**Exercice :**

Vous êtes chargé de créer un système pour une chaîne de café où les boissons peuvent être personnalisées avec divers ajouts (comme du lait, du caramel, du chocolat, etc.). Utilisez le motif décorateur pour garantir que le système est extensible, c'est-à-dire que de nouveaux ajouts peuvent être introduits sans avoir à modifier la structure existante des classes de boissons.

1. Créez un diagramme UML représentant les classes principales ainsi que leurs relations.
2. Implémentez les classes en C#.

**Diagramme UML (Simplifié)**:

* Classe Boisson
  + propriété : description (String)
  + propriété : prix (Double)
  + méthode : GetDescription() -> String
  + méthode : GetPrix() -> Double
* Classe DecoreBoisson (abstraite)
  + hérite de Boisson
  + propriété : boisson (Boisson)
* Classe Lait
  + hérite de DecoreBoisson
* Classe Chocolat
  + hérite de DecoreBoisson
* Classe Caramel
  + hérite de DecoreBoisson

**Implémentation en C#**:

csharp

public abstract class Boisson

{

protected string description = "Boisson Inconnue";

public virtual string GetDescription()

{

return description;

}

public abstract double GetPrix();

}

public abstract class DecoreBoisson : Boisson

{

protected Boisson boisson;

public DecoreBoisson(Boisson boisson)

{

this.boisson = boisson;

}

}

public class Cafe : Boisson

{

public Cafe()

{

description = "Café";

}

public override double GetPrix()

{

return 1.50;

}

}

public class Lait : DecoreBoisson

{

public Lait(Boisson boisson) : base(boisson) { }

public override string GetDescription()

{

return boisson.GetDescription() + ", Lait";

}

public override double GetPrix()

{

return 0.50 + boisson.GetPrix();

}

}

public class Chocolat : DecoreBoisson

{

public Chocolat(Boisson boisson) : base(boisson) { }

public override string GetDescription()

{

return boisson.GetDescription() + ", Chocolat";

}

public override double GetPrix()

{

return 0.75 + boisson.GetPrix();

}

}

public class Caramel : DecoreBoisson

{

public Caramel(Boisson boisson) : base(boisson) { }

public override string GetDescription()

{

return boisson.GetDescription() + ", Caramel";

}

public override double GetPrix()

{

return 0.60 + boisson.GetPrix();

}

}

**Utilisation**:

csharp

var monCafe = new Cafe();

Console.WriteLine($"{monCafe.GetDescription()} coûte {monCafe.GetPrix()}€");

var monCafeAuLait = new Lait(monCafe);

Console.WriteLine($"{monCafeAuLait.GetDescription()} coûte {monCafeAuLait.GetPrix()}€");

var monCafeAuLaitChocolat = new Chocolat(monCafeAuLait);

Console.WriteLine($"{monCafeAuLaitChocolat.GetDescription()} coûte {monCafeAuLaitChocolat.GetPrix()}€");

Le principal avantage du design pattern décorateur est qu'il nous permet d'ajouter de nouvelles fonctionnalités à un objet existant sans altérer sa structure.

Haut du formulaire

Bas du formulaire