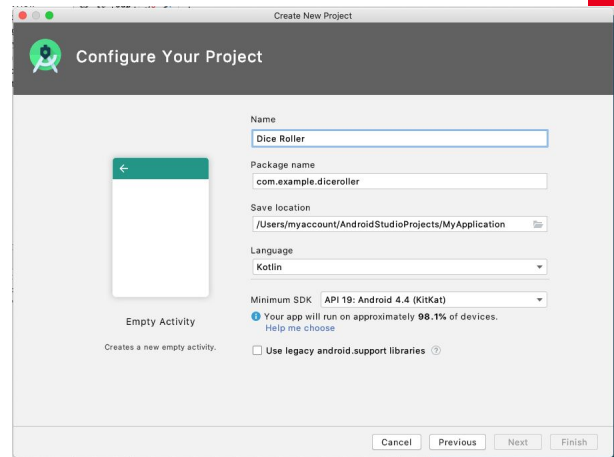


PdAM-I

Programação de Aplicativos Mobile - I

Passo a Passo

Parte 1. Layout



Criar o layout

Neste codelab, você criará 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.

Criar o layout

11:44

Tip Time

Cost of Service

How was the service?

☒ Amazing (20%)

☐ Good (18%)

☐ OK (15%)

Round up tip?☒

CALCULATE

Tip Amount

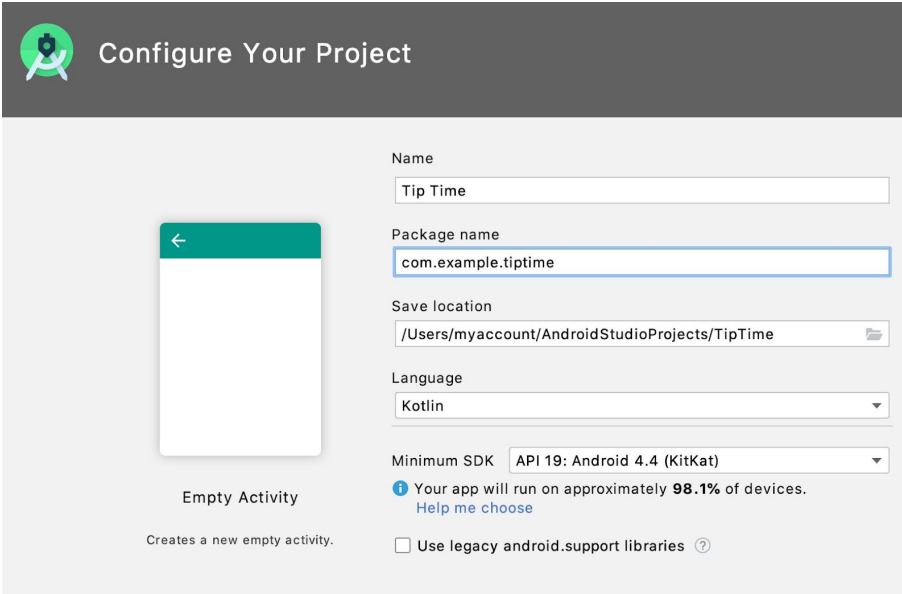
Criar o layout


Você usará estes elementos de IU fornecidos pelo Android:

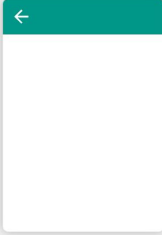
- EditText: para inserir e editar texto.
- TextView: para exibir um texto, como uma pergunta sobre o serviço e o valor da gorjeta.
- RadioButton: um botão de opção selecionável para cada opção de gorjeta.
- RadioGroup: para agrupar os botões de opção.
- Switch: um botão liga/desliga para definir se a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

Criar o layout

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**.



 Configure Your Project



Empty Activity
Creates a new empty activity.


Name
Tip Time


Package name
com.example.tiptime

Save location
/Users/myaccount/AndroidStudioProjects/TipTime

Language
Kotlin

Minimum SDK
API 19: Android 4.4 (KitKat)

 Your app will run on approximately **98.1%** of devices.
[Help me choose](#)

☐ Use legacy android.support libraries 

Criar o layout

Em vez de usar o **Layout Editor** que você já conhece, crie o layout do aplicativo modificando o [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.

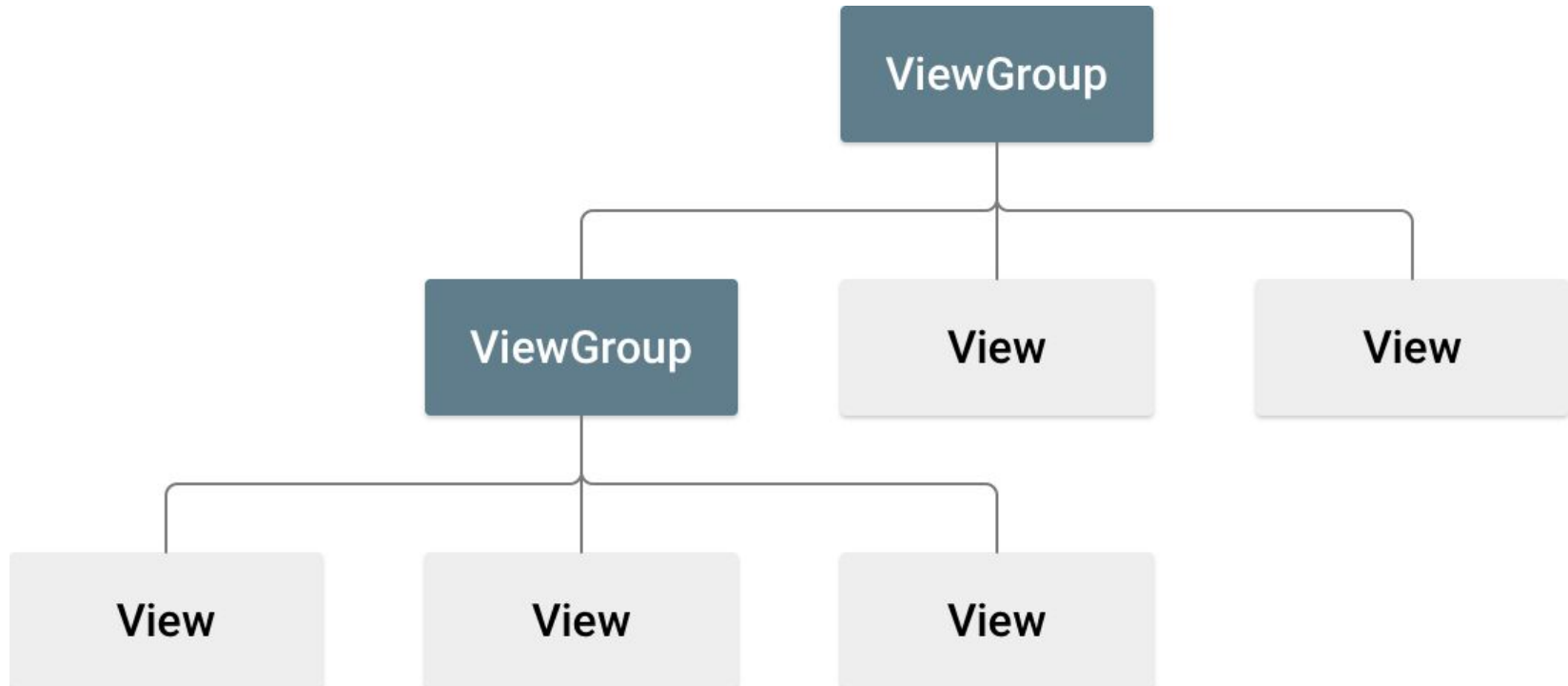
Você vai visualizar e editar o arquivo XML que define o layout da IU desse app. XML significa *eXtensible Markup Language*, que é uma maneira de descrever dados usando um documento de texto. Como o XML é extensível e muito flexível, ele é usado para muitas coisas, incluindo a definição do layout da IU 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.

Criar o layout

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

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

Criar o layout



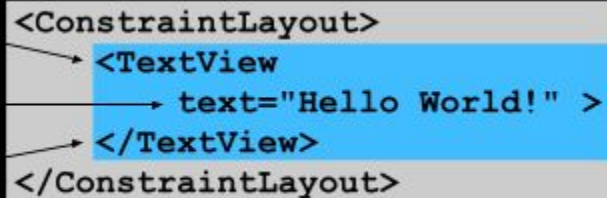
Criar o layout

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

Criar o layout

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>
```



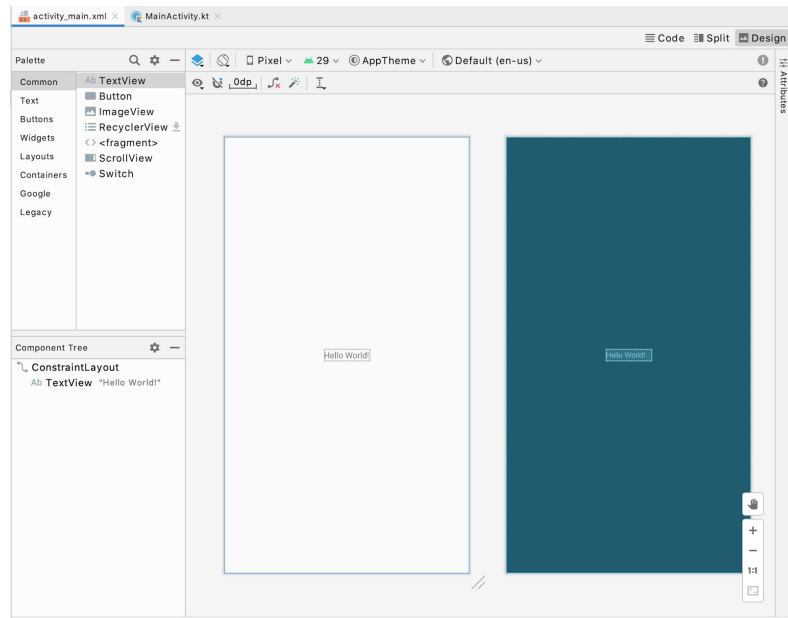
```
<ConstraintLayout>
    <TextView
        text="Hello World!" >
    </TextView>
</ConstraintLayout>
```



TextView (child)

Criar o layout

1. Abra o arquivo `activity_main.xml` (**res > layout > activity_main.xml**).
2. Você verá que o app exibe uma `TextView` com "Hello World!" em um `ConstraintLayout`, como visto em projetos anteriores criados com este modelo.
3. As opções para as visualizações **Code**, **Split** e **Design** ficam no canto superior direito do Layout Editor.
4. Selecione a visualização **Code**.
5. 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.
6. 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.



Criar o layout

```
<?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>
```

Criar o layout

Tags, elementos e atributos XML

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

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

Criar o layout

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

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

Criar o layout

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>
```

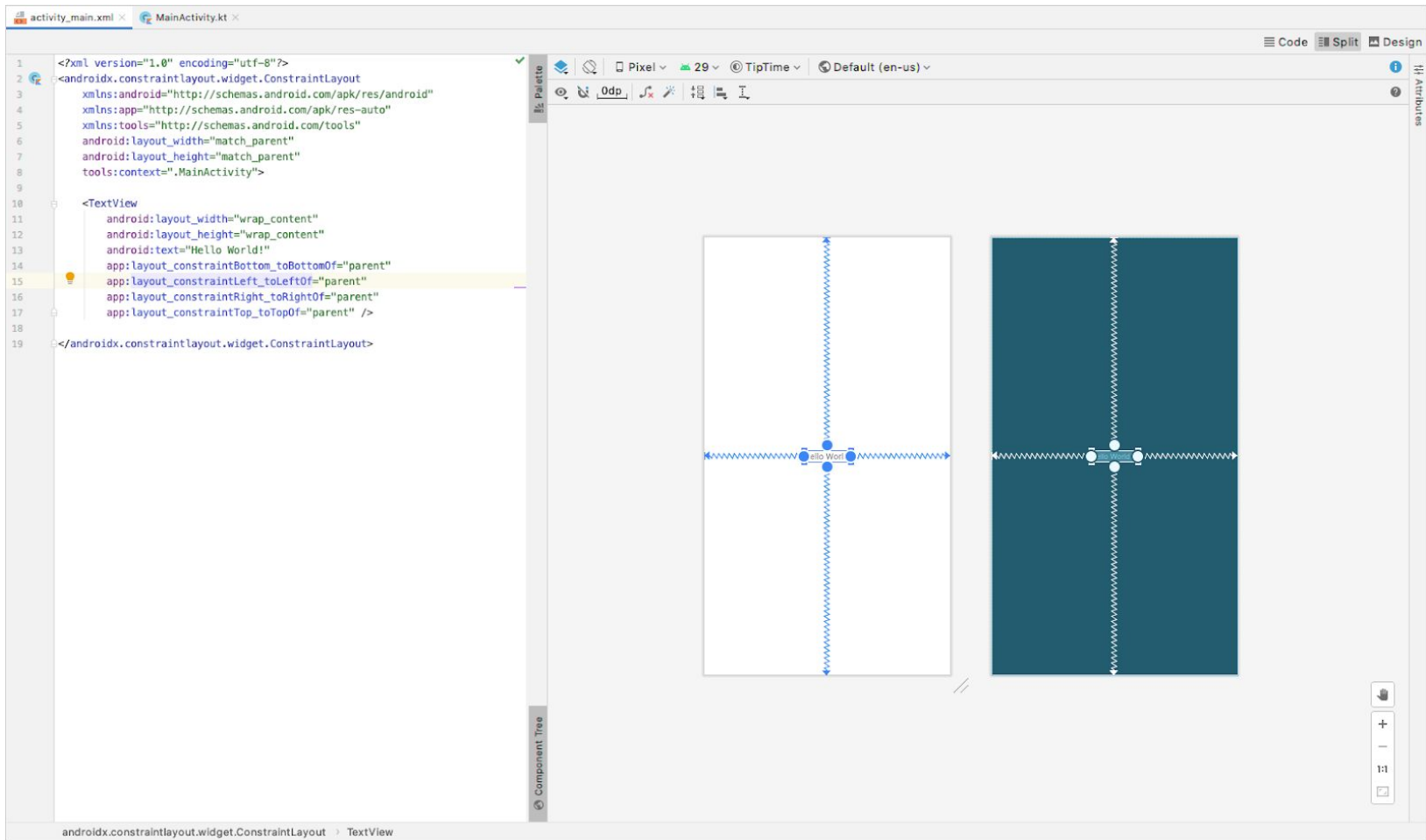

Criar o layout

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>
```

Criar o layout

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



Criar o layout

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>
```

Criar o layout

2. 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">
```

Adicionando um TextView

Adicionar um campo de texto do custo do serviço

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

1. Veja a documentação da classe [EditText](#) 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.

Adicionando um TextView

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"
    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>
```

Adicionando um TextView

4. Observe que a EditText está sublinhada em vermelho.
5. 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.

<EditText

an
an
an
an

This view is not constrained vertically: at runtime it will jump to the top unless you add a vertical constraint

Suppress: Add tools:ignore="MissingConstraints" attribute ↵ ⌵ ⌵ More actions... ↵ ⌵

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

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


Adicionando um TextView

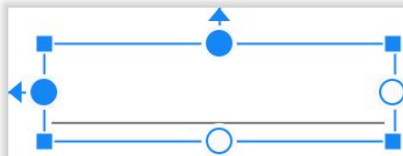
Com as novas restrições adicionadas, o elemento EditText ficará assim:

```
<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"/>
```

Adicionando um TextView

Verifique todos os atributos EditText que você colocou 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`.
3. 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.
4. 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.
5. 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.).



Adicionando um TextView

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

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

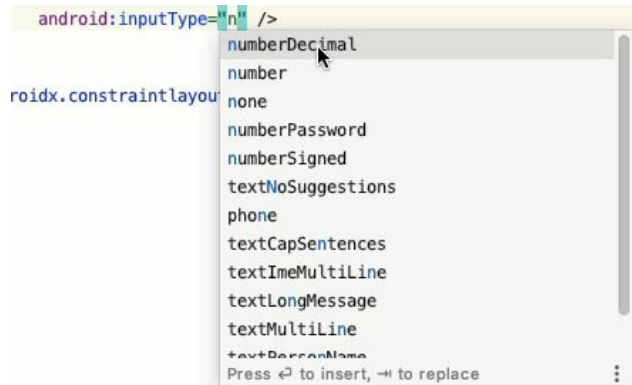
Ao fazer referência aos IDs de recurso no código do app, use `R.<type>.<name>`, por exemplo, `R.string.roll`. Para IDs de View, o `<type>` é `id`, por exemplo: `R.id.button`.

Adicionando um TextView

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

6. Apague a palavra text, mas mantenha as aspas.
7. Digite number no lugar dela. Após digitar a letra "n", o Android Studio mostrará uma lista de possíveis opções que incluem "n".
8. Escolha numberDecimal, o que limita os valores a números com um ponto decimal.

```
android:inputType="numberDecimal"
```



Adicionando um TextView

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

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

```
android:hint="Cost of Service"
```

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



Cost of Service



Adicionando um TextView

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>
```



Cost of Service



Adicionando um TextView

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 ajudarão você.

1. Após o término da tag EditText, />, adicione uma nova linha e comece a digitar <TextView.
2. Selecione TextView nas sugestões, e o Android Studio adicionará 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 "Como foi o serviço?"
5. Feche a tag com />.
6. No **Design Editor**, a TextView se sobrepõe à EditText.

```
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Como foi o serviço?"  
    android:layout_width="match_parent"
```

Adicionando um TextView

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.

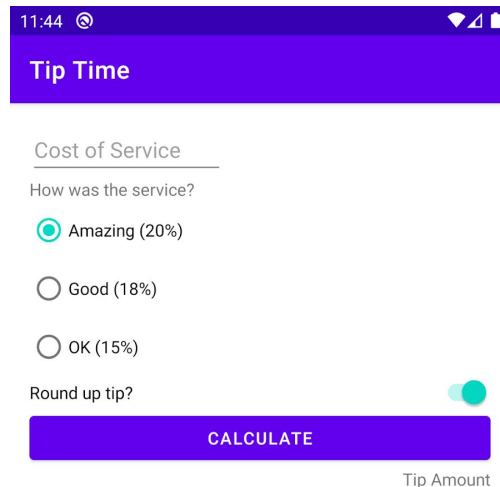
7. Adicione uma restrição horizontal à TextView para restringir a borda inicial da visualização à borda inicial do pai.

```
app:layout_constraintStart_toStartOf="parent"
```

8. 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"
```

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



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"
    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>
```

Adicionando o RadioButton

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%)

Os botões de opção permitem que o usuário selecione uma opção em um conjunto delas. Os botões de opção são usados para conjuntos opcionais que são mutuamente exclusivos, em casos em que você acredita que o usuário precisa ver todas as opções disponíveis lado a lado. Se não for necessário mostrar todas as opções lado a lado, use um *switch*.

Para isso é interessante lermos a documentação:

<https://developer.android.com/guide/topics/ui/controls/radiobutton?hl=pt-br>

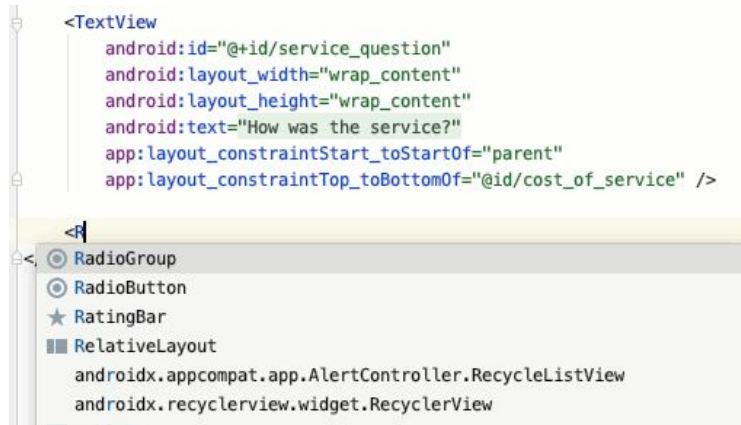
Adicionando o RadioButton

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.



Adicionando o RadioButton

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 elemento `RadioGroup` terá outros elementos dentro dele, então você pode 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.



Adicionando o RadioButton

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>
```

Adicionando o RadioButton

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>
```

2. Configure a layout_width e a layout_height como wrap_content.
3. Atribua um ID de recurso de @+id/option_twenty_percent ao RadioButton.
4. Defina o texto como Amazing (20%).
5. Feche a tag com />.

Adicionando o RadioButton

```
<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 a Good (18%) e Okay (15%)?

Adicionando o RadioButton

```
<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="OK (15%" />

</RadioGroup>
```


Adicionando o RadioButton

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

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

1. No `RadioGroup`, defina o atributo `android:checkedButton` como `@id/option_twenty_percent`.

```
<RadioGroup
    android:id="@+id/tip_options"
    android:checkedButton="@id/option_twenty_percent"
    ...
```

Adicionando o RadioButton

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.

```
<?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">

    <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%" />

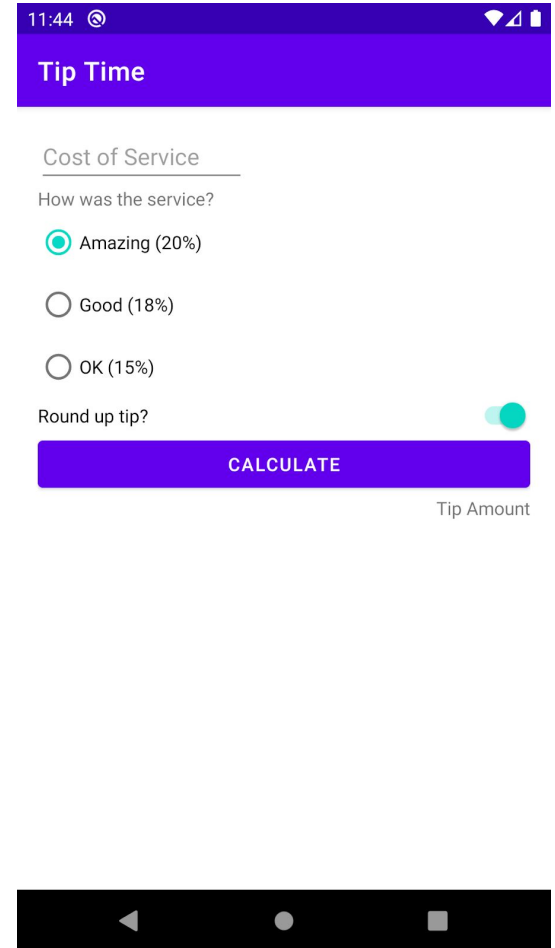
    <RadioButton
        android:id="@+id/option_fifteen_percent"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="OK (15%" />

</RadioGroup>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Terminando o Layout

Você está na última parte do layout.

Falta adicionar um Switch, um Button e uma TextView para exibir o valor da gorjeta.



Terminando o Layout

Em seguida, você usará um widget Switch 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.

Terminando o Layout

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

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

9. Defina o atributo `android:checked` como `true`.

Terminando o Layout

<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="parent"  
    app:layout_constraintTop_toBottomOf="@id/tip_options" />
```

Terminando o Layout

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

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

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

Terminando o Layout

<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" />
```

Terminando o Layout

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

1. Adicione uma `TextView` com um ID de recurso chamado `tip_result` e o texto `Tip Amount`.
2. Restrinja a borda final da `TextView` à borda final do pai.
3. Restrinja a borda superior da visualização à borda inferior do botão **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" />
```

Tip Time

Cost of Service

How was the service?

☒ Amazing (20%)

☐ Good (18%)

☐ Okay (15%)

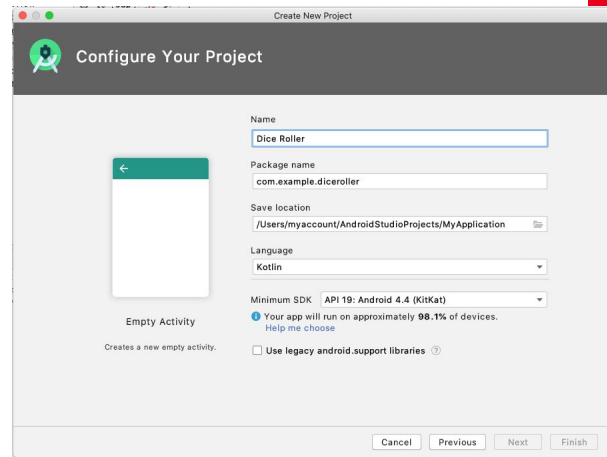
Round up tip?



CALCULATE

Passo a Passo

Parte 2. Otimização



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 `okay_service` para tornar os nomes mais descritivos.

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

2. Se a janela **Project** não for exibida, clique na guia **Project** no lado esquerdo da janela.
3. 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.

```
<?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="@string/cost_of_service"
        android:inputType="numberDecimal"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/service_question"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/how_was_the_service"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/cost_of_service" />
```



```

<RadioGroup
    android:id="@+id/tip_options"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checkedButton="@id/option_twenty_percent"
    android:orientation="vertical"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/service_question">
    <RadioButton
        android:id="@+id/option_twenty_percent"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/amazing_service" />
    <RadioButton
        android:id="@+id/option_eighteen_percent"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/good_service" />

    <RadioButton
        android:id="@+id/option_fifteen_percent"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/ok_service" />
</RadioGroup>
<Switch
    android:id="@+id/round_up_switch"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:checked="true"
    android:text="@string/round_up_tip"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/tip_options" />

<Button
    android:id="@+id/calculate_button"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="@string/calculate"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/round_up_switch" />

<TextView
    android:id="@+id/tip_result"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/tip_amount"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toBottomOf="@id/calculate_button" />
</androidx.constraintlayout.widget.ConstraintLayout>

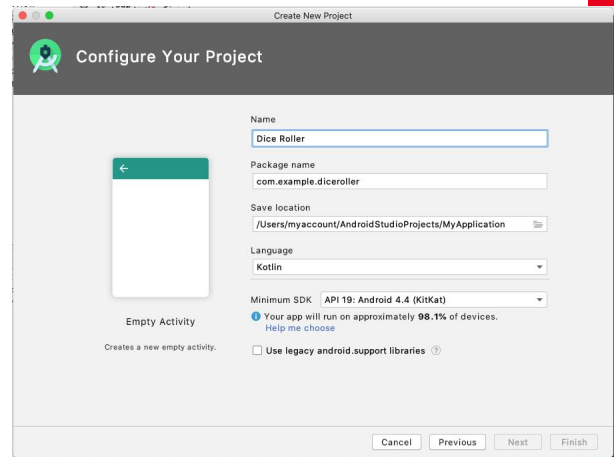
```

res/values/strings.xml

```
<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>
```

Passo a Passo

Parte 3. Adicionando uma lógica



Calculando o Custo do Serviço

O app tem toda a IU necessária para uma calculadora de gorjetas, mas nenhum código para calcular a gorjeta. O app tem o botão **Calculate**, mas ele ainda não funciona.

O campo EditText **Cost of Service** permite que o usuário insira o custo do serviço.

Uma lista de RadioButtons permite que o usuário selecione a porcentagem da gorjeta e um Switch permite que ele escolha se a gorjeta será arredondada para cima ou não.

O valor da gorjeta é exibido em uma TextView. Por fim, um Button **Calculate** solicita que o app colete os dados dos outros campos e calcule o valor da gorjeta

Calculando o Custo do Serviço

Estrutura do projeto do app

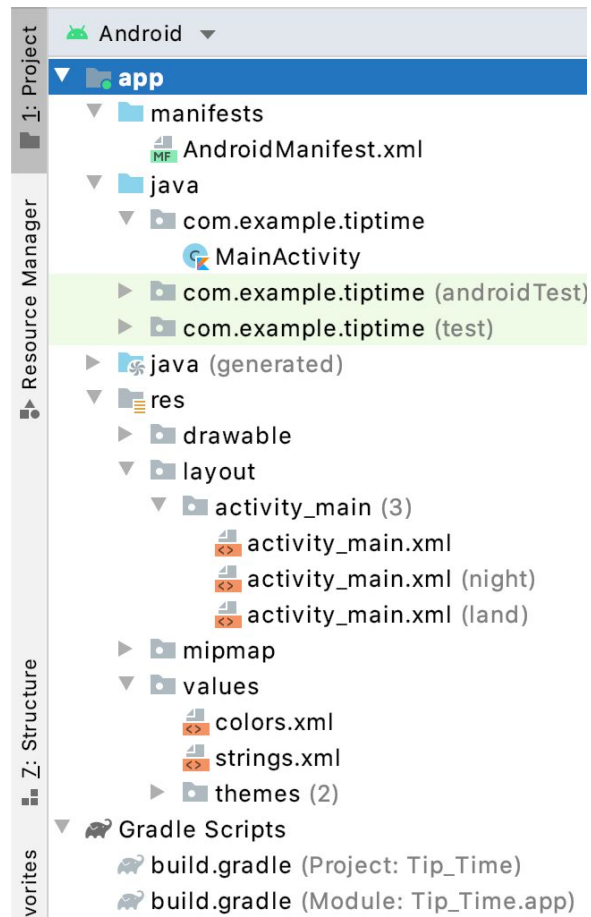
Um projeto de app no seu ambiente de desenvolvimento integrado consiste em várias partes, incluindo o código Kotlin, layouts XML e outros recursos, como strings e imagens. Antes de fazer mudanças no app, é bom aprender como ele funciona.

1. Abra o projeto **Tip Time** no Android Studio.
2. Se a janela **Project** não for exibida, selecione a guia **Project** no lado esquerdo do Android Studio.
3. Escolha a visualização **Android** na lista suspensa, caso ela ainda não esteja selecionada.

Calculando o Custo do Serviço

O *Gradle* é o sistema de compilação automatizado usado pelo Android Studio. Sempre que você mudar o código, adicionar um recurso ou fizer outras mudanças no app, o Gradle descobrirá o que mudou e executará as etapas necessárias para recriá-lo. Ele também instala o app no emulador ou dispositivo físico e controla a execução.

Há outras pastas e arquivos envolvidos na criação do app, mas esses são os principais itens usados neste codelab e os seguintes.



Vinculando as Views

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 um método, `findViewById()`, que faz exatamente o que você precisa, ele 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. 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.

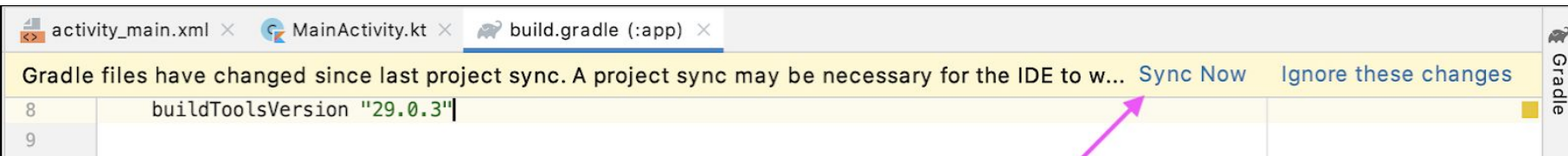
Ativar a vinculação de visualizações

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  
  
}
```

Vinculando as Views

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



Após alguns instantes, você verá uma mensagem na parte inferior da janela do Android Studio, **Gradle sync finished**. Feche o arquivo build.gradle, se quiser.

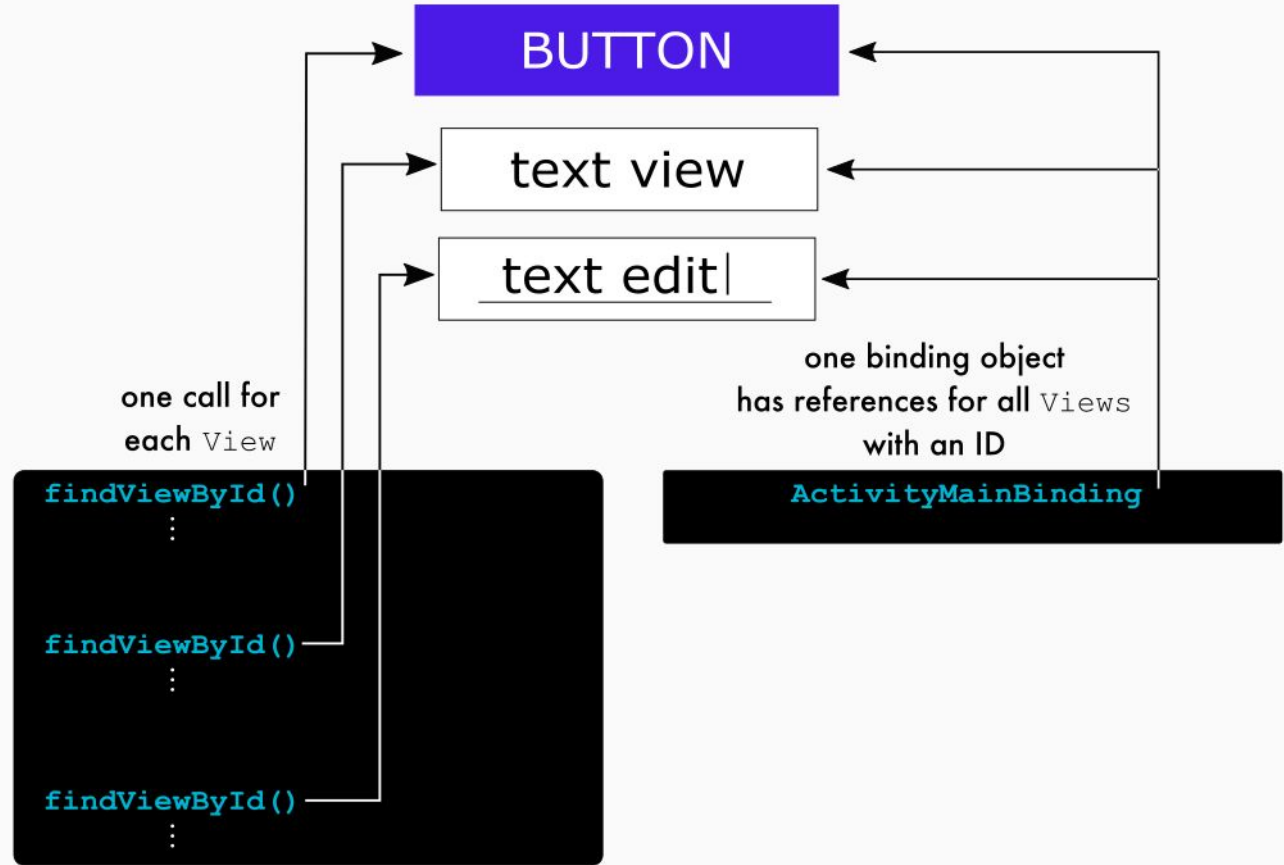
Vinculando as Views

O método `onCreate()` na classe `MainActivity` 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 criará e inicializa um objeto de vinculação apenas uma vez.

findViewById

View Binding



Vinculando as Views

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)  
  
    }  
  
}
```

3. 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.

Vinculando as Views

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 você não precisará criar uma variável para armazenar a referência de uma View, bastará usá-la diretamente com o objeto de vinculação.

```
// 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"
```

Adicionando Listeners

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.

O primeiro passo é adicionar um listener de clique para especificar o que o botão **Calculate** fará 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() }
```

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

```
fun calculateTip() {  
  
}
```

Calcular a gorjeta

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() {  
  
    }  
}
```

Calcular a gorjeta

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.

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. Lembre-se de 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.

Calcular a gorjeta

2. 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.

Calcular a gorjeta

3. 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 momento, o método `calculateTip()` será semelhante a:

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

Calcular a 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`.

2. 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  
}
```

Calcular a gorjeta

Neste momento, o método `calculateTip()` será semelhante a:

```
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

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. $\text{Gorjeta} = \text{custo do serviço} * \text{porcentagem da gorjeta}$. 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.

Calcular a gorjeta

2. 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`.

3. 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 momento, o método `calculateTip()` será semelhante a:

```
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)  
    }  
}
```

Calcular a gorjeta

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.

Calcular a gorjeta

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.

Calcular a gorjeta

2. 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)
```

3. Observe que `NumberFormat` é exibido em vermelho. Isso ocorre porque o Android Studio não consegue descobrir automaticamente qual versão de `NumberFormat` você quer usar.
4. Passe o cursor sobre `NumberFormat` e escolha **Import** no pop-up que será exibido.
5. Na lista de possíveis importações, escolha **NumberFormat (java.text)**. O Android Studio adicionará uma instrução `import` na parte superior do arquivo `MainActivity` e o `NumberFormat` não estará mais vermelho

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

Unresolved reference: NumberFormat

Import More actions...

Exibir o valor

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. 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.

3. 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 momento, o método `calculateTip()` será semelhante a:

```
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.

4. Abra activity_main.xml (**app > res > layout > activity_main.xml**)
5. Localize a TextView do tip_result.
6. Remova a linha com o atributo android:text.

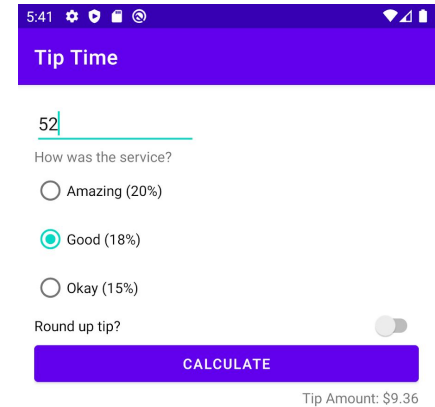
```
android:text="@string/tip_amount"
```

7. 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.

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



The screenshot shows the 'Tip Time' app interface. At the top, there's a status bar with the time 5:41 and various icons. Below it, the app title 'Tip Time' is displayed in a blue header. A text input field contains the number '52'. Below the input field, the question 'How was the service?' is followed by three radio button options: 'Amazing (20%)', 'Good (18%)', and 'Okay (15%)'. The 'Good (18%)' option is selected. Below these options, there is a toggle switch for 'Round up tip?'. At the bottom, there is a large blue button labeled 'CALCULATE'. Below the button, the text 'Tip Amount: \$9.36' is displayed.

5:41

Tip Time

52

How was the service?

☐ Amazing (20%)

☒ Good (18%)

☐ Okay (15%)

Round up tip? ☐

CALCULATE

Tip Amount: \$9.36

Modificando o layout

O Material Design

O Material Design é o Design System da Google, e é inspirado no mundo físico e nas texturas dele, incluindo a forma como os objetos refletem a luz e projetam sombras.

Ele apresenta **diretrizes** sobre como criar a IU do app de uma forma legível, atraente e consistente.

Com o Material Theming, ferramenta disponibilizada pelo google, você pode adaptar o Material Design ao app seguindo as orientações de personalização de cores, tipografia e formas. O Material Design vem com um tema de referência, que pode ser usado sem mudanças.

Você pode personalizá-lo o quanto quiser para que o Material Design se adapte ao seu app.

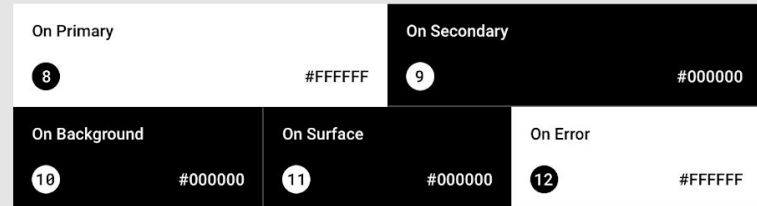
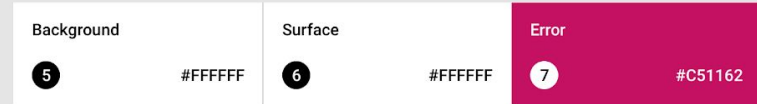
<https://material.io/design/material-theming/overview.html>

Um *estilo* pode especificar atributos para uma View, como cor e tamanho da fonte, cor do segundo plano e muito mais.

O *tema* é uma coleção de estilos aplicados a um app, atividade ou hierarquia de visualização inteiros, não apenas a uma View individual. Quando você aplica um tema a um app, atividade, visualização individual ou em grupo, o tema é aplicado a esse elemento e a todos os filhos deles. Os temas também podem aplicar estilos a elementos que não são de visualização, como a barra de status e o segundo plano da janela.

Cada parte da IU dos apps Android usa cores diferentes. Para ajudar você a usar as cores de forma significativa no app e aplicá-las consistentemente, o sistema de tema agrupa as cores em **12 atributos nomeados** relacionados à cor que será usada pelo texto, pelos ícones e muito mais. Seu tema não precisa especificar todos eles. Você escolherá as cores primárias e secundárias, além das cores do texto e dos ícones desenhados com essas cores.

Atributos do Material Design



Nº	Nome	Atributo do tema
1	Primary	colorPrimary
2	Primary Variant	colorPrimaryVariant
3	Secondary	colorSecondary
4	Secondary Variant	colorSecondaryVariant
5	Background	colorBackground
6	Surface	colorSurface
7	Error	colorError
8	On Primary	colorOnPrimary
9	On Secondary	colorOnSecondary
10	On Background	colorOnBackground
11	On Surface	colorOnSurface
12	On Error	colorOnError

1. No Android Studio, abra o arquivo themes.xml (**app > res > values > themes.xml**).
2. O nome do tema, Theme.TipTime, é baseado no nome do app.

```
<style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
```

Observe que a linha do XML também especifica um tema pai, Theme.MaterialComponents.DayNight.DarkActionBar. DayNight é um tema predefinido na biblioteca de componentes do Material Design.

DarkActionBar significa que a barra de ações usa uma cor escura. Assim como uma classe herda os atributos da classe mãe, um tema herda os atributos do tema pai.

Referências

<https://developer.android.com/codelabs/basic-android-kotlin-training-xml-layouts#0>

<https://developer.android.com/codelabs/basic-android-kotlin-training-tip-calculator?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-tip-calculator#0>

Analise os itens no arquivo e observe que os nomes são semelhantes aos do diagrama acima: colorPrimary, colorSecondary e assim por diante. themes.xml

```
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
        <!-- Primary brand color. -->
        <item name="colorPrimary">@color/purple_500</item>
        <item name="colorPrimaryDark">@color/purple_700</item>
        <item name="colorOnPrimary">@color/white</item>
        <!-- Secondary brand color. -->
        <item name="colorSecondary">@color/teal_200</item>
        <item name="colorSecondaryVariant">@color/teal_700</item>
        <item name="colorOnSecondary">@color/black</item>
        <!-- Status bar color. -->
        <item name="android:statusBarColor" tools:targetApi="1"?attr/colorPrimaryVariant</item>
        <!-- Customize your theme here. -->
    </style>
</resources>
```

Nem todos os atributos de tema de cor são definidos. As cores que não forem definidas herdarão a cor do tema pai.

O Android Studio desenha uma cor pequena de exemplo na margem esquerda.

4		<code><!-- Primary brand color. --></code>
5		<code><item name="colorPrimary">@color/purple_500</item></code>
6		<code><item name="colorPrimaryVariant">@color/purple_700</item></code>
7		<code><item name="colorOnPrimary">@color/white</item></code>
8		<code><!-- Secondary brand color. --></code>
9		<code><item name="colorSecondary">@color/teal_200</item></code>
10		<code><item name="colorSecondaryVariant">@color/teal_700</item></code>
11		<code><item name="colorOnSecondary">@color/black</item></code>

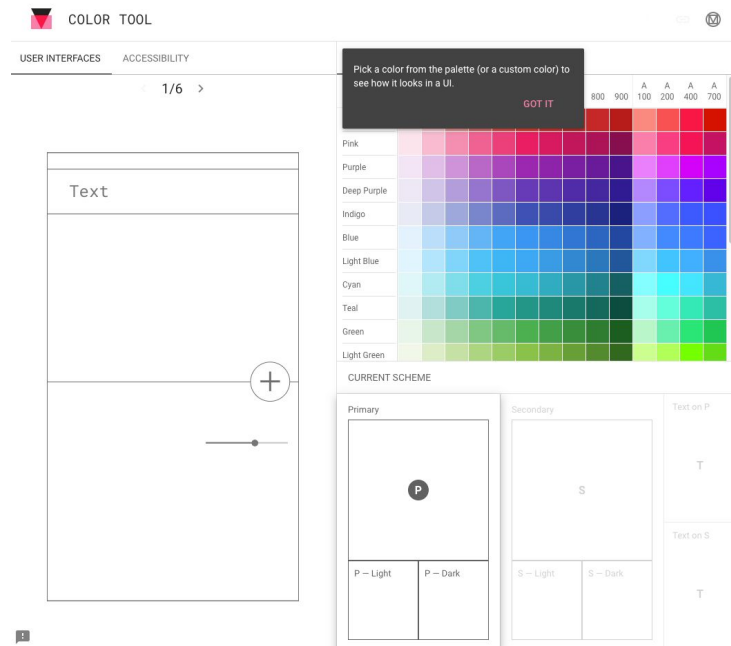
5. Por fim, as cores são especificadas como recursos de cor, por exemplo, `@color/purple_500`, em vez de usar um valor RGB diretamente.
6. Abra o arquivo `colors.xml` (**app > res > values > colors.xml**). Você verá os valores hexadecimais de cada recurso de cor. Lembre-se de que o `#FF` à esquerda é o valor alfa, o que significa que a cor é 100% opaca. `colors.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
</resources>
```

Modificar as cores do App

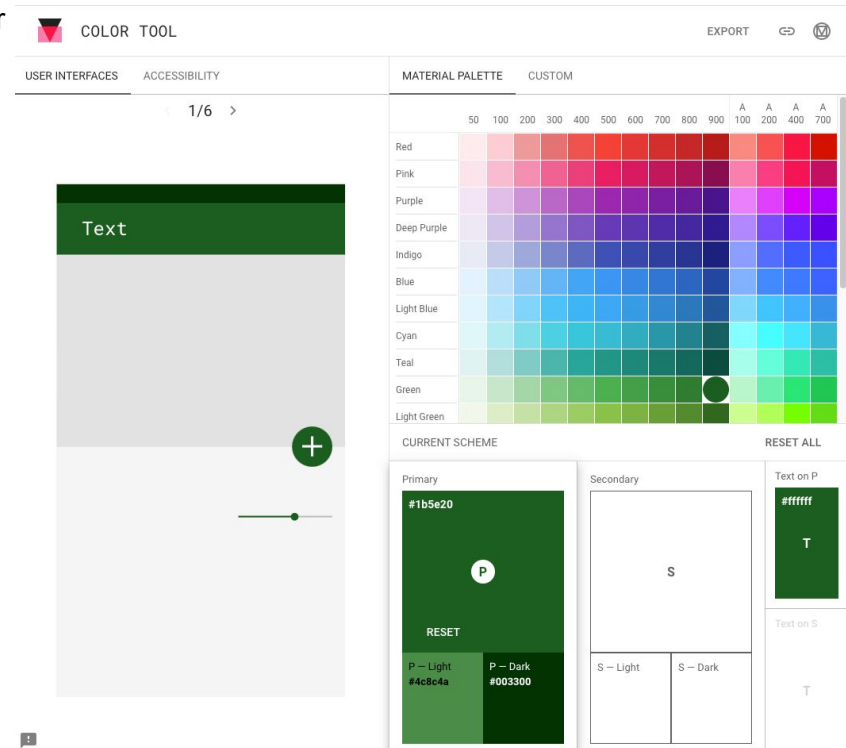
Agora que você já conhece um pouco sobre os atributos do tema, é hora de escolher algumas cores. A maneira mais fácil de fazer isso é com o [Color Tool](#), um app baseado na Web fornecido pela equipe do Material Design.

A ferramenta oferece uma paleta de cores predefinidas e permite ver facilmente como elas serão exibidas quando usadas em diferentes elementos da IU.



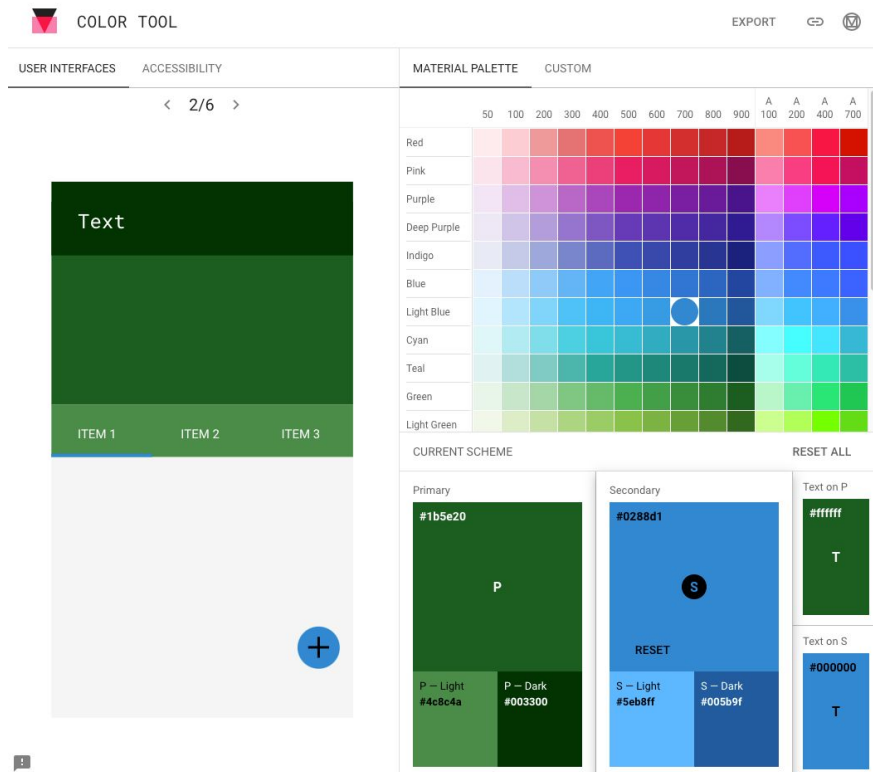
Modificar as cores do App

1. Comece selecionando uma cor principal na seção **Primary** (colorPrimary), por exemplo, **Green 900**. A ferramenta de cor mostrará a aparência em um modelo do app, além de selecionar as variantes **Light** e **Dark**.



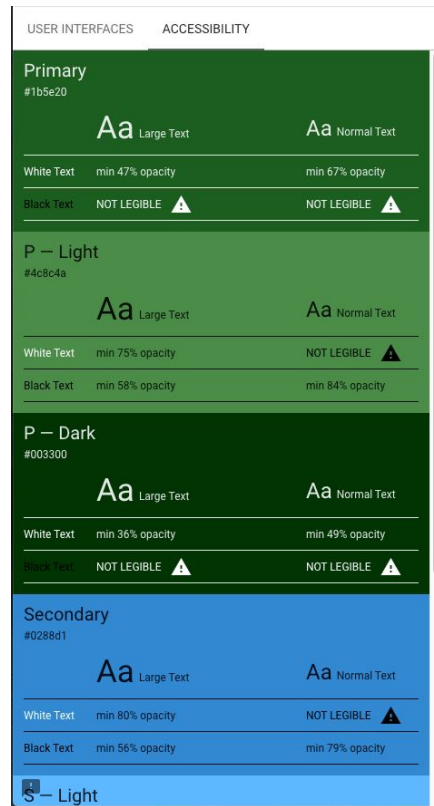
Modificar as cores do App

2. Toque na seção **Secondary** e escolha uma cor secundária (colorSecondary) de sua preferência, por exemplo, **Light Blue 700**. A cor mostra o que será exibido no modelo do app e seleciona novamente as variantes **Light** e **Dark**.
3. O modelo do app inclui seis simulações de "telas". Veja como suas opções de cor serão exibidas nas diferentes telas tocando nas setas acima do modelo.



Modificar as cores do App

4. A ferramenta de cores também fornece a guia **Accessibility** para informar se as cores têm contraste suficiente para serem lidas quando usadas com texto preto ou branco. Para tornar seu app mais acessível, é preciso garantir que o contraste de cores seja alto o suficiente: 4,5:1 ou maior para textos pequenos e 3,0:1 ou maior para textos grandes
5. Para `primaryColorVariant` e `secondaryColorVariant`, você pode escolher a variante clara ou escura sugerida.



Modificar as cores do App

A definição de recursos de cores facilita a reutilização consistente das mesmas cores em partes diferentes do app.

1. No Android Studio, abra o arquivo colors.xml (**app > res > values > colors.xml**).
2. Após as cores existentes, defina um recurso de cor chamado green usando o valor selecionado acima, #1B5E20.

```
<color name="green">#1B5E20</color>
```

3. Continue definindo recursos para as outras cores. A maioria dos valores é gerada pela ferramenta de cores. Observe que os valores para green_light e blue_light são diferentes daqueles mostrados pela ferramenta. Você vai usá-los em uma etapa futura.

green	#1B5E20
green_dark	#003300
green_light	#A5D6A7
blue	#0288D1
blue_dark	#005B9F
blue_light	#81D4FA

Modificar as cores do App

O arquivo colors.xml do app agora terá esta aparência:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>

    <color name="green">#1B5E20</color>
    <color name="green_dark">#003300</color>
    <color name="green_light">#A5D6A7</color>
    <color name="blue">#0288D1</color>
    <color name="blue_dark">#005B9F</color>
    <color name="blue_light">#81D4FA</color>
</resources>
```

Modificar as cores do App

Agora que você definiu nomes para as cores selecionadas, é hora de usá-las no seu tema.

1. Abra o arquivo themes.xml (**app > res > values > themes > themes.xml**).
2. Mude a colorPrimary para a cor principal que você selecionou, @color/green.
3. Mude colorPrimaryVariant para @color/green_dark.
4. Mude colorSecondary para @color/blue.
5. Mude colorSecondaryVariant para @color/blue_dark.
6. Verifique se as opções **Text on P** e **Text on S** ainda estão definidas como branco (#FFFFFF) e preto (#000000). Se você estiver usando a ferramenta de cores por conta própria e selecionar outras cores, talvez seja necessário definir mais recursos de cor.

Modificar as cores do App

```
<resources xmlns:tools="http://schemas.android.com/tools">
  <!-- Base application theme. -->
  <style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
    <!-- Primary brand color. -->
    <item name="colorPrimary">@color/green</item>
    <item name="colorPrimaryVariant">@color/green_dark</item>
    <item name="colorOnPrimary">@color/white</item>
    <!-- Secondary brand color. -->
    <item name="colorSecondary">@color/blue</item>
    <item name="colorSecondaryVariant">@color/blue_dark</item>
    <item name="colorOnSecondary">@color/black</item>
    <!-- Status bar color. -->
    <item name="android:statusBarColor" tools:targetApi="1">?attr/colorPrimaryVariant</item>
    <!-- Customize your theme here. -->
  </style>
</resources>
```

Adicionando um tema escuro

O modelo de app incluiu um tema claro padrão e também uma variante de tema escuro. Um tema escuro usa cores mais escuras e discretas e:

- Pode reduzir significativamente o consumo de energia (dependendo da tecnologia da tela do dispositivo).
- Melhora a visibilidade para usuários com problemas de visão e que tenham sensibilidade ao brilho da luz.
- Facilita o uso do dispositivo em um ambiente com pouca luz.

Adicionando um tema escuro

Como escolher cores para o tema escuro

As cores de um tema escuro precisam ser legíveis. Os temas escuros usam uma cor de superfície escura com tons de cores limitados. Para garantir a legibilidade, as cores primárias geralmente são versões menos saturadas das cores primárias do tema claro.

Para oferecer mais flexibilidade e usabilidade em um tema escuro, recomendamos usar tons mais claros (200 a 50) em um tema escuro, em vez do tema de cores padrão (tons de saturação de 900 a 500). Anteriormente, você escolheu verde 200 e azul claro 200 como as cores claras. Para o app, você usará as cores claras como as principais e as cores primárias como as variantes.

Adicionando um tema escuro

1. Abra o arquivo themes.xml (night) (**app > res > values > themes > themes.xml (night)**).

Observação: este arquivo themes.xml é diferente do arquivo themes.xml anterior. Ele contém a versão escura do tema. Os recursos neste arquivo serão usados quando o **tema escuro** estiver ativado no dispositivo.

2. Mude a colorPrimary para a variante clara da cor primária que você selecionou, @color/green_light.
3. Mude colorPrimaryVariant para @color/green.
4. Mude colorSecondary para @color/blue_light.
5. Mude colorSecondaryVariant para @color/blue_light.

Quando você terminar, seu arquivo themes.xml (night) terá esta aparência:

```
<resources xmlns:tools="http://schemas.android.com/tools">
  <!-- Application theme for dark theme. -->
  <style name="Theme.TipTime" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
    <!-- Primary brand color. -->
    <item name="colorPrimary">@color/green_light</item>
    <item name="colorPrimaryVariant">@color/green</item>
    <item name="colorOnPrimary">@color/black</item>
    <!-- Secondary brand color. -->
    <item name="colorSecondary">@color/blue_light</item>
    <item name="colorSecondaryVariant">@color/blue_light</item>
    <item name="colorOnSecondary">@color/black</item>
    <!-- Status bar color. -->
    <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>
    <!-- Customize your theme here. -->
  </style>
</resources>
```

Agora, as cores originais definidas no arquivo colors.xml, por exemplo, purple_200, não são mais usadas; então você pode excluí-las.

PMobile - Atividade Bonus

<https://developer.android.com/codelabs/basic-android-kotlin-training-change-app-theme?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-2-pathway-2%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-change-app-theme#0>