MANIPULAÇÃO DE EVENTOS

Ao término desse capítulo você terá aprendido:

- ✓ *Tipos de eventos;*
- ✓ *Manipulação de eventos de componentes;*
- ✓ *Manipulação de eventos do mouse;*
- ✓ *Manipulação de eventos do teclado;*
- ✓ *Interfaces Listener;*
- ✓ *Classes Adaptadoras de eventos.*

Programas baseados em janelas são assíncronos, isto significa que qualquer coisa pode acontecer a qualquer tempo. Por exemplo, o programa não "sabe" predizer quando o usuário poderá clicar sobre um botão.

Por causa disso, programas gráficos são orientados a eventos (modelo de delegação de eventos), desse modo, interações realizadas pelo usuário são passadas para o programa processar.

1.1 FUNCIONAMENTO BÁSICO

Basicamente, os eventos trabalham dessa maneira:

- ✓ Qualquer sistema operacional que suporta GUI (interface gráfica) monitora eventos tais como clique de mouse, teclas pressionadas etc;
- ✓ O sistema operacional informa esses eventos ao programa que está executando;
- ✓ O programa decide o que fazer com esses eventos.

O programador precisa construir um número de tratadores de eventos capaz de processar um tipo particular de evento:

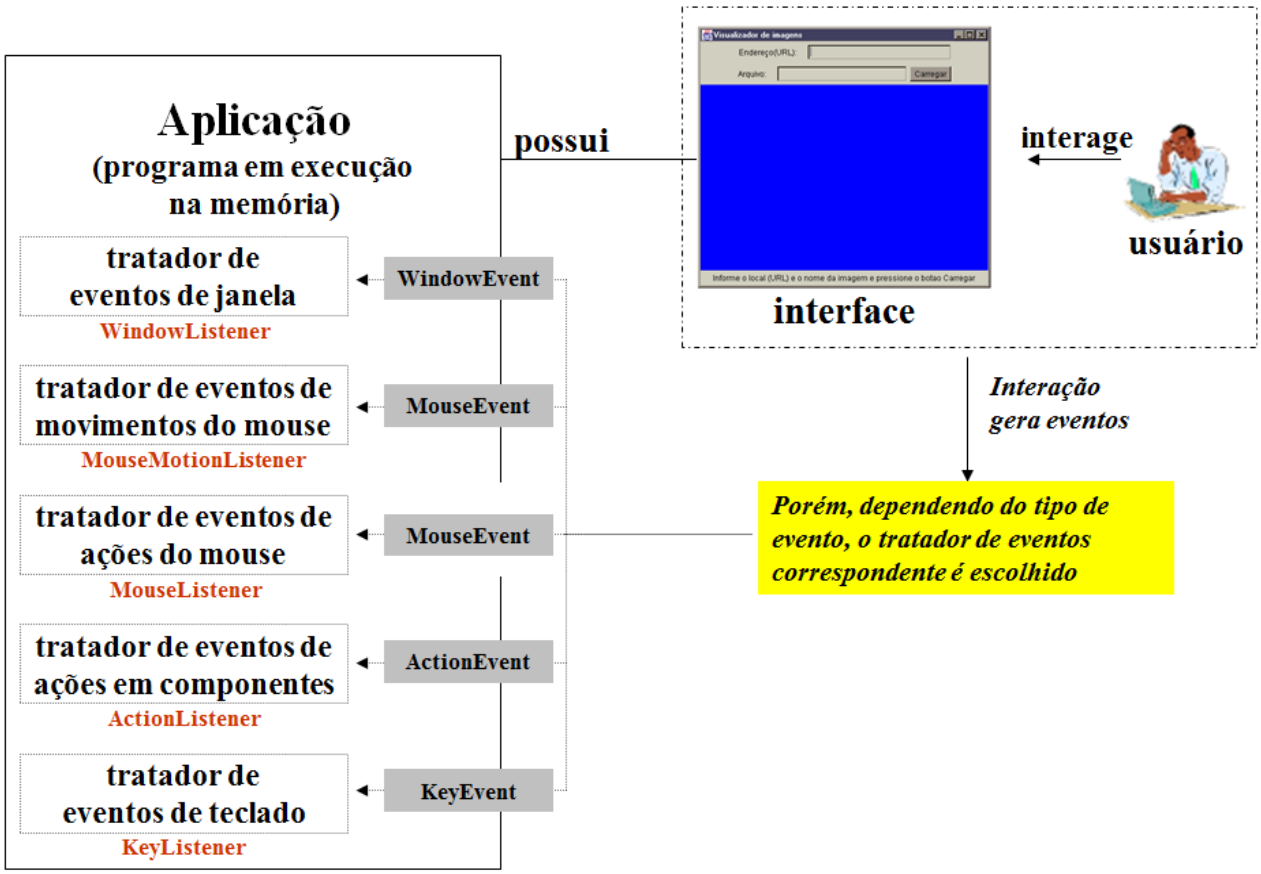
- ✓ Clique do mouse;
- ✓ Movimentação do mouse;
- ✓ Tecla pressionada

Uma vez tenha sido definido o tratamento do evento, ele deve ser registrado para que o máquina virtual Java possa usá-lo.

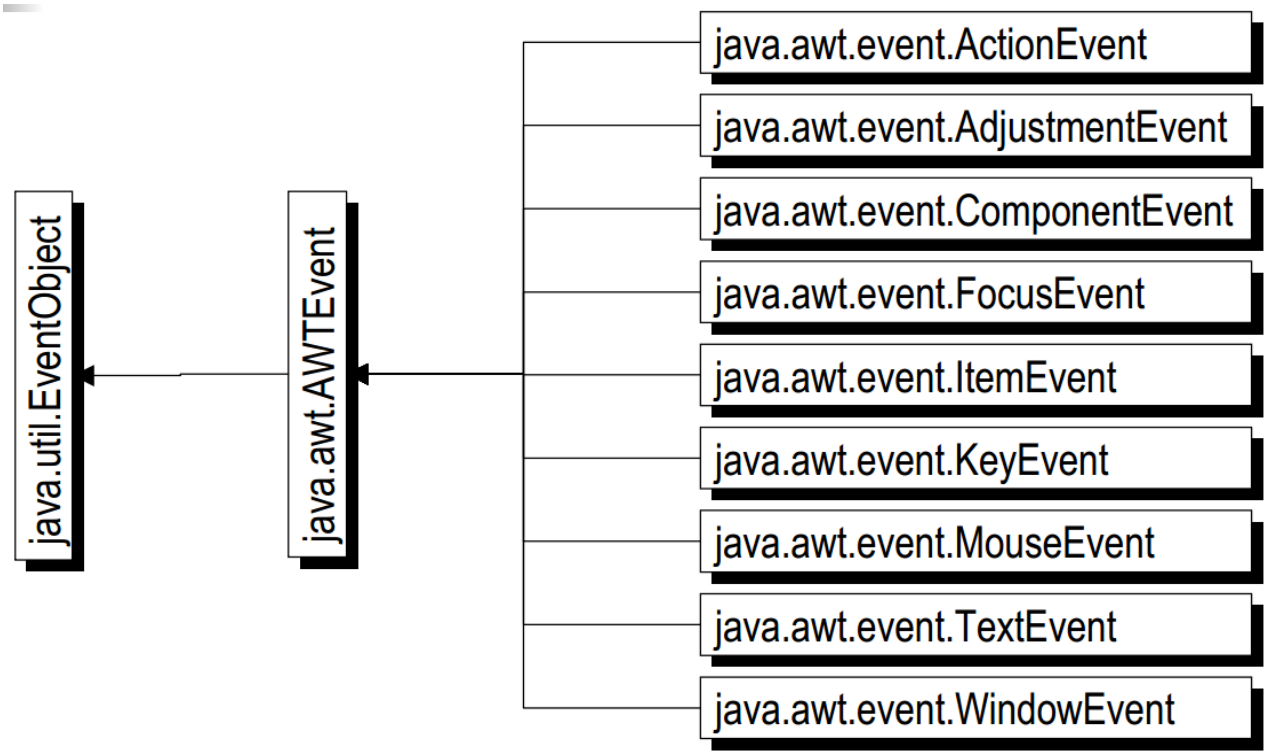
Normalmente, cada componente (botão, menu, item de lista, janela etc) tem um número de tratadores de eventos associados. Por exemplo: Quando um mouse dá entrada em um componente, o evento "***MouseEntered***" é gerado. Logo após a ocorrência do evento, a máquina virtual Java verifica se o componente, no qual o mouse deu a

entrada, possui um evento registrado para esse tipo de ocorrência. Havendo o evento de entrada do mouse registrado, é passado o controle para esse componente e o método associado é executado.

A figura a seguir ilustra o processo de interação do usuário com uma determinada janela, contendo as respectivas aplicações e tratadores de eventos incidentes.



1.2 HIERARQUIA DAS CLASSES DE EVENTOS



Categoria	Eventos	Interface
Mouse	Arrastar/Soltar o mouse	MouseMotionListener
	Clique sobre um botão etc	MouseListener
Teclado	Tecla pressionada	KeyListener
Seleção de itens	Quando seleciona um item	ItemListener
Controle de texto	Nova linha em um controle	TextListener
Controle Scrolling	Quando deslisa a barra de rolagem	AdjustmentListener
Botões, menus, etc	Quando pressiona ou seleciona	ActionListener
Mudanças janela	Abrir/fechar/minimizar etc	WindowListener
Mudança de foco	Tabulação/Enter/Foco	FocusListener
Componente	Tamanho/esconder/mover	ComponentListener
Container	Adicionar/mover um componente	ContainerListener

1.3 EVENTOS DE COMPONENTES

Existe uma classe em Java que será a responsável pelo tratamento de eventos. Ou seja, é nela que se identifica o evento que ocorreu e é nela que se define as ações que os aplicativos devem executar quando tal evento ocorrer.

A interface ***ActionListener*** é composta apenas de um *método abstrato,* chamado ***actionPerformed***. Assim, a interface fica na espera de algum evento, e caso ocorra, ele é imediatamente passado para o método ***actionPerformed***. É no corpo desse método que se definem as ações de um determinado componente. Por exemplo: a programação de um botão “gravar”.

Listener pode ser traduzido como 'ouvinte'.

Para fazer uso dessas funcionalidades, deve-se importar:

- ✓ *import java.awt.event.ActionEvent;*
- ✓ *import java.awt.event.ActionListener;*

Existem diversas formas de tratamento de ações em Java. Pode-se criar uma classe específica para tratar todos os eventos, de todos os componentes de uma janela (JFrame), por exemplo.

Também pode-se definir, em cada componente, o seu próprio tratamento para cada tipo de evento necessário.

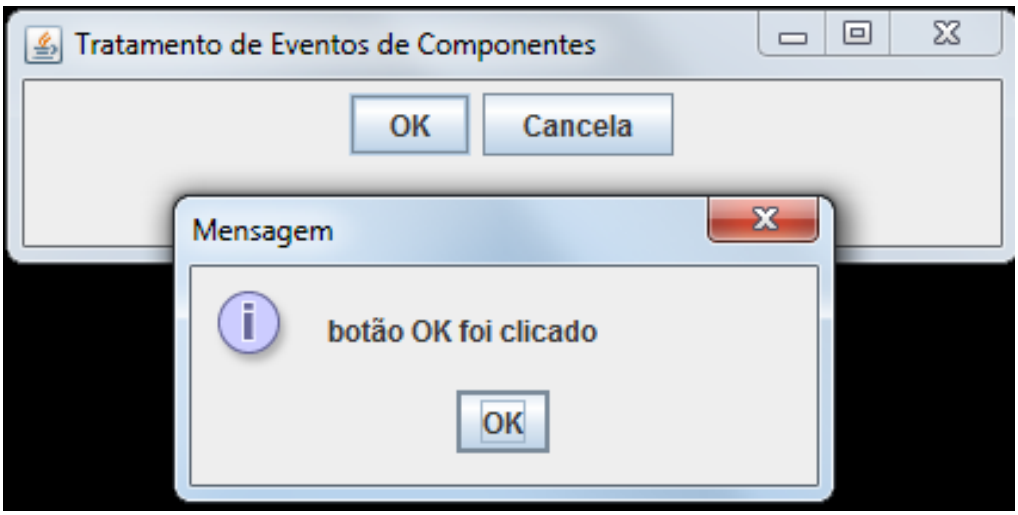
A seguir serão demonstradas essas duas formas:

6.3.1 Através de uma classe que implemente a interface *ActionListener*

```
1 package eventos;
2
3 import java.awt.event.ActionEvent;
4 import java.awt.event.ActionListener;
5 import javax.swing.JButton;
6 import javax.swing.JFrame;
7 import javax.swing.JOptionPane;
8
9 public class TrataEventoComponente extends JFrame implements ActionListener {
10     private JButton botaoOK, botaoCancela;
11
12     public TrataEventoComponente(JButton botaoOK, JButton botaoCancela) {
13         this.botaoOK = botaoOK;
14         this.botaoCancela = botaoCancela;
15     }
16
17     @Override
18     public void actionPerformed(ActionEvent evento) {
19         if (evento.getSource() == botaoOK) {
20             JOptionPane.showMessageDialog(null, "botão OK foi clicado");
21         }
22
23         if (evento.getSource() == botaoCancela) {
24             JOptionPane.showMessageDialog(null, "botão CANCELA foi clicado");
25         }
26     } // fim do método actionPerformed
27 } // fim da classe TrataEventoComponente
```

```
 1 package eventos;
 2
 3 import java.awt.FlowLayout;
 4 import javax.swing.JFrame;
 5 import javax.swing.JButton;
 7 public class EventoComponente extends JFrame {
 9     private JButton btnOK = new JButton("OK");
10     private JButton btnCancela = new JButton("Cancela");
11     private TrataEventoComponente evento;
12
13     public EventoComponente() {
14         super("Tratamento de Eventos de Componentes");
15         setLayout(new FlowLayout());
16         evento = new TrataEventoComponente(btnOK, btnCancela);
17
18         btnOK.addActionListener(evento);
19         add(btnOK);
20
21         btnCancela.addActionListener(evento);
22         add(btnCancela);
23     }
24
25     public static void main(String[] args) {
26         EventoComponente obj = new EventoComponente();
27
28         obj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
29         obj.setSize(350, 200);
30         obj.setLocation(300, 300);
31         obj.setVisible(true);
32     }
33 }
```

✓ Resultado da execução da classe ***EventoComponente***



6.3.2 Adicionar diretamente em cada componente o tratamento de evento

```
1 package eventos;
2
3 import java.awt.FlowLayout;
4 import java.awt.event.ActionEvent;
5 import java.awt.event.ActionListener;
6 import javax.swing.JFrame;
7 import javax.swing.JButton;
8 import javax.swing.JOptionPane;
9
10 public class EventoComponenteDireto extends JFrame {
11
12     private JButton btnOK = new JButton("OK");
13     private JButton btnCancela = new JButton("Cancela");
14     private TrataEventoComponente evento;
15
16     public EventoComponenteDireto() {
17         super("Tratamento de Eventos de Componentes");
18         setLayout(new FlowLayout());
19
20         add(btnOK);
21         add(btnCancela);
22
23         // adicionando tratamento de evento ao botao OK
24         btnOK.addActionListener(
25             new ActionListener() {
26                 @Override
27                 public void actionPerformed(ActionEvent e) {
28                    JOptionPane.showMessageDialog(null, "Clicou no Botão
OK","Tratamento de Evento", JOptionPane.WARNING_MESSAGE);
30                 }
31             }
32         );
33         // adicionando tratamento de evento ao botao CANCELAR
34         btnCancela.addActionListener(
35             new ActionListener() {
36                 @Override
37                 public void actionPerformed(ActionEvent event) {
38                    JOptionPane.showMessageDialog(null, "Clicou no Botão
CANCELA", "Tratamento de Evento", JOptionPane.WARNING_MESSAGE);
40                         System.exit(0);
41                  }
42                 }
43         );
44     }
45
46     public static void main(String[] args) {
47         EventoComponenteDireto obj = new EventoComponenteDireto();
48
49         obj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
50         obj.setSize(350, 200);
51         obj.setLocation(300, 300);
52         obj.setVisible(true);
53     }
54 }
```

- ✓ O resultado da execução da classe ***EventoComponenteDireto*** é o mesmo do exemplo da classe ***EventoComponente***.

1.4 EVENTOS DO MOUSE

Os eventos gerados por alguma ação do mouse (clicar um botão, por exemplo), são tratados pela classe *MouseListener* ou *MouseAdapter* e suas classes-filhas, definidas pelo programador. Essas classes disponibilizam os seguintes métodos:

- ✓ **mouseClicked(MouseEvent ev)** → quando um botão do mouse é *clicado*;
- ✓ **mouseEntered(MouseEvent ev)** → quando o mouse entra em um componente (passa por cima);
- ✓ **mouseExited(MouseEvent ev)** → quando o mouse sai de um componente;
- ✓ **mousePressed(MouseEvent ev)** → quando um botão do mouse é pressionado;
- ✓ **mouseReleased(MouseEvent ev)** → quando um botão do mouse é solto.

Já os eventos gerados por algum movimento do mouse são tratados pela classe *MouseMotionListener* ou *MouseMotionAdapter* e suas classes-filhas, definidas pelo programador. Essas classes disponibilizam os seguintes métodos:

- ✓ **mouseMoved(MouseEvent ev)** → quando o mouse é movido dentro de um componente ou janela;
- ✓ **mouseDraged(MouseEvent ev)** → quando o mouse é movido com um botão pressionado dentro de um componente ou janela (para implementar o recurso *arrastar-e-soltar*);

A seguir segue um exemplo contendo a implementação completa para todos os métodos da interface MouseListener e MouseMotionListener.

```
 1 package eventos;
 2
 3 import java.awt.Color;
 4 import java.awt.BorderLayout;
 5 import java.awt.event.MouseListener;
 6 import java.awt.event.MouseMotionListener;
 7 import java.awt.event.MouseEvent;
 8 import javax.swing.JFrame;
 9 import javax.swing.JLabel;
10 import javax.swing.JPanel;
11
12 public class EventosMouse extends JFrame {
13
14    private JPanel mousePanel; // painel em que os eventos de mouse ocorrerão
15    private JLabel statusBar;  // rótulo que exibe informações de evento
16
17    // construtor MouseTrackerFrame configura GUI e
18    // registra handlers de evento de mouse
19    public EventosMouse() {
20        super("Demonstração de Eventos do Mouse");
21
22        mousePanel = new JPanel(); // cria o painel
23        mousePanel.setBackground(Color.WHITE); // configura cor de fundo
24        add(mousePanel, BorderLayout.CENTER); // adiciona painel ao JFrame
25
26        statusBar = new JLabel("Mouse fora do JPanel");
27        add(statusBar, BorderLayout.SOUTH); // adiciona rótulo ao JFrame
28
```

```
29          // cria e registra listener para mouse e eventos de movimento
30          MouseHandler handler = new MouseHandler();
31          mousePanel.addMouseListener(handler);
32          mousePanel.addMouseMotionListener(handler);
33      } // fim do construtor de MouseTrackerFrame
34
35      private class MouseHandler implements MouseListener,
36              MouseMotionListener {
37          // Handlers de evento de MouseListener
38          // trata evento quando o mouse é liberado logo depois de pressionado
39          public void mouseClicked(MouseEvent event) {
40              statusBar.setText(String.format("Clicado em [%d, %d]",
41                      event.getX(), event.getY()));
42          } // fim do método mouseClicked
43
44          // trata evento quando mouse é pressionado
45          public void mousePressed(MouseEvent event) {
46              statusBar.setText(String.format("Pressionado em [%d, %d]",
47                      event.getX(), event.getY()));
48          } // fim do método mousePressed
49
50          // trata evento quando mouse é liberado depois de arrastado
51          public void mouseReleased(MouseEvent event) {
52              statusBar.setText(String.format("Liberado em [%d, %d]",
53                      event.getX(), event.getY()));
54          } // fim do método mouseReleased
55
56          // trata evento quando mouse entra na área
57          public void mouseEntered(MouseEvent event) {
58              statusBar.setText(String.format("Mouse na área [%d, %d]",
59                      event.getX(), event.getY()));
60              mousePanel.setBackground(Color.BLUE);
61          } // fim do método mouseEntered
62
63          // trata evento quando mouse sai da área
64          public void mouseExited(MouseEvent event) {
65              statusBar.setText("Mouse fora do JPanel");
66              mousePanel.setBackground(Color.WHITE);
67          } // mouseExited fim do método
68
69          // Handlers de evento de MouseMotionListener
70          // trata evento quando arrasta o mouse com o botão pressionado
71          public void mouseDragged(MouseEvent event) {
72              statusBar.setText(String.format("Arrastando em [%d, %d]",
73                      event.getX(), event.getY()));
74          } // fim do método mouseDragged
75
76          // trata evento quando usuário move o mouse
77          public void mouseMoved(MouseEvent event) {
78              statusBar.setText(String.format("Movido em [%d, %d]",
79                      event.getX(), event.getY()));
80          } // fim do método mouseMoved
81      } // fim da classe interna MouseHandler
82
83      public static void main(String args[]) {
84          EventosMouse eventos = new EventosMouse();
85          eventos.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
86          eventos.setSize(500, 300); // configura o tamanho da janela
87          eventos.setLocation(300, 200); // posiciona a janela
88          eventos.setVisible(true); // exibe a janela
89      } // fim do método main
90  } // fim da classe EventosMouse
```

1.5 EVENTOS DE TECLADO

Eventos de teclado são gerados quando as teclas do teclado são pressionadas e liberadas, sendo controlados através da interface KeyListener, que possui os seguintes métodos:

- ✓ **void keyPressed(KeyEvent e)** → Invocado quando uma tecla é pressionada.
- ✓ **void keyReleased(KeyEvent e)** → Invocado quando uma tecla é solta (após ser pressionada).
- ✓ **void keyTyped(KeyEvent e)** → Invocado quando uma tecla é pressionada (para capturar a tecla).

A seguir é apresentado um exemplo para manipulação de eventos de teclado através da interface *KeyListener*.

```
 1 package eventos;
 2
 3 import java.awt.Color;
 4 import java.awt.event.KeyListener;
 5 import java.awt.event.KeyEvent;
 6 import javax.swing.JFrame;
 7 import javax.swing.JTextArea;
 8
 9 public class EventosTeclado extends JFrame implements KeyListener {
10
11     private String line1 = ""; // primeira linha de textarea
12     private String line2 = ""; // segunda linha de textarea
13     private String line3 = ""; // terceira linha de textarea
14     private JTextArea textArea; // textarea a exibir saída
15
16     // construtor KeyDemoFrame
17     public EventosTeclado() {
18         super("Demonstrando Eventos de Teclado em Java");
19
20         textArea = new JTextArea(10, 15); // configura JTextArea
21         textArea.setText("Pressione uma tecla...");
22         textArea.setEnabled(false); // desativa textarea
23         textArea.setDisabledTextColor(Color.BLACK); // configura cor de texto
24         add(textArea); // adiciona textarea ao JFrame
25
26         addKeyListener(this); // permite que o frame processe os eventos de
teclado
27     } // fim do construtor KeyDemoFrame
28
29     // trata o pressionamento de qualquer tecla
30     public void keyPressed(KeyEvent event) {
31         line1 = String.format("Tecla pressionada: %s",
32                 event.getKeyText(event.getKeyCode())); // gera saída de tecla
pressionada
33         setLines2and3(event); // configura a saída das linhas dois e três
34     } // fim do método keyPressed
35
36     // trata liberação de qualquer tecla
37     public void keyReleased(KeyEvent event) {
38         line1 = String.format("Tecla liberada: %s",
39                 event.getKeyText(event.getKeyCode())); // gera saída de tecla
liberada
40         setLines2and3(event); // configura a saída das linhas dois e três
```

```
41      } // fim do método keyReleased
42
43      // trata pressionamento de uma tecla de ação
44      public void keyTyped(KeyEvent event) {
45          line1 = String.format("Tecla pressionada: %s", event.getKeyChar());
46          setLines2and3(event); // configura a saída das linhas dois e três
47      } // fim do método keyTyped
48
49      // configura segunda e terceira linhas de saída
50      private void setLines2and3(KeyEvent event) {
51          line2 = String.format("\nEssa tecla %sé uma tecla de função",
52                  (event.isActionKey() ? "" : "não "));
53
54          String temp = event.getKeyModifiersText(event.getModifiers());
55
56          line3 = String.format("\nTecla modificadora pressionada: %s",
57                  (temp.equals("") ? "nenhuma" : temp)); // modificadores de
saída
58
59          textArea.setText(String.format("%s\n%s\n%s\n",
60                  line1, line2, line3)); // gera saída de três linhas de texto
61      } // fim do método setLines2and3
62
63      public static void main(String args[]) {
64          EventosTeclado teclado = new EventosTeclado();
65          teclado.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
66          teclado.setSize(500, 300); // set frame size
67          teclado.setLocation(300, 300);
68          teclado.setVisible(true); // display frame
69      } // end main
70  } // end class KeyDemoFrame
```

1.6 INTERFACES x CLASSES ADAPTADORAS

Muitas das interfaces listeners de eventos fornecem muitos métodos, mas nem sempre é necessário implementar todos os métodos disponíveis nessas interfaces. Por exemplo, o programa pode precisar apenas do tratamento do evento arrastar botões do mouse. Utilizando a interface *MouseMotionListener*, precisamos, obrigatoriamente, implementar todos os métodos, mesmo que deixando-os sem corpo (sem código, apenas a assinatura do método).

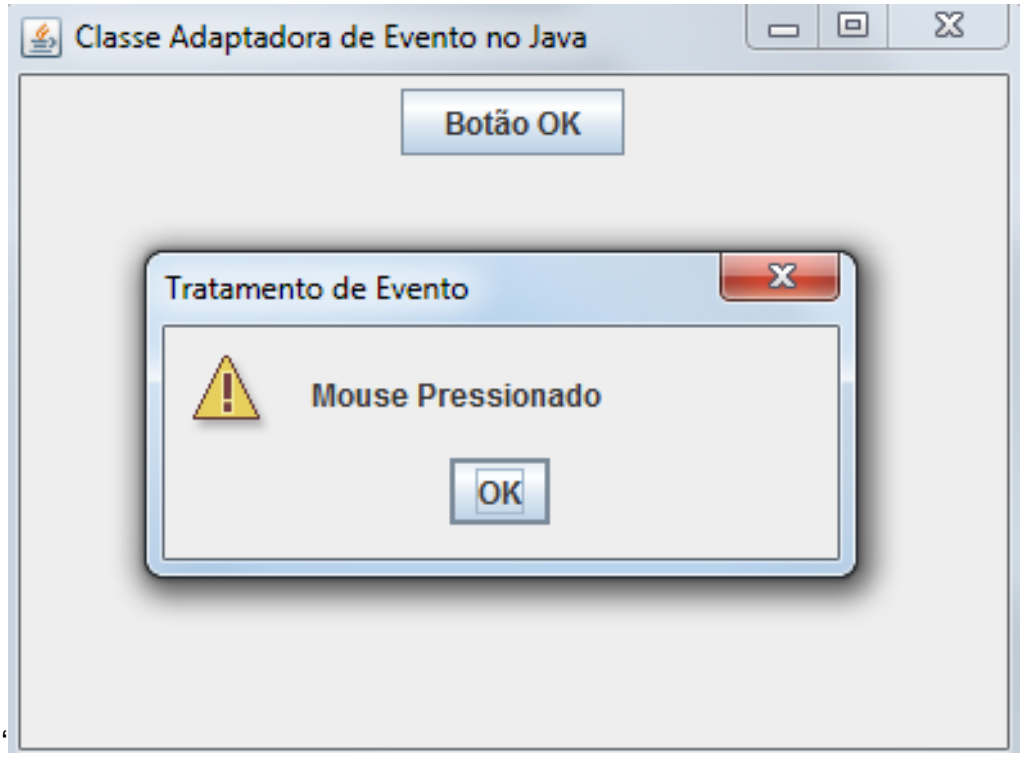
Para suprir essa necessidade, existem as classes adaptadoras, que fornecem a capacidade de implementação individual dos métodos necessários para tratamento de eventos.

Uma classe adaptadora implementa uma interface e fornece uma implementação default (com o corpo do método vazio) de cada método da interface. O desenvolvedor pode estender a classe adaptadora para herdar a implementação default de cada método, e depois sobrescrever o(s) método(s) necessário(s) para o tratamento de eventos.

Algumas classes adaptadoras *java.awt.event* são apresentadas abaixo:

Classe Adaptadora	Interface Listener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener
ActionListener	não possui classe adaptadora pois só possui um método.

A seguir, um exemplo de uma classe que estende a classe adaptadora ***MouseAdapter***, com uma subclasse (classe interna) para a criação de uma janela Swing com um botão. Importante ressaltar que, para fins didáticos, o tratamento de evento não está configurado para o botão, e sim para a janela. Dessa forma, para que o método ***mousePressed*** seja acionado, deve-se clicar em qualquer área da janela (JFrame), e não no botão "Botão OK".



```java
package eventos;

import java.awt.FlowLayout;
import java.awt.event.*;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class EventoClasseAdaptadora extends MouseAdapter {

    public static void main(String[] args) {
        ClasseAdaptadoraJanela janela = new ClasseAdaptadoraJanela();
        janela.addMouseListener(new MouseAdapter() {
            @Override
            public void mousePressed(MouseEvent e) {
                JOptionPane.showMessageDialog(null, "Mouse Pressionado",
                        "Tratamento de Evento", JOptionPane.WARNING_MESSAGE);
            }
        });
    }
}
// subclasse (classe interna)
class ClasseAdaptadoraJanela extends JFrame {
    JButton btnOK = new JButton("Botão OK");

    public ClasseAdaptadoraJanela() {
        setTitle("Classe Adaptadora de Evento no Java");
        setSize(400, 300);
        setLocation(300, 300);
        setLayout(new FlowLayout());
        add(btnOK);
        setVisible(true);
    }
}
```