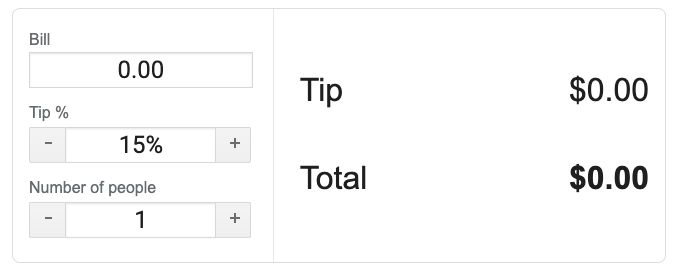
[2. Iniciar o projeto](https://developer.android.com/codelabs/basic-android-kotlin-training-xml-layouts?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-1%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-xml-layouts"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

Fonte: <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#1>

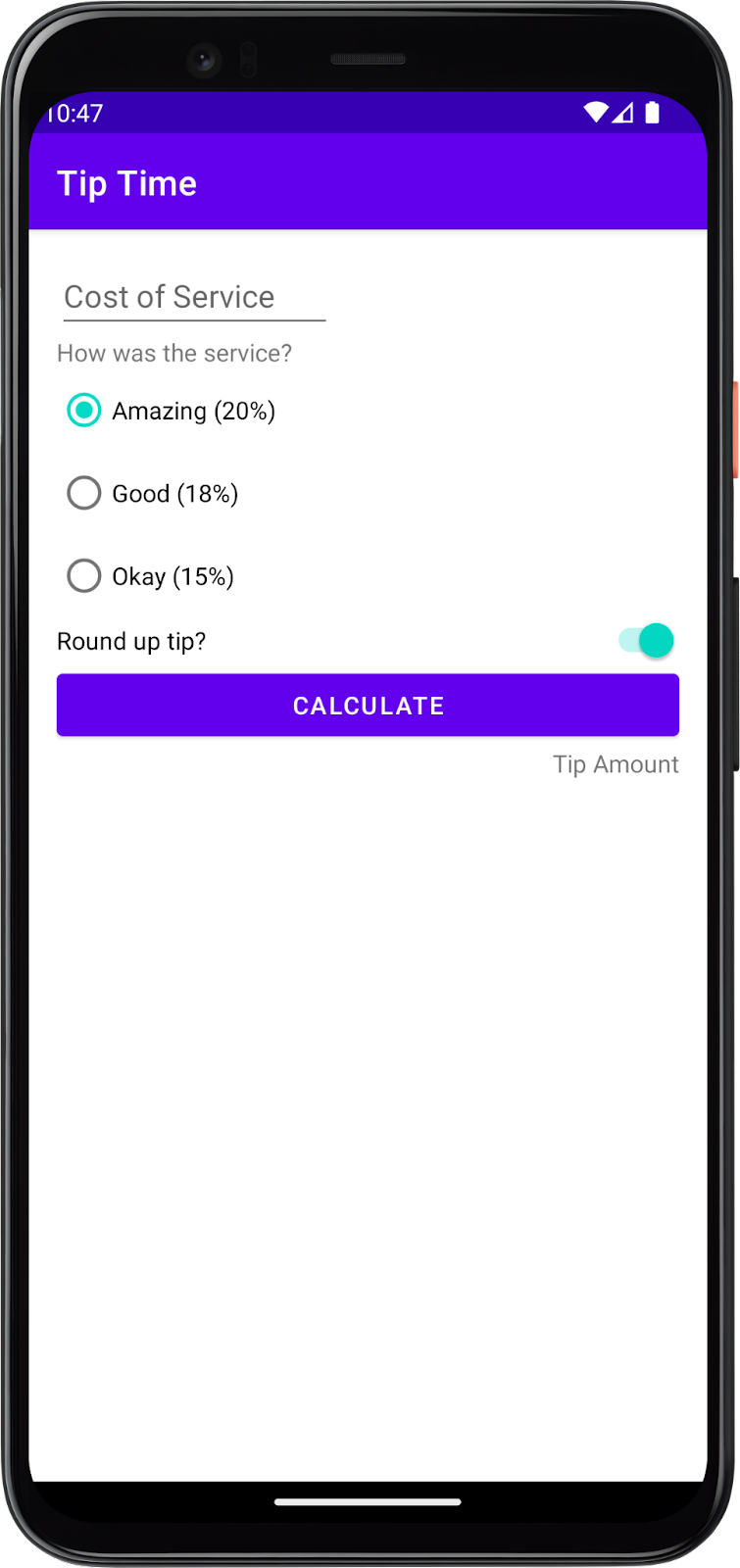
Confira a calculadora de gorjeta no Google: [https://www.google.com/search?q=tip+calculator](https://www.google.com/search?q=tip+calculator&hl=pt-br)



Neste codelab, você vai criar uma versão simples de uma calculadora de gorjetas como um app Android.

Os desenvolvedores geralmente trabalham assim: primeiro deixam pronta uma versão simples do app, parcialmente funcional (mesmo que não tenha uma aparência muito boa), e depois a tornam totalmente funcional e com um visual mais elaborado.

No fim deste codelab, seu app de calculadora de gorjetas vai ficar assim:

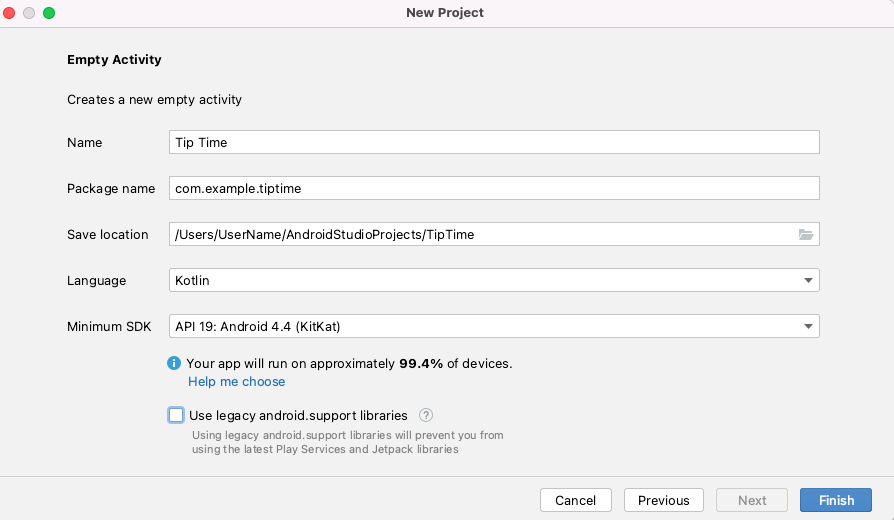


Você vai usar estes elementos de IU fornecidos pelo Android:

* EditText: para inserir e editar texto.
* TextView: para mostrar 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 ativar/desativar 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 "Tip Time" ao app e defina o nível mínimo da API como 19 (KitKat). O nome do pacote é **com.example.tiptime**.



1. Clique em **Finish** para criar o app.

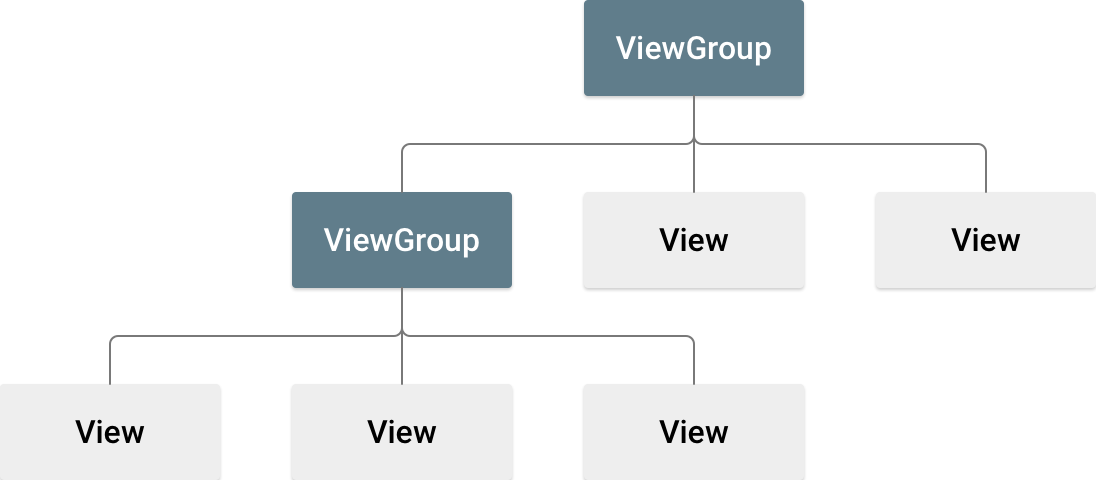
[3. Ler e entender o XML](https://developer.android.com/codelabs/basic-android-kotlin-training-xml-layouts?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-1%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-xml-layouts#2)

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) (link em inglês) 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.

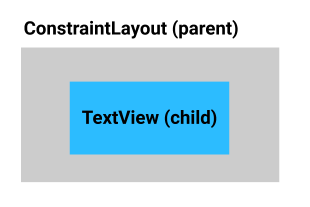
A IU para um app Android é criada como uma hierarquia de contêineres de componentes (widgets) e os layouts na tela desses componentes. Esses layouts também são componentes da IU.

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



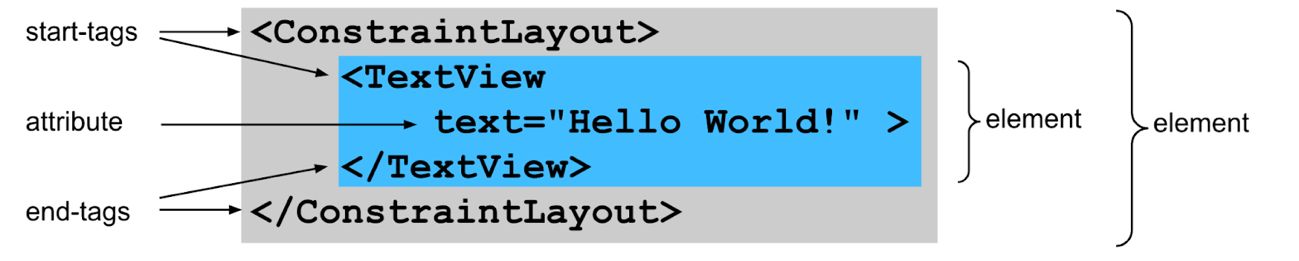
Hierarquia de contêineres de um app Android

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 "mãe" e "filha" são usados, mas o contexto aqui é que visualizações mãe (grupos de visualização) contêm visualizações filhas, que por sua vez podem conter mais visualizações filhas.



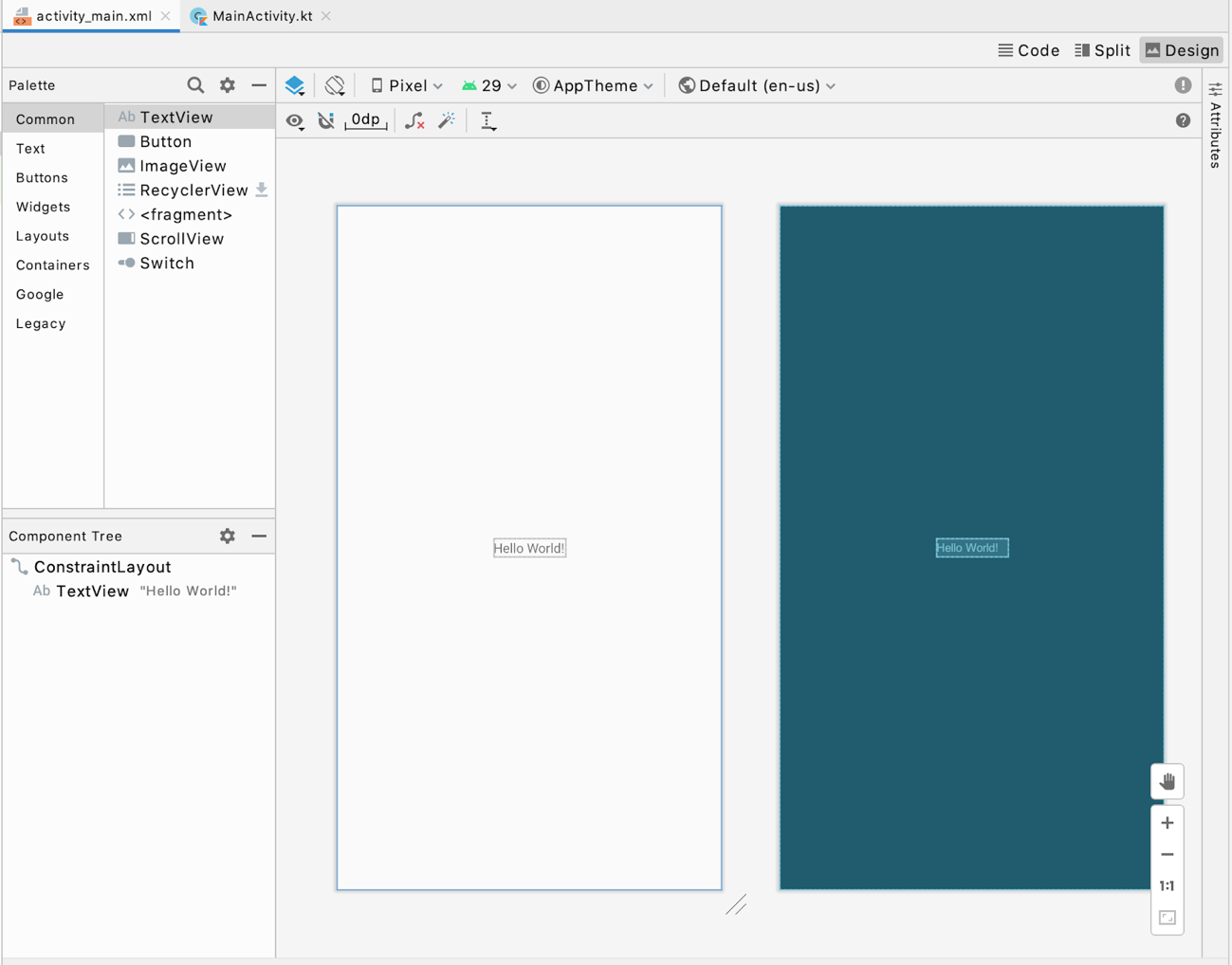
Cada elemento de IU é representado por um *elemento* XML no arquivo XML. Cada elemento começa e termina com uma tag, e cada tag começa com < e termina com >. Assim como você pode definir atributos em elementos da IU usando o **Layout Editor (design)**, os elementos XML também podem ter *atributos*. De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

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



Vamos conferir um exemplo real.

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



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



O XML em activity\_main.xml vai ficar 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 mostrado 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 vai aprender em seguida 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.

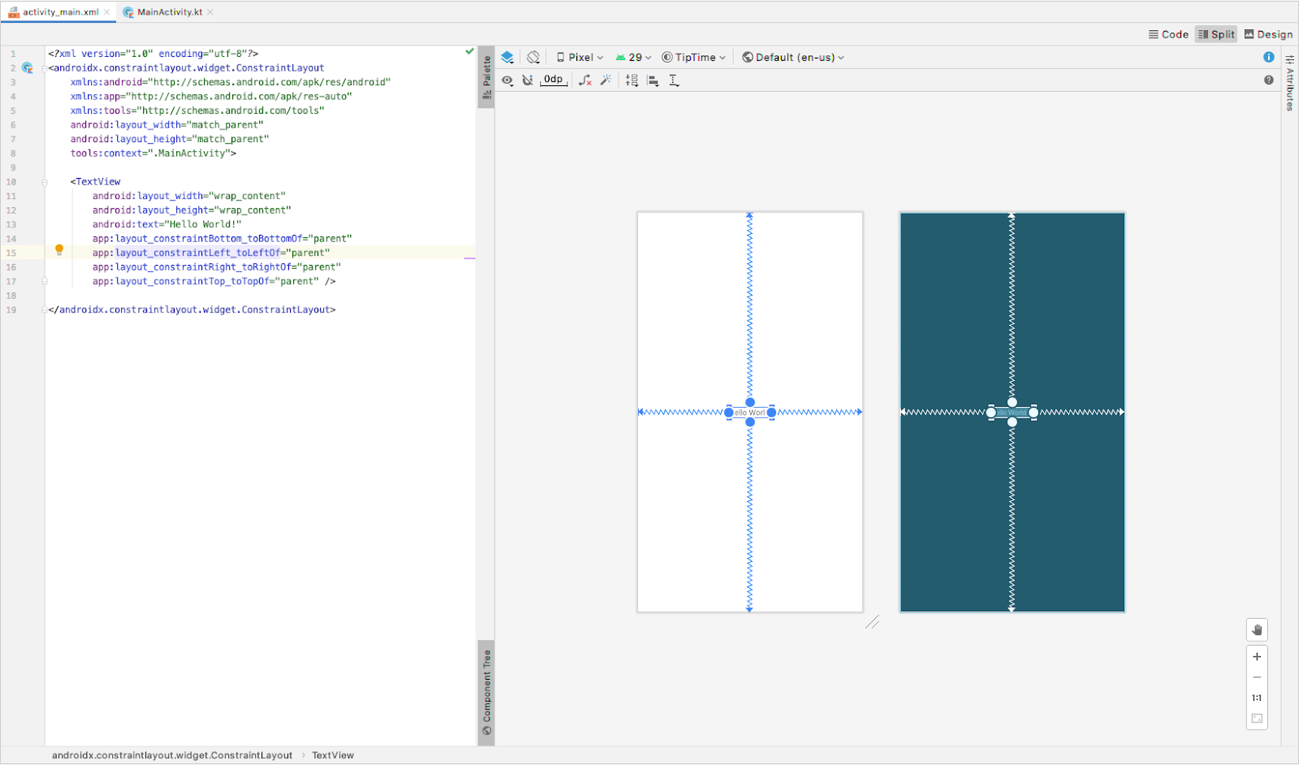
**Observação**: o Android Studio sinaliza qualquer problema que possa haver com o XML do app com texto em vermelho. Se você mover o mouse sobre o texto em vermelho, o Android Studio vai mostrar mais informações sobre o problema. Se o problema não for óbvio, observe o recuo e as cores da programação, que podem dar uma ideia do que está errado.

[4. 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, neste codelab, use a visualização **Split** para mostrar 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 vai ser 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.

<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" />

Só o ConstraintLayout será mantido no arquivo:

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

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

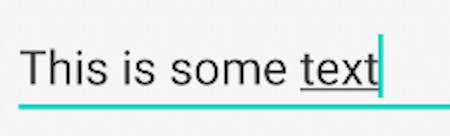
O padding é semelhante às margens, mas acrescenta espaço ao interior do ConstraintLayout, em vez de adicionar espaço ao lado externo.

<androidx.constraintlayout.widget.ConstraintLayout  
    ...  
    android:layout\_width="match\_parent"  
    android:layout\_height="match\_parent"  
    android:padding="16dp"  
    tools:context=".MainActivity">

**Observação**: alguns snippets de código deste codelab não mostram tudo por questões de simplicidade. O código que não for mudado ou não for relevante para a etapa atual será representado por reticências (três pontos consecutivos ...), para que você possa se concentrar nas partes mais importantes.

Adicionar um campo de texto do custo do serviço

Nesta etapa, vamos adicionar o elemento da IU para permitir que o usuário insira o custo do serviço no app. Você vai usar um elemento EditText, que permite que o usuário insira ou modifique texto em um app.



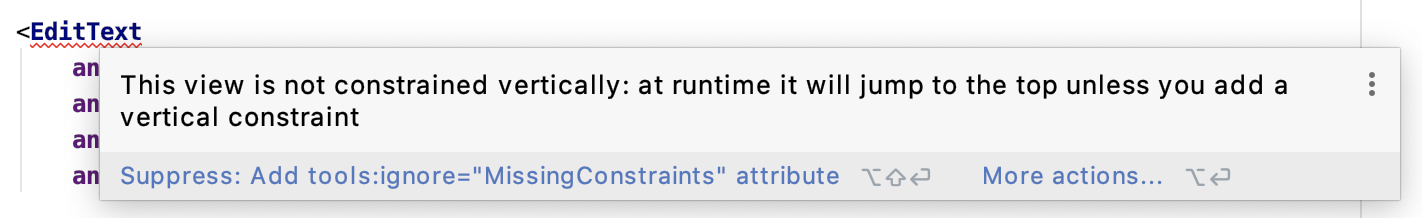
1. Confira a documentação da classe [EditText](https://developer.android.com/reference/kotlin/android/widget/EditText?hl=pt-br" \t "_blank) e analise o XML de exemplo.
2. Encontre um espaço em branco entre as tags de início e término do ConstraintLayout.
3. Copie e cole o XML da documentação nesse espaço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

O arquivo de layout ficará assim:

<?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\_height="wrap\_content"  
        android:layout\_width="match\_parent"  
        android:inputType="text"/>  
  
</androidx.constraintlayout.widget.ConstraintLayout>

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.

app:layout\_constraintStart\_toStartOf="parent"  
app:layout\_constraintTop\_toTopOf="parent"

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".

**Observação**: o nome das restrições segue o formato layout\_constraint<Source>\_to<Target>Of, em que <Source> se refere à visualização atual. <Target> se refere à borda da visualização de destino à qual a visualização atual está limitada, seja o contêiner pai ou outra visualização.

Com as novas restrições adicionadas, o elemento EditText ficará assim:

<EditText  
    android:id="@+id/plain\_text\_input"  
    android:layout\_height="wrap\_content"  
    android:layout\_width="match\_parent"  
    app:layout\_constraintStart\_toStartOf="parent"  
    app:layout\_constraintTop\_toTopOf="parent"  
    android:inputType="text"/>

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.

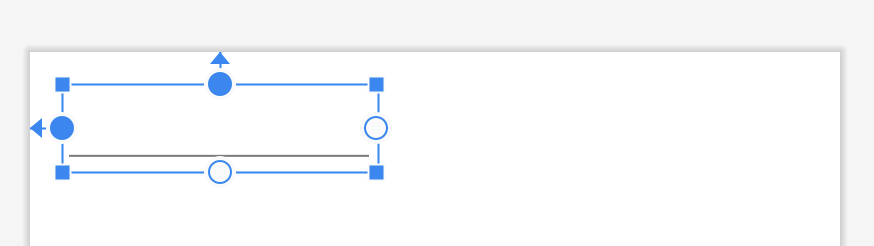
1. Localize o atributo id, que está definido como @+id/plain\_text\_input.
2. Mude o atributo id para um nome mais adequado, @+id/cost\_of\_service.

**Observação**: um ID de recurso é um nome de recurso exclusivo para o elemento. Ele é atribuído automaticamente pelo Android Studio quando você adiciona View ou outro recurso ao **Layout Editor**. 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.).

android:inputType="text"

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. Depois que você digitar a letra "n", o Android Studio vai mostrar uma lista de possíveis opções que incluem "n".



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

android:inputType="numberDecimal"

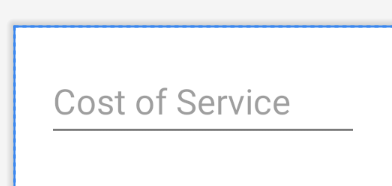
Para ver outras opções de tipos de entrada, consulte [Especificar o tipo de método de entrada](https://developer.android.com/training/keyboard-input/style?hl=pt-br) na documentação para desenvolvedores.

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

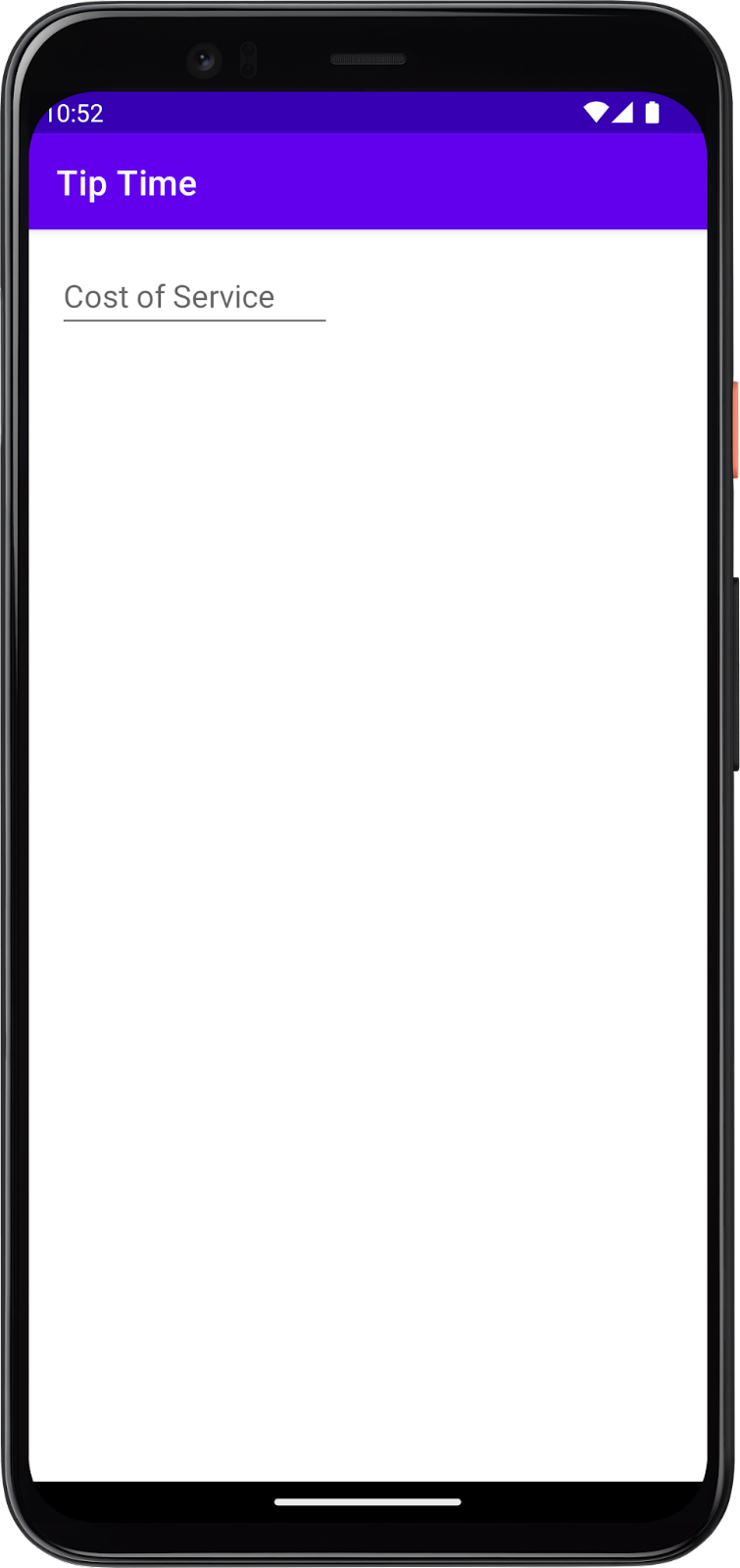
1. Adicione um atributo hint à EditText descrevendo o que o usuário precisa inserir no campo.

android:hint="Cost of Service"

A atualização vai ser exibida no **Design Editor** também.



1. Execute o app no emulador. Ele vai ter esta aparência:

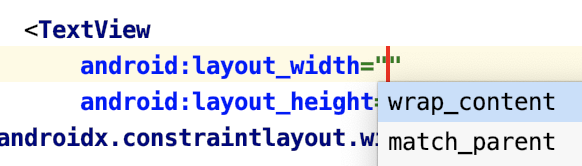


Muito bem! Ele ainda não tem muitas funcionalidades, mas você já tem uma boa base e aprendeu a editar um XML. O XML do seu layout será parecido com este:

<?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/cost\_of\_service"  
        android:layout\_width="160dp"  
        android:layout\_height="wrap\_content"  
        android:hint="Cost of Service"  
        android:inputType="numberDecimal"  
        app:layout\_constraintStart\_toStartOf="parent"  
        app:layout\_constraintTop\_toTopOf="parent" />  
  
</androidx.constraintlayout.widget.ConstraintLayout>

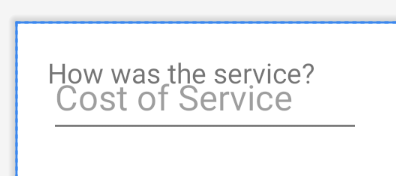
Adicionar pergunta sobre o serviço

Nesta etapa, você adicionará uma TextView para a pergunta "Como foi o serviço?". Tente digitar isso, sem copiar e colar. As sugestões do Android Studio vão ajudar você.

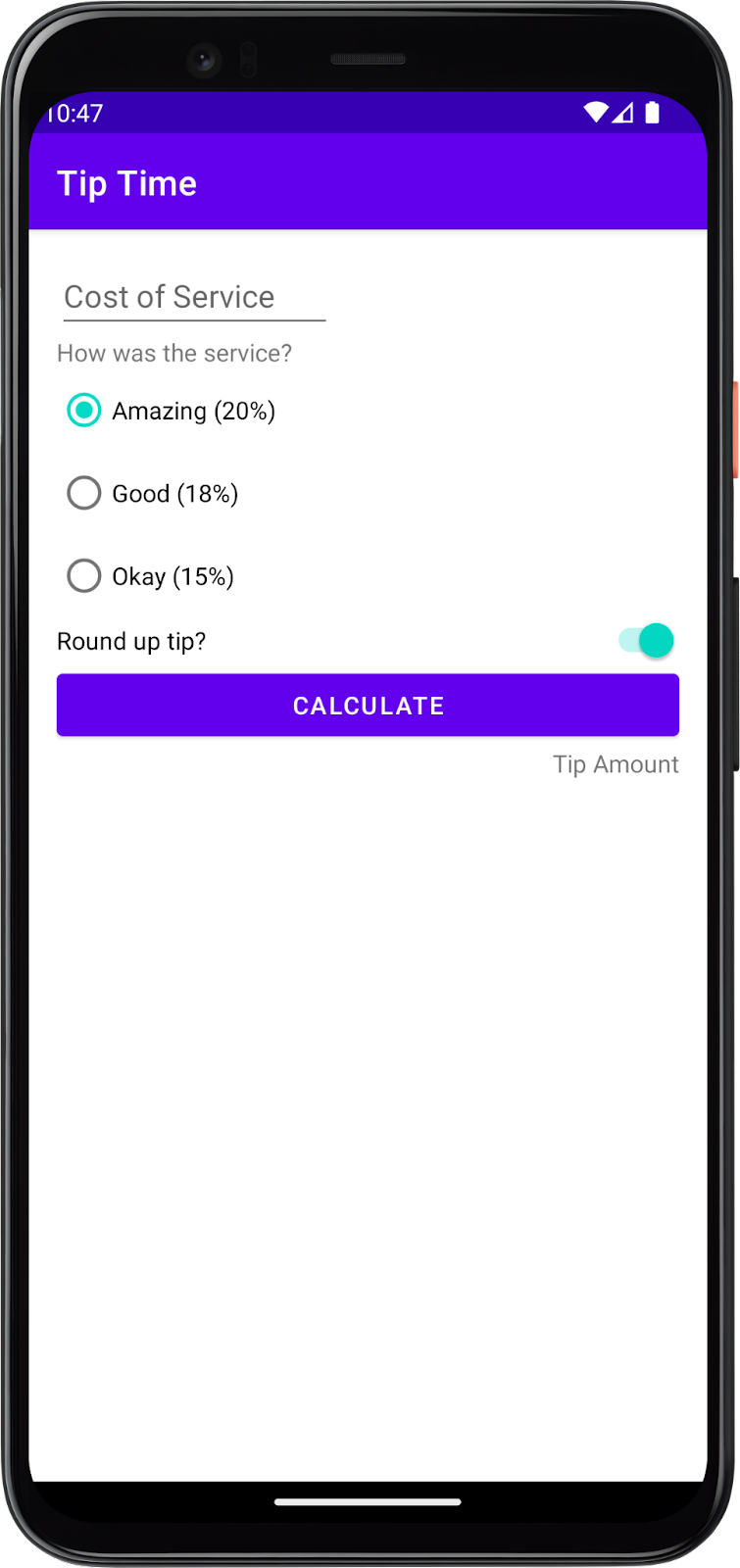
1. Após o fechamento da tag EditText, />, adicione uma nova linha e comece a digitar <TextView.
2. Selecione TextView nas sugestões e o Android Studio vai adicionar automaticamente os atributos layout\_width e layout\_height à TextView.
3. Escolha wrap\_content para ambos os valores, já que você só precisa que a TextView seja do tamanho do conteúdo de texto dentro dela. 
4. Adicione o atributo text com a pergunta "How was the service?"

<TextView  
    android:layout\_width="wrap\_content"  
    android:layout\_height="wrap\_content"  
    android:text="How was the service?"

1. Feche a tag com />.
2. No **Design Editor**, a TextView se sobrepõe à EditText.



Parece que tem algo errado, por isso você adicionará restrições à TextView em seguida. Pense em quais restrições você precisa. Onde a TextView precisa ser posicionada horizontalmente e verticalmente? Para ajudar, confira a captura de tela do app.



No sentido vertical, é recomendável que a TextView esteja abaixo do campo de texto do custo do serviço. No sentido horizontal, você quer que a TextView esteja alinhada com a borda inicial do pai.

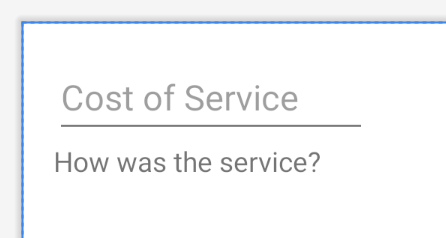
1. Adicione uma restrição horizontal à TextView para restringir a borda inicial da visualização à borda inicial do pai.

app:layout\_constraintStart\_toStartOf="parent"

1. Adicione uma restrição vertical à TextView para limitar a borda superior da TextView à borda inferior da View do custo do serviço.

app:layout\_constraintTop\_toBottomOf="@id/cost\_of\_service"

Não há mais um sinal de adição em @id/cost\_of\_service porque o ID já foi definido.



Não é a melhor aparência possível, mas não se preocupe com isso por enquanto. Basta verificar se todos os elementos necessários estão na tela e se o app funciona. Você melhorará a aparência nos próximos codelabs.

1. Adicione um ID de recurso à TextView. Você precisará referenciá-la mais tarde, à medida que adicionar mais visualizações e criar restrições para elas.

android:id="@+id/service\_question"

Seu XML ficará 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/cost\_of\_service"  
        android:hint="Cost of Service"  
        android:layout\_height="wrap\_content"  
        android:layout\_width="160dp"  
        app:layout\_constraintStart\_toStartOf="parent"  
        app:layout\_constraintTop\_toTopOf="parent"  
        android:inputType="numberDecimal"/>  
  
    <TextView  
        android:id="@+id/service\_question"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:text="How was the service?"  
        app:layout\_constraintStart\_toStartOf="parent"  
        app:layout\_constraintTop\_toBottomOf="@id/cost\_of\_service"/>  
  
</androidx.constraintlayout.widget.ConstraintLayout>

[5. 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)

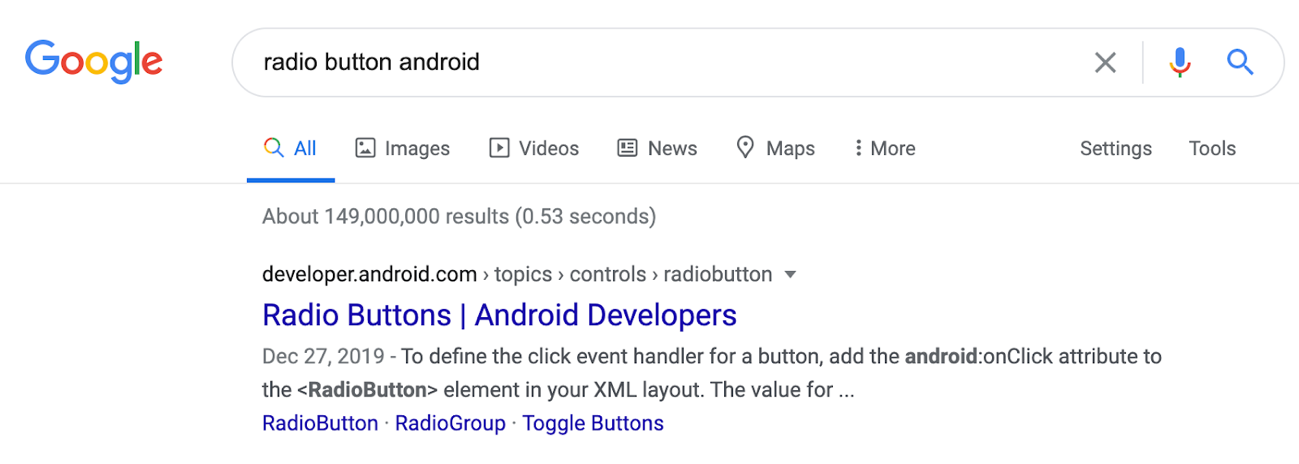
Em seguida, você adicionará botões de opção para as diferentes opções de gorjeta que o usuário poderá escolher.

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.

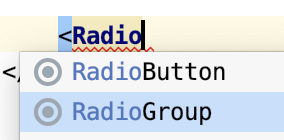
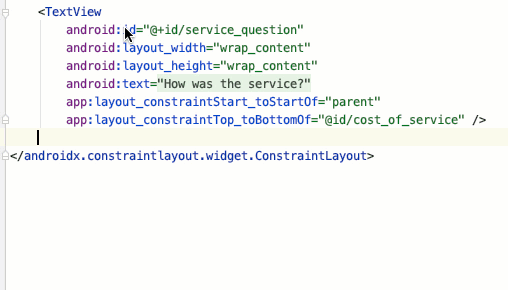
1. Pesquise radio button android no Google. O principal resultado é um guia do site para desenvolvedores Android sobre como usar os botões de opção. Perfeito!



1. Consulte o [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. 
3. Defina os atributos layout\_width e layout\_height do RadioGroup como wrap\_content.
4. Adicione um ID de recurso definido como @+id/tip\_options.
5. Feche a tag de início com >.
6. O Android Studio adicionará o </RadioGroup>. Assim como o ConstraintLayout, o RadioGroup tem outros elementos dentro dele. Assim, é possível movê-lo para uma linha própria. 
7. Restrinja o RadioGroup abaixo da pergunta sobre o serviço (verticalmente) e no início do pai (horizontalmente).
8. 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:

<RadioGroup  
    android:id="@+id/tip\_options"  
    android:layout\_width="wrap\_content"  
    android:layout\_height="wrap\_content"  
    android:orientation="vertical"  
    app:layout\_constraintStart\_toStartOf="parent"  
    app:layout\_constraintTop\_toBottomOf="@id/service\_question">  
  
</RadioGroup>

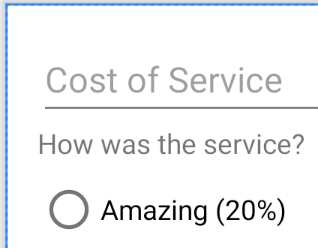
Adicionar RadioButtons

1. Após o último atributo do RadioGroup, mas antes da tag de término </RadioGroup>, adicione um RadioButton.

<RadioGroup  
    android:id="@+id/tip\_options"  
    android:layout\_width="wrap\_content"  
    android:layout\_height="wrap\_content"  
    android:orientation="vertical"  
    app:layout\_constraintStart\_toStartOf="parent"  
    app:layout\_constraintTop\_toBottomOf="@id/service\_question">  
  
    <!-- add RadioButtons here -->  
  
</RadioGroup>

1. Configure a layout\_width e a layout\_height como wrap\_content.
2. Atribua um ID de recurso de @+id/option\_twenty\_percent ao RadioButton.
3. Defina o texto como Amazing (20%).
4. Feche a tag com />.

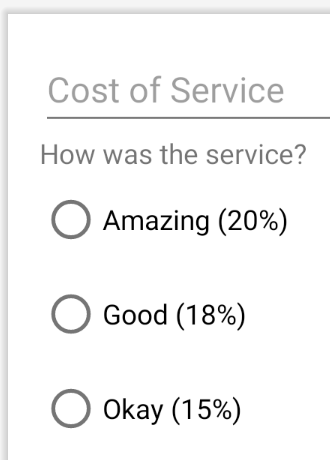
<RadioGroup  
   android:id="@+id/tip\_options"  
   android:layout\_width="wrap\_content"  
   android:layout\_height="wrap\_content"  
   app:layout\_constraintTop\_toBottomOf="@id/service\_question"  
   app:layout\_constraintStart\_toStartOf="parent"  
   android:orientation="vertical">  
  
   <RadioButton  
       android:id="@+id/option\_twenty\_percent"  
       android:layout\_width="wrap\_content"  
       android:layout\_height="wrap\_content"  
       android:text="Amazing (20%)" />  
  
</RadioGroup>



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

O XML para RadioGroup e RadioButtons vai ficar assim:

<RadioGroup  
   android:id="@+id/tip\_options"  
   android:layout\_width="wrap\_content"  
   android:layout\_height="wrap\_content"  
   app:layout\_constraintTop\_toBottomOf="@id/service\_question"  
   app:layout\_constraintStart\_toStartOf="parent"  
   android:orientation="vertical">  
  
   <RadioButton  
       android:id="@+id/option\_twenty\_percent"  
       android:layout\_width="wrap\_content"  
       android:layout\_height="wrap\_content"  
       android:text="Amazing (20%)" />  
  
   <RadioButton  
       android:id="@+id/option\_eighteen\_percent"  
       android:layout\_width="wrap\_content"  
       android:layout\_height="wrap\_content"  
       android:text="Good (18%)" />  
  
   <RadioButton  
       android:id="@+id/option\_fifteen\_percent"  
       android:layout\_width="wrap\_content"  
       android:layout\_height="wrap\_content"  
       android:text="Okay (15%)" />  
  
</RadioGroup>



Adicionar uma seleção padrão

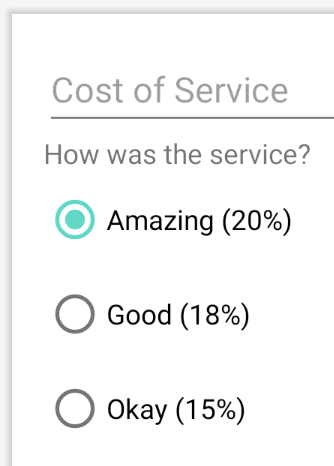
No momento, nenhuma das opções de gorjeta está selecionada. É uma boa ideia selecionar uma das opções de botão como padrão.

Há um atributo no RadioGroup em que é possível especificar qual botão fica 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 como @id/option\_twenty\_percent.

<RadioGroup  
   android:id="@+id/tip\_options"  
   android:checkedButton="@id/option\_twenty\_percent"  
   ...

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.

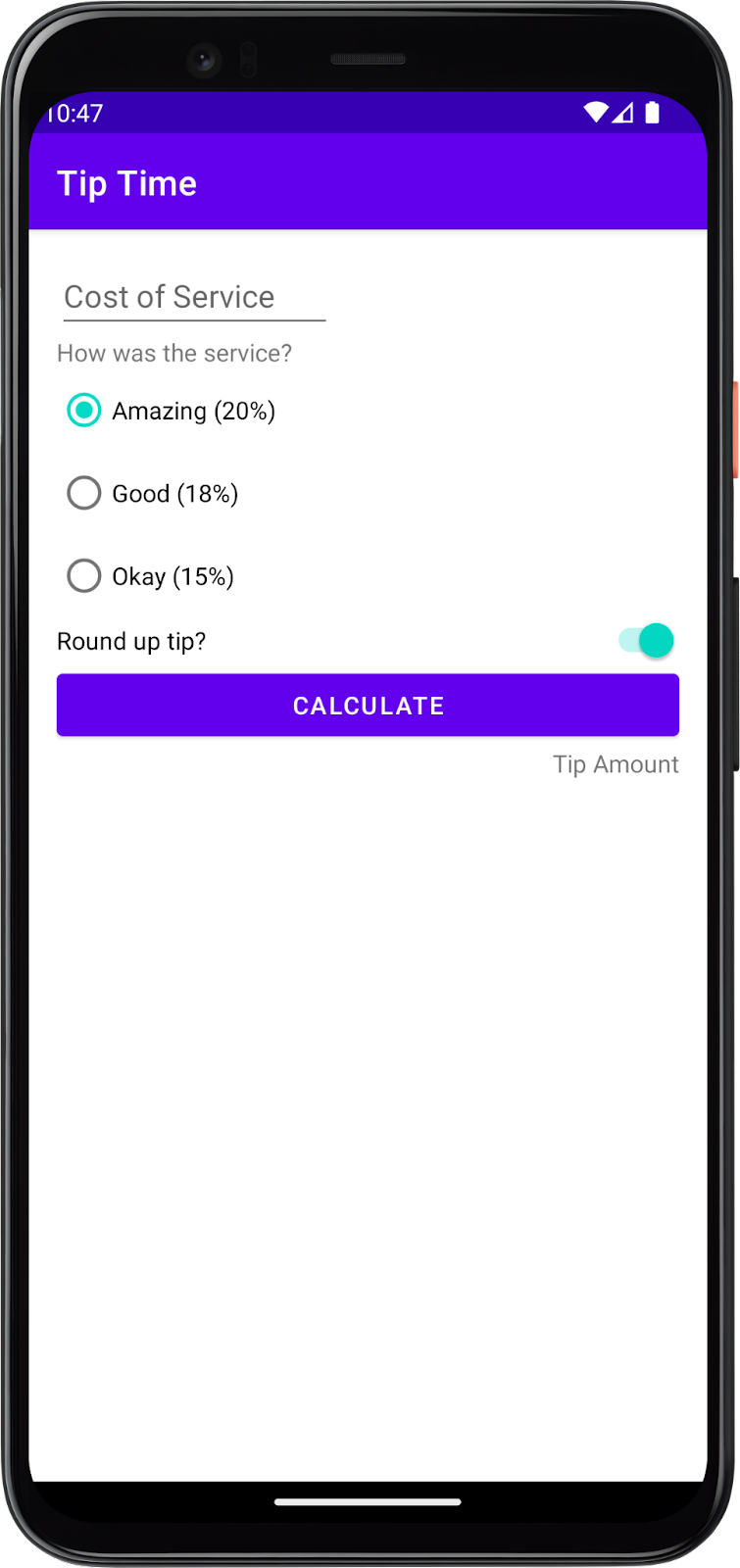


Confira a aparência do XML até agora:

<?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/cost\_of\_service"  
        android:hint="Cost of Service"  
        android:layout\_height="wrap\_content"  
        android:layout\_width="160dp"  
        app:layout\_constraintStart\_toStartOf="parent"  
        app:layout\_constraintTop\_toTopOf="parent"  
        android:inputType="numberDecimal"/>  
  
    <TextView  
        android:id="@+id/service\_question"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:text="How was the service?"  
        app:layout\_constraintTop\_toBottomOf="@id/cost\_of\_service"  
        app:layout\_constraintStart\_toStartOf="parent" />  
  
    <RadioGroup  
        android:id="@+id/tip\_options"  
        android:checkedButton="@id/option\_twenty\_percent"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        app:layout\_constraintTop\_toBottomOf="@id/service\_question"  
        app:layout\_constraintStart\_toStartOf="parent"  
        android:orientation="vertical">  
  
        <RadioButton  
            android:id="@+id/option\_twenty\_percent"  
            android:layout\_width="wrap\_content"  
            android:layout\_height="wrap\_content"  
            android:text="Amazing (20%)" />  
  
        <RadioButton  
            android:id="@+id/option\_eighteen\_percent"  
            android:layout\_width="wrap\_content"  
            android:layout\_height="wrap\_content"  
            android:text="Good (18%)" />  
  
        <RadioButton  
            android:id="@+id/option\_fifteen\_percent"  
            android:layout\_width="wrap\_content"  
            android:layout\_height="wrap\_content"  
            android:text="Okay (15%)" />  
    </RadioGroup>  
</androidx.constraintlayout.widget.ConstraintLayout>

[6. Concluir o restante do layout](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#5)

Você está na última parte do layout. Falta adicionar um Switch, um Button e uma TextView para mostrar o valor da gorjeta.



Adicionar uma chave 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. Defina a layout\_height como wrap\_content. Assim, a altura da visualização Switch vai ser a mesma do conteúdo.
4. Defina o atributo id como @+id/round\_up\_switch.
5. Defina o atributo text como Round up tip?. Ele vai ser usado como um rótulo para o Switch.
6. Restrinja a borda inicial do Switch à borda inicial do tip\_options, e a borda final ao fim do pai.
7. Restrinja a parte de cima do Switch à parte de baixo de tip\_options.
8. Feche a tag com />.

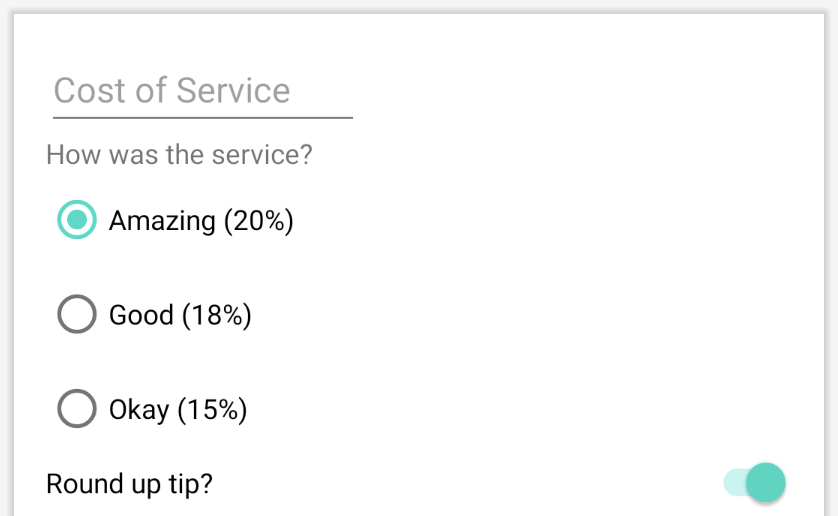
Seria bom se o interruptor fosse ativado por padrão. Há um atributo para isso, android:checked, em que os valores possíveis são true (ativado) ou false (desativado).

1. Defina o atributo android:checked como true.

Juntando tudo isso, o XML do elemento Switch vai ficar assim:

<Switch  
    android:id="@+id/round\_up\_switch"  
    android:layout\_width="0dp"  
    android:layout\_height="wrap\_content"  
    android:checked="true"  
    android:text="Round up tip?"  
    app:layout\_constraintEnd\_toEndOf="parent"  
    app:layout\_constraintStart\_toStartOf="@id/tip\_options"  
    app:layout\_constraintTop\_toBottomOf="@id/tip\_options" />

**Observação**: se você passar o cursor sobre o elemento Switch do seu layout XML, vai ver uma sugestão no Android Studio com a mensagem: "Use SwitchCompat from AppCompat or SwitchMaterial from Material library". Essa sugestão vai ser implementada em um próximo codelab para criar o app de calculadora de gorjetas. Ignore o aviso.



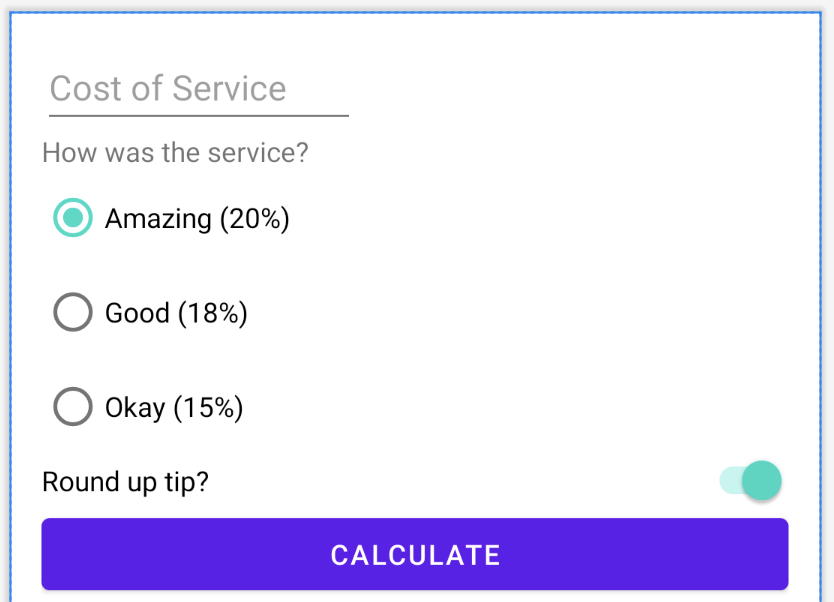
Adicionar o botão "Calcular"

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 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 elemento pai e a borda final ao fim do pai.
7. Feche a tag com />.

Veja como o XML para o Button **Calcular** vai ficar:

<Button  
   android:id="@+id/calculate\_button"  
   android:layout\_width="0dp"  
   android:layout\_height="wrap\_content"  
   android:text="Calculate"  
   app:layout\_constraintTop\_toBottomOf="@id/round\_up\_switch"  
   app:layout\_constraintStart\_toStartOf="parent"  
   app:layout\_constraintEnd\_toEndOf="parent" />



Adicionar o resultado da gorjeta

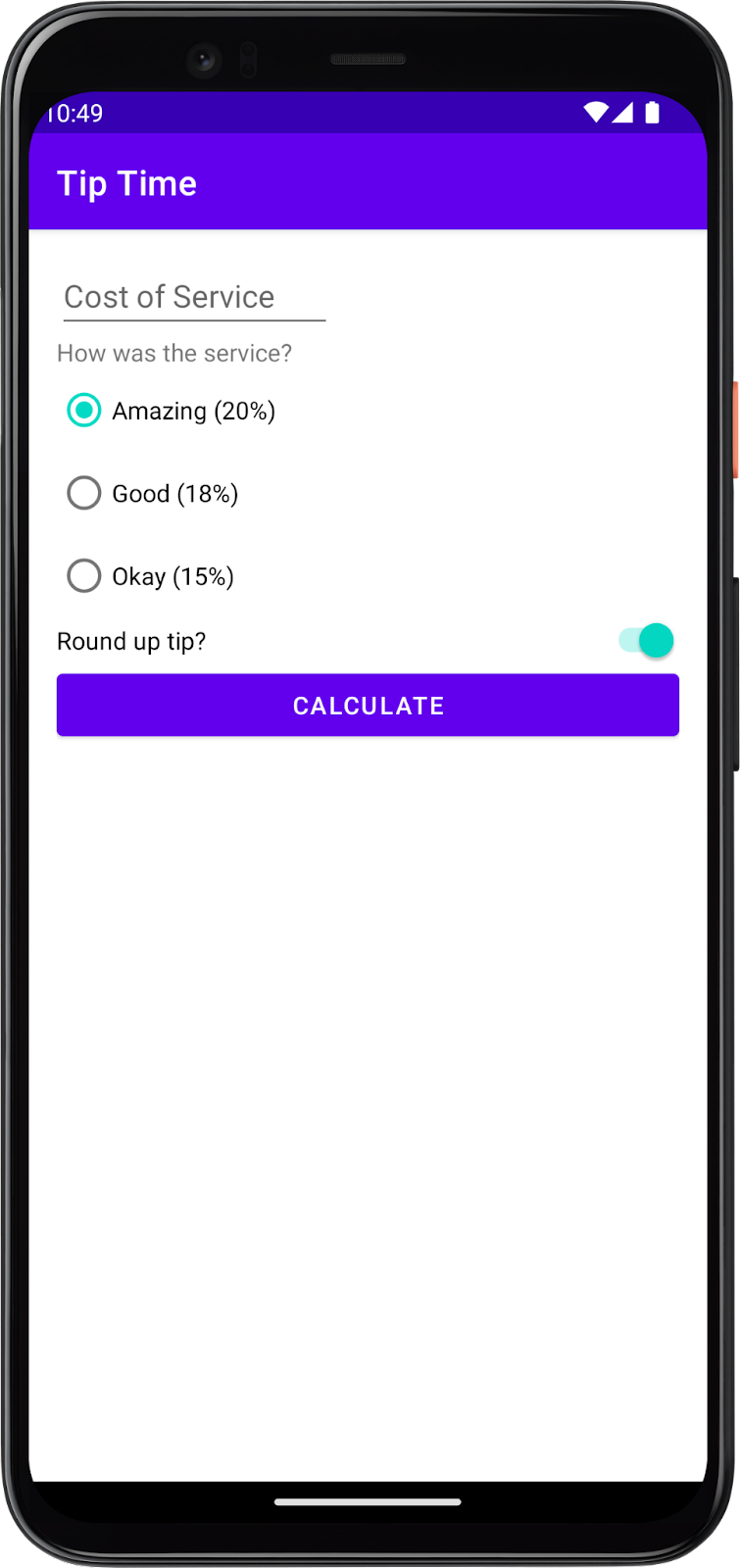
Você está quase terminando o layout! Nesta etapa, você vai adicionar uma TextView para o resultado da gorjeta. Ela será colocada abaixo do botão **Calcular** e alinhada com o final, e não o início, como nos outros elementos da IU.

1. Adicione uma TextView com um ID de recurso chamado tip\_result e o texto Tip Amount.
2. Restrinja a borda final da TextView à borda final do pai.
3. Restrinja a borda de cima da visualização à borda de baixo do botão **Calculate**.

<TextView  
    android:id="@+id/tip\_result"  
    android:layout\_width="wrap\_content"  
    android:layout\_height="wrap\_content"  
    app:layout\_constraintEnd\_toEndOf="parent"  
    app:layout\_constraintTop\_toBottomOf="@id/calculate\_button"  
    android:text="Tip Amount" />



1. Execute o app. Ele vai ficar parecido com esta captura de tela.



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 **Calculate** 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.

[7. Adotar boas práticas de programação](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#6)

Extrair as strings

Você pode ter percebido os avisos sobre strings codificadas. Relembre dos codelabs anteriores que extrair strings para um arquivo de recurso facilita a tradução do app para outros idiomas e a reutilização de strings. Extraia todos os recursos de string do arquivo activity\_main.xml.

1. Clique em uma string, passe o cursor sobre o ícone de lâmpada amarela exibido e, em seguida, clique no triângulo ao lado dele, depois escolha **Extract String Resource**. Os nomes padrão dos recursos de string são bons. Se quiser, nas opções de gorjeta, você pode usar amazing\_service, good\_service e ok\_service para tornar os nomes mais descritivos.

Agora, verifique os recursos de string que você acabou de adicionar.

1. Se a janela **Project** não for exibida, clique na guia **Project** no lado esquerdo da janela.
2. Abra **app > res > values > strings.xml** para ver todos os recursos de string da IU.

<resources>  
    <string name="app\_name">Tip Time</string>  
    <string name="cost\_of\_service">Cost of Service</string>  
    <string name="how\_was\_the\_service">How was the service?</string>  
    <string name="amazing\_service">Amazing (20%)</string>  
    <string name="good\_service">Good (18%)</string>  
    <string name="ok\_service">Okay (15%)</string>  
    <string name="round\_up\_tip">Round up tip?</string>  
    <string name="calculate">Calculate</string>  
    <string name="tip\_amount">Tip Amount</string>  
</resources>

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.

[3. Vinculação de visualizações](https://developer.android.com/codelabs/basic-android-kotlin-training-tip-calculator?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-tip-calculator#2)

Para calcular a gorjeta, o código precisa acessar todos os elementos da IU para ler a entrada do usuário. Você pode se lembrar de codelabs anteriores em que o código precisava encontrar uma referência a uma View, como um Button ou uma TextView, para que seu código pudesse chamar métodos na View ou acessar os atributos. O framework do Android fornece o método findViewById(), que faz exatamente o que você precisa: recebe o ID de uma View e retorna uma referência a ela. Essa abordagem funciona, mas, à medida que você adiciona mais visualizações ao app e a IU fica mais complexa, o uso do método findViewById() pode ficar complicado.

Por conveniência, o Android também oferece um recurso chamado [*vinculação de visualizações*](https://developer.android.com/topic/libraries/view-binding?hl=pt-br). Com um pouco mais de trabalho no início, a vinculação de visualizações faz com que seja muito mais fácil e rápido chamar métodos nas visualizações da IU. Será necessário ativar a vinculação de visualizações do app no Gradle e fazer algumas mudanças no código.

Ativar a vinculação de visualizações

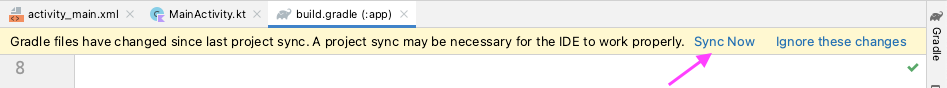
1. Abra o arquivo build.gradle do app (**Gradle Scripts > build.gradle (Module: Tip\_Time.app)**)
2. Na seção android, adicione as seguintes linhas:

buildFeatures {

viewBinding = true

}

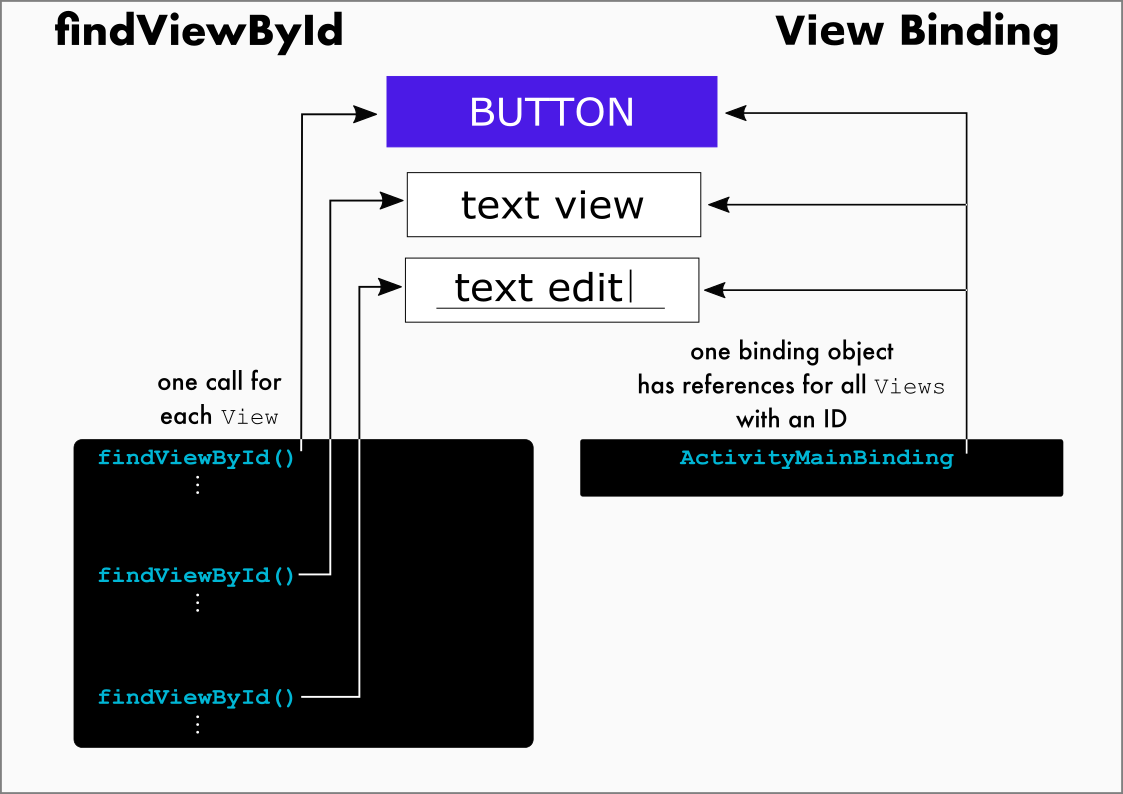
1. A mensagem **Gradle files have changed since last project sync**, que informa que os arquivos mudaram desde a última sincronização, deve aparecer.
2. Pressione **Sync Now**.



Após alguns instantes, vai aparecer uma mensagem na parte de baixo da janela do Android Studio, **Gradle sync finished**. Feche o arquivo build.gradle, se quiser.

Inicializar o objeto de vinculação

Nos codelabs anteriores, você viu o método onCreate() na classe MainActivity. Esse método é um dos primeiros a serem chamados quando o app é iniciado e a MainActivity é inicializada. Em vez de chamar findViewById() para cada View no seu app, você vai criar e inicializar um objeto de vinculação apenas uma vez.



1. Abra MainActivity.kt (**app > java > com.example.tiptime > MainActivity**).
2. Substitua todo o código da classe MainActivity por este código para configurar a MainActivity para usar a vinculação de visualização:

class MainActivity : AppCompatActivity() {  
  
    lateinit var binding: ActivityMainBinding  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        binding = ActivityMainBinding.inflate(layoutInflater)  
        setContentView(binding.root)  
    }  
}

1. Esta linha declara uma variável de nível superior na classe para o objeto de vinculação. Ela é definida nesse nível porque será usada em vários métodos da classe MainActivity.

lateinit var binding: ActivityMainBinding

A palavra-chave lateinit é nova. Ela é uma promessa de que seu código vai inicializar a variável antes de usá-la. Se você não fizer isso, o app falhará.

1. Essa linha inicializa o objeto binding que você usará para acessar as Views no layout activity\_main.xml.

binding = ActivityMainBinding.inflate(layoutInflater)

1. Defina a visualização de conteúdo da atividade. Em vez de transmitir o ID de recurso do layout, R.layout.activity\_main, esse código especifica a raiz da hierarquia de visualizações no app, binding.root.

setContentView(binding.root)

É possível que você se lembre da ideia de visualizações mães e filhas. A raiz se conecta a todas elas.

Agora, quando você precisar de uma referência a uma View no app, poderá acessá-la no objeto binding, em vez de chamar o método findViewById(). O objeto binding define automaticamente as referências para cada View do app que tenha um ID. Usar a vinculação de visualizações é tão mais conciso que muitas vezes não será necessário criar uma variável para armazenar a referência de uma View, usá-la diretamente com o objeto de vinculação vai ser suficiente.

// Old way with findViewById()  
val myButton: Button = findViewById(R.id.my\_button)  
myButton.text = "A button"  
  
// Better way with view binding  
val myButton: Button = binding.myButton  
myButton.text = "A button"  
  
// Best way with view binding and no extra variable  
binding.myButton.text = "A button"

Não é demais?

**Observação**: o nome da classe de vinculação é gerado pela conversão do nome do arquivo XML no padrão PascalCase e da adição da palavra "Binding" ao final. Da mesma forma, a referência para cada visualização é gerada removendo sublinhados e convertendo o nome da visualização para camelCase. Por exemplo, em PascalCase, activity\_main.xml se torna ActivityMainBinding, e você pode acessar @id/text\_view como binding.textView.

[4. Calcular a gorjeta](https://developer.android.com/codelabs/basic-android-kotlin-training-tip-calculator?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-tip-calculator#3)

O cálculo da gorjeta começará quando o usuário tocar no botão **Calculate**. Isso envolve verificar a IU para ver o custo do serviço e o percentual de gorjeta que o usuário quer deixar. Com essas informações, você calcula o valor total cobrado pelo serviço e exibe o valor da gorjeta.

Adicionar listeners de clique ao botão

O primeiro passo é adicionar um listener de clique para especificar o que o botão **Calculate** vai fazer quando o usuário tocar nele.

1. Em MainActivity.kt no método onCreate(), após a chamada de setContentView(), defina um listener de clique no botão **Calculate** e faça com que ele chame o método calculateTip().

binding.calculateButton.setOnClickListener{ calculateTip() }

1. Ainda na classe MainActivity, mas fora do método onCreate(), adicione um método auxiliar com o nome calculateTip().

fun calculateTip() {  
  
}

É aqui que você adicionará o código para verificar a IU e calcular a gorjeta.

MainActivity.kt

class MainActivity : AppCompatActivity() {  
  
    lateinit var binding: ActivityMainBinding  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        binding = ActivityMainBinding.inflate(layoutInflater)  
        setContentView(binding.root)  
        binding.calculateButton.setOnClickListener{ calculateTip() }  
    }  
  
    fun calculateTip() {  
  
    }  
}

Acessar o custo do serviço

Para calcular a gorjeta, a primeira coisa de que você precisa é o custo do serviço. O texto é armazenado no EditText, mas você precisa que ele seja um número para poder usá-lo em cálculos. Você pode se lembrar do tipo Int de outros codelabs, mas um Int só pode conter números inteiros. Para usar um número decimal no app, use o tipo de dados Double em vez de Int. Leia mais sobre [tipos de dados numéricos em Kotlin](https://kotlinlang.org/docs/reference/basic-types.html#numbers) (em inglês) na documentação. O Kotlin fornece um método para converter uma String em um Double, conhecido como toDouble().

1. Primeiro, acesse o texto do custo do serviço. No método calculateTip(), acesse o atributo de texto EditText **Cost of Service**e atribua-o a uma variável com o nome stringInTextField. Não esqueça que é possível acessar o elemento de IU usando o objeto binding e fazendo referência ao elemento da IU com base no nome do ID de recurso em letras concatenadas.

val stringInTextField = binding.costOfService.text

Observe o .text no final. A primeira parte, binding.costOfService, faz referência ao elemento da IU para o custo do serviço. Adicionar .text no final instrui o app a usar o resultado (um objeto EditText) e acessar a propriedade text dele. Isso é conhecido como *encadeamento* e é um padrão muito comum no Kotlin.

1. Depois, converta o texto em número decimal. Chame o método toDouble() em stringInTextField e armazene-o em uma variável com o nome cost.

val cost = stringInTextField.toDouble()

No entanto, isso não funcionará. toDouble() precisa ser chamado em uma String. Perceba que o atributo text de um EditText é do tipo Editable, porque representa um texto que pode ser modificado. Felizmente, é possível converter um Editable em uma String chamando o método toString() nele.

1. Chame toString() em binding.costOfService.text para convertê-lo em uma String:

val stringInTextField = binding.costOfService.text.toString()

Agora o método stringInTextField.toDouble() funcionará.

Neste ponto, o método calculateTip() vai ficar assim:

fun calculateTip() {  
    val stringInTextField = binding.costOfService.text.toString()  
    val cost = stringInTextField.toDouble()  
}

Acessar a porcentagem da gorjeta

Até o momento, você tem o custo do serviço. Agora você precisa da porcentagem de gorjeta, que o usuário selecionou em um RadioGroup de RadioButtons.

1. Em calculateTip(), acesse o atributo checkedRadioButtonId do RadioGroup tipOptions e atribua-o a uma variável com o nome selectedId.

val selectedId = binding.tipOptions.checkedRadioButtonId

Agora você sabe que uma das opções entre R.id.option\_twenty\_percent, R.id.option\_eighteen\_percent ou R.id.fifteen\_percent foi o RadioButton selecionado, mas precisa da porcentagem correspondente. É possível escrever uma série de instruções if/else, mas é muito mais fácil usar uma expressão when.

1. Adicione as linhas abaixo para acessar a porcentagem da gorjeta.

val tipPercentage = when (selectedId) {  
    R.id.option\_twenty\_percent -> 0.20  
    R.id.option\_eighteen\_percent -> 0.18  
    else -> 0.15  
}

Neste ponto, o método calculateTip() vai ficar assim:

fun calculateTip() {  
    val stringInTextField = binding.costOfService.text.toString()  
    val cost = stringInTextField.toDouble()  
    val selectedId = binding.tipOptions.checkedRadioButtonId  
    val tipPercentage = when (selectedId) {  
        R.id.option\_twenty\_percent -> 0.20  
        R.id.option\_eighteen\_percent -> 0.18  
        else -> 0.15  
    }  
}

Calcular a gorjeta e arredondar para cima

Agora que você tem o custo do serviço e o percentual de gorjeta, calcular a gorjeta é simples: multiplique o custo pela porcentagem da gorjeta. Gorjeta = custo do serviço \* porcentagem da gorjeta. Esse valor pode ser arredondado para cima.

1. No método calculateTip(), depois do outro código que você adicionou, multiplique a tipPercentage pelo cost e atribua o resultado a uma variável com o nome tip.

var tip = tipPercentage \* cost

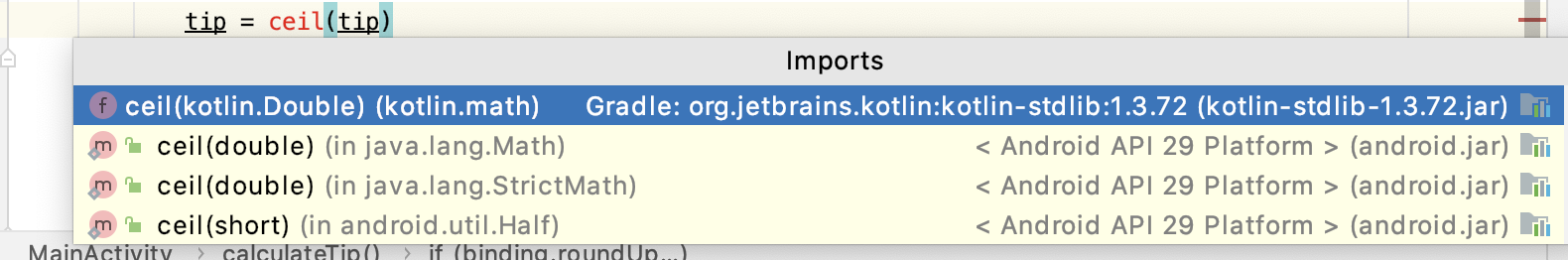
Observe que var é usado em vez de val. Isso ocorre porque pode ser necessário arredondar o valor se o usuário tiver selecionado essa opção. Assim, o valor poderá mudar.

Em um elemento Switch, verifique o atributo isChecked para ver se a chave está "ativada".

1. Atribua o atributo isChecked do interruptor de arredondamento a uma variável com o nome roundUp.

val roundUp = binding.roundUpSwitch.isChecked

Arredondar significa ajustar um número decimal para cima ou para baixo para que seja o valor inteiro mais próximo. Porém, nesse caso, você só quer arredondar para cima ou encontrar o teto. Para isso, use a função ceil(). Há várias funções com esse nome, mas a que você quer usar está definida em kotlin.math. É possível adicionar uma instrução import, mas, neste caso, é mais simples apenas informar ao Android Studio a função que você quer usando kotlin.math.ceil().



Se você quisesse usar várias funções matemáticas, seria mais fácil adicionar uma instrução import.

1. Adicione uma instrução if que atribui o teto da gorjeta à variável tip se roundUp for verdadeiro.

if (roundUp) {  
    tip = kotlin.math.ceil(tip)  
}

Neste ponto, o método calculateTip() vai ficar assim:

fun calculateTip() {  
    val stringInTextField = binding.costOfService.text.toString()  
    val cost = stringInTextField.toDouble()  
    val selectedId = binding.tipOptions.checkedRadioButtonId  
    val tipPercentage = when (selectedId) {  
        R.id.option\_twenty\_percent -> 0.20  
        R.id.option\_eighteen\_percent -> 0.18  
        else -> 0.15  
    }  
    var tip = tipPercentage \* cost  
    val roundUp = binding.roundUpSwitch.isChecked  
    if (roundUp) {  
        tip = kotlin.math.ceil(tip)  
    }  
}

Formatar a gorjeta

Seu app está quase pronto. Você calculou a gorjeta, então só precisa formatá-la e exibi-la.

Como é de se esperar, o Kotlin fornece métodos para formatar diferentes tipos de número. Mas o valor da gorjeta é um pouco diferente: ele representa um valor de moeda. Cada país usa uma moeda diferente e tem regras diferentes para formatar números decimais. Por exemplo, em dólares americanos, 1234.56 seria formatado como $1,234.56, mas em euros, o formato seria €1.234,56. Felizmente, o framework do Android fornece métodos para formatar números como moedas, de modo que você não precisa conhecer todas as possibilidades. O sistema formata automaticamente a moeda com base no idioma e em outras configurações escolhidas pelo usuário no smartphone. Leia mais sobre [NumberFormat](https://developer.android.com/reference/java/text/NumberFormat?hl=pt-br" \t "_blank) na documentação do desenvolvedor Android.

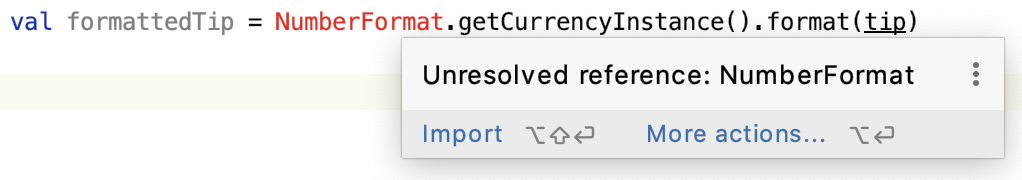
1. Em calculateTip(), após o outro código, chame NumberFormat.getCurrencyInstance().

NumberFormat.getCurrencyInstance()

Isso fornece um formatador numérico que pode ser usado para formatar os números como moedas.

1. Usando o formatador de números, encadeie uma chamada para o método format() com a tip e atribua o resultado a uma variável com o nome formattedTip.

val formattedTip = NumberFormat.getCurrencyInstance().format(tip)

1. Observe que NumberFormat é mostrado em vermelho. Isso ocorre porque o Android Studio não consegue descobrir automaticamente qual versão de NumberFormat você quer usar.
2. Passe o cursor sobre NumberFormat e escolha **Import** no pop-up que vai aparecer. 
3. Na lista de possíveis importações, escolha **NumberFormat (java.text)**. O Android Studio vai adicionar uma instrução import na parte de cima do arquivo MainActivity e o NumberFormat não estará mais vermelho.

Exibir a gorjeta

Agora é hora de exibir a gorjeta no elemento TextView do valor de gorjeta do app. Você pode atribuir formattedTip ao atributo text, mas seria bom marcar o que o valor representa. Nos EUA, em inglês, você pode mostrar **Tip Amount: $12.34**, mas em outros idiomas o número pode precisar aparecer no início ou no meio da string. O framework do Android fornece um mecanismo para isso chamado *parâmetros de string*. Assim, alguém que traduzir seu app poderá mudar onde o número aparece, se necessário.

1. Abra strings.xml (**app > res > values > strings.xml**)
2. Mude a string tip\_amount de Tip Amount para Tip Amount: %s.

<string name="tip\_amount">Tip Amount: %s</string>

O %s é onde a moeda formatada será inserida.

1. Agora, defina o texto do tipResult. De volta ao método calculateTip() em MainActivity.kt, chame getString(R.string.tip\_amount, formattedTip) e o atribua ao atributo text do resultado da gorjeta da TextView.

binding.tipResult.text = getString(R.string.tip\_amount, formattedTip)

Neste ponto, o método calculateTip() vai ficar assim:

fun calculateTip() {  
    val stringInTextField = binding.costOfService.text.toString()  
    val cost = stringInTextField.toDouble()  
    val selectedId = binding.tipOptions.checkedRadioButtonId  
    val tipPercentage = when (selectedId) {  
        R.id.option\_twenty\_percent -> 0.20  
        R.id.option\_eighteen\_percent -> 0.18  
        else -> 0.15  
    }  
    var tip = tipPercentage \* cost  
    val roundUp = binding.roundUpSwitch.isChecked  
    if (roundUp) {  
        tip = kotlin.math.ceil(tip)  
    }  
    val formattedTip = NumberFormat.getCurrencyInstance().format(tip)  
    binding.tipResult.text = getString(R.string.tip\_amount, formattedTip)  
}

Você está quase lá. Ao desenvolver seu app e exibir a visualização, é útil ter um marcador para essa TextView.

1. Abra activity\_main.xml (**app > res > layout > activity\_main.xml**).
2. Localize a TextView do tip\_result.
3. Remova a linha com o atributo android:text.

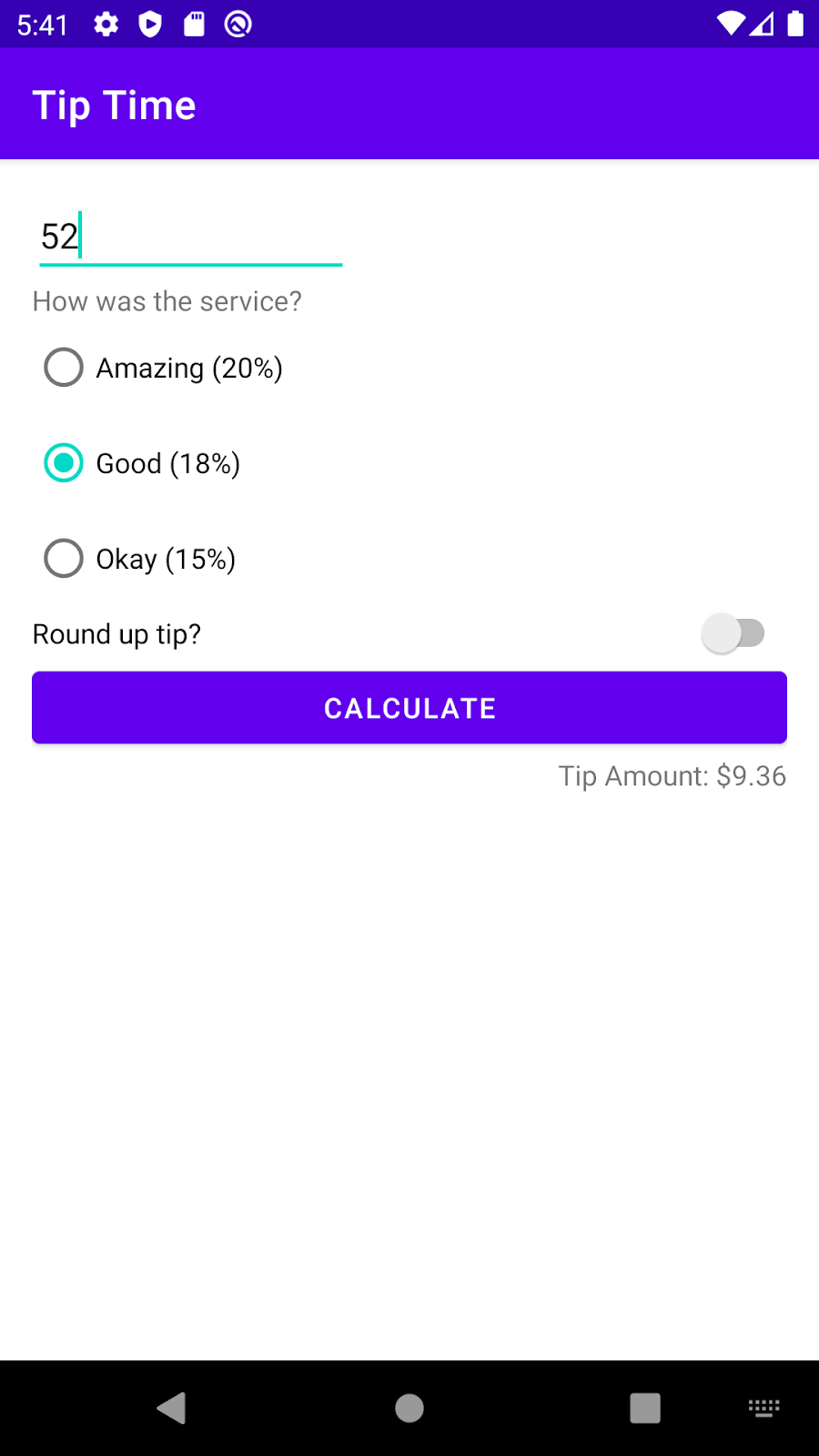
android:text="@string/tip\_amount"

1. Adicione uma linha para o atributo tools:text definido como Tip Amount: $10.

tools:text="Tip Amount: $10"

Como este é apenas um marcador, não é necessário extrair a string para um recurso. Ela não aparecerá quando você executar o app.

1. Observe que o texto das ferramentas é exibido no **Layout Editor**.
2. Execute o app. Insira um valor para o custo, selecione algumas opções e pressione o botão **Calculate**.



Parabéns, o app está funcionando! Se você não estiver recebendo o valor da gorjeta correto, volte para a etapa 1 desta seção e confira se fez todas as mudanças necessárias no código.

[5. Testar e depurar](https://developer.android.com/codelabs/basic-android-kotlin-training-tip-calculator?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-tip-calculator#4)

Você executou o app em várias etapas para garantir que ele faça o que você quer, mas agora é hora de fazer outros testes.

Por enquanto, pense em como as informações passam pelo app no método calculateTip() e o que poderia dar errado em cada etapa.

Por exemplo, o que aconteceria nessa linha:

val cost = stringInTextField.toDouble()

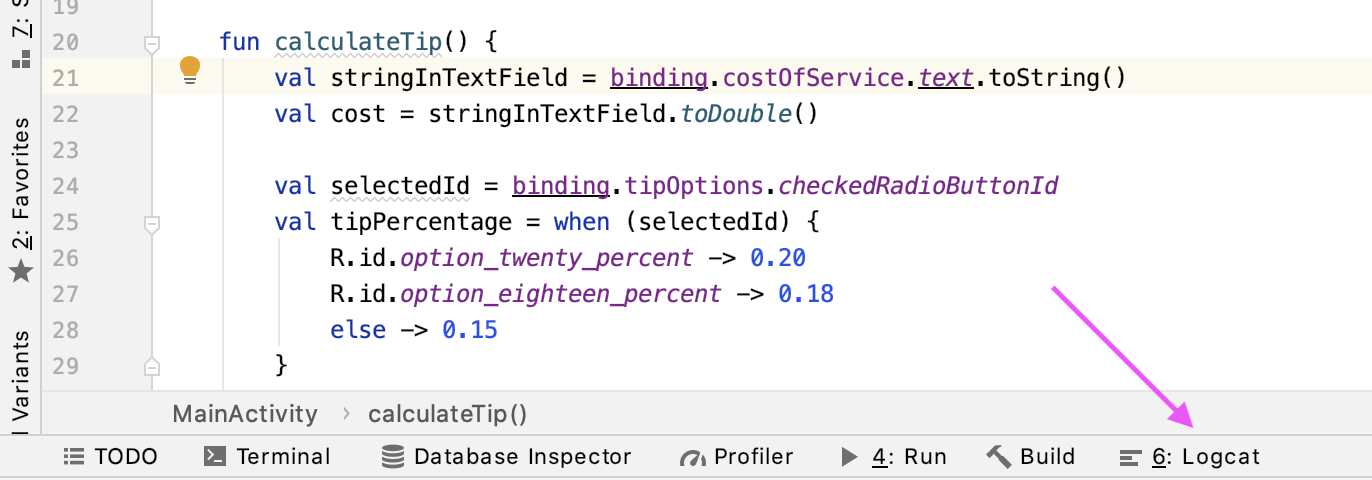
se stringInTextField não representasse um número? O que aconteceria se o usuário não tivesse inserido texto e o campo stringInTextField estivesse vazio?

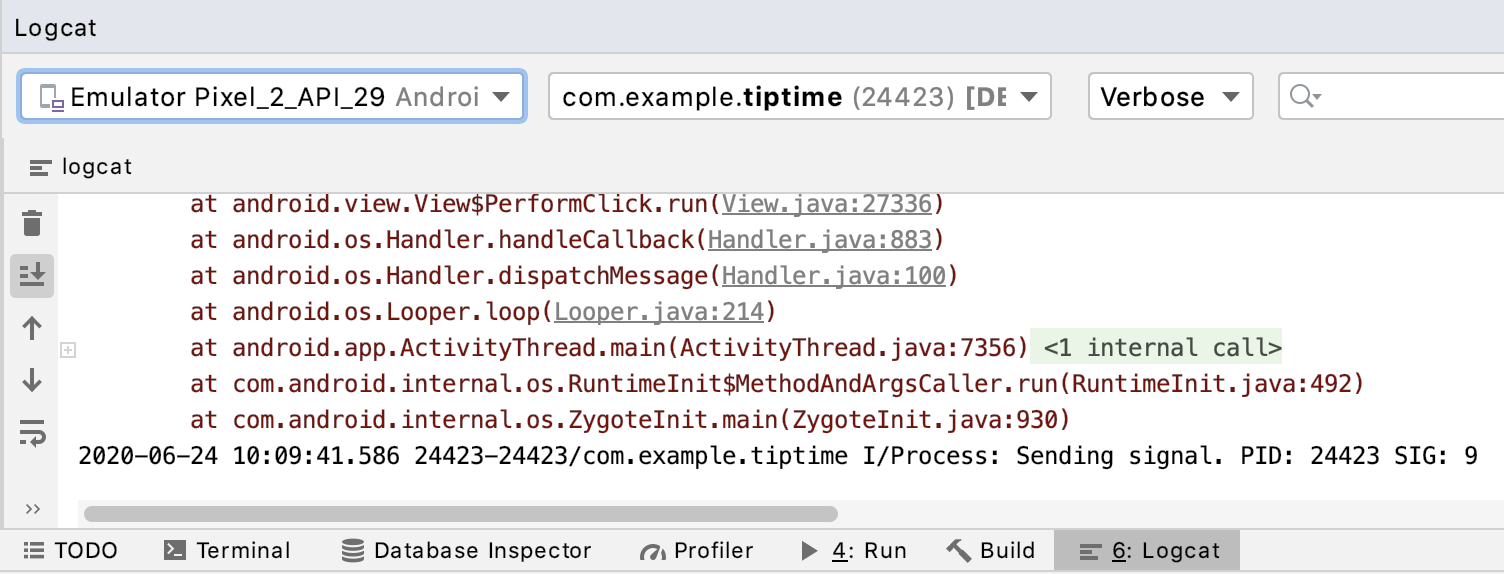
1. Execute o app no emulador, mas, em vez de usar **Run > Run 'app'**, use **Run > Debug 'app'**.
2. Ao tocar em **Calculate**, tente usar diferentes combinações de custo, valor da gorjeta, arredondando para cima ou não, e verifique se você recebeu o resultado esperado para cada caso.
3. Agora tente excluir todo o texto no campo **Cost of Service** e toque em **Calculate**. Ops! O programa falhou.

Depurar a falha

O primeiro passo para lidar com um bug é descobrir o que aconteceu. O Android Studio mantém um [registro](https://developer.android.com/studio/debug/am-logcat?hl=pt-br) do que está acontecendo no sistema, que você pode usar para descobrir o que deu errado.

1. Pressione o botão **Logcat** na parte de baixo do Android Studio ou selecione **View > Tool Windows > Logcat** nos menus.



1. A janela **Logcat** vai aparecer na parte de baixo do Android Studio, preenchida com um texto estranho .

O texto é um *stack trace*, uma lista dos métodos que estavam sendo chamados quando a falha ocorreu.

1. Role para cima no **Logcat** até encontrar uma linha que inclua o texto FATAL EXCEPTION.

2020-06-24 10:09:41.564 24423-24423/com.example.tiptime E/AndroidRuntime: FATAL EXCEPTION: main  
    Process: com.example.tiptime, PID: 24423  
    java.lang.NumberFormatException: empty String  
        at sun.misc.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:1842)  
        at sun.misc.FloatingDecimal.parseDouble(FloatingDecimal.java:110)  
        at java.lang.Double.parseDouble(Double.java:538)  
        at com.example.tiptime.MainActivity.calculateTip(MainActivity.kt:22)  
        at com.example.tiptime.MainActivity$onCreate$1.onClick(MainActivity.kt:17)

1. Deslize para baixo até encontrar a linha com NumberFormatException.

java.lang.NumberFormatException: empty String

À direita, está escrito empty String. O tipo da exceção informa que está relacionado a um formato de número e o restante mostra a base do problema: uma String vazia foi encontrada quando deveria ser uma String com um valor.

1. Continue descendo para ver as chamadas para parseDouble().
2. Abaixo dessas chamadas, encontre a linha com calculateTip. Observe que ela também inclui sua classe MainActivity.

at com.example.tiptime.MainActivity.calculateTip(MainActivity.kt:22)

1. Observe atentamente essa linha e você vai ver exatamente onde a chamada foi feita no código, linha 22 em MainActivity.kt. Se você digitou seu código de outra maneira, o número pode ser diferente. Essa linha converte a String em um Double e atribui o resultado à variável cost.

val cost = stringInTextField.toDouble()

1. Na documentação do Kotlin, procure o método toDouble() que funciona em uma String. O método tem o nome [String.toDouble()](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.text/to-double.html" \t "_blank) (link em inglês).
2. A página exibirá "Exceptions: NumberFormatException, se a string não for uma representação válida de um número".

Uma *exceção* é a maneira como o sistema informa que há um problema. Nesse caso, o problema é que o método toDouble() não conseguiu converter a String vazia em um Double. Mesmo que o EditText tenha inputType=numberDecimal, ainda é possível inserir alguns valores que toDouble() não consegue processar, como uma string vazia.

Saiba mais sobre o valor nulo

Não é possível chamar o método toDouble() em uma string vazia ou em uma string que não representa um número decimal válido. Felizmente, o Kotlin também oferece o método toDoubleOrNull() que gerencia esses problemas. Ele retorna um número decimal, se puder, ou retorna null se houver um problema.

*Null* é um valor especial que significa "sem valor". Ele é diferente de um Double com valor 0.0 ou uma String vazia com zero caractere, "". Null significa que não há valor, nenhum Double ou nenhuma String. Muitos métodos que esperam um valor podem não saber como processar null e serão interrompidos, o que significa que o app falhará. Por isso, o Kotlin tenta limitar onde null é usado. Você aprenderá mais sobre isso nas próximas lições.

Seu app pode verificar se null é retornado pelo método toDoubleOrNull() e fazer coisas de forma diferente nesse caso para que o app não falhe.

1. Em calculateTip(), mude a linha que declara a variável cost para chamar toDoubleOrNull(), em vez de toDouble().

val cost = stringInTextField.toDoubleOrNull()

1. Após essa linha, adicione uma instrução para verificar se cost é null e, em caso afirmativo, para retornar do método. A instrução return significa sair do método sem executar o restante das instruções. Se o método precisar retornar um valor, especifique-o com uma instrução return com uma expressão.

if (cost == null) {  
    return  
}

1. Execute o app novamente.
2. Sem adicionar texto ao campo **Cost of Service**, toque em **Calculate**. Desta vez, seu app não falhou. Bom trabalho! Você encontrou e corrigiu o bug.

Gerenciar outro caso

Nem todos os bugs causam falhas no app. Às vezes, os resultados podem ser confusos para o usuário.

Veja outro caso. O que acontecerá se o usuário:

1. inserir um valor válido para o custo;
2. tocar em **Calculate** para calcular a gorjeta;
3. excluir o custo;
4. tocar em **Calculate** novamente?

Na primeira vez, a gorjeta será calculada e exibida conforme o esperado. Na segunda vez, o método calculateTip() retornará antes devido à verificação que você acabou de adicionar, mas o app ainda mostrará o valor da gorjeta anterior. Isso pode confundir o usuário. Então, adicione algum código para limpar o valor da gorjeta se houver algum problema.

1. Para confirmar o problema, digite um custo válido, toque em **Calculate**. Em seguida, exclua o texto e toque em **Calculate** novamente. O primeiro valor da gorjeta ainda será exibido.
2. Na expressão if adicionada, antes da instrução return, adicione uma linha para definir o atributo text do tipResult como uma string vazia.

if (cost == null) {  
    binding.tipResult.text = ""  
    return  
}

Isso limpará o valor da gorjeta antes de retornar do método calculateTip().

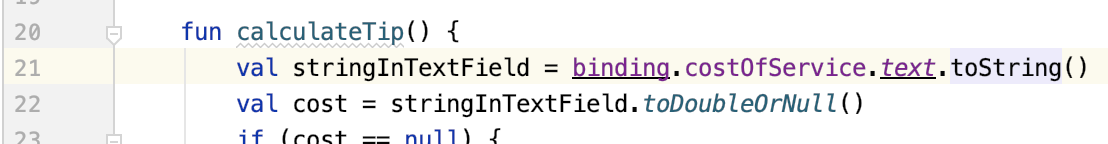
1. Execute o app novamente e repita o processo acima. O primeiro valor da gorjeta desaparecerá quando você tocar em **Calculate** pela segunda vez.

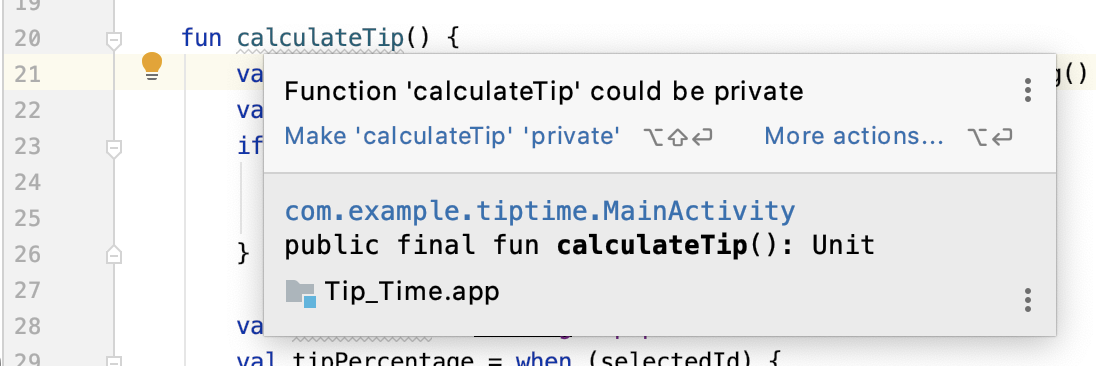
Parabéns! Você criou um app de calculadora de gorjetas funcional para Android e resolveu alguns possíveis problemas.

[6. Adotar boas práticas de programação](https://developer.android.com/codelabs/basic-android-kotlin-training-tip-calculator?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-tip-calculator#5)

Sua calculadora de gorjetas funciona agora, mas você pode melhorar o código e facilitar a utilização no futuro adotando práticas de programação adequadas.

1. Abra MainActivity.kt em **app > java > com.example.tiptime > MainActivity**.
2. Veja o início do método calculateTip(). Ele está sublinhado com uma linha ondulada cinza.



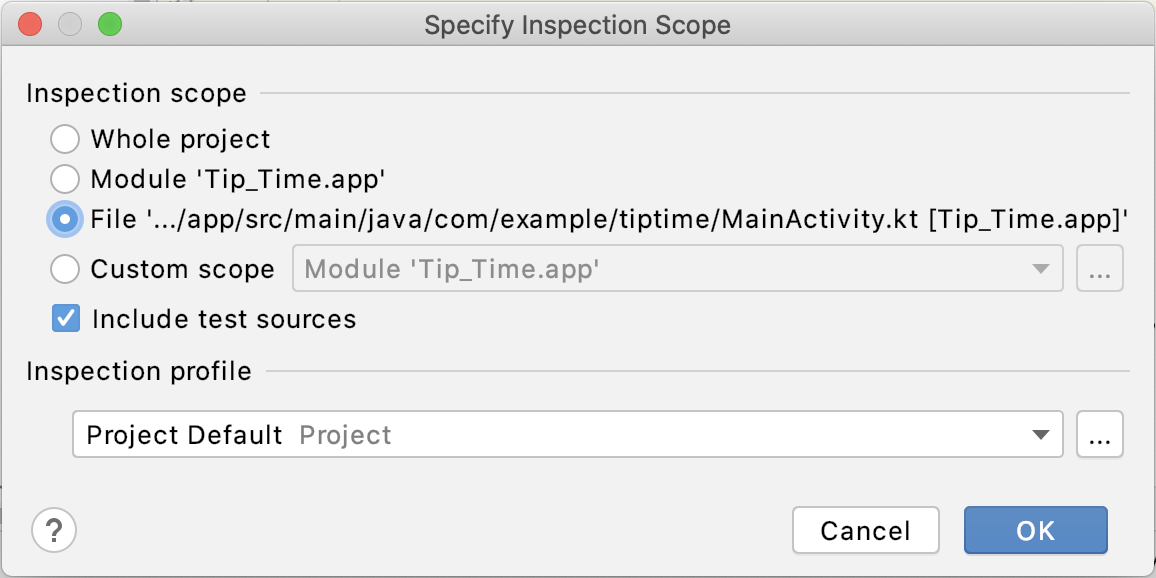
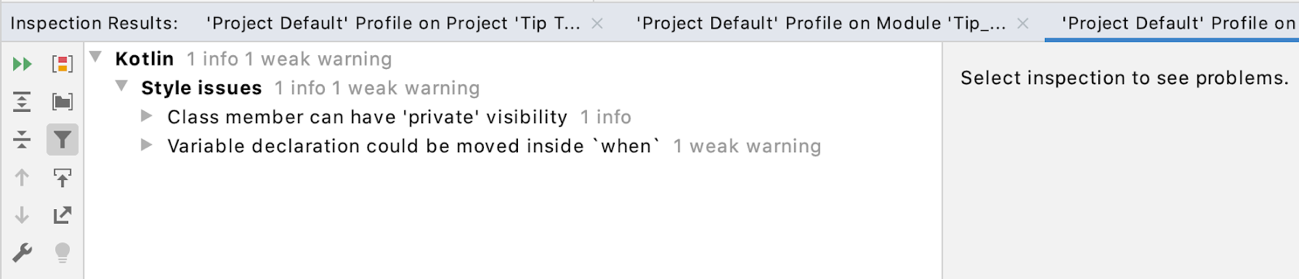
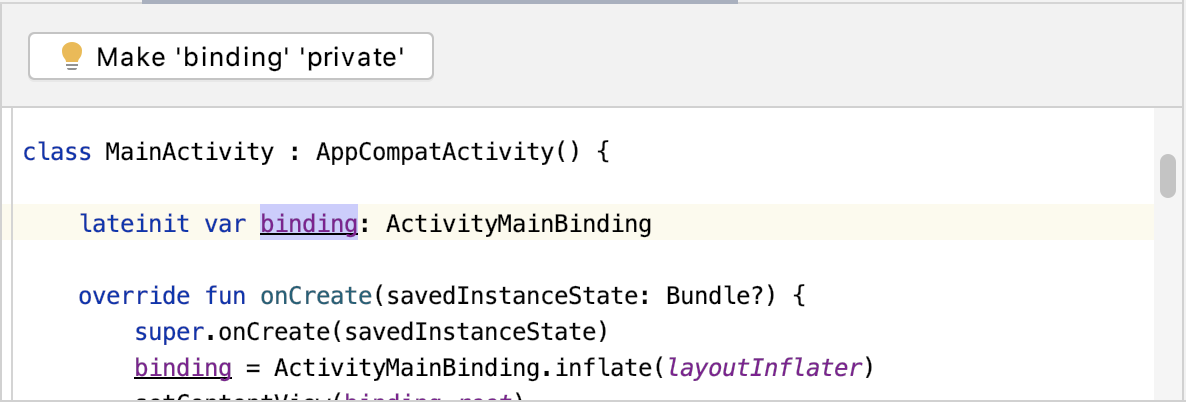
1. Passe o cursor sobre calculateTip(). Você verá a mensagem **Function 'calculateTip' could be private** com uma sugestão abaixo para que você faça com que a função seja particular: **Make 'calculateTip' 'private'**. 

Lembre-se de codelabs anteriores em que private significa que o método ou a variável ficam visíveis apenas para o código dessa classe, neste caso, a MainActivity. Não há motivo para que o código fora da MainActivity chame calculateTip(). Você pode tornar essa função private com segurança.

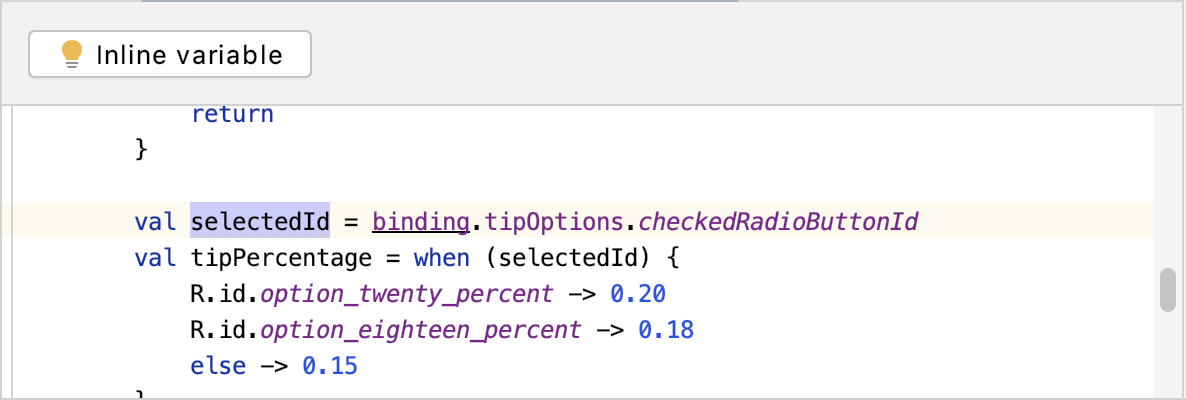
1. Escolha **Make ‘calculateTip' ‘private'** ou adicione a palavra-chave private antes de fun calculateTip(). A linha cinza em calculateTip() desaparecerá.

Inspecionar o código

A linha cinza é muito fina e pode passar despercebida. Você pode procurar essas linhas em todo o arquivo, mas há uma maneira mais simples de garantir que todas elas sejam encontradas.

1. Com o arquivo MainActivity.kt ainda aberto, escolha **Analyze > Inspect Code…** nos menus. Uma caixa de diálogo **Specify Inspection Scope** será exibida. 
2. Escolha a opção que começa com **File** e pressione **OK**. Isso limitará a inspeção apenas ao arquivo MainActivity.kt.
3. Uma janela com os **Inspection Results** será mostrada na parte de baixo.
4. Clique nos triângulos cinza ao lado de **Kotlin** e, ao lado de **Style issues**, até encontrar duas mensagens. A primeira diz **Class member can have 'private visibility'**. 
5. Clique nos triângulos cinza até ver a mensagem **Property ‘binding' could be private** e clique na mensagem. O Android Studio exibirá a parte do código da MainActivity e destacará a variável binding. 
6. Pressione o botão **Make ‘binding' ‘private'**. O Android Studio removerá o problema dos **Inspection Results**.
7. Se você verificar binding no seu código, verá que o Android Studio adicionou a palavra-chave private antes da declaração.

private lateinit var binding: ActivityMainBinding

1. Clique nos triângulos cinza nos resultados até ver a mensagem **Variable declaration could be inlined**. O Android Studio mostrará novamente a parte do código, mas, desta vez, vai destacar a variável selectedId. 
2. Se você analisar seu código, vai notar que o selectedId é usado apenas duas vezes: primeiro, na linha destacada em que o valor de tipOptions.checkedRadioButtonId é atribuído a ele e na próxima linha, na expressão when.
3. Pressione o botão **Inline variable**. O Android Studio substituirá o selectedId na expressão when pelo valor atribuído na linha anterior. Depois, ele removerá a linha anterior completamente, porque ela não é mais necessária.

val tipPercentage = when (binding.tipOptions.checkedRadioButtonId) {  
    R.id.option\_twenty\_percent -> 0.20  
    R.id.option\_eighteen\_percent -> 0.18  
    else -> 0.15  
}

Isso é bem legal! Seu código tem uma linha e uma variável a menos.

Como remover variáveis desnecessárias

O Android Studio não tem mais resultados da inspeção. No entanto, se você olhar seu código atentamente, verá um padrão semelhante ao que acabou de mudar: a variável roundUp é atribuída em uma linha, usada na seguinte e não é usada em nenhum outro lugar.

1. Copie a expressão à direita de = da linha em que roundUp é atribuído.

val roundUp = binding.roundUpSwitch.isChecked

1. Substitua roundUp na próxima linha pela expressão que você acabou de copiar, binding.roundUpSwitch.isChecked.

if (binding.roundUpSwitch.isChecked) {  
    tip = kotlin.math.ceil(tip)  
}

1. Exclua a linha com roundUp porque ela não é mais necessária.

Você fez o mesmo que o Android Studio ajudou a fazer com a variável selectedId. Novamente, o código tem uma linha e uma variável a menos. Essas mudanças são pequenas, mas ajudam a tornar seu código mais conciso e legível.

(Opcional) Eliminar o código repetitivo

Quando seu app estiver em execução corretamente, você poderá procurar outras oportunidades de limpar o código e torná-lo mais conciso. Por exemplo, quando você não insere um valor no custo do serviço, o app atualiza o tipResult para ser uma string vazia "". Quando houver um valor, você usará NumberFormat para formatá-lo. Essa funcionalidade pode ser aplicada em outro lugar no app, por exemplo, para mostrar uma gorjeta de 0.0 em vez de uma string vazia.

Para reduzir a duplicação de código muito semelhante, extraia essas duas linhas de código para funções próprias. Essa função auxiliar pode receber como entrada um valor de gorjeta, como Double, formatá-la e atualizar a TextView do tipResult na tela.

1. Identifique o código duplicado no arquivo MainActivity.kt. Essas linhas de código podem ser usadas várias vezes na função calculateTip(), uma vez para o caso do valor 0.0 e outra para o caso geral.

val formattedTip = NumberFormat.getCurrencyInstance().format(0.0)  
binding.tipResult.text = getString(R.string.tip\_amount, formattedTip)

1. Mova o código duplicado para a própria função. Uma mudança no código é usar uma gorjeta como parâmetro para que o código funcione em vários lugares.

private fun displayTip(tip : Double) {  
   val formattedTip = NumberFormat.getCurrencyInstance().format(tip)  
   binding.tipResult.text = getString(R.string.tip\_amount, formattedTip)  
}

1. Atualize a função calculateTip() para usar a função auxiliar displayTip() e verifique o valor 0.0 também.

MainActivity.kt

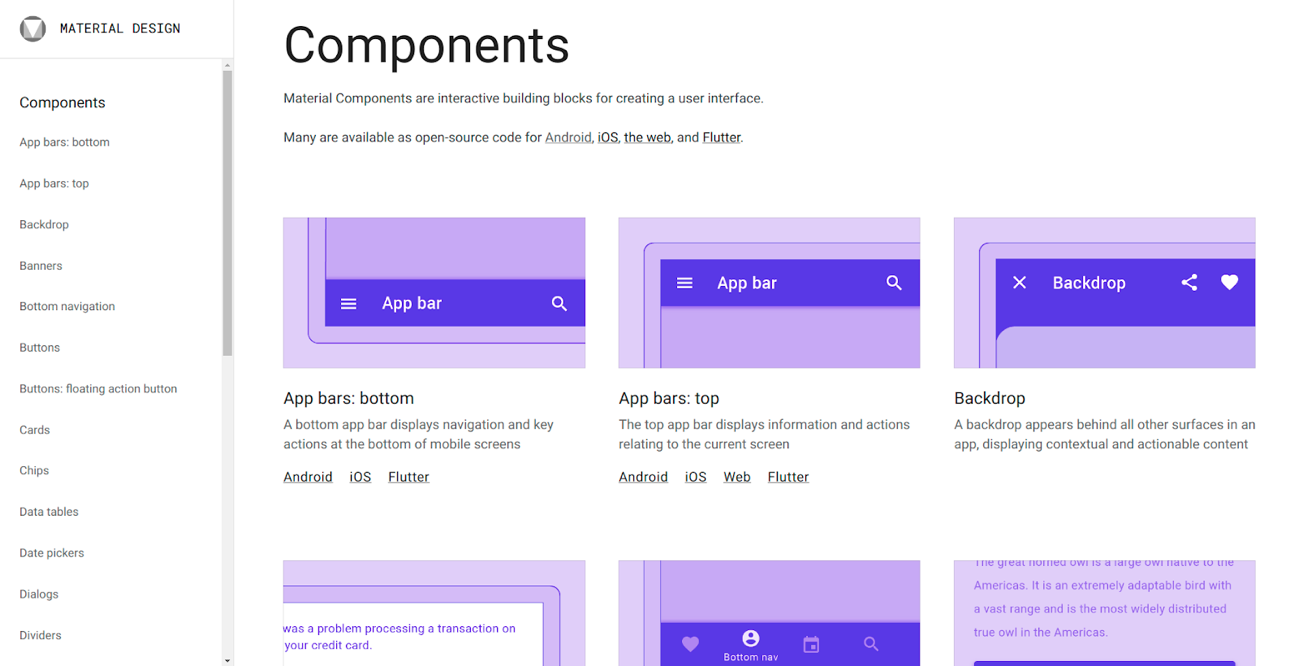
private fun calculateTip() {  
    ...  
  
        // If the cost is null or 0, then display 0 tip and exit this function early.  
        if (cost == null || cost == 0.0) {  
            displayTip(0.0)  
            return  
        }  
  
    ...  
    if (binding.roundUpSwitch.isChecked) {  
        tip = kotlin.math.ceil(tip)  
    }  
  
    // Display the formatted tip value on screen  
    displayTip(tip)  
}

Observação

Mesmo que o app esteja funcionando agora, ele ainda não está pronto para produção. Você precisa fazer mais testes. E precisa adicionar algumas melhorias visuais e seguir as diretrizes do Material Design. Você também var aprender a mudar o tema e o ícone do app nos codelabs a seguir.

[3. Componentes do Material Design](https://developer.android.com/codelabs/basic-android-kotlin-training-polished-user-experience?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-2&hl=pt-br#2)

Os [Componentes do Material Design](https://material.io/components) (link em inglês) são widgets comuns de IU que facilitam a implementação do estilo do Material Design no seu app. A documentação traz informações sobre como usar e personalizar os Componentes do Material Design. Há diretrizes gerais do Material Design para cada componente e orientações específicas para a Plataforma Android para os componentes disponíveis no Android. Os diagramas com etiquetas fornecem informações suficientes para recriar um componente, caso ele não exista na plataforma escolhida.



Ao usar os componentes do Material Design, seu app operará de forma mais consistente, junto dos outros aplicativos do dispositivo do usuário. Dessa forma, os padrões de IU que você aprendeu para um app podem ser transferidos para o próximo. Portanto, os usuários poderão aprender a usar o app muito mais rapidamente. É [recomendável usar componentes do Material Design](https://medium.com/androiddevelopers/we-recommend-material-design-components-81e6d165c2dd) (link em inglês) sempre que possível (em vez de widgets que não são do Material Design). Eles também são mais flexíveis e personalizáveis, como você aprenderá na próxima tarefa.

A biblioteca de Componentes do Material Design (MDC) precisa ser incluída como dependência no projeto. Se você estiver usando o Android Studio 4.1 ou uma versão mais recente, essa linha já estará presente no projeto por padrão. No arquivo build.gradle do seu app, verifique se essa dependência está incluída na versão mais recente da biblioteca. Para mais detalhes, consulte a página [Primeiros passos](https://material.io/develop/android/docs/getting-started) (link em inglês) no site do Material Design.

app/build.gradle

dependencies {  
    ...  
    implementation 'com.google.android.material:material:<version>'  
}

Campos de texto

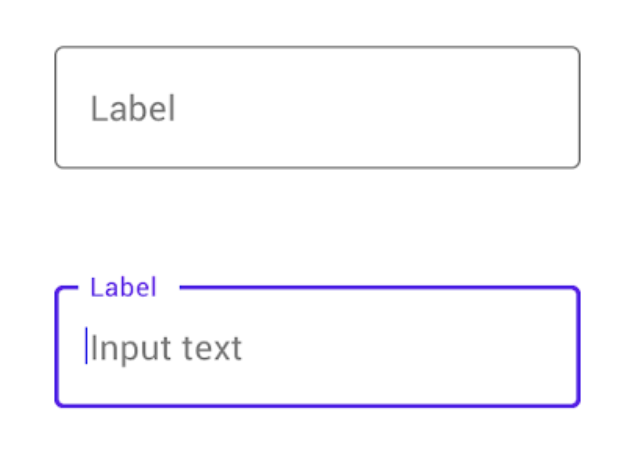
No app de calculadora de gorjetas, na parte superior do layout, você tem um campo EditText para o custo do serviço. Esse campo EditText funciona, mas não segue as orientações recentes do Material Design sobre a aparência e o comportamento dos campos de texto.

Aprenda sobre os componentes novos que você pretende usar no site do Material Design. No guia sobre [Campos de texto](https://material.io/develop/android/components/text-fields) (link em inglês), há dois tipos de campo de texto:

Campo de texto preenchido



Campo de texto delineado



Para criar um campo de texto como mostrado acima, use um TextInputLayout com um TextInputEditText da biblioteca MDC. O campo de texto "Material" pode ser facilmente personalizado para:

* exibir texto ou um marcador de entrada que sempre fica visível;
* exibir um ícone no campo de texto;
* exibir mensagens auxiliares ou de erro.

Na primeira tarefa deste codelab, você substituirá o custo do serviço do EditText por um campo de texto do Material Design, composto por um TextInputLayout e um TextInputEditText.

1. Com o app **Tip Time** aberto no Android Studio, acesse o arquivo de layout activity\_main.xml. Ela precisa conter um ConstraintLayout com o layout da calculadora de gorjetas.
2. Para ver um exemplo de como o XML de um campo de texto do Material Design se parece, volte para a orientação do Android sobre [Campos de texto](https://material.io/develop/android/components/text-fields) (link em inglês). Você verá snippets como este:

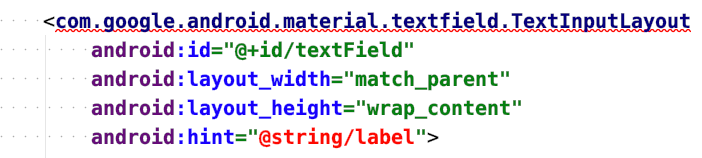
<com.google.android.material.textfield.TextInputLayout  
    android:id="@+id/textField"  
    android:layout\_width="match\_parent"  
    android:layout\_height="wrap\_content"  
    android:hint="@string/label">  
  
    <com.google.android.material.textfield.TextInputEditText  
        android:layout\_width="match\_parent"  
        android:layout\_height="wrap\_content"  
    />  
  
</com.google.android.material.textfield.TextInputLayout>

1. Depois de ver esse exemplo, insira um campo de texto do Material como o primeiro filho do ConstraintLayout (antes do campo EditText). Você excluirá o campo EditText em uma etapa posterior.

É possível digitar isso no Android Studio e usar o preenchimento automático para facilitar o trabalho. Outra opção é copiar o XML de exemplo da página de documentação e colá-lo no seu layout. Observe como o TextInputLayout tem uma visualização filha, TextInputEditText. Lembre-se de que as reticências (...) são usadas para abreviar snippets. Assim, você pode se concentrar nas linhas de XML que realmente mudaram.

<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout  
    ...>  
  
    <com.google.android.material.textfield.TextInputLayout  
        android:id="@+id/textField"  
        android:layout\_width="match\_parent"  
        android:layout\_height="wrap\_content"  
        android:hint="@string/label">  
  
        <com.google.android.material.textfield.TextInputEditText  
            android:layout\_width="match\_parent"  
            android:layout\_height="wrap\_content"  
        />  
  
    </com.google.android.material.textfield.TextInputLayout>  
  
    <EditText  
        android:id="@+id/cost\_of\_service" ... />  
  
    ...

Você verá erros no elemento TextInputLayout. Você ainda não restringiu adequadamente essa visualização no ConstraintLayout pai. Além disso, o recurso de string não é reconhecido. Você corrigirá esses erros nas próximas etapas.



1. Adicione restrições verticais e horizontais ao campo de texto para posicioná-las corretamente no ConstraintLayout pai. Como você ainda não excluiu o EditText, recorte e cole os seguintes atributos do EditText e coloque-os no TextInputLayout: as restrições, o ID do recurso cost\_of\_service, a largura do layout de 160dp, a altura do layout de wrap\_content e o texto de dica @string/cost\_of\_service.

...  
  
<com.google.android.material.textfield.TextInputLayout  
   android:id="@+id/cost\_of\_service"  
   android:layout\_width="160dp"  
   android:layout\_height="wrap\_content"  
   android:hint="@string/cost\_of\_service"  
   app:layout\_constraintStart\_toStartOf="parent"  
   app:layout\_constraintTop\_toTopOf="parent">  
  
   <com.google.android.material.textfield.TextInputEditText  
       android:layout\_width="match\_parent"  
       android:layout\_height="wrap\_content"/>  
  
</com.google.android.material.textfield.TextInputLayout>  
  
...

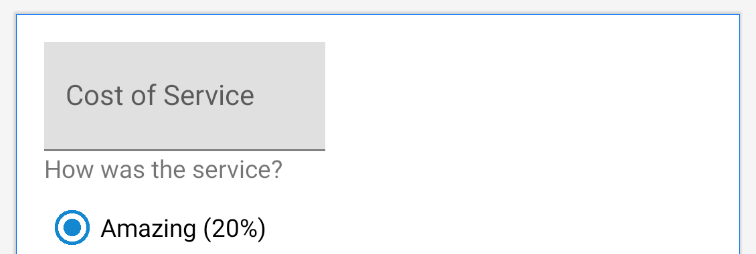
Você pode ver um erro de que o ID cost\_of\_service é igual ao ID do recurso do EditText, mas isso pode ser ignorado por enquanto. O EditText será removido nas próximas etapas.

1. Em seguida, verifique se o elemento TextInputEditText tem todos os atributos adequados. Recorte e cole o tipo de entrada do EditText para o TextInputEditText.. Mude o ID do recurso do elemento TextInputEditText para cost\_of\_service\_edit\_text.

<com.google.android.material.textfield.TextInputLayout ... >  
  
   <com.google.android.material.textfield.TextInputEditText  
       android:id="@+id/cost\_of\_service\_edit\_text"  
       android:layout\_width="match\_parent"  
       android:layout\_height="wrap\_content"  
       android:inputType="numberDecimal" />  
  
</com.google.android.material.textfield.TextInputLayout>

A largura match\_parent e a altura wrap\_content não precisam ser alteradas. Ao definir uma largura como match\_parent, o TextInputEditText terá a mesma largura que o elemento pai TextInputLayout, que é 160dp.

1. Agora que você copiou todas as informações relevantes do EditText, exclua o EditText do layout.
2. Na visualização **Design** do seu layout, essa visualização será exibida. O campo de custo de serviço agora se parece com um campo de texto do Material Design.

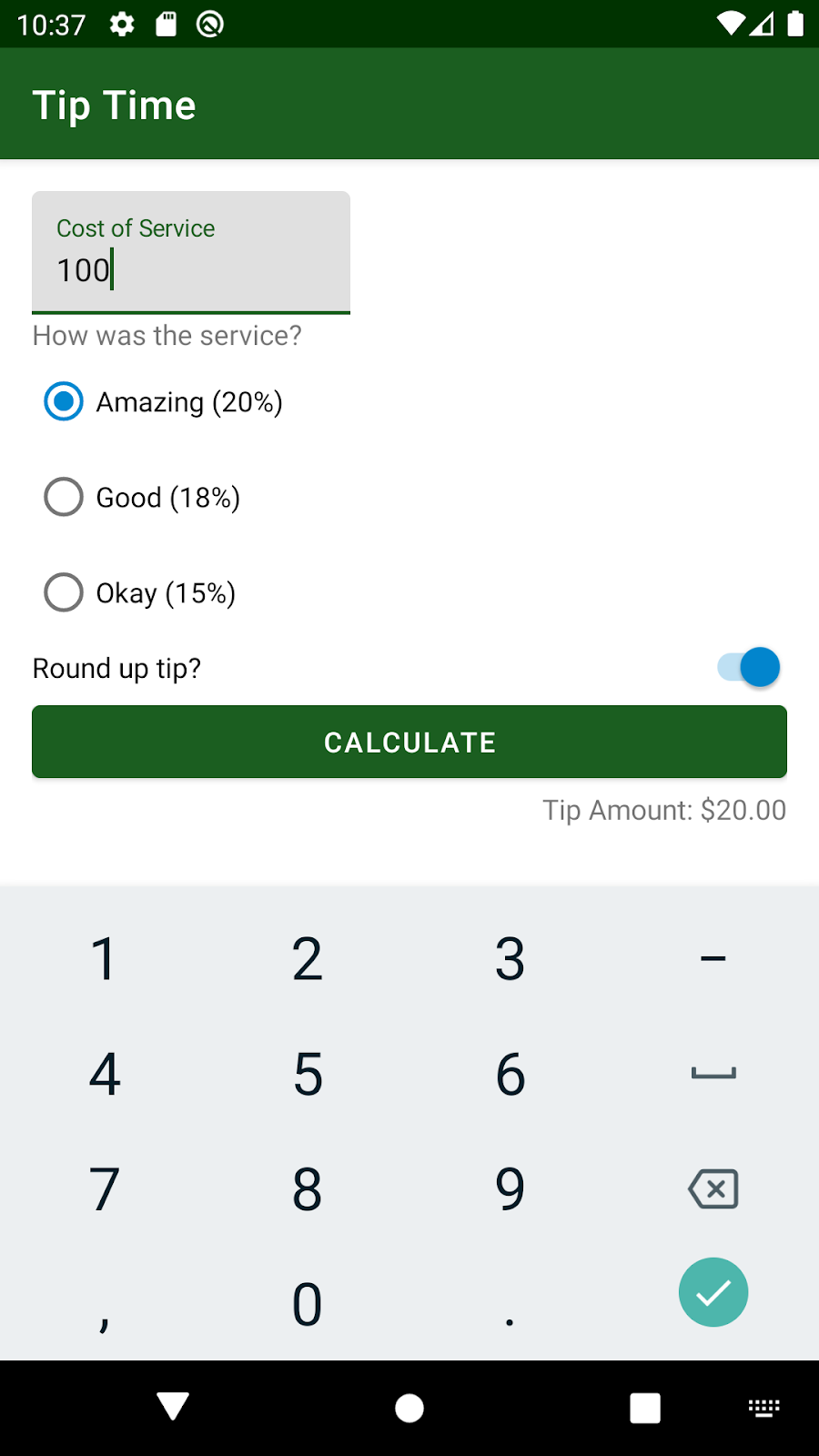


1. Ainda não é possível executar o app porque há um erro no arquivo MainActivity.kt no método calculateTip(). Lembre-se de um codelab anterior que, com a vinculação de visualização ativada para o projeto, o Android criava propriedades em um objeto de vinculação com base no nome do ID do recurso. O campo do qual extraímos o custo de serviço foi alterado no layout XML. Portanto, o código Kotlin precisa ser atualizado de acordo com isso.

Agora, você recuperará a entrada do usuário do elemento TextInputEditText com o ID de recurso cost\_of\_service\_edit\_text. Na MainActivity, use binding.costOfServiceEditText para acessar a string de texto armazenada nela. O restante do método calculateTip() pode permanecer o mesmo.

private fun calculateTip() {  
    // Get the decimal value from the cost of service text field  
    val stringInTextField = binding.costOfServiceEditText.text.toString()  
    val cost = stringInTextField.toDoubleOrNull()  
  
    ...  
}

1. Bom trabalho! Agora, execute o app e teste se ele continua funcionando. O marcador "Cost of service" agora será exibido acima da entrada à enquanto você digita. A gorjeta ainda será calculada conforme o esperado.



**Interruptores**

Nas diretrizes do Material Design, você também encontra orientações sobre [interruptores](https://material.io/components/selection-controls#switches) (link em inglês). Um interruptor é um widget para ativar ou desativar uma configuração.

1. Confira as orientações do Android sobre [interruptores](https://material.io/develop/android/components/switches) (link em inglês) do Material Design. Você aprenderá sobre o widget SwitchMaterial da biblioteca MDC, que fornece estilos do Material Design para interruptores. Se continuar rolando o guia, você verá um exemplo de XML.
2. Para usar SwitchMaterial, é necessário especificar explicitamente o SwitchMaterial no layout e usar o nome do caminho totalmente qualificado.

No layout activity\_main.xml, mude a tag XML de Switch para com.google.android.material.switchmaterial.SwitchMaterial.

...  
  
<com.google.android.material.switchmaterial.SwitchMaterial  
    android:id="@+id/round\_up\_switch"  
    android:layout\_width="0dp"  
    android:layout\_height="wrap\_content" ... />  
  
...

1. Execute o app para verificar se ele ainda é compilado. Não há mudanças visíveis no app. No entanto, uma vantagem de usar o SwitchMaterial da biblioteca MDC (em vez do Switch da Plataforma Android) é que, quando a implementação da biblioteca do SwitchMaterial é atualizada (por exemplo, se as diretrizes do Material Design mudarem), você receberá o widget atualizado gratuitamente, sem nenhuma mudança necessária. Isso ajuda a preparar o app para o futuro.

Agora, você já viu dois exemplos de como sua IU pode se beneficiar do uso de componentes do Material Design prontos para uso e de como alinhar seu app com as diretrizes do Material Design. Lembre-se de que é possível explorar outros componentes do Material Design fornecidos no Android [neste site](https://material.io/develop/android) (link em inglês).

## [4. Ícones](https://developer.android.com/codelabs/basic-android-kotlin-training-polished-user-experience?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-2&hl=pt-br#3)

Ícones são símbolos que ajudam os usuários a entender a interface do usuário, pois eles oferecem uma comunicação visual da função pretendida. Eles costumam se inspirar em objetos do mundo físico que um usuário pode já ter visto. Muitas vezes, o design do ícone reduz o nível de detalhes para o mínimo necessário para ser facilmente reconhecido pelo usuário. Por exemplo, um lápis no mundo físico é usado para escrever, portanto, o ícone correspondente geralmente indica a criação, adição ou edição de um item.

|  |  |
| --- | --- |
| Foto de [Angelina Litvin](https://unsplash.com/@linalitvina?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText) no [Unsplash](https://unsplash.com/?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText" \t "_blank) (links em inglês). |  |

Algumas vezes, os ícones são vinculados a objetos mundanos obsoletos, como é o caso do ícone de disquete. Esse ícone é a representação universal para salvar um arquivo ou um registro de banco de dados. No entanto, embora os disquetes fossem populares na década de 1970, eles deixaram de ser comuns a partir dos anos 2000. Mas o uso contínuo atual demonstra como um visual forte pode transcender a vida útil da forma física.

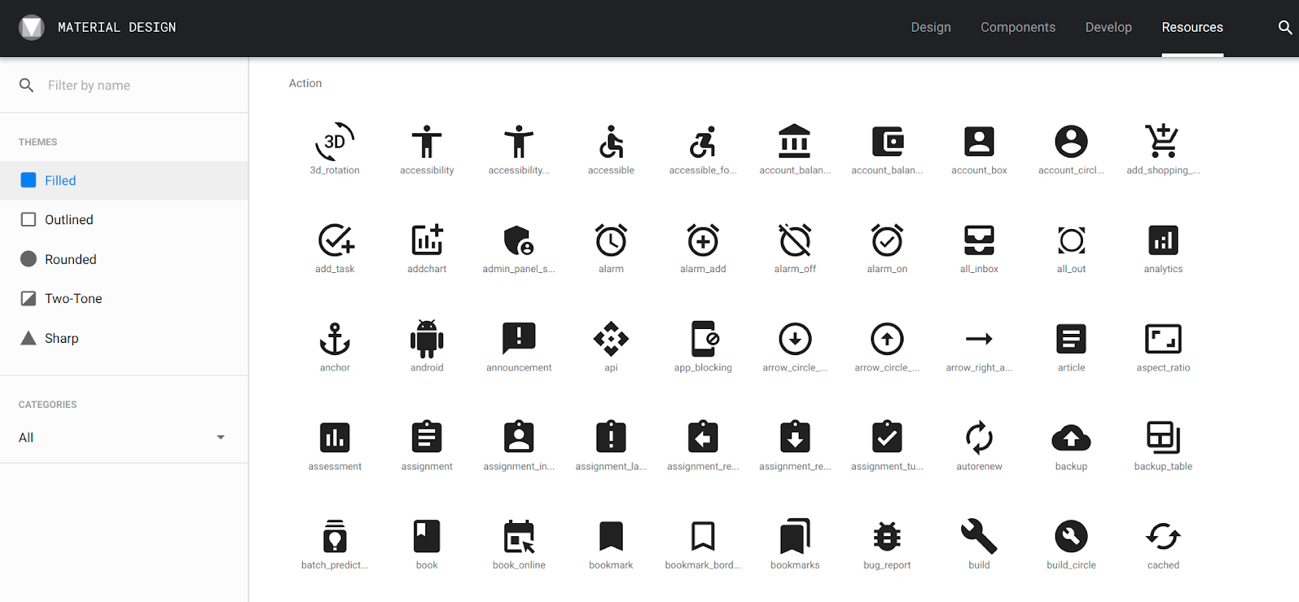
|  |  |
| --- | --- |
| Foto de [Vincent Botta](https://unsplash.com/@0asa?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText) no [Unsplash](https://unsplash.com/?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText" \t "_blank) (links em inglês). |  |

## Como representar ícones no app

Para ícones no app, em vez de fornecer diferentes versões de uma imagem de bitmap para diferentes densidades de tela, a prática recomendada é usar drawables vetoriais. Drawables vetoriais são representados como arquivos XML que armazenam instruções sobre como criar uma imagem em vez de salvar os pixels reais que a compõem. Drawables vetoriais podem ser escalonados verticalmente ou reduzidos sem perda de qualidade visual ou aumento no tamanho do arquivo.

## Ícones fornecidos

O Material Design oferece vários ícones organizados em categorias comuns para a maioria das suas necessidades. [Veja a lista de ícones](https://material.io/resources/icons/?style=baseline).



Esses ícones também podem ser desenhados usando um dos cinco temas (Sólido, Circunscrito, Arredondado, Dois tons ou Nítido) e podem ser coloridos.

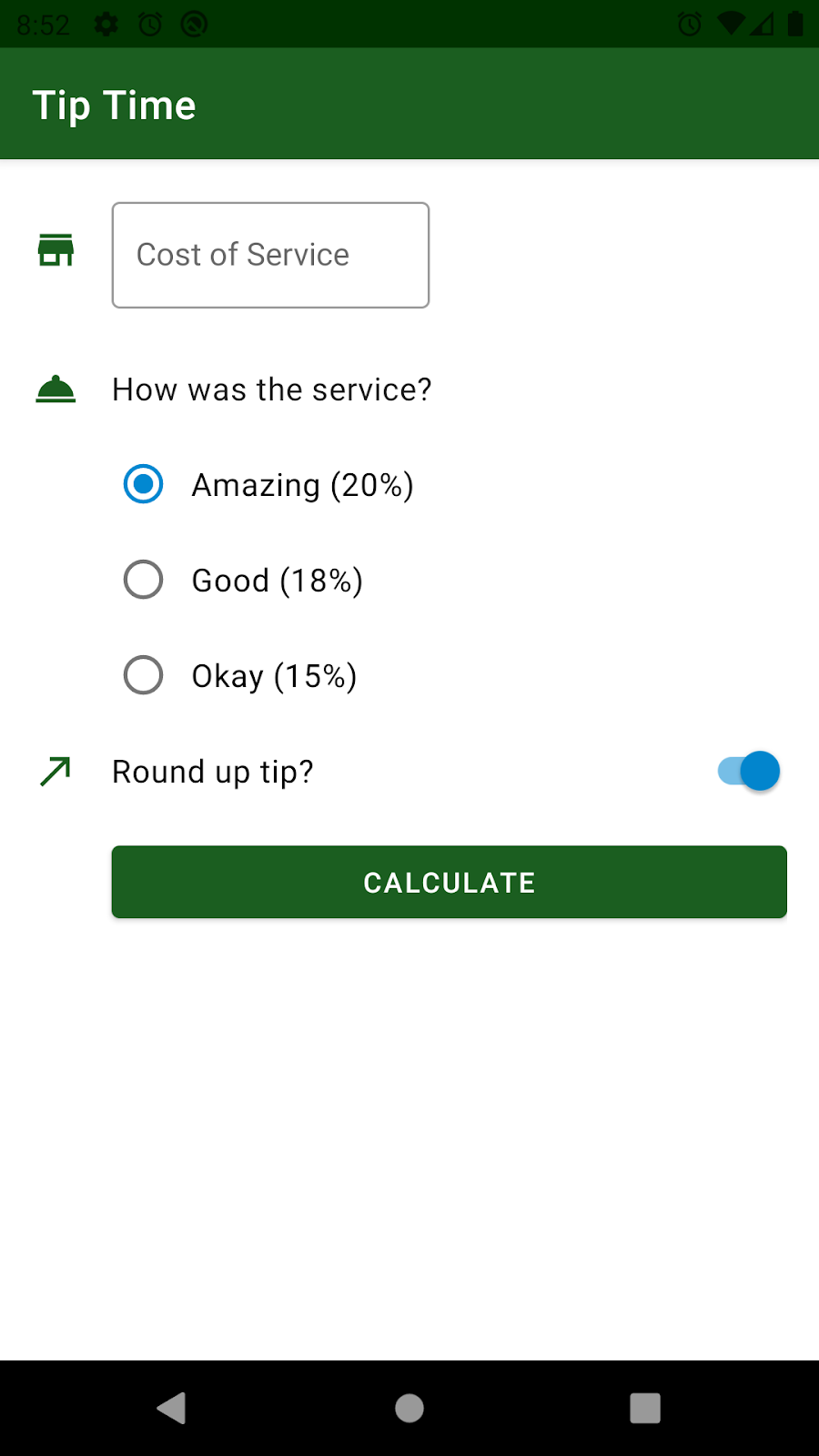
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sólido | Delineado | Arredondado | Dois tons | Nítido |
|  |  |  |  |  |

## Como adicionar ícones

Nesta tarefa, você adicionará três ícones de drawable vetorial ao app:

1. Ícone ao lado do campo de texto do custo do serviço
2. Ícone ao lado da pergunta sobre o serviço
3. Ícone ao lado da solicitação de arredondar a gorjeta

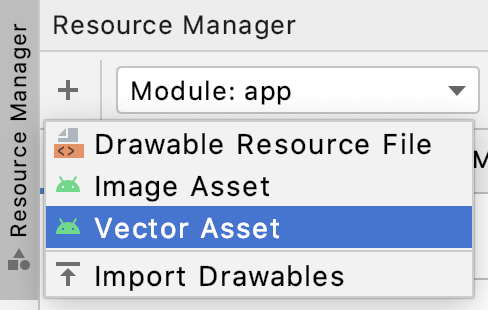
Veja abaixo uma captura de tela da versão final do app. Depois de adicionar os ícones, você ajustará o layout para acomodar o posicionamento desses ícones. Observe como os campos e o botão para calcular a gorjeta são movidos para a direita com a adição dos ícones.



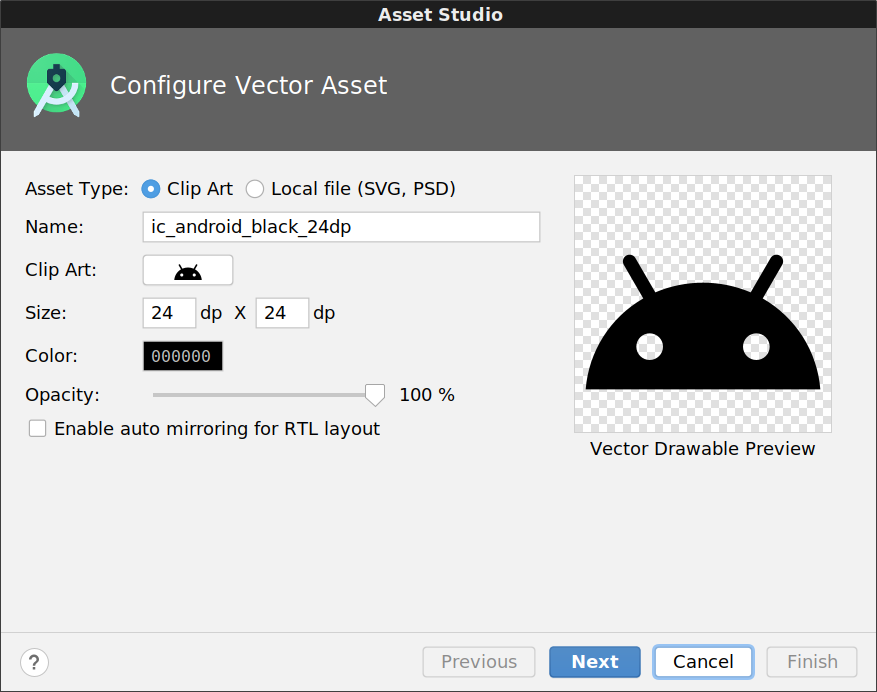
## Adicionar recursos de drawable vetorial

Você pode criar esses ícones como drawables vetoriais diretamente no **Asset Studio** no Android Studio.

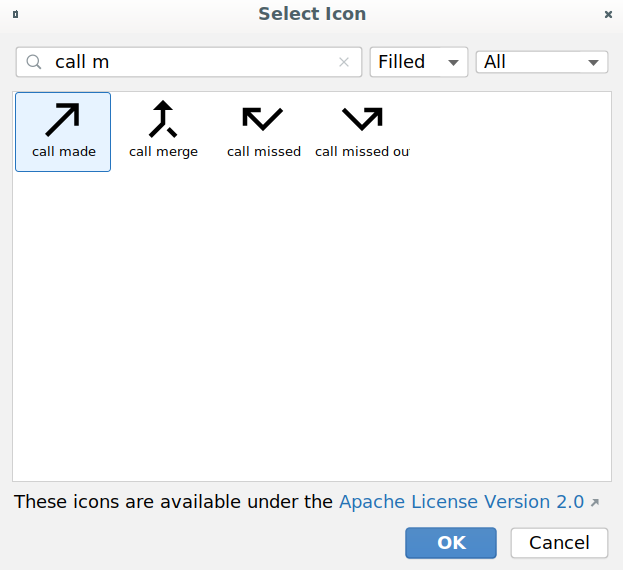
1. Abra a guia **Resource Manager** à esquerda da janela do aplicativo.
2. Clique no ícone + e selecione **Vector Asset**.



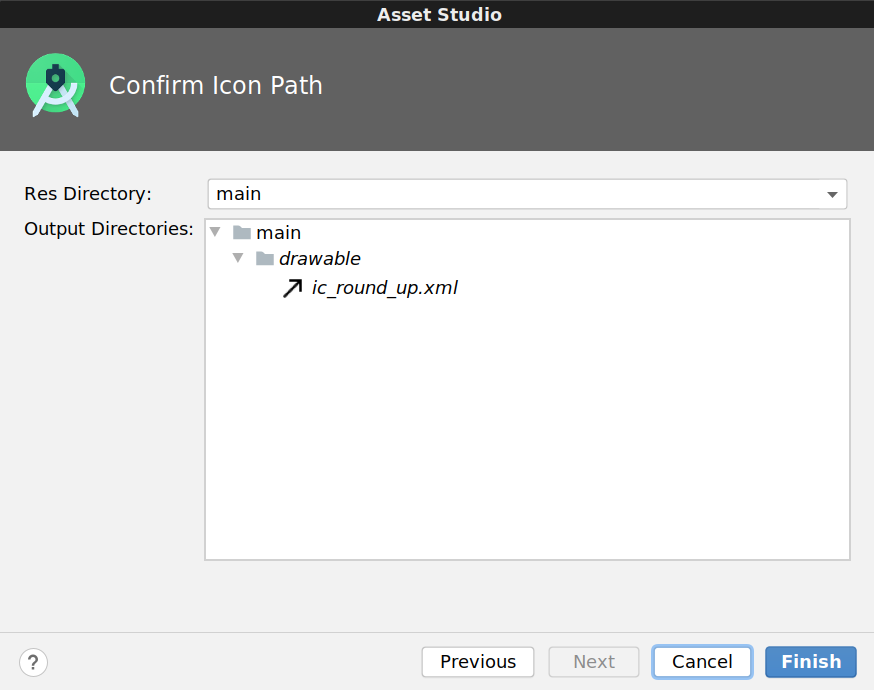
1. Em **Asset Type**, verifique se o botão de opção **Clip Art** está selecionado.



1. Clique no botão ao lado de **Clip Art:** para selecionar outra imagem clip art. Na solicitação exibida, digite "call made" na janela exibida. Você usará esse ícone de seta para a opção de arredondar a gorjeta para cima. Selecione-o e clique em **OK**.



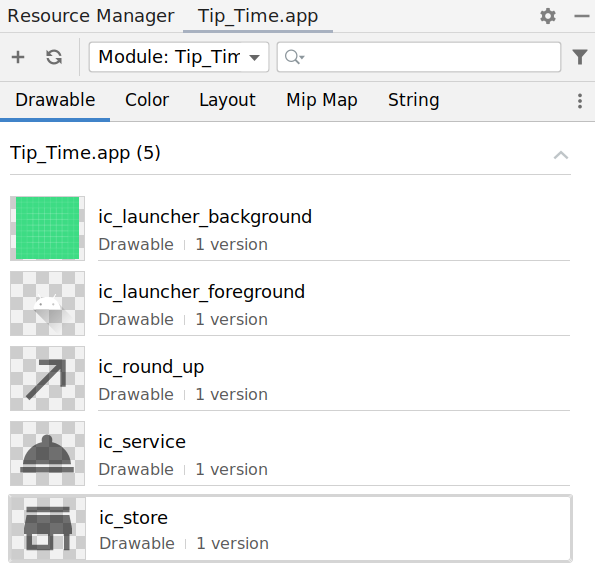
1. Renomeie o ícone como ic\_round\_up. É recomendável usar o prefixo ic\_ ao nomear os arquivos de ícone. Você pode deixar o campo \*\*Size\*\* como 24 dp x 24 dp e o campo \*\*Color\*\* como preto 000000.
2. Clique em **Next**.
3. Aceite o local do diretório padrão e clique em **Finish**.



1. Repita as etapas 2 a 7 para os outros dois ícones:

* **Ícone da pergunta sobre o serviço**: procure o ícone "room service" e salve-o como ic\_service.
* **Ícone de custo do serviço**: procure o ícone "store" e salve-o como ic\_store.

1. Quando terminar, o **Resource Manager** ficará parecido com a captura de tela abaixo. Esses três drawables vetoriais (ic\_round\_up, ic\_service e ic\_store) também serão listados na sua pasta res/drawable.



### **Compatibilidade com versões anteriores do Android**

Você acabou de adicionar drawables vetoriais ao seu app. É importante notar que a compatibilidade com drawables vetoriais na Plataforma Android não foi adicionada até o [Android 5.0 (nível 21 da API)](https://developer.android.com/guide/topics/graphics/vector-drawable-resources?hl=pt-br).

Dependendo de como você configurar o projeto, a versão mínima do SDK para o app Tip Time será a API 19. Isso significa que o app poderá ser executado em dispositivos Android com a versão 19 ou mais recente da Plataforma Android.

Para que o app funcione nessas versões mais antigas do Android (o que é conhecido como compatibilidade com versões anteriores), adicione o elemento vectorDrawables ao arquivo build.gradle do app. Isso permite usar drawables vetoriais em versões da plataforma anteriores à API 21 em vez de converter as imagens em PNGs quando o projeto for criado. Veja [mais detalhes neste link](https://developer.android.com/guide/topics/graphics/vector-drawable-resources?hl=pt-br#vector-drawables-backward-solution).

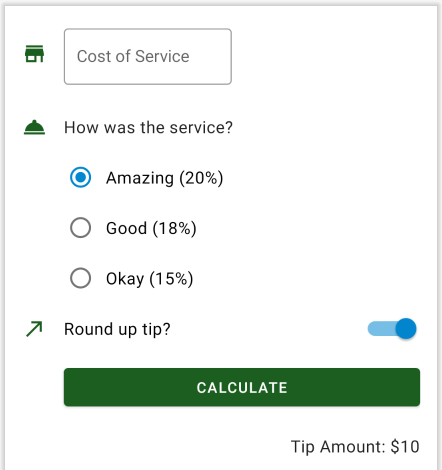
app/build.gradle

android {  
  defaultConfig {  
    ...  
    vectorDrawables.useSupportLibrary = true  
   }  
   ...  
}

Depois de configurar corretamente o projeto, você pode adicionar os ícones ao layout.

## Inserir ícones e posicionar elementos

Você usará ImageViews para exibir ícones no app. Sua IU final ficará assim.



1. Abra o layout activity\_main.xml.
2. Primeiro, posicione o ícone da loja ao lado do campo de texto do custo do serviço. Insira uma nova ImageView como a primeira visualização filha do ConstraintLayout, antes do TextInputLayout.

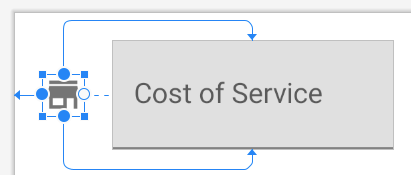
<androidx.constraintlayout.widget.ConstraintLayout  
   ...>  
  
   <ImageView  
       android:layout\_width=""  
       android:layout\_height=""  
  
   <com.google.android.material.textfield.TextInputLayout  
       android:id="@+id/cost\_of\_service"  
       ...

1. Configure os atributos corretos na ImageView para armazenar o ícone ic\_store. Defina o ID como icon\_cost\_of\_service. Defina o atributo app:srcCompat como o recurso drawable @drawable/ic\_store para que você veja uma prévia do ícone próximo à linha do XML. Também defina android:importantForAccessibility="no", porque essa imagem é usada somente para fins decorativos.

<ImageView  
    android:id="@+id/icon\_cost\_of\_service"  
    android:layout\_width="wrap\_content"  
    android:layout\_height="wrap\_content"  
    android:importantForAccessibility="no"  
    app:srcCompat="@drawable/ic\_store" />

Haverá um erro na ImageView porque a visualização ainda não está restrita. Você corrigirá isso em seguida.

1. Posicione o icon\_cost\_of\_service em duas etapas. Primeiro, adicione restrições à ImageView (esta etapa) e atualize as restrições no TextInputLayout ao lado dela (etapa 5). Este diagrama mostra como configurar as restrições.



Na ImageView, você quer que a borda inicial dela fique restrita à borda inicial da visualização mãe (app:layout\_constraintStart\_toStartOf="parent").

O ícone será centralizado verticalmente em comparação com o campo de texto ao lado. Portanto, restrinja a parte superior desta ImageView (layout\_constraintTop\_toTopOf) à parte superior do campo de texto. Restrinja a parte inferior desta ImageView (layout\_constraintBottom\_toBottomOf) à parte inferior do campo de texto. Para se referir ao campo de texto, use o ID de recurso @id/cost\_of\_service. O comportamento padrão é que, quando duas restrições forem aplicadas a um widget na mesma dimensão (como uma restrição superior e inferior), as restrições são aplicadas igualmente. O resultado é que o ícone é centralizado verticalmente em relação ao custo do campo de serviço.

<ImageView  
    android:id="@+id/icon\_cost\_of\_service"  
    android:layout\_width="wrap\_content"  
    android:layout\_height="wrap\_content"  
    android:importantForAccessibility="no"  
    app:srcCompat="@drawable/ic\_store"  
    app:layout\_constraintStart\_toStartOf="parent"  
    app:layout\_constraintTop\_toTopOf="@id/cost\_of\_service"  
    app:layout\_constraintBottom\_toBottomOf="@id/cost\_of\_service" />

O ícone e o campo de texto ainda estarão sobrepostos na visualização **Design**. Isso será corrigido na próxima etapa.

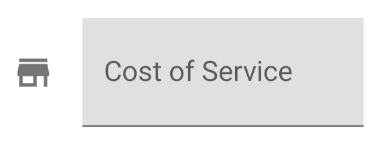
1. Antes de adicionar o ícone, o campo de texto estava posicionado no início do elemento pai. Agora ele precisa ser movido para a direita. Atualize as restrições no campo de texto cost\_of\_service em relação ao icon\_cost\_of\_service.



A borda inicial do TextInputLayout precisa ser restringida à borda final da ImageView (@id/icon\_cost\_of\_service). Para adicionar espaçamento entre as duas visualizações, adicione uma margem inicial de 16dp no TextInputLayout.

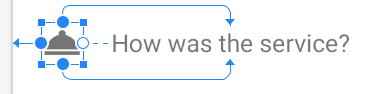
<com.google.android.material.textfield.TextInputLayout  
    android:id="@+id/cost\_of\_service"  
    ...  
    android:layout\_marginStart="16dp"  
    app:layout\_constraintStart\_toEndOf="@id/icon\_cost\_of\_service">  
  
    <com.google.android.material.textfield.TextInputEditText ... />  
  
</com.google.android.material.textfield.TextInputLayout>

Depois de todas essas mudanças, o ícone será posicionado corretamente ao lado do campo de texto.



1. Depois, insira o ícone de sino de serviço ao lado da pergunta "Como foi o serviço?" da TextView. Embora você possa declarar a ImageView em qualquer lugar do ConstraintLayout, seu layout XML será mais fácil de ler se você inserir a nova ImageView no layout XML após o TextInputLayout, mas antes da TextView da service\_question.

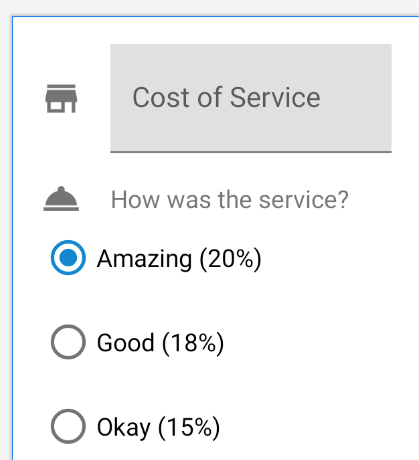
Para a nova ImageView, atribua um ID de recurso de @+id/icon\_service\_question. Defina as restrições adequadas na ImageView e na pergunta sobre o serviço da TextView.



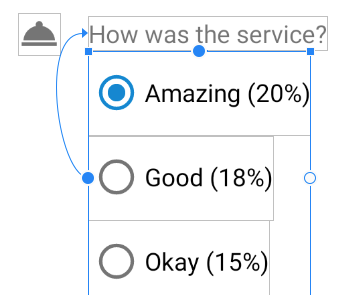
Adicione também uma margem superior de 16dp à service\_question TextView para que haja mais espaço vertical entre a pergunta sobre o serviço e o campo de texto do custo do serviço acima dela.

...  
  
   <ImageView  
        android:id="@+id/icon\_service\_question"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:importantForAccessibility="no"  
        app:srcCompat="@drawable/ic\_service"  
        app:layout\_constraintStart\_toStartOf="parent"  
        app:layout\_constraintTop\_toTopOf="@id/service\_question"  
        app:layout\_constraintBottom\_toBottomOf="@id/service\_question" />  
  
    <TextView  
        android:id="@+id/service\_question"  
        ...  
        android:layout\_marginTop="16dp"  
        app:layout\_constraintStart\_toStartOf="@id/cost\_of\_service"  
        app:layout\_constraintTop\_toBottomOf="@id/cost\_of\_service"/>  
  
...

1. Nesse momento, a visualização **Design** ficará assim. O campo do custo de serviço e da pergunta sobre o serviço (e os respectivos ícones) ficaram ótimos, mas os botões de opção agora estão fora de lugar. Eles não estão alinhados verticalmente com o conteúdo acima deles.



1. Melhore o posicionamento dos botões de opção movendo-os para a direita, abaixo da pergunta sobre o serviço. Isso significa atualizar uma restrição do RadioGroup. Restrinja a borda inicial do RadioGroup à borda inicial da TextView da service\_question. Todos os outros atributos no RadioGroup podem ficar como estão.

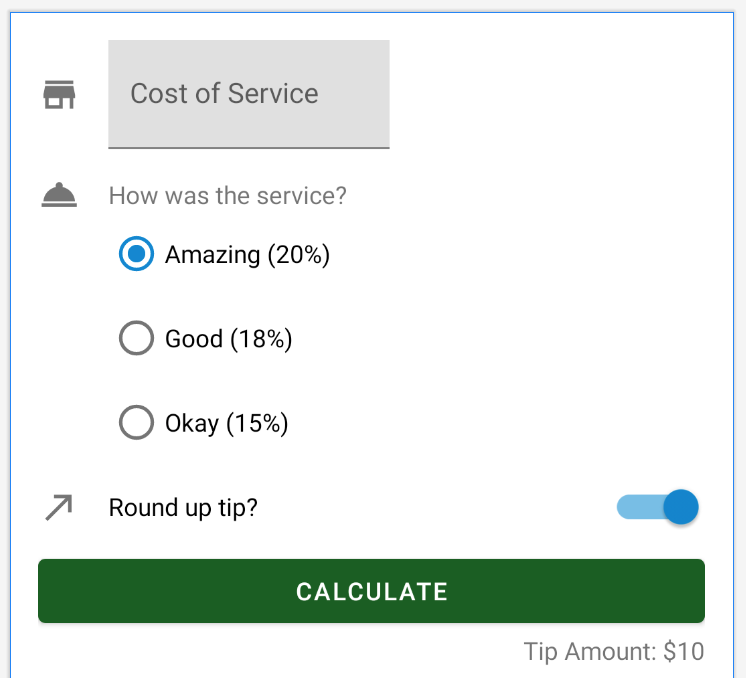


...  
  
<RadioGroup  
    android:id="@+id/tip\_options"  
    ...  
    app:layout\_constraintStart\_toStartOf="@id/service\_question">  
  
...

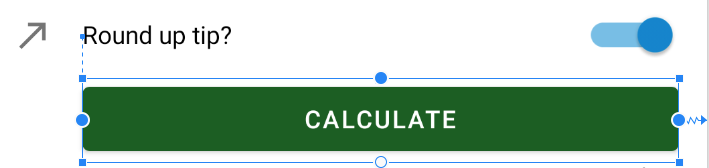
1. Em seguida, adicione o ícone ic\_round\_up ao layout ao lado do interruptor "Arredondar gorjeta?" Tente fazer isso por conta própria e, se tiver dificuldades, consulte o XML abaixo. Você pode atribuir um ID de recurso icon\_round\_up à nova ImageView.
2. No XML de layout, insira uma nova ImageView depois do RadioGroup, mas antes do widget SwitchMaterial.
3. Atribua um ID de recurso de icon\_round\_up à ImageView e defina a srcCompat como o drawable do ícone @drawable/ic\_round\_up. Restrinja o início da ImageView ao início do elemento pai e também centralize o ícone em relação ao SwitchMaterial.
4. Atualize o SwitchMaterial para ficar ao lado do ícone e ter uma margem inicial de 16dp. O XML resultante para icon\_round\_up e round\_up\_switch ficará assim.

...  
  
   <ImageView  
        android:id="@+id/icon\_round\_up"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:importantForAccessibility="no"  
        app:srcCompat="@drawable/ic\_round\_up"  
        app:layout\_constraintStart\_toStartOf="parent"  
        app:layout\_constraintTop\_toTopOf="@id/round\_up\_switch"  
        app:layout\_constraintBottom\_toBottomOf="@id/round\_up\_switch" />  
  
    <com.google.android.material.switchmaterial.SwitchMaterial  
        android:id="@+id/round\_up\_switch"  
        ...  
        android:layout\_marginStart="16dp"  
        app:layout\_constraintStart\_toEndOf="@id/icon\_round\_up" />  
  
...

1. A visualização **Design** precisa ser semelhante a esta. Todos os três ícones estão posicionados corretamente.

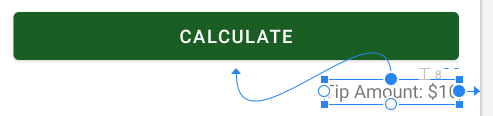


1. Se você comparar isso com a captura de tela final do app, perceberá que o botão para calcular a gorjeta também mudou de lugar para se alinhar verticalmente com o campo do custo de serviço, a pergunta sobre o serviço, os botões de opção e a pergunta de arredondamento da gorjeta. Para fazer isso, restrinja o início do botão para calcular a gorjeta ao início do round\_up\_switch. Adicione também 8dp de margem vertical entre o botão e o interruptor acima dele.



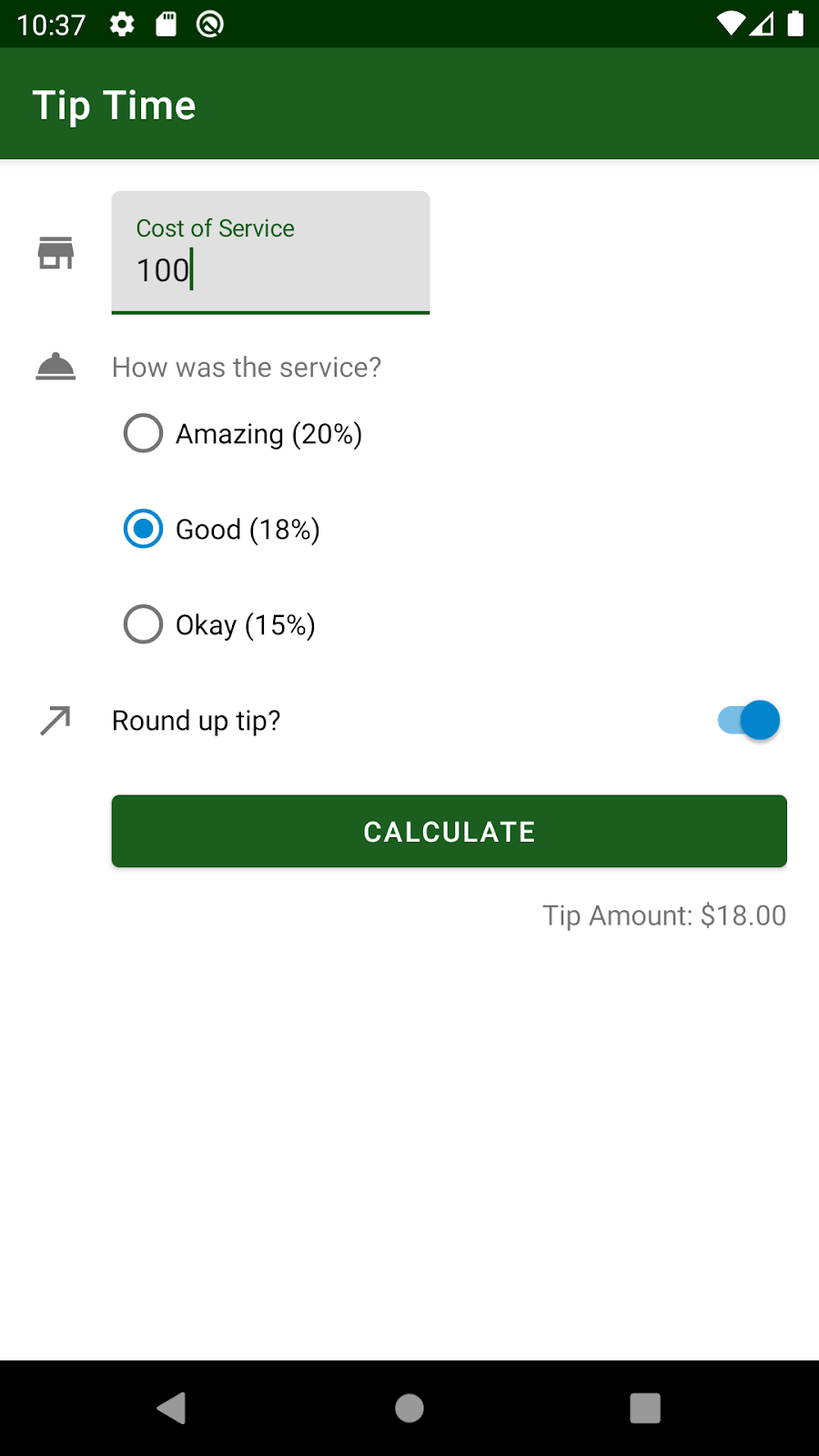
...  
  
<Button  
   android:id="@+id/calculate\_button"  
   ...  
   android:layout\_marginTop="8dp"  
   app:layout\_constraintStart\_toStartOf="@id/round\_up\_switch" />  
  
...

1. Por último, mas não menos importante, posicione o tip\_result adicionando 8dp de margem superior à TextView.



...  
  
<TextView  
   android:id="@+id/tip\_result"  
   ...  
   android:layout\_marginTop="8dp" />  
  
...

1. São muitas etapas. Bom trabalho por concluí-las passo a passo. É necessária muita atenção aos detalhes para alinhar os elementos corretamente no layout, mas isso faz com que o resultado final fique muito melhor! Execute o app. Ele deve ficar parecido com a captura de tela abaixo. Ao alinhar verticalmente e aumentar o espaçamento entre os elementos, eles não ficam tão amontoados.



Você ainda não terminou. Você pode ter notado que o tamanho e a cor da fonte da pergunta sobre o serviço e o valor da gorjeta são diferentes do texto dos botões de opção e do interruptor. Usaremos estilos e temas para torná-los consistentes na próxima tarefa.

[5. Estilos e temas](https://developer.android.com/codelabs/basic-android-kotlin-training-polished-user-experience?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-2&hl=pt-br#4)

Um [estilo](https://medium.com/androiddevelopers/android-styling-themes-vs-styles-ebe05f917578) (link em inglês) é uma coleção de valores de atributos de visualização para um único tipo de widget. Por exemplo, um estilo de uma TextView pode especificar a cor e o tamanho da fonte e a cor do plano de fundo, para citar alguns exemplos. Ao extrair esses atributos em um estilo, você pode aplicá-lo facilmente a várias visualizações no layout e mantê-lo em um único local.

Nesta tarefa, você primeiro criará estilos para os widgets da visualização de texto, do botão de opção e do interruptor.

**Criar estilos**

1. Crie um novo arquivo com o nome styles.xml no diretório **res > values** se ainda não houver um. Crie-o clicando com o botão direito no diretório **valores** e selecione **New > Values Resource File**. Dê o nome styles.xml a ele. O novo arquivo terá o seguinte conteúdo.

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

1. Crie um novo estilo TextView para que o texto seja consistente em todo o app. Defina o estilo uma vez em styles.xml para que ele possa ser aplicado a todas as TextViews no layout. Embora seja possível definir um estilo do zero, você pode estender um estilo TextView já existente usando a biblioteca MDC.

Ao criar o estilo de um componente, geralmente é possível estender o estilo pai do tipo de widget usado. Isso é importante por dois motivos. Primeiro, garante que todos os valores padrão importantes sejam definidos no componente. Depois, o estilo continuará a herdar quaisquer mudanças futuras desse estilo pai.

Você pode nomear seu estilo como quiser, mas há uma convenção recomendada. Se você herdar de um estilo pai do Material Design, nomeie seu estilo de maneira paralela substituindo MaterialComponents pelo nome do seu app (TipTime). Isso move as mudanças para um namespace próprio, o que elimina a possibilidade de conflitos futuros se os componentes do Material Design introduzirem novos estilos. Exemplo:

O nome do seu estilo: Widget.TipTime.TextView herda do estilo pai: Widget.MaterialComponents.TextView

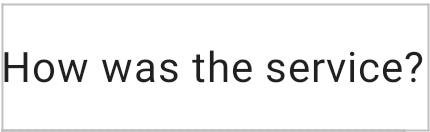
Adicione esta informação ao seu arquivo styles.xml entre as tags de abertura e fechamento de resources.

<style name="Widget.TipTime.TextView" parent="Widget.MaterialComponents.TextView">  
</style>

1. Configure o estilo TextView para que ele modifique os seguintes atributos: android:minHeight,android:gravity, e android:textAppearance.

O android:minHeight define uma altura mínima de 48dp na TextView. A menor altura de qualquer linha precisa ser de 48dp de acordo com as [diretrizes do Material Design](https://material.io/components/lists#specs) (link em inglês).

É possível centralizar o texto na TextView verticalmente definindo o atributo android:gravity. Veja a captura de tela abaixo. Esse atributo de gravidade controla como o conteúdo em uma visualização será posicionado. Como o conteúdo do texto real não ocupa o tamanho total da altura de 48dp, o valor center\_vertical centralizará o texto na TextView verticalmente (mas não mudará a posição horizontal dele). Outros valores de gravidade possíveis incluem center, center\_horizontal, top e bottom. Teste os outros valores de gravidade para ver o efeito no texto.



Defina o valor do atributo de aparência de texto como ?attr/textAppearanceBody1. TextAppearance é um conjunto de estilos predefinidos sobre o tamanho do texto, fontes e outras propriedades do texto. Para ver outras possíveis aparências de texto fornecidas pelo Material Design, consulte esta [lista de escalas de tipo](https://material.io/develop/android/theming/typography) (link em inglês).

<style name="Widget.TipTime.TextView" parent="Widget.MaterialComponents.TextView">  
    <item name="android:minHeight">48dp</item>  
    <item name="android:gravity">center\_vertical</item>  
    <item name="android:textAppearance">?attr/textAppearanceBody1</item>  
</style>

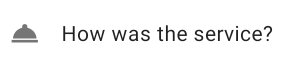
1. Aplique o estilo Widget.TipTime.TextView à TextView da service\_question adicionando um atributo de estilo em cada TextView no arquivo activity\_main.xml.

<TextView  
    android:id="@+id/service\_question"  
    style="@style/Widget.TipTime.TextView"  
    ... />

Antes de adicionar um estilo, a TextView tinha essa aparência com uma fonte de tamanho pequeno e de cor cinza:

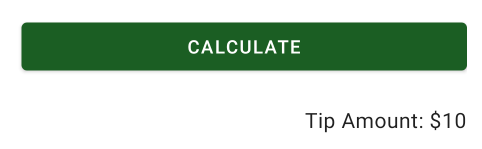


Depois de adicionar o estilo, a TextView ficará assim. Agora, essa TextView é mais consistente com o restante do layout.



1. Aplique o mesmo estilo Widget.TipTime.TextView à TextView do tip\_result.

<TextView  
    android:id="@+id/tip\_result"  
    style="@style/Widget.TipTime.TextView"  
    ... />



**Observação**: se você especificar um atributo em um estilo (por exemplo, definir android:textSize como 18sp) e especificar esse mesmo atributo no arquivo do layout de forma diferente (por exemplo, android:textSize como 14sp), o valor definido no layout (14sp) será aplicado ao que é visto na tela.

1. O mesmo estilo de texto precisa ser aplicado à etiqueta do texto do interruptor. No entanto, não é possível definir um estilo TextView em um widget SwitchMaterial. Os estilos TextView só podem ser aplicados em TextViews. Portanto, crie um novo estilo para o interruptor. Os atributos são os mesmos em termos de minHeight, gravity e textAppearance. O que muda é o nome do estilo e o pai porque você estará herdando o estilo Switch da biblioteca MDC. O nome do estilo também precisa espelhar o nome do estilo pai.

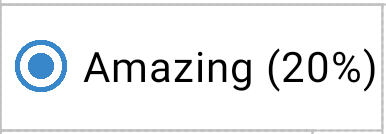
O nome do seu estilo: Widget.TipTime.CompoundButton.Switch. Herda do estilo pai: Widget.MaterialComponents.CompoundButton.Switch

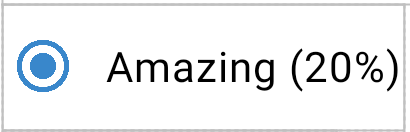
<style name="Widget.TipTime.CompoundButton.Switch" parent="Widget.MaterialComponents.CompoundButton.Switch">  
   <item name="android:minHeight">48dp</item>  
   <item name="android:gravity">center\_vertical</item>  
   <item name="android:textAppearance">?attr/textAppearanceBody1</item>  
</style>

Também é possível especificar atributos adicionais específicos para interruptores nesse estilo, mas não é necessário fazer isso neste app.

1. O texto do botão de opção é o último local em que você quer garantir que o texto apareça visualmente consistente. Não é possível aplicar um estilo TextView ou Switch em um widget RadioButton. Em vez disso, você precisa criar um novo estilo para botões de opção. Você pode estender o estilo RadioButton da biblioteca MDC.

Ao criar o estilo, adicione também padding entre o texto do botão de opção e o visual do círculo. O paddingStart é um novo atributo que você ainda não usou. O padding é a quantidade de espaço entre o conteúdo de uma visualização e os limites dela. O atributo paddingStart define o padding somente no início do componente. Veja a diferença entre 0dp e 8dp de paddingStart em um botão de opção.





<style name="Widget.TipTime.CompoundButton.RadioButton"  
parent="Widget.MaterialComponents.CompoundButton.RadioButton">  
   <item name="android:paddingStart">8dp</item>  
   <item name="android:textAppearance">?attr/textAppearanceBody1</item>  
</style>

1. (Opcional) Crie um arquivo dimens.xml para melhorar o gerenciamento dos valores usados com frequência. É possível criar o arquivo da mesma forma que você criou o arquivo styles.xml acima. Selecione o diretório de valores, clique com o botão direito e selecione **New > Values Resource File**.

Neste app pequeno, você repetiu a configuração de altura mínima duas vezes. Por enquanto, isso é fácil de gerenciar, mas você perderia o controle rapidamente se tivéssemos 4, 6, 10 ou mais componentes compartilhando esse valor. Lembre-se de que o processo de mudar todos eles é tedioso e propenso a erros. É possível criar outro arquivo de recursos útil em **res > values** com o nome dimens.xml que contém dimensões comuns que você pode nomear. Ao padronizar valores comuns como dimensões nomeadas, fica mais fácil gerenciar nosso app. O TipTime é pequeno, por isso não usaremos esse arquivo fora dessa etapa opcional. No entanto, com apps mais complexos em um ambiente de produção, onde você pode trabalhar com uma equipe de design, o arquivo dimens.xml permitirá mudanças nesses valores com mais frequência.

dimens.xml

<resources>  
   <dimen name="min\_text\_height">48dp</dimen>  
</resources>

Você atualizaria o arquivo styles.xml para usar @dimen/min\_text\_height em vez de 48dp diretamente.

...  
<style name="Widget.TipTime.TextView" parent="Widget.MaterialComponents.TextView">  
    <item name="android:minHeight">@dimen/min\_text\_height</item>  
    <item name="android:gravity">center\_vertical</item>  
    <item name="android:textAppearance">?attr/textAppearanceBody1</item>  
</style>  
...

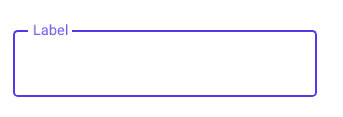
Adicionar estes estilos aos seus temas

Você pode ter percebido que ainda não aplicou os novos estilos RadioButton e Switch aos respectivos widgets. Isso acontece porque você usará atributos do tema para definir o radioButtonStyle e o switchStyle no tema do app. Vamos ver novamente o que é um tema.

Um [tema](https://medium.com/androiddevelopers/android-styling-themes-vs-styles-ebe05f917578) (link em inglês) é um conjunto de recursos nomeados (conhecidos como atributos de tema) que pode ser consultado posteriormente em estilos, layouts etc. É possível especificar um tema para um app, uma atividade ou uma hierarquia de visualização, não apenas uma View. individual. Você modificou o tema do app no arquivo themes.xml definindo atributos de tema como colorPrimary e colorSecondary, que são usados em todo o app e nos componentes dele.

Os radioButtonStyle e switchStyle são outros atributos de tema que você pode definir. Os recursos de estilo fornecidos para esses atributos de tema serão aplicados a todos os botões de opção e interruptores na hierarquia de visualização a que o tema se aplica.

Há também um atributo de tema para o textInputStyle, em que o recurso de estilo especificado será aplicado a todos os campos de entrada de texto no app. Para que um TextInputLayout apareça como um campo de texto descrito (como mostrado nas diretrizes do Material Design), há um estilo OutlinedBox definido na biblioteca MDC como Widget.MaterialComponents.TextInputLayout.OutlinedBox. Esse é o estilo que você usará.



1. Modifique o arquivo themes.xml para que o tema se refira aos estilos desejados. A definição de um atributo de tema é feita da mesma maneira que você declarou os atributos de tema colorPrimary e colorSecondary em um codelab anterior. No entanto, desta vez, os atributos de tema relevantes são textInputStyle, radioButtonStyle e switchStyle. Você usará os estilos criados anteriormente para RadioButton e Switch com o estilo do campo de texto OutlinedBox do Material Design.

Copie o seguinte no res/values/themes.xml para a tag de estilo do tema do app.

<item name="textInputStyle">@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox</item>  
<item name="radioButtonStyle">@style/Widget.TipTime.CompoundButton.RadioButton</item>  
<item name="switchStyle">@style/Widget.TipTime.CompoundButton.Switch</item>

1. Seu res/values/themes.xml ficará assim Adicione comentários no XML se quiser (indicado por <!- e -->).

<resources xmlns:tools="http://schemas.android.com/tools">  
  
    <!-- Base application theme. -->  
    <style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">  
        ...  
        <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>  
        <!-- Text input fields -->  
        <item name="textInputStyle">@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox</item>  
        <!-- Radio buttons -->  
        <item name="radioButtonStyle">@style/Widget.TipTime.CompoundButton.RadioButton</item>  
        <!-- Switches -->  
        <item name="switchStyle">@style/Widget.TipTime.CompoundButton.Switch</item>  
    </style>  
  
</resources>

1. Faça as mesmas mudanças no tema escuro em **themes.xml (noite)**. Seu arquivo res/values-night/themes.xml ficará assim.

<resources xmlns:tools="http://schemas.android.com/tools">  
  
    <!-- Application theme for dark theme. -->  
    <style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">  
        ...  
        <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>  
        <!-- Text input fields -->  
        <item name="textInputStyle">@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox</item>  
        <!-- For radio buttons -->  
        <item name="radioButtonStyle">@style/Widget.TipTime.CompoundButton.RadioButton</item>  
        <!-- For switches -->  
        <item name="switchStyle">@style/Widget.TipTime.CompoundButton.Switch</item>  
    </style>  
  
</resources>

1. Execute o app e veja as mudanças. O estilo OutlinedBox é muito melhor para o campo de texto, e todo o texto agora está consistente.

