Portal Bruxa Academica - P2

Nome: Luna Leão de Maria, Lucas Aguena Gatto, Victor Trindade dos Santos

Código das Classes

Classe Usuario

```
public class Usuario {

// Atributo protegido: nome de usuário (acessível pela própria
classe e subclasses).
protected String username;

// Atributo protegido: senha do usuário.
protected String senha;

// Construtor da classe que inicializa os atributos username e
senha.
public Usuario(String username, String senha) {
    this.username = username;
    this.senha = senha;
}

// Método público que retorna o nome de usuário.
public String getUsername() {
    return username;
}

// Método público que retorna a senha.
public String getSenha() {
    return senha;
}

// Método público que indica se o usuário é administrador.
// Neste caso, sempre retorna false, pois é um usuário comum.
public boolean isAdmin() {
    return false;
```

```
}


}
```

Classe Admin

```
public class Admin extends Usuario {

// Construtor da classe Admin que chama o construtor da classe pai
(Usuario).
public Admin(String username, String senha) {
    super(username, senha);  // Passa os dados para o construtor da
classe Usuario
}

// Sobrescreve o método isAdmin da classe Usuario.
// Indica que este usuário é um administrador.
@Override
public boolean isAdmin() {
    return true;
}


}
```

Classe Artigo

```
public class Artigo {

// Atributo privado que armazena o título do artigo.
private String titulo;

// Atributo privado que armazena o conteúdo do artigo.
private String conteudo;

// Construtor que inicializa título e conteúdo do artigo.
public Artigo(String titulo, String conteudo) {
    this.titulo = titulo;
    this.conteudo = conteudo;
```

```
}

// Getter para obter o título do artigo.
public String getTitulo() {
    return titulo;
}

// Getter para obter o conteúdo do artigo.
public String getConteudo() {
    return conteudo;
}


}
```

Classe BancoFake

```
public class BancoFake {

// Lista pública e estática de usuários (compartilhada por toda a
aplicação)
public static ArrayList<Usuario> usuarios = new ArrayList<>();

// Lista pública e estática de artigos
public static ArrayList<Artigo> artigos = new ArrayList<>();

// Representa o usuário atualmente logado (inicialmente null)
public static Usuario usuarioLogado = null;

// Bloco estático executado uma vez ao carregar a classe
static {
    // Adiciona um usuário administrador padrão à lista
    usuarios.add(new Admin("admin", "123"));
}


}
```

Classe PortalBruxa

```
public class PortalBruxa {

public static void main(String[] args) {
    // Executa o código da interface gráfica na thread correta (EDT
- Event Dispatch Thread)
    javax.swing.SwingUtilities.invokeLater(() -> {
        // Cria e exibe a tela de login
        new TelaLogin().setVisible(true);
    });
}


}
```

Explicação Simples dos Conceitos

Classe Usuário

Encapsulamento

- **Como está aplicado:**
  - Os atributos `username` e `senha` são `protected`, não `public`.
  - O acesso é feito por meio de **métodos públicos (`getUsername()` e `getSenha()`)**, os chamados **getters**.

Classe Admin

Herança

- **Como está aplicado:**
  - A classe `Admin` declara `extends Usuario`, indicando que herda de `Usuario`.
  - Herda atributos (`username`, `senha`) e métodos (`getUsername()`, `getSenha()`, etc.) da classe `Usuario`.
- **Explicação:**

- Herança permite reaproveitar código e criar especializações.
- `Admin` é uma especialização de `Usuario` que mantém a estrutura básica, mas pode adicionar ou modificar comportamentos.

Polimorfismo (sobrescrita de método)

- **Como está aplicado:**
  - A classe `Admin` sobrescreve o método `isAdmin()` da classe pai com `@Override`.
  - Enquanto `Usuario.isAdmin()` retorna `false`, `Admin.isAdmin()` retorna `true`.
- **Explicação:**
  - Isso possibilita tratar objetos de diferentes tipos (aqui, `Usuario` e `Admin`) de forma uniforme, mas com comportamentos específicos.

Classe Artigo

Encapsulamento

- **Como está aplicado:**
  - Os atributos `titulo` e `conteudo` são declarados como `private`, ou seja, **não acessíveis diretamente fora da classe**.
  - O acesso a esses atributos é feito por meio de métodos públicos chamados **getters** (`getTitulo()`, `getConteudo()`).
- **Explicação:**
  - O encapsulamento protege os dados internos da classe, controlando como eles são acessados e modificados.
  - Isso aumenta a segurança e a integridade dos dados, além de permitir alterar a implementação interna sem afetar o código externo

Classe BancoFake

Abstração

- **Como está aplicado:**
  - A classe `BancoFake` **representa uma abstração de um banco de dados**, mas sem acesso real a um banco externo.
  - Ela oferece apenas o necessário: listas de objetos e um usuário logado.
- **Explicação:**
  - A abstração permite **esconder detalhes técnicos de persistência** e modelar o conceito de um "repositório de dados" com simplicidade.
  - Isso facilita o uso da classe em outras partes do sistema, como se fosse uma fonte real de dados.

Classe PortalBruxa

Abstração

- **Como está aplicado:**
  - A classe `PortalBruxa` representa a **entrada principal da aplicação**, sem expor os detalhes da interface gráfica ou da lógica de negócio.
  - A criação da tela (`TelaLogin`) e a manipulação da interface ficam escondidas dentro do `invokeLater()`.
- **Explicação:**
  - A abstração está em **separar o ponto de entrada (main)** da lógica da aplicação.
  - O restante do sistema (ex.: autenticação, dados) está oculto, e só o necessário para iniciar a aplicação está exposto.

Classe Navbar

Encapsulamento

- A classe `Navbar` encapsula o comportamento de uma barra de navegação — ela tem seus atributos (`btCriar`, `btLogout`, etc.) e métodos (`btLogoutActionPerformed`, etc.) organizados dentro de si.

Classe TelaVisualizarArtigo

Encapsulamento

- A tela (`TelaVisualizarArtigo`) encapsula seus próprios componentes (`navbar1`, `pArtigos`) e comportamentos (`carregarArtigos`).
- Usa métodos privados para manter a responsabilidade interna da classe, como `carregarArtigos()`.

Composição

- Usa **composição** ao incluir objetos de outras classes: `Navbar`, `Card`, `Artigo`.