

[feedback](https://developer.android.com/preview?hl=pt-br) (<https://developer.android.com/preview?hl=pt-br>).

Iniciar outra atividade

Quando concluir a [lição anterior](#)

(<https://developer.android.com/training/basics/firstapp/building-ui?hl=pt-br>), você terá um app que exibe uma atividade que consiste em uma só tela, com um campo de texto e um botão **Send**. Nesta lição, você adicionará um código à **MainActivity** para iniciar uma nova atividade para exibir uma mensagem quando o usuário tocar no botão **Send**.

Observação: esta lição pressupõe que você usa o Android Studio v3.0 ou versão posterior.

Responder ao botão "Send"

Siga as etapas a seguir para adicionar um método à classe **MainActivity** chamada quando o botão **Send** é tocado:

1. No arquivo **app > java > com.example.myfirstapp > MainActivity**, adicione o stub de método `sendMessage()` a seguir:

KOTLIN (#KOTLIN)**JAVA**

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    /** Called when the user taps the Send button */
    public void sendMessage(View view) {
        // Do something in response to button
    }
}
```

Talvez seja exibido um erro, porque o Android Studio não pode resolver a classe **View** usada como argumento do método. Para limpar o erro, clique na declaração **View**, coloque o cursor sobre ela e pressione **Alt+Enter** ou **Option+Enter** no Mac, para executar uma correção rápida. Se um menu for exibido, selecione **Import class**.

2. Retorne ao arquivo **activity_main.xml** para chamar o método pelo botão:

- a. Selecione o botão no Layout Editor.
- b. Na janela **Attributes**, localize a propriedade **onClick** e selecione **sendMessage [MainActivity]** na lista suspensa.

Agora, ao tocar no botão, o sistema chamará o método **sendMessage()**.

Observe os detalhes desse método. Eles são necessários para que o sistema reconheça o método como compatível com o atributo **android:onClick** (https://developer.android.com/reference/android/view/View?hl=pt-br#attr_android:onClick). Especificamente, o método tem as seguintes características:

- Acesso público.
- Um vazio ou, na linguagem Kotlin, um valor de retorno **unit** (<https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-unit/index.html>) (link em inglês) implícito.
- Uma **View** (<https://developer.android.com/reference/android/view/View?hl=pt-br>) como o único parâmetro. Esse é o objeto **View** (<https://developer.android.com/reference/android/view/View?hl=pt-br>) em que você clicou no final da Etapa 1.

3. A seguir, preencha esse método para ler o conteúdo do campo de texto e enviar esse texto a outra atividade.

Criar um intent

Um **Intent** (<https://developer.android.com/reference/android/content/Intent?hl=pt-br>) é um objeto que fornece vínculos de tempo de execução entre componentes separados, como duas atividades. O **Intent** (<https://developer.android.com/reference/android/content/Intent?hl=pt-br>) representa uma “intenção de fazer algo” do aplicativo. Você pode usar intents para uma ampla variedade de tarefas, mas, nesta lição, o intent iniciará outra atividade.

Em **MainActivity**, adicione a constante **EXTRA_MESSAGE** e o código **sendMessage()**, conforme mostrado:

KOTLIN (#KOTLIN) JAVA

```
public class MainActivity extends AppCompatActivity {  
    public static final String EXTRA_MESSAGE = "com.example.myfirstapp.MESSAGE"  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
  
    /** Called when the user taps the Send button */  
    public void sendMessage(View view) {  
        Intent intent = new Intent(this, DisplayMessageActivity.class);  
        EditText editText = (EditText) findViewById(R.id.editText);  
        String message = editText.getText().toString();  
        intent.putExtra(EXTRA_MESSAGE, message);  
        startActivity(intent);  
    }  
}
```

É provável que o Android Studio encontre erros **Cannot resolve symbol** novamente. Para limpar os erros, pressione Alt+Enter ou Option+Return no Mac. Suas importações precisam terminar como as seguintes:

KOTLIN (#KOTLIN)JAVA

```
import androidx.appcompat.app.AppCompatActivity;  
import android.content.Intent;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.EditText;
```

Um erro ainda permanece para `DisplayMessageActivity`, mas isso não é um problema. Você poderá corrigi-lo na próxima seção.

Veja o que está acontecendo em `sendMessage()`:

- O criador do `Intent` (<https://developer.android.com/reference/android/content/Intent?hl=pt-br>) usa dois parâmetros, um `Context`

(<https://developer.android.com/reference/android/content/Context?hl=pt-br>) e uma **Class** (<https://developer.android.com/reference/java/lang/Class?hl=pt-br>).

O parâmetro **Context**

(<https://developer.android.com/reference/android/content/Context?hl=pt-br>) é usado primeiro porque a classe **Activity**

(<https://developer.android.com/reference/android/app/Activity?hl=pt-br>) é uma subclasse de **Context** (<https://developer.android.com/reference/android/content/Context?hl=pt-br>).

Nesse caso, o parâmetro **Class**

(<https://developer.android.com/reference/java/lang/Class?hl=pt-br>) do componente do app, ao qual o sistema entrega o **Intent**

(<https://developer.android.com/reference/android/content/Intent?hl=pt-br>), é a atividade a ser iniciada.

- O método **putExtra()**

([https://developer.android.com/reference/android/content/Intent?hl=pt-br#putExtra\(java.lang.String,%20java.lang.String\)](https://developer.android.com/reference/android/content/Intent?hl=pt-br#putExtra(java.lang.String,%20java.lang.String)))

adiciona o valor de **EditText** ao intent. Um **Intent** pode carregar tipos de dados como pares de chave-valor chamados de *extras*.

Sua chave é uma **EXTRA_MESSAGE** pública constante porque a próxima atividade usa a chave para recuperar o valor de texto. É recomendável definir chaves para intents extras com o nome do pacote do app como prefixo. Isso garante que as chaves sejam únicas caso seu app interaja com outros.

- O método **startActivity()**

([https://developer.android.com/reference/android/app/Activity?hl=pt-br#startActivity\(android.content.Intent\)](https://developer.android.com/reference/android/app/Activity?hl=pt-br#startActivity(android.content.Intent)))

inicia uma instância da **DisplayMessageActivity** especificada pelo **Intent**

(<https://developer.android.com/reference/android/content/Intent?hl=pt-br>). Em seguida, você precisa criar essa classe.

Observação: o Navigation Architecture Component permite usar o Navigation Editor para associar uma atividade a outra. Depois que o relacionamento é criado, você pode usar a API para iniciar a segunda atividade quando o usuário acionar a ação associada, como clicar em um botão. Para saber mais, consulte **Navegação** (<https://developer.android.com/topic/libraries/architecture/navigation?hl=pt-br>).

Criar a segunda atividade

Para criar a segunda atividade, siga as etapas a seguir:

1. Na janela **Project** clique com o botão direito do mouse na pasta **app** e selecione **New > Activity > Empty Activity**.
2. Na janela **Configure Activity**, insira "DisplayMessageActivity" em **Activity Name**. Deixe todas as outras propriedades definidas como padrão e clique em **Finish**.

O Android Studio realizar estas três ações automaticamente:

- Cria o arquivo **DisplayMessageActivity**.
- Cria o arquivo de layout **activity_display_message.xml**, que corresponde ao arquivo **DisplayMessageActivity**.
- Adiciona o elemento `<activity>` (<https://developer.android.com/guide/topics/manifest/activity-element?hl=pt-br>) necessário em **AndroidManifest.xml**.

Se você executar o app e tocar no botão na primeira atividade, a segunda atividade será iniciada, mas estará vazia. Isso ocorre porque a segunda atividade utiliza o layout vazio fornecido pelo modelo.

Adicionar uma visualização de texto

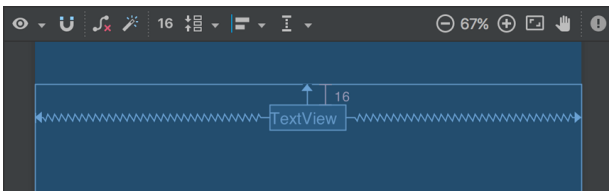



Figura 1. A visualização de texto centralizada no topo do layout.

A nova atividade inclui um arquivo de layout vazio. Siga estas etapas para adicionar uma exibição de texto ao local em que a mensagem aparece:

1. Abra o arquivo **app > res > layout > activity_display_message.xml**.
2. Clique em **Enable Autoconnection to Parent**  na barra de ferramentas. Isso habilita a conexão automática. Veja a Figura 1.
3. No painel **Palette**, clique em **Text**, arraste uma **TextView** para o layout e solte-a próximo ao centro superior do layout, para que ela se encaixe na linha vertical exibida. O Autoconnect adiciona limitações esquerda e direita para colocar a visualização no centro horizontal.

4. Crie mais uma limitação do topo da visualização de texto para o topo do layout, de forma que ela apareça como é mostrado na figura 1.

Opcionalmente, faça ajustes ao estilo do texto expandindo **textAppearance** no painel **Common Attributes** da janela **Attributes** e altere atributos como **textSize** e **textColor**.

Exibir a mensagem

Nesta etapa, você modificará a segunda atividade para exibir a mensagem que foi passada pela primeira.

1. Em **DisplayMessageActivity**, adicione o seguinte código ao método **onCreate()**:

KOTLIN (#KOTLIN)**JAVA**

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_display_message);

    // Get the Intent that started this activity and extract the string
    Intent intent = getIntent();
    String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);

    // Capture the layout's TextView and set the string as its text
    TextView textView = findViewById(R.id.text_view);
    textView.setText(message);
}
```

2. Pressione **Alt+Enter** ou **Option+Return** no Mac para importar as outras classes necessárias:

KOTLIN (#KOTLIN)**JAVA**

```
import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;
```

Adicionar navegação para cima

Todas as que não são a tela inicial precisam oferecer uma navegação que direcione o usuário à tela do pai lógico na hierarquia do app. Para fazer isso, adicione um botão **Up** na barra de apps (<https://developer.android.com/training/appbar?hl=pt-br>).

Para adicionar um botão **Up**, é necessário declarar qual atividade é o pai lógico no arquivo AndroidManifest.xml

(<https://developer.android.com/guide/topics/manifest/manifest-intro?hl=pt-br>). Abra o arquivo em **app > manifests > AndroidManifest.xml**, localize a tag `<activity>` de `DisplayMessageActivity` e substitua-a com o seguinte:

```
<activity android:name=".DisplayMessageActivity"
          android:parentActivityName=".MainActivity">
    <!-- The meta-data tag is required if you support API level 15 and lo
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity" />
</activity>
```

O sistema Android adicionará automaticamente o botão **Up** à barra de apps.

Executar o app

Clique em **Apply changes**

na barra de ferramentas para executar o app. Quando ele for aberto, digite uma mensagem no campo de texto e toque em **Send** para que ela seja exibida na segunda atividade.



Figura 2. App aberto, com texto inserido na tela esquerda e exibido na direita.

Pronto, você criou seu primeiro app Android!

Para continuar aprendendo as noções básicas de desenvolvimento de apps para Android, volte para [Criar seu primeiro app](https://developer.android.com/training/basics/firstapp?hl=pt-br) (https://developer.android.com/training/basics/firstapp?hl=pt-br) e siga os outros links fornecidos lá.

[Anterior](#)



[Criar uma IU simples](https://developer.android.com/training/basics/firstapp/building-ui?hl=pt-br)

(https://developer.android.com/training/basics/firstapp/building-ui?hl=pt-br)

Content and code samples on this page are subject to the licenses described in the [Content License](https://developer.android.com/license?hl=pt-br) (https://developer.android.com/license?hl=pt-br). Java is a registered trademark of Oracle and/or its affiliates.

Last updated 2019-12-27.