SABLONUL DE DESIGN SINGLETON

GHICA MADALINA

GR. 30239

1. PREZENTARE GENERALA

Singleton este un șablon de proiectare în ingineria software care are ca scop restricționarea instanțierii unei clase la o singură instanță și furnizarea unui punct global de acces la acea instanță. Acest șablon este utilizat atunci când este necesară o singură instanță a unei clase pentru a coordona acțiuni la nivelul sistemului.

Principalele caracteristici ale șablonului Singleton includ:

* O singură instanță: Singleton garantează că o clasă are doar o singură instanță și furnizează un mecanism pentru a accesa acea instanță.
* Acces global: Singleton oferă un punct global de acces la instanța sa, astfel încât orice parte a programului poate obține referința la această instanță unică.
* Inițializare amânată: Instantierea clasei Singleton poate fi amânată până în momentul în care este necesară, oferind economie de resurse.

2.AVANTAJE

* Controlul instanțierii: Singleton restricționează instanțierea unei clase la o singură instanță, asigurându-se că nu există mai multe obiecte ale aceleiași clase în cadrul sistemului. Aceasta este utilă în situații în care trebuie evitate conflictele de resurse sau problemele de coerență a datelor.
* Economia de resurse: Singleton permite amânarea instanțierii până în momentul în care este necesară. Astfel, resursele nu sunt utilizate în mod inutil până când instanța Singleton este cu adevărat necesară, facilitând economia de resurse.
* Inițializare amânată: Singletonul oferă posibilitatea de a amâna inițializarea instanței până în momentul în care este cerută. Aceasta este utilă în cazul în care crearea instanței poate fi costisitoare sau dacă inițializarea trebuie să fie realizată într-un anumit moment al programului.
* Evitarea conflictelor de concurență: Singletonul poate include mecanisme pentru a gestiona accesul concurent la instanța sa, evitând conflictele și asigurând o instanțiere sigură în medii cu concurență.
* Flexibilitate în modificare: Implementarea Singletonului poate fi ajustată pentru a satisface nevoile schimbătoare ale sistemului. De exemplu, o implementare poate fi modificată pentru a permite instanțierea pigră sau pentru a gestiona mai bine situațiile de concurență.

3.Secventa de cod

public class CabinetMedicalApp {

private static CabinetMedicalApp instance;

private CabinetMedicalApp() {

// Inițializarea aplicației aici

}

public static CabinetMedicalApp getInstance() {

if (instance == null) {

instance = new CabinetMedicalApp();

}

return instance;

}

public void authenticate(String username, String password, UserType userType) {

// Logica de autentificare aici

// Verificați tipul de utilizator și redirecționați către pagina corespunzătoare

switch (userType) {

case ADMIN:

redirectToAdminPage();

break;

case ASISTENT:

redirectToAsistentPage();

break;

case MEDIC:

redirectToMedicPage();

break;

case PACIENT:

redirectToPacientPage();

break;

}

}

private void redirectToAdminPage() {

// Implementați redirecționarea către pagina de administrator

}

private void redirectToAsistentPage() {

// Implementați redirecționarea către pagina de asistent

}

private void redirectToMedicPage() {

// Implementați redirecționarea către pagina de medic

}

private void redirectToPacientPage() {

// Implementați redirecționarea către pagina de pacient

}

public static void main(String[] args) {

// Exemplu de utilizare

CabinetMedicalApp app = CabinetMedicalApp.getInstance();

app.authenticate("admin", "parola", UserType.ADMIN);

}

}

enum UserType {

ADMIN,

ASISTENT,

MEDIC,

PACIENT

}