Chapitre 4
Les bases du langage

1. Introduction

Comme tous les langages de programmation, Visual Basic .NET impose certaines règles au développeur. Ces règles se manifestent au travers de la syntaxe du langage mais elles couvrent le large spectre fonctionnel proposé par VB.NET. Avant d'explorer en profondeur les fonctionnalités du langage au chapitre suivant, nous étudierons donc les notions essentielles et fondamentales de VB.NET : la création de données utilisables par une application et le traitement de ces données.

2. Les variables

Les données utilisables dans un programme VB.NET sont représentées par des variables. Une variable est un espace mémoire réservé auquel on assigne arbitrairement un nom et dont le contenu est une valeur dont le type est fixé. On peut manipuler ce contenu dans le code en utilisant le nom de la variable.

2.1 Nommage des variables

La spécification du langage établit quelques règles à prendre en compte lorsque l'on nomme une variable :

– Le nom d'une variable ne peut comporter que des chiffres, des caractères de l'alphabet latin, accentués ou non, le caractère ç ou les caractères spéciaux suivants : _ µ
– Le nom d'une variable ne peut en aucun cas commencer par un chiffre. Il peut en revanche en comporter un ou plusieurs à toute autre position.
– Le langage est insensible à la casse, c'est-à-dire qu'il ne fait aucune distinction entre majuscules et minuscules : les variables `unevariable` et `uneVariable` sont équivalentes.
– Un nom de variable ne peut pas excéder 511 caractères. Il n'est évidemment pas recommandé d'avoir des noms de variable aussi longs, le maximum en pratique étant plus souvent de l'ordre de la trentaine de caractères.
– Le nom d'une variable ne peut pas être un mot-clé du langage. Il est toutefois possible d'encadrer un mot-clé par les caractères [et] pour utiliser un nom de variable similaire.

Les noms de variables suivants sont acceptés par le compilateur Visual Basic :

– `maVariable`
– `maVariableNumero1`
– `[Next]` (`Next` est un mot-clé de VB.NET)
– `µn3_VàRïãbl3`

De manière générale, il est préférable d'utiliser des noms de variables explicites, c'est-à-dire permettant de savoir à quoi correspond la valeur stockée dans la variable, comme `nomClient`, `montantAchat` ou `ageDuCapitaine`.

2.2 Type des variables

Une des caractéristiques de VB.NET est la notion de typage statique : chaque variable correspond à un type de données et ne peut en changer. De plus, ce type doit être déterminable au moment de la compilation.

2.2.1 Types valeurs et types références

Les différents types utilisables avec VB.NET peuvent être décomposés en deux familles : les types valeurs et les types références. Cette notion peut être déconcertante au premier abord, puisqu'une variable représente justement une donnée et donc une valeur. Ce concept est en fait lié à la manière dont est stockée l'information en mémoire.

Lorsque l'on utilise une variable de type valeur, on accède directement à la zone mémoire stockant la donnée. Au moment de la création d'une variable de type valeur, une zone mémoire de la taille correspondant au type est allouée. Chaque octet de cette zone est automatiquement initialisé avec la valeur binaire 00000000. Notre variable aura donc une suite de 0 pour valeur binaire.

Dans le cas d'une variable de type référence, le comportement est différent. La zone mémoire allouée à notre variable contient une adresse mémoire à laquelle est stockée la donnée. On passe donc par un intermédiaire pour accéder à notre donnée. L'adresse mémoire est initialisée avec une valeur spéciale qui ne pointe sur rien, tandis que la zone mémoire contenant les données n'est pas initialisée. Elle sera initialisée lorsque la variable sera instanciée. Dans le même temps, l'adresse mémoire stockée dans notre variable sera mise à jour.

Une variable de type référence pourra donc ne contenir aucune donnée, tandis qu'une variable de type valeur aura forcément une valeur correspondant par défaut à une suite de 0 binaires.

Cette différence de fonctionnement a une conséquence importante : la copie de variable se fait par valeur ou par référence, ce qui signifie qu'une variable de type valeur sera effectivement copiée, tandis que pour un type référence, c'est l'adresse contenue par la variable qui sera copiée, et il sera donc possible d'agir sur la donnée réelle indifféremment à partir de chacune des variables pointant sur ladite donnée.

2.2.2 Types intégrés

Le framework .NET embarque plusieurs milliers de types différents utilisables par les développeurs. Parmi ces types, nous en avons une quinzaine que l'on peut considérer comme fondamentaux : ce sont les types intégrés (aussi nommés types primitifs). Ce sont les types de base à partir desquels sont construits les autres types de la BCL ainsi que ceux que le développeur crée dans son propre code. Ils permettent de définir des variables contenant des données très simples.

Ces types ont comme particularité d'avoir chacun un alias intégré au langage.

Types numériques

Ces types permettent de définir des variables numériques entières ou décimales. Ils couvrent des plages de valeurs différentes et ont chacun une précision spécifique. Certains types seront donc plus adaptés pour les calculs entiers, d'autres pour les calculs dans lesquels la précision décimale est très importante, comme les calculs financiers.

Les différents types numériques sont énumérés ci-dessous avec leurs alias ainsi que les plages de valeurs qu'ils couvrent.

Type .NET	Alias VB	Plage de valeurs couverte	Taille en mémoire
System.Byte	Byte	0 à 255	1 octet
System.SByte	SByte	-128 à 127	1 octet
System.Int16	Short	-32768 à 32767	2 octets
System.UInt16	UShort	0 à 65535	2 octets
System.Int32	Integer	-2147483648 à 2147483647	4 octets
System.UInt32	UInteger	0 à 4294967295	4 octets
System.Int64	Long	-9223372036854775808 à 9223372036854775807	8 octets
System.UInt64	ULong	0 à 18446744073709551615	8 octets
System.Single	Single	±1,5e-45 à ±3,4e38	4 octets

Type .NET	Alias VB	Plage de valeurs couverte	Taille en mémoire
System.Double	Double	±5,0e-324 à ±1,7e308	8 octets
System.Decimal	Decimal	±1,0e-28 à ±7,9e28	16 octets

Les types numériques primitifs sont tous des types valeurs. Une variable de type numérique et non initialisée par le développeur aura pour valeur par défaut 0.

Remarque

.NET 4 a apporté le type `System.Numerics.BigInteger` afin de manipuler des entiers d'une taille arbitraire. Ce type est aussi un type valeur, mais il ne fait pas partie des types intégrés.

Types textuels

Il existe dans la BCL deux types permettant de manipuler des caractères Unicode et des chaînes de caractères Unicode : `System.Char` et `System.String`. Ces types ont respectivement pour alias VB.NET `Char` et `String`.

Le type `Char` est un type valeur encapsulant les mécanismes nécessaires au traitement d'un caractère Unicode. Par conséquent, une variable de type char est stockée en mémoire sur 2 octets.

Les valeurs de type `Char` doivent être encadrées par les caractères `" "` et suffixées par le caractère `c` (qui les différencie ainsi des chaînes de caractères) : `"a"c`.

Le type `String` manipule en interne des tableaux d'objets de type `Char` pour permettre le traitement de chaînes pouvant représenter jusqu'à 4 Go, soit 2147483648 caractères. Contrairement aux types intégrés vus jusqu'ici, le **type String est un type référence**, ce qui signifie donc qu'une variable de ce type peut ne pas avoir de valeur.

Une chaîne de caractères s'écrit entre les caractères `" "` : `"une chaîne de caractères sympathique et un peu longue"`.

Pour utiliser le caractère " dans une variable de type `Char` ou `String` sans causer une erreur de compilation, il est nécessaire de le doubler : `"le ""deuxième"" mot est entre guillemets"`.

D'autres caractères doivent être représentés à l'aide de constantes VB.NET particulières. Le tableau suivant résume les constantes qui peuvent être utilisées.

Constante VB.NET	Caractère associé
`vbBack`	Retour arrière
`vbCr`	Retour chariot
`vbLf`	Saut de ligne
`vbCrLf`	Concaténation d'un retour chariot et d'un saut de ligne
`vbNewLine ( = vbCrLf )`	Concaténation d'un retour chariot et d'un saut de ligne
`vbNullChar`	Caractère nul
`vbNullString`	Chaîne de caractères nulle
`vbTab`	Tabulation
`vbVerticalTab`	Tabulation verticale

Il est à noter que les chaînes de caractères sont invariables, c'est-à-dire qu'elles ne peuvent pas être modifiées. Heureusement, le framework nous permet de le faire mais à un certain prix. Chaque modification effectuée sur une chaîne de caractères entraîne la création d'une nouvelle chaîne incluant la modification. Ce comportement, transparent pour nous, peut entraîner des problèmes de performance lorsqu'il est question de traitements importants sur ce type d'objets. Il est donc important de connaître cette subtilité afin d'éviter ce type de problèmes.

Type booléen

Une valeur booléenne représente un état vrai ou faux. Le type `Boolean` est essentiel au développement car c'est lui qui va permettre d'effectuer des vérifications de conditions.