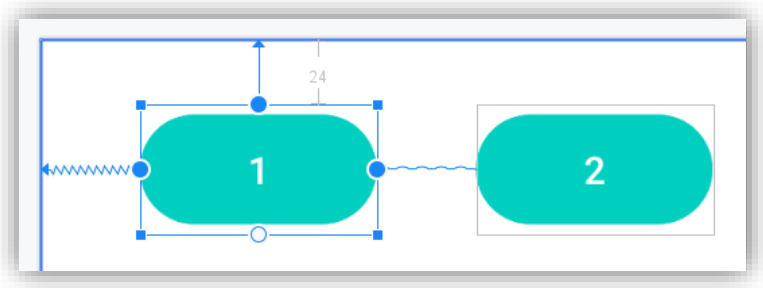
Контейнер ConstraintLayout

Контейнер ConstraintLayout позволяет создавать сложные макеты. Важно разобраться с его использованием и возможностями, потому что он является контейнером по умолчанию в новых проектах, и Google рекомендует использовать именно его.

Ограничения

Чтобы определить положение элемента внутри контейнера, нужно добавить ему как минимум одно горизонтальное и одно вертикальное *ограничение*, то есть *привязать* или *выровнять* его по отношению к другому элементу или контейнеру:

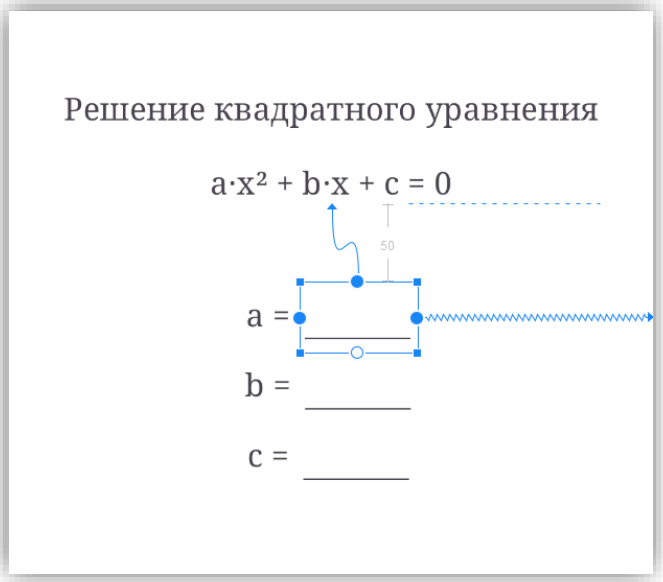


На рисунке кнопка с текстом «1» привязана своей левой стороной к левому краю контейнера ConstraintLayout, верхним краем – к верхнему краю контейнера с отступом 24dp, а правым краем – к левому краю кнопки с текстом «2».

Если у элемента нет ни одного ограничения, Android Studio будет указывать на отсутствие ограничений, однако программа всё равно может быть скомпилирована и запущена. Такой элемент без ограничений окажется в левом верхнем углу контейнера.

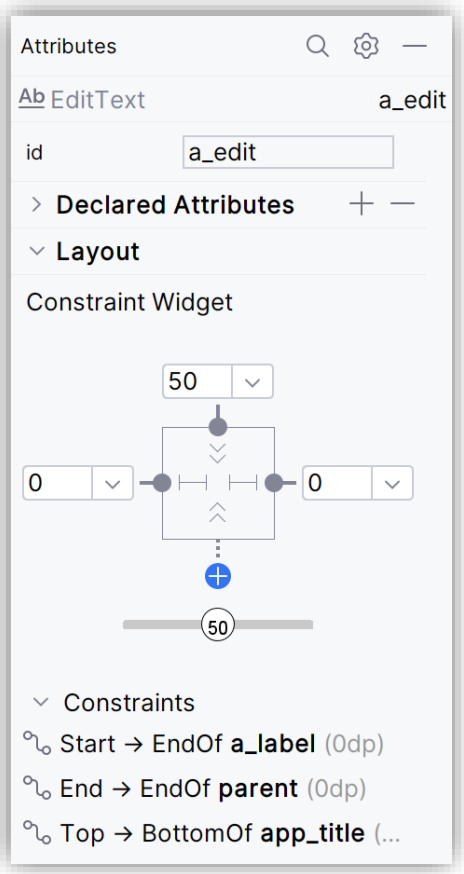
Работа с ограничениями из XML-кода довольно сложна, потому что обычно ограничений у каждого элемента несколько, и перемещение элемента порой требует перепривязать элемент к другому элементу. Тем не менее, умение работать с XML-кодом может сделать работу более эффективной, поэтому далее разбираются оба варианта установки ограничений.

Для установки ограничений в режиме редактирования макета, нужно разместить требуемые элементы внутри контейнера ConstraintLayout, а затем выбрать мышкой какой-либо из этих элементов:



Синим цветом Android Studio обозначит рамку элемента, на каждой стороне рамки будут синие кружки. Закрашенный кружок ● означает что данная сторона элемента привязана к чему-либо, пустой кружок ○ означает отсутствие ограничений.

Чтобы привязать сторону элемента, нужно нажать мышкой в выбранном кружке и не отпуская кнопку тянуть его до другого элемента или края контейнера, а затем отпустить. В панели Attributes при этом появится информация об ограничении:



В разделе Constraint Widget серыми закрашенными кружками показаны привязанные стороны, непривязанные стороны обозначены синим кружком со знаком + (он привяжет

сторону к ближайшему элементу с этой стороны, а если таких нет – то к границе контейнера). Около каждой стороны приведено расстояние в единицах `dp` до элемента, к которому осуществлена привязка. Чуть ниже в разделе Constraints приведена информация к какому элементу привязана каждая из сторон.

В режиме кода ограничения задаются свойствами вида (обратите внимание, что вместо привычного пространства имён `android:` используется `app:`):

```
app:layout_constraintЧТО_toКУДАOf="ЭЛЕМЕНТ"
```

Например, чтобы привязать *верхнюю* сторону элемента к *верху* контейнера, нужно использовать такое свойство:

```
app:layout_constraintTop_toTopOf="parent"
```

Указать сторону можно следующими ключевыми словами:

- `Top` – верхняя
- `Bottom` – нижняя
- `Start` – левая (если направление текста слева направо)
- `End` – правая (если направление текста слева направо)
- `Baseline` – линия шрифта

Элемент, к которому осуществляется привязывание стороны, указывается справа в кавычках. Если это обычный элемент, то нужно указать его идентификатор, а если привязывание осуществляется к контейнеру, то используется ключевое слово `parent`. Например, чтобы связать правую сторону элемента с левой стороной другого элемента (с идентификатором `app_title`), можно использовать такой код:

```
app:layout_constraintStart_toEndOf="@id/app_title"
```

Обязательно следует убедиться, что присутствуют хотя бы два ограничения – одно горизонтальное, и одно вертикальное.

Чтобы добавить отступ от элемента или границы, используется свойство `layout_margin` или одно из его разновидностей (`layout_marginStart`, `layout_marginTop` и т. д.). Например, следующий код разместит кнопку в правом нижнем углу контейнера, на указанном расстоянии:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    android:layout_marginBottom="10dp"
    android:layout_marginEnd="20dp"/>
```

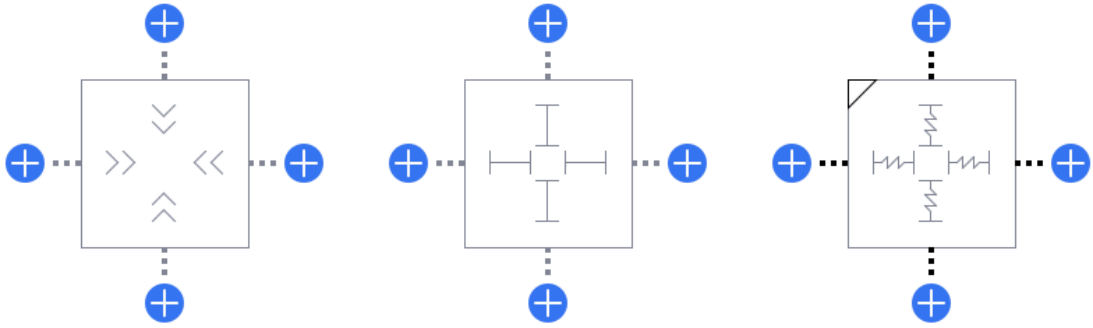
Порой возникает сильное искушение не тратить время на задание ограничений, а привязать все элементы, например, к границам контейнера, и с помощью отступов отрегулировать положение элементов внутри контейнера. Делать так крайне не рекомендуется! На устройствах с другими разрешениями экрана все элементы могут съехать самым

непредсказуемым образом, и в лучшем случае на экране появятся пустые пространства, а в худшем элементы будут перекрываться или выходить за пределы экрана!

Размеры элемента

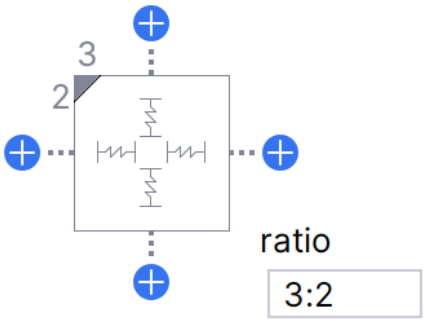
Размер элемента в ConstraintLayout может задаваться следующими способами:

- `wrap_content` – размер элемента зависит от содержимого элемента, в панели Attributes отображается *угловыми скобками*;
- `fixed` – размер элемента задаётся фиксированным числом (в единицах `dp`), в панели Attributes отображается *отрезком*;
- `0dp` – элемент старается занять всё доступное ему пространство, в панели Attributes отображается *зигзагом*.



Для смены режима размера в панели Attributes можно щёлкнуть линию (скобки, отрезок или зигзаг), а в режиме редактирования кода нужно просто задать свойства `layout_width` или `layout_height`.

Если размер элемента по какой-либо из осей задан как `0dp`, то появляется возможность настроить пропорции сторон. Для этого в панели Attributes можно щёлкнуть небольшой треугольник в левом верхнем углу условного элемента, а затем задать соотношение сторон:



В режиме же редактирования кода соотношение задаётся свойствами:

```
app:layout_constraintDimensionRatio="h,3:2"
```

Здесь при изменении ширины элемента будет меняться высота (`h`), которая изначально задана как `0dp`. И наоборот, в коде:

```
app:layout_constraintDimensionRatio="w,16:9"
```

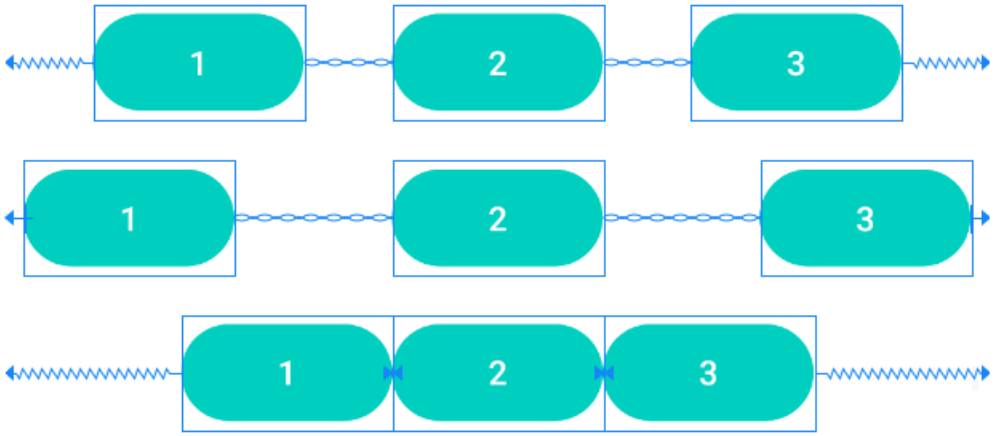
при изменении высоты элемента будет меняться ширина (`w`), она изначально должна быть задана как `0dp`.

Цепочки

Элементы в ConstraintLayout можно сгруппировать друг с другом, для этого они объединяются в цепочки (*англ.* chains). Чтобы создать цепочку в графическом режиме, нужно выбрать элементы для связывания, нажать правой кнопкой мыши и в контекстном меню выбрать Chains → Create Horizontal Chain (или Vertical для вертикальной цепочки).

Цепочки бывают трёх видов:

- `spread` – элементы внутри цепочки распределяются равномерно, сама цепочка центрируется внутри контейнера;
- `spread_inside` – элементы внутри цепочки распределяются равномерно, сама цепочка растянута внутри контейнера вплотную к соответствующим сторонам;
- `packed` – элементы внутри цепочки соединяются без отступов, сама цепочка центрируется внутри контейнера.



В коде вид цепочки можно задать с помощью свойства `app:layout_constraintHorizontal_chainStyle` и `app:layout_constraintVertical_chainStyle`, присвоив им одно из перечисленных значений. Можно указать это свойство только у одного из связанных элементов цепочки, вся цепочка автоматически будет использовать этот вид.

Задание

Напишите приложение для нахождения корней квадратного уравнения.

Интерфейс программы должен быть построен с помощью контейнера `ConstraintLayout`, без использования других контейнеров. Элементы должны быть привязаны друг к другу, по вертикальной оси допускается использовать свойство `layout_marginTop` для отступов, другие свойства для отступов использоваться не должны.

Программа должна отслеживать изменения в текстовых полях, в которые вводятся коэффициенты a, b и c. Когда в этих полях оказываются введены значения, которые можно распознать как числа (целые или вещественные) – программа без дополнительных действий со стороны пользователя вычисляет и выводит корни уравнения (или информацию об отсутствии одного или обоих корней).

Для вычисления корней уравнения можно использовать любой известный (и даже неизвестный) метод. При написании примера использовался метод вычисления корней с помощью дискриминанта:

1. Вычисляется дискриминант:

$$D = b^2 - 4ac$$

2. Далее, в зависимости от значения дискриминанта, вычисляется один или несколько корней:

$$\begin{cases} \text{корней нет, если } D < 0 \\ x = -\dfrac{b}{2a}, \text{если } D = 0 \\ x_{1,2} = \dfrac{-b \pm \sqrt{D}}{2a}, \text{если } D > 0 \end{cases}$$

Контрольные примеры для проверки правильности работы программы:

Решение квадратного уравнения a·x² + b·x + c = 0 a = 1 b = 4 c = -21 Найдены два корня: 3.0 -7.0	Решение квадратного уравнения a·x² + b·x + c = 0 a = 1 b = -2 c = 1 Найден один корень: 1.0	Решение квадратного уравнения a·x² + b·x + c = 0 a = 2 b = -1 c = 1 Корней нет

Если для задания ограничений использовался визуальный редактор, следует разобраться с текстовым представлением свойств, чтобы понимать, что каждое из них означает.