1.Antes de Começar

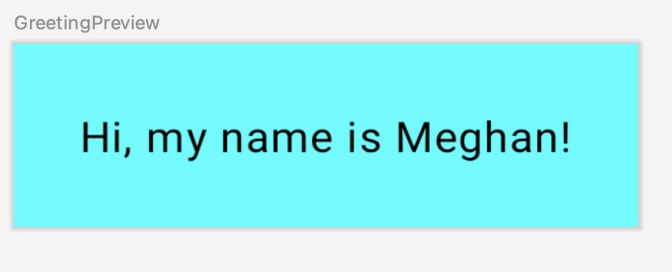
Instale o Android Studio no seu computador, caso ainda não tenha feito isso. Verifique se o computador atende aos requisitos do sistema necessários para executar o Android Studio (na parte inferior da página de download).
Neste tutorial, você vai criar seu primeiro app Android usando um modelo de projeto fornecido pelo Android Studio. Use o Kotlin e o Jetpack Compose para personalizar o app. O Android Studio é atualizado e, às vezes, a UI muda. Por isso, é normal o Android Studio estar um pouco diferente das capturas de tela deste tutorial.

Pré-requisitos
•Conhecimentos básicos sobre Kotlin

•O que é necessário
•A versão mais recente do Android Studio.

•O que você vai aprender
•Como criar um app Android usando o Android Studio.
•Como executar apps com a ferramenta de visualização no Android Studio.
•Como atualizar textos com o Kotlin.
•Como atualizar uma interface do usuário (IU) usando o Jetpack Compose.
•Como visualizar o app no Jetpack Compose.

O que você vai criar
•Um app que permite personalizar sua apresentação.
O app vai ficar assim quando você concluir este tutorial, mas ele vai mostrar seu nome personalizado:

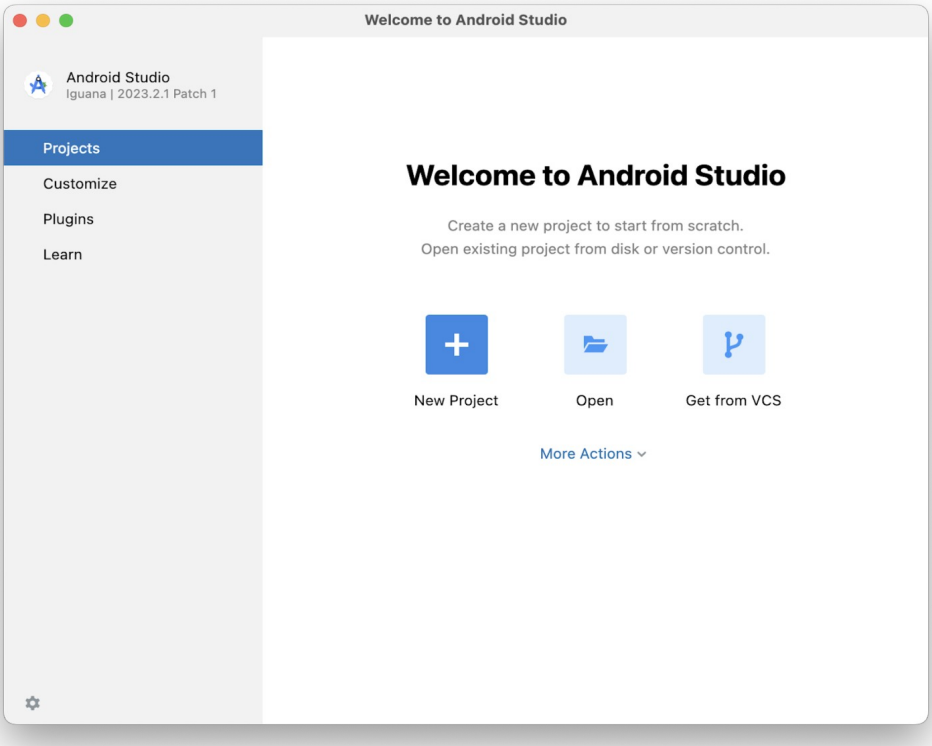


O que é necessário
•Um computador com o Android Studio instalado.

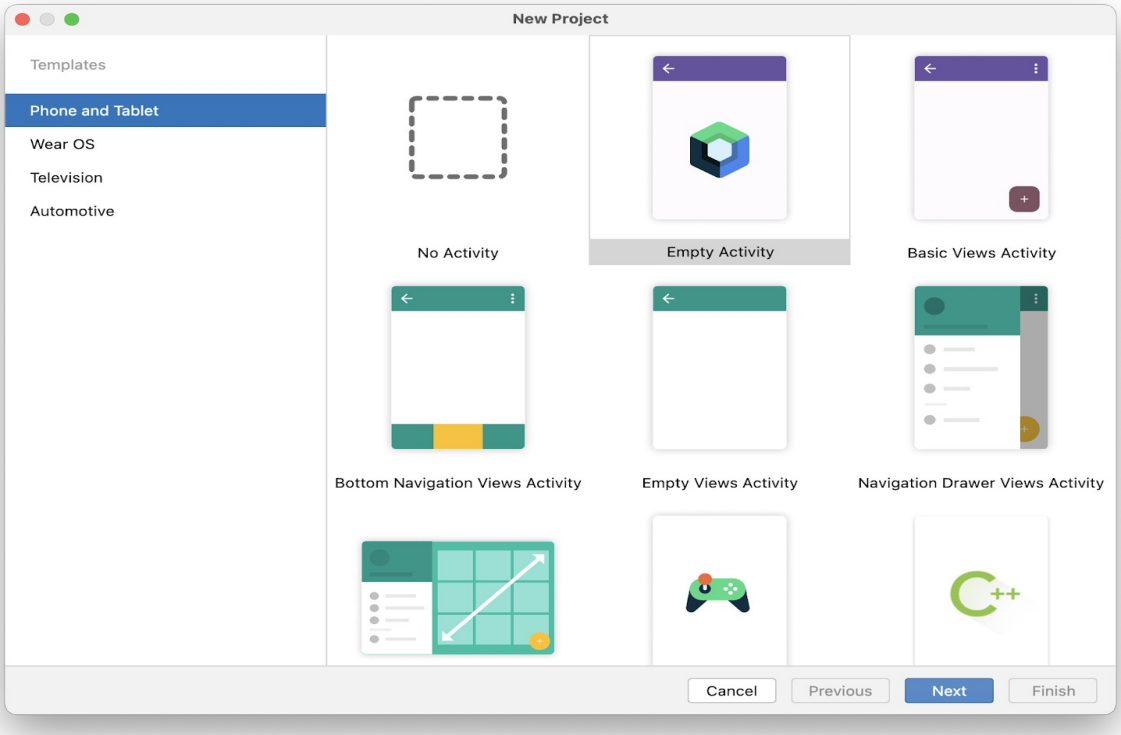
2. Criar um projeto usando o modelo

Neste tutorial, você vai criar um app Android com o modelo de projeto **Empty Activity** fornecido pelo Android Studio.
Para criar um projeto no Android Studio:
1.Clique duas vezes no ícone do Android Studio para iniciar.

2.Na caixa de diálogo **Welcome to Android Studio**, clique em **New Project**.

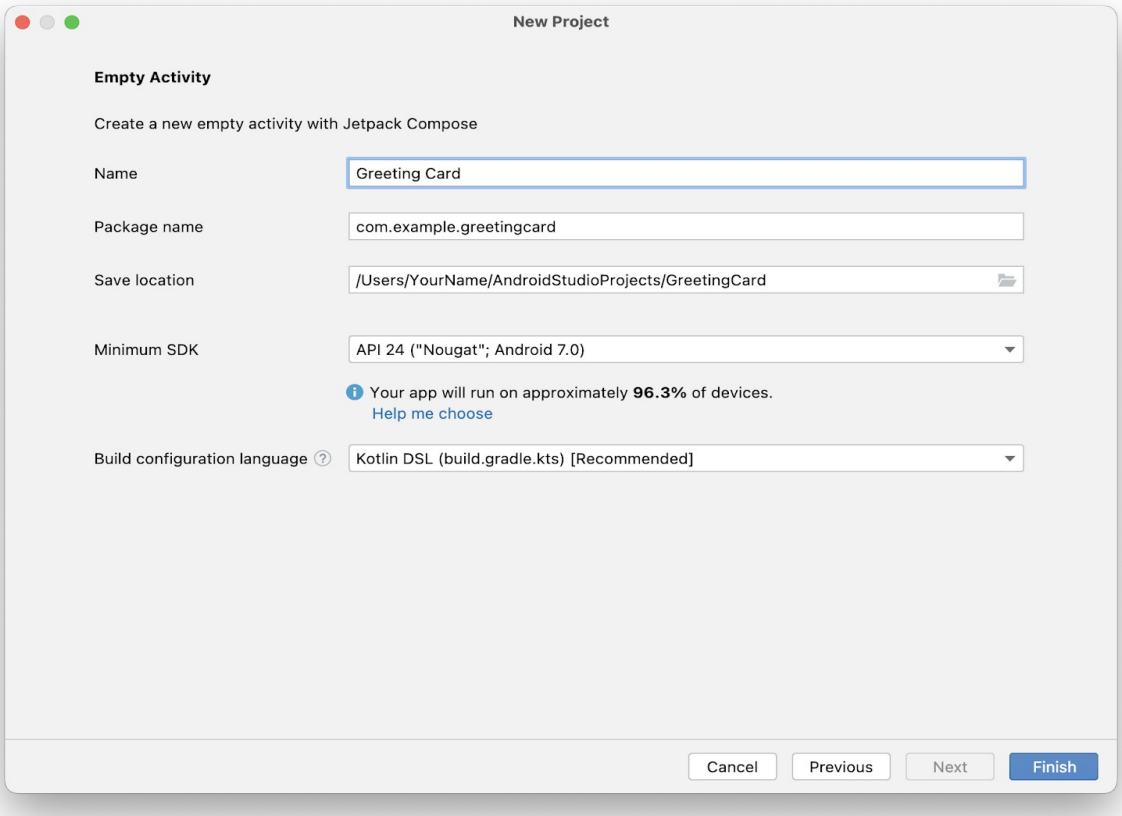


A janela **New Project** será aberta com uma lista de modelos do Android Studio.



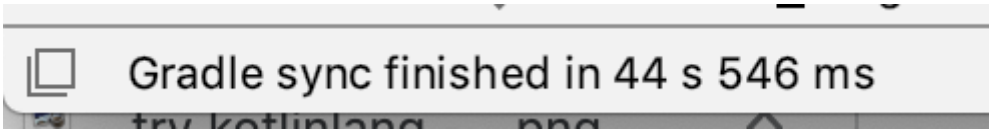
No Android Studio, um modelo de projeto é um projeto do Android que fornece a base para um determinado tipo de app. Os modelos criam a estrutura do projeto e os arquivos necessários para que ele seja criado pelo Android Studio. O modelo escolhido fornece um código inicial para acelerar o processo.
3.Confira se a guia **Phone and Tablet** está selecionada.

4.Clique no modelo **Empty Activity** para selecioná-lo como modelo do projeto. O **Empty Activity** é o modelo para criar um projeto simples, que você pode usar para criar um app do Compose. Ele tem uma única tela e mostra o texto **"Hello Android!"**.
5.Clique em **Próxima**. A caixa de diálogo **New Project** vai ser aberta. Há alguns campos para configurar o projeto.
6.Configure o projeto desta maneira:
O campo **Name** é usado para inserir o nome do projeto. Para este tutorial, digite "Greeting Card" (Cartão de apresentação).
Deixe o campo **Package name** como está. É assim que seus arquivos vão ser organizados na estrutura do arquivo. Nesse caso, o nome do pacote será **com.example.greetingcard**.
Deixe o campo **Save location** como está. Ele contém o local onde todos os arquivos relacionados ao projeto são salvos. Anote onde o conteúdo está salvo no computador para poder encontrar os arquivos.
Selecione **API 24: Android 7.0 (Nougat)** no menu no campo **Minimum SDK**. O campo Minimum SDK indica a versão mínima do Android em que o app pode ser executado.

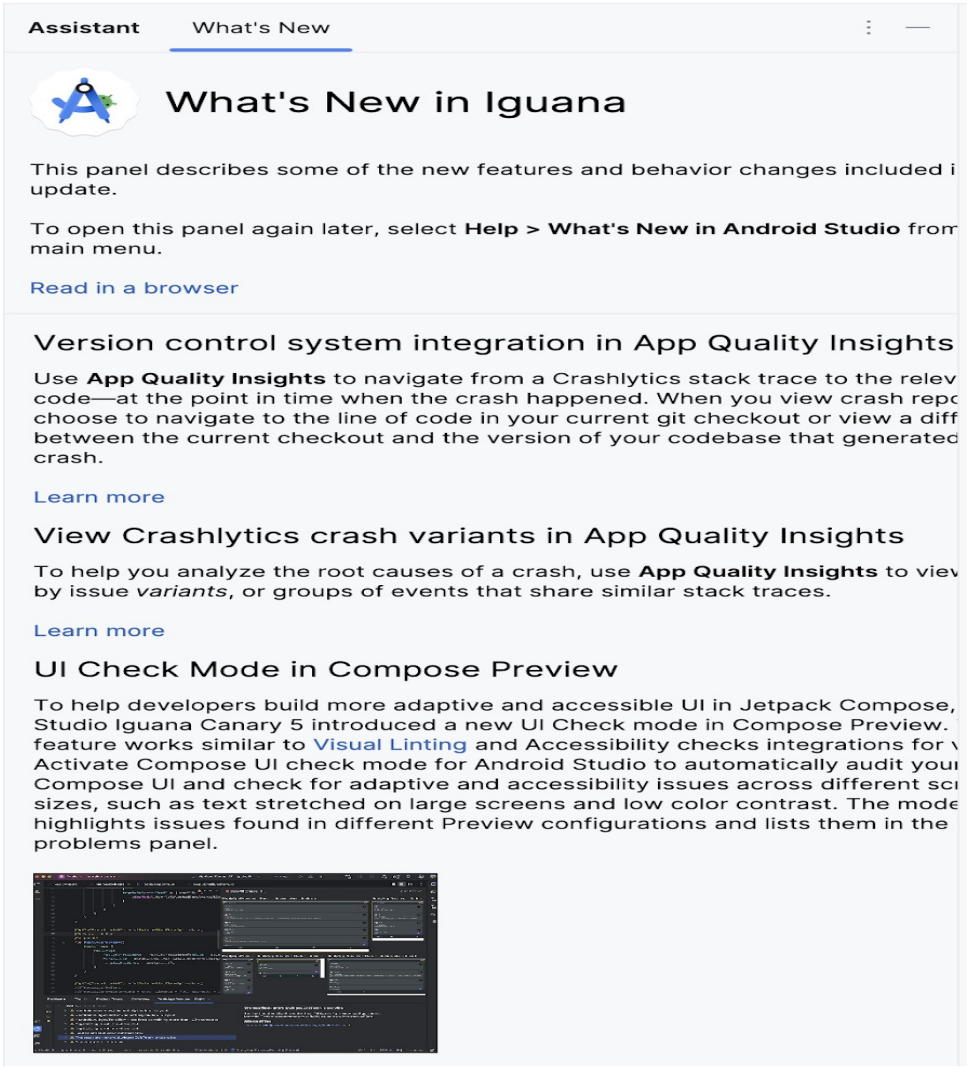


7.Clique em **Finish**. Talvez isso demore um pouco. Pode ser um bom momento para tomar um café. Enquanto isso, uma barra de progresso e uma mensagem indicam se o Android Studio ainda está configurando o projeto. Ela pode ter esta aparência:

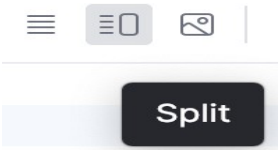
Uma mensagem semelhante a essa informa quando a configuração do projeto é criada.



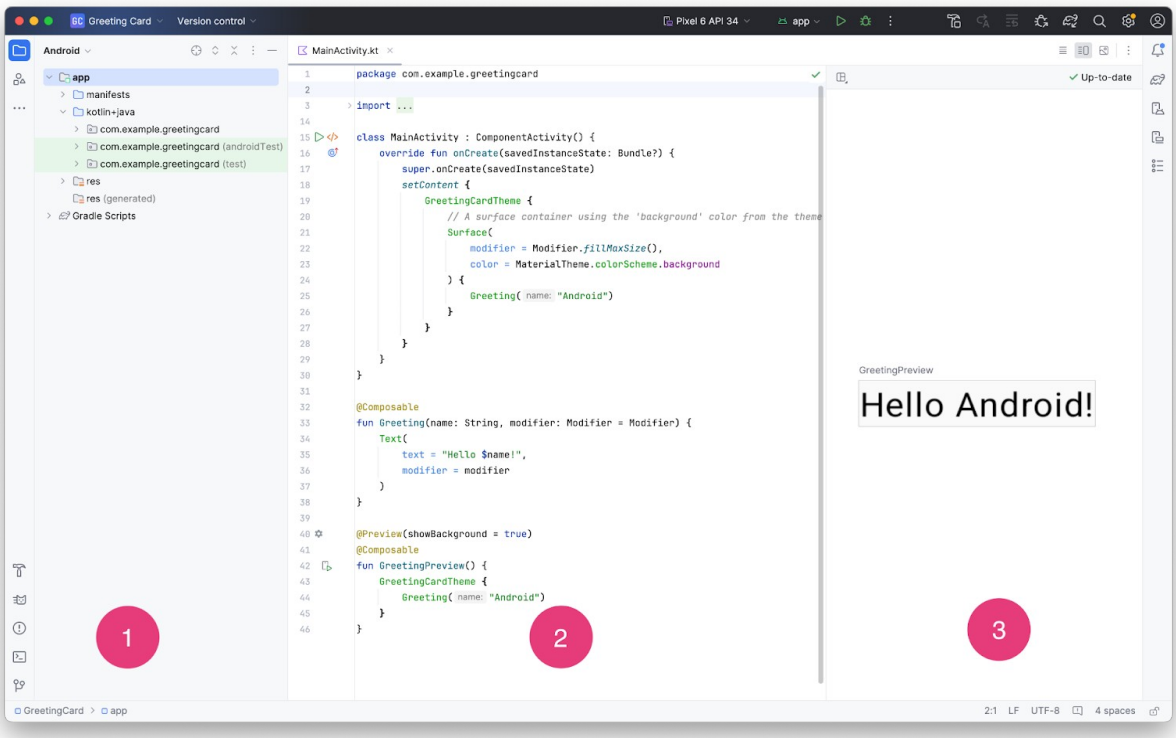
8.Talvez você veja um painel **What's New** com atualizações sobre novos recursos do Android Studio. Feche-o por enquanto.



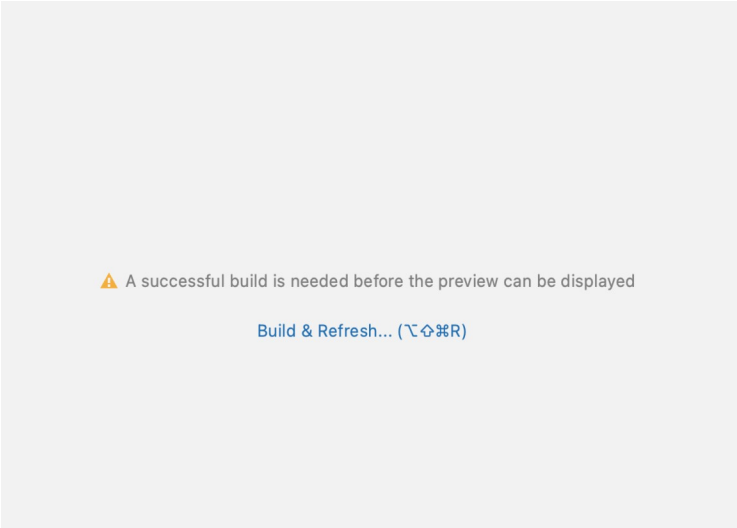
9.Clique em **Split** no canto direito de cima do Android Studio para ver o código e o design. Você também pode clicar em **Code** para ver somente o código ou em **Design** para ver apenas o design.



Depois de pressionar **Split**, você verá três áreas:



•A visualização **Project** (1) mostra os arquivos e as pastas do projeto.
•A visualização **Code** (2) é onde você edita o código.
•A visualização **Design** (3) é onde você tem uma prévia da aparência do app.
Na visualização **Design**, você pode ver um painel em branco com o texto abaixo:



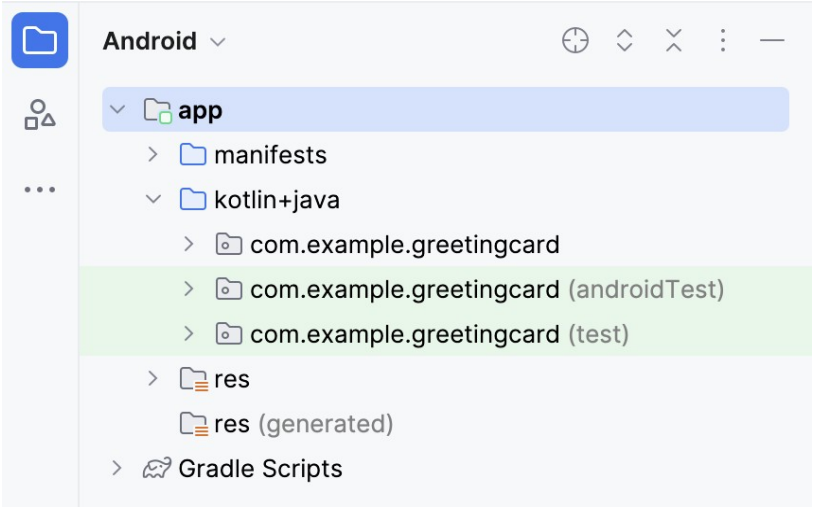
10.Clique em **Build & Refresh**. A criação pode demorar um pouco, mas quando terminar, a visualização vai mostrar uma caixa de texto com a mensagem "**Hello Android!**". A atividade vazia do Compose contém todo o código necessário para criar esse app.

3. Encontrar arquivos de projetos

Nesta seção, você continuará aprendendo sobre o Android Studio conhecendo a estrutura de arquivos.
1.No Android Studio, veja a guia **Project**. A guia **Project** mostra os arquivos e as pastas do projeto. Durante a configuração do projeto, o nome do pacote era **com.example.greetingcard**. Veja esse pacote na

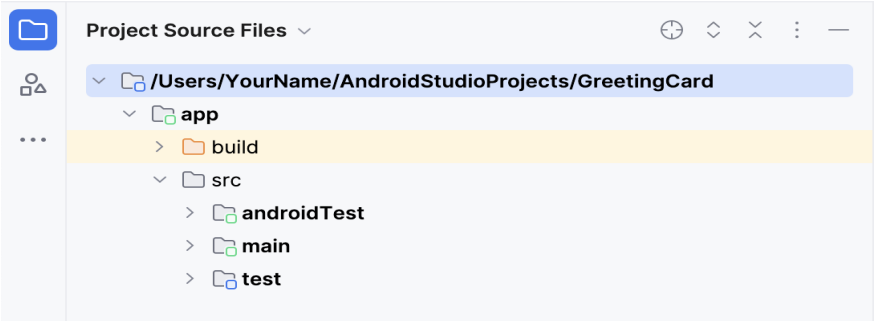
guia **Project**. Um pacote é basicamente uma pasta em que o código está localizado. O Android Studio organiza o projeto em uma estrutura de diretórios composta por um conjunto de pacotes.

2.Se necessário, selecione **Android** no menu suspenso da guia **Project**.



Essa é a visualização e a organização padrão dos arquivos que você usa. Ela é útil ao escrever códigos para o projeto, porque você consegue acessar facilmente os arquivos com que vai trabalhar no app. No entanto, em um navegador como o Finder ou o Windows Explorer, a hierarquia é organizada de forma muito diferente.

3.Selecione **Project Source Files** no menu suspenso. Agora você pode procurar os arquivos da mesma maneira que faria em qualquer navegador de arquivos.



4.Selecione **Android** novamente para retornar à visualização anterior. Você usa a visualização do **Android** neste curso. Caso a estrutura de arquivos pareça estranha, confira se você ainda está na visualização do **Android**.

4. Atualizar o texto

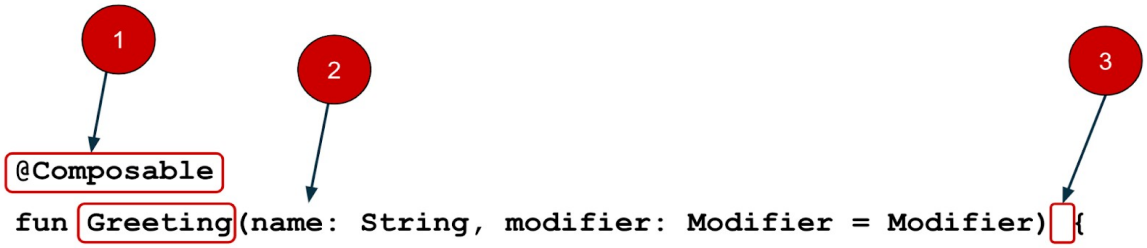
Agora que você conhece o Android Studio, é hora de começar a criar seu cartão de apresentação.
Veja a visualização **Code** do arquivo **MainActivity.kt**. Há algumas funções geradas automaticamente nesse código, especificamente **onCreate()** e **setContent()**.
Observação: função é a seção de um programa que realiza uma tarefa específica.

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            GreetingCardTheme {
                // A surface container using the 'background' color from the theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
```

```
            ) {
                Greeting("Android")
            }
        }
    }
  }
}
```

A função **onCreate()** é o ponto de entrada do app Android e chama outras funções para criar a interface do usuário. Nos programas Kotlin, a função **main()** é o ponto de entrada/de partida da execução. Em apps Android, a função **onCreate()** cumpre esse papel.
A função **setContent()** na função **onCreate()** é usada para definir o layout usando funções de composição. Todas as funções marcadas com a anotação **@Composable** podem ser chamadas na função **setContent()** ou em outras funções de composição. A anotação diz ao compilador Kotlin que essa função é usada pelo Jetpack Compose para gerar a IU.
Observação: o compilador usa o código Kotlin que você escreveu, analisa cada linha e traduz em algo que o computador entenda. Esse processo é chamado de compilação do código.
Agora, observe a função **Greeting()**. A função **Greeting()** é combinável e, por esse motivo, a anotação **@Composable** aparece acima dela. Essa função recebe uma entrada e gera o que será mostrado na tela.

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Text(
        text = "Hello $name!",
        modifier = modifier
    )
}
```



1.A anotação **@Composable** é adicionada antes da função.
2.Nomes de funções **@Composable** usam letras maiúsculas.
3.As funções **@Composable** não podem retornar nada.

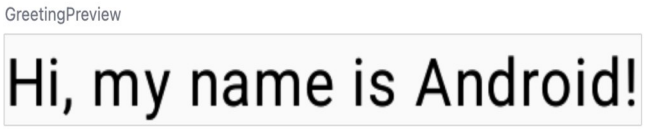
```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Text(
        text = "Hello $name!",
        modifier = modifier
    )
}
```

No momento, a função **Greeting()** recebe um nome e mostra **Hello** para essa pessoa.
1.Atualize a função **Greeting()** para apresentar você em vez de dizer "Hello":

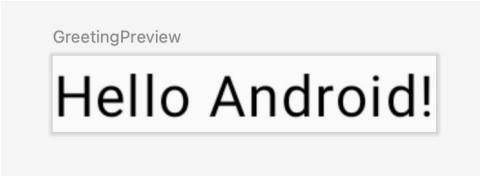
```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Text(
        text = "Hi, my name is $name!",
        modifier = modifier
```

```
    )
}
```

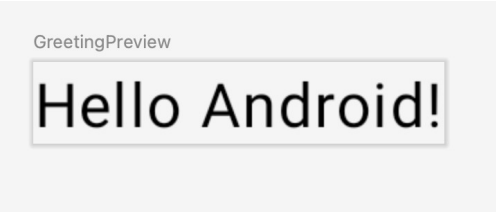
2.O Android deve atualizar automaticamente a visualização.



Bom trabalho! Você mudou o texto, mas ele ainda apresenta você como "Android", que provavelmente não é seu nome. Depois, personalize o app para se apresentar com seu nome.
A função **GreetingPreview()** é um recurso interessante que permite ver a aparência do seu elemento combinável sem precisar criar todo o app. Para ativar uma visualização de elemento combinável, anote-o com **@Composable** e **@Preview**. A anotação **@Preview** informa ao Android Studio que esse elemento combinável deve ser mostrado na visualização de design desse arquivo.
Como você pode ver, a anotação **@Preview** recebe um parâmetro conhecido como **showBackground**. Se **showBackground** for definido como **true**, um plano de fundo vai ser adicionado à visualização do elemento combinável.
Como o Android Studio usa um tema claro por padrão no editor, pode ser difícil ver a diferença entre **showBackground = true** e **showBackground = false**. No entanto, este é um exemplo de como é a diferença. Observe o fundo branco na imagem com o parâmetro definido como **true**.



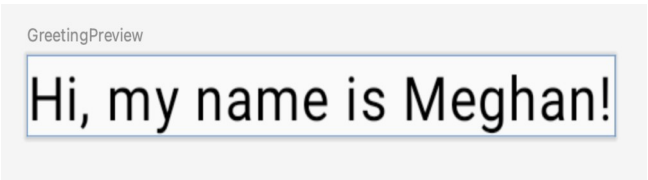
showBackground = true



showBackground = false

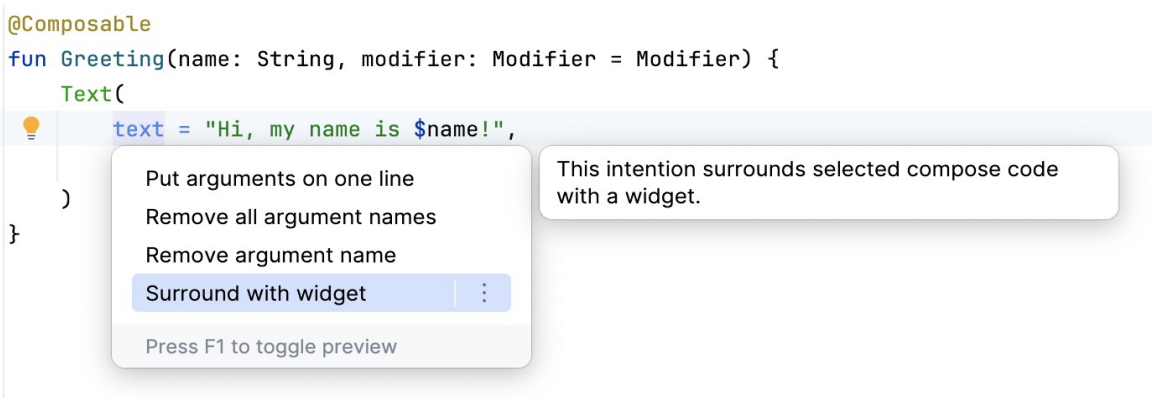
3.Atualize a função **GreetingPreview()** com seu nome. Depois, recrie e confira seu cartão de apresentação personalizado.

```
@Preview(showBackground = true)
@Composable
fun GreetingPreview() {
    GreetingCardTheme {
        Greeting("Meghan")
    }
}
```

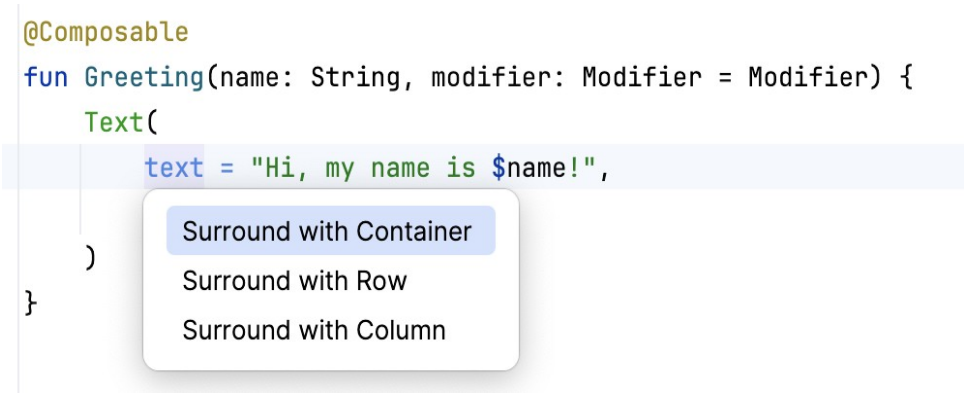


5. Mudar a cor do plano de fundo

Agora você tem um texto de introdução, mas ele está um pouco sem graça. Nesta seção, você vai aprender a mudar a cor do plano de fundo.
Para definir uma cor de plano de fundo diferente no cartão de apresentação, você vai precisar cercar o texto com uma **Surface**. Uma **Surface** é um contêiner que representa uma seção da IU em que você pode mudar a aparência, como a cor ou a borda do plano de fundo.
1.Para cercar o texto com uma **Surface**, destaque a linha de texto, pressione (**Alt+Enter** para Windows ou **Option+Enter** no Mac) e selecione **Surround with widget**.



2.Escolha a opção **Surround with Container**.



O contêiner padrão fornecido é **Box**, mas ele pode ser mudado para outro tipo de contêiner. Você aprenderá sobre o layout **Box** mais tarde no curso.

3.Exclua **Box** e digite **Surface()**.

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Surface() {
        Text(
            text = "Hi, my name is $name!",
            modifier = modifier
        )
    }
```

```
}
```

4.Para o contêiner **Surface**, adicione um parâmetro **color** e defina como **Color**.

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Surface(color = Color) {
        Text(
            text = "Hi, my name is $name!",
            modifier = modifier
        )
    }
}
```

5.Ao digitar **Color**, ele é exibido em vermelho, o que significa que o Android Studio não consegue resolver isso. Para resolver esse problema, role até a parte de cima do arquivo, onde está escrito "import", e pressione os três pontos.

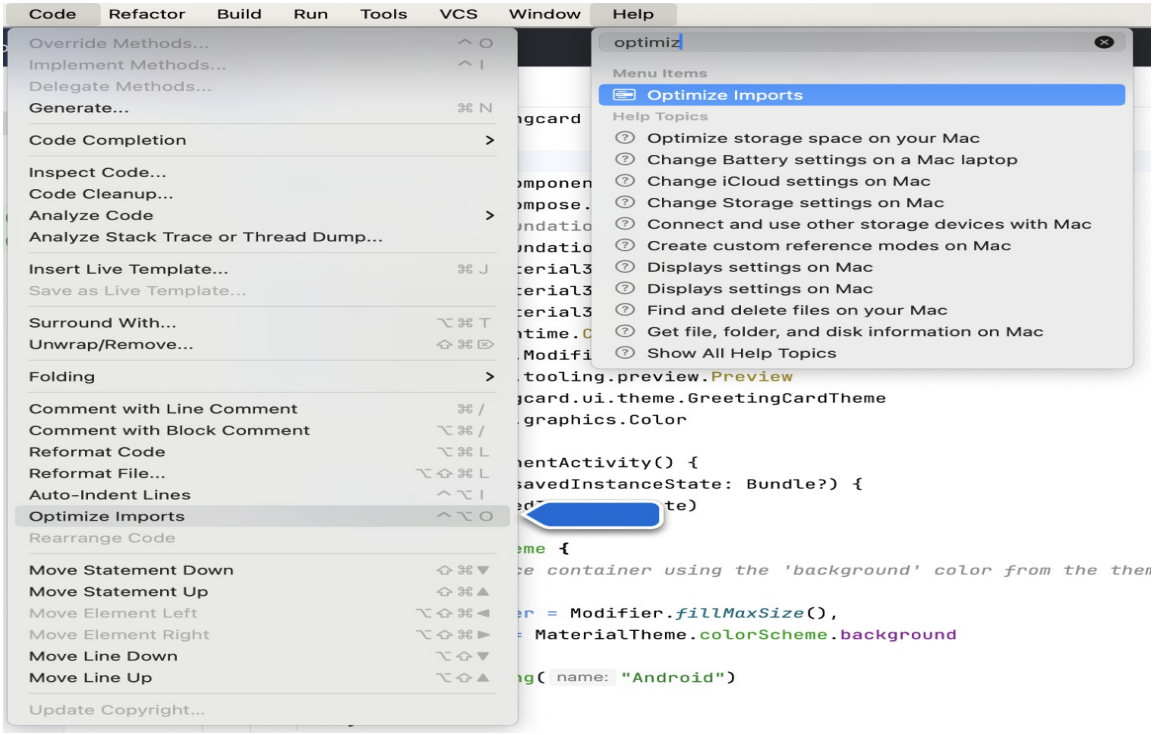
6.Adicione essa instrução ao fim da lista de importações.

```
import androidx.compose.ui.graphics.Color
```

A lista completa de importações será semelhante a esta.

```
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.tooling.preview.Preview
import com.example.greetingcard.ui.theme.GreetingCardTheme
import androidx.compose.ui.graphics.Color
```

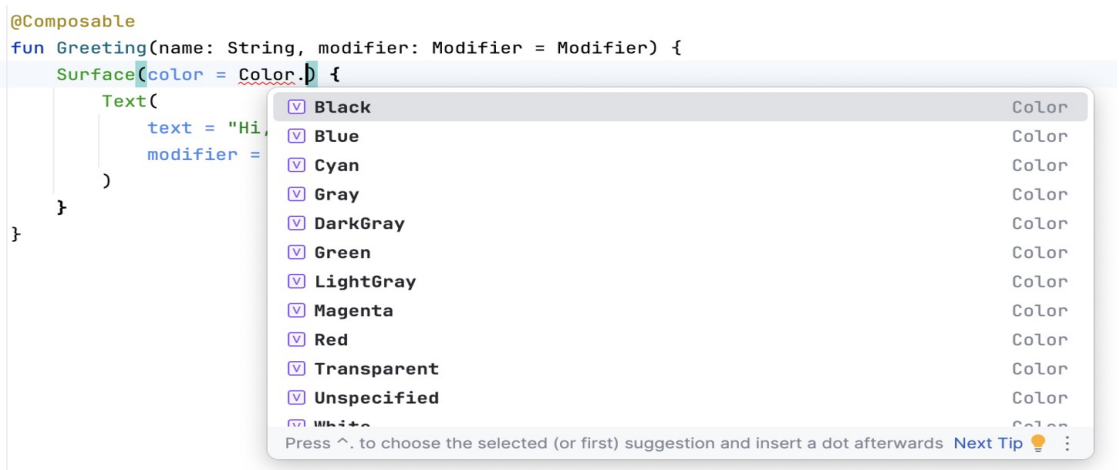
7.No código, a prática recomendada é manter suas importações listadas em ordem alfabética e remover as importações não utilizadas. Para isso, pressione **Help** na barra de ferramentas na parte de cima da tela, digite **optimize imports** e, em seguida, clique em **Optimize Imports**.



Abra **Optimize Imports** diretamente no menu: **Code** > **Optimize Imports.** Com a opção de pesquisa da Ajuda, você vai localizar um item de menu se não se lembrar de onde ele está.
A lista completa de importações vai ficar assim:

```
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.tooling.preview.Preview
import com.example.greetingcard.ui.theme.GreetingCardTheme
```

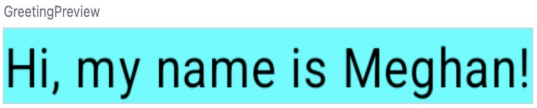
8.Observe que "Color", inserido entre parênteses depois de "Surface", não está mais em vermelho. Há apenas um sublinhado vermelho abaixo da palavra agora. Para corrigir esse problema, adicione um ponto depois do parâmetro. Vai aparecer um pop-up mostrando diferentes opções de cor.
Esse é um dos recursos mais interessantes do Android Studio. Ele é inteligente e vai ajudar você quando puder. Neste caso, ele sabe que você quer especificar uma cor, então vai sugerir cores diferentes.



9.Escolha uma cor para a superfície. Este tutorial usa **ciano**, mas você pode escolher sua cor favorita.

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
  Surface(color = Color.Cyan) {
    Text(
      text = "Hi, my name is $name!",
      modifier = modifier
    )
  }
}
```

10.Observe a visualização atualizada.



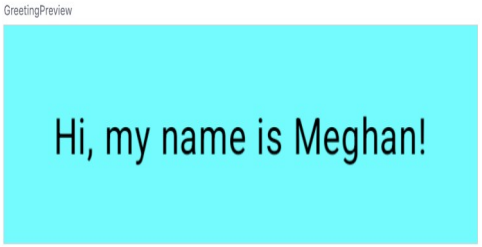
6. Adicionar padding

Agora, o texto tem uma cor de segundo plano. Depois, você vai adicionar um espaço (padding) ao redor dele.
Um `Modifier` é usado para aumentar ou decorar um elemento combinável. Um modificador que pode ser usado é o **padding**, que adiciona espaço ao redor do elemento. Neste caso, ele adiciona espaço ao redor do texto. Isso é feito com a função `Modifier.padding()`.
Cada elemento combinável precisa ter um parâmetro opcional do tipo **Modifier**, que deve ser o primeiro.
1.Adicione um padding ao **modifier** com um tamanho de **24.dp**.

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
   Surface(color = Color.Cyan) {
     Text(
       text = "Hi, my name is $name!",
       modifier = modifier.padding(24.dp)
     )
   }
}
```

2.Adicione essas importações à seção de instruções de importação.
Use **Optimize Imports** para listar as novas importações em ordem alfabética.

```
import androidx.compose.ui.unit.dp
import androidx.compose.foundation.layout.padding
```



Parabéns! Você criou seu primeiro app Android no Compose. Essa é uma grande conquista. Tente usar diferentes cores e textos do jeito que quiser.