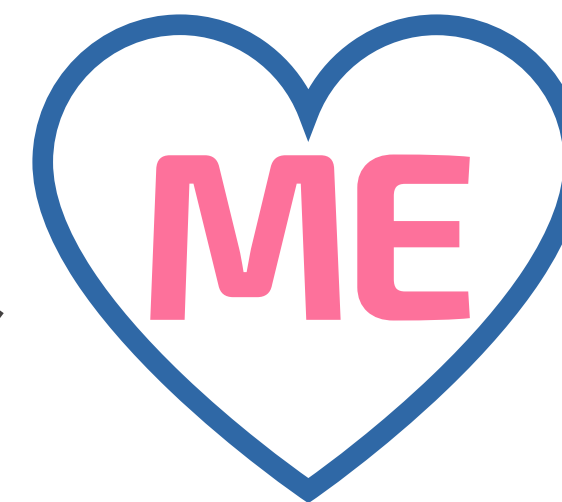


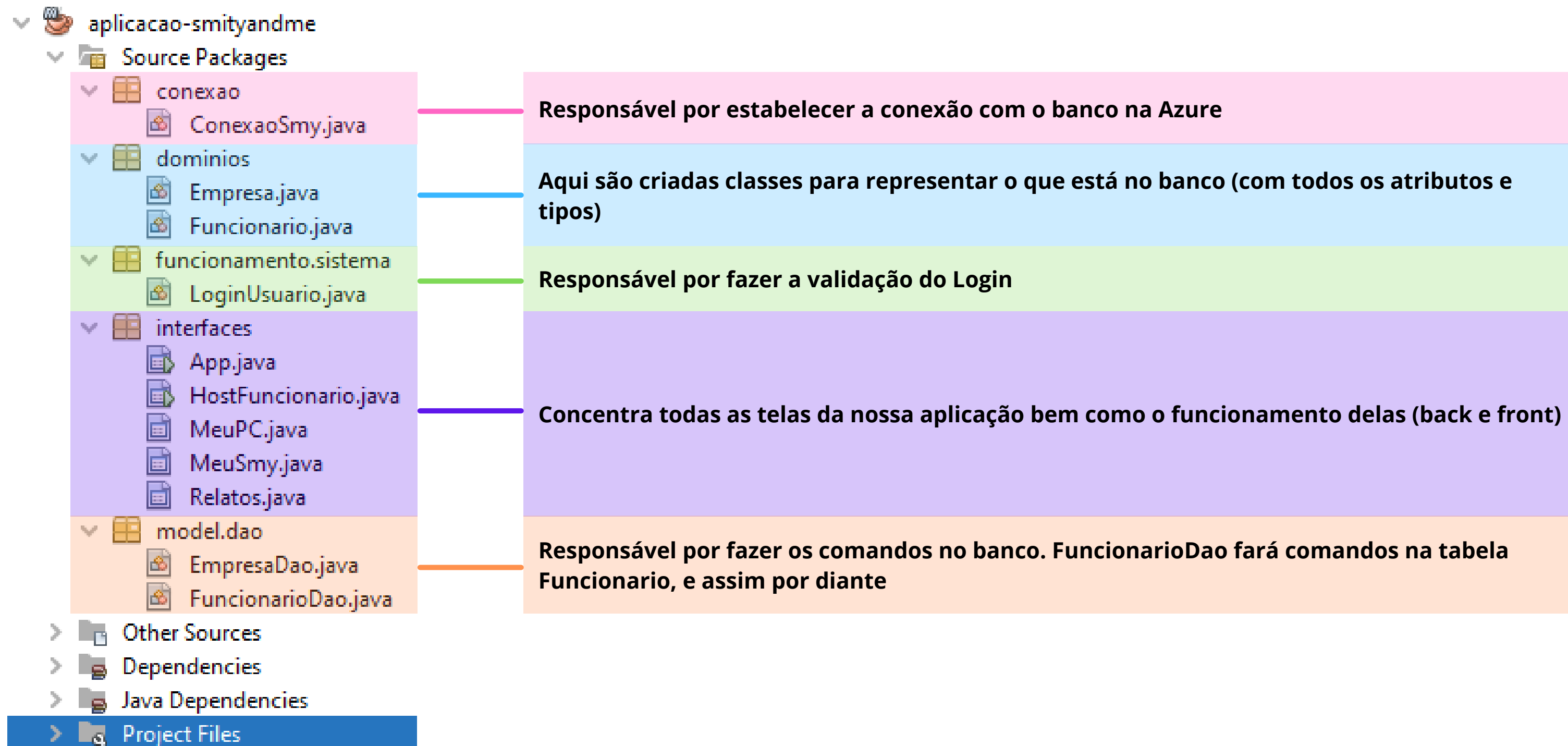


SMITY &

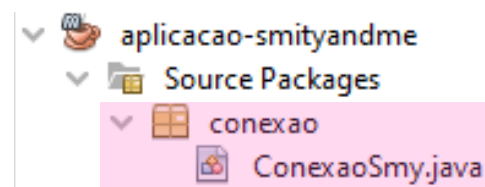


Documentação - Java

# Organização do projeto



# Pacote "conexao"



Responsável por estabelecer a conexão com o banco na Azure

```
public class ConexaoSmy {
```

```
    // Connect to your database.  
    // Replace server name, username, and password with your credentials
```

```
    String connectionUrl = null;
```

```
    Connection connection = null;
```

```
    public ConexaoSmy() {
```

```
        connectionUrl
```

```
        = "jdbc:sqlserver://bdprojeto2sem.database.windows.net:1433;"  
        + "database=bdprojeto2sem;"  
        + "user=smy;"  
        + "password=#Gfgrupo4;"  
        + "encrypt=true;"  
        + "trustServerCertificate=false;"  
        + "hostNameInCertificate=*.database.windows.net;"  
        + "loginTimeout=30;"
```

```
    }
```

```
    public Connection Conectar() {
```

```
        try {
```

```
            connection = DriverManager.getConnection(getConnectionUrl());
```

```
            System.out.println("Conexão com Azure estabelecida com sucesso!");
```

```
        } // Lida com todos os erros que podem ter ocorrido
```

```
        catch (SQLException e) {
```

```
            System.err.println("Erro ao conectar-se à Azure. Tente novamente mais tarde");
```

```
        }
```

```
        return connection;
```

```
    }
```

```
    public String getConnectionUrl() {
```

```
        return connectionUrl;
```

```
    }
```

```
    public void setConnectionUrl(String connectionUrl) {
```

```
        this.connectionUrl = connectionUrl;
```

```
    }
```

```
}
```

GETTERS e SETTERS

**connectionUrl** -> só pra armazenar os parâmetros da conexão

Connection é uma classe que faz parte da dependência JDBC que o Diego e a Giu passaram. O objeto "connection" que vai fazer a conexão

Aqui atribuímos um valor para o connection. No caso, o "DriverManager" (gerenciador de driver) também faz parte da classe connection e ele que vai escolher qual driver precisa usar para o JDBC (se é MySQL, SQL Server, h2 e tals). No nosso caso, é SQL Server pq está na Azure. O DriverManager tem um método chamado "getConnection", que recebe como parâmetro todos aqueles atributos da conexão que colocamos na variável connectionUrl e, com eles, vai tentar acessar o banco

```
try {
```

(vai tentar fazer todos os códigos que estão aqui. No nosso caso, quando chama DriverManager.getConnection ele vai tentar fazer a conexão com o banco)

```
}
```

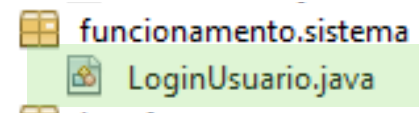
```
catch (SQLException e) {
```

Caso a parte ali do try dê algum erro de conexão do SQL, ele vai pegar esse erro (que aparece com letras vermelhas no console) e vai tirar ele do console. Então você pode colocar alguma mensagem bonitinha no lugar, que é o que fizemos ali. SQLException é um tipo de erro quando ele não consegue acessar o SQL, então o "e" que tá ali é tipo uma variável do tipo SQLException. Você pode usar o "e" lá no print pra printar o erro, mas não é necessário

```
}
```

Retorna a conexão que foi estabelecida

# Pacote "funcionamento.sistema"



Responsável por fazer a validação do Login

```
/**
 *
 * @author Victor
 */
public class LoginUsuario {

    private Boolean loginValidado = false;

    public Funcionario validarLogin(Funcionario usuario, String email, String senha) {

        if (usuario.getIdFuncionario() == null) {
            JOptionPane.showMessageDialog(null, "Email incorreto, tente novamente.");
        } else {
            if ((usuario.getEmailFuncionario().equals(email)) && (usuario.getSenhaFuncionario().equals(senha))) {
                // passar usuario, login e senha
                loginValidado = true;
                HostFuncionario telaInicio = new HostFuncionario(usuario);
                telaInicio.setVisible(true);
            } else {
                JOptionPane.showMessageDialog(null, "Senha incorreta, tente novamente.");
            }
        }

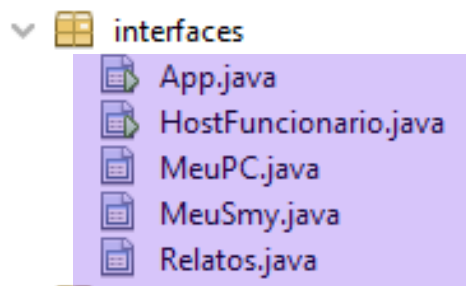
        return usuario;
    }

    public Boolean getLoginValidado() {
        return loginValidado;
    }
}
```

Recebe como parâmetro um objeto funcionário (que é construído após o select no banco), o email digitado no input e a senha digitada no input

PROCESSO DE VALIDAÇÃO SERÁ  
EXPLICADO MAIS PRA FRENTE

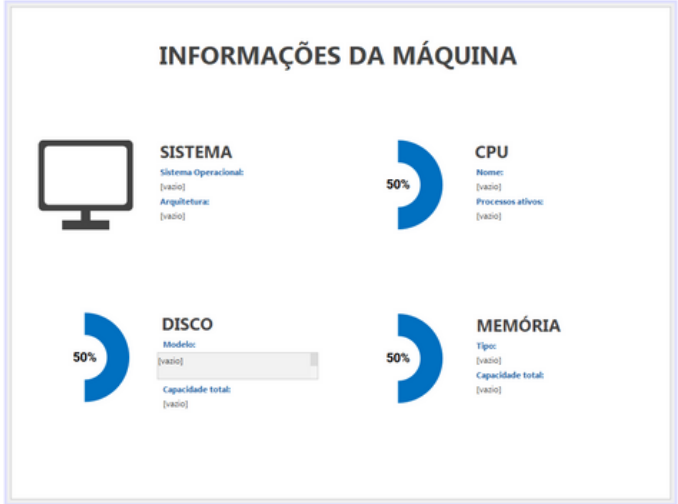
# Pacote "interfaces"



Concentra todas as telas da nossa aplicação bem como o funcionamento delas (back e front)



App.java (login)



MeuPC.java



Relatos.java

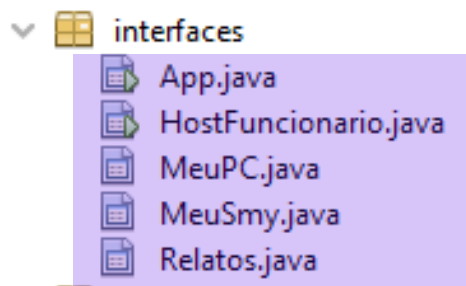


HostFuncionario.java



MeuSmy.java

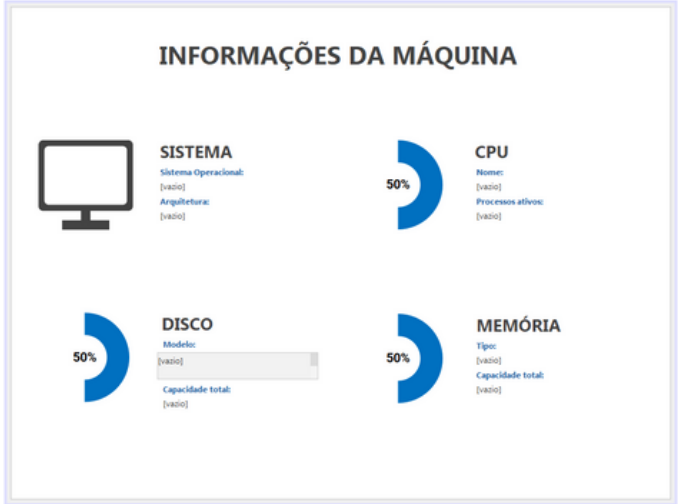
# Pacote "interfaces"



Concentra todas as telas da nossa aplicação bem como o funcionamento delas (back e front)



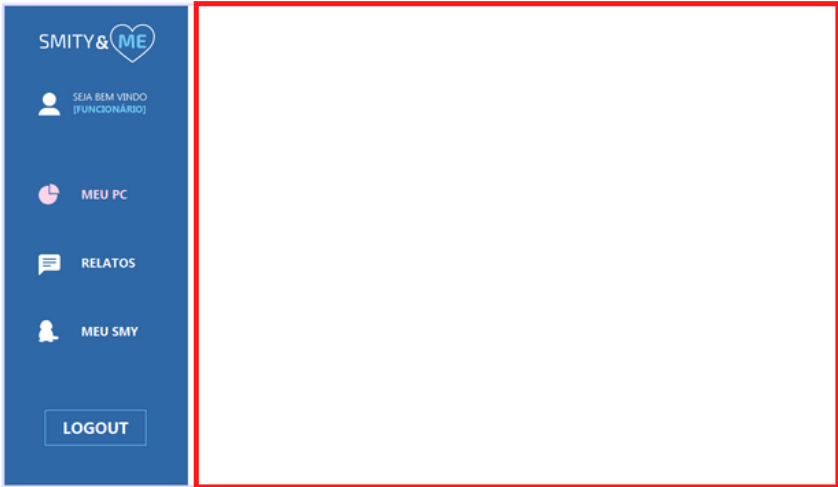
JFRAME



JInternalFrame



JInternalFrame



JFRAME



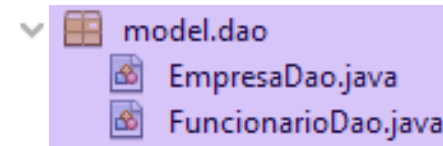
JInternalFrame

Eles são JInternalFrame porque vão aparecer no lugar do JDesktopPane da tela de HostFuncionario (quadrado com contorno vermelho ao lado). Quando clica no menu de navegação da dash, ele vai esconder ou mostrar a tela. Mostra a tela que foi clicada e esconde as demais, e por aí vai

Essa tela em branco é um JDesktopPane (ele serve como um recipiente para outras telas. No caso, será um recipiente para as telas MeuPC, MeuSmy e Relatos)



# Pacote "model.dao"



Responsável por fazer os comandos no banco. FuncionarioDao fará comandos na tabela Funcionario, e assim por diante

```
public class FuncionarioDao {

    ConexaoSmy conexao = new ConexaoSmy();
    Connection con = conexao.Conectar();
    PreparedStatement pstmt = null;
    ResultSet resultadoQuery = null;

    public Funcionario buscarDados(Funcionario usuarioQuerendoLogar) {
        Funcionario usuario = usuarioQuerendoLogar;

        try {
            String queryEmail = "SELECT * FROM funcionario WHERE emailFuncionario = ?";
            pstmt = con.prepareStatement(queryEmail);
            // O setString vai pegar como parâmetro os ? que você coloca na query
            // e vai definir o valor dela no segundo parâmetro. Se eu tivesse
            // colocado dois ??, teria que fazer o setString(1, valor) e setString(2, valor)
            pstmt.setString(1, usuario.getEmailFuncionario());
            resultadoQuery = pstmt.executeQuery();

            while (resultadoQuery.next()) {
                usuario.setIdFuncionario(resultadoQuery.getInt("idFuncionario"));
                usuario.setNomeFuncionario(resultadoQuery.getString("nomeFuncionario"));
                usuario.setLoginFuncionario(resultadoQuery.getString("loginFuncionario"));
                usuario.setSenhaFuncionario(resultadoQuery.getString("senhaFuncionario"));
                usuario.setEmailFuncionario(resultadoQuery.getString("emailFuncionario"));
                usuario.setTelefoneFuncionario(resultadoQuery.getString("telefoneFuncionario"));
                usuario.setPermissaoAdmin(resultadoQuery.getInt("permissaoAdmin"));
                usuario.setFkEmpresa(resultadoQuery.getInt("fkEmpresa"));
            }

        } catch (SQLException e) {
            System.err.println("Query malsucedida");
        }
        return usuario;
    }
}
```

Variável conexao do tipo ConexaoSmy

Variável con do tipo Connection vai pegar o conexao e chamar o método conectar (que retorna o resultado da conexão, como visto anteriormente, isto é, se deu certo ou errado).

PreparedStatement é uma classe que vai armazenar comandos do SQL para, depois, serem executados. Tem que usar isso porque ele tem métodos que personalizam o script e o executam. Então criou-se um objeto do tipo PreparedStatement chamado pstmt.

ResultSet é uma classe do java que vai armazenar o resultado de uma query de um banco de dados, isto é, vai armazenar a tabela resultado do comando executado. Foi criado um objeto resultadoQuery para armazenar esse resultado

try catch já foi explicado. Vai tentar fazer o select, aí se não encontrar o email no banco ou se o comando estiver escrito errado, o catch vai pegar esse erro e mostrar bonitinho pro usuário como "Query malsucedida".

queryEmail é só uma string pra armazenar o comando  
pstmt = con.prepareStatement(queryEmail) vai chamar o método prepareStatement (que já existe por padrão) e vai passar como parâmetro o queryEmail que é a query em si. Até então, estamos preparando a query para ser executada. Como tem o "?" na query, precisamos definir o que vai ser esse "?". E é por isso que usamos o pstmt.setString. Ele tem o 1 porque a gente tá se referindo ao primeiro ? que aparece, aí o usuario.getEmailFuncionario() é uma string do emailFuncionario. Ou seja, o ? agora terá o valor de emailFuncionario.

Por fim, a query deve ser executada e seu resultado armazenado. resultadoQuery vai receber o resultado da execução (que é feita com pstmt.executeQuery). Se a query der certo, vai entrar no laço while. O ".next()" é um método do resultadoQuery (classe ResultSet) que vai repetir os comandos que estão dentro do while até que haja um próximo registro no banco (executa tudo para o primeiro item de id 1, aí se tiver um segundo de id 2 executa pra ele também, e assim por diante). O que esses comandos fazem é pegar o resultado da query e passar para o objeto usuario que foi criado logo abaixo do (public Funcionario buscarDados)

# Validação do Login

Para fazer a validação do Login, precisamos das 4 classes e do JFrame da tela de Login abaixo:

```
public class ConexaoSmy {

    // Connect to your database.
    // Replace server name, username, and password with your credentials
    String connectionUrl = null;
    Connection connection = null;

    public ConexaoSmy() {
        connectionUrl
            = "jdbc:sqlserver://bdprojeto2sem.database.windows.net:1433;"
            + "database=bdprojeto2sem;"
            + "user=smy;"
            + "password=#Gfggrupo4;"
            + "encrypt=true;"
            + "trustServerCertificate=false;"
            + "hostNameInCertificate=.database.windows.net;"
            + "loginTimeout=30;";
    }

    public Connection Conectar() {
        try {
            connection = DriverManager.getConnection(getConnectionUrl());
            System.out.println("Conexão com Azure estabelecida com sucesso!");
        } // Lida com todos os erros que podem ter ocorrido
        catch (SQLException e) {
            System.err.println("Erro ao conectar-se à Azure. Tente novamente mais tarde");
        }
        return connection;
    }

    // public static void main(String[] args) {
    //     ConexaoFactory c = new ConexaoFactory();
    //     c.Conectar();
    // }

    public String getConnectionUrl() {
        return connectionUrl;
    }

    public void setConnectionUrl(String connectionUrl) {
        this.connectionUrl = connectionUrl;
    }

}
```

Classe ConexaoSmy

```
public class Funcionario {

    Integer idFuncionario;
    String nomeFuncionario;
    String loginFuncionario;
    String senhaFuncionario;
    String emailFuncionario;
    String telefoneFuncionario;
    Integer permissaoAdmin;
    Integer fkEmpresa;

    public Integer getIdFuncionario() {
        return idFuncionario;
    }

    public void setIdFuncionario(Integer idFuncionario) {
        this.idFuncionario = idFuncionario;
    }

    public String getNomeFuncionario() {
        return nomeFuncionario;
    }

    public void setNomeFuncionario(String nomeFuncionario) {
        this.nomeFuncionario = nomeFuncionario;
    }

    public String getLoginFuncionario() {
        return loginFuncionario;
    }

    public void setLoginFuncionario(String loginFuncionario) {
        this.loginFuncionario = loginFuncionario;
    }

    public String getSenhaFuncionario() {
        return senhaFuncionario;
    }

    public void setSenhaFuncionario(String senhaFuncionario) {
```

Funcionario

```
public class FuncionarioDao {

    ConexaoSmy conexao = new ConexaoSmy();
    Connection con = conexao.Conectar();
    PreparedStatement pstmt = null;
    ResultSet resultadoQuery = null;

    public Funcionario buscarDados(Funcionario usuarioQuerendoLogar) {
        Funcionario usuario = usuarioQuerendoLogar;

        try {
            String queryEmail = "SELECT * FROM funcionario WHERE emailFuncionario = ?";
            pstmt = con.prepareStatement(queryEmail);
            // O setString vai pegar como parâmetro os ? que você coloca na query
            // e vai definir o valor dela no segundo parâmetro. Se eu tivesse
            // colocado dois ??, teria que fazer o setString(1, valor) e setString(2, valor)
            pstmt.setString(1, usuario.getEmailFuncionario());
            resultadoQuery = pstmt.executeQuery();

            while (resultadoQuery.next()) {
                usuario.setIdFuncionario(resultadoQuery.getInt("idFuncionario"));
                usuario.setNomeFuncionario(resultadoQuery.getString("nomeFuncionario"));
                usuario.setLoginFuncionario(resultadoQuery.getString("loginFuncionario"));
                usuario.setSenhaFuncionario(resultadoQuery.getString("senhaFuncionario"));
                usuario.setEmailFuncionario(resultadoQuery.getString("emailFuncionario"));
                usuario.setTelefoneFuncionario(resultadoQuery.getString("telefoneFuncionario"));
                usuario.setPermissaoAdmin(resultadoQuery.getInt("permissaoAdmin"));
                usuario.setFkEmpresa(resultadoQuery.getInt("fkEmpresa"));
            }

        } catch (SQLException e) {
            System.err.println("Query malsucedida");
        }
        return usuario;
    }

}
```

FuncionarioDao

```
public class LoginUsuario {

    private Boolean loginValidado = false;

    public Funcionario validarLogin(Funcionario usuario, String email, String senha) {

        if (usuario.getIdFuncionario() == null) {
            JOptionPane.showMessageDialog(null, "Email incorreto, tente novamente.");
        } else {
            if ((usuario.getEmailFuncionario().equals(email)) && (usuario.getSenhaFuncionario().equals(senha))) {
                // passar usuario, login e senha
                loginValidado = true;
                HostFuncionario telaInicio = new HostFuncionario(usuario);
                telaInicio.setVisible(true);
            } else {
                JOptionPane.showMessageDialog(null, "Senha incorreta, tente novamente.");
            }
        }

        return usuario;
    }

    public Boolean getLoginValidado() {
        return loginValidado;
    }

}
```

LoginUsuario

```
private void btnEntrarActionPerformed(java.awt.event.ActionEvent evt) {

    if (!(txtValorEmail.getText().equalsIgnoreCase("")) && !String.valueOf(txtValorSenha.getPassword()).equalsIgnoreCase("")) {
        FuncionarioDao funcionarioDao = new FuncionarioDao();
        Funcionario funcionario = new Funcionario();
        LoginUsuario tentativaDeLogin = new LoginUsuario();

        funcionario.setEmailFuncionario(txtValorEmail.getText());
        funcionarioDao.buscarDados(funcionario);

        try {
            tentativaDeLogin.validarLogin(funcionario, String.valueOf(txtValorEmail.getText()), String.valueOf(txtValorSenha.getPassword()));
            if (tentativaDeLogin.getLoginValidado()) {
                this.dispose();
            }
        } catch (Exception e) {
            System.out.println(e);
        }
    } else {
        if (txtValorEmail.getText().equals("")) {
            JOptionPane.showMessageDialog(null, "Digite seu e-mail!");
        } else {
            JOptionPane.showMessageDialog(null, "Digite sua senha!");
        }
    }

}
```

Tela de login (App.java), mais especificamente a função chamada quando clica no botão "Entrar"



# Validação do Login

## Utilização da Classe ConexaoSmy

```
public class ConexaoSmy {

    // Connect to your database.
    // Replace server name, username, and password with your credentials
    String connectionUrl = null;
    Connection connection = null;

    public ConexaoSmy() {
        connectionUrl
            = "jdbc:sqlserver://bdprojeto2sem.database.windows.net:1433;"
            + "database=bdprojeto2sem;"
            + "user=smy;"
            + "password=#Gfgrupo4;"
            + "encrypt=true;"
            + "trustServerCertificate=false;"
            + "hostNameInCertificate=*.database.windows.net;"
            + "loginTimeout=30;";
    }

    public Connection Conectar() {
        try {
            connection = DriverManager.getConnection(getConnectionUrl());
            System.out.println("Conexão com Azure estabelecida com sucesso!");
        } // Lida com todos os erros que podem ter ocorrido
        catch (SQLException e) {
            System.err.println("Erro ao conectar-se à Azure. Tente novamente mais tarde");
        }
        return connection;
    }

    // public static void main(String[] args) {
    //     ConexaoFactory c = new ConexaoFactory();
    //     c.Conectar();
    // }

    public String getConnectionUrl() {
        return connectionUrl;
    }

    public void setConnectionUrl(String connectionUrl) {
        this.connectionUrl = connectionUrl;
    }

}
```

A utilização da classe ConexaoSmy já foi explicada anteriormente. De forma resumida, é ela que estabelece a conexão com o nosso banco de dados SQL Server hospedado na Azure

# Validação do Login

## Utilização da Classe Funcionario

```
public class Funcionario {

    Integer idFuncionario;
    String nomeFuncionario;
    String loginFuncionario;
    String senhaFuncionario;
    String emailFuncionario;
    String telefoneFuncionario;
    Integer permissaoAdmin;
    Integer fkEmpresa;

    public Integer getIdFuncionario() {
        return idFuncionario;
    }

    public void setIdFuncionario(Integer idFuncionario) {
        this.idFuncionario = idFuncionario;
    }

    public String getNomeFuncionario() {
        return nomeFuncionario;
    }

    public void setNomeFuncionario(String nomeFuncionario) {
        this.nomeFuncionario = nomeFuncionario;
    }

    public String getLoginFuncionario() {
        return loginFuncionario;
    }

    public void setLoginFuncionario(String loginFuncionario) {
        this.loginFuncionario = loginFuncionario;
    }

    public String getSenhaFuncionario() {
        return senhaFuncionario;
    }

    public void setSenhaFuncionario(String senhaFuncionario) {
```

A utilização da classe Funcionario já foi explicada anteriormente. De forma resumida, vai representar o que está no banco (com todos os atributos e tipos iguais). Vai servir para armazenar as informações do funcionário caso o select no banco com o email e senha fornecidos tenha retorno positivo

# Validação do Login

## Utilização da Classe FuncionarioDao

```
public class FuncionarioDao {

    ConexaoSmy conexao = new ConexaoSmy();
    Connection con = conexao.Conectar();
    PreparedStatement pstm = null;
    ResultSet resultadoQuery = null;

    public Funcionario buscarDados(Funcionario usuarioQuerendoLogar) {
        Funcionario usuario = usuarioQuerendoLogar;

        try {
            String queryEmail = "SELECT * FROM funcionario WHERE emailFuncionario = ?";
            pstm = con.prepareStatement(queryEmail);
            // O setString vai pegar como parâmetro os ? que você coloca na query
            // e vai definir o valor dela no segundo parâmetro. Se eu tivesse
            // colocado dois ??, teria que fazer o setString(1, valor) e setString(2, valor)
            pstm.setString(1, usuario.getEmailFuncionario());
            resultadoQuery = pstm.executeQuery();

            while (resultadoQuery.next()) {
                usuario.setIdFuncionario(resultadoQuery.getInt("idFuncionario"));
                usuario.setNomeFuncionario(resultadoQuery.getString("nomeFuncionario"));
                usuario.setLoginFuncionario(resultadoQuery.getString("loginFuncionario"));
                usuario.setSenhaFuncionario(resultadoQuery.getString("senhaFuncionario"));
                usuario.setEmailFuncionario(resultadoQuery.getString("emailFuncionario"));
                usuario.setTelefoneFuncionario(resultadoQuery.getString("telefoneFuncionario"));
                usuario.setPermissaoAdmin(resultadoQuery.getInt("permissaoAdmin"));
                usuario.setFkEmpresa(resultadoQuery.getInt("fkEmpresa"));
            }

        } catch (SQLException e) {
            System.err.println("Query malsucedida");
        }
        return usuario;
    }
}
```

Responsável por fazer os comandos no banco. FuncionarioDao fará comandos na tabela Funcionario e, caso o retorno seja positivo (isto é, exista no banco), vai preencher as informações de todos os atributos do objeto usuario da classe Funcionario

Atenção ao método **buscarDados**, ele é super importante. Note que ele **recebe como parâmetro um objeto (usuarioQuerendoLogar) do tipo Funcionario**

# Validação do Login

## Utilização da Classe LoginUsuario

```
public class LoginUsuario {

    private Boolean loginValidado = false;

    public Funcionario validarLogin(Funcionario usuario, String email, String senha) {

        if (usuario.getIdFuncionario() == null) {
            JOptionPane.showMessageDialog(null, "Email incorreto, tente novamente.");
        } else {
            if ((usuario.getEmailFuncionario().equals(email)) && (usuario.getSenhaFuncionario().equals(senha))) {
                // passar usuario, login e senha
                loginValidado = true;
                HostFuncionario telaInicio = new HostFuncionario(usuario);
                telaInicio.setVisible(true);
            } else {
                JOptionPane.showMessageDialog(null, "Senha incorreta, tente novamente.");
            }
        }

        return usuario;
    }

    public Boolean getLoginValidado() {
        return loginValidado;
    }

}
```

Vai validar se os campos de e-mail e de senha digitados na tela de login existem no banco de dados. Em caso afirmativo, efetua o login e direciona para a tela de monitoramento. Em caso negativo, mostra uma janela com mensagem de erro (e-mail ou senha incorretos). Possui um método "getLoginValidado" para saber se o login foi feito com sucesso ou não

Atenção ao método **validarLogin**, ele é super importante. Note que ele **recebe como parâmetro um objeto (usuario) do tipo Funcionario, um email e uma senha (que são os inputs do usuário na tela de login)**

O getter **getLoginValidado** também é importante, pois **retorna se o login foi um sucesso ou não**. Isso será usado na função de clique do botão "Entrar" na tela de login

# Validação do Login

## Utilização do JFrame App

```
190 private void btnEntrarActionPerformed(java.awt.event.ActionEvent evt) {
191
192     if ((!txtValorEmail.getText().equalsIgnoreCase("") && !String.valueOf(txtValorSenha.getPassword()).equalsIgnoreCase("")) {
193         FuncionarioDao funcionarioDao = new FuncionarioDao();
194         Funcionario funcionario = new Funcionario();
195         LoginUsuario tentativaDeLogin = new LoginUsuario();
196
197         funcionario.setEmailFuncionario(txtValorEmail.getText());
198         funcionarioDao.buscarDados(funcionario);
199
200         try {
201             tentativaDeLogin.validarLogin(funcionario, String.valueOf(txtValorEmail.getText()), String.valueOf(txtValorSenha.getPassword()));
202             if (tentativaDeLogin.getLoginValidado()) {
203                 this.dispose();
204             }
205         } catch (Exception e) {
206             System.out.println(e);
207         }
208     } else {
209         if (txtValorEmail.getText().equals("")) {
210             JOptionPane.showMessageDialog(null, "Digite seu e-mail!");
211         } else {
212             JOptionPane.showMessageDialog(null, "Digite sua senha!");
213         }
214     }
215 }
```



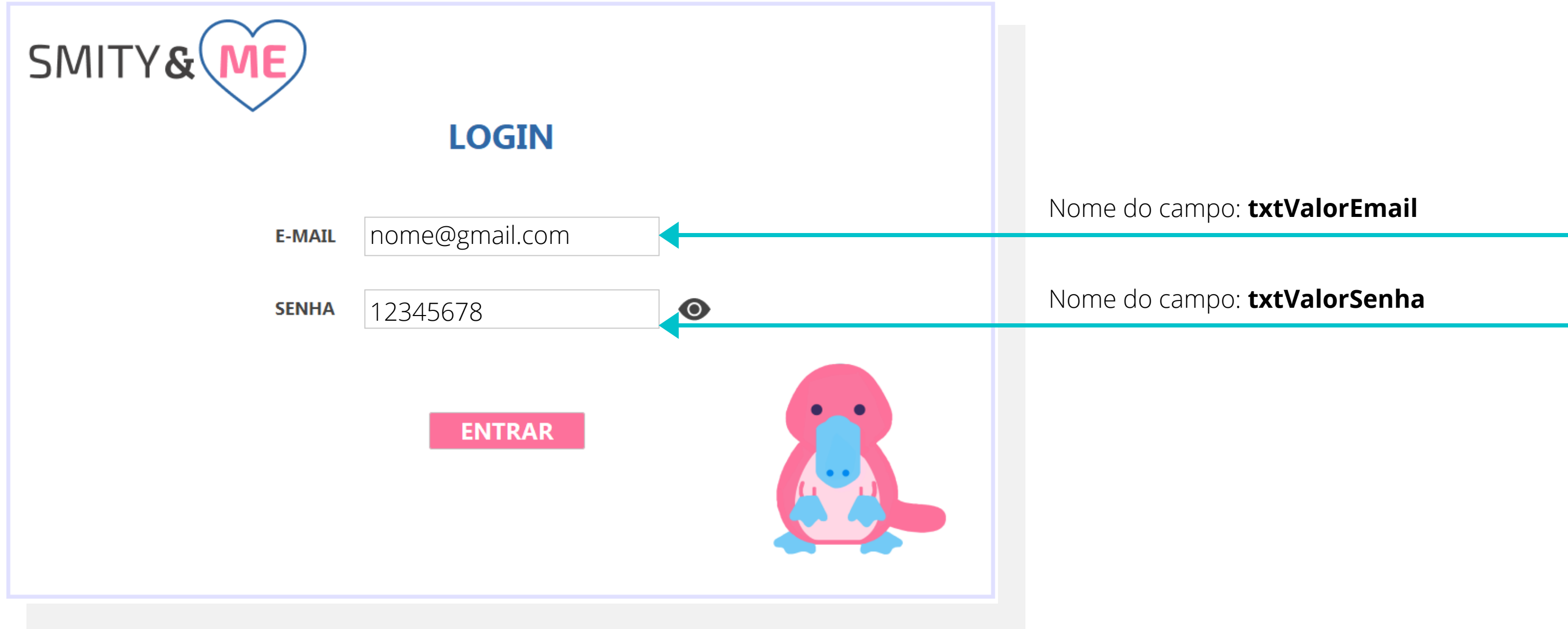
Ao clicar no botão "Entrar", todo o código acima é executado. E é exatamente aqui que começaremos a explicação do Login



# Validação do Login

## Explicação do fluxo de login

**Primeiro passo:** usuário tenta fazer o login. Digita seu e-mail e sua senha e aperta o botão "Entrar"



The image shows a login form for a brand named "SMITY & ME". The form is titled "LOGIN" and contains two input fields: "E-MAIL" and "SENHA". The "E-MAIL" field contains the text "nome@gmail.com" and is pointed to by a red arrow with the label "Nome do campo: **txtValorEmail**". The "SENHA" field contains the text "12345678" and is pointed to by a red arrow with the label "Nome do campo: **txtValorSenha**". Below the input fields is a red button labeled "ENTRAR". To the right of the button is a cartoon illustration of a pink dinosaur. The form is set against a light gray background with a white border.

SMITY & ME

LOGIN

E-MAIL nome@gmail.com

SENHA 12345678

ENTRAR

Nome do campo: **txtValorEmail**

Nome do campo: **txtValorSenha**

# Validação do Login

## Explicação do fluxo de login

**Segundo passo:** clicando no botão entrar, é chamada a função (btnEntrarActionPerformed) atrelada ao clique dele. Essa função é criada automaticamente pelo java swing, por isso o nome grande. Só se preocupe com o conteúdo dela

```
190 private void btnEntrarActionPerformed(java.awt.event.ActionEvent evt) {  
191  
192     if ((!txtValorEmail.getText().equalsIgnoreCase("") && !String.valueOf(txtValorSenha.getPassword()).equalsIgnoreCase("")) {  
193         FuncionarioDao funcionarioDao = new FuncionarioDao();  
194         Funcionario funcionario = new Funcionario();  
195         LoginUsuario tentativaDeLogin = new LoginUsuario();  
196  
197         funcionario.setEmailFuncionario(txtValorEmail.getText());  
198         funcionarioDao.buscarDados(funcionario);  
199  
200         try {  
201             tentativaDeLogin.validarLogin(funcionario, String.valueOf(txtValorEmail.getText()), String.valueOf(txtValorSenha.getPassword()));  
202             if (tentativaDeLogin.getLoginValidado()) {  
203                 this.dispose();  
204             }  
205         } catch (Exception e) {  
206             System.out.println(e);  
207         }  
208     } else {  
209         if (txtValorEmail.getText().equals("")) {  
210             JOptionPane.showMessageDialog(null, "Digite seu e-mail!");  
211         } else {  
212             JOptionPane.showMessageDialog(null, "Digite sua senha!");  
213         }  
214     }  
}
```

Validação do Login

Explicação do fluxo de login

**Terceiro passo:** a primeira coisa que a função irá fazer, antes de tudo, é verificar se o usuário deixou o campo de e-mail ou o de senha em branco (linha 192). Para o campo senha, é necessário converter para String (por isso o uso de String.valueOf), já que ele é um vetor de caracteres, então não tem como comparar com um vazio "" sem antes converter para string

```
192 |         if ((!txtValorEmail.getText().equalsIgnoreCase("")) && !String.valueOf(txtValorSenha.getPassword()).equalsIgnoreCase("")) {
```

**Quarto passo:** caso nenhum dos campos seja vazio, ele entra no if e executa as linhas de 193 a 198. Caso um deles seja vazio, vai para o else na linha 209. Se o e-mail for vazio, mostra um popup falando para digitar o email. Caso a senha seja vazia, mostra um popup pedindo para digitar a senha. Caso os dois sejam vazios, vai sempre pedir o e-mail primeiro

```
193 |         FuncionarioDao funcionarioDao = new FuncionarioDao();
194 |         Funcionario funcionario = new Funcionario();
195 |         LoginUsuario tentativaDeLogin = new LoginUsuario();
196 |
197 |         funcionario.setEmailFuncionario(txtValorEmail.getText());
198 |         funcionarioDao.buscarDados(funcionario);
```

- FuncionarioDao funcionarioDao = new FuncionarioDao() ➡ objeto funcionarioDao sendo criado para poder acessar a classe FuncionarioDao e fazer as consultas na tabela Funcionario
- Funcionario funcionario = new Funcionario() ➡ objeto funcionario sendo criado para armazenar as informações do funcionário que está fazendo login
- LoginUsuario tentativaDeLogin = new LoginUsuario() ➡ objeto tentativaDeLogin sendo criado para poder acessar o método de validação de login dentro da classe LoginUsuario

# Validação do Login

## Explicação do fluxo de login

**Quarto passo:** caso nenhum dos campos seja vazio, ele entra no if e executa as linhas de 193 a 198

```
193 FuncionarioDao funcionarioDao = new FuncionarioDao();
194 Funcionario funcionario = new Funcionario();
195 LoginUsuario tentativaDeLogin = new LoginUsuario();
196
197 funcionario.setEmailFuncionario(txtValorEmail.getText());
198 funcionarioDao.buscarDados(funcionario);
```

funcionario.setEmailFuncionario(txtValorEmail.getText())



está colocando no atributo emailFuncionario do funcionario criado na linha 194 o email que foi digitado no input lá no primeiro passo

funcionarioDao.buscarDados(funcionario)



está chamando o método buscarDados da classe FuncionarioDao e passando como parâmetro o funcionario criado na linha 194. Vale lembrar que, até agora, este funcionário só tem um de seus atributos preenchidos (o emailFuncionario). Todos os demais atributos estão nulos ainda

Vamos para a classe FuncionarioDao agora para analisar o que vai acontecer quando o método buscarDados é chamado

# Validação do Login

## Explicação do fluxo de login

**Quinto passo:** o método buscarDados vai fazer a consulta no banco e ver se esse e-mail existe no banco de dados.

```
21 public Funcionario buscarDados(Funcionario usuarioQuerendoLogar) {
22     Funcionario usuario = usuarioQuerendoLogar;
23
24     try {
25         String queryEmail = "SELECT * FROM funcionario WHERE emailFuncionario = ?";
26         pstmt = con.prepareStatement(queryEmail);
27         // O setString vai pegar como parâmetro os ? que você coloca na query
28         // e vai definir o valor dela no segundo parâmetro. Se eu tivesse
29         // colocado dois ??, teria que fazer o setString(1, valor) e setString(2, valor)
30         pstmt.setString(1, usuario.getEmailFuncionario());
31         resultadoQuery = pstmt.executeQuery();
32
33         while (resultadoQuery.next()) {
34             usuario.setIdFuncionario(resultadoQuery.getInt("idFuncionario"));
35             usuario.setNomeFuncionario(resultadoQuery.getString("nomeFuncionario"));
36             usuario.setLoginFuncionario(resultadoQuery.getString("loginFuncionario"));
37             usuario.setSenhaFuncionario(resultadoQuery.getString("senhaFuncionario"));
38             usuario.setEmailFuncionario(resultadoQuery.getString("emailFuncionario"));
39             usuario.setTelefoneFuncionario(resultadoQuery.getString("telefoneFuncionario"));
40             usuario.setPermissaoAdmin(resultadoQuery.getInt("permissaoAdmin"));
41             usuario.setFkEmpresa(resultadoQuery.getInt("fkEmpresa"));
42         }
43
44     } catch (SQLException e) {
45         System.err.println("Query malsucedida");
46     }
47     return usuario;
48
49 }
```



# Validação do Login

## Explicação do fluxo de login

Este método exige um parâmetro (definido como `usuarioQuerendoLogar`, do tipo `Funcionario`) para ser chamado. Quando ele foi chamado lá na função do botão "Entrar", foi passado o objeto `funcionario` com somente o e-mail preenchido. Todos os outros atributos são nulos. Na linha 22, foi criado um novo objeto do tipo `Funcionario` (`usuario`) para facilitar as validações. Logo na criação, já estamos falando que ele vai ser igual ao `usuarioQuerendoLogar` (que foi o objeto passado como parâmetro na tela anterior). Desta forma, já é possível notar que o `usuario` também terá apenas o seu e-mail preenchido

```
21 public Funcionario buscarDados(Funcionario usuarioQuerendoLogar) {  
22     Funcionario usuario = usuarioQuerendoLogar;
```

As linhas 26 a 31 são apenas formalidades para preparar o select que será feito no banco. Note que, na linha 30, estamos definindo que o ? do select será o e-mail do objeto `Funcionario` que criamos (`usuario`). Lembre-se que o `usuario` possui apenas um email por enquanto (o que foi digitado na tela de login)

```
24     try {  
25         String queryEmail = "SELECT * FROM funcionario WHERE emailFuncionario = ?";  
26         pstmt = con.prepareStatement(queryEmail);  
27         // O setString vai pegar como parâmetro os ? que você coloca na query  
28         // e vai definir o valor dela no segundo parâmetro. Se eu tivesse  
29         // colocado dois ??, teria que fazer o setString(1, valor) e setString(2, valor)  
30         pstmt.setString(1, usuario.getEmailFuncionario());  
31         resultadoQuery = pstmt.executeQuery();  
32  
33         while (resultadoQuery.next()) {  
34             usuario.setIdFuncionario(resultadoQuery.getInt("idFuncionario"));  
35             usuario.setNomeFuncionario(resultadoQuery.getString("nomeFuncionario"));  
36             usuario.setLoginFuncionario(resultadoQuery.getString("loginFuncionario"));  
37             usuario.setSenhaFuncionario(resultadoQuery.getString("senhaFuncionario"));  
38             usuario.setEmailFuncionario(resultadoQuery.getString("emailFuncionario"));  
39             usuario.setTelefoneFuncionario(resultadoQuery.getString("telefoneFuncionario"));  
40             usuario.setPermissaoAdmin(resultadoQuery.getInt("permissaoAdmin"));  
41             usuario.setFkEmpresa(resultadoQuery.getInt("fkEmpresa"));  
42         }
```

# Validação do Login

## Explicação do fluxo de login

Como todos estes comandos estão dentro de um "try", isso quer dizer que o código vai tentar executar os comandos, mas caso ocorra algum erro (por exemplo, o select dar erro porque o email não existe no banco), aí o código já pula direto para o catch, que está na linha 44. Bom, caso o select seja positivo (isto é, o email existe lá no banco de dados), o código vai executar a linha 33 (que também já foi explicada anteriormente). O que as linhas 34 a 41 estão fazendo é preencher o resto das informações deste objeto usuario (que antes estavam nulas) com os valores que estão lá no banco. Terminados os preenchimentos, o código vai para a linha 47, retornando o usuario que agora já tem todas as informações preenchidas. Este retorno permite que o objeto usuario possa ser utilizado em outros métodos. Desta forma, agora temos um objeto do tipo Funcionario com TODAS as informações preenchidas. **VAMOS VOLTAR PARA A TELA DE LOGIN AGORA**

```
24 try {
25     String queryEmail = "SELECT * FROM funcionario WHERE emailFuncionario = ?";
26     pstmt = con.prepareStatement(queryEmail);
27     // O setString vai pegar como parâmetro os ? que você coloca na query
28     // e vai definir o valor dela no segundo parâmetro. Se eu tivesse
29     // colocado dois ??, teria que fazer o setString(1, valor) e setString(2, valor)
30     pstmt.setString(1, usuario.getEmailFuncionario());
31     resultadoQuery = pstmt.executeQuery();
32
33     while (resultadoQuery.next()) {
34         usuario.setIdFuncionario(resultadoQuery.getInt("idFuncionario"));
35         usuario.setNomeFuncionario(resultadoQuery.getString("nomeFuncionario"));
36         usuario.setLoginFuncionario(resultadoQuery.getString("loginFuncionario"));
37         usuario.setSenhaFuncionario(resultadoQuery.getString("senhaFuncionario"));
38         usuario.setEmailFuncionario(resultadoQuery.getString("emailFuncionario"));
39         usuario.setTelefoneFuncionario(resultadoQuery.getString("telefoneFuncionario"));
40         usuario.setPermissaoAdmin(resultadoQuery.getInt("permissaoAdmin"));
41         usuario.setFkEmpresa(resultadoQuery.getInt("fkEmpresa"));
42     }
43
44 } catch (SQLException e) {
45     System.err.println("Query malsucedida");
46 }
47 return usuario;
48
49 }
```

# Validação do Login

## Explicação do fluxo de login

**Sexto passo:** voltando para a tela de login, agora temos acesso ao objeto usuario que tem todas as informações preenchidas (não só o email). Lembre-se que ele só tem todas as informações preenchidas porque o email digitado foi encontrado no banco. Agora temos que **prestar bastante atenção**. Veja que quando chamamos o método buscarDados, passamos como parâmetro um objeto Funcionario chamado funcionario. O método recebeu esse objeto e, lá dentro dele, passou a chamá-lo de usuario. No final, retornou para nós o objeto usuario. Quando esse retorno é feito, no entanto, o nome do objeto continua sendo funcionario aqui nesta função, porque foi esse o nome que passamos como parâmetro da função na linha 198. Então agora temos um objeto funcionario com todas as informações preenchidas. O próximo passo é fazer a validação em si, verificando se a senha digitada no input corresponde ao que está cadastrado no banco. Isso é feito nas linhas 200 a 206.

```
190 private void btnEntrarActionPerformed(java.awt.event.ActionEvent evt) {
191
192     if ((!txtValorEmail.getText().equalsIgnoreCase("") && !String.valueOf(txtValorSenha.getPassword()).equalsIgnoreCase("")) {
193         FuncionarioDao funcionarioDao = new FuncionarioDao();
194         Funcionario funcionario = new Funcionario();
195         LoginUsuario tentativaDeLogin = new LoginUsuario();
196
197         funcionario.setEmailFuncionario(txtValorEmail.getText());
198         funcionarioDao.buscarDados(funcionario);
199
200         try {
201             tentativaDeLogin.validarLogin(funcionario, String.valueOf(txtValorEmail.getText()), String.valueOf(txtValorSenha.getPassword()));
202             if (tentativaDeLogin.getLoginValidado()) {
203                 this.dispose();
204             }
205         } catch (Exception e) {
206             System.out.println(e);
207         }
208     } else {
209         if (txtValorEmail.getText().equals("")) {
210             JOptionPane.showMessageDialog(null, "Digite seu e-mail!");
211         } else {
212             JOptionPane.showMessageDialog(null, "Digite sua senha!");
213         }
214     }
}
```

# Validação do Login

## Explicação do fluxo de login

Aqui temos o try catch novamente. Ele vai tentar fazer o que está escrito em 201, caso dê algum erro vai para o catch na linha 205 e printa o erro no console. Em 201, é possível ver que estamos chamando o objeto tentativaDeLogin (instância da classe LoginUsuario) e estamos chamando o método validarLogin deste objeto, passando 3 elementos como parâmetro

```
200     try {
201         tentativaDeLogin.validarLogin(funcionario, String.valueOf(txtValorEmail.getText()), String.valueOf(txtValorSenha.getPassword()));
202         if (tentativaDeLogin.getLoginValidado()) {
203             this.dispose();
204         }
205     } catch (Exception e) {
206         System.out.println(e);
207     }
```

Temos que passar esses parâmetros porque o método validarLogin exige que eles sejam passados. No caso, vamos relembrar o que este método pede logo abaixo:

```
public Funcionario validarLogin(Funcionario usuario, String email, String senha)
```

O método pede um objeto do tipo Funcionario, um email String e uma senha String. Quando estamos chamando este método na linha 201, estamos passando um funcionario (que é aquele que agora tem todas as informações preenchidas), o texto que foi digitado no input de email (que está sendo convertido para String) e o texto que foi digitado no input de senha (que está sendo convertido para String também). Como todos os parâmetros estão de acordo com o que o método necessita, agora **VAMOS PARA A CLASSE LoginUsuario** para verificar o que este método irá fazer

# Validação do Login

## Explicação do fluxo de login

Na classe LoginUsuario, temos o método validarLogin que acabou de ser chamado na tela de login, passando todos os parâmetros necessários.

```
17     private Boolean loginValidado = false;
18
19     public Funcionario validarLogin(Funcionario usuario, String email, String senha) {
20
21         if (usuario.getIdFuncionario() == null) {
22             JOptionPane.showMessageDialog(null, "Email incorreto, tente novamente.");
23         } else {
24             if ((usuario.getEmailFuncionario().equals(email)) && (usuario.getSenhaFuncionario().equals(senha))) {
25                 // passar usuario, login e senha
26                 loginValidado = true;
27                 HostFuncionario telaInicio = new HostFuncionario(usuario);
28                 telaInicio.setVisible(true);
29             } else {
30                 JOptionPane.showMessageDialog(null, "Senha incorreta, tente novamente.");
31             }
32         }
33
34         return usuario;
35     }
36 }
```

Este método está recebendo um objeto usuario do tipo Funcionario (no caso passamos o funcionario que já tem todas as informações preenchidas), o email e a senha digitados. Ele vai primeiro verificar se o idFuncionario é nulo (linha 21). Por que isso? Lembra que quando ele clica em entrar a primeira coisa que fazemos é criar um objeto funcionario e atribuir APENAS o email para ele? Aí todos os outros campos (incluindo o idFuncionario) permanecem nulos e só são preenchidos caso o select retorne positivo? Então... caso o select não tenha encontrado o e-mail que o usuário entrou, isso significa que os outros campos não serão preenchidos. Logo, o idFuncionario vai ser null. Desta forma, conseguimos verificar de início se o e-mail é válido. Caso o e-mail seja válido, ele vai verificar se o email e a senha que estão no objeto funcionario (dados que foram coletados do banco e armazenados no objeto funcionario, sendo obtidos pelo getEmailFuncionario e pelo getSenhaFuncionario na linha 24) são iguais ao e-mail e à senha que o usuario digitou naqueles inputs da tela de login. Caso sejam iguais, ele muda o valor da variável loginValidado para true e abre a tela de da dashboard já (linhas 27 e 28)



# Validação do Login

## Explicação do fluxo de login

Na linha 27 estamos criando uma instância daquela tela de monitoramento. Observe que estamos passando como parâmetro o usuario (que representa o objeto do tipo Funcionario com todas as informações já preenchidas. Isso é feito para poder passar as informações dele para esta tela e, assim, poder mostrar o nome dele no início e poder fazer os inserts certinho de acordo com o id do funcionario). Na linha 28 ele só está fazendo essa instância que acabamos de criar ficar visível. Na linha 34, observe que estamos retornando o usuario (com todas as informações preenchidas).

```
17 private Boolean loginValidado = false;
18
19 public Funcionario validarLogin(Funcionario usuario, String email, String senha) {
20
21     if (usuario.getIdFuncionario() == null) {
22         JOptionPane.showMessageDialog(null, "Email incorreto, tente novamente.");
23     } else {
24         if ((usuario.getEmailFuncionario().equals(email)) && (usuario.getSenhaFuncionario().equals(senha))) {
25             // passar usuario, login e senha
26             loginValidado = true;
27             HostFuncionario telaInicio = new HostFuncionario(usuario);
28             telaInicio.setVisible(true);
29         } else {
30             JOptionPane.showMessageDialog(null, "Senha incorreta, tente novamente.");
31         }
32     }
33
34     return usuario;
35
36 }
```

Agora **VAMOS VOLTAR PARA A TELA DE LOGIN** para verificar o que acontece depois de chamar essa função validarLogin

# Validação do Login

## Explicação do fluxo de login

O que acabamos de ver acontecendo foi o que está escrito na linha 201. Como a validação deu tudo certo, lembra que mudamos o valor da variável loginValidado para true? Então agora vamos fazer um if na linha 202 perguntando se essa variável está como true. Caso ela esteja como true (ou seja, o login deu certo como vimos no slide anterior e foi aberta a janela de monitoramento), ele vai entrar no if e vai executar o comando "this.dispose()". Este comando é responsável por encerrar a tela que está sendo atualmente mostrada. Como esta função está na tela de login (App.java), quando a gente chama o "this.dispose()" o this está se referindo à tela atual. Desta forma, estamos falando "java, encerre a tela de App.java agora". Assim, garantimos que quando o login é feito com sucesso, a tela de login some e o usuário já acessa a dashboard.

```
190 private void btnEntrarActionPerformed(java.awt.event.ActionEvent evt) {
191
192     if ((!txtValorEmail.getText().equalsIgnoreCase("") && !String.valueOf(txtValorSenha.getPassword()).equalsIgnoreCase("")) {
193         FuncionarioDao funcionarioDao = new FuncionarioDao();
194         Funcionario funcionario = new Funcionario();
195         LoginUsuario tentativaDeLogin = new LoginUsuario();
196
197         funcionario.setEmailFuncionario(txtValorEmail.getText());
198         funcionarioDao.buscarDados(funcionario);
199
200         try {
201             tentativaDeLogin.validarLogin(funcionario, String.valueOf(txtValorEmail.getText()), String.valueOf(txtValorSenha.getPassword()));
202             if (tentativaDeLogin.getLoginValidado()) {
203                 this.dispose();
204             }
205         } catch (Exception e) {
206             System.out.println(e);
207         }
208     } else {
209         if (txtValorEmail.getText().equals("")) {
210             JOptionPane.showMessageDialog(null, "Digite seu e-mail!");
211         } else {
212             JOptionPane.showMessageDialog(null, "Digite sua senha!");
213         }
214     }
215 }
```