

Chapitre 3

Programmation orientée objet

1. Principes de la programmation orientée objet

La programmation orientée objet (POO) est un paradigme très répandu en développement logiciel. Il vient compléter un panorama déjà riche du paradigme procédural ainsi que du paradigme fonctionnel.

La POO est une forme de conception de code visant à représenter les données et les actions comme faisant partie de classes, elles-mêmes devenant des objets lors de leur création en mémoire. Cette notion a été rapidement présentée dans le chapitre précédent, il est maintenant temps de comprendre son fonctionnement plus en détail.

1.1 Qu'est-ce qu'une classe ?

Une classe est un élément du système que forme votre application. Une classe contient deux types d'élément de code : des données ainsi que des méthodes, représentant des actions. Il faut voir la classe comme étant une boîte dans laquelle il est possible de ranger ces deux types d'éléments. Pour faire un parallèle avec la vie réelle, nous pouvons facilement comprendre que la définition d'une classe s'applique à un objet comme un ordinateur, par exemple. Ce dernier dispose de méthodes (allumer, éteindre...) ainsi que des propriétés (nombre d'écrans, quantité de RAM...).

Conceptuellement, une classe n'est qu'une définition. Une fois que vous avez statué sur ce qu'elle doit contenir ainsi que ses méthodes, il convient de la créer. Cette action s'appelle l'instanciation. À la suite de cette opération, nous obtenons une instance en mémoire d'un objet.

Pour tenter une comparaison, prenons l'exemple d'une usine de fabrication d'objets en bois. Afin de pouvoir créer un objet, il faut un plan (la classe). Grâce à ce dernier, la machine peut découper et assembler les divers éléments (les données et méthodes) afin de créer une nouvelle instance (instanciation).

En C#, la déclaration d'une classe se fait grâce au mot-clé `class`. Il y a quelques spécificités possibles, notamment la portée, que nous étudierons juste après, dans la section *Que peut-on déclarer dans une classe ?* - Les méthodes, ainsi que les concepts de `static`, `sealed` et celui de `partial`. La syntaxe complète de la déclaration d'une classe est la suivante :

```
PORTÉE [static] [sealed] [partial] class NOM_CLASSE
```

Le nom de la classe est libre mais répond à deux règles :

- Il ne peut contenir que des caractères alphanumériques et le signe underscore (« _ »).
- Il ne peut pas commencer par un chiffre.

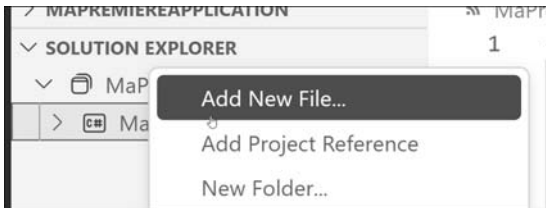
En plus de ces règles, les développeurs C# respectent souvent une convention syntaxique : l'utilisation du *PascalCase*. Cela indique que le nom commence par une majuscule et chaque mot est symbolisé par une majuscule également, par exemple, `OrdinateurPortable`. Le langage et le compilateur n'interdisent pas d'écrire `ordinateurPortable`, `Ordinateurportable` ou encore `ordinateurportable`, mais ces différentes déclarations ne respectent pas la convention largement admise et appliquée. Finalement, bien que ce soit possible, il est recommandé d'éviter tout caractère accentué dans le nom d'une classe. Par exemple, il est préférable d'appeler sa classe `Pieton` plutôt que `Piéton`, cela afin que le code C# produit soit le plus proche possible de ce que nous aurions en anglais.

Dans le programme de base créé en C# dans le précédent chapitre, une classe `Program` est créée par défaut. Nous pouvons constater qu'il n'y a ni notion de portée ni notion de `partial` ou `static`. Une fois qu'une classe est déclarée, elle définit un bloc, dans lequel nous pouvons implémenter les données et les méthodes dont notre programme a besoin pour fonctionner.

1.1.1 Les classes dans Visual Studio Code

Pour créer une classe dans Visual Studio Code, il est nécessaire de suivre les étapes suivantes :

- ❑ Dépliez la vue **Solution Explorer** du projet.
- ❑ Placez-vous sur le dossier où vous souhaitez créer la nouvelle classe (ou directement sur le nom du projet si vous souhaitez la créer à la racine).
- ❑ Faites un clic droit pour sélectionner l'élément de menu **Add New File**.
- ❑ Renseignez le nom de la classe dans la petite pop-up qui s'est ouverte en haut au centre de l'écran (toujours sans espaces ni caractères spéciaux).



Ajout d'une nouvelle classe avec Visual Studio Code

À la suite de ces manipulations, un nouveau fichier, portant le nom de la classe suivi de l'extension `.cs`, est disponible dans la hiérarchie à gauche. Par défaut, ce fichier sera ouvert et contient la classe qui a été déclarée dans l'espace de noms correspondant au dossier de destination.

1.1.2 L'héritage

Il existe un concept extrêmement important en POO : l'héritage. Globalement, si vous avez la possibilité de dire « X est un Y », l'équivalent pourrait être de dire « X hérite de Y ». X reprend toutes les propriétés et tous les comportements de Y, mais le spécifie. Donnons un exemple concret : « Un Mac est un ordinateur ». Donc, au niveau du développement orienté objet, un Mac reprend toutes les propriétés d'un ordinateur ainsi que ses comportements, mais les spécifie en y apportant ses propres éléments. On dit dans ces cas-là que Mac est une classe fille de la classe Ordinateur.

En C#, cette notion est centrale car tous les éléments que vous allez manipuler héritent naturellement de la classe `System.Object`, qui définit le comportement de base de n'importe quel objet. De surcroît, contrairement à d'autres langages (comme le C++), il n'est pas possible, en C#, d'hériter de plusieurs classes : une seule classe mère est possible. Si aucune classe mère n'est spécifiée, c'est par définition la classe `System.Object` qui constitue la classe mère (sans qu'une quelconque manipulation soit requise).

■ Remarque

En C#, il n'est pas possible d'hériter de plusieurs classes. Il faut donc choisir la classe dont on hérite. En l'absence de précision, le compilateur génère automatiquement, de façon transparente, un héritage de la classe `System.Object`, comme décrit ci-dessus. Si nous spécifions un héritage, cela ne veut pas dire que la classe hérite de `System.Object` ET de la classe héritée, mais uniquement de la classe héritée, qui remplace l'héritage généré par le compilateur. La classe héritée, elle-même, hérite soit d'une autre classe, soit directement de `System.Object`. En finalité, toutes les classes en C# héritent d'une façon ou d'une autre de `System.Object`.

Afin d'indiquer qu'une classe hérite d'une autre, il faut utiliser le deux-points, suivi de la classe dont on souhaite hériter :

```
class Ordinateur { }  
class Mac : Ordinateur { }
```

Bien entendu, ce n'est pas parce qu'une classe hérite d'une autre qu'elle a forcément accès à tout ce qui a été défini au sein de la classe mère.

1.1.3 L'encapsulation

Tout ce qui se trouve à l'intérieur d'une classe est désigné par un terme bien spécifique : l'encapsulation. Avec celle-ci vient également la notion de portée, qui indique comment les choses sont perçues d'un point de vue extérieur à la classe.

La portée permet de définir la visibilité d'un élément d'une classe ou de la classe elle-même. Il existe en tout sept portées en C# :

- `public` : définit que l'élément est totalement visible dans et en dehors de la classe.
- `private` : définit que l'élément n'est visible qu'au sein de la classe où il est déclaré alors qu'il est totalement invisible de l'extérieur.
- `internal` : définit que l'élément est visible uniquement au sein du projet où il est déclaré. Nous pouvons considérer l'élément comme étant `public`, mais simplement au sein du projet dans lequel il est déclaré. Un autre projet qui référence notre projet n'a pas connaissance d'un élément déclaré comme `internal`. Par défaut, en l'absence de portée explicite sur une classe, c'est la portée `internal` qui est sélectionnée par le compilateur.
- `protected` : définit que l'élément est visible uniquement au sein de la classe où il est déclaré ainsi que dans sa hiérarchie de classes filles. Cela rejoint le concept de l'héritage, que nous verrons plus loin dans ce chapitre.
- `protected internal` : définit un cumul entre `protected` et `internal`. Un élément déclaré avec cette portée est visible par la classe concernée ainsi que ses classes filles, tout comme par toutes les autres classes au sein du même projet. Cela signifie également que si une classe fille est déclarée en dehors du projet actuel, elle peut accéder à un élément `protected internal`, tout comme n'importe quelle classe du même projet.
- `private protected` : définit une intersection entre `protected` et `internal`. Un élément déclaré avec cette portée n'est visible que par la classe concernée ainsi que les classes filles qui sont définies au sein du même projet. Cela veut dire qu'une classe fille définie en dehors du projet actuel ne pourra pas accéder à cet élément.

- `file` : ajoutée en C# 11, cette portée définit une visibilité uniquement dans le cadre du fichier en cours. Cette portée est très particulière car elle n'a pas vocation à être directement utilisée par les développeurs. Elle existe surtout pour les outils de génération automatique de code. Néanmoins, dans de très rares cas, il peut être utile de déclarer un élément qui n'existe que dans le cadre d'un fichier pour un algorithme précis. Il est à noter également que, contrairement aux autres portées, cette portée n'est valide que pour la déclaration d'un type. Elle ne peut pas s'appliquer sur une méthode, un champ ou une propriété.

Avec toutes ces portées, il est possible de créer la classe qui correspond finement au besoin de votre application, pour éviter que certains éléments ne sortent du périmètre de la classe. En reprenant notre exemple, considérons que la classe `Ordinateur` dispose d'un booléen indiquant si la machine est allumée ou non. Afin d'éviter que quelqu'un ne puisse manipuler directement cette donnée, la manière de procéder est de la définir comme étant publiquement accessible en lecture, mais privée pour ce qui est de l'écriture. En conséquence, seule une méthode publique, définie dans cette classe, comme par exemple `Allumer` ou `Eteindre`, peut changer la valeur de cet indicateur. Nous nous préservons ainsi d'un changement d'état non maîtrisé (car nous pouvons considérer que l'opération d'extinction nécessite d'effectuer quelques opérations en amont avant de basculer le booléen).

1.2 Que peut-on déclarer dans une classe ?

Nous l'avons vu, il existe deux types d'éléments que nous pouvons déclarer dans une classe : des méthodes (actions) et des données. Voyons rapidement comment les déclarer.

1.2.1 Les méthodes

Une méthode traduit une action qu'il est possible d'invoquer sur la classe. Lors de la déclaration d'une méthode, il faut se poser les questions suivantes :

- S'agit-il d'une action qui doit pouvoir être réalisée depuis l'extérieur ou uniquement depuis l'intérieur de la classe ?
- Est-ce qu'une valeur de retour particulière est attendue ?

- Certaines informations sont-elles nécessaires pour que cette méthode fonctionne ?

Vous avez déjà eu un aperçu d'un appel de méthode dans le premier chapitre, sur la classe `Console` : `WriteLine` et `ReadLine`. Ces deux méthodes illustrent bien les points cités précédemment :

- `WriteLine` doit pouvoir être appelée depuis l'extérieur. Nous n'attendons pas de valeur en retour à son appel, mais il est nécessaire de lui transmettre l'information que nous souhaitons écrire.
- `ReadLine` doit également pouvoir être appelée depuis l'extérieur. Nous avons besoin de récupérer l'information saisie par l'utilisateur uniquement, sans besoin de lui transmettre une quelconque information.

La syntaxe de déclaration d'une méthode dans une classe est la suivante :

```
PORTÉE [static] TYPE_RETOUT NOM_METHODE([PARAMÈTRES])
```

Le type de retour doit correspondre à un type C# connu. Par exemple, si nous souhaitons créer une méthode qui réalise l'addition de deux nombres et renvoie le résultat, le tout accessible publiquement, nous la déclarons comme suit :

```
public int Addition(int premier, int second) {}
```

■ Remarque

Dès lors que nous déclarons une méthode avec une valeur de retour sans écrire le contenu de la méthode, le compilateur émet immédiatement une erreur de compilation. Ceci est dû au fait que chaque méthode retournant un résultat doit obligatoirement comporter une instruction `return`.

Lorsqu'une méthode doit renvoyer une valeur, il faut utiliser le mot-clé `return` afin de définir la valeur que nous souhaitons renvoyer. L'instruction `return` peut être utilisée directement avec une valeur ou alors nous pouvons nous servir d'une variable du type de retour attendu. Dans le cas de l'exemple ci-dessus, ces deux façons d'écrire la méthode sont valides :

```
public int Addition(int premier, int second)
{
    return premier + second;
}
```

```
public int Addition(int premier, int second)
{
    int resultat = premier + second;
    return resultat;
}
```

Un élément important à garder en mémoire : à l'instar de ce que nous avons vu dans le chapitre précédent avec la déclaration de classes du même nom au sein du même espace de noms, il n'est pas possible de déclarer deux fois la même méthode à l'intérieur d'une même classe. Si les noms sont identiques et que les paramètres le sont également, alors le compilateur C# considère qu'il s'agit de la même méthode. La valeur de retour ne constitue pas un élément distinctif. Ainsi, la déclaration des deux méthodes suivantes dans la même classe est impossible et cela provoque une erreur de compilation :

```
public int Addition (int premier, int second)
{
    return premier + second;
}
public void Addition (int premier, int second)
{
}
```

Remarque

Comme nous pouvons le constater dans l'exemple ci-dessus, le mot-clé `void` précise que la méthode ne renvoie aucun résultat. La notion de type de retour étant obligatoire, il faut utiliser ce mot-clé pour indiquer les cas où il n'y en a pas.

Si la méthode ne prend pas de paramètres, la présence de parenthèses ouvrantes et fermantes accolées au nom de la méthode est malgré tout nécessaire pour signifier qu'il s'agit d'une méthode :

```
public void MaMethode()
{
}
```

Au sein d'une méthode qui déclare son propre bloc, il est possible de déclarer des variables et constantes qui sont considérées uniquement comme locales (c'est-à-dire visibles au sein de la méthode et de tous ses sous-blocs, mais invisibles dans les blocs parents, directs ou indirects).

Chapitre 3

Les ordres du SQL

1. Les bases du langage SQL

1.1 Les expressions

Dans la plupart des syntaxes Transact-SQL, il est possible d'utiliser des expressions ou des combinaisons d'expressions pour gérer des valeurs ou pour tirer parti de la programmabilité du langage. Les expressions peuvent prendre différentes formes :

Les valeurs constantes

Exemple

■ **Caractère** 'AZERTY', 'Ecole Nantaise d''Informatique'

Les chaînes de caractères sont entourées d'apostrophes. S'il doit y avoir une apostrophe dans la chaîne de caractères, il est nécessaire d'en mettre deux pour ne pas le confondre avec la fin de la chaîne de caractères.

■ **Numérique** 10, -15.26, 1.235e-5

Les valeurs numériques peuvent être écrites avec des chiffres en utilisant le point pour séparer la partie entière de la partie décimale. Il est possible d'utiliser la notation scientifique en utilisant le *e* avant la puissance de :

Date		
constante	date	heure
'231205'	5 Décembre 2023	00:00:00.000
'05/12/2023'	5 Décembre 2023	00:00:00.000
'05-12-23 8:30'	5 Décembre 2023	08:30:00:000
'8:30:2'	1 Janvier 1900	08:30:02:000
'5.12.2023 8:30pm'	5 Décembre 2023	20:30:00:000

Le format français jj/mm/aaaa est l'un des formats possibles pour les dates.

■ **Binaire** 0x05, 0xFF, 0x5aef1b
■ **Nulle** NULL

Les données binaires doivent être données en hexadécimal et sont préfixées par 0x.

Les noms de colonne

Un nom de colonne pourra être employé comme expression, la valeur de l'expression étant la valeur "stockée" de la colonne.

Les fonctions

On peut utiliser comme expression n'importe quelle fonction, la valeur de l'expression est le résultat retourné par la fonction.

Exemple

expression	valeur
SQRT(9)	3
substring('ABCDEF',2,3)	'BCD'

Les variables

Les variables peuvent être employées en tant qu'expression ou dans une expression, sous la forme @nom_de_variable. La valeur de l'expression est la valeur de la variable.

Exemple

```
DECLARE @x CHAR(10);
SELECT @x='AZERTY';
SELECT LOWER(@x);
```

Résultats		Messages
(Aucun nom de colonne)		
1	azerty	

Les sous-requêtes

Une requête SELECT entre parenthèses peut être employée en tant qu'expression ayant pour valeur le résultat de la requête, soit une valeur unique, soit un ensemble de valeurs. Les requêtes SELECT sont présentées à la section L'extraction de lignes de ce chapitre.

Exemple

Stockage du nombre de clients dans une variable :

```
DECLARE @nombre INT;
SELECT @nombre = (SELECT COUNT(*) FROM Clients);
SELECT @nombre;
```

Résultats		Messages
(Aucun nom de colonne)		
1	0	

Les expressions booléennes

Elles sont destinées à tester des conditions (IF, WHILE, WHERE, etc.). Ces expressions sont composées de la manière suivante :

```
expression1 opérateur expression2
```

1.2 Les opérateurs

Les opérateurs vont permettre de constituer des expressions calculées, des expressions booléennes ou des combinaisons d'expressions.

1.2.1 Les opérateurs arithmétiques

Ils permettent d'effectuer des calculs élémentaires et de renvoyer un résultat.

+ Addition

- Soustraction

* Multiplication

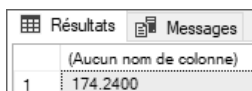
/ Division : division entière si les deux opérandes sont des entiers, division réelle sinon.

% Modulo (reste de la division entière)

(...) Parenthèses

Exemple

```
DECLARE @prixArticle NUMERIC(6,2) = 145.2;  
DECLARE @TVA NUMERIC(2,2) = 20.0/100;  
SELECT @prixArticle*(1+@TVA);
```



Résultats		Messages
(Aucun nom de colonne)		
1	174.2400	

```
DECLARE @nb INT = (SELECT COUNT(*) FROM Clients);  
IF @nb%2=0 PRINT 'nombre de clients pair';  
ELSE PRINT 'nombre de clients impair';
```

nombre de clients pair

Remarque

Les opérateurs + et - fonctionnent également sur les dates. L'unité est alors le jour. Exemple : `GETDATE() + 1` permet d'obtenir la date du lendemain.

1.2.2 La concaténation de chaînes de caractères

La concaténation permet de constituer une seule chaîne de caractères à partir de plusieurs expressions de type caractère. L'opérateur de concaténation est le signe plus (+).

Exemple

```
DECLARE @nom CHAR(4) = 'LEVY';  
DECLARE @prenom CHAR(5) = 'Simon';  
SELECT 'M. ' + @nom + ' ' + @prenom;
```

Résultats		Messages
	(Aucun nom de colonne)	
1	M. LEVY Simon	

Il existe de plus la fonction `CONCAT()` qui permet de réaliser des concaténations. Contrairement à l'opérateur `+`, cette fonction appartient à la norme SQL. Elle a également l'avantage de pouvoir être utilisée avec des nombres sans que la somme soit effectuée.

1.2.3 Les opérateurs binaires

Ils permettent le calcul entre entiers, traduits implicitement en valeurs binaires.

& ET

| OU

^ OU Exclusif

~ NON

1.2.4 Les opérateurs de comparaison

Ils permettent la constitution d'expressions booléennes en comparant des expressions. Ces expressions peuvent être placées entre parenthèses.

`exp1 = exp2`

Égal.

`exp1 > exp2`

Supérieur.

`exp1 >= exp2`

Supérieur ou égal.

■ Remarque

`exp1 != exp2` est également possible mais n'est pas dans la norme.

`exp1 < exp2`

Inférieur.

`exp1 <= exp2`

Inférieur ou égal.

■ Remarque

`exp1 != exp2` est également possible mais n'est pas dans la norme.

`exp1 <> exp2`

Différent.

■ Remarque

`exp1 != exp2` est également possible mais n'est pas dans la norme.

`exp IN (exp1, exp2, ...)`

Indique si la valeur est parmi celles de la liste de valeurs.

`exp IS NULL` ou `exp IS NOT NULL`

Teste l'absence de valeur (NULL). Pour tester si une variable n'a pas de valeur affectée, il est indispensable d'utiliser l'opérateur IS NULL.

`exp LIKE 'masque'`

Filtre la chaîne de caractères ou la date suivant le masque spécifié.

Le masque peut être composé de :

—

Un caractère quelconque.

%

n caractères quelconques, avec n compris entre zéro et l'infini.

[ab...]

Un caractère dans la liste ab... .

[a-z]

Un caractère dans l'intervalle a-z.

[^ab...]

Un caractère en dehors de la liste ou de l'intervalle spécifié.

ab...

Le caractère lui-même.

Pour utiliser `_`, `%`, `[` et `^` en tant que caractères de recherche, il faut les encadrer de `[]`.

Exemple

masque	Chaînes correspondantes
'G%'	commençant par "G"
'_X%1'	deuxième caractère "X", dernier "1"
'%[1-9]'	se terminant par un chiffre compris entre "1" et "9"
'[^XW]%'	ne commençant ni par X ni par W
'LE[_]%'	commençant par "LE_"

exp LIKE 'masque' ESCAPE '\'

Lorsque le masque contient un caractère spécial comme `_` ou `&`, il est nécessaire de les faire précéder d'un caractère spécifique appelé caractère d'échappement. Afin de s'adapter à toutes les situations possibles, le caractère d'échappement est libre et il est simplement nécessaire de le spécifier après le mot-clé ESCAPE.

`exp BETWEEN min AND max`

Teste si la valeur de `exp` est incluse dans l'intervalle compris entre les valeurs `min` et `max` (bornes incluses).

`EXISTS (sous-requête)`

Renvoie VRAI si la sous-requête a renvoyé au moins une ligne.

Les opérateurs booléens

Ils permettent de combiner des expressions booléennes (`expb`) en renvoyant une valeur booléenne.

`expb1 OR expb2`

Vrai si une des deux expressions est vraie.

`expb1 AND expb2`

Vrai si les deux expressions sont vraies.

`NOT expb`

Vrai si `expb` est fausse.

1.3 Les fonctions

De nombreuses fonctions sont disponibles pour valoriser des colonnes ou pour effectuer des tests. Des exemples sont donnés avec l'instruction `SELECT`.

Parmi toutes les fonctions proposées en standard par SQL Server, il est possible de les regrouper par type : rowset, agrégation, ranking, scalaire. Pour ce dernier type de fonctions, il faut en plus les classer par catégories : mathématique, chaîne de caractères, date... car elles sont très nombreuses.

Certaines fonctions, plus particulièrement les fonctions permettant de manipuler les données de type caractère, prennent en compte le classement défini au niveau du serveur.