Cours : Programmation Orientée Objet (P+ C) en C#

Madani Bezoui

CESI, Campus de Nancy.

Objectifs du cours

- Introduire les concepts de base de la programmation orientée objet (POO) en C#.
- Apprendre à utiliser UML pour représenter des classes et des objets.
- Comprendre les attributs, méthodes et les différents niveaux d'accès (privé, public, protégé).
- Explorer l'héritage, les classes abstraites et les interfaces.
- Comparer les principaux langages orientés objet.
- Découvrir les structures de données (dictionnaire, liste, tableau, tuple) et leur utilisation avec le Standard Template Library (STL).

1 Introduction à la Programmation Orientée Objet (POO)

La **POO** est un paradigme de programmation où les concepts sont organisés autour des **objets**, qui peuvent contenir des données (attributs) et des méthodes (comportements). En C#, tout est un objet, et chaque programme consiste à manipuler ces objets.

Premier exemple en C# :

```
public class Voiture {
    public string marque;
    public string modele;
    public int annee;

    public void AfficherInfo() {
        Console.WriteLine($"Marque: {marque}, Modele: {modele},
            Annee: {annee}");
```

```
    }
}

public class Program {
    public static void Main() {
        Voiture maVoiture = new Voiture();
        maVoiture.marque = "Toyota";
        maVoiture.modele = "Corolla";
        maVoiture.annee = 2020;
        maVoiture.AfficherInfo();
    }
}
```

Explication :

- `Voiture` est une classe avec trois attributs : `marque`, `modele` et `annee`.
- `AfficherInfo()` est une méthode pour afficher les informations d'une voiture.
- Un objet `maVoiture` est créé et ses attributs sont initialisés.

2 Utilisation de l'UML pour la POO

UML (Unified Modeling Language) permet de modéliser les relations entre les objets sous forme de **diagrammes de classes**.

Classe UML :

- Attributs : Caractéristiques d'une classe.
- Méthodes : Fonctions exécutées par les objets.

Exemple UML :
Une classe `Voiture` aura :

- Attributs : `marque`, `modele`, `annee`.
- Méthodes : `AfficherInfo()`.

```
+----------------+
|   Voiture      |
+----------------+
| - marque       |
| - modele       |
| - annee        |
+----------------+
| + AfficherInfo()|
+----------------+
```

3 Les Attributs et les Méthodes

Attributs : Ce sont des variables membres d'une classe qui stockent des informations relatives aux objets de cette classe.

Méthodes : Ce sont des fonctions définies dans une classe qui exécutent des actions.

Visibilité :

- **public** : Accessible par tous.
- **private** : Accessible uniquement dans la classe.
- **protected** : Accessible dans la classe et ses sous-classes.

Exemple en C# :

```
public class Voiture {
    private string marque;
    protected string modele;
    public int annee;

    public void SetMarque(string m) {
        marque = m;
    }
}
```

Dans cet exemple :

- `marque` est privée, donc accessible uniquement via la méthode `SetMarque`.
- `modele` est protégée, donc accessible dans les classes héritées.
- `annee` est publique, donc accessible directement.

4 Héritage, Classes Abstraites et Interfaces

Héritage permet à une classe de dériver d'une autre classe pour réutiliser, étendre ou modifier ses fonctionnalités.

Exemple d'héritage en C# :

```
public class Vehicule {
    public int roues;
    public void Rouler() {
        Console.WriteLine("Le v hicule roule.");
    }
}

public class Voiture : Vehicule {
```

```
    public string marque;
}
```

Classe Abstraite :

C'est une classe qui ne peut pas être instanciée et doit être héritée. Elle peut contenir des méthodes abstraites, qui doivent être implémentées dans les classes dérivées.

```
public abstract class Animal {
    public abstract void Cri();
}

public class Chien : Animal {
    public override void Cri() {
        Console.WriteLine("Le chien aboie.");
    }
}
```

Interface :

C'est une collection de méthodes sans corps. Une classe qui implémente une interface doit définir toutes ses méthodes.

```
public interface IAnimal {
    void Cri();
}

public class Chat : IAnimal {
    public void Cri() {
        Console.WriteLine("Le chat miaule.");
    }
}
```

5 Comparaison des principaux langages orientés objet

Caractéristique	Java	C++	C#	Python	JavaSc
Héritage	Oui	Oui	Oui	Oui	Prototy
Interfaces	Oui	Oui	Oui	Non (modules)	Non
Gestion mémoire	Automatique (GC)	Manuelle	Automatique (GC)	Automatique (GC)	Automatiqu
Support du POO	Complet	Complet	Complet	Complet	Parti

Remarques :

- **GC (Garbage Collector)** : Gestion automatique de la mémoire en C#, Java, Python.
- C++ utilise une gestion manuelle de la mémoire (pointeurs).

6 Structures de données : Dictionnaire, Liste, Tableau, Tuple

Dictionnaire : Une collection d'éléments clés-valeurs.

Exemple en C# :

```
Dictionary<string, string> capitales = new Dictionary<string,
    string>();
capitales.Add("France", "Paris");
capitales.Add("Italie", "Rome");
```

Liste : Une collection d'éléments ordonnés.

Exemple :

```
List<string> fruits = new List<string>() { "Pomme", "Banane", "
    Orange" };
```

Tableau : Une collection d'éléments de taille fixe.

Exemple :

```
int[] nombres = { 1, 2, 3, 4, 5 };
```

Tuple : Une collection ordonnée de valeurs de types différents.

Exemple :

```
var tuple = (1, "Deux", true);
Console.WriteLine(tuple.Item1); // Affiche 1
```

Conclusion

À la fin de ce cours, vous aurez une compréhension solide des concepts fondamentaux de la programmation orientée objet en C#. Vous saurez modéliser des systèmes en utilisant UML et comprendre les différentes structures de données essentielles.

7 Exercices avec solutions

Exercice 1

Soit la classe `vecteur3d` qui contient les coordonnées du vecteur (float), un constructeur initialisant les données membres avec des valeurs par défaut à 0.

- Définir une méthode `plus1vec` fournissant la somme de deux vecteurs.
- Définir une méthode `ProdScalaire()` pour qu'elle fournisse le produit scalaire de deux vecteurs.
- Créer un programme permettant de tester cette classe.

Exercice 2

Écrire une classe `vecteur` comportant :

- en membres donnée privés : trois composantes de type double.
- un constructeur qui initialise les valeurs des composantes.
- des méthodes permettant :
  - de multiplier les composantes par une valeur,
  - de comparer 2 vecteurs,
  - d'afficher et de lire les composantes d'un vecteur.

Créer un programme permettant de tester la classe `vecteur`.

Solutions

Solution Exercice 1

```
class vecteur3d {
    private:
        float x, y, z;

    public:
        // Constructeur par défaut
        vecteur3d() : x(0), y(0), z(0) {}

        // Constructeur avec paramètres
        vecteur3d(float a, float b, float c) : x(a), y(b), z(c) {}

        // Méthode pour ajouter deux vecteurs
        vecteur3d plus1vec(const vecteur3d& v) {
            return vecteur3d(x + v.x, y + v.y, z + v.z);
        }

        // Méthode pour le produit scalaire
        float ProdScalaire(const vecteur3d& v) {
            return x * v.x + y * v.y + z * v.z;
        }

        // Afficher les coordonnées du vecteur
        void afficher() {
            std::cout << "(" << x << ", " << y << ", " << z << ")" << std::endl;
        }
};

// Programme principal pour tester la classe
int main() {
    vecteur3d v1(1.0, 2.0, 3.0);
    vecteur3d v2(4.0, 5.0, 6.0);

    vecteur3d somme = v1.plus1vec(v2);
    float produit = v1.ProdScalaire(v2);

    std::cout << "Somme des vecteurs : ";
    somme.afficher();

    std::cout << "Produit scalaire : " << produit << std::endl;

    return 0;
}
```

Solution Exercice 2

```
class vecteur {
    private:
        double x, y, z;

    public:
        // Constructeur
        vecteur(double a = 0, double b = 0, double c = 0) : x(a), y(b), z(c) {}

        // Méthode pour multiplier les composantes
        void multiplier(double facteur) {
            x *= facteur;
            y *= facteur;
            z *= facteur;
        }

        // Méthode pour comparer deux vecteurs
        bool comparer(const vecteur& v) {
            return (x == v.x && y == v.y && z == v.z);
        }

        // Méthodes pour afficher et lire les composantes
        void afficher() const {
            std::cout << "(" << x << ", " << y << ", " << z << ")" << std::endl;
        }

        void lire() {
            std::cout << "Entrez les composantes (x, y, z) : ";
            std::cin >> x >> y >> z;
        }
};

// Programme principal pour tester la classe
int main() {
    vecteur v1(1.0, 2.0, 3.0);
    vecteur v2;

    v2.lire();

    v1.afficher();
    v2.afficher();

    if (v1.comparer(v2)) {
        std::cout << "Les vecteurs sont égaux." << std::endl;
    } else {
```

```
        std::cout << "Les vecteurs sont différents." << std::endl;
    }

    v1.multiplier(2.0);
    std::cout << "Vecteur v1 après multiplication : ";
    v1.afficher();

    return 0;
}
```