



# INF1636 – PROGRAMAÇÃO ORIENTADA A OBJETOS

Departamento de Informática – PUC-Rio

Ivan Mathias Filho

[ivan@inf.puc-rio.br](mailto:ivan@inf.puc-rio.br)

## Programa – Capítulo 12



- Orientação a Eventos
- Tratadores de Eventos
- Adaptadores

Laboratório de Engenharia de Software

## Programa – Capítulo 12




- **Orientação a Eventos**
- Tratadores de Eventos
- Adaptadores

© LES/PUC-Rio

Laboratório de Engenharia de Software

## Orientação a eventos




- O funcionamento de um sistema gráfico se dá por meio de ações sobre diversos elementos da interface;
- A cada ação correspondem um ou mais eventos, enviados para a fonte geradora dos mesmos (componentes visuais);
- Estes eventos devem ser identificados e tratados de maneira adequada pelo programa de aplicação.

© LES/PUC-Rio

Laboratório de Engenharia de Software

## Mecanismo de callback




- Para que um programa possa tratar um evento gerado na interface com o usuário, ele deve registrar uma função com esse objetivo;
- Esse mecanismo é chamado de callback, pois a função registrada será chamada de volta pelo runtime da aplicação;
- O uso de callbacks é não exclusivo de linguagens orientadas a objetos, pois funções ordinárias, encontradas em praticamente todas as linguagens de programação, podem ser usadas para tal;
- Dessa forma, como uma linguagem orientada a objetos pura, como Java, implementa callbacks, uma vez que nesse tipo de linguagem as funções são definidas no escopo de alguma classe (métodos)?

Laboratório de Engenharia de Software

© LES/PUC-Rio

Laboratório de Engenharia de Software

## Callback – Exemplo Python



```

from tkinter import *

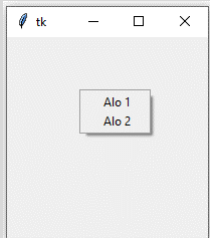
def alo():
    print('Alo!')

def popup(e):
    menu.post(e.x_root, e.y_root)

root = Tk()
menu = Menu(root, tearoff=0)
menu.add_command(label='Alo 1', command=alo)
menu.add_command(label='Alo 2', command=alo)

frame = Frame(root, width=200, height=200)
frame.pack()
frame.bind('<Button-3>', popup)

root.mainloop()
```



Laboratório de Engenharia de Software

© LES/PUC-Rio

6

## Callbacks em Java



- Como Java é uma linguagem orientada a objetos pura, o mecanismo de callback deve ser implementado por meio de objetos;
- Um objeto que implemente um callback é chamado de listener.


## Programa – Capítulo 12



- Orientação a Eventos
- **Tratadores de Eventos**
- Adaptadores

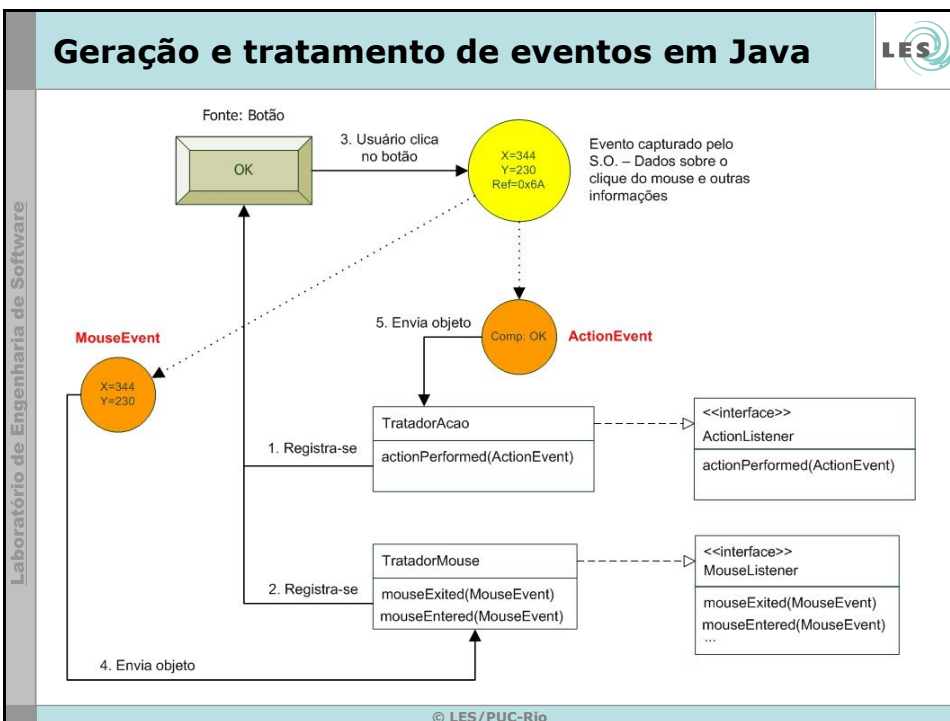
Laboratório de Engenharia de Software

## Listeners e eventos



- Em Java, um listener é um objeto responsável pelo tratamento de um evento;
- Um listener é um objeto que implementa uma determinada interface e, por conseguinte, pode tratar vários eventos;
- Quando um listener tem um de seus métodos chamados, ele recebe um parâmetro que descreve o evento ocorrido;
- A API nativa do Java fornece várias classes para representar diferentes tipos de eventos.

© LES/PUC-Rio



Laboratório de Engenharia de Software

## Eventos de interface gráfica



- Descendentes de `java.awt.event.AWTEvent`;
- Divididos em categorias (pacote `java.awt.event`):
  - `ActionEvent` (fonte: componentes de ação);
  - `MouseEvent` (fonte: componentes afetados pelo mouse);
  - `ItemEvent` (fonte: *checkboxes* e similares);
  - `AdjustmentEvent` (fonte: *scrollbars*);
  - `TextEvent` (fonte: componentes de texto);
  - `WindowEvent` (fonte: janelas);
  - `FocusEvent` (fonte: componentes em geral);
  - `KeyEvent` (fonte: componentes afetados pelo teclado).

© LES/PUC-Rio

Laboratório de Engenharia de Software

## Listeners




- Cada evento tem uma interface listener correspondente, que declara as operações usadas para tratá-lo:
  - `ActionEvent`: `ActionListener`;
  - `MouseEvent`: `MouseListener` e `MouseMotionListener`;
  - `ItemEvent`: `ItemListener`;
  - `AdjustmentEvent`: `AdjustmentListener`;
  - `TextEvent`: `TextListener`;
  - `WindowEvent`: `WindowListener`;
  - `FocusEvent`: `FocusListener`;
  - `KeyEvent`: `KeyListener`.

© LES/PUC-Rio

Laboratório de Engenharia de Software

## Implementação de um listener




- Para implementar um listener deve-se seguir os seguintes passos:
  - Definir a classe (nova ou já existente) que irá implementar a interface do *listener* adequado;
  - Implementar cada uma das operações declaradas na interface;
  - Instanciar um objeto da classe definida no primeiro passo do processo;
  - Registrar o objeto junto à fonte geradora de eventos.

© LES/PUC-Rio

Laboratório de Engenharia de Software

## Listener – Exemplo



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

public class TratadorInclusao implements ActionListener {
    Component c;


    public TratadorInclusao(Component x) {
        c=x;
    }

    public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(c,"Inclusão Efetuada");
    }
}
```

© LES/PUC-Rio

Laboratório de Engenharia de Software

## Ligação do listener à fonte (1)




- Na ocorrência de um evento, todos os listeners registrados junto à fonte geradora são notificados;
- Para isso, é preciso registrar os listeners junto ao componente gerador de eventos (fonte):

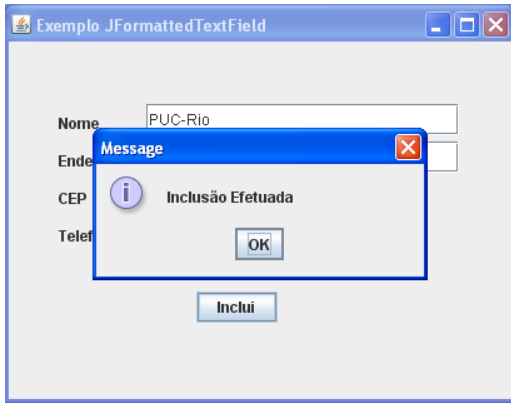
```
 JButton inc=new JButton("Inclui");  
 inc.addActionListener(new TratadorInclusao(this));
```
- O objeto gerador de eventos também pode ser o tratador dos seus eventos, desde que a sua classe implemente as interfaces adequadas.

© LES/PUC-Rio

Laboratório de Engenharia de Software

## Ligação do listener à fonte (2)





The screenshot shows a Java Swing application window titled "Exemplo JFormattedTextField". Inside the window, there are four labels: "Nome", "Ende", "CEP", and "Telef", each followed by a text input field. The "Nome" field contains the text "PUC-Rio". A "Message" dialog box is overlaid on the window, displaying an information icon and the text "Inclusão Efetuada" with an "OK" button. Below the input fields, there is a button labeled "Inclui".

© LES/PUC-Rio



Alguns eventos, listeners e operações			LES
ActionEvent	ActionListener	actionPerformed(ActionEvent)	Laboratório de Engenharia de Software
ItemEvent	ItemListener	itemStateChanged(ItemEvent)	
KeyEvent	KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)	
MouseEvent	MouseListener	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)	
	MouseMotionListener	mouseDragged(MouseEvent) mouseMoved(MouseEvent)	
TextEvent	TextListener	textValueChanged(TextEvent)	
WindowEvent	WindowListener	windowActivated(WindowEvent) windowClosed(WindowEvent) windowClosing(WindowEvent) windowDeactivated(WindowEvent) windowDeiconified(WindowEvent) windowIconified(WindowEvent) windowOpened(WindowEvent)	
© LES/PUC-Rio			

# Listeners – Operações declaradas (1)

The logo for LES (Laboratório de Engenharia de Software) at PUC-Rio, featuring the letters 'LES' in a bold, sans-serif font next to a circular graphic with concentric arcs.


- Interface ActionListener
  - `void actionPerformed(ActionEvent e)` – acionado quando uma ação ocorre.
- Interface AdjustmentListener
  - `void adjustmentValueChanged(AdjustmentEvent e)` – acionado quando houver mudança no estado de um componente ajustável.
- Interface ComponentListener
  - `void componentHidden(ComponentEvent e)` – acionado quando um componente se torna invisível;

© LES/PUC-Rio

Laboratório de Engenharia de Software

Laboratório de Engenharia de Software

## Listeners – Operações declaradas (2)




- Interface ComponentListener (cont.)
  - `void componentMoved(ComponentEvent e)` – acionado quando a posição de um componente for alterada;
  - `void componentResized(ComponentEvent e)` – acionado quando as dimensões de um componente forem alteradas;
  - `void componentShown(ComponentEvent e)` – acionado quando um componente se torna visível.
- Interface ItemListener
  - `void itemStateChanged(ItemEvent e)` – acionado quando um componente é selecionado ou desmarcado pelo usuário.

© LES/PUC-Rio

Laboratório de Engenharia de Software

## Listeners – Operações declaradas (3)




- Interface FocusListener
  - `void focusGained(FocusEvent e)` – acionado quando um componente ganha o foco do teclado;
  - `void focusLost(FocusEvent e)` – acionado quando um componente perde o foco do teclado.
- Interface TextListener
  - `void textValueChanged(TextEvent e)` – acionado quando o valor de um texto for alterado.

© LES/PUC-Rio

Laboratório de Engenharia de Software

## Listeners – Operações declaradas (4)




- Interface `KeyListener`
  - `void keyPressed(KeyEvent e)` – acionado quando uma tecla for pressionada;
  - `void keyReleased(KeyEvent e)` – acionado quando uma tecla for liberada;
  - `void keyTyped(KeyEvent e)` – acionado quando uma tecla for digitada.
- Interface `WindowListener`
  - `void windowActivated(WindowEvent e)` – acionado quando um janela for ativada;

© LES/PUC-Rio

Laboratório de Engenharia de Software

## Listeners – Operações declaradas (5)




- Interface `WindowListener` (cont.)
  - `void windowClosed(WindowEvent e)` – acionado quando um janela for fechada através de uma chamada ao método `dispose()`;
  - `void windowClosing(WindowEvent e)` – acionado quando o usuário tentar fechar uma janela através do *system menu*;
  - `void windowDeactivated(WindowEvent e)` – acionado quando um janela for mais a janela ativa;
  - `void windowIconified(WindowEvent e)` – acionado quando uma janela for minimizada;
  - `void windowOpened(WindowEvent e)` – acionado na primeira vez que uma janela se tornar visível.

© LES/PUC-Rio

Laboratório de Engenharia de Software

## Programa – Capítulo 12




- Orientação a Eventos
- Tratadores de Eventos
- **Adaptadores**

© LES/PUC-Rio

Laboratório de Engenharia de Software

## Adaptadores (1)



- Em algumas situações pode ser muito cansativo implementar todas as operações declaradas em interfaces relativas a listeners;
- Para evitar trabalho desnecessário, alguns listeners possuem uma classe adaptadora correspondente;
- A classe adaptadora possui um método com implementação vazia (`{ }`) para cada operação declarada na interface;
- Dessa forma, pode-se criar uma subclasse do adaptador e sobrescrever apenas os métodos correspondentes aos eventos que interessam.

© LES/PUC-Rio

## Adaptadores (2)



- **public abstract class** MouseAdapter **extends** Object
  - Implements: MouseListener, MouseWheelListener, MouseMotionListener;
- **public abstract class** KeyAdapter **extends** Object
  - Implements: KeyListener;
- **public abstract class** WindowAdapter **extends** Object
  - Implements: WindowListener, WindowStateListener, WindowFocusListener.

© LES/PUC-Rio

## Classe Anônima (1)



- Favorece a escrita de código mais conciso;
- Permite declarar e instanciar uma classe ao mesmo tempo;
- Deve ser empregada quando não se quiser criar uma classe completa apenas para tratar um evento;
- No exemplo abaixo, uma classe completa foi criada apenas para tratar o evento de clique sobre um botão:

```
public class TratadorInclusao implements ActionListener {
    Component c;
    public TratadorInclusao(Component x) {
        c=x;
    }
    public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(c,"Inclusão Efetuada");
    }
}
```

© LES/PUC-Rio

- Essa classe poderia ser substituída por uma classe anônima, instanciada no momento do registro do tratador de evento junto a botão:

```
inc.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        JOptionPane.showMessageDialog(inc, "Inclusão Efetuada");  
    }  
});
```