FACAMP
FACULDADES DE CAMPINAS

Linguagens de Programação

Aula 10

Tratamento de exceções e eventos

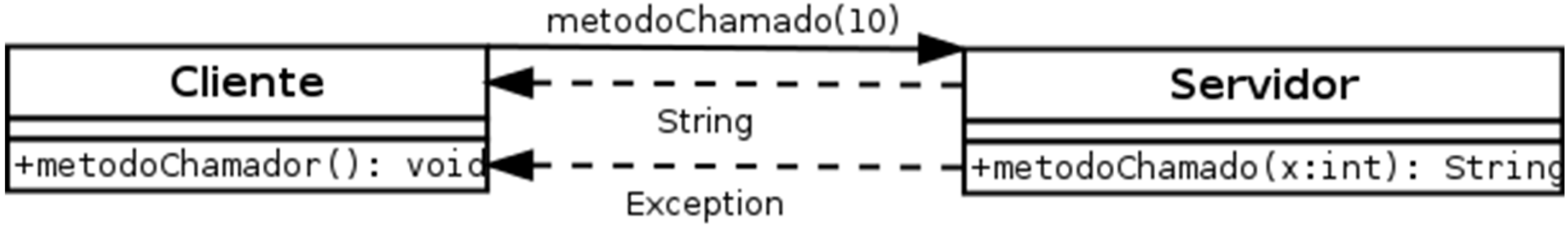
2º semestre de 2019
Prof José Martins Junior

Motivação

- Muitas linguagens modernas incluem um mecanismo para comunicação de exceções
  - Ada, C++, Java, C#
- A forma geral é a mesma, exceções podem ser
  - Implementadas/estendidas
  - Previstas e lançadas
  - Tratadas ou repassadas

Exceções em Java

- Java possui recursos para manipulação de erros ou exceções
  - Estabelece um fluxo alternativo para retorno de um método, em regime de exceção



- Alguns exemplos de erros
  - Erros de parâmetros de entrada (formato incorreto de números, p.e.)
  - Erros em acesso a dispositivos (I/O de arquivos e rede, p.e.)
  - Limitações físicas (memória, tamanhos de Strings e arrays, p.e.)
  - Erros de código (casting inapropriado, objetos nulos, p.e.)

Tipos de exceções

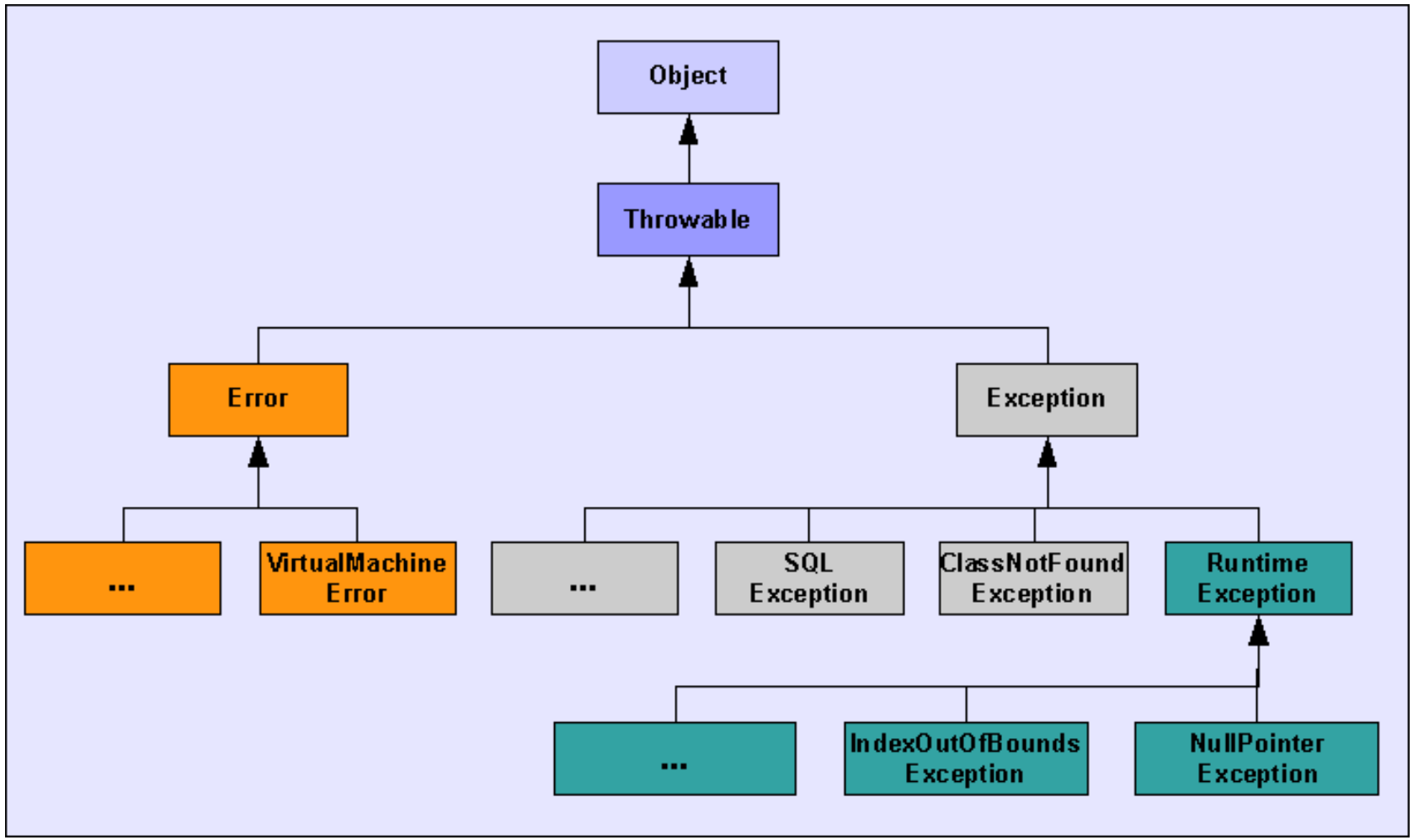
- Toda exceção em Java é uma subclasse de Throwable

- Existem dois filhos diretos dessa classe
  - Error
    - Erros internos da JVM, como a falta de recursos no sistema, em tempo de execução
    - Não podem ser previstos, tratados ou lançados
    - Exemplos: AWTError, VirtualMachineError
  - Exception
    - Define uma hierarquia de exceções que podem/devem ser previstas, lançadas ou tratadas
    - Existem dois tipos: runtime (**unchecked**) e **checked** exceptions

Exceptions

- **RuntimeException**
  - Exceções em tempo de execução e, portanto, não são verificadas pelo compilador (conhecidas como unchecked exceptions)
  - Como consequência, não precisam ser obrigatoriamente anunciadas e podem ser lançadas ou obtidas/tratadas independentemente disso
  - Exemplos: NullPointerException, ClassCastException, IndexOutOfBoundsException
- **Checked exceptions**
  - Representam exceções previsíveis, de acordo com a situação (acesso a arquivos, conexão com BD ou rede, p.e.)
  - São verificadas pelo compilador e, portanto, devem ser anunciadas e obtidas/tratadas ou repassadas
  - Exemplos: FileNotFoundException, SQLException, NumberFormatException

Hierarquia de exceções em Java



Error

Checked Exceptions

Unchecked Exceptions

Manipulando exceções

- Exceções em métodos
  - Podem (no caso das checked exceptions, devem) ser **anunciadas** (**throws**) e definem um caminho alternativo à execução do método chamado
  - Quando o método em execução se deparar com a ocorrência da situação prevista, ele pode **lançar** (**throw**) a exceção, o que interrompe sua execução naquele momento
  - O chamador pode **tentar** (**try**) executar o método que anuncia a exceção, em caso dela ocorrer, pode **pegá-la** (**catch**) e **tratá-la**
  - Alternativamente, o método chamador pode também **anunciá-la** (**throws**), e **repassar** a obrigação para o método que o chamar

Classes e exceções

- Como já dito, exceções são classes em Java e podem ser estendidas (subclasses de exceções)

- Implicações em herança
  - Quando um método declara que lança uma exceção de um tipo (classe de exceção), pode também lançar exceções estendidas desse tipo (de subclasses do tipo de exceção declarado)
  - Um método de uma subclasse, reescrito de uma superclasse, não poderá anunciar mais exceções explícitas que o método original (se o método da classe-pai não anunciar qualquer exceção explícita, o método da classe-filha também não poderá fazê-lo)

Rastro da pilha de execução

- Quando uma exceção surge, dentro da execução de algum método, em tempo de execução
  - Se um método detectar/obter uma exceção, se não a tratar, pode repassar ao chamador
  - O processo continua, até que um método pegue e trate a exceção

- O caminho percorrido pela exceção é chamado de stacktrace (rastro da pilha)
  - Se não tratada a exceção (uma exceção runtime, p.e.), o rastro é exibido na interface e pode ser utilizado para depuração do erro

Exemplo de método com erro

```
class TesteErro {
    public static void main(String[] args) {
        System.out.println("inicio do main");
        metodo1();
        System.out.println("fim do main");
    }
    static void metodo1() {
        System.out.println("inicio do metodo1");
        metodo2();
        System.out.println("fim do metodo1");
    }
    static void metodo2() {
        System.out.println("inicio do metodo2");
        int[] array = new int[10];
        for (int i = 0; i <= 15; i++) {
            array[i] = i;
            System.out.println(i);
        }
        System.out.println("fim do metodo2");
    }
}
```

Obtido da apostila da Caelum FJ11

Resultado da execução

```
run:
inicio do main
inicio do metodo1
inicio do metodo2
0
1
2
3
4
5
6
7
8
9
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
        at javaapplication7.TesteErro.metodo2(TesteErro.java:29)
        at javaapplication7.TesteErro.metodo1(TesteErro.java:21)
        at javaapplication7.TesteErro.main(TesteErro.java:15)
Java Result: 1
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Anunciando exceções

- Utiliza-se a sentença throws na declaração do método para informar os tipos de exceções que pode retornar
  - Ex. (método da classe DataInputStream)

```
public String readLine() throws IOException
```

- Uma exceção deve ser anunciada sempre que um método
  - Prever uma situação de erro e lançar (throw) uma checked exception
  - Realizar chamada a outro método que anuncia uma checked exception, que o método chamador não pretende tratar

Lançando exceções

- Como dito, uma exceção pode ser lançada
  - Utiliza-se o operador throw
    - Exemplo: `throw new Exception("Ocorreu um erro!");`
  - Se a exceção lançada for do tipo checked (como a do exemplo acima), ela deve ser anunciada (throws) pelo método que a lançar, se este não a tratar localmente (em um try-catch)
    - Exemplo:

```
public void teste() throws Exception {
    //...
    throw new Exception("Ocorreu um erro!");
    //...
}
```

  - Exceções unchecked podem ser lançadas sem prévio anúncio

Pegando (catch) exceções

- Exceções podem ser obtidas e tratadas
  - Quando anunciadas explicitamente por um método, ou quando se prevê uma exceção em tempo de execução (RuntimeException)
  - **try**
    - Demarca o bloco de instruções que realiza chamada a métodos que podem retornar exceções
    - Se ocorrer uma exceção, a execução do bloco é interrompida e é assumida a condição catch (compatível com a exceção)
  - **catch(*TipoExceção* nome)**
    - Delimita um bloco que pode conter o código para tratar a exceção ou passá-la adiante
    - Recebe em nome a instância da exceção para o tipo declarado

Exemplo

- Complementando o exemplo do slide 9

```
static void metodo2() {
    System.out.println("inicio do metodo2");
    int[] array = new int[10];
    try {
        for (int i = 0; i <= 15; i++) {
            array[i] = i;
            System.out.println(i);
        }
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("erro: " + e.getMessage());
    }
    System.out.println("fim do metodo2");
}
```

Resultado da execução

```
run:
inicio do main
inicio do metodo1
inicio do metodo2
0
1
2
3
4
5
6
7
8
9
erro: java.lang.ArrayIndexOutOfBoundsException: 10
fim do metodo2
fim do metodo1
fim do main
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Criando exceções

- Exemplos de exceções que estendem Exception

```
public class LessThanException extends Exception {
    @Override
    public String getMessage() {
        return "O valor fornecido é menor!";
    }
}

public class GreaterThanException extends Exception {
    @Override
    public String getMessage() {
        return "O valor fornecido é maior!";
    }
}
```

Lançando e pegando exceções

- Exceções podem ser lançadas e imediatamente tratadas

```
public void teste(int i) {
        try {
            if (i < 10) {
                throw new LessThanException();
            } else if(i > 10) {
                throw new GreaterThanException();
            } else {
                System.out.println("O valor é " + i);
            }
        } catch (LessThanException ex) {
            System.out.println(ex.getMessage());
        } catch (GreaterThanException ex) {
            System.out.println(ex.getMessage());
        }
}
```

A cláusula finally

- O operador finally é executado em ambas situações
  - Quando um bloco de instruções delimitado por try é executado por completo, sem exceções
  - Quando da recepção de uma exceção (e execução das instruções contidas em catch)
- Artifício importante para executar ações em qualquer condição de saída do método (normal ou erro), como liberar recursos do sistema
  - Arquivos abertos
  - Conexões de rede
  - Conexões com BD

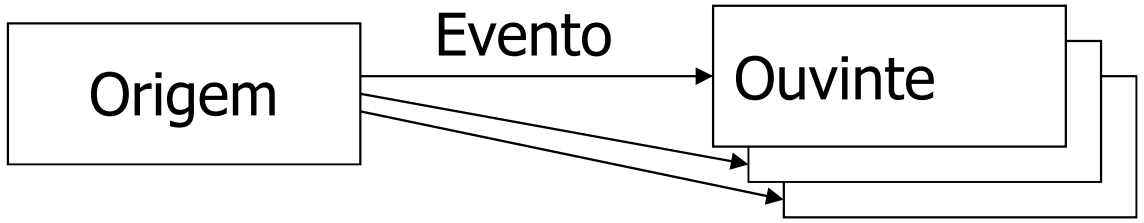
A cláusula finally

- Exemplo

```
FileWriter writer = null;
try {
    writer = new FileWriter("teste.txt");
    // outros códigos que podem obter exceções
} catch (Exception ex) {
    System.out.println("Ocorreu um erro!");
} finally {
    try {
      writer.close();
    } catch (Exception ex) {
      System.out.println("Erro: " + ex.getMessage());
    }
}
```

Eventos no Java

- A versão 1.1 de Java trouxe mudanças significativas no tratamento de eventos pela AWT (Swing usa tal versão)
- Modelo de eventos em Java



- Objetos envolvidos
  - Origem - registra, através de métodos próprios, objetos ouvintes para determinado evento. Ex.: origem.addActionListener(ouvinte)
  - Eventos - objetos enviados aos ouvintes registrados, quando o evento ocorre
  - Ouvinte - instância de objeto que implementa uma listener interface compatível com o tipo de evento envolvido. Deve implementar cada método da interface ouvinte que será chamado pela origem para a passagem dos eventos, podendo assim tratá-los

Exemplificando o modelo

- Os objetos de eventos derivam de java.util.EventObject
  - Ex.: ActionEvent - enviado por botões
- Exemplo: associando o evento de um botão ao painel

```
MyPanel painel = new MyPanel();
JButton botao = new JButton("Limpar");
botao.addActionListener(painel);
```

  - A classe MyPanel deve implementar a interface ActionListener e o método actionPerformed para estar apta a receber um objeto ActionEvent (quando ocorrer o click no botão)

```
public class MyPanel extends Jpanel implements
ActionListener {
   public void actionPerformed(ActionEvent evt) {
       //tratamento do evento ...
   }
}
```

Distinguindo eventos do mesmo tipo

- Um objeto ouvinte pode receber eventos de várias origens do mesmo tipo
- Pode-se chamar seu método getSource

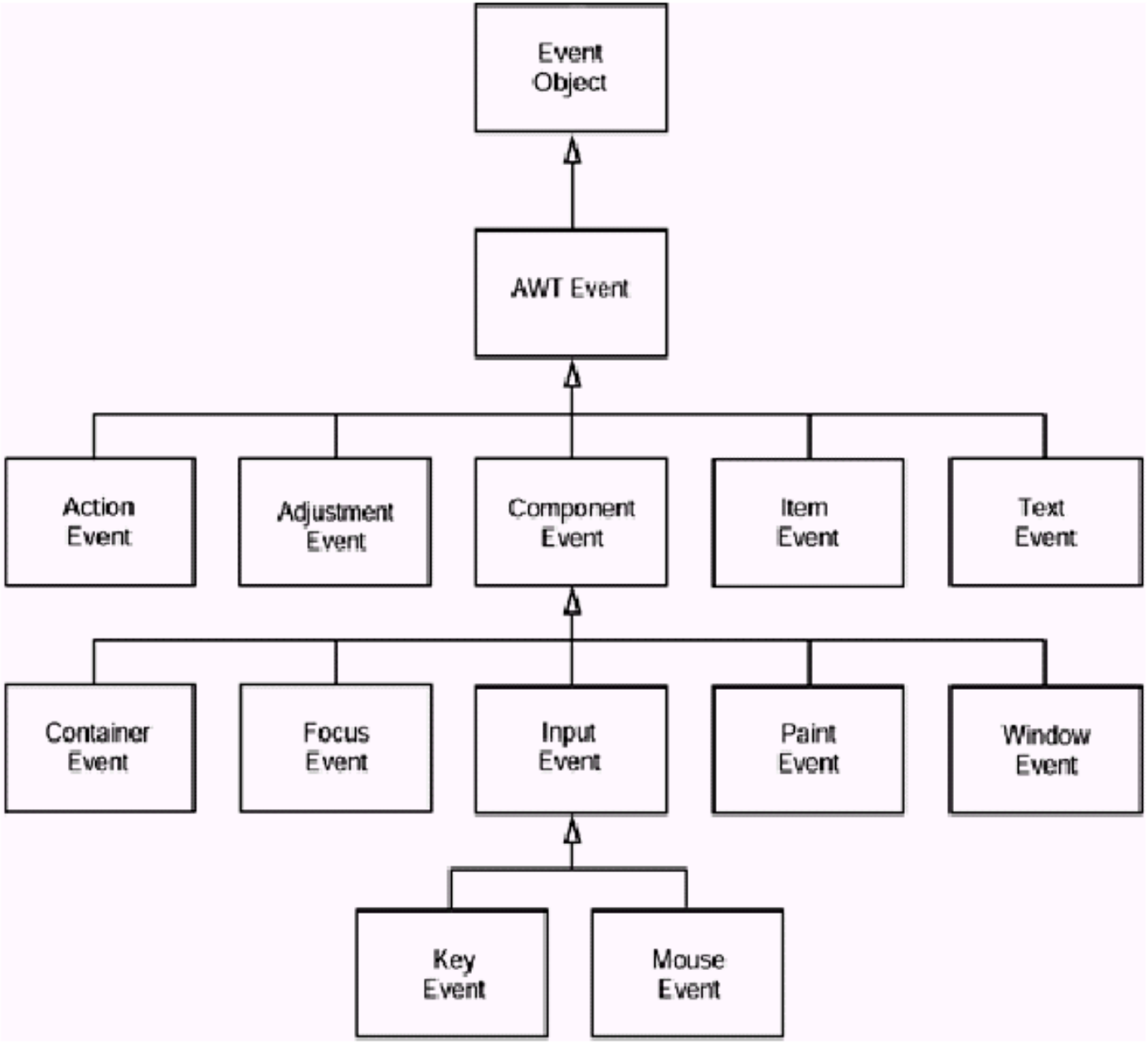
```
Object source = evt.getSource();
if(source == botao1) {
...
} else if(source == botao2) {
...
```

Eventos de janelas

- Um WindowListener tem 7 métodos para eventos de janelas!
- Usa-se uma classe (adaptadora) abstrata WindowAdapter, que possui implementação vazia para os 7 métodos
  - Cria-se uma classe que a estenda e implementa-se apenas os métodos desejados. Ex. (classe adaptadora interna anônima):

```
class MyFrame extends Jframe {
    public MyFrame() {
       addWindowListener(
         new WindowAdapter() {
           public void windowClosing(WindowEvent e){
              System.exit(0);
           }
         }
       );
    }
}
```

Hierarquia de eventos no AWT



Eventos, Interfaces e Adaptadoras

Evento	Interface	Adaptadora
ActionEvent	ActionListener	
AdjustmentEvent	AdjustmentListener	
ComponentEvent	ComponentListener	ComponentAdapter
ContainerEvent	ContainerListener	ContainerAdapter
FocusEvent	FocusListener	FocusAdapter
ItemEvent	ItemListener	
KeyEvent	KeyListener	KeyAdapter
MouseEvent	MouseListener	MouseAdapter
	MouseMotionListener	MouseMotionAdapter
TextEvent	TextListener	
WindowEvent	WindowListener	WindowAdapter

Eventos semânticos

- Semântico: expressa uma ação do usuário
  - ActionEvent: click de botão, seleção de menu, click duplo em um item da lista, <enter> pressionado em um campo texto
  - AdjustmentEvent: usuário ajusta uma barra de rolagem
  - ItemEvent: usuário selecionou em um conjunto de caixas de seleção ou itens de uma lista
  - TextEvent: conteúdo de um campo (ou área) de texto modificado

Eventos de baixo nível

- Baixo nível: eventos que não expressam semanticamente uma ação do usuário
  - ComponentEvent: componente redimensionado, movido, exibido ou ocultado
  - KeyEvent: tecla foi pressionada ou liberada
  - MouseEvent: botão do mouse pressionado, liberado, movido ou arrastado
  - FocusEvent: componente recebeu ou perdeu o foco
  - WindowEvent: janela foi ativada, desativada, minimizada, restaurada ou fechada
  - ContainerEvent: componente foi adicionado ou removido

Resumo sobre manipulação de eventos 1/2

Interface	Methods	Parameter/Accessors	Events Generated By
ActionListener	actionPerformed	ActionEvent getActionCommand getModifiers	AbstractButton JComboBox JTextField Timer
AdjustmentListener	adjustmentValueChanged	AdjustmentEvent getAdjustable getAdjustmentType getValue	JScrollbar
ItemListener	itemStateChanged	ItemEvent getItem getItemSelectable getStateChange	AbstractButton JComboBox
TextListener	textValueChanged	TextEvent	
ComponentListener	componentMoved componentHidden componentResized componentShown	ComponentEvent getComponent	Component

Resumo sobre manipulação de eventos 2/2

ContainerListener	componentAdded componentRemoved	ContainerEvent getChild getContainer	Container
FocusListener	focusGained focusLost	FocusEvent isTemporary	Component
KeyListener	keyPressed keyReleased keyTyped	KeyEvent getKeyChar getKeyCode getKeyModifiersText getKeyText isActionKey	Component
MouseListener	mousePressed mouseReleased mouseEntered mouseExited mouseClicked	MouseEvent getClickCount getX getY getPoint translatePoint isPopupTrigger	Component
MouseMotionListener	mouseDragged mouseMoved	MouseEvent	Component
WindowListener	windowClosing windowOpened windowIconified windowDeiconified windowClosed windowActivated windowDeactivated	WindowEvent getWindow	Window

Eventos de foco

- Muitos componentes de interface podem receber o foco
  - Foco pode ser chaveado através do click do mouse, como pelo movimento do mouse sobre o componente ou com a navegação usando a tecla <tab>
- FocusListener provê dois métodos: focusGained e focusLost
- Pode-se mudar o foco através dos métodos do componente
  - requestFocus: solicita o foco
  - transferFocus: muda o foco para o componente anterior
  - Outros métodos
    - getComponent retorna o componente que gerou o evento
    - isTemporary (método de FocusEvent): retorna se a mudança de foco é temporária

Eventos do teclado

- A interface KeyListener provê o acesso a eventos KeyEvent, pelos métodos keyPressed e keyReleased
  - O método keyTyped combina os dois eventos, possibilitando o acesso ao valor da tecla pelos métodos getKeyChar e getKeyCode de KeyEvent
- A classe KeyEvent define constantes para o valor das teclas virtuais (VK_...) ou mnemônicas
- Outros métodos de KeyEvent, herdados de InputEvent
  - isShiftDown
  - isControlDown
  - isAltDown
  - isMetaDown
  - Os métodos acima retornam um boolean que significa se a respectiva tecla também está pressionada

Eventos do mouse 1/2

- Não é necessário tratar eventos semânticos do mouse quando forem relacionados a ações sobre componentes (botões...)
- 3 métodos ouvintes de MouseListener podem ser chamados
  - mousePressed - o mouse foi pressionado
  - mouseReleased - o botão do mouse foi liberado
  - mouseClicked - que descreve os dois anteriores juntos
- Métodos de MouseEvent (ou de seus antecessores)
  - getX e getY: retornam as coordenadas X e Y do evento
  - getClickCount: descreve quantos clicks ocorreram no evento (click simples, duplo ou triplo)

Eventos do mouse 2/2

- Outros métodos de MouseListener
  - mouseEntered: método chamado quando o mouse entra em um componente
  - mouseExited: chamado quando o mouse sai do componente
- Métodos de MouseMotionListener
  - mouseMoved: recebe fluxos de eventos toda vez que o mouse é movimentado
  - mouseDragged: recebe fluxos de eventos, descrevendo a ação de arrastar um componente (clicar sobre e mover)

Eventos no NetBeans

- O ambiente de edição GUI do NetBeans facilita a programação de eventos
  - Oferece opção para adição de eventos para cada objeto gráfico
  - A partir da adição do evento, automaticamente cria um método para tratá-lo, escondendo a reimplementação do método do Listener (ou Adapter) na área protegida do formulário

Exercício: criação de evento

- MessageEvent: para comunicação de mensagem entre Frames
- Serão implementadas
  - MessageEvent
    - Classe de evento que estende EventObject
  - MessageEventListener
    - Interface que declara o método messageArrived
    - Deve ser implementada pela origem e pelos ouvintes
  - JFrame
    - Uma das janelas terá o comportamento de origem e adicionará novos ouvintes com um método addMessageListener
    - Quando ocorrer um evento, deve iterar a lista de ouvintes e comunicar todos, passando o evento pelo método comum messageArrived
    - A outra janela será do tipo ouvinte e serão criadas várias instâncias pela janela principal (origem)

Bibliografia

GOSLING, J.; ARNOLD, K.; HOLMES, D. A Linguagem de Programação Java. 4ª edição, Porto Alegre: Bookman, 2007.

HORSTMANN, C. Conceitos de Computação com Java, 5ª edição. Porto Alegre: Bookman, 2009.

CAELUM. Apostila do curso FJ-11 – Java e Orientação a Objetos. Disponível em https://www.caelum.com.br/apostila-java-orientacao-objetos/, acessado em 13/10/2019.