Tema 1

Sctructura fisierelor

```
├── ConsoleProject.csproj
├── Models
│   ├── Book.cs
│   ├── LibraryItem.cs
│   ├── LibraryUser.cs
│   └── Member.cs
├── Program.cs
├── Services
│   ├── BookRepository.cs
│   ├── IBookRepository.cs
│   ├── ILoanService.cs
│   ├── IMemberRepository.cs
│   ├── LoanService.cs
│   └── MemberRepository.cs
├── System
│   ├── ILibraryLoanSystem.cs
│   ├── ILibraryStorage.cs
│   └── LibrarySystem.cs
└──────────────────────────
```

Am șters fișierul `ILibrarySystem.cs` și am separat această interfață în două interfețe distincte:

- `ILibraryStorage` – responsabilă pentru gestionarea stocării cărților
- `ILibraryLoanSystem` – responsabilă pentru gestionarea împrumuturilor de cărți

Această modificare respectă **Interface Segregation Principle (ISP)**, asigurând că fiecare interfață conține doar metode relevante pentru o responsabilitate specifică.

Codul actualizat al interfețelor:

`ILibraryStorage`

```
public interface ILibraryStorage
{
    void AddBook(LibraryItem book);
    void RemoveBook(LibraryItem book);
}
```

`ILibraryLoanSystem`

```
public interface ILibraryLoanSystem
{
    void BorrowBook(LibraryUser user, LibraryItem item);
    void ReturnBook(LibraryUser user, LibraryItem item);
    bool IsBookAvailable(LibraryItem item);
}
```

Astfel, am eliminat dependențele inutile și am împărțit clar funcționalitățile, respectând principiul **ISP**.

Ai aplicat corect **Liskov Substitution Principle (LSP)** prin introducerea claselor abstracte `LibraryItem` și `LibraryUser`, ceea ce permite tratarea obiectelor derivate într-un mod uniform.

Modificările aduse:

✅ **Generalizare**:

- `LibraryItem` este o clasă abstractă pentru toate tipurile de obiecte din bibliotecă (ex. cărți, reviste, CD-uri etc.).
- `LibraryUser` reprezintă un utilizator generic al bibliotecii, permițând extinderea cu tipuri diferite de utilizatori.

✅ **Respectarea LSP:**

- În loc de a folosi un `string` pentru membri, metoda `AddMember` primește acum un obiect de tip `LibraryUser`, ceea ce permite extinderea cu clase derivate, cum ar fi `Member`.
- **Orice instanță a unei clase derivate trebuie să poată fi utilizată în locul clasei de bază fără a afecta corectitudinea programului**, iar această schimbare respectă acest principiu.

LibraryItem.cs

```
public abstract class LibraryItem
{
    public string Title { get; set; }
    public string Author {get; set; }
    public string ISBN { get; set; }

    public LibraryItem(string Title, string ISBN, string author){
        this.Title = Title;
        this.ISBN = ISBN;
        this.Author = author;
    }

}
```

LibraryUser.cs

```
public abstract class LibraryUser
{
    public string Name { get; set; }
    public string Email { get; set; }
    public string PhoneNumber { get; set; }
    public string Address { get; set; }

    public LibraryUser(
                string name,
                string email,
                string phoneNumber,
                string address)
    {
        Name = name;
        Email = email;
        PhoneNumber = phoneNumber;
        Address = address;
    }
}
```

Din care deriva doua clase `Member` şi `Book`

A fost

```
public void AddMember(string m
ember)
{
      members.Add(member);
}
```

A devenit

```
public void AddMember(LibraryU
ser member)
{
   memberRepository.Add(mem
ber);
}
```

Tasks

Am aplicat principiul **Single Responsibility Principle (SRP)** prin introducerea a trei interfețe:

- `IBookRepository` – responsabilă pentru gestionarea cărților
- `ILoanService` – responsabilă pentru gestionarea împrumuturilor de cărți
- `IMemberRepository` – responsabilă pentru gestionarea utilizatorilor

De asemenea, am creat următoarele clase care implementează aceste interfețe:

- `BookRepository` – implementează `IBookRepository`
- `LoanService` – implementează `ILoanService`
- `MemberRepository` – implementează `IMemberRepository`

Astfel, fiecare clasă are o singură responsabilitate, respectând principiul SRP.

```
public class LibrarySystem : ILibraryStorage, ILibraryLoanSystem
{
   private readonly IBookRepository bookRepository;
   private readonly IMemberRepository memberRepository;
   private readonly ILoanService loanService;
}
```

Am rezolvat **Dependency Inversion Principle (DIP)** adăugând peste tot clase abstracte (e.g. `LibraryItem.cs`, `LibraryUser.cs`) și interfețe (e.g. `IBookRepository`, `ILoanService`, `IMemberRepository`) astfel încât nu voi avea dependență directă de clase.

De asemenea, am implementat **Dependency Injection** prin constructor pentru a separa complet abstracțiile de implementări, reducând astfel cuplarea și făcând sistemul mai flexibil și testabil.

Înainte, metoda `AddBook` primea un **string**, ceea ce forța modificarea codului ori de câte ori se adăuga un nou tip de carte:

```
public void AddBook(string book)
{
   books.Add(book);
}
```

Această metodă **nu permite extinderea sistemului fără modificare**, deoarece, dacă în viitor s-ar adăuga alte tipuri de materiale (ex: reviste, CD-uri), codul ar trebui schimbat.

În loc de `string`, am creat o clasă abstractă `LibraryItem`, ceea ce permite ca orice tip de item din bibliotecă să fie reprezentat ca o subclasă:

```
public abstract class LibraryItem
{
   public string Title { get; set; }
   public string Author { get; set; }
   public string ISBN { get; set; }

   public LibraryItem(string Title, string ISBN, string author)
   {
      this.Title = Title;
      this.ISBN = ISBN;
      this.Author = author;
   }
}
```

Astfel, în viitor, dacă dorim să adăugăm **reviste, CD-uri sau alte tipuri de obiecte**, putem face acest lucru **fără a modifica metodele existente**, ci doar adăugând noi clase care extind `LibraryItem`.

Am modificat metoda `AddBook` pentru a accepta `LibraryItem`

Acum, `AddBook` primește un obiect de tip `LibraryItem`, permițând adăugarea oricărui tip de material bibliotecar **fără modificări ulterioare ale metodei**:

```
public void AddBook(LibraryItem book)
{
   bookRepository.Add(book);
}
```

Aceasta înseamnă că în viitor, dacă dorim să adăugăm **revista, CD-uri sau alte resurse**, trebuie doar să creăm clase noi care extind `LibraryItem`, fără să atingem metoda `AddBook`.