Ministerul Educației, Culturii și Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică
Departamentul Ingineria Software și Automatica

Disciplina: Tehnici si mecanisme de proiectare software

RAPORT

Proiect de curs

Tema: “Gestionarea vânzărilor de abonamente”

A efectuat: st. gr. Apostol Mihail,TI-204
A verificat: asist. univ. Mihai Gaidau

Chișinău – 2023

Cuprins

Introducere: 3

Tehnologii utilizate: 4

Clase Uml 6

Alte detalii 10

Concluzie 11

Introducere:

Proiectul de curs la obiectul Tehnici si mecanisme de proiectare software (TMPS) se axează pe dezvoltarea unei aplicații în limbajul JavaScript pentru gestionarea vânzării de abonamente la o bibliotecă pentru studenți. În cadrul acestui proiect, am implementat șapte design pattern-uri pentru a asigura o arhitectură robustă, modulară și ușor de extins.

Scopul proiectului:

De a oferi o soluție eficientă pentru gestionarea abonamentelor la bibliotecă, oferind studenților trei tipuri de abonamente: Basic, Ultimate și Premium. Fiecare abonament oferă diferite beneficii și niveluri de acces la resursele bibliotecii, adaptate nevoilor și preferințelor utilizatorilor.
Implementarea a șapte design pattern-uri ne permite să abordăm diverse aspecte ale proiectului, cum ar fi gestionarea abonamentelor, validarea și autentificarea utilizatorilor, precum și integrarea cu alte sisteme sau servicii. Aceste pattern-uri reprezintă soluții bine-cunoscute și testate în dezvoltarea software și asigură un cod ușor de întreținut și extins.

Teorie:

Creational Design Patterns:

1. Singleton: Singleton este un pattern de design creational care permite crearea unei singure instanțe a unei clase și furnizează un punct global de acces la acea instanță.
2. Prototype: Prototype este un pattern de design creational care permite crearea de noi obiecte prin clonarea unui obiect existent, în loc de crearea lor directă.
3. Factory Method: Factory Method este un pattern de design creational care oferă o interfață pentru crearea de obiecte, dar permite subclaselor să decidă tipul exact al obiectului care va fi creat.

Structural Design Patterns:

1. Facade: Facade este un pattern de design structural care oferă o interfață simplificată pentru a interacționa cu un sistem complex, ascunzând detaliile și complexitatea acestuia.
2. Decorator: Decorator este un pattern de design structural care permite atașarea de comportamente suplimentare la un obiect într-un mod dinamic, fără a afecta alte instanțe ale aceleiași clase.

Behavioral Design Patterns:

1. Iterator: Iterator este un pattern de design behavioral care oferă o modalitate de a parcurge elementele unei colecții fără a expune structura internă a colecției.
2. Memento: Memento este un pattern de design behavioral care permite salvarea și restaurarea stării interne a unui obiect fără a dezvălui detalii de implementare, oferind posibilitatea de revenire la o stare anterioară a obiectului.

Tehnologii utilizate:

Pentru proiectul de gestionare a vânzării de abonamente la o bibliotecă pentru studenți, am ales să utilizăm limbajul JavaScript datorită multiplelor avantaje pe care le oferă în dezvoltarea aplicațiilor web.

Folosirea JavaScript în dezvoltarea unei aplicații CLI are și ea avantaje:

1. Portabilitate: JavaScript poate fi rulat în mediul Node.js, care este un mediu de execuție JavaScript de tip server. Acesta oferă posibilitatea de a rula JavaScript în afara browserului, ceea ce permite dezvoltarea de aplicații JavaScript care rulează în terminal.
2. Acces la funcționalități de sistem: JavaScript în Node.js oferă acces la funcționalități de sistem de operare precum citirea și scrierea fișierelor, manipularea directoriilor, comunicarea prin rețea și multe altele. Aceasta îți permite să construiești o aplicație CLI puternică și flexibilă, care poate interacționa cu mediul de sistem.
3. Ușurința de dezvoltare și depanare: JavaScript oferă o sintaxă familiară și unelte puternice pentru dezvoltarea și depanarea aplicațiilor. Există o serie de biblioteci și framework-uri disponibile pentru Node.js, care pot accelera dezvoltarea și îmbunătăți productivitatea dezvoltatorului.
4. Integrare cu alte servicii sau aplicații: JavaScript în Node.js facilitează integrarea cu alte servicii sau aplicații prin intermediul API-urilor sau a bibliotecilor disponibile. Aceasta permite comunicarea cu baze de date, servicii de autentificare, servicii web și multe altele.

Patternurile folosite:

Implementarea celor șapte design pattern-uri aduce numeroase avantaje în cadrul proiectului nostru:

1. Decorator Pattern: Acest pattern ne permite să extindem funcționalitatea abonamentelor de bază prin adăugarea de funcționalități suplimentare. De exemplu, putem adăuga opțiuni de upgrade sau caracteristici suplimentare la abonamentele existente, oferind astfel flexibilitate și scalabilitate în gestionarea abonamentelor.
2. Factory Method Pattern: Acest pattern a fost utilizat pentru a crea diverse tipuri de abonamente în funcție de cerințele utilizatorului. Prin intermediul unei fabrici, putem gestiona crearea și inițializarea abonamentelor într-un mod flexibil și extensibil. Astfel,

putem adăuga cu ușurință noi tipuri de abonamente în viitor fără a modifica în mod direct codul existent.

3. Iterator Pattern: Acest pattern a fost folosit pentru a itera și accesa elementele colecției de abonamente. Prin intermediul unui iterator, putem parcurge și manipula colecția de abonamente într-un mod uniform și eficient, fără a dezvălui detalii de implementare.
4. Memento Pattern: Acest pattern a fost implementat pentru a permite salvarea și restaurarea stării abonamentelor. Astfel, utilizatorii pot reveni la o anumită stare anterioară a abonamentului, fără a pierde date sau progres.
5. Prototype Pattern: Acest pattern a fost folosit pentru a crea noi instanțe de abonamente pe baza unui prototip existent. Astfel, putem clona abonamente existente și să le personalizăm în funcție de nevoile utilizatorilor, fără a crea fiecare abonament de la zero.
6. Singleton Pattern: Acest pattern ne asigură că există o singură instanță a serviciului de gestionare a abonamentelor în întreaga aplicație. Aceasta asigură coerența și consistența în manipularea și accesarea abonamentelor

Clase Uml

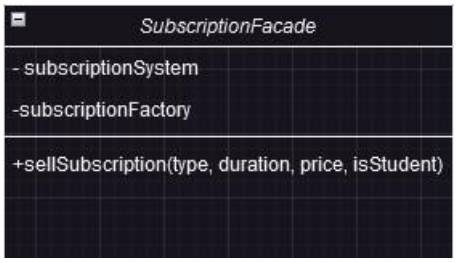

Figura 1

Facade

Explicație:

System este clasa principală care gestionează vânzarea de abonamente.

Atributele subscriptionSystem și subscriptionFactory sunt marcate ca private (-), ceea ce înseamnă că pot fi accesate doar în interiorul clasei System.

Metoda sellSubscription() este marcată ca publică (+), ceea ce înseamnă că poate fi accesată din exteriorul clasei System.

sellSubscription() primește patru parametri: type (tipul abonamentului), duration (durata în luni), price (prețul) și isStudent (o valoare booleană care indică dacă abonatul este student).

Metoda sellSubscription() creează un abonament utilizând subscriptionFactory și adaugă abonamentul în subscriptionSystem.

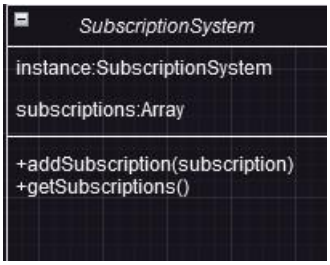

Figura 2

Singleton

Explicație:

SubscriptionSystem este clasa care gestionează abonamentele.

Atributul instance este marcat ca privat (-), ceea ce înseamnă că poate fi accesat doar în interiorul clasei SubscriptionSystem.

Atributul subscriptions este marcat ca privat (-), ceea ce înseamnă că poate fi accesat doar în interiorul clasei SubscriptionSystem.

Metoda addSubscription(subscription: Subscription) este marcată ca publică (+) și permite adăugarea unui obiect subscription de tip Subscription în lista subscriptions.

Metoda getSubscriptions(): Subscription[] este marcată ca publică (+) și returnează lista subscriptions de tip array care conține obiecte de tip Subscription.

Această implementare presupune că există o instanță Singleton a clasei SubscriptionSystem, care este accesibilă prin intermediul metodei statice getInstance().

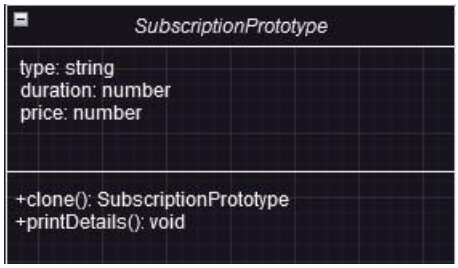

Figura 3

Prototype

Explicație:

SubscriptionPrototype este clasa care reprezintă un prototip de abonament.

Atributele type, duration și price sunt marcate ca private (-), ceea ce înseamnă că pot fi accesate doar în interiorul clasei SubscriptionPrototype.

Metoda clone() este marcată ca publică (+) și returnează o nouă instanță a SubscriptionPrototype, reprezentând o clonă a obiectului curent.

Metoda printDetails() este marcată ca publică (+) și este responsabilă de afișarea detaliilor abonamentului.

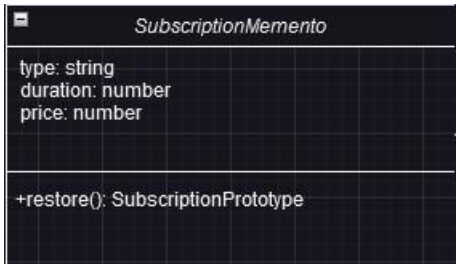

Figura 4

Memento

Explicație:

SubscriptionMemento este clasa care reprezintă un memento (o stare salvată) a unui abonament.

Atributele type, duration și price sunt marcate ca private (-), ceea ce înseamnă că pot fi accesate doar în interiorul clasei SubscriptionMemento.

Metoda restore() este marcată ca publică (+) și returnează o instanță de SubscriptionPrototype, utilizând valorile stocate în memento pentru a crea un nou abonament.

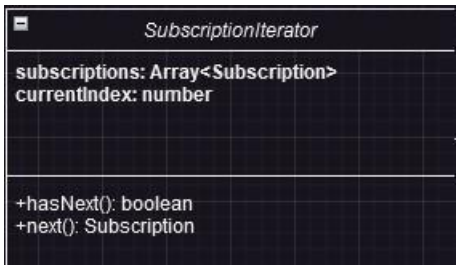

Figura 5

Iterator

Explicație:

SubscriptionIterator este clasa care implementează un iterator pentru parcurgerea unei colecții de abonamente.

Atributele subscriptions și currentIndex sunt marcate ca private (-), ceea ce înseamnă că pot fi accesate doar în interiorul clasei SubscriptionIterator.

Metoda hasNext() este marcată ca publică (+) și returnează true dacă mai există un abonament disponibil în iterație, sau false în caz contrar.

Metoda next() este marcată ca publică (+) și returnează următorul abonament din iterație.

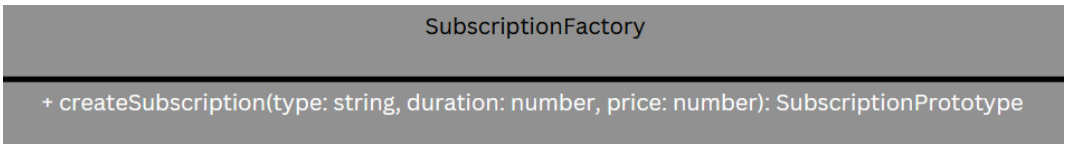

Figura 6

Factory Method

Explicație:

SubscriptionFactory este clasa care implementează un Factory Method pentru crearea de obiecte de tip SubscriptionPrototype.

Metoda createSubscription(type: string, duration: number, price: number): SubscriptionPrototype este marcată ca publică (+) și primește trei parametri: type (de tip string), duration (de tip number) și price (de tip number). Această metodă creează și returnează un obiect de tip SubscriptionPrototype pe baza parametrilor furnizați.

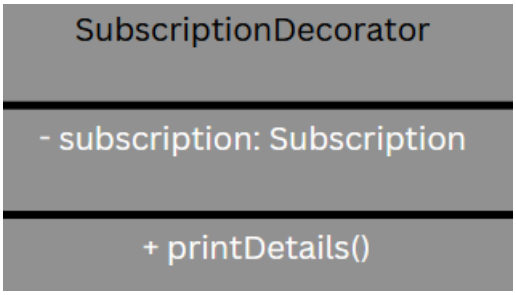

Figura 7

Decorator

Explicație:

SubscriptionDecorator este clasa decorator care extinde funcționalitatea abonamentului de bază.

Atributul subscription este marcat ca privat (-), ceea ce înseamnă că poate fi accesat doar în interiorul clasei SubscriptionDecorator.

Metoda printDetails() este marcată ca publică (+) și apelează metoda printDetails() a obiectului subscription pentru a afișa detaliile abonamentului decorat.

Alte detalii

Clasa SubscriptionSystem implementează Singleton pattern-ul.Constructorul verifică dacă există deja o instanță a clasei SubscriptionSystem. Dacă există,returnează instanța existentă.

Clasa Subscription reprezintă un abonament și este folosită în Prototype pattern.
Metoda printDetails() afișează detaliile abonamentului în consolă.
Clasa SubscriptionPrototype nu este menționată explicit în cod, dar se presupune că este disponibilă pentru crearea prototipurilor.
Clasa SubscriptionFactory reprezintă o fabrică care creează obiecte Subscription.
Are o metodă createSubscription() care primește type, duration și price

Decorator Pattern:

Clasa SubscriptionDecorator reprezintă un decorator pentru obiectele Subscription.
Constructorul primește un obiect subscription și îl stochează intern.
Metoda printDetails() apelează metoda printDetails() a obiectului subscription intern.

Iterator Pattern:

Clasa SubscriptionIterator reprezintă un iterator pentru colecția de abonamente
Constructorul primește un array de abonamente și inițializează currentIndex cu 0.
Metoda hasNext() verifică dacă mai există elemente în iterator.
Metoda next() returnează următorul abonament din secvența de iterare.

Memento Pattern:

Clasa SubscriptionMemento reprezintă un memento care stochează starea abonamentelor.

Concluzie

Proiectul prezent implică implementarea mai multor pattern-uri de design într-un sistem de gestionare a abonamentelor. Iată o concluzie generală despre acest proiect:Proiectul utilizează mai multe pattern-uri de design, inclusiv Singleton,Façade, Factory Method, Prototype, Decorator și Iterator, pentru a crea și gestiona abonamentele într-un sistem.Clasa System acționează ca o interfață simplificată (facadă) către subsistemele de abonamente.Clasa SubscriptionSystem este implementată ca un Singleton, asigurând că există o singură instanță a sistemului de abonamente în cadrul aplicației.Clasa SubscriptionFactory oferă un Factory Method pentru crearea de obiecte de tip SubscriptionPrototype.Clasa SubscriptionPrototype implementează Pattern-ul Prototype, permițând clonarea abonamentelor existente.Clasa SubscriptionDecorator implementează Pattern-ul Decorator, extinzând funcționalitatea abonamentelor de bază prin decorare.Clasa SubscriptionIterator implementează Pattern-ul Iterator, permițând parcurgerea colecției de abonamente.Clasa SubscriptionMemento implementează Pattern-ul Memento, salvând starea abonamentelor pentru restaurarea ulterioară.Prin aplicarea acestor pattern-uri de design, sistemul de gestionare a abonamentelor devine mai flexibil, modular și ușor de extins în viitor.În ansamblu, acest proiect demonstrează utilizarea eficientă a diferitelor pattern-uri de design.