

LCP - Aula 06

Prof. Lucas Guerreiro

Aulas anteriores

CONTEÚDO PROGRAMÁTICO (título e discriminação das unidades)

1. Recursos computacionais para implementação de sistemas comerciais: Características, Histórico e Ferramentas Computacionais Disponíveis
2. Estudo detalhado de uma linguagem de programação comercial orientada a objetos.
 - 2.1 A tecnologia Java
 - 2.2. Ambientes de Desenvolvimento de Aplicações em Java
 - 2.3. Conceitos Básicos da Linguagem Java
 - 2.4 Conceitos da programação orientada a objetos : abstração, classe e objeto, herança, polimorfismo e interface
 - 2.5 Conceitos de vetores e coleções
 - 2.6 Manipulação de Strings
 - 2.7 Tratamento de Exceções
 - 2.8. Arquivos e Fluxos
3. Conectividade JDBC
 - 3.1 Conexão com o Banco de Dados
 - 3.2 Manipulação dos dados
 - 3.3 DataSource
4. Componentes Visuais (Swing)
5. Implementação de um sistema comercial em Java

Aula 06

*Objetivos: Interfaces Swing (GUI) - Parte 1. Componentes essenciais (rótulo, campos, botões).
Tratamento de Eventos.*

Interfaces Gráficas / GUI

- Mecanismos amigáveis para interação com o usuário.
- Componentes GUI, também chamados de controles ou widgets, são objetos que interagem com o usuário através de algum dispositivo de entrada.
- GUIs podem ser construídas via comandos ou IDEs.
- Exemplos pacotes GUI para Java: AWT, **Swing** e Java FX.
- GUIs devem ser propostas de modo a serem intuitivas ao usuário e condizentes com a necessidade dos usuários.

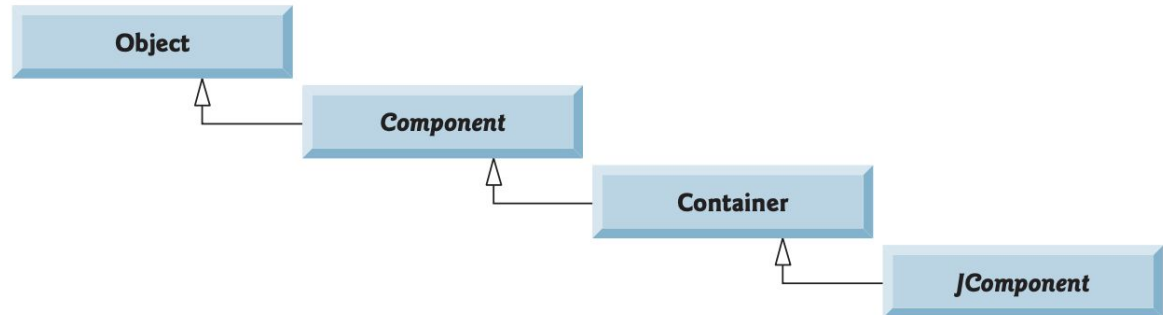
AWT (java.awt)

- **Abstract Window Toolkit:** "aparência" nativa das plataformas em que executam, ou seja, botões rodando em aplicações no Windows se parecem com os componentes do Windows e o mesmo botão tem a aparência do MacOS quando rodando nos dispositivos da Apple.
- Costumam ser aplicações mais "pesadas".
- Existente desde as primeiras versões do Java, com isso, deu origem às demais GUI.

awt.Component -> awt.Container -> swing.JComponent

Swing (javax.swing)

- Possui base nos componentes AWT, porém mais leve e com mais funcionalidades.
- Performance maior do que AWT.
- Possui interface e interação independente da plataforma em que está executando.
- Suporte a MVC.



Swing

- Qualquer objeto que é um **Container** (pacote `java.awt`) pode ser usado para organizar *Components* anexando os *Components* ao **Container**.
- Containers podem compor outros containers.
- `JComponent` é uma subclasse de `Container`.
- `JComponent` é a superclasse dos componentes Swing.
- Componentes podem ter suporte a personalização de sua aparência, recursos de atalho via teclado, suporte a acessibilidade e localização, etc.

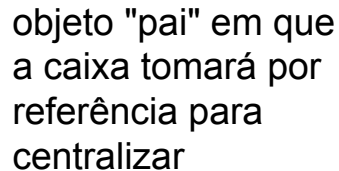
Componentes Swing

- **JLabel**: exibição de caixa de texto (não editável)
- **TextField**: caixa de texto utilizada para receber entrada do usuário, podendo ser editável ou estática
- **Button**: botão que dispara evento após clique
- **CheckBox**: opção que pode ser selecionada ou não
- **ComboBox**: lista (dropdown) para seleção pelo usuário
- **List**: lista de itens que podem ser selecionados (múltiplas seleções)
- **Panel**: área para organizar outros componentes





JOptionPane

- Exibição de caixa de diálogo para os usuários;
- Pode ser utilizada para receber uma confirmação/interação com os usuários;
- Podem ser exibidas caixas de diferentes propósitos: confirmação (Confirm), entrada (Input), aviso (Message), opção (Option).
- Exemplo de chamada:

```
JOptionPane.showMessageDialog(null, "Mensagem") ;
```



objeto "pai" em que
a caixa tomará por
referência para
centralizar

Tipo de diálogo de mensagem	Ícone	Descrição
ERROR_MESSAGE		Indica um erro.
INFORMATION_MESSAGE		Indica uma mensagem informativa.
WARNING_MESSAGE		Alerta de um potencial problema.
QUESTION_MESSAGE		Faz uma pergunta. Normalmente, esse diálogo exige uma resposta, como clicar em um botão Yes ou No.
PLAIN_MESSAGE	Sem ícone	Um diálogo que contém uma mensagem, mas nenhum ícone.

Exemplo

```
import java.util.Scanner;
import javax.swing.JOptionPane;
public class Swing0 {

    public static void main(String[] args) {

        String nome;
        Scanner scan = new Scanner(System.in);
        System.out.print("Digite seu nome: ");
        nome = scan.nextLine();
        JOptionPane.showMessageDialog(null, "Olá, " + nome);
        scan.close();

    }

}
```

Exemplo

```
import javax.swing.JOptionPane;

public class Swing1 {

    public static void main(String[] args) {

        String nome;
        nome = JOptionPane.showInputDialog("Digite seu nome: ");
        JOptionPane.showMessageDialog(null, "Olá, " + nome);

    }

}
```

Swing1.java

Exemplo

```
String nome, nota_string;  
int nota1, nota2;  
nome = JOptionPane.showInputDialog("Digite o nome do aluno: ");  
nota_string = JOptionPane.showInputDialog(String.format("Digite  
a primeira nota do aluno %s: ", nome));  
nota1 = Integer.parseInt(nota_string);  
nota_string = JOptionPane.showInputDialog(String.format("Digite  
a segunda nota do aluno %s: ", nome));  
nota2 = Integer.parseInt(nota_string);  
float media = (float) (nota1+nota2) / 2;  
JOptionPane.showMessageDialog(null, String.format("A média do  
aluno %s foi %.2f", nome, media));
```

Swing2.java

```
entrada = JOptionPane.showConfirmDialog(null, "Mensagem de confirmação");
    entrada = JOptionPane.showConfirmDialog(null, "Mensagem de confirmação", "Este
parâmetro é do título da mensagem", JOptionPane.YES_NO_CANCEL_OPTION);
    entrada = JOptionPane.showConfirmDialog(null, "Mensagem de confirmação",
"Sim/Não", JOptionPane.YES_NO_OPTION);
    entrada = JOptionPane.showConfirmDialog(null, "Mensagem de confirmação",
"Ok/Cancelar", JOptionPane.OK_CANCEL_OPTION);
    entrada = JOptionPane.showConfirmDialog(null, "Vamos ver os ícones?", "Opção
única", JOptionPane.DEFAULT_OPTION);
    entrada = JOptionPane.showConfirmDialog(null, "Mensagem de confirmação",
"Sim/Não/Cancelar (simples)", JOptionPane.YES_NO_CANCEL_OPTION,
JOptionPane.PLAIN_MESSAGE);
    entrada = JOptionPane.showConfirmDialog(null, "Mensagem de confirmação",
"Sim/Não/Cancelar (aviso)", JOptionPane.YES_NO_CANCEL_OPTION,
JOptionPane.WARNING_MESSAGE);
    entrada = JOptionPane.showConfirmDialog(null, "Mensagem de confirmação",
"Sim/Não/Cancelar (erro)", JOptionPane.YES_NO_CANCEL_OPTION, JOptionPane.ERROR_MESSAGE);
    int respostaCerta;
    do {
        respostaCerta = JOptionPane.showConfirmDialog(null, "O Brasil vai ser Hexa
em 2022?", "Sim/Não/Cancelar (pergunta)", JOptionPane.YES_NO_OPTION,
JOptionPane.QUESTION_MESSAGE);
    } while(respostaCerta == 1);
    ImageIcon icon = new ImageIcon("images/copa.png");
    JOptionPane.showMessageDialog(null, "Eu também acho!", "Concordo",
JOptionPane.PLAIN_MESSAGE, icon);
```

Swing3.java

```
int entrada;
Integer[] valores = {3, 5, 8, 9, 13};

int resposta;

resposta = JOptionPane.showOptionDialog(null,
    "Qual desses valores é par?",
    "Teste de matemática",
    JOptionPane.DEFAULT_OPTION,
    JOptionPane.INFORMATION_MESSAGE,
    null,
    valores,
    valores[1]);

if (resposta == 2) {
    JOptionPane.showMessageDialog(null, "Parabéns! Você acertou!");
}else {
    JOptionPane.showMessageDialog(null, String.format("Foi escolhida
a resposta %d e a correta é 8.", valores[resposta]));
}
```

JFrame

- JFrame é uma subclasse indireta de `java.awt.Window` e fornece atributos e comportamentos de janelas (barra de título e opções de minimizar, maximizar e fechar janelas).
- Nos nossos exemplos vamos colocar componentes dentro dos frames (`JFrame`).
- Podemos definir comportamentos e atributos para a janela (tamanho, ações, etc).

Exemplo

```
import javax.swing.JFrame;
public class Swing5 {
    public static void main(String[] args) {
        JFrame janela = new JFrame();
        janela.setTitle("Janela JFrame");
        janela.setBounds(500, 400, 300, 300);
        janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        janela.setResizable(false);
        janela.setVisible(true);
    }
}
```

Exemplo

```
import javax.swing.JFrame;
public class Swing6 extends JFrame {
    public Swing6() {
        super("Janela JFrame");
        setBounds(500, 400, 300, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setResizable(false);
        setVisible(true);
    }
    public static void main(String[] args) {
        new Swing6();
    }
}
```

Gerenciadores de Layout

- Componentes são adicionados a um Container (exemplo: `JFrame`).
- Podemos posicionar e redimensionar cada componente individualmente, mas também utilizar gerenciadores de layout que fornecem uma padronização de disposição/organização.
- *Exemplo: `FlowLayout`*: exibe componentes da esquerda para a direita até completar cada linha do Layout. `FlowLayout` também tem mecanismos de reajuste quando a janela é redimensionada.
- Método para definir layout: `setLayout (Objeto Layout)`.

Gerenciadores de Layout

- **FlowLayout:** layout padrão do `JPanel`. Adiciona os componentes sequencialmente da esquerda para a direita na mesma ordem em que são inseridos no layout.
- **BorderLayout:** layout padrão do `JFrame`, organiza o layout em 5 áreas: NORTH, SOUTH, EAST, WEST e CENTER.
- **GridLayout:** dispõe os componentes em linhas e colunas.

JLabel

- Componente para exibição de textos (estáticos) na tela.
- Exemplos de uso: explicação de algum outro componente (rotular um `JTextField`, por exemplo) ou exibir informações para o usuário.
- `JLabels` não podem ser editados via interface gráfica. Suas alterações de conteúdo devem ser feitas via código.
- Pode ser utilizado para exibir imagens na tela.

Exemplo

```
public class LabelFrame extends JFrame {  
    private JLabel lb1, lb2, lb3;  
  
    public LabelFrame() {  
        super("Testando JLabel");  
        setLayout(new FlowLayout( ));  
        lb1 = new JLabel("Primeiro label");  
        lb1.setToolTipText ("Dica para o label");  
        add(lb1);  
        lb2 = new JLabel("Segundo label");  
        lb2.setToolTipText ("Você está no segundo label");  
        add(lb2);  
        lb3 = new JLabel();  
        lb3.setText("Defini o terceiro label");  
        lb3.setHorizontalTextPosition(SwingConstants.RIGHT);  
        add(lb3);  
    }  
}
```

Exemplo

```
import javax.swing.JFrame;

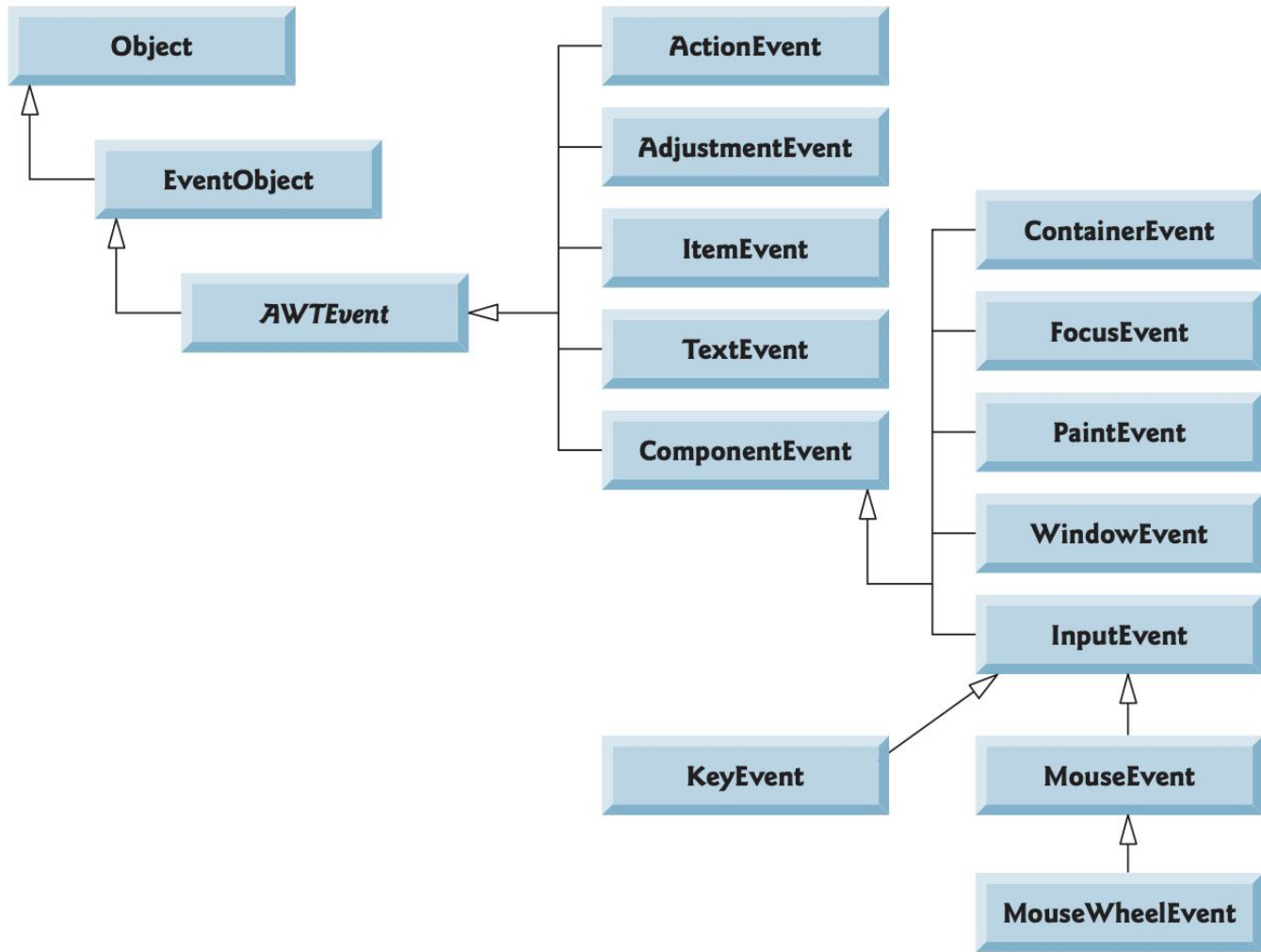
public class LabelFrameTest {
    public static void main(String[] args) {
        LabelFrame labelFrame = new LabelFrame();
        labelFrame. setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        labelFrame.setSize(150,150);
        labelFrame.setVisible(true);
    }
}
```

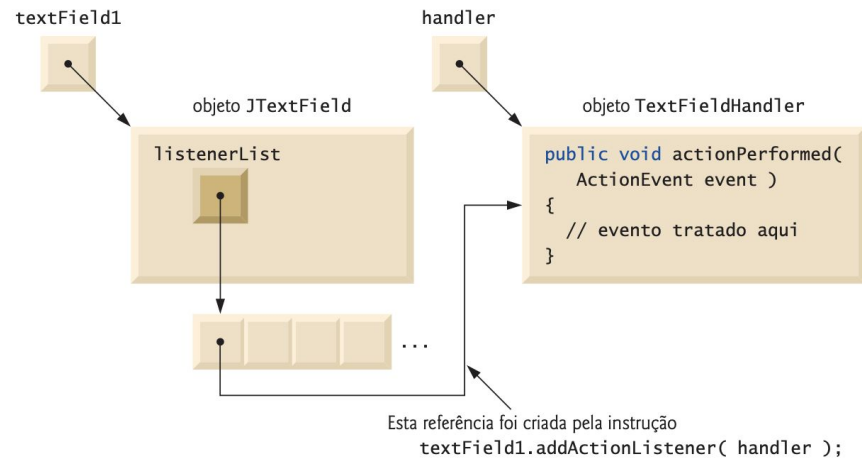
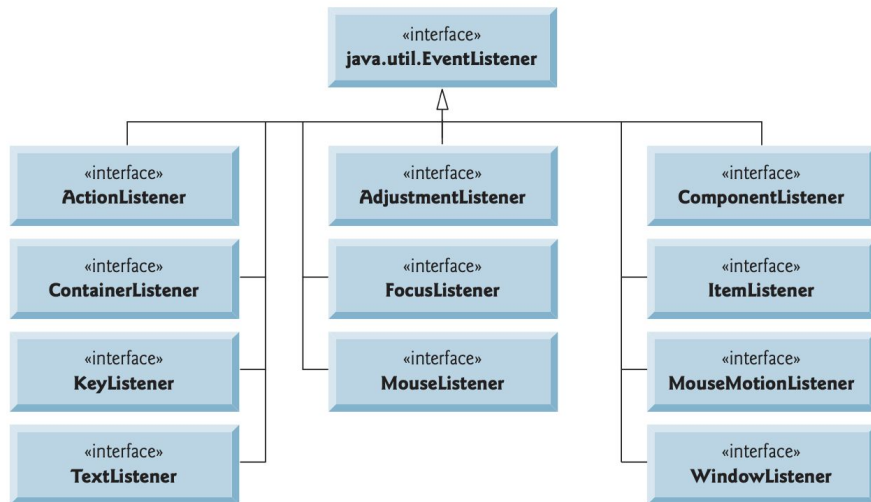
Classes internas

- Podemos ter classes dentro de outras classes. Estas podem ser `static` ou não `static`, este último tipo é conhecido também como classe interna.
- Com isso, os objetos de classes internas serão criados pelas classes de "primeiro nível".
- Objetos de classe interna podem acessar as variáveis e métodos das classes de primeiro nível.
- Classes internas podem ainda ser anônimas (conforme veremos mais adiante).

Ações e eventos de componentes

- Além de manipulações "estáticas", por exemplo: alterar o texto de um label, podemos capturar os eventos de componentes para controlar ações.
- Exemplos: clique de um botão ou "enter" em um JTextField.
- Para lidar com esses tratamentos de eventos devemos ter:
 - 1) uma classe para a rotina de tratamento do evento e que irá "ouvir" a ação esperada (listener);
 - 2) um objeto da classe especificada anteriormente que será notificado quando o evento esperado ocorrer.





TextField

- **JTextField**: interface para usuário digitar conteúdo (String) para, em geral, ser recebida e tratada pelo programa.
- **JPasswordField**: similar a `JTextField` (estende `JTextField`) mas adiciona comportamentos e tratativas para lidar com campos de senha.
- **JTextArea**: exibição (ou entrada) de textos mais longos (múltiplas linhas).
- Método `setEditable(boolean b)` define se o TextField será editável ou não.

Exemplo

```
public class TextFieldFrame extends JFrame{
    private final JTextField textField1;
    private final JTextField textField2;
    private final JTextField textField3;
    private final JPasswordField passwordField;

    public TextFieldFrame(){
        super("Testando JTextField e JPasswordField");
        setLayout(new FlowLayout());
        textField1 = new JTextField(10);
        add(textField1);
        textField2 = new JTextField("Insira seu texto aqui");
        add(textField2);
        textField3 = new JTextField("Texto não editável", 20);
        textField3.setEditable(false);
        add(textField3);
        passwordField = new JPasswordField("Texto escondido");
        add(passwordField);
        TextFieldHandler handler = new TextFieldHandler();
        textField1.addActionListener(handler);
        textField2.addActionListener(handler);
        textField3.addActionListener(handler);
        passwordField.addActionListener(handler);
        setVisible(true);
    }
}
```

Exemplo

```
private class TextFieldHandler implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent event) {
        String string = "";
        if (event.getSource() == textField1)
            string = String.format("textField1: %s", event.getActionCommand());
        else if (event.getSource() == textField2)
            string = String.format("textField2: %s", event.getActionCommand());
        else if (event.getSource() == textField3)
            string = String.format("textField3: %s", event.getActionCommand());
        else if (event.getSource() == passwordField)
            string = String.format("passwordField: %s", event.getActionCommand());
        JOptionPane.showMessageDialog(null, string);
    }

}

public static void main(String[] args) {
    TextFieldFrame textFieldFrame = new TextFieldFrame();
    textFieldFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    textFieldFrame.setSize(350, 100);
    textFieldFrame.setVisible(true);
}

}
```

Exercício

Fazer um validador simples de login e senha.

Vamos ter duas constantes `loginCerto` e `senhaCerta`.

Apresentar ao usuário para preencher dois campos: **login** e **senha**.

O usuário terá 5 tentativas de acesso ao sistema. Se ele errar, exibir o número de tentativas restantes (com mensagem de erro) até acabar as tentativas e sair do sistema.

Se acertar, exibir mensagem de sucesso e também sair ao pressionar ok.

Ao pressionar “enter” no campo de login deverá mudar o foco para o campo de senha.
Se pressionar “enter” no campo de senha, deverá fazer a tentativa de acesso.

Exercício

```
public class Exerciciol extends JFrame{

private final JLabel label1 = new JLabel();
private final JLabel label2 = new JLabel();
private final JTextField fieldLogin;
private final JPasswordField passwordField;

private static final String loginCerto = "usuario";
private static final String senhaCerta = "senha";

private static int tentativasSobrando = 5;

public Exerciciol(){
    super("Acesso ao sistema");
    setLayout(new FlowLayout());

    label1.setText("Login: ");
    add(label1);
    fieldLogin = new JTextField(10);
    add(fieldLogin);

    label2.setText("Senha: ");
    add(label2);
    passwordField = new JPasswordField("",20);
    add(passwordField);

    loginHandler handler = new loginHandler();
    fieldLogin.addActionListener(handler);
    passwordField.addActionListener(handler);

    setVisible(true);
}
```


Exercício

```
private static void validaLogin(String logText, String passText) {
    if (logText.equals(loginCerto) && passText.equals(senhaCerta)) {
        JOptionPane.showMessageDialog(null, "Senha válida!");
        System.exit(1);
    }else {
        tentativasSobrando--;
    }
    if (tentativasSobrando > 0) {
        JOptionPane.showMessageDialog(null, String.format("Senha inválida, você tem mais %d tentativas.",
tentativasSobrando), "Acesso Negado!", JOptionPane.ERROR_MESSAGE);
    } else {
        JOptionPane.showMessageDialog(null, "Acesso inválido. Clique para sair.");
        System.exit(1);
    }
}
private class loginHandler implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent event) {
        if (event.getSource() == fieldLogin)
            passwordField.requestFocus();
        else if (event.getSource() == passwordField) {
            validaLogin(fieldLogin.getText(), String.format("%s", event.getActionCommand()));
        }
    }
}
public static void main(String[] args) {
    Exerciciol ex1 = new Exerciciol();
    ex1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    ex1.setSize(600, 100);
    ex1.setVisible(true);
}
```

JButton

- Implementação de botão em Swing.
- Podemos definir propriedades/comportamentos como tamanho, cor, texto, etc.
- **Evento** "clique" deve ser tratado para que execute ações esperadas.
- Em geral, iremos utilizar as ações de botões para disparar eventos/comportamentos para outros componentes/objetos.

Exemplo

- No exercício anterior:
 - adicionar a validação de senha no botão “Acessar”;
 - adicionar um botão “Limpar” para limpar os campos de login e senha e mudar foco para o campo de login;

```
import javax.swing.JButton;  
  
private final JButton botaoLimpar = new JButton();  
  
private final JButton botaoOk = new JButton();  
  
botaoLimpar.addActionListener(handler);  
  
botaoOk.addActionListener(handler);
```

Exemplo

```
private class loginHandler implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent event) {
        if (event.getSource() == fieldLogin)
            passwordField.requestFocus();
        else if (event.getSource() == passwordField) {
            botaoOk.requestFocus();
        }
        else if (event.getSource() == botaoLimpar) {
            fieldLogin.setText("");
            passwordField.setText("");
            fieldLogin.requestFocus();
        }
        else if (event.getSource() == botaoOk ) {
            validaLogin(fieldLogin.getText(), new String(passwordField.getPassword()));
        }
    }
}
```

Exercício 2

Criar uma tela com um label com o texto "Label 1" e outro label com o texto "Label 2".

Ao clicar em um botão os valores do botões deverão ser trocados (obtendo o valor a partir do label e não com textos pré-definidos).

Exercício 2

```
private final JLabel label1 = new JLabel();
private final JLabel label2 = new JLabel();
private final JButton trocar = new JButton("Trocar");

public Exercicio2(){
    super("Troca texto");
    setLayout(new FlowLayout());
    label1.setText("Label 1");
    add(label1);
    label2.setText("Label 2");
    add(label2);
    add(trocar);

    BotaoHandler handler = new BotaoHandler();
    trocar.addActionListener(handler);

    setVisible(true);
}
```

Exercício 2

```
private class BotaoHandler implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent event) {
        String texto1, texto2;
        if (event.getSource() == trocar) {
            texto1 = new String(label1.getText());
            texto2 = new String(label2.getText());
            label1.setText(texto2);
            label2.setText(texto1);
        }
    }
}
```

JScrollPane

Componente que engloba outros componentes para permitir rolagem nestes.

Ao definir o JScrollPane podemos definir qual componente irá implementar suas funcionalidades.

Com isso, podemos ter um tamanho fixo para o componente e o usuário pode navegar pelo conteúdo com barras de rolagem.

Podemos, ainda, definir comportamentos específicos para a rolagem.

Exemplo: atrelar um `JTextArea` ao `ScrollPane`.


```
public class TextAreaFrame extends JFrame {
    private final JTextArea taOrigem;
    private final JTextArea taDestino;
    private final JButton copiar;
    public TextAreaFrame() {
        super("Testando TextArea e ScrollPane");
        Box box = Box.createHorizontalBox();
        String textoCompleto = "Olá a todos! \n" +
            "este é um textArea que tem \n" +
            "um texto longo que pode ser \n" +
            "copiado para outro textArea!";
        taOrigem = new JTextArea(textoCompleto, 10, 20);
        box.add(new JScrollPane(taOrigem));
        copiar = new JButton("copiar >>");
        box.add(copiar);
        copiar.addActionListener(
            new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    taDestino.setText(taOrigem.getSelectedText());
                }
            }
        );
        taDestino = new JTextArea(10, 20);
        taDestino.setEditable(false);
        box.add(new JScrollPane(taDestino));
        add(box);
    }
}
```

```
public class TestaTextAreaFrame {  
  
    public static void main(String[] args) {  
        TextAreaFrame textAreaFrame = new TextAreaFrame();  
        textAreaFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        textAreaFrame.setSize(425,200);  
        textAreaFrame.setVisible(true);  
    }  
  
}
```

JCheckBox

- Componente utilizado para caixas de seleção/marcação.
- Exibição para o usuário de existência ou não de determinada característica.
- Podemos ter várias caixas de seleção em um mesmo formulário/tela.
- É visto como um botão que mantém seu estado.
- Método comum utilizado no Listener: `isSelected()`

```

private final JTextField textField;
private final JCheckBox cbNegrito;
private final JCheckBox cbItalico;
public CheckBoxFrame() {
    super("Seleções com JCheckBox");
    setLayout(new FlowLayout());
    textField = new JTextField("Observe as mudanças na fonte", 20);
    add(textField);
    cbNegrito = new JCheckBox("Negrito");
    cbItalico = new JCheckBox("Itálico");
    add(cbNegrito);
    add(cbItalico);
    CheckBoxHandler cbHandler = new CheckBoxHandler();
    cbNegrito.addItemListener(cbHandler);
    cbItalico.addItemListener(cbHandler);
}
private class CheckBoxHandler implements ItemListener{
    Font fonte = null;
    @Override
    public void itemStateChanged(ItemEvent e) {
        if (cbNegrito.isSelected() && cbItalico.isSelected())
            fonte = new Font("Serif", Font.BOLD + Font.ITALIC, 14);
        else if (cbNegrito.isSelected())
            fonte = new Font("Serif", Font.BOLD, 14);
        else if (cbItalico.isSelected())
            fonte = new Font("Serif", Font.ITALIC, 14);
        else
            fonte = new Font("Serif", Font.PLAIN, 14);
        textField.setFont(fonte);
    }
}

```

Exemplo

```
public class TestaTextAreaFrame {  
  
    public static void main(String[] args) {  
        TextAreaFrame textAreaFrame = new TextAreaFrame();  
        textAreaFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        textAreaFrame.setSize(425,200);  
        textAreaFrame.setVisible(true);  
    }  
  
}
```

Exercício 3

Criar uma tela com um checkbox e um botão.

Ao clicar no botão deve-se exibir em uma caixa de diálogo indicando quantas vezes o checkbox foi clicado (independe do estado).

```

public class Exercicio3 {

    private final static JCheckBox cbMarcador = new JCheckBox();
    private final static JButton btnContador = new JButton("Exibir Contagem");
    private static int contagem = 0;

    private static class Ex3Handler implements ActionListener{
        @Override
        public void actionPerformed(ActionEvent e) {
            if (e.getSource() == cbMarcador) {
                contagem++;
            } else if (e.getSource() == btnContador) {
                JOptionPane.showMessageDialog(null, String.format("Marcador foi clicado %d vezes", contagem));
            }
        }
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame("Contador de cliques");
        frame.setLayout(new FlowLayout());
        frame.add(cbMarcador);
        frame.add(btnContador);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(350, 100);
        Ex3Handler handler = new Ex3Handler();
        cbMarcador.addActionListener(handler);
        btnContador.addActionListener(handler);
        frame.setVisible(true);
    }
}

```

JRadioButton

- Componente de seleção única (entre mais de uma opção).
- Utilizado em um grupo de escolhas que é controlado por um `ButtonGroup` (que é um objeto invisível ao usuário).
- Com isso, o método `add` do `ButtonGroup` é utilizado para alocar os `JRadioButton`.
- Assim como `JCheckBox`, `JRadioButton` é um botão (ou grupo de botões) com estado mantido após a seleção.


```
public RadioButtonFrame() {
    super("Seleções com RadioButton");
    setLayout(new FlowLayout());
    textField = new JTextField("Observe as mudanças na fonte", 25);
    add(textField);
    rbSimples = new JRadioButton("Simples", true);
    rbNegrito = new JRadioButton("Negrito", false);
    rbItalico = new JRadioButton("Itálico", false);
    rbNegritoItalico = new JRadioButton("Negrito e Itálico", false);
    rbGrupo = new ButtonGroup();
    rbGrupo.add(rbSimples);
    rbGrupo.add(rbNegrito);
    rbGrupo.add(rbItalico);
    rbGrupo.add(rbNegritoItalico);
    add(rbSimples);
    add(rbNegrito);
    add(rbItalico);
    add(rbNegritoItalico);
    fonteSimples = new Font("Serif", Font.PLAIN, 14);
    fonteNegrito = new Font("Serif", Font.BOLD, 14);
    fonteItalico = new Font("Serif", Font.ITALIC, 14);
    fonteNegritoItalico = new Font("Serif", Font.BOLD + Font.ITALIC, 14);
    rbSimples.addItemListener(new RadioHandler(fonteSimples));
    rbNegrito.addItemListener(new RadioHandler(fonteNegrito));
    rbItalico.addItemListener(new RadioHandler(fonteItalico));
    rbNegritoItalico.addItemListener(new RadioHandler(fonteNegritoItalico));
}
```

```
private class RadioHandler implements ItemListener{
    private Font fonte;
    public RadioHandler(Font f) {
        fonte = f;
    }
    @Override
    public void itemStateChanged(ItemEvent e) {
        textField.setFont(fonte);
    }
}
```

RadioButtonFrame.java

```
public class TestaRadioButton {
```

TestaRadioButton.java

```
    public static void main(String[] args) {
        RadioButtonFrame rbFrame = new RadioButtonFrame();
        rbFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        rbFrame.setSize(350, 100);
        rbFrame.setVisible(true);
    }
}
```

JComboBox

- Seleção de um item em uma lista (dropdown).
- `JComboBox` cria uma representação `String` de cada objeto que está associado à `JComboBox`.
- Com isso, `JComboBox` manipula itens que podem ser adicionados através do método `addItem(Object)` e deixar um item pré-selecionado com o método `setSelectedItem(index)`

```

public class ComboBoxFrame extends JFrame{
    private final JComboBox<String> comboCores;
    private static final String[] cores = {"Azul", "Amarelo", "Vermelho", "Branco",
"Verde"};
    private static final Color[] listaCores = {Color.BLUE, Color.YELLOW, Color.RED,
Color.WHITE, Color.GREEN};
    public ComboBoxFrame() {
        super("Seleção Cores");
        setLayout(new FlowLayout());
        getContentPane().setBackground(Color.BLUE);
        comboCores = new JComboBox<String>(cores);
        comboCores.setMaximumRowCount(3);
        comboCores.addItemListener(
            new ItemListener() {
                public void itemStateChanged(ItemEvent e) {
                    if (e.getStateChange() == ItemEvent.SELECTED)
                        getContentPane().setBackground(listaCores[comboCores.getSelected
Index()])
                }
            }
        );
        add(comboCores);
    }
}

```

```
public class TestaComboBox {  
  
    public static void main(String[] args) {  
        ComboBoxFrame comboFrame = new ComboBoxFrame();  
        comboFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        comboFrame.setSize(500,500);  
        comboFrame.setVisible(true);  
  
    }  
  
}
```

JList

- Lista com valores que podem ser selecionados.
- Permite visualização de diversos itens pelos usuários.
- Pode-se ainda selecionar e emitir ações a partir das seleções.
- Evento de seleção: `ListSelectionEvent`
- Listas podem ser de seleção única (`SINGLE_SELECTION`), com um único intervalo (`SINGLE_INTERVAL_SELECTION`) ou múltiplos intervalos (`MULTIPLE_INTERVAL_SELECTION`).

```

public class ListFrame extends JFrame {
    private final JList<String> listaCores;
    private final Color[] refCores = {Color.BLUE, Color.YELLOW, Color.RED,
Color.WHITE, Color.GREEN};
    private final String[] todasCores = {"Azul", "Amarelo", "Vermelho", "Branco",
"Verde"};
    public ListFrame() {
        super("Listagem de cores");
        setLayout(new FlowLayout());
        listaCores = new JList<String>(todasCores);
        listaCores.setVisibleRowCount(3);
        listaCores.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        add(new JScrollPane(listaCores));
        listaCores.addListSelectionListener(
            new ListSelectionListener() {
                @Override
                public void valueChanged(ListSelectionEvent e) {
                    getContentPane().setBackground(refCores[listaCores.getSelectedIndex()])
                }
            }
        );
    }
}

```

```
public class TestaListFrame {  
  
    public static void main(String[] args) {  
        ListFrame lf = new ListFrame();  
        lf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        lf.setSize(400,200);  
        lf.setVisible(true);  
  
    }  
  
}
```


Exercício 4

Implementar um programa para o usuário adivinhar números.

Deve-se obter um número inteiro aleatório (classe Random, método nextInt()) entre 0 e 100.

Se o usuário acertar o número, mostrar mensagem e sair do programa.

Se errar, dizer se o número correto está acima ou abaixo e voltar pra tela.

```

public Exercicio4() {
    super("Adivinha Número");
    setLayout(new FlowLayout());
    textNumero = new JTextField(15);
    Random rand = new Random();
    numero = rand.nextInt(100);
    botao = new JButton("Dar palpite");
    botao.addActionListener(new BotaoHandler());
    add(textNumero);
    add(botao);
}

private class BotaoHandler implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        int palpite = Integer.parseInt(textNumero.getText());
        if (palpite == numero) {
            JOptionPane.showMessageDialog(null, "Você acertou!");
            System.exit(1);
        } else if (palpite > numero) {
            JOptionPane.showMessageDialog(null, "O número é menor do que " + palpite, "Errou",
JOptionPane.ERROR_MESSAGE);
        } else if (palpite < numero) {
            JOptionPane.showMessageDialog(null, "O número é maior do que " + palpite, "Errou",
JOptionPane.ERROR_MESSAGE);
        }
        textNumero.requestFocus();
    }
}

```