Programação Orientada e Objetos II

Anhanguera


AVALIE
SUA PROFISSÃO
QUANDO APARECER EM SEU PORTAL UMA AVALIAÇÃO SOBRE SEU CURSO, RESPONDA:
NOTAS
9 OU 10
SIGNIFICA QUE VOCÊ INDICA
NOTAS
7 OU 8
SIGNIFICA QUE VOCÊ NÃO INDICA
Anhanguera

Anhanguera

Na seção anterior tratamos as formas de configurar uma interface gráfica e adicionar os componentes nessa janela. É importante ressaltar que as interfaces gráficas representam como o usuário percebe o sistema, e para se criar essa interface é necessário seguir alguns passos (HORSTMANN, 2016; DEITEL e DEITEL, 2016):

1. Criar uma relação de especialização com a classe que representa sua tela.
2. Declarar como atributos os elementos que serão adicionados à tela.
3. Definir a forma de alocação dos elementos gráficos na tela.
4. No construtor, instanciar, configurar e posicionar os itens na tela.
5. Tratar os eventos dos componentes para tratar as ações do usuário com a interface gráfica.

Interface gráfica x Interface

Seguindo esses passos é possível criar e posicionar os componentes, todavia, para que essa interface seja interativa, é necessário tratar os eventos decorrentes das ações do usuário na janela. No Java Swing existem diversos eventos que podem ser tratados através do uso de interfaces da orientação a objetos. Segundo Deitel e Deitel (2016) interfaces são coleções de métodos relacionados que informam aos objetos o que estes devem fazer, mas não como fazer. É importante perceber que esse novo elemento apresenta conceito e implementação distintos de uma interface gráfica. As interfaces são elementos que padronizam um conjunto de métodos; dessa forma é possível que uma classe apresente qualquer nível de complexidade, e essa interface propicia as formas de fazer o acesso padronizado e encapsulado.

Para criar uma interface, clique com o botão direito no projeto, selecione New >> Interface, escolha a pasta e o nome. O código no Quadro 1.5 apresenta um exemplo desse item da orientação a objetos. Repare que nesse tipo de código não temos nenhuma implementação de código, e sim um conjunto de métodos; essas descrições de implementação representam uma forma de padronizar o método que será o responsável por tratar um evento na interface gráfica (FURGERI, 2015). Na linha 1 se define qual será o nome da interface, e nas linhas 2 e 3 são descritos quais métodos as classes que utilizarem essa interface deverão ter. Portanto, quando uma classe implementar a interface AcessoElementos, esta deverá, obrigatoriamente, implementar os métodos getElemento e setElemento.

```
1. public interface AcessoElementos {
2. public int getElemento (int index);
3. public void setElemento (int index);
4.}
```

Para uma classe implementar uma interface, esta deve usar o comando implements, conforme ilustra o Quadro 1.6. Veja que na linha 2, a classe Aluno implementa a interface AcessoElementos, portanto os métodos da interface tiveram que ser declarados. A notação @Override é usada para indicar que os métodos serão sobrescritos.

```
1. import javax.swing.*;
2. public class Aluno extends JFrame implements
AcessoElementos{
3. @Override
4. public int getElemento(int index) {
5. return 0;
6. }
7. @Override
8. public void setElemento(int index) {
9. //códigos
10. }
11. }
```

Para evitar confusões na definição dos termos, chamaremos de interface os elementos da orientação a objetos que descrevem um conjunto de métodos, e de interface gráfica os elementos como botões, janelas, campos e outros. Ao se construir uma interface gráfica usando Java Swing, acrescentamos as interfaces para definir qual método será utilizado no momento de chamada que um evento é iniciado.

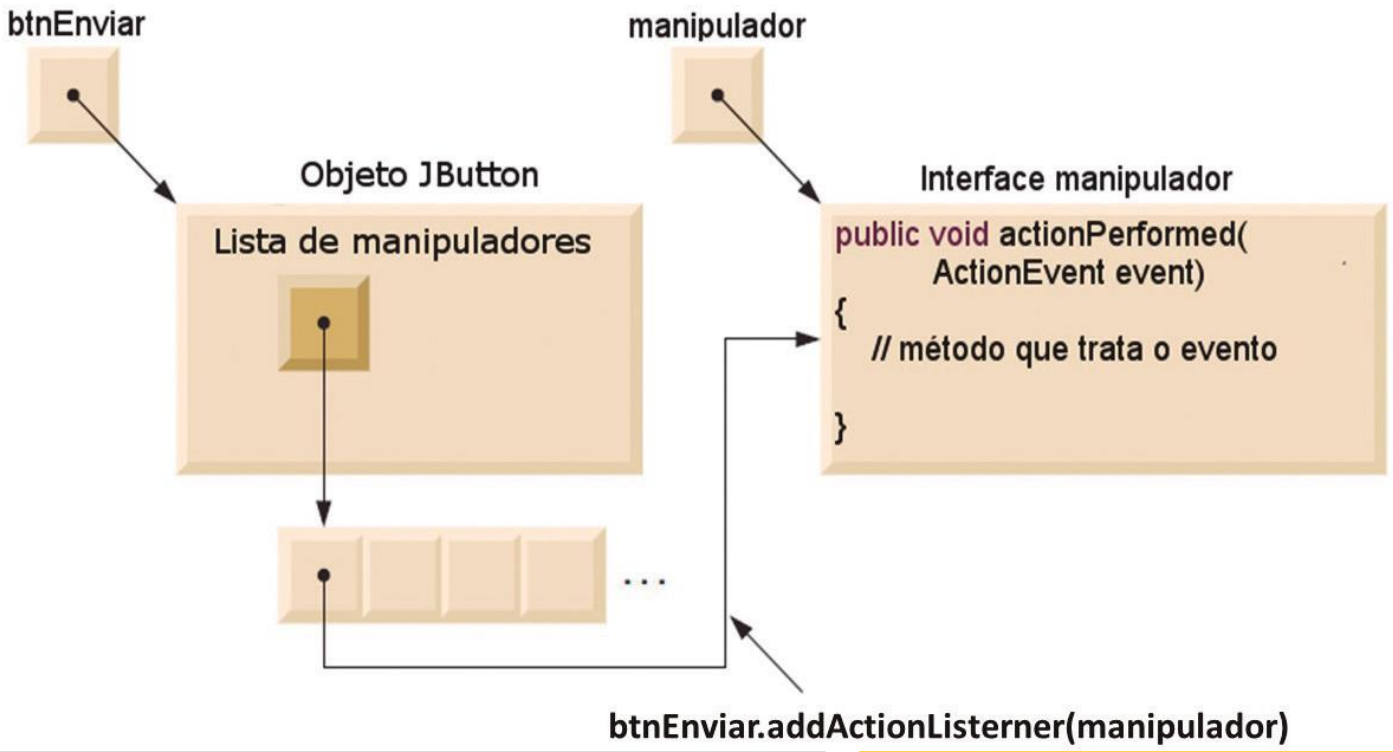
Tratamento de eventos

A criação da interface gráfica representa o mecanismo de interação do usuário com o sistema. Cada ação do usuário gerará uma reação no sistema e do sistema. Esse conjunto de ações (interações) é o que se chama de eventos, e existem diversos tipos já predefinidos, por exemplo, clicar em um botão, pressionar uma tecla, maximizar uma janela, etc. O Quadro 1.7 traz alguns eventos suportados pela linguagem Java. É importante ter em mente que para cada evento é necessário implementar um método; ao longo do livro veremos diversos métodos.

Quadro 1.7 | Eventos na linguagem Java

Evento	Descrição
`ActionListener`	Evento gerado pelo clique com o mouse em um botão.
`ItemListener`	Evento gerado quando um item em uma lista é selecionado.
`FocusListener`	Evento gerado quando um elemento ganha ou perde foco.
`WindowListener`	Evento gerado quando ocorre uma mudança na janela.

Para tratar a ação de “clique em um botão” em uma janela gráfica feita em Java Swing, utiliza-se a interface ActionListener. Na Figura 1.5 temos a representação de um objeto da classe JButton, instância que pode tratar diversos eventos. A instância "btnEnviar" registra uma interface do manipulador, nesse caso o ActionListener, para tratar um de seus possíveis eventos.



Para demonstrar, utilizaremos a tela da Figura 1.6. Essa interface gráfica dispõe de um JButton, um JLabel e um JTextField. O objetivo é verificar se o botão foi clicado pelo usuário, e para isso é possível utilizar a interface ActionListener de três maneiras distintas (MANZANO, 2014):

1. Implementar a interface ActionListener diretamente na classe que especializa o JFrame.
2. Criar uma classe que implementa a interface e trata todos os eventos dos componentes gráficos.
3. Fazer que cada componente crie uma classe anônima que implementa a interface ActionListener.

Os itens 1 e 2 utilizam apenas um método para tratar os eventos de diversos componentes; dessa forma torna-se necessário utilizar os parâmetros das interfaces que tratam os eventos e detectar qual o objeto fez a chamada. O item 3 cria uma interface específica para cada componente, sem a necessidade de avaliar qual componente fez a chamada.

Vamos começar implementando o código necessário para gerar a interface gráfica. O Quadro 1.8 apresenta a parte que cria a interface da Figura 1.6. As linhas 1 e 2 descrevem as classes que serão importadas; a definição da classe que especializa a classe JFrame está na linha 3, e entre as linhas 4 a 7 são criados os atributos que representam os componentes da tela, nesse caso o JLabel, JButton e JTextField. As linhas 10 e 11 representam a configuração da tela em termos de tamanho (setSize) e título (setTitle), os elementos das linhas 12 e 13 são os pontos que tratam da posicionamento em local absoluto na tela ctn. setLayout(null); e ctn = getContentPane();. Nas linhas 14 a 22 os objetos são alocados e configurados. Veja que na linha 23 deixamos indicado onde será inserido parte do tratamento de evento ao clicar no botão. Na linha 24, a visibilidade da tela, e nas linhas 26 a 28, a função inicial do programa. Na linha 29 também deixamos indicado que adicionaremos um código para tratamento do evento.

Anhanguera

```
1. import java.awt.Container;
2. import javax.swing.*;
3. public class PrimeiraTela extends JFrame{
4.    private JButton btnok;
5.    private JTextField txtNome;
6.    private JLabel lblNome;
7.    private Container ctn;
8.    public PrimeiraTela()
9.    {
10.       setSize(300,140);
11.       setTitle(“Eventos em Java Swing”);

12.       ctn = getContentPane();
13.       ctn.setLayout(null);

14.       btnok = new JButton(“Enviar”);
15.       lblNome = new JLabel(“Nome”);
16.       txtNome = new JTextField();

17.       lblNome.setBounds(10,10,100,25);
18.       txtNome.setBounds(70,10,200,25);
19.       btnok.setBounds(90,50,80,40);

20.       ctn.add(lblNome);
21.       ctn.add(txtNome);
22.       ctn.add(btnok);
23.       // nesse ponto vamos inserir tratamento dos eventos
24.       setVisible(true);
25.   }
26.   public static void main(String[] args) {
27.       PrimeiraTela tela = new PrimeiraTela();
28.   }
29.    // nesse ponto vamos inserir tratamento dos eventos
30. }
```

Com a interface gráfica criada, vamos tratar o evento de clique no botão “Enviar” utilizando a primeira opção, ou seja, implementando a interface ActionListener diretamente na classe que especializa o JFrame. Para isso é necessário alterar a declaração da classe e implementar uma interface, nesse caso utilizando uma ActionListener conforme o Quadro 1.9 (esse código deverá ser acrescentado ao Quadro 1.8, nas linhas indicadas). Nas linhas 1 e 2 foram importadas as referências para o tratamento de eventos que faremos. Na linha 3 é possível verificar a utilização do comando implements indicando a ActionListener, e com isso essa classe é forçada a implementar o método actionPerformed() feito na linha 30. Na linha 23, adicionamos o método addActionListener() no botão, e o parâmetro this define que a classe que tratará os eventos é a própria classe, nesse caso a PrimeiraTela. Essa abordagem é interessante quando se tem poucos eventos a serem tratados, pois se a janela apresentar mais botões e ambos definirem que a classe que desenha a interface gráfica cuidará de seus eventos, será necessário comparar a resposta do método getActionCommand() com o texto do botão em questão, como o if na linha 32.

Anhanguera

```
1. import java.awt.event.ActionEvent;
2. import java.awt.event.ActionListener;
3. public class PrimeiraTela Tela extends JFrame implements
ActionListener{
…
23.     btnok.addActionListener(this);
…
29.     @Override
30.     public void actionPerformed(ActionEvent e)
31.     {
            // caso seja necessário tratar eventos de mais de um
botão
32.     if(e.getActionCommand().equals(“Enviar”))
33.     {
34.             txtNome.setText(“Botão clicado”);
35.     }
36.   }
37. }
```

A segunda forma de tratar esses eventos consiste em criar outra classe que implemente o ActionListener, porém existe um problema na forma de tratar e diferenciar eventos que ainda não foi resolvido, sendo necessário criar diversas instâncias dessa classe de tratamento, e ainda fazer a comparação com o retorno do método getActionCommand da classe ActionEvent, o que torna essa opção de pouco utilidade.

Para ter uma solução mais precisa é interessante utilizar o conceito de classe anônima (terceira opção), que como sua própria classificação indica, não tem nome e cada elemento gráfico terá uma classe separada para fazer o seu próprio tratamento (DEITEL e DEITEL, 2016).

Veja no Quadro 1.10 a implementação de uma classe anônima dentro da classe Tela. Na linha 3 repare que não existe mais a implementação da interface, mesmo assim é necessário incluir as referências (linhas 1 e 2). Na linha 5 o ActionListener é adicionado ao botão pelo método addActionListener() usando uma classe anônima para fazer o tratamento do evento. Como a classe anônima e a classe Tela estão no mesmo arquivo, a classe anônima pode acessar os elementos da classe que a engloba (WINDER, 2009). Na linha 7 é feita a chamada de um método da classe que fará o tratamento da ação no JButton. O método trataBotaoOk() foi criado na linha 9 e na linha 10 foi especificada a ação do método, que consiste em acessar o método setText() do objeto txtNome adicionando um texto.

Anhanguera

```
1. import java.awt.event.ActionEvent;
2. import java.awt.event.ActionListener;

3. public class Tela extends JFrame {
4.       public Tela()
         {
5.       btnok.addActionListener(new ActionListener() {
6.                public void actionPerformed(ActionEvent e) {
7.                       trataBotaoOk();
8.                }
            });
         }
9.       public void trataBotaoOk()
         {
10.               txtNome.setText("Botão clicado");
         }
    }
```

O modelo de tratamento que o Quadro 1.10 apresenta é interessante, pois para cada elemento gráfico da tela é possível criar um método específico, garantindo assim a coesão de cada elemento da classe.

Em se tratando de eventos relacionados a uma caixa de texto é possível verificar se ocorreram alterações nas propriedades, remoções ou inserções. Essas alterações também podem ser vinculadas à interface ActionListener, bem como a outros eventos como, por exemplo, ao FocusListener e ao DocumentListener, entre outros. Observe o código no Quadro 1.11: na linha 3 o objeto txtNome é vinculado à interface DocumentListener em uma classe anônima, pelo método addDocumentListener(). Essa interface especifica a implementação de três métodos: removeUpdate(), insertUpdate() e changedUpdate(), e mesmo que você não vá implementar os três a chamada é obrigatória. O removeUpdate(), na linha 4, executa ações quando algo é apagado de uma caixa de texto. O método insertUpdate(), na linha 5, executa ações quando algo é inserido na caixa de texto e o método changedUpdate(), quando algo é alterado. Os três métodos implementam a interface DocumentEvent, portanto também é necessário importar a referência javax.swing.event. DocumentEvent.

Anhanguera

```
1. import javax.swing.event.DocumentEvent;
2. import javax.swing.event.DocumentListener;
3.    txtNome.getDocument().addDocumentListener(new
       DocumentListener() {
4.         public void removeUpdate(DocumentEvent e) {
          // ações quando texto for apagado
          }
5.         public void insertUpdate(DocumentEvent e) {
              // ações quando texto for inserido
          }
6.         public void changedUpdate(DocumentEvent e) {
              // ações quando texto for alterado
          }
    });
```

Para componentes gráficos que oferecem opções de escolha para o usuário, como o JComboBox por exemplo, pode ser interessante efetuar tratamento nos dados assim que o usuário seleciona uma das opções. O tratamento de eventos nesse objeto pode ser feito implementando a interface ItemListener (disponível em java.awt.event.ItemListener) e o método addItemListener(), conforme exemplifica o código do Quadro 1.12. Na linha 3 é feita a criação da classe anônima para a implementação da interface ItemListener. O uso dessa interface obriga a implementação do método itemStateChanged(), que apresenta como parâmetro a interface ItemEvent (disponível em java.awt.event.ItemEvent), conforme linha 4. Na linha 5, verificamos se o evento disparado (e) é igual a “selecionar um item”; se for, o método trataJmbTipos() é invocado na linha 7. Na linha 11 é declarado o método para tratar o evento, na linha 13 é mostrada uma mensagem gráfica exibindo qual item foi selecionado pelo JComboBox.

Anhanguera

```
1. import java.awt.event.ItemEvent;
2. import java.awt.event.ItemListener;
3. jmbTipos.addItemListener(new ItemListener() {
4.     public void itemStateChanged(ItemEvent e) {
5.         if (e.getStateChange() == ItemEvent.SELECTED)
6.         {
7.                 trataJmbTipos();
8.         }
9.      }
10. });
11. public void trataJmbTipos()
12.{
13.                 JOptionPane.showMessageDialog(this,
                    “Item selecionado: “+ jmbTipos.
                     getSelectedItem());
14.}
```

Anhanguera

O tratamento de eventos em Java Swing deve sempre seguir os elementos da orientação a objetos, e os métodos devem ter alta coesão, sendo que cada uma dessas ações deve se limitar a funções específicas. Nesse sentido, a implementação das interfaces para tratar os eventos deve apenas recebê-los e chamar outros métodos para que cada ação seja tratada de forma independente.

Para tratar os eventos de componentes no Java Swing é necessário utilizar uma interface que força a criação de métodos no código. Esses métodos são chamados quando o evento acontece na interface gráfica. Assim, existem diversas formas de implementar essa interface e tratar esses eventos, como por exemplo, implementar a interface que trata o evento
escolhido na classe que estende o JFrame.
No caso de implementar a interface ActionListener na classe que estende o JFrame e essa interface gráfica dispor de 30 botões, qual dos itens a seguir representa o tratamento necessário no método ActionPerformed?

a) É necessário utilizar o parâmetro ActionEvent do método ActionPerformed para diferenciar cada chamado de cada botão.
b) A interface ActionPeformed não deve ser utilizada para tratar cliques de botão.
c) O parâmetro ActionEvent não existe na interface ActionPerformed.
d) Não é possível detectar as diferentes chamadas, sendo assim a tela suporta apenas um botão.
e) O método do parâmetro ActionEvent a ser utilizado é o getText.

Das diversas formas de tratar um evento em interfaces gráficas utilizando Java Swing, é interessante destacar a técnica em que se adiciona uma interface diretamente para cada componente. Dessa forma, o tratamento dos eventos não necessita da detecção de qual componente fez o chamado, tratando assim a coesão dos métodos.

Para o tratamento em que cada elemento gráfico pode utilizar umainterface específica, qual tipo de classe é necessária?

a) Classe abstrata.

b) Classe anônima.

c) Classe interface.

d) Classe primitiva.

e) Classe estática.