C#

Structure d'un programme

SOMMAIRE

LANGAGE C# : STRUCTURE D'UN PROGRAMME

Objectifs

Comprendre les généralités de l'environnement .NET et savoir créer une première application C# avec Visual Studio .NET.
Connaître les règles et les conventions de syntaxe et de présentation d'un programme écrit en langage C#.

- Namespace et directive Using
- La fonction Main, point d'entrée de l'application.
- Comment déclarer des variables et des constantes, comment les utiliser dans le corps d'un programme.
- Comment commenter vos programmes.

Le namespace

Même si les concepts objets ne seront traités que dans une phase ultérieure de l'apprentissage C#, il est nécessaire d'introduire ici le concept de classe ainsi que le mot clé du langage C# qui permet de l'implémenter.
Ainsi, un programme C# est d'abord constitué par un **namespace** (espace de nom en français). Un namespace est un ensemble d'éléments, qui se déclare par le mot clé **`namespace`** de la façon suivante

```
namespace TPCSharp
{
    //Déclaration des classes et fonctions de l'application
}
```

Le nom du namespace (ici **TPCSharp**) est choisi par le programmeur. La construction d'un nom de namespace obéit à la règle standard régissant tous les identificateurs du langage C# (voir ci-dessous).
Ce namespace est bien un simple regroupement de classes avec leurs méthodes (fonctions) ayant un lien logique ou fonctionnel entre elles. Par exemple, le namespace prédéfini **`System`** de C# regroupera l'ensemble des classes ayant trait au système lui-même.

Selon le standard C# (ECMA-334), il est fortement déconseillé de mettre le même nom à un namespace et à une classe.

Les noms de classe et de namespace commencent par une **Majuscule :** voir notation **PascalCase**

(http://msdn.microsoft.com/en-us/library/x2dbyw72(v=vs.71).aspx)
.
http://fr.wikipedia.org/wiki/CamelCase

Structure d'un programme

La directive USING

Le langage C# est un langage peu implémenté. C'est-à-dire qu'il n'existe que très peu de mots réservés, une cinquantaine, dont la liste est publiée dans la documentation du compilateur.

Par contre, il existe plusieurs bibliothèques de classes ou namespaces (qui correspondent aux packages de Java) qui offrent au programmeur un grand nombre de fonctionnalités. Ces namespaces sont utilisés grâce à l'instruction **`using`**.

Pour pouvoir utiliser les fonctions ou les objets des namespaces, il faut que celles-ci soient reconnues par le compilateur (qui ne les connaît pas par défaut). C'est le rôle de la directive **`using`**. C'est pourquoi on commence très souvent (voire toujours) un programme C# par la directive suivante :

```
using System;
```

Encore une fois, ceci n'est pas obligatoire, mais facilite grandement la tâche du développeur.

En l'absence de directive **`using`** et à chaque fois que l'on souhaite utiliser une classe ou une fonctionnalité d'un namespace, il faudrait préfixer cette classe ou cette fonctionnalité par le nom du namespace où elle est définie

Par exemple : **`System.`**`Console`**`.ReadLine()`** en l'absence de la directive **`using System`**.

En utilisant la directive **`using`**, le préfixe n'est plus obligatoire. Le compilateur ira chercher les fonctionnalités dans les différents namespaces des directives **`using`**. Dans notre exemple, en mettant **`using System`**, on pourra utiliser directement et simplement `Console`**`.ReadLine()`** dans l'ensemble du programme.

Structure d'un programme

Le programme principal

Une application C# possède toujours au moins une classe dite « classe application » et une fonction **`Main.`**

La classe application est simplement la dénomination donnée à la classe contenant la fonction **`Main`**.

La fonction **`Main`** est obligatoire et constitue le point d'entrée du programme principal de l'application. Elle se déclare dans la classe de l'application de la façon suivante :

```
namespace TPCsharp
{
  class Program
  {
    static void Main(string[] args)
     {
        // Instructions
     }
  }
}
```

La fonction **`Main`** est le point d'entrée du programme. Elle est donc obligatoire pour toute application C# et doit être définie de façon unique (il ne peut pas y avoir deux fonctions **`Main`** dans un programme). Elle doit s'appeler obligatoirement **`Main`** et doit être suivie par des parenthèses ouvrante et fermante encadrant les arguments (comme toutes les fonctions en C#).

Par défaut, la fonction **`Main`** reçoit du système d'exploitation un seul argument (**`args`**) qui est un tableau de chaînes de caractères, chacune d'elles correspondant à un paramètre de la ligne de commande (unix ou MS-DOS).

Le corps du programme suit la fonction **`Main`** entre deux accolades ouvrante et fermante. Les instructions constituant le programme sont à insérer entre ces deux accolades.

Les mots clés **`static`** et **`void`** sont obligatoires pour la fonction **`Main`**. Ces notions seront plus faciles à comprendre lors de la partie objet du cours (concepts objets proprement dits), mais on peut simplement préciser que :

- **`static`** : permet au système d'exploitation de ne pas avoir besoin de créer une instance (un objet) de la classe définie ;
- **`void`** : ce mot clé sera également plus longuement explicité dans la suite du cours. Pour résumer, il s'agit de la déclaration du type de retour de la fonction, à savoir : aucun retour pour le type **`void`**.

Structure d'un programme

Les variables

Les données en mémoire sont stockées dans des variables. Il peut par exemple s'agir de données saisies par l'utilisateur ou de résultats obtenus par le programme, intermédiaires ou définitifs.

Pour employer une image, une variable est une **boîte**, que le programme (l'ordinateur) va repérer par une **étiquette**. Pour avoir accès au contenu de la boîte, il suffit de la désigner par son étiquette

Comme tous les autres identificateurs (identificateurs de classes, entre autres) en langage C#, les noms de variables peuvent être choisis librement par le programmeur (sauf parmi les *mots-clés*). La liste des mots clés se trouve dans la documentation en ligne, mais dans un problème, une variable est choisie pour jouer un rôle. Il est souhaitable que l'auteur de l'algorithme s'en souvîenne en choisissant des noms évocateurs et en respectant, autant que possible, l'unicité du rôle pour chaque variable.

Les caractères suivants peuvent être utilisés :

- de **A** à **Z**
- de **a** à **z** (les minuscules sont considérées comme des caractères différents des majuscules)
- de **0** à **9**
- le caractère souligné **_** et le caractère @ .

Exemple : nom des variables à retenir dans un problème de calcul de moyenne de notes

totalNotes	: total des notes d'un(e) élève
nombreNotes	: nombre des notes de cet(te) élève
moyenneNotes	: moyenne des notes de cet(te) élève

Remarquez le style de nommage des variables : (convention CamelCase)

Une variable est typée: elle peut représenter un nombre (entier ou réel), une chaîne de caractères, etc.

Une variable peut être déclarée à tout moment à l'intérieur du corps du programme. Cependant, la déclaration et l'initialisation d'une variable doivent impérativement précéder la première utilisation de celle-ci. Il est également et généralement préférable de déclarer les variables en début de bloc (juste après l'accolade ouvrante) pour une question de lisibilité

Chaque déclaration de variable est construite sur le modèle suivant :

```
type      nomDeLaVariable ;
```

Une variable peut être initialisée lors de sa déclaration :

```
type    nomDeLaVariable = valeurInitiale;
```

Les variables ne sont visibles (reconnues par le compilateur) que dans le bloc d'instructions (défini par **{** et **}**) dans lequel elles sont définies.

Structure d’un programme

Comme toutes les instructions du langage C#, chaque déclaration de variable DOIT absolument être terminée par un point-virgule **`;`** Le point-virgule ne constitue pas un séparateur, mais plutôt un *terminateur* d'instructions.

Le chapitre suivant vous donnera plus de détails sur l'ensemble des types de données C# et .NET.

Les constantes

Une constante est une variable dont la valeur ne peut changer. Son rôle est de noter des repères, des dimensions, des références invariants au cours d'un programme.

Les constantes sont définies par un identificateur, et par une valeur. Le nom représente la manière de faire référence à la valeur. La valeur représente le contenu de notre constante. Cette valeur est **invariante**.

Les constantes se déclarent comme les variables initialisées précédées du mot-clef **const**. Leur valeur ne pourra pas être modifiée pendant l'exécution du programme.

```
const Type nomDeLaConstante = valeurInitiale;
```

Structure d'un programme

Les instructions

Une instruction est une ligne de traitement

Le caractère **;** est un *terminateur*. TOUTES les instructions doivent se terminer par un **;**.

Les différents identificateurs sont séparés par un *séparateur*.

Les *séparateurs* peuvent être indifféremment l'*espace*, la *tabulation* et le *saut de ligne*.

Les commentaires

Les caractères compris entre **/*** et ***/** ne seront pas interprétés par le compilateur.

Les caractères compris entre **//** et un *saut de ligne* ne seront pas interprétés par le compilateur.

Les caractères compris entre **///** et un *saut de ligne* ne seront pas interprétés par le compilateur. Le texte de commentaire peut être utilisé pour générer une documentation de façon automatique.

Les conventions d'écriture

Ces conventions ne sont pas des contraintes de syntaxe du langage C#. Elles n'existent que pour en faciliter la lecture et font partie d'une sorte de norme implicite que tous les bons développeurs s'obligent à respecter.

Une seule instruction par ligne. Même si tout un programme en langage C# peut être écrit sur une seule ligne.

Les délimiteurs d'un bloc **{** et **}** doivent se trouver sur des lignes différentes et être alignés sur la première colonne de sa déclaration.

A l'intérieur d'un bloc **{ }** les instructions sont indentées (décalées) par un caractère *tabulation*.

A l'intérieur d'un bloc **{ }** la partie *déclaration des variables* et la partie *instructions* sont séparées par une ligne vide.

Les entrées sorties standard

Les fonctions de lecture et d'écriture standard (clavier/écran) sont définies dans le namespace **`System`**.

La classe **`Console`** est définie dans ce namespace et dispose, notamment de trois fonctionnalités :

- **`Console.ReadLine()`** : qui retourne **une chaîne de caractères** (de type **`string`**) saisie au clavier jusqu'à la frappe de la touche "Entrée";
- **`Console.Write()`** : qui affiche sur l'écran a priori la chaîne de caractères (de type **`string`**) envoyée en paramètre ;
- **`Console.WriteLine()`** : qui réalise la même opération en ajoutant un caractère retour-chariot après l'affichage de la chaîne de caractères (de type **`string`**) envoyée en paramètre.

Structure d'un programme

Un exemple de code

Le code ci-dessous calcule et restitue la valeur de la circonférence d'un cercle à partir de la saisie à la console de son rayon.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace TPCSharp
{
  class Cercle
  {
  static void Main()❶
    {
      //Declaration de variables.❷
      string saisie;     // variable recevant la saisie
      double rayon ;     // rayon
      double perimetre;  // périmètre

      // Etape 1 : lecture du rayon
      Console.WriteLine("Entrez la valeur du rayon : ");❸
      saisie = Console.ReadLine();❹
      // Etape 2 : calcul et affichage du périmètre
      rayon = Convert.ToDouble(saisie);❺
      //Calcul du perimetre
      perimetre = 2 * Math.PI * rayon; ❻
      Console.Write("Le cercle de rayon " + rayon);❼
      Console.Write("a pour périmetre : " + perimetre);❽
      Console.WriteLine();
      // Permet de conserver l'affichage de la console
      Console.ReadLine(); ❾
    }
  }
}
```

Structure d’un programme

Le programme décrypté, instruction après instruction…

❶ La fonction Main, point d’entrée du programme

❷ La variable **saisie** est déclarée de type **chaîne de caractères**, les variables **rayon** et **perimetre** de type **réel**

❸ Affichage à la console du texte « Entrez la valeur du rayon »

❹ Dans la variable de nom saisie, sera stockée la chaine de caracteres entrée par l’utilisateur

❺ Les variables faisant partie d’une opération (+, -,*,/) doivent être des nombres, entier ou rééls. Cette instruction a pour but de **convertir le contenu de la variable saisie de type chaîne en un nombre réel de type double**.

❻ La formule magique …. La valeur de PI est obtenue à partir d’une bibliothèque du framework, la classe Maths, dont le rôle est de fournir des constantes et des fonctions mathématiques et trigononométriques.

❼ Affichage à la console du texte « Le cercle de rayon », suivi de la valeur de la variable **rayon :** notez l’opérateur **+** qui est ici l’opérateur de concaténation.

❽ Affichage à la console du texte « a pour périmetre :», suivi de la valeur de la variable **perimetre** .

❾ Permet de ne pas terminer le programme pour visualiser l’affichage.

Structure d’un programme

Exercices

1) Saisissez le code fourni et exécutez le.

2) Rajouter une fonctionnalité de calcul et d’affichage de la surface du cercle.

3) Afficher la fonctionnalité et la version du programme, dès son démarrage :
**** Périmètre et surface du Cercle (V1.0, XX/XX/XX) ****
On conservera cette saine habitude dans tous les développements suivants, pour éviter de mettre au point avec le code d’un autre programme, ou avec le code du mois précédent !

4) Aérer la présentation du résultat de l’exécution du programme, en insérant des retours à la ligne avec *Console.Writeline()*