VOLTAR

# Lidando com Eventos de GUI em Java

Não basta desenvolvermos telas, precisamos dar comportamentos a elas, ou seja, os componentes de nossas telas precisam ter eventos associados, como eventos de cliques em botões, em campos de texto, checkboxes etc. Aprenderemos aqui como assimilar os componentes de nossa interface gráfica com eventos na programação.

NESTE TÓPICO



## O que são eventos de GUI?

Eventos de GUI são as ações que ocorrem ao se acionar algum componente da interface gráfica como, por exemplo, apertar botões, passar o mouse sobre algum label ou campo de texto, alterar um campo de texto, trocar a seleção em botões de radio etc.

Os eventos são associados, sempre, a algum método. Para cada tipo de evento em cada componente, há um método para trata-lo. Vamos explorar, aqui, os principais eventos dos principais componentes do pacote **Swing**.

## Eventos de cliques em botões

Começaremos tratando eventos de cliques em botões pois é um dos eventos mais comuns quando estamos trabalhando com interfaces gráficas.

Eventos de cliques são as ações que ocorrem quando clivamos em algum botão. Ao entendermos o funcionamento deste processo, implementar o tratamento de eventos de outros componentes será bastante fácil.

Segundo Teruel, os eventos de ação são tratados, normalmente, a partir de métodos assinados na interface *ActionListener,* ou seja, um método que "escuta" a ação esperada (por isso nome de *Listener).* Estes eventos são gerados quando se clica em objetos do formulário como botões, listas, opções de menu entre outros. A maneira mais comum para uma classe tratar os eventos de ação é implementando o método *actionPerformed* da interface *ActionListener*.

Sendo assim, a classe deve implementar a interface *ActionListener* e, reescrever obrigatoriamente (lembre-se do conceito de classes tipo interface), o método *actionPerformed*. Além disso, o elemento que será alvo da ação de clique do mouse deverá se "auto monitorar" constantemente para que identifique quando ocorreu o clique do mouse. Isso é feito pelo método *addActionListener*.

Para exemplificar isso, vamos criar uma tela com três campos: Um de texto para o usuário preencher seu nome, um de telefone e outro sendo uma caixa de múltipla seleção onde o usuário seleciona a cidade dele. Nesta tela haverá dois botões: Um para adicionar à lista as informações inseridas e outro para limpar os campos do formulário.

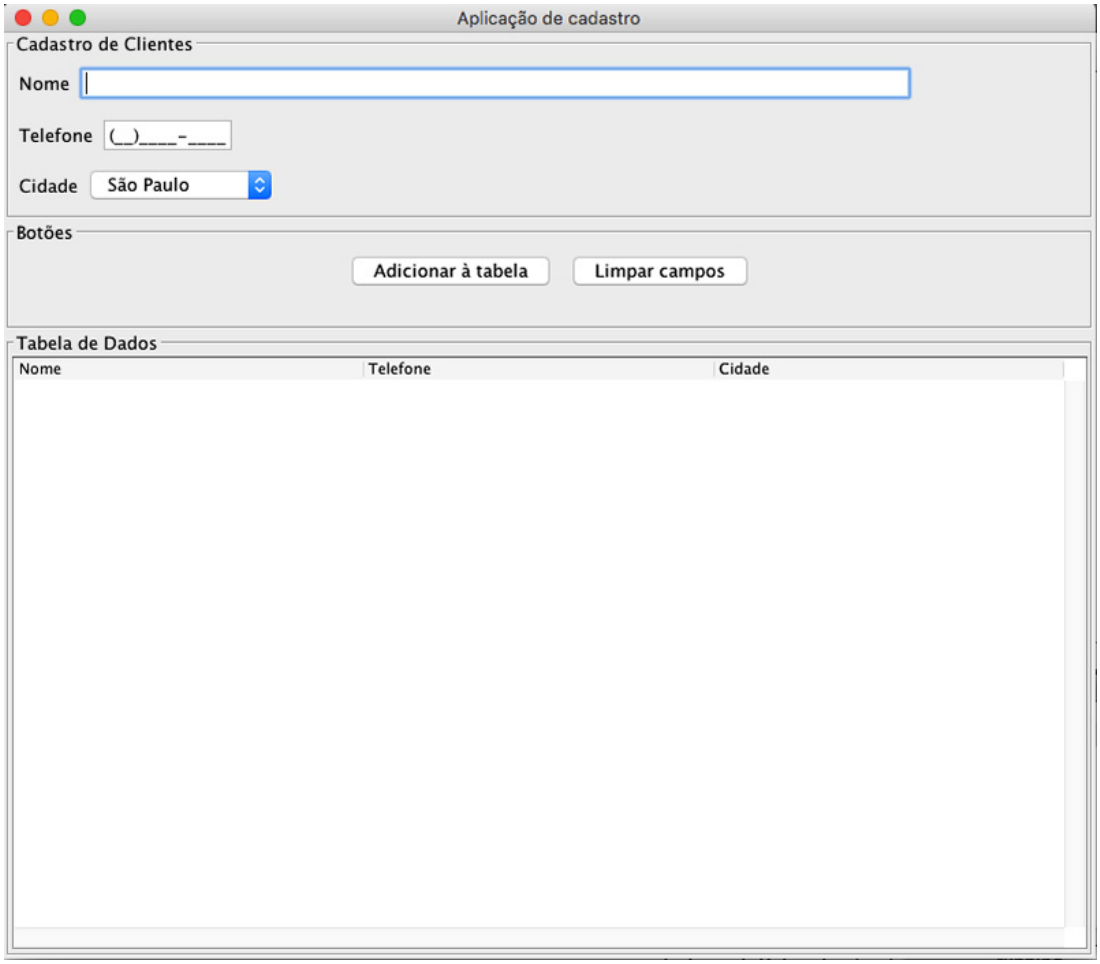
Veja a implementação abaixo dessa tela. Não deixe de ler os comentários do código-fonte, pois são bastante explicativos em relação as linhas mais importantes, especialmente pois este é um código bastante completo (Teruel):

```
1.  //Importações necessárias
2.
3.  import javax.swing.*;
4.  import java.awt.*;
5.  import java.awt.event.ActionEvent;
6.  import java.awt.event.ActionListener;
7.  import java.text.ParseException;
8.  import static javax.swing.JFrame.EXIT_ON_CLOSE;
9.  import javax.swing.border.TitledBorder;
10. import javax.swing.table.DefaultTableModel;
11. import javax.swing.text.MaskFormatter;
12.
13. //Como temos eventos, a classe herda de jFrame E implementa a ActionListener
14. public class Aplicativo extends JFrame implements ActionListener {
15.
16.     //Atributos locais da classe:
17.     private JPanel pnlCampos, pnlTabela, pnlBotoes, pnlNome, pnlTelefone, pnlCidade;
18.     private TitledBorder tituloPnlCampos, tituloPnlTabela, tituloPnlBotoes;
19.     private JLabel lblNome, lblTelefone, lblCidade;
20.     private JTextField txtNome;
21.     private JFormattedTextField txtTelefone;
22.     private MaskFormatter mskTelefone;
23.     private JTable tblClientes;
24.     private DefaultTableModel tblClientesModel;
25.     JButton btnAdicionar, btnLimpar;
26.     private JComboBox cmbCidades;
27.     private DefaultComboBoxModel cmbCidadesModel;
28.
29.     //Construtor, que coloca tudo no lugar
30.     public Aplicativo() {
31.         definirJanela(); //Chama o método que seta as propriedades da janela
32.         pnlCampos = new JPanel(new GridLayout(3, 1)); //Um painel para os campos
33.         tituloPnlCampos = BorderFactory.createTitledBorder("Cadastro de Clientes");
34.         pnlCampos.setBorder(tituloPnlCampos);
35.
36.         //Painel para os botões:
37.         pnlBotoes = new JPanel(new FlowLayout());
38.         tituloPnlBotoes = BorderFactory.createTitledBorder("Botões");
39.         pnlBotoes.setBorder(tituloPnlBotoes);
40.
41.         //Painel para a tabela
42.         pnlTabela = new JPanel(new GridLayout(1, 1));
43.         tituloPnlTabela = BorderFactory.createTitledBorder("Tabela de Dados");
44.         pnlTabela.setBorder(tituloPnlTabela);
45.
46.         //Seta os labels
47.         lblNome = new JLabel("Nome");
48.         lblTelefone = new JLabel("Telefone");
49.         lblCidade = new JLabel("Cidade");
50.         txtNome = new JTextField(50);
51.
52.         //Cria um campo de texto com uma máscara para o telefone
53.         try {
54.             mskTelefone = new MaskFormatter("(##)####-####");
55.             mskTelefone.setPlaceholderCharacter('_');
56.         } catch (ParseException ex) {
57.             JOptionPane.showMessageDialog(null, "Máscara incorreta");
58.         }
59.         txtTelefone = new JFormattedTextField(mskTelefone);
60.         cmbCidadesModel = new DefaultComboBoxModel();
61.         cmbCidades = new JComboBox(cmbCidadesModel);
62.
63.         //Cidades para o combo de seleção
64.         String cidade1 = "São Paulo";
65.         String cidade2 = "Rio de Janeiro";
66.         String cidade3 = "Campinas";
67.
68.         //Acrescenta as cidades
69.         cmbCidadesModel.addElement(cidade1);
70.         cmbCidadesModel.addElement(cidade2);
71.         cmbCidadesModel.addElement(cidade3);
```

```
72.          pnlNome = new JPanel(new FlowLayout(FlowLayout.LEFT));
73.          pnlNome.add(lblNome);
74.          pnlNome.add(txtNome);
75.          pnlCampos.add(pnlNome);
76.
77.          pnlTelefone = new JPanel(new FlowLayout(FlowLayout.LEFT));
78.          pnlTelefone.add(lblTelefone);
79.          pnlTelefone.add(txtTelefone);
80.          pnlCampos.add(pnlTelefone);
81.
82.          pnlCidade = new JPanel(new FlowLayout(FlowLayout.LEFT));
83.          pnlCidade.add(lblCidade);
84.          pnlCidade.add(cmbCidades);
85.          pnlCampos.add(pnlCidade);
86.
87.          //Seta os botões
88.          btnAdicionar = new JButton("Adicionar à tabela");
89.          btnAdicionar.addActionListener(this); //Adiciona o Listener ao botão
90.          btnLimpar = new JButton("Limpar campos");
91.          btnLimpar.addActionListener(this); //Adiciona o Listener ao botão
92.          pnlBotoes.add(btnAdicionar);
93.          pnlBotoes.add(btnLimpar);
94.
95.          //Define a tabela
96.          String[] cols = {"Nome", "Telefone", "Cidade"};
97.          tblClientesModel = new DefaultTableModel(cols, 3);
98.          tblClientes = new JTable(tblClientesModel);
99.          tblClientesModel.setNumRows(0);
100.
101.         JScrollPane scrRolagem = new JScrollPane(tblClientes,
102.                 JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
103.                 JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
104.         pnlTabela.add(scrRolagem);
105.
106.         add(pnlCampos, BorderLayout.NORTH);
107.         add(pnlBotoes, BorderLayout.CENTER);
108.         add(pnlTabela, BorderLayout.SOUTH);
109.     }
110.
111.     //mëtodo para setar algumas proprietdades da janela
112.     private void definirJanela() {
113.         setTitle("Aplicação de cadastro");
114.         setDefaultCloseOperation(EXIT_ON_CLOSE);
115.         setLayout(new BorderLayout());
116.         setSize(800, 700);
117.         setLocation(200, 80);
118.     }
119.
120.     //Método main
121.     public static void main(String args[]) {
122.         Aplicativo ap = new Aplicativo();
123.         ap.setVisible(true);
124.     }
125.
126.     //Sobrescrita obrigatória da interface ActionListener
127.     @Override
128.     public void actionPerformed(ActionEvent e) {
129.         /*
130.          A ideia é assim: É um listerner para todos os botões.
131.          O Listener deve identificar qual botão foi clicado e disparar
132.          uma ação específica para cada um
133.          */
134.         if (e.getSource() == btnAdicionar) { //Para o botão adicionar
135.             String[] linha = {txtNome.getText(),
136.                 txtTelefone.getText(),
137.                 cmbCidadesModel.getSelectedItem().toString()};
138.             tblClientesModel.addRow(linha);
139.              JOptionPane.showMessageDialog(this, "Dados adicionados", "Feito", JOptio
    nPane.PLAIN_MESSAGE);
```

```
140.          } else if (e.getSource() == btnLimpar) { //Para o botão limpar
141.              int op = JOptionPane.showConfirmDialog(this, "Quer, realmente, limpar os
      campos?", "Limpar",
142.                      JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE);
143.              if (op == 0) {
144.                  txtNome.setText(null);
145.                  txtTelefone.setText(null);
146.                  cmbCidadesModel.setSelectedItem("São Paulo");
147.                  txtNome.requestFocus(); //Coloca o cursor no txtNome
148.              }
149.          }
150.      }
151.  }
```

E o resultado da execução deste código pode ser visto abaixo:




Resultado da execução do código com tratamento de eventos

## Caixas de mensagens

As caixas de mensagens são pequenas caixas que exibem alguma mensagem ao usuário e podem possuir conjuntos diferentes de botões que, por sua vez, podem ser tratados em código, ou seja, podemos saber qual botão o usuário clicou ao fechar a caixa de diálogo.

Em Java, temos 4 tipos de métodos para caixas de diálogo prontas, implementadas pela API *Swing*, que são:

- **showConfirmDialog**: Caixa com dois botões, um para confirmar outro para cancelar (ok e cancelar).
- **showInputDialog**: Caixa que permite que o usuário digite um valor e possui um botão para confirmar.
- **showMessageDialog**: Caixa com uma mensagem simples e com um único botão, para fecha-la.
- **showOptionDialog**: Caixa a qual você pode personalizar os botões e tratar o evento de cada um.

Para exemplificar, vamos acrescentar duas caixas de diálogo para cada um dos botões do projeto anterior. Quanto o usuário clicar no botão “Adicionar à Tabela” uma caixa de mensagem será exibida confirmando o acréscimo dos dados. Quando o usuário clicar em “Limpar campos”, uma caixa de diálogo irá perguntar para o usuário se ele realmente quer fazer isso.

O novo código do *ActionPerformed* (que trata os botões de nossa aplicação), ficará assim:

```
1.      @Override
2.      public void actionPerformed(ActionEvent e) {
3.          /*
4.           A ideia é assim: É um listerner para todos os botões.
5.           O Listener deve identificar qual botão foi clicado e disparar
6.           uma ação específica para cada um
7.           */
8.          if (e.getSource() == btnAdicionar) { //Para o botão adicionar
9.              String[] linha = {txtNome.getText(),
10.                 txtTelefone.getText(),
11.                 cmbCidadesModel.getSelectedItem().toString()};
12.             tblClientesModel.addRow(linha);
13.              JOptionPane.showMessageDialog(this, "Dados adicionados", "Feito", JOptio
    nPane.PLAIN_MESSAGE);
14.         } else if (e.getSource() == btnLimpar) { //Para o botão limpar
15.              int op = JOptionPane.showConfirmDialog(this, "Quer, realmente, limpar os
    campos?", "Limpar",
16.                     JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE);
17.             if (op == 0) {
18.                 txtNome.setText(null);
19.                 txtTelefone.setText(null);
20.                 cmbCidadesModel.setSelectedItem("São Paulo");
21.                 txtNome.requestFocus(); //Coloca o cursor no txtNome
22.             }
23.         }
24.     }
```

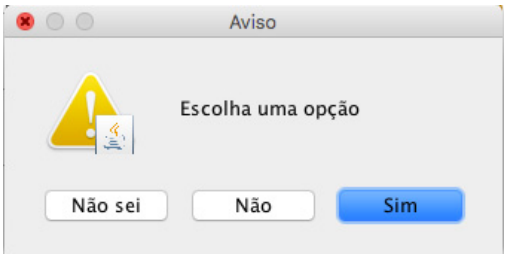
Repare, no código acima que usamos duas caixas de mensagem: Uma apenas mostrando a confirmação e outra com o tratamento.

É importante salientar que a caixa de mensagem que faz o tratamento do vento, retorna para o código um valor inteiro indicando qual opção o usuário escolheu. Se ele escolheu “sim”, o retorno será 0 e 1 caso ele escolha “não".

Você pode, também, personalizar os botões que aparecem na caixa de mensagem, tanto em quantidade de botões como no texto que aparece em cada botão. Para isso, vamos usar o **showOptionDialog**, exemplificado pelo trecho de código abaixo (Teruel):

```
1.          Object[] opt = {"Sim", "Não", "Não sei"};  //Lista de opções tipo vetor de o
    bjetos
2.          int op = JOptionPane.showOptionDialog(null, "Escolha uma opção", "Aviso",
3.                   JOptionPane.DEFAULT_OPTION, JOptionPane.WARNING_MESSAGE, null, opt,
    opt[0]);
4.
5.          //Trata as opções:
6.          if (op == 0) {
7.              JOptionPane.showMessageDialog(null, "Você clicou na opção Sim");
8.
9.          } else if (op == 1) {
10.             JOptionPane.showMessageDialog(null, "Você clicou na opção Não");
11.
12.         } else if (op == 2) {
13.             JOptionPane.showMessageDialog(null, "Você clicou na opção Não Sei");
14.         }
```

E o resultado será:




Caixa de mensagem personalizada, com três botões

## Resumo da Aula

Nesta aula você aprendeu a lidar com eventos de GUI, dos mais diversos, através de um *listenert*, que "escuta" o evento e o disparada com ocorre.

Você aprendeu, também como criar e personalizar suas próximas caixas de mensagens, bem como tratar os eventos de cliques em diferentes botões das caixas.

Pratique bastante a codificação com tratamento de eventos. A prática lhe levará a perfeição, certamente.

## Quiz

Exercício Final

Lidando com Eventos de GUI em Java

INICIAR

## Referências

Teruel, E. C., 2015, Programação Orientada a Objetos com Java - sem mistérios - 1ª Ed., Editora Uninove

Deitei P. e Deitel H., 2010, Java : Como programar, 8ª Edição, Pearson Pretice Hall

Schildt, H., 2015, Schildt, Java para iniciantes : crie, compile e execute programas Java rapidamente, Bookman

Avalie este tópico

ANTERIOR
Conceitos Básicos de GUI em Java

Índice

PRÓXIMO
Usando Gerenciadores de Layout

Biblioteca (https://www.uninove.br/conheca-a-uninove/biblioteca/sobre-a-biblioteca/apresentacao/)
Portal Uninove (http://www.uninove.br)
Mapa do Site



Ajuda? (https://ava.un idCurso=)