет иметь для свойства android:layout_weight значение 2, а другой – значение 1, то в сумме они дадут 3, поэтому первый элемент будет занимать 2/3 оставшегося пространства, а второй - 1/3.

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:text="Hello"
        android:background="#e0e0e0"
        android:layout_weight="1"
        android:textSize="30sp" />

    <TextView
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:text="World"
        android:background="#bdbdbd"
        android:layout_weight="3"
        android:textSize="30sp" />
</LinearLayout>
```

2. **RelativeLayout:** Позволяет располагать элементы относительно друг друга или относительно родительского контейнера. Это обеспечивает большую гибкость в определении местоположения и размера элементов.

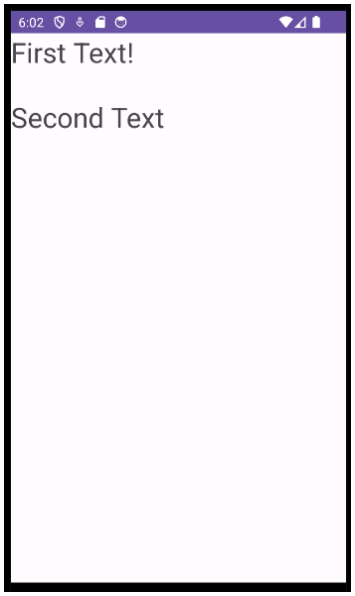
```
<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView android:text="Left Top"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:textSize="30sp"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true" />

    <TextView android:text="Right Bottom"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:textSize="30sp"
        android:layout_alignParentRight="true"
        android:layout_centerInParent="true" />
</RelativeLayout>
```



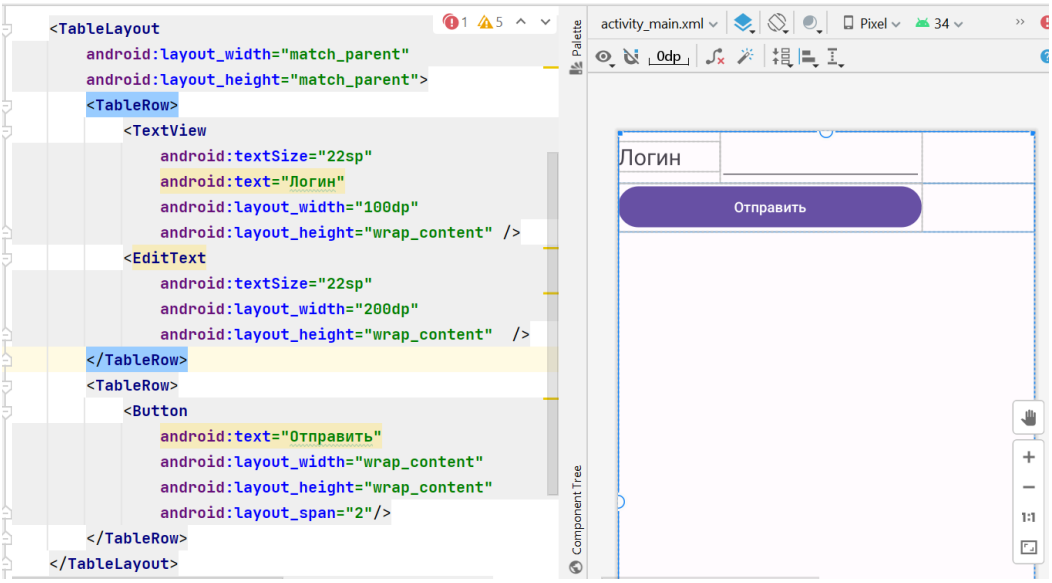
3. **FrameLayout:** Предназначен для размещения одного элемента поверх другого. Это может быть полезно для создания сложных перекрывающихся

интерфейсов, например, для отображения загрузочных индикаторов или для создания пользовательских компонентов интерфейса.

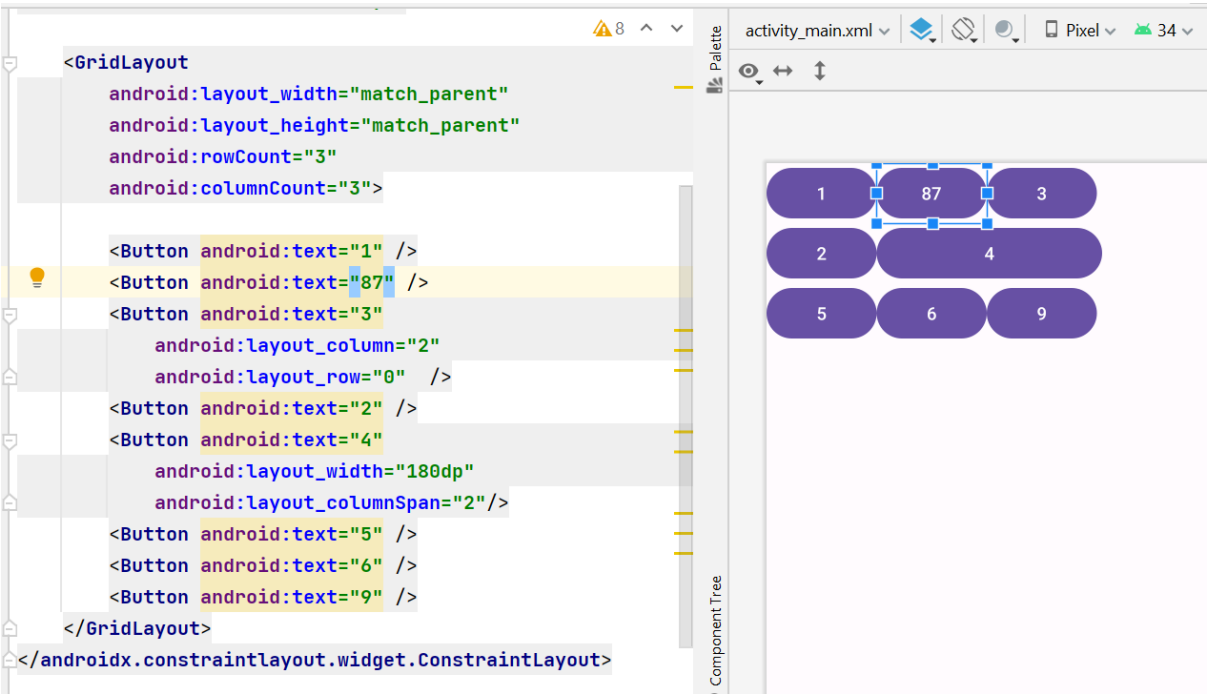
```
<FrameLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="First Text!"
        android:textSize="30sp"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Second Text"
        android:textSize="30sp"
        android:layout_marginTop="70dp"/>

</FrameLayout>
```



4. **TableLayout:** Структурирует элементы управления в виде таблицы по столбцам и строкам.



5. **GridLayout:** Представляет еще один контейнер, который позволяет создавать табличные представления. GridLayout состоит из коллекции строк, каждая из которых состоит из отдельных ячеек.

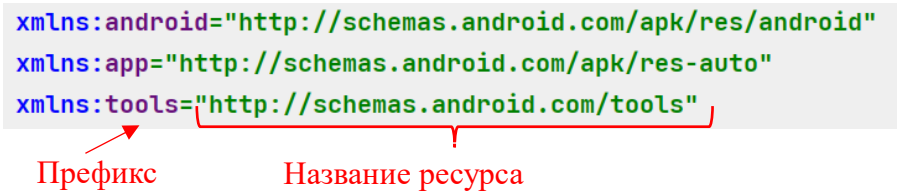



```
<GridLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:rowCount="3"
    android:columnCount="3">

    <Button android:text="1" />
    <Button android:text="87" />
    <Button android:text="3"
        android:layout_column="2"
        android:layout_row="0"  />
    <Button android:text="2" />
    <Button android:text="4"
        android:layout_width="180dp"
        android:layout_columnSpan="2"/>
    <Button android:text="5" />
    <Button android:text="6" />
    <Button android:text="9" />
</GridLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

6. **ConstraintLayout:** Относительно новый тип контейнера компоновки, который позволяет создавать сложные и гибкие интерфейсы с помощью набора ограничений для определения положения и размера виджетов. По умолчанию при открытии Activity все элементы располагаются в этом контейнере.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:ar
    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">
    <EditText
        android:id="@+id/editText"
        android:layout_width="180dp"
        android:layout_height="wrap_content"
        android:hint="Введите ваш логин"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Войти"
        app:layout_constraintLeft_toRightOf="@id/editText"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Корневой элемент содержит определение используемых пространств имен XML. Например, в коде по умолчанию в **ConstraintLayout** мы можем увидеть такие атрибуты:



Каждое пространство имен задается следующим образом: `xmlns:префикс="название_ресурса"`.

Через префикс мы сможем ссылаться на функциональность этого пространства имен.

Каждое пространство имен определяет некоторую функциональность, которая используется в приложении, например, предоставляют теги и атрибуты, которые необходимые для построения приложения.

- `xmlns:android="http://schemas.android.com/apk/res/android"`: содержит основные атрибуты, которые предоставляются платформой Android, применяются в элементах управления и определяют их визуальные свойства (например, размер, позиционирование).
- `xmlns:app="http://schemas.android.com/apk/res-auto"`: содержит атрибуты, которые определены в рамках приложения.
- `xmlns:tools="http://schemas.android.com/tools"`: применяется для работы с режиме дизайнера в Android Studio

Это наиболее распространенные пространства имен.

Для организации элементов внутри контейнера используются параметры разметки. Для их задания в файле xml используются атрибуты, которые начинаются с префикса **layout_**. В частности, к таким параметрам относятся атрибуты **layout_height** и **layout_width**, которые используются для установки размеров. Данные атрибуты могут иметь следующие значения:

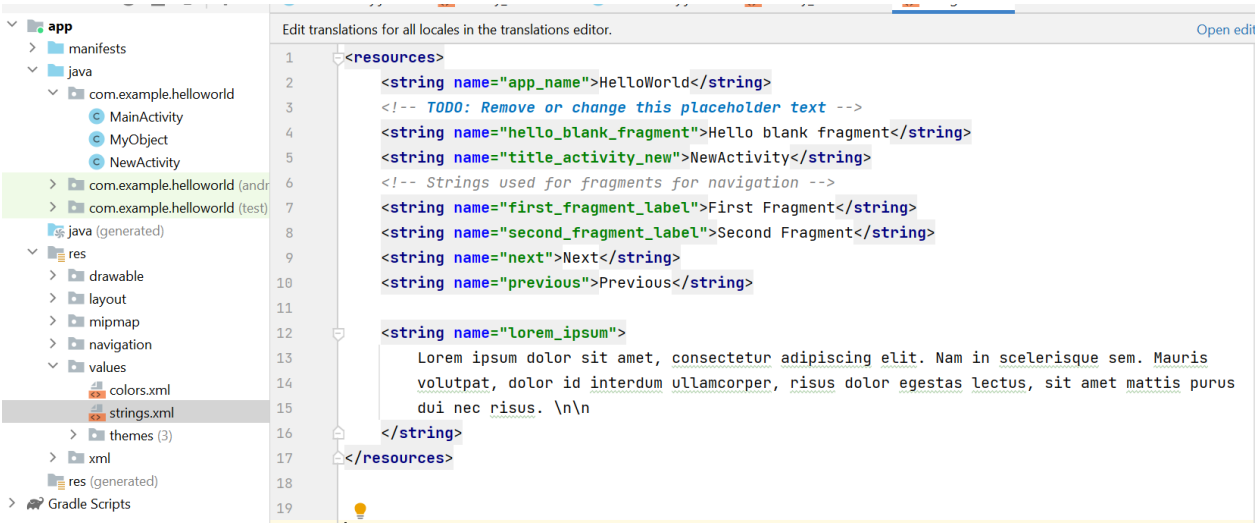
1. **match_parent**: Установка данного значения позволяет растянуть элемент по всей ширине или высоте контейнера. Данное значение применяется ко всем контейнерам, кроме ConstraintLayout. **match_parent** также не рекомендуется применять к элементам внутри ConstraintLayout. Вместо этого можно использовать значение **0dp**, чтобы растянуть элемент по горизонтали или вертикали.

2. **wrap_content**: Устанавливает те значения для ширины или высоты, которые необходимы, чтобы разместить на экране содержимое элемента.
3. Точные значения.

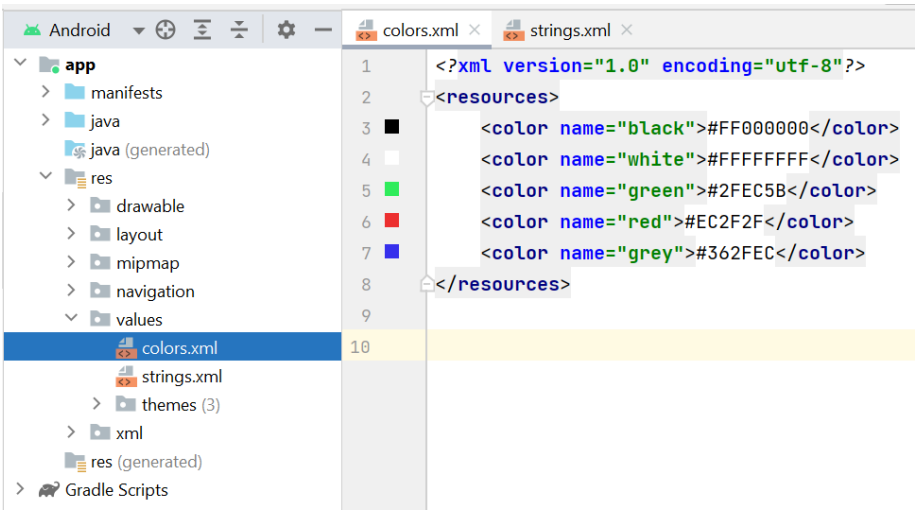
Часть 4. Ресурсы в Android

Ресурсы в Android — это специальные элементы, используемые для хранения различных типов данных, отделённых от кода приложения. Эти ресурсы включают в себя такие элементы, как строки, цвета, изображения, макеты и другие. Важной особенностью ресурсов является их удобство управления и изменения без необходимости перекомпиляции всего приложения.

Строковые ресурсы используются для хранения текстовой информации, такой как сообщения, названия кнопок и прочее. Хранение строк в отдельных ресурсах облегчает локализацию приложений, позволяя легко заменять текст на различные языки без изменения исходного кода.



Цветовые ресурсы используются для определения цветовой палитры приложения, включая цвета фона, текста, кнопок и других элементов интерфейса. Хранение цветов в отдельных ресурсах позволяет унифицировать и централизовать управление цветовой схемой и упрощая процесс дизайна. Это также упрощает изменения цветовой схемы для разных тем или режимов (например, дневного и ночного), позволяя менять цвета во всем приложении без необходимости редактирования каждого элемента вручную.

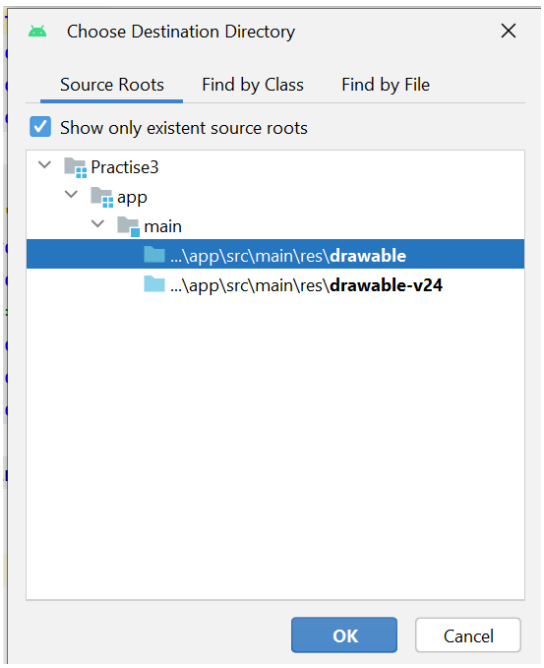


Оба ресурса находятся в пакете values и создаются по одному шаблону: сначала указывается тип данных, например color или string, затем в поле name указывается название ресурса и то значение, которое он будет содержать.

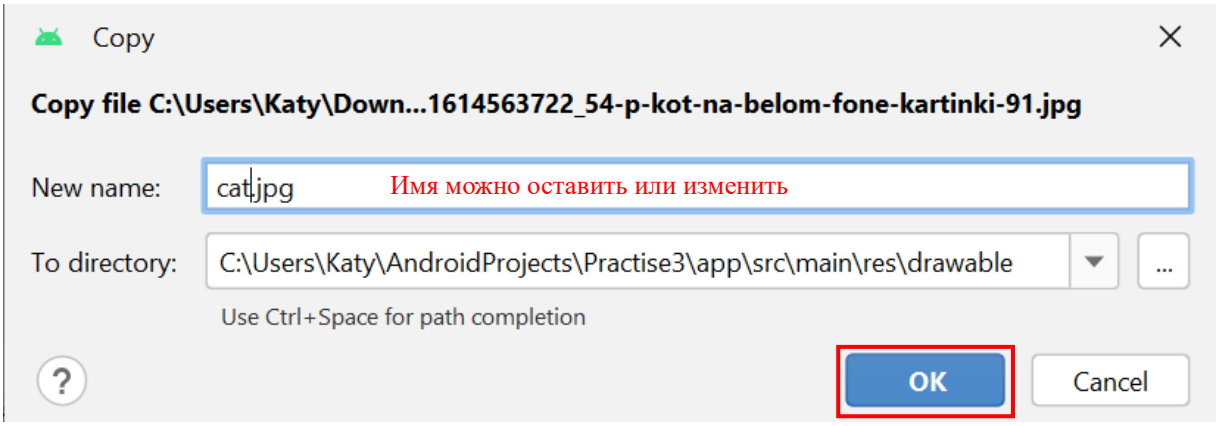
Изображения и графика, обычно хранятся в пакете drawable, в форматах PNG и JPEG, с остальными в Android Studio могут возникать ошибки. Они используются для улучшения визуальной составляющей приложения. Размещение графических ресурсов в отдельной папке позволяет легко обновлять и изменять изображения, не затрагивая остальную часть приложения.

Для добавления в проект в папку **res/drawable** какого-нибудь файла изображения необходимо скопировать на жестком диске какой-нибудь файл с расширением png или jpg и вставим его в папку **res/drawable** (для копирования в проект используется простой Copy-Paste).

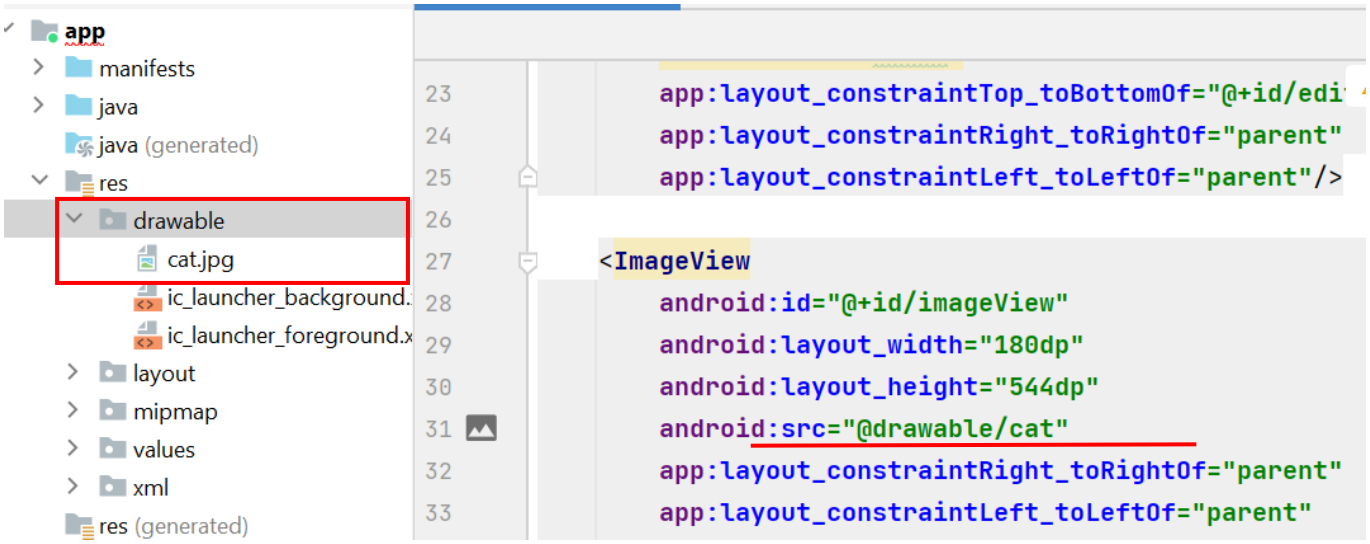
Далее нам будет предложено выбрать папку - **drawable** или **drawable-24**. Для добавления обычных файлов изображений выберем **drawable**:



При копировании файла нам будет предложено установить для него новое имя.



После этого в папку **drawable** будет добавлен выбранный нами файл изображения. Для отображения файла в **ImageView** у элемента устанавливается атрибут **android:src**.

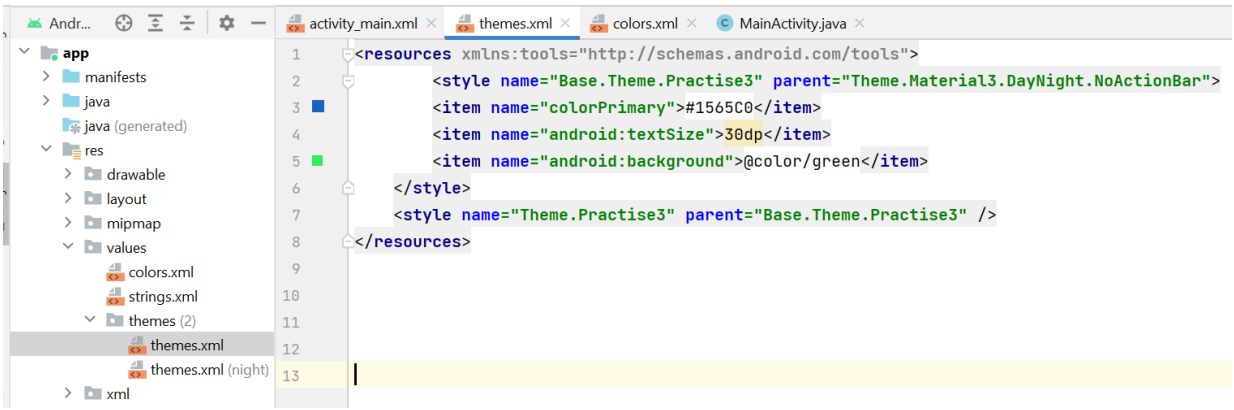


Макеты описывают структуру пользовательского интерфейса приложения, определяют расположение элементов, таких как кнопки, текстовые поля, и другие виджеты. Хранятся такие макеты в пакете layout.

Ресурсы позволяют детальнее индивидуализировать приложение, поменяв тему или локализировать его для конкретной страны или региона.

Для смены темы необходимо в пакете values перейти в пакет theme, где будут находится файлы для светлой и темной темы приложения, состоящие из специальных стилей, со своим набором атрибутов, позволяющих настраивать внешний вид отдельных компонентов.

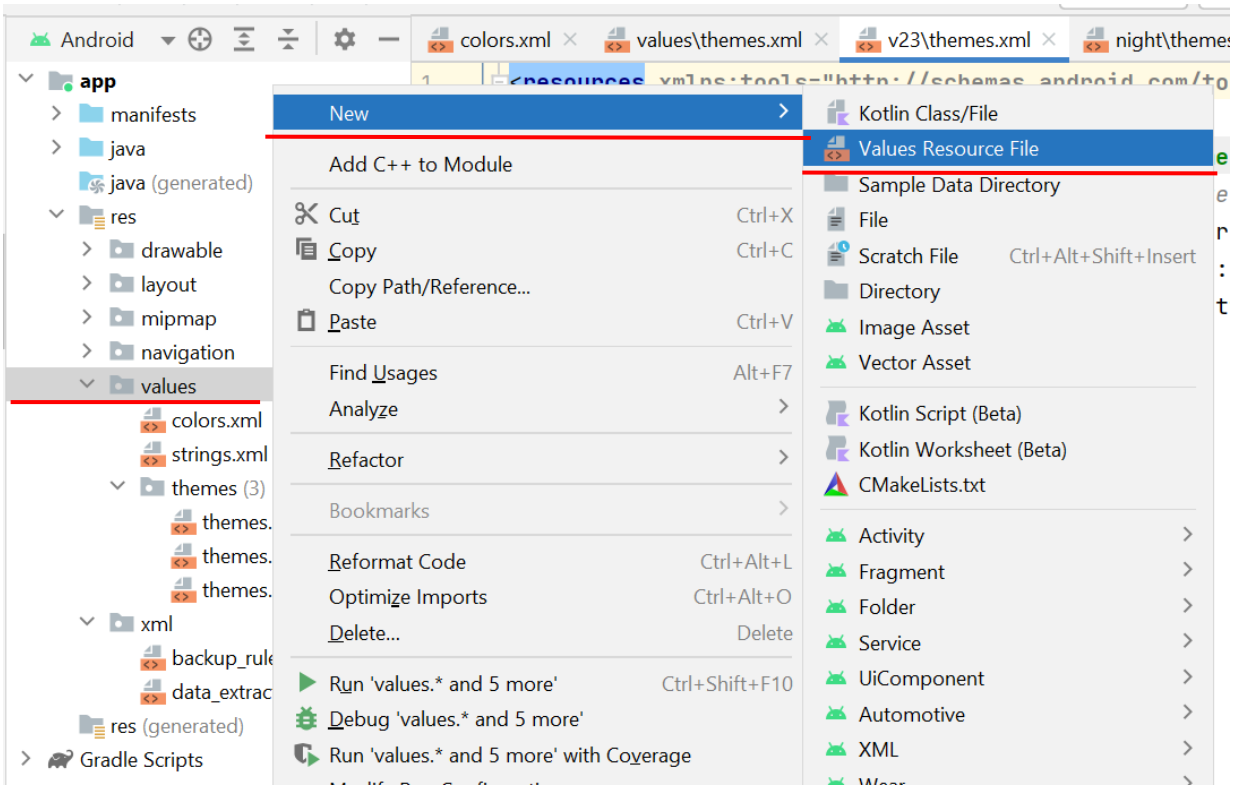
Эти атрибуты включают цвета, шрифты, размеры, отступы и многие другие параметры, которые определяют визуальное представление элемента. Примеры таких атрибутов включают colorPrimary для определения основного цвета темы, textSize для размера текста и background для фона элемента.



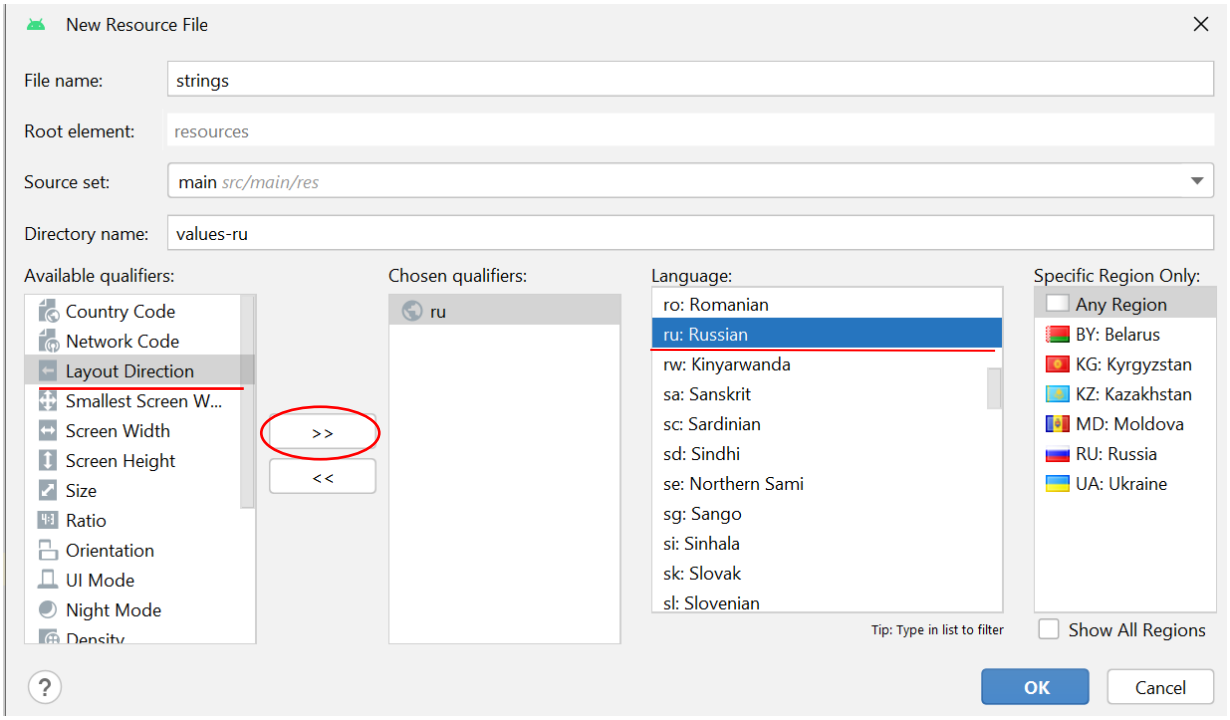
Результатом станет измененные параметры для всей темы.

Немаловажным элементом в иерархии стилей является параметр parent. Он позволяет определить родительский стиль для текущего стиля, наследуя его атрибуты и позволяя модифицировать или расширять их.

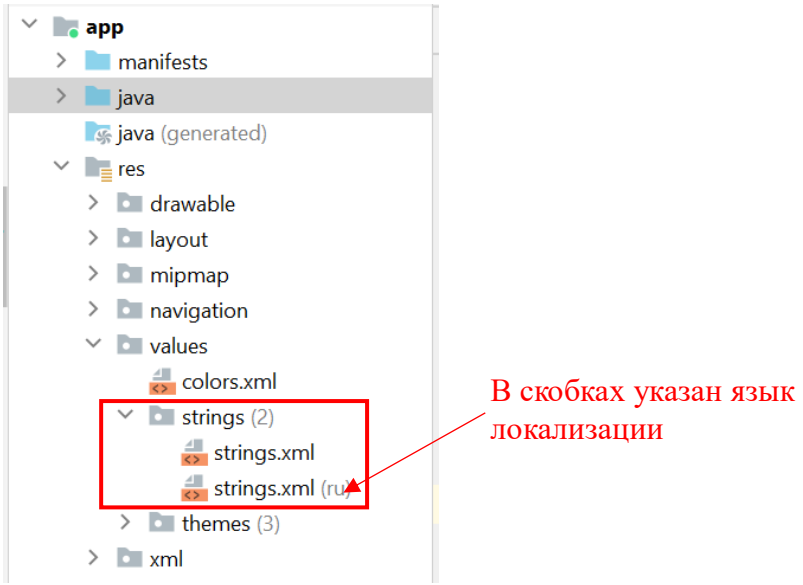
Локализация приложения осуществляется похожим образом, отличием является то, что для каждого региона необходимо создавать свой отдельный файл со строками. Для этого нужно нажать по пакету values правой кнопкой мыши и в открывшимся окне выбрать параметр Locale и регион, например ru (название файла может отличаться).



В диалоговом окне в левой части **Available qualifiers:** выбираем пункт **Locale** и переносим его в правую часть **Chosen qualifiers:** с помощью кнопки с двумя стрелками вправо. В появившейся третьей колонке выбираем нужные языки, например, русский. Вы увидите, что в поле **Directory name** автоматически появится нужное название папки.



В результате создания ресурсных файлов в структуре проекта будут добавлены записи в раздел **res/values/strings**.



Для корректной работы перевода необходимо созданные ранее строки перенести в новый файл. Таким образом, в данном файле будет содержаться текст для российского региона.

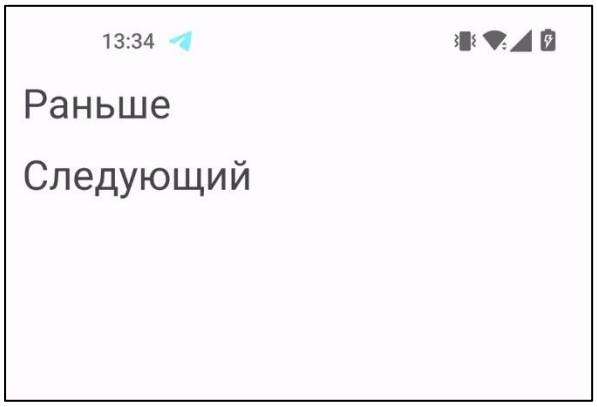
Листинг файла values-ru/string.xml

```xml
<resources>
    <string name="app_name">Здравствуй мир!</string>
    <!-- TODO: Remove or change this placeholder text -->
    <string name="lbl_login">Логин</string>
    <string name="lbl_password">пароль</string>
    <string name="hint_login">Учетная запись</string>
    <string name="previous">Раньше</string>
    <string name="next">Следующий</string>
</resources>
```

Листинг файла values/string.xml

```xml
<resources>
    <string name="app_name">HelloWorld</string>
    <!-- TODO: Remove or change this placeholder text -->
    <string name="lbl_login">Login</string>
    <string name="lbl_password">Password</string>
    <string name="hint_login">Account</string>
    <string name="next">Next</string>
    <string name="previous">Previous</string>
</resources>
```

Если запустить приложение, то локализация будет работать.



Задание

1. Реализовать несколько файлов разметки с применением следующих контейнеров: **LinerLayout, RelativeLayout, Constraint Layout, FrameLayout** (При желании можно расширить перечень другими контейнерами). Расположить в файлах разметки различные элементы управления (кнопки, текстовые поля и т.д.).
2. Реализовать переход между несколькими Activity с передачей данных (попробуйте передавать данные в определенные поля, а не сообщением, как в предыдущей практике) и возможностью возврата на предыдущую страницу.

3. Добавить в проект несколько строковых, размерных, цветовых и drawable ресурсов. Произвести изменение настроек темы. Добавить локализацию на другой язык.

Источники

1) https://developer.android.com/guide/components/activities/tasks-and-back-stack
2) https://developer.alexanderklimov.ru/android/layout/linearlayout.php
3) https://developer.alexanderklimov.ru/android/layout/relativelayout.php
4) https://developer.alexanderklimov.ru/android/layout/framelayout.php
5) https://developer.alexanderklimov.ru/android/layout/constraintlayout.php
6) https://developer.android.com/guide/topics/resources/providing-resources