# Módulo 2

Introdução à Programação II


# Lição 8

Tratamento de Eventos em Interfaces Gráficas

***Versão 1.0 - Mar/2007***

**Autor**
Rebecca Ong

**Equipe**
Joyce Avestro
Florence Balagtas
Rommel Feria
Rebecca Ong
John Paul Petines
Sun Microsystems
Sun Philippines

# Necessidades para os Exercícios

## Sistemas Operacionais Suportados

**NetBeans IDE 5.5** para os seguintes sistemas operacionais:

- Microsoft Windows XP Profissional SP2 ou superior
- Mac OS X 10.4.5 ou superior
- Red Hat Fedora Core 3
- Solaris™ 10 Operating System (SPARC® e x86/x64 Platform Edition)

**NetBeans Enterprise Pack**, poderá ser executado nas seguintes plataformas:

- Microsoft Windows 2000 Profissional SP4
- Solaris™ 8 OS (SPARC e x86/x64 Platform Edition) e Solaris 9 OS (SPARC e x86/x64 Platform Edition)
- Várias outras distribuições Linux

## Configuração Mínima de Hardware

**Nota:** IDE NetBeans com resolução de tela em 1024x768 pixel

| Sistema Operacional | Processador | Memória | HD Livre |
|---|---|---|---|
| Microsoft Windows | 500 MHz Intel Pentium III workstation ou equivalente | 512 MB | 850 MB |
| Linux | 500 MHz Intel Pentium III workstation ou equivalente | 512 MB | 450 MB |
| Solaris OS (SPARC) | UltraSPARC II 450 MHz | 512 MB | 450 MB |
| Solaris OS (x86/x64 Platform Edition) | AMD Opteron 100 Série 1.8 GHz | 512 MB | 450 MB |
| Mac OS X | PowerPC G4 | 512 MB | 450 MB |

## Configuração Recomendada de Hardware

| Sistema Operacional | Processador | Memória | HD Livre |
|---|---|---|---|
| Microsoft Windows | 1.4 GHz Intel Pentium III workstation ou equivalente | 1 GB | 1 GB |
| Linux | 1.4 GHz Intel Pentium III workstation ou equivalente | 1 GB | 850 MB |
| Solaris OS (SPARC) | UltraSPARC IIIi 1 GHz | 1 GB | 850 MB |
| Solaris OS (x86/x64 Platform Edition) | AMD Opteron 100 Series 1.8 GHz | 1 GB | 850 MB |
| Mac OS X | PowerPC G5 | 1 GB | 850 MB |

## Requerimentos de Software

NetBeans Enterprise Pack 5.5 executando sobre Java 2 Platform Standard Edition Development Kit 5.0 ou superior (JDK 5.0, versão 1.5.0_01 ou superior), contemplando a Java Runtime Environment, ferramentas de desenvolvimento para compilar, depurar, e executar aplicações escritas em linguagem Java. Sun Java System Application Server Platform Edition 9.

- Para **Solaris**, **Windows**, e **Linux**, os arquivos da JDK podem ser obtidos para sua plataforma em http://java.sun.com/j2se/1.5.0/download.html
- Para **Mac OS X**, Java 2 Plataform Standard Edition (J2SE) 5.0 Release 4, pode ser obtida diretamente da Apple's Developer Connection, no endereço: http://developer.apple.com/java (é necessário registrar o download da JDK).

Para mais informações: http://www.netbeans.org/community/releases/55/relnotes.html

### *Colaboradores que auxiliaram no processo de tradução e revisão*

Alexandre Mori
Alexis da Rocha Silva
Aline Sabbatini da Silva Alves
Allan Wojcik da Silva
André Luiz Moreira
Andro Márcio Correa Louredo
Antoniele de Assis Lima
Antonio Jose R. Alves Ramos
Aurélio Soares Neto
Bruno da Silva Bonfim
Bruno dos Santos Miranda
Bruno Ferreira Rodrigues
Carlos Alberto Vitorino de Almeida
Carlos Alexandre de Sene
Carlos André Noronha de Sousa
Carlos Eduardo Veras Neves
Cleber Ferreira de Sousa
Cleyton Artur Soares Urani
Cristiano Borges Ferreira
Cristiano de Siqueira Pires
Derlon Vandri Aliendres
Fabiano Eduardo de Oliveira
Fábio Bombonato
Fernando Antonio Mota Trinta
Flávio Alves Gomes
Francisco das Chagas
Francisco Marcio da Silva
Gilson Moreno Costa
Givailson de Souza Neves
Gustavo Henrique Castellano
Hebert Julio Gonçalves de Paula
Heraldo Conceição Domingues
Hugo Leonardo Malheiros Ferreira
Ivan Nascimento Fonseca
Jacqueline Susann Barbosa
Jader de Carvalho Belarmino
João Aurélio Telles da Rocha
João Paulo Cirino Silva de Novais
João Vianney Barrozo Costa
José Augusto Martins Nieviadonski
José Leonardo Borges de Melo
José Ricardo Carneiro
Kleberth Bezerra G. dos Santos
Lafaiete de Sá Guimarães
Leandro Silva de Morais
Leonardo Leopoldo do Nascimento
Leonardo Pereira dos Santos
Leonardo Rangel de Melo Filardi
Lucas Mauricio Castro e Martins
Luciana Rocha de Oliveira
Luís Carlos André
Luís Octávio Jorge V. Lima
Luiz Fernandes de Oliveira Junior
Luiz Victor de Andrade Lima
Manoel Cotts de Queiroz
Marcello Sandi Pinheiro
Marcelo Ortolan Pazzetto
Marco Aurélio Martins Bessa
Marcos Vinicius de Toledo
Maria Carolina Ferreira da Silva
Massimiliano Giroldi
Mauricio Azevedo Gamarra
Mauricio da Silva Marinho
Mauro Cardoso Mortoni
Mauro Regis de Sousa Lima
Namor de Sá e Silva
Néres Chaves Rebouças
Nolyanne Peixoto Brasil Vieira
Paulo Afonso Corrêa
Paulo José Lemos Costa
Paulo Oliveira Sampaio Reis
Pedro Antonio Pereira Miranda
Pedro Henrique Pereira de Andrade
Renato Alves Félix
Renato Barbosa da Silva
Reyderson Magela dos Reis
Ricardo Ferreira Rodrigues
Ricardo Ulrich Bomfim
Robson de Oliveira Cunha
Rodrigo Pereira Machado
Rodrigo Rosa Miranda Corrêa
Rodrigo Vaez
Ronie Dotzlaw
Rosely Moreira de Jesus
Seire Pareja
Sergio Pomerancblum
Silvio Sznifer
Suzana da Costa Oliveira
Tásio Vasconcelos da Silveira
Thiago Magela Rodrigues Dias
Tiago Gimenez Ribeiro
Vanderlei Carvalho Rodrigues Pinto
Vanessa dos Santos Almeida
Vastí Mendes da Silva Rocha
Wagner Eliezer Roncoletta

### *Auxiliadores especiais*

Revisão Geral do texto para os seguintes Países:

- **Brasil** – Tiago Flach
- **Guiné Bissau** – Alfredo Cá, Bunene Sisse e Buon Olossato Quebi – ONG Asas de Socorro

### *Coordenação do DFJUG*

- **Daniel deOliveira** – JUGLeader responsável pelos acordos de parcerias
- **Luci Campos -** Idealizadora do DFJUG responsável pelo apoio social
- **Fernando Anselmo -** Coordenador responsável pelo processo de tradução e revisão, disponibilização dos materiais e inserção de novos módulos
- **Regina Mariani -** Coordenadora responsável pela parte jurídica
- **Rodrigo Nunes -** Coordenador responsável pela parte multimídia
- **Sérgio Gomes Veloso -** Coordenador responsável pelo ambiente JEDI™ (Moodle)

## Agradecimento Especial

**John Paul Petines –** Criador da Iniciativa JEDI™
**Rommel Feria** – Criador da Iniciativa JEDI™

# 1. Objetivos

Neste módulo, aprenderemos como tratar eventos disparados quando o usuário interage com a interface gráfica de sua aplicação (GUI). Após completar este módulo seremos capazes de desenvolver aplicações gráficas que responderão às interações do usuário.

Ao final desta lição, o estudante será capaz de:

- Enumerar os componentes do modelo de delegação de eventos
- Explicar como o modelo de delegação de eventos funciona
- Criar aplicações gráficas que interajam com o usuário
- Discutir os benefícios das classes *adapter*
- Discutir as vantagens de utilizar *inner* e *anonymous inner class*

# 2. Modelo de Delegação de Eventos

O modelo de Delegação de eventos descreve o modo como sua classe pode responder a uma interação do usuário. Para compreender o modelo, estudaremos primeiro três importantes componentes:

1. ***Event Source*** **(Gerador de Evento)**
   O *event source* refere-se ao componente da interface que origina o evento. Por exemplo, se o usuário pressiona um botão, o *event source* neste caso é o botão.

2. **Event Listener/Handler (Monitor de Eventos/Manipulador)**
   O *event listener* recebe informações de eventos e processa as interações do usuário. Quando um botão é pressionado, o *event listener* pode tratá-lo exibindo uma informação útil ao usuário.

3. **Event Object (Objeto Evento)**
   Quando um evento ocorre (por exemplo, quando o usuário interage com um componente da interface gráfica), um objeto *Event* é criado. Este objeto contém todas as informações necessárias sobre o evento gerado. As informações incluem o tipo de evento, digamos, "o mouse foi clicado". Há diversas classes de evento para diferentes categorias de ações do usuário. Um objeto *event* tem o tipo de dado de uma dessas classes.

Aqui está representado o modelo de delegação de eventos:

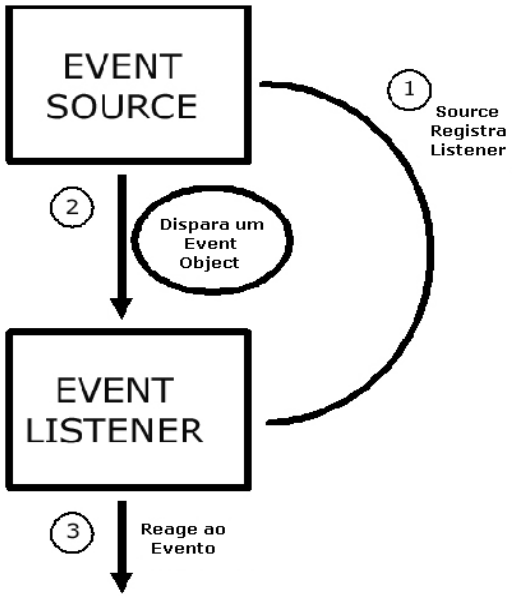



*Figura 1: Modelo de Delegação de Eventos*

Inicialmente, um *listener* deverá ser registrado pelo *event source*. Assim ele poderá receber informações sobre os eventos quando ocorrerem no *event source*. Somente um *listener* registrado poderá receber notificações dos eventos. Uma vez registrado, um *listener* simplesmente aguarda até que ocorra um evento.

Quando alguma coisa acontece no *event source*, um objeto *event*, que descreve o evento, é criado. O evento é então disparado pelo gerador para os *listener* registrados.

Quando o *listener* recebe um objeto *event* (ou seja, uma notificação) de um *event source*, ele executa sua função. Ele decifra a notificação e processa o evento ocorrido.

## *2.1. Registro de Listeners*

O *event source* registra um *listener* através dos métodos *add<Tipo>Listener.*

```
void add<Tipo>Listener(<Tipo>Listener listenerObj)
```

*<Tipo>* depende do tipo de *event source*. Ele pode ser *Key*, *Mouse*, *Focus*, *Component*, *Action* e outros.

Diversos *listener* podem ser registrados para um *event source* para recepção de notificações de eventos.

Um *listener* registrado pode também ser removido através do método *remove<Tipo>Listener.*

```
void remove<Tipo>Listener(<Tipo>Listener listenerObj)
```

# 3. Classes *Event*

Um *EventObject* tem uma classe de evento que indica seu tipo. Na raiz da hierarquia das classes *E*vent estão a classe *EventObject*, que é encontrada no pacote *java.util*. Uma subclasse imediata da classe *EventObject* é a classe *AWTEvent*. A classe *AWTEvent* está declarada no pacote *java.awt*. Ela é a raiz de todos os eventos baseados em AWT. A seguir temos algumas das classes de evento *AWT*:

| ***Classes de Evento*** | ***Descrição*** |
|---|---|
| ComponentEvent | Estende a classe *AWTEvent*. Instanciada quando um componente é movido, redimensionado, tornado visível ou invisível. |
| InputEvent | Estende a classe *ComponentEvent*. Classe abstrata de evento, raiz a partir da qual são implementadas todas as classes de componentes de entrada de dados. |
| ActionEvent | Estende a classe *AWTEvent*. Instanciada quando um botão é pressionado, um item de uma lista recebe duplo-clique ou quando um item de menu é selecionado. |
| ItemEvent | Estende a classe *AWTEvent*. Instanciada quando um item é selecionado ou desmarcado pelo usuário, seja numa lista ou caixa de seleção (checkbox). |
| KeyEvent | Estende a classe *InputEvent*. Instanciado quando uma tecla é pressionada, liberada ou digitada (pressionada e liberada). |
| MouseEvent | Estende a classe *InputEvent*. Instanciada quando um botão do mouse é pressionado, liberado, clicado (pressionado e liberado) ou quando o cursor do mouse entra ou sai de uma parte visível de um componente. |
| TextEvent | Estende a classe *AWTEvent*. Instanciada quando o valor de um campo texto ou area de texto sofre alteração. |
| WindowEvent | Estende a classe *ComponentEvent*. Instanciada quando um objeto *Window* é aberto, fechado, ativado, desativado, minimizado, restaurado ou quando o foco é transferido para dentro ou para fora da janela. |

*Tabela 1: Classes Event*

Tome nota: todas as subclasses de *AWTEvent* seguem esta convenção de nomeação:

```
<Tipo>Event
```

# 4. *Listeners* de Evento

*Listeners* de evento são classes que implementam as interfaces <*Tipo*>*Listener*. A tabela a seguir algumas interfaces utilizadas com maior freqüência:

| ***Listeners* de Evento** | ***Descrição*** |
|---|---|
| ActionListener | Recebe eventos de ação, estes podem ser um pressionamento do mouse ou da barra de espaço sobre o objeto. |
| MouseListener | Recebe eventos do mouse. |
| MouseMotionListener | Recebe eventos de movimento do mouse, que incluem arrastar e mover o mouse. |
| WindowListener | Recebe eventos de janela (abrir, fechar, minimizar, entre outros). |

*Tabela 2: Event Listeners*

## *4.1. Método de ActionListener*

Este é o método da interface *ActionListener* que deve ser implementado:

| **Médoto de *ActionListener*** |
|---|
| `public void actionPerformed(ActionEvent e)` |
| Chamado quando o mouse é pressionado ou foi utilizada a barra de espaço sobre um botão por exemplo. |

*Tabela 3: Método de ActionListener*

## *4.2. Métodos de MouseListener*

Estes são os métodos da interface *MouseListener* que devem ser implementados:

| **Métodos de *MouseListener*** |
|---|
| `public void mouseClicked(MouseEvent e)` |
| Chamado quando o mouse é pressionado (pressionar e soltar). |
| `public void mouseEntered(MouseEvent e)` |
| Chamado quando o cursor do mouse entra em um componente. |
| `public void mouseExited(MouseEvent e)` |
| Chamado quando o cursor do mouse sai de um componente. |
| `public void mousePressed(MouseEvent e)` |
| Chamado quando o botão do mouse é pressionado sobre um componente (pressionar). |
| `public void mouseReleased(MouseEvent e)` |
| Chamado quando o botão do mouse é solto sobre um componente (soltar). |

*Tabela 4: Métodos de MouseListener*

## *4.3. Métodos de MouseMotionListener*

Estes são os métodos da interface *MouseMotionListener* que devem ser implementados:

| ***Métodos MouseListener*** |
|---|
| `public void mouseDragged(MouseEvent e)` |
| Chamado quando o mouse é pressionado sobre um componente e então arrastado (*dragged*). É chamado tantas vezes quanto o mouse for arrastado. |
| `public void mouseMoved(MouseEvent e)` |
| Chamado quando o cursor do mouse é movido sobre um componente, sem que o mouse esteja pressionado. Será chamado múltiplas vezes, tantas quantas o mouse for movido. |

*Tabela 5: Métodos MouseMotionListener*

## *4.4. Métodos WindowListener*

Estes são os métodos da interface *WindowListener* que devem ser implementados:

| ***Métodos WindowListener*** |
|---|
| `public void windowOpened(WindowEvent e)` |
| Chamado quando uma janela é aberta (quando o objeto *Window* torna-se visível pela primeira vez) |
| `public void windowClosing(WindowEvent e)` |
| Chamado quando uma janela é encerrada (objeto *Window*). |
| `public void windowClosed(WindowEvent e)` |
| Chamado quando a janela foi fechada após a liberação de recursos utilizados pelo objeto (*EventSource*). |
| `public void windowActivated(WindowEvent e)` |
| Chamado quando uma janela é ativada, ou seja, está em uso. |
| `public void windowDeactivated(WindowEvent e)` |
| Chamado quando uma janela deixa de ser a janela ativa. |
| `public void windowIconified(WindowEvent e)` |
| Chamado quando a janela é iconificada (Este evento é utilizado especialmente para o ambiente Macintosh). |
| `public void windowDeiconified(WindowEvent e)` |
| Chamado quando a janela retorna do estado iconificado para o normal (Este evento é utilizado especialmente para o ambiente Macintosh). |

*Tabela 6: Métodos WindowListener*

## *4.5. Guia para criação de Aplicações Gráficas com tratamento de eventos*

Estes são os passos a serem lembrados ao criar uma aplicação gráfica com tratamento de eventos.

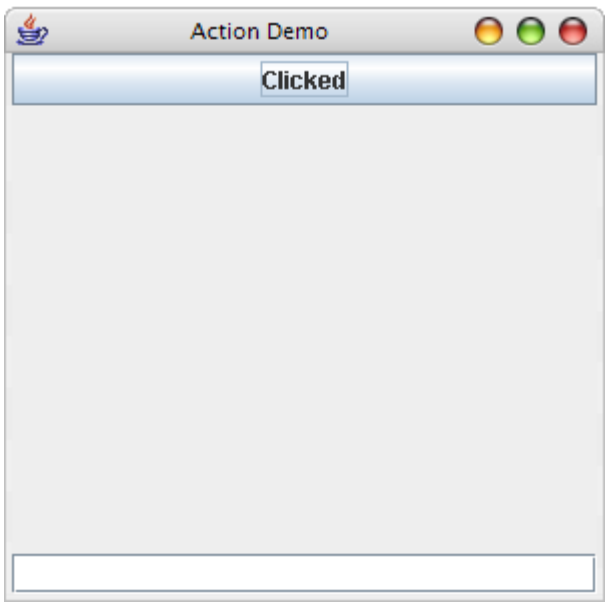
1. Criar uma classe que descreva e mostre a aparência da sua aplicação gráfica.
2. Criar uma classe que implemente a interface *listener* apropriada. Esta classe poderá estar inserida na classe do primeiro passo.
3. Na classe implementada, sobrepor TODOS os métodos da interface *listener*. Descrever em cada método como o evento deve ser tratado. Podemos deixar vazio um método que não desejamos tratar.
4. Registrar o objeto *listener* (a instância da classe *listener* do passo 2) no *EventSource* utilizando o método *add*<Tipo>*Listener*.

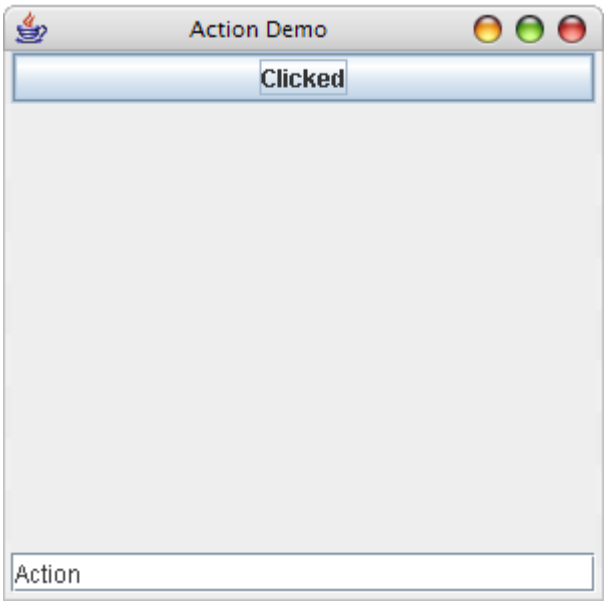
## 4.6. Exemplo do Evento Action

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class ActionDemo extends JFrame implements ActionListener {
    private JTextField tf;
    private JButton bt;
    public ActionDemo(String title){
        super(title);
        tf = new JTextField();
        bt = new JButton("Clicked");
        add(tf, BorderLayout.SOUTH);
        add(bt, BorderLayout.NORTH);
        setSize(300,300);
        bt.addActionListener(this);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent actionEvent) {
        if (tf.getText().equals("Action"))
            tf.setText("");
        else
            tf.setText("Action");
    }
    public static void main(String args[]) {
        new ActionDemo("Action Demo");
    }
}
```

A seguir vemos algumas imagens da janela criada pela classe *ActionDemo*:




## 4.7. Anonymous inner class

*Anonymous inner class* são classes internas que não são declaradas. O uso de *anonymous inner class* simplifica a classe. A seguir temos o mesmo exemplo apresentado anteriormente modificado para utilizar este recurso.

## 4.8. Exemplo do Evento Action

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class ActionDemo extends JFrame {
    private JTextField tf;
    private JButton bt;
    public ActionDemo(String title){
        super(title);
        tf = new JTextField();
        bt = new JButton("Clicked");
        add(tf, BorderLayout.SOUTH);
        add(bt, BorderLayout.NORTH);
        setSize(300,300);
        bt.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent actionEvent) {
                method();
            }
        });
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public void method() {
        if (tf.getText().equals("Action"))
            tf.setText("");
        else
            tf.setText("Action");
    }
    public static void main(String args[]) {
        new ActionDemo("Action Demo");
    }
}
```

Observe que a classe se encontra de forma mais organizada, criamos um objeto em tempo de execução para o evento, este fará a chamada um método, chamado *method*, que se encarregará de executar as instruções.

## 4.9. Exemplo de Eventos do Mouse

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class MouseEventsDemo extends JFrame {
    private TextField tf;
    public MouseEventsDemo(String title){
        super(title);
        tf = new TextField(60);
        add(tf, BorderLayout.SOUTH);
        setSize(300,300);
        this.addMouseListener(new MouseListener() {
            public void mouseClicked(MouseEvent mouseEvent) {
                eventMouse1(mouseEvent);
            }
            public void mouseEntered(MouseEvent mouseEvent) {
                eventMouse2(mouseEvent);
            }
            public void mouseExited(MouseEvent mouseEvent) {
                eventMouse3(mouseEvent);
```

```
                    }
                    public void mousePressed(MouseEvent mouseEvent) {
                        eventMouse4(mouseEvent);
                    }
                    public void mouseReleased(MouseEvent mouseEvent) {
                        eventMouse5(mouseEvent);
                    }
                });
                this.addMouseMotionListener(new MouseMotionListener() {
                    public void mouseDragged(MouseEvent mouseEvent) {
                        eventMouse6(mouseEvent);
                    }
                    public void mouseMoved(MouseEvent mouseEvent) {
                        eventMouse7(mouseEvent);
                    }
                });
                this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                setVisible(true);
            }
            public void eventMouse1(MouseEvent me) {
                tf.setText("Mouse clicked.");
            }
            public void eventMouse2(MouseEvent me) {
                tf.setText("Mouse entered component.");
            }
            public void eventMouse3(MouseEvent me) {
                tf.setText("Mouse exited component.");
            }
            public void eventMouse4(MouseEvent me) {
                tf.setText("Mouse pressed.");
            }
            public void eventMouse5(MouseEvent me) {
                tf.setText("Mouse released.");
            }
            public void eventMouse6(MouseEvent me) {
                tf.setText("Mouse dragged at " + me.getX() + "," + me.getY());
            }
            public void eventMouse7(MouseEvent me) {
                tf.setText("Mouse moved at " + me.getX() + "," + me.getY());
            }
            public static void main(String args[]) {
                new MouseEventsDemo("Mouse Events Demo");
            }
        }
```

A seguir vemos algumas imagens da janela criada pela classe *MouseEventsDemo*:

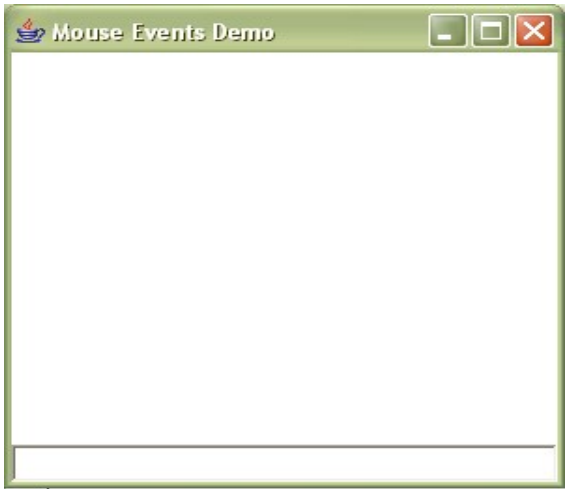



*Figura 2: Executando MouseEventsDemo*

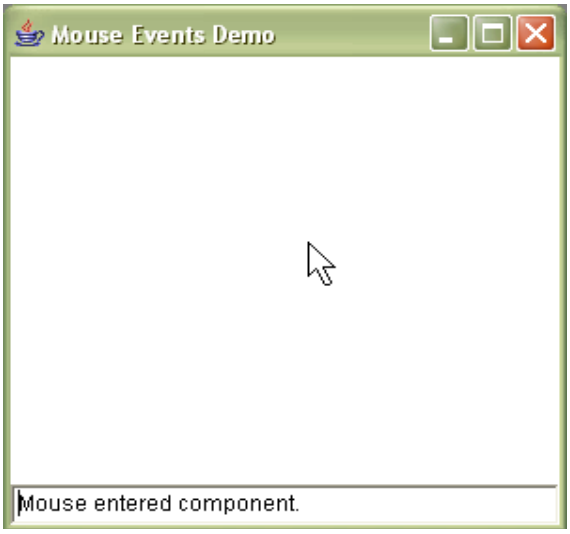



*Figura 3: Executando MouseEventsDemo: Mouse "entra" na janela*

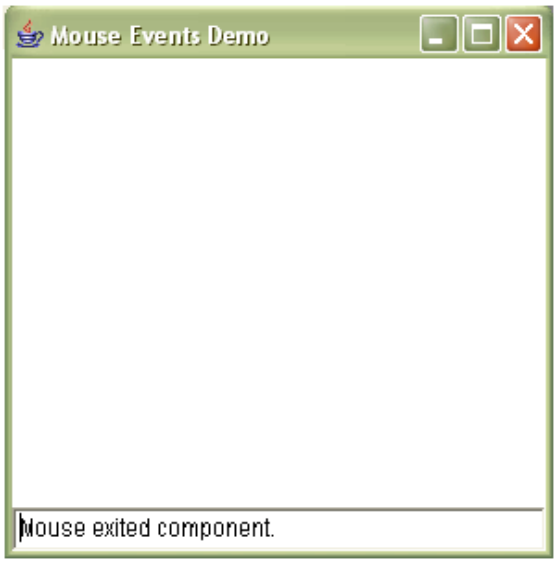



*Figura 4: Executando MouseEventsDemo: Mouse sai da janela*

## 4.10. Exemplo Close Window

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class WindowDemo extends JFrame {
    public WindowDemo(String title) {
        super(title);
        this.setSize(300,300);
        this.addWindowListener(new WindowListener() {
            public void windowActivated(WindowEvent windowEvent) {
            }
            public void windowClosed(WindowEvent windowEvent) {
            }
            public void windowClosing(WindowEvent windowEvent) {
                closed();
            }
            public void windowDeactivated(WindowEvent windowEvent) {
            }
            public void windowDeiconified(WindowEvent windowEvent) {
            }
            public void windowIconified(WindowEvent windowEvent) {
            }
            public void windowOpened(WindowEvent windowEvent) {
            }
        });
        this.setVisible(true);
    }
    public void closed() {
        System.exit(0);
    }
    public static void main(String args[]) {
        new WindowDemo("Close Window Example");
    }
}
```

Executando esta classe, será exibida a janela abaixo:


*Figura 5: Executando CloseFrame*

# 5. Classes *Adapter*

Implementar todos os métodos de uma interface é muito trabalhoso. Freqüentemente necessitaremos implementar somente alguns métodos da interface. Felizmente, Java nos oferece as classes *adapter* que implementam todos os métodos das interfaces *listener* que possuem mais de um método. As implementações dos métodos são vazias.

## *5.1. Exemplo Close Window*

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class WindowDemo extends JFrame {
    private JLabel label;
    public WindowDemo(String title) {
        super(title);
        this.setSize(300,300);
        label = new JLabel("Close the frame.");
        this.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                closed();
            }
        });
        this.setVisible(true);
    }
    public void closed() {
        System.exit(0);
    }
    public static void main(String args[]) {
        new WindowDemo("Close Window Example");
    }
}
```

Comparando com o exemplo anterior, repare que não existe mais a necessidade de implementar os outros métodos do *listener* deixando-os em branco, implementamos apenas os métodos que realmente iremos utilizar.

# 6. Inner Class

Nesta seção faremos uma revisão de conceitos que já foram abordados no primeiro módulo do curso. *Inner class* é muito útil no tratamento de eventos em interfaces gráficas.

## 6.1. Inner Class

Aqui está uma breve revisão sobre *inner class*. Uma *inner class*, como o nome já diz, é uma classe declarada dentro de outra classe. O uso de uma *inner class* pode auxiliar na simplificação de suas classes, especialmente quanto ao tratamento de eventos, como será mostrado no exemplo a seguir:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class WindowDemo extends JFrame {
    private JLabel label;
    public WindowDemo(String title) {
        super(title);
        this.setSize(300,300);
        label = new JLabel("Close the frame.");
        this.addWindowListener(new CFListener());
        this.setVisible(true);
    }
    private class CFListener extends WindowAdapter {
        public void windowClosing(WindowEvent e) {
            closed();
        }
    }
    public void closed() {
        System.exit(0);
    }
    public static void main(String args[]) {
        new WindowDemo("Close Window Example");
    }
}
```

# Parceiros que tornaram JEDI™ possível


***Instituto CTS***
Patrocinador do DFJUG.

***Sun Microsystems***
Fornecimento de servidor de dados para o armazenamento dos vídeo-aulas.

***Java Research and Development Center da Universidade das Filipinas***
Criador da Iniciativa JEDI™.

***DFJUG***
Detentor dos direitos do JEDI™ nos países de língua portuguesa.

***Banco do Brasil***
Disponibilização de seus *telecentros* para abrigar e difundir a Iniciativa JEDI™.

***Politec***
Suporte e apoio financeiro e logístico a todo o processo.

***Borland***
Apoio internacional para que possamos alcançar os outros países de língua portuguesa.

***Instituto Gaudium/CNBB***
Fornecimento da sua infra-estrutura de hardware de seus servidores para que os milhares de alunos possam acessar o material do curso simultaneamente.