**Exercice:**

Créez un système qui représente une hiérarchie de groupes d'employés dans une entreprise. Un employé peut être soit un travailleur individuel, soit un gestionnaire qui peut avoir plusieurs employés sous sa responsabilité. Le système doit être capable de calculer les salaires de tous les employés dans la hiérarchie. Nous allons utiliser le pattern Composite pour représenter cette hiérarchie.

Ensuite, nous voulons également protéger l'accès à certaines informations sur les employés. Par exemple, seulement les gestionnaires peuvent voir les informations sur les salaires de leurs subordonnés. Nous utiliserons le pattern Proxy pour cela.

Finalement, nous voulons pouvoir effectuer différentes opérations sur la hiérarchie d'employés, comme calculer le total des salaires ou afficher les détails de tous les employés. Pour cela, nous utiliserons le pattern Visiteur.

Je vais vous donner un aperçu de comment on pourrait commencer à implémenter cela en C#.

csharp

// L'interface Employee pour le pattern Composite.

public interface IEmployee

{

void Accept(IVisitor visitor);

}

// L'interface Visiteur pour le pattern Visiteur.

public interface IVisitor

{

void Visit(Employee employee);

void Visit(Manager manager);

}

// La classe Employee.

public class Employee : IEmployee

{

public string Name { get; set; }

public decimal Salary { get; set; }

public Employee(string name, decimal salary)

{

this.Name = name;

this.Salary = salary;

}

public void Accept(IVisitor visitor)

{

visitor.Visit(this);

}

}

// La classe Manager pour le pattern Composite.

public class Manager : IEmployee

{

public string Name { get; set; }

public decimal Salary { get; set; }

private List<IEmployee> \_employees = new List<IEmployee>();

public Manager(string name, decimal salary)

{

this.Name = name;

this.Salary = salary;

}

public void Add(IEmployee employee)

{

\_employees.Add(employee);

}

public void Accept(IVisitor visitor)

{

visitor.Visit(this);

foreach (var employee in \_employees)

{

employee.Accept(visitor);

}

}

}

// La classe Proxy pour le pattern Proxy.

public class EmployeeProxy : IEmployee

{

private IEmployee \_realEmployee;

public EmployeeProxy(IEmployee realEmployee)

{

\_realEmployee = realEmployee;

}

public void Accept(IVisitor visitor)

{

if (visitor is SalaryVisitor)

{

Console.WriteLine("Unauthorized access!");

}

else

{

\_realEmployee.Accept(visitor);

}

}

}

// Un visiteur qui calcule le total des salaires.

public class SalaryVisitor : IVisitor

{

private decimal \_totalSalary = 0;

public void Visit(Employee employee)

{

\_totalSalary += employee.Salary;

}

public void Visit(Manager manager)

{

\_totalSalary += manager.Salary;

}

public decimal GetTotalSalary()

{

return \_totalSalary;

}

}

// Un visiteur qui affiche les détails des employés.

public class DetailsVisitor : IVisitor

{

public void Visit(Employee employee)

{

Console.WriteLine($"Employee: {employee.Name}, Salary: {employee.Salary}");

}

public void Visit(Manager manager)

{

Console.WriteLine($"Manager: {manager.Name}, Salary: {manager.Salary}");

}

}

N'oubliez pas qu'il est important de comprendre chaque design pattern et de savoir comment ils fonctionnent avant de les implémenter dans un projet.