

Base de la Programmation Objet avec C#

Définition

Membres

La programmation orientée objet consiste à rapprocher les traitements (fonctions) des données (variables). Cela permet de modéliser des situations de façon plus logique et naturelle. La POO s'articule donc autour de structures appelées **classes** qui possède un état (des attributs) et des comportements (les méthodes). L'ensemble des attributs et méthodes d'une classe sont appelées les membres de la classe.

En C#, on peut définir une classe à l'aide du mot clé `class`. Une classe doit être définie dans un fichier qui porte son nom. Pas convention, les noms des classes sont en PascalCase. Les noms des membres camelCase.

Définition d'une classe :

```
class Cat {  
  
}
```

Pour définir un attribut, on précise d'abord son type, puis son nom :

```
class Cat {  
    string name;  
}
```

Pour définir une méthode, on précise d'abord son type de retour, puis son nom, puis ses arguments :

```
class Cat {  
    string name;  
  
    void Pet(int duration) {  
        // Corps de la méthode  
    }  
}
```

Constructeur

Le constructeur est un membre (plus spécifiquement une méthode) particulier de la classe, qui est appelé à son instantiation et qui sert à initialiser l'état de l'objet.

Pour définir un constructeur on définit une méthode qui a le nom de la classe :

```
class Cat {  
    string name;  
  
    Cat(string catName){  
        name = catName;  
    }  
}
```

Une classe peut avoir plusieurs constructeurs (avec différents prototypes). Un constructeur peut faire appel à un autre par un appel à `this()`.

```
class Cat {  
    string name;  
  
    Cat() : this("Felix"){  
    }  
  
    Cat(string catName){  
        name = catName;  
    }  
}
```

Classes et Instances

Instances

Une classe est un type qui définit de quels membres seront dotés ses instances. Une instance d'une classe (aussi appelée objet), est une variable du type de la classe et possède tous les membres définis par cette dernière, avec des valeurs qui lui sont propres. On peut instancier un objet avec le mot clé `new`, suivi d'un appel au constructeur de la classe :

```
Cat cat = new Cat("Félix");
```

On peut accéder aux membres (attributs ou méthodes) de la classe avec l'opérateur `.` sur la référence de l'objet :

```
Console.WriteLine(cat.name);  
cat.Pet(10);
```

Membres de classe

On peut aussi déclarer dans une classe des membres (attributs ou méthodes) qui seront communs à toutes les instances d'une classe (une valeur pour toutes les instances) grâce au mot clé `static` :

```
class Cat {  
    static numberOfCats = 0;  
  
    Cat(string catName){  
        name = catName;  
        numberOfCats++;  
    }  
}
```

Les membres de classe peuvent être utilisés au sein de la classe, ou alors à l'extérieur avec l'opérateur `.` sur le nom de la classe :

```
Console.WriteLine(Cat.numberOfCats);
```

Référence `this`

La référence `this` est accessible dans toute classe dans les contextes non `static`. Elle pointe vers l'instance courante. Elle est utilisée pour lever des ambiguïtés de nommage :

```
class Cat {  
    Cat(string name){  
        this.name = name;  
    }  
}
```

Elle est également utilisée pour des raisons de visibilité afin de permettre de dissocier rapidement une variable locale d'un attribut d'instance.

Références & Garbage Collection

En C#, toutes les classes sont de types référence. Les `struct` ainsi que les primitifs :

- int
- char
- long
- double
- float
- bool
- short
- byte

sont des types valeur.

Tous les classes sont des types références, c'est-à-dire que les variables contiennent les adresses mémoires des objets. Tous les objets sont alloués en mémoire sur le tas, et c'est le runtime .NET qui s'occupe de les désallouer lorsqu'ils ne sont plus référencés via un mécanisme appelé la **Garbage Collection**.

Encapsulation

Namespaces

Les namespaces sont les espaces de noms pour organiser le code. Par convention on les fait correspondre à la structure de dossiers. Leur nomage est par convention en PascalCase.

Pour déclarer le namespace dans un fichier (en haut par convention) :

```
namespace MonNamespace;
```

Visibilité

La notion de visibilité permet de restreindre l'accès aux membres d'une classe. Il existe 5 modificateurs d'accès :

Modificateur	Classe	Projet	Classe Enfant	Tout le reste
<code>public</code>	Oui	Oui	Oui	Oui
<code>internal</code>	Oui	Oui	Non	Non
<code>protected</code>	Oui	Non	Oui	Non
<code>protected internal</code>	Oui	Oui	Oui	Non
<code>private</code>	Oui	Non	Non	Non

La visibilité par défaut (pas de modification de visibilité) est la visibilité `private`.

Principe d'encapsulation

Le principe d'encapsulation dicte que seul les informations que l'interface publique (l'ensemble des membres publics) d'une classe doit être la plus restreinte possible. C'est pourquoi tous les attributs doivent être privés. Les méthodes sont publiques que si elles ont besoin de l'être.

Pour exposer avec une granularité fine les attributs, on utilise des méthodes un peu spéciales, les propriétés. Les propriétés peuvent contenir :

- Un **accesseur**, pour lire un champs. Avec le mot clé `get`
- Un **mutateur**, pour écrire un champs. Avec le mot clé `set`

Exemple de champs correctement encapsulé :

```
namespace Animals;

public class Cat {
    private string name;

    public string Name {
        get => this.name;
        set => this.name = value;
    }

    public Cat(String name){
        this.name = name;
    }
}
```

```
}  
}
```

On peut simplifier cette syntaxe, avec les propriétés automatiques, elles déclarent automatiquement leur champ privé :

```
namespace Animals;  
  
public class Cat {  
    public string Name {get;set;}  
  
    public Cat(String name){  
        this.Name = name;  
    }  
}
```

Il ne faut déclarer un accesseurs ou un mutateur sur une propriétés que si besoin, par exemple :

```
namespace Animals;  
  
public class Cat {  
    public string Name {get;}  
  
    public Cat(String name){  
        this.Name = name;  
    }  
}
```

On peut aussi jouer sur la visibilité :

```
namespace Animals;  
  
public class Cat {  
    public string Name {get;private set;}  
  
    public Cat(String name){  
        this.name = name;  
    }  
}
```