Le design pattern Iterator fournit une façon d'accéder aux éléments d'un objet agrégat séquentiellement sans exposer sa représentation sous-jacente. Voici un exercice pratique autour de ce motif.

**Exercice :**

Vous devez développer un système pour une bibliothèque. Cette bibliothèque contient plusieurs étagères, et chaque étagère contient plusieurs livres. Vous devez permettre aux utilisateurs de parcourir tous les livres de toutes les étagères de manière séquentielle.

1. Créez un diagramme UML pour ce système en utilisant le design pattern Iterator.
2. Implémentez le système en C#.

**Diagramme UML (Simplifié)**:

* **Interface** IIterator
  + méthode : Next() -> Object
  + méthode : HasNext() -> bool
* **Interface** IAggregate
  + méthode : CreateIterator() -> IIterator
* Classe Livre
  + propriété : titre (String)
* Classe Etagere (implémente IAggregate)
  + propriété : livres (List<Livre>)
  + méthode : CreateIterator() -> IIterator
* Classe EtagereIterator (implémente IIterator)
  + propriété : etagere (Etagere)
  + propriété : index (int)
  + méthode : Next() -> Object
  + méthode : HasNext() -> bool

**Implémentation en C#**:

csharp

using System.Collections.Generic;

public interface IIterator

{

object Next();

bool HasNext();

}

public interface IAggregate

{

IIterator CreateIterator();

}

public class Livre

{

public string Titre { get; }

public Livre(string titre)

{

Titre = titre;

}

}

public class Etagere : IAggregate

{

private List<Livre> \_livres;

public Etagere()

{

\_livres = new List<Livre>();

}

public void AjouterLivre(Livre livre)

{

\_livres.Add(livre);

}

public IIterator CreateIterator()

{

return new EtagereIterator(this);

}

public int Count => \_livres.Count;

public Livre this[int index]

{

get => \_livres[index];

}

}

public class EtagereIterator : IIterator

{

private Etagere \_etagere;

private int \_index;

public EtagereIterator(Etagere etagere)

{

\_etagere = etagere;

\_index = 0;

}

public object Next()

{

Livre livre = \_etagere[\_index];

\_index++;

return livre;

}

public bool HasNext()

{

return \_index < \_etagere.Count;

}

}

// Test

var etagere = new Etagere();

etagere.AjouterLivre(new Livre("Livre A"));

etagere.AjouterLivre(new Livre("Livre B"));

etagere.AjouterLivre(new Livre("Livre C"));

IIterator iterator = etagere.CreateIterator();

while (iterator.HasNext())

{

Livre livre = (Livre)iterator.Next();

System.Console.WriteLine(livre.Titre);

}

En utilisant le design pattern Iterator, nous avons pu encapsuler la responsabilité de l'itération au sein de la classe EtagereIterator, permettant ainsi à la classe Etagere de se concentrer uniquement sur la gestion des livres.