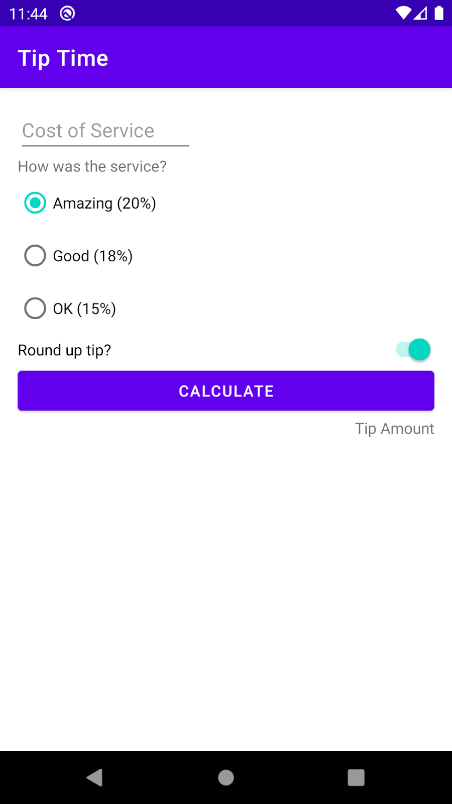
# Calculadora de Gorjeta



Você usará estes elementos de IU fornecidos pelo Android:

* EditText: para inserir e editar texto.
* TextView: para exibir um texto, como uma pergunta sobre o serviço e o valor da gorjeta.
* RadioButton: um botão de opção selecionável para cada opção de gorjeta.
* RadioGroup: para agrupar os botões de opção.
* Switch: um botão liga/desliga para definir se a gorjeta será arredondada ou não.

Criar um projeto Empty Activity

1. Para começar, crie um novo projeto Kotlin no Android Studio usando o modelo **Empty Activity**.
2. Dê o nome "deneros" ao app e defina o nível mínimo da API como 19 (KitKat). O nome do pacote é **com.example.deneros**.
3. Clique em **Finish** para criar o app.

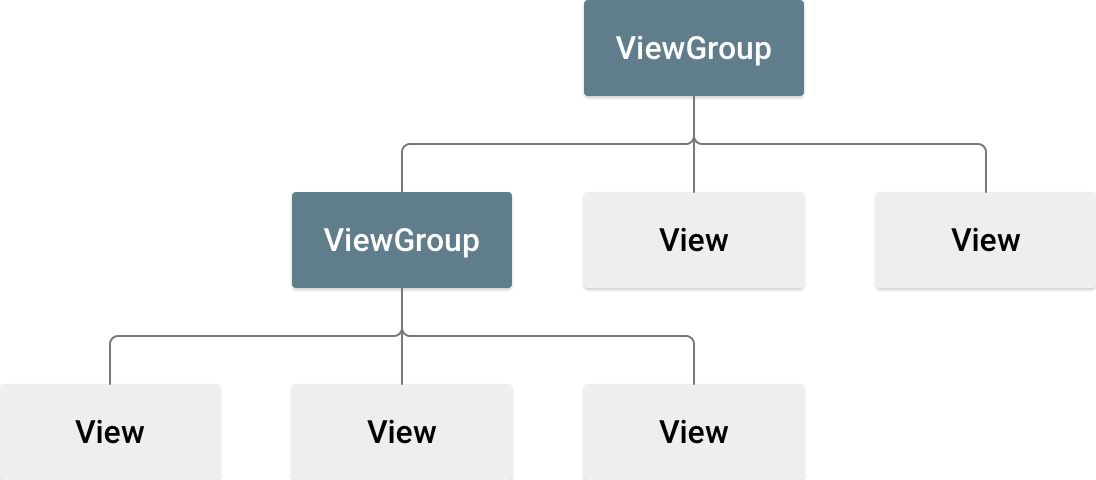
[3. Ler e entender o XML](https://developer.android.com/codelabs/basic-android-kotlin-training-xml-layouts?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-1%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-xml-layouts#2)

Em vez de usar o **Layout Editor** que você já conhece, crie o layout do aplicativo modificando o [XML](https://en.wikipedia.org/wiki/XML) que descreve a IU. Aprender a entender e modificar layouts de IU usando XML é importante para todos os desenvolvedores Android.

Você vai visualizar e editar o arquivo XML que define o layout da IU desse app. XML significa *eXtensible Markup Language*, que é uma maneira de descrever dados usando um documento de texto. Como o XML é extensível e muito flexível, ele é usado para muitas coisas, incluindo a definição do layout da IU dos apps Android. Você pode se lembrar de codelabs anteriores em que outros recursos, como strings do app, também eram definidos em um arquivo XML chamado strings.xml.

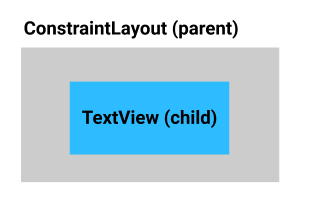
A IU para um app Android é criada como uma hierarquia de contêineres de componentes (widgets) e os layouts na tela desses componentes. Esses layouts também são componentes da IU.

É você que descreve a hierarquia de visualização dos elementos da IU na tela. Por exemplo, uma ConstraintLayout (o pai) pode conter Buttons, TextViews, ImageViews ou outras visualizações (os filhos). A ConstraintLayout é uma subclasse da ViewGroup. Ela permite posicionar ou dimensionar as visualizações filhas de forma flexível.



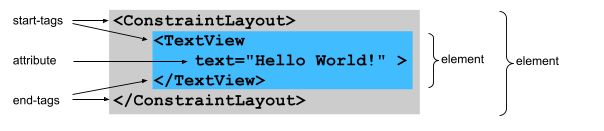
Hierarquia de contêineres de um app Android

OBSERVAÇÃO: a hierarquia de IU visível é baseada em confinamento, ou seja, um componente tem um ou mais componentes dentro dele. Isso não está relacionado à hierarquia de classes e subclasses ensinada anteriormente. Às vezes, os termos "pai" e "filho" são usados, mas o contexto aqui é que visualizações pai (grupos de visualização) contêm visualizações filhas, que por sua vez podem conter mais visualizações filhas.

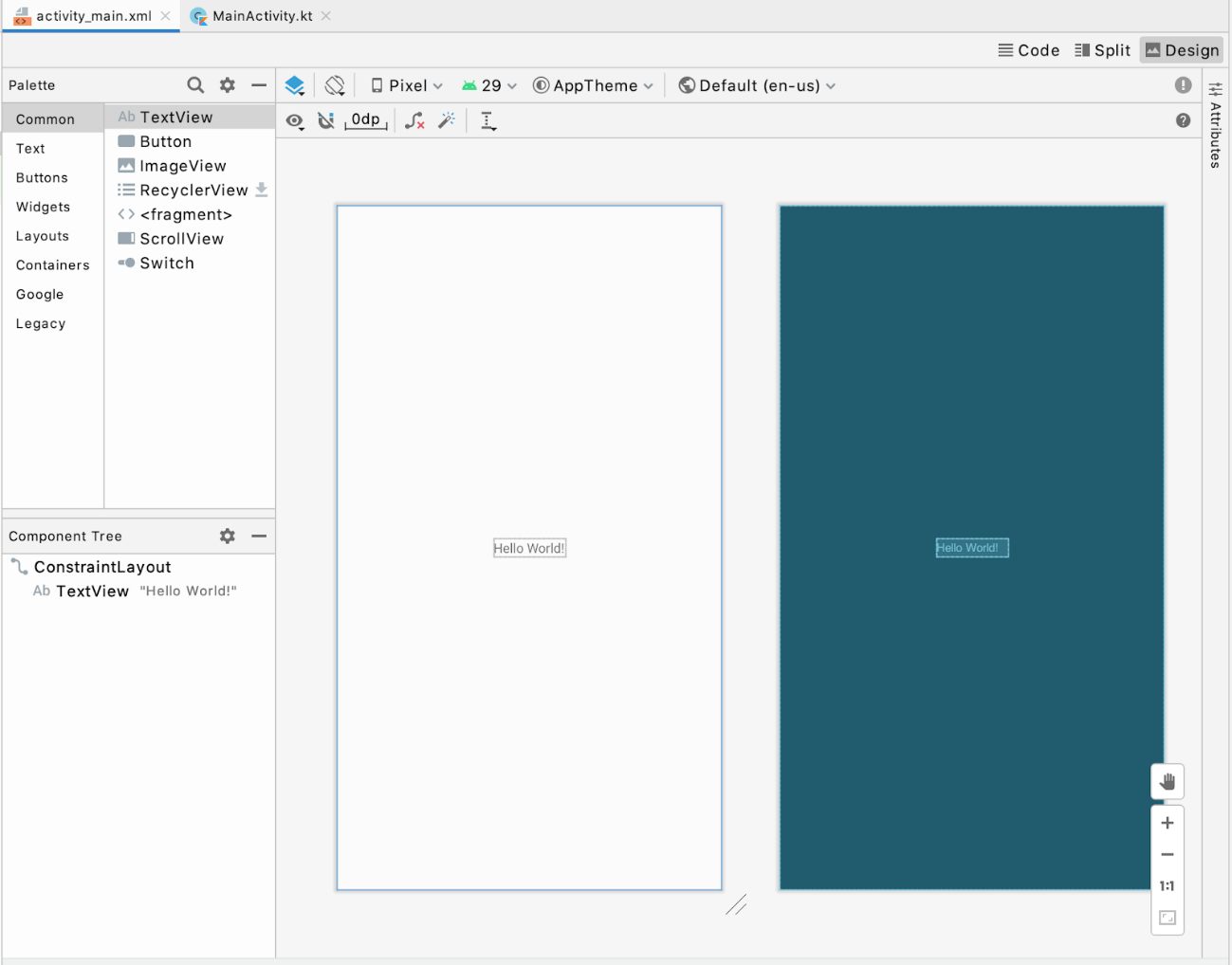


Cada elemento de IU é representado por um *elemento* XML no arquivo XML. Cada elemento começa e termina com uma tag, e cada tag começa com < e termina com >. Assim como você pode definir atributos em elementos da IU usando o **Layout Editor (design)**, os elementos XML também podem ter *atributos*. De modo simples, o XML dos elementos da IU acima pode ser algo como:

<ConstraintLayout>  
    <TextView  
        text="Hello World!">  
    </TextView>  
</ConstraintLayout>

 Vejamos um exemplo específico.

1. Abra o arquivo activity\_main.xml (**res** > **layout** > **activity\_main.xml**).
2. Você verá que o app exibe uma TextView com "Hello World!" em um ConstraintLayout, como visto em projetos anteriores criados com este modelo.



1. As opções para as visualizações **Code**, **Split** e **Design** ficam no canto superior direito do Layout Editor.
2. Selecione a visualização **Code**. 

O XML em activity\_main.xml estará assim:

<?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:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:text="Hello World!"  
        app:layout\_constraintBottom\_toBottomOf="parent"  
        app:layout\_constraintLeft\_toLeftOf="parent"  
        app:layout\_constraintRight\_toRightOf="parent"  
        app:layout\_constraintTop\_toTopOf="parent" />  
  
</androidx.constraintlayout.widget.ConstraintLayout>

Há *muito* mais elementos do que no exemplo simplificado, mas o Android Studio ajuda a tornar o XML mais legível, assim como seu código Kotlin.

1. Observe o recuo. O Android Studio faz isso automaticamente para mostrar a hierarquia dos elementos. A TextView tem um recuo porque está contida no ConstraintLayout. O ConstraintLayout é o pai, e a TextView é a filha. Os atributos de cada elemento são recuados para mostrar que fazem parte dele.
2. Observe as cores da programação. Algumas coisas estão em azul, outras em verde e assim por diante. As partes semelhantes do arquivo são desenhadas na mesma cor para ajudar você a agrupá-las. O Android Studio desenha o início e o fim das tags dos elementos usando a mesma cor. Observação: as cores usadas neste codelab podem não ser iguais as que você verá no Android Studio.

# Tags, elementos e atributos XML

Esta é uma versão simplificada do elemento TextView para você analisar algumas partes importantes:

<TextView  
    android:text="Hello World!"  
/>

A linha com <TextView é o início da tag, e a linha com /> é o fim dela. A linha com android:text="Hello World!" é um atributo da tag. Ela representa o texto que será exibido pela TextView. Essas três linhas são um atalho usado com frequência chamado *tag de elemento vazio*. Seria como se você tivesse escrito uma tag de *início* e *término* separadamente, da seguinte forma:

<TextView  
    android:text="Hello World!"  
></TextView>

Também é comum usar uma tag de elemento vazio para escrever o menor número possível de linhas e combinar o final da tag com a linha anterior. Por isso, talvez você veja uma tag de elemento vazio em duas linhas (ou até mesmo em uma linha, se ela não tiver atributos):

<!-- with attributes, two lines -->  
<TextView  
    android:text="Hello World!" />

O elemento ConstraintLayout é escrito com tags de início e término separadas, porque precisa conter outros elementos dentro dele. Veja uma versão simplificada do elemento ConstraintLayout com o elemento TextView dentro dele:

<androidx.constraintlayout.widget.ConstraintLayout>  
    <TextView  
        android:text="Hello World!" />  
</androidx.constraintlayout.widget.ConstraintLayout>

Se você quiser adicionar outra View como filha do ConstraintLayout, como um Button abaixo da TextView, ele precisa ser colocado após o final da tag TextView /> e antes do final da tag do ConstraintLayout, desta forma:

<androidx.constraintlayout.widget.ConstraintLayout>  
    <TextView  
        android:text="Hello World!" />  
    <Button  
        android:text="Calculate" />  
</androidx.constraintlayout.widget.ConstraintLayout>

# Mais sobre XML para layouts

1. Observe a tag do ConstraintLayout e veja que ela usa androidx.constraintlayout.widget.ConstraintLayout, em vez de apenas ConstraintLayout como a TextView. Isso ocorre porque o ConstraintLayout faz parte do Android Jetpack, que contém bibliotecas de código que oferecem mais funcionalidades do que a Plataforma Android principal. O Jetpack tem funcionalidades úteis que você pode aproveitar para facilitar a criação de apps. Você pode perceber que um componente de IU é parte do Jetpack porque começa com "androidx".
2. Talvez você tenha percebido as linhas que começam com xmlns:, seguido por android, app e tools.

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"

O xmlns representa o namespace do XML, e cada linha define um *esquema* ou vocabulário para atributos relacionados a essas palavras. O namespace android:, por exemplo, marca atributos definidos pelo sistema Android. Todos os atributos no XML do layout começam com um desses namespaces.

1. O espaço em branco entre elementos XML não muda o significado para o computador, mas pode facilitar a leitura do XML.

O Android Studio adicionará automaticamente espaços em branco e recuo para facilitar a leitura. Você aprenderá mais tarde sobre como usar o Android Studio para garantir que o XML siga as convenções de estilo de codificação.

1. Você pode adicionar comentários em XML, da mesma forma que faz com o código Kotlin. Use <!-- no começo e --> no fim.

<!-- this is a comment in XML -->  
  
<!-- this is a  
multi-line  
Comment.  
And another  
Multi-line comment -->

1. Observe a primeira linha do arquivo:

<?xml version="1.0" encoding="utf-8"?>

Ela indica que o arquivo é XML, mas nem todos os XML a incluem.

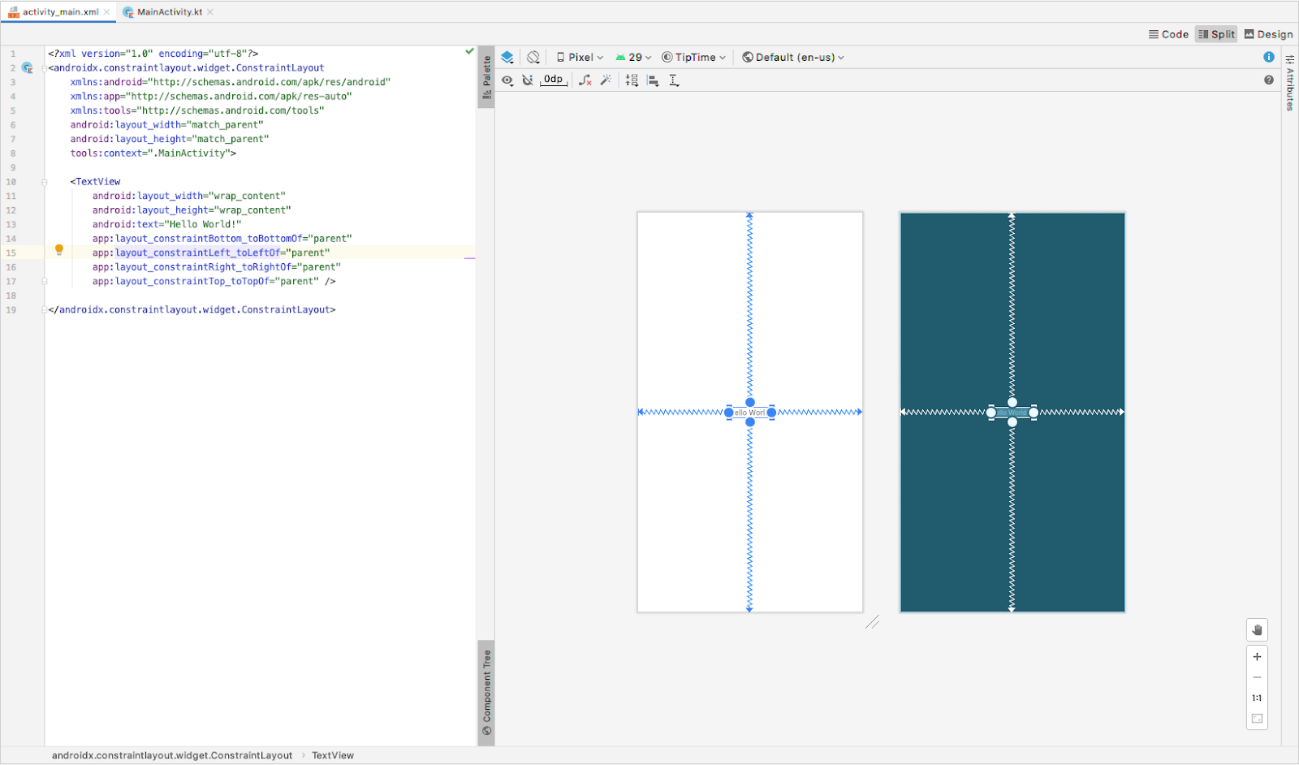
**Observação**: se houver um problema com o XML do app, o Android Studio o sinalizará desenhando o texto em vermelho. Se você mover o mouse sobre o texto em vermelho, o Android Studio mostrará mais informações sobre o problema. Se o problema não for óbvio, observe o recuo e as cores da programação, que podem dar uma ideia do que está errado.

# [Criar o layout em XML](https://developer.android.com/codelabs/basic-android-kotlin-training-xml-layouts?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-1%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-xml-layouts#3)

1. Ainda no arquivo activity\_main.xml, mude para a visualização de tela **Split** para ver o XML ao lado do **Design Editor**. O **Design Editor** permite visualizar o layout da IU.

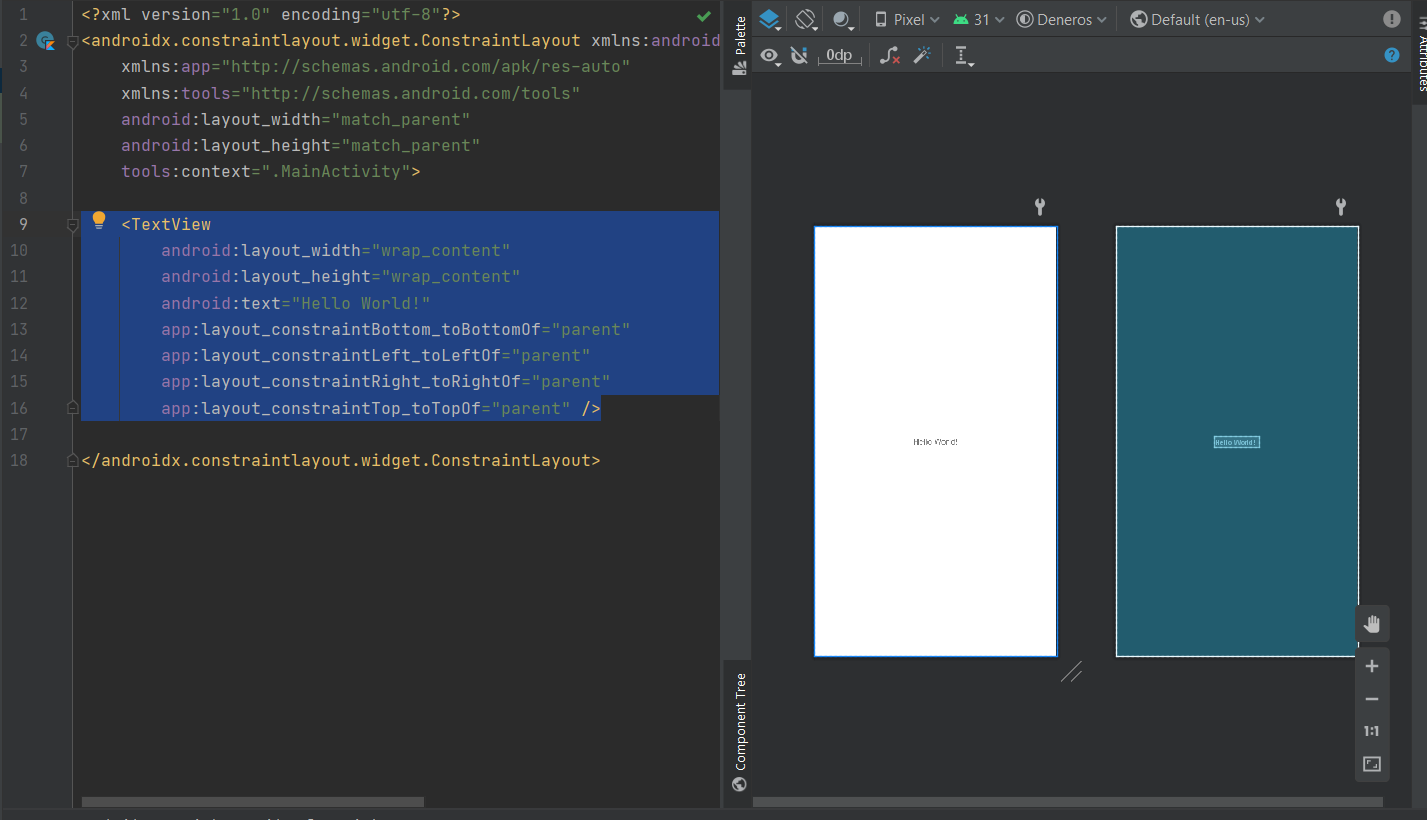


1. A visualização que você vai usar é uma preferência pessoal, mas, use a visualização **Split** para ver tanto o XML editado quanto as mudanças feitas no **Design Editor**.
2. Clique em linhas diferentes, uma abaixo do ConstraintLayout e outra abaixo da TextView. Observe que a visualização correspondente é selecionada no **Design Editor**. O inverso também funciona. Por exemplo, se você clicar na TextView no **Design Editor**, o XML correspondente será destacado.



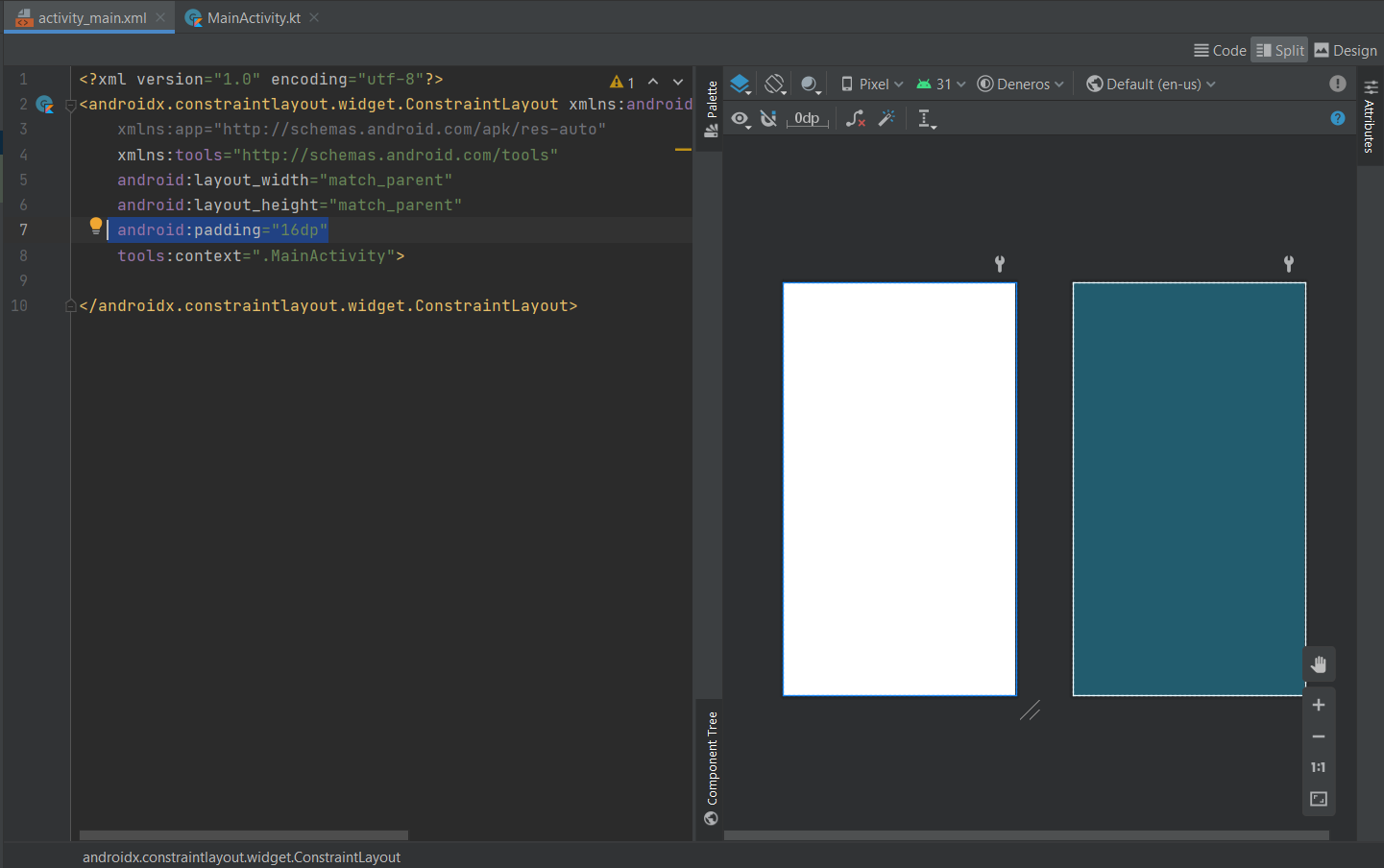
# Excluir a TextView

1. Não é preciso usar a TextView agora, então exclua-a. Certifique-se de excluir tudo desde <TextView até a tag /> de término.



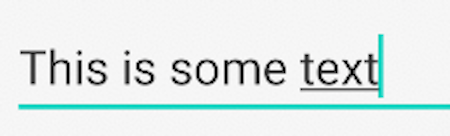
1. Adicione 16 dp de padding ao ConstraintLayout para que a IU não fique poluída na borda da tela.

O padding é semelhante às margens, mas acrescenta espaço ao interior do ConstraintLayout, em vez de adicionar espaço ao lado externo.



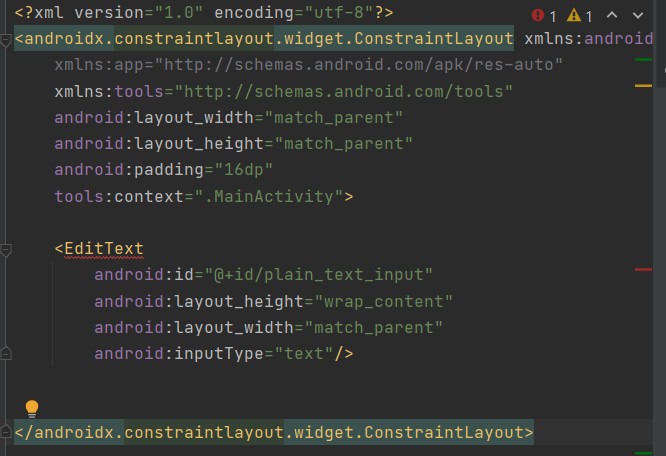
# Adicionar um campo de texto do custo do serviço

Nesta etapa, você adicionará o elemento da IU para permitir que o usuário insira o custo do serviço no app. Você usará um elemento EditText, que permite que o usuário insira ou modifique texto em um app.



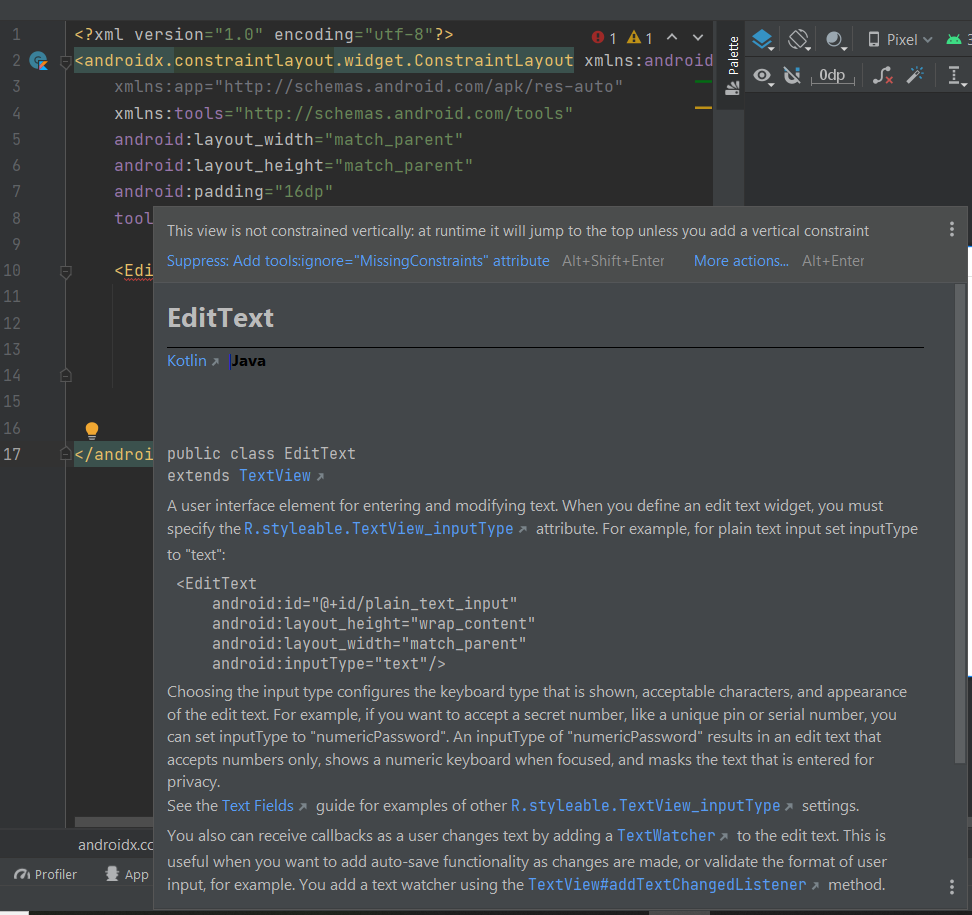
1. Veja a documentação da classe [EditText](https://developer.android.com/reference/kotlin/android/widget/EditText?hl=pt-br" \t "_blank) e analise o XML de exemplo.
2. Encontre um espaço em branco entre as tags de início e término do ConstraintLayout.
3. Copie e cole o XML da documentação nesse espaço do layout no Android Studio.

O arquivo de layout ficará assim:

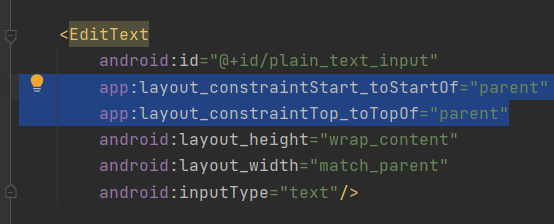


Talvez você não entenda tudo isso ainda, mas isso será explicado nas etapas a seguir.

1. Observe que a EditText está sublinhada em vermelho.
2. Passe o cursor sobre ela para ver o erro "view is not constrained", que você já deve conhecer dos codelabs anteriores. Lembre-se de que os filhos de um ConstraintLayout precisam de restrições para que o layout saiba organizá-los.



1. Adicione essas restrições à EditText para fixá-la no canto superior esquerdo do pai.



Se você estiver escrevendo em inglês ou outro idioma com escrita da esquerda para a direita (direção LTR), a borda inicial será a esquerda. Mas alguns idiomas, como o árabe, são escritos da direita para a esquerda (direção RTL), de modo que a borda inicial será a direita. É por isso que a restrição usa o elemento "start", para que funcione em direções LTR ou RTL. Da mesma forma, as restrições usam "end" em vez de "right".

# Revisar os atributos da EditText

Verifique todos os atributos EditText que você colou para garantir que eles funcionem quanto ao modo como serão usados no seu app.

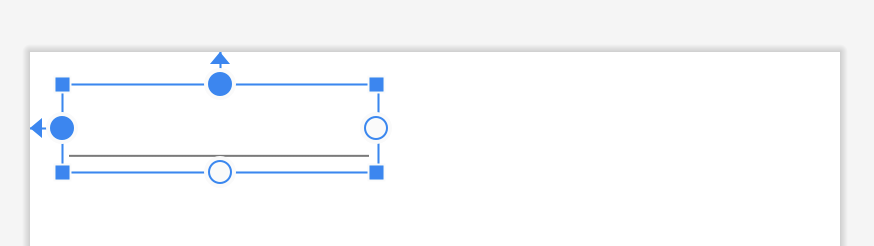
1. Localize o atributo id, que está definido como @+id/plain\_text\_input.
2. Mude o atributo id para um nome mais adequado, @+id/cost\_of\_service.

**Observação**: um ID de recurso é um nome de recurso exclusivo para o elemento. Quando você adiciona uma **View** ou outro recurso com o **Layout Editor**, o Android Studio atribui IDs de recurso automaticamente para eles. Ao digitar o XML manualmente, você precisa declarar o ID do recurso explicitamente. Os novos IDs de visualização no arquivo XML precisam ser definidos com o prefixo **@+id**, que instrui o Android Studio a adicionar o ID como o novo ID do recurso.

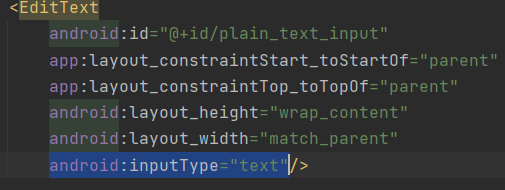
Escolha nomes descritivos para os recursos, de modo que você saiba ao que eles se referem. Os nomes precisam ser escritos em letras minúsculas, e as palavras precisam ser separadas com um sublinhado.

Ao fazer referência aos IDs de recurso no código do app, use **R.<type>.<name>**, por exemplo, **R.string.roll**. Para IDs de **View**, o **<type>** é **id**, por exemplo: **R.id.button**.

1. Veja o atributo layout\_height. Ele é definido como wrap\_content, o que significa que a altura será igual a do conteúdo dentro dele. Não há problemas nisso, porque haverá somente uma linha de texto.
2. Veja o atributo layout\_width. Ele é definido como match\_parent, mas não é possível definir match\_parent em um filho do ConstraintLayout. Além disso, o campo de texto não precisa ser tão largo. Configure-o com uma largura fixa de 160dp, que oferece bastante espaço para o usuário inserir o custo do serviço.

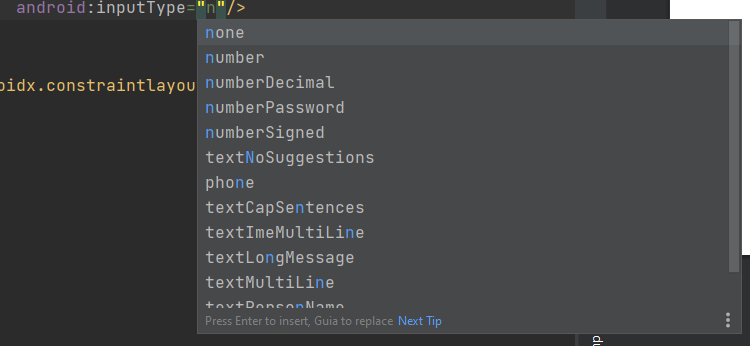


1. Observe o novo atributo inputType. O valor dele é "text", o que significa que o usuário pode digitar qualquer caractere de texto no campo na tela (caracteres alfabéticos, símbolos etc.).



No entanto, você quer que eles insiram apenas números na EditText, porque o campo representa um valor monetário.

1. Apague a palavra text, mas mantenha as aspas.
2. Digite number no lugar dela. Após digitar a letra "n", o Android Studio mostrará uma lista de possíveis opções que incluem "n".

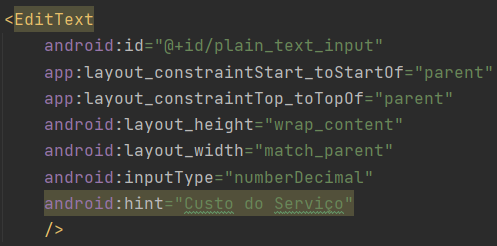


1. Escolha numberDecimal, o que limita os valores a números com um ponto decimal.

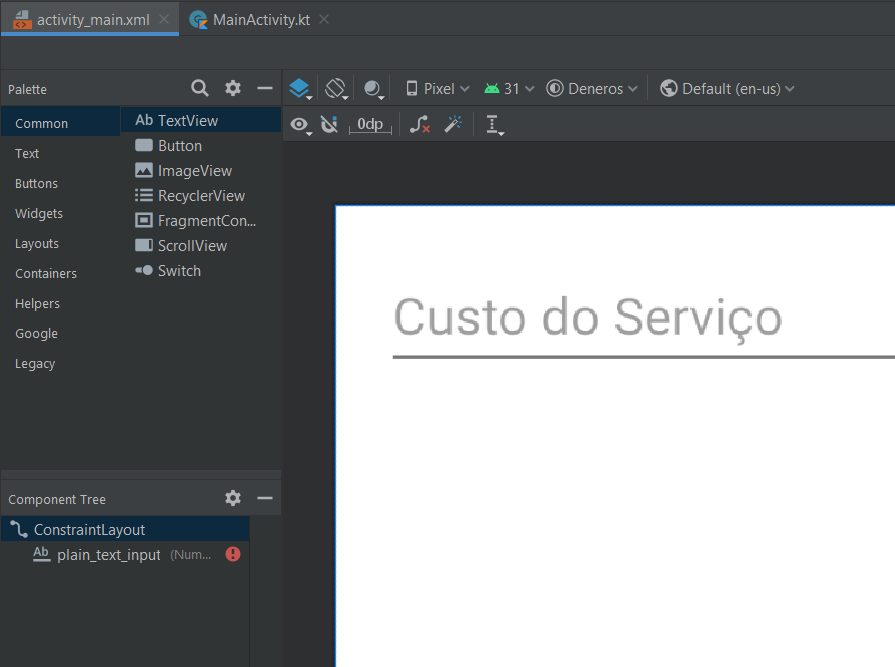
Para ver outras opções de tipos de entrada, consulte [Especificar o tipo de método de entrada](https://developer.android.com/training/keyboard-input/style?hl=pt-br) na documentação para desenvolvedores.

Há mais uma mudança a ser feita, porque é útil mostrar uma dica sobre o que o usuário precisa inserir nesse campo.

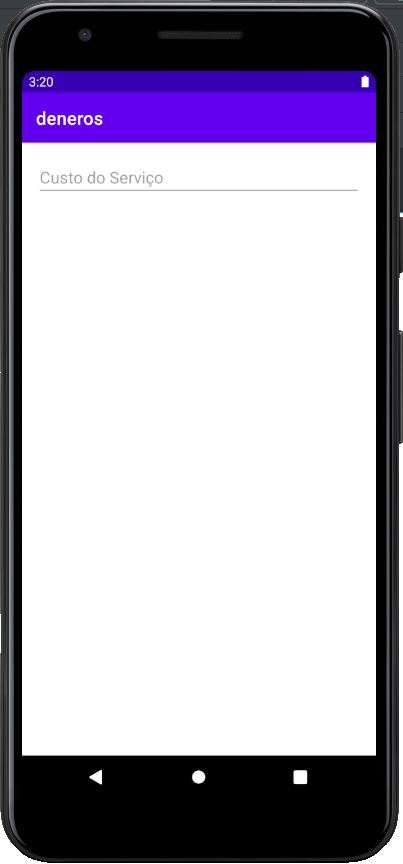
1. Adicione um atributo hint à EditText descrevendo o que o usuário precisa inserir no campo.



A atualização será exibida no **Design Editor** também.



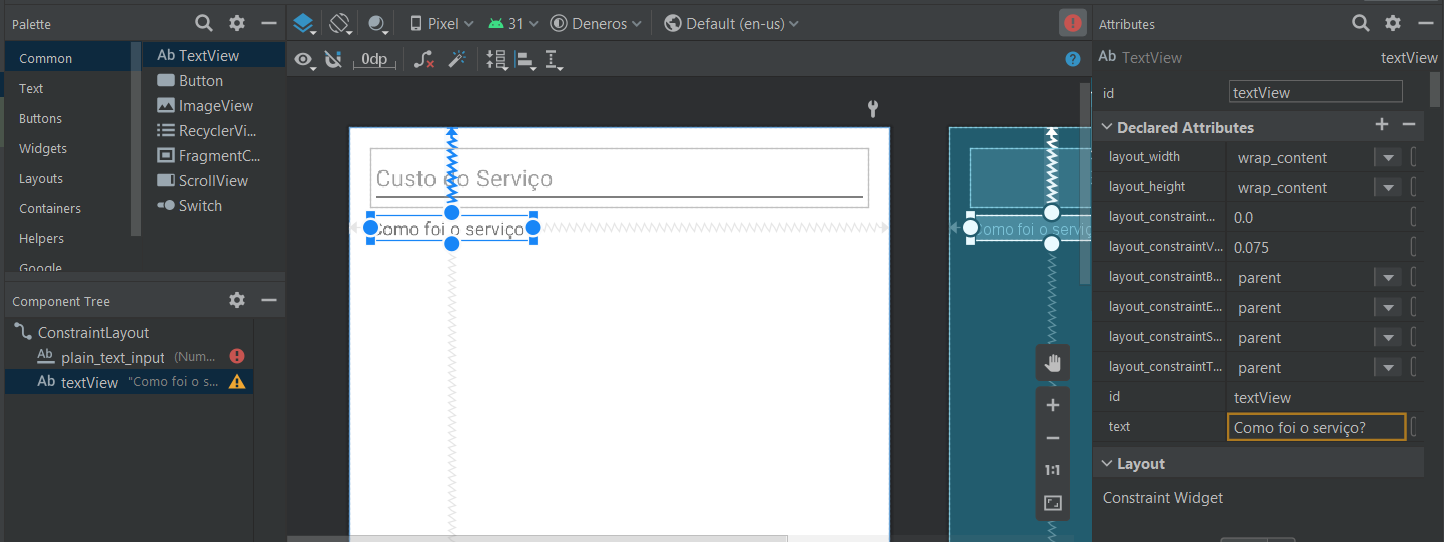
Execute o app no emulador. Ele ficará assim:



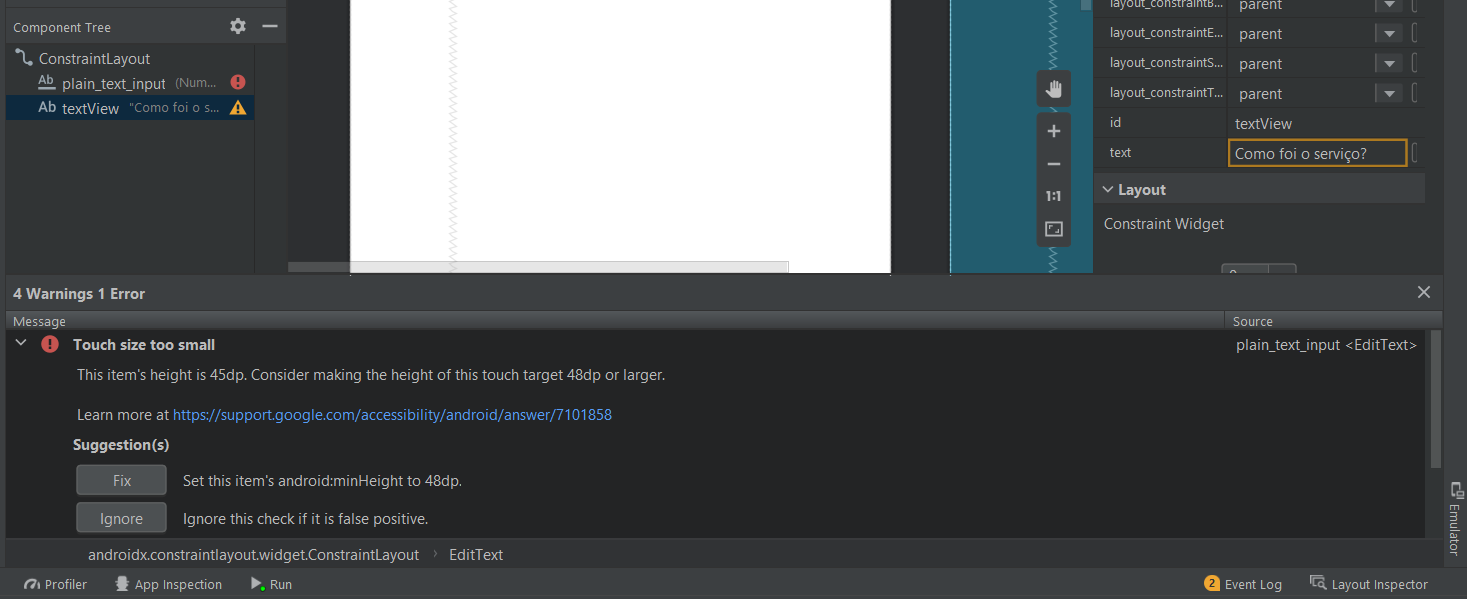
Muito bem! Ele ainda não tem muitas funcionalidades, mas você já tem uma boa base e aprendeu a editar um XML.

# Adicionar pergunta sobre o serviço

Nesta etapa, você adicionará uma TextView para a pergunta "Como foi o serviço?".



Vamos seguir algumas sugestões



Clique em fix



# [Adicionar opções de gorjeta](https://developer.android.com/codelabs/basic-android-kotlin-training-xml-layouts?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-1%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-xml-layouts#4)

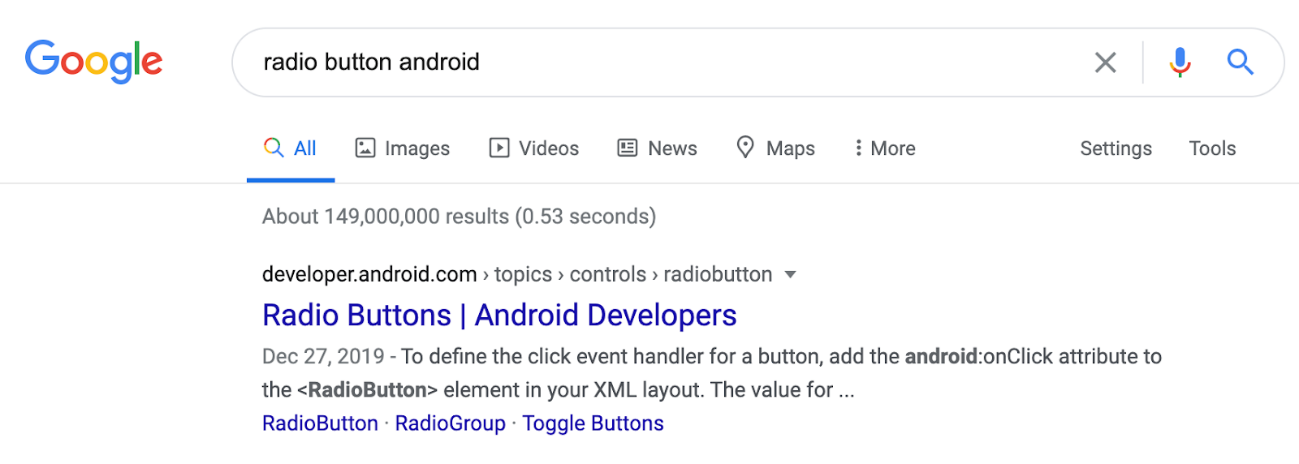
Em seguida, você adicionará botões de opção para as diferentes opções de gorjeta que o usuário poderá escolher.

Há três opções:

* Incrível (20%)
* Boa (18%)
* Regular (15%)

Se você não souber como fazer isso, faça uma pesquisa no Google. Essa é uma ótima ferramenta que os desenvolvedores usam quando encontram problemas.

1. Pesquise radio button android no Google. O principal resultado é um guia do site para desenvolvedores Android sobre como usar os botões de opção. Perfeito!

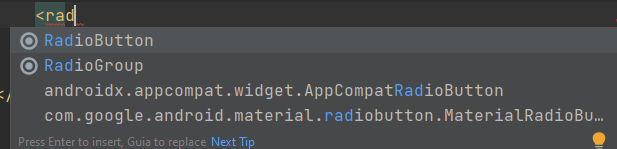


1. Dê uma olhada no [guia de botões de opção](https://developer.android.com/guide/topics/ui/controls/radiobutton?hl=pt-br#kotlin).

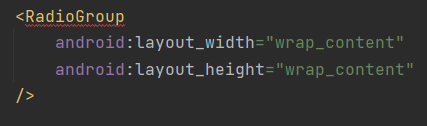
Ao ler a descrição, você pode confirmar se pode usar um elemento de IU RadioButton no seu layout para cada botão de opção necessário. Além disso, você também precisa agrupar os botões de opção em um RadioGroup porque apenas uma opção pode ser selecionada por vez.

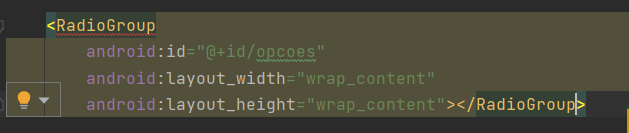
Há um XML que se encaixa nas suas necessidades. Continue lendo e veja como o RadioGroup é a visualização pai e os RadioButtons são as visualizações filhas dele.

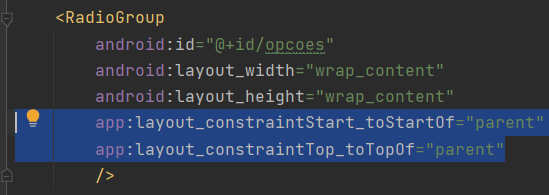
1. Volte ao layout no Android Studio para adicionar RadioGroup e RadioButton ao app.
2. Depois do elemento TextView, mas ainda dentro do ConstraintLayout, comece a digitar <RadioGroup. O Android Studio fornecerá sugestões úteis para ajudar você a completar o XML.



1. Feche a tag de início com />.
2. Defina os atributos layout\_width e layout\_height do RadioGroup como wrap\_content.

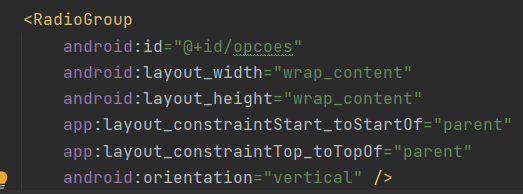


1. Adicione um ID de recurso definido como @+id/tip\_options.
2. O Android Studio adicionará o </RadioGroup>. 
3. Adicione os constraintLayout dessa forma:

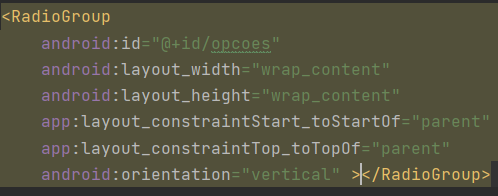


1. Restrinja o RadioGroup abaixo da pergunta sobre o serviço (verticalmente) e no início do pai (horizontalmente).
2. Defina o atributo android:orientation como vertical. Se você quiser usar RadioButtons em uma linha, defina a orientação como horizontal.

O XML do RadioGroup ficará assim:

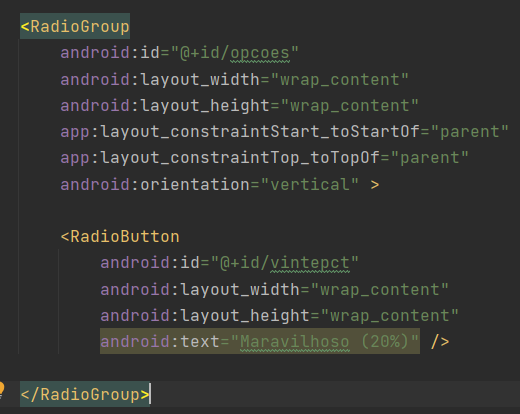


Para adicionar os radiobuttons vamos mudar a forma como a tag fecha, apague então o /> e digite apenas > e veja o que acontece

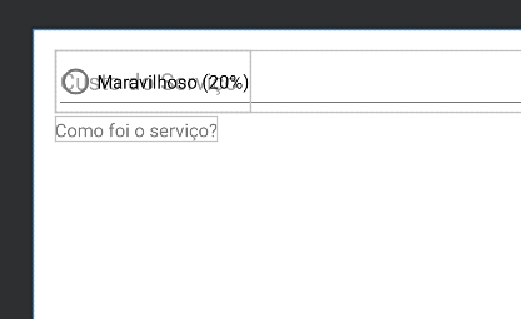


Adicionar RadioButtons

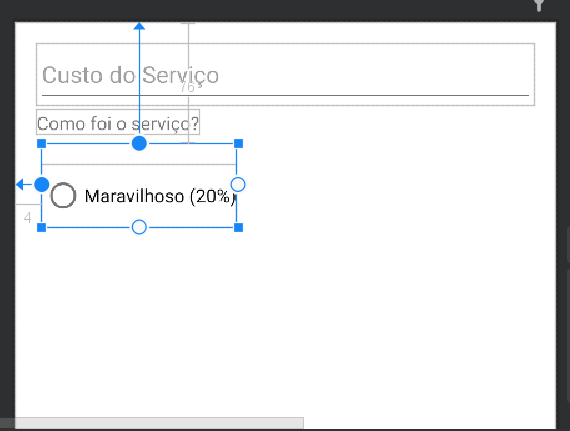
1. Após o último atributo do RadioGroup, mas antes da tag de término </RadioGroup>, adicione um RadioButton.
2. Configure a layout\_width e a layout\_height como wrap\_content.
3. Atribua um ID de recurso de @+id/option\_twenty\_percent ao RadioButton.
4. Defina o texto como Amazing (20%).
5. Feche a tag com />.



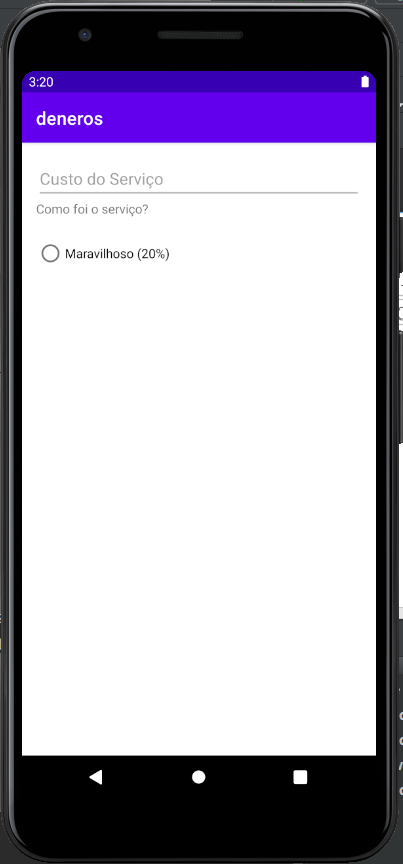
Note que o radiobutton esta em cima das outras coisas



Arraste manualmente e alinhe da forma que julga melhor, segue uma sugestão:



E teremos como resultado:

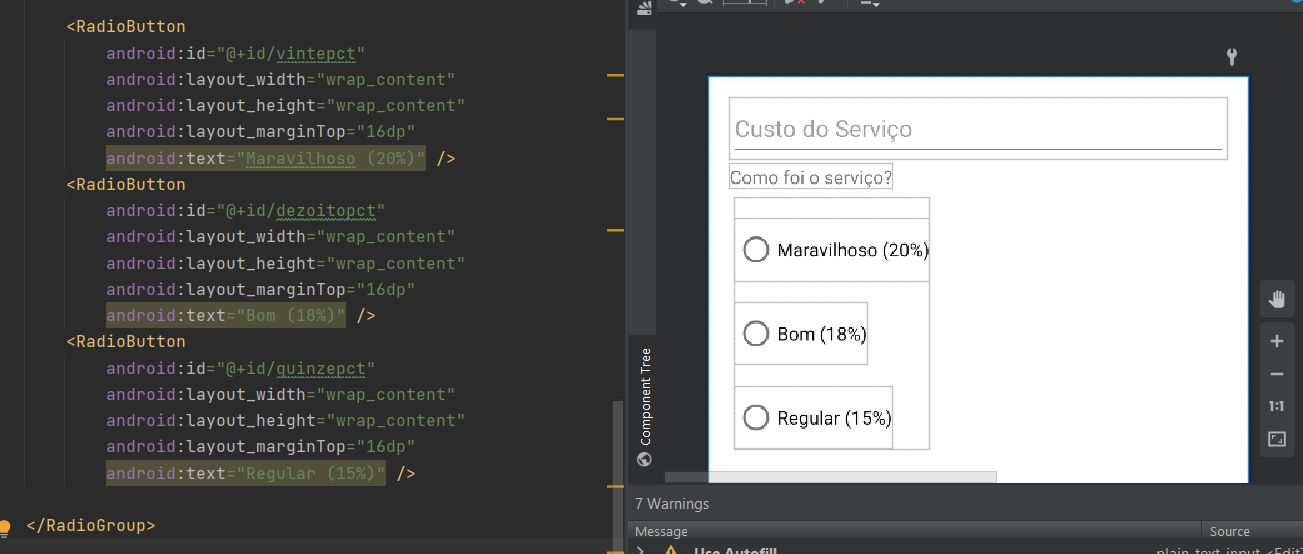


Até o momento nosso código esta assim:

<?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"  
 android:padding="16dp"  
 tools:context=".MainActivity">  
  
 <EditText  
 android:id="@+id/plain\_text\_input"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:hint="Custo do Serviço"  
 android:inputType="numberDecimal"  
 android:minHeight="48dp"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />  
  
 <TextView  
 android:id="@+id/textView"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Como foi o serviço?"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintHorizontal\_bias="0.0"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 app:layout\_constraintVertical\_bias="0.075" />  
  
 <RadioGroup  
 android:id="@+id/opcoes"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginStart="4dp"  
 android:layout\_marginTop="76dp"  
 android:orientation="vertical"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent">  
  
 <RadioButton  
 android:id="@+id/vintepct"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginTop="16dp"  
 android:text="Maravilhoso (20%)" />  
  
 </RadioGroup>  
  
  
</androidx.constraintlayout.widget.ConstraintLayout>

Agora que você adicionou um RadioButton, pode modificar o XML para adicionar mais dois botões de opção a Good (18%) e Okay (15%)?

O XML para RadioGroup e RadioButtons ficará assim:

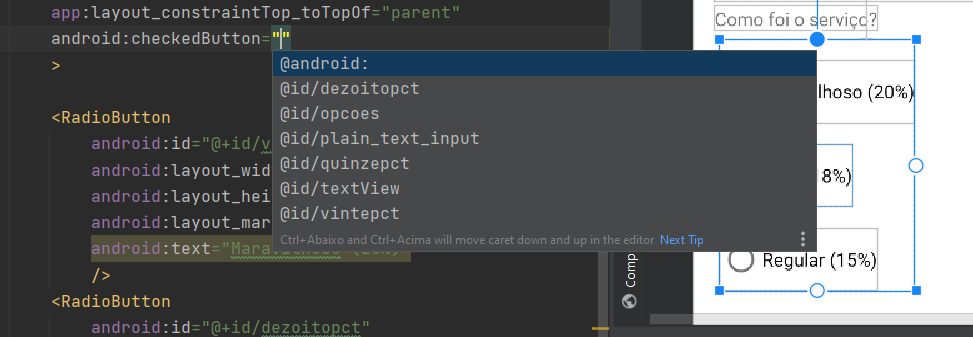


# Adicionar uma seleção padrão

No momento, nenhuma das opções de gorjeta está selecionada. É uma boa ideia selecionar uma das opções de botão como padrão.

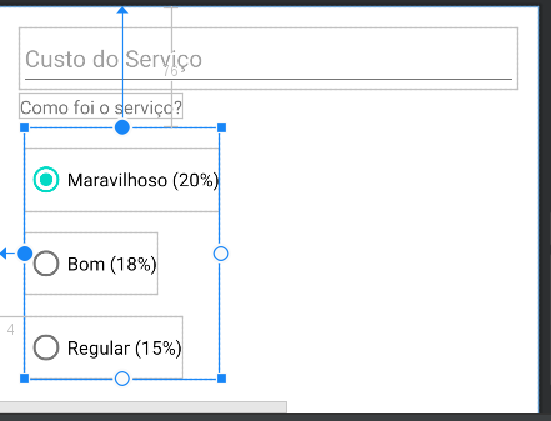
Há um atributo no RadioGroup em que é possível especificar qual botão ficará marcado inicialmente. Ele se chama checkedButton e é definido como o ID do recurso do botão de opção que você quer selecionar.

1. No RadioGroup, defina o atributo android:checkedButton para @id/vintepct.



android:checkedButton="@id/vintepct"

O layout foi atualizado no **Design Editor**. A opção de 20% de gorjeta foi selecionada por padrão. Agora o app está começando a se parecer com uma calculadora de gorjetas.



# Adicionar um Switch para arredondar a gorjeta

Em seguida, você usará um widget [Switch](https://developer.android.com/reference/kotlin/android/widget/Switch?hl=pt-br) para permitir que o usuário selecione "sim" ou "não" para arredondar a gorjeta.

Você quer que o Switch tenha a largura do pai, então pode considerar que a largura precisa ser definida como match\_parent. Conforme visto anteriormente, não é possível definir match\_parent nos elementos da IU em um ConstraintLayout. Em vez disso, é necessário restringir as bordas de início e fim da visualização e definir a largura como 0dp. Definir a largura como 0dp instrui o sistema a não calcular a largura, apenas tentar manter as restrições que estão na visualização.

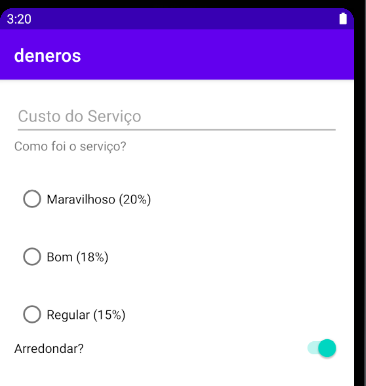
**Observação**: não é possível usar **match\_parent** para nenhuma visualização do **ConstraintLayout**. Em vez disso, use **0dp**, o que significa que as restrições serão igualadas.

1. Adicione um elemento Switch após o XML do RadioGroup.
2. Conforme mencionado acima, defina a layout\_width como 0dp..
3. Configure a layout\_height como wrap\_content Isso fará com que a visualização Switch tenha a mesma altura do conteúdo dela.
4. Defina o atributo id como @+id/round\_up\_switch.
5. Defina o atributo text como Round up tip?. Ele será usado como um rótulo para o Switch.
6. Restrinja a borda inicial do Switch à borda inicial do pai e a borda final ao fim do pai.
7. Restrinja a parte superior do Switch à parte inferior de tip\_options.
8. Feche a tag com />.

Seria bom se o interruptor fosse ativado por padrão. Há um atributo para isso, android:checked, em que os valores possíveis são true (ativado) ou false (desativado).

1. Defina o atributo android:checked como true.

Juntando tudo isso, o XML do elemento Switch ficará assim:

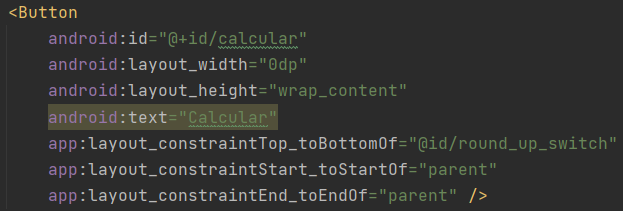


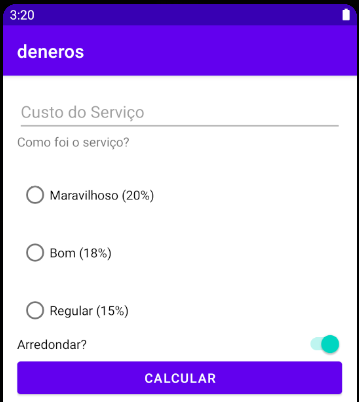
# Adicionar o botão "Calcular"

Em seguida, você adicionará um Button para que o usuário possa pedir que a gorjeta seja calculada. Você quer que o botão seja tão largo quanto o pai, então as restrições e a largura horizontais são as mesmas do Switch.

1. Adicione um Button depois do Switch.
2. Defina a largura como 0dp, como você fez no Switch.
3. Defina a altura como wrap\_content.
4. Use um ID de recurso de @+id/calculate\_button, com o texto "Calculate".
5. Restrinja a borda superior do Button à borda inferior do Switch **Arredondar gorjeta?**Switch.
6. Restrinja a borda inicial do botão à borda inicial do pai e a final ao fim do pai.
7. Feche a tag com />.

Veja como o XML para o Button **Calcular** ficará.

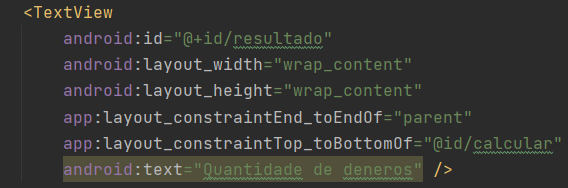




# Adicionar o resultado da gorjeta

Você está quase terminando o layout! Nesta etapa, você adicionará uma TextView para o resultado da gorjeta, colocando-a abaixo do botão **Calcular** e alinhada com o fim em vez do início, como nos outros elementos da IU.

1. Adicione uma TextView com um ID de recurso chamado quantidade de deneros e o texto Tip Amount.
2. Restrinja a borda final da TextView à borda final do pai.
3. Restrinja a borda superior da visualização à borda inferior do botão **Calcular**.





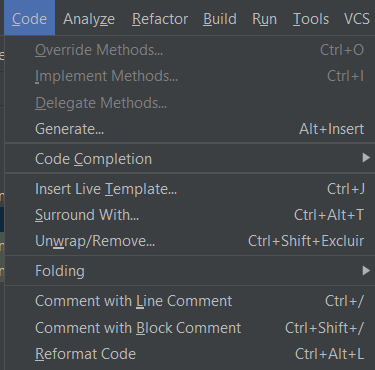
Bom trabalho, principalmente se esta for a primeira vez que você está trabalhando com XML.

Observe que o app pode não ser exatamente igual à captura de tela, porque os modelos podem ter sido mudados em versões mais recentes do Android Studio. O botão **Calcular** ainda não faz nada, mas é possível digitar o custo, selecionar a porcentagem da gorjeta e alternar a opção para arredondá-la ou não. Você fará o botão **Calcular** funcionar no próximo codelab. Não se esqueça de voltar.

Reformatar o XML

O Android Studio oferece várias ferramentas para organizar seu código e garantir que ele siga as convenções de programação recomendadas.

1. No arquivo activity\_main.xml, selecione **Edit > Select All**.
2. Selecione **Code > Reformat Code**.



Isso garante que o recuo seja consistente e pode reordenar parte do XML de elementos da IU para que agrupe itens, por exemplo, colocando todos os atributos android: em um único elemento.

[9. Resumo](https://developer.android.com/codelabs/basic-android-kotlin-training-xml-layouts?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-1%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-xml-layouts#8)

* O XML (eXtensible Markup Language) é uma maneira de organizar textos e é composto por tags, elementos e atributos.
* Use o XML para definir o layout de um app Android.
* Use a classe EditText para permitir que o usuário insira ou edite texto.
* Uma EditText pode ter uma dica para informar ao usuário o que é esperado nesse campo.
* Especifique o atributo android:inputType para limitar o tipo de texto que o usuário pode inserir em um campo EditText.
* Crie uma lista de opções exclusivas com RadioButtons, agrupadas em um RadioGroup.
* Um RadioGroup pode ser vertical ou horizontal, e você pode especificar qual RadioButton será selecionado inicialmente.
* Use um Switch para permitir que o usuário alterne entre duas opções.
* É possível adicionar um rótulo a um Switch sem usar uma TextView separada.
* Cada filho de um ConstraintLayout precisa ter restrições verticais e horizontais.
* Use as restrições de "início" e "término" para lidar com os idiomas com leitura da esquerda para a direita (LTR) e da direita para a esquerda (RTL).
* Os nomes dos atributos de restrição seguem o formato layout\_constraint<Source>\_to<Target>Of.
* Para definir uma View com a largura máxima do ConstraintLayout, restrinja o início e o final dela como o início e o fim do elemento pai e defina a largura como 0 dp.