2. СОЗДАНИЕ ИНТЕРФЕЙСА. МАКЕТЫ. ЭЛЕМЕНТЫ UI. РЕСУРСЫ

2.1. 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.

2.1.1. Разработка Layout

Макет определяет визуальную структуру пользовательского интерфейса. Существует два способа объявить макет (рис. 2.1):

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

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.

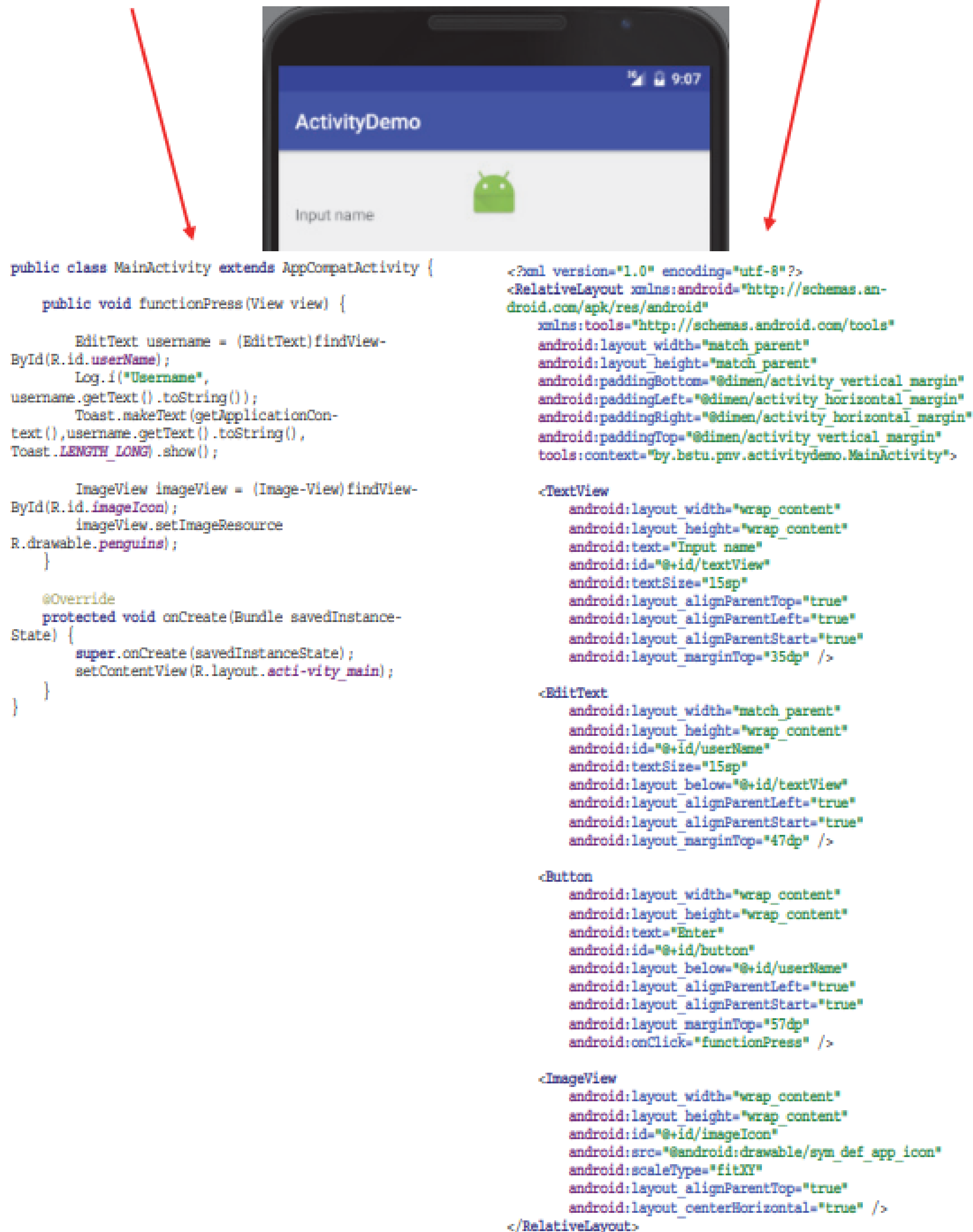

Рис. 2.1. Разработка макета пользовательского интерфейса Android-приложения

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

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. 2.2).

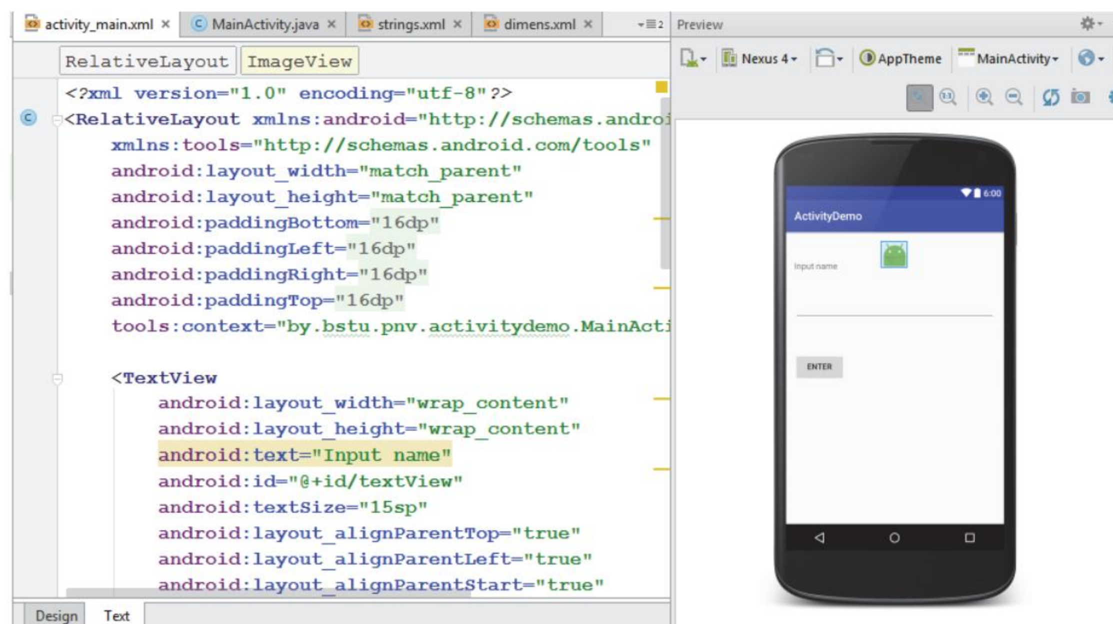

Рис. 2.2. Разработка макета пользовательского интерфейса на основе XML

2.1.2. 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. 2.3).

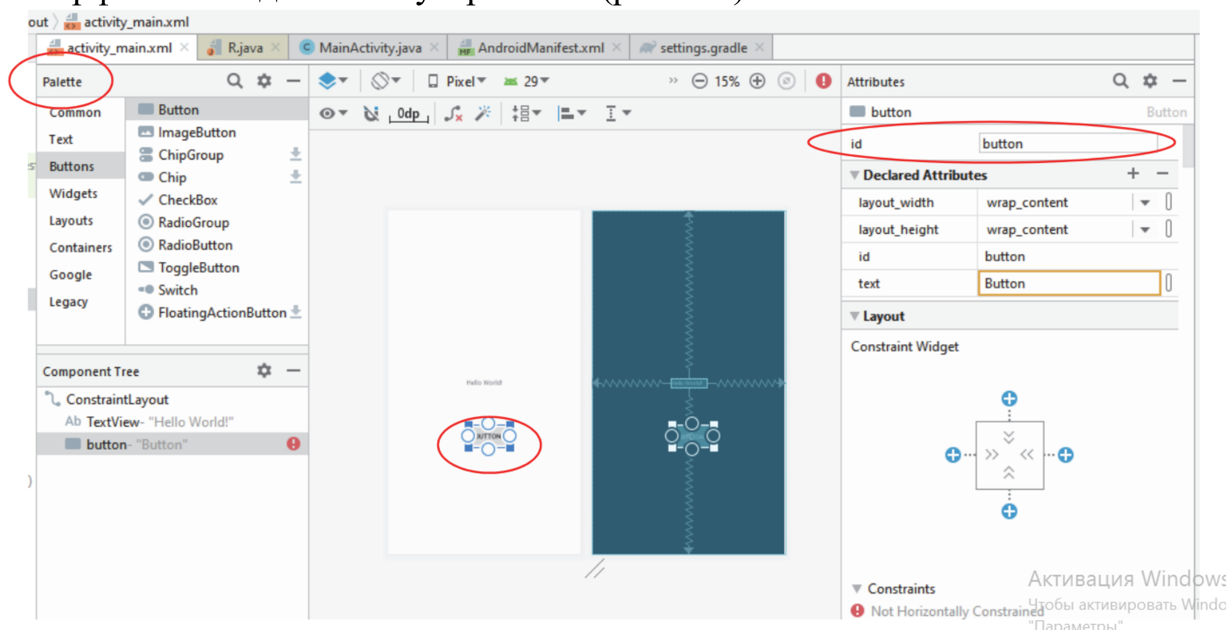

Рис. 2.3. Работа в режиме *Design*

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.

2.2. 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 1 dp = 1 px. Если размер экрана больше или меньше, чем 160 dpi, количество пикселей,

которые применяются для отрисовки 1 dp, 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, величина базируется на физических размерах экрана;

– *mm* – миллиметры; – *in* – дюймы.

Предпочтительными единицами для использования являются dp.

Для упрощения работы с размерами все они разбиты на несколь- ко групп:

– ldpi (low) ~120 dpi;

– mdpi (medium) ~160 dpi;

– hdpi (high) ~240 dpi;

– xhdpi (extra-high) ~320 dpi;

– xxhdpi (extra-extra-high) ~480 dpi;

– xxxhdpi (extra-extra-extra-high) ~640 dpi.

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. 2.4);

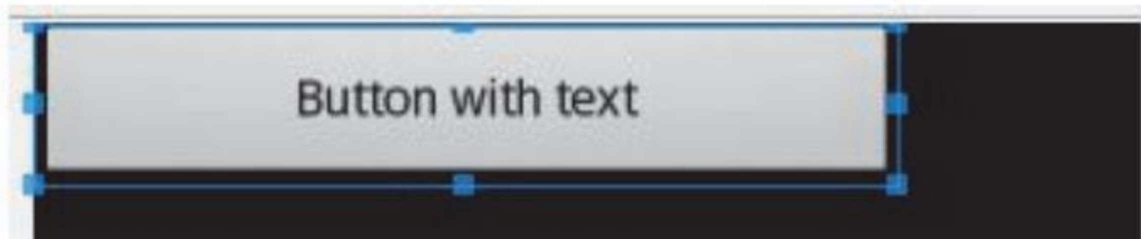

Рис. 2.4. Определение элемента с точными размерами

– *значение wrap_content*: элемент растягивается до тех границ, которые достаточны, чтобы вместить все его содержимое (рис. 2.5);

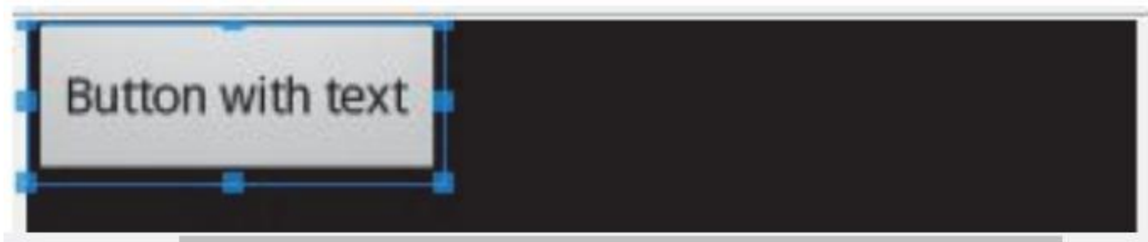

Рис. 2.5. Определение элемента в соответствии с содержимым

– *значение match_parent*: элемент заполняет всю область родительского контейнера (рис. 2.6).

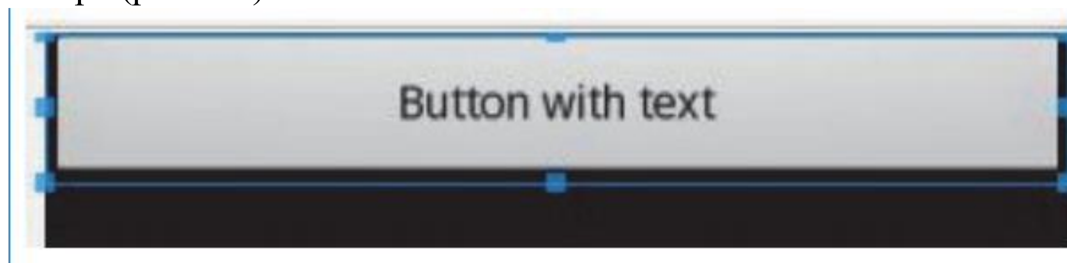

Рис. 2.6. Определение элемента в соответствии с родительским контейнером

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

```
android:minWidth="200dp" android:maxWidth="250dp"
android:minHeight="100dp" android:maxHeight="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. 2.7).

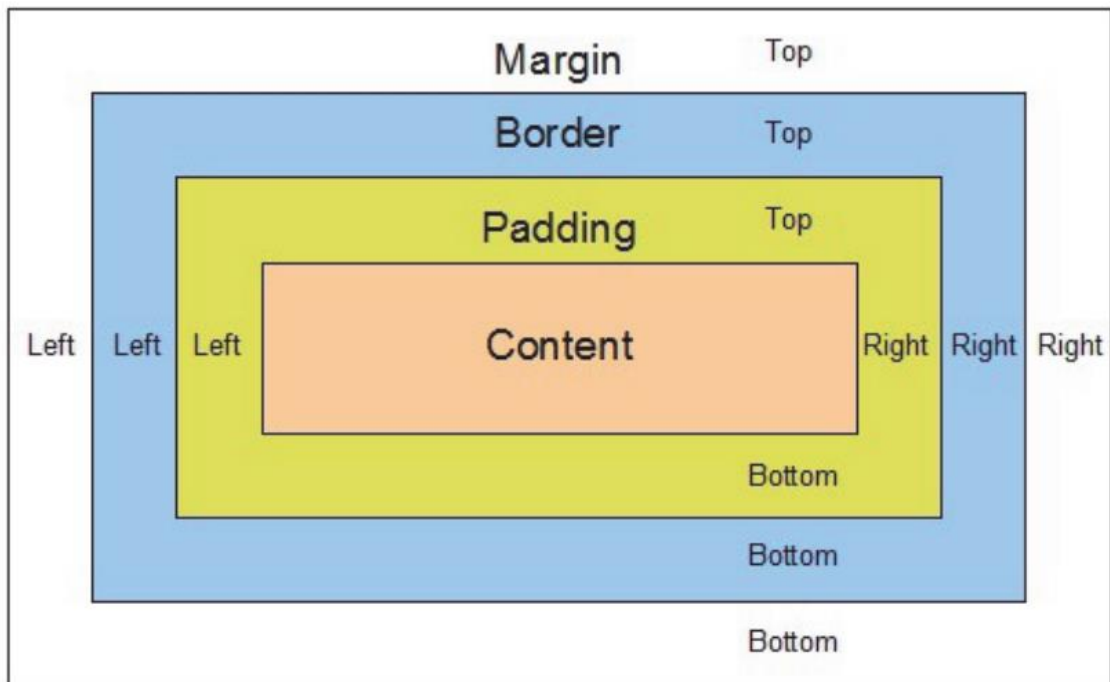

Рис. 2.7. Отступы элементов управления

Для установки внешних остступов 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. 2.7):

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

Для программной установки внутренних отступов вызыватся метод *setPadding*(*left*, *top*, *right*, *bottom*), в который передаются четыре значения для каждой из сторон.

2.3. Виды Layout

2.3.1. LinearLayout

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. 2.8.

```
<?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">
    <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>
```

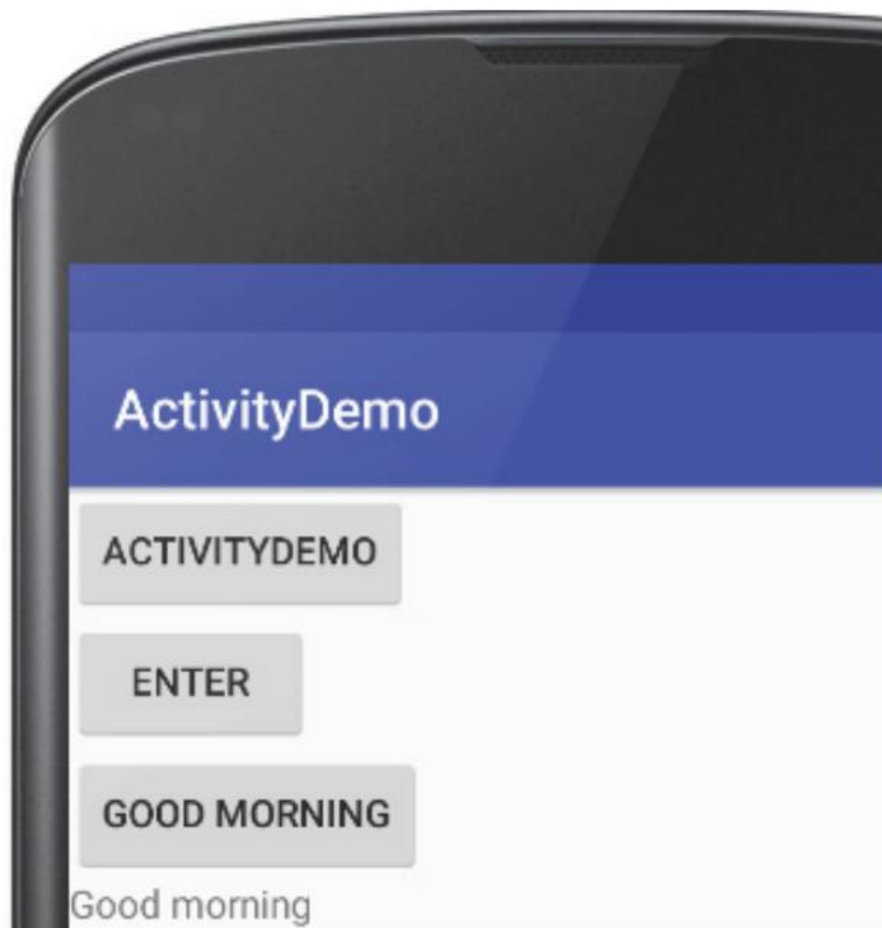

Рис. 2.8. Пример интерфейса с контейнером *LinearLayout*

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 они будут равномерно распределены по всей площади контейнера.

2.3.2. RelativeLayout

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. В настоящее время он считается устаревшим и распологается в папке Legacy.

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* – располагает элемент слева от элемента с указанным ID;
– *android:layout_toRightOf* – располагает элемент 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* элемент выравнивается по центру по горизонтали;
–*android:layout_centerVertical* – при значении *true* элемент выравнивается по центру по вертикали.
Пример отображения разметки, представленной ниже, приведен на рис. 2.9.

```
<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>
```

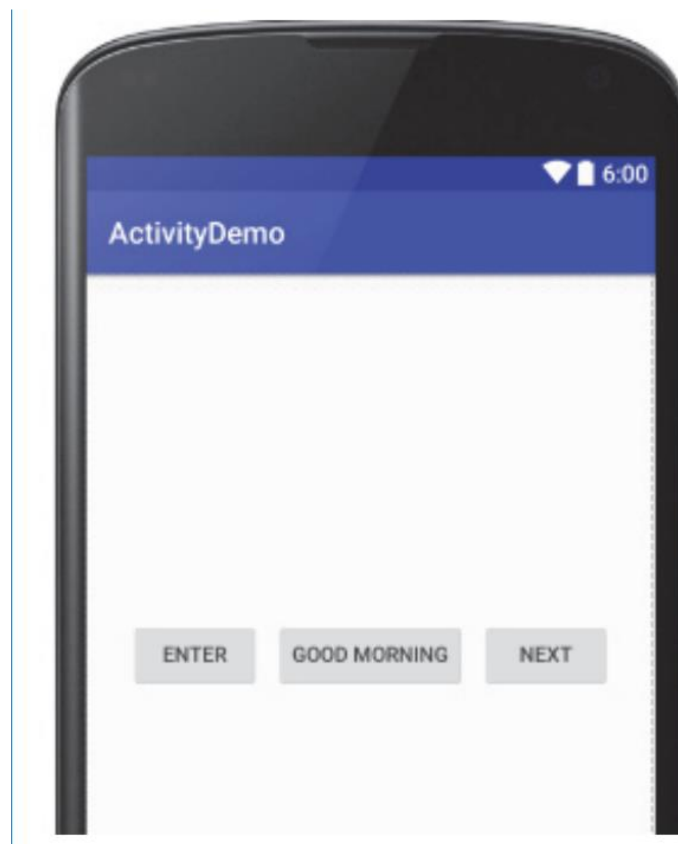

Рис. 2.9. Пример интерфейса с контейнером *RelativeLayout*

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* – элемент размещается вверху;
– *bottom* – элемент размещается внизу;
– *left* – элемент размещается в левой стороне;
– *right* – элемент размещается в правой стороне контейнера;
– *center_vertical* – выравнивает элементы по центру по вертикали;
– *center_horizontal* – выравнивает элементы по центру по горизонтали;
– *center* – элемент размещается 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 границы элементов; – *clip_horizontal* – обрезает правую и левую границы элементов;
– *start* – элемент позиционируется в начале контейнера (в верхнем левом углу);
– *end* – элемент позиционируется в конце контейнера (в верхнем правом углу).

2.3.3. TableLayout

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 три виджета, то, соответственно, столбцов было бы также три, даже если в другой строке осталось бы два виджета. Например, определены три строки, в каждой из которых изменяется два или три элемента (рис. 2.10):

```
<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>
```

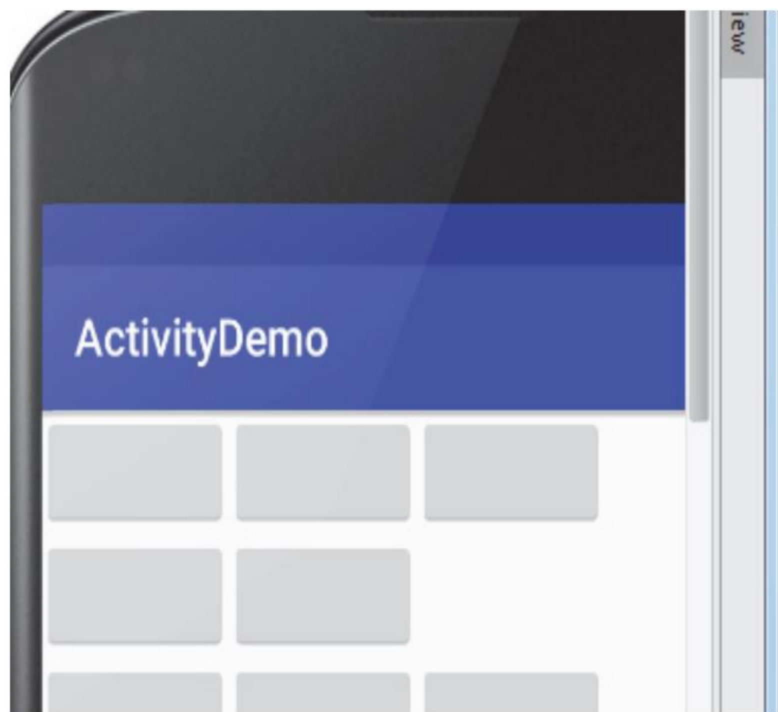

Рис. 2.10. Пример интерфейса с контейнером *TableLayout*

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".

2.3.4. FrameLayout

Контейнер *FrameLayout* предназначен для вывода на экран одного помещенного в него визуального элемента. Если поместить несколько элементов, то они будут накладываться друг на друга (рис. 2.11):

```
<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" />
```

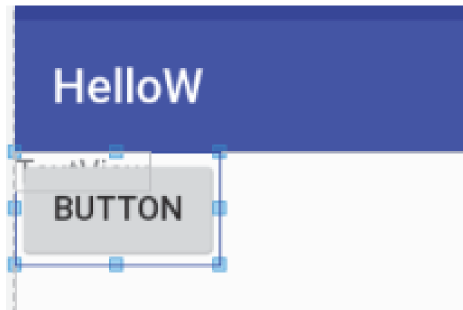

Рис. 2.11. Пример интерфейса с контейнером *FrameLayout*

Элементы управления, которые помещаются в контейнер *FrameLayout*, могут установить свое позиционирование с помощью атрибута *android*:*layout_gravity* (рис. 2.12).

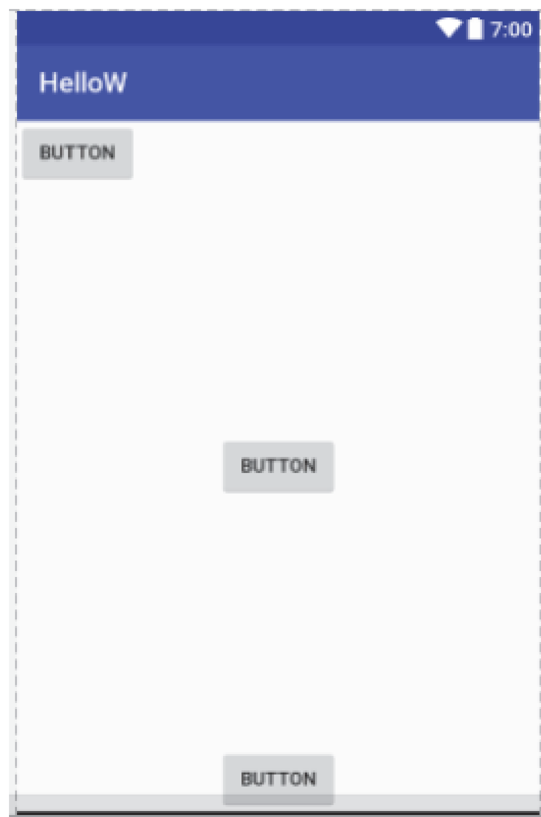

Рис. 2.12. Пример интерфейса с контейнером *FrameLayout* и атрибутом *android*:*layout_gravity*

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>
```

2.3.5. GridLayout

GridLayout – это контейнер, который позволяет создавать табличные представления. В настоящее время он считается устаревшим и распологается в папке Legacy.

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 слудеющем 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. 2.13):

```
<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. 2.13. 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.

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 позиционирование, 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. 2.14). Эти круги используются, чтобы создавать привязки.

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

Чтобы создать горизонтальную привязку, нужно привязать 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, следует нажать мышкой на *Button* и выделить ее. Затем зажать левой кнопкой мыши левый кружок и тащить его к левой границе (рис. 2.14). *Button* также сдвигается влево. Она привязывается к левой границе своего родителя.

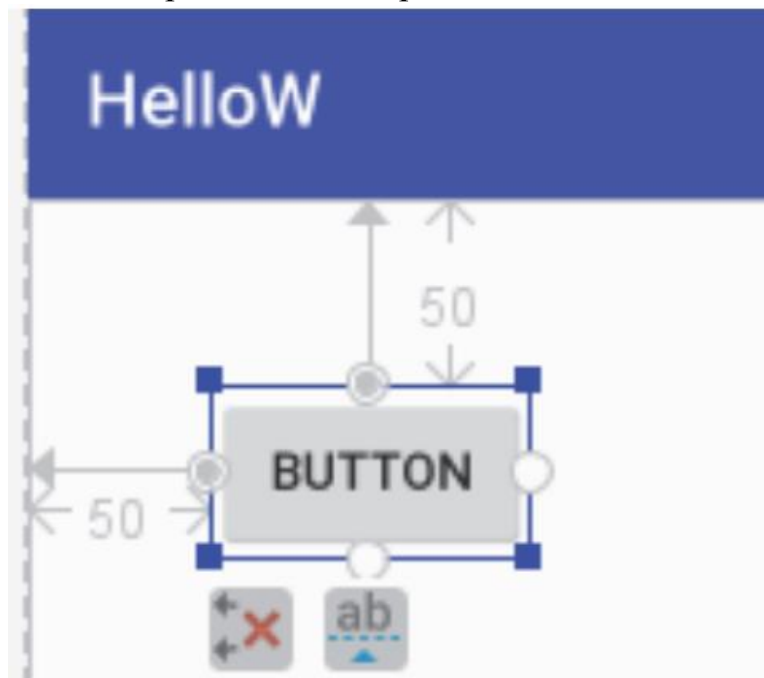

Рис. 2.14. Установка привязок к границам контейнера

Чтобы закрепить *Button* и по вертикали, можно создать вертикальную привязку. Теперь *View* привязан и по горизонтали, и по вертикали (рис. 2.14). То есть 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 его перемещении, кнопка также будет перемещаться (рис. 2.15).

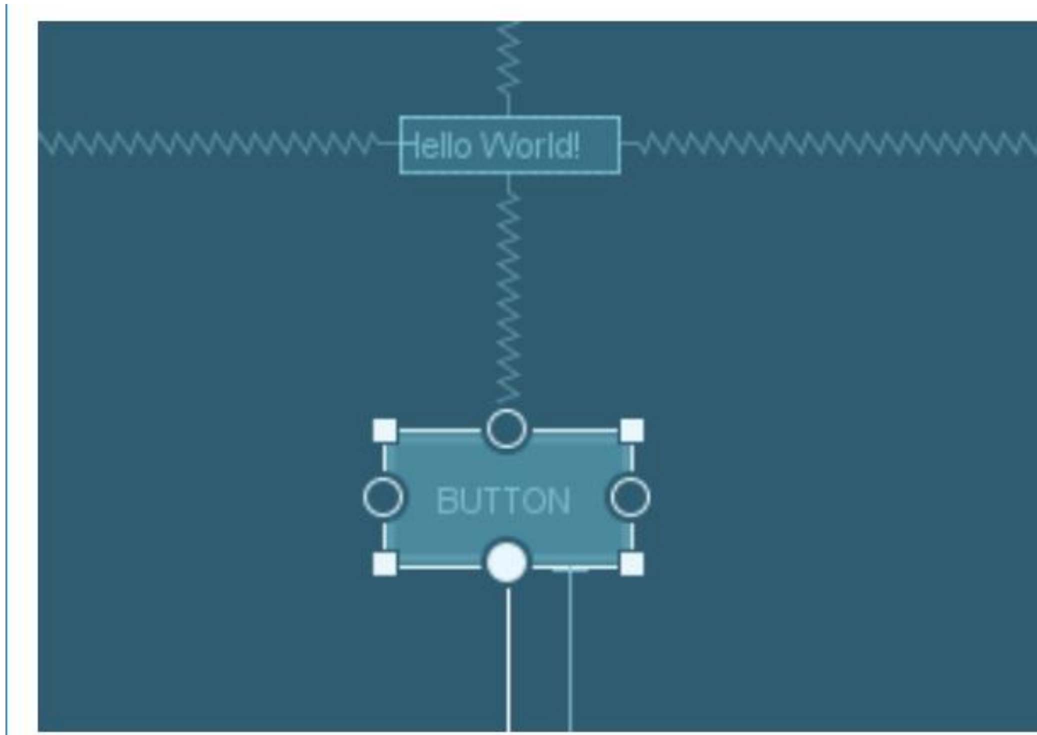

Рис. 2.15. Установка привязок к *View*

Чтобы удалить привязку, надо просто нажать на нее и выбрать *Clear* (рис. 2.16).

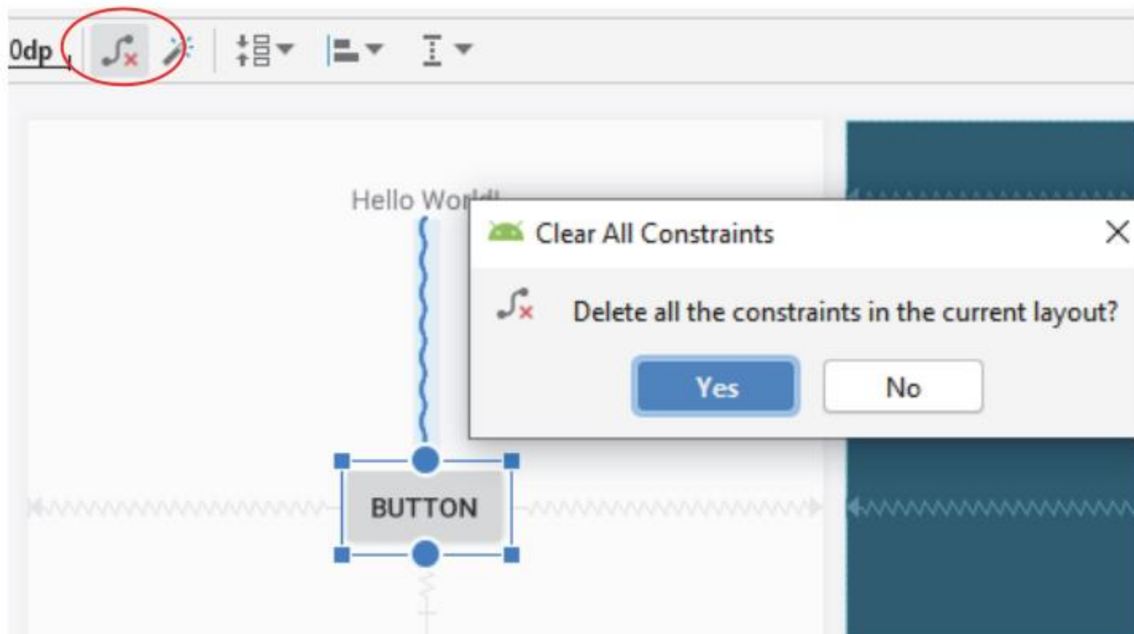

Рис. 2.16. Удаление привязки

Привяжем кнопку к левой и правой границам родителя. *Button* сначала уйдет влево, так как была сделана привязка к левой границе, но после создания привязки к правой границе она выровняется и теперь будет располагаться по центру (рис. 2.17). То есть привязки уравнивают друг друга, и *View* будет

находиться ровно посередине между тем, к чему он привязан слева, и тем, к чему он привязан справа. Следует обратить внимание, что такие двусторонние привязки отображаются как пружинки, а не линии.

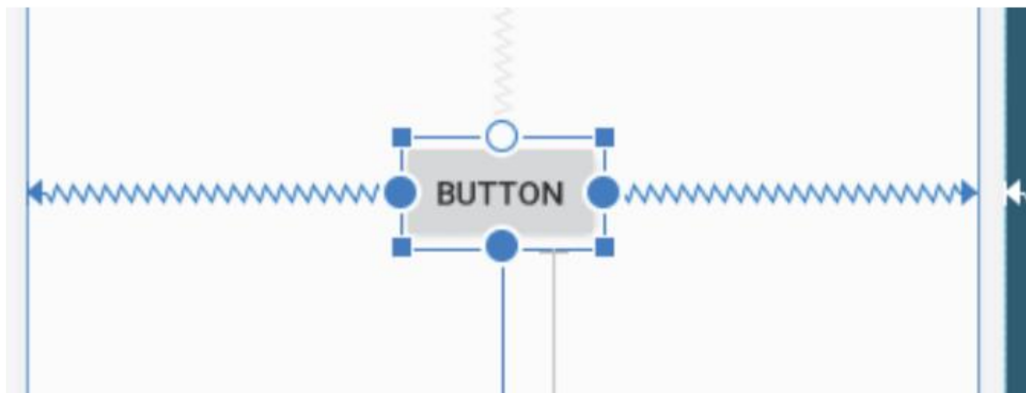

Рис. 2.17. Привязка к левой и правой границам

При создании ограничений необходимо придерживаться следующих правил:

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

– можно 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 создавать несколько ограничений (с различных точек зрения) к одной и той же точке привязки.

На панели есть несколько инструментов, которые могут помочь в работе (рис. 2.18) (перечислены слева направо).

Рис. 2.18. 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 – при нажатии 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. 2.19).

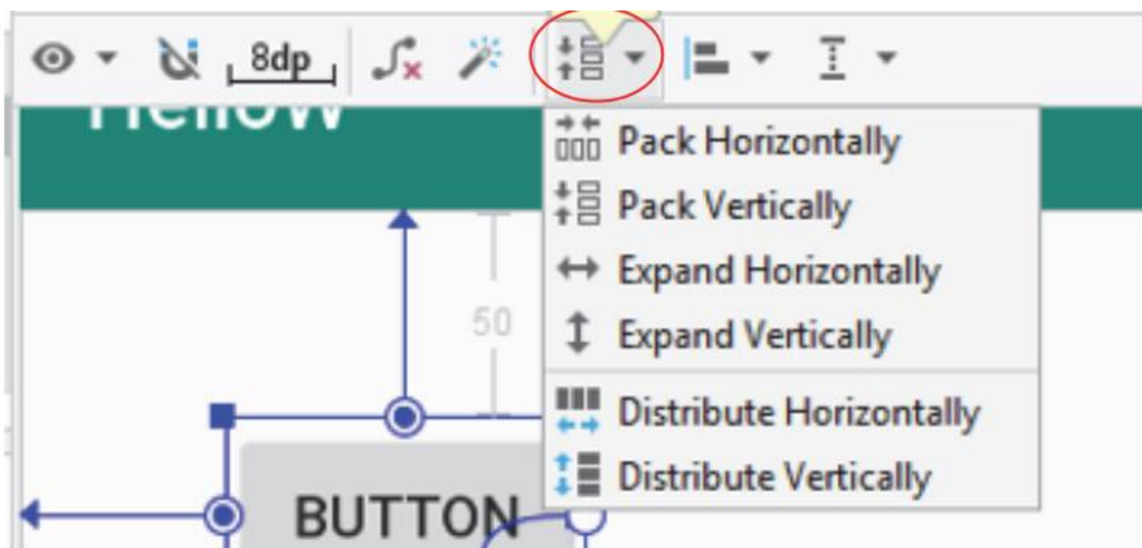

Рис. 2.19. Элемент *Собрать/Растянуть* на панели инструментов

Выравнивание – по горизонтали: по левому краю, по центру, по правому краю. Нижний ряд кнопок – это центрирование. Оно создает двустороннюю привязку (рис. 2.20).

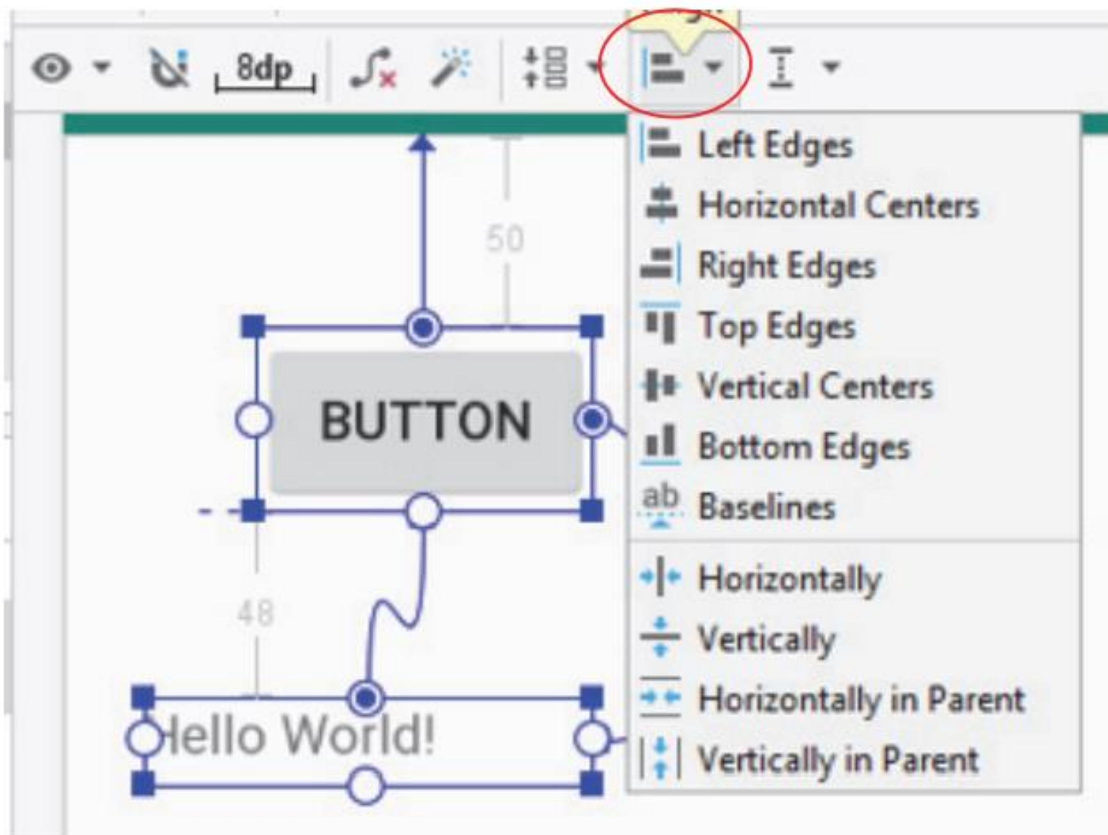

Рис. 2.20. Элемент *Выравнивание* на панели инструментов

Направляющие – это линии, которые можно использовать для создания привязок (рис. 2.21).

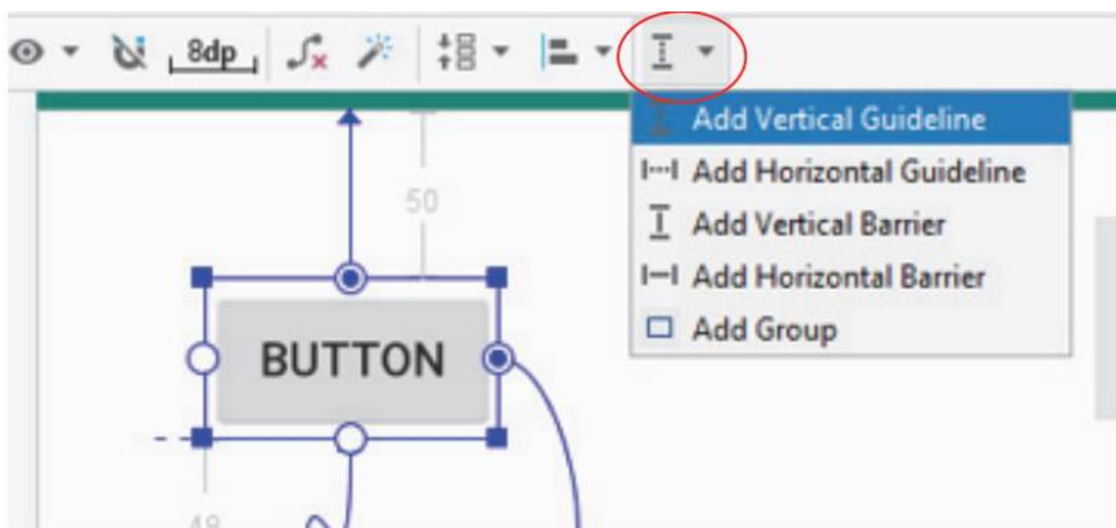

Рис. 2.21. Элемент *Направляющие* на панели инструментов

Если у *View* есть двусторонняя вертикальная привязка и значение высоты установлено в *match_constraints* (0 dp), то *View* растянется по высоте между

объектами привязки. При использовании *aspect ratio* можно настроить элемент так, чтобы высота не растягивалась, а зависила от ширины *View*. Для *View* с двусторонней вертикальной привязкой надо выставить значение высоты в 0 dp. При этом *View* растягивается по высоте. Затем необходимо включить режим соотношения сторон, нажав на треугольник (рис. 2.22). Задать соотношение ширины к высоте, например 3 : 1. То есть высота теперь будет в три раза меньше ширины. С изменением ширины меняется и высота, чтобы соблюдалось установленное соотношение сторон 3 : 1 (рис. 2.22).

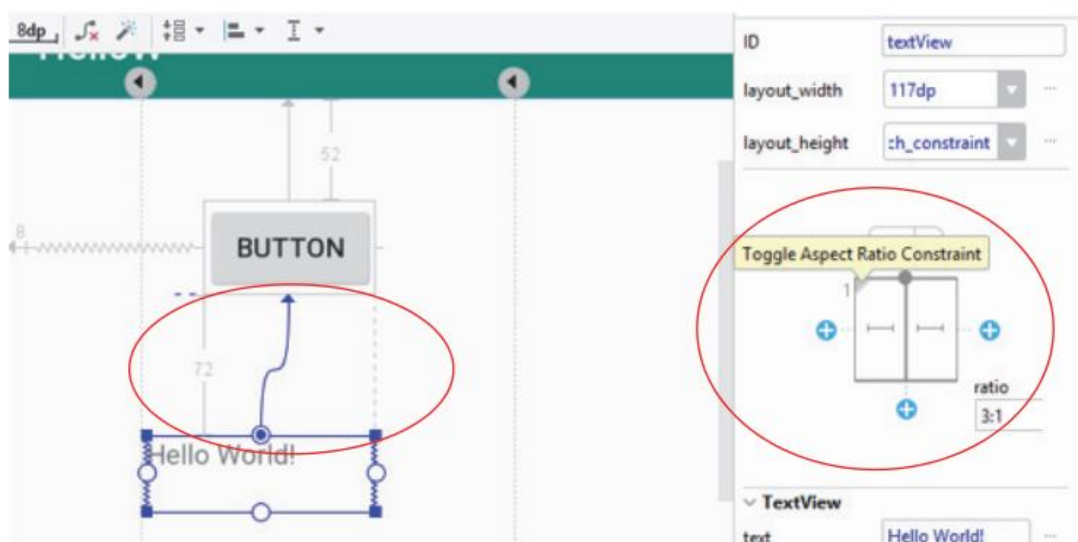

Рис. 2.22. Установка режима соотношения сторон

Цепочка позволит равномерно распределить несколько *View* в имеющемся свободном пространстве. Чтобы создать цепочку, необходимо выделить *View* и центрировать их по горизонтали или вертикали
(рис. 2.23).

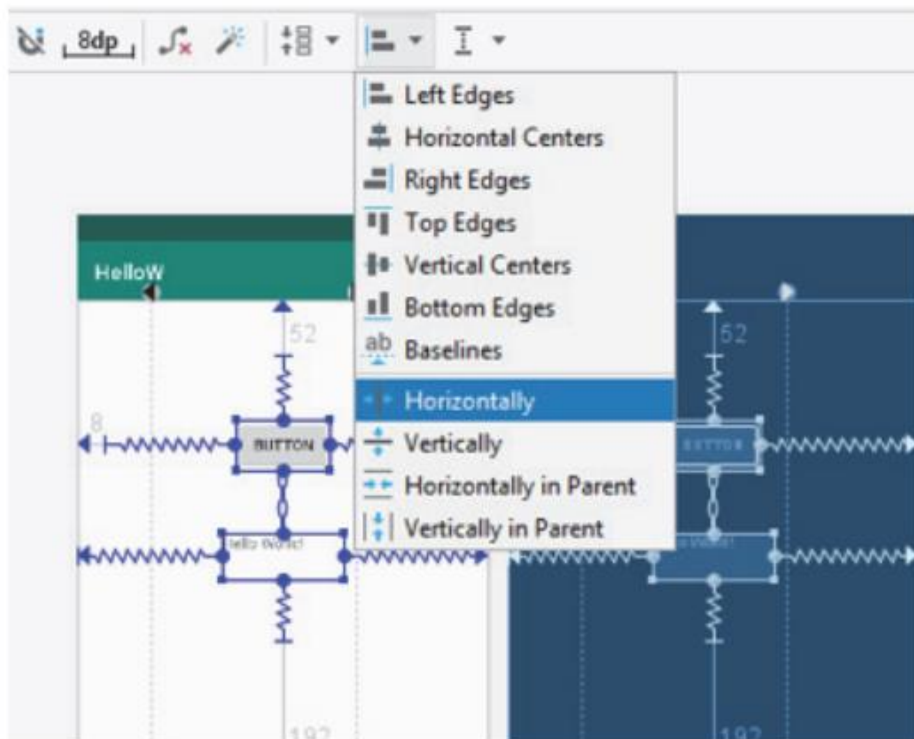

Рис. 2.23. Выбор режима выравнивания

Цепочка может быть в одном из трех режимов.

1. *Spread*: свободное пространство равномерно распределяется между *View* и границами родителя.

2. *Spread_inside*: свободное пространство равномерно распределяется только между *View*. Крайние *View* прижимаются к границам родителя.

3. *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. 2.24).

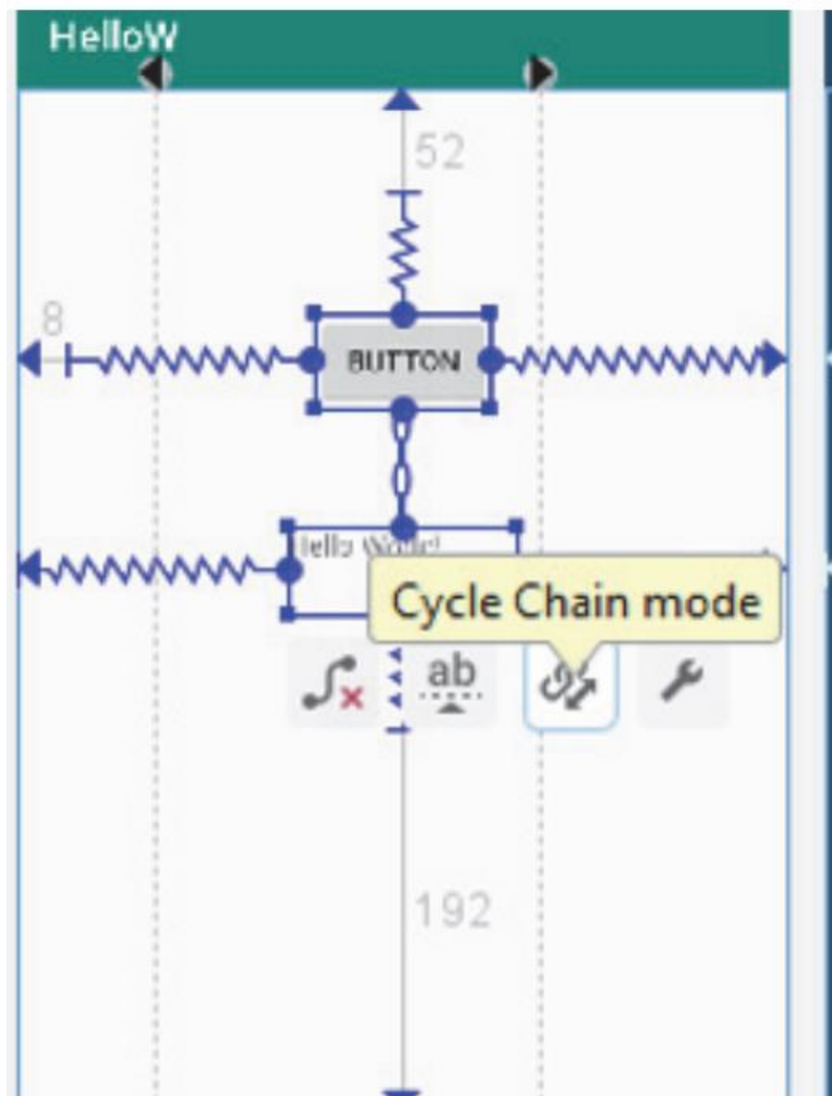

Рис. 2.24. Создание цепочек

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*. Чтобы увидеть все атрибуты, необходимо нажать на значок с двумя стрелками (рис. 2.25).

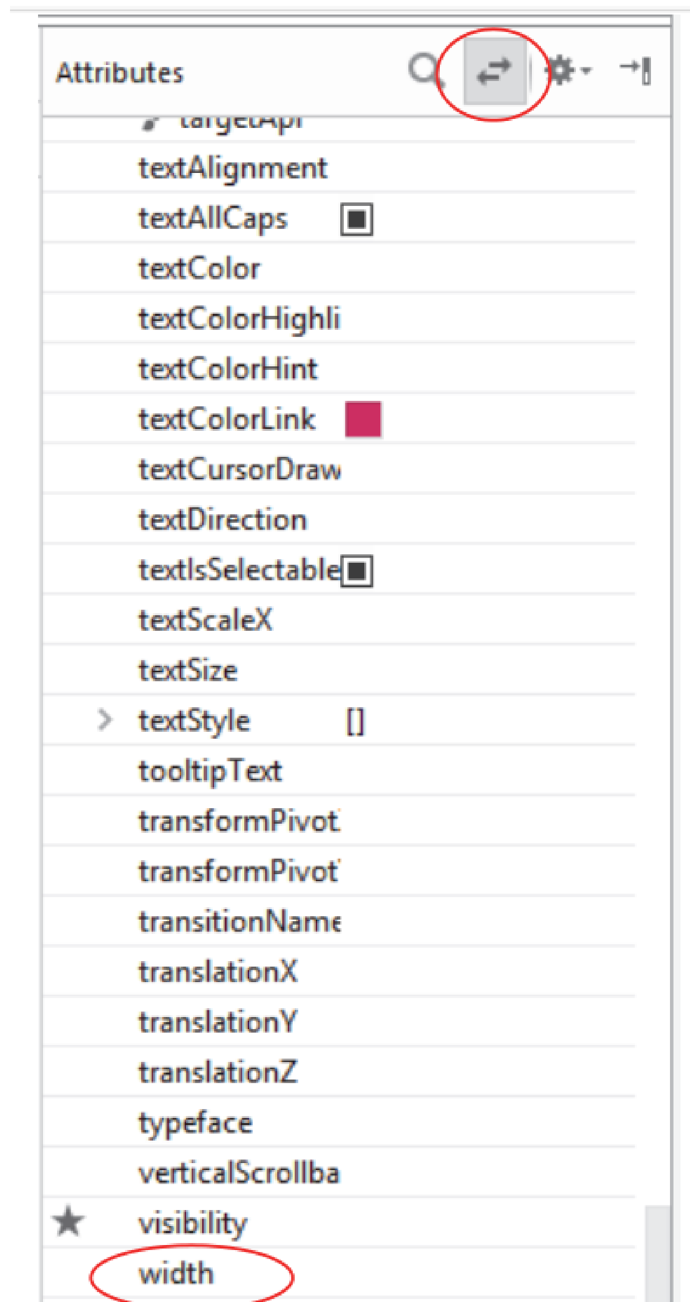

Рис. 2.25. Окно управления атрибутами

Для оптимизации производительности UI необходимо использовать как можно меньшее количество разных *ViewGroup*. Для относительного расположения группы элементов можно выбрать барьеры (рис. 2.26).

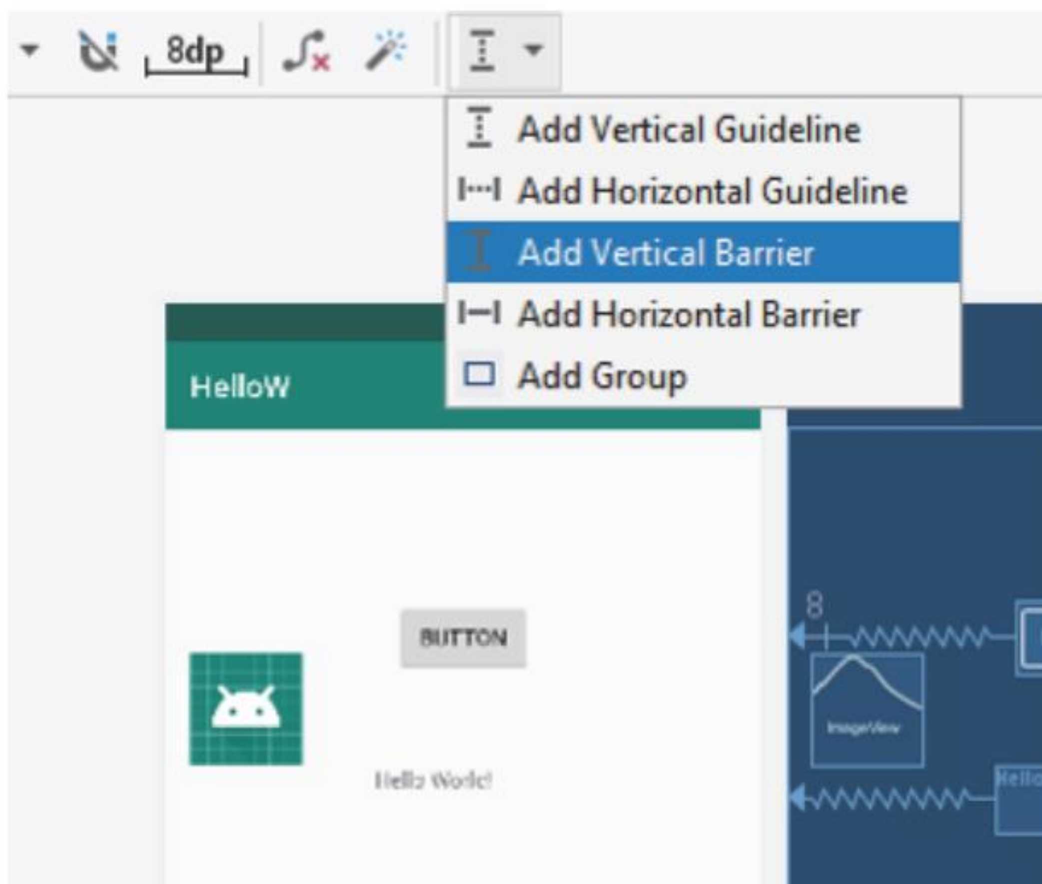

Рис. 2.26. Добавление барьеров

2.3.7. ScrollView

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

2.4. Обращение к View

Чтобы обратиться к элементу *View* 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 класс *R.java*, если он там еще не существует (рис. 2.27). Идентификатор – целое уникальное число.

Рис. 2.27. Индентификация *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("ISIT");
}
```

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

2.5. Ресурсы

2.5.1. 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. 2.28). Для различных типов ресурсов, определенных в проекте, в *res* создаются подпапки (представлены на рис. 2.28).

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.

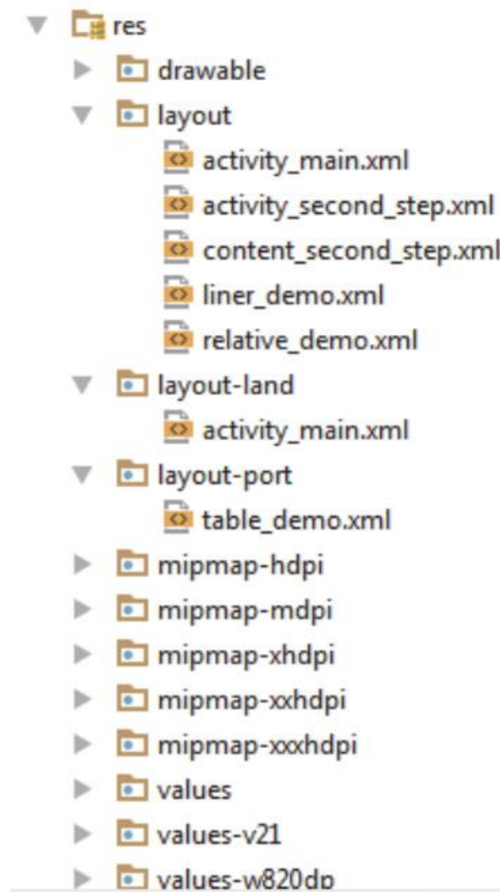

Рис. 2.28. Ресурсы приложения

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/layoutland/*. 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 конфигура- циями. Чтобы

указать, что группа ресурсов предназначена для определенной конфигурации, добавьте соответствующий квалификатор к имени каталога.

2.5.2. Предоставление альтернативных ресурсов и квалификаторы конфигурации

Квалификатор *hdpi* (см. рис. 2.28) 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 его конфигурации 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 пикселях, 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.

2.5.3. Строковые ресурсы

По умолчанию для ресурсов строк (хранятся в виде пар «имя – значение строки») применяется файл *strings.xml*, но разработчики могут добавлять дополнительные файлы ресурсов в каталог проекта *res/values*. При этом

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>
```

2.5.4. Ресурсы plurals

Plurals предназначены 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*, указывающий, когда данная 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 толко в коде Java.

2.5.5. 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_horizontal_margin); int
topPadding = (int)getResources()
                   .getDimension(R.dimen.activity_vertical_margin);
```

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

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

2.5.6. 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:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color> </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*(), где в качестве первого параметра принимается текущий объект *activity*, а вторым параметром является идентификатор цветового ресурса:

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

2.5.7. Доступ к ресурсам

Когда происходит компиляция проекта сведения обо всех ресурсах добавляются *R.java*, который можно найти в проекте по пути *app/build/generated/not_namespaced_r_class_sources/debug/r/[пакет_приложения]* (рис. 2.29).
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 типа *android.content.res.Resources*. Чтобы получить ресурс, надо у объекта *Resources* вызвать один из методов:
– *getString*() – возвращает строку из файла *strings.xml* по числовому идентификатору;
– *getDimension*() – возвращает числовое значение из ресурса *dimen*; – *getDrawable*() – возвращает графический файл;
– *getBoolean*() – возвращает значение *boolean* и т. д.

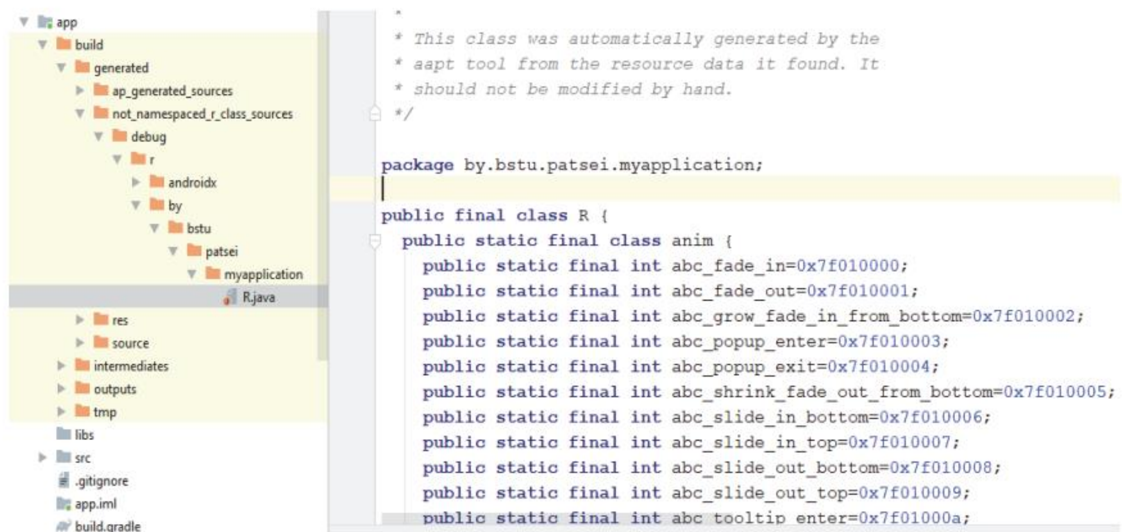

Рис. 2.29. Файл *R.java*

2.5.8. Добавление layout-файла 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. 2.30).

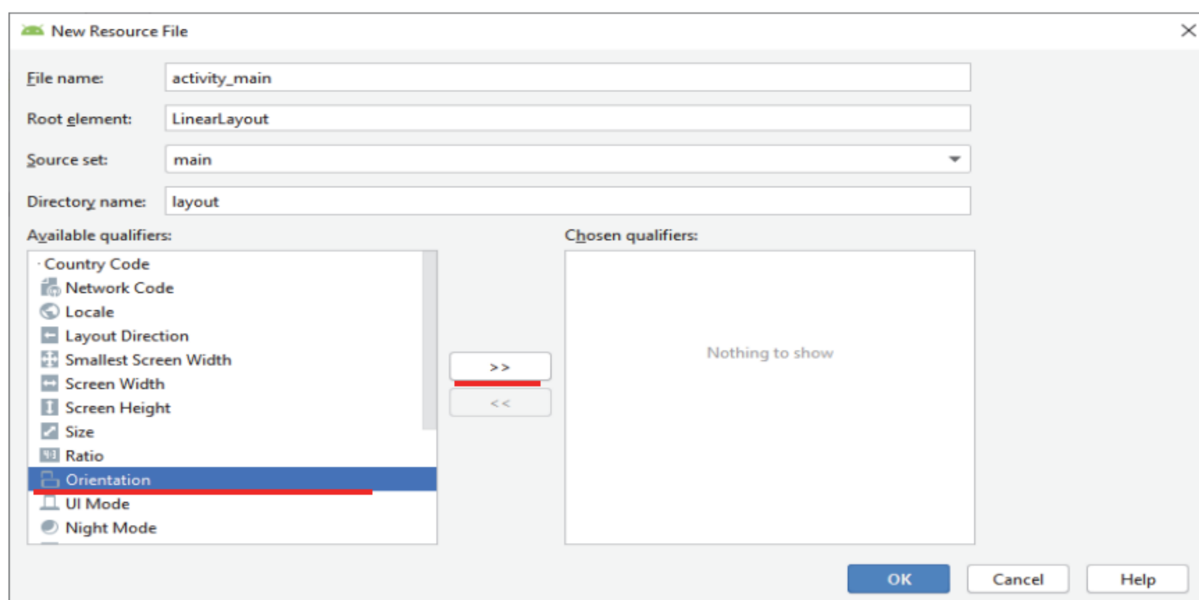

Рис. 2.30. Окно добавления *layout*-файла

Обратите внимание, значение поля *Directory name* изменилось (рис. 2.31). Новый *layout*-файл будет создан в папке *res/layout-land*, а не *res/layout*, как обычно.

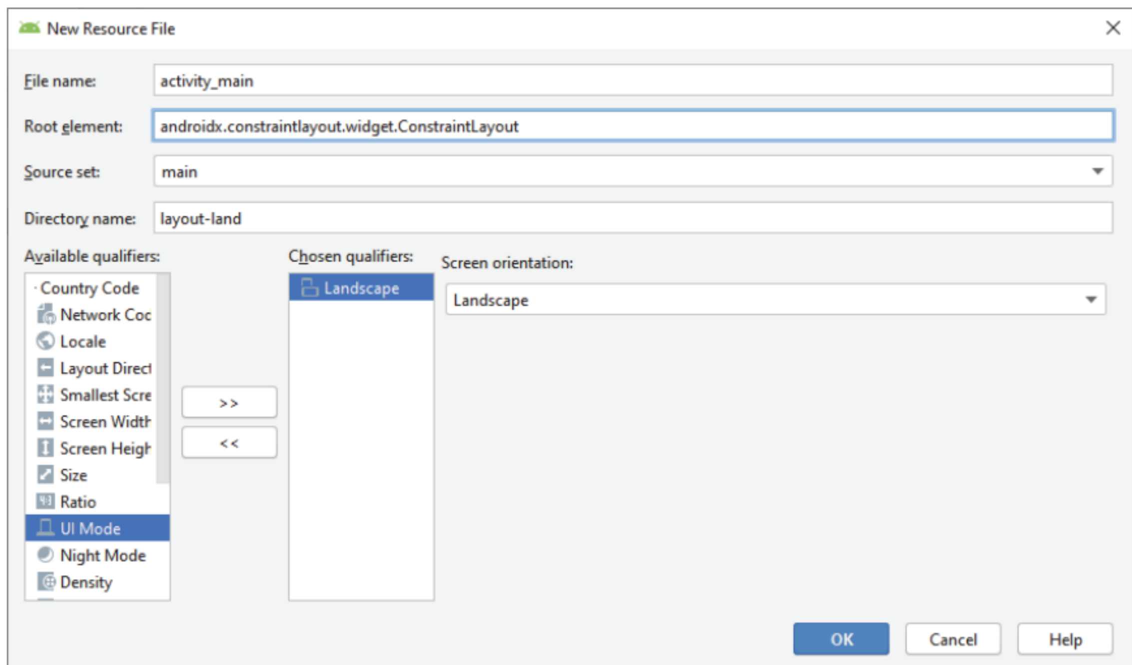

Рис. 2.31. Добавление *layout*-файла для горизонтальной ориентации экрана

Посмотрим на структуру папок проекта (рис. 2.32). Теперь там два файла *activity_main*: обычный и *land*. Каждый из файлов можно менять независимо и соответственно получать разное представление.

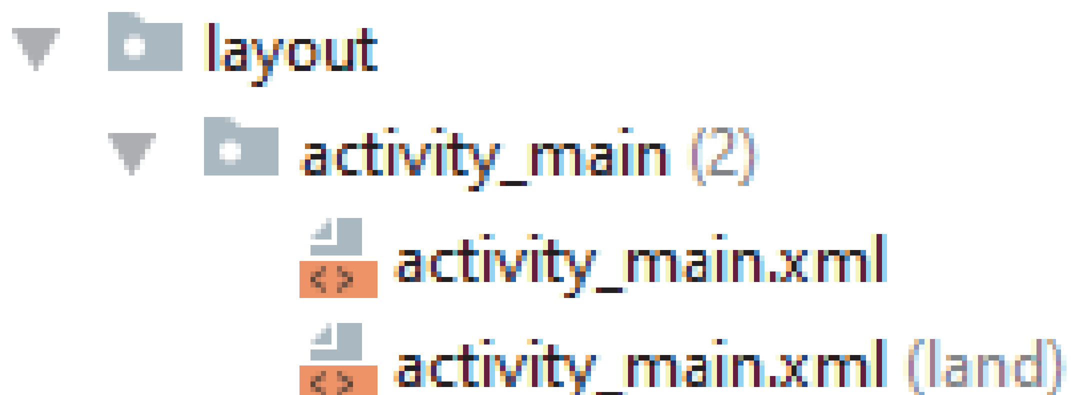

Рис. 2.32. Структура папок проекта

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

Можно управлять сменой шаблона ориентации вручную, если воспользоваться методами, предоставляемыми классом *Configuration*. Также

можно запретить смену ориентации экрана для своей активности с помощью атрибута *android:screenOrientation*. Если ограничить приложение одной ориентацией, то шаблон должен располагаться в папке *res/layout*.

2.6. Обработка нажатий View

Пусть есть следующий макет:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.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=".MainActivity">
    <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" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

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

```
import android.os.Bundle; import
android.widget.Button; import
android.widget.TextView;
 public class MainActivity extends AppCompatActivity {

    TextView txtView;
    Button btnBy;

    @Override     protected void onCreate(Bundle
savedInstanceState) { ...
```

В методе *onCreate* эти объекты заполняются методом *findViewById*. В методе *onCreate* класса *MainActivity* надо написать следующий код:

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

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

```
import android.view.View;

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

Можно установить обработчик по-другому:

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

Откомпилируем код. Если произошла ошибка при запуске кода, то можно перейти на вкладку *Logcat* и прочитать информацию об ошибке
(рис. 2.33).

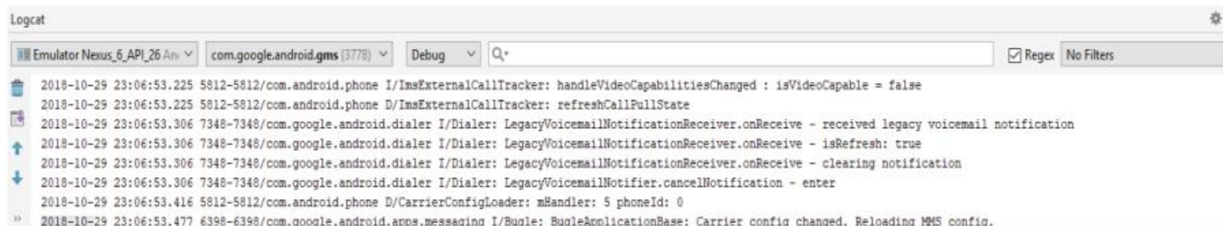

Рис. 2.33. Окно *Logcat*

Если ошибок нет, то после запуска приложения и нажатия на кнопку на экране появится сообщение (рис. 2.34).

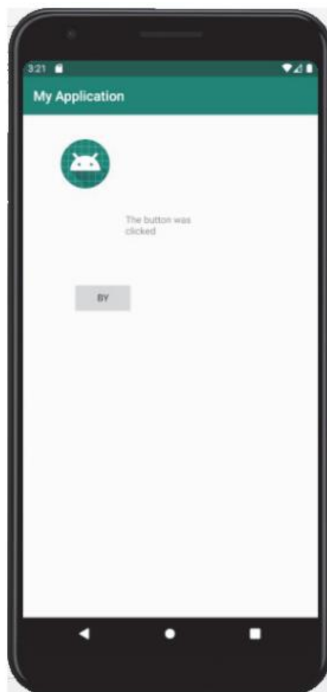

Рис. 2.34. Результат выполнения приложения

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*.

Затем добавляется еще одна кнопка (рис. 2.35), то есть на экране появятся *TextView* с текстом и две кнопки. Необходимо сделать так, чтобы по нажатию кнопки менялось содержимое *TextView*. По нажатию кнопки *BY* нужно ввести текст «By FIT», по нажатию *Hello* – «Hello FIT». Делается это с помощью одного обработчика, который будет обрабатывать нажатия для обеих кнопок.

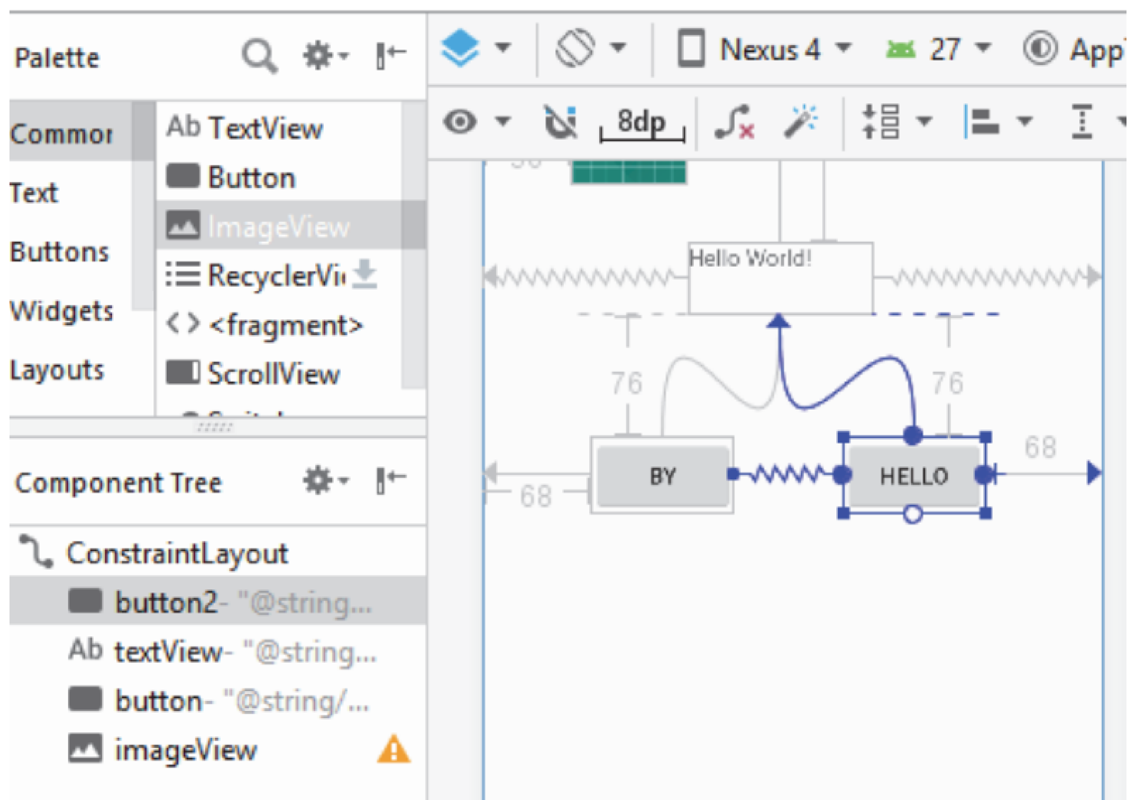

Рис. 2.35. Добавление новой кнопки в активность

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

```
import android.widget.Button; import
android.widget.TextView; import
android.view.View;
 public class MainActivity 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_main);
// Находим View-элементы          txtView = (TextView)
findViewById(R.id.textView);          btnBy = (Button)
findViewById(R.id.button);        btnHello = (Button)
findViewById(R.id.button2);

// Создаем обработчик нажатия
// Присваиваем 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;
                }
            }
        };
        // Назначение обработчика
        btnBy.setOnClickListener(onCLBtn);
        btnHello.setOnClickListener(onCLBtn);
    }
}
```

В примере использовался обработчик с реализацией метода *onClick*. В параметрах ему подается объект класса *View*. Это объект, на котором произошло нажатие и вызов обработчика. В данном случае это будет либо кнопка *BY,* либо *HELLO*. Поэтому надо узнать *ID* элемента *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 обеим кнопкам. Результат представлен на рис. 2.36.

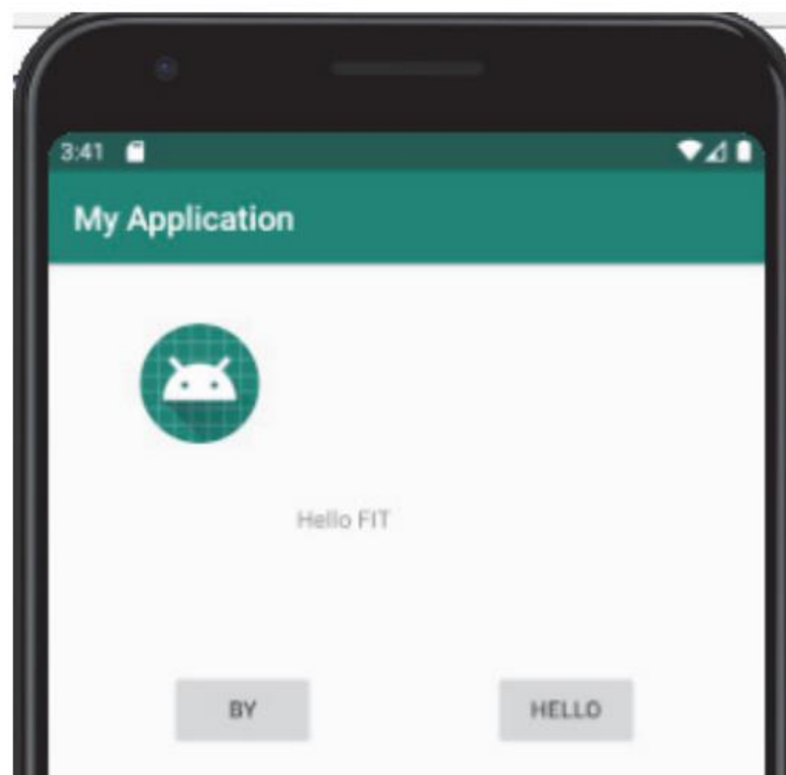

Рис. 2.36. Результат выполнения приложения с двумя кнопками

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

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

Есть еще один способ. В *layout*-файле при определении кнопки используется атрибут *onClick*. В нем указывается имя метода активности. Этот метод срабатывает 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 добавляется этот метод в *activity* (*MainActivity.java*). Требования к методу: модификатор *public*, возвращаемое значение *void* и параметр типа *View*:

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

2.7. 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 – тег и текст сообщения. Тег – это метка, чтобы позволяющая быстро находить нужное сообщение в системных логах.

Затем следует изменить код активности и вывести лог с помощью метода *Log*.d:

```
public class MainActivity extends AppCompatActivity {
 public  final String TAG = "HelloW";

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

После запуска приложения можно изучить сообщения в *Logcat* (рис. 2.37). Когда логов много, их можно отфильтровать по типу сообщения (рис. 2.38, 2.39).

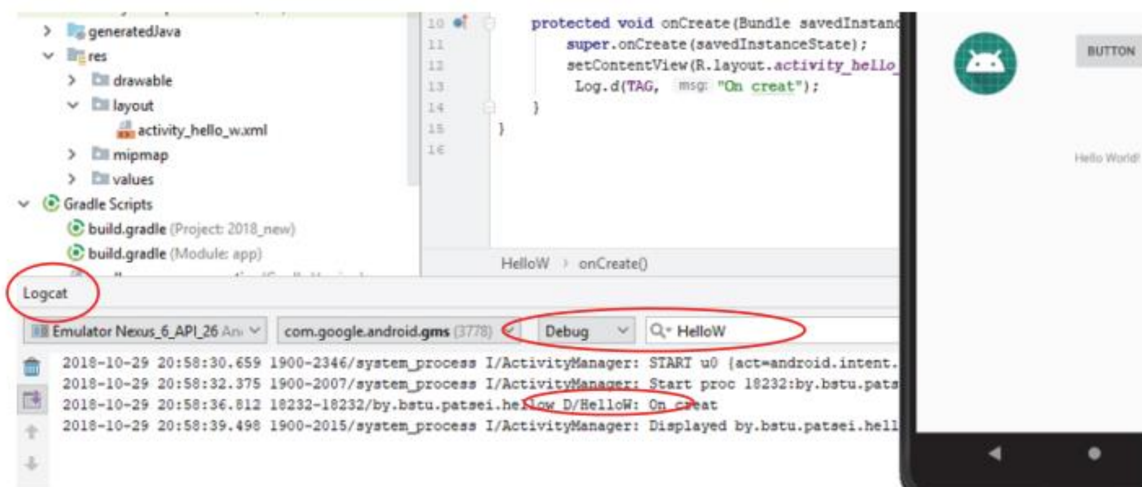

Рис. 2.37. Вывод *Log*-сообщений

Рис. 2.38. Фильтрация *Log*-сообщений

Можно изменить код *HelloW.java* и добавить *Info*-логи:

```
View.OnClickListener onCLBtn = new View.OnClickListener() {
    // Создается обработчик
    @Override     public void
onClick(View v) {
         switch (v.getId()) {
case R.id.button:
// Кнопка BY
txtView.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;
        }
    }
} ;
```

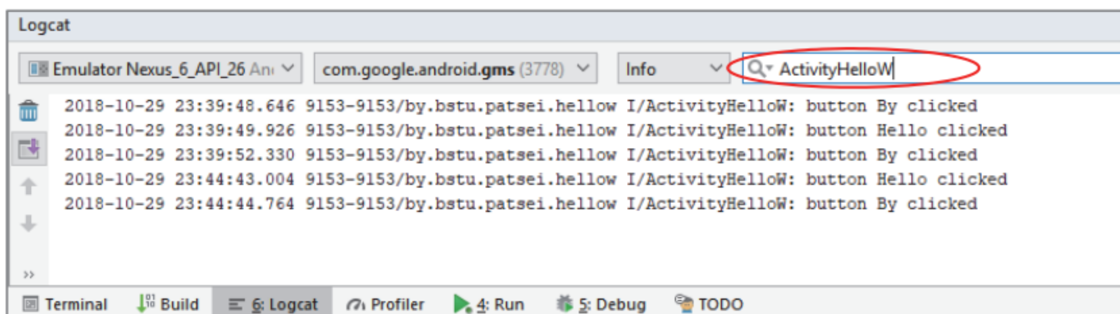

Рис. 2.39. Фильтрация сообщений

2.8. 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 мс, *Toast.LENGTH*_SHORT – 2000 мс):

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

При редактировании метода *onClick* следует сделать так, чтобы всплывало сообщение о том, какая кнопка была нажата:


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

Результат выполнения представлен на рис. 2.40.

Рис. 2.40. Результат выполнения приложения с всплывающими сообщениями

2.9. Элементы управления

2.9.1. 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 шестнадцатеричной 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.

2.9.2. 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*. Он 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* – первый введенный символ слова при вводе 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.
Элемент *EditText* определяется так (результат представлен на рис. 2.41):

```
<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"  />
```

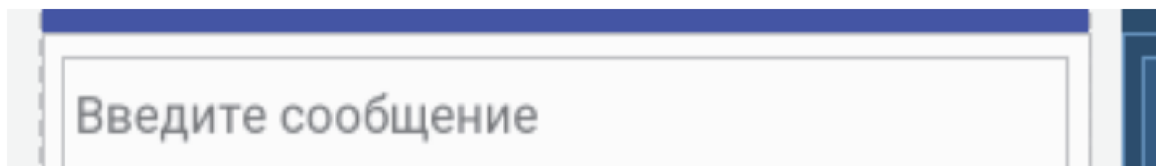

Рис. 2.41. Вывод элемента *EditText*

2.9.3. Button

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

Ниже представлены некоторые ключевые атрибуты, которые можно присвоить кнопкам:

- *text* – задает текст на кнопке;
- *textColor* – задает цвет текста;
- *background* – задает фоновый цвет;
- *textAllCaps* – при значении *true* устанавливает текст в верхнем регистре;
- *onClick* – определяет обработчик нажатия кнопки.

2.9.4. 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.

2.9.5. 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.

2.9.6. ToggleButton

Атрибуты *android*:*textOn* и *android*:*textOff* задают текст кнопки в отмеченном и неотмеченном состоянии соответственно. И так же, 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"/>
```