



# Desenvolvimento de Sistemas

---

## Usabilidade e acessibilidade: conceitos, aplicabilidade e técnicas

Nos conteúdos anteriores você aprendeu quão importantes são a usabilidade e acessibilidade para o desenvolvimento de aplicações. Sendo assim, a utilização desses dois conceitos em aplicações *desktop* com Java torna-se fundamental, não apenas para criar uma melhor experiência para os usuários, mas também para possibilitar a criação de aplicações para que sejam utilizadas por pessoas com alguma deficiência ou particularidade que influencie no uso dessa aplicação.

De modo geral, a usabilidade está muito relacionada à acessibilidade, pois criar uma aplicação com recursos de acessibilidade representa preocupação em dar ao usuário uma melhor experiência na utilização do sistema, e, para que isso ocorra, os princípios de usabilidade acabam sendo utilizados de modo complementar. Quando se desenvolve um sistema com acessibilidade, busca-se a inclusão das pessoas com alguma deficiência. A principal forma de aplicar a acessibilidade é por meio das tecnologias assistivas (TA), que podem ser empregadas tanto em aplicações *desktop* quanto para *web*.

Entre os usos das aplicações criadas com tecnologias assistivas está a utilização de recursos de classes Java específicas para auxiliar pessoas com deficiência visual. Isso ocorre porque essas classes auxiliam os *softwares* de leitores de tela, que são utilizados para apoiar a navegação na interface da aplicação. O funcionamento desses leitores ocorre com a leitura dos componentes de interface por meio de sintetizadores que analisam a interface gráfica do usuário (GUI). Alguns exemplos de leitores disponíveis no mercado são: Jaws, Virtual Vision, Orca.

## Iniciando uma aplicação com recursos de usabilidade e acessibilidade

Para entender essas classes que auxiliam na criação de aplicações *desktop* tanto na usabilidade quanto na acessibilidade, será desenvolvida uma tela com componentes para um cadastro de usuário. Para isso, serão utilizadas propriedades de acessibilidade e usabilidade para os seguintes componentes: **JTextField**, **JLabel**, **JMenu**, **JMenuItem**, **JBButton** e **JLayeredPane**.

### Criação da tela inicial com JFrame

Para iniciar a aplicação, é necessário criar uma tela de interface principal. Crie um projeto Java Ant, como os construídos nos conteúdos anteriores, e em seguida crie um **JFrame**. Dê a ele o nome de “Main”.

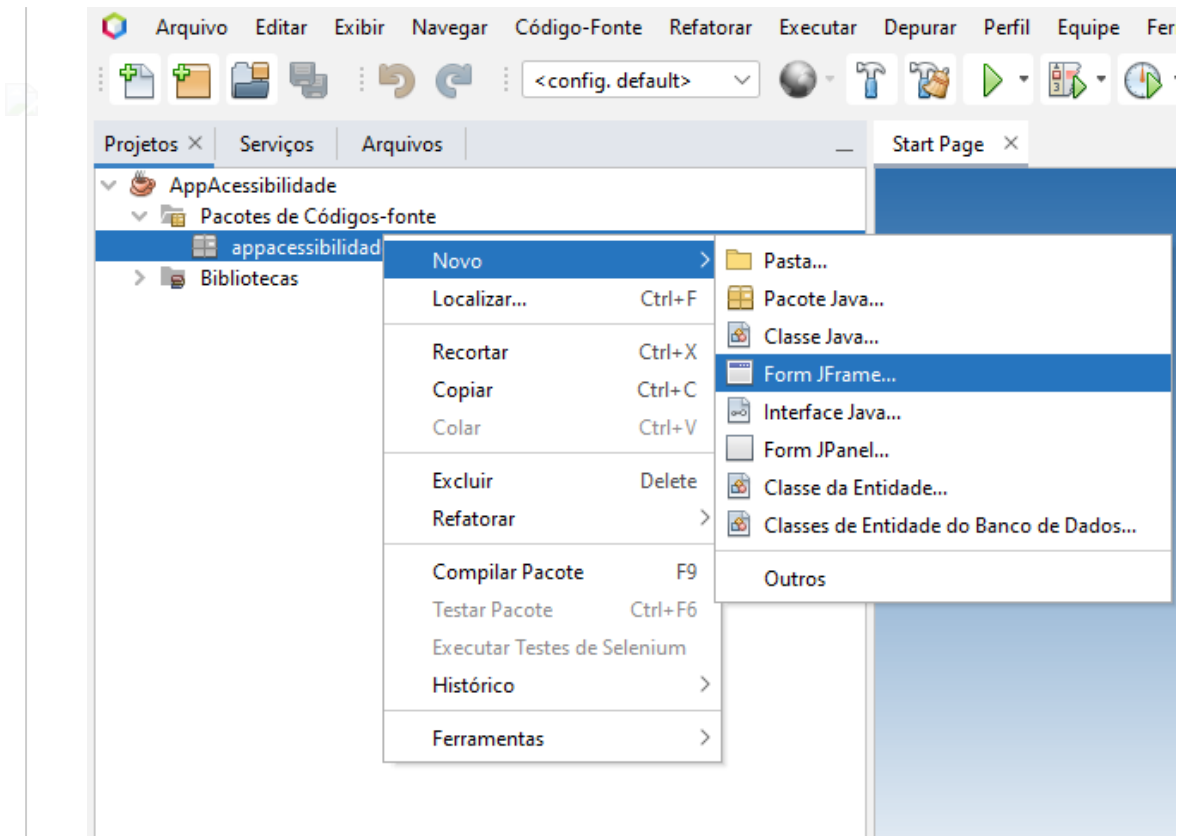
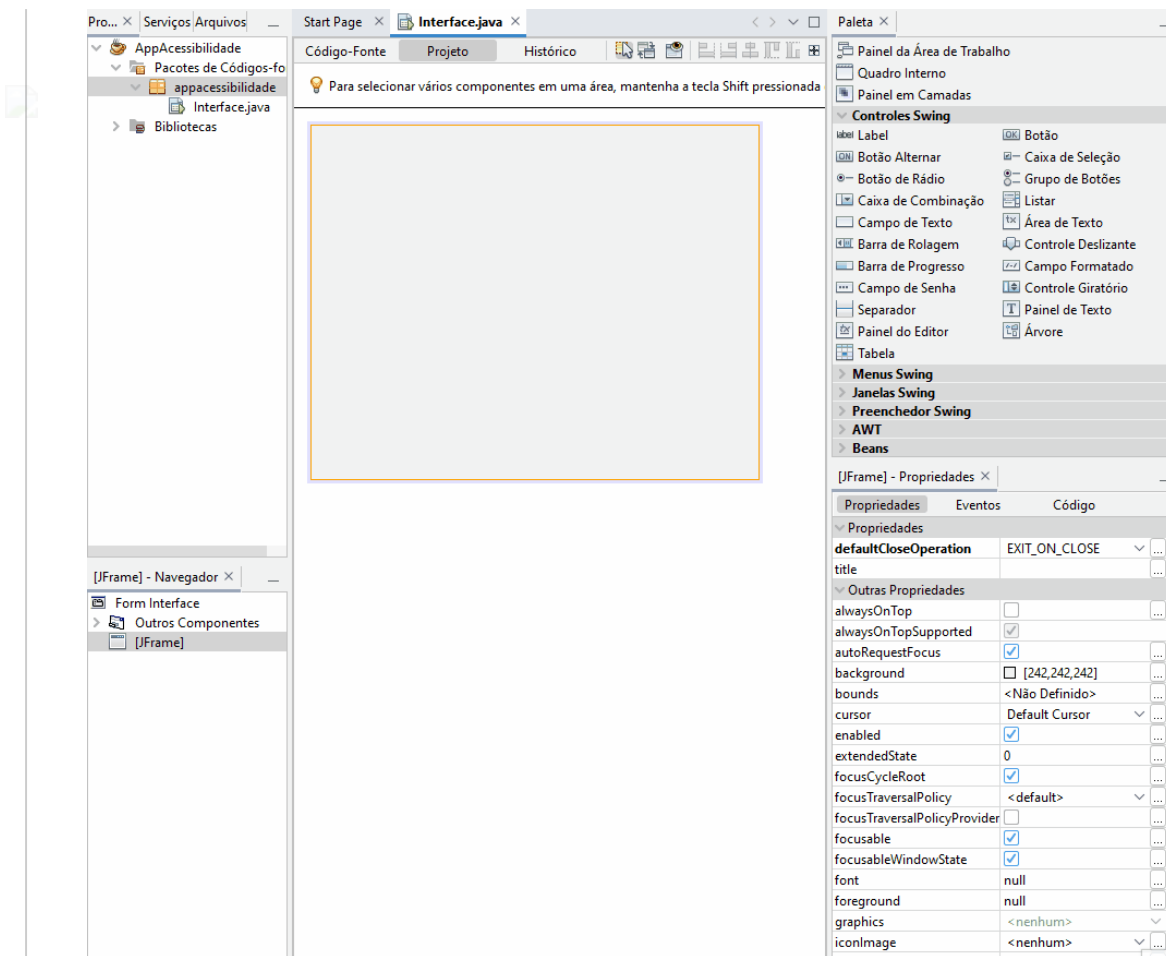


Figura 1 – Criação da tela inicial com **JFrame**

Fonte: NetBeans (2022)

Em seguida, arraste um *panel* em camada (**JLayeredPane**). Para deixar mais nítido o componente, defina uma borda (essa definição encontra-se na aba **Propriedades**, opção **border**). Escolha a opção **borda com título** e defina o texto para “Informações Cadastrais”. **A criação desse texto descritivo auxilia as tecnologias assistivas a localizarem o componente ao qual o rótulo está associado, assim como proporciona uma melhor usabilidade.**

Confira esse processo no GIF (*graphics interchange format*) a seguir:



## Texto de dica da ferramenta – Método `setToolTipText(String)`

Com a base da tela inicial já construída, comece agora a desenvolver sua aplicação com os princípios da acessibilidade e usabilidade para apoiar uma melhor utilização pelos leitores de tela e usuários com alguma deficiência visual. Para isso, inicie com a definição **`setToolTipText`**. Essa propriedade define um texto sobre o elemento quando o usuário colocar o cursor sobre esse elemento, mostrando assim uma descrição do componente. Para definir essa propriedade, abra as propriedades do **`JLayeredPane`** criado e informe na opção **`toolTipTex`** o texto “Insira os seus dados para o cadastro”.

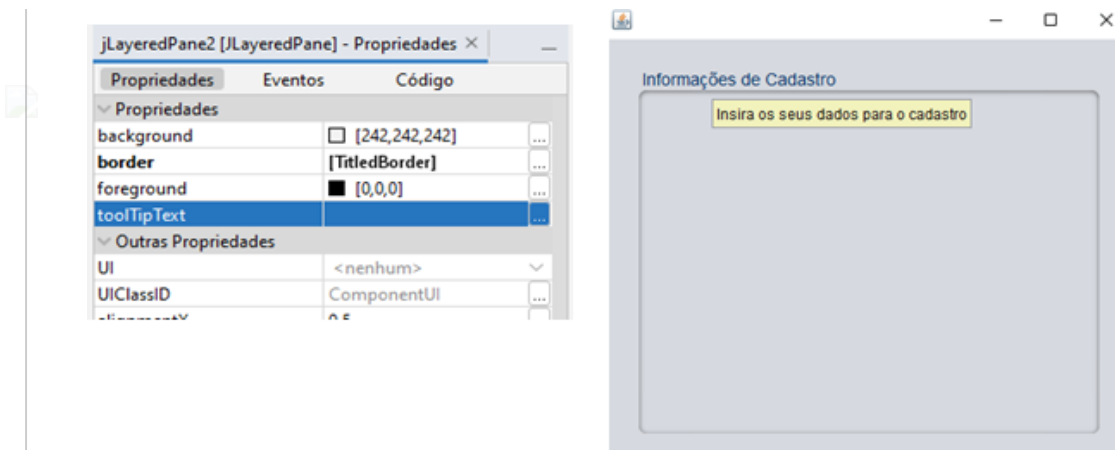


Figura 2 – Definição do texto descritivo do componente

Fonte: NetBeans (2022)

Quando se cria a propriedade, um código é gerado automaticamente com a definição **setToolTipText** para o elemento. Esse código só pode ser mudado na aba de propriedades da interface.

O **setToolTipText** acaba auxiliando também em termos de usabilidade para o usuário conseguir uma opção de ajuda, pois ele cria explicações sobre o elemento e consequentemente sobre sua funcionalidade. O código gerado é definido a seguir:

```
public class Main extends javax.swing.JFrame {

    public Main() {
        initComponents();
    }

    @SuppressWarnings("unchecked")
    private void initComponents() {

        interfaceApp = new javax.swing.JLayeredPane();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        interfaceApp.setToolTipText("Insira os seus dados para o cadastro");
    }
}
```

Como você pôde observar, o código dentro do método  **initComponents()** é gerado devido à propriedade definida na interface. O nome do componente **JLayeredPane** aparecerá como padrão ainda, mas é possível, na aba **Código**, na propriedade **Nome da Variável**, modificar seu nome para "interfaceApp", sendo o **JLayerdPane** definido com novo nome agora.

## Definição do nome de acessibilidade do componente – Método `setAccessibleName(String)`

Este método define o nome que será lido quando o leitor de tela estiver com o foco no componente, ou seja, quando o elemento estiver sendo selecionado para leitura. Isso especifica de maneira mais geral qual é o componente.

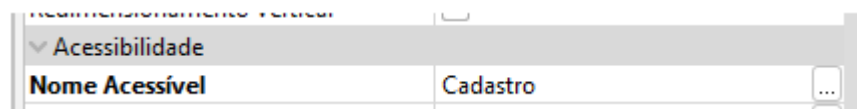


Figura 3 – Propriedade Nome Acessível

Fonte: NetBeans (2022)

Este é código gerado:

```
InterfaceApp.getAccessibleContext().setAccessibleName("Cadastro");
```

## Descrições de acessibilidade – Método `setAccessibleDescription(String)`

A utilização de textos descritivos aos componentes faz com que os leitores de tela, ao lerem esse componente, consigam fazer uma descrição mais ampla dele. Essa propriedade torna-se assim fundamental para o início do processo de deixar a aplicação com uma boa acessibilidade. O método `setAccessibleDescription()` faz parte do **`getAccessibleContext()`**, que recupera o contexto acessível.

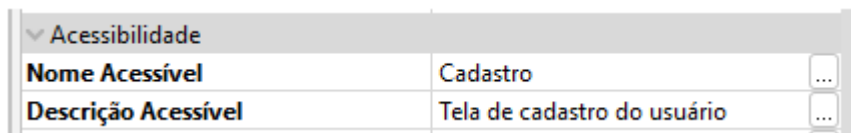


Figura 4 – Propriedade Descrição Acessível

Fonte: NetBeans (2022)

Confira o código gerado:

```
InterfaceApp.setToolTipText("Insira os seus dados para o cadastr  
o");  
    //Códigos adicionais da interface...  
    InterfaceApp.getAccessibleContext().setAccessibleName("Cadastro");  
    InterfaceApp.getAccessibleContext().setAccessibleDescription("Tela  
de cadastro do usuário");
```

## Rótulos associados ao seu componente

Quando se cria um componente na tela com um rótulo, como na situação da criação de uma **`JLabel`** e um **`JTextField`**, é preciso associar esses dois elementos entre si. Por padrão, esses dois componentes não ficam relacionados. No momento que for criado esse vínculo, os leitores de tela, caso não achem um texto de acessibilidade para o elemento, lerão o texto definido na *label*. Para fazer essa implementação, utilize o método **`setLabelFor()`** do **`JLabel`**. Essa propriedade se encontra na barra de

ferramentas **Propriedades**, na opção **labelFor**. Ao selecionar o combo **labelFor**, serão mostrados todos os componentes da tela, então você deve selecionar qual componente quer associar à *label*. Neste caso, associe ao campo de inserir o nome ou **txtNome**, que foi o nome definido para esse **JTextField**.

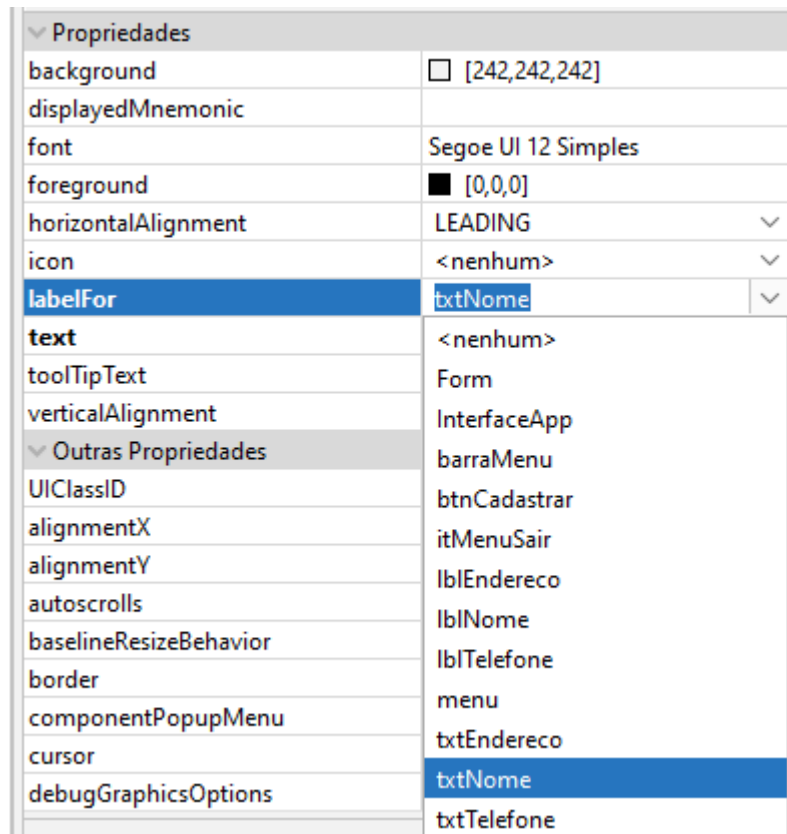


Figura 5 – Propriedade labelFor

Fonte: NetBeans (2022)

O código gerado será este:

```
lblNome.setLabelFor(txtNome);  
lblNome.setText("Nome completo");
```



## Navegação entre os componentes pelo teclado

Os leitores de tela têm seu funcionamento baseado nos elementos que podem ter foco na tela, o que é observado quando se clica em **Alt + Tab** na tela. Quando se faz isso, pode-se notar que haverá navegação entre cada elemento. Contudo, é muito trabalhoso o usuário que utiliza leitores de tela navegar até um elemento específico, então, criar um atalho permitirá a ele ir direto ao componente de tela desejado.

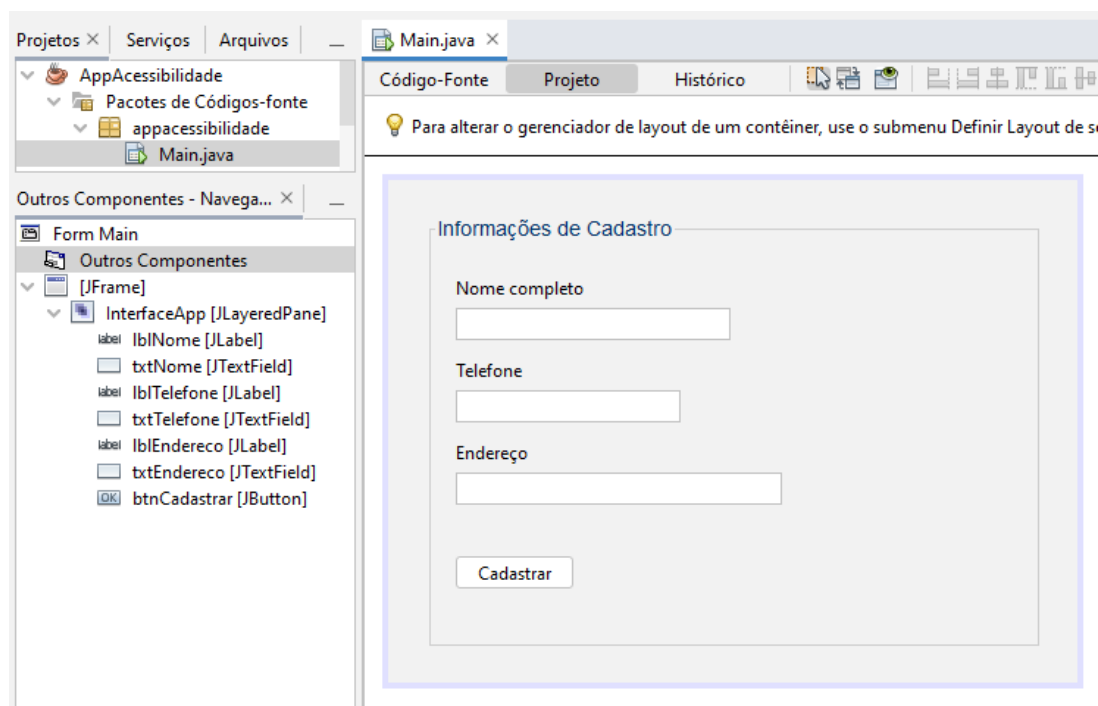


Figura 6 – Interface com novos componentes adicionados

Fonte: NetBeans (2022)

Para exemplificar essa situação, crie mais componentes na interface a partir dos conceitos já vistos nos conteúdos anteriores sobre **JLabel**, **Jbutton**, **JTextField**.

Na imagem anterior foi definida a interface inicial de cadastro apresentando também os nomes dos componentes na navegação à esquerda.

## Criação de método para funções de acessibilidade

Após o ajuste de sua tela inicial com os novos componentes, você pode agora criar alguns códigos. Defina então a propriedade de navegação de teclado para o botão **Cadastrar**. Neste exemplo, quando o usuário apertar as teclas de atalho **Alt + C**, ele irá direto para o botão **Cadastrar**.

Para isso, procure o código-fonte da sua tela e, logo após o método construtor da classe, crie o método **geraAcessibilidade()**. Dentro desses métodos, crie algumas definições de acessibilidade associadas aos componentes da tela.

```
public class Main extends javax.swing.JFrame {

    /**
     * CONSTRUTOR COM MÉTODO geraAcessibilidade SENDO CHAMDO
     */
    public Main() {
        initComponents();
        geraAcessibilidade();
    }

    //MÉTODO SEM RETORNO COM DEFINIÇÕES DE ACESSIBILIDADE PARA INTE
RFACE
    public void geraAcessibilidade(){
    }
```

Com o método criado, é preciso definir agora a ação da tecla de atalho. Para utilizar as ações que envolvem o uso de teclas, deve-se importar a classe **KeyEvent**, que criará um evento associado a teclas

pressionadas.

```
//IMPORT DA CLASSE PARA USO DAS TECLAS DE ATALHO
import java.awt.event.KeyEvent;

public class Main extends javax.swing.JFrame {
    /**
     * CONSTRUTOR COM MÉTODO geraAcessibilidade SENDO CHAMDO
     */
    public Main() {
        initComponents();
        geraAcessibilidade();
    }

    //MÉTODO SEM RETORNO COM DEFINIÇÕES DE ACESSIBILIDADE PARA INTE
RFACE
    public void geraAcessibilidade() {
    }
```

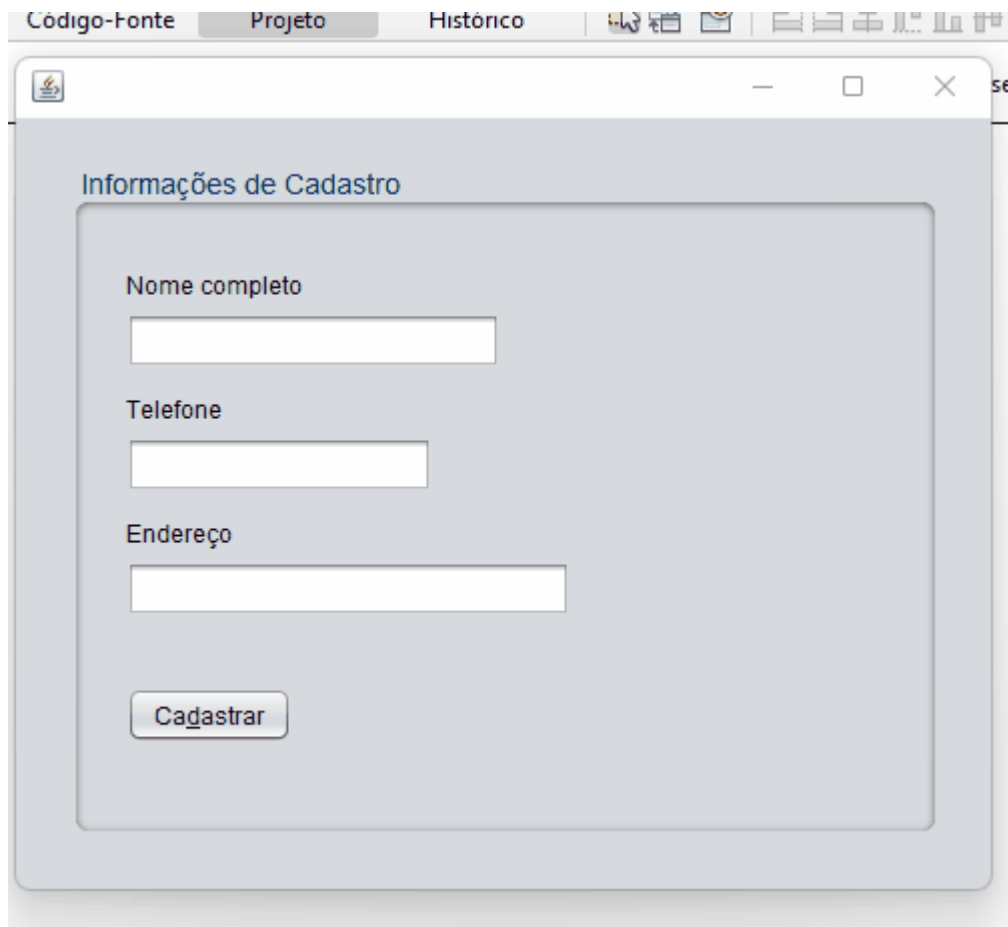
Criado o import, defina em seguida o **mnemonic**, ou alternativa do teclado, por meio do método **setMnemonic()**. Essa propriedade será associada ao botão **Cadastrar (btnCadastrar)**. Para definir o evento associado à tecla D, utilize os virtual-key codes, que são uma padronização de códigos associada a cada tecla. Neste caso, o código será “VK\_D”, que representa a letra **D** do teclado.

O código ficará da seguinte forma:

```
//IMPORT DA CLASSE PARA USO DAS TECLAS DE ATALHO
import java.awt.event.KeyEvent;
```

```
public class Main extends javax.swing.JFrame {  
  
    /**  
     * CONSTRUTOR COM MÉTODO geraAcessbilidade SENDO CHAMDO  
     */  
    public Main() {  
        initComponents();  
        geraAcessbilidade();  
    }  
  
    //MÉTODO SEM RETORNO COM DEFINIÇÕES DE ACESSIBILIDADE PARA INTERFAC  
    public void geraAcessbilidade() {  
  
        //DEFINE MÉTODO QUE ASSOCIA TECLA DE ATALHO "ATL + D" PARA O BC  
        btnCadastrar.setMnemonic(KeyEvent.VK_D);  
    }  
}
```

Compreenda melhor com o GIF a seguir:



## Atalhos associado ao menu

A criação de atalhos para o menu é também muito importante para o desenvolvimento de aplicações com uma boa acessibilidade, pois o usuário poderá navegar diretamente a um item de menu específico. Porém, o método associado a essa ação é o **setAccelerator()**. Esse método permite aos usuários acessarem diretamente um item de menu sem que o seu menu apareça. Esse método permite também definir diferentes teclas de atalhos, não sendo obrigatória a utilização do **Alt**.

Neste exemplo de teclas de atalho para o menu, será definido um item de menu para o usuário sair do sistema. Serão utilizados nesta criação os componentes **JMenuBar** e **JMenu**.

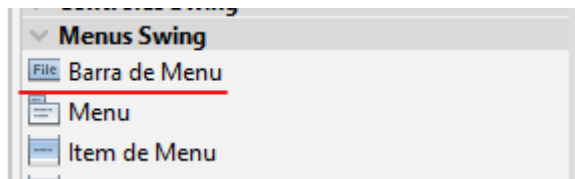


Figura 7 – Componentes de criação do menu

Fonte: NetBeans (2022)

Apague um dos menus que são gerados juntos ao arrastar a barra para sua interface e renomeie um deles com o nome “Menu”. Crie dentro dele um item de menu chamado “Sair”.

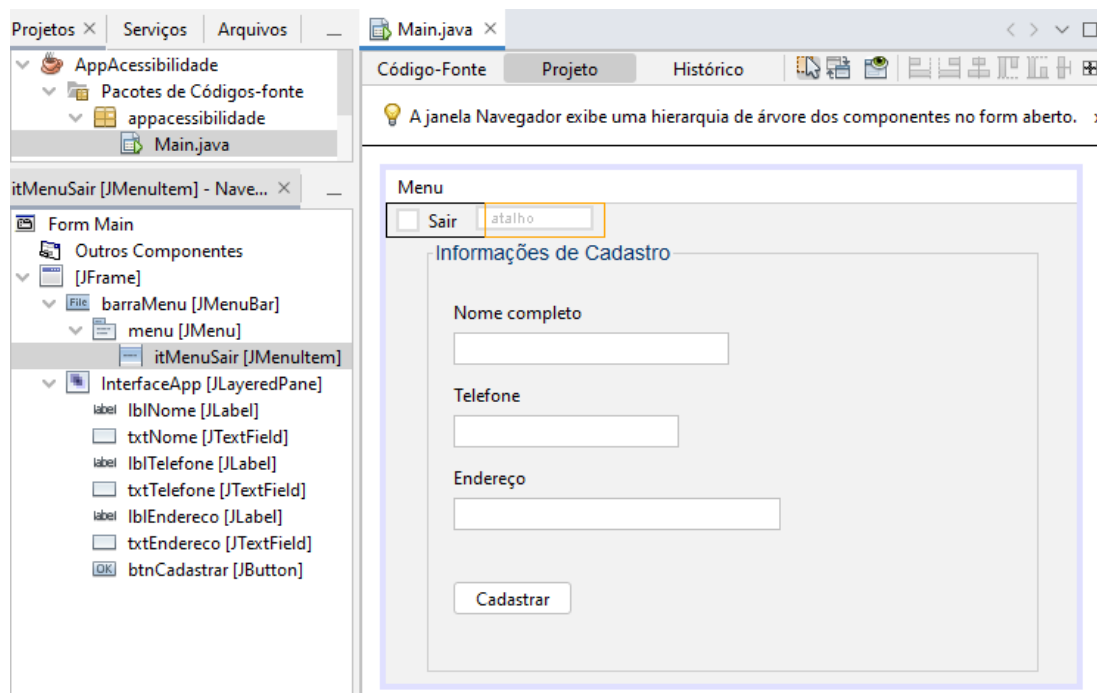
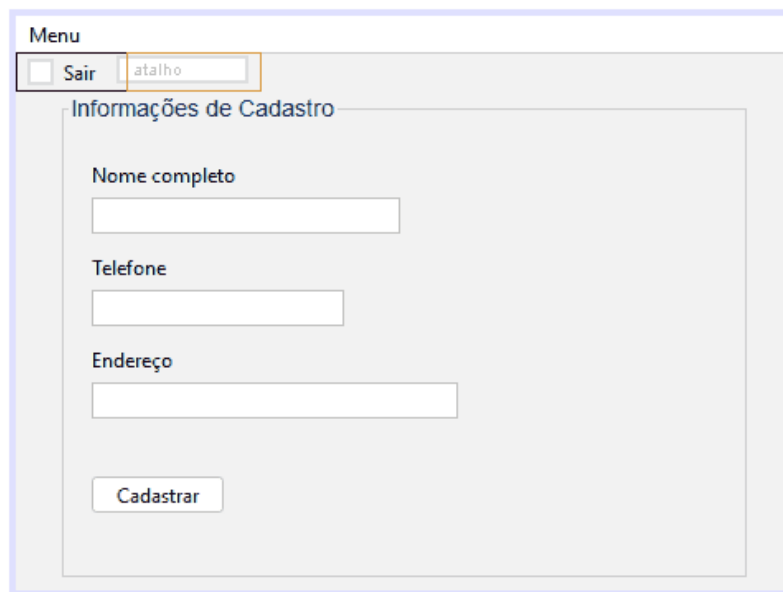


Figura 8 – Item de menu com texto “Sair”

Fonte: NetBeans (2022)

Ao criar um item de menu, observa-se um texto ao lado desse item com o termo “atalho”. Dois cliques sobre o texto “atalho” abrem uma tela na qual se pode definir as teclas de atalhos. Essa tela para criar as teclas de atalho é a interface de configuração relacionada ao método **setAccelerator()**. Para esse atalho, selecione a tecla **Shift** seguida da tecla **S**, o que possibilita a criação do comando **Shif + S** para acessar esse item de menu.

Analise o GIF a seguir:



Menu

☐ Sair    atalho

Informações de Cadastro

Nome completo

Telefone

Endereço

Cadastrar

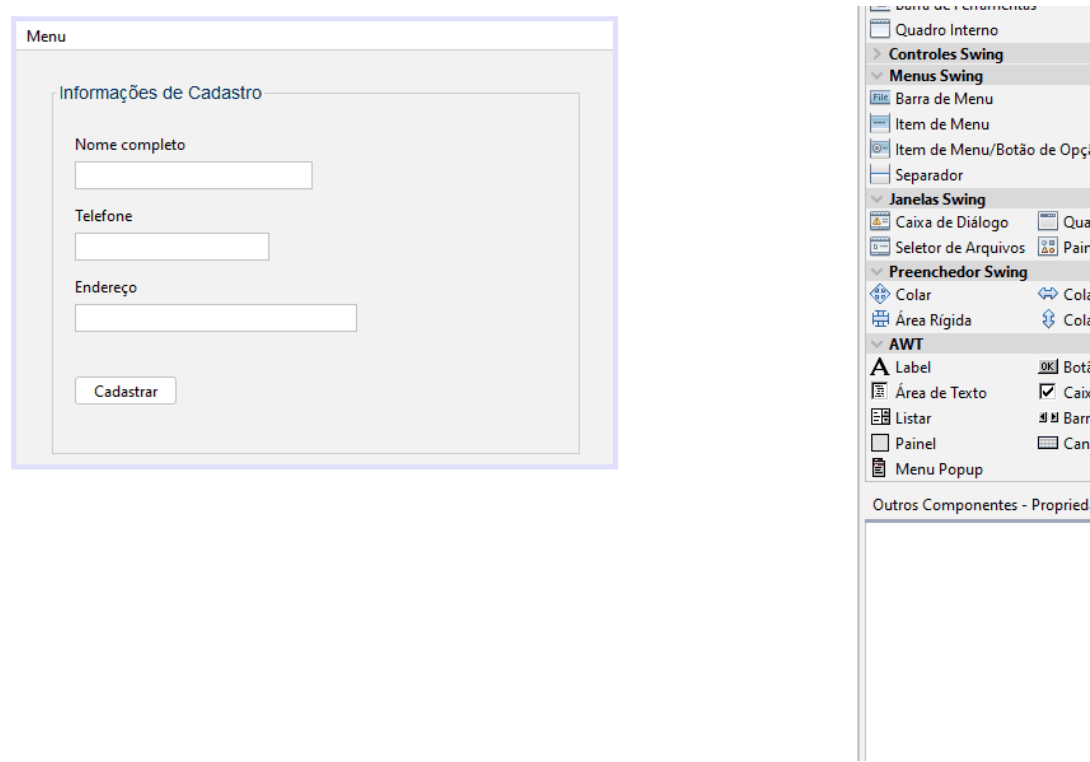
O código gerado será o seguinte:

```
/*METODO PARA CRIAR ATALHO SEM ABRIR MENU,  
VK_S REPRESENTA A LETRA "S"  
E SHIFT_DOWN_MASK REPRESENTA A TECLA "SHIFT"  
TEMOS DEPOIS O CÓDIGO DE DEFINIÇÃO DO TEXTO O ITEM DE MENU,  
setText(), E A ADIÇÃO DO ITEM DE MENU AO MENU COM menu.add()*/  
itMenuSair.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.a  
wt.event.KeyEvent.VK_S, java.awt.event.InputEvent.SHIFT_DOWN_MASK));  
itMenuSair.setText("Sair");  
menu.add(itMenuSair);
```

Depois de definir a tecla de atalho, é preciso criar ainda a ação de clique. Para o item de menu, isso será feito com o método **ActionPerformed()**. Dentro do método será criado um comando para sair

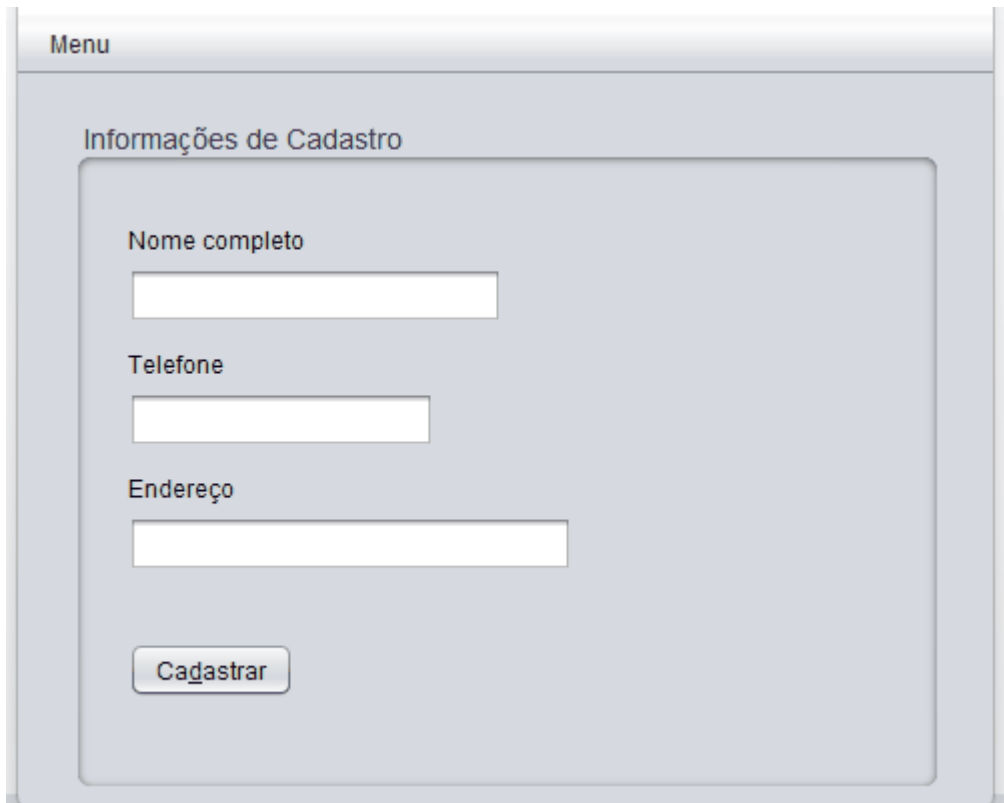
do sistema com a especificação de código **System.exit(0)**.

Veja a seguir um GIF com a implementação dessas etapas:



Após essa definição, é possível executar a aplicação e, ao apertar o atalho **Shift + S**, este fechará a aplicação.





Menu

Informações de Cadastro

Nome completo

Telefone

Endereço

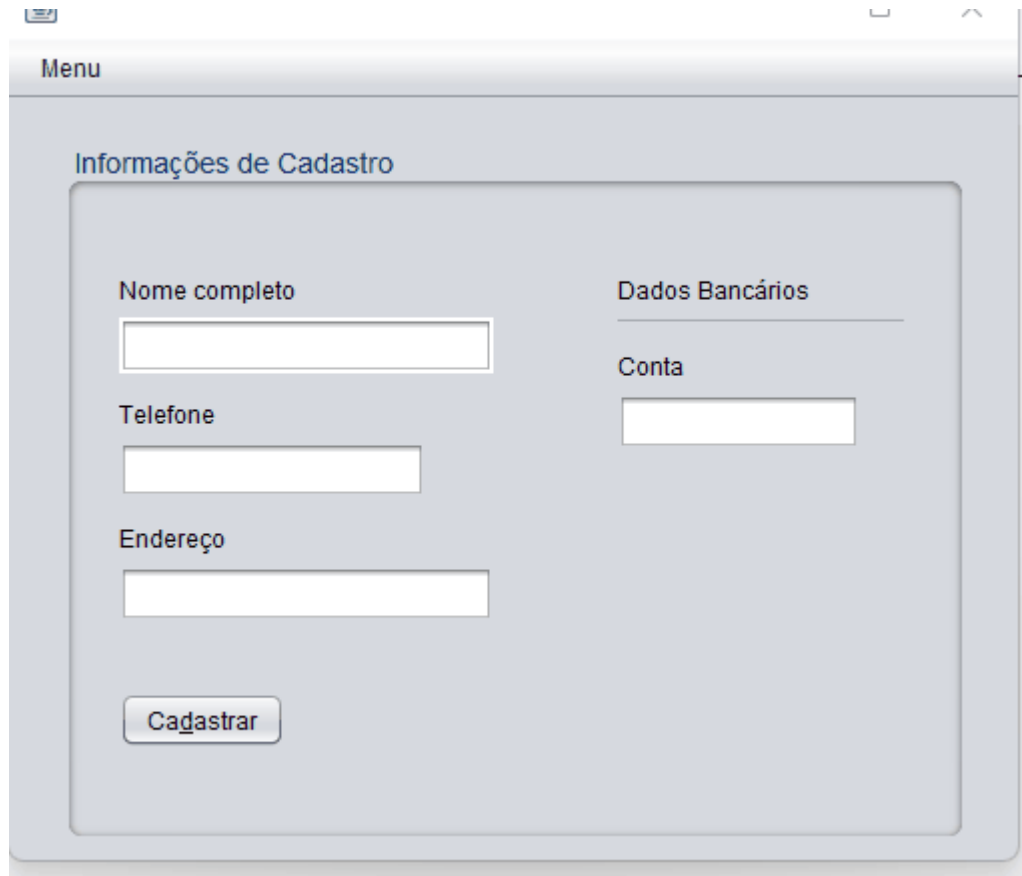
Cadastrar

## Políticas de prioridade – Método `FocusTraversalPolicy()`

A classe **FocusTraversalPolicy** lida com a definição da prioridade em que os elementos da tela são percorridos, mudando-se o foco por meio das teclas de atalho. Nem sempre a interface consegue seguir uma ordem que represente a maneira mais adequada para navegar entre as informações. Assim, quando desejar alterar o padrão da ordem de percorrer os elementos com a tecla **Tab**, você poderá utilizar um método específico que receba como parâmetro o componente da tela que será o próximo a receber o foco.

Para utilização da política de prioridade, será utilizado o método **NextFocusableComponent()**. Esse método recebe por parâmetro o componente que será o próximo a ser selecionado quando é pressionada a tecla **Tab**.

Para exemplificar essa utilização, defina mais alguns componentes em sua aplicação. Adicione mais um **TextField**, uma *label* e um separador; esses componentes criarão o *layout* para receber os dados de uma conta bancária. Observe esse processo no seguinte GIF:



The image shows a screenshot of a Java Swing window titled "Menu". Inside the window is a panel titled "Informações de Cadastro". This panel is divided into two columns. The left column, under the heading "Informações de Cadastro", contains three text input fields labeled "Nome completo", "Telefone", and "Endereço". The right column, under the heading "Dados Bancários", contains a single text input field labeled "Conta". At the bottom left of the panel is a button labeled "Cadastrar". The layout demonstrates a sequential flow of focus from the name field to the phone field, then to the address field, and finally to the account field.

Ao observar a animação, note que os dados do banco estão em uma coluna separada, sendo a forma sequencial mais correta ir do campo do nome para o do telefone, depois para o campo do endereço e por fim o da conta. Contudo, a animação mostra que a sequência percorre do nome para a conta e retorna ao telefone.

Para realizar a modificação na sequência de prioridades que se quer definir, você pode utilizar a propriedade **nextFocusableComponent**. Nela, você especificará qual componente receberá o próximo foco. Defina o **textField** do telefone, que contém o nome **txtTelefone**. Observe que, ao clicar nas opções da propriedade, um combo aparecerá com os nomes de todos os componentes da tela que podem ser selecionados.

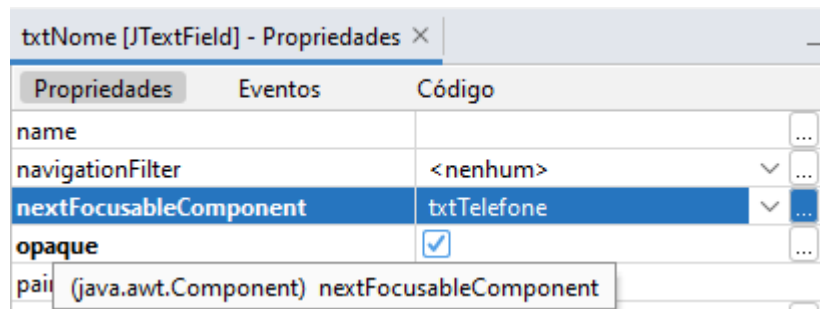


Figura 9 – Definição da propriedade que recebe o próximo foco após “Nome”

Fonte: NetBeans (2022)

Será definido também no **textField** de endereço, para que o foco seja alterado na sequência para a conta. Se você não definir assim, ele irá para o botão **Cadastrar** e somente depois para a conta.

A alteração ficará da seguinte forma:

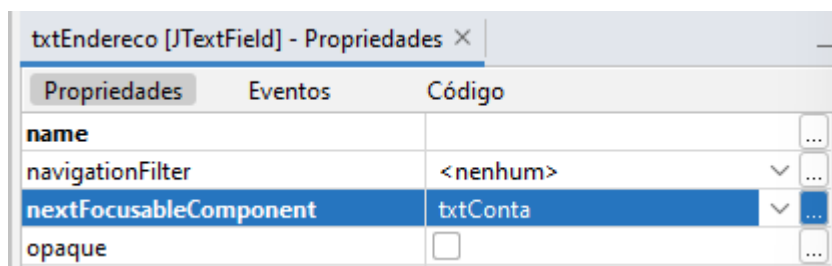


Figura 10 – Definição da propriedade que recebe o próximo foco após endereço

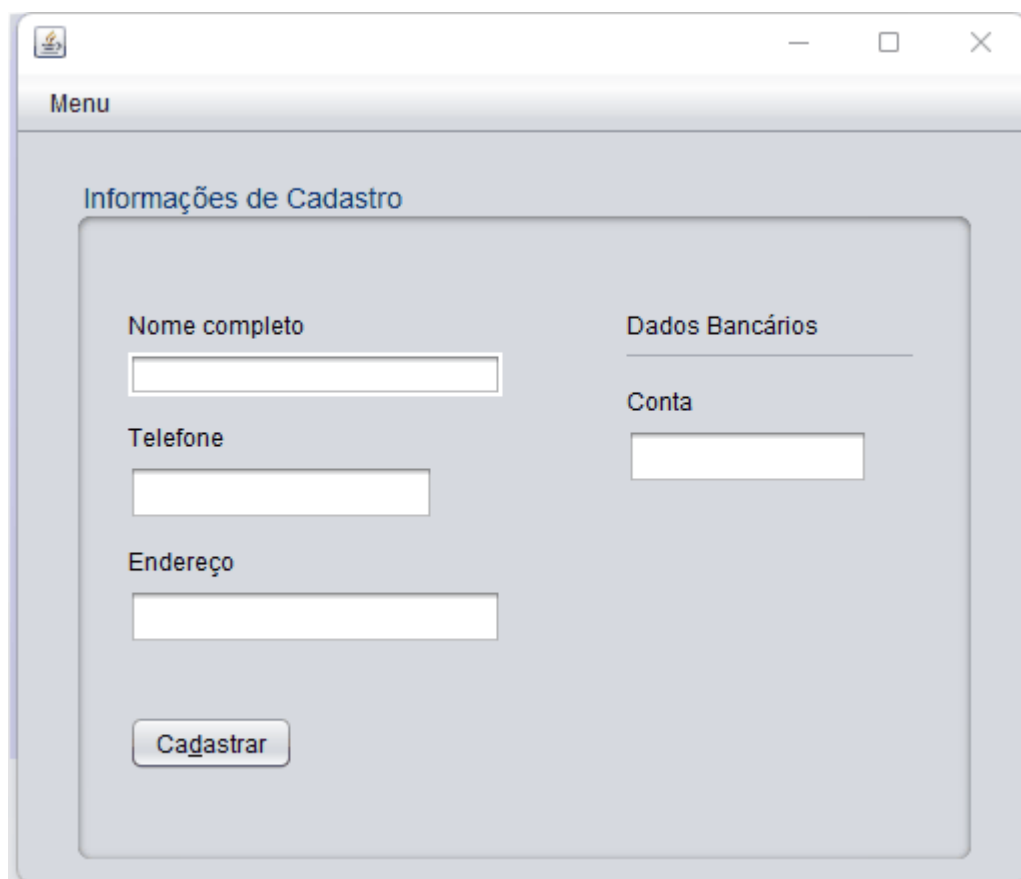
Fonte: NetBeans (2022)

Por fim, será definido também para o **textField** da conta, para direcionar o foco na sequência ao botão **Cadastrar**.

O código gerado, associado a essas propriedades, ficará assim:

```
txtNome.setNextFocusableComponent(txtTelefone);  
  
txtEndereco.setNextFocusableComponent(txtConta);  
  
txtConta.setNextFocusableComponent(btnCadastrar);
```

A organização sequencial de navegação ficará, portanto, como mostra o GIF a seguir, sendo executada agora da forma como foi planejada:



The screenshot shows a Java Swing window titled "Menu" with standard window controls (minimize, maximize, close). Inside the window is a panel titled "Informações de Cadastro". This panel contains two columns of input fields. The left column has three text boxes labeled "Nome completo", "Telefone", and "Endereço". The right column has one text box labeled "Conta" under the heading "Dados Bancários". At the bottom left of the panel is a button labeled "Cadastrar".

## Encerramento

Este conteúdo abordou a importância da utilização de acessibilidade e usabilidade na aplicação *desktop*. As propriedades trabalhadas neste material compõem um exemplo para desenvolver uma melhor experiência a usuários portadores de deficiência, especificamente aos usuários que necessitam de *softwares* leitores de tela. Reforça-se neste sentido a precaução no desenvolvimento de uma aplicação que atenda à diversidade de usuários que utilizarão o sistema.