D'accord, je vais vous donner un exercice UML basé sur un système de commandes pour un restaurant, en utilisant le patron de conception "Command". Ensuite, je fournirai un exemple d'implémentation en C#.

**Exercice UML :**

**Objectif** : Modéliser et implémenter un système de prise de commande pour un restaurant avec la possibilité de revenir en arrière (annuler une commande).

**Entités** :

1. Serveur
2. Commande
3. CommandeConcrète (par exemple, CommanderPizza, CommanderBoisson)
4. CommandeInvoker
5. Cuisine (Receiver)

**Diagramme UML de classes** :

1. Commande (Interface)
   * + executer() : void
     + annuler() : void
2. CommandeConcrète (Classes qui implémentent Commande)
   * + executer() : void
     + annuler() : void
3. CommandeInvoker
   * + prendreCommande(Commande c) : void
     + executerCommande() : void
     + annulerCommande() : void
4. Cuisine (Receiver)
   * + preparerPizza() : void
     + preparerBoisson() : void
5. Serveur
   * + créerCommande() : void

**Implémentation en C#** :

csharp

// Commande Interface

public interface ICommande

{

void Executer();

void Annuler();

}

// Receiver

public class Cuisine

{

public void PreparerPizza()

{

Console.WriteLine("Pizza est prête!");

}

public void PreparerBoisson()

{

Console.WriteLine("Boisson est prête!");

}

}

// Commande Concrète

public class CommanderPizza : ICommande

{

private readonly Cuisine \_cuisine;

public CommanderPizza(Cuisine cuisine)

{

\_cuisine = cuisine;

}

public void Executer()

{

\_cuisine.PreparerPizza();

}

public void Annuler()

{

Console.WriteLine("Annulation de la pizza!");

}

}

public class CommanderBoisson : ICommande

{

private readonly Cuisine \_cuisine;

public CommanderBoisson(Cuisine cuisine)

{

\_cuisine = cuisine;

}

public void Executer()

{

\_cuisine.PreparerBoisson();

}

public void Annuler()

{

Console.WriteLine("Annulation de la boisson!");

}

}

// Invoker

public class CommandeInvoker

{

private ICommande \_commande;

public void PrendreCommande(ICommande commande)

{

\_commande = commande;

}

public void ExecuterCommande()

{

\_commande.Executer();

}

public void AnnulerCommande()

{

\_commande.Annuler();

}

}

// Client (Serveur)

public class Serveur

{

public void CreerCommande()

{

Cuisine cuisine = new Cuisine();

ICommande commandePizza = new CommanderPizza(cuisine);

CommandeInvoker invoker = new CommandeInvoker();

invoker.PrendreCommande(commandePizza);

invoker.ExecuterCommande();

ICommande commandeBoisson = new CommanderBoisson(cuisine);

invoker.PrendreCommande(commandeBoisson);

invoker.ExecuterCommande();

}

}

public class Program

{

public static void Main(string[] args)

{

Serveur serveur = new Serveur();

serveur.CreerCommande();

}

}

Avec ce design, le serveur crée différentes commandes pour différents plats ou boissons. Chaque commande sait comment s'exécuter et comment être annulée, et le CommandeInvoker est responsable de l'exécution ou de l'annulation des commandes.