

# Tópicos adicionais

## Interfaces gráficas

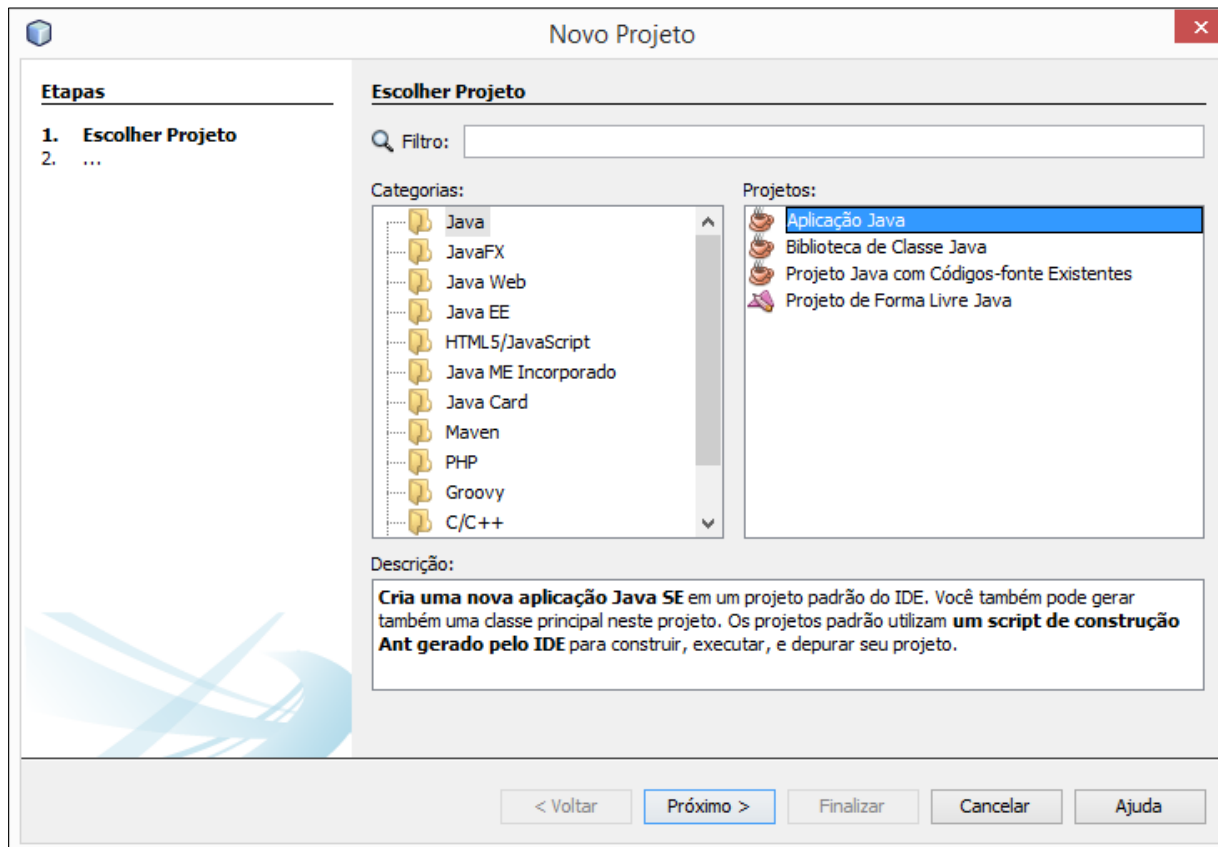
# Interfaces gráficas em Java

- O Java fornece bibliotecas de componentes para auxiliar na construção de interfaces gráficas.
  - **Componente:** objeto com capacidade de interagir com o usuário.
- As bibliotecas mais conhecidas são o **AWT** (Abstract Window Toolkit) e o **SWING**. Este último é mais atual, possuindo mais componentes e recursos, mas ainda permite a utilização em conjunto com o anterior.
- Estas bibliotecas são disponibilizadas através dos pacotes **javax.swing.\*** e **java.awt.\***.
- Algumas IDEs possuem ferramentas para suporte à construção de interfaces gráficas, que permitem ao desenvolvedor construí-la de forma visual, arrastando seus componentes e implementando suas ações.



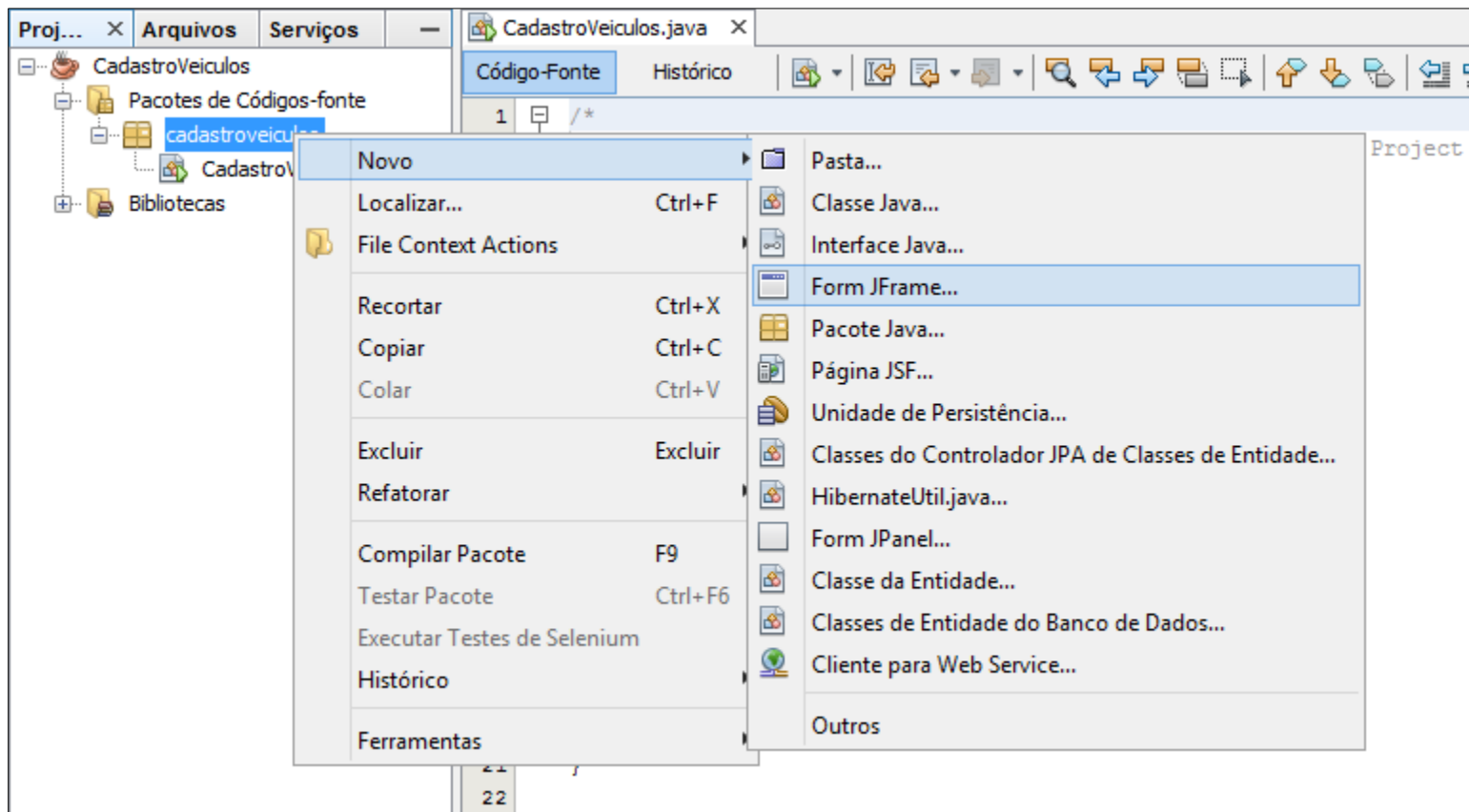
# Construção usando o NetBeans

- Como exemplo, criaremos um sistema para cadastro e consulta de veículos. Para criar uma aplicação com interface gráfica, crie um novo projeto, selecionando a categoria **Java** e o tipo **Aplicação Java**.

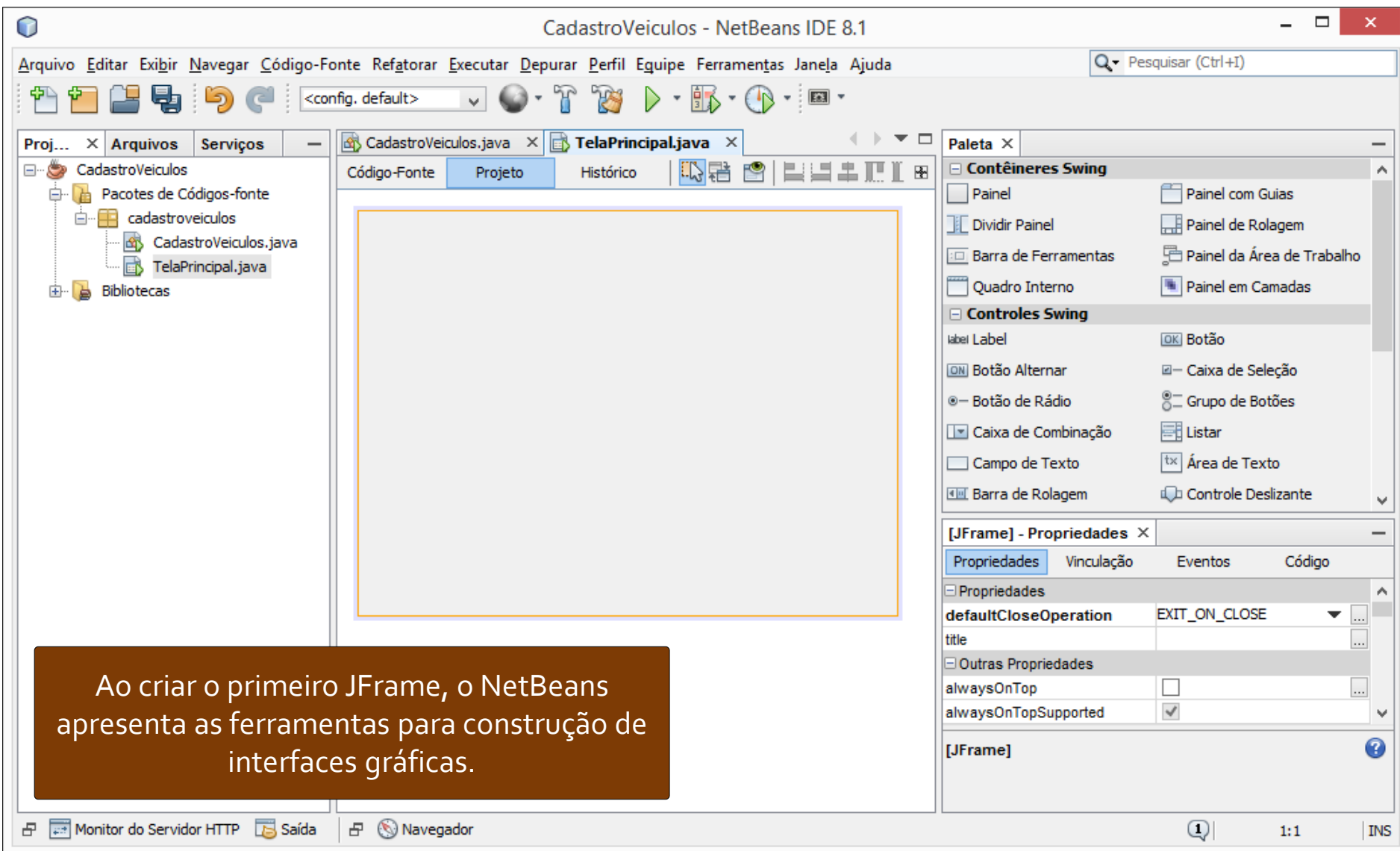


# Construção usando o NetBeans

- No pacote de códigos-fonte, clique com o botão direito > **Novo** > **Form JFrame**. Caso esta opção não apareça, busque-a em **Outros**. Chame a classe de **TelaPrincipal**.

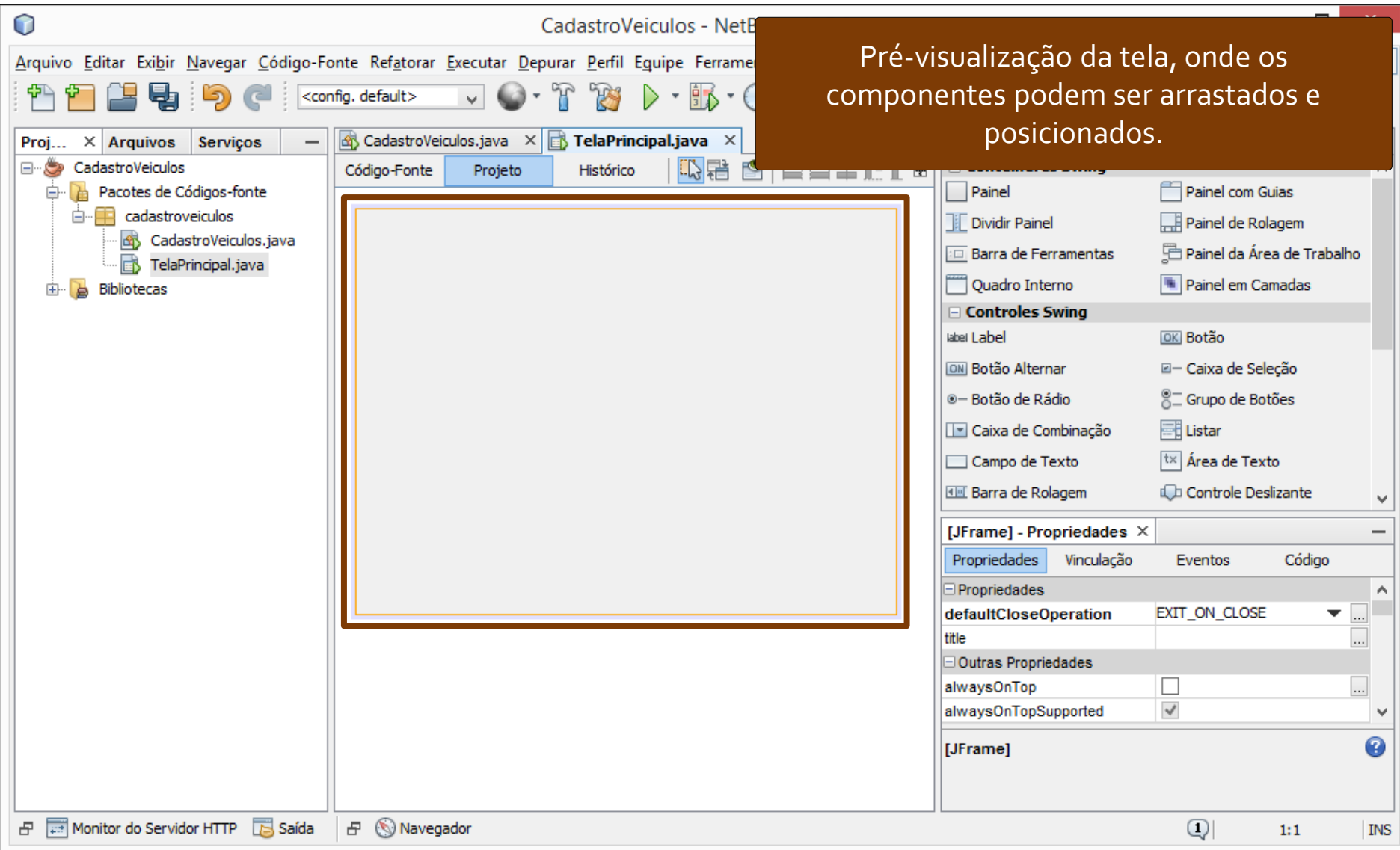


# Construção usando o NetBeans

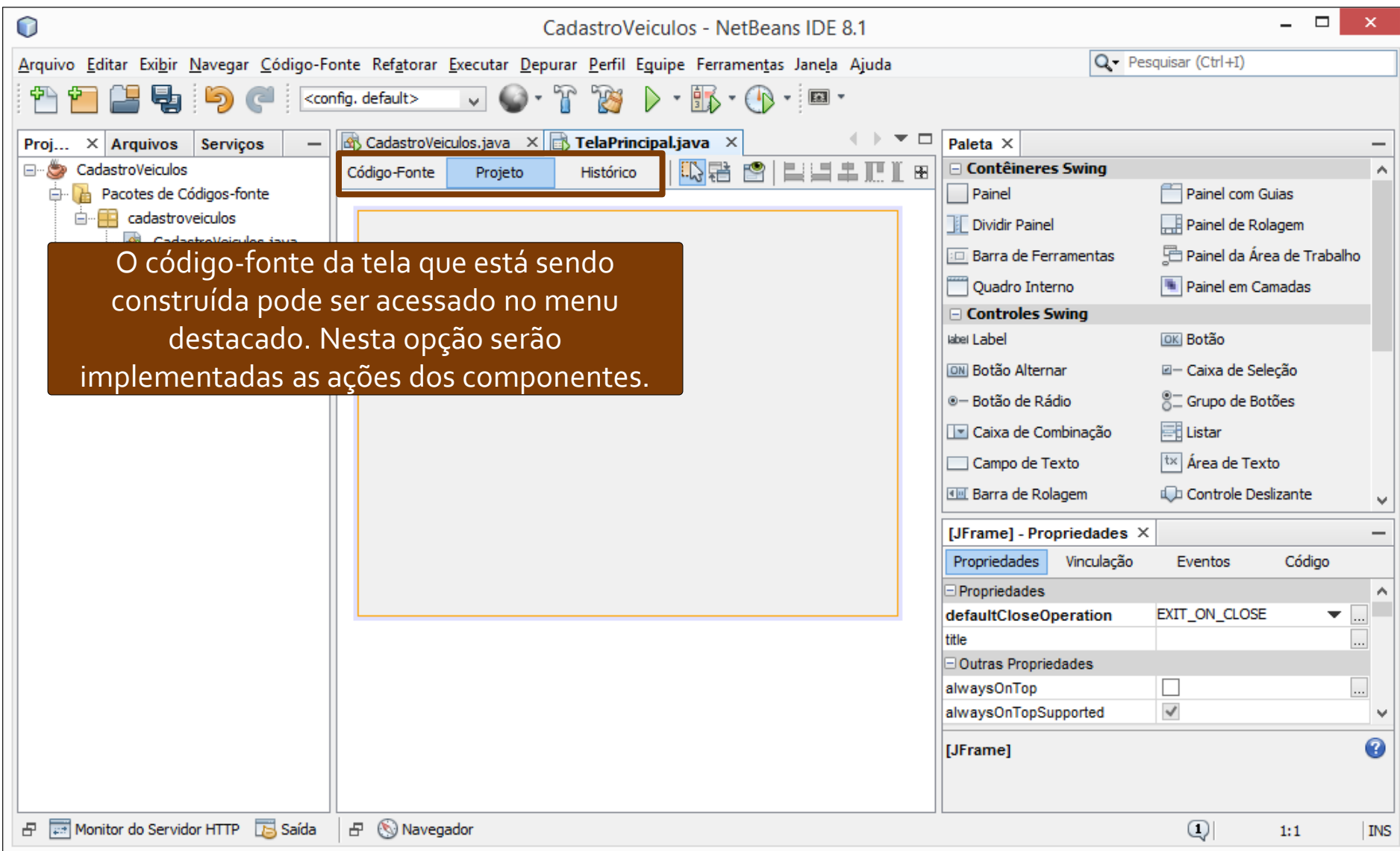


# Construção usando o NetBeans

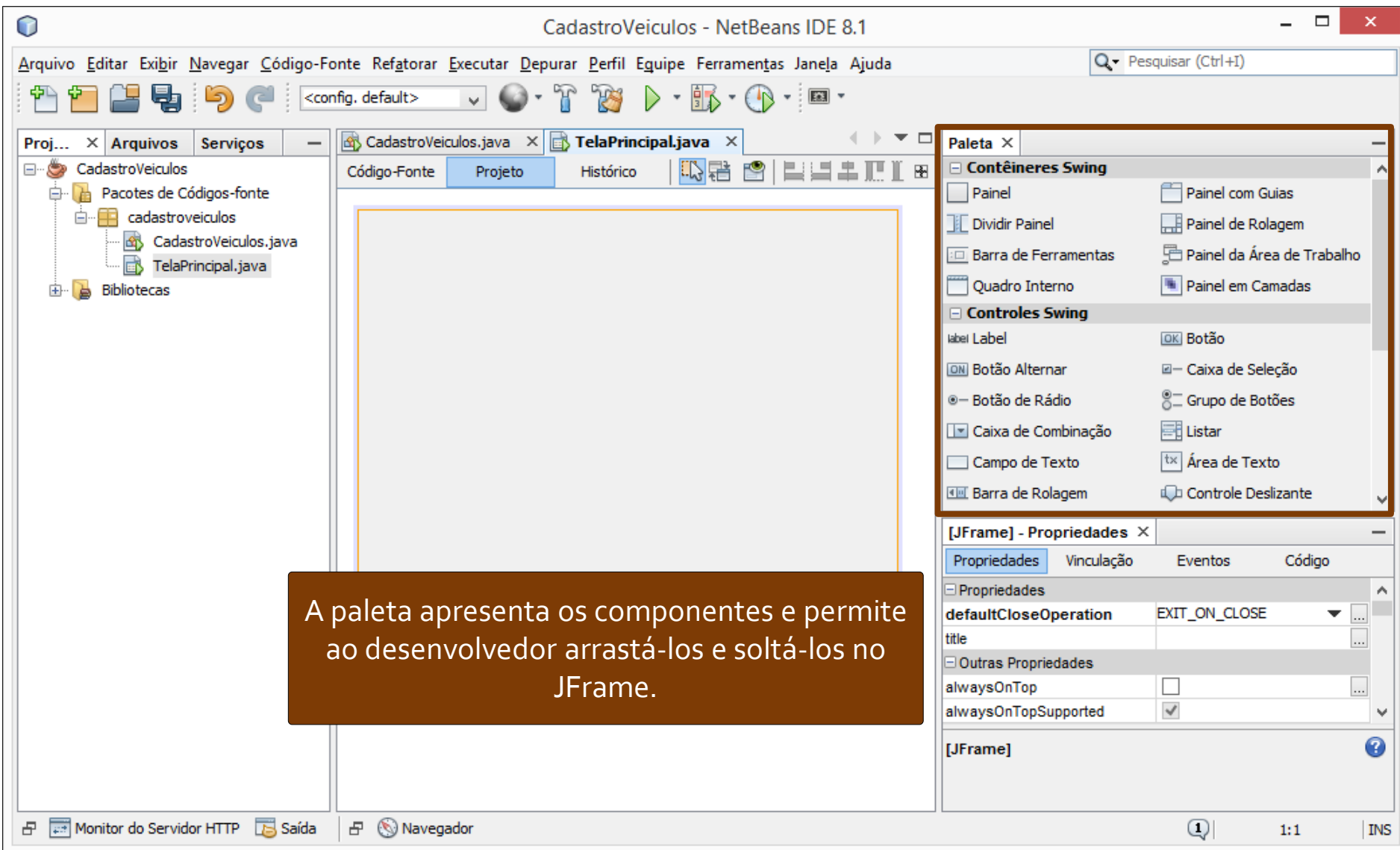
Pré-visualização da tela, onde os componentes podem ser arrastados e posicionados.



# Construção usando o NetBeans



# Construção usando o NetBeans





# Construção usando o NetBeans

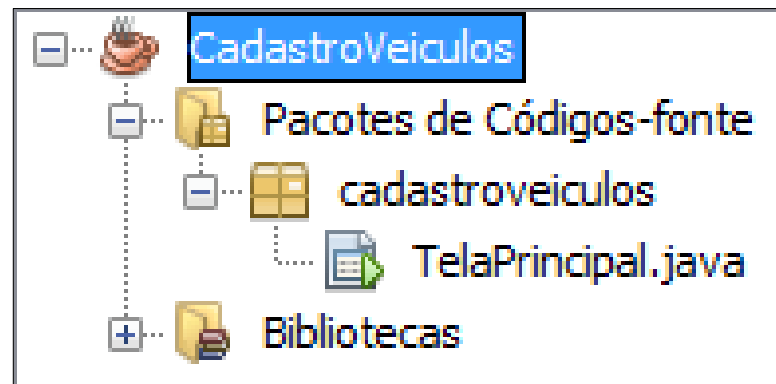
Para cada componente selecionado, existe uma série de propriedades que podem ser definidas na área destacada (ex: título do JFrame).

The screenshot shows the NetBeans IDE 8.1 interface. The project 'CadastroVeiculos' is open, and the 'TelaPrincipal.java' file is selected. The 'Projeto' tab is active, showing a visual representation of the component. The 'Paleta' (Palette) on the right lists various Swing components and controls. The 'Propriedades' (Properties) window for the selected component is also visible, showing properties like 'defaultCloseOperation', 'title', 'alwaysOnTop', and 'alwaysOnTopSupported'.

[JFrame] - Propriedades	
Propriedades	Vinculação
defaultCloseOperation	EXIT_ON_CLOSE
title	
Outras Propriedades	
alwaysOnTop	<input type="checkbox"/>
alwaysOnTopSupported	<input checked="" type="checkbox"/>

# Construção usando o NetBeans

- O **JFrame** consiste, portanto, em um componente swing que produz uma janela, através da qual o usuário interage com o sistema.
- Ele é criado com um método **main**, permitindo que a aplicação inicie por ele. O ideal é que uma classe de controle seja responsável por iniciar o sistema e criar as suas telas. Porém, por questões de simplicidade, utilizaremos a **TelaPrincipal** como ponto de partida da aplicação. Logo, a classe principal criada no projeto pode ser excluída.



# Construção usando o NetBeans

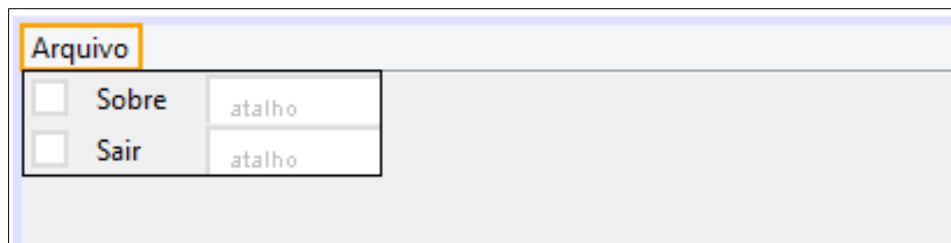
- Mude para a opção de código-fonte e veja o código Java gerado.

```
public class TelaPrincipal extends javax.swing.JFrame {  
  
    public TelaPrincipal() {  
        initComponents();  
    }  
  
    @SuppressWarnings("unchecked")  
    //Generated code  
  
    public static void main(String args[]) {  
        java.awt.EventQueue.invokeLater(new Runnable() {  
            public void run() {  
                new TelaPrincipal().setVisible(true);  
            }  
        });  
    }  
}
```

A classe estende JFrame. O método main cria a janela e a torna visível. No construtor, todos os componentes inseridos na janela são inicializados. Nesta classe podemos definir nossas variáveis, como a lista de veículos e os objetos necessários à sua manipulação.

# Componentes – menu

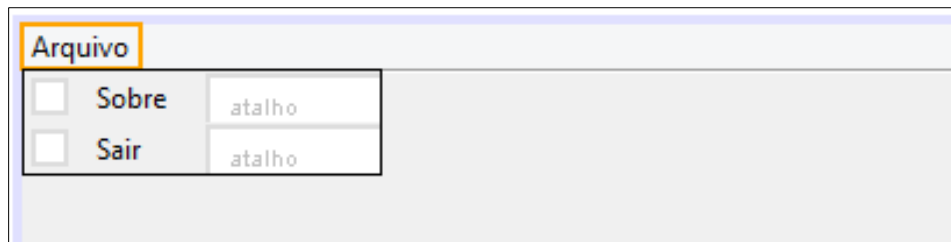
- Utilizando a visão de **Projeto**, na categoria **Menus Swing** da **paleta**, arraste o componente **Barra de Menu** para a janela. Será criado um menu com as opções **File** e **Edit**.
- Clicando com o botão direito sobre estes itens é possível:
  - Editar o texto que é apresentado em tela.
  - Alterar o nome da variável (objeto) do componente.).
  - Adicionar itens a esta opção do menu (**Adicionar da Paleta > Item de Menu**).
  - Para cada item adicionado, é possível realizar as mesmas ações.
- Crie a seguinte configuração do menu:



# Componentes – menu

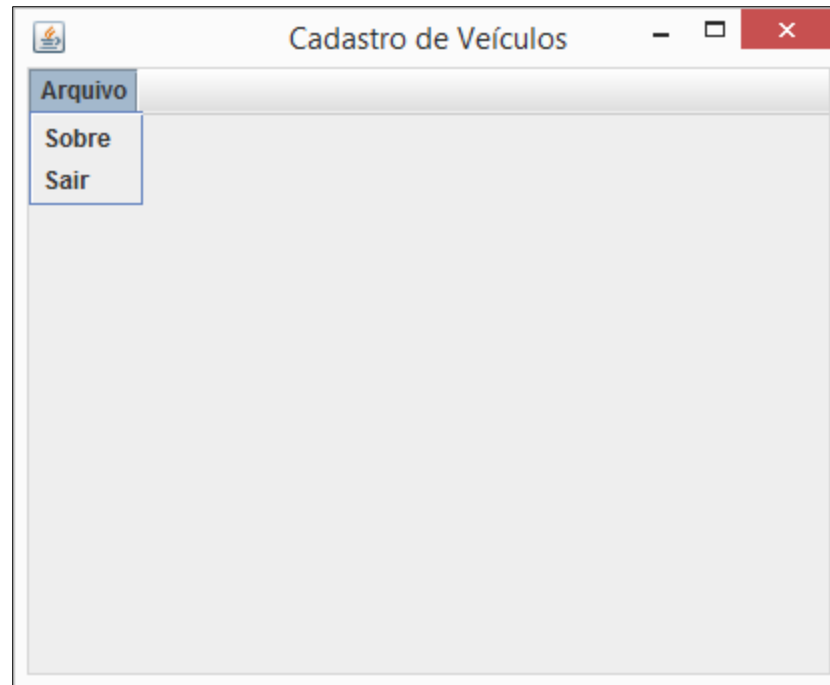
- Utilizando a visão de **Projeto**, na categoria **Menus Swing** da **paleta**, arraste o componente **Barra de Menu** para a janela. Será criado um menu com as opções **File** e **Edit**.
- Clicando com o botão direito sobre estes itens é possível:
  - Editar o texto que é apresentado em tela.
  - Alterar o nome da variável (objeto) do componente.).
  - Adic
  - Para
- Crie a se

É importante definirmos nomes sugestivos a cada objeto que será manipulado, de forma a dar legibilidade ao código.  
Exemplos: menuArquivo, menuSobre e menuSair.



# Componentes – menu

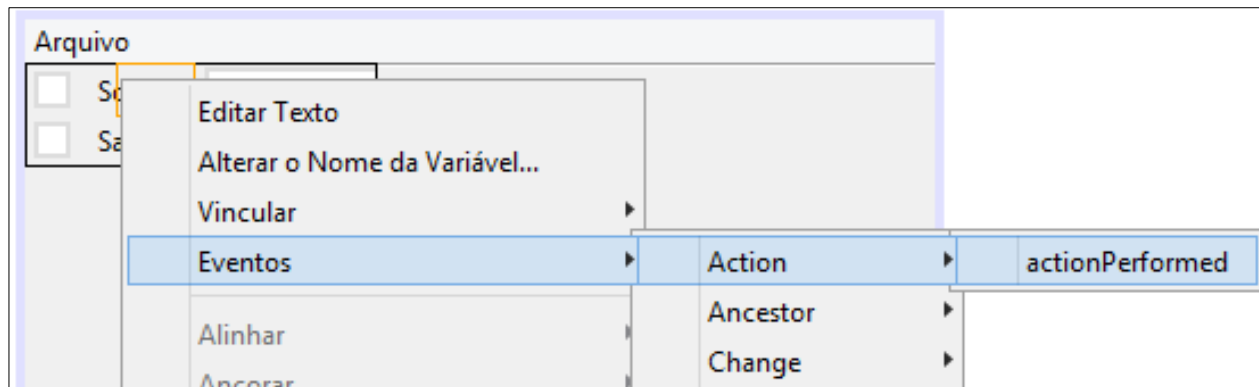
- Execute a aplicação e veja o resultado



- Observe que a janela tem como título "Cadastro de Veículos". Isso pode ser definido na propriedade **title** com a janela selecionada.

# Componentes – menu

- Nosso objetivo é apresentar uma janela com as informações de desenvolvimento da aplicação quando o usuário seleciona a opção **Sobre**.
- Temos que definir o evento de seleção desta opção, que consiste em um método que é chamado quando o usuário clica no respectivo item.
- Para isso, clique com o botão direito no item **Sobre** > **Eventos** > **Action** > **actionPerformed**.



# Componentes – menu

- Com isso, o NetBeans gera um método que é invocado sempre que o item de menu for selecionado pelo usuário.
- Nele podemos implementar o comportamento desejado. Neste caso, usaremos o componente **JOptionPane** para criar uma caixa de diálogo com as informações da aplicação.

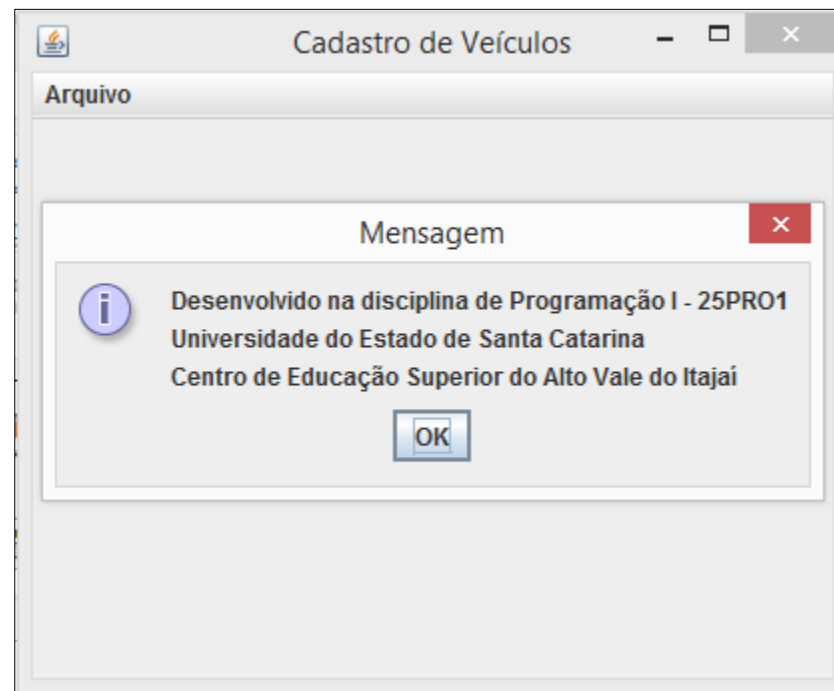
```
private void menuSobreActionPerformed(java.awt.event.ActionEvent evt) {  
    JOptionPane.showMessageDialog(this,  
        "Desenvolvido na disciplina de Programação I - 25PR01\n"  
        + "Universidade do Estado de Santa Catarina\n"  
        + "Centro de Educação Superior do Alto Vale do Itajaí");  
}
```

- Repare que a caixa de diálogo é um componente filho da janela, pois a definimos no primeiro argumento do método **showMessageDialog**.



# Componentes – menu

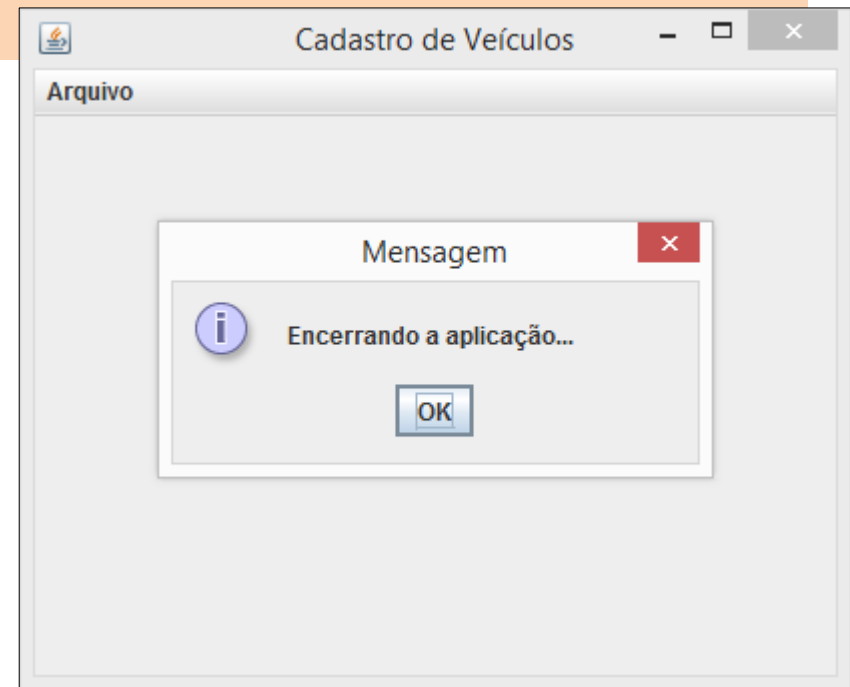
- Como resultado, ao clicar no item **Sobre**, a caixa de diálogo aparece centralizada em relação à janela da aplicação, mostrando as informações previamente definidas.



# Componentes – menu

- Faremos o mesmo processo para o item Sair, cuja tarefa é fechar a janela e, conseqüentemente, finalizar a aplicação.
- Fazemos isso com o comando **dispose()**;

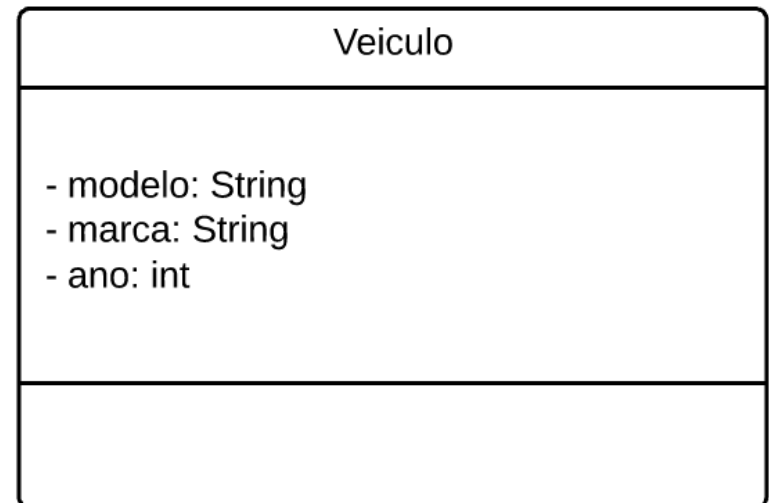
```
private void menuSairActionPerformed(java.awt.event.ActionEvent evt) {  
    JOptionPane.showMessageDialog(this, "Encerrando a aplicação...");  
    dispose();  
}
```



# Entidade Veiculo

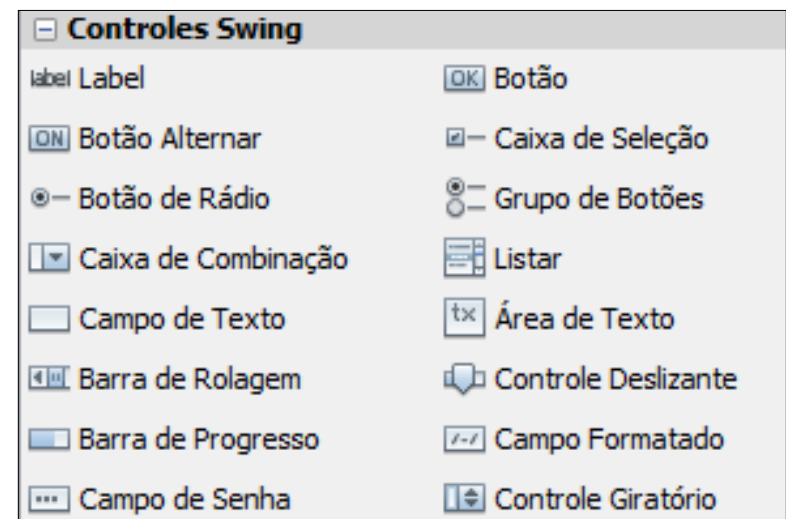
- A classe veículo possui os atributos **modelo**, **marca** e **ano**, bem como seus métodos acessores.

```
public class Veiculo {  
  
    private String modelo;  
    private String marca;  
    private int ano;  
  
    //Métodos acessores  
  
}
```



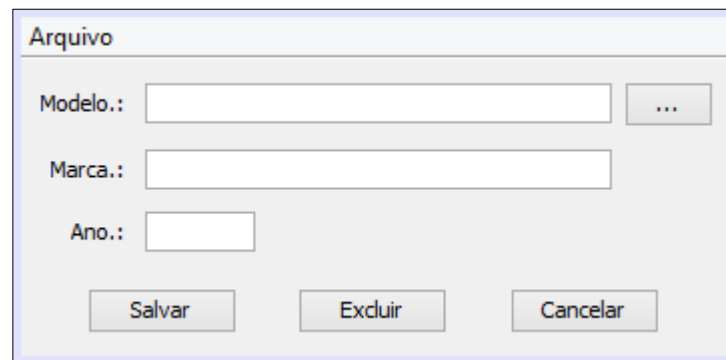
# Componentes – formulários

- A tela principal deve apresentar um formulário através do qual o usuário poderá cadastrar, consultar, alterar e excluir veículos.
- Os componentes necessários para isso são:
  - **Label (JLabel):** rótulo ou saída de texto na tela.
  - **TextField (JTextField):** campo de entrada de texto.
  - **Button (JButton):** botão de ação.



# Componentes – formulários

- Posicione cada elemento e defina os textos de exibição e nomes sugestivos aos objetos.
- **Resultado esperado:**



Arquivo

Modelo.:  ...

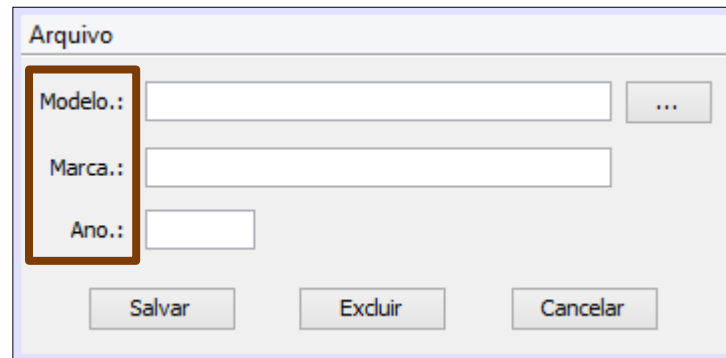
Marca.:

Ano.:

Salvar Excluir Cancelar

# Componentes – formulários

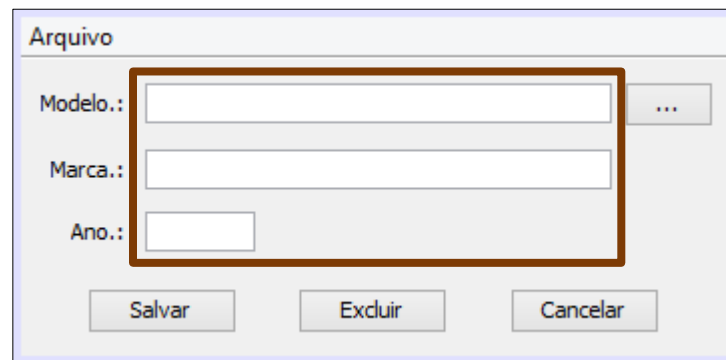
- Posicione cada elemento e defina os textos de exibição e nomes sugestivos aos objetos.
- **Resultado esperado:**



Campos de saída de texto.

# Componentes – formulários

- Posicione cada elemento e defina os textos de exibição e nomes sugestivos aos objetos.
- **Resultado esperado:**



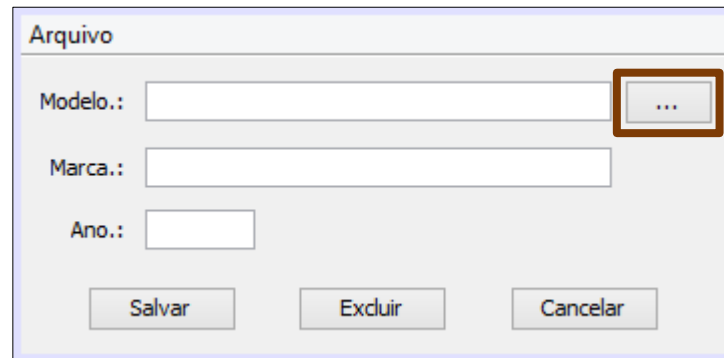
The image shows a dialog box titled "Arquivo". It contains three text input fields labeled "Modelo:", "Marca:", and "Ano:". The "Modelo:" and "Marca:" fields are grouped together by a brown rectangular border. To the right of the "Modelo:" field is a button with three dots "...". At the bottom of the dialog are three buttons: "Salvar", "Excluir", and "Cancelar".

Campos de entrada de texto: **edModelo**,  
**edMarca** e **edAno**.

# Componentes – formulários

- Posicione cada elemento e defina os textos de exibição e nomes sugestivos aos objetos.
- **Resultado esperado:**

Botão para pesquisar nos veículos cadastrados pelo modelo informado pelo usuário (**btPesquisar**).



The screenshot shows a form titled "Arquivo" with three input fields: "Modelo.:", "Marca.:", and "Ano.:". Below these fields are three buttons: "Salvar", "Excluir", and "Cancelar". A small button with three dots (...) is located to the right of the "Modelo.:" field, and it is highlighted with a red rectangle.



# Funcionalidades

- Para implementar as funcionalidades a classe **TelaPrincipal** deverá manter uma lista de veículos (onde os registros serão armazenados) e um objeto da classe Veiculo (que representará o registro manipulado pelo usuário).

```
public class TelaPrincipal extends javax.swing.JFrame {  
  
    private List<Veiculo> listaVeiculos = new ArrayList<Veiculo>();  
    private Veiculo veiculo = new Veiculo();  
  
    //...  
}
```

# Funcionalidades – salvar

- Da mesma forma como é feito com os menus, é possível definir métodos que serão executados quando os botões forem clicados. O processo é o mesmo, definindo um evento do tipo **actionPerformed**.
- O botão **Salvar** serve para gravar na lista um novo registro informado pelo usuário, ou gravar as alterações feitas pelo usuário em um objeto previamente recuperado da lista.
- Logo, os passos que devem ser implementados são:
  - Recuperar os valores digitados pelo usuário no formulário.
  - Atualizar os atributos do objeto **veiculo** com os valores recuperados.
  - Se o objeto não se encontra na lista, é um novo registro e deve ser incluído na mesma. Caso contrário, trata-se de uma alteração e a atualização do objeto já é suficiente.
  - Instanciar um novo objeto em **veiculo**, permitindo o cadastro de um novo registro.
  - Limpar os campos do formulário para que o usuário possa cadastrar um novo registro.

# Funcionalidades – salvar

- Código resultante:

```
private void btSalvarActionPerformed(java.awt.event.ActionEvent evt) {  
    String modelo = edModelo.getText();  
    String marca = edMarca.getText();  
    int ano = Integer.parseInt(edAno.getText());  
  
    veiculo.setModelo(modelo);  
    veiculo.setMarca(marca);  
    veiculo.setAno(ano);  
  
    if(!listaVeiculos.contains(veiculo)) {  
        listaVeiculos.add(veiculo);  
    }  
    veiculo = new Veiculo();  
  
    edModelo.setText("");  
    edMarca.setText("");  
    edAno.setText("");  
}
```

# Funcionalidades – salvar

- Código resultante:

```
private void btSalvarActionPerformed(java.awt.event.ActionEvent evt) {  
    String modelo = edModelo.getText();  
    String marca = edMarca.getText();  
    int ano = Integer.parseInt(edAno.getText());  
  
    veiculo.setModelo(modelo);  
    veiculo.setMarca(marca);  
    veiculo.setAno(ano);  
  
    if(!listaVeiculos.contains(veiculo)) {  
        listaVeiculos.add(veiculo);  
    }  
    veiculo = new Veiculo();  
  
    edModelo.setText("");  
    edMarca.setText("");  
    edAno.setText("");  
}
```

Recuperar os valores digitados pelo usuário no formulário.

# Funcionalidades – salvar

- Código resultante:

```
private void btSalvarActionPerformed(java.awt.event.ActionEvent evt) {  
    String modelo = edModelo.getText();  
    String marca = edMarca.getText();  
    int ano = Integer.parseInt(edAno.getText());  
  
    veiculo.setModelo(modelo);  
    veiculo.setMarca(marca);  
    veiculo.setAno(ano);  
  
    if(!listaVeiculos.contains(veiculo)) {  
        listaVeiculos.add(veiculo);  
    }  
    veiculo = new Veiculo();  
  
    edModelo.setText("");  
    edMarca.setText("");  
    edAno.setText("");  
}
```

Atualizar os atributos do objeto **veiculo** com os valores recuperados.

# Funcionalidades – salvar

- Código resultante:

```
private void btSalvarActionPerformed(java.awt.event.ActionEvent evt) {  
    String modelo = edModelo.getText();  
    String marca = edMarca.getText();  
    int ano = Integer.parseInt(edAno.getText());  
  
    veiculo.setModelo(modelo);  
    veiculo.setMarca(marca);  
    veiculo.setAno(ano);  
  
    if(!listaVeiculos.contains(veiculo)) {  
        listaVeiculos.add(veiculo);  
    }  
    veiculo = new Veiculo();  
  
    edModelo.setText("");  
    edMarca.setText("");  
    edAno.setText("");  
}
```

Se o objeto não se encontra na lista, é um novo registro e deve ser incluído na mesma. Caso contrário, trata-se de uma alteração e a atualização do objeto já é suficiente.

# Funcionalidades – salvar

- Código resultante:

```
private void btSalvarActionPerformed(java.awt.event.ActionEvent evt) {  
    String modelo = edModelo.getText();  
    String marca = edMarca.getText();  
    int ano = Integer.parseInt(edAno.getText());  
  
    veiculo.setModelo(modelo);  
    veiculo.setMarca(marca);  
    veiculo.setAno(ano);  
  
    if(!listaVeiculos.contains(veiculo)) {  
        listaVeiculos.add(veiculo);  
    }  
    veiculo = new Veiculo();  
  
    edModelo.setText("");  
    edMarca.setText("");  
    edAno.setText("");  
}
```

Instanciar um novo objeto em **veiculo**, permitindo o cadastro de um novo registro.

# Funcionalidades – salvar

- Código resultante:

```
private void btSalvarActionPerformed(java.awt.event.ActionEvent evt) {  
    String modelo = edModelo.getText();  
    String marca = edMarca.getText();  
    int ano = Integer.parseInt(edAno.getText());  
  
    veiculo.setModelo(modelo);  
    veiculo.setMarca(marca);  
    veiculo.setAno(ano);  
  
    if(!listaVeiculos.contains(veiculo)) {  
        listaVeiculos.add(veiculo);  
    }  
    veiculo = new Veiculo();  
  
    edModelo.setText("");  
    edMarca.setText("");  
    edAno.setText("");  
}
```

Limpar os campos do formulário para que o usuário possa cadastrar um novo registro.



# Funcionalidades – cancelar

- O botão **Cancelar** deve limpar todos os campos. Caso os valores nos campos sejam referentes a um objeto pesquisado pelo usuário, a referência armazenada em **veiculo** deve ser removida e uma nova instância atribuída ao objeto.

```
private void btCancelarActionPerformed(java.awt.event.ActionEvent evt) {  
    edModelo.setText("");  
    edMarca.setText("");  
    edAno.setText("");  
    veiculo = new Veiculo();  
}
```

# Funcionalidades – excluir

- O botão **Excluir** remove da lista de veículos um registro pesquisado anteriormente. Como o registro pesquisado se encontra armazenado no objeto **veiculo**, basta removê-lo da lista, atribuir uma nova instância a ele e limpar os campos da tela.

```
private void btExcluirActionPerformed(java.awt.event.ActionEvent evt) {  
    listaVeiculos.remove(veiculo);  
    this.veiculo = new Veiculo();  
  
    edModelo.setText("");  
    edMarca.setText("");  
    edAno.setText("");  
}
```

# Funcionalidades – pesquisar

- O botão **Pesquisar** permite recuperar um objeto da lista de veículos e preencher os campos do formulário com os valores dos seus atributos. Essa pesquisa é feita com base no modelo informado pelo usuário, por isso o botão se localiza ao lado deste campo.
- Caso o veículo encontrado, sua referência deve ser armazenada no objeto **veiculo** e os campos do formulário preenchidos.
- Com isso, caso o usuário modifique os valores dos campos e clique em **Salvar**, o objeto da lista de veículos será atualizado, ao invés de incluído um novo objeto. Caso o usuário clique em **Excluir**, o objeto será removido da lista. Caso o usuário clique em **Cancelar**, nada é realizado e os campos e o objeto são “reiniciados”.

# Funcionalidades – pesquisar

- Código resultante:

```
private void btPesquisarActionPerformed(java.awt.event.ActionEvent evt) {  
    String modelo = edModelo.getText();  
  
    for(Veiculo v : listaVeiculos) {  
        if(v.getModelo().equals(modelo)) {  
            veiculo = v;  
            edModelo.setText(veiculo.getModelo());  
            edMarca.setText(veiculo.getMarca());  
            edAno.setText(String.valueOf(veiculo.getAno()));  
            break;  
        }  
    }  
}
```

# Funcionalidades – botão de exclusão

- Uma característica importante do botão **Excluir** é o fato de ele não fazer sentido quando o usuário está cadastrando um novo veículo. Sua funcionalidade só pode ser executada quando o usuário carregar um veículo previamente cadastrado.
- Para evitar problemas, podemos **desabilitar** o botão quando o mesmo não deve ser clicado, impedindo que o usuário execute o método vinculado a ele e deixando claro sua impossibilidade de uso.
- Isso é feito modificando a propriedade **enabled** através do método **setEnabled(true)** ou **setEnabled(false)**, habilitando e desabilitando o componente.

```
btExcluir.setEnabled(true);  
btExcluir.setEnabled(false);
```

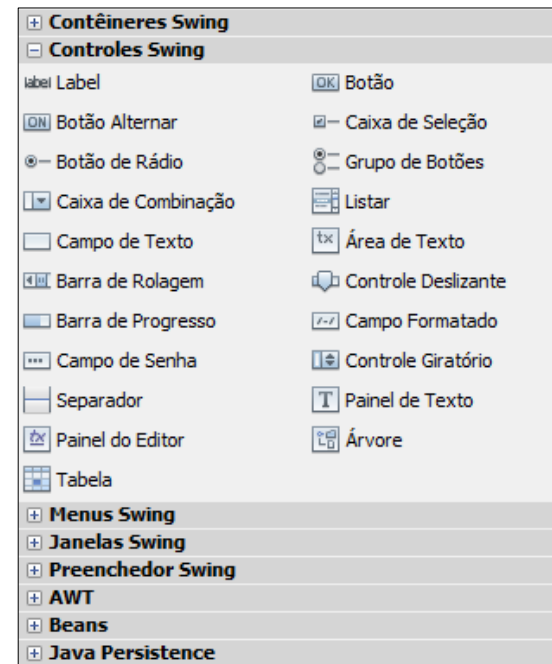
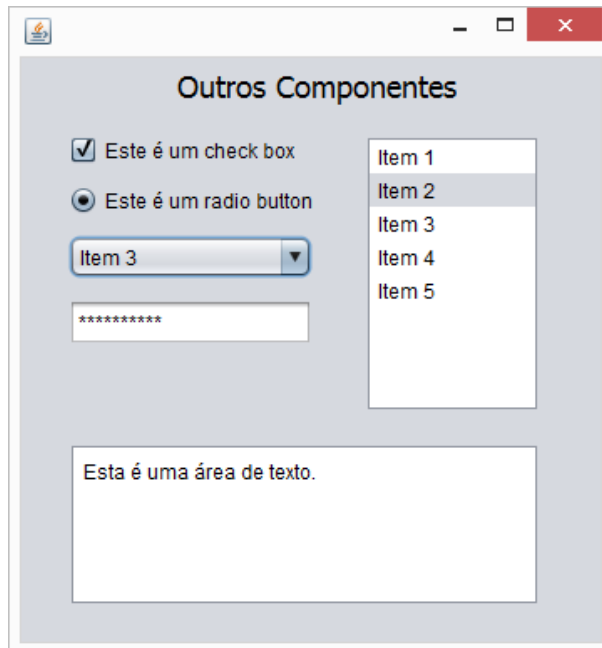
# Funcionalidades – botão de exclusão

- Logo, o botão deve ser **habilitado** nos seguintes casos:
  - Quando a ação de **pesquisa** encontra o objeto e o carrega na tela, permitindo ao usuário excluir o veículo, se desejado.
- O botão deve ser **desabilitado** nos seguintes casos:
  - Quando a ação de **salvar** é realizada, pois pode se tratar de um objeto carregado da lista, com suas alterações sendo salvas.
  - Quando a ação de **cancelar** é realizada, pois pode se tratar de um objeto carregado da lista, onde o usuário está cancelando sua alteração.
  - Quando a ação de **exclusão** é realizada, pois ao concluir a exclusão de um registro, uma nova instância é definida e não é possível excluir novamente.
  - Quando a **aplicação inicia**, pois o botão deve estar desabilitado por padrão. Isso pode ser feito no construtor da classe, após a inicialização dos seus componentes (abaixo).

```
public TelaPrincipal() {  
    initComponents();  
    btExcluir.setEnabled(false);  
}
```

# Outros componentes

- Existem muitos outros componentes swing para a construção de interfaces gráficas interativas.



- Consulte a documentação dos componentes que deseja utilizar em <http://docs.oracle.com/javase/8/docs/api/>.

# Referências

DEITEL, H. M. **Java: como programar**. H. M Deitel e P. J. Deitel - 8a ed. Porto Alegre: Prentice-Hall, 2010.

## Leitura complementar

TutorialsPoint Java (<http://www.tutorialspoint.com/java>).