



Редактируем макет: устанавливаем для текста размер 70sp, а для изображения — ширину `match_parent` и подбираем высоту. Центрируем объекты по горизонтали и вертикали. Готово. Запускаем приложение — теперь на 7-дюймовом экране планшета контент смотрится так же, как и на смартфоне.

Теперь наш интерфейс может считаться действительно адаптивным. В будущем нужно стараться делать адаптивными интерфейсы всех наших приложений. Но, как несложно догадаться, копировать ва-

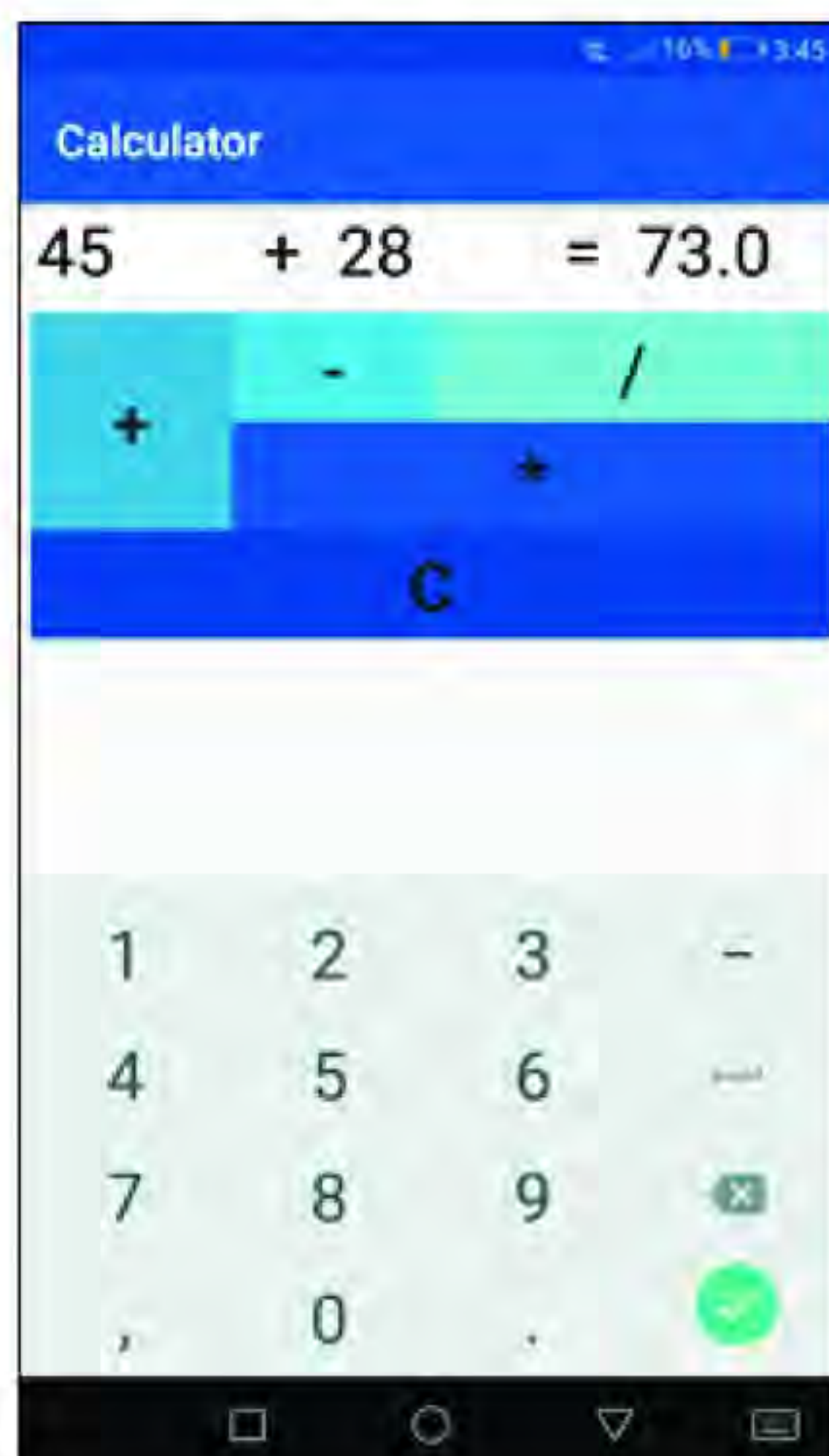
риации экрана нужно в последнюю очередь. Так в них придется вносить наименьшее количество изменений.

Задания

1. Внести аналогичные изменения для `sw720dp`, чтобы контент корректно отображался на 10-дюймовых экранах.
2. Разрешить приложению снова отображаться во всех вариантах ориентации — удалить соответствующую строку из файла манифеста.

2.4. Приложение «Калькулятор»

Для того чтобы обобщить и закрепить все изученное в этой главе, создадим полноценное работающее приложение — калькулятор.



Концепция приложения: простой непрограммируемый калькулятор. Два поля для ввода чисел и одно для вывода результата. Панель с кнопками для проведения различных арифметических операций.

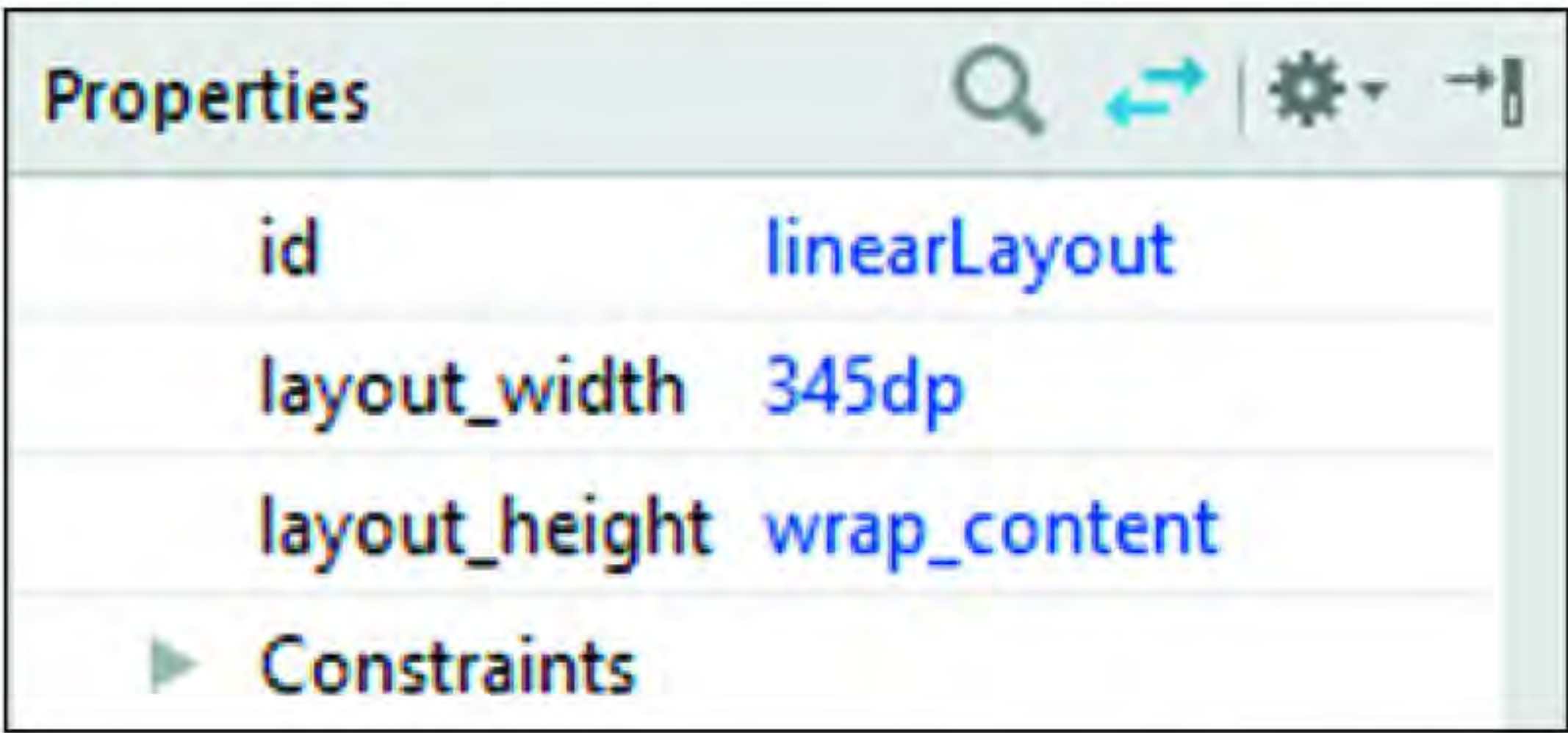
При нажатии на поля ввода появляется числовая клавиатура. При выборе операции автоматически происходит расчет. Кнопка «С» очищает все поля.

Выполнив все описанные ниже шаги, мы получим настоящий «готовый продукт» — собственное приложение, которым смело можно поделиться.

Шаг 1. Создать новый Android Studio проект с именем Calculator и одной Empty Activity.

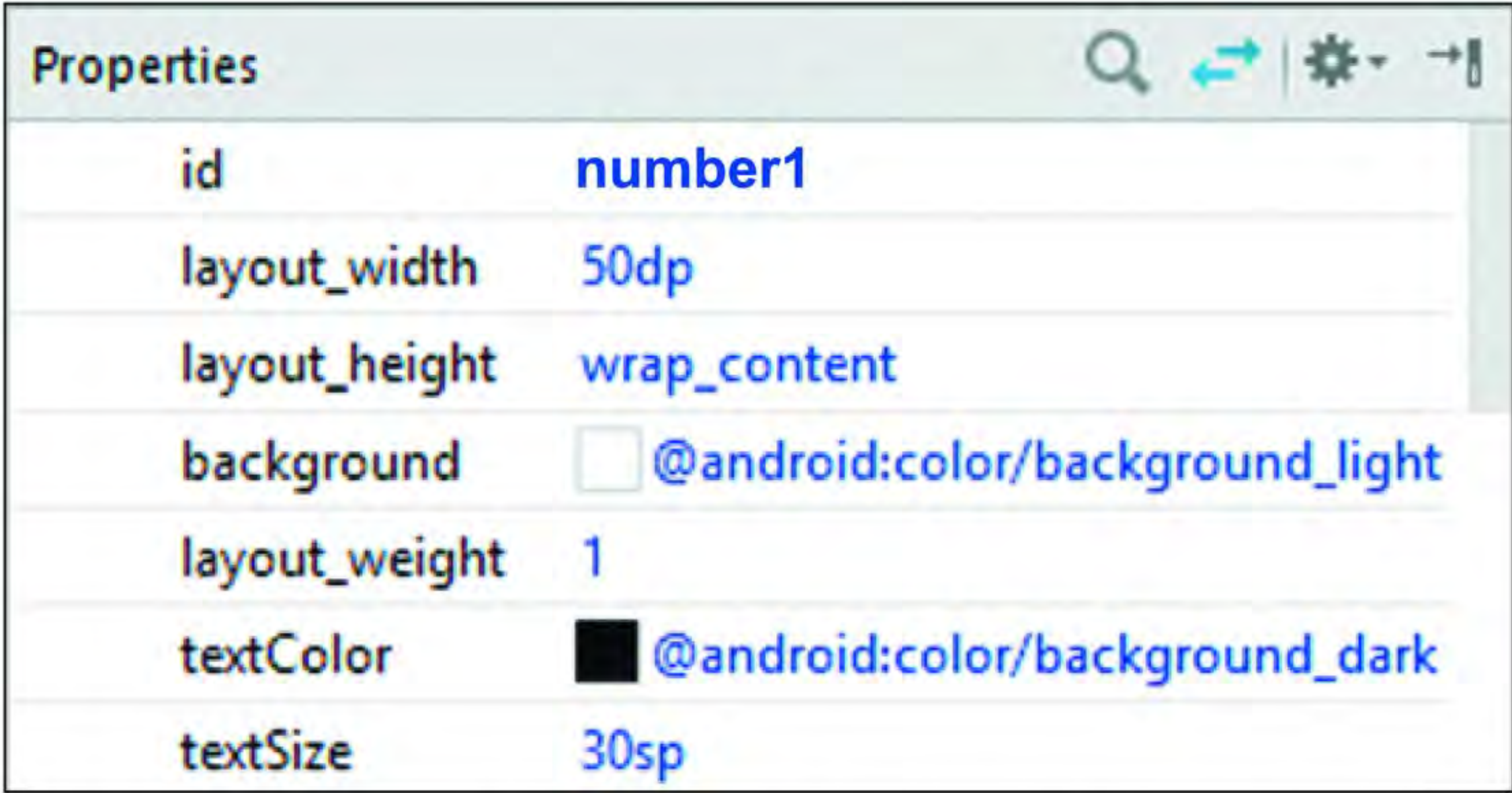
Шаг 2. Реализовать область вычислений.

Для этого добавить на экран горизонтальный **LinearLayout**. С помощью панели свойств задать ему идентификатор `linearLayout`, ширину `345dp` и высоту `wrap_content`:

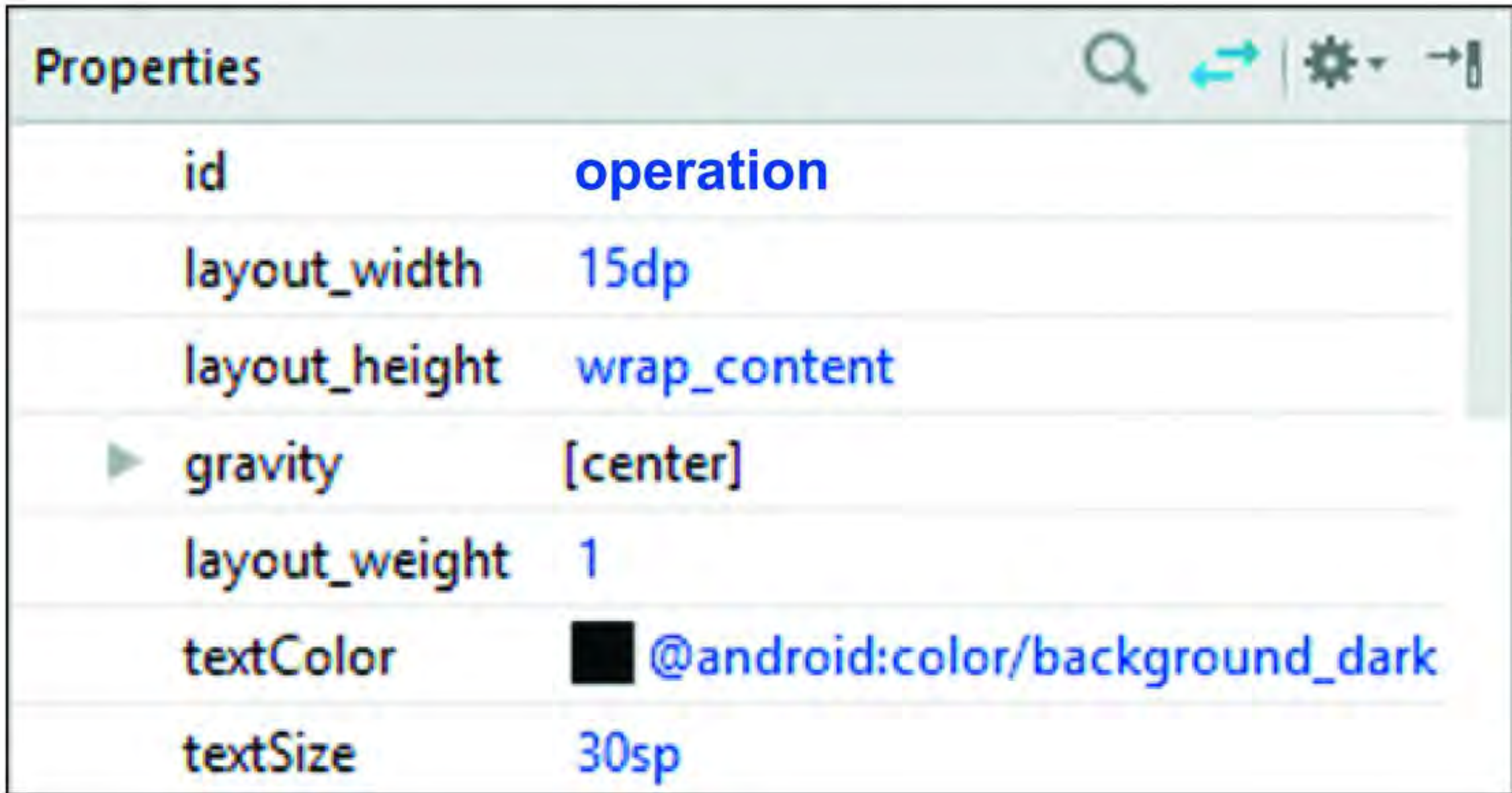


Шаг 3. В добавленном макете **LinearLayout** разместить пять элементов **TextView**:

1) **number1** (*число 1*) — для ввода первого числа:



2) **operation** (*операция*) — для назначения операции (умножение, деление, сложение, вычитание):



3) **number2** (*число 2*) — для ввода второго числа:

Properties	
id	number2
layout_width	50dp
layout_height	wrap_content
background	<input type="checkbox"/> @android:color/background_light
layout_weight	1
textColor	<input checked="" type="checkbox"/> @android:color/background_dark
textSize	30sp

4) **equal** (*равно*) — для вывода символа «=» перед результатом:

Properties	
id	equal
layout_width	15dp
layout_height	wrap_content
gravity	[center]
layout_weight	1
text	=
textColor	<input checked="" type="checkbox"/> @android:color/background_dark
textSize	30sp

5) **result** (*результат*) — для вывода результатов вычислений:

Properties	
id	result
layout_width	50dp
layout_height	wrap_content
layout_weight	1
textColor	<input checked="" type="checkbox"/> @android:color/background_dark
textSize	30sp

Шаг 4. Изменить элемент **TextView** для числа 1 и числа 2 таким образом, чтобы в них можно было вводить текст.

Перейти в режим кода. Найти элементы и изменить их названия с **TextView** на **EditText**. В конце описания элементов доба-

вить атрибут **inputType** (*тип ввода*) со значением **numberSigned** (строка № 26).

```

18 <EditText
19     android:id="@+id/number1"
20     android:layout_width="50dp"
21     android:layout_height="wrap_content"
22     android:layout_weight="1"
23     android:background="@android:color/background_light"
24     android:textColor="@android:color/background_dark"
25     android:textSize="30sp"
26     android:inputType="numberSigned"/>
27

```

Этот атрибут отвечает за ввод данных в элемент, то есть какая клавиатура будет отображаться при заполнении элемента. Если не указывать этот атрибут, то отобразится клавиатура, выбранная по умолчанию пользователем для всего смартфона, если **numberDecimal** — клавиатура только с цифрами, если же **numberPassword** — клавиатура для ввода числового пароля, которая к тому же заменяет вводимые цифры точками. Мы выбрали **numberSigned** — числовую клавиатуру с возможностью ввода десятичного разделителя (точки или запятой) и минуса, то есть отрицательных и дробных чисел, — то, что нужно для калькулятора.

Шаг 5. Реализовать кнопки для выбора операции.





Добавить на экран макет **GridLayout**, с помощью панели свойств задать ему идентификатор **gridLayout**, ширину **350dp**, высоту **wrap_content** и количество столбцов **3** (свойство **columnCount**):

Properties	
id	gridLayout
layout_width	350dp
layout_height	wrap_content
columnCount	3





Центрировать оба макета (**LinearLayout** и **GridLayout**) по горизонтали.

Шаг 6. Разместить внутри **GridLayout** пять кнопок:





- 1) **buttonAdd** (*кнопка Сложить*) — кнопка «+»:

Properties		   
id	buttonAdd	
layout_width	wrap_content	
layout_height	wrap_content	
text	+	
textSize	30sp	





2) **buttonSubtract** (*кнопка Вычитать*) — кнопка «-»:

Properties		   
id	buttonSubtract	
layout_width	wrap_content	
layout_height	wrap_content	
textSize	30sp	
text	-	

3) **buttonDivide** (*кнопка Разделить*) — кнопка «/»:

Properties		   
id	buttonDivide	
layout_width	wrap_content	
layout_height	wrap_content	
text	/	
textSize	30sp	

4) **buttonMultiply** (*кнопка Умножить*) — кнопка «*»:

Properties		   
id	buttonMultiply	
layout_width	wrap_content	
layout_height	wrap_content	
text	*	
textSize	30sp	

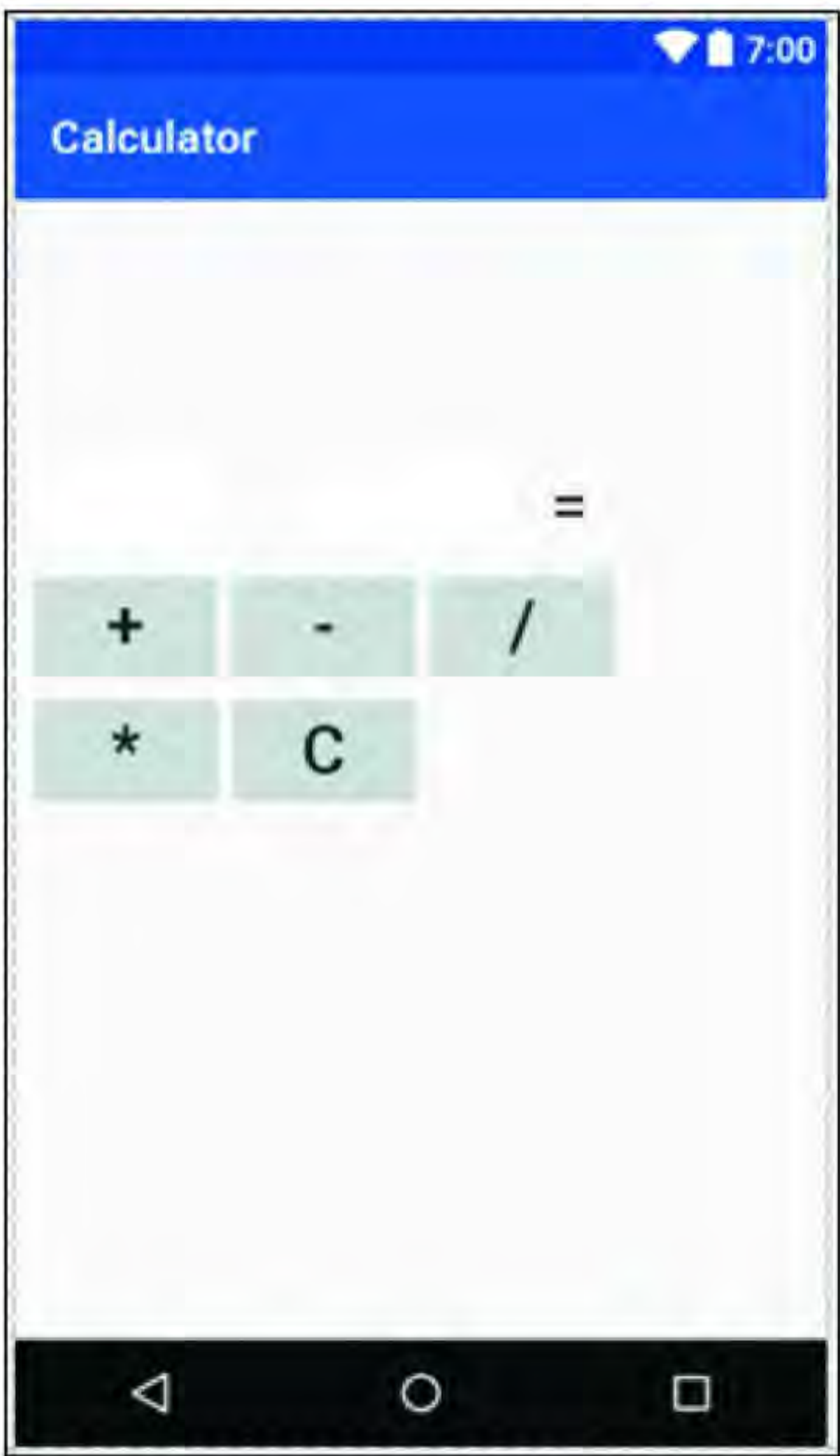
5) **buttonClean** (кнопка *Очистить*) — кнопка «C»:

Properties	
id	buttonClean
layout_width	wrap_content
layout_height	wrap_content
text	C
textSize	30sp

Шаг 7. Изменить размер и расположение кнопок.

Перейдем в режим кода.

Сейчас наши кнопки расположены не самым удобным и презентабельным образом:



Это поправимо. У макета **GridLayout** есть функция объединения ячеек: элементы в нем могут занимать больше одной ячейки, как при работе с таблицами в текстовых и табличных процессорах (MS Office Word, MS Office Excel и так далее) или как для атрибута `span` в HTML. За это отвечают три атрибута: **layout_columnSpan** (*макет_диапазон столбцов*) для определения числа столбцов, занимаемых элементом (то есть размер по горизонтали), **layout_rowSpan** (*макет_диапазон строк*) — число строк, **layout_gravity** (*макет_притяжение*) для «растягивания» элемента на выбранное количество ячеек.

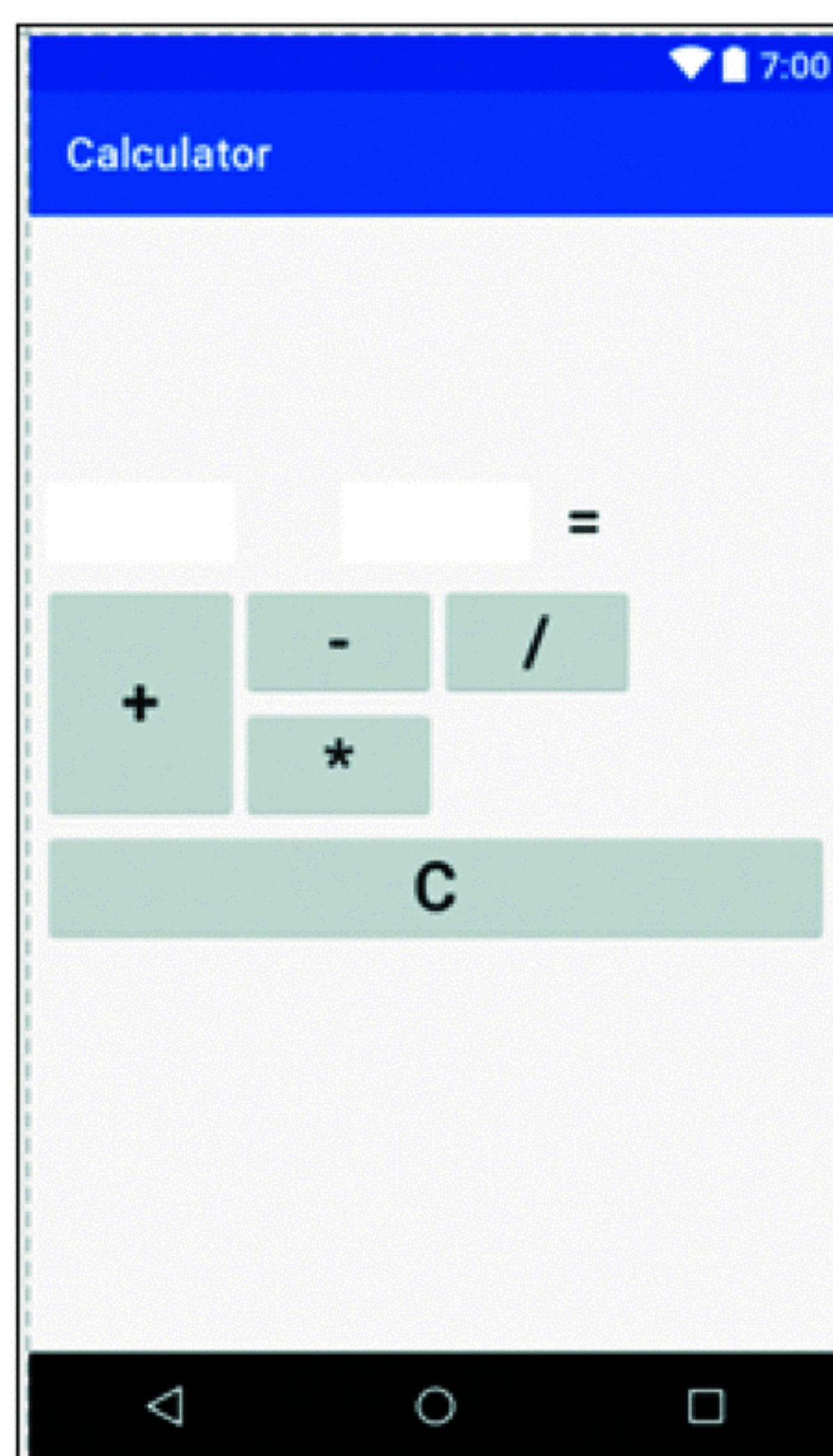
Отведем кнопке «+» две ячейки по вертикали. Для этого найдем ее описание в xml-разметке и добавим в него строки:

```
android:layout_rowSpan="2"  
android:layout_gravity="fill_vertical"
```

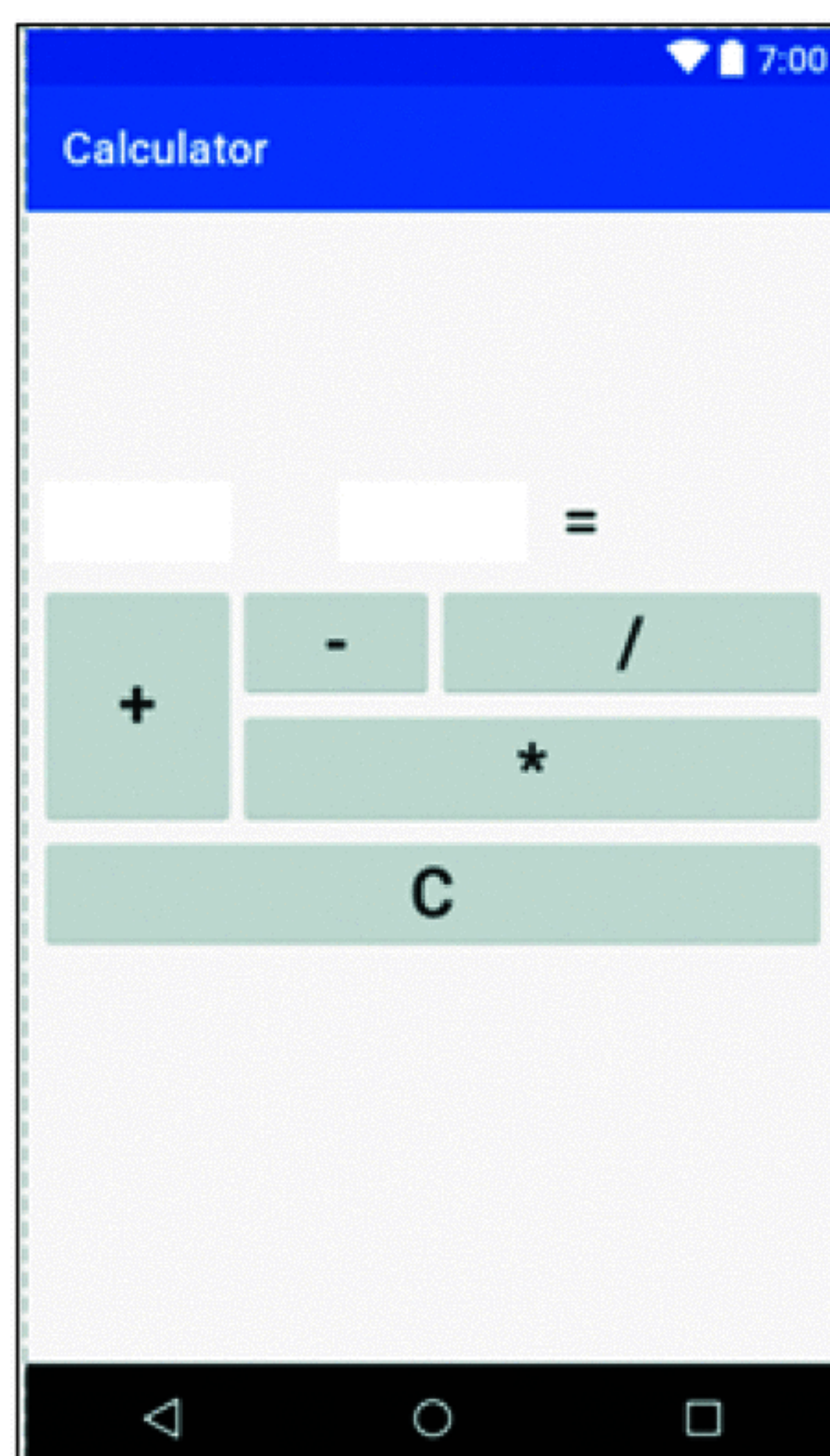
Первой строкой мы определили новую высоту кнопки — две строки, а второй — растянули кнопку на эту высоту по вертикали (**fill_vertical** — *заполнение по вертикали*).

А кнопку «C», например, растянем на три столбца по горизонтали:

```
android:layout_columnSpan="3"  
android:layout_gravity="fill_horizontal"
```




Теперь по аналогии нужно изменить разметку оставшихся трех кнопок так, чтобы они расположились следующим образом:



С разметкой мы закончили, осталось написать Java-код, чтобы наш калькулятор работал.

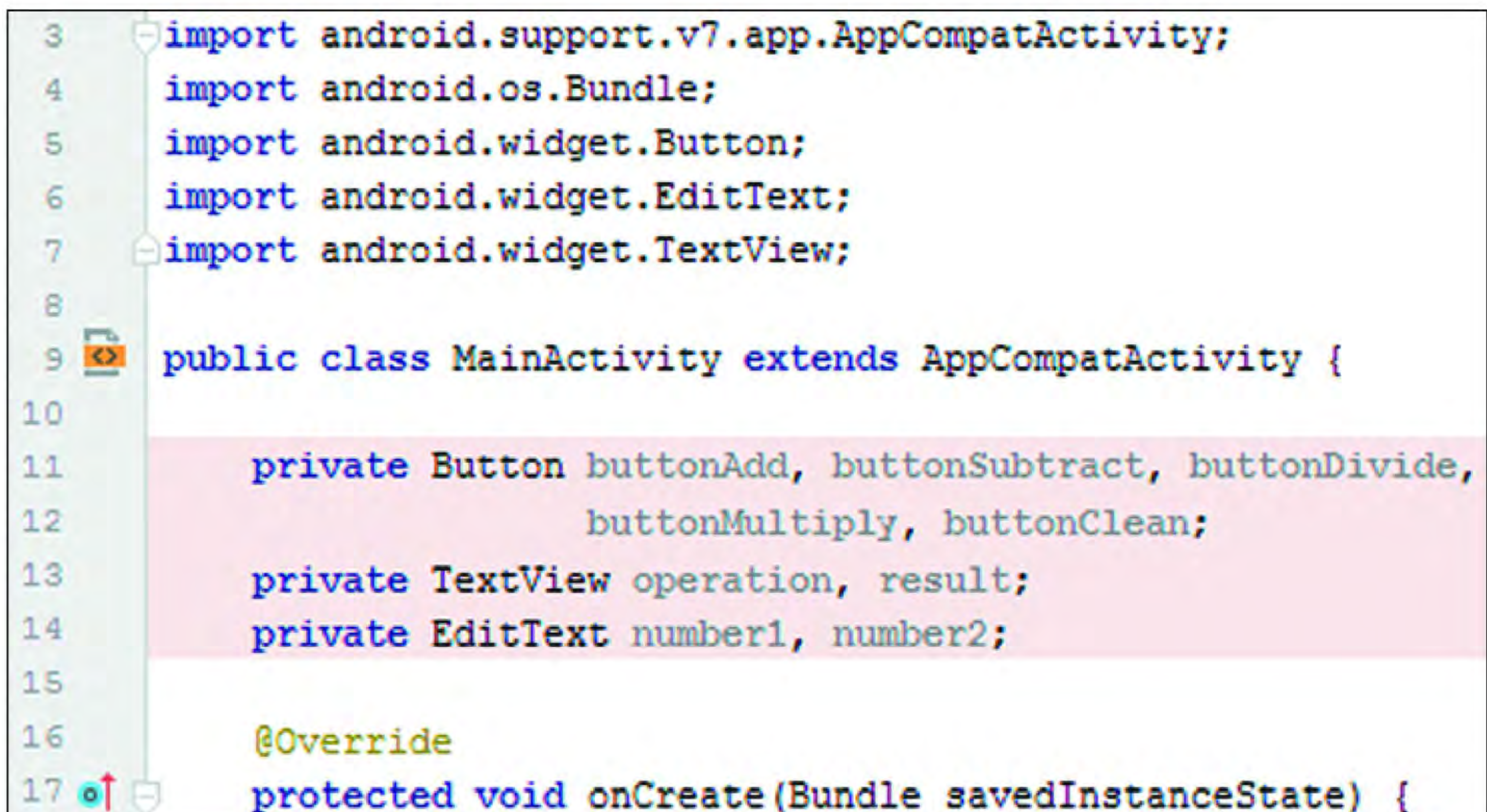
Шаг 8. Объявить переменные и определить элементы.

Откроем файл **MainActivity.java**:

A screenshot of the Android Studio IDE showing the MainActivity.java file. The code defines the MainActivity class, which extends AppCompatActivity. It includes imports for AppCompatActivity and Bundle. The onCreate method is overridden to call super.onCreate and setContentView(R.layout.activity_main).

```
1 package com.marusyafm.calculator;
2
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5
6 public class MainActivity extends AppCompatActivity {
7
8     @Override
9     protected void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         setContentView(R.layout.activity_main);
12     }
13 }
```

В классе **MainActivity** объявим переменные для всех наших кнопок и текстовых полей (не забывая импортировать предлагаемые **Android Studio** библиотеки):

A screenshot of the Android Studio IDE showing the MainActivity.java file. The code defines the MainActivity class, which extends AppCompatActivity. It includes imports for AppCompatActivity, Bundle, Button, EditText, and TextView. The onCreate method is overridden. Variable declarations for buttons, text view, and edit texts are added to the class.

```
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5 import android.widget.Button;
6 import android.widget.EditText;
7 import android.widget.TextView;
8
9 public class MainActivity extends AppCompatActivity {
10
11     private Button buttonAdd, buttonSubtract, buttonDivide,
12         buttonMultiply, buttonClean;
13     private TextView operation, result;
14     private EditText number1, number2;
15
16     @Override
17     protected void onCreate(Bundle savedInstanceState) {
```

Затем, как мы уже умеем, с помощью метода **findViewById()** определим элементы в методе **onCreate()**:


```

15     @Override
16     protected void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18         setContentView(R.layout.activity_main);
19
20         buttonAdd = (Button) findViewById(R.id.buttonAdd);
21         // ... остальные кнопки
22         operation = (TextView) findViewById(R.id.operation);
23         number1 = (EditText) findViewById(R.id.number1);
24         // ... остальные текстовые элементы
25     }
26 }

```

Шаг 9. Написать обработчики событий нажатия кнопок.

Для добавления обработчиков событий нажатия кнопок нужно сначала подключить «прослушиватели» нажатий. Для кнопок нажатие является сигналом для начала выполнения заданных действий. Для этого в название класса **MainActivity** допишем **implements View.OnClickListener** (реализующий прослушиватель нажатий).

```

9
10 public class MainActivity extends AppCompatActivity implements View.OnClickListener {
11
12

```

Class 'MainActivity' must either be declared abstract or implement abstract method 'onClick(View)' in 'OnClickListener'

Android Studio подчеркивает измененную строку и сообщает нам, что **Class 'MainActivity' must either be declared abstract or implement abstract method 'onClick(View)' in 'OnClickListener'** (класс *MainActivity* должен быть либо объявлен абстрактным, либо реализовывать абстрактный метод *onClick()* в *OnClickListener*). Следуя его подсказкам, нажимаем **Alt+Enter** и выбираем первый пункт — **Implement methods** (имплементировать, реализовать методы).

```

public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    private Button buttonAdd, buttonSubtract, buttonDivide;
    private TextView operation, result;
    private EditText number1, number2;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}

```

Implement methods

- Make 'MainActivity' abstract
- Remove breakpoint Ctrl+F8
- Create Test
- Create subclass
- Insert App Indexing API Code
- Make package-private

В конце программы появится метод **onClick()**, в котором мы сможем описать необходимые действия при нажатии любой кнопки.



Примечание

Java-классы бывают обычными и абстрактными.

Абстрактные классы в первую очередь существуют для того, чтобы предоставлять свои методы другим классам в качестве шаблона. Абстрактные классы могут содержать как обычные, так и абстрактные методы. Абстрактные методы — методы без прописанного внутри них функционала.

Таким образом, *имплементируя* (реализуя) методы абстрактного класса, мы получаем что-то вроде черновика для написания кода и доступ к использованию всех возможностей абстрактного класса внутри нашего класса.

Прежде чем что-то прописать в методе **onClick()**, нужно установить «прослушиватели» нажатия для всех кнопок в методе **onCreate()**. Если кнопок много, они расположены в одном блоке и выполняют однотипные действия, то прописывать отдельный метод **onClick()** для каждой кнопки не всегда удобно. В таких случаях прописывают только один **onClick()**, в который помещают инструкции для всех кнопок. А определением, какая именно кнопка была нажата, занимаются прослушиватели.

Прослушиватели нажатия кнопок объявляются в методе **onCreate()** с помощью метода **setOnClickListener()** (*установить прослушиватель нажатия*). В качестве параметра передадим методу ключевое слово **this** (*этот*), указывающее на текущий объект данного класса:

```
16      @Override
17      protected void onCreate(Bundle savedInstanceState) {
18          super.onCreate(savedInstanceState);
19          setContentView(R.layout.activity_main);
20
21          buttonAdd = (Button) findViewById(R.id.buttonAdd);
22          buttonSubtract = (Button) findViewById(R.id.buttonSubtract);
23          buttonDivide = (Button) findViewById(R.id.buttonDivide);
24          buttonMultiply = (Button) findViewById(R.id.buttonMultiply);
25          buttonClean = (Button) findViewById(R.id.buttonClean);
26          operation = (TextView) findViewById(R.id.operation);
27          result = (TextView) findViewById(R.id.result);
28          number1 = (EditText) findViewById(R.id.number1);
29          number2 = (EditText) findViewById(R.id.number2);
30
31          buttonAdd.setOnClickListener(this);
32          buttonSubtract.setOnClickListener(this);
33          // ... ОСТАЛЬНЫЕ КНОПКИ
34      }
```


Затем в методе `onClick()` объявляем переменные, необходимые нам для расчетов: `num1` для первого числа, `num2` для второго числа и `res` для результата вычислений. Используем тип данных `float`, потому что работаем не только с целыми значениями:

```
38      @Override
39      public void onClick(View v) {
40          float num1 = 0;
41          float num2 = 0;
42          float res = 0;
43      }
44  }
```

Действия для кнопок установим через оператор выбора `switch`.

Switch (*переключатель*) — условный оператор языка Java, который в отличие от операторов `if` и `if-else` применяется для списка известных значений, а также предусматривает ситуацию по умолчанию (**default**). Действия для каждого возможного значения описываются в своем операторе `case` (*случай*). Принятое на вход выражение сравнивается со значением каждого `case`, и, если совпадение найдено, выполняется команда именно этого `case`. В конце каждого `case` обычно ставится команда `break` для «выхода» из оператора `switch` и перехода к выполнению кода, стоящего после него. Для случаев, не описанных ни в одном `case`, выполняются действия, описанные в **default** (по умолчанию).

Но в операторе `switch` могут использоваться только типы данных `int`, `char`, `byte`, `short`, `enum` и `String`. Использование типа `float` приведет к ошибке выполнения. Поэтому, прежде чем писать `switch`, добавим в метод `onClick()` строки, в которых получим значения из `num1` и `num2` и «переведем» их из типа данных `float` в `String` (строки № 44–45):

```
38      @Override
39      public void onClick(View v) {
40          float num1 = 0;
41          float num2 = 0;
42          float res = 0;
43
44          num1 = Float.parseFloat(number1.getText().toString());
45          num2 = Float.parseFloat(number2.getText().toString());
46      }
47  }
```

Метод `getText()` (*получить текст*) «извлекает» текстовое содержание `TextView` (то, что прописано в свойстве `text` элемента).

Затем метод `toString()` (*в строку*) определяет этот извлеченный текст как строку. А метод `parseFloat()` (*распознать тип данных float*) «распознает» в получившейся строке число с плавающей точкой.

Вернемся к оператору `switch`. На его вход в качестве выражения для сравнения подается идентификатор нажатой кнопки (`getId()` — *получить идентификатор*). Значения для оператора `case` — идентификаторы всех имеющихся у нас кнопок: `buttonAdd`, `buttonSubtract`, `buttonDivide`, `buttonMultiply` и `buttonClean`. И поскольку мы опишем все возможные значения, для действия по умолчанию остается только выход из оператора:

```

44     num1 = Float.parseFloat(number1.getText().toString());
45     num2 = Float.parseFloat(number2.getText().toString());
46
47     switch (v.getId()) {
48         case R.id.buttonAdd:
49             operation.setText("+");
50             res = num1 + num2;
51             break;
52         // ... по аналогии для "-", "*" и "/"
53         case R.id.buttonClean:
54             number1.setText("");
55             operation.setText("");
56             number2.setText("");
57             result.setText("");
58             break;
59         default:
60             break;
61     }
62 }
63 }

```

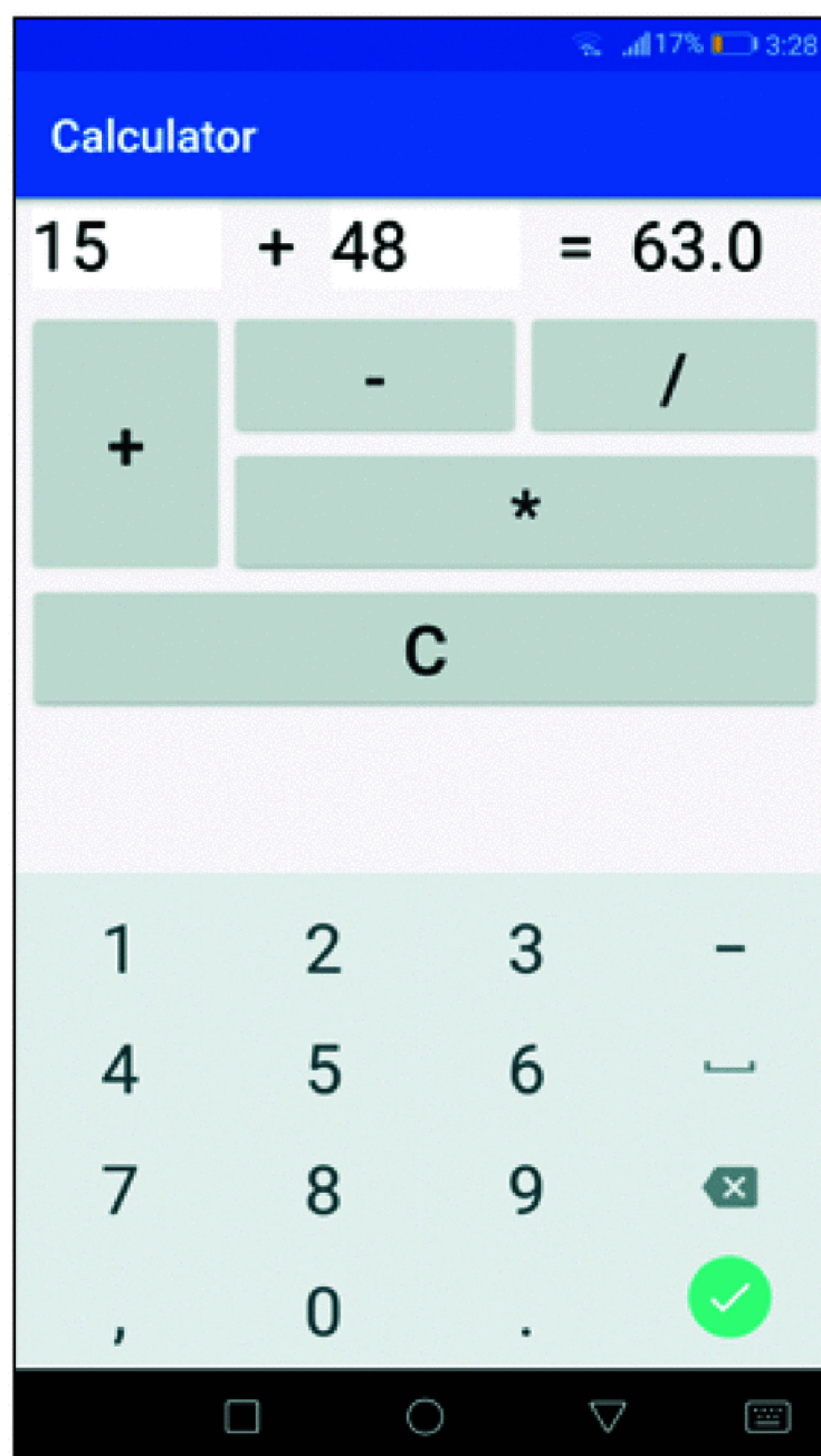
Для кнопки сложения в `TextView operation` устанавливаем текстовое значение «+» (строка № 49), а переменной `res` присваиваем значение суммы `num1` и `num2` (строка № 50). По аналогии опишем значения `case` для кнопок вычитания, умножения и деления. Для кнопки очистки всем текстовым полям устанавливаем пустое значение (строки № 54–57).

В самом конце метода `onClick()`, после оператора `switch`, добавляем строку вывода результата:

```
result.setText(res+"");
```

Готово!

Шаг 10. Собрать проект, запустить приложение и проверить работу нашего калькулятора через эмулятор или на реальном устройстве.



Задания

1. Доработать приложение до корректного отображения при горизонтальной ориентации экрана.
2. Доработать приложение до корректного отображения на больших экранах.
3. Добавить функцию возведения в степень.
4. Проработать дизайн приложения.

Итоги главы 2

В главе 2 мы освоили одну из основополагающих вещей Android-разработки — основы проектирования интерфейса. Если не уделить этому вопросу должного внимания, каким бы грамотным ни был код — элементы могут начать наслаиваться друг на друга или выстраиваться не в том порядке. В результате с ними будет гораздо сложнее работать на этапе разработки приложения. Само же приложение не будет корректно отображаться на устройствах с разной диагональю экрана и не будет реагировать на изменение ориентации экрана.