

Roteiro de Laboratório 04


José Cassiano Grassi Gunji


Intent

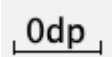
Neste roteiro vamos explorar alguns usos do recurso *intent* do Android. Com *intent* é possível fazer nosso aplicativo navegar para outras *activities*, interagir com recursos do telefone (como fazer ligações, enviar mensagens de texto, tirar uma foto) e interagir com outros aplicativos (como compartilhar algo com um aplicativo de rede social).

Criando a primeira Activity

Inicie o Android Studio, crie um novo aplicativo usando o assistente para uma *Empty Activity* assim como fizemos nos exemplos anteriores. Escolha um nível mínimo de API apropriado e a linguagem Java.

Clique em  (View Options) na barra de ferramentas do *Layout Editor* e verifique se a opção *Show All Constraints* está marcada.

Certifique-se de que a opção *Autoconnect* está desativada. Uma dica na barra de ferramentas exibirá  (Enable Autoconnection to Parent) quando essa opção estiver desativada.

Clique em  (Default Margins) na barra de ferramentas e selecione **16**. Se necessário, é possível ajustar as margens de cada visualização posteriormente.

Remova o *TextView* que foi colocado no layout automaticamente.

Da palheta *Text*, arraste um *Plain Text* para a parte superior de seu design. Um *EditText* é adicionado ao seu layout.

Clique na visualização no editor de design. Agora você pode usar as alças quadradas para redimensionar a visualização em cada canto e as âncoras circulares de limitação em cada lado. Para ter um controle melhor, recomenda-se aumentar o zoom no editor. Para isso, use os botões **Zoom** na barra de ferramentas do *Layout Editor*.

Clique e mantenha pressionada a âncora na lateral superior, arraste-a para cima até que ela se encaixe na parte superior do layout e solte-a. Essa é uma *constraint* (restrição): ela limita a visualização dentro da margem padrão que foi definida. Nesse caso, você a define a 16 dp a partir do topo do layout.

Use o mesmo processo para criar uma limitação do lado esquerdo da visualização para o lado esquerdo do layout.

O resultado deve ser semelhante ao apresentado na Figura 1.

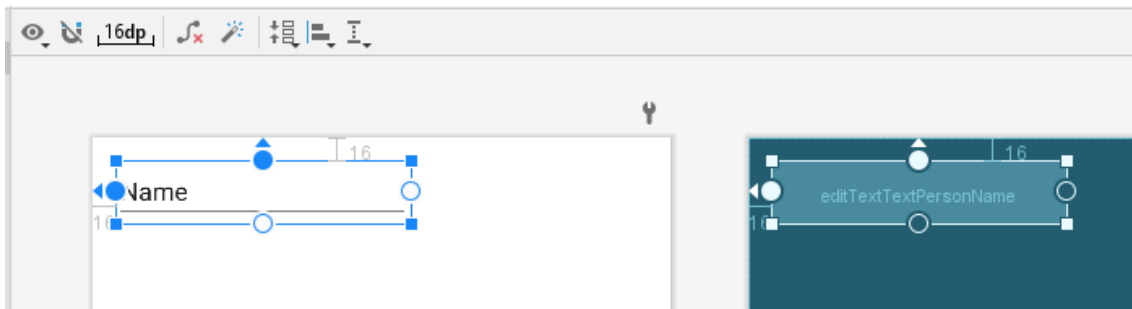



Figura 1: A caixa de diálogo é limitada ao topo e à esquerda do layout pai.

No painel *Palette*, clique em *Buttons*. Arraste o *widget Button* para o editor de design e solte-o perto do lado direito. Crie uma restrição do lado esquerdo do botão para o lado direito da caixa de texto.

Para restringir as views em um alinhamento horizontal, crie uma restrição entre as linhas de base do texto. Para fazer isso, clique com o botão direito do mouse no botão e selecione *Show Baseline* . A âncora da linha de base aparece dentro do botão. Clique e mantenha essa âncora pressionada e arraste-a para a âncora da linha de base que aparece na caixa de texto adjacente.

O resultado será semelhante ao mostrado na Figura 2.

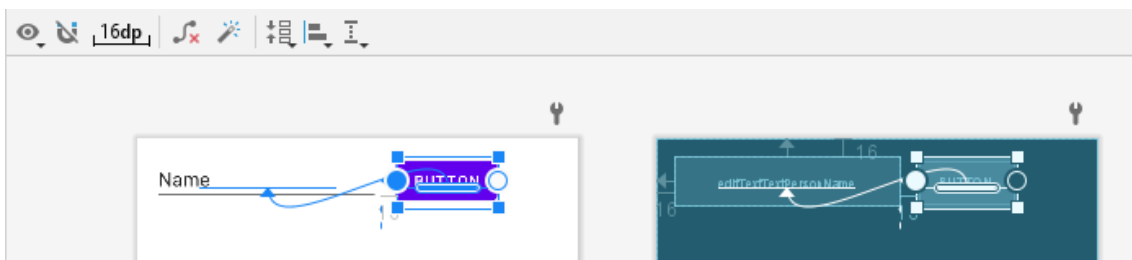


Figura 2: O botão é restrito à direita da caixa de texto e da linha de base.

Para criar um layout responsivo a diferentes tamanhos de tela, é necessário esticar a caixa de texto para preencher todo o espaço horizontal que sobre depois de considerar botão e as margens.

Selecione as duas visualizações. Para fazer isso, clique em uma visualização, mantenha a tecla Shift pressionada, clique na outra visualização, em seguida, clique com o botão direito do mouse em uma delas e selecione *Chains > Create Horizontal Chain*. O layout aparecerá como mostrado na Figura 3.

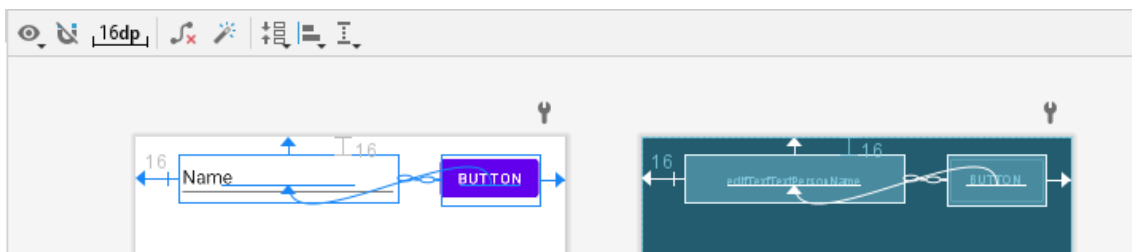


Figura 3: O resultado da escolha de *Create Horizontal Chain*.

Uma *chain* é uma restrição bidirecional entre duas ou mais views que permite que você disponha as views encadeadas em conjunto.

Clique no botão. Na janela *Attributes*, use o *Constraint Widget* para definir a margem direita como 16 dp. Observe a Figura 4.

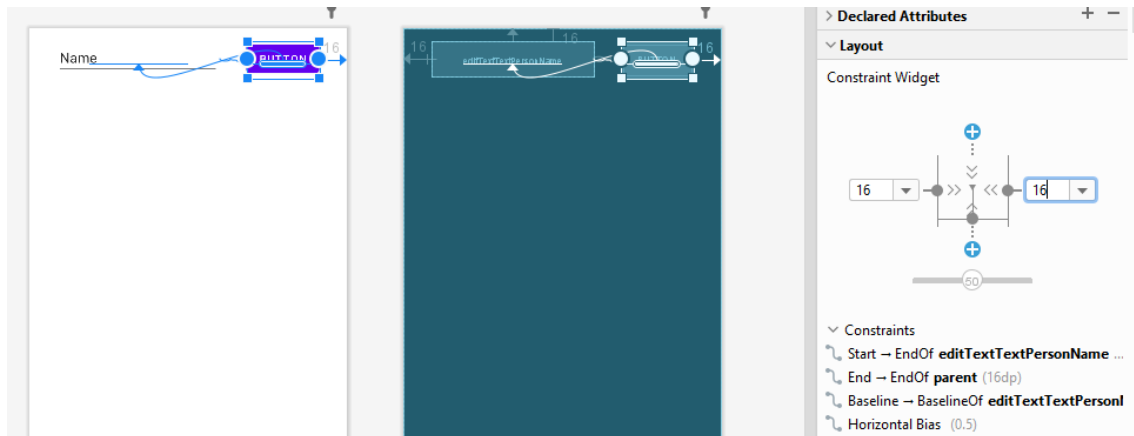


Figura 4: Definindo a restrição do lado direito do botão para 16 dp.

Clique na caixa de texto para visualizar os atributos. Em seguida, clique no indicador de largura duas vezes para que ele seja definido como uma linha em zigue-zague (Match Constraints), conforme indicado pela seta na Figura 5.

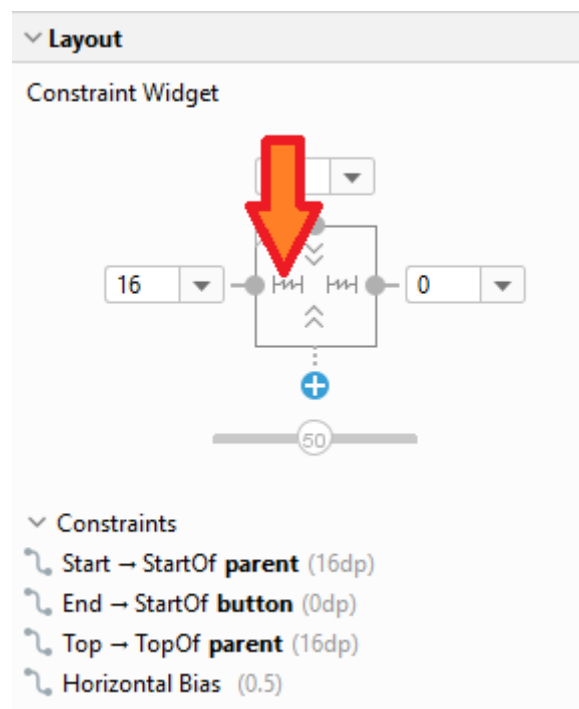


Figura 5: Alterando a restrição para Match Constraints.

Match constraints significa que a largura se expande para atender à definição das limitações horizontais e das margens. Portanto, a caixa de texto será esticada para preencher o espaço horizontal que permanece depois de considerar o botão e todas as margens.

Agora o layout deve estar como o da figura Figura 6.

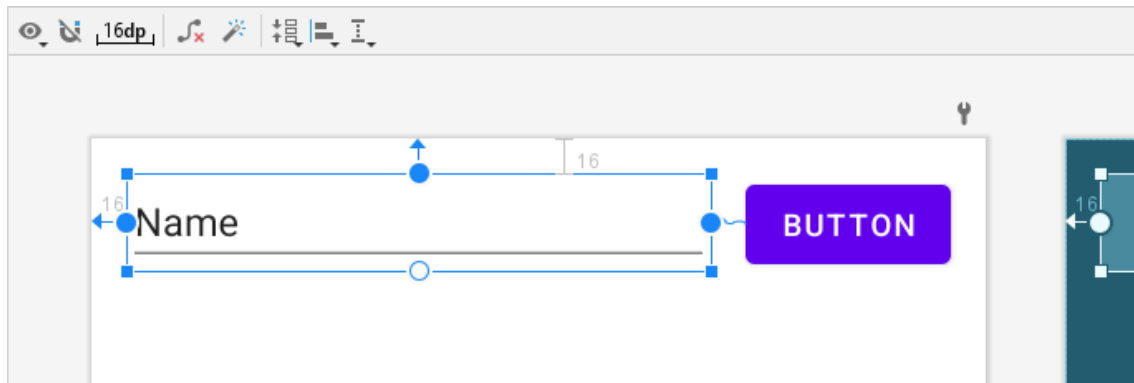


Figura 6: Layout responsivo do aplicativo.

Vamos configurar os textos de nosso aplicativo. Para que ele esteja preparado para o mercado internacional, vamos definir os textos padrões em inglês. Depois disso, vamos criar uma localização para o português.

Clique no *EditText*. Apague o conteúdo do atributo *Text*. Crie um novo recurso de texto para o atributo *hint* chamado **digiteAlgo** com o valor **Type something**.

Clique no botão. Crie um novo recurso de texto para o atributo *Text* chamado **enviar** com o valor **Send**.

Na aba *Project*, abra seu arquivo *strings.xml* a partir de *app->res->values->strings.xml*. A seguir, abra o *Translator Editor*. Acrescente uma nova localização clicando no botão indicado pela seta na Figura 7. A seguir, digite **ptBR** para encontrar a localização para português brasileiro.

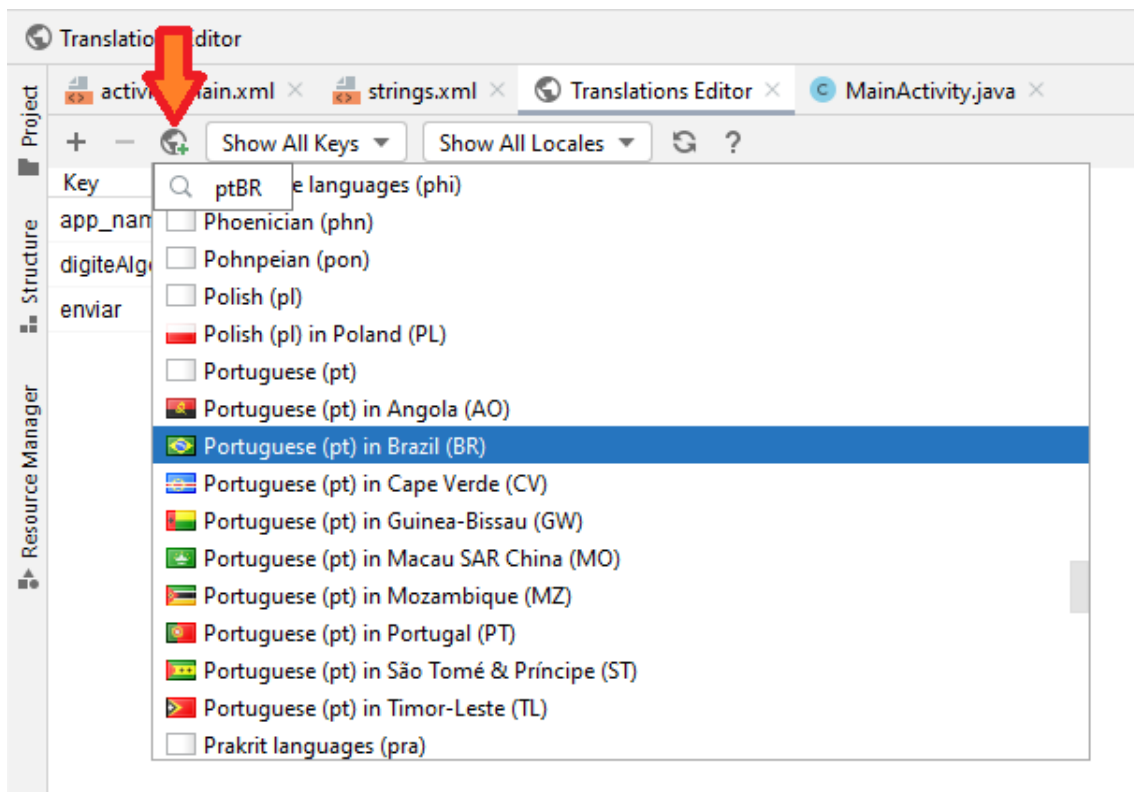


Figura 7: Adicionando a localização para português brasileiro.

Agora faça as traduções dos recursos de texto como mostrado na Figura 8.

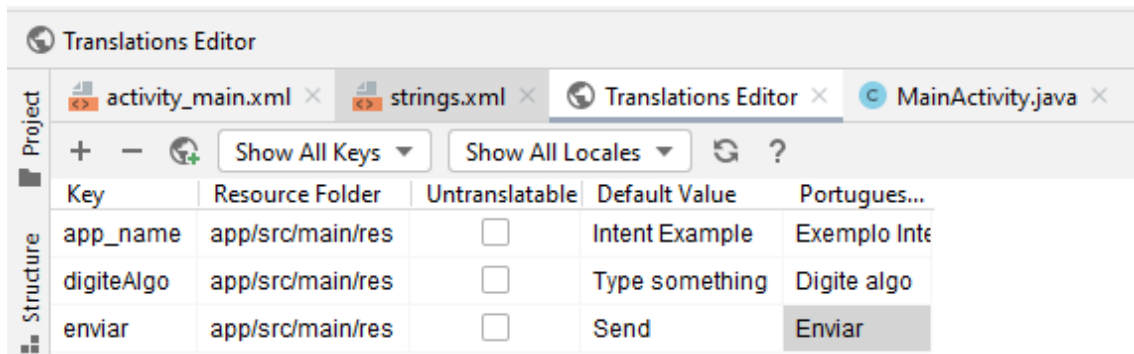



Figura 8: Criando as traduções dos textos do aplicativo.

Criando a segunda Activity

Na janela *Project* clique com o botão direito do mouse na pasta *app* e selecione *New > Activity > Empty Activity*.

Na janela *New Android Activity*, insira "MostreMensagemActivity" em *Activity Name*. Deixe todas as outras propriedades definidas como padrão e clique em *Finish*.

Abra o arquivo *app > res > layout > activity_display_message.xml*.

Clique em *Enable Autoconnection to Parent*  na barra de ferramentas. Isso habilita a conexão automática. Veja a Figura 1.

No painel *Palette*, clique em *Text*, arraste uma *TextView* para o layout e solte-a próximo ao centro da parte superior do layout para que ela se encaixe na linha vertical exibida. O *Autoconnect* adiciona restrições esquerda, direita e superior para colocar a visualização no centro horizontal.

O layout da nova atividade deve estar como na Figura 9.

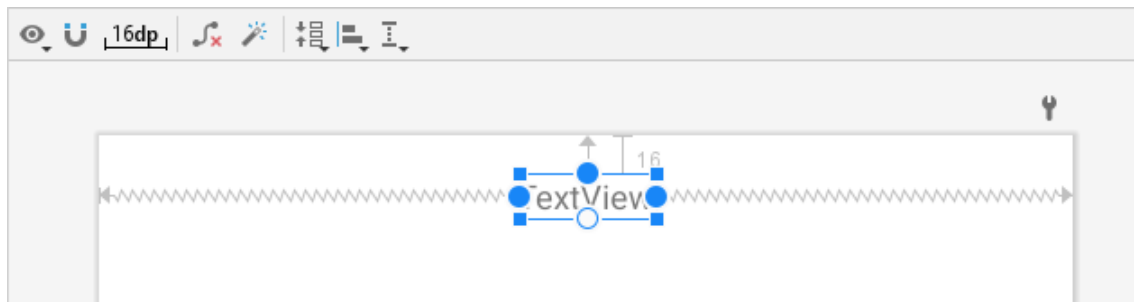


Figura 9: A view de texto centralizada no topo do layout.

Iniciando uma nova atividade

Vamos criar um método que será chamado quando o usuário clicar no botão *Enviar* da atividade principal. Desta vez, não vamos usar ouvintes de eventos. Vamos criar um método e associá-lo diretamente ao evento *onClick* do botão.

No arquivo *app->java->br.edu.cruzeirosul.exemplointent->MainActivity.java*, adicione o *stub* de método *envieMensagem()* a seguir:

```

1 package br.edu.cruzeirosul.exemplointent;
2
3 import ...
4
5
6
7
8 public class MainActivity extends AppCompatActivity {
9
10     @Override
11     protected void onCreate(Bundle savedInstanceState) {
12         super.onCreate(savedInstanceState);
13         setContentView(R.layout.activity_main);
14     }
15
16     /**
17      * Chamado quando o usuário clica no botão Enviar
18      * @param view O botão Enviar
19      */
20     public void envieMensagem(View view){
21         // Faça algo em resposta ao clique do botão
22     }
23 }

```

Figura 10: Stub do método `envieMensagem()`.

Retorne ao arquivo **activity_main.xml** para chamar o método com o 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](#). Especificamente, o método tem as seguintes características:

- c. Acesso público.
- d. Um valor vazio.
- e. [View](#) como o único parâmetro.

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

Um [Intent](#) é um objeto que fornece vínculos de tempo de execução entre componentes separados, como duas atividades. O [Intent](#) representa a intenção do app de fazer algo. Você pode usar intents para uma ampla variedade de tarefas, mas, nesta lição, a intent iniciará outra atividade.

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

```

10 public class MainActivity extends AppCompatActivity {
11     public static final String EXTRA_MESSAGE = "br.edu.cruzeirodosul.exemplointent.MESSAGE";
12     @Override
13     protected void onCreate(Bundle savedInstanceState) {
14         super.onCreate(savedInstanceState);
15         setContentView(R.layout.activity_main);
16     }
17
18     /**
19      * Chamado quando o usuário clica no botão Enviar
20      * @param view O botão Enviar
21      */
22     public void envieMensagem(View view){
23         Intent intent = new Intent( packageContext: this, MostreMensagemActivity.class);
24         EditText editText = findViewById(R.id.editTextTextPersonName);
25         String mensagem = editText.getText().toString();
26         intent.putExtra(EXTRA_MESSAGE, mensagem);
27         startActivity(intent);
28     }
29 }

```

Figura 11: Escrevendo a lógica do botão Enviar.

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

- O construtor do [Intent](#) usa dois parâmetros, um [Context](#) e um [Class](#). O parâmetro [Context](#) é usado primeiro porque a classe [Activity](#) é uma subclasse de [Context](#). Nesse caso, o parâmetro [Class](#) do componente do app, ao qual o sistema envia o [Intent](#), que, nesse caso, é a atividade a ser iniciada.
- O método [putExtra\(\)](#) adiciona o valor de `EditText` à `intent`. Uma `Intent` pode carregar tipos de dados como pares de chave-valor chamados de *extras*. Sua chave é uma constante `EXTRA_MESSAGE` pública porque a próxima atividade usará a chave para recuperar o valor do 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\(\)](#) inicia uma instância da `DisplayMessageActivity` especificada pela [Intent](#). Em seguida, você precisa criar essa classe.

Exibindo a mensagem

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

Em `MostreMensagemActivity`, adicione o seguinte código ao método `onCreate()`:

```

11      @Override
12      protected void onCreate(Bundle savedInstanceState) {
13          super.onCreate(savedInstanceState);
14          setContentView(R.layout.activity_mostre_mensagem);

15          // Obtém a intent que iniciou esta activity e extrai o string
16          Intent intent = getIntent();
17          String mensagem = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);

18          // Captura o TextView do layout e define o string como seu Text.
19          TextView textView = findViewById(R.id.textview);
20          textView.setText(mensagem);
21      }
22
23

```

Figura 12: Exibindo a mensagem recebida pelo intent.

Adicionando navegação para cima

Todas as telas que não são o ponto de entrada principal, ou seja, 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](#).

Para adicionar um botão **Up**, é necessário declarar qual atividade é o pai lógico no arquivo [AndroidManifest.xml](#). Abra o arquivo em **app > manifests > AndroidManifest.xml**, localize a tag `<activity>` para `MostreMensagemActivity` e a substitua-a pelo seguinte:

```

15      <activity
16          android:name=".MostreMensagemActivity"
17          android:parentActivityName=".MainActivity"
18          android:exported="false">
19          <meta-data
20              android:name="android.app.lib_name"
21              android:value="" />
22      </activity>

```

Figura 13: Configurando a hierarquia entre as activities.

Execute o aplicativo

Nas figuras abaixo, são apresentadas capturas de tela do aplicativo sendo executado com a localização português brasileiro e com tema diurno. Tente outras variações e verifique o resultado.

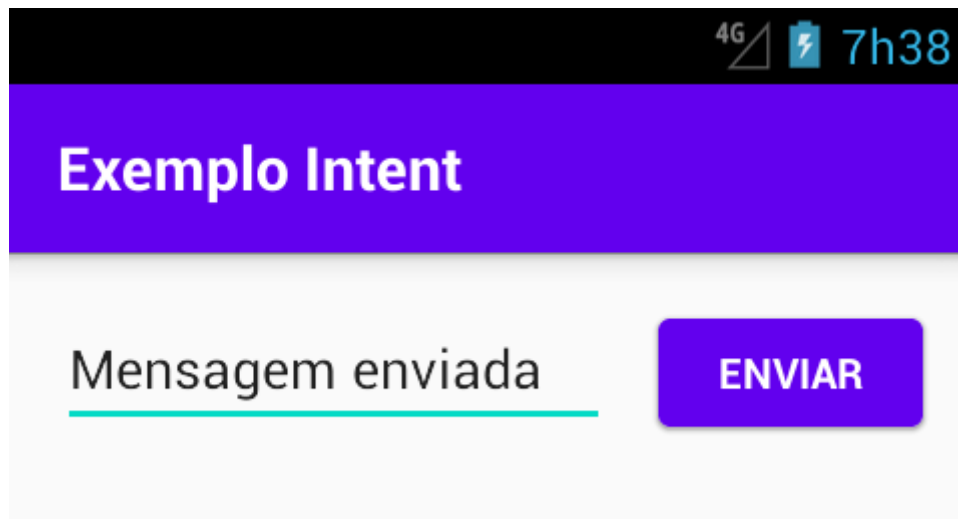


Figura 14: Atividade principal.

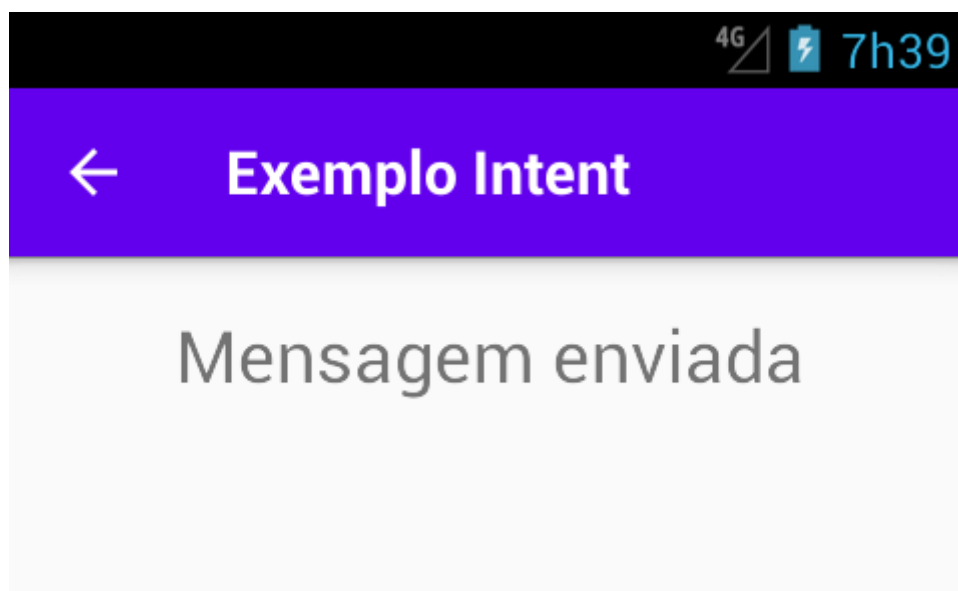


Figura 15: Segunda atividade.

Veja que há uma pequena diferença entre o comportamento de seu aplicativo quando você pressiona o botão “up” (↵) e quando você pressiona o botão voltar (ou o gesto voltar) no próprio dispositivo. O botão (ou gesto) voltar faz com que a tela inicial seja apresentada com o texto digitado. Já ao clicar no botão “up”, a tela inicial é apresentada sem o texto digitado.

À primeira vista, parece ser uma situação em que o estado do aplicativo deve ser salvo sobrescrevendo-se o método `onSaveInstanceState()` e recuperando o estado salvo com o método `onCreate()` ou o método `onRestoreInstanceState()`. Mas se você fizer isso, vai notar que o problema continua.

Neste caso, o botão “up” está na verdade iniciando um *intent* para a atividade principal, o que faz com que uma nova instância da atividade seja criada. Por ser uma nova instância, ela terá novos campos de persistência e não conseguirá recuperar os campos gravados pela instância original.

Uma maneira de se resolver isso é modificar ou incluir a propriedade *launchMode* da atividade principal com o valor “singleTop” no arquivo `app->manifests->AndroidManifest.xml` como mostrado na Figura 16.

```

<activity
    android:name=".MainActivity"
    android:launchMode="singleTop"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>

    <meta-data
        android:name="android.app.lib_name"
        android:value="" />
</activity>

```

Figura 16: Configurando o `launchMode` da `MainActivity`.

Cada *activity* é instanciada e colocada na pilha de navegação do aplicativo (back stack). Assim, cada vez que o usuário pressionar o botão voltar do dispositivo, a *activity* atual é retirada da pilha, também da memória, e a próxima *activity* é trazida para o primeiro plano.

Com o *launchMode* configurado para *singleTop*, se uma atividade estiver sendo instanciada e já houver uma instância da mesma atividade no topo da pilha de navegação, o aplicativo vai ativar a instância já existente ao invés de criar uma nova.

Para saber mais sobre a navegação na pilha de atividades:

<https://developer.android.com/guide/components/activities/tasks-and-back-stack>

Para saber mais sobre os modos de lançamento de uma atividade:

<https://developer.android.com/guide/topics/manifest/activity-element#lmode>

Referências

<https://developer.android.com/training/basics/firstapp>

DEITEL, Paul et al. Android para Programadores: uma abordagem baseada em aplicativos, 1ª Edição, 2013.