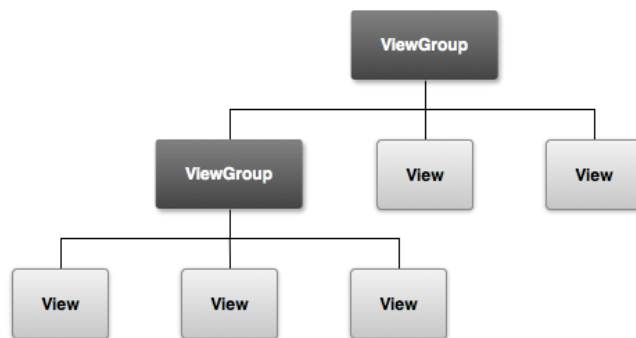


3. СОЗДАНИЕ ИНТЕРФЕЙСА. МАКЕТЫ. ЭЛЕМЕНТЫ UI. РЕСУРСЫ

1 Организация пользовательского интерфейса

Все элементы интерфейса в приложении Android создаются с помощью объектов *View* и *ViewGroup*. Объект *View* формирует на экране элемент, с которым пользователь может взаимодействовать. Объект *ViewGroup* содержит другие объекты *View* (и *ViewGroup*) для определения макета интерфейса.

Android предоставляет коллекцию подклассов *View* и *ViewGroup*, которая включает в себя обычные элементы ввода и различные модели макета. Каждая группа представляет собой невидимый контейнер, в котором объединены дочерние виды. Эта древовидная иерархия может быть простой или сложной (чем проще, тем лучше для производительности).



Для отладки макетов можно воспользоваться инструментом *Hierarchy Viewer* — с его помощью можно просмотреть значения свойств, рамки с индикаторами заполнения или поля, а также полностью отрисованные представления прямо во время отладки приложения на эмуляторе или на устройстве.

1.1 Разработка layout

Макет определяет визуальную структуру пользовательского интерфейса. Существует два способа объявить макет (рис 1):

- объявление элементов пользовательского интерфейса в XML;
- создание экземпляров элементов во время выполнения (приложение может программным образом создавать объекты *View* и *ViewGroup*).

В приложениях Android визуальный интерфейс загружается из специальных файлов XML, которые хранят разметку. Эти файлы являются ресурсами разметки.

Объявление пользовательского интерфейса в файлах XML позволяет

отделить интерфейс приложения от кода. В приложении могут быть определены разметки в файлах XML для различных ориентаций монитора, размеров устройств, языков и т.д. Кроме того, объявление разметки в XML позволяет легче визуализировать структуру интерфейса и облегчает отладку (рис. 2).

Активность - специальный класс Java, который решает, какой макет следует использовать, и описывает, как приложение должно реагировать на действия пользователя

Макеты могут включать компоненты графических интерфейсов: кнопки, текстовые поля, подписи и т. д

```
package by.bstu.pnv.activi-
tydemo;

import android.sup-
port.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.Toast;

public class MainActivity ex-
tends AppCompatActivity {

    public void function-
    Press(View view) {

        EditText username =
        (EditText)findViewById(
        R.id.userName);
        Log.i("Username",
        username.getText().toString());
        Toast.makeText(getAppli-
        cationCon-
        text(),username.getText().toStri-
        ng(),Toast.LENGTH_LONG).show();

        ImageView imageView =
        (ImageView)findViewById(R.id.im-
        ageIcon);
        imageView.setImageRe-
        source(R.drawable.penguins);
    }

    @Override
    protected void onCreate(Bun-
    dle savedInstanceState) {
        super.onCreate(savedIn-
        stanceState);
        setContentView(R.lay-
```

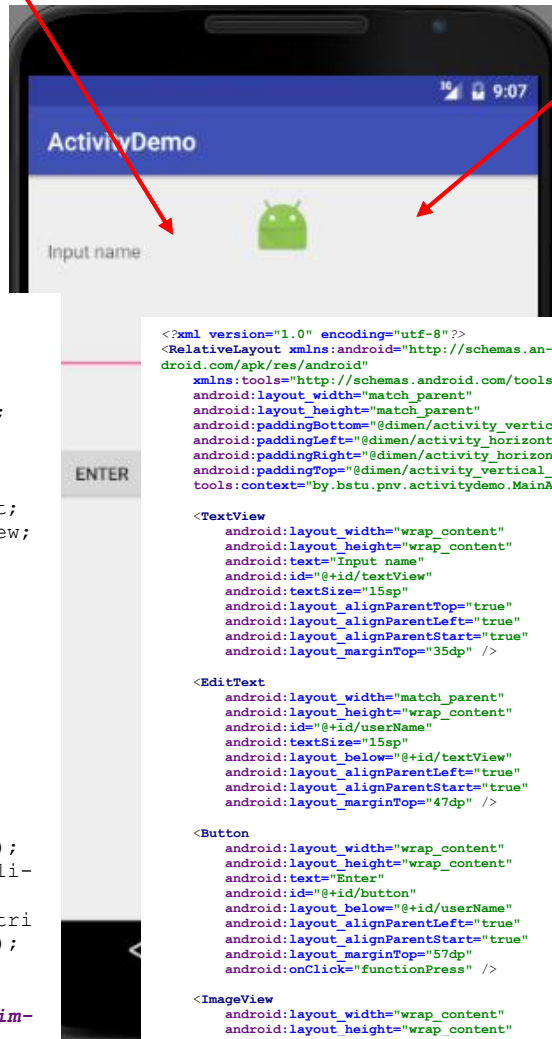


Рис. 1 Разработка макета пользовательского интерфейса Android-приложения

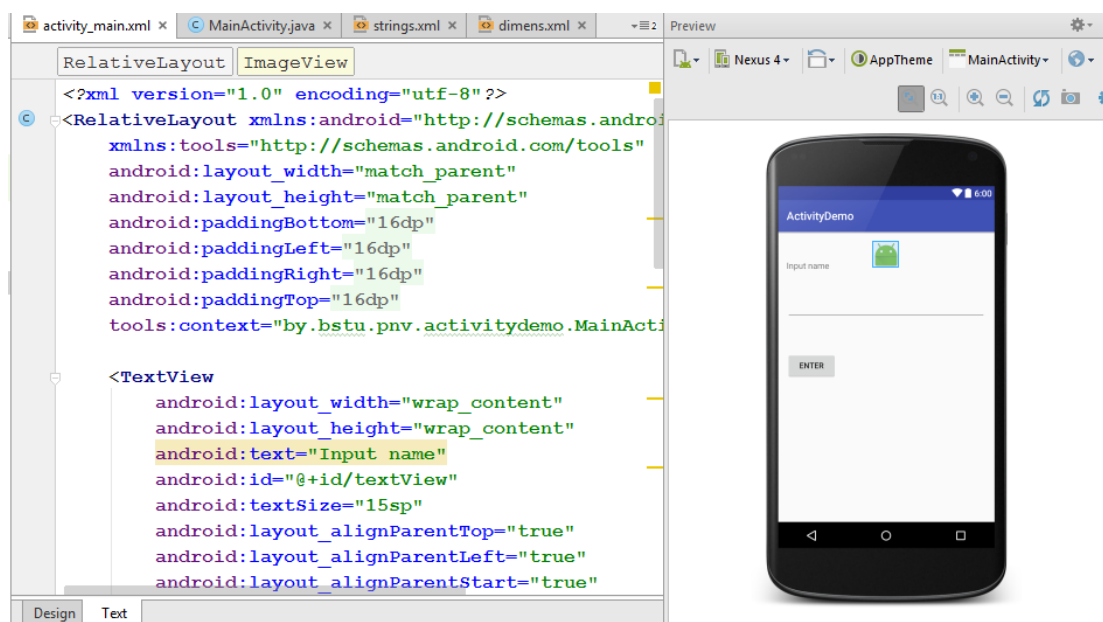


Рис. 2 Разработка макета пользовательского интерфейса основе XML

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/in"
    android:id="@+id/textView"
    android:textSize="15sp"
    android:layout_alignParentTop="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_marginTop="35dp" />
```

Файлы разметки графического интерфейса располагаются в проекте в каталоге *res/layout*.

1.2 Разработка интерфейса в режиме дизайнера

Android Studio имеет инструментарий, который облегчает разработку графического интерфейса. Можно открыть файл XML и с помощью кнопки *Design* переключиться в режим дизайнера к графическому представлению интерфейса в виде эскиза устройства (рис.3).

Слева будет находиться панель инструментов, из которой можно перенести нужный элемент мышкой на эскиз. Все перенесенные элементы будут автоматически добавляться в файл XML.

При выделении элемента справа появится окно *Properties* – панель свойств выделенного элемента. Здесь можно изменить значения свойств элемента.

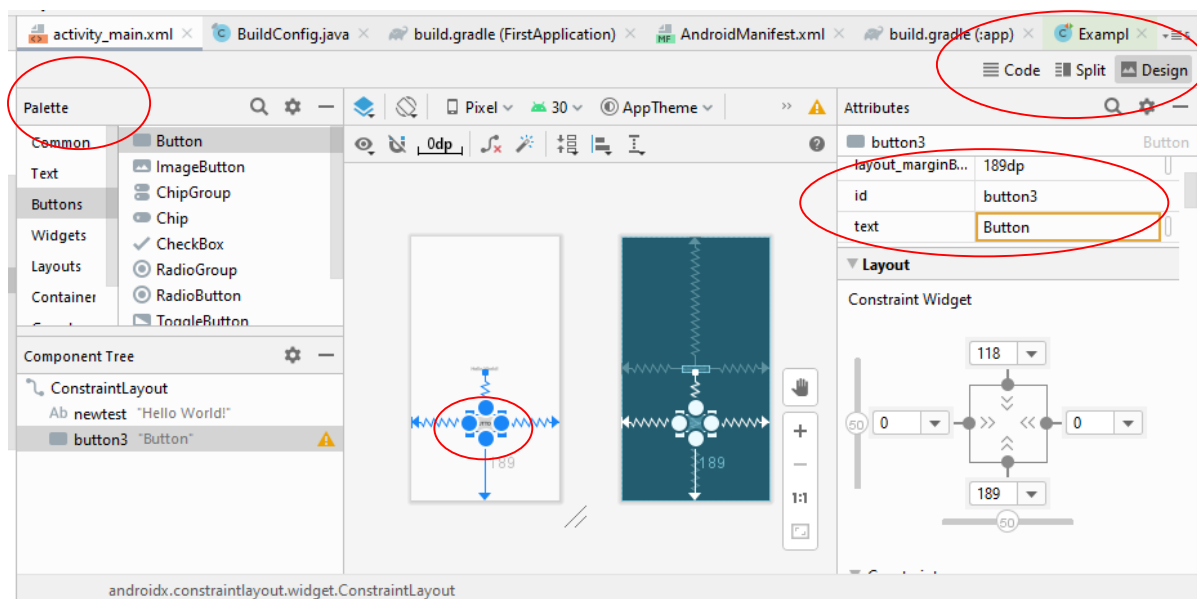


Рис. 3 Работа в режиме Design

2 Установка размеров

В операционной системе Android можно использовать различные типы измерений:

- *px*: пиксели текущего экрана. Эта единица измерения не рекомендуется, так как реальное представление внешнего вида может изменяться в зависимости от устройства;

- *dp*: (device-independent pixels) независимые от плотности экрана пиксели. Абстрактная единица измерения, основанная на физической плотности экрана с разрешением 160 dpi (точек на дюйм). В этом случае $1dp = 1px$. Если размер экрана больше или меньше, чем 160dpi, количество пикселей, которые применяются для отрисовки 1dp соответственно увеличивается или уменьшается. Общая формула для получения количества физических пикселей из **dp**: $px = dp * (dpi / 160)$;

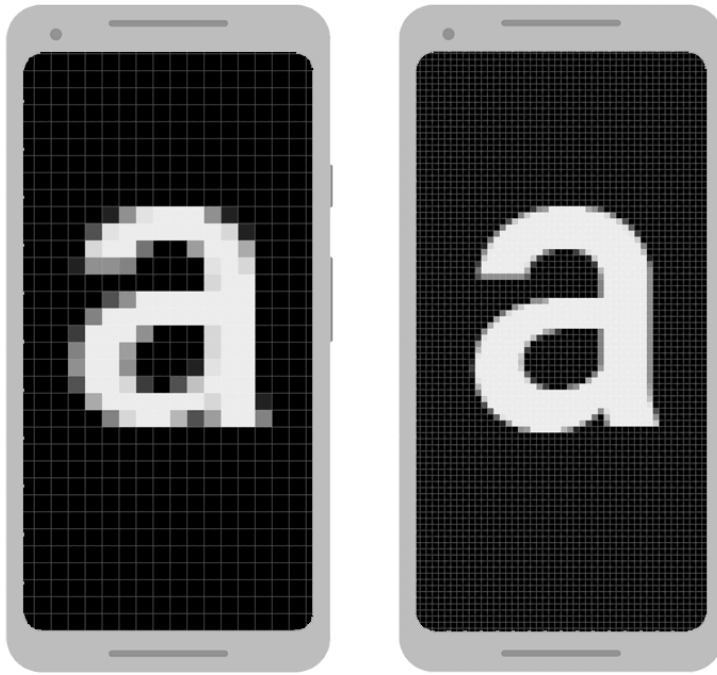
- *sp*: (scale-independent pixels) независимые от масштабирования пиксели. Допускают настройку размеров, производимую пользователем. Рекомендуются для работы со шрифтами;

- *pt*: 1/72 дюйма, базируются на физических размерах экрана;

- *mm*: миллиметры;

- *in*: дюймы.

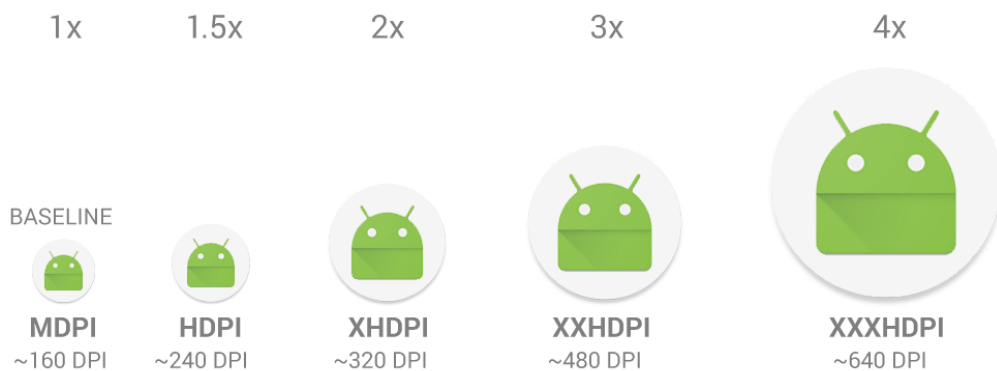
Предпочтительными единицами для использования являются *dp*.



Два экрана одного размера могут иметь разное число пикселей.

Для упрощения работы с размерами все размеры разбиты на несколько групп:

- ldpi (low): ~120dpi;
- mdpi (medium): ~160dpi;
- hdpi (high): ~240dpi;
- xhdpi (extra-high): ~320dpi;
- xxhdpi (extra-extra-high): ~480dpi;
- xxxhdpi (extra-extra-extra-high): ~640dpi.



Все визуальные элементы, которые используются в приложении, как правило, упорядочиваются на экране с помощью контейнеров. В Android подобными контейнерами служат классы *RelativeLayout*, *LinearLayout*, *GridLayout*, *TableLayout*, *ConstraintLayout*, *FrameLayout* и др. Все они по-

разному располагают элементы и управляют ими, но есть некоторые общие моменты при компоновке визуальных компонентов.

Для организации элементов внутри контейнера используются параметры разметки. Для их задания в файле XML используются атрибуты, которые начинаются с префикса *layout_*. К таким параметрам относятся атрибуты *layout_height* и *layout_width*, которые используются для установки размеров и могут принимать одно из следующих значений:

- *точные размеры* элемента (например, 96 dp) (рис.4);

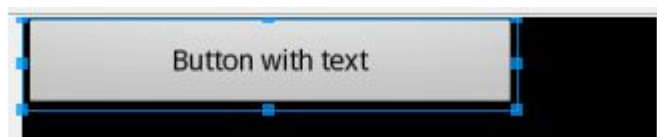


Рис. 4 Определение элемента с точными размерами

- *значение wrap_content*: элемент растягивается до тех границ, которые достаточны, чтобы вместить все его содержимое (рис. 5);



Рис. 5 Определение элемента в соответствии с содержимым

- *значение match_parent*: элемент заполняет всю область родительского контейнера (рис 6).

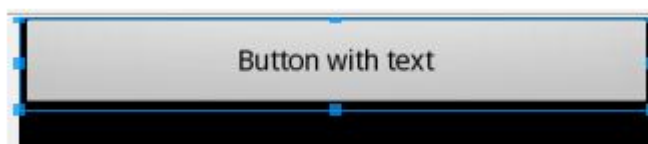


Рис. 6 Определение элемента в соответствии с родительским контейнером

Можно дополнительно ограничить минимальные и максимальные значения с помощью атрибутов *minWidth/maxWidth* и *minHeight/maxHeight*:

```
android:minWidth="200dp"
android:maxLength="250dp"
android:minHeight="100dp"
android:maxLength="200dp"
android:layout_height="wrap_content"
android:layout_width="wrap_content"
```

Если элемент, создается в коде java, то для установки высоты и ширины можно использовать метод *setLayoutParams()*:

```

TextView textView1 = new TextView(this);
textView1.setText("Hello Android");
textView1.setTextSize(26);

// устанавливаем размеры
textView1.setLayoutParams(new ViewGroup.LayoutParams
(ViewGroup.LayoutParams.WRAP_CONTENT,
ViewGroup.LayoutParams.WRAP_CONTENT));

```

Параметры разметки позволяют задать отступы как от внешних границ элемента до границ контейнера, так и внутри самого элемента между его границами и содержимым.

Для установки внутренних отступов применяется атрибут *android:padding*. Он устанавливает отступы контента от всех четырех сторон контейнера. Можно устанавливать отступы только от одной стороны контейнера, применяя атрибуты: *android:paddingLeft*, *android:paddingRight*, *android:paddingTop* и *android:paddingBottom* (рис.7).

Для установки внешних отступов используется атрибут *layout_margin*. Он имеет модификации, которые позволяют задать отступ только от одной стороны: *android:layout_marginBottom*, *android:layout_marginTop*, *android:layout_marginLeft* и *android:layout_marginRight* (рис.7):

```

android:layout_marginTop="50dp"
android:layout_marginBottom="60dp"
android:layout_marginLeft="60dp"
android:layout_marginRight="60dp"

```

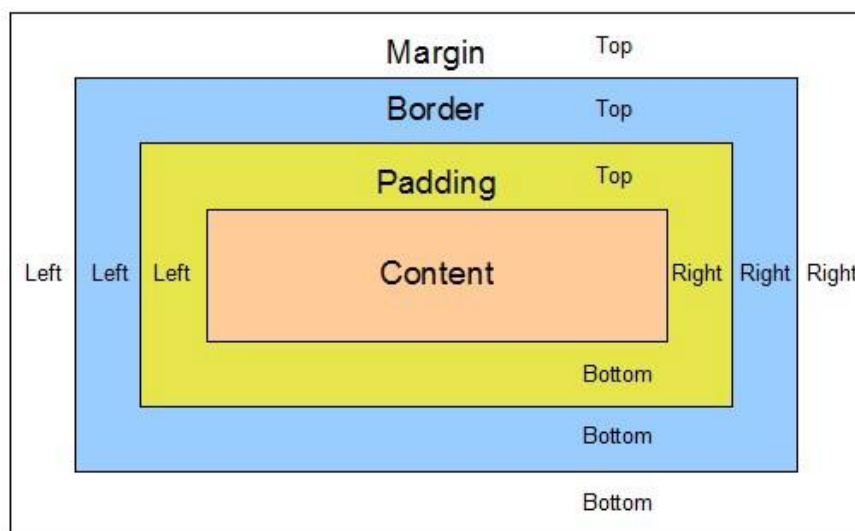


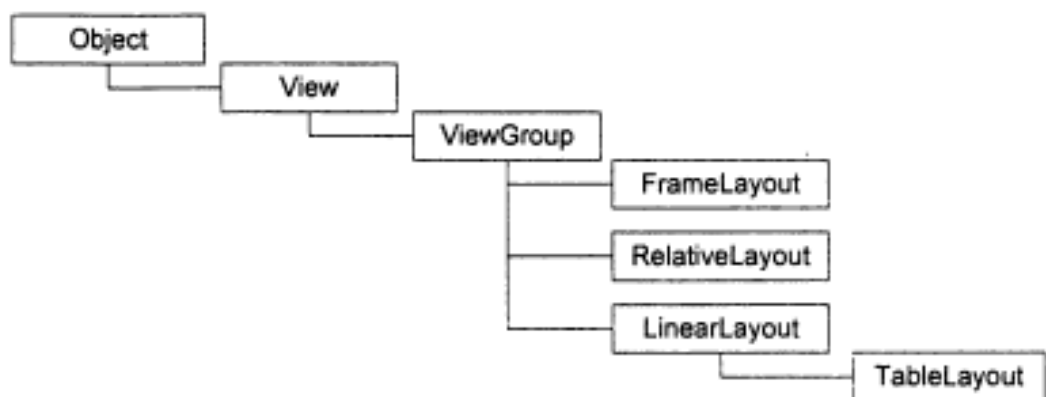
Рис. 7 Отступы элементов управления

Для программной установки внутренних отступов вызывается метод *setPadding(left, top, right, bottom)*, в который передаются четыре значения для каждой из сторон и для получения *getPaddingLeft()*, *getPaddingTop()*, *getPaddingRight()* , *getPaddingBottom()*.

3 Виды Layout

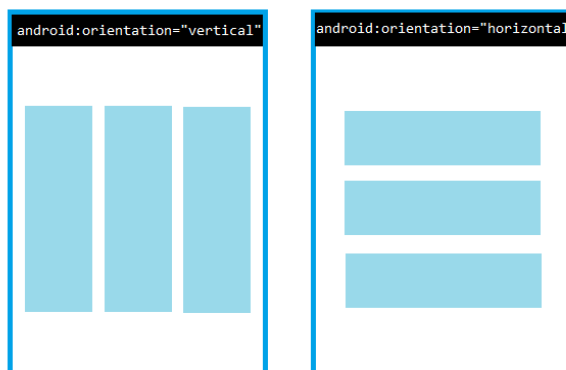
Распространены следующие макеты:

- *FrameLayout*
- *LinearLayout*
- *TableLayout*
- *RelativeLayout*
- *GridLayout*
- *ConstraintLayout* и др.



3.1 Linear Layout

Контейнер *LinearLayout* представляет объект *ViewGroup*, который упорядочивает все дочерние элементы в одном направлении: по горизонтали или по вертикали.



Все элементы расположены один за другим. Направление разметки указывается с помощью атрибута *android:orientation*. Пример отображения приведенной ниже разметки представлен на рис.8:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/an-
droid"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/Button01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/app_name"/>
    <Button
        android:id="@+id/Button02"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/enter"/>
    <Button
        android:id="@+id/Button04"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/greeting"/>
    <TextView
        android:id="@+id/TextView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/greeting"/>
</LinearLayout>
```

LinearLayout поддерживает свойство – вес элемента, которое передается атрибутом *android:layout_weight*. Это свойство принимает значение, указывающее, какую часть оставшегося свободного места контейнера по отношению к другим объектам займет данный элемент. Например, если один элемент будет иметь для свойства *android:layout_weight* значение 2, а другой – значение 1, то в сумме они дадут 3, поэтому первый элемент будет занимать 2/3 оставшегося пространства, а второй – 1/3. Если все элементы имеют значение *android:layout_weight="1"*, то все элементы будут равномерно распределены по всей площади контейнера.

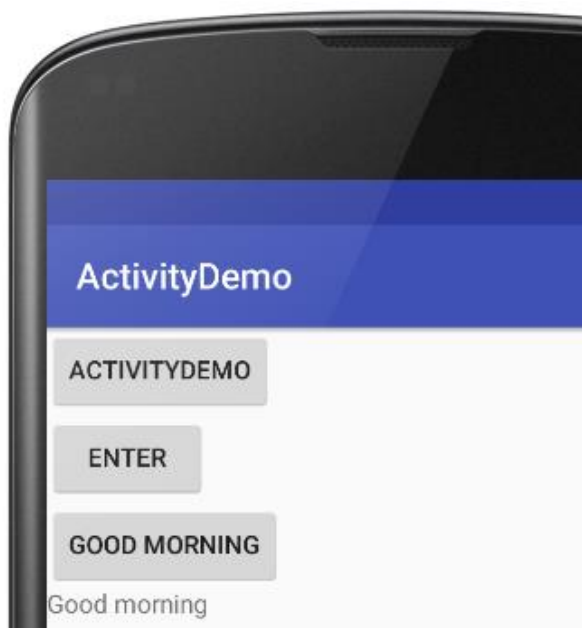
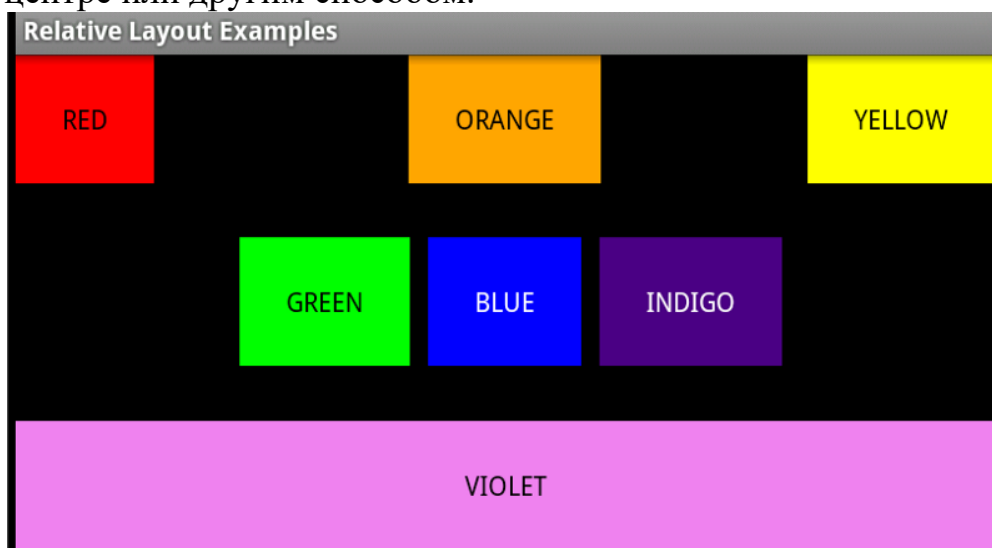


Рис. 8 Пример интерфейса с контейнером *LinearLayout*

3.2 *Relative Layout*

RelativeLayout представляет объект *ViewGroup*, который располагает дочерние элементы относительно позиции других дочерних элементов разметки или относительно области самой разметки *RelativeLayout*. Используя относительное позиционирование, можно установить элемент по правому краю, в центре или другим способом.



Для установки элемента в файле XML применяются следующие атрибуты:

- *android:layout_above*: располагает элемент над элементом с указанным Id;
- *android:layout_below*: располагает элемент под элементом с указанным Id;
- *android:layout_toLeftOf*: располагается слева от элемента с указанным Id;
- *android:layout_toRightOf*: располагается справа от элемента с указанным Id;
- *android:layout_alignBottom*: выравнивает элемент по нижней границе другого элемента с указанным Id;
- *android:layout_alignLeft*: выравнивает элемент по левой границе другого элемента с указанным Id;
- *android:layout_alignRight*: выравнивает элемент по правой границе другого элемента с указанным Id;
- *android:layout_alignTop*: выравнивает элемент по верхней границе другого элемента с указанным Id;
- *android:layout_alignBaseline*: выравнивает базовую линию элемента по базовой линии другого элемента с указанным Id;
- *android:layout_alignParentBottom*: если атрибут имеет значение *true*, то элемент прижимается к нижней границе контейнера;
- *android:layout_alignParentRight*: если атрибут имеет значение *true*, то элемент прижимается к правому краю контейнера;
- *android:layout_alignParentLeft*: если атрибут имеет значение *true*, то элемент прижимается к левому краю контейнера;
- *android:layout_alignParentTop*: если атрибут имеет значение *true*, то элемент прижимается к верхней границе контейнера;
- *android:layout_centerInParent*: если атрибут имеет значение *true*, то элемент располагается по центру родительского контейнера;
- *android:layout_centerHorizontal*: при значении *true* выравнивает элемент по центру по горизонтали;
- *android:layout_centerVertical*: при значении *true* выравнивает элемент по центру по вертикали.

Пример отображения разметки, представленной ниже приведен на рис.9:

```

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent">

    <Button
        android:id="@+id/button_center"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:height="20pt"
        android:longClickable="true"
        android:text="@string/greeting"
        android:layout_centerVertical="true"
        android:layout_centerInParent="true"/>

    <Button
        android:id="@+id/button_next"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
        android:layout_marginLeft="30sp"
        android:height="20pt"
        android:longClickable="true"
        android:text="@string/enter"
        android:width="20pt"/>

    <Button
        android:id="@+id/button_right"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/button_center"
        android:layout_alignBottom="@+id/button_center"
        android:layout_alignParentRight="true"
        android:layout_alignWithParentIfMissing="false"
        android:text="Next"
        android:layout_marginRight="30sp" />
</RelativeLayout>

```

Для управления позиционированием элемента при определении интерфейса можно использовать атрибуты *gravity* и *layout_gravity*. Атрибут *gravity* задает позиционирование содержимого внутри объекта. Он может принимать следующие значения:

- *top*: элементы размещаются вверху;
- *bottom*: элементы размещаются внизу;
- *left*: элементы размещаются в левой стороне;
- *right*: элементы размещаются в правой стороне контейнера;

- *center_vertical*: выравнивает элементы по центру по вертикали;
- *center_horizontal*: выравнивает элементы по центру по горизонтали;
- *center*: элементы размещаются по центру;
- *fill_vertical*: элемент растягивается по вертикали;
- *fill_horizontal*: элемент растягивается по горизонтали;
- *fill*: элемент заполняет все пространство контейнера;
- *clip_vertical*: обрезает верхнюю и нижнюю границу элементов;
- *clip_horizontal*: обрезает правую и левую границу элементов;
- *start*: элемент позиционируется в начале (в верхнем левом углу) контейнера;
- *end*: элемент позиционируется в конце контейнера(в верхнем правом углу).

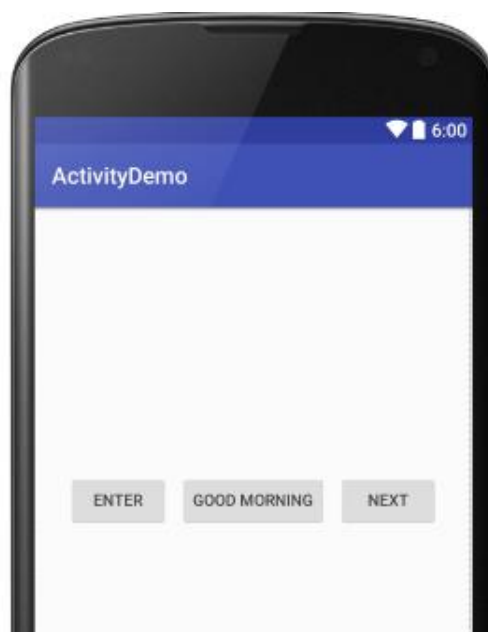


Рис. 9 Пример интерфейса с контейнером *RelativeLayout*

3.3 Table Layout

Контейнер *TableLayout* структурирует элементы по столбцам и строкам. Android находит строку с максимальным количеством виджетов одного уровня, и это количество будет означать количество столбцов. Если бы в какой-нибудь из них было бы три виджета, то соответственно столбцов было бы также три, даже если в другой строке осталось бы два виджета. Например, определены три строки и в каждой по два или три элемента (рис.10):

```

<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent">
    <TableRow>
        <Button />
        <Button />
        <Button />
    </TableRow>
    <TableRow>
        <Button />
        <Button />
    </TableRow>
    <TableRow>
        <Button />
        <Button />
        <Button />
    </TableRow>
</TableLayout>

```

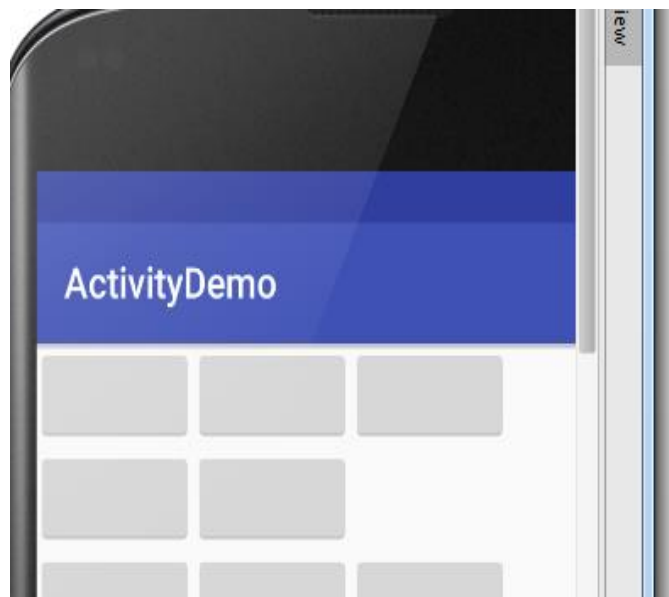


Рис. 10 Пример интерфейса с контейнером *TableLayout*

Элемент *TableRow* наследуется от класса *LinearLayout*, поэтому можно к нему применять тот же функционал, что и к *LinearLayout*.

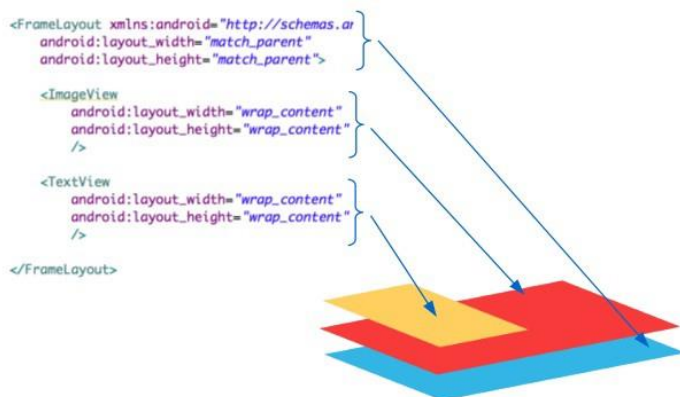
Если какой-то элемент должен быть растянут на ряд столбцов, то это выполняется с помощью атрибута *layout_column*, который указывает на какое количество столбцов надо растянуть элемент.

Также элемент можно растянуть на всю строку, установив атрибут *android:layout_weight="1"*.

3.4 *FrameLayout*

Контейнер *FrameLayout* предназначен для вывода на экран одного помещенного в него визуального элемента.

FrameLayout



Если поместить несколько элементов, то они будут накладываться друг на друга (рис. 11):

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
```

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button" />
```

```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="TextView" />
```



Рис. 11 Пример интерфейса с контейнером *FrameLayout*

Элементы управления, которые помещаются в *FrameLayout*, могут установить свое позиционирование с помощью атрибута *android:layout_gravity* (рис.12). При указании значения атрибута можно комбинировать ряд значений, разделяя их вертикальной чертой *bottom|center_horizontal*:

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent">
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="top"
        android:text="Button" />
    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Button" />
    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|center_horizontal"
        android:text="Button" />
</FrameLayout>
```


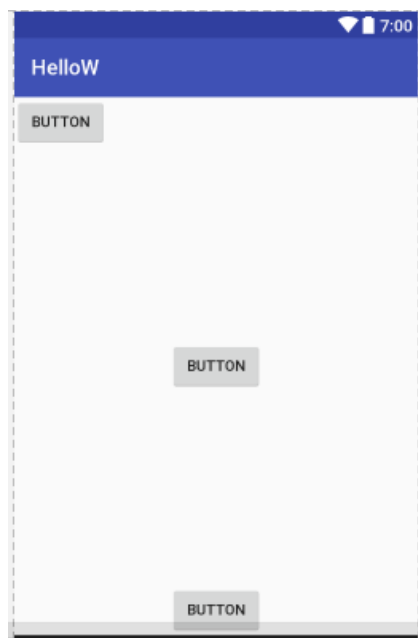



Рис. 12 Пример интерфейса с контейнером *FrameLayout* и атрибутом *android:layout_gravity*

3.5 GridLayout (legacy)

GridLayout это контейнер, который позволяет создавать табличные представления. *GridLayout* состоит из коллекции строк, каждая из которых состоит из отдельных ячеек. С помощью атрибутов *android:rowCount* и *android:columnCount* устанавливается число строк и столбцов соответственно. *GridLayout* автоматически может позиционировать вложенные элементы управления по строкам. При этом ширина столбцов устанавливается автоматически по ширине самого широкого элемента. В следующем примере устанавливается 2 строки и 3 столбца, первая кнопка попадает в первую ячейку (первая строка первый столбец), вторая кнопка – во вторую ячейку и так далее (рис. 13):

```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:rowCount="2"
    android:columnCount="3">

    <Button android:text="1" />
    <Button android:text="2" />
    <Button android:text="3" />
    <Button android:text="4" />
    <Button android:text="5" />
    <Button android:text="6" />
    <Button android:text="7" />
    <Button android:text="8" />
    <Button android:text="9" />
    <Button android:text="sin"/>
    <Button android:text="cos"/>
    <Button android:text="/" />
    <Button android:text="*" />
</GridLayout>
```



Рис. 13 Пример интерфейса с контейнером *FrameLayout*

Можно явно задать номер столбца и строки для определенного элемента, а при необходимости растянуть на несколько столбцов или строк. Для этого используют атрибуты:

- *android:layout_column*: номер столбца (отсчет идет от нуля);
- *android:layout_row*: номер строки;
- *android:layout_columnSpan*: количество столбцов, на которые растягивается элемент;
- *android:layout_rowSpan*: количество строк, на которые растягивается элемент.

```
<Button android:text="sin"
        android:layout_width="180dp"
        android:layout_columnSpan="2"/>
<Button android:text="cos"
        android:layout_height="95dp"
        android:layout_rowSpan="2"/>
```



3.6 ConstraintLayout

ConstraintLayout относительно новый тип контейнера, который является развитием *RelativeLayout* и позволяет создавать гибкие и масштабируемые интерфейсы.

Для позиционирования элемента внутри *ConstraintLayout* необходимо указать ограничения (constraints). Есть несколько типов ограничений. Для установки позиции относительно определенного элемента используются следующие ограничения:

- *layout_constraintLeft_toLeftOf*: левая граница позиционируется относительно левой границы другого элемента (аналог *layout_constraintStart_toStartOf*);
- *layout_constraintLeft_toRightOf*: левая граница позиционируется относительно правой границы другого элемента (аналог *layout_constraintStart_toEndOf*);
- *layout_constraintRight_toLeftOf*: правая граница позиционируется относительно левой границы другого элемента (аналог *layout_constraintEnd_toStartOf*);
- *layout_constraintRight_toRightOf*: правая граница позиционируется относительно правой границы другого элемента (аналог *layout_constraintEnd_toEndOf*);
- *layout_constraintTop_toTopOf*: верхняя граница позиционируется относительно верхней границы другого элемента;

- *layout_constraintBottom_toBottomOf*: нижняя граница позиционируется относительно нижней границы другого элемента;
- *layout_constraintBaseline_toBaselineOf*: базовая линия позиционируется относительно базовой линии другого элемента;
- *layout_constraintTop_toBottomOf*: верхняя граница позиционируется относительно нижней границы другого элемента;
- *layout_constraintBottom_toTopOf*: нижняя граница позиционируется относительно верхней границы другого элемента.

Позиционирование может производиться относительно границ самого контейнера *ContentLayout* (в этом случае ограничение имеет значение "parent"), либо же относительно любого другого элемента внутри *ConstraintLayout*, тогда в качестве значения ограничения указывается *id* этого элемента.

Чтобы указать отступы от элемента, относительно которого производится позиционирования, применяются следующие атрибуты:

- *android:layout_marginLeft*: отступ от левой границы;
- *android:layout_marginRight*: отступ от правой границы;
- *android:layout_marginTop*: отступ от верхней границы;
- *android:layout_marginBottom*: отступ от нижней границы;
- *android:layout_marginStart*: отступ от левой границы;
- *android:layout_marginEnd*: отступ от правой границы.

Если выделить на экране *Button*, то можно видеть 4 круга по его бокам (рис. 14). Эти круги используются, чтобы создавать привязки.

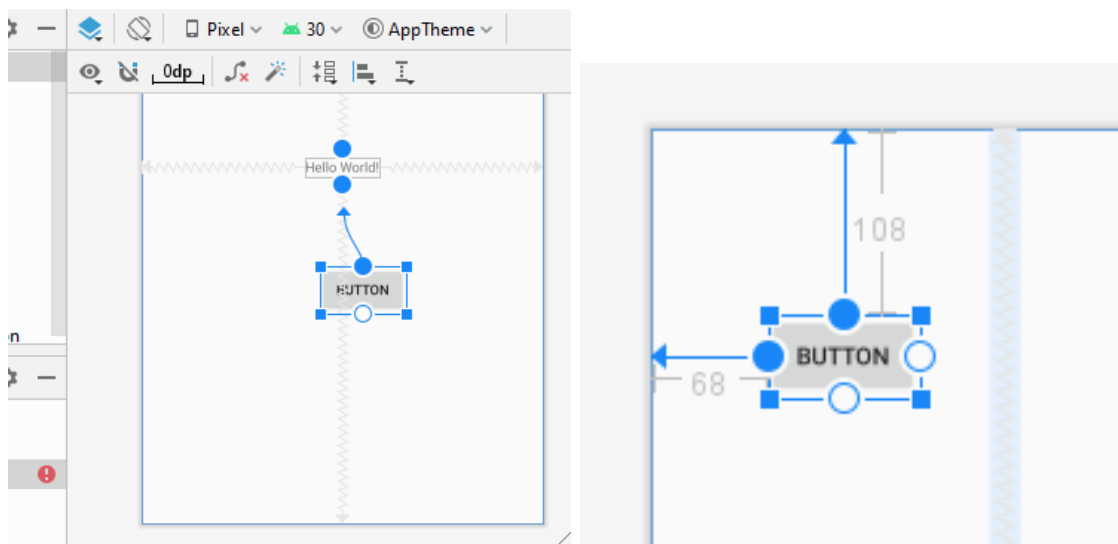


Рис.14 Установка привязок к границам контейнера

Существует два типа привязок. Одни задают положение *View* по горизонтали, а другие – по вертикали.

Создадим горизонтальную привязку. Привяжем положение *Button* к левому краю его родителя. Родителем *Button* является *ConstraintLayout*, который в нашем случае занимает весь экран. Поэтому края *ConstraintLayout* совпадают с краями экрана. Чтобы создать привязку, нажмите мышкой на *Button*, чтобы выделить его. Затем зажмите левой кнопкой мыши левый кружок и тащите его к левой границе (рис.14). *Button* также уехал влево. Он привязался к левой границе своего родителя.

Чтобы закрепить *Button* и по вертикали можно создать вертикальную привязку. Теперь *View* привязан и по горизонтали, и по вертикали (рис.14). Т.е. он точно знает, где он должен находиться на экране во время работы приложения.

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="50dp"
    android:layout_marginTop="50dp"
    android:text="Button"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Можно привязываться не только к границам родителя, но и к другим *View*. Например, привяжем кнопку к *TextView*. Так как кнопка привязана к *TextView*, то, при перемещении *TextView*, кнопка будет также перемещаться (рис.15).

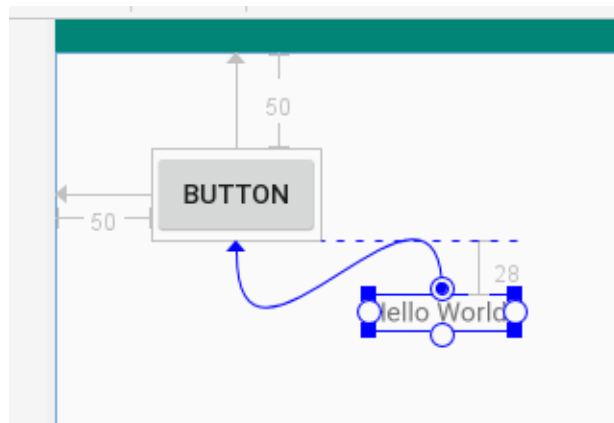


Рис. 15 Установка привязок к *View*

Чтобы удалить привязку, надо просто нажать на соответствующий кружок (рис.16).

Привяжем кнопку к левой и правой границам родителя. *Button* сначала ушел влево, т.к. была привязка к левой границе, но после создания привязки к правой границе она выровнялась и теперь расположена по центру (рис.17). Т.е. привязки уравнивали друг друга, и *View* находится ровно посередине между тем, к чему он привязан слева, и тем, к чему он привязан справа. Обратите внимание, что такие двусторонние привязки отображаются как пружинки, а не линии.

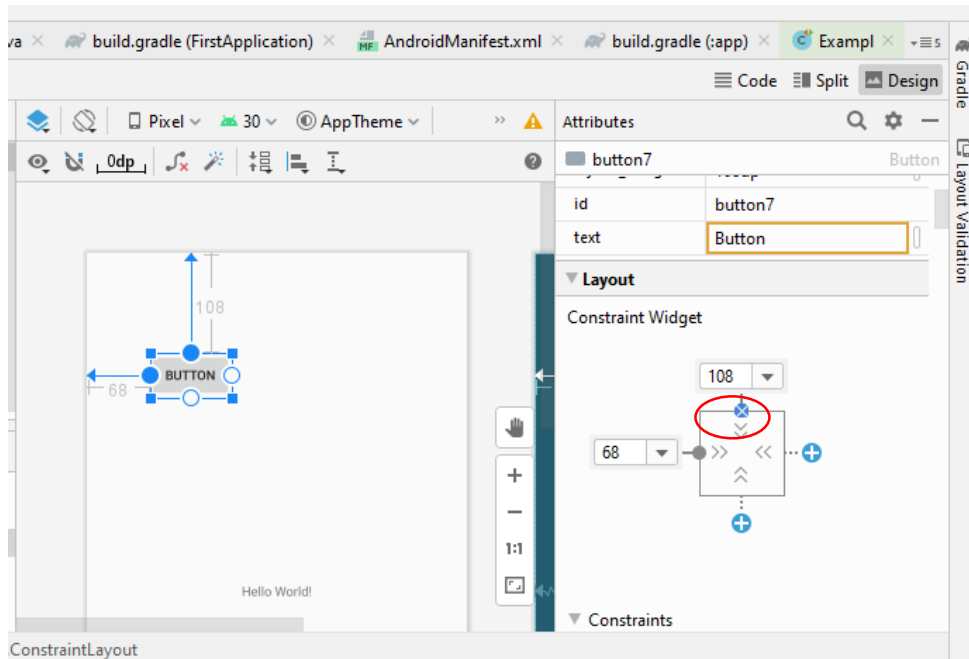


Рис. 16 Удаление привязки

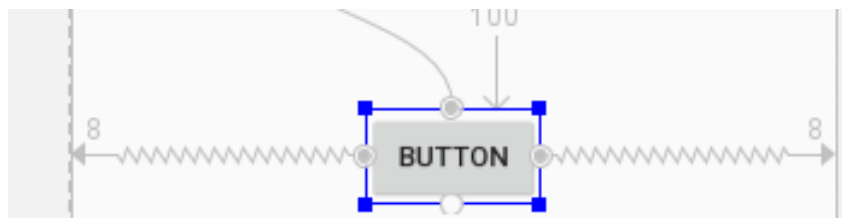


Рис. 17 Привязка к левой и правой границам

При создании ограничений придерживайтесь следующих правил:

— у каждого *View* должно быть как минимум два ограничения: одно горизонтальное и одно вертикальное;

— вы можете создавать ограничения только между дескриптором ограничения и точкой привязки, которые используют одну и ту же плоскость. Таким образом, вертикальная плоскость (левая и правая стороны) *View* может быть ограничена только другой вертикальной плоскостью; и базовые линии могут ограничивать только другие базовые линии;

— каждое правило может быть использовано только для одного ограничения, но можно создать несколько ограничений (с различных точек зрения) к одной и той же точке привязки.

В панели есть несколько инструментов, которые могут помочь в работе (рис.18) (перечислены слева направо).

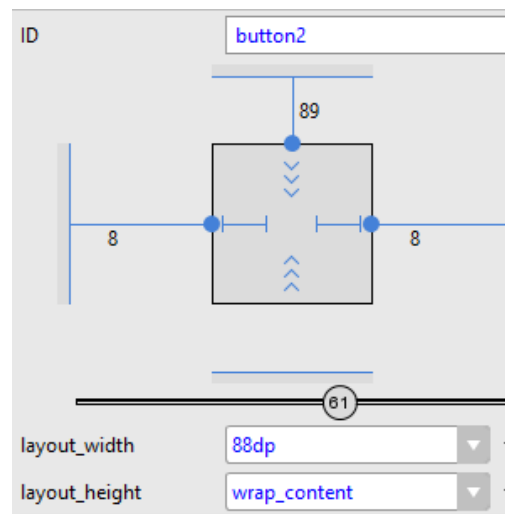


Рис. 18 Панель инструментов для работы с привязками

Показать/скрыть привязки – если включено, то все привязки будут видны на экране, если выключено, то видны будут только привязки выделенного *View*.

Автопривязки – если включено, то можно создавать привязки к родителю.

Отступ – здесь можно автоматически задать, какой отступ будет использован по умолчанию при создании привязки.

Удалить все привязки – по нажатию этой кнопки все привязки на экране будут удалены.

Создать привязки – создает привязки для всех *View* на экране.

Собрать-растянуть – собирает вместе несколько выделенных *View* сначала по горизонтали, затем по вертикали. Эта операция не создает никаких привязок (рис.19).

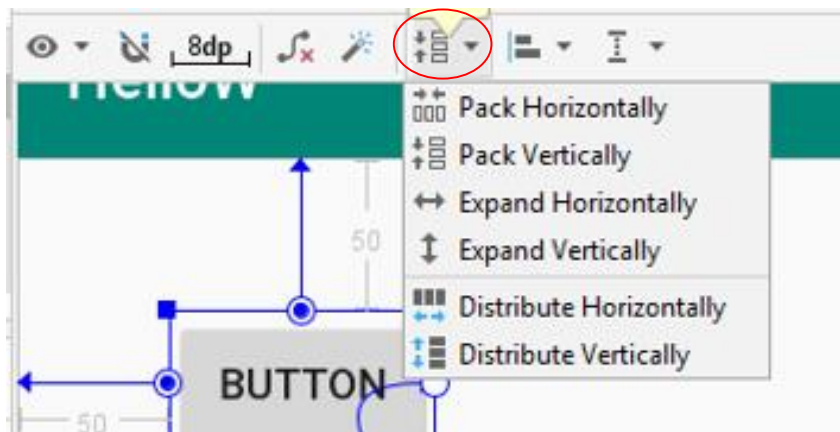


Рис. 19 Панель инструментов, элемент собрать -растянуть

Выравнивание – по горизонтали: по левому краю, по центру, по правому краю. Нижний ряд кнопок – это центрирование. Оно создает двухстороннюю привязку (рис.20).

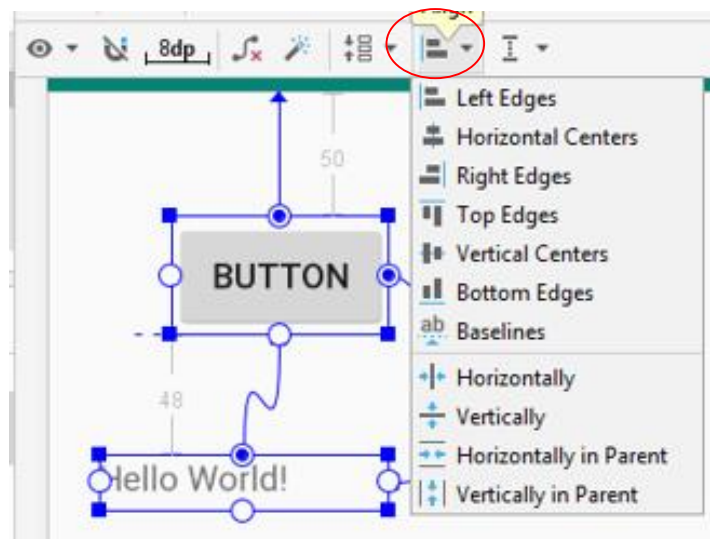


Рис. 20 Панель инструментов, выравнивание

Направляющие – это линии, которые можно использовать для создания привязок (рис.21).

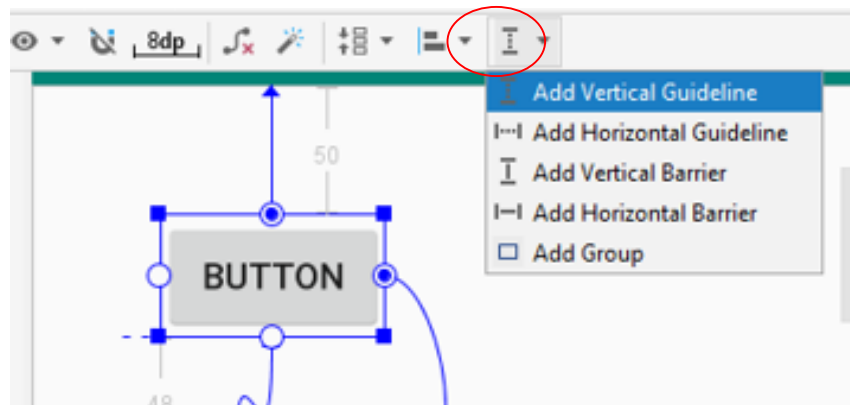


Рис.21 Панель инструментов, направляющее

Если у *View* есть двусторонняя вертикальная привязка и значение высоты установлено в *match_constraints* (0 dp), то *View* растянется по высоте между объектами привязки. Используя *aspect ratio* можно настроить элемента так, чтобы высота в этом случае не растягивалась, а зависела от ширины *View*. Для *View* с двусторонней вертикальной привязкой надо выставить значение высоты в 0 dp. *View* растягивается по высоте. Затем, включаем режим соотношения сторон, нажав на треугольник (рис. 22). Задаем соотношение ширины к высоте, например, 3 к 1. Т.е. высота теперь будет в три раза меньше ширины. С изменением ширины меняется и высота, чтобы соблюдалось установленное соотношение сторон 3:1 (рис.22).

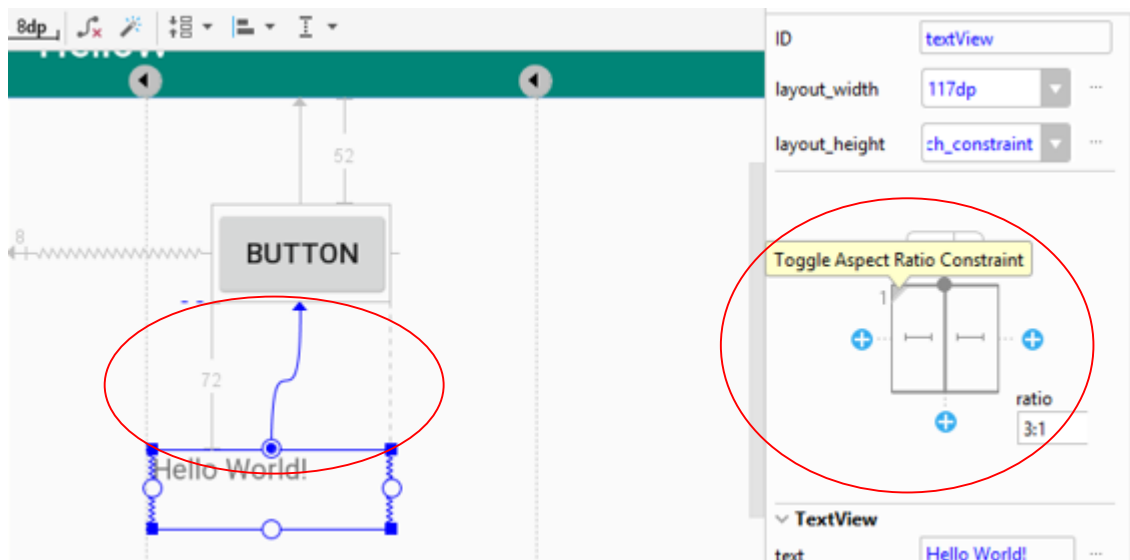


Рис. 22 Установка соотношения сорон

Цепочка позволит равномерно распределить несколько *View* в имеющемся свободном пространстве. Чтобы создать цепочку, необходимо выделить *View* и центрировать их по горизонтали или вертикали (рис.23).

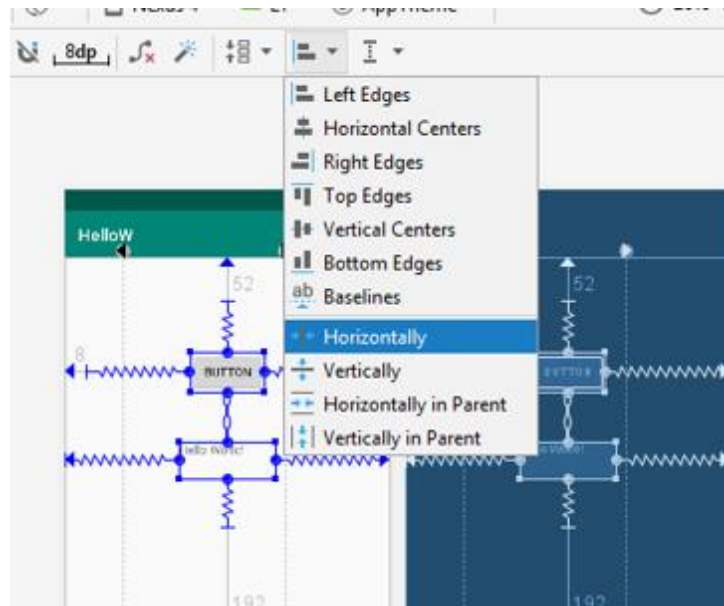


Рис. 23 Выбор режима выравнивания

Цепочка может быть в одном из трех режимов. *Spread* – свободное пространство равномерно распределяется между *View* и границами родителя. *Spread_inside* – свободное пространство равномерно распределяется только между *View*. Крайние *View* прижимаются к границам родителя. *Packed* – свободное пространство равномерно распределяется между крайними *View* и границами родителя. Можно использовать *margin*, чтобы сделать отступы между *View*. Режимы цепочки переключаются нажатием на значок цепи (рис.24).

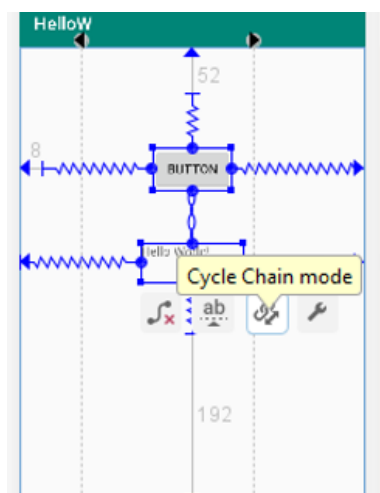


Рис. 24 Создание цепочек

В качестве объектов для привязки можно использовать не только границы родителя, но и другие объекты.

Цепочка позволяет указывать для *View* значение веса – *weight*. По умолчанию их нет в основном списке *Properties*.

Для оптимизации производительности UI необходимо использовать как можно меньшее количество разных *ViewGroup*. Для относительного расположения группы элементов можно использовать барьеры (рис.26).

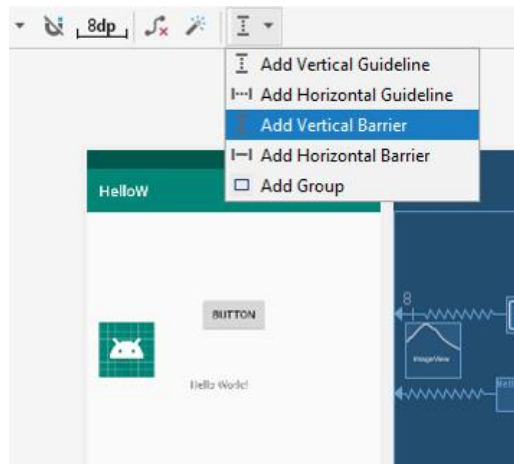


Рис. 26 Добавление барьеров

3.7 ScrollView

Контейнер *ScrollView* предназначен для создания прокрутки интерфейса, все элементы которого одновременно не могут поместиться на экране устройства.

4. Обращение к View

Чтобы обратиться к элементу *View* экрана из кода, нужен его *ID*. Он прописывается или в *Properties*, или в *layout-файлах*. 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 (рис.27). Идентификатор – целое уникальное число.

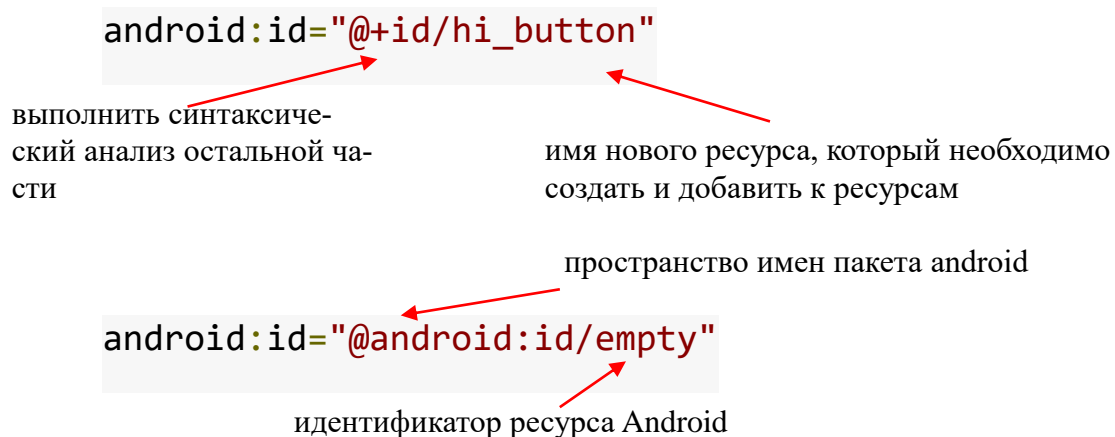


Рис. 27 Идентификация *View*

Названия элементов соответствуют названиям классов, а названия атрибутов соответствуют методам. Атрибуты бывают общие, специфические.

Листинг инициализации *layout*:

```
public class HelloWActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // устанавливаем в качестве интерфейса файл
        setContentView(R.layout.activity_hello_w);

        // получаем элемент
        Button buttonHi = (Button) findViewById(R.id.buttonHi);
        // переустанавливаем у него текст
        buttonHi.setText("IT");
    }
}
```

Чтобы обратиться к элементу программно понадобится метод *findViewById*. Он по *ID* возвращает *View* (см. приведенный выше листинг).

5. Ресурсы

5.1 Группирование ресурсов

Ресурсы представляют собой файлы разметки, отдельные строки, звуковые файлы, файлы изображений и т.д. Все ресурсы находятся в проекте в папке *res* (рис.28). Для различных типов ресурсов, определенных в проекте, в *res* создаются подпапки (представлены на рис. 28).

Если возьмем стандартный проект Android Studio, который создается по умолчанию, то там уже есть несколько папок для различных ресурсов в *res*:

- *animator/*: анимация 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/*: макеты;

- *menu/*: меню приложения;
- *raw/*: различные файлы, которые сохраняются в исходном виде;
- *values/*: xml-файлы, которые содержат различные значения, например, ресурсы строк;
- *xml/*: Произвольные xml-файлы.

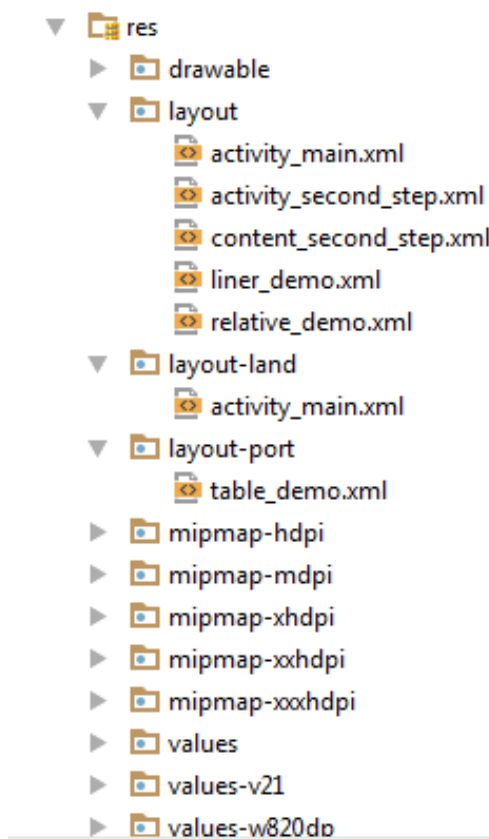


Рис.28 Ресурсы приложения

Макет пользовательского интерфейса по умолчанию сохранен в каталоге *res/layout/*, можно указать другой макет для использования на экране с альбомной ориентацией, сохранив его в каталоге *res/layout-land/*. Android автоматически применяет соответствующие ресурсы, сопоставляя текущую конфигурацию устройства с именами каталогов ресурсов.

Существуют ресурсы по умолчанию и альтернативные ресурсы. Ресурсы по умолчанию должны использоваться независимо от конфигурации устройства или в том случае, когда отсутствуют альтернативные ресурсы, соответствующие текущей конфигурации. Альтернативные ресурсы предназначены для работы с определенными конфигурациями. Чтобы указать, что группа ресурсов предназначена для определенной конфигурации, добавьте соответствующий квалификатор к имени каталога.

5.2 Предоставление альтернативных ресурсов и квалификаторы конфигурации

Квалификатор *hdpi* (на рис.28) указывает, что ресурсы в этом каталоге предназначены для устройств, оснащенных экраном высокой плотности. Идентификатор ресурса всегда одинаков, но Android выбирает версию каждого ресурса, которая оптимально соответствует текущему устройству, сравнивая информацию о конфигурации устройства с квалификаторами в имени каталога ресурсов.

Язык задается двухбуквенным кодом языка ISO 639-1, к которому можно добавить двухбуквенный код региона ISO 3166-1-alpha-2 (которому предшествует строчная буква *r*, для обозначения кода региона). Например *en*, *fr*, *en-rUS*.

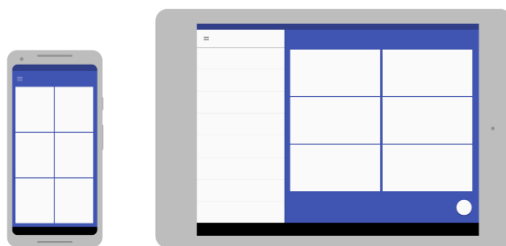
Квалификатор *ldrtl* означает «направление макета справа налево». Квалификатор *ldltr* означает «направление макета слева направо» и используется по умолчанию. Эти квалификаторы можно применять к любым ресурсам, таким как макеты, графические элементы или значения.

w<N>dp – указывает минимальную доступную ширину экрана в единицах *dp*, для которой должен использоваться ресурс, заданный значением *<N>* (например, *w720dp*). Это значение конфигурации будет изменяться в соответствии с текущей фактической шириной при изменении альбомной/книжной ориентации. Когда приложение предоставляет несколько каталогов ресурсов с разными значениями этой конфигурации, система использует ширину, ближайшую к текущей ширине экрана устройства, но не превышающую ее.

h<N>dp – указывает минимальную доступную высоту экрана в пикселах, для которой должен использоваться ресурс, заданный значением *<N>* (например, *h720dp*). Это значение конфигурации будет изменяться в соответствии с текущей фактической высотой при изменении альбомной/книжной ориентации.

port – устройство в портретной (вертикальной) ориентации.

land – устройство в книжной (горизонтальной) ориентации. Ориентация может измениться за время работы приложения, если пользователь поворачивает экран.



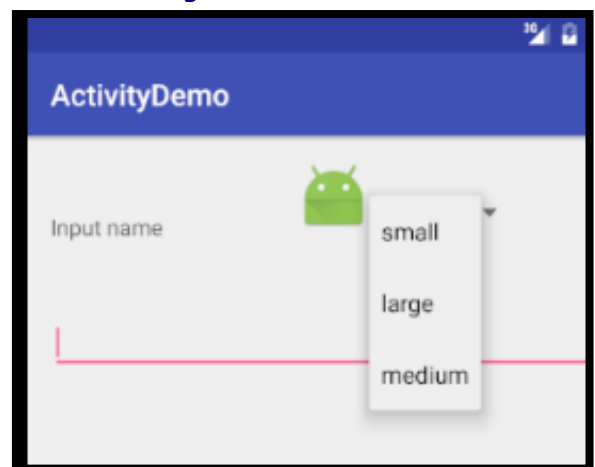
5.3 Строковые ресурсы

По умолчанию для ресурсов строк (хранится в виде пар «имя/значение строки») применяется файл *strings.xml*, но разработчики могут добавлять дополнительные файлы ресурсов в каталог проекта *res/values*. Необходимо соблюдать структуру файла, иметь корневой узел *<resources>* и иметь один или несколько элементов *<string>*:

```
<resources>
    <string name="app_name">ActivityDemo</string>
    <string name="enter">Enter</string>
    <sting name ="login"> Login</sting>
    <string name ="in"> Input name </string>
    <string name="greeting">
Good morning</string>
</resources>
```

Для хранения массива строк используется *<string-array>*:

```
<string-array name="images">
    <item>small</item>
    <item>large</item>
    <item>medium</item>
</string-array>
```



5.4 Ресурсы plurals

Plurals предназначен для описания количества элементов (при изменении окончания в зависимости от числительного, которое с ним употребляется). Для задания ресурса используется элемент *<plurals>*, для которого существует атрибут *name*, получающий в качестве значения произвольное название, по которому потом ссылаются на данный ресурс. Сами наборы строк вводятся дочерними элементами *<item>*. Этот элемент имеет атрибут *quantity*, указывающее, когда эта строка используется:

```
?xml version="1.0" encoding="utf-8" ?>
<resources>
    <plurals name="student">
        <item quantity="one">%d студент</item>
        <item quantity="few">%d студентов</item>
        <item quantity="many">%d студентов</item>
    </plurals>
</resources>
```

Использование данного ресурса возможно только в коде java.

5.5 Ресурсы *dimension*

Определение размеров должно находиться в каталоге *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, *dimension* определяется в корневом элементе *<resources>*. Тег *<dimen>* обозначает ресурс и в качестве значения принимает некоторое значение размера в одной из принятых единиц измерения. Например:

```
<dimen name="activity_horizontal_margin">16dp</dimen>
<dimen name="activity_vertical_margin">16dp</dimen>
<dimen name="text_size">16sp</dimen>
```

Здесь определены два ресурса для отступов *activity_horizontal_margin* и *activity_vertical_margin* и ресурс *text_size*, (высота шрифта 16 sp). Названия ресурсов могут быть произвольными.

К ресурсу можно обратиться следующим образом:

```
int leftPadding = (int) getResources()
                    .getDimension(R.dimen.activity_hori-
tal_margin);
int topPadding = (int) getResources()
                    .getDimension(R.dimen.activity_verti-
cal_margin);
```

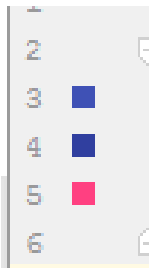
Для получения ресурса в XML после "*@dimen/*" указывается имя ресурса:

```
<TextView
    android:textSize="@dimen/text_size"
    android:layout_width="wrap_content"/>
```


5.6 Ресурсы color

Ресурсы цветов (color) должны храниться в файле по пути *res/values* и также, как и ресурсы строк, заключены в тег `<resources>`. По умолчанию при создании самого простого проекта в папку *res/values* добавляется файл *colors.xml*.

Цвет определяется с помощью элемента `<color>`. Атрибут *name* устанавливает название цвета, которое будет использоваться в приложении, а шестнадцатеричное число – значение цвета:



```
1
2 <?xml version="1.0" encoding="utf-8"?>
3 <resources>
4     <color name="colorPrimary">#3F51B5</color>
5     <color name="colorPrimaryDark">#303F9F</color>
6     <color name="colorAccent">#FF4081</color>
7 </resources>
```

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).

Для получения цвета применяется метод `ContextCompat.getColor()`, который в качестве первого параметра принимает текущий объект *activity*, а второй параметр – идентификатор цветового ресурса:

```
int textColor = ContextCompat.getColor(this, R.color.colorPrimary);
```

5.7 Доступ к ресурсам

Когда происходит компиляция проекта сведения обо всех ресурсах добавляются *R.java*, который можно найти в проекте по пути *app\build\generated\source\r\debug\[назет_приложения]*.

Существует два способа доступа к ресурсам: в файле исходного кода: *тип_ресурса.имя* (например *R.string.hello*) и в файле XML: *тип_ресурса/имя* (*@string/hello*).

Для получения ресурсов в классе активности используется метод `getResources()`, который возвращает объект *android.content.res.Resources*. Чтобы получить ресурс, надо у полученного объекта *Resources* вызвать один из методов:

- `getString()`: получает строку из файла *strings.xml* по числовому идентификатору;

- *getDimension()*: получает числовое значение из ресурса *dimen*;
- *getDrawable()*: получает графический файл;
- *getBoolean()*: получает значение *boolean* и т.д.

5.8 Добавление *layout* для смены ориентации экрана

По умолчанию *layout*-файл настраивается под вертикальную ориентацию экрана. Если интерфейс насыщен, то для удобства необходим еще один *layout*-файл, который был бы ориентирован под горизонтальную ориентацию и возможно вывел бы элементы по-другому.

Для этого необходимо в папке *res/layout* создать новый *Layout resource file*. Название файла указывается такое же. Затем, добавляется спецификатор, который даст приложению понять, что этот *layout*-файл надо использовать в горизонтальной ориентации. В списке спецификаторов слева снизу находим *Orientation* и включаем использование спецификатора ориентации (рис.30).

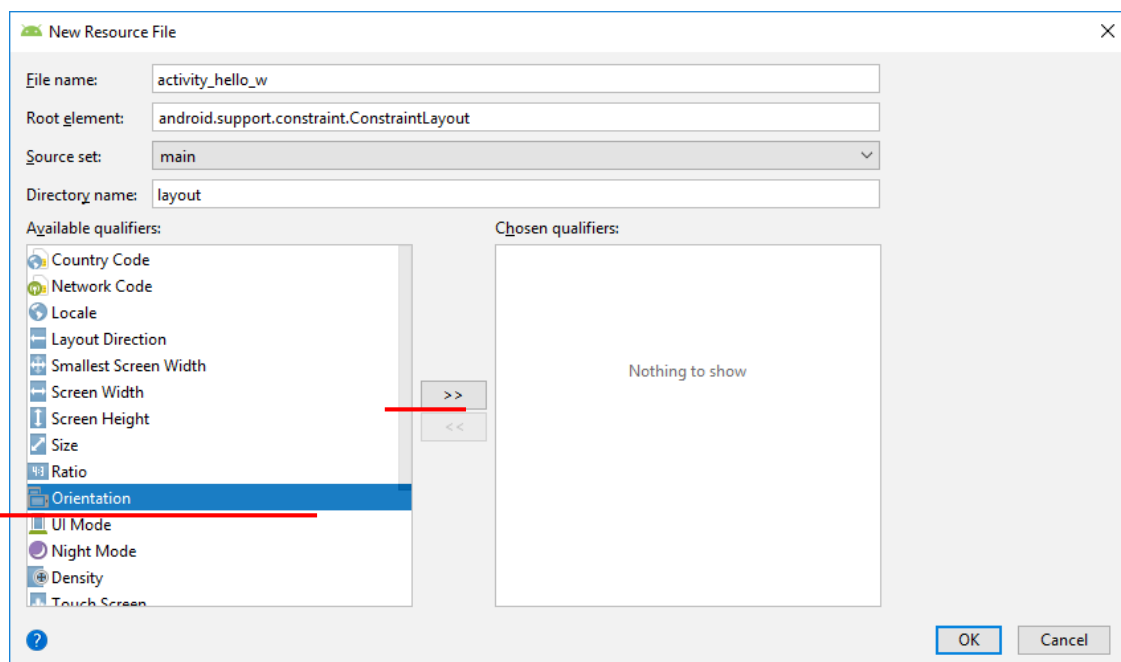


Рис. 30 окно добавления *layout*-файла

Обратите внимание, значение поля *Directory name* изменилось (рис.31). Новый *layout*-файл будет создан в папке *res/layout-land*, а не *res/layout*, как обычно.

Посмотрим на структуру папок проекта (рис.32). Теперь там два файла *activity_main*: обычный и *land*. Каждый из файлов можно менять независимо и соответственно получать разное представление.

31 Добавление *layout*-файла для горизонтальной ориентации экрана

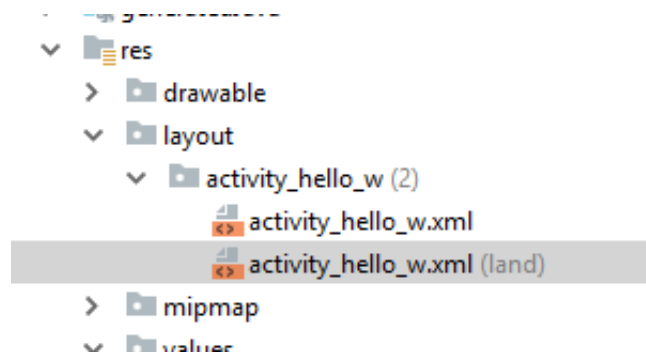


Рис. 32 Структура папок проекта

Активность читает *layout*-файл, который указан в методе *setContentView* и отображает его содержимое. При этом учитывается ориентация устройства, и в случае горизонтальной ориентации берет файл из папки *res/layout-land* (если он существует).

6 Обработка нажатий View

Пусть есть следующий макет:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".HelloW">

    <TextView
        android:id="@+id/textView"
        android:layout_width="114dp"
        android:layout_height="44dp"
        android:layout_marginTop="144dp"
        android:text="@string/helloW"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.474"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
```

```

        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="244dp"
        android:text="@string/by"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.197"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_chainStyle="packed" />

<ImageView
    android:id="@+id/imageView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="56dp"
    android:layout_marginTop="36dp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:srcCompat="@mipmap/ic_launcher_round" />

</android.support.constraint.ConstraintLayout>

```

В файле *HelloW.java* описание объектов вынесем за пределы метода *onCreate* для того, чтобы можно было из любого метода обращаться к ним:

```

public class HelloW extends AppCompatActivity {
    TextView txtView;
    Button btnBy;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
    }
}

```

В методе *onCreate* эти объекты заполняются методом *findViewById*. В методе *onCreate* класса *HelloW* напишем следующий код:

```

txtView = (TextView) findViewById(R.id.textView);
btnBy = (Button) findViewById(R.id.button);

```

После того как установили *layout* и захватили *View* на экране, с объектами можно работать. Например, научить кнопку реагировать на нажатие. Для этого у кнопки есть метод *setOnClickListener*:

```
// создаем обработчик нажатия
// присвоим обработчик кнопке By
btnBy.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Меняем текст в TextView
        txtView.setText("The button was clicked");
    }
});
```

или так:

```
btnBy.setOnClickListener((View v) -> {
    txtView.setText("The button was clicked");
});
```

Запустите код. Если произошла ошибка, то можно перейти на вкладку *Logcat* и прочитать информацию об ошибке (рис.3).

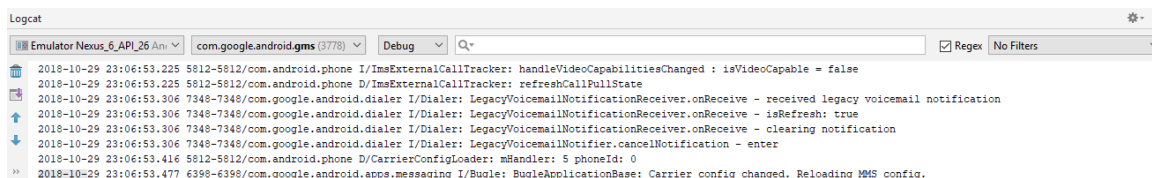


Рис. 33 Окно *Logcat*

Если ошибок нет, то после запуска приложения и нажатия на кнопку вы увидите сообщение на экране (рис.34).

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*.

Добавим еще одну кнопку (рис.35). Итак, есть *TextView* с текстом и две кнопки. Сделаем так, чтобы по нажатию кнопки менялось содержимое *TextView*. По нажатию кнопки *BY* – будем выводить текст: «*By FIT*», по нажатию *Hello* – «*Hello FIT*». Сделаем это с помощью одного обработчика, который будет обрабатывать нажатия для обеих кнопок.

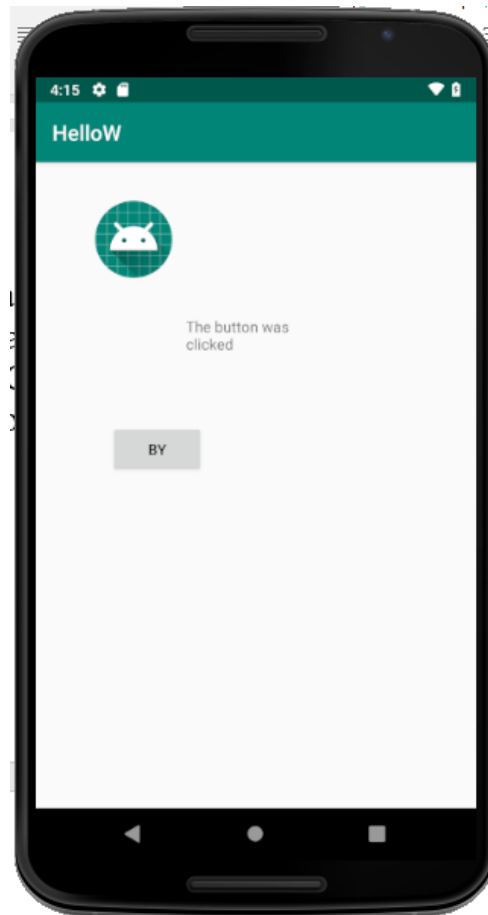


Рис. 34 Результат выполнения приложения

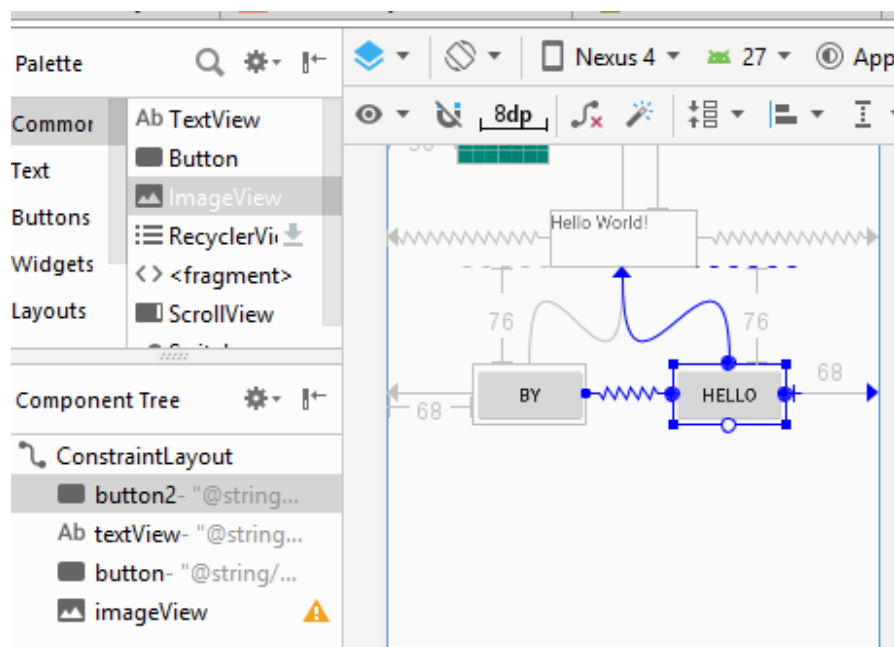


Рис. 35 Добавление новой кнопки в активность

Для реализации необходимо выполнить следующие шаги: создать обработчик; заполнить метод *onClick*; присвоить обработчик кнопке. Перепишем код следующим образом:

```
public class HelloW extends AppCompatActivity {

    public final String TAG = "HelloW";
    TextView txtView;
    Button btnBy;
    Button btnHello;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_hello_w);
        // найдем View-элементы
        txtView = (TextView) findViewById(R.id.textview);
        btnBy = (Button) findViewById(R.id.button);
        btnHello = (Button) findViewById(R.id.button2);

        // создаем обработчик нажатия
        // присвоим обработчик кнопке By
        View.OnClickListener onCLBtn = new View.OnClickListener() {
            // создаем обработчик
            @Override
            public void onClick(View v) {
                switch (v.getId()) {
                    case R.id.button:
                        // кнопка BY
                        txtView.setText("BY FIT");
                        break;
                    case R.id.button2:
                        // кнопка Hello
                        txtView.setText("Hello FIT");
                        break;
                }
            }
        };
        // назначаем обработчик
        btnBy.setOnClickListener(onCLBtn);
        btnHello.setOnClickListener(onCLBtn);
    }
}
```

В примере использовался обработчик с реализацией метода *onClick*. В параметрах ему подается объект класса *View*. Это *View*, на которой произошло нажатие, и, который вызвал обработчик. В данном случае это будет либо кнопка *BY*, либо *HELLO*. Поэтому надо узнать *ID* этой *View* и сравнить его с *R.id.button* и *R.id.button2*, чтобы определить какая кнопка была нажата. Чтобы получить *ID* используется метод *getId*.

После создания обработчика его присвоили кнопкам. Обеим кнопкам присваиваем один и тот же обработчик. Результат на рис.36.

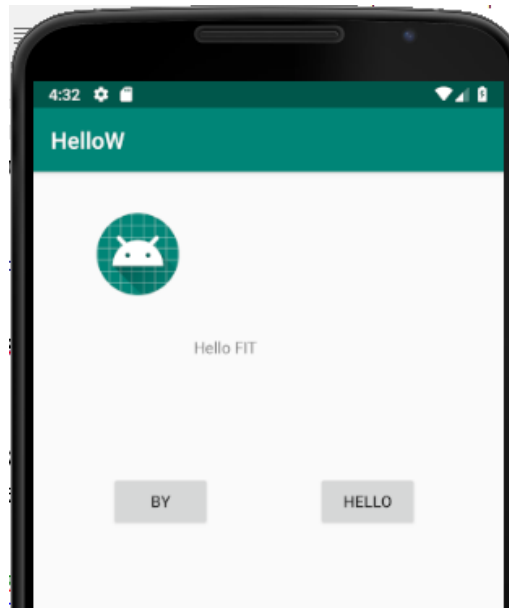


Рис. 36 Результат выполнения приложения

Есть еще один способ создания обработчика, который не потребует создания объектов. Можно использовать уже созданный объект активности. Указываем, что класс активности реализует интерфейс *View.OnClickListener* и реализуем метод *onCreate*:

```
public class HelloW extends AppCompatActivity implements
View.OnClickListener{
...
@Override
    public void onClick(View v) {
    }
}
```

Есть третий способ. В *layout*-файле при определении кнопки используем атрибут *onClick*. В нем указываем имя метод активности. Этот метод сработает при нажатии на кнопку:


```

<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginTop="76dp"
    android:layout_marginEnd="68dp"
    android:text="@string/hello"
    android:onClick="onClick"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.927"
    app:layout_constraintStart_toEndOf="@+id/button"
    app:layout_constraintTop_toBottomOf="@+id/textView"
    app:layout_constraintVertical_chainStyle="packed" />

```

Далее, добавляем этот метод в *activity* (*HelloW.java*). Требования к методу: модификатор *public*, возвращаемое значение *void* и параметр тип *View*:

```

public void onClick (View v) {
    // действия при нажатии на кнопку
}

```

6 Вывод Log-сообщений

Когда тестируется работа приложения, полезно видеть логи работы. Они отображаются в окне *LogCat*. Логи имеют разные уровни важности: *ERROR*, *WARN*, *INFO*, *DEBUG*, *VERBOSE* (в порядке убывания критичности). Писать логи можно с помощью класса *Log* и его методов *Log.v()*, *Log.d()*, *Log.i()*, *Log.w()* и *Log.e()*. Названия методов соответствуют уровню логов, которые они пишут. Параметры методов – тэг и текст сообщения. Тэг – это метка, чтобы легче было в системных логах найти нужное сообщение.

Изменим код активности и выведем лог с помощью метода *Log.d*:

```

public class HelloW extends AppCompatActivity {

    public final String TAG = "HelloW";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_hello_w);
        Log.d(TAG, "On creat");
    }
}

```

}

После запуска приложения изучите сообщения в *Logcat* (рис.37). Когда логов много, их можно отфильтровать по типу сообщения (рис.38-39).

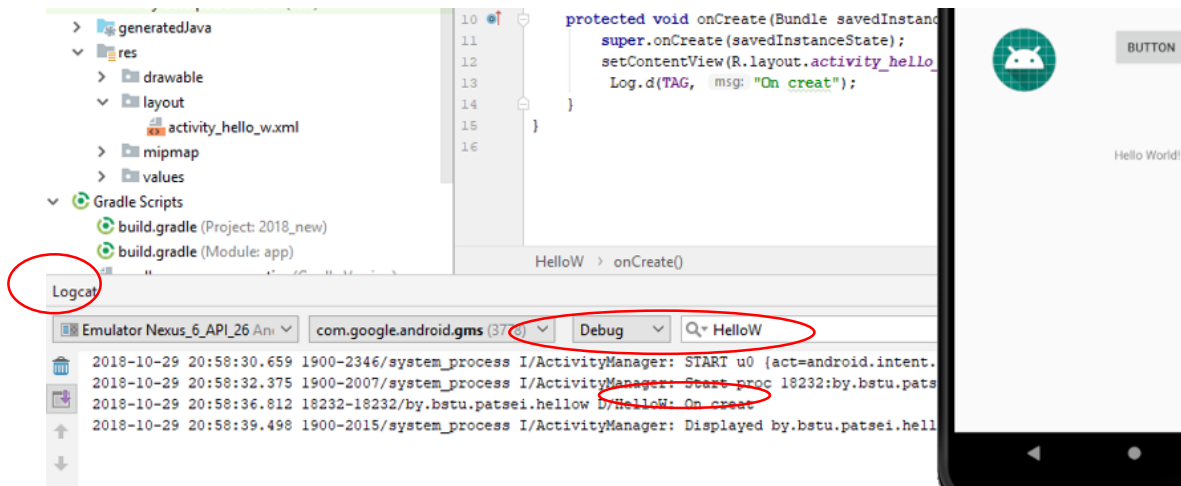


Рис. 37 Вывод Log-сообщений



Рис. 38 Фильтрация Log-сообщений

Изменим код *HelloW.java*. Добавим *Info*-логи:

```
View.OnClickListener onCLBtn = new View.OnClickListener() {  
    // создаем обработчик  
    @Override  
    public void onClick(View v) {  
  
        switch (v.getId()) {  
  
            case R.id.button:  
                // кнопка BY  
                textView.setText("BY FIT");  
                Log.i ("ActivityHelloW", "button By clicked");  
                break;  
  
            case R.id.button2:
```

```

        // кнопка Hello
        txtView.setText("Hello FIT");
        Log.i ("ActivityHelloW", "button Hello
clicked");
        break;
    }
}
};

```

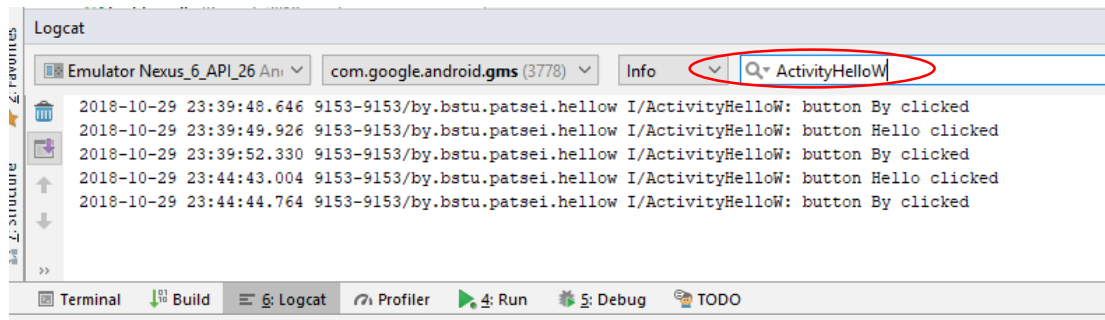


Рис. 39 Фильтрация сообщений

7 Всплывающие окна Toast

Для создания простых уведомлений в Android используется класс *Toast*. Фактически *Toast* представляет всплывающее окно с текстом, которое отображается в течение некоторого времени.

Объект *Toast* нельзя создать в коде разметки XML. Для его создания применяется метод *Toast.makeText()*, в который передается три параметра: *context* – текущий контекст (текущий объект активности), *text*, отображаемый текст и *duration* – время отображения окна (*Toast.LENGTH_LONG* – 3500 миллисекунд, *Toast.LENGTH_SHORT* – 2000 миллисекунд):

```

Toast.makeText(getApplicationContext(), "HelloW", Toast.LENGTH_LONG)
    .show();

```

Отредактируем метод *onClick*. Сделаем так, чтобы всплывало сообщение о том, какая кнопка была нажата (результата выполнения представлен на рис.40):

```

View.OnClickListener onCLBtn = new View.OnClickListener() {

    // создаем обработчик
    @Override
    public void onClick(View v) {
        switch (v.getId()) {

```

```

        case R.id.button:
            txtView.setText("BY FIT");
            Log.i ("ActivityHelloW", "button By clicked");
            Toast.makeText(getApplicationContext(), "But-
ton By clicked", Toast.LENGTH_LONG).show();
            break;

        case R.id.button2:
            txtView.setText("Hello FIT");
            Log.i ("ActivityHelloW", "button Hello
clicked");
            Toast.makeText(getApplicationContext(), "But-
ton Hello clicked", Toast.LENGTH_LONG).show();
            break;
    }
}
};

```

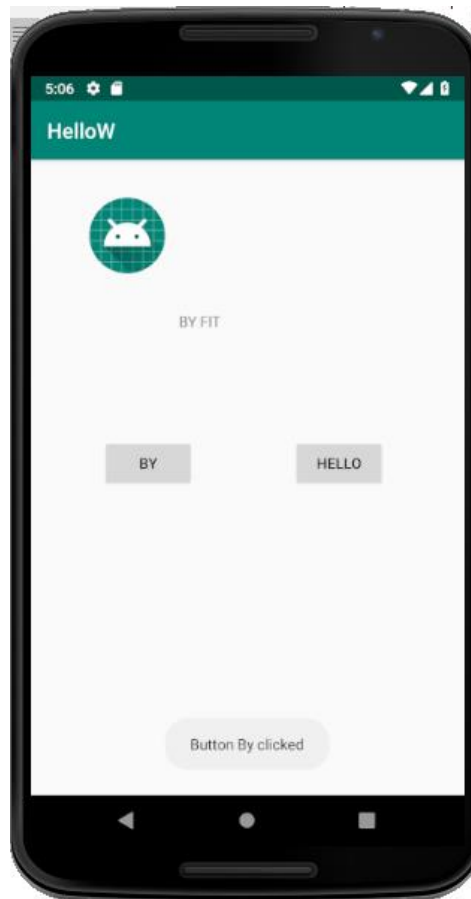


Рис. 40 Результат выполнения приложения

8. Элементы управления

8.1 *TextView*

TextView предназначен для вывода текста на экран без возможности его редактирования. Его основные атрибуты:

- *android:text*: устанавливает текст элемента;
- *android:textSize*: устанавливает высоту текста;
- *android:background*: задает фоновый цвет элемента в виде цвета в шестнадцатиричной записи или в виде цветового ресурса;
- *android:textColor*: задает цвет текста;
- *android:textAllCaps*: при значении *true* делает все символы в тексте заглавными;
- *android:textDirection*: устанавливает направление текста. По умолчанию используется направление слева направо;
- *android:textAlignment*: задает выравнивание текста;
- *android:fontFamily*: устанавливает тип шрифта.

Иногда необходимо вывести на экран какую-нибудь ссылку, либо телефон, по нажатию на которые выполнялось бы определенное действие. Для этого в *TextView* определен атрибут *android:autoLink*, который принимает значения:

- *none*: отключает все ссылки;
- *web*: включает все веб-ссылки;
- *email*: включает ссылки на электронные адреса;
- *phone*: включает ссылки на номера телефонов;
- *map*: включает ссылки на карту;
- *all*: включает все вышеперечисленные ссылки.

8.2 *EditText*

Элемент *EditText* является подклассом класса *TextView*. Он также представляет текстовое поле, но с возможностью ввода и редактирования текста. В *EditText* можно использовать все те же возможности, что и в *TextView*.

Из тех атрибутов, что не рассматривались, следует отметить *android:hint*. Позволяет задать текст, который будет отображаться в качестве подсказки, если элемент *EditText* пуст. Кроме того, можно использовать атрибут *android:inputType*, который позволяет задать клавиатуру для ввода:

- *text*: обычная клавиатура для ввода однострочного текста;

- *textMultiLine*: многострочное текстовое поле;
 - *textEmailAddress*: обычная клавиатура, на которой присутствует символ @, ориентирована на ввод email;
 - *textUri*: обычная клавиатура, на которой присутствует символ /, ориентирована на ввод интернет-адресов;
 - *textPassword*: клавиатура для ввода пароля;
 - *textCapWords*: при вводе первый введенный символ слова представляет заглавную букву, остальные – строчные;
 - *number*: числовая клавиатура;
 - *phone*: клавиатура в стиле обычного телефона;
 - *date*: клавиатура для ввода даты;
 - *time*: клавиатура для ввода времени;
 - *datetime*: клавиатура для ввода даты и времени.
- Определим *EditText* (результат представлен на рис. 41):

```
<EditText
    android:layout_marginTop="16dp"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:hint="Введите сообщение"
    android:inputType="textMultiLine"
    android:gravity="top" />
```

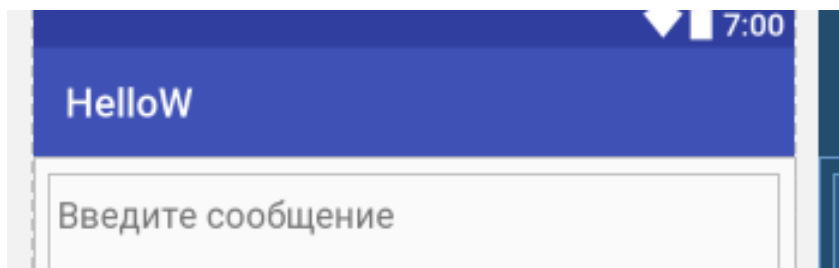


Рис. 41 Вывод элемента *EditText*

8.3 *Button*

Одним из часто используемых элементов являются кнопки, которые представлены классом *android.widget.Button*. Ключевой особенностью кнопок является возможность взаимодействия с пользователем через нажатия.

Некоторые ключевые атрибуты, которые можно задать у кнопок:

- *text*: задает текст на кнопке;
- *textColor*: задает цвет текста;
- *background*: задает фоновый цвет;

- *textAllCaps*: при значении *true* устанавливает текст в верхнем регистре;
- *onClick*: задает обработчик нажатия кнопки.

8.4 Checkboxes

Элементы *Checkbox* представляют собой флажки, которые могут находиться в отмеченном и неотмеченном состоянии. Флажки позволяют производить множественный выбор. С помощью слушателя *OnCheckedChangeListener* можно отслеживать изменения флажка.

8.5 RadioButton

Схожую с флажками функциональность предоставляют переключатели, которые представлены классом *RadioButton*. Чтобы создать список переключателей для выбора, вначале надо создать объект *RadioGroup*, который будет включать в себя все переключатели.

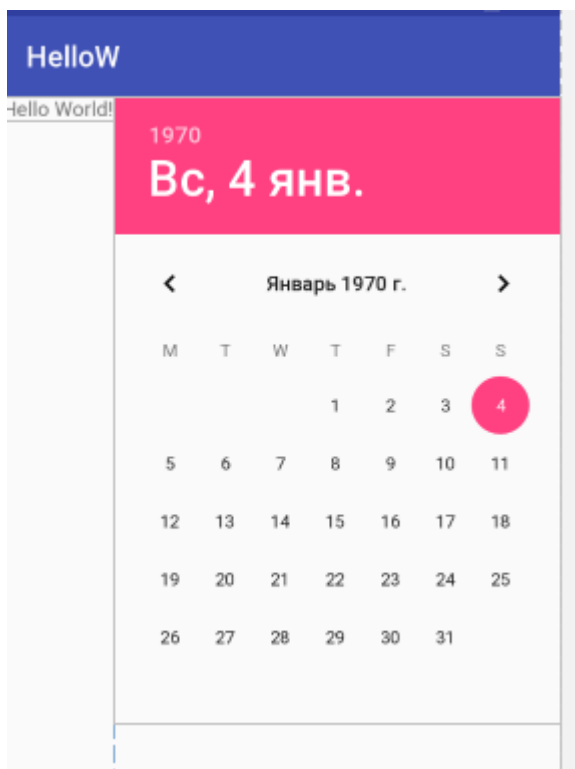
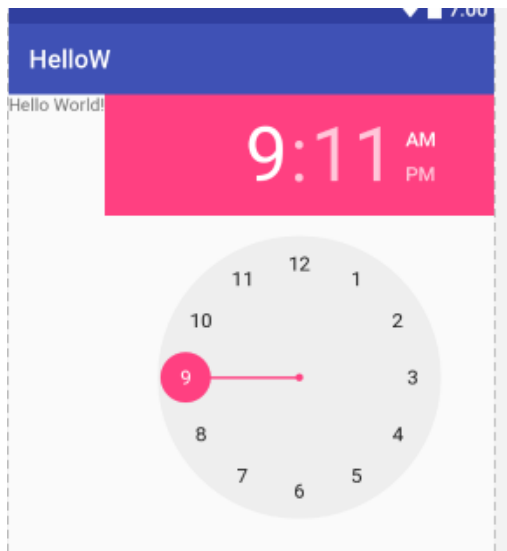
8.6 ToggleButton

Атрибуты *android:textOn* и *android:textOff* задают текст кнопки в отмеченном и неотмеченном состоянии соответственно. И также, как и для других кнопок, можно обработать нажатие на элемент с помощью события *onClick*:

```
<ToggleButton
    android:id="@+id/toggleButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:textOn="ВКЛ"
    android:textOff="ВЫКЛ" />
```

DatePicker и TimePicker

```
TimePicker android:id="@+id/timePicker"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```



```
DatePicker android:id="@+id/datePicker"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

```
DatePicker dp = (DatePicker)this.findViewById(R.id.datePicker);
// Месяц начиная с нуля. Для отображения добавляем 1.
textView1.setText("Дата по умолчанию " + dp.getDayOfMonth() + "/" +
    (dp.getMonth() + 1) + "/" + dp.getYear());
```



```
dp.init(2018, 03, 01, null); //апрель
```

```
TimePicker tp = (TimePicker)this.findViewById(R.id.timePicker);  
java.util.Formatter timeF = new java.util.Formatter();  
timeF.format("Время по умолчанию %d:%02d", tp.getHour(),  
            tp.getMinute());
```

```
textView2.setText(timeF.toString());  
tp.setIs24HourView(true);
```

Ц

и
ф

<DigitalClock

о android:layout_width="wrap_content"

в android:layout_height="wrap_content" />

<AnalogClock

е android:layout_width="wrap_content"

android:layout_height="wrap_content" />

и

а

н

а

л

о

г

о

в

ы

е

<SeekBar

ч android:id="@+id/seekBar"

а android:layout_width="265dp"

с android:layout_height="wrap_content"

ы />

Кроме элемента TimePicker Android поддерживает для отображения времени такие эле-
менты: Spinner (аналоговые и цифровые часы (элементы AnalogClock и DigitalClock):

<Spinner

а android:id="@+id/spinner"

а android:layout_width="match_parent"

а android:layout_height="wrap_content"

ы />

<ProgressBar

а android:id="@+id/progressBar"

а style="?android:attr/progressBarStyle"

```

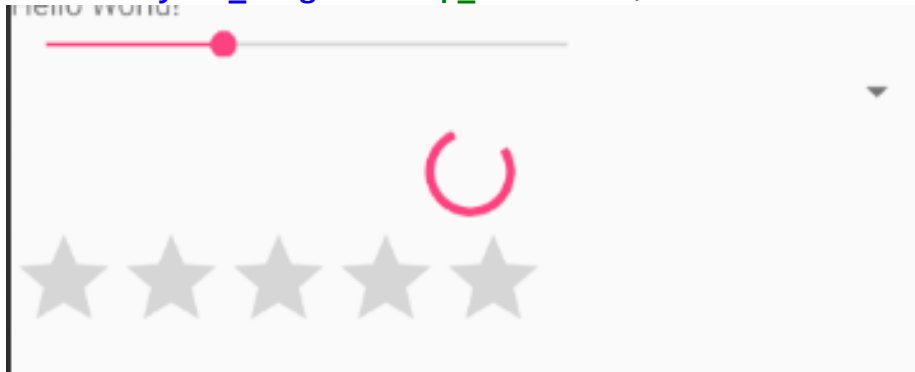
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />

```

```

<RatingBar
    android:id="@+id/ratingBar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

```



Spinner

Составной элемент, отображающий TextView в сочетании с соответствующим ListView, которое позволяет выбрать элемент списка для отображения в текстовой строке. Сама строка состоит из объекта TextView и кнопки, при нажатии на которую всплывает диалог выбора. Внешне этот элемент напоминает тэг <SELECT> в HTML.

```

Spinner spinner = (Spinner) findViewById(R.id.spinner);

ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromRe-
source(this,
    R.array.planets_array, android.R.layout.simple_spin-
ner_item);

// Определение макета для вывода
adapter.setDropDownViewResource(android.R.layout.simple_spin-
ner_dropdown_item);

// Применить адаптер
spinner.setAdapter(adapter);

```