Ministerul Educației și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică şi Microelectronică

Departamentul Ingineria Software și Automatică

RAPORT

Lucrare de laborator nr.3

Disciplina: Tehnici și Mecanisme de Proiectare Software

Tema: Șabloane structurale

A efectuat: st. gr. TI-202, Aliona Tatar

A verificat: asist. univ., Mihai Gaidău

Chişinău 2023

**Elaborarea sarcinii**

**Patternurile structurale** se concentrează pe relația dintre obiecte, în special pe modul în care acestea sunt legate între ele pentru a forma structuri mai mari și mai complexe. Acestea sunt utile atunci când este necesar să se creeze relații între obiecte pentru a rezolva probleme specifice, cum ar fi gestionarea dependențelor și interfețelor între obiecte.

Unele dintre cele mai cunoscute pattern-uri structurale includ:

**Adapter** - convertește interfețele unui obiect în interfețe compatibile cu cele ale altor obiecte, permițând astfel obiectelor să comunice și să lucreze împreună.

În codul prezentat, se observă implementarea pattern-ului structural Adapter prin intermediul interfeței ILibraryAdapter și a clasei LibraryAdapter. Această clasă servește ca o interfață între clasa Library și alte clase care doresc să adauge sau să elimine elemente din bibliotecă prin intermediul metodelor Add și Remove. Astfel, alte clase pot să utilizeze această interfață în loc să utilizeze direct clasa Library și să modifice astfel funcționarea bibliotecii fără să modifice codul existent.

public interface ILibraryAdapter

{

void Add(ILibraryItem item);

void Remove(ILibraryItem item);

}

public class LibraryAdapter : ILibraryAdapter

{

private readonly Library \_library;

public LibraryAdapter(Library library)

{

\_library = library;

}

public void Add(ILibraryItem item)

{

\_library.AddItem(item);

}

public void Remove(ILibraryItem item)

{

\_library.RemoveItem(item);

}

}

Astfel, prin intermediul acestei clase adapter, se poate adapta clasa existentă Library la interfața ILibraryAdapter, pentru a putea fi utilizată într-un mod mai ușor și mai flexibil în cadrul altor clase care folosesc interfața ILibraryAdapter.

**Decorator** - permite adăugarea de funcționalități suplimentare la obiecte existente prin încapsularea acestora în obiecte decorator.

În exemplul dat, clasa BorrowableDecorator este decoratorul pentru obiectele de tip ILibraryItem, care oferă posibilitatea de a împrumuta și returna aceste obiecte.

BorrowableDecorator implementează aceeași interfață ca și clasa de bază ILibraryItem și stochează un obiect Item de tip ILibraryItem ca o proprietate. Clasa BorrowableDecorator extinde comportamentul inițial al obiectului Item, adăugând abilitatea de a fi împrumutat și returnat.

public class BorrowableDecorator : ILibraryItemDecorator

{

public ILibraryItem Item { get; }

public string Borrower { get; set; }

public DateTime BorrowDate { get; set; }

public int NumberOfCopies { get => Item.NumberOfCopies; set => Item.NumberOfCopies = value; }

public string Title => Item.Title;

public BorrowableDecorator(ILibraryItem item)

{

Item = item;

}

public void BorrowItem(string borrower)

{

Borrower = borrower;

BorrowDate = DateTime.Now;

NumberOfCopies--;

}

public void ReturnItem()

{

Borrower = null;

BorrowDate = default;

NumberOfCopies++;

}

public void Display()

{

Item.Display();

if (!string.IsNullOrEmpty(Borrower))

{

Console.WriteLine($"Borrower: {Borrower}, Borrow Date: {BorrowDate.ToShortDateString()}");

}

}

}

public interface ILibraryItemDecorator : ILibraryItem

{

ILibraryItem Item { get; }

}

De exemplu, atunci când o carte este împrumutată, BorrowableDecorator scade numărul de exemplare disponibile pentru cartea respectivă și stochează informațiile despre împrumutător și data împrumutului. Când cartea este returnată, proprietățile Borrower și BorrowDate sunt resetate, iar numărul de exemplare disponibile pentru carte este crescut.

De asemenea, BorrowableDecorator suprascrie metoda Display() din clasa de bază, afișând informațiile despre obiectul ILibraryItem și, dacă obiectul a fost împrumutat, afișează informațiile despre împrumutător și data împrumutului.

public interface ILibraryItem

{

string Title { get; }

int NumberOfCopies { get; set; }

void Display();

}

În cadrul clasei Library, decoratorul BorrowableDecorator poate fi adăugat la orice obiect de tip ILibraryItem, ceea ce permite împrumutul și returnarea obiectelor din bibliotecă. Acest lucru face ca clasa Library să fie mai flexibilă și mai ușor de întreținut, deoarece nu este nevoie să se modifice clasa de bază ILibraryItem atunci când se dorește adăugarea de noi responsabilități.

**Façade** - furnizează o interfață simplificată către un subsistem mai mare, permițând astfel utilizatorilor să acceseze aceste subsisteme fără a cunoaște detaliile interne.

Principiul facadei se aplică prin intermediul clasei LibraryFacade. Această clasă oferă o interfață simplificată și unificată pentru utilizatorii bibliotecii pentru a accesa funcționalitățile oferite de clasa Library și clasele decorator asociate.

Prin intermediul metodelor precum AddBook, RemoveBook, BorrowBook și ReturnBook, utilizatorii bibliotecii pot interacționa cu biblioteca prin intermediul obiectelor de tip LibraryFacade în loc să interacționeze direct cu obiectele din interiorul bibliotecii.

public class LibraryFacade

{

private readonly Library \_library;

private readonly ILibraryAdapter \_adapter;

public LibraryFacade()

{

\_library = new Library();

\_adapter = new LibraryAdapter(\_library);

}

public void AddBook(string title, int numberOfCopies)

{

var book = new Book(title, numberOfCopies);

\_adapter.Add(book);

}

public void RemoveBook(string title)

{

var book = \_library.GetItem(title);

\_adapter.Remove(book);

}

public void BorrowBook(string title, string borrower)

{

var book = \_library.GetItem(title);

var borrowableBook = new BorrowableDecorator(book);

borrowableBook.BorrowItem(borrower);

\_adapter.Remove(book);

\_adapter.Add(borrowableBook);

}

public void ReturnBook(string title)

{

var book = \_library.GetItem(title);

var borrowableBook = book as BorrowableDecorator;

borrowableBook.ReturnItem();

\_adapter.Remove(book);

\_adapter.Add(borrowableBook.Item);

}

public Library GetLibrary()

{

return \_library;

}

public void DisplayItems()

{

Console.WriteLine("Library Items:");

}

}

Astfel, utilizatorii nu trebuie să aibă cunoștințe despre cum funcționează biblioteca sau despre clasele decorator asociate pentru a efectua operațiile simple, cum ar fi adăugarea sau ștergerea de cărți sau împrumutul și returnarea lor.

Prin intermediul clasei LibraryFacade, utilizatorii pot utiliza o interfață simplificată pentru a accesa funcționalitățile bibliotecii, fără a fi nevoiți să interacționeze direct cu clasa Library și clasele decorator asociate.

**Proxy** - un pattern structural care permite crearea unui obiect intermediar care acționează ca o înlocuire pentru alt obiect. Scopul său este de a controla accesul la obiectul original, fără a-l modifica sau înlocui, oferind o interfață similară cu obiectul original.

Clasa CachingProxyBookRepository implementează pattern-ul Proxy. Este un proxy de memorare cache care stochează în cache cărți pentru acces rapid. Dacă o carte este cerută și se află în cache, aceasta este returnată direct. În caz contrar, proxy-ul va accesa depozitul de cărți de bază pentru a obține cartea și o va adăuga în cache pentru accesul ulterior.

public class CachingProxyBookRepository : IBookRepository

{

private readonly IBookRepository \_bookRepository;

private readonly Dictionary<string, Book> \_cache;

public CachingProxyBookRepository(IBookRepository bookRepository)

{

\_bookRepository = bookRepository;

\_cache = new Dictionary<string, Book>();

}

public void AddBook(Book book)

{

\_bookRepository.AddBook(book);

\_cache[book.Title] = book;

}

public void RemoveBook(Book book)

{

\_bookRepository.RemoveBook(book);

\_cache.Remove(book.Title);

}

public Book GetBookByTitle(string title)

{

if (\_cache.ContainsKey(title))

{

return \_cache[title];

}

var book = \_bookRepository.GetBookByTitle(title);

if (book != null)

{

\_cache[title] = book;

}

return book;

}

public IEnumerable<Book> GetAllBooks()

{

// Notă: Caching proxy nu stochează întreaga colecție de cărți,

// ci extrage cărțile din depozitul de bază de fiecare dată.

return \_bookRepository.GetAllBooks();

}

Metodele implementate sunt:

* AddBook: Adaugă o carte atât în depozitul de cărți de bază, cât și în cache.
* RemoveBook: Înlătură o carte atât din depozitul de cărți de bază, cât și din cache.
* GetBookByTitle: Returnează o carte după titlu. Verifică mai întâi în cache și, dacă cartea este găsită, o returnează. În caz contrar, accesează depozitul de cărți de bază pentru a obține cartea și o adaugă în cache înainte de a o returna.
* GetAllBooks: Returnează toate cărțile din depozitul de cărți de bază. Notă: Proxy-ul de memorare cache nu stochează întreaga colecție de cărți, ci extrage cărțile din depozitul de bază de fiecare dată.

Acest proxy permite o accesare mai rapidă a cărților deja existente în cache și minimizează accesul la depozitul de cărți de bază, îmbunătățind performanța sistemului în cazul în care aceleași cărți sunt solicitate frecvent.

**Concluzie**

În cea de-a treia lucrare de laborator am studiat șabloanele de proiectare structurale care sunt utile la organizarea și gestionarea eficientă a diferitelor entități și funcționalități ale sistemului, având drept scop reutilizarea codului și reducerea dependențele strânse între componente, facilitând astfel adaptabilitatea și extensibilitatea.

Patternul Adapter a facilitat interoperabilitatea între obiectele existente în bibliotecă, permițând să fie adăugate și să fie eliminate elemente într-un mod uniform prin intermediul unui adaptor. Acesta a oferit o interfață comună pentru obiecte incompatibile.

Patternul Decorator a permis adăugarea de comportamente suplimentare sau informații la obiectele bibliotecii prin intermediul decorațiunilor. Acesta a oferit flexibilitate și extensibilitate în gestionarea împrumuturilor și afișarea informațiilor despre cărți.

Patternul Facade a furnizat o interfață simplificată și unificată pentru utilizarea bibliotecii. Acesta a ascuns complexitatea sistemului sub o interfață simplă, permițând utilizatorilor să interacționeze ușor cu biblioteca fără a cunoaște detaliile interne.

În ansamblu, prin utilizarea acestor patternuri structurale, am reușit să creez un sistem bine organizat și ușor de utilizat pentru gestionarea bibliotecii. Aceste patternuri au adus beneficii semnificative în ceea ce privește modularitatea, reutilizarea și extensibilitatea codului.