Ресурсы

Работа с ресурсами

Ресурс в приложении Android представляет собой файл, например, файл разметки интерфейса или некоторое значение, например, простую строку. То есть ресурсы представляют собой и файлы разметки, и отдельные строки, и звуковые файлы, файлы изображений и т.д. Все ресурсы находятся в проекте в папке **res**. Для различных типов ресурсов, определенных в проекте, в папке **res** создаются подпапки. Поддерживаемые подпапки:

- **animator/**: xml-файлы, определяющие анимацию свойств
- **anim/**: xml-файлы, определяющие tween-анимацию
- **color/**: xml-файлы, определяющие список цветов
- **drawable/**: Графические файлы (*.png*, *.jpg*, *.gif*)
- **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>

Ссылка на ресурсы в коде

Тип ресурса в данной записи ссылается на одно из пространств (вложенных классов), определенных в файле R.java, которые имеют соответствующие им типы в xml:

- R.drawable (ему соответствует тип в xml drawable)

- R.id (id)
- R.layout (layout)
- R.string (string)
- R.attr (attr)
- R.plural (plurals)
- R.array (string-array)

Например, для установки ресурса activity_main.xml в качестве графического интерфейса в коде MainActivity в методе onCreate() мы уже познакомились с этой строкой:

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

Для получения ресурсов в классе Activity мы можем использовать метод **getResources()**, который возвращает объект **android.content.res.Resources**. Но чтобы получить сам ресурс, нам надо у полученного объекта Resources вызвать один из методов. Некоторые из его методов:

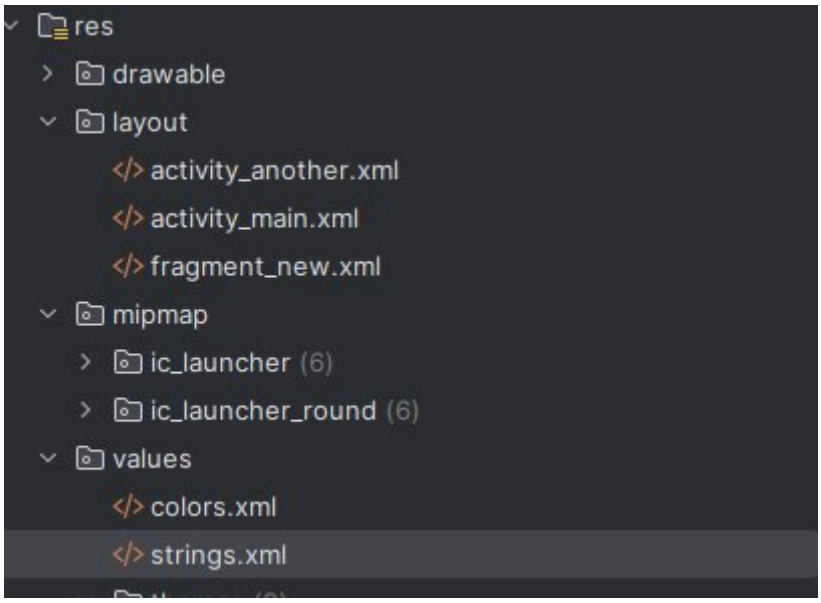
- **getString()**: возвращает строку из файла strings.xml по числовому идентификатору
- **getDimension()**: возвращает числовое значение - ресурс dimen
- **getDrawable()**: возвращает графический файл в виде объекта Drawable
- **getBoolean()**: возвращает значение boolean
- **getColor()**: возвращает определение цвета
- **getColorStateList()**: возвращает объект ColorStateList - набор цветов
- **getFont()**: возвращает определение шрифта в виде объекта Typeface
- **getFloat()**: возвращает значение float
- **getLayout()**: возвращает объект XmlResourceParser, связанный с файлом layout

Ресурсы строк

Ресурсы строк - один из важных компонентов приложения. Мы используем их при выведении названия приложения, различного текста, например, текста кнопок и т.д.

XML-файлы, представляющие собой ресурсы строк, находятся в проекте в папке **res/values**.

Давайте создадим себе новую строку. Перейдём в strings.xml



И там пропишем к примеру:

```
<string name="roflan_lico">Roflan lico</string>
```

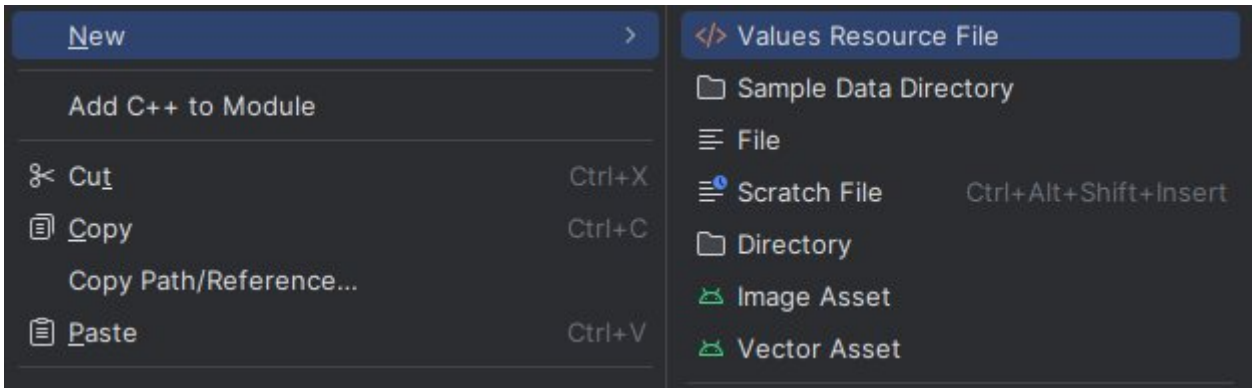
Теперь мы можем использовать свою строку в разметке. Пропишите вместо текста @string/ваше_название:

```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/roflan_lico"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.498"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.157" />
```

Если мы хотим получить свою строку в коде, то пропишем:

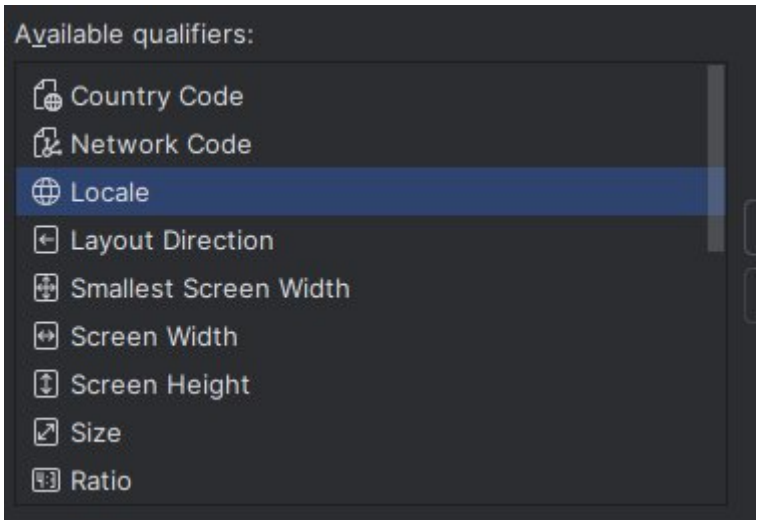
```
String lico = getResources().getString(R.string.roflan_lico);
```

Но, т.к у нас только один файл strings, это исключает возможность локализации под другие языки. Но это можно исправить. Кликните правой кнопкой мыши по папке **/values** и выберите values resource file:

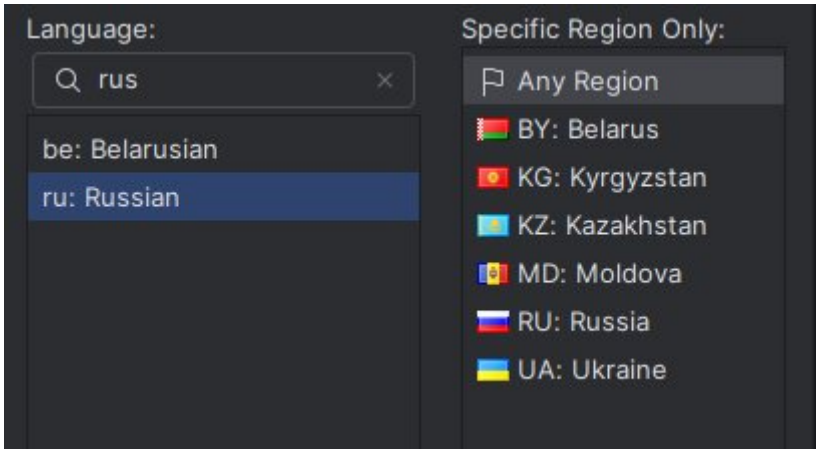


Файл назовите также strings.

В нём выберем Locale в этом меню:



И при выборе пропишем rus для поиска русского языка:



У вас создастся ваш файл с локализацией уже под русскую локаль. Теперь надо продублировать все строки из основного файла strings.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Моё приложение</string>
    <string name="roflan_lico">Рофлан лицо</string>
</resources>
```

Ресурсы Color и Themes. Установка цвета

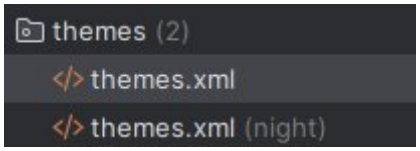
В приложении Android также можно определять ресурсы цветов (Color). Они должны храниться в файле по пути **res/values** и также, как и ресурсы строк, заключены в тег `<resources>`. Так, по умолчанию при создании самого простого проекта в папку **res/values** добавляется файл **colors.xml**.

Базово у нас в наличии только чёрный и белый.

Создайте ещё 4 цвета для своего приложения. К примеру:

```
<resources>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
    <color name="red">#FF0000</color>
    <color name="blue">#001AFF</color>
    <color name="yellow">#FBFF00</color>
    <color name="light_blue">#00EDEA</color>
</resources>
```

Далее перейдём в папку /themes:



Тут лежат как раз таки тёмная и светлая тема. Откройте themes.xml не ночной. Выглядит он так:

```
<resources xmlns:tools="http://schemas.android.com/tools">
    <style name="Base.Theme.MyApplication" parent="Theme.Material3.DayNight.NoActionBar">
        <!-- <item name="colorPrimary">@color/my_light_primary</item> -->
    </style>

    <style name="Theme.MyApplication" parent="Base.Theme.MyApplication" />
</resources>
```

Раскомментируйте строку с colorPrimary и пропишите любой свой цвет. Добавьте ещё базовых цветов для своей темы. К примеру:

```
<item name="colorPrimary">@color/white</item>
<item name="colorPrimaryInverse">@color/black</item>
<item name="colorSecondary">@color/light_blue</item>
<item name="colorContainer">@color/blue</item>
<item name="colorSecondaryContainer">@color/red</item>
```

А также продублируйте их для тёмной темы, изменив цвета, чтобы они различались.

```
<item name="colorPrimary">@color/black</item>
<item name="colorPrimaryInverse">@color/white</item>
<item name="colorSecondary">@color/light_blue</item>
<item name="colorContainer">@color/red</item>
<item name="colorSecondaryContainer">@color/blue</item>
```

Поясню для тех кто в танке. Данные переменные позволят нам менять цвета в зависимости от темы. colorPrimary обычно – основной цвет, использующийся для фона и т.п. colorPrimaryInverse – для текста и т.д. Переменные отражают их смысл в вашем дизайне. Так как же их использовать? <— ответ на вопрос.

```
android:textColor="?colorPrimaryInverse"
```

С помощью знака вопроса мы обращаемся ко всем атрибутам темы, а не на конкретные значения ресурсов.

Задание

Добавьте в ваши разметки item константные значения строк. Перекрасьте все добавленные в ваше приложение до этого элементы.

Parcelable

До этого при передаче элементов из активности в активность мы использовали простые данные.Но также мы можем передавать более сложные данные. В этом случае используется механизм сериализации. Android также предоставляет интерфейс **Parcelable**, позволяет сериализовать объекты и является более оптимизированным.И подобные объекты Parcelable можно передавать между двумя activity или использовать каким-то иным образом.

На прошлых парах мы создавали свой класс сущности и его список. Теперь, давайте его обновим. Добавьте ему использование интерфейса Parcelable:
public class User implements Parcelable

От нас требуется реализовать его методы describeContents() и writeToParcel(Parcel dest, int flags) а также реализовать CREATOR, который будет

собирать переданные данные в новый объект. Теперь ваш класс должен выглядеть примерно так:

```
public class User implements Parcelable {

    private String name;
    private String company;
    private int age;

    public static final Creator<User> CREATOR = new Creator<User>() {
        @Override
        public User createFromParcel(Parcel source) {
            String name = source.readString();
            String company = source.readString();
            int age = source.readInt();
            return new User(name, company, age);
        }

        @Override
        public User[] newArray(int size) {
            return new User[size];
        }
    };

    public User(String name, String company, int age){
        this.name = name;
        this.company = company;
        this.age = age;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getCompany() {
        return company;
    }
    public void setCompany(String company) {
        this.company = company;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
```

```
    }

    @Override
    public int describeContents() {
        return 0;
    }

    @Override
    public void writeToParcel(Parcel dest, int flags) {
        dest.writeString(name);
        dest.writeString(company);
        dest.writeInt(age);
    }
}
```

Если мы захотим иметь самописный класс внутри класса, к примеру у меня это Info, то другой класс тоже должен иметь реализацию Parcelable. А внутри нашего основного класса мы используем методы **writeParcelable()** и **readParcelable().** Выглядеть будет так:

```
Info info = source.readParcelable(Info.class.getClassLoader());

dest.writeParcelable(info,flags);
```

Задание

Создайте новый класс, который будет как-либо дополнять вашу основную сущность и добавьте его в неё. Сделайте, чтобы при нажатии на элемент вашего списка в новую Activity передавался класс сущности и выводился в textView.