2018

Технологии разработки мобильных приложений

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 1

УЧЕБНЫЕ ВОПРОСЫ:

1. Установка и настройка операционной системы, Java SDK (JDK), среды разработки и пакетов разработки Android SDK. 2

2. Создание AVD. Первое приложение. Структура Android-проекта. 3

3. Компоненты экрана и их свойства. 6

4. Layout-файл в Activity. Смена ориентации экрана. 12

5. Виды Layouts. Ключевые отличия и свойства. 15

6. Обращение из кода к элементам экрана. Обработчики событий. 20

7. Обработчики событий на примере Button. 22

1. *Установка и настройка операционной системы, Java SDK (JDK), среды разработки и пакетов разработки Android SDK.*

Java SDK (JDK)

Т.к. разработка приложений ведется на Java, требуется скачать и установить соответствующее SDK, называемое еще JDK.

Скачать можно здесь. Нажимать там ближайшую кнопку JDK Download, выбирать версию под вашу операционную систему, скачать и устанавить.

После установки рекомендую перезагрузить компьютер.

Среда разработки + Android SDK

В среде разработки мы будем создавать программу и получать на выходе готовое приложение. Сейчас существует несколько сред разработки, мы выберем Android Studio.

От нас требуется указать два пути. Первый путь будет использован для установки Android Studio. Второй - для установки Android SDK. После выполнения этих шагов мы получили среду разработки.

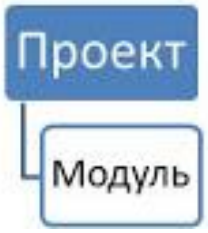
AVD – это эмулятор смартфона с операционной системой Android, на котором можно запускать и тестировать приложения.

2. *Создание AVD. Первое приложение. Структура Android-проекта.*

Для тестирования приложения, нам понадобится Android Virtual Device (AVD). Это эмулятор Android-смартфона, на который мы сможем устанавливать созданные нами приложения, и запускать их там.

Для начала создадим приложение. Чтобы создать приложение, нам нужно в Android Studio создать проект. При создании проекта, в нем создается модуль. В этом модуле мы рисуем экраны приложения и пишем код. И при запуске этого модуля мы получаем готовое приложение. Поэтому модуль по сути и является приложением. А проект - контейнер для модуля.

Т.е. в самом простом случае структура проекта такова:



Есть проект, и в нем есть модуль. При запуске проекта запускается модуль и мы получаем Android-приложение, которое создано в этом модуле.

В этом случае: один проект = одно Android-приложение (один модуль).

Но в одном проекте может быть несколько модулей. Да и проектов можно создать несколько.

Здесь в первом проекте созданы два модуля, а во втором проекте – три модуля.

При запуске какого-либо проекта необходимо будет указать, какой именно модуль вы хотите запустить.

Каждый модуль является отдельным Android-приложением.

Т.е. в этом случае: один проект = несколько Android-приложений (несколько модулей).

Application name – имя проекта. Оно будет отображаться в списке проектов при открытии Android Studio. Напишем здесь Android lessons (т.е. Android уроки).

Company Domain – имя сайта.

Package name – это понятие из Java. Вкратце – это префикс для имени классов нашего приложения. Как видите, пакет автоматически составился из

имени сайта и имени проекта. Его всегда можно отредактировать вручную нажав на ссылку edit справа.

Project location – папка на компе, где будут находиться все файлы проекта.

App – это модуль. По умолчанию при создании проекта создается модуль app.

Итак, проект создан. Теперь создадим в проекте свой модуль. Для каждого урока мы будем создавать модуль в этом проекте. Сейчас создадим модуль (приложение) для этого текущего урока. Эта процедура будет частично похожа на создание проекта, но с небольшими отличиями.

Чтобы создать модуль – в меню выбираем **File -> New -> New module**

Тип модуля выбираем **Phone and Tablet Application**

Application/Library name – непосредственно имя приложения, которое будет отображаться в списке приложений в смартфоне. Пишем тут FirstProject.

Module name – это название модуля. Т.е. это название будет отображаться слева в списке модулей, там, где сейчас есть app. Давайте придумаем шаблон для названия модулей.

Например: p<номер урока(002)><номер проекта в уроке(1)>.

На номер урока выделим три цифры, на номер проекта – одну. Также, будем добавлять название приложения - FirstProject. И все это напишем маленькими буквами и без пробелов. Получится такое имя модуля: p0021firstproject.

Package name – имя пакета отредактируем вручную, нажав edit справа. Оставим там ru.mirea и добавим точку и имя модуля.

Minimum SDK оставляйте без изменений.

Empty activity – без создания дополнительных элементов.

Жмем **Next**

Заходим в настройки эмулятора – создаем и запускаем проект.

P.S. Если эмулятор не показал ваше приложение, то убедитесь, что Android Studio "видит" этот эмулятор. Для этого снизу слева нажмите вкладку Android Monitor. Если эмулятор есть в списке, а приложение не отобразилось,

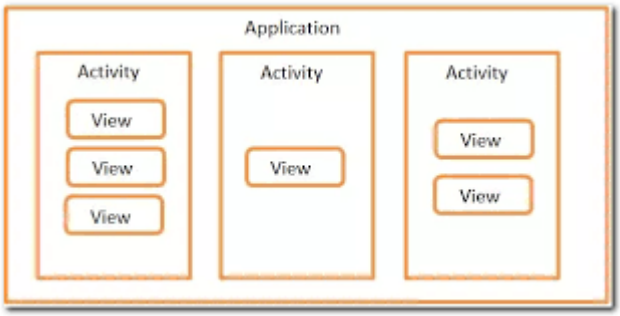
то попробуйте снова запустить приложение, нажав зеленый треугольник (Shift+F10).

Если эмулятора в списке нет, то закройте эмулятор и попробуйте снова запустить приложение.

3. Компоненты экрана и их свойства.

Если проводить аналогию с Windows, то приложение состоит из окон, называемых Activity. В конкретный момент времени обычно отображается одно Activity и занимает весь экран, а приложение переключается между ними. В качестве примера можно рассмотреть почтовое приложение. В нем одно Activity – список писем, другое – просмотр письма, третье – настройки ящика. При работе вы перемещаетесь по ним.

Содержимое Activity формируется из различных компонентов, называемых View. Самые распространенные View - это кнопка, поле ввода, чекбокс и т.д.

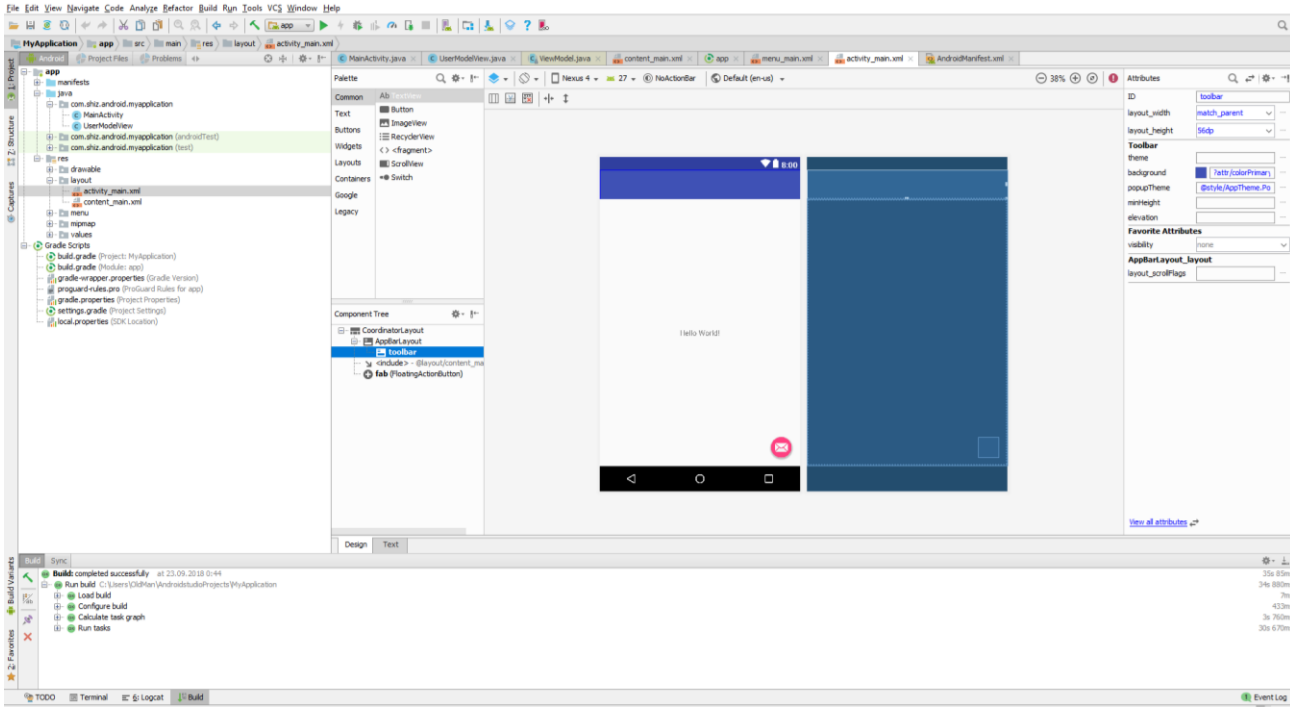


Необходимо заметить, что View обычно размещаются в ViewGroup. Самый распространенный пример ViewGroup – это Layout. Layout бывает различных типов и отвечает за то, как будут расположены его дочерние View на экране (таблицей, строкой, столбцом и т.д.).

Создадим модуль. В меню ***File >New > New Module > Phone & Tablet Module > Empty Activity***. В этом модуле нам интересен файл: *res > layout > activity_main.xml*. Это layout-файл. В нем мы определяем набор и расположение View компонентов, которые хотим видеть на экране. При запуске приложения, Activity читает этот файл и отображает нам то, что мы настроили. Скорее всего, он у вас уже открыт на редактирование, но на всякий случай давайте еще раз откроем его двойным кликом и посмотрим, как он выглядит.

Компоновка (также используются термины разметка или макет) хранится в виде XML-файла в папке /res/layout. Это сделано для того, чтобы отделить код от дизайна, как это принято во многих технологиях (HTML и CSS). Кроме основной компоновки для всего экрана, существуют дочерние элементы компоновки для группы элементов. По сути, компоновка – это некий визуальный шаблон для пользовательского интерфейса вашего приложения, который позволяет управлять элементами управления, их свойствами и расположением. Если вы будет обращаться к элементам управления через

Java-код, то необходимо присваивать элементам уникальный идентификатор через атрибут android:id. Сам идентификатор назначается через выражение @+id/your_value. После этого вы можете обращаться к элементу через код при помощи метода findViewById(R.id.your_value).



•Design и Text

Desing - это графическое представление экрана. Text - это текстовое представление. Оно выглядит так:

•Режимы отображения экрана

На скриншоте в области 3 вы видите два экрана. Обычный белый и синий. Это один и тот же экран, но он отображен в двух разных режимах: Design - в нем мы видим View компоненты так, как они обычно выглядят на экране. Blueprint - отображаются только контуры View компонентов

Кнопки в области 2 позволяют вам переключать режимы:

- *Design;*
- *Blueprint;*
- *Design + Blueprint.*

- Экран

Здесь вы можете видеть, как выглядит экран вашего приложения. Сюда мы будем добавлять различные компоненты из области Pallete.

- Палитра

Это список всех View компонентов, которые вы можете добавлять на ваш экран: кнопки, поля ввода, чекбоксы, прогрессбары и прочее.

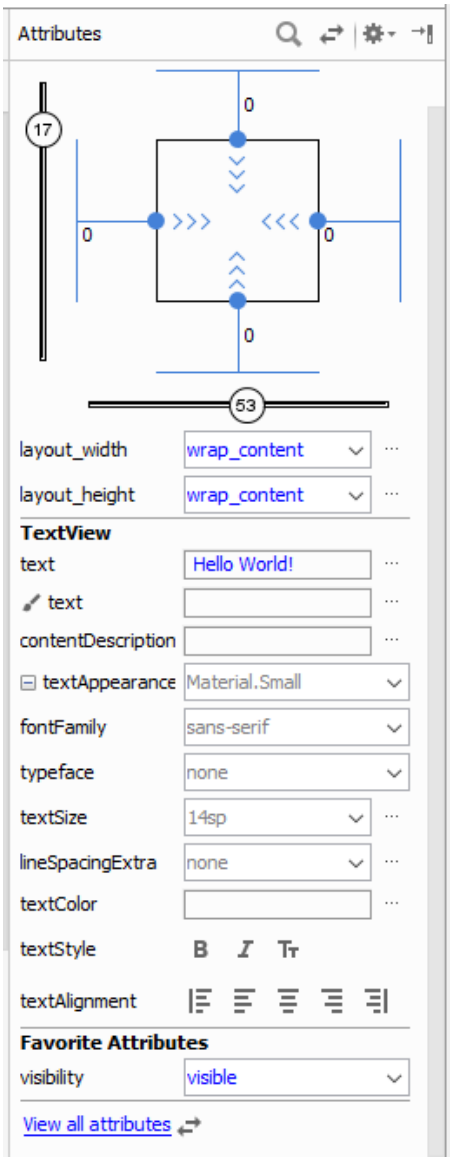
- Дерево компонентов

Здесь представлена иерархия View компонентов вашего экрана. Сейчас, например, корневой элемент - это ConstraintLayout. А в него вложен TextView.

- Свойства

При работе с каким-либо View компонентом здесь будут отображаться свойства этого компонента. С помощью свойств вы сможете настраивать внешний вид, расположение и содержимое View компонента.

Задание:



Разместить на текущую разметку (res>layout>activity_main.xml) из меню Palette следующие элементы и изучить их свойства (область Attributes)

- ***Text > TextView, PlainText (EditText);***
- ***Buttons > Button, ImageButton, CheckBox;***
- ***Widgets > ImageView (установить изображение).***

P.s.: Откройте вкладку Text (рядом с Design) – xml-описание всех View нашего layout-файла. Названия xml-элементов - это классы View-элементов, xml-атрибуты - это параметры View-элементов, т.е. все те параметры, что мы меняем через вкладку Attributes. Также вы можете вносить изменения прямо сюда и изменения будут отображаться во вкладке Design. Например, изменим текст у TextView.

Нам интересен xml-код. Каждый объект View и ViewGroup поддерживают свои собственные атрибуты XML. Некоторые атрибуты характерны только для объекта View (например, объект TextView поддерживает атрибут textSize), однако эти атрибуты также наследуются

любыми объектами View, которые могут наследовать этот класс. Некоторые атрибуты являются общими для всех объектов View, поскольку они наследуются от корневого класса View (такие как атрибут id). Любые другие атрибуты рассматриваются как «параметры макета». Такие атрибуты описывают определенные ориентации макета для объекта View, которые заданы родительским объектом ViewGroup такого объекта.

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:id="@+id/entry"
```

- слово android в названии каждого атрибута – это namespace, говорит о принадлежности к элементам android

- id – это целочисленный идентификатор, который служит для обозначения уникальности объекта View в иерархии. Во время компиляции приложения этот идентификатор используется как целое число, однако идентификатор обычно назначается в файле XML макета в виде строки в атрибуте id. Этот атрибут XML является общим для всех объектов View (определенных классом View), который вы будете использовать довольно часто. Символ @ в начале строки указывает на то, что обработчику XML следует выполнить синтаксический анализ остальной части идентификатора, выполнить ее синтаксический анализ и определить ее в качестве ресурса идентификатора. Символ плюса (+) обозначает, что это имя нового ресурса, который необходимо создать и добавить к нашим ресурсам (в файле R.java)

- layout_width (ширина элемента) и layout_height (высота элемента) могут задаваться в абсолютных значениях, а могут быть следующими: match_parent (максимально возможная ширина или высота в пределах родителя) и wrap_content (ширина или высота определяется по содержимому элемента).

Сейчас вернемся к нашим элементам. Ниже приведены наиболее часто используемые атрибуты.

TextView

android:id="@+id/label" - ID

android:layout_width="match_parent" - занимает всю доступную ему ширину (хоть это и не видно на экране);

android:layout_height="wrap_content" - высота по содержимому;

EditText

android:id="@+id/entry" - ID

android:layout_width="match_parent" - вся доступная ему ширина

android:layout_height="wrap_content" - высота по содержимому

android:layout_below="@+id/label" - расположен ниже TextView (ссылка по ID)

Button_OK

android:id="@+id/ok" – ID

android:layout_width="wrap_content" - ширина по содержимому

android:layout_height="wrap_content" – высота по содержимому

android:layout_below="@+id/entry" - расположен ниже EditText

android:layout_alignParentRight="true" - выравнен по правому краю родителя

android:layout_marginLeft="10dip" – имеет отступ слева (чтобы Button_Cancel был не впритык)

Button_Cancel

android:layout_width="wrap_content" - ширина по содержимому

android:layout_height="wrap_content" – высота по содержимому

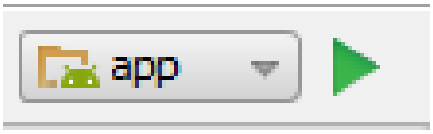
android:layout_toLeftOf="@+id/ok" - расположен слева от Button_OK

android:layout_alignTop="@+id/ok" - выравнен по верхнему краю Button_OK

Вы можете подобавлять элементы и поэкспериментировать с их размещением.

Обратите внимание, что у View-элемента может не быть ID (android:id). Например, для TextView он обычно не нужен, т.к. они чаще всего статичны, и мы к ним почти не обращаемся при работе приложения. Другое дело EditText – мы работаем с содержимым текстового поля, и Button – нам надо обрабатывать нажатия и соответственно знать, какая именно кнопка нажата. В будущем мы увидим еще одну необходимость задания ID для View-элемента.

Запустить приложение с добавленными элементами:



4. Layout-файл в Activity. Смена ориентации экрана.

Откуда Activity знает, какой именно layout-файл читать.

При разработке, каждому Activity сопоставляется одноименный java-класс (наследник класса android.app.Activity). При запуске приложения, когда система должна показать Activity и в дальнейшем работать с ним, она будет вызывать методы этого класса. И от того, что мы в этих методах напишем, зависит поведение Activity.

Откроем файл MainActivity:

метод onCreate – он вызывается, когда приложение создает и отображает Activity. Посмотрим код реализации onCreate.

Первая строка:

```
super.onCreate(savedInstanceState);
```

это вызов метода родительского класса, выполняющий необходимые процедуры по созданию Activity.

Перейдем к следующей строке:

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

Метод setContentView(int) – устанавливает содержимое Activity из layout-файла. Но в качестве аргумента мы указываем не путь к layout-файлу (res/layout/activity_main.xml), а константу, которая является ID файла. Эта константа генерируется автоматически в файле ***R.java****. В этом классе будут храниться сгенерированные ID для всех ресурсов проекта (из папки res/*), чтобы мы могли к ним обращаться. Имена этих ID-констант совпадают с именами файлов ресурсов (без расширений). Файл res/layout/activity_main.xml был создан средой разработки вместе с Activity. Его название запрашивалось на том же экране, где и название Activity.*

Попробуем отобразить содержимое другого файла. Создадим еще один layout-файл, например, **activity_second.xml**.

Для этого выделим папку res/layout в нашем модуле и нажмем на ней правую кнопку мыши. В появившемся меню выбираем:

```
New > Layout resource file (либо ALT+Insert) > Layout resource file.
```

В папке layout должен появиться новый файл activity_second.xml. Этот новый layout-файл должен сразу открыться на редактирование. Добавим на экран элемент PlainTextView из списка слева и через *Attributes* изменим его текст на: *«new life for activity» и 3 кнопки Button*. Сохраняем (CTRL+S).

При создании нового layout-файла activity_second, среда добавила в R.java новую константу для этого файла - R.layout. activity_second. И мы теперь в коде сможем через эту константу указать на этот новый layout-файл.

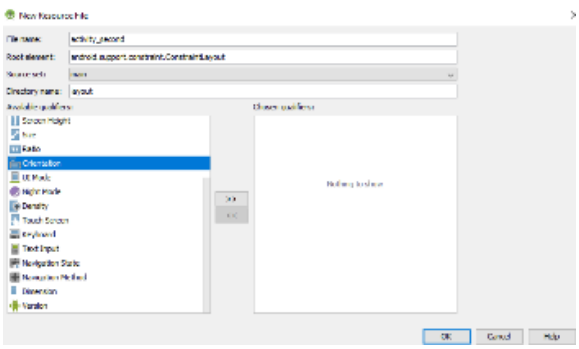
Настроим так, чтобы Activity использовало новый файл activity_second.xml, а не activity_main.xml, который был изначально. Откроем MainActivity.java
и поменяем аргумент метода *setContentView*. Замените «R.layout.activity_main», на «R.layout.activity_second» (ID нового layout-файла). Должно получиться так:

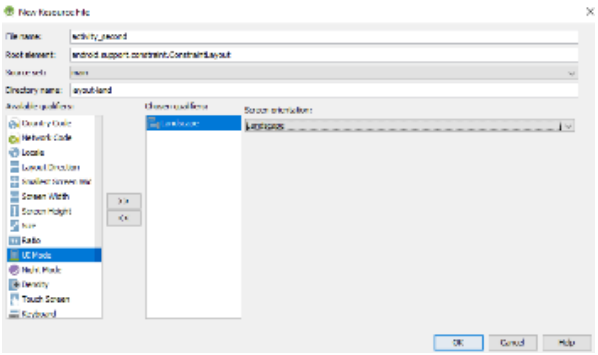
```
6   public class MainActivity extends AppCompatActivity {
7
8       @Override
9       protected void onCreate(Bundle savedInstanceState) {
10          super.onCreate(savedInstanceState);
11          setContentView(R.layout.activity_second);
12      }
13  }
```

Сохраняем код (CTRL+S) и запускаем приложение (SHIFT+F10).

Нажмем в эмуляторе CTRL+F12, ориентация сменилась на горизонтальную. Т.е. нам необходим еще один layout-файл, который был бы заточен под горизонтальную. Но как дать знать Activity, что она в вертикальной ориентации должна использовать один layout-файл, а в горизонтальной – другой? Имеется возможность создать layout-файл, который будет использоваться приложением, когда устройство находится в горизонтальной ориентации.

Создание такого файла почти не отличается от создания обычного layout-файла. Выделяем папку res/layout и создаем новый Layout resource file. Название файла указываем то же самое: activity_second. Осталось добавить спецификатор, который даст приложению понять, что этот layout-файл требуется использовать в горизонтальной ориентации. Для этого в списке спецификаторов (Available qualifiers) слева снизу находим Orientation:

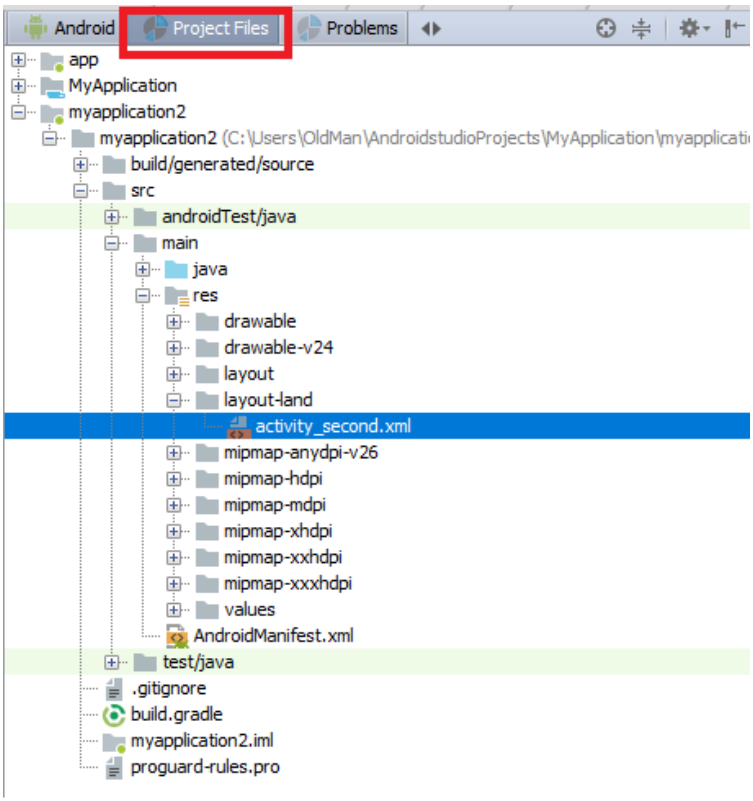




Нажимаем кнопку со стрелкой вправо. Тем самым мы включили использование спецификатора ориентации. Нам надо указать, что нас интересует горизонтальная ориентация: Landscape. Выберите это значение из выпадающего списка.

Обратите внимание, что изменилось значение поля Directory name

Настройкой спецификатора мы указали, что наш новый layout-файл будет создан в папке res/layout-land, а не res/layout, как обычно (переключитесь во вкладке Project Files). Т.е. спецификатор –land указывает на то, что layout-файлы из этой папки будут использованы в горизонтальной ориентации устройства.



Посмотрим на структуру модуля (переключимся обратно в вкладку Android). Видим, что у нас теперь два файла activity_second: обычный и land. Открыть файл activity_second.xml (land) и поменяем его содержимое:

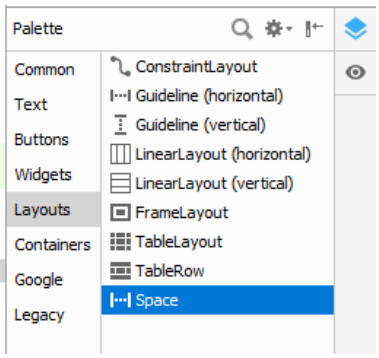
Сохраняем код (CTRL+S) и запускаем приложение (SHIFT+F10).

Activity читает layout-файл, который мы указывали в методе setContentView, т.е. activity_second*.xml и отображает его содержимое. При этом оно учитывает ориентацию устройства, и в случае горизонтальной ориентации берет* activity_second *из папки res/layout-land (если он, конечно, там существует).*

Переключим ориентацию CTRL+F12.

5. *Виды Layouts. Ключевые отличия и свойства.*

Для возможности размещения на экране различных компонентов (кнопки, поля ввода, чекбоксы и т.п.), необходимо использовать специальный контейнер. Именно в него вы будете помещать компоненты. В Android компоненты называются View, а контейнер - ViewGroup. Существуют несколько типов ViewGroup: LinearLayout, RelativeLayout, FrameLayout, TableLayout, ConstraintLayout и т.д. Они различаются тем, как они будут упорядочивать компоненты внутри себя.



LinearLayout – отображает View-элементы в виде одной строки (если он Horizontal) или одного столбца (если он Vertical).

TableLayout – отображает элементы в виде таблицы, по строкам и столбцам.

RelativeLayout – для каждого элемента настраивается его положение относительно других элементов.

FrameLayout – самый простой тип разметки. Все дочерние элементы прикрепляются к верхнему левому углу экрана. В разметке нельзя определить различное местоположение для дочернего объекта. Последующие дочерние объекты View будут просто рисоваться поверх предыдущих компонентов, частично или полностью затеняя их.

ConstraintLayout – размещает элементы view путем привязки к дочерним элементам или самому себе.

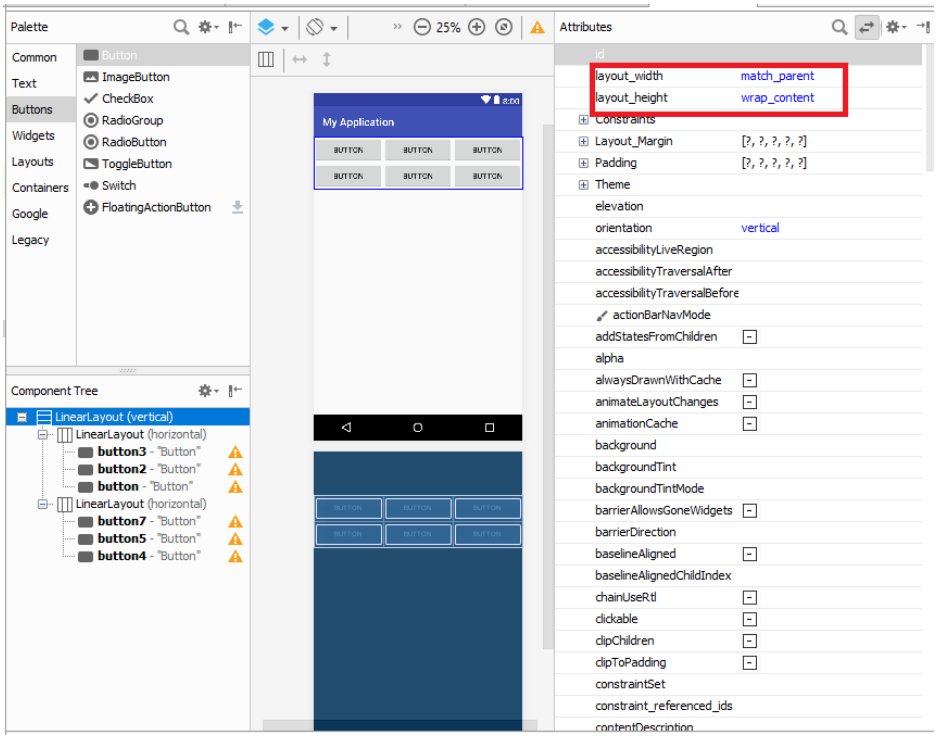
Задание:

*Выделяем папку res/layout и создаем новый Layout resource file. В Root element **указываем**:*

- ***LinearLayout** - имеет свойство Orientation, которое определяет, как будут расположены дочерние элементы – горизонтальной или вертикальной линией. Разместите button и изменяйте ориентацию LL.*

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="vertical"
              android:layout_width="match_parent"
              android:layout_height="match_parent">

</LinearLayout>
```

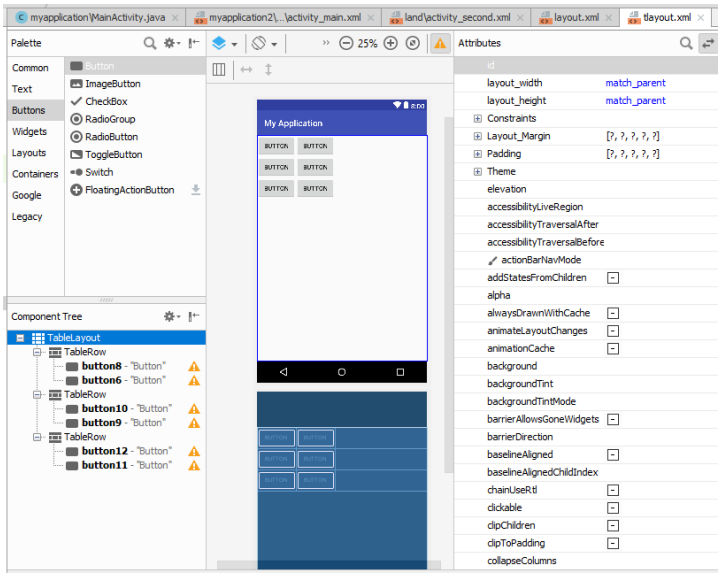
GroupView можно вкладывать друг в друга. Вложим в один LL два других. Удалите в main.xml все элементы кроме корневого LL. Ориентацию корневого LL укажем ***вертикальную*** *и добавим в него два новых* ***горизонтальных*** *LL. В списке элементов слева они находятся в разделе Layouts. Напоминаю, что вы можете перетаскивать элементы из списка не только на экран, но и на конкретный элемент на вкладке Outline. Обратите внимание на параметры ширины и высоты экрана -*



- ***TableLayout***,

TL состоит из строк TableRow (TR). Каждая TR в свою очередь содержит View-элементы, формирующие столбцы. Т.е. количество View в TR - это количество столбцов. Но количество столбцов в таблице должно быть равным для всех строк. Поэтому, если в разных TR разное количество View-элементов (столбцов), то общее количество определяется по TR с максимальным количество. Рассмотрим на примере.

Создадим layout-файл tlayout.xml. с корневым элементом TableLayout. Добавим в корневой TableLayout три TableRow-строки (из раздела Layouts слева) и в каждую строку добавим по две кнопки. Результат: наша таблица имеет три строки и два столбца. Добавьте еще различные элементы View (TL может содержать не только TR, но и обычные View. Добавьте, например, Button прямо в TL, а не в TR и увидите, что она растянулась на ширину всей таблицы.).



- ***RelativeLayout***,

В этом виде Layout каждый View-элемент может быть расположен определенным образом относительно указанного View-элемента.

Виды отношений:

- слева, справа, сверху, снизу указанного элемента (layout_toLeftOf, layout_toRightOf, layout_above, layout_below)

- выравненным по левому, правому, верхнему, нижнему краю указанного элемента (layout_alignLeft, layout_alignRight, layout_alignTop, layout_alignBottom)

- выравненным по левому, правому, верхнему, нижнему краю родителя (layout_alignParentLeft, layout_alignParentRight, layout_alignParentTop, layout_alignParentBottom)

- выравненным по центру вертикально, по центру горизонтально, по центру вертикально и горизонтально относительно родителя (layout_centerVertical, layout_centerHorizontal, layout_centerInParent)

Добавить различные view.

- ***ConstraintLayout***.

Добавьте на экран какой-нибудь компонент, например, снова TextView. Для этого просто перетащите компонент мышкой из Palette на экран. После этого TextView появился на экране и в Component Tree.

Запустим приложение (SHIFT+F10). Видим, что TextView уехал влево и вверх. Что-то явно пошло не так.

Если вы откроете текстовое представление вашего экрана (вкладка Text слева-снизу), то увидите, что элемент TextView подчеркнут красной линией. Если навести на него мышкой, то он покажет ошибку:

This view is not constrained, it only has designtime positions, so it will jump to (0,0) unless you add constraints.

Этим сообщением среда разработки сообщает, что View не привязано. Его текущее положение на экране актуально только для разработки (т.е. только в студии). А при работе приложения, это положение будет проигнорировано, и View уедет в точку (0,0), т.е. влево-вверх (что мы и наблюдали при запуске).

Как сделать так, чтобы View в ConstraintLayout оставалось на месте и не смещалось в угол? Необходимо добавить привязки (constraints). Они будут задавать положение View на экране относительно каких-либо других элементов или относительно родительского View. Давайте добавим привязки для нашего TextView.

Если вы выделите на экране TextView, то можете видеть 4 круга по его бокам.Эти круги используются, чтобы создавать привязки.

Существует два типа привязок: одни задают положение View по горизонтали, а другие - по вертикали.

Создадим горизонтальную привязку. Привяжем положение TextView к левому краю его родителя. Напомню, что родителем TextView является ConstraintLayout, который в нашем случае занимает весь экран. Поэтому края ConstraintLayout совпадают с краями экрана.

Чтобы создать привязку, нажмите мышкой на TextView, чтобы выделить его. Затем зажмите левой кнопкой мыши левый кружок и тащите его к левой границе. TextView также уехал влево. Он привязался к левой границе своего родителя.

Но вовсе необязательно они должны быть вплотную. Мы можем задать отступ. Для этого просто зажмите левой кнопкой мыши TextView, перетащите вправо и отпустите. Обратите внимание на число, которое меняется. Это величина отступа TextView от объекта, к которому он привязан (в нашем случае - от левой границы родителя).

Запустим приложение.

Раньше у нас TextView уезжал влево-вверх, а теперь он уехал только вверх. Влево он не уехал, т.к. мы создали для него горизонтальную привязку. И TextView теперь знает, что по горизонтали он должен располагаться с определенным отступом от левого края.

Давайте создадим вертикальную привязку, чтобы закрепить TextView и по вертикали. Используем верхний кружок и тащим его к верхней границе. TextView привязывается по вертикали к верхней границе родителя. После

этого можно перетащить TextView куда вам нужно, чтобы настроить горизонтальный и вертикальный отступы. При перетаскивании вы видите значения отступов.

Теперь TextView привязан и по горизонтали, и по вертикали. Т.е. он точно знает, где он должен находиться на экране во время работы приложения.

Запускаем, чтобы проверить.

TextView никуда не уехал, а находится там, где мы и настроили с помощью привязок. Давайте добавим еще одно View, например, кнопку - Button. Мы можем привязывать не только к границам родителя, но и к другим View. Давайте привяжем кнопку к TextView.

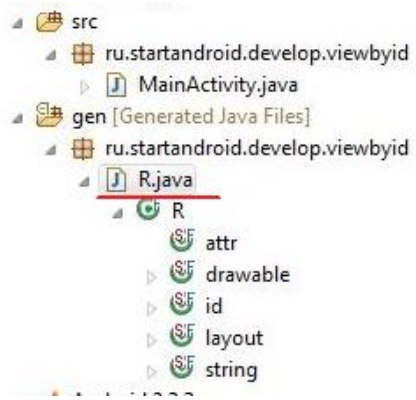
Мы рассмотрели примеры, когда View было привязано по каждой оси с одной стороны. Т.е. только слева или справа по горизонтали, и сверху или снизу по вертикали. Но мы можем привязать View с обоих сторон по каждой оси. Пока рассмотрим только горизонтальную привязку.
Но, разумеется, все это будет работать и для вертикальной привязки. Давайте попробуем, например, левый край привязать к левой границе родителя, а правый край - к правой границе родителя. TextView сначала ушел влево, т.к. была привязка к левой границе, но после создания привязки к правой границе он выровнялся и теперь расположен по центру. Т.е. привязки уравняли друг друга, и View находится ровно посередине между тем, к чему он привязан слева, и тем, к чему он привязан справа. Т.е. в нашем случае View находится посередине между левой и правой границами его родителя. Обратите внимание, что такие двусторонние привязки отображаются как пружинки, а не линии.

6. *Обращение из кода к элементам экрана. Обработчики событий.*

Чтобы обратиться к элементу экрана из кода, нам нужен его ID. Он прописывается либо в Properties, либо в layout-файлах, как вам удобнее. Для ID существует четкий формат - @+id/name, где + означает, что это новый ресурс и он должен добавиться в R.java класс, если он там еще не существует.

Давайте откроем activity_main.xml, для TextView укажем ID = @+id/ textView и сохраним и скомпилируем(Ctrl+S)

Теперь откроем R.java и видим, что для класса id появилась константа myText. Т.е. чтобы к ней обратиться, надо написать R.id.textView.



Она связана с элементом TextView и мы можем ее использовать, чтобы обратиться к элементу программно. Для этого нам понадобится метод findViewById. Он по ID возвращает View. Давайте напишем вызов этого метода. Напомню, что пока мы пишем наш код в методе onCreate. Это метод, который вызывается при создании Activity. Если вдруг непонятно, куда писать, можно подсмотреть в конец урока, там я выложил код.

Откроем MainActivity.java и после строки с вызовом метода setContentView напишем:

```
TextView myTextView = (TextView) findViewById(R.id.textView);
```

P.S. Если View подчеркнуто красным, то скорей всего этот класс не добавлен в секцию import. Нажмите CTRL+SHIFT+O для автоматического обновления импорта.

Теперь **myTextView** имеет тип **TextView**, а результат метода findViewById мы преобразуем из View в TextView. Теперь мы можем применять к myTextView методы класса TextView. Для примера возьмем метод setText. Сейчас отображаемый текст = *Hello World, MainActivity!*. Мы его программно поменяем на *New text in TextView:*

```
myTextView.setText("New text in TextView");
```

Сохраняем, запускаем (CTRL+F11) и видим, что текст изменился.

Добавим на экран кнопку (Button), Id = @+id/myBtn, текст оставим по умолчанию. Сохраняем - CTRL+SHIFT+S (если не сохранить, то в R.java не появится ID).

Пишем код:

```
Button myBtn = (Button) findViewById(R.id.myBtn);
```

Они друг другу не мешают и так делать даже логичнее. Это остается на ваше усмотрение. Так, кнопку мы нашли, теперь давайте изменим ее текст:

```
myBtn.setText("My button");
```

Запустим приложение. Текст на кнопке поменялся, на кнопку можно понажимать, но ничего происходить не будет. Т.к. мы нигде не указывали, что надо делать при нажатии. А пока давайте сделаем кнопку неактивной.

```
myBtn.setEnabled(false);
```

Мы поменяли параметр Enabled. Теперь на кнопку нельзя нажать. Сохраним, запустим и убедимся.

Добавим CheckBox, id = @+id/myChb. По умолчанию галочка не стоит. Давайте поставим ее программно, для этого используется метод setChecked, который меняет параметр Checked.

```
CheckBox myChb = (CheckBox) findViewById(R.id.myChb);
myChb.setChecked(true);
```

Запустив приложение видим, что код сработал.

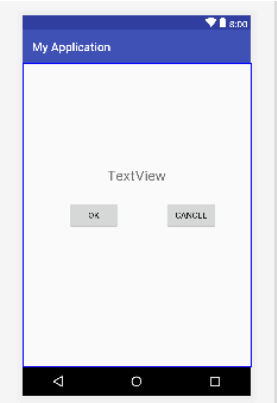
Как видите – все несложно. Используем метод findViewById, чтобы по ID получить объект соответствующий какому-либо View-элементу (Button, TextView, CheckBox) и далее вызываем необходимые методы объектов (setText, setEnabled, setChecked).

В итоге должен получиться такой код:

```
package ru.startandroid.develop.viewbyid;
import android.app.Activity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.TextView;

public class MainActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        TextView myTextView = (TextView) findViewById(R.id.myText);
        myTextView.setText("New text in TextView");

        Button myBtn = (Button) findViewById(R.id.myBtn);
        myBtn.setText("My button");
        myBtn.setEnabled(false);

        CheckBox myChb = (CheckBox) findViewById(R.id.myChb);
        myChb.setChecked(true);
    }
}
```

7. Обработчики событий на примере Button.

Создадим новый модуль. В меню File >New > New Module > Phone & Tablet Module > Empty Activity. Проект назовем onclickbuttons.



Добавим TextView (id = tvOut) с текстом и две кнопки: OK (id = btnOk) и Cancel (id = btnCancel). Мы сделаем так, чтобы по нажатию кнопки менялось содержимое TextView. По нажатию кнопки OK – будем выводить текст: «Нажата кнопка OK», по нажатию Cancel – «Нажата кнопка Cancel».

Открываем MainActivity.java. Описание объектов вынесем за пределы метода onCreate. Это сделано для того, чтобы мы могли из любого метода обращаться к ним. В onCreate мы эти объекты заполним с помощью уже пройденного нами метода *findViewById*. В итоге должен получиться такой код:

```
1  public class MainActivity extends Activity {
2
3      TextView tvOut;
4      Button btnOk;
5      Button btnCancel;
6
7      /** Called when the activity is first created. */
8      @Override
9      public void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         setContentView(R.layout.main);
12
13         // найдем View-элементы
14         tvOut = (TextView) findViewById(R.id.tvOut);
15         btnOk = (Button) findViewById(R.id.btnOk);
16         btnCancel = (Button) findViewById(R.id.btnCancel);
17
18     }
19 }
```

Обновляем секцию import (CTRL+SHIFT+O). Объекты tvOut, btnOk и btnCancel соответствуют View-элементам экрана и мы можем с ними работать. Нам надо научить кнопку реагировать на нажатие. Для этого у кнопки есть метод setOnClickListener (View.OnClickListener l). На вход подается объект с интерфейсом View.OnClickListener. Именно этому объекту кнопка поручит обрабатывать нажатия. Давайте создадим такой объект. Код продолжаем писать в onCreate:

```
1  OnClickListener oclBtnOk = new OnClickListener() {
2      @Override
3      public void onClick(View v) {
4          // TODO Auto-generated method stub
5
6      }
7  };
```

Жмем CTRL+SHIFT+O, нам нужен View.OnClickListener, т.к. метод кнопки setOnClickListener принимает на вход именно его.

Итак, мы создали объект oclBtnOk, который реализует интерфейс View.OnClickListener. Объект содержит метод onClick – это как раз то, что нам нужно. Именно этот метод будет вызван при нажатии кнопки. Мы решили, что по нажатию будем выводить текст: «Нажата кнопка ОК» в TextView (tvOut). Реализуем это. В методе onClick пишем:

```
tvOut.setText("Нажата кнопка ОК");
```

Обработчик нажатия готов. Осталось «скормить» его кнопке с помощью метода setOnClickListener.

```
btnOk.setOnClickListener(oclBtnOk);
```

В итоге должен получится такой код:

```
public class MainActivity extends Activity {

    TextView tvOut;
    Button btnOk;
    Button btnCancel;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // найдем View-элементы
        tvOut = (TextView) findViewById(R.id.tvOut);
        btnOk = (Button) findViewById(R.id.btnOk);
        btnCancel = (Button) findViewById(R.id.btnCancel);

        // создаем обработчик нажатия
        OnClickListener oclBtnOk = new OnClickListener() {
            @Override
            public void onClick(View v) {
                // Меняем текст в TextView (tvOut)
                tvOut.setText("Нажата кнопка ОК");
            }
        };

        // присвоим обработчик кнопке ОК (btnOk)
        btnOk.setOnClickListener(oclBtnOk);
    }
}
```

Нажатие на Cancel пока ни к чему не приводит, т.к. для нее мы обработчик не создали и не присвоили. Давайте сделаем это аналогично, как для кнопки ОК. Сначала мы создаем обработчик:

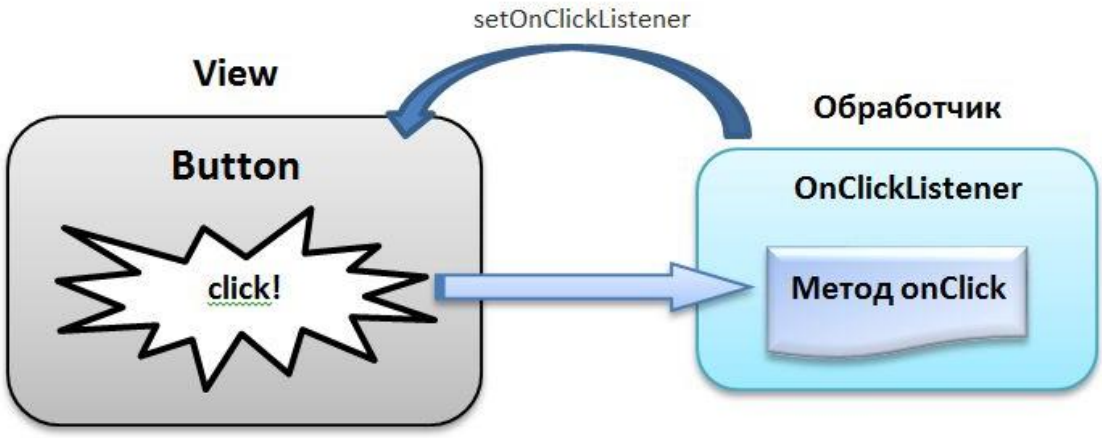
```
OnClickListener oclBtnCancel = new OnClickListener() {
    @Override
    public void onClick(View v) {
        // Меняем текст в TextView (tvOut)
        tvOut.setText("Нажата кнопка Cancel");
    }
};
```

Потом присваиваем его кнопке:

```
btnCancel.setOnClickListener(oclBtnCancel);
```

Запускаем, проверяем. Обе кнопки теперь умеют обрабатывать нажатия.

Давайте еще раз проговорим механизм обработки событий на примере нажатия кнопки. Сама кнопка обрабатывать нажатия не умеет, ей нужен обработчик (его также называют слушателем - listener), который присваивается с помощью метода setOnClickListener. Когда на кнопку нажимают, обработчик реагирует и выполняет код из метода onClick. Это можно изобразить так:



Соответственно для реализации необходимо выполнить следующие шаги:

- создаем обработчик;
- заполняем метод onClick;
- присваиваем обработчик кнопке.