Ressourcesinformatiques

C# 6

et Visual Studio 2015

Les fondamentaux du langage

Sébastien PUTIER

Table des matières

Les éléments à télécharger sont disponibles à l'adresse suivante :
http://www.editions-eni.fr
Saisissez la référence ENI de l'ouvrage **RI15CSHA** dans la zone de recherche et validez. Cliquez sur le titre du livre puis sur le bouton de téléchargement.

Chapitre 2
Visual Studio 2015

Chapitre 3
L'organisation d'une application

Chapitre 4
Les bases du langage

Chapitre 5
La programmation orientée objet avec C#

Chapitre 6
Débogage et gestion des erreurs

Chapitre 7
Développement d'applications Windows

Chapitre 8
Accès aux données

Chapitre 9
LINQ

Chapitre 4
Les bases du langage

1. Introduction

Comme tous les langages de programmation, C# 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 C#. Avant d'explorer en profondeur les fonctionnalités du langage au chapitre suivant, nous étudierons donc les notions essentielles et fondamentales de C# : la création de données utilisables par une application et le traitement de ces données.

2. Les variables

Les données utilisables dans un programme C# 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 C# 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 _ et 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 sensible à la casse, c'est-à-dire qu'il fait la distinction entre majuscules et minuscules : les variables `unevariable` et `uneVariable` sont donc différentes.

– 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 de préfixer un mot-clé par un caractère autorisé ou par le caractère @ pour utiliser un nom de variable similaire.

Les noms de variables suivants sont acceptés par le compilateur C# :

– `maVariable`

– `maVariableNumero1`

– `@void` (`void` est un mot-clé de C#)

– `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 C# 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 en C# 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 la valeur spéciale `null`, 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 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 que contient 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 à C#.

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	Alias C#	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	int	-2147483648 à 2147483647	4 octets
System.UInt32	uint	0 à 4294967295	4 octets
System.Int64	long	-9223372036854775808 à 9223372036854775807	8 octets
System.UInt64	ulong	0 à 18446744073709551615	8 octets
System.Single	float	±1,5e-45 à ±3,4e38	4 octets

Type	Alias C#	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 `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 deux octets.

Les valeurs de type `char` doivent être encadrées par les caractères ' ' : 'a'.

Certains caractères ayant une signification particulière pour le langage doivent être utilisés avec le caractère d'échappement \ afin d'être correctement interprétés. D'autres caractères n'ayant pas de représentation graphique doivent être aussi déclarés avec des séquences spécifiques. Le tableau suivant résume les séquences qui peuvent être utilisées.

Séquence d'échappement	Caractère associé
\'	Simple quote '
\"	Double quote "
\\	Backslash \

Séquence d'échappement	Caractère associé
\a	Alerte sonore
\b	Retour arrière
\f	Saut de page
\n	Saut de ligne
\r	Retour chariot
\t	Tabulation horizontale
\v	Tabulation verticale
\0	Caractère nul
\uXXXX	Caractère Unicode dont le code hexadécimal est XXXX

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 2 147 483 648 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 avoir la valeur `null`.

Une chaîne de caractère s'écrit entre les caractères " " : `"une chaîne de caractères sympathique et un peu longue"`. Tout comme pour les `char`, certains caractères nécessitent d'utiliser une séquence d'échappement.

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.