2.3. Макет активности

Сайт: Samsung Innovation Campus
Курс: Мобильная разработка на Kotlin
Книга: 2.3. Макет активности

Напечатано:: Murad Rezvan
Дата: понедельник, 3 июня 2024, 17:42

Оглавление

2.3.1 Описание элементов активности

Файлы, содержащие код по созданию и работе с активностями, размещаются в папке *src* пакета проекта. Для каждого класса представления и активности в пакете инструментов Android-разработчика предусмотрено текстовое описание объектов этого класса на языке xml с заготовленными тегами для самих экземпляров класса и именами их свойств, что позволяет создание xml-вёрстки макета активности. Для указания того, что свойство принадлежит объекту из пакета Android, используется префикс "android:", значение свойства присваивается в соответствии со стандартом xml через знак "=" в виде текста. Обязательными свойствами всех представлений являются размеры выделяемой области экрана: ширина и высота View. Минимальное описание элемента «Кнопка», достаточное для её определения выглядит так:

```
<Button
    android:layout_width="70dp"
    android:layout_height="33dp"/>
```

Ресурсы, используемые при описании объектов, так же входят в состав проекта. Папка ресурсов *res* содержит xml-описания составляющих приложения, размещённые в соответственных папках. К таким элементам относятся:

- изображения, используемые в приложении;
- стили объектов приложения;
- различные константы (цвета, строки и т.п.);
- разметки экрана - слои (layouts), в которых размещаются представления.

2.3.2 Ресурсы проекта

Под описания констант в папке res выделяется папка с именем *values*. Для создания приложения, переводимого на язык устройства, папки values нагружаются постфиксами с языковым расширением. Так для русского языка папка будет носить имя values-ru, а для немецкого values-de. В основном это необходимо для строковых констант. При работе приложения с несколькими разноязычными папками values константы будут считаны из папки, соответствующей языку, настроенному на устройстве запуска приложения. Само описание констант проекта размещается в xml-файлах с соответственными названиями. Корневым элементом этих описаний является `<resources>`, что указывает на факт описания ресурса. В корневой объект вкладываются объекты, заключаемые в теги их типа с обязательным указанием имени объекта. Обращение к ресурсам из кода и других описаний производится по типу и имени вызываемого ресурса: ="@typeResource/nameResource". При работе мастер создания android-проекта генерирует три таких файла с описаниями стилей (styles.xml), строк (strings.xml) и цветов (colors.xml). В них уже описаны текстовые константы, стили активностей и их элементов и значения цветовых констант, использованных в проекте. При создании собственных констант их описания можно добавлять в эти существующие файлы, а можно создать собственные. Можно даже все ресурсы поместить в один файл, что не будет считаться ошибкой, но создаст неудобство разработчику в поисках соответствующих констант. Именно по этому рекомендуется каждый тип констант описывать в отдельном файле с соответственным именем. Для создания нового файла ресурсов достаточно выполнить для соответственной папки values команду New... -> Values resource file и дать ему имя. При этом будет создан xml-файл с корневым элементом <resources></resources>.

Описание нового ресурса - это xml-описание нового объекта в xml-нотации:

```
<тип_ресурса name = "имя_объекта">описание содержимого объекта</тип_ресурса>
```

Примеры описания ресурсов и обращения к ним

Описание строки в ресурсах:

```
<string name = "author_name">Академия Samsung</string>
```

Обращение к строке из кода:

```
var author: String = getString(R.string.author_name)
```

Обращение к строке из xml-описания свойства:

```
android:text = "@string/author_name"
```

Описание цвета в файле ресурсов:

```
<color name = "text_color">#010101</color>
```

Обращение из кода:

```
var color = getColor(R.color.text_color)
```

Взятие цвета из xml-описания свойства:

```
android:textColor = "@color/text_color"
```

Для описания объекта типа *Страна* можно ввести новый тег `<country name = "rus">Россия</country>`, тогда обращение к этому ресурсу будет выглядеть соответственно `var rus = getString(R.country.rus)` и `android:text = "@country/rus"`

Описание других стран тоже разместятся в тег *country*, тем самым получится описание массива стран. В этом случае удобнее все объекты собрать в один структурированный объект-массив и каждую страну сделать его элементом . Это позволит не именовать каждый ресурс отдельно и при обращении из кода вернуть сразу массив наименований стран.

```
<resources>
<array name = "countries">
<item>Россия</item>
<item>England</item>
<item>France</item>
<item>Azerbaycan</item>
</array>
</resources>
```

Обращаться к этому ресурсу нужно уже как к массиву данных

```
val countries: Array <String> = resources.getStringArray(R.array.countries)
```

Kotlin требует, чтобы при чтении из ресурсов тип данных элементов массива был чётко определён. По этому желательно при описании массива сразу пользоваться тегами объектов `<string-array>` и `<integer-array>`.

Поскольку стиль представляет собой набор настроек, то стиль так же описывается в виде массива

```
<style name = "my_stand_style">
    <item name="android:layout_height">300dp</item>
    <item name="android:layout_width">50dp</item>
    <item name="android:textColor">@color/colorPrimary</item>
</style>
```

При необходимости группировки файлов описаний их помещают в отдельную папку. Так в проекте изначально все изображения собраны в общие папки: *drawable* для xml-описаний векторных картинок и *mipmap* для хранения изображений в графическом формате. Форматами растровых изображений, корректно обрабатываемых операционной системой Android, являются *.png* и *.jpg*. Имена файлов обязательно начинаются с латинского знака в нижнем регистре

При добавлении в проект меню его разделы тоже помещаются в текстовый файл и для них в папке res создаётся папка *menu*. При таком именовании Android Studio в текстовые файлы этой папки поместит описание объекта `<menu>`, содержащего дочерние элементы . Подробнее о создании меню можно почитать в документации Android-разработчика

Аналогично происходит описание анимации в папке *anim* с автоматической генерацией описания ресурса `<set>`. Подробнее об описании анимации можно прочитать в документации.

Упражнение 2.3.1 Редактирование ресурсов приложения

1. В проекте My_app измените текстовые константы в файле res/values/string.xml:

- ”Hello, World” на строку ”Hello, Name”, вместо Name поставьте свое имя;
- ”app-name” на строку ”My first app”.

Сохраните проект и запустите приложение на устройстве.

2. Скопируйте папку res/values и назовите копию res/values-ru. Переведите значения строковых констант в файле string.xml на русский язык и сохраните проект. Переключите на устройстве язык на русский и запустите приложение.

3. Создайте в папке values новый файл ресурсов с именем *week.xml*. Опишите в нём ресурс - строковый массив дней недели

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="week">
        <item>Понедельник</item>
        <item>Вторник</item>
        <item>Среда</item>
        <item>Четверг</item>
        <item>Пятница</item>
        <item>Суббота</item>
        <item>Воскресенье</item>
    </string-array>
</resources>
```

В методе onCreate() создайте переменную массива строк, прочитайте в неё массив week и выведите его в консоль с использованием логирования.

```
var s = resources.getStringArray(R.array.week)
Log.d("week", s.contentToString())
```

2.3.3 Представления (Views)

Как описывалось ранее, представления являются содержимым виджетов и занимают на активности определённую прямоугольную область. Создать объект класса View можно как в коде активности, на которой размещается данное представление, так и в её макете. Для создания используется конструктор от аргумента - контекста, в котором создаётся объект. Контекстом для любого view является активность, в которой он размещается либо родительский виджет.

```
var view =View(this)
```

Программное описание view позволяет в любой момент после создания объекта заполнить необходимое свойство. Таким образом, в коде не требуется группировать инициализацию свойств представлений, однако, желательно производить эту группировку, поскольку разрозненность инициализаций сразу же сказывается на читаемости кода. В макете активности представления являются дочерними объектами и размещаются в контейнеры (ViewGroup). При описании объекта производится инициализация всех необходимых свойств сразу внутри открывающего тега объекта. У каждого объекта обязательно должны быть указаны свойства высоты `android:layout_height` и ширины `android:layout_width`, без которых невозможно выделить необходимую область на активности. Однако, для установления привязки xml-описания к переменной в коде объекту требуется его идентификатор - свойство `android:id`, по которому и будет устанавливаться соответствие. Для привязки макета к переменной в коде используется функция *findViewById(id: Int)*, применять которую следует только после того, как в классе активности указан файл разметки, из которого берётся макет. Передача источника для макета активности осуществляется в методе *setContentView(View view)*

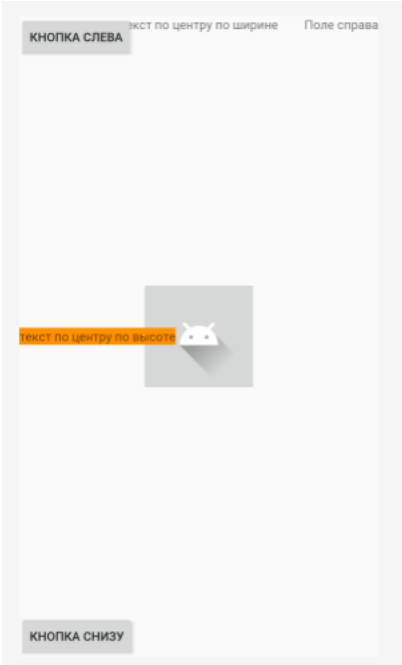
```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)  //установили макет из файла разметки activity_main.xml
    val root = findViewById<LinearLayout>(R.id.root) // нашли корневую линейную разметку по её id
    val tv: TextView = findViewById(R.id.textView) // нашли текстовое поле по его id
val b = Button(this) // создали программно кнопку, которая не описана в макете активности
b.text = "Новая кнопка"
root.addView(b) //добавили кнопку в корневую разметку
}
```

2.3.4 Разметка (Layout)

На активности объекты размещаются в контейнерах (ViewGroup).Для каждой активности обязательно наличие ровно одного корневого элемента View - контейнера или единственного элемента. У каждого контейнера, так же как и у каждого виджета имеется свой Kotlin-класс, в котором описана структура этого представления. Макет активности можно описать в файле разметки, размещаемом в папке *layout* внутри папки res, а можно создать программно. Основным (в том числе и корневым) контейнером макета активности выступает слой или разметка (layuot), в которой размещаются представления. В зависимости от количества и положения размащаемых объектов выбирают одну из разметок в качестве корневой. Разметка может быть и дочерним элементом другой разметки, то есть имеет место вложенность слоёв.

Самый простой вариант разметки - *рамка (FrameLayout)*. Этот контейнер выделяет место на экране для размещения малого числа объектов, поскольку не предоставляет возможности выравнивать объекты относительно друг друга. Все дочерние элементы выравниваются внутри рамки относительно левого верхнего угла, что не даёт возможности размещения сложной и широкой иерархии объектов. Однако, есть возможность выравнивания объектов свойством *gravity* относительно полей активности. Описание разметки в xml-файле определяется тегом `<FrameLayout>`:

```
<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=".MainActivity">
</FrameLayout>
```



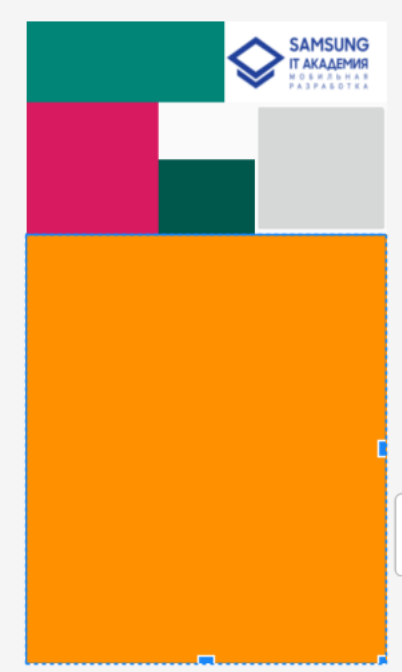
Весьма популярна *линейная разметка (LinearLayout)*, внутри которой объекты располагаются в ряд и отображаются в порядке описания. То есть свойство выравнивания по верхней и нижней границам активности уже не имеют значения, но работает выравнивание по центру, по левому и правому краю. Такой вид разметки имеет обязательное свойство `android:orientation`, принимающее значение `vertical` или `horizontal` и указывающее направление ряда: в столбец или в строку.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:orientation="horizontal"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
</LinearLayout>
```

Если в вертикально ориентированной разметке нужно разместить горизонтальную строку объектов, то в столбец добавляют объект LinearLayout с горизонтальной ориентацией и в нём размещают строку из объектов. Такое решение является нерациональным, поскольку при построении активности тратится время на создание каждого layout.

На основе LinearLayout были построены классы *табличной разметки (TableLayout)* и *относительной разметки (RelativeLayout)*.

Табличная разметка представляет из себя линейную разметку вертикальной ориентации, содержащую объекты `<TableRow>` - линейные разметки горизонтальной ориентации.



В относительной разметке появилась возможность располагать объекты внутри контейнера относительно друг друга и границ самой layout, устанавливая привязки по четырём сторонам view.

2.3.5 Основные свойства экранных элементов

Поскольку все экранные элементы являются экземплярами наследников одного родительского класса View, то они имеют общие свойства, которые наследуются от этого класса. Общие основные свойства и их описания представлены в таблице.

Свойство	Формат значений	Описание	Пример применения
android:id	@+id/идентификационное_имя	Идентификатор объекта (имя, используемое при обращении) . Должно начинаться с латинского символа нижнего регистра, не может содержать разделителей	android:id = “@+id/button_OK” android:id = “@+id/tv”
android:layout_width android:layout_height	in, mm, pt,px; dp; wrap_content; match_parent	Ширина и высота объекта в единицах измерения дюймы, миллиметры, пункты, пиксели - неизменяемые величины,определяются по размеру экрана устройства; Density-independent Pixels, шкалированные пиксели - адаптируют размеры объектов к размеру экрана приложения; размер объекта равен размеру отображаемых данных; размер объекта равен размеру родительского объекта	android:layout_width="263dp" android:layout_width="wrap_content" android:layout_height="match_parent"
android:layout_gravity	top, bottom, left, right, center	Выравнивание объекта внутри активности по верхнему/нижнему/правому/левому краю или по центру	android:layout_gravity="center_vertical"
android:gravity	top, bottom, left, right, center	Выравнивание содержимого объекта (текст, изображение) внутри самого объекта	android:gravity="center"
android:layout_weight	Положительное целое число	«Вес» объекта: доля родительского контейнера, выделяемая под этот объект. Рассчитывается от суммы долей всех дочерних объектов	android:layout_weight="1" android:layout_weight="4"
android:layout_margin	in, mm, pt,px, dp	Отступ от границы объекта до его содержимого. Если не указана граница, то отступ делается по всем сторонам объекта	android:layout_margin="1dp" android:layout_marginBottom=”1mm”
android:text	Любая текстовая строка или обращение текстовому к ресурсу	Текстовая надпись на объекте	android:text="@string/app_name" android:text="Нажми”
android:src	Цветовой или графический ресурс приложения	Содержимое графического view	android:src="@color/colorPrimary" android:src="@drawable/ic_launcher_foreground"
android:textSize	in, mm, pt,px, dp(sp)	Размер шрифта надписи в абсолютных или относительных единицах измерения	android:textSize="34sp"
android:background android:textColor	Изображение или цвет из ресурса, код цветовой константы в форматах #RRGGBB #RGB	Цветовое оформление объекта или текста надписи на объекте	android:background="@color/colorAccent" android:background="@mipmap/logo" android:textColor="#12A08D"
style	ресурс с тегом style	Задание объекту стиля, описанного массивом свойств в ресурсах приложения	style="@style/my_stand_style"

Для получения экрана, представленного на рисунке 2.3.3, кроме высоты и ширины объектов нужно настроить свойства весов и цвета заливки фона.

```
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:orientation="vertical"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
<TableRow>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="92dp"/>
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:src="@drawable/ic_launcher_foreground"/>
</TableRow>
    <TableRow>
    <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Кнопка снизу"
    android:layout_gravity="bottom"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Поле справа"
        android:layout_gravity="right"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="left"
        android:text="Кнопка слева" />
    </TableRow>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="текст по центру по высоте"
        android:background="#ff9000"
        android:layout_gravity="center_vertical"/>
</TableLayout>
```

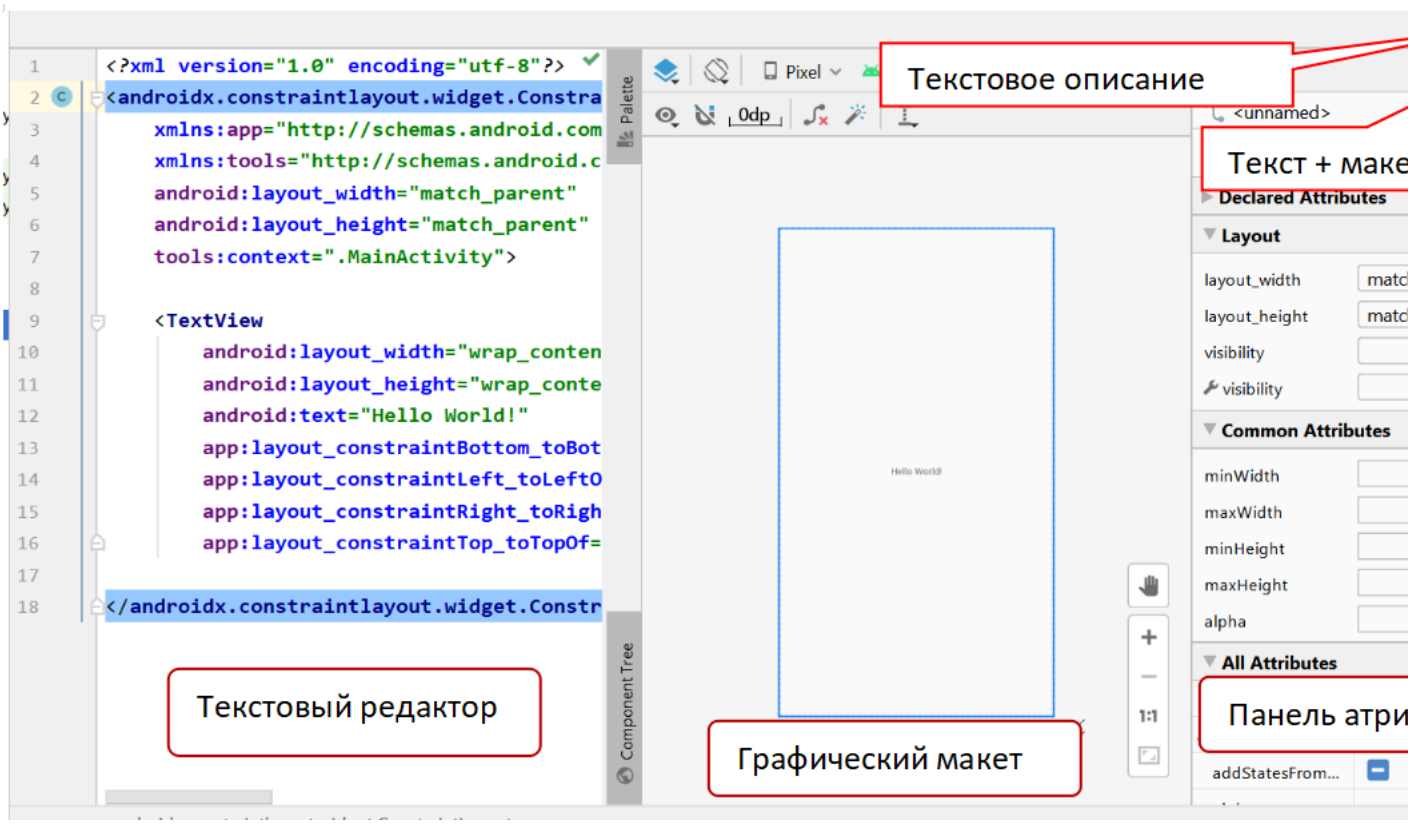
Упражнение 2.3.2 Редактирование макета активности

1. Откройте файл res/layout/activity_main и замените корневую разметку ConstraintLayout на линейную LinearLayout; внутри нее создайте еще две линейные разметки по ширине родительской и по высоте относящиеся 1:2 сверху вниз. Вторую разметку разделите на две равные рамки FrameLayout. Самым нижним элементом корневой разметки должно остаться текстовое поле. Установите для всех объектов различные значения цвета фона.

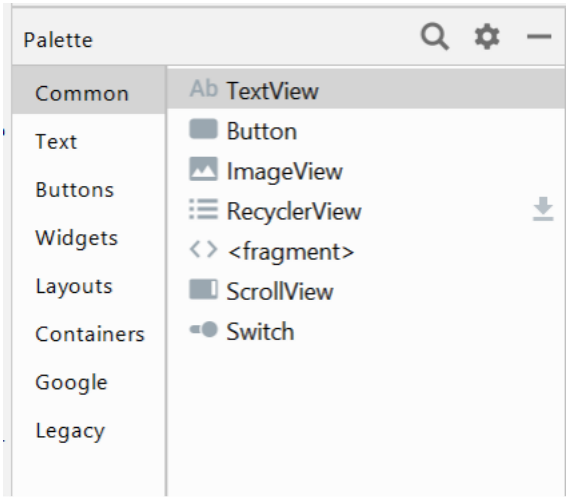
2. Используя табличную разметку, нарисуйте экран игры «Пятнашки» размером 4x4.

2.3.6 Редактор макета

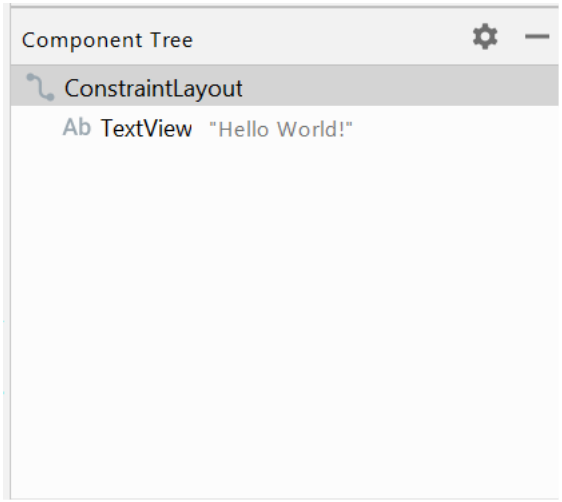
XML- описание макета активности содержится в файлах разметки в папке res/layout. Редактирование макета можно выполнить, изменяя текстовый файл, то есть отредактировать xml-описание объекта: добавить, изменить или удалить свойство. Рамка и линейная (табличная) разметки свободно редактируются в текстовом описании. Но текстовое описание относительной разметки очень громоздко, поскольку кроме свойств самого объекта нужно описывать его положение внутри разметки. То есть нужно указать расположение относительно границ самой активности и относительно соседних элементов. Это достаточно трудоёмко. Android Studio имеется графический редактор макета активности, отображающий схематический вид активности на устройстве. С помощью этого редактора можно просматривать и изменять макет в визуальном режиме. Редактор содержит три представления: текст, текст + дизайн, дизайн. Переключатели выбора режима отображения расположены справа в строке заголовка редактора (Рисунок 2.3.4)



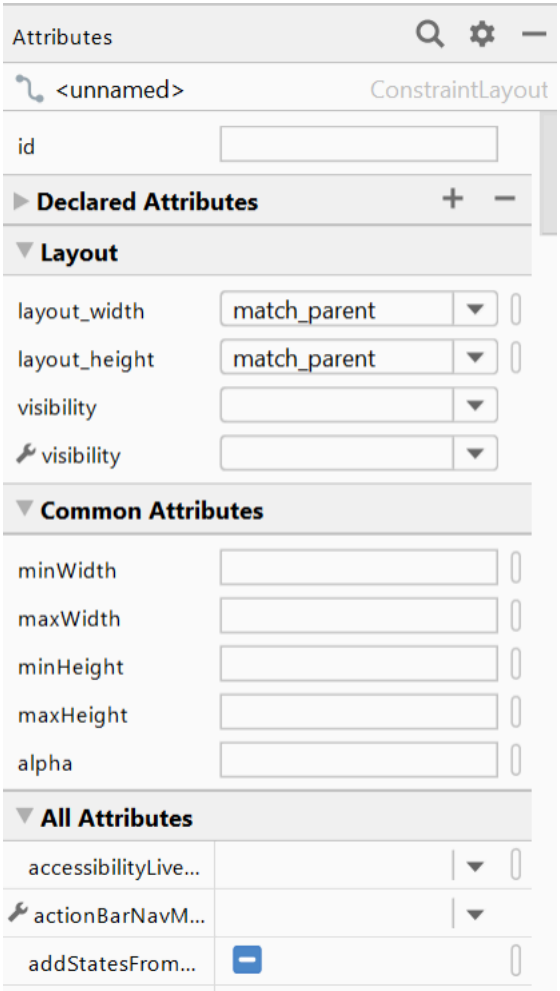
Редактор дизайна макета очень удобен для размещения объектов в относительной разметке. В графическом представлении редактора имеются панели компонентов *Palette* и *Component Tree*, а так же панель свойств *Attributes* этих компонентов. На панели Palette представлены в графическом виде все возможные View и ViewGroup, размещение которых возможно на активности. Выбор и размещение элементов выполняется по технологии drag-and-drop выбором элемента в Palette и переносом его на активность. Все представления в панели сгруппированы по назначению, а так же есть полный список всех объектов.



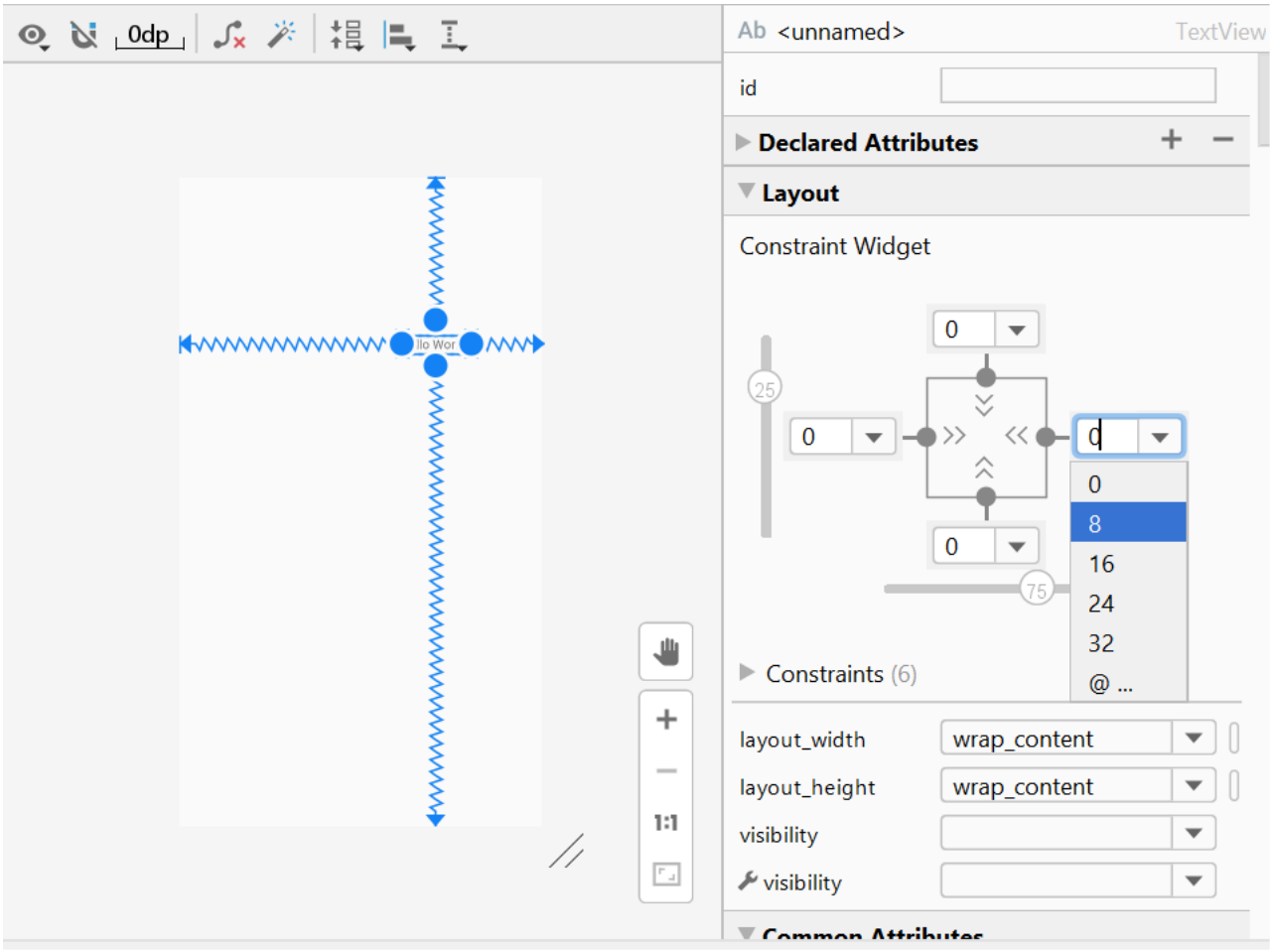
Иерархия объектов макета отображается в панели компонентов Component Tree. В данной панели отражается не только иерархия объектов, но и ошибки разметки элементов.



Свойства активного (выделенного) объекта отображаются в панели Attributes, где их можно редактировать.



Поскольку относительная разметка требует обязательной привязки объекта хотя бы по двум границам, то режим дизайна обеспечивает возможность задать эти привязки непосредственно на самом макете или в панели свойств.Для этого нужно маркер привязки «тянуть» к элементу, по которому производится связывание.


0dp
Ab <unnamed>
TextView
id
Declared Attributes
Layout
Constraint Widget
0
25
0
0
0
75
0
8
16
24
32
@ ...
Constraints (6)
layout_width
wrap_content
layout_height
wrap_content
visibility
visibility
1:1

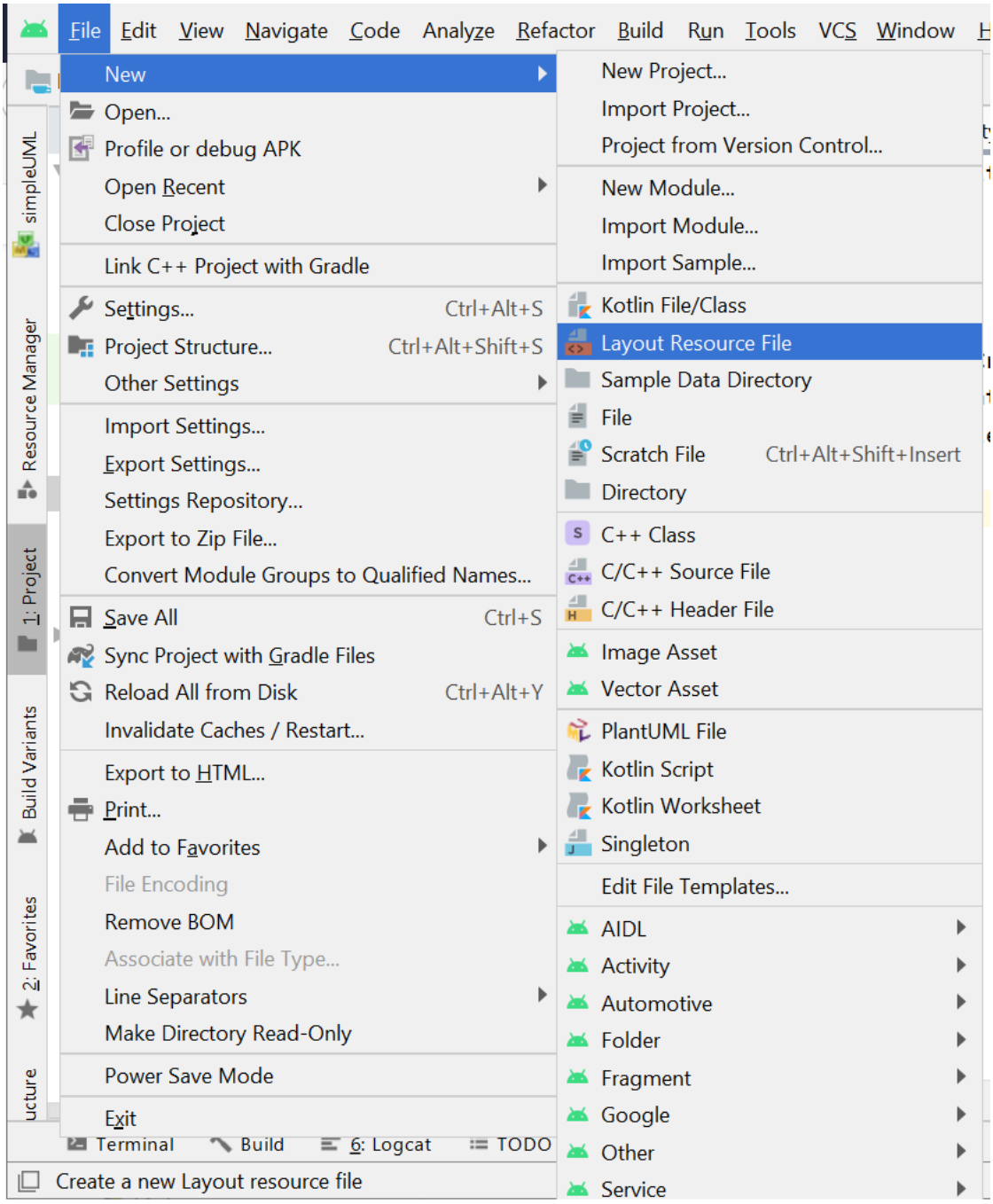
2.3.7 Несколько макетов для одной активности

В зависимости от размеров экрана, его ориентации и возможности разделения вид приложения может меняться. Так, активности приложения, написанного под горизонтальную ориентацию при повороте экрана в вертикальное положение могут сжать контент до нечитаемого вида, а могут и просто отобразить только часть своего содержимого, попадающую в область экрана. А при разделении экрана велика вероятность потери контента. Для решения этой проблемы в операционной системе Android предусмотрена возможность наличия нескольких разметок для одной активности.

Чтобы воспользоваться этой возможностью нужно создать в папке *res/layout* дополнительные файлы разметки с тем же самым именем, что и у исходного и настроить соответствующие параметры файла. Для этого:

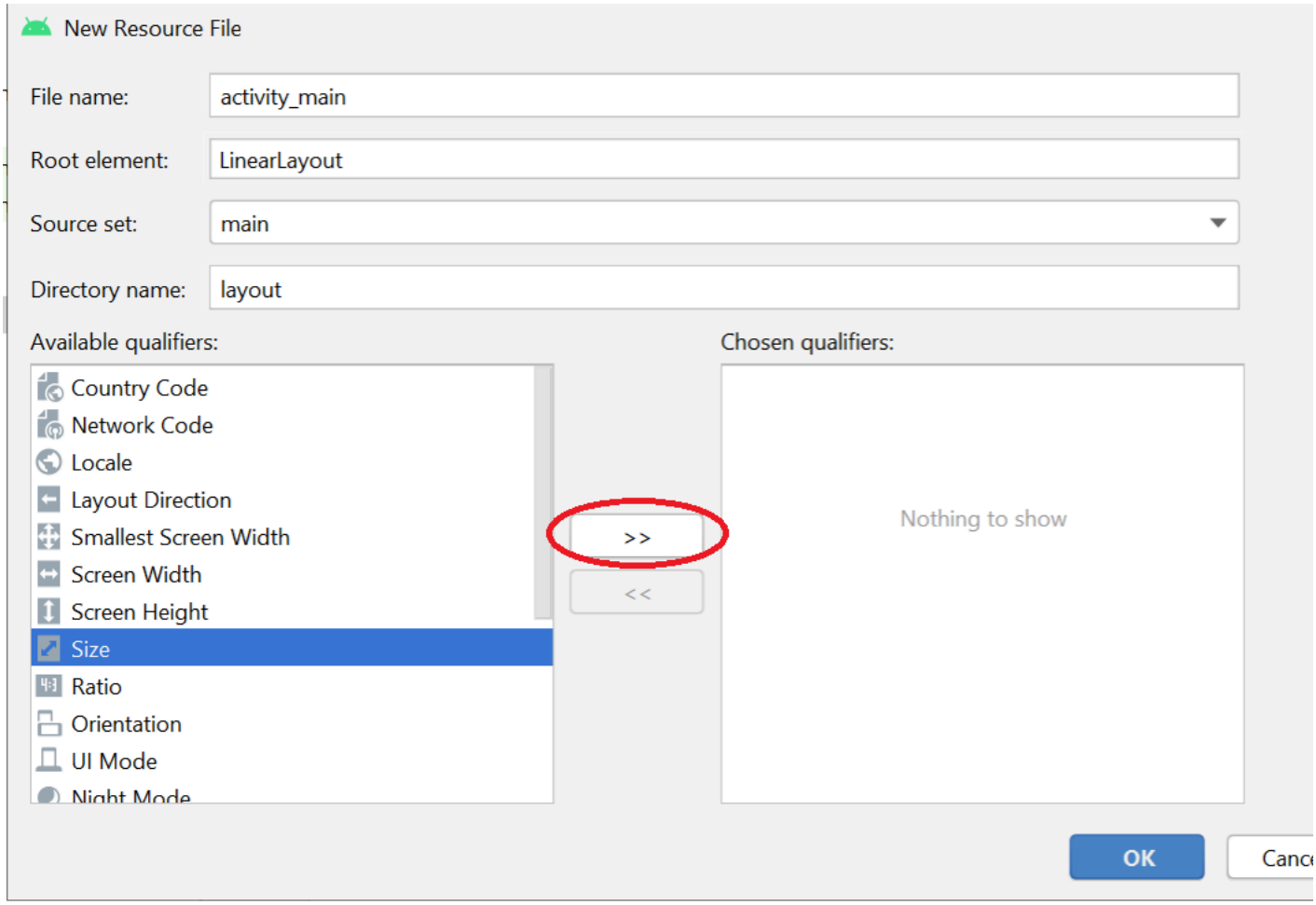
1. в окне проекта (Project) нужно выделить папку res/layout;

2. выбрать в меню программы раздел File -> New -> Layout Resource File;

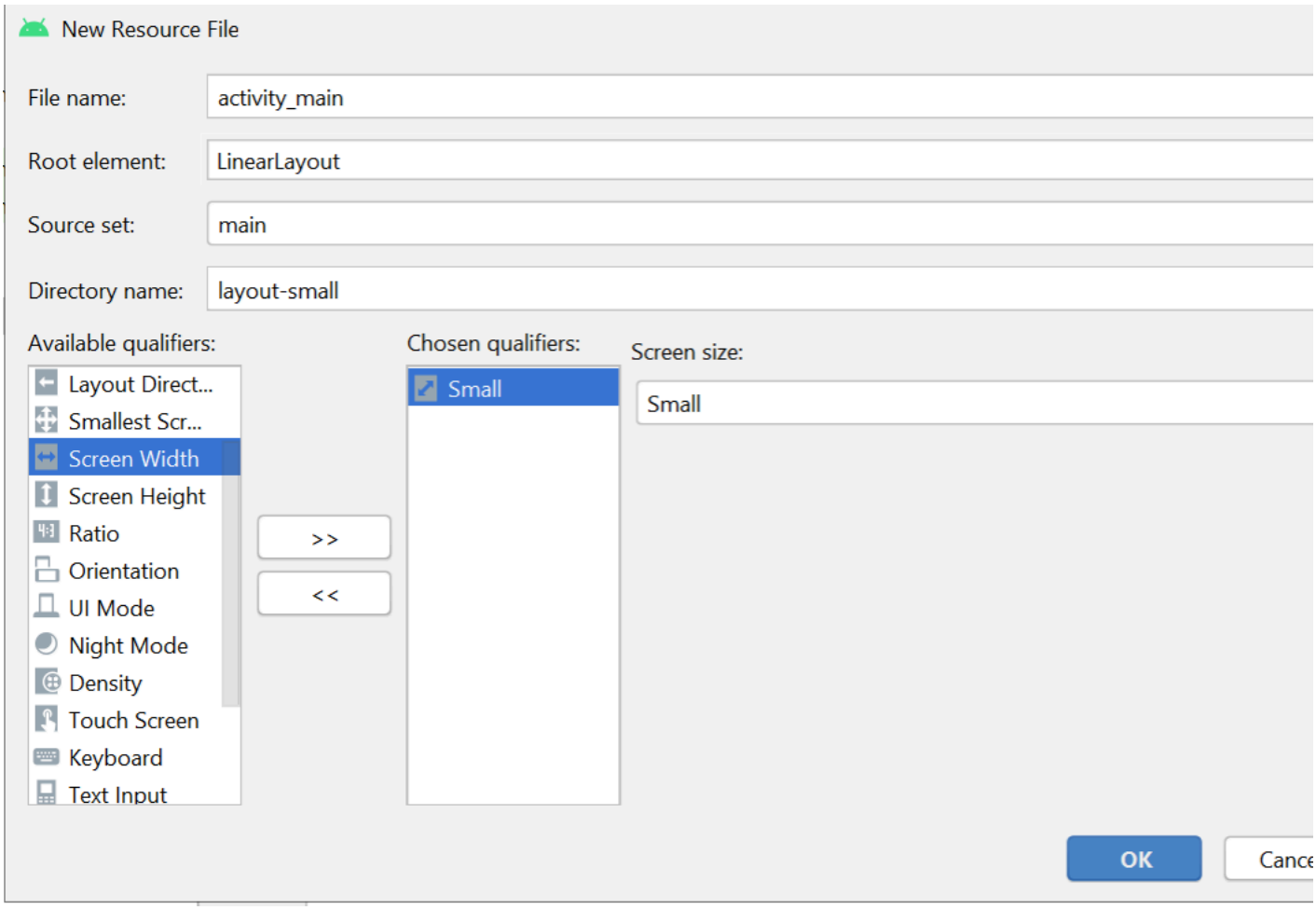


3. ввести имя файла, выбрать корневую разметку;

4. выбрать в разделе *Available Qyalifiers* необходимые свойства и перенести их в зону *Chosen Qualifiers* стрелками переноса.



В появившейся справа области нужно выбрать необходимое значение. На рисунке 2.3.11 показан выбор размера окна отображения активности при разделении экрана.



Для изменения макета активности в ночном режиме нужно выбирать описание *Night Mode*;

чтобы создать экраны разных ориентаций, нужно выбрать и настроить свойство *Orientation*.

Если же создать файл разметки с именем, отличным от исходного файла, то заменой аргумента функции `setContentView()` можно заменить источник разметки активности на новый.

Упражнение 2.3.3 Создание нескольких разметок активности

Создадим простое приложение, которое разделяет день на "До обеда" и "После обеда".

1. Создайте новый проект, назовите его ManyLayout.
2. Разверните в окне обозревателя проекта папку res/layout, выделите файл activity_main.xml, скопируйте его (через контекстное меню, через меню Edit->Copy или комбинацией клавиш Ctrl+C) и вставьте в эту же папку, но с именем second.xml.

Теперь в проекте имеется два файла разметки, но используется только один: `setContentView(R.layout.activity_main)`. Если заменить аргумент на `R.layout.second`, то макет будет загружаться из файла second.xml, но это никак не отразится на приложении, поскольку сейчас файлы одинаковые.

3. Изменим содержимое текстовых полей в каждом файле для их отличия. Откроем файл activity_main.xml в редакторе кода, найдём строку

```
android:text="Hello, World!"
```

и заменим на строку

```
android:text="До обеда"
```

Аналогично в second.xml поставим надпись "После обеда".

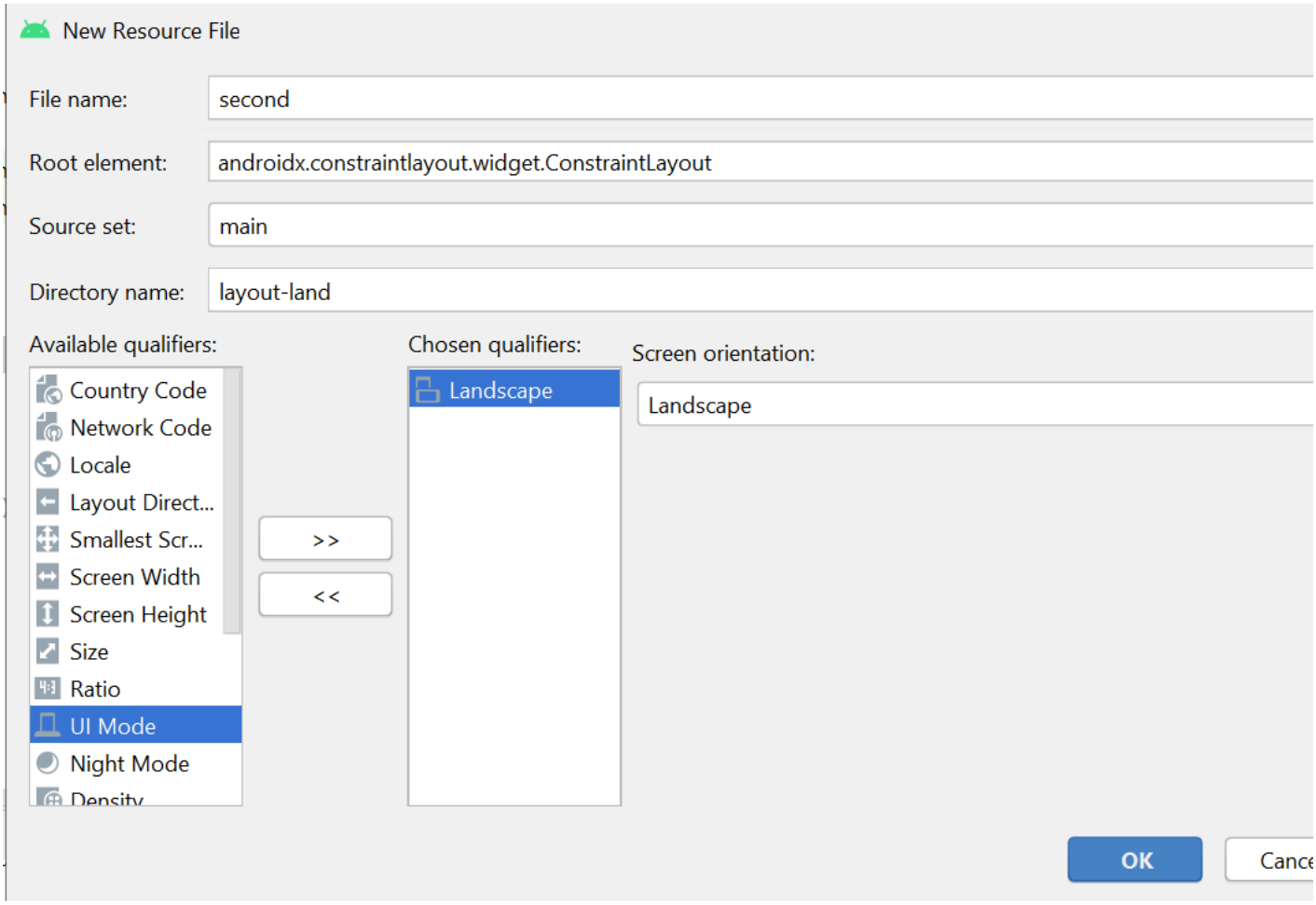
Теперь экраны отличаются. Проверьте это самостоятельно, заменяя источники макета активности в методе `setContentView`.

4. Сделаем так, чтобы по времени в устройстве до 14:00 приложение запускалось с макетом activity_main.xml, а после 14:00 с макетом second.xml. Для этого добавим в активности переменную даты: `val date = Date()` и получим из неё текущее время в часах: `date.hours`. Если это время меньше 14, то будем выгружать первый макет, иначе - второй:

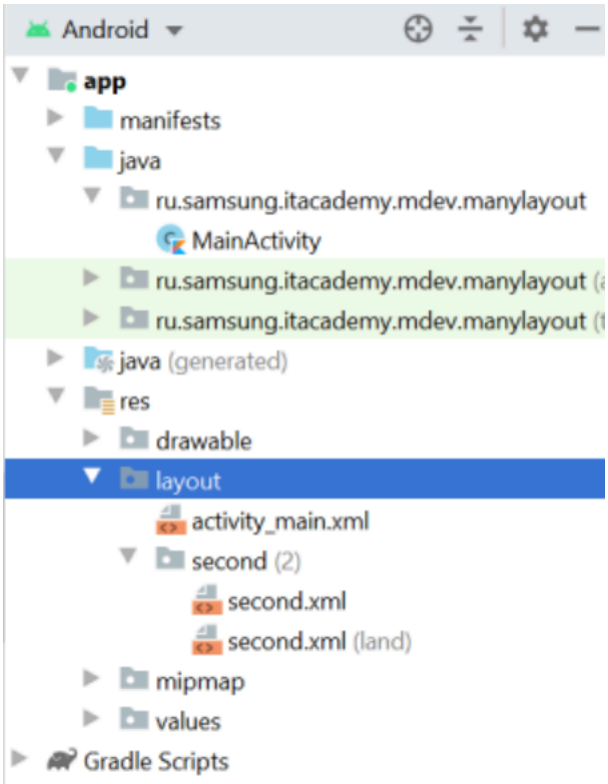
```
val date = Date()
        var time: Int  //переменная для id макета
        if ( date.hours < 14 )  time = R.layout.activity_main
        else time = R.layout.second
        setContentView(time)
```

Запустите приложение и проверьте его работу. Затем измените знак неравенства, чтобы убедиться в корректности работы приложения.

5. Для разметки second.xml создадим горизонтальную вариацию без экранных элементов. Создадим новый файл ресурса (File ->New ->Layout Resourse File), назовём его second, разметку оставим без изменения, выберем определение Orientation и зададим ему значение Landscape.



После нажатия кнопки ОК в структуре проекта можно увидеть, что файлов second.xml стало два.



Поставьте знак неравенства в условии так, чтобы отображался макет second.xml запустите приложение на устройстве. Поверните экран из вертикального в горизонтальное положение.

Готовое приложение

Начать тур для пользователя на этой странице