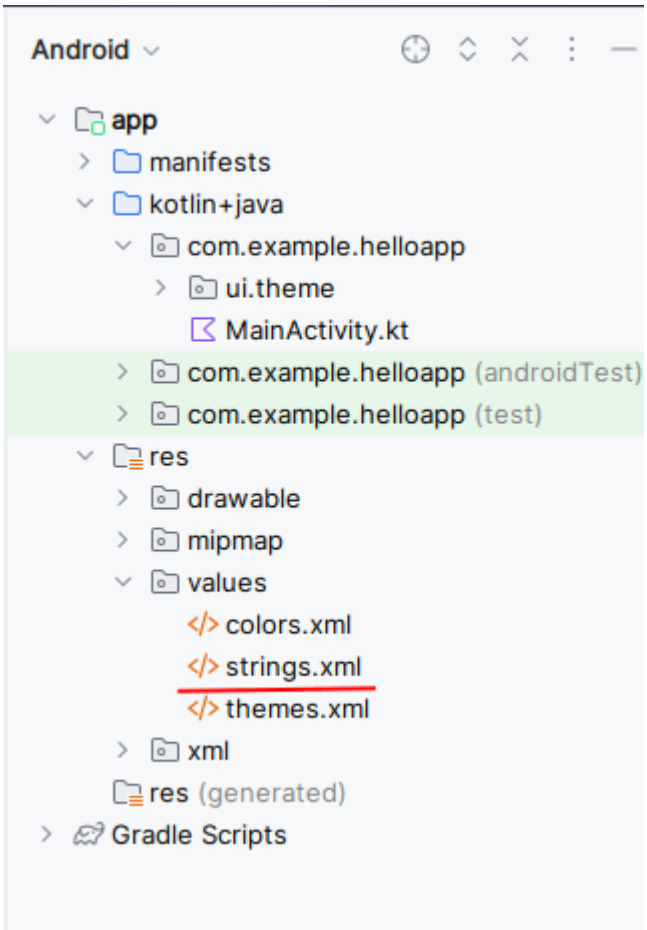
Ресурсы в Jetpack Compose

Ресурсы строк

Кроме файлов кода на языке Kotlin проект может включать дополнительные файлы, которые используются в приложении, например, файлы изображений, определения иконок, файлы xml и так далее. Подобные файлы называются ресурсами. Все ресурсы находятся в проекте в каталоге res.

Одним из наиболее применяемых типов ресурсов являются ресурсы строк. Они используются при выведении названия приложения, различного текста, например, текста кнопок и т.д. Например, нам нужно вывести в различных местах приложения один и тот же текст. Вместо того, чтобы по несколько раз определять его в коде Kotlin, мы можем определить его один раз в виде строкового ресурса и использовать в любом месте приложения.

По умолчанию строковые ресурсы хранятся в проекте в каталоге res/values. При создании проекта в этот каталог по умолчанию добавляется файл строковых ресурсов strings.xml.



Если мы откроем файл, то мы найдем в нем строки наподобие следующих:

```
<resources>
    <string name="app_name">helloapp</string>
</resources>
```

Каждый отдельный строковый ресурс определяется с помощью элемента string, а его атрибут name содержит название ресурса. Так, в самом простом виде этот файл определяет один ресурс "app_name", который устанавливает название приложения и который в моем случае имеет значение "helloapp" (по умолчанию значение этого ресурса соответствует названию проекта). Но естественно мы можем определить любые строковые ресурсы.

Затем в приложении в файлах кода мы можем ссылаться на эти ресурсы через их идентификатор, который имеет следующий вид:

```
R.string.название_ресурса
```

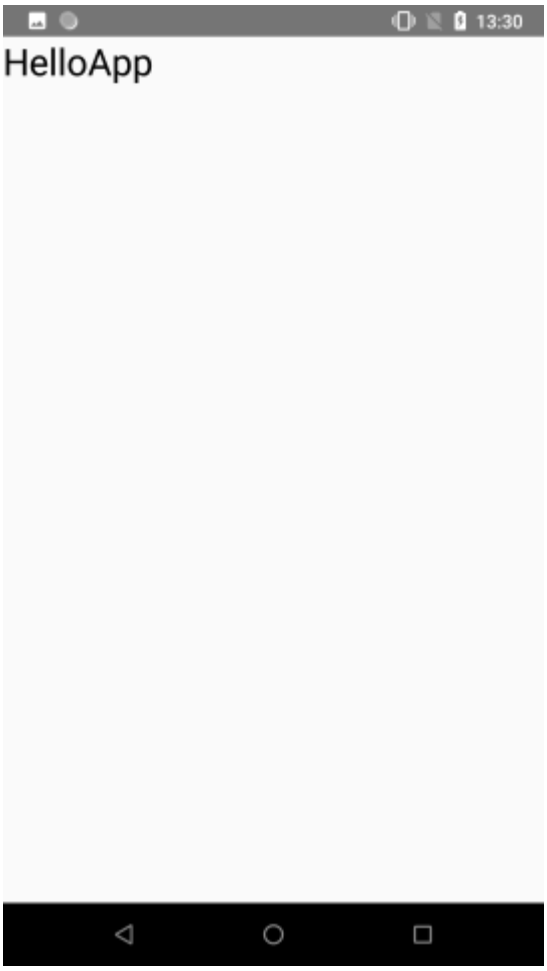
Например, для обращения к определенному по умолчанию строковому ресурсу app_name применяется идентификатор

```
R.string.app_name
```

Чтобы получить строковый ресурс в коде Kotlin, применяется встроенная функция androidx.compose.ui.res.stringResource(), в которую передается идентификатор ресурса и которая возвращает строку в виде объекта String. Например, получим в коде ресурс app_name:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.Text
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Text(
                text = stringResource(R.string.app_name),
                fontSize = 28.sp
            )
        }
    }
}
```

В данном случае параметру text компонента Text передается строка из строкового ресурса app_name:



Подобным образом при необходимоси мы можем определять и свои ресурсы. Например, изменим файл res/values/strings.xml следующим образом:

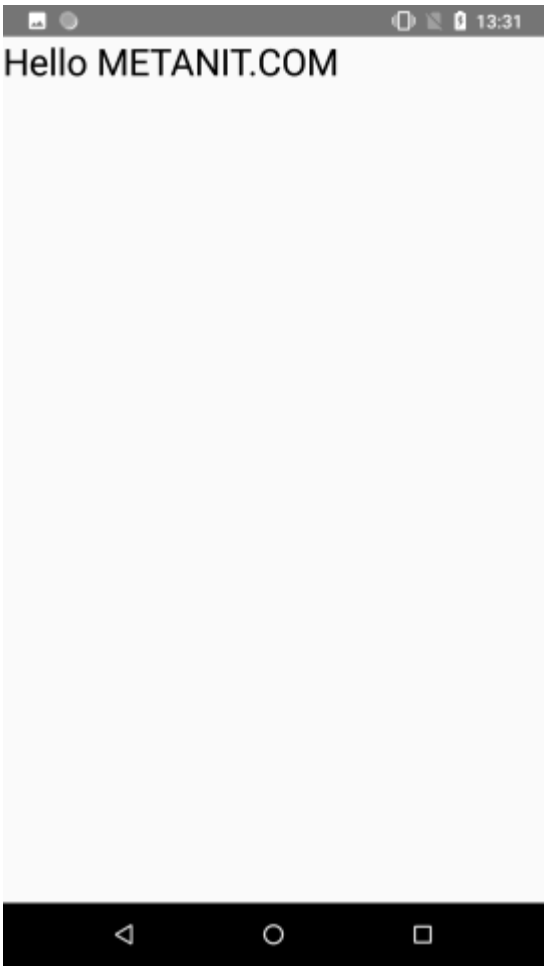
```
<resources>
    <string name="app_name">helloapp</string>
    <string name="message">Hello METANIT.COM</string>
</resources>
```

Здесь добавлен строковый ресурс "message", который имеет значение "Hello METANIT.COM"

Используем этот строковый ресурс в коде:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.Text
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
```

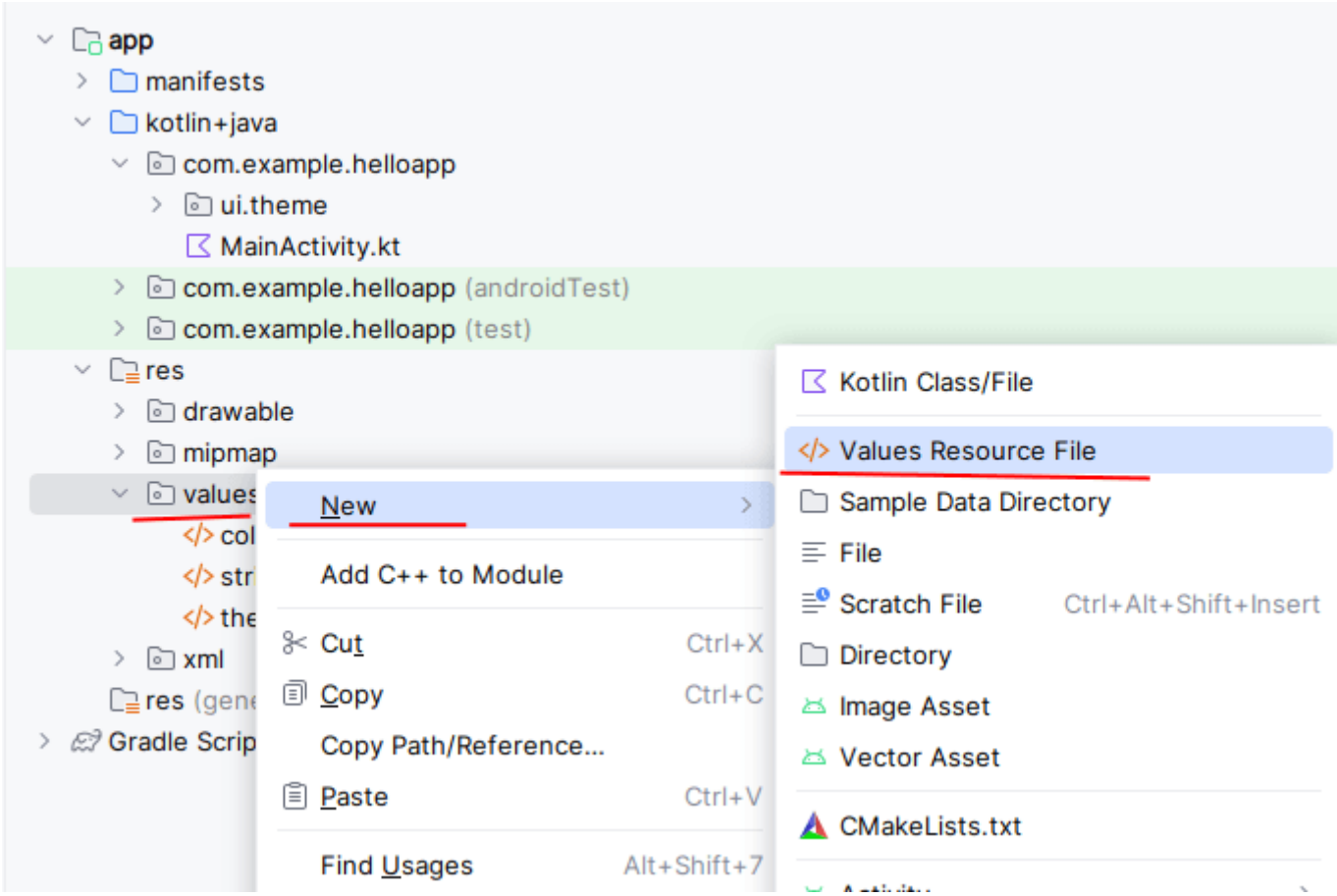
```
            setContent {
                Text(
                    text = stringResource(R.string.message),
                    fontSize = 28.sp
                )
            }
        }
    }
```



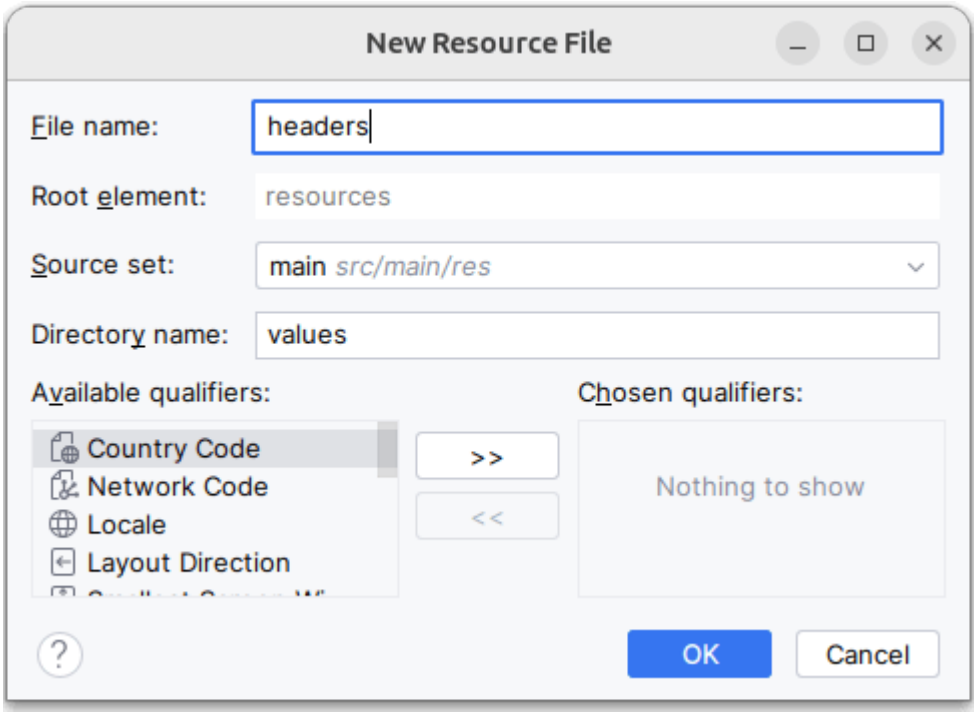
Добавление файла ресурсов строк

Хотя по умолчанию для ресурсов строк применяется файл strings.xml, но можно добавлять дополнительные файлы ресурсов в каталог проекта res/values. При этом достаточно соблюдать структуру файла: он должен иметь корневой узел и иметь один или несколько элементов .

Так, нажмем на папку res/values правой кнопкой мыши и в появившемся списке выберем пункт New -> Value Resource File:



После этого нам будет предложено определить для файла имя:



Назовем, к примеру, headers.xml (название файла произвольное), а для всех остальных полей оставим значения по умолчанию. И в папку res/values будет добавлен новый файл headers.xml. Определим в нем пару ресурсов:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
```

```
    <string name="welcome">Добро пожаловать</string>
    <string name="click_button">Отправить</string>
</resources>
```

И после этого мы также сможем использовать эти ресурсы в коде Kotlin, например:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.material.Button
import androidx.compose.material.Text
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Column{
                Text(
                    text = stringResource(R.string.welcome),
                    fontSize = 28.sp
                )
                Button(onClick={}){
                    Text(
                        text = stringResource(R.string.click_button),
                        fontSize = 22.sp
                    )
                }
            }
        }
    }
}
```

Форматирование строк

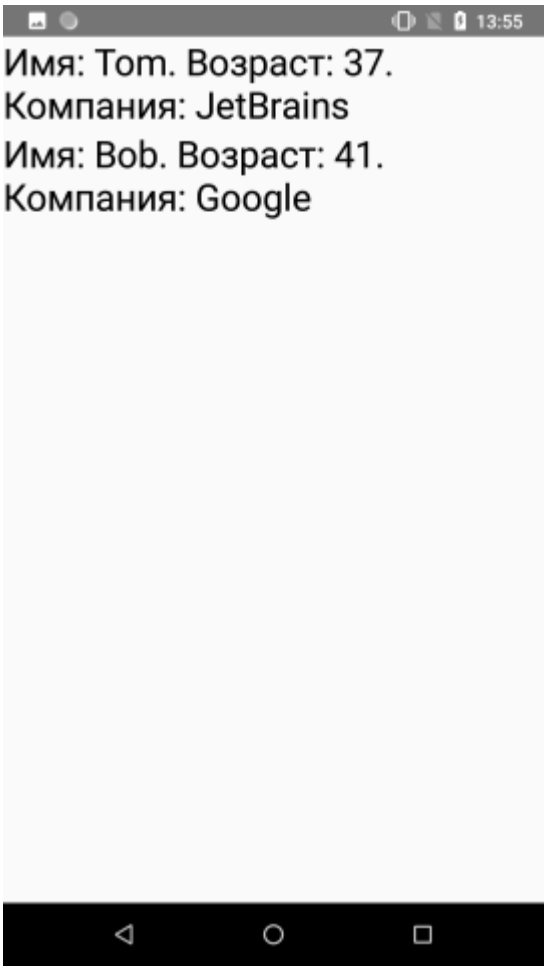
Функция stringResource() позволяет применять к ресурсам строк форматирование. Преимуществом форматирования является то, что мы можем определить общий щаблон и затем подставлять в него необходимые значения. Например, изменим файл strings.xml:

```
<resources>
    <string name="app_name">helloapp</string>
    <string name="user_data">Имя: %1$s. Возраст: %2$d. Компания: %3$s</string>
</resources>
```

Здесь ресурс user_data представляет строку с форматированием. Так, она содержит такие символы как %1$s, %2$d и %3$ds. Что они означают? %1$s указывает, что это первый аргумент, а символ "s" говорит, что этот аргумент представляет строку. %2$d представляет второй аргумент, а символ "d" в конце указывает, что это будет целое число. Аналогично %3$s указывает, что это третий аргумент, который представляет строку.

Получим ресурс в коде Kotlin:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.material.Text
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Column{
                Text(
                    text = stringResource(R.string.user_data, "Tom", 37,
"JetBrains"),
                    fontSize = 28.sp
                )
                Text(
                    text = stringResource(R.string.user_data, "Bob", 41,
"Google"),
                    fontSize = 28.sp
                )
            }
        }
    }
}
```



Вызов функции stringResource(R.string.user_data, "Tom", 37, "JetBrains") получает ресурс "user_data" и в качестве последующих параметров передает в строку вставляемые значения. Так, вместо первого аргумента-строки в user_data вставляется второй аргумент функции - строка "Tom", вместо второго аргумента-числа в user_data вставляется число 37, а вместо третьего аргумента в user_data передается строка "JetBrains".

Ресурсы Plurals

Ресурсы Plurals представляют еще один вид набора строк, в которым применяется форматирование. Он предназначен для описания количества элементов. Для чего это надо? К примеру, возьмем существительное: нередко оно изменяет окончание в зависимости от числительного, которое с ним употребляется: 1 цветок, 2 цветка, 7 цветков. Для подобных случаев и используется ресурс plurals.

Посмотрим на примере. Добавим в папку res/values новый ресурс flowers.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <plurals name="flowers">
        <item quantity="one">%d цветок</item>
        <item quantity="few">%d цветка</item>
        <item quantity="many">%d цветков</item>
    </plurals>
</resources>
```

Для задания ресурса используется элемент . Его атрибут name хранит название ресурса.

Сами наборы строк вводятся дочерними элементами . Этот элемент имеет атрибут quantity, который имеет значение, указывающее, когда эта строка используется. Данный атрибут может принимать следующие значения:

- zero: строка для количества в размере 0
- one: строка для количества в размере 1 (для русского языка - для задания всех количеств, оканчивающихся на 1, кроме 11)
- two: строка для количества в размере 2
- few: строка для небольшого количества
- many: строка для больших количеств
- other: все остальные случаи

Причем в данном случае многое зависит от конкретного языка. А система сама позволяет определить, какое значение брать для того или иного числа.

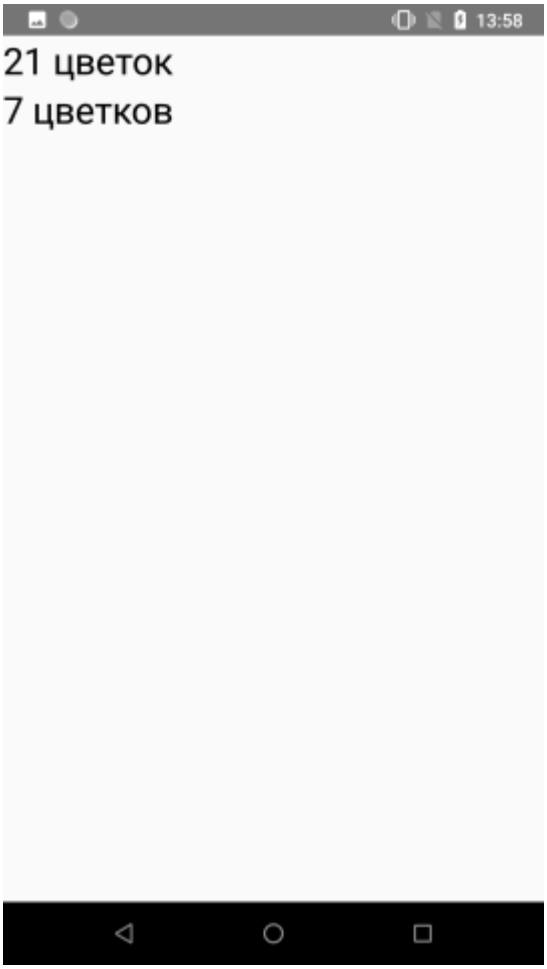
Получим и используем этот ресурс в коде Kotlin:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.material.Text
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val resources = LocalContext.current.resources
            Column{
                Text(
                    text = resources.getQuantityString(R.plurals.flowers, 21, 21),
                    fontSize = 22.sp
                )
                Text(
                    text = resources.getQuantityString(R.plurals.flowers, 7, 7),
                    fontSize = 22.sp
                )
            }
        }
    }
}
```

На данный момент Jetpack Compose не поддерживает напрямую получение подобных ресурсов. Поэтому для их извлечения вначале необходимо получить текущий контекст через свойство

LocalContext.current и затем обратиться к его свойству resource, которое представляет класс Resources и ассоциирован со всеми ресурсами приложения.

Далее с помощью метода getQuantityString класса Resources получаем значение ресурса. Первым параметром передаем идентификатор ресурса. Вторым параметром идет значение. для которого нужно найти нужную строку. Третий параметр представляет собой значение, которое будет вставляться на место плейсхолдера %d. То есть мы получаем строку для числа 21.



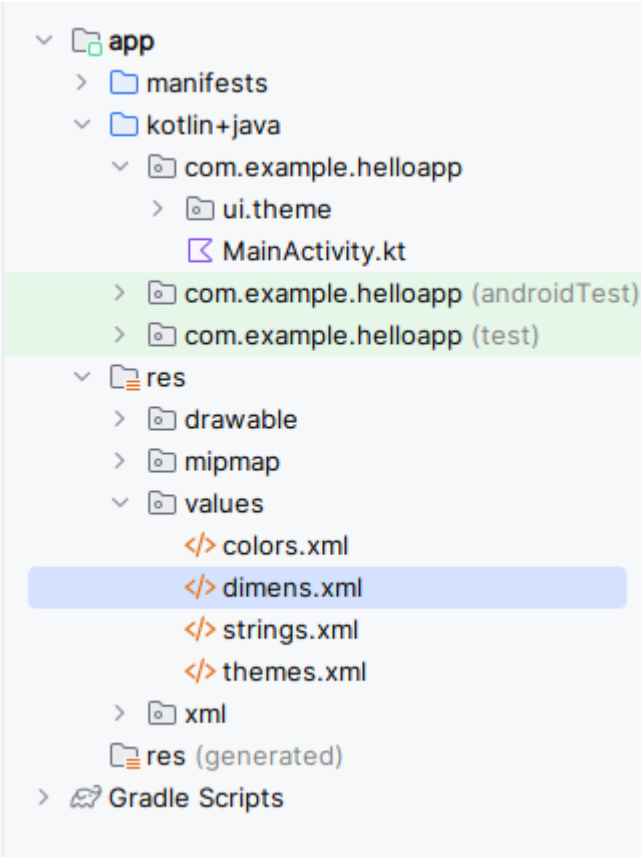
Ресурсы dimension

Ресурсы dimension задают размеры. Определение размеров должно находиться в папке res/values в файле с любым произвольным именем. Общий синтаксис определения ресурса следующий:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="имя_ресурса">используемый_размер</dimen>
</resources>
```

Как и другие ресурсы, ресурс dimension определяется в корневом элементе . Тег обозначает ресурс и в качестве значения принимает некоторое значение размера в единицах dp.

Так, добавим в Android Studio в папку res/values новый элемент Values Resources File, который назовем dimens.xml.



Определим в нем следующее содержимое:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="padding">30.dp</dimen>
    <dimen name="width">200.dp</dimen>
    <dimen name="height">150.dp</dimen>
</resources>
```

Здесь определены три ресурса. Первый ресурс - padding имеет значение 30.dp, второй ресурс - width - 200.dp, третий ресурс - height - 150.dp. Названия ресурсов могут быть произвольными. А в качестве единиц измерения могут применяться только единицы dp.

Затем в файлах кода мы можем ссылаться на эти ресурсы через его идентификатор, который имеет следующий вид:

```
R.dimen.название_ресурса
```

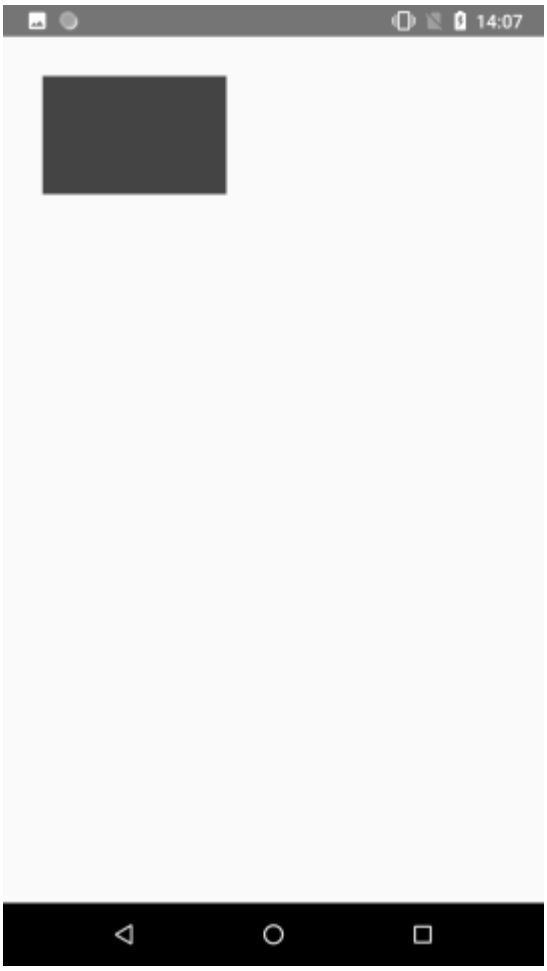
Например, обращение к ресурсу padding:

```
R.dimen.padding
```

Чтобы получить ресурс dimen в коде Kotlin, применяется встроенная функция androidx.compose.ui.res.dimensionResource(), в которую передается идентификатор ресурса и которая возвращает объект Dp.

Используем ресурс в коде Kotlin:

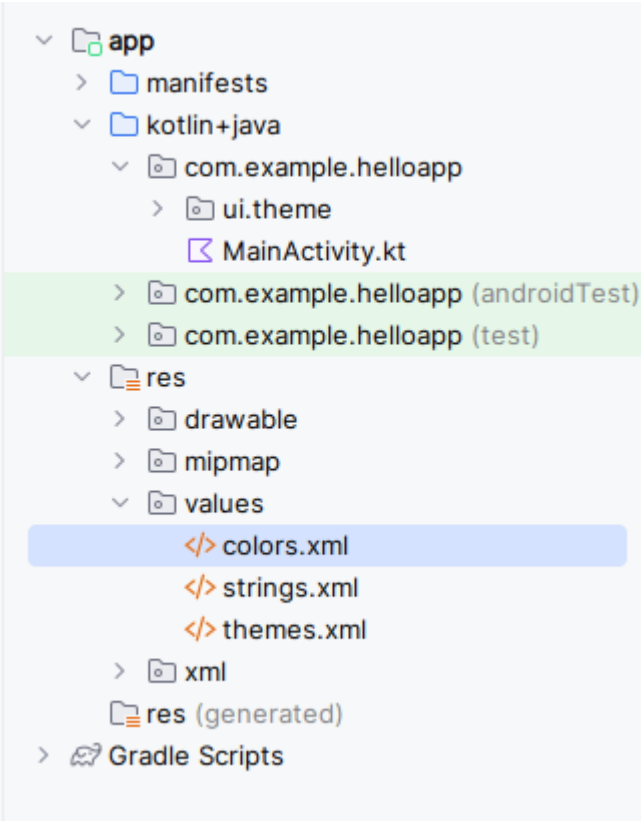
```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.res.dimensionResource

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {

            Box(
                Modifier.width(dimensionResource(R.dimen.width))
                    .height(dimensionResource(R.dimen.height))
                    .padding(dimensionResource(R.dimen.padding))
                    .background(Color.Red)
            )
        }
    }
}
```



Здесь создается объект Box, в функциях-модификаторах которого для установки ширины, высоты и отступов применяются ресурсы dimen.

Ресурсы Color

Ресурсы Color хранят определения цветов. Они должны храниться в проекте в каталоге res/values и также, как и ресурсы строк, заключены в тег . Так, по умолчанию при создании самого простого проекта в папку res/values добавляется файл colors.xml:



По умолчанию он имеет следующее определение:

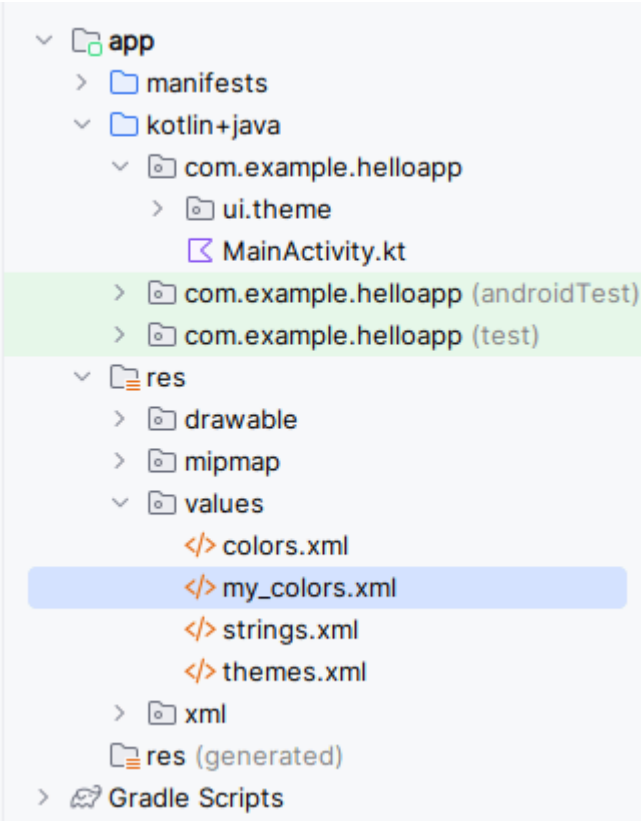
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
</resources>
```

Цвет определяется с помощью элемента . Его атрибут name устанавливает название цвета, которое будет использоваться в приложении, а шестнадцатеричное число - значение цвета.

Для задания цветовых ресурсов можно использовать следующие форматы:

- #RGB (#F00 - 12-битное значение)
- #ARGB (#8F00 - 12-битное значение с добавлением альфа-канала)
- #RRGGBB (#FF00FF - 24-битное значение)
- #AARRGGBB (#80FF00FF - 24-битное значение с добавлением альфа-канала)

Чтобы не трогать и не портить данный файл, определим свой новый файл ресурсов и для этого добавим в папку res/values новый файл ресурсов, который назовем my_colors.xml.



Изменим файл my_colors.xml, добавив в него пару цветов:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="textViewBackColor">#A0EAE1</color>
    <color name="textViewFontColor">#00695C</color>
</resources>
```

В файлах кода мы можем ссылаться на эти ресурсы через их идентификатор, который имеет следующий вид:

```
R.color.название_ресурса
```

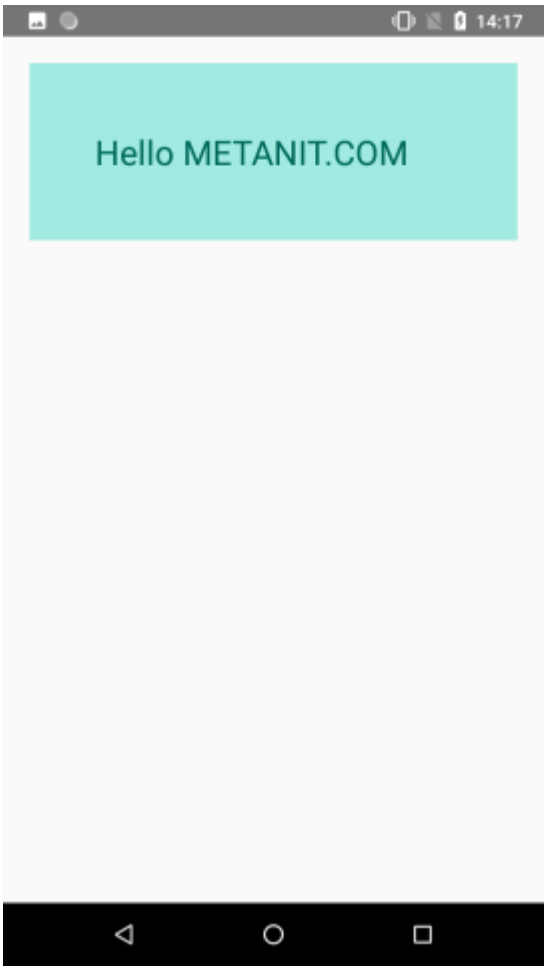
Например, обращение к ресурсу textViewBackColor:

```
R.color.textViewBackColor
```

Чтобы получить ресурс color в коде Kotlin, применяется встроенная функция androidx.compose.ui.res.colorResource(), в которую передается идентификатор ресурса и которая возвращает объект Color.

Используем ресурсы Color в коде Kotlin:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.material.Text
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.colorResource
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Text(
                text="Hello METANIT.COM",
                fontSize= 26.sp,
                color = colorResource(R.color.textViewFontColor),
                modifier = Modifier.padding(20.dp)
                    .fillMaxWidth()
                    .background(colorResource(R.color.textViewBackColor))
                    .padding(50.dp)
            )
        }
    }
}
```


14:17
Hello METANIT.COM