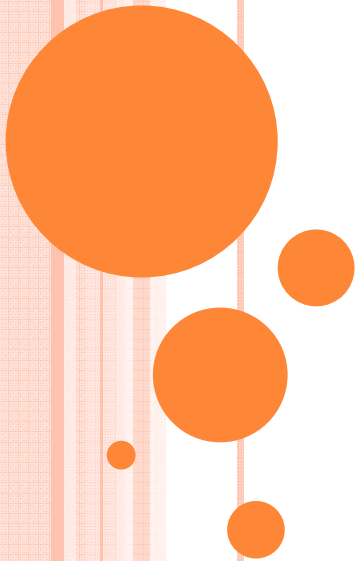
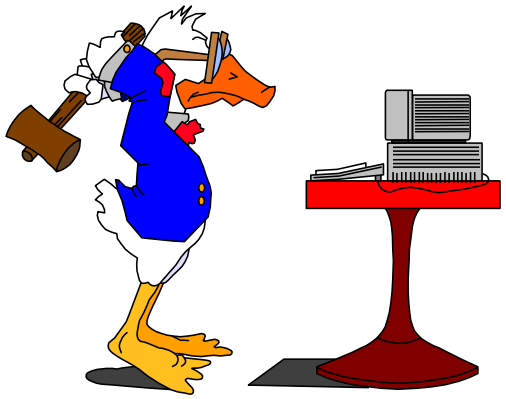


Tratamento de Eventos em Java usando Swing

George Gomes Cabral

Incrementando nossas interfaces

- Vamos ver mais componentes que vão nos permitir uma maior funcionalidade.
- Depois, vamos melhorar a maneira como tratamos os eventos.
- Assim, seremos capazes de fazer melhores interfaces, que não fazem o usuário querer destruir o micro!



Nossas telas e os eventos

- Sem eventos, nossas telas não fazem absolutamente nada - precisamos dar vida a elas, acrescentando a capacidade de lidar com eventos...
- Na programação orientada a eventos a ordem de execução de um programa é indeterminada.
- Os componentes da tela causam eventos quando interagem com o usuário ou com o SO.
- Estes eventos devem ser tratados, fazendo com que o programa responda e dirigindo a execução através dos tratadores de eventos.

Eventos

- Cada vez que o usuário bate numa tecla, movimenta o mouse ou aperta um botão deste, um evento ocorre.

- A programação orientada a eventos (*Event-driven programming*) consiste em fazer programas cuja ordem de execução seja determina pela ocorrência destes eventos.

- Todo objeto pode ser notificado por um evento
  - Basta implementar a interface apropriada e então registrar estes métodos como um event listener do evento apropriado.

Programação Orientada a Eventos

- Na programação orientada a eventos, nós precisamos de um loop eterno que fica eternamente esperando por uma entrada do usuário.
- Isto é feito no Java sem nosso conhecimento...
- Só pelo fato de uma GUI estar ativa, este pseudo-loop está rodando.
- Quando um evento ocorre, o gerenciador de janelas (*window manager*) cria um evento e passa para um tratador de eventos definido pelo programador.
- Este processo é chamado de callback.
- No final das contas, nosso trabalho é definir os tratadores de eventos...

Eventos

- Os componentes Swing podem gerar vários tipos de eventos diferentes.

- Alguns exemplos:
  - Usuário clica em um botão ⇨ ActionListener
  - Usuário fecha um frame ⇨ WindowListener
  - Usuário pressiona um botão do mouse ⇨ MouseListener
  - Usuário move o mouse ⇨ MouseMotionListener
  - Componentes se tornam visíveis ⇨ ComponentListener

Entendendo Eventos

- Cada evento é representado por um objeto que contém informações sobre este evento.
- Event source: é quem gera o evento, normalmente um componente da GUI
- Event listener
  - Responde ao evento.
  - Pode ser qualquer classe Java que implemente a interface correta.
  - Uma única fonte pode ter múltiplos *listeners*.
  - Um único *listener* pode responder a múltiplas sources.

Básico do tratamento de Eventos em Java

- Escreva uma classe que implemente a interface associada ao evento a ser tratada.
- Normalmente, esta interface tem o nome do formato `SomethingListener`.
- Exemplo: tratadores de eventos de janelas são `WindowListener`, e os de mouse, `MouseListener`
- Crie um objeto da classe que você definiu.
- Registre o objeto que você criou como tratador de eventos de um determinado objeto usando o método apropriado.
- Normalmente, este método tem o formato `addSomethingListener(listener)`

Hierarquia de Eventos em java

```
java.lang.Object
  +--java.util.EventObject
        +--java.awt.AWTEvent
              +--java.awt.event.ActionEvent
              +--java.awt.event.TextEvent
              +--java.awt.event.ComponentEvent
                    +--java.awt.event.FocusEvent
                    +--java.awt.event.WindowEvent
                    +--java.awt.event.InputEvent
                          +--java.awt.event.KeyEvent
                          +--java.awt.event.MouseEvent
```

- Precisamos colocar no nosso cabeçalho a statement **`import java.awt.event.*;`**

ActionEvents

- Tipo de evento mais simples e comum no Swing
- Representa um ação qualquer ocorrendo em um componente da GUI
- Criado por:
  - cliques em botão
  - mudanças em checkboxes
  - cliques de menu
  - digitar [Enter] em uma textbox
  - etc.

Esperando por ActionEvents

- Acrescente um *listener* ao componente
- O método listener apropriado será chamado quando o evento ocorrer (pro exemplo, quando o botão for clicado)
- Para eventos de ação, use a classe `ActionListener`

Básico para ouvir Eventos

- Para fazermos um tratador de eventos precisamos de três ações básicas

  ① Especificar uma classe que implemente uma interface de listener:

```
public class MyClass implements ActionListener {
```

  ② Código que implemente métodos dentro da interface listener

```
public void actionPerformed(ActionEvent e) {
    ...//code that responds to the event... }
```

  ③ Executar um código que registre uma instância desta classes como um listener de um ou mais componentes

```
someComponent.addActionListener(instanceOfMyClass)
  ;
```

Escrevendo um actionListener

- Precisamos implementar a interface `ActionListener`, que especifica o método `actionPerformed` que deve ser implementado em nossa classe.

- Não interessa se sua classe extender alguma outra classe.

- O código básico é o seguinte:

```
// Prints a message when the button is clicked.
public class MyActionListener implements
  ActionListener {
   public void actionPerformed(ActionEvent event) {
     System.out.println("Event occurred!");
   }
}
```

Escrevendo um actionListener

- Agora precisamos associar nosso `ActionListener` ao elemento gráfico cujo evento de ação nós trataremos.

```
JButton button = new JButton(“button 1”);
MyActionListener listener = new
  MyActionListener();
button.addActionListener(listener);
```

- Agora, quando o botão `button` for clicado, veremos a mensagem “`Event occurred!`” impressa
- O método `addActionListener` existe em vários componentes (basicamente todos aqueles que podem ter um evento de ação associado).

Onde Colocar um ActionListener

- Podemos colocar em uma classe interna

```
public class Outer {
  private class Inner implements ActionListener {
    public void actionPerformed(ActionEvent event)
    {
      ...
    }
  }

  public Outer() {
    JButton myButton = new JButton();
    myButton.addActionListener(new Inner());
  }
}
```

Onde Colocar um ActionListener

- Podemos colocar em uma classe interna anônima

```
public class Outer {
  public Outer() {
    JButton myButton = new JButton();
    myButton.addActionListener(
      new ActionListener() {
        public void actionPerformed(ActionEvent e) {
          ...
        }
      }
    );
  }
}
```

Exemplo

```
public class GUIDemo extends JFrame implements
  ActionListener{
  protected int numCl=0;
  protected JButton meuJButton = null;
  protected JLabel meuJLabel = null;

  public GUIDemo(String title) {
         super(title);
         initialize();
  }

  public GUIDemo() {this("Demonstrando eventos");}
```

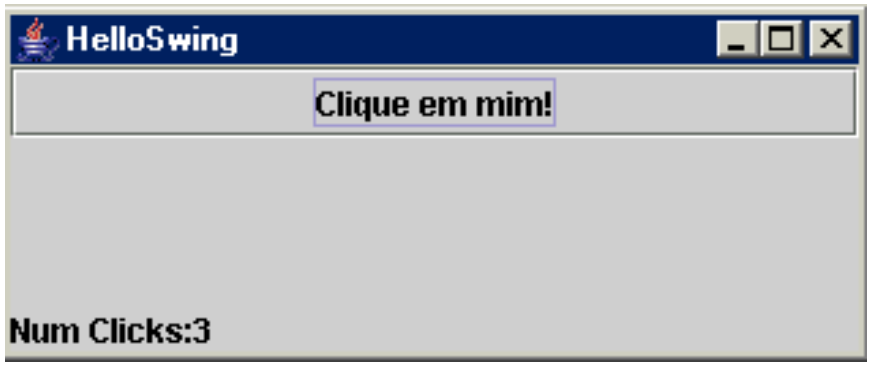
Exemplo (cont.)

```
protected void initialize() {
     this.setSize( new Dimension(200,75) );
     meuJButton = new JButton("Clique em mim!");
     meuJButton.addActionListener(this);
     meuJLabel = new JLabel("Num. clicks="+numCl);
     this.getContentPane().add(meuJButton,
                               BorderLayout.NORTH);
     this.getContentPane().add(meuJLabel,
                               BorderLayout.SOUTH);

     this.setDefaultCloseOperation(JFrame.EXIT_ON_CLO
SE);
     this.pack();
     this.setVisible(true);
}
```

Exemplo (cont.)

```
  public void actionPerformed(ActionEvent e) {
        numCl++;
        meuJLabel.setText("Num Clicks:"+numCl);
  }
}
```

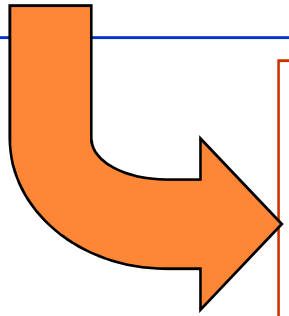


Eventos de Mouse

- Motivação:
  - Capturar os clicks e os movimentos do mouse dentro de um componente de uma GUI.
  - Responder a atividade de mouse com ações apropriadas.
  - Criar programas interativos que são dirigidos pela atividade do mouse.

A INTERFACE MOUSELISTENER

```
package java.awt.event;
public interface MouseListener {
  public void mouseClicked(MouseEvent event);
  public void mouseEntered(MouseEvent event);
  public void mouseExited(MouseEvent event);
  public void mousePressed(MouseEvent event);
  public void mouseReleased(MouseEvent event);
}
```



```
public class ML implements MouseListener {
  public void mouseClicked(MouseEvent e) {}
  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e)  {}
  public void mousePressed(MouseEvent e) {
    System.out.println("Pressionou!");  }
  public void mouseReleased(MouseEvent e) {}
}
```

USANDO MOUSELISTENER

```
// Dado um panel qualquer
MyPanel panel = new MyPanel();
panel.addMouseListener(new
  MyMouseListener());
```

- Problemas:
  - Temos que implementar toda a interface
  - Isto é tedioso, especialmente se só queremos usar um único método, como no exemplo.

MouseAdapter

- É uma classe com implementações vazias de todos os métodos da interface `MouseListener`
- Para usar, extenda a classe `MouseAdapter` e sobreescreva os métodos que lhe interessam.
- Evita a necessidade de implementar um monte de métodos vazios que não nos interessam.
- Exemplo:

```
public class MyMouseAdapter extends MouseAdapter {
  public void mousePressed(MouseEvent event) {
    System.out.println("User pressed mouse button!");
  }
}
// usando a classe que definimos (MyMouseAdapter)
MyPanel panel = new MyPanel();
panel.addMouseListener(new MyMouseAdapter());
```

Objetos MouseEvent

- Repare que todos os métodos de MouseAdapter recebem um parâmetro da classe MouseEvent.

- Esta é uma classe pré-definida para que possamos saber de onde o evento que diparou o método foi proveniente e certas circunstâncias modificadoras associadas a ele

- Constantes em **`InputEvent`** (mãe de MouseEvent)
  - `public static int` **`BUTTON1_MASK, BUTTON2_MASK, BUTTON3_MASK, CTRL_MASK, ALT_MASK, SHIFT_MASK`**

USANDO O MOUSEEVENT

- Métodos interessantes em `MouseEvent`
  - `public int getClickCount()`
  - `public Point` **`getPoint`**`()`
  - `public int` **`getX`**`(),` **`getY`**`()`
  - `public Object` **`getSource`**`()`
  - `public int` **`getModifiers`**`()`
- Exemplo de uso:

```
public class MyMouseAdapter extends MouseAdapter {
  public void mousePressed(MouseEvent event) {
    Point p = event.getPoint();
    Object source = event.getSource();
    if (source == myPanel  &&  p.getX() < 10)
      JOptionPane.showMessageDialog(null, “Lado
  esquerdo!”);
  }
}
```

Capturando movimento do mouse: MouseMotionListener

```
package java.awt.event;
public interface MouseMotionListener {
   public void mouseDragged(MouseEvent
  event);
   public void mouseMoved(MouseEvent event);
}
```

- A classe abstrata `MouseMotionAdapter` provê uma implementação vazia de ambos os métodos para que possamos sobreescrevê-los.
- A idéia é a mesma da classe `MouseAdapter` com a interface `MouseListener`

EXEMPLO MOUSEMOTIONADAPTER

```
public class MyAdapter extends
  MouseMotionAdapter {
   public void mouseMoved(MouseEvent event)
   {
      Point p = event.getPoint();
      int x   = event.getX();
      int y   = event.getY();
      System.out.println("Mouse is at " + p);
      System.out.println("x is " + x);
      System.out.println("y is " + y);
   }
}

// usando o método
myPanel.addMouseMotionListener(new
  MyAdapter());
```

MouseInputListener

- A interface MouseInputListener extende tanto a interface MouseListener quanto a interface MouseMotionListener
- Código:

```
package javax.swing.event;
public interface MouseInputListener extends
     MouseListener, MouseMotionListener {}
```

- Logo, se você quiser usar as duas, você pode implementar apenas `MouseInputListener`.
- Assim como nos casos anteriores, existe uma classe Adapter que implementa versões vazias de todos os métodos desta interface.
- Neste caso a classe a ser extendida é `MouseInputAdapter`

Exemplo de MouseInputAdapter

```
public class MyMouseInputAdapter extends
  MouseInputAdapter {
  public void mousePressed(MouseEvent event) {
    System.out.println("Mouse was pressed");
  }

  public void mouseMoved(MouseEvent event) {
    Point p = event.getPoint();
    System.out.println("Mouse is at " + p);
  }
}

// using the listener
MyMouseInputAdapter adapter = new
  MyMouseInputAdapter();
myPanel.addMouseListener(adapter);
myPanel.addMouseMotionListener(adapter);
```

Eventos de Teclado

- São usados para ouvir atividade de teclado dentro de um componente UI (geralmente um panel)
- Com eles respondemos a atividade de teclado com as ações apropriadas.

A INTERFACE KEYLISTENER

- A interface KeyListener deve ser implementada para ouvirmos entradas do teclado.
- Código:

```
package java.awt.event;
public interface KeyListener {
  public void keyPressed(KeyEvent event);
  public void keyReleased(KeyEvent event);
  public void keyTyped(KeyEvent event);
}
```

- Assim como nos casos anteriores, existe uma classe Adapter que implementa versões vazias de todos os métodos desta interface.
- Neste caso a classe a ser extendida é KeyAdapter

A CLASSE KEYEVENT

- Objetos da classe KeyEvent são enviados para nossos tratadores de eventos de teclado.
- **InputEvent**
  - ⇨ public static int **CTRL_MASK**, **ALT_MASK**, **SHIFT_MASK**

- **KeyEvent (descendente da InputEvent)**
  - ⇨ public static int VK_A .. VK_Z, VK_0 .. VK_9, VK_F1 .. VK_F10,VK_UP, VK_LEFT, ..,VK_TAB, VK_SPACE,VK_ENTER,... *(um para cada tecla)*
  - ⇨ public char **getKeyChar**()
  - ⇨ public int **getKeyCode**()
  - ⇨ public Object **getSource**()
  - ⇨ public int **getModifiers**() *(use as máscaras definidas em InputEvent)*

EXEMPLO DE KEYADAPTER

```
class PacManKeyListener extends KeyAdapter {
  public void keyPressed(KeyEvent event) {
    char keyChar = event.getKeyChar();
    int keyCode  = event.getKeyCode();

    if (keyCode == KeyEvent.VK_RIGHT) {
      pacman.setX(pacman.getX() + 1);
      pacpanel.repaint();
    } else if (keyChar == 'Q')
      System.exit(0);
  }
}

PacPanel panel = new PacPanel();
panel.addKeyListener(new PacKeyListener());
```

Quer dizer: se a tecla da seta à direita foi pressionada...

Eventos de Janelas

- Os eventos de janela são tratados por classes que implementem a interface `WindowListener`.

- Definição:

```
public interface WindowListener {
    public void windowClosing(WindowEvent e)
    public void windowClosed(WindowEvent e)
    public void windowOpened(WindowEvent e)
    public void windowIconified(Window Event e)
    public void windowDeIconified(Window Event e)
    public void windowActivated(Window Event e)
    public void windowDeactivated(Window Event e)
}
```

- Como seria de se esperar, existe uma classe chamada `WindowAdapter` que tem uma implementação vazia de cada um destes métodos.