Desenvolvimento de Sistemas

Interface *desktop*: construção de interface de usuário, manipulação de eventos, uso de controles, manipulação de janelas, construção de formulários e listagens (Parte 1)

Introdução

Comumente, quando se começa a programar, os primeiros *softwares* que são criados têm foco no console em formato de texto. Essa é uma ótima forma de se aprender a programar inicialmente, pois o foco está em conhecer a linguagem e desenvolver a lógica de programação. Mas, quando se precisa desenvolver um sistema que será usado por um usuário que não tem conhecimentos técnicos, é preciso apresentar uma interação mais apropriada que o console. É aí que são utilizadas as interfaces gráficas para criar elementos visuais que permitirão ao usuário interagir com o sistema. No decorrer desse conteúdo, você aprenderá como construir interfaces gráficas para *desktop*. Para isso, serão utilizados componentes de um *kit* de ferramentas chamado Swing, por meio do Apache NetBeans IDE.

Construção de interface de usuário

Falar de “interfaces gráficas” significa falar de uma forma visual de interagir com um sistema. Em programação, essas interfaces são chamadas de **GUI** (***g****raphical* ***u****ser* ***i****nterface* ou “interface gráfica do usuário”, em português).

O Apache NetBeans IDE contém um construtor de interface gráfica integrado chamado GUI Builder, que permite construir GUIs em projetos Java apenas arrastando e soltando elementos na tela. Para isso, o GUI Builder utiliza dois *toolkits* (*kits* de

ferramentas) com um rico conjunto de classes prontas: o AWT e o Swing.

AWT (abstract windowing toolkit)

Foi introduzido no JDK (Java Development Kit) 1.0. Sua proposta é utilizar a ambientação gráfica do sistema operacional que está sendo rodado no programa para renderizar elementos visuais que tenham a aparência idêntica, passando a sensação de "*software* nativo". Na implementação, o AWT usa as APIs (*application programming interfaces*, ou "interfaces de programação de aplicação", em português) de componentes visuais do sistema operacional, nas quais a JVM (Máquina Virtual Java) está rodando para criar componentes do sistema operacional. Por se tratar da primeira solução para construção de interfaces gráficas no Java, seu uso é extremamente limitado: a maioria dos seus componentes tornou-se obsoleta e eles devem ser substituídos por componentes de GUI mais atuais.

Swing

O Swing é um conjunto muito mais abrangente de bibliotecas gráficas que aprimora o AWT, trazendo componentes mais leves e flexíveis. Foi introduzido como parte do Java Foundation Classes (JFC) após o lançamento do JDK 1.1. Diferentemente do AWT, o Swing tem componentes que são independentes de plataforma, o que permite padronizar a exibição das interfaces em diferentes sistemas operacionais. Essa tecnologia é mais completa que o AWT, e por isso você aprenderá a construir interfaces gráficas para *desktop* utilizando-a.

Criando um projeto Java com interface gráfica para *desktop*

Para começar, você criará um novo projeto Java no Apache NetBeans IDE. Esse projeto será usado para testar e praticar os exemplos deste conteúdo. Logo, você poderá nomeá-lo como desejar.

1. Clique em **File** > **New Project**. Como alternativa, você pode clicar no ícone de **Novo Projeto**, localizado na barra de ferramentas do NetBeans IDE.
2. Selecione **Java with Ant** > **Java Application.e** e clique em **Next**.
3. Digite o nome do projeto no campo **Project Name** e clique em **Finish**.

Agora que o seu projeto foi criado, desenvolva sua primeira tela:

1. Clique com o botão direito do *mouse* sobre o pacote do projeto e selecione as opções **New** > **JFrame Form**.
2. Na nova janela que será aberta, especifique o nome da classe (**Class Name**). Para manter uma boa organização no projeto, o ideal é criar as classes com um nome referente ao conteúdo da tela. Depois de informar o nome da classe, clique em **Finish**.

Após concluir a criação da classe, uma nova tela será aberta no NetBeans IDE, na qual você será apresentado ao GUI Builder.

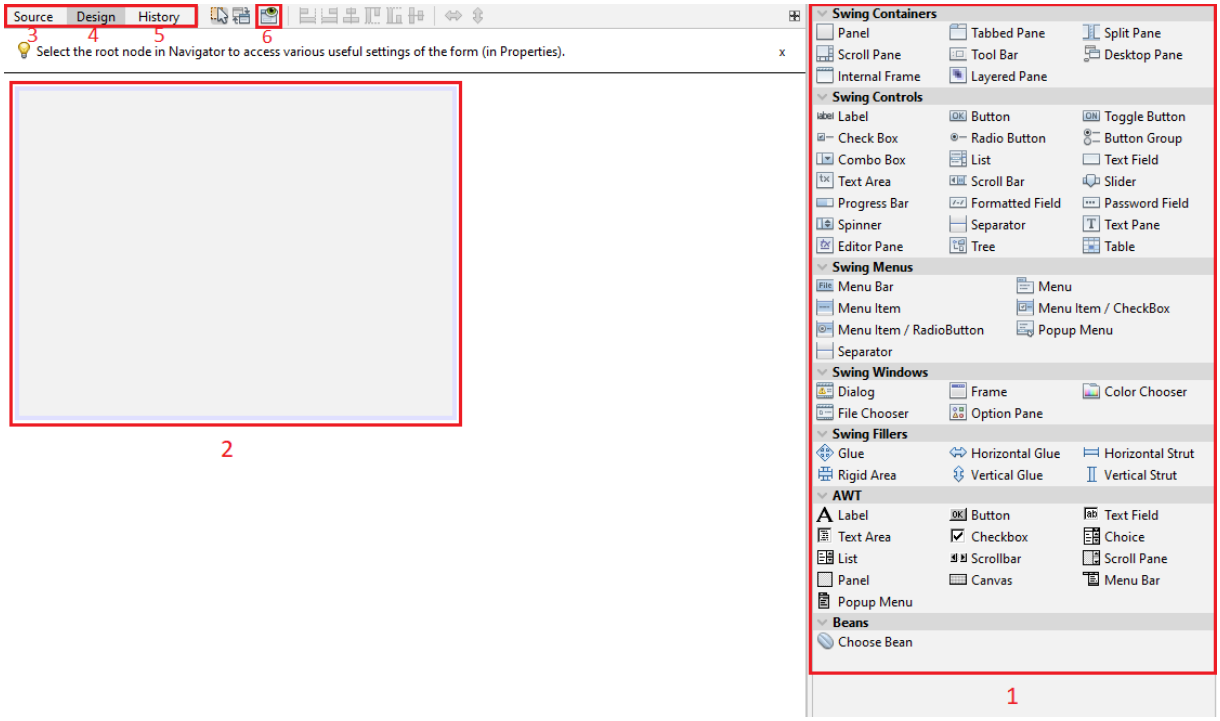

Figura 1 – Tela do GUI Builder

Fonte: Apache NetBeans IDE (2022)

Nessa tela, você pode identificar os seguintes recursos:

O GUI Builder do Apache NetBeans IDE consegue simplificar o fluxo de trabalho de criação de interfaces gráficas, liberando os desenvolvedores das complexidades dos gerenciadores de *layout* do Swing. Você pode criar seus formulários simplesmente

colocando os componentes onde quiser. O GUI Builder descobre quais atributos de *layout* são necessários e gera o código para você automaticamente – o código pode ser acessado na aba de código-fonte **Source code**.

Para desenvolver uma interface amigável para o usuário, são necessários diversos elementos, como caixas de texto, botões, locais para entrada de dados e muito mais. Esses elementos que compõem a interface gráfica do usuário são chamados de componentes.

O GUI Builder do Apache NetBeans IDE traz várias opções de componentes para você construir a interface gráfica da sua aplicação. Tudo o que precisa fazer é selecionar o componente e arrastar para a tela em branco. Ao fazer isso, um código Java será escrito na aba do **código-fonte** para o componente que você escolher ser criado na tela com todas as suas respectivas configurações. Você pode usar a aba de código-fonte para editar valores e chamar métodos para configurar cada componente. Outra opção é você clicar com o botão direito sobre um componente no GUI Builder e acessar a opção **Properties** para configurar os atributos e métodos do componente por meio de uma interface visual.

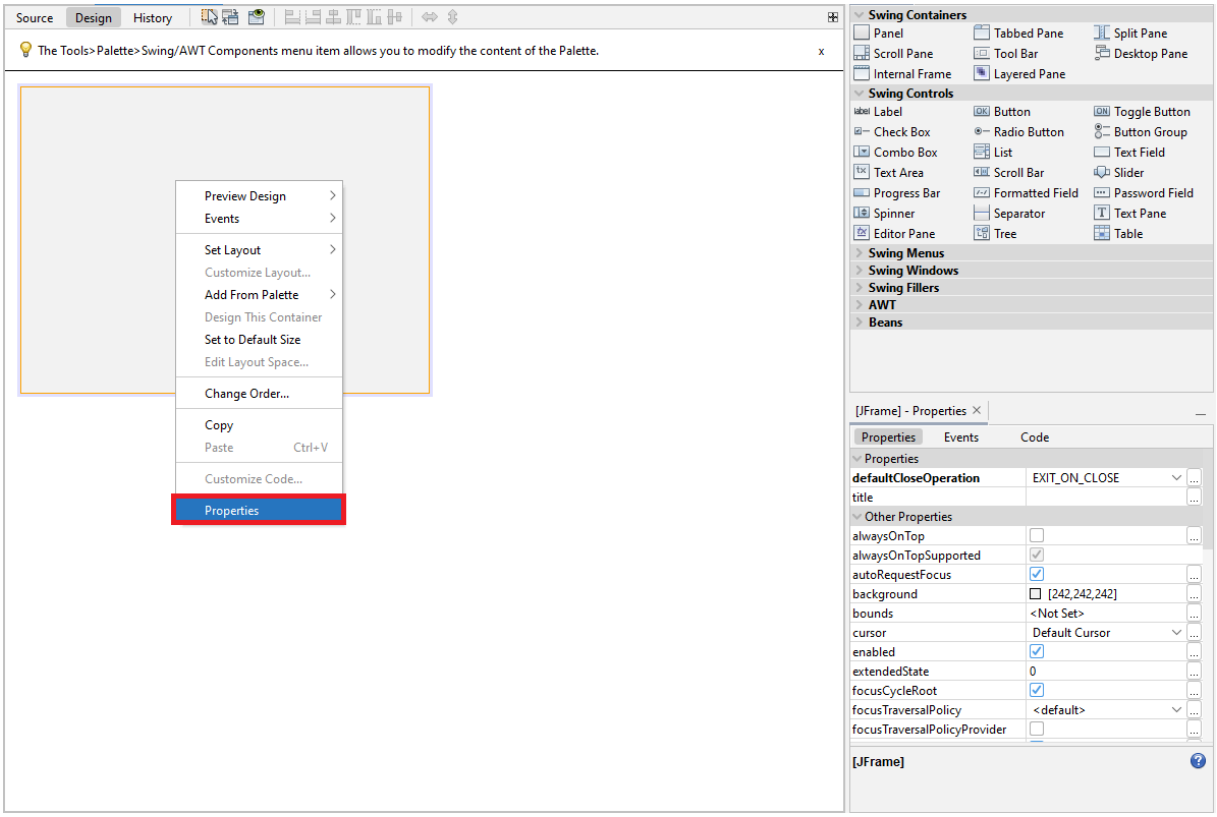

Figura 2 – Tela do GUI Builder
Fonte: Apache NetBeans IDE (2022)

Todos os componentes do Swing, como **JButton**, **JComboBox**, **JList**, **JLabel** são herdados da classe **JComponent**, que pode ser adicionada às classes contêineres. Os contêineres são as janelas ,como quadros e caixas de diálogo. Os componentes básicos do Swing são os blocos de construção de qualquer aplicativo de GUI. Métodos como **setLayout** substituem o *layout* padrão em cada contêiner. Contêineres como **JFrame** e **JDialog** só podem adicionar um componente a si mesmo.

Agora que você se familiarizou com a interface do construtor de GUI, confira alguns principais componentes disponíveis para construir interfaces gráficas.

Observação!

Apesar de o GUI Builder apresentar componentes do AWT, o uso desses componentes não é recomendado. Sendo assim, o foco deste estudo será o uso dos componentes do Swing.

Principais componentes do Java Swing

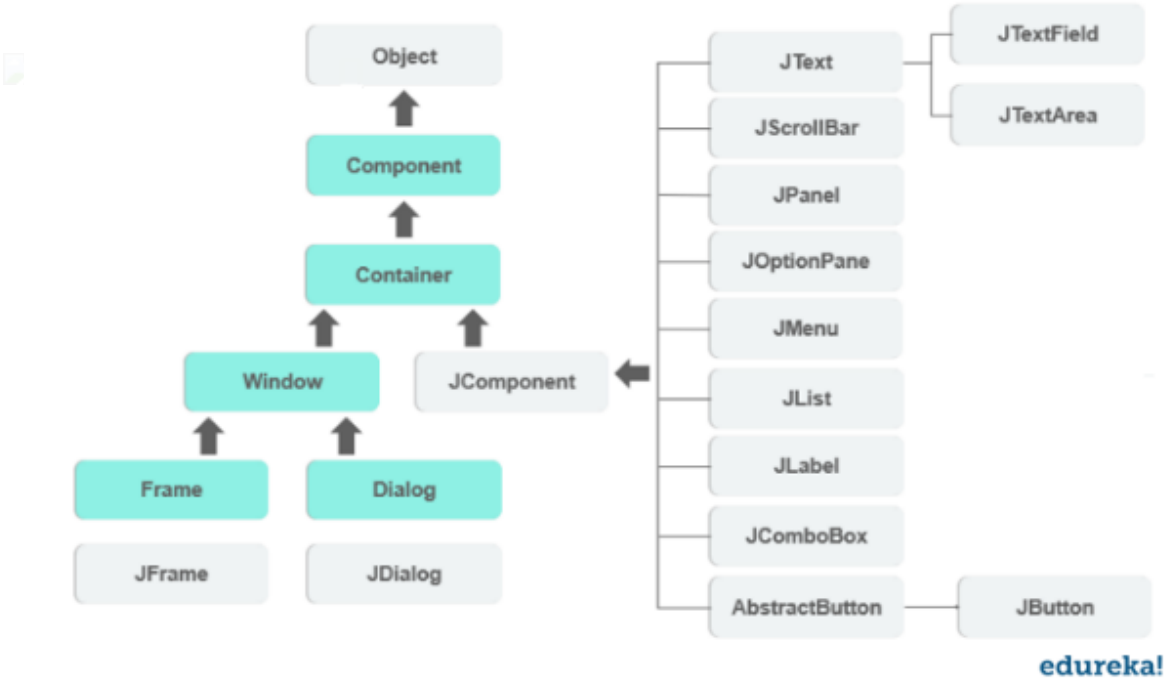

Figura 3 – Hierarquia de componentes do Java Swing
Fonte: Wassem (2021)

A imagem ilustra resumidamente a hierarquia de classes do Java Swing, deixando claro que todos os componentes são derivados direta ou indiretamente de **JComponent**. A seguir estão alguns componentes com exemplos para você entender como pode usá-los.

Frame

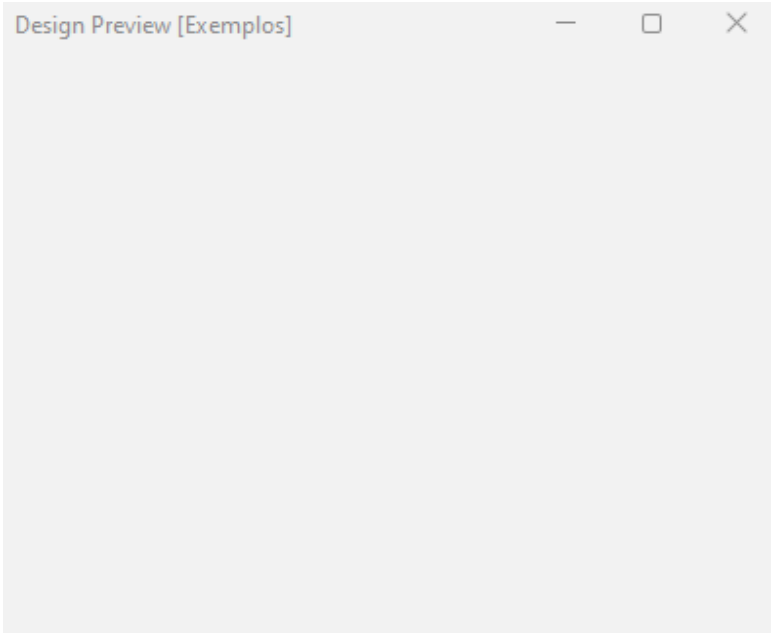

Figura 4 – Exemplo de **JFrame**
Fonte: Apache NetBeans IDE (2022)

Nome da classe: JFrame

Descrição: é a janela do programa na qual podem ser inseridos outros componentes, como botões de comando, ícones, textos etc.

Principais métodos:

pack(): compacta a janela para o tamanho dos componentes.

setSize(int, int): define a largura e altura da janela.

setLocation(int, int): define a posição em que a janela do programa ficará quando estiver rodando.

setBounds(int, int, int, int): define a posição e o tamanho da janela.

setVisible(boolean): é o método para habilitar (*true*) ou desabilitar (*false*) a exibição da janela.

setDefaultCloseOperation(): define o que ocorre quando o usuário tenta fechar a janela.

Panel

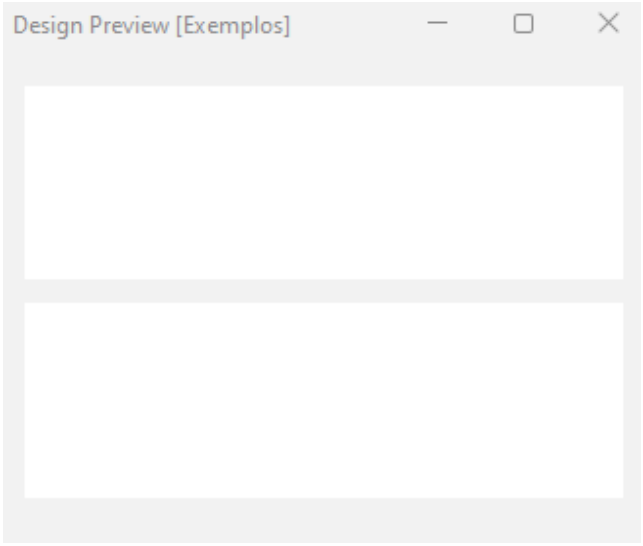

Figura 5 – Exemplo de **JPanel**
Fonte: Apache NetBeans IDE (2022)

Nome da classe: JPanel

Descrição: representa um contêiner básico para componentes a serem inseridos.

Principais métodos:

add(Component, int): adiciona um componente definindo a sua posição.

setLayout(LayoutManager): altera o tipo de *layout*.

Dicas de uso:

Antes de inserir qualquer componente no **JFrame**, recomenda-se começar com um **JPanel** para garantir o posicionamento correto dos componentes.

Caso você não utilize o **JPanel**, os componentes ficarão “soltos” e poderão se comportar de maneira inesperada quando o usuário aumentar ou diminuir a janela do programa.

Você pode alterar a cor do **JPanel** clicando com o botão direito sobre o componente e acessando a opção **Properties**. Nela, você terá o campo **Background**, que permite você escolher uma cor para o seu **JPanel**.

Label

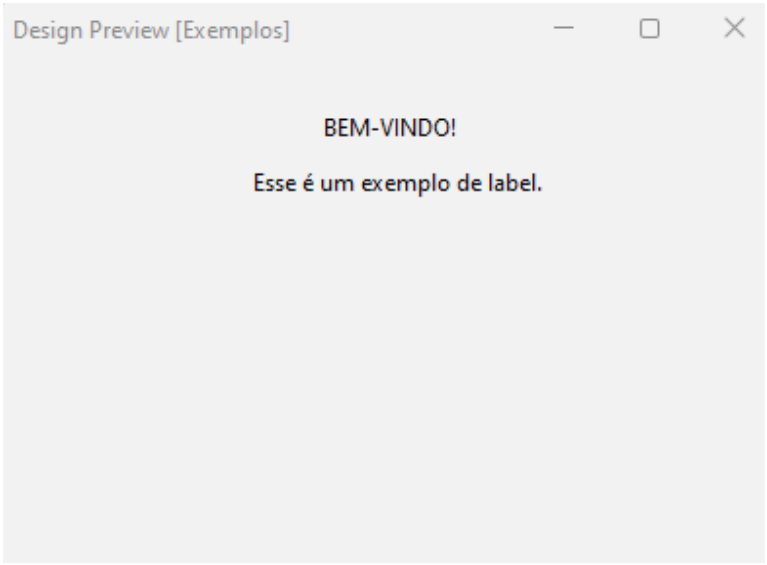

Figura 6 – Exemplo de **JLabel**
Fonte: Apache NetBeans IDE (2022)

Nome da classe: JLabel

Descrição: Permite exibir um texto simples não editável, podendo ser utilizada também para exibir ícones e imagens.

Principais métodos:

setText(String): altera o texto do componente.

getText(): retorna o texto do componente.

setIcon(IconImage icon): adiciona um ícone ao **JLabel** ao passar como parâmetro o caminho da imagem.

Dicas de uso:

Você pode alterar o texto das *labels* sem precisar ir até o código-fonte. Para isso, dê um duplo clique sobre a *label* inserida na tela e então digite a palavra que você deseja inserir.

A maneira mais simples de alterar a cor, a fonte e o tamanho do texto na *label* é por meio da janela de propriedades da *label*, que pode ser acessada clicando-se com o botão direito sobre o componente inserido e selecionando a opção **Properties**. Nessa tela, você terá acesso às propriedades **Foreground** (para alterar a cor da fonte) e **Font** (para alterar fonte, estilo e tamanho do texto).

Text field

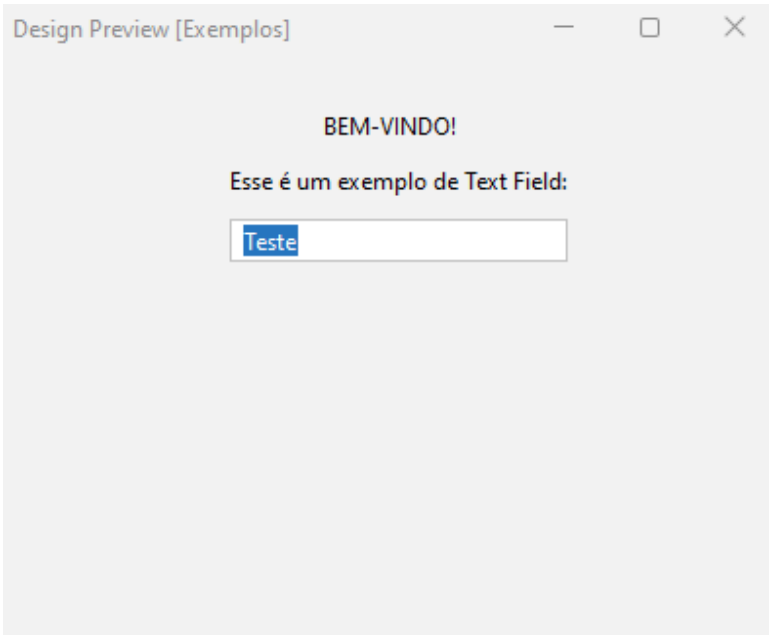

Figura 7 – Exemplo de **JTextField**
Fonte: Apache NetBeans IDE (2022)

Nome da classe: JTextField

Descrição: é o campo de texto no qual o usuário pode digitar um texto em uma linha.

Principais métodos:

setText(String): altera o texto do componente.

getText(): retorna o texto do componente.

Password field

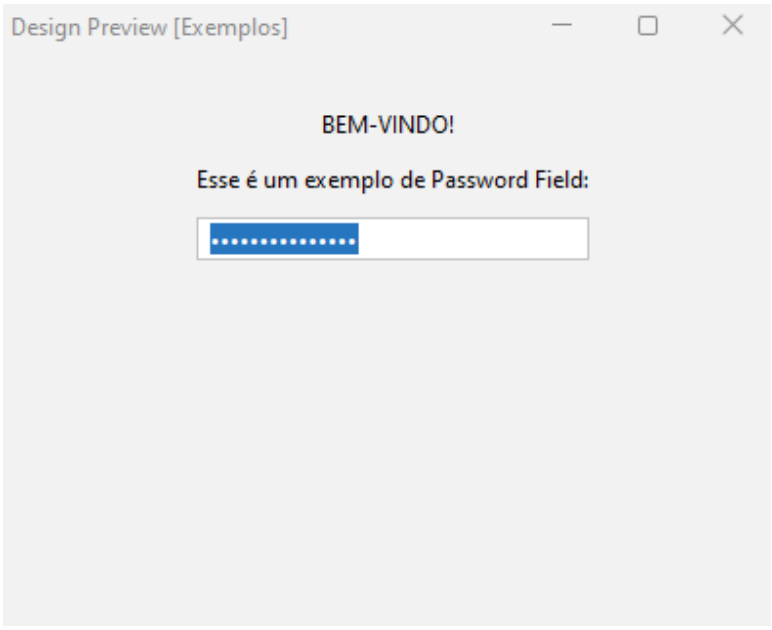

Figura 8 – Exemplo de **JPasswordField**
Fonte: Apache NetBeans IDE (2022)

Nome da classe: JPasswordField

Descrição: permite ao usuário inserir uma entrada de texto, mas que será exibida de modo camuflado, sendo cada caractere um asterisco.

Principais métodos:

setEchoChar(char): define o caractere que será exibido ao digitar.

Text area

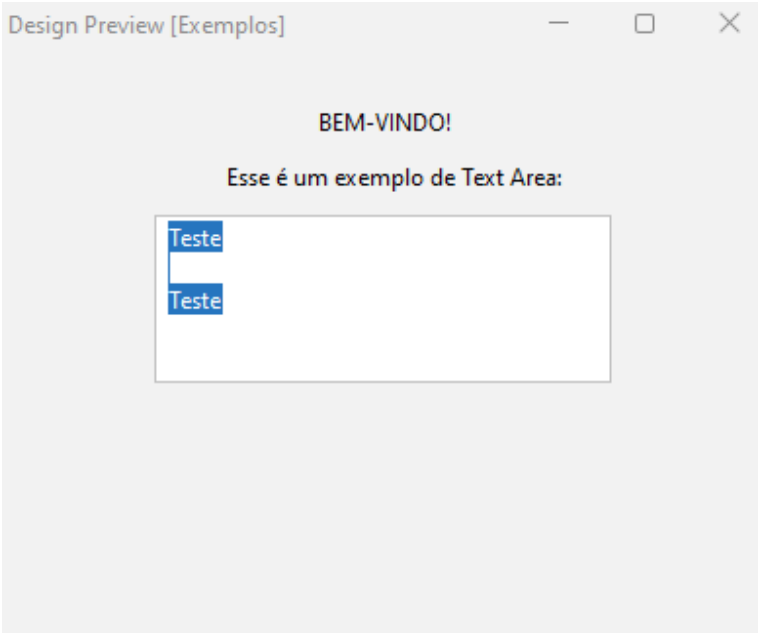

Figura 9 – Exemplo de **JTextArea**
Fonte: Apache NetBeans IDE (2022)

Nome da classe: JTextArea

Descrição: é similar à classe **JTextField**, que permite ao usuário digitar algo. Porém, ao invés de digitar apenas uma linha de texto, o **JTextArea** cria uma caixa de texto que permite a digitação em várias linhas.

Principais métodos:

setText(String): altera o texto do componente.

getText(): retorna o texto do componente.

getSelectedText(): retorna o texto selecionado pelo usuário.

insert(String, int): insere um texto na linha especificada.

Checkbox

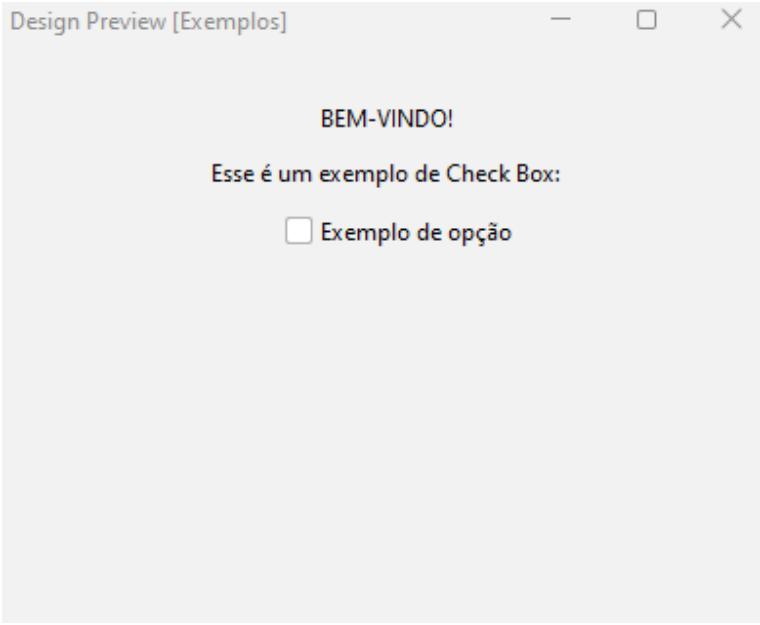

Figura 10 – Exemplo de **JCheckBox**
Fonte: Apache NetBeans IDE (2022)

Nome da classe: JCheckBox

Descrição: caixa de seleção que permite ao usuário selecionar ou não uma opção.

Principais métodos:

setText(String string): define o texto do item.

setSelected(Boolean b): se o parâmetro for *true*, o item fica selecionado.

Radio button

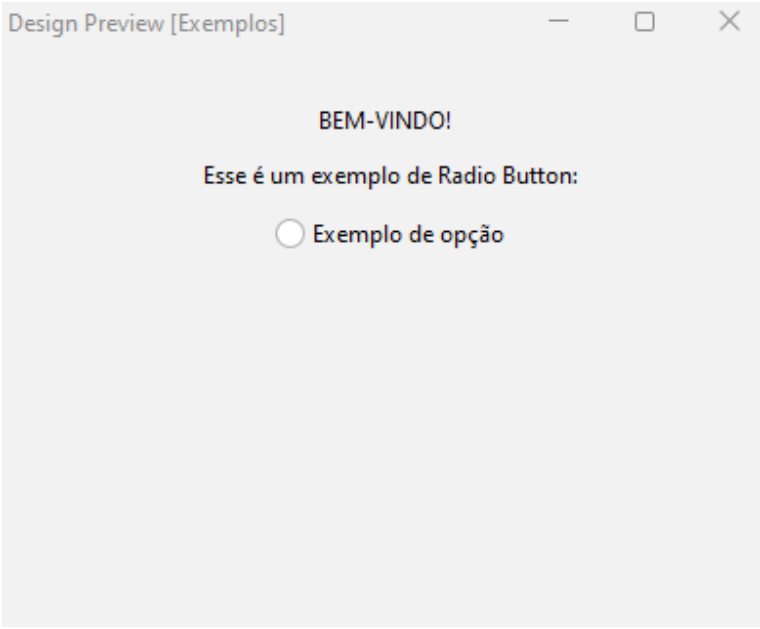

Figura 11 – Exemplo de **JRadioButton**
Fonte: Apache NetBeans IDE (2022)

Nome da classe: JRadioButton

Descrição: componente que permite selecionar uma entre várias opções. É semelhante à classe **JCheckBox** e por isso contém os mesmos construtores e métodos.

Principais métodos:

setText(String string): define o texto do item.

setSelected(Boolean b): se o parâmetro for *true*, o item fica selecionado.

Combobox

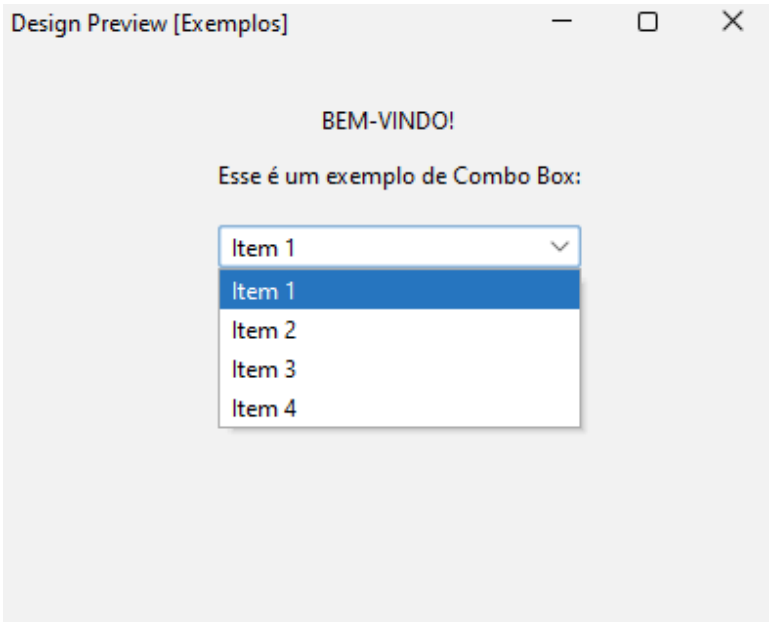

Figura 12 – Exemplo de **JComboBox**

Fonte: Apache NetBeans IDE (2022)

Nome da classe: JComboBox

Descrição: é uma caixa de combinação na qual o usuário pode selecionar uma opção.

Principais métodos:

addItem(Object obj): é utilizado para adicionar um item no *combo box*. Normalmente neste caso utiliza-se uma *string*.

setSelectedIndex(int index): define qual item do *combo box* começa selecionado.

Dica de uso:

Você consegue alterar as opções apresentadas no **JComboBox** diretamente pelo GUI Builder do Apache NetBeans IDE. Para isso, clique com o botão direito sobre o componente e acesse as propriedades clicando em **Properties**. Uma nova janela abrirá e na *model* você terá acesso aos itens da lista – cada linha representa um item. Depois de fazer as alterações, basta clicar em **OK**.

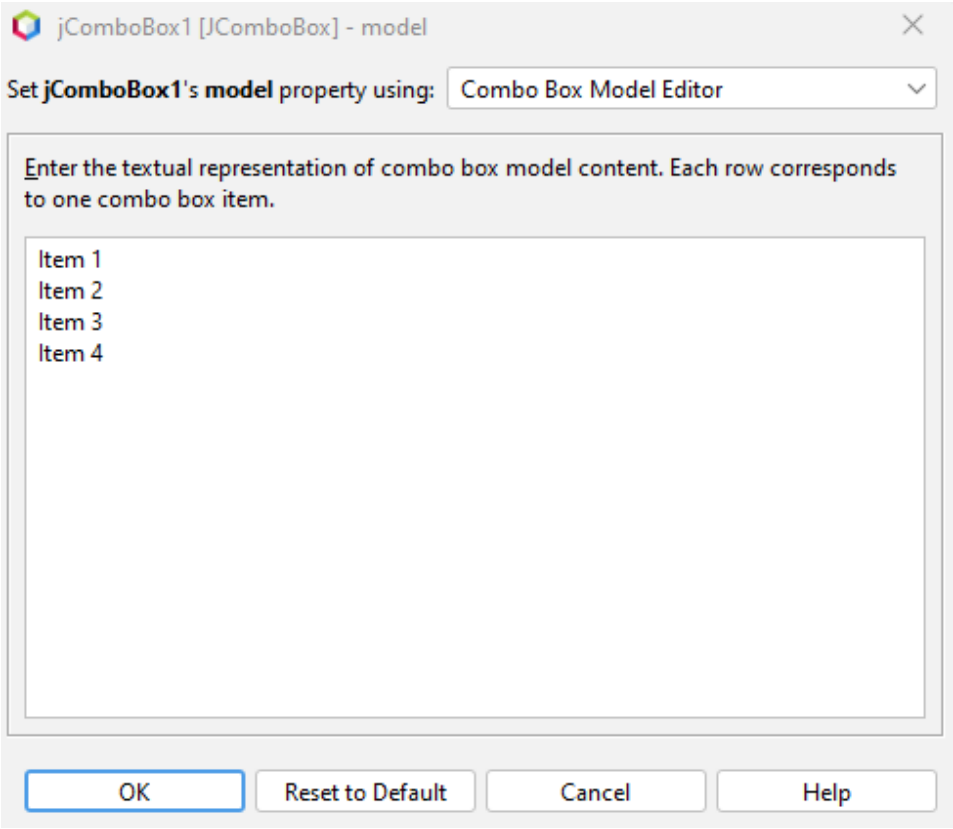

Figura 13 – Propriedades do componente **JComboBox**

Fonte: Apache NetBeans IDE (2022)

List

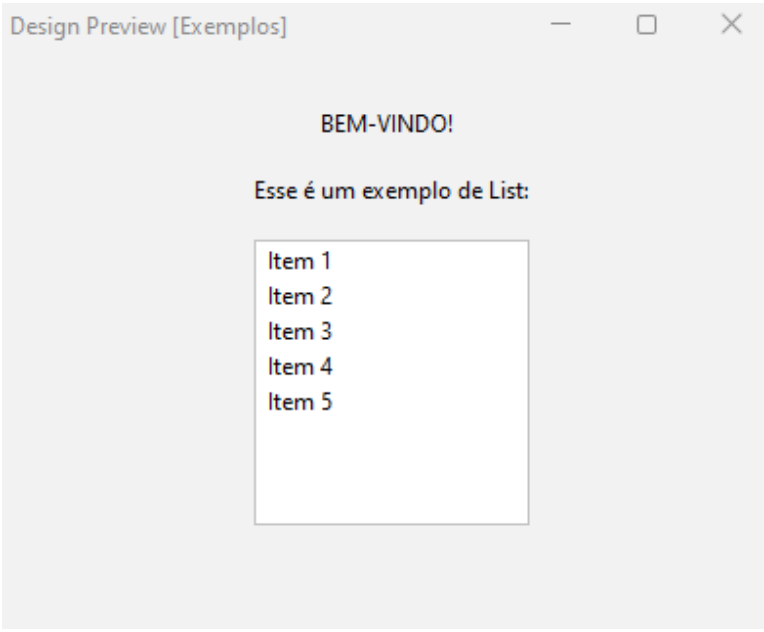

Figura 14 – Exemplo de **JList**
Fonte: Apache NetBeans IDE (2022)

Nome da classe: JList

Descrição: é uma lista de opções que permite a seleção de mais de um item simultaneamente.

Principais métodos:

setModel(ListModel list): permite adicionar/atualizar os dados na lista, ou seja, os tipos de dados passados como parâmetros em um **ListModel**. Para adicionar valores ao **ListModel**, utilize o método **addElement()**.

Dica de uso:

Assim como em **JComboBox**, é possível definir os itens que estarão presentes no componente **JList** clicando com o botão direito do mouse sobre o componente, selecionando item "Properties" e depois a propriedade "model".

Button

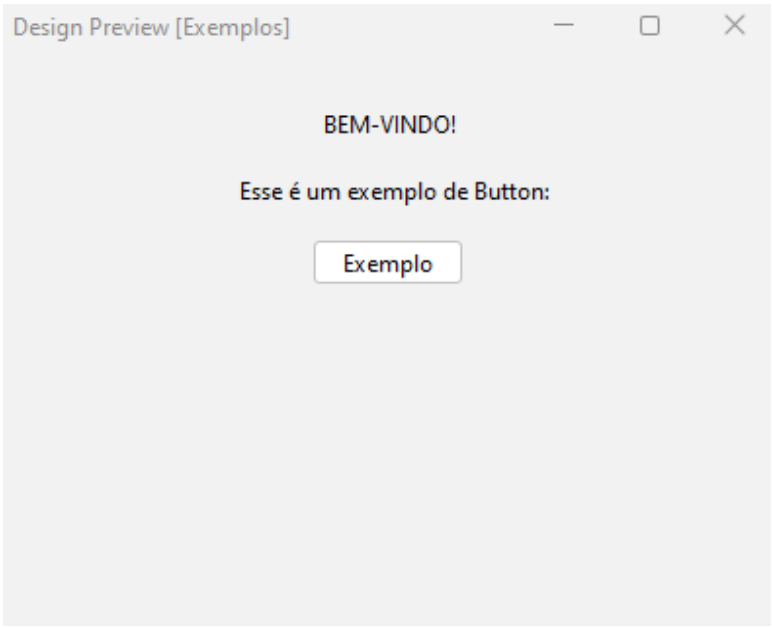

Figura 15 – Exemplo de **JButton**
Fonte: Apache NetBeans IDE (2022)

Nome da classe: JButton

Descrição: é um botão com um texto, que permite a interação do usuário por meio de um clique.

Principais métodos:

setText(String string): define o texto que será exibido no botão.

setEnabled(Boolean enabled): se o parâmetro for *true*, significa que o usuário pode clicar no botão. O botão ficará bloqueado, caso o valor seja *false*.

setIcon(Icon defaultIcon): define um ícone para o botão – deve-se passar o caminho do ícone.

Table

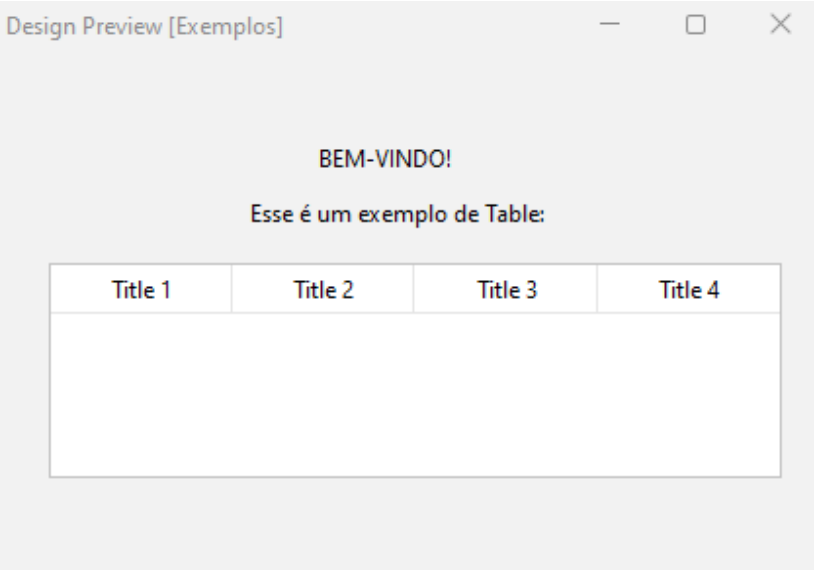

Figura 16 – Exemplo de **JTable**
Fonte: Apache NetBeans IDE (2022)

Nome da classe: JTable

Descrição: cria uma tabela composta por cabeçalho, linhas e colunas.

Principais métodos:

addColumn(TableColumn[] column): adiciona colunas ao fim da tabela.

clearSelection(): limpa seleção da tabela.

setValueAt(int linha, int coluna): altera o valor da célula na linha e coluna informadas.

Dica de uso: normalmente, as tabelas são responsáveis por exigir uma grande quantidade de dados. Sendo assim, o tamanho padrão da janela do programa nem sempre será o suficiente para exigir todas as linhas. Por isso, recomenda-se que as tabelas sejam inseridas dentro do componente **Scroll Pane**, que automaticamente criará uma barra de rolagem quando os dados não couberem na tela.

Por exemplo:

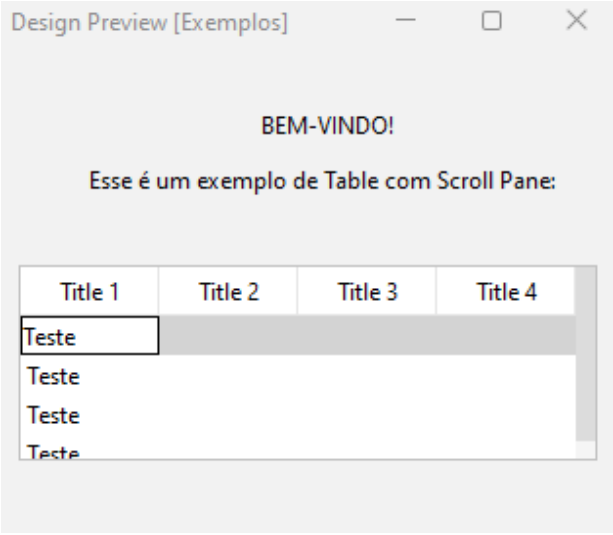

Figura 17 – Exemplo de **JScrollPane**

Fonte: Apache NetBeans IDE (2022)

Dentro do GUI Builder do Apache NetBeans IDE existem muitos outros componentes à disposição para você construir suas telas. Agora que você já sabe como usar os principais componentes, sinta-se livre para experimentá-los e, sempre que possível, confira as atualizações diretamente na documentação do Java.

Manipulação de eventos e uso de controles

Para o funcionamento completo de um *software*, não basta ter uma interface gráfica bonita, é preciso que o sistema responda às interações do usuário corretamente. Para isso, considere o seguinte algoritmo:

1. Enquanto o programa estiver rodando, verifica-se se houve alguma interação do usuário. Por exemplo: se o *mouse* foi movido ou se alguma tecla foi pressionada.
2. Caso tenha havido interação, realiza-se alguma ação.
3. Segue-se repetindo essa verificação enquanto o programa estiver rodando.

Em Java, para esse caso, haveria mais ou menos a seguinte estrutura de código:

```
while(programaRodando == true) {
  boolean resposta = verificaSeHouveInteracao();

  if(resposta == true) {
      // Realizamos alguma ação
  }
}
```

Porém, essa solução não é considerada eficiente, pois seria necessário explicitamente “ler” os periféricos (*mouse* e teclado) para tratar as ações do usuário dentro do programa. Caso você siga essa ideia, perderia muito tempo e

processamento para fazer essas leituras, causando lentidões nas outras tarefas que deveriam ser feitas pelo sistema. Felizmente, a linguagem de programação Java contém um paradigma no qual se pode executar trechos de códigos com o disparo de eventos. Isso se chama “programação orientada a eventos”.

A linguagem de programação orientada a eventos tem o usuário como o papel principal na execução das rotinas que compõem um programa. Essas ações ou rotinas são disparadas por ele por meio da interação com o programa. Após isso, os eventos são capturados pelo ambiente para acionar funções escritas pelo programador. Na prática, isso ocorreria da seguinte forma:

1. Sempre que o usuário interage com o *mouse* ou o teclado, o sistema operacional gera um **evento**.
2. Se a aplicação (imagine uma aplicação Java) estiver interessada em um evento específico (por exemplo, clique do *mouse* no botão **Salvar**), deve solicitar ao sistema operacional para **escutar** o evento.
3. Se aplicação não estiver interessada, seu processamento continuará normalmente.

Perceba que a aplicação não é responsável por identificar a ocorrência de eventos, pois o responsável é o sistema operacional.

Esse paradigma da programação orientada a eventos em que o sistema operacional avisa o programa que determinado evento aconteceu é baseado no Princípio de Hollywood: “*don’t call us, we’ll call you*”, ou, em português, “não nos chame, nós chamaremos você”.

Alguns exemplos de eventos são:

Clicar sobre um botão: evento *click*

Digitar caracteres dentro de uma caixa de texto: evento *change*

Passar o ponteiro do *mouse* sobre um componente: evento *MouseMove*

Pressionar uma tecla: evento *KeyPress* ou evento *KeyDown*

Liberar um botão do *mouse* após tê-lo pressionado: evento *MouseUp*

Focalizar um componente: evento *SetFocus*

Na linguagem de programação Java, há diversas classes para se trabalhar com o tratamento de eventos. Essas classes são chamadas de interfaces ouvintes de eventos e fazem parte do pacote **java.awt.event**.

As interfaces ouvintes de eventos são chamadas assim porque são responsáveis por “escutar” se determinado evento ocorreu. Se o evento ocorrer, o sistema operacional irá informá-la e outro evento poderá ocorrer. Em outras palavras, elas exercem o papel de um ***listener*** ou “ouvinte”. Cada *listener* utiliza uma classe específica para representar o evento, de acordo com o tipo que será gerado.

Figura 18 – Hierarquia de classes do pacote **AWTEvent**
Fonte: https://www.falkhausen.de/Java-8/java.awt/event/Event-Hierarchy.html

Observação

O pacote **AWTEvent** oferece várias interfaces para ouvintes, mas, para fins didáticos, este estudo focará apenas nas principais interfaces.

Ação do usuário	Evento disparado	Ouvinte do evento
Clicar em um componente	**ActionEvent**	**ActionListener**
Clicar em um item de escolha (**JCheckBox**)	**ItemEvent**	**ItemListener**
Pressionar uma tecla do teclado	**KeyEvent**	**KeyListener**
Clicar em um componente (**JComponent**)	**MouseEvent**	**MouseListener**
Abrir, fechar, minimizar ou maximizar uma janela	**WindowEvent**	**WindowListener**

Para que você compreenda como cada recurso funciona na prática, será criado um novo projeto Java no Apache NetBeans e serão apresentados alguns casos de uso para cada evento.

Observação

Para fins didáticos, você verá exemplos em código Java que não utilizam o construtor de interface (GUI Builder) do Apache NetBeans IDE. Para isso, uma classe chamada **Exemplo** será criada e utilizada para escrever os códigos dos exemplos a seguir.

Encerramento

Com isso, foi concluída a primeira parte do conteúdo sobre a criação de interfaces gráficas em Java para *desktop*. Até aqui, você aprendeu a teoria e a prática referentes a como construir telas com o GUI Builder e manipular eventos com seus respectivos controles. Na próxima parte, você colocará em prática tudo que aprendeu em um projeto completo com janelas, formulários, listagens e funcionalidades de cadastro e listagem de dados.