Pour créer un modèle UML complexe basé sur le concept de dossiers et de fichiers en utilisant le patron de conception Command, nous allons suivre ces étapes :

1. **Identification des Classes**:
   * Commande : interface pour toutes les commandes
   * CréerDossierCommande : crée un dossier
   * CréerFichierCommande : crée un fichier
   * Invocateur : stocke et exécute les commandes
   * Dossier : représente un dossier qui peut contenir d'autres dossiers ou fichiers
   * Fichier : représente un fichier
2. **Modèle UML** :
   * **Commande** : Une interface avec une méthode Exécuter().
   * **CréerDossierCommande** :
     + **Dépendances** : Commande, Dossier
     + **Méthodes** : Exécuter()
   * **CréerFichierCommande** :
     + **Dépendances** : Commande, Fichier
     + **Méthodes** : Exécuter()
   * **Invocateur** :
     + **Attributs** : Liste de commandes
     + **Méthodes** : StockerCommande(), ExécuterCommandes()
   * **Dossier** :
     + **Attributs** : Nom, Liste de dossiers, Liste de fichiers
     + **Méthodes** : AjouterDossier(), AjouterFichier()
   * **Fichier** :
     + **Attributs** : Nom, Contenu
     + **Méthodes** : Écrire()
3. **Implémentation en C#** :

csharp

// Commande Interface

public interface ICommande {

void Exécuter();

}

// Commandes concrètes

public class CréerDossierCommande : ICommande {

private Dossier \_dossier;

private string \_nom;

public CréerDossierCommande(Dossier dossier, string nom) {

this.\_dossier = dossier;

this.\_nom = nom;

}

public void Exécuter() {

\_dossier.AjouterDossier(new Dossier(\_nom));

}

}

public class CréerFichierCommande : ICommande {

private Dossier \_dossier;

private string \_nom;

private string \_contenu;

public CréerFichierCommande(Dossier dossier, string nom, string contenu) {

this.\_dossier = dossier;

this.\_nom = nom;

this.\_contenu = contenu;

}

public void Exécuter() {

\_dossier.AjouterFichier(new Fichier(\_nom, \_contenu));

}

}

// Invocateur

public class Invocateur {

private List<ICommande> \_commandes = new List<ICommande>();

public void StockerCommande(ICommande cmd) {

\_commandes.Add(cmd);

}

public void ExécuterCommandes() {

foreach(var cmd in \_commandes) {

cmd.Exécuter();

}

\_commandes.Clear();

}

}

// Récepteurs

public class Dossier {

public string Nom { get; }

private List<Dossier> \_dossiers = new List<Dossier>();

private List<Fichier> \_fichiers = new List<Fichier>();

public Dossier(string nom) {

Nom = nom;

}

public void AjouterDossier(Dossier dossier) {

\_dossiers.Add(dossier);

}

public void AjouterFichier(Fichier fichier) {

\_fichiers.Add(fichier);

}

}

public class Fichier {

public string Nom { get; }

public string Contenu { get; }

public Fichier(string nom, string contenu) {

Nom = nom;

Contenu = contenu;

}

}

Avec cette approche, vous pouvez facilement ajouter, modifier ou supprimer des commandes sans avoir à modifier le code existant. C'est l'un des avantages majeurs du patron de conception Command.