Pacotes Importados

- **javax.swing.***: Utilizado para criar e gerenciar componentes gráficos da interface.
- **java.awt.***: Fornece classes para criar elementos gráficos e manipulação de eventos.
- **java.awt.event.ActionEvent e java.awt.event.ActionListener**: Usados para capturar ações de eventos de interface.
- **java.sql.Timestamp**: Representa um ponto no tempo (data/hora) para registrar o momento do logout.
- **java.time.LocalDateTime**: Utilizado para obter a data e hora atuais.
- **controller.SessionController**: Controlador responsável pelo gerenciamento de sessões.
- **models.User**: Representa o modelo de usuário.

Atributos

- **private final User user;**
  Armazena o objeto User, contendo informações do usuário logado.
- **private final JPanel homePanel;**
  Painel principal que organiza os componentes da interface.
- **private final JLabel homeLabel;**
  Exibe o título "Home" no centro da interface.
- **private final JButton logoutButton;**
  Botão que, quando pressionado, realiza o logout do usuário.

Construtor

- **java**
- **Copiar código**
- **public Home(User user) {**
- **this.user = user;**
- **this.initComponents();**
- **}**
- **Recebe um objeto User como parâmetro e inicializa os componentes da interface chamando o método initComponents.**

Método

`initComponents()`

Responsável por configurar e organizar os componentes da interface.

1. **Configuração da Janela:**
   - Define o título, tamanho e comportamento ao fechar.
   - Posiciona a janela no centro da tela.
2. **Configuração do `JLabel`:**
   - Define a fonte, tamanho e alinhamento do título "Home".
3. **Configuração do `JButton`:**
   - Adiciona um `ActionListener` ao botão de logout, que chama o método `logoutButtonActionPerformed` quando acionado.
4. **Layout:**
   - Utiliza `GroupLayout` para organizar os componentes dentro do painel `homePanel`.
   - Configura a disposição horizontal e vertical dos componentes.

Método

`logoutButtonActionPerformed(ActionEvent e)`

Executa o processo de logout do usuário.

1. **Captura o Timestamp Atual:**
   - Registra o momento do logout usando `LocalDateTime.now()` convertido para `Timestamp`.
2. **Chama o Controlador de Sessão:**
   - Invoca o método `logout()` do `SessionController`, passando o ID do usuário e o `Timestamp`.
3. **Exibe Mensagem de Resultado:**
   - Se o logout for bem-sucedido, exibe uma mensagem de confirmação.
   - Se houver erro, exibe uma mensagem de erro.
4. **Redireciona para a Tela de Login:**
   - Abre a janela de login (`new Login().setVisible(true);`).
   - Fecha a janela atual (`this.dispose();`).

Estrutura Visual

- **Título:** "Home" centralizado no topo da tela.
- **Botão de Logout:** Posicionado centralmente abaixo do título.
- **Espaçamento:** Utiliza margens e espaçamento uniforme entre os componentes.

Observações

- **Modularidade:** A classe segue boas práticas, separando a lógica de interface da lógica de negócio (controlador de sessão).
- **Extensibilidade:** Fácil de adicionar novos componentes ou funcionalidades à interface.
- **Segurança:** O processo de logout inclui um timestamp, possibilitando auditoria ou monitoramento de sessões.

Classe `LOGIN`

A classe `Login` representa a tela de autenticação de um sistema, desenvolvida com Java Swing. Nela, os usuários inserem suas credenciais para acessar o sistema ou podem navegar para a tela de registro.

Pacotes Importados

- **`javax.swing.*`**: Utilizado para criar componentes de interface gráfica.
- **`java.awt.*`**: Contém classes para manipulação de componentes gráficos.
- **`java.awt.event.*`**: Fornece classes para tratamento de eventos de interface, como cliques de botão.

- **`java.security.NoSuchAlgorithmException`**: Exceção lançada ao lidar com algoritmos de criptografia.
- **`java.sql.Timestamp`**: Representa um ponto temporal específico, usado para registrar o horário de login.
- **`java.time.LocalDateTime`**: Obtém a data e hora atuais.
- **`controller.SessionController`** e **`controller.UserController`**: Controladores para gerenciar sessões e usuários.
- **`models.User`**: Modelo que representa as informações do usuário.
- **`utils.PasswordUtil`**: Utilitário para verificação de senhas.
- **`utils.Util`**: Contém métodos utilitários, como a geração de tokens.

Atributos

- **`loginPanel`**: Painel principal que agrupa os componentes da interface.
- **`loginLabel`**: Título "Login" exibido no topo da tela.
- **`usernameLabel`** e **`passwordLabel`**: Labels para os campos de entrada de usuário e senha.
- **`usernameField`** e **`passwordField`**: Campos de entrada para o nome do usuário e a senha, respectivamente.
- **`loginButton`** e **`registerButton`**: Botões para realizar login ou navegar até a tela de registro.

Construtor

```
java
Copiar código
public Login() {
    this.initComponents();
}
```

Inicializa a interface gráfica chamando o método `initComponents()`.

Método `initComponents()`

Responsável pela configuração e organização dos componentes da interface:

1. **Configuração da Janela:**
   - Define título, tamanho da janela e comportamento ao fechar.
   - Centraliza a janela na tela.
2. **Configuração do Layout:**
   - Usa `GroupLayout` para organizar componentes dentro do painel `loginPanel`.
   - Componentes são dispostos vertical e horizontalmente, centralizados na tela.
3. **Ações dos Botões:**
   - **Botão `loginButton`:** Associa um `ActionListener` que chama `loginButtonActionPerformed`.
   - **Botão `registerButton`:** Navega para a tela de registro ao ser clicado.

4. **Atalho de Teclado:**
    - Associa a tecla **Enter** ao botão focado (`loginButton` ou `registerButton`).

Método `loginButtonActionPerformed(ActionEvent e)`

Executa o processo de autenticação do usuário:

1. **Captura as Credenciais:**
    - Obtém o nome de usuário e senha dos campos de entrada.
2. **Validação das Credenciais:**
    - Utiliza `UserController` para buscar o usuário.
    - Verifica se a senha fornecida corresponde à armazenada usando `PasswordUtil`.
3. **Gerenciamento de Sessão:**
    - Gera um token de sessão e registra o horário atual.
    - Invoca o método `login` de `SessionController`.
4. **Resultado da Autenticação:**
    - Se o login for bem-sucedido, abre a tela `Home`.
    - Se falhar, exibe uma mensagem de erro.

Método `registerButtonActionPerformed(ActionEvent e)`

Abre a tela de registro (`Register`) e fecha a tela atual de login.

Estrutura Visual

- **Título:** "Login", centralizado no topo.
- **Campos de Entrada:** Nome de usuário e senha, com rótulos identificando cada campo.
- **Botões:**
    - **`Login`**: Realiza a autenticação.
    - **`Register`**: Abre a tela de cadastro.

Considerações Finais

- **Usabilidade:** Interface simples e clara, com atalhos úteis (Enter).
- **Segurança:** Implementa verificação segura de senhas e gerencia tokens de sessão.
- **Extensibilidade:** Fácil adicionar funcionalidades, como recuperação de senha ou log de tentativas falhas.

Classe `Register`

A classe `Register` representa a tela de cadastro de usuários de um sistema Java Swing. Ela permite que novos usuários criem uma conta fornecendo um nome de usuário, e-mail, senha e confirmação de senha.

Pacotes Importados

- **`javax.swing.*`**: Utilizado para criar componentes de interface gráfica.
- **`java.awt.*`**: Contém classes para manipulação de componentes gráficos.
- **`java.awt.event.*`**: Fornece classes para tratamento de eventos de interface, como cliques de botão.
- **`java.security.NoSuchAlgorithmException`**: Exceção lançada ao lidar com algoritmos de criptografia.
- **`controller.UserController`**: Controlador responsável por gerenciar operações de usuário.
- **`utils.PasswordUtil`**: Utilitário para manipulação segura de senhas (hashes e verificação).
- **`utils.Util`**: Fornece métodos utilitários, como validação de e-mails.

Atributos

- **`registerPanel`**: Painel principal que agrupa os componentes da interface.
- **`registerLabel`**: Título "Register" exibido no topo da tela.
- **`usernameLabel`, `emailLabel`, `passwordLabel`, `confirmPasswordLabel`**: Labels para os campos de entrada.
- **`usernameField`, `emailField`, `passwordField`, `confirmPasswordField`**: Campos de entrada para informações do usuário.
- **`registerButton` e `cancelButton`**: Botões para registrar um novo usuário ou cancelar o registro.

Construtor

```
java
Copiar código
public Register() {
    this.initComponents();
}
```

Inicializa a interface gráfica chamando o método `initComponents()`.

Método `initComponents()`

Responsável pela configuração e organização dos componentes da interface:

1. **Configuração da Janela:**
   - Define título, tamanho da janela e comportamento ao fechar.
   - Centraliza a janela na tela.
2. **Configuração do Layout:**
   - Usa `GroupLayout` para organizar os componentes dentro do painel `registerPanel`.
   - Componentes são dispostos vertical e horizontalmente, centralizados.
3. **Ações dos Botões:**
   - **Botão `registerButton`:** Associa um `ActionListener` que chama `registerButtonActionPerformed`.
   - **Botão `cancelButton`:** Retorna à tela de login ao ser clicado.
4. **Atalho de Teclado:**
   - Associa a tecla **Enter** aos botões focados (`registerButton` ou `cancelButton`).

Método `registerButtonActionPerformed(ActionEvent e)`

Executa o processo de registro do usuário:

1. **Captura as Informações:**
   - Obtém o nome de usuário, e-mail, senha e confirmação de senha dos campos de entrada.
2. **Validação dos Dados:**
   - **Campos vazios:** Verifica se todos os campos foram preenchidos.
   - **Usuário ou e-mail já existentes:** Utiliza `UserController` para checar a existência.
   - **Validação de senha:** Confere se a senha tem ao menos 8 caracteres.
   - **Validação de e-mail:** Utiliza o método `isValidEmail` da classe `Util`.
3. **Registro do Usuário:**
   - Se a senha e a confirmação corresponderem, gera o hash da senha.
   - Chama o método `register` de `UserController` para registrar o usuário.
4. **Resultado do Registro:**
   - Exibe mensagens informativas dependendo do sucesso ou falha.
   - Se bem-sucedido, abre a tela de login (`Login`) e fecha a atual.

Método `cancelButtonActionPerformed(ActionEvent e)`

Retorna à tela de login:

- Abre a janela de login (`Login`) usando `SwingUtilities.invokeLater`.
- Fecha a janela de registro atual.

Estrutura Visual

- **Título:** "Register", centralizado no topo.
- **Campos de Entrada:** Nome de usuário, e-mail, senha e confirmação de senha, com rótulos descritivos.
- **Botões:**
  - **`Register`**: Realiza o cadastro do usuário.
  - **`Cancel`**: Retorna à tela de login.

Considerações Finais

- **Usabilidade:** Interface intuitiva com mensagens de erro claras para diferentes tipos de falhas.
- **Segurança:** Garante senhas seguras com hashes e validação.
- **Extensibilidade:** Pode-se adicionar validações adicionais, como verificação de força de senha ou CAPTCHA.

Classe **`Main`**

A classe `Main` é o ponto de entrada principal da aplicação. Ela verifica a conexão com o banco de dados e inicia a interface gráfica, abrindo a tela de login.

Pacotes Importados

- **`utils.DBConnection`**: Fornece o método para obter a conexão com o banco de dados.
- **`view.Login`**: Representa a tela de login que será exibida ao iniciar a aplicação.
- **`javax.swing.SwingUtilities`**: Utilizado para garantir que a interface gráfica seja criada na thread de despacho de eventos (EDT).

Atributo **`database`**

```
java
Copiar código
private static final Connection database =
DBConnection.getConnection();
```

- **Descrição:** Estabelece a conexão com o banco de dados utilizando a classe `DBConnection`.
- **Tipo:** `Connection`, representando a conexão com o banco.
- **Uso:** Verifica se a conexão foi bem-sucedida.

Método `main(String[] args)`

Ponto de entrada da aplicação:

1. **Verificação da Conexão com o Banco de Dados:**

```
java
Copiar código
if (database != null) {
    System.out.println("Connected to the database");
} else {
    System.out.println("Failed to connect to the database");
}
```

   - **Objetivo:** Garante que a aplicação só continue se a conexão com o banco de dados estiver ativa.
   - **Saída no Console:** Exibe mensagens de sucesso ou falha na conexão.

2. **Inicialização da Interface Gráfica:**

```
java
Copiar código
SwingUtilities.invokeLater(new Runnable() {
    @Override
    public void run() {
        new Login().setVisible(true);
    }
});
```

   - **Uso do `SwingUtilities.invokeLater`:** Garante que a criação e manipulação da interface gráfica ocorram na **Event Dispatch Thread (EDT)**, conforme recomendado para o Swing.
   - **Criação da Tela de Login:** Instancia a classe `Login` e torna a janela visível.

Fluxo de Execução:

1. **Conexão com o Banco de Dados:**
   - A aplicação tenta se conectar ao banco de dados através do método `DBConnection.getConnection()`.
   - Se a conexão falhar, o usuário é informado no console, e a interface pode não funcionar corretamente.
2. **Inicialização da Interface:**
   - Após garantir a conexão, a tela de login é exibida.
   - O processo de exibição ocorre de forma assíncrona, garantindo que a interface não bloqueie a thread principal.

Estrutura do Código:

- **Organização Clara:** Separação das responsabilidades — verificação da conexão e inicialização da interface.

- **Boas Práticas:** Uso de `SwingUtilities.invokeLater` assegura que a interface gráfica seja gerenciada corretamente, evitando problemas de concorrência.
- **Simples e Direto:** Foco na funcionalidade básica de conexão e exibição da tela inicial.

Possíveis Melhorias:

- **Tratamento de Erro Detalhado:** Implementar mensagens mais específicas em caso de falha na conexão.
- **Log de Conexão:** Integrar um sistema de log para registrar tentativas e falhas de conexão.
- **Configuração Dinâmica:** Permitir que os parâmetros de conexão ao banco de dados sejam configurados externamente (ex.: arquivo `properties`).

Exemplo de Saída no Console

```
plaintext
Copiar código
Connected to the database
```

ou

```
plaintext
Copiar código
Failed to connect to the database
```

Considerações Finais:

A classe `Main` desempenha um papel crucial ao iniciar a aplicação, verificando a conexão com o banco de dados e lançando a interface gráfica de forma segura e eficiente.