**Лекция 2 Ресурсы и макеты**

В этой лекции обсуждаются [**Ресурсы**](https://developer.android.com/guide/topics/resources/overview.html) , которые используются для представления элементов или данных, которые отделены от поведения (функциональной логики) приложения. В частности, в этой лекции основное внимание уделяется тому, как ресурсы используются для определения **Макетов** для пользовательских интерфейсов. В этой лекции основное внимание уделяется исходному коду на основе XML в приложении Android; лекция [Activity](https://info448.github.io/activities.html#activities) начинает детализировать исходный код, написанный на Java.

В этой лекции используется код, найденный по адресу <https://github.com/info448/lecture02-layouts> .

**2.1 Ресурсы**

[Ресурсы](https://developer.android.com/guide/topics/resources/overview.html) можно найти в **res/**папке, и они представляют элементы или данные, которые являются «внешними» по отношению к коду. Вы можете думать о них как о «медиаконтенте»: часто изображения, но также и такие вещи, как текстовые вырезки (или короткие строковые константы), обычно определенные в XML-файлах. Ресурсы представляют компоненты, которые *отделены* от поведения приложения, поэтому хранятся отдельно от кода Java для поддержки **принципа разделения ответственности**

* Определяя ресурсы в XML, их можно разрабатывать (работать) *без* инструментов кодирования (например, с помощью таких систем, как вкладка графического «дизайна» в Android Studio). Теоретически вы могли бы поручить графическому дизайнеру создать эти ресурсы, которые затем можно интегрировать в код без необходимости дизайнеру владеть Java.
* Аналогично, разделение ресурсов позволяет вам выбирать, какие ресурсы включать *динамически* . Вы можете выбрать показ разных изображений в зависимости от разрешения экрана устройства или выбрать разные строки в зависимости от языка устройства (интернационализация!) — поведение приложения то же самое, но «контент» другой!

Каким должен быть ресурс? В общем:

* Макеты **всегда** следует определять как ресурсы.
* Элементы управления пользовательского интерфейса (кнопки и т. д.) *в основном* следует определять как ресурсы (они являются частью макетов), хотя поведение будет определяться программно в Java.
* Любые графические изображения (рисунки) должны быть определены как ресурсы.
* Любые строки *, доступные пользователю,* должны быть определены как ресурсы.
* Информация о стиле и тематике должна быть определена как ресурсы.

Как было показано в [лекции 1](https://info448.github.io/introduction.html#introduction) , в Android используется ряд различных [типов ресурсов](https://developer.android.com/guide/topics/resources/available-resources.html) , которые можно найти в res/папке проекта Android по умолчанию, в том числе:

* res/drawable/: содержит графику (PNG, JPEG и т. д.)
* res/layout/: содержит файлы макета пользовательского интерфейса XML
* res/mipmap/: содержит файлы иконок запуска в разных разрешениях
* res/values/: содержит определения XML для общих констант, которые могут включать:
  + /strings.xml: короткие строковые константы (например, метки кнопок)
  + /colors.xml: константы цвета
  + /styles.xml: константы для деталей [стиля и темы](https://developer.android.com/guide/topics/ui/themes.html)
  + /dimen.xml: размерные константы (например, поля по умолчанию); не создаются по умолчанию в Android Studio 2.3+.

Подробная информация об этих различных типах ресурсов немного разбросана по всей документации, но хорошим началом будет раздел [«Типы ресурсов](http://developer.android.com/guide/topics/resources/available-resources.html)[2](https://info448.github.io/resources-and-layouts.html#fn2)[», а также раздел «Предоставление ресурсов»](https://developer.android.com/guide/topics/resources/providing-resources.html#ResourceTypes) .

**Р**

Ресурсы обычно определяются как [XML](https://en.wikipedia.org/wiki/XML) (который по синтаксису похож на HTML). Когда приложение компилируется, инструменты сборки (например, Gradle) генерируют **дополнительный** класс Java, называемый **R**(для «ресурса»). Этот класс содержит то, что по сути является гигантским списком статических «констант» — по крайней мере, по одной для каждого элемента XML.

Например, рассмотрим strings.xmlресурс, который используется для определения строковых констант. Предоставленный strings.xmlопределяет две константы типа <string>. nameАтрибут указывает имя, которое будет принимать переменная, а содержимое элемента дает значение этой переменной. Таким образом

**<string** name="app\_name"**>**Layout Demo**</string>**

**<string** name="greeting"**>**Hello Android!**</string>**

фактически будут скомпилированы в константы, подобные:

**public** static final String app\_name = "My Application";

**public** static final String greeting = "Hello Android!";

Все константы ресурсов компилируются во *внутренние классы* внутри R, по одному для каждого типа ресурса. Таким образом, Rфайл, содержащий вышеуказанные строки, будет иметь следующую структуру:

**public** **class** R {

**public** static **class** string {

**public** static final String app\_name = "My Application";

**public** static final String greeting = "Hello Android!";

}

}

Это позволяет использовать **точечную нотацию** для ссылки на каждый ресурс на основе его типа (например, R.string.greeting) — аналогично синтаксису, используемому для ссылки на вложенные объекты JSON!

* Для большинства ресурсов идентификатор определяется как атрибут элемента ( nameатрибут для значений, таких как строки; idдля определенных элементов View в макетах). Для более сложных ресурсов, таких как целые макеты или чертежи, идентификатором является имя *файла* (без расширения файла): например, R.layout.activity\_mainотносится к корневому элементу файла layout/activity\_main.xml.
* В более общем смысле, каждый ресурс можно обозначить с помощью [(package\_name).]R.resource\_type.identifier.
* Обратите внимание, что имя файла string.xml— это просто соглашение для удобства чтения; все дочерние элементы элемента <resource>компилируются в Rзависимости от их типа, а не от расположения исходного кода. Таким образом, возможно иметь множество различных файлов ресурсов, в зависимости от ваших потребностей. Файл robot\_list.xmlне является стандартным ресурсом.

Сгенерированный файл можно найти R.javaвнутри app/build/generated/source/r/debug/...(используйте представление «Файлы проекта» в Android Studio).

Если вы действительно откроете R.javaфайл, вы увидите, что статические константы на самом деле являются просто **int**значениями, которые являются *указателями* на ссылки на элементы (аналогично передаче a pointer\*в языке C); содержимое значения хранится в другом месте (поэтому его можно корректировать во время выполнения; см. ниже). Это означает, что в нашем коде Java мы обычно работаем с intтипом данных для ресурсов XML, таких как строки, потому что мы фактически работаем с указателями *на* эти ресурсы.

* Например, setContentView()вызов в Activity onCreate()принимает ресурс int.
* Вы можете думать о каждой intконстанте как о «ключе» или «индексе» для этого ресурса (в списке всех ресурсов). Android выполняет тяжелую работу int, беря его, просматривая во внутренней таблице ресурсов, находя связанный XML-файл, а затем извлекая нужный элемент из этого XML. (Под тяжелой работой я подразумеваю реализацию. Android ищет эти ссылки непосредственно в памяти, поэтому поиск выполняется быстро).

Поскольку Rкласс включен в Java, мы можем получить доступ к этим intконстантам напрямую в нашем коде (как R.resource\_type.identifier), как в setContentView()методе. Однако, если вы хотите получить значение String, вы можете найти его, используя объект приложения Resources():

Resources res = **this**.getResources(); *//get access to application's resources*

String myString = res.getString(R.string.myString); *//look up value of that resource*

* Другим распространенным методом, использующим ресурсы, является findViewById(int)метод , который применяется для ссылки на Viewэлемент (например, кнопку), указанный в ресурсе макета, чтобы вызывать для него методы в Java, как в примере из предыдущей лекции.

Расширение [Kotlin для Android](https://kotlinlang.org/docs/tutorials/android-plugin.html) позволит вам создавать importмакеты непосредственно в Kotlin, поэтому вам не нужно использовать findViewById().

Класс Rрегенерируется все время (всякий раз, когда вы меняете ресурс, что случается часто); когда Eclipse был рекомендуемой Android IDE, вам часто приходилось вручную регенерировать класс, чтобы индекс IDE оставался актуальным! Вы можете выполнить аналогичную задачу в Android Studio, используя Build > Clean Projectи Build > Rebuild Project.

Также возможно ссылаться на один ресурс из другого в XML с помощью **@**символа, следуя схеме @[<package\_name>:]<resource\_type>/<resource\_name>. Например, в Манифесте вы можете видеть, что метка приложения ссылается через @string/app\_name.

* Вы также можете использовать **+**символ для создания *нового* ресурса, на который мы можем ссылаться; это немного похоже на объявление переменной внутри атрибута XML. Это чаще всего используется с android:idатрибутом ( android:id="@+id/identifier") для создания переменной, ссылающейся на это представление; подробности см. ниже.

**Альтернативные Ресурсы**

Одним из главных преимуществ отделения ресурсов от кода Java является то, что это позволяет локализовать их **и** изменять в зависимости от устройства! Android позволяет разработчику указывать папки для [«альтернативных» ресурсов](https://developer.android.com/guide/topics/resources/providing-resources.html#AlternativeResources) , например, для разных языков или разрешений экрана устройства. **Во время выполнения** Android проверит конфигурацию устройства и попытается найти альтернативный ресурс, соответствующий этой конфигурации. Если он *не сможет* найти соответствующий альтернативный ресурс, он вернется к ресурсу «по умолчанию».

Существует множество различных конфигураций, которые можно использовать для влияния на ресурсы; см. [Предоставление ресурсов](https://developer.android.com/guide/topics/resources/providing-resources.html#AlternativeResources)[3.](https://info448.github.io/resources-and-layouts.html#fn3) Чтобы выделить несколько вариантов, вы можете указать различные ресурсы на основе:

* Язык и регион (например, с помощью двухбуквенных кодов ISO)
* Размер экрана( small, normal, medium, large, xlarge)
* Ориентация экрана ( portдля портретной, landдля альбомной)
* Конкретная плотность пикселей экрана (dpi) ( ldpi, mdpi, hdpi, xhdpi, xxhdpi, и т. д.) xxhdpiдовольно распространена для высококлассных устройств. Обратите внимание, что dpi — это «точек на дюйм», поэтому эти значения представляют собой количество пикселей *относительно* размера устройства!
* Версия платформы ( v1, v4, v7… для каждого номера API)

Конфигурации указываются с помощью **имени каталога** , придавая папкам форму <resource\_name>(-<config\_qualifier>)+. Например, values-fr/будет содержать постоянные значения для устройств с конфигурацией на французском языке.

* Важно, чтобы сам файл ресурсов был одинаковым *как* для квалификатора, так и для неквалифицированного имени ресурса (например, values/strings.xmlи values-fr/strings.xml). Это связано с тем, что Android загрузит файл внутри квалифицированного ресурса, если он соответствует конфигурации устройства *вместо* «стандартного» неквалифицированного ресурса. Имена должны быть одинаковыми, чтобы одно могло заменить другое!
* Вы можете увидеть это в действии, используя мастер создания *нового ресурса* ( File > New > Android resource file) для создания строкового ресурса (например, для app\_name) на [другом языке](https://www.webucator.com/blog/2010/03/saying-hello-world-in-your-language-using-javascript/) . Измените языковые настройки устройства (через Settings > Language & Input > Language), чтобы увидеть автоматическую настройку содержимого!
* **<?xml** version="1.0" encoding="utf-8"**?>**
* **<resources>**
* **<string** name="app\_name"**>**Mon Application**</string>**

**</resources>**

* Вы можете просмотреть структуру каталогов, которая поддерживает это, переключившись в режим Packageпросмотра проекта в Android Studio.

**2.2 Просмотры**

Наиболее распространенный тип элемента, который вы определите как ресурс, — это [**Views**](https://developer.android.com/reference/android/view/View.html)[4](https://info448.github.io/resources-and-layouts.html#fn4) . View— это суперкласс для визуальных элементов интерфейса — визуальный компонент на экране — это View. Конкретные типы Views включают: [TextViews](https://developer.android.com/reference/android/widget/TextView.html) , [ImageViews](https://developer.android.com/reference/android/widget/ImageView.html) , [Buttons](https://developer.android.com/reference/android/widget/Button.html) и т. д.

* Viewявляется суперклассом для этих компонентов, поскольку он позволяет нам использовать **полиморфизм** для обработки всех этих визуальных элементов как экземпляров одного и того же типа. Мы можем выкладывать их, рисовать их, щелкать по ним, перемещать их и т. д. И все поведение будет одинаковым (хотя подклассы также могут иметь «дополнительные» функции).

Вот в чем главный трюк: один подкласс Viewis [5](https://info448.github.io/resources-and-layouts.html#fn5) . A is a View может содержать другие «дочерние» Views. Но поскольку is a … он может содержать больше внутри себя! Таким образом, мы можем **вкладывать** Views в Views, следуя [Composite Pattern](http://www.oodesign.com/composite-pattern.html) . В конечном итоге это работает во многом как HTML (который может иметь элементы DOM как внутри других элементов DOM), позволяя создавать сложные пользовательские интерфейсы.[ViewGroup](https://developer.android.com/reference/android/view/ViewGroup.html)ViewGroupViewGroupViewViewGroups<div>

* Подобно HTML DOM, представления Android структурированы в виде [*дерева*](https://en.wikipedia.org/wiki/Tree_(data_structure)) , которое называется **иерархией представлений** .

Представления определяются внутри [макетов](https://developer.android.com/guide/topics/ui/declaring-layout.html) — то есть внутри ресурса макета, который является XML-файлом, описывающим представления. Эти ресурсы *«раздуваются»* (визуализируются) в объекты пользовательского интерфейса, которые являются частью приложения.

Технически, a Layout— это просто ViewGroup, которые предоставляют информацию об «упорядочивании» и «позиционировании» для Views внутри него. Макеты позволяют системе «размещать» Views разумно и эффективно. *Отдельные Views не должны знать свою собственную позицию* ; это следует из хорошего объектно-ориентированного дизайна и сохраняет Views инкапсулированными.

Android Studio поставляется с графическим [редактором макетов](https://developer.android.com/studio/write/layout-editor.html) (вкладка «Дизайн»), который можно использовать для создания макетов. Однако большинство разработчиков продолжают писать макеты в XML. Это в основном потому, что ранние инструменты дизайна были жалкими и непригодными для использования, поэтому XML был всем, что у нас было. Хотя графический редактор Android Studio может быть эффективным, для этого курса вам следует создавать макеты «вручную» в XML. Это полезно для того, чтобы убедиться, что вы понимаете части, лежащие в основе разработки, и это навык, с которым вы в любом случае должны быть комфортно (аналогично тому, как мы призываем людей использовать gitиз командной строки).

**Просмотр свойств**

Прежде чем мы перейдем к тому, как группировать Views, давайте сосредоточимся на отдельных базовых Viewклассах. В качестве примера рассмотрим activity\_mainмакет в коде лекции. Этот макет содержит два отдельных Viewэлемента (внутри a Layout): a TextViewи a Button.

Все View имеют **свойства** , которые определяют состояние View. Свойства обычно указываются в XML-ресурсе как *атрибуты* элемента . Некоторые примеры этих атрибутов свойств описаны ниже.

* **android:id**указывает уникальный идентификатор для View. Этот идентификатор должен быть уникальным в пределах макета, хотя в идеале он должен быть уникальным в пределах всего приложения для ясности.

Синтаксис @+используется для определения *нового* ресурса View id— почти как если бы вы объявляли переменную внутри атрибута элемента! Вам нужно будет использовать всякий @+раз, когда вы указываете новый id, что позволит ссылаться на него либо из кода Java (как R.id.identifier), либо из других ресурсов XML (как @id/identifier).

Идентификаторы должны быть допустимыми именами переменных Java (потому что они преобразуются в имя переменной в Rклассе) и по соглашению именуются в lower\_caseформате .

* + *Совет по стилю* : полезно добавлять к идентификатору каждого представления его тип (например, btn, txt, edt). Это помогает сделать код самодокументируемым!

Каждому интерактиву следует присвоить Viewуникальный идентификатор, который позволит автоматически сохранять его состояние при уничтожении Activity. Подробности см. [здесь .](https://developer.android.com/guide/components/activities/activity-lifecycle.html#saras)

* **android:layout\_width**и **android:layout\_height**используются для указания размера View на экране (см. документацию [ViewGroup.LayoutParams](http://developer.android.com/reference/android/view/ViewGroup.LayoutParams.html) ). Эти значения могут быть определенным значением (например, 12dp), но чаще всего это одно из двух специальных значений:
  + wrap\_content, то есть размер должен быть таким, как того требует содержимое, плюс отступ.
  + match\_parent, то есть размер должен быть таким же большим, как *родительский* (контейнерный) элемент, за вычетом отступа. Это значение было переименовано из fill\_parent(теперь устарело).

Android использует следующие [измерения](https://developer.android.com/guide/topics/resources/more-resources.html#Dimension) или [единицы](https://www.google.com/design/spec/layout/units-measurements.html) :

* **dp**является «пикселем, не зависящим от плотности». На экране с разрешением 160 точек на дюйм (dpi) 1dpравно 1px(пиксель). Но по мере увеличения dpi увеличивается и количество пикселей на дюйм dp. Эти значения следует использовать вместо px, поскольку это позволяет измерениям работать независимо от dpi оборудования (которое *сильно* варьируется).
* **px**это фактический пиксель экрана. *НЕ ИСПОЛЬЗУЙТЕ ЭТО* (используйте dpвместо этого!)
* **sp**является «пикселем, не зависящим от масштаба». Это значение похоже на dp, но масштабируется в соответствии с системными настройками шрифта (например, если пользователь выбрал, что устройство должно отображать более крупный шрифт, 1spбудет охватывать больше dp). ***Всегда****следует использовать для размеров текста, чтобы поддерживать настройки пользователя и доступность.sp*
* **pt**составляет 1/72 дюйма физического экрана. Аналогичные единицы mmи inдоступны. *Не рекомендуется для использования.*
* **android:padding**, **android:paddingLeft**, **android:margin**, **android:marginLeft**, и т. д. используются для указания полей и отступов для Views. Они работают в основном так же, как и в CSS: отступ — это пространство между содержимым и «краем» View, а margin — это пространство между Views. Обратите внимание, что в отличие от CSS поля между элементами не схлопываются.
* **android:textSize**указывает «размер шрифта» текстовых представлений (используйте spединицы измерения!), **android:textColor**указывает цвет текста (рекомендуется ссылаться на цветовой ресурс!) и т. д.
* Также есть много других свойств! Вы можете увидеть список общих свойств в документации [6](https://info448.github.io/resources-and-layouts.html#fn6) , просмотреть параметры на вкладке «Design» в Android Studio или просмотреть параметры автозаполнения в IDE. Каждый отдельный класс (например, , , и т. д.) также будет иметь свой собственный набор свойств.[View](https://developer.android.com/reference/android/view/View.html#lattrs)ViewTextViewImageView

Обратите внимание, что в отличие от CSS, свойства стилей, указанные в ресурсах XML макета, *не* наследуются: вы фактически указываете встроенный styleатрибут для этого элемента, и тот, который не повлияет на дочерние элементы. Чтобы определить общие свойства стилей, вам нужно будет использовать [ресурсы стилей](https://developer.android.com/guide/topics/ui/themes.html#DefiningStyles) , которые обсуждаются в следующей лекции.

**Виды и Java**

Отображение View на экране называется **инфляцией** этого View. Процесс называется «инфляцией» на основе идеи, что он «распаковывает» или «разворачивает» компактное описание ресурса в сложный Java Object. Когда View инфляциируется, он инстанциируется как объект: процесс инфляции изменяет XML <Button>в new Button()объект в Java, при этом атрибуты свойств передаются в качестве [параметра этому конструктору](https://developer.android.com/reference/android/view/View.html#View(android.content.Context,%20android.util.AttributeSet)) . Таким образом, вы можете думать о каждом элементе XML как о представлении конкретного Java Object, который будет инстанцирован и на который будет сделана ссылка во время выполнения.

* Это почти то же самое, что и компоненты JSX в React, которые являются отдельными объектами!
* Помните, что вы можете получить ссылку на эти объекты из кода Java с помощью findViewById()метода.

После того, как у вас есть ссылка на объект View в Java, можно динамически указывать визуальные свойства с помощью методов Java (например, setText(), setPadding()). Однако следует использовать методы Java для указания свойств View **только тогда, когда они***должны* быть динамическими (например, текст изменяется в ответ на нажатие кнопки) — гораздо чище и эффективнее указывать как можно больше визуальных деталей в файлах ресурсов XML. Также можно динамически заменять один ресурс макета другим (см. ниже).

* Представления также имеют методы проверки, такие как isVisible()и , hasFocus()если вам нужно проверить состояние представления.
* Если вы используете расширение Kotlin для Android, вы можете получить доступ к этим свойствам напрямую.

**НЕ** создавайте экземпляры или не определяйте представления или появления представлений в методе Activity onCreate(), если только свойства (например, содержимое) действительно не могут быть определены до времени выполнения! Вместо этого **указывайте** макеты в XML.

**Упражняться**

Добавьте новый [ImageView](http://developer.android.com/reference/android/widget/ImageView.html)элемент, содержащий картинку. Обязательно укажите ее idразмер (поэкспериментируйте с разными вариантами).

Содержимое изображения следует указать в XML-ресурсе с помощью android:srcатрибута (используйте @для ссылки на drawable), но вы можете указать содержимое динамически в коде Java, если захотите изменить его позже.

*//java*

ImageView imageView = (ImageView)findViewById(R.id.img\_view);

imageView.setImageResource(R.drawable.my\_image);

*//kotlin*

**val** imageView:ImageView = findViewById<ImageView>(R.id.img\_view)

imageView.setImageResource(R.drawable.my\_image);

**2.3 Макеты**

Как упоминалось выше, [Layout](https://developer.android.com/guide/topics/ui/declaring-layout.html) — это группа Views (в частности, ViewGroup). Layout действует как контейнер для других Views, помогая структурировать элементы на экране. Все Layouts являются подклассами [ViewGroup](https://developer.android.com/reference/android/view/ViewGroup.html), поэтому вы можете использовать его документацию по наследованию, чтобы увидеть (в основном) полный список опций, хотя многие из перечисленных классов устарели в пользу более поздних, более общих/мощных опций.

**Линейная компоновка**

Вероятно, самый простой для понимания Layout — это [LinearLayout](https://developer.android.com/guide/topics/ui/layout/linear.html). Этот Layout упорядочивает дочерние Views в линию («линейно»). Все дочерние элементы располагаются в одном направлении, но вы можете указать, будет ли оно горизонтальным или вертикальным, с помощью свойства android:orientation. См. [LinearLayout.LayoutParams](https://developer.android.com/reference/android/widget/LinearLayout.LayoutParams.html) для списка всех параметров атрибутов!

* Помните: поскольку a Layout— это a — ViewGroupэто a View, вы также можете использовать все свойства, обсуждавшиеся выше, такие как отступ или цвет фона; поддержка атрибутов наследуется! (Но помните, что сами свойства не наследуются дочерними элементами: вы не можете задать для textSizeLayout и применить его ко всем дочерним Views).

Другое общее свойство, которое вы можете захотеть контролировать в LinearLayout, — это то, сколько оставшегося пространства должны занимать элементы (например, должны ли они расширяться). Это делается с помощью android:layout\_weghtсвойства. После того, как все размеры элементов рассчитаны (через их индивидуальные свойства), оставшееся пространство в Layout делится пропорционально каждого layout\_weightэлемента (что по умолчанию, 0поэтому элементы по умолчанию не получают дополнительного пространства). Подробнее см. [пример в руководстве.](https://developer.android.com/guide/topics/ui/layout/linear.html#Weight)

* *Полезный совет* : задайте элементам 0dpширину или высоту, а также 1вес, чтобы сделать все в макете одинакового размера!
* Это поведение похоже на flex-growсвойство в фреймворке [CSS Flexbox](https://css-tricks.com/snippets/css/a-guide-to-flexbox/) .

Вы также можете использовать это [android:layout\_gravity](https://developer.android.com/reference/android/widget/LinearLayout.LayoutParams.html#attr_android:layout_gravity)свойство для указания «выравнивания» элементов в макете (например, куда они «попадают»). Обратите внимание, что это свойство объявляется для отдельных дочерних Views, чтобы указать, где они расположены; android:gravityсвойство указывает, где должно быть выровнено содержимое элемента.

**Важный момент** Поскольку макеты *являются* представлениями, вы, конечно, можете вкладывать их LinearLayoutsдруг в друга! Таким образом, вы можете создавать «сетки», создавая вертикальный LinearLayout, содержащий «строки» горизонтальных LinearLayouts (содержащих представления). Как и в случае с HTML, существует множество различных вариантов для достижения любой конкретной компоновки интерфейса.

**ОтносительныйLayout**

A [RelativeLayout](https://developer.android.com/guide/topics/ui/layout/relative.html)более гибкий (и, следовательно, мощный), но может быть более сложным в использовании. В a RelativeLayoutдочерние элементы располагаются «относительно» родительского **ИЛИ***друг друга* . Все дочерние элементы по умолчанию находятся в верхнем левом углу макета, но вы можете задать им свойства from, [RelativeLayout.LayoutParams](https://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html)чтобы указать, куда они должны идти вместо этого.

Например: android:layout\_verticalCenterцентрирует View по вертикали внутри родительского элемента. android:layout\_toRightOfразмещает View справа от View с указанным идентификатором ресурса (используйте @ссылку для ссылки на View по его идентификатору):

**<TextView**

android:id="@+id/first"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="FirstString" **/>**

**<TextView**

android:id="@+id/second"

android:layout\_height="wrap\_content"

android:layout\_below="@id/first"

android:layout\_alignParentLeft="true"

android:text="SecondString" **/>**

Вам не нужно указывать и то, toRightOfи другое toLeftOf; подумайте о размещении одного элемента на экране, а затем о размещении другого элемента относительно того, что было до этого. Это может быть сложно. По этой причине автор предпочитает использовать LinearLayouts, поскольку вы всегда можете создать относительное позиционирование, используя достаточное количество LinearLayouts (и большинство макетов в конечном итоге в некотором роде становятся линейными!)

**ConstraintLayout**

[ConstraintLayout](https://developer.android.com/training/constraint-layout/index.html)— это макет, предоставляемый как часть дополнительной библиотеки поддержки, и то, что используется инструментом «Design» Android Studio (и, таким образом, является макетом по умолчанию для новых ресурсов макета). ConstraintLayoutработает концептуально аналогично RelativeLayout, в том смысле, что вы указываете расположение Views по отношению друг к другу. Однако ConstraintLayoutпредлагает более мощный набор отношений в форме *ограничений* , которые можно использовать для создания высокоадаптивных макетов. Более подробную информацию и примеры ограничений, которые можно добавить, см. [в документации по классу.](https://developer.android.com/reference/android/support/constraint/ConstraintLayout.html)

Главное преимущество заключается в ConstraintLayoutтом, что он поддерживает разработку с помощью инструмента дизайна Android Studio. Однако, поскольку этот курс фокусируется на реализации XML-файлов ресурсов, а не на использовании конкретного инструмента (который может измениться через год), мы в основном будем использовать другие макеты.

**Другие макеты**

Есть также много других макетов, хотя мы не будем рассматривать их все подробно. Все они работают схожим образом; проверьте документацию отдельного класса для получения подробной информации.

* [FrameLayout](https://developer.android.com/reference/android/widget/FrameLayout.html) — это своего рода макет «заполнителя», который содержит **один** дочерний View (второй дочерний View не будет показан). Вы можете думать об этом макете как о способе добавления простого контейнера для использования в качестве отступа и т. д. Он также очень полезен в ситуациях, когда фреймворк требует указать ресурс Layout вместо простого отдельного View.
* [GridLayout](https://developer.android.com/reference/android/widget/GridLayout.html) упорядочивает Views в Grid. Он похож на LinearLayout, но размещает элементы в сетке, а не в линии.

Обратите внимание, что это отличается от [Grid *View*](https://developer.android.com/guide/topics/ui/layout/gridview.html) , который представляет собой прокручиваемый, адаптируемый список (похожий на ListView, который обсуждается в следующей лекции).

* [TableLayout](http://developer.android.com/guide/topics/ui/layout/grid.html) действует как HTML-таблица: вы определяете TableRowмакеты, которые могут быть заполнены содержимым. Это представление обычно не используется.
* [CoordinatorLayout](https://developer.android.com/reference/android/support/design/widget/CoordinatorLayout.html) — это класс, предоставляемый как часть дополнительной библиотеки поддержки, и обеспечивающий поддержку виджетов и анимаций [Material Design](https://developer.android.com/design/material/index.html) . Подробнее см. в Лекцию 5.

**Объединение и расширение макетов**

Можно объединить несколько файлов ресурсов макета. Это полезно, если вы хотите динамически изменять, какие представления включены, или реорганизовать части макета в различные файлы XML для улучшения организации кода.

В качестве одного из вариантов вы можете *статически* включать XML-макеты в другие макеты, используя [<include>](https://developer.android.com/training/improving-layouts/reusing-layouts.html)элемент:

**<include** layout="@layout/sub\_layout"**>**

Но также возможно динамически загружать представления «вручную» (например, в коде Java) с помощью [LayoutInflator](https://developer.android.com/reference/android/view/LayoutInflater.html). Это класс, который выполняет работу по «раздуванию» (рендерингу) представлений. LayoutInflator неявно используется в setContentView()методе, но может также использоваться независимо со следующим синтаксисом:

*//java*

LayoutInflator inflator = getLayoutInflator(); *//access the inflator (called on the Activity)*

View myLayout = inflator.inflate(R.layout.my\_layout, parentViewGroup, **true**); *//to attach*

*//kotlin*

**val** inflator: LayoutInflator = getLayoutInflator(); *//access the inflator (called on the Activity)*

**val** myLayout: View = inflator.inflate(R.layout.my\_layout, parentViewGroup, **true**); *//to attach*

Обратите внимание, что мы никогда не создаем экземпляр LayoutInflator, мы просто получаем доступ к объекту, который определен как часть Activity.

Метод [inflate()](http://developer.android.com/reference/android/view/LayoutInflater.html#inflate(int,%20android.view.ViewGroup,%20boolean))принимает несколько аргументов:

* Первый параметр — это ссылка на ресурс для инфляции ( intсохраненный в Rклассе)
* Второй параметр — это a, ViewGroupкоторый будет выступать в качестве «родителя» для этого View — например, в какой макет View должен быть развернут внутри? Это может быть, nullесли контекст макета еще не существует; например, вы хотите раздуть View, но пока не отображать его на экране.
* Третий (необязательный) параметр — следует ли фактически прикрепить инфляционный View к этому родителю (если нет, родитель просто предоставляет контекст и параметры макета для использования). Если не назначать родительскому при инфляции, вы можете позже прикрепить View с помощью методов в ViewGroup(например, addView(View)).

Ручное расширение представления работает для динамической загрузки ресурсов, и мы часто будем видеть шаблоны реализации пользовательского интерфейса, использующие Inflators.

Однако для динамического создания View явная инфляция, как правило, запутанна и сложна в поддержке (работа пользовательского интерфейса должна быть полностью определена в XML, без необходимости множественных ссылок на родительские и дочерние View), поэтому не так распространена в современной разработке. Гораздо более чистым решением является использование 7 [.](https://info448.github.io/resources-and-layouts.html#fn7) A похож на [«находящийся на палубе»](https://en.wikipedia.org/wiki/On-deck) Layout: он записан в XML, но фактически не отображается, пока вы не решите раскрыть его через код Java. С помощью Android инфляция во время выполнения, но затем удаляет его из родителя (оставляя «заглушку» на его месте). Когда вы вызываете (или ) для этой заглушки, она снова присоединяется к дереву View и отображается:[ViewStub](https://developer.android.com/training/improving-layouts/loading-ondemand.html)ViewStubViewStubViewinflate()setVisible(View.VISIBLE)

*<!-- XML -->*

**<ViewStub** android:id="@+id/stub"

android:inflatedId="@+id/subTree"

android:layout="@layout/mySubTree"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content" **/>**

*//Java*

ViewStub stub = (ViewStub)findViewById(R.id.stub);

View inflated = stub.inflate();

*//kotlin*

**val** stub = findViewById<ViewStub>(R.id.stub);

**val** inflated: View = stub.inflate();

Есть много других вариантов отображения или изменения содержимого View. Просто не забудьте определить как можно больше View в XML, чтобы код Java оставался простым и отдельным.

**2.4 Входные данные**

До сих пор мы использовали некоторые базовые представления, такие как TextView, ImageView, и Button.

A Button— пример элемента [управления вводом](https://developer.android.com/guide/topics/ui/controls.html) . Это простые (читай, одноцелевые; не обязательно лишенные сложности) виджеты, которые позволяют пользователю вводить данные. Существует много таких виджетов в дополнение к Button, в основном они находятся в [android.widget](https://developer.android.com/reference/android/widget/package-summary.html)пакете. Многие соответствуют <input>элементам HTML, но Android также предоставил дополнительные виджеты.

Вы можете изменить коды лекций MainActivity, чтобы отобразить Вид R.id.input\_control\_layout, чтобы увидеть пример многих виджетов (а также демонстрацию более сложной компоновки!). Эти виджеты включают в себя:

* [Кнопка](https://developer.android.com/guide/topics/ui/controls/button.html) , виджет, который позволяет щелкнуть. Кнопки могут отображать текст, изображения или и то, и другое.
* [EditText](https://developer.android.com/reference/android/widget/EditText.html) , виджет для ввода текста пользователем. Обратите внимание, что вы можете использовать android:inputTypeсвойство для [указания типа](https://developer.android.com/training/keyboard-input/style.html) ввода, аналогичного HTML <input>.
* [Флажок](https://developer.android.com/guide/topics/ui/controls/checkbox.html) — виджет для выбора состояния «вкл.-выкл.».
* [RadioButton](https://developer.android.com/guide/topics/ui/controls/radiobutton.html) , виджет для выбора из набора вариантов. Поместите RadioButtonэлементы внутрь RadioGroupэлемента, чтобы сделать кнопки взаимоисключающими.
* [ToggleButton](https://developer.android.com/guide/topics/ui/controls/togglebutton.html) — еще один виджет для выбора состояния «включено-выключено».
* [Switch](https://developer.android.com/reference/android/widget/Switch.html) , еще один виджет для выбора состояния «вкл-выкл». Это просто ToggleButtonUI со слайдером. Он был представлен в API 14 и является «современным» способом поддержки ввода «вкл-выкл».
* [Spinner](https://developer.android.com/guide/topics/ui/controls/spinner.html) , виджет для выбора из массива вариантов, похожий на выпадающее меню. Обратите внимание, что вы должны определить варианты как ресурс (например, в strings.xml).
* [Pickers](https://developer.android.com/guide/topics/ui/controls/pickers.html) : составной элемент управления для некоторых конкретных входных данных (даты, время и т. д.). Обычно они используются во всплывающих диалоговых окнах, которые будут обсуждаться в будущей лекции.
* …и многое другое! [android.widget](http://developer.android.com/reference/android/widget/package-summary.html)Дополнительные опции см. в пакете.

Все эти элементы управления вводом в основном работают одинаково: вы определяете (создаете) их в ресурсе макета, а затем обращаетесь к ним в Java, чтобы определить поведение взаимодействия.

Существует два способа взаимодействия с элементами управления (и представлениями в целом) из кода Java:

1. Вызов **методов** View для управления им. Это представляет собой коммуникацию «извне вовнутрь» (по отношению к View).
2. Прослушивание **событий,** производимых View, и реагирование на них. Это представляет собой коммуникацию «изнутри наружу» (по отношению к View).

Пример второго, событийно-ориентированного подхода был представлен в [Лекции 1.](https://info448.github.io/introduction.html#introduction) Он включает в себя *регистрацию прослушивателя* для события (после получения ссылки на View с помощью findViewById()), а затем указание **метода обратного вызова** (путем создания экземпляра интерфейса Listener), который будет «вызван обратно» при возникновении события.

Также возможно указать метод обратного вызова в самом ресурсе XML, используя, например, android:onClickатрибут. Это значение этого атрибута должно быть *именем* метода обратного вызова:

**<Button**

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:onClick="handleButtonClick" **/>**

Метод обратного вызова объявлен в коде Java как принимающий Viewпараметр (который будет ссылкой на любой View, вызвавший событие) и возвращающий void:

*//java*

**public** void handleButtonClick(View view) { }

*//kotlin*

**fun** handleButtonClick(view: View) { }

В этом классе мы будем использовать сочетание обеих этих стратегий (определяя обратные вызовы как в Java, так и в XML).

*Мнение автора* : спорно, какой подход «лучше». Указание метода обратного вызова в коде Java помогает сохранить внешний вид и поведение раздельными и избежать введения скрытых зависимостей для ресурсов (Activity должна предоставлять требуемый обратный вызов). Однако, поскольку кнопки создаются для нажатия, не неразумно дать «имя» в ресурсе XML относительно того, что будет делать кнопка, особенно потому, что соответствующий метод Java может быть просто методом «запуска», который вызывает что-то еще. Указание обратного вызова в ресурсе XML часто может показаться более быстрым и простым, и мы будем использовать тот вариант, который лучше всего поддерживает ясность в нашем коде.

Обратные вызовы событий используются для реагирования на все виды виджетов управления вводом. CheckBoxes используют onClickобратный вызов, ToggleButtons используют onCheckedChangedи т. д. Другие общие события можно найти в [документации View](https://developer.android.com/reference/android/view/View.html#nestedclasses) и обрабатываются с помощью слушателей, таких как OnDragListener(для перетаскиваний), OnHoverListener(для событий «наведения»), OnKeyListener(для ввода пользователем) или OnLayoutChangeListener(для изменения макета).

В дополнение к прослушиванию событий, можно вызывать методы напрямую в указанных Views для доступа к их состоянию. В дополнение к общим методам View, таким как isVisible()или hasFocus(), можно напрямую запрашивать состояние предоставленного ввода. Например, isChecked()метод возвращает, отмечен ли флажок.

Это также хороший способ получить доступ к введенному содержимому из Java Code. Например, вызовите getText()элемент EditTextуправления, чтобы получить содержимое этого View.

* Для практики попробуйте вывести содержимое включенного элемента EditTextуправления при Buttonнажатии !

Между прослушиванием событий и запросом состояния мы можем полностью взаимодействовать с элементами управления вводом. Проверьте официальную документацию для получения более подробной информации о том, как использовать конкретные отдельные виджеты.

1. <https://developer.android.com/guide/topics/resources/available-resources.html>[↩](https://info448.github.io/resources-and-layouts.html#fnref2)
2. <http://developer.android.com/guide/topics/resources/providing-resources.html>[↩](https://info448.github.io/resources-and-layouts.html#fnref3)
3. <http://developer.android.com/reference/android/view/View.html>[↩](https://info448.github.io/resources-and-layouts.html#fnref4)
4. <http://developer.android.com/reference/android/view/ViewGroup.html>[↩](https://info448.github.io/resources-and-layouts.html#fnref5)
5. <http://developer.android.com/reference/android/view/View.html#lattrs>[↩](https://info448.github.io/resources-and-layouts.html#fnref6)
6. <http://developer.android.com/training/improving-layouts/loading-ondemand.html>[↩](https://info448.github.io/resources-and-layouts.html#fnref7)