Linguagem de Programação Orientada a Objetos II

Marcos Lapa dos Santos
marcoslapa@gmail.com

Estácio | FIB

Tratamento de Eventos em Java

Estácio | FIB

Tratamento de Eventos

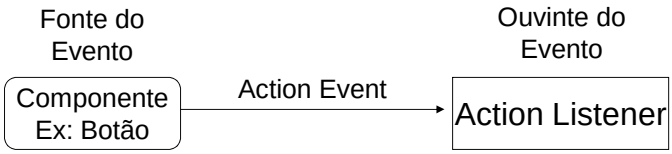
- Toda ação de um usuário (um clique em um botão, um caracter digitado) causa um evento no ambiente operacional.
- Qualquer objeto pode ser notificado de que um evento ocorreu.
- O esquema de tratamento de eventos contará sempre com o conceito de *Event Source* e *Event Listener*

Tratamento de Eventos

- O modelo de eventos da linguagem java atual é diferente do usado na versão 1.0 do java.
- Qualquer sistema operacional com suporte a interfaces de usuário gráficas – GUIs (exemplos windows, linux, mac os) – precisa monitorar constantemente o ambiente buscando por eventos.
- Exemplos de Eventos: teclas pressionadas, cliques de mouse.

Tratamento de Eventos

- Modelo de Tratamento de eventos:



Tratamento de Eventos

- O SO fica encarregado de informar esses eventos aos programas que estão em execução.
- Cada programa decide então o que fazer em resposta a esses eventos.

POO

Tratamento de Eventos

- Para tratar eventos de componentes gráficos são necessários dois passos :
  - **Registrar** um event listener e **implementar** um event handler.
  - Um **event listener** é um objeto de uma classe que implementa uma ou mais interfaces event listener do pacote java.awt.event ou do pacote javax.swing.event.
  - Um **event handler** é um método chamado automaticamente em resposta a um evento.

Estácio | FIB

POO

Tratamento de Eventos

- Todo evento tem um objeto que é sua fonte.
- Métodos de ouvintes (listeners) que desejam tratar eventos, recebem eventos como argumento.
- Ouvintes precisam ser registrados nas fontes
  - Quando ocorre um evento, um método de todos os ouvintes registrados é chamado e o evento é passado como argumento.

Estácio | FIB

Tratamento de Eventos

- // método ouvinte registrado e o evento passado como argumento

```
jButtonNovo.addActionListener(
    new    java.awt.event.ActionListener() {
           public void actionPerformed(ActionEvent e) {
                  jButtonNovo_actionPerformed(e);
           }
    }
);
```


Tratamento de Eventos

- // implementação da resposta ao evento

```
void jButtonNovo_actionPerformed(ActionEvent e) {
    ...
}
```

Exercício

- Crie uma tela (JFrame) contendo um JPanel e 3 botões com os seguintes textos (“Azul”, “Amarelo” e “Verde”).

- O comportamento da aplicação deverá ser o seguinte:
  - Ao clicar em cada botão, o JPanel terá sua cor de fundo modificada de acordo com o botão.
  - Uma caixa de diálogo (JOptionPane) com o seguinte texto: “Deseja manter a nova cor? (Sim) (Não)” deverá ser exibida em seguida.
  - Caso o usuário clique na opção **Sim** a cor se manterá, caso seja **Não** a cor anterior deverá ser restaurada.

- **Dicas**:
  - use o método setBackGround(Color.NOME_DA_COR) do JPanel para modificar sua cor adequadamente.
  - Analogamente, para capturar a cor corrente do JPanel utilize o seu método getBackground()

Hierarquia de eventos do AWT

- Todos eventos herdam da classe EventObject do pacote java.util.

- Lista com alguns tipos de eventos AWT:
  - ActionEvent
  - AdjustmentEvent
  - ComponentEvent
  - ContainerEvent
  - FocusEvent
  - ItemEvent
  - KeyEvent
  - MouseEvent
  - MouseWheelEvent
  - TextEvent
  - WindowEvent

Interfaces

- Existem algumas interfaces ouvintes reunidas no pacote java.awt.event
  - ActionListener;
  - AdjustmentListener;
  - ComponentListener;
  - ContainerListener;
  - FocusListener;
  - ItemListener;
  - KeyListener;
  - MouseListener;
  - MouseMotionListener;
  - MouseWheelListener;
  - TextListener;
  - WindowListener;

Eventos Semânticos e de baixo nível no AWT

- Um evento semântico é aquele que expressa o que o usuário está fazendo.
  - Exemplo: clicando um botão (ActionEvent)
- Um evento de baixo nível é aquele que torna possível um evento semântico.
- No clique do botão acontece o seguinte:
  - 1) o botão do mouse é pressionado;
  - 2) vários movimentos do mouse são realizados
  - 3) botão do mouse é liberado (somente se ele estiver na área do botão);

Eventos Semânticos

- Existem quatro classes de eventos semânticos no pacote java.awt.event:
  - **ActionEvent** (para o clique do botão, seleção de um menu, clique duplo em um item de lista, tecla *enter* pressionada em um campo de texto)
  - **AdjustmentEvent** (o usuário ajusta uma barra de rolagem)
  - **ItemEvent** (o usuário fez uma seleção num conjunto de caixas de seleção ou itens de uma lista)
  - **TextEvent** (o conteúdo de um campo de texto ou área de texto foi modificado)

Eventos de baixo nível

- Existem sete classes de eventos de baixo nível:
  - **ComponentEvent** (o o componente foi redimensionado, movido, exibido ou ocultado); é a classe base para todos os eventos de baixo nível
  - **KeyEvent** (uma tecla foi pressionada ou liberada)
  - **MouseEvent** (o botão do mouse foi pressionado, liberado, movido ou arrastado)
  - **MouseWheelEvent** (a barra de rolagem do mouse foi rodada)
  - **FocusEvent** (um componente recebeu ou perdeu o foco)
  - **WindowEvent** (uma janela foi ativada, desativada, minimizada, restaurada ou fechada)
  - **ContainerEvent** (um componente foi adicionado ou removido)

Eventos de Foco

- Um componente tem o foco se puder receber pressionamentos de teclas
  - Campo de texto tem o foco quando o cursor de inserção (I) torna-se visível
  - Botão tem o foco geralmente quando se encontra marcado com uma borda de seleção;
  - Somente um componente pode ter um foco de cada vez;

Eventos de Janela

- Uma janela abriu
- Uma janela fechou
- Uma janela tornou-se ativa
- Uma janela tornou-se inativa
- Uma janela Minimizada ou Restaurada

Eventos de Teclado

- Quando o usuário pressiona uma tecla um evento keyEvent KEY_PRESSED é gerado;
- Quando o usuário solta a tecla um evento keyEvent KEY_RELEASED é gerado;
- Esses eventos são capturados pelos métodos keyPressed e keyReleased de qualquer classe que implementa a interface keyListener

Eventos do Mouse

- Não é necessário processar explicitamente os eventos do mouse caso você só queira que o usuário seja capaz de clicar em um **botão** ou **menu**;
- Caso queira permitir que o usuário **desenhe** com o mouse terá de capturar os eventos de **movimento**, **clique** e **arrastar** do mouse

Exemplo de Eventos do Mouse

Classes Adaptadoras

Classes Adaptadoras

- Para simplificar o uso dos listeners que nos obrigam a implementar vários métodos foram criadas as classes adaptadoras.
- Cada classe dessa implementa todos os métodos de um determinado listener no estilo *do-nothing*
- Ex: A **WindowListener** nos obriga a implementar:
  - windowClosing, windowOpened, windowClosed, windowIconified, windowDeiconified,windowActivated, windowDeactivated

Classes Adaptadoras

- Com a classe adaptadora WindowAdapter o desenvolvedor foca apenas no evento que ele precisa implementar:

```
// Adicionando um window listener.
this.addWindowListener (
    new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    }
);
```

Classes Adaptadoras

- Algumas classes adaptadoras frequentemente utilizadas:

- FocusAdapter
- KeyAdapter
- MouseAdapter
- MouseMotionAdapter
- WindowAdapter

Consumir Eventos

- Muitas vezes pode-se interceptar um evento de modo que ele não chegue até um certo componente de usuário

- Ex: um campo de texto que só aceita números, caso se digite letras devemos bloquear o evento.

- Para isso é necessário usar o KeyEvent.consume();

Exercício

- Fazer uma tela com um botão e um campo de texto que só aceite números e caso não seja preenchido nada neste campo, ao sair dele, deve-se exibir a mensagem em um JLabel: “Campo de preenchimento obrigatório”

- Dicas:
  - Use uma classe KeyListener ou a sua adaptadora KeyAdapter e capture o evento KeyTyped.
  - Verifique o caracter digitado através do parâmetro evt.getKeyChar()
  - Para barrar o aparecimento de uma letra utilize o evt.consume()
  - Use a classe focusListener ou a sua adaptadora FocusAdapter para manipular o evento focusLost