Slides 14: GUI, Eventos e classes internas

Baseado nos slides oficiais do livro Java – Como programar – Deitel e Deitel – 10ª edição

Introdução

- Conceito
  - GUI - Graphical User Interface ou interface gráfica com o usuário: forma de interação do usuário com o programa através de mouse, teclado ou outro dispositivo de entrada
- Swing
  - Bibilioteca de componentes gráficos padrão para desktop do Java
  - Usa o padrão Nimbus como modelo de look-and-feel
  - SwingSet: demonstração das principais classes da biblioteca com códigos fontes
    - https://mlapshin.com/swingset3/SwingSet3.jnlp

Principais classes

- Component
  - Atributos e comportamentos comuns em classes GUI
- Container
  - Subclasse de Component
  - Pode conter outros componentes dentro dele
  - Componentes internos podem ser mostrados e organizados
- JComponent
  - Superclasse de todos os itens leves da interface gráfica com Swing
  - Características: look-and-feel plugável, suporte para acessibilidade e localização (país, linguagem, …), suporte para texto tooltip, atalhos (mnemônicos) e tratamento de eventos

Tratamento de eventos

- Conceito
  - Resposta do sistema a uma interação do usuário (evento) através de um dispositivo de entrada ou interrupção do sistema
  - Cada evento de cada fonte causadora pode ser tratada de maneira diferente
- Passos
  - Criar uma classe (handler) que vai ser responsável por tratar (determinar o comportamento)
  - Implementer uma interface que determina os métodos dos comportamentos que serão observados (listener)
  - Registrar o handler junto ao componente gráfico de alto nível que terá seus eventos escutados

Hierarquia de classes de eventos

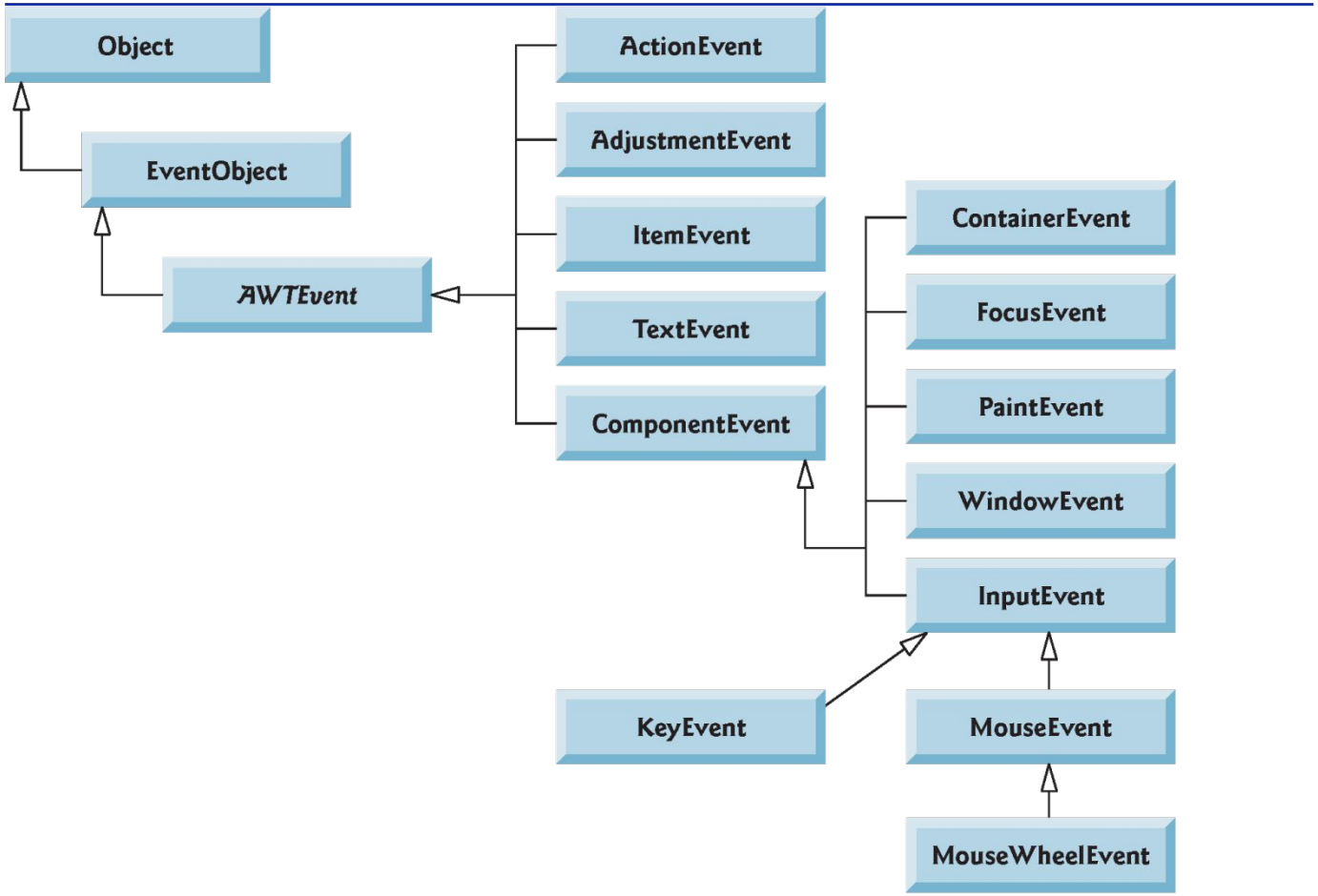

Fig. 12.11 | Some event classes of package `java.awt.event`.

Hierarquia de interfaces de Listeners

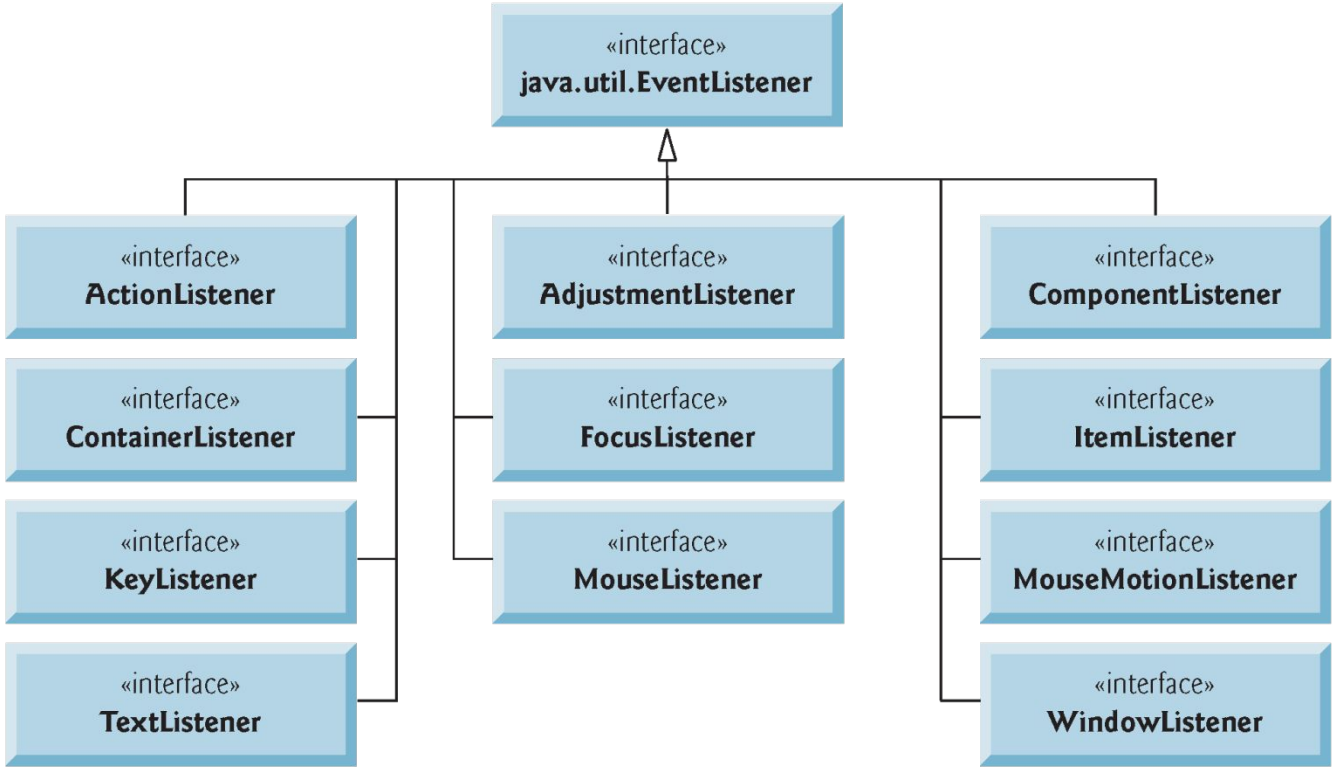

Fig. 12.12 | Some common event-listener interfaces of package `java.awt.event`.

Como funciona?

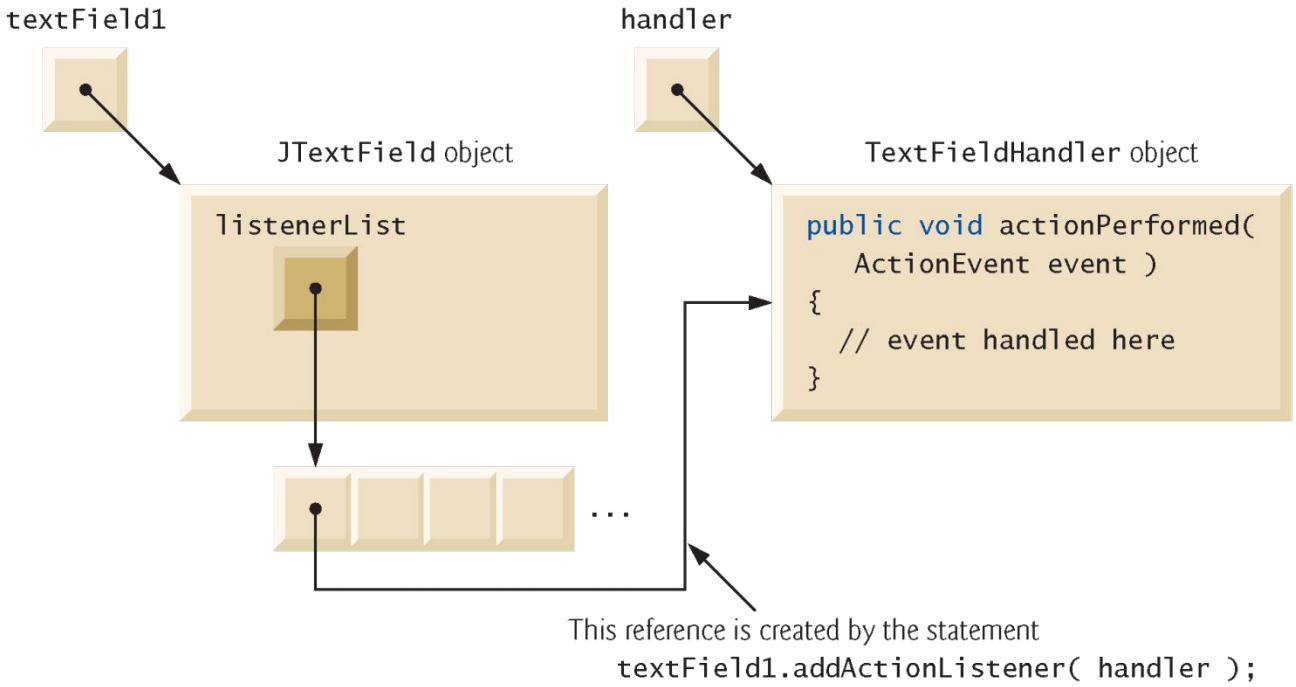

Fig. 12.13 | Event registration for `JTextField textField1`.

Métodos de implementação Classes Adapters

- As classes Adapters de eventos são classes abstratas que implementam comportamentos padrão dos Listeners
- Para criar um handler basta criar uma classe que herda do Adapter desejado e sobrescrever os métodos relativos ao comportamento que se deseja tratar

```
public class KeyHandler extends KeyAdapter{
    @Overrride
    public void KeyTyped (KeyEvent e) { … }
```

- Para registrar o tratador no componente GUI de alto nível, chama-se um método addXXXXListener de acordo com o evento

```
jTextField1.addKeyListener (new KeyHandler ());
```

- Todo método tratador recebe um objeto do tipo do evento gerado com informações específicas.

```
char tecla = e.getKeyChar( );
```

Exemplo

Criar uma classe TextColorHandler que implementa a interface KeyListener. Ao digitar uma tecla, o handler deve trocar a cor da fonte do componente. Criar um programa com GUI com uma caixa de texto e registar um TextColorHandler através do método addListener

Métodos de implementação Usando interfaces Listeners

- Caso não se queira / possa usar a herança, pode-se implementar o Listener implementando a interface do evento desejado
- Nesse caso, a classe Handler tem que implementar todos os métodos da interface implementada

```
public class FocusHandler implements FocusListener{
    void focusGained(FocusEvent e) { … }
    void focusLost(FocusEvent e) { … }
```

- Usando esse método, o próprio componente pode ser usado como Handler

```
public class EmailTextField extends JFormattedTextField
implements FocusListener { … }
```

Exemplo 2

Criar uma subclasse da classe JFormattedTextField para receber um e-mail. Implementar a interface FocusListener e a cada vez que ela perder o foco, verificar se o e-mail esta correto. Se não estiver avaisar com um JOptionPane e retomar o foco.

Classe internas

- Podemos declarar classes dentro de outras classes quando, por exemplo, só utilizaremos objetos delas nesse local
- Essas classes são chamadas de classes internas (inner classes) e podem inclusive ter encapsulamento privado
- Um caso em que isso é usado é no tratamento de eventos onde o comportamento de um componente é específico e o handler não é reutilizado para outros componentes

```
public class Janela extends JFrame {

    private class KeyHandler extends KeyAdapter { … }

    public Janela(){
        addKeyListener (new KeyHandler () );
```

Exemplo 3

Refatorar o exemplo 2 colocando o tratamento de evento como uma classe interna.

Objetos anônimos

- Chamamos de objetos anônimos aqueles que são declarados sem se atribuir a uma variável, ou seja, não há um nome para chamá-lo diretamente

addKeyListener (**new KeyHandler ( )**);

- Nesses casos, a única forma de referenciar os objetos anônimos é se eles tiverem sido atribuídos a referências nas classes e possam ser acessados
  - Ex: getKeyListener(), getComponents
- É possível chamar um método em objetos anônimos logo após sua instanciação:
  - new JFrame("Janela").setVisible(true);
- Nas classes containers GUI do Java é possível usar o método getComponents ( ), que retorna um vetor com as referências para todos os componentes que estão dentro deles.

Exemplo 4

Criar um handler de ActionEvent para um botão que altera as cores de background de campos de texto para vermelho se os mesmos não estiverem preenchidos

Classes internas anônimas

- Ainda em relação a eventos (e casos semelhantes) um handler só é criado visando determinar o comportamento de um objeto específico
- Usando a sintaxe de criação de objetos anônimos podemos definir uma classe interna anônima que herda de uma classe ou implementa uma interface
- Criamos um objeto anônimo com o tipo da superclasse ou interface a ser implementada e abrimos uma chave onde serão inseridos os métodos

```
new JFrame("Janela") {
    public void mudaCor(){
        setBackground(Color.blue);
    }
}.setVisible(true);
```

Classes internas anônimas e eventos

- É comum usar classes internas anônimas quando um evento é específico para um componente
- A classe é criada quando se registra o handler para o listener do evento

```
addKeyListener( new KeyAdapter ( ) {
        @Override
        public void keyTyped(KeyEvent e){
            ...
        }
    }
```

- É importante lembrar que uma classe interna anônima tem o escopo limitado ao método na qual ela está sendo declarada

Exemplo 5

Criar uma classe interna anônima de MouseAdapter para um botão em que toda vez que o mouse estiver sobre ele a fonte do texto fique em negrito e quando o mouse sair a fonte volte ao normal