Cégep de Saint-Hyacinthe
Département d'informatique

Programmation orientée objet

420-2DP-HY

(3-3-3)

Les méthodes #2

(Version 1.0)

3 heures

Préparé par

Martin Lalancette

Comprendre les éléments suivants :

- La signature d'un membre
- Surcharge des méthodes
- Définir une méthode avec paramètre(s) variables
- Définir une méthode avec paramètre(s) par référence

Table des matières

Introduction

Cette séquence a pour but de vous initier aux notions de base nécessaires à l'introduction de la programmation. Nous commencerons par énoncer les éléments théoriques appuyés d'exemples simples et faciles à reproduire. Afin d'axer l'attention sur la compréhension de ces notions, il y aura des exercices à faire tout au long de cette séquence. **La prochaine séquence portera sur la définition et l'utilisation de méthodes #2.** Voici une suggestion d'arborescence :

Exercice 1. : S'assurer d'avoir créé l'arborescence ici haut mentionnée sur votre C ou votre clé USB. Créer la solution **Les méthodes #2**. Elle sera utilisée dans les prochains exercices.

La signature d'un membre

Pour permettre à C#, de déterminer quel membre exécuter, il doit se fier à sa signature. Donc chaque membre doit être conçu avec sa propre signature. Qu'est-ce que la signature d'une méthode? Elle est l'assemblage de :

- L'**identificateur** de la méthode (son nom)
- La **séquence des types** de la liste des paramètres

Ce qui est exclu de la signature :

- Modificateur d'accès (public, private, …)
- Les identificateurs de paramètres (c.-à-d. leurs noms)
- Le type de retour

```
public int Multiplier( int iValeur1, int iValeur2 )
{
   return iValeur1 * iValeur2;
}
```

La signature est : Multiplier(int, int)

La surcharge d'un membre

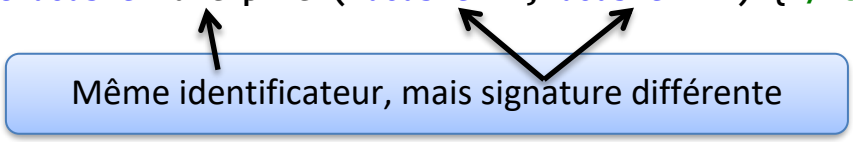
Lorsque deux ou plusieurs membres d'un type représentent le même type de membre (méthode, constructeur, etc.), qu'ils possèdent le même nom, mais des listes de paramètres différentes, le membre est dit **surchargé**.

La surcharge d'une méthode

La surcharge d'une méthode est la possibilité de déclarer **plusieurs méthodes avec le même identificateur**. Cependant, chaque signature doit demeurer unique.

Exemple :

```
public int Multiplier( int f1, int f2 ) { /*code...*/ }
public float Multiplier( float f1, float f2 ) { /*code...*/ }
public double Multiplier( double f1, double f2 ) { /*code...*/ }
```



Dans le code, lorsqu'une méthode est appelée, le compilateur accomplit les étapes suivantes :

1. Il **détermine sa signature** en tenant compte de son **identificateur** et en analysant le **type de chaque paramètre** à l'appel.
2. Il vérifie si cette signature existe, **sinon erreur : le nom "Id?" n'existe pas dans le contexte actuel**.
3. Si l'appel vient de l'extérieur, il vérifie si les droits d'accès sont suffisants (public, private, …), **sinon erreur : "Id?" est inaccessible en raison de son niveau de protection**.

Exercice 2. : Écrire et programmer deux fonctions nommées **TirerDé** selon les paramètres suivants :

- **TirerDé**() : Par défaut, représente un dé à 6 faces. Elle doit retourner une valeur entre 1 et 6.
- **TirerDé**(*nombre de faces*) : Elle doit retourner une valeur entre 1 et le nombre de faces.

Définir une méthode avec des paramètres variables

Jusqu'à maintenant nous avons développé ou utilisé des méthodes qui déclarent 0, 1 ou plusieurs paramètres. Il existe un mot clé qui permet d'indiquer à une méthode d'accepter un nombre variable de paramètres lors de l'appel. Ce mot clé est **params** qui permet à 1 paramètre de recevoir **entre 0 à N valeur de type attendu**. Règles pour **params** :

- **params** est utilisé **dans la déclaration** des paramètres d'une méthode (entre les parenthèses).
- **params** est utilisé **une fois**, avec le **dernier paramètre**, avant **le type**, et le type **doit être un tableau**.
- À l'appel, un paramètre qualifié avec **params** accepte **une liste d'éléments** ou **un tableau d'éléments** tous du même type (celui du paramètre).

Exemple #1 : Additionner plusieurs nombres à partir d'un tableau

Code C# :

```
private int Additionner(params int[] valeurs)
{
    int iSomme = 0;

    // Préparer et effectuer l'opération.
    for (int iIndex = 0; iIndex < valeurs.Length; iIndex++)
        iSomme += valeurs[iIndex];

    // Ou bien utiliser un foreach comme suit:
    foreach (int valeur in valeurs)
        iSomme += valeur;

    // Retourner le résultat.
    return iSomme;
}
…

// Exemples d'appel.
int iResultat = 0;
iResultat = Additionner(3);
iResultat = Additionner(4, 6);
iResultat = Additionner(4, 8, 12, 34, 2, 56);
iResultat = Additionner(); // Ici va retourner 0;

int[] entiers = new int[] { 30, 40, 50 };
iResultat = Additionner(entiers);
iResultat = Additionner(new int[] { 50, 60, 90, 100 });
```

Choisir un ou l'autre des deux algorithmes.

Exercice 3. Écrire et programmer une fonction nommée **CalculerMoyenne** qui doit faire la moyenne des nombres passés en paramètres variable. Cependant, la méthode doit au minimum accepter deux valeurs.

Exercice 4. : Écrire et programmer une fonction nommée **Concatener** qui a comme **premier paramètre** une variable indiquant un **séparateur (caractère)**. Par la suite, les autres paramètres sont variables et de type string. Cette méthode doit prendre toutes les chaines de caractères et les coller ensemble en utilisant le premier paramètre comme séparateur.

Un petit rappel concernant les paramètres par valeur

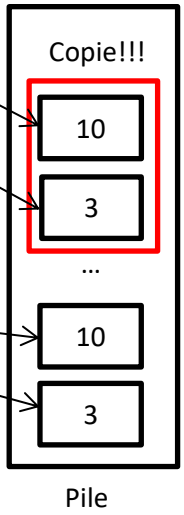
Un appel à une méthode avec des paramètres définis par valeur **consiste à passer une copie des valeurs** aux paramètres utilisés.

Exemple #1: Paramètres de type valeur

```
// iNb1 et iNb2 reçoivent une copie des valeurs.
static public int Calculer( int iNb1, int iNb2 )
{
       iNb1 += 10;
       iNb2 *= iNb1;
       return iNb2;
}

static void Main(string[] args)
{
    int iValeur1 = 3;
    int iValeur2 = 10;

    // Au retour de l'appel à Calculer(), les valeurs
    // que possédaient iValeur1 et iValeur2 sont inchangées.
    int iResultat = Calculer(iValeur1, iValeur2);
}
```



Définir une méthode avec des paramètres par référence

Nous avons vu jusqu'à maintenant l'utilisation des paramètres par valeur qui consiste à passer des valeurs à une méthode qui va s'en servir pour effectuer différentes opérations ou des calculs. Le contenu de ces paramètres ne change pas au retour de la méthode. Il existe un type de paramètre, ap-

pelé « référence », qui permet à une méthode d'en modifier son contenu et le retourner à la fin de l'exécution de celle-ci.

Le type de retour est excessivement pratique, car il permet de récupérer le résultat provenant d'une méthode. Il peut être intéressant, voire nécessaire, de récupérer plus d'un résultat, mais il n'y a qu'un seul type de retour. Le passage de paramètres par référence vient combler ce besoin particulier et bien plus.

Le **passage de paramètres par référence** fournit l'argument même. Autrement dit, on passe le contrôle complet de l'argument (variable) et non une copie de la valeur qu'il possède.

Par défaut, un paramètre est passé par valeur.

Il existe en C# deux mots clés possibles qui peuvent être ajoutés à la déclaration du paramètre, en voici les détails.

Mot clé « ref »

« Le mot clé **ref** fait en sorte que les arguments soient passés par référence. La conséquence est **que toute modification apportée au paramètre dans la méthode est reflétée dans cette variable lorsque la méthode appelante récupère le contrôle**. Pour utiliser un paramètre ref, la définition de méthode et la méthode d'appel doivent toutes deux utiliser le mot clé ref explicitement. » (MSDN, n.d.). **La variable passée en paramètre avec ref doit absolument être initialisée**.

Règles pour l'utilisation de « ref » :

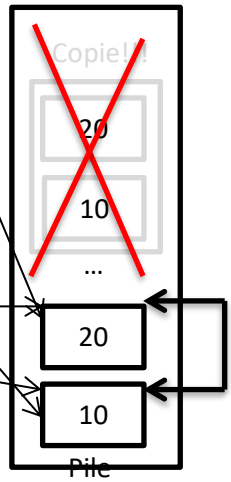
- **ref est un mot clé** utilisé **à la déclaration** d'un paramètre de la méthode, **mais aussi à l'appel**.
- **ref** est **associé à un 1 paramètre à la fois**.
- **ref** est spécifié **avant le type** du paramètre à la déclaration, et **avant la variable** à l'appel.

- À l'appel, **la variable doit être initialisée** (assignée), que son type soit valeur ou référence.

Exemple #1 : Passage **par référence** de **types valeurs**

```
// iNb1 et iNb2 reçoivent le contrôle des 2 arguments
// de l'appelant.
static public void Initialiser( ref int iNb1, ref int iNb2 )
{
        iNb1 = 0;
        iNb2 = 0;
}
static void Main(string[] args)
{
        int x = 10;
        int y = 20;

        // À l'appel x = 10 et y = 20
        Initialiser( ref x, ref y );
        // Au retour x = 0 et y = 0
}
```



- Le mot clé **ref** **fait partie de la signature pour chaque paramètre** avec lequel il est déclaré. Exemple :

```
public int DiviserReste(ref int iReste, int iDividende, int iDiviseur)
{
    iReste = iDividende % iDiviseur;
    return iDividende / iDiviseur;
}
```

La **signature** est DiviserReste(ref int, int, int)

Exercice 5. : Ajouter une méthode nommée « **SéparerTéléphone** » qui permet de prendre comme premier paramètre un **numéro de téléphone** selon le format 450-234-1234. Elle aura comme fonction de séparer les éléments composant ce numéro en trois : **l'indicatif régional**, le **préfix** et le **suffixe**. Retourner ces trois informations via des paramètres par référence. Faire un exemple d'appel dans la méthode principale.

Exercice 6. : Écrire et programmer une méthode nommée **Permuter** qui prend en paramètre deux variables entières et qui interchange leurs valeurs. Exemple : V1=3 et V2=5 deviennent V1=5 et V2=3 après l'appel de cette méthode.

Écrire et programmer la même méthode, mais qui prend 2 variables de type chaine de caractères.

Mot clé « out »

La fonctionnalité est semblable au mot clé **ref**, mais **out NE nécessite PAS que la variable soit initialisée avant d'être passée en paramètre**.

Règles pour l'utilisation de « out » :

- **out** est un mot clé dont l'effet est complémentaire à ce que fait **ref**. Les règles sont similaires.
- À l'appel, la variable **peut être initialisée ou non** (assignée ou non), que son type soit valeur ou référence.
- Cependant, la variable doit **absolument avoir été initialisée ou réinitialisée** avant de retourner le contrôle à l'appelant.
- La signature tient compte du mot clé **out**.
- Essentiellement, le mot clé **out** a été conçu **pour ne pas contraindre l'appelant à initialiser** la variable qui, de toute évidence, doit être affectée ou réaffectée avec une nouvelle valeur. Sinon, il serait plus logique d'utiliser **ref** que **out**.

Attention : les mots clés **ref** et **out** sont considérés comme **étant équivalents au niveau de la signature d'une méthode**. Autrement dit, une signature tient compte de ces 2 mots clés, mais ne s'en sert pas pour les différencier. Exemple :

```
public int DiviserReste(ref int iReste, int iDividende, int iDiviseur) {/*...*/}
public int DiviserReste(out int iReste, int iDividende, int iDiviseur) {/*...*/}
```

Pour le compilateur, ces 2 méthodes possèdent une signature identique et une erreur compilation sera provoquée.

En conclusion, le passage de paramètres par référence vient appuyer les limitations d'un seul type de retour. On peut conclure ainsi cette technique est intéressante, même si elle n'est pas fréquemment utilisée.

Exercice 7. : Ajouter une méthode appelée « **SéparerChemin** » qui va prendre en premier paramètre le chemin d'un fichier (ex. : C:\Temp\texte.txt) et en extraire les éléments suivants : **lecteur**, **répertoire** et **nom du fichier**. **Ici les variables passées en paramètre ne sont pas initialisées**. Faire la déclaration et faire un exemple d'appel dans le programme principal.

Exercice 8. : Écrire et programmer une méthode nommée **DemanderValeur** ayant comme premier paramètre une chaine de caractères qui servira à afficher la question à l'usager et comme deuxième paramètre une variable de type entière qui servira à recevoir la réponse de l'usager. Cette fonction doit valider la réponse et quitter seulement lorsque la valeur est de type entier et respecte ses limites. Lors d'une erreur, afficher un message et reposer la question.

Faire une surcharge de cette méthode pour retourner une réponse de type Réel et une autre de type caractère.

Faire des exemples d'appel dans le programme principal.

Bibliographie

MSDN. (s.d.). *ref (référence C#)*. Consulté le 11 26, 2012, sur Visual Studio - MSDN: http://msdn.microsoft.com/fr-fr/library/vstudio/14akc2c7(v=vs.110).aspx