Module 5:
Méthodes et paramètres

Vue d'ensemble

- Utilisation de méthodes
- Utilisation de paramètres
- Utilisation de méthodes surchargées

Utilisation de méthodes

- Qu’est-ce qu’une méthode
- Appel de méthodes
- Utilisation de l'instruction de “return”
- Utilisation des variables locales
- Retour de valeurs

Qu'est-ce qu'une méthode

- Main est une méthode
  - Utilisez la même syntaxe pour définir vos propres méthodes

```
using System;

class ExampleClass
{
    static void ExampleMethod( )
    {
        Console.WriteLine("Example method");
    }
    static void Main( )
    {
        // ...
    }
}
```

Appel de méthodes

- Après avoir défini une méthode, vous pouvez:
  - Appeler une méthode à partir de la même classe
    - Utilisez le nom de la méthode suivie par une liste de paramètres entre parentheses
  - Appeler une méthode qui est dans une classe différente
    - Vous devez indiquer au compilateur quelle classe contient la méthode à appeler
    - La méthode à appeler doit être déclarée avec le mot-clé **public**
  - Appels imbriqués
    - Des méthodes peuvent appeler des méthodes, qui peuvent appeler d'autres méthodes, et ainsi de suite…

Utilisation de l'instruction de “return”

- Retour immédiat si une condition est vraie

```
static void ExampleMethod( )
{
    int numBeans;
    //...

    Console.WriteLine("Hello");
    if (numBeans < 10)
        return;
    Console.WriteLine("World");
}
```

Utilisation des variables locales

- Les variables locales
  - Créée lorsque la méthode commence
  - Privée à la méthode
  - Détruite à la sortie
- Les variables partagées
  - Les variables de classe sont utilisés pour le partage
- Les conflits de portée
  - Le compilateur n'affichera pas d'erreurs si les nom des variables locales et de classes sont identiques. **Les variables locales seront utilisées prioritairement.**

Retour de valeurs

- Déclarer la méthode avec comme type de retour, autre chose que “void”
- Ajoutez une expression avec une instruction de retour pour:
  - Définir la valeur de retour
  - Retourner à l'appellant

- Les méthodes “non-void” doivent **absolument renvoyer une valeur :**

```
static int TwoPlusTwo( ) {
    int a,b;
    a = 2;
    b = 2;
    return a + b;
}
```

```
int x;
x = TwoPlusTwo( );
Console.WriteLine(x);
```

Utilisation de paramètres

- Déclaration et utilisation de paramètres
- Mécanismes pour passer des paramètres
- Passer par valeur
- Passer par référence
- Paramètres de sortie
- Utilisation des listes de paramètres de longueur variable
- Guidelines pour passer des paramètres
- Utilisation de méthodes récursives

Déclaration et utilisation de paramètres

- Déclaration des paramètres
  - Placer entre parenthèses après le nom de la méthode
  - Définir le type et le nom de chaque parameter
  - Séparer les parametres par une “,”
- Appel de méthodes avec des paramètres
  - Fournir une valeur de même type pour chaque paramètre

```
static void MethodWithParameters(int n, string y)
{ ... }

MethodWithParameters(2, "Hello, world");
```

Mécanismes pour passer des paramètres

- Il y a 3 façons de passer des paramètres

in	Passer par valeur
In out	Passer par référence
out	Les paramètres de sortie

Passez par valeur

- Mécanisme par défaut pour passer des paramètres:
  - La valeur du paramètre est copié
  - Variable peut être modifiée à l'intérieur de la méthode
  - N'a pas d'effet sur la valeur en dehors de la méthode
  - Le paramètre doit être du même type

```
static void AddOne(int x)
{
        x++; // Increment x
}
static void Main( )
{
        int k = 6;
        AddOne(k);
        Console.WriteLine(k); // Display the value 6, not 7
}
```

Passez par référence

- Que sont les paramètres de référence?
  - Une référence à l'emplacement de mémoire
- Utilisation des paramètres de référence
  - Utilisez le mot-clé **ref** dans la déclaration de méthode et appel
  - types Match et les valeurs des variables
  - Les modifications apportées à la méthode affectent l'appelant
  - Affectez la valeur du paramètre avant d'appeler la méthode

```
static void increment(ref int i)
{
    i++;
}
static void Main( )
{
    int val = 0;
    increment(ref val)
    // the value of val is 1 !
}
```

Paramètres de sortie

- Que sont les paramètres de sortie?
  - Valeurs sont définies dans la méthode
- Utilisation des paramètres de sortie
  - Comme **ref**, mais les valeurs ne sont pas transmise à la méthode
  - Utilisez **dehors** mot-clé dans la déclaration de méthode et appel

```
static void OutDemo(out int p)
{
        // ...
}
int n = 1;
OutDemo(out n);
```

Utilisation des listes de paramètres de longueur variable

- Utiliser le mot clé params
- Déclarer un tableau à la fin de la liste de paramètres
- Toujours passer par valeur

```
static long AddList(params long[ ] v)
{
    long total, i;
    for (i = 0, total = 0; i < v.Length; i++)
        total += v[i];
    return total;
}
static void Main( )
{
    long x = AddList(63,21,84);
}
```

Guidelines pour passer des paramètres

- Mécanismes
  - Passer par valeur est la plus courante
  - La valeur de retour de la méthode est utile pour les valeurs simples
  - Utilisez **ref** et/ou **out** pour plusieurs valeurs de retour
  - Utilisez uniquement **ref** si les données sont transférées dans les deux sens
- Efficacité
  - Le passage par valeur est généralement le plus efficace

Utilisation de méthodes récursives

- Une méthode peut s'appeler
  - Directement
  - Indirectement
- Utile pour résoudre certains problèmes

Utilisation de méthodes surchargées

- Déclaration de méthodes surchargées
- Les signatures de méthode
- Utilisation de méthodes surchargées

Déclaration de méthodes surchargées

- Méthodes qui partagent un nom dans une classe
  - Distingué par l'examen des listes de paramètres

```
class OverloadingExample
{
    static int Add(int a, int b)
    {
        return a + b;
    }
    static int Add(int a, int b, int c)
    {
        return a + b + c;
    }
    static void Main( )
    {
        Console.WriteLine(Add(1,2) + Add(1,2,3));
    }
}
```

Les signatures de méthode

- Les signatures de méthode doivent être uniques au sein d'une classe
- Signature de la méthode :

Signature

- Nom de la méthode
- Type de paramètre
- Modificateur de paramètre (ref ou out)

Pas d'effet sur Signature

- Nom du paramètre
- type de méthode de retour

Utilisation de méthodes surchargées

- Pensez à utiliser les méthodes surchargées lorsque:
  - Vous avez des méthodes similaires qui nécessitent des paramètres différents
  - Vous souhaitez ajouter de nouvelles fonctionnalités au code existant
- N‘en abusez pas parce que c’est:
  - Difficile à déboguer
  - Difficile à maintenir

Lab 5.1: Création et utilisation de méthodes

technocité
creative industries knowledge center

WAVENET
Intelligent IT Solutions