|  | FTCE  Faculdade de Tecnologia e Ciências Exatas |
| --- | --- |

USJT – 2021/1 – Programação de Soluções Computacionais

**Aula 11 – Conexão com banco de dados e interface gráfica (GUI)**

**1 Introdução a Interfaces gráficas**

Neste material daremos continuidade no desenvolvimento da aplicação da última aula. Começamos implementando a funcionalidade de login e ilustramos o uso de outros componentes visuais do pacote javax.swing.

**2 Desenvolvimento**

**2.1 (Implementando a funcionalidade de login)** Os dados de usuários do sistema serão armazenados em uma base relacional gerenciada pelo MySQL. Como sabemos, a fim de obter esses dados, a aplicação Java precisa estabelecer uma conexão com o MySQL Server, o que pode ser feito utilizando a API JDBC.

- Começamos criando um database para o sistema. No Workbench, uma vez conectado com o MySQL Server, use

**CREATE DATABASE nome\_do\_seu\_db;**

**USE nome\_do\_seu\_db;**

para criar o novo database e informar ao MySQL Server que as próximas instruções deverão ter impacto sobre ele.

- A seguir, crie uma tabela para armazenar os dados de usuários com

**CREATE TABLE tb\_usuario (id INT PRIMARY KEY AUTO\_INCREMENT, nome VARCHAR(200), senha VARCHAR(200));**

- Faça a inserção de um usuário com

**INSERT INTO tb\_usuario (nome, senha) VALUES (‘admin’, ‘admin’);**

- Segundo o princípio conhecido como **alta coesão**, cada classe que criamos deve ter um único propósito, uma única razão de ser. Sendo assim, criaremos uma classe cuja única responsabilidade será a de gerenciar conexões com o banco. Veja a Listagem 2.1.1.

Listagem 2.1.1

| import java.sql.Connection;  import java.sql.DriverManager;  public class ConexaoBD {  private static String host = "localhost";  private static String porta = "3306";  private static String db = "**seubd**";  private static String usuario = "**seusuario**";  private static String senha = "**suasenha**";    public Connection obtemConexao (){  try{  Connection c = DriverManager.getConnection(  "jdbc:mysql://" + host + ":" + porta + "/" + bd,  usuario,  senha  );  return c;  }  catch (Exception e){  e.printStackTrace();  return null;  }  }  } |
| --- |

- Lembre-se de abrir o arquivo **pom.xml** e especificar que o driver do MySQL deve ser baixado pelo Maven. O ajuste a ser feito é exibido na Listagem 2.1.2.

Listagem 2.1.2

| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>br.com.bossini</groupId>  <artifactId>pessoal\_sistema\_academico\_com\_netbeans\_para\_montar\_pdf</artifactId>  <version>1.0-SNAPSHOT</version>  <packaging>jar</packaging>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <maven.compiler.source>14</maven.compiler.source>  <maven.compiler.target>14</maven.compiler.target>  </properties>  <dependencies>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>8.0.20</version>  </dependency>  </dependencies>  </project> |
| --- |

- Um sistema desenvolvido com linguagem que tem suporte à orientação a objetos é uma representação simplificada do mundo real (lembra do **minimundo**?). Assim, vamos criar uma classe para representar o que é um usuário do sistema. No momento, usuários possuem apenas duas coisas de interesse: login e senha. Veja a definição da classe que descreve o que é um usuário na Listagem 2.1.3.

Listagem 2.1.3

| public class Usuario {    private String nome;  private String senha;  public Usuario(String nome, String senha) {  this.nome = nome;  this.senha = senha;  }  public String getNome() {  return nome;  }  public void setNome(String nome) {  this.nome = nome;  }    public String getSenha() {  return senha;  }  public void setSenha(String senha) {  this.senha = senha;  }  } |
| --- |

- A seguir, vamos criar uma classe que será responsável por todas as operações de persistência de dados da aplicação. Embora não seja a melhor prática possível, fazê-lo nesse momento tende a dar origem a código de mais fácil compreensão. No futuro (em semestres mais avançados) aprenderemos a escrever código mais organizado, que usa padrões de projeto e que tende a ter maior nível de reusabilidade e tende a ser mais fácil de se manter. Neste momento, o que nos é mais importante é a simplicidade. Assim, crie a classe da Listagem 2.1.4. No momento, o único método que ela possui se encarrega de verificar se um determinado usuário existe ou não na base de dados.

**Nota:** DAO é um acrônimo para **Data Access Object**, ou seja, Objeto de Acesso aos Dados. Trata-se de um dos padrões de desenvolvimento de software mais antigos. Uma classe DAO tem a finalidade de encapsular código de acesso a bases de dados. É comum que um projeto possua muitas classes DAO, cada qual apropriada para a manipulação de diferentes tipos de objetos. Por simplicidade, como mencionado, teremos uma única classe DAO.

Listagem 2.1.4

| public class DAO {  public boolean existe (Usuario usuario) throws Exception{  String sql = "SELECT \* FROM tb\_usuario WHERE nome = ? AND senha = ?";  try (Connection conn = ConexaoBD.obterConexao();  PreparedStatement ps = conn.prepareStatement(sql)){  ps.setString(1, usuario.getNome());  ps.setString(2, usuario.getSenha());  try (ResultSet rs = ps.executeQuery()){  return rs.next();  }  }  }  } |
| --- |

**2.2 (Criando a tela de login - revisão)** A tela do sistema permitirá que o usuário insira seus dados de acesso. Para criá-la, clique com o direito no pacote principal da aplicação e escolha **New >> JFrame Form**. Seu nome será **LoginTela**.

**2.3 (Campo para login)** O usuário irá digitar seu login em um JTextField. Note que há uma paleta de componentes à direita. Arraste um componente do tipo **Text Field** para a tela. Faça os seguintes ajustes:

- **Largura**: 270

- **Altura**: 54

- Clique com o direito, escolha **Edit Text** e apague o texto que ele exibe por padrão.

- Clique com o direito, escolha **Change Variable Name** e digite **loginTextField**.

- Mantenha-o selecionado e veja suas propriedades na parte inferior direita da tela. Encontre a propriedade **border**. Escolha **Titled Border** e digite **Digite seu login** no campo **Title**.

**2.4 (Campo para senha)** Arraste e solte um componente do tipo **Password Field** logo **abaixo** do **loginTextField**. Faça os seguintes ajustes:

- Posição, largura e altura iguais aos do **loginTextField**.

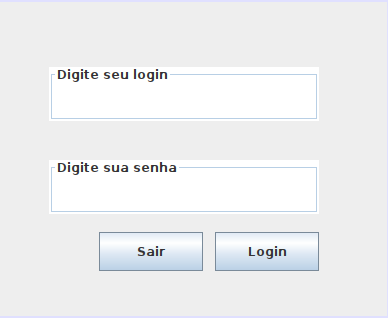
- Clique com o direito, escolha **Edit Text** e apague o texto que ele exibe por padrão.

- Clique com o direito, escolha **Change Variable Name** e digite **senhaPasswordField**.

- Mantenha-o selecionado e veja suas propriedades na parte inferior direita da tela. Encontre a propriedade **border**. Escolha **Titled Border** e digite **Digite sua senha** no campo **Title**.

**2.5 (Botões para sair e para fazer login)** Arraste e solte dois componentes **Button** e faça ajustes para que o resultado seja parecido com o que exibe a Figura 2.4.5.1.

Figura 2.5.1



- Clique com o direito em cada botão, escolha **Change Variable Name** e altere seus nomes para **sairButton** e **loginButton.**

**2.6 (Tratando o evento “clique” dos botões)** Há uma tarefa específica a ser executada quando cada um dos botões for clicado.

- No caso do botão **sairButton**, apenas desejamos encerrar a aplicação. Para isso, basta clicar duas vezes sobre ele e completar o corpo do método que irá aparecer, como mostra a Listagem 2.6.1. Note que o registro do observador já foi feito automaticamente.

Listagem 2.6.1

| private void sairButtonActionPerformed(java.awt.event.ActionEvent evt) {  this.dispose();  } |
| --- |

- O método acionado quando o botão de login é clicado será cliente do método existe. Assim ele passa a validar os dados do usuário em função do que existe realmente na base. Veja a Listagem 2.6.2.

Listagem 2.6.2

| private void loginButtonActionPerformed(java.awt.event.ActionEvent evt) {  //pega o login do usuário  String login = loginTextField.getText();  //pega a senha do usuário como char[] e converte para String  String senha = new String (senhaPasswordField.getPassword());    try{  //verifica se o usuário é válido  Usuario usuario = new Usuario (login, senha);  DAO dao = new DAO();  if (dao.existe(usuario)){  JOptionPane.showMessageDialog (null, "Bem vindo, " + usuario.getNome() + "!");  }  else{  JOptionPane.showMessageDialog(null, "Usuário inválido");  }  }  catch (Exception e){  JOptionPane.showMessageDialog (null, "Problemas técnicos. Tente novamente mais tarde");  e.printStackTrace();  }    } |
| --- |

***Referências***

DEITEL, P. e DEITEL, H. **Java Como Programar**. 8ª Edição. São Paulo, SP: Pearson, 2010.