Cégep de Saint-Hyacinthe
Département d'informatique

Programmation orientée objet

420-2DP-HY

(3-3-3)

Notions sur les objets - Classes statiques et méthodes d'extension

(Version 1.3)

3 heures

Préparé par

Martin Lalancette

Comprendre les éléments suivants :

- Membres statiques
- Classes statiques
- Méthodes d'extension

Table des matières

Introduction

Cette séquence a pour but de vous initier aux nécessaires à la programmation à base d'objets et d'événements. **Cette séquence traitera des notions reliées aux objets axées principalement sur les classes et méthodes statiques ainsi que les méthodes d'extension.**Pour bien suivre les instructions qui vont être mentionnées tout au long des séquences d'apprentissage, une préparation de base s'impose. Voici une suggestion d'arborescence :

Exercice 1. : S'assurer d'avoir créé l'arborescence ici haut mentionnée sur votre C ou votre clé USB. Récupérer une **solution de départ** disponible dans LÉA.

Les membres statiques

Les membres statiques sont des membres (propriétés, champs, méthodes) qui sont **accessibles sans devoir instancier un objet** de ladite classe. Donc toutes les instances associées (c.-à-d. les variables) à une classe auront en commun les membres statiques qui seront accessibles uniquement via le nom de la classe (et non via la variable associée à une instance). Le mot-clé à utiliser est *static*. À savoir :

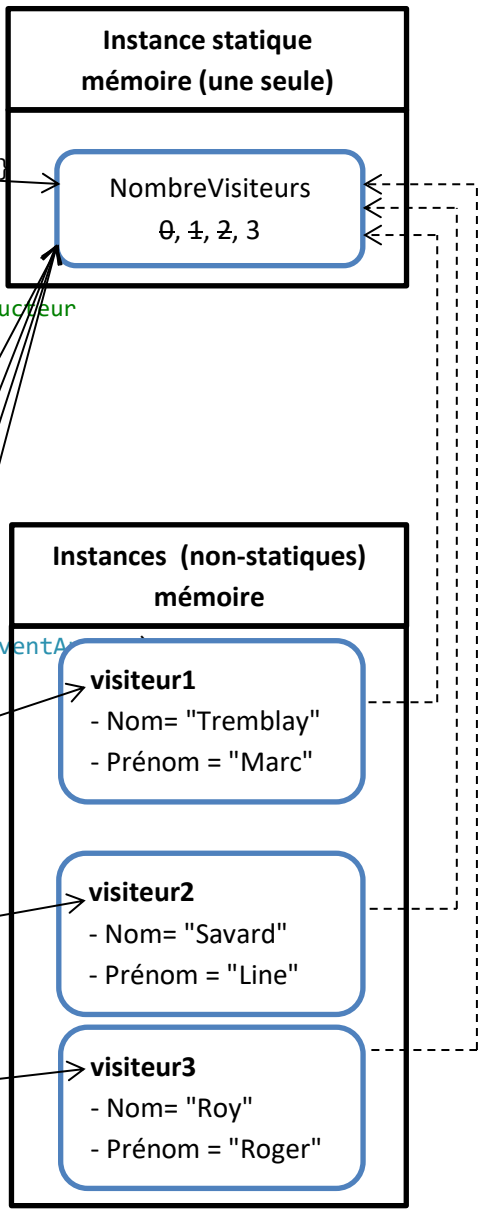
- Une méthode statique **ne peut utiliser que des attributs statiques**, et **ne peut appeler à l'interne que des méthodes également déclarées comme statiques**.
- Une méthode statique pourra s'exécuter dès que la classe qui la contient est chargée en mémoire, y compris en l'absence de toute création d'objet.
- Les membres déclarés **sans le modificateur *static*** sont appelés : membres d'instances, ou membres non statiques.
- Les membres déclarés **avec le modificateur *static*** sont appelés : membres statiques, ou membres de classe.

- Le modificateur *static* est combiné au modificateur d'accès comme *public* ou *private*. Chacun joue un rôle très différent. L'ordre dans la déclaration des modificateurs n'a pas d'importance.

Toutes les **instances non statiques sont indépendantes,** car elles possèdent une **copie de leurs propres membres** alors que l'**instance statique est commune à toutes les instances**. Il n'existe **qu'une copie de chaque membre statique d'une classe**.

Exemple #1 : Je souhaite connaître le nombre de visiteurs qui sont instanciés dans mon programme.

```
public class Visiteur
{
    // Propriétés
    static public int NombreVisiteurs { get; private set; }

    public string Nom { get; set; }
    public string Prénom { get; set; }

    public Visiteur(string sNom, string sPrénom) // Constructeur
    {
        NombreVisiteurs++;
        Nom = sNom;
        Prénom = sPrénom;
    }

    ~Visiteur()  // Destructeur
    {
        NombreVisiteurs--;
    }
}
private void btnExercice01_Click(object sender, RoutedEventA
{

    MessageBox.Show("Nombre de visiteurs: " +
                    Visiteur.NombreVisiteurs);

    // Instancier le visiteur #1.
    Visiteur visiteur1 = new Visiteur("Tremblay", "Marc");
    MessageBox.Show("Nombre de visiteurs: " +
                      Visiteur.NombreVisiteurs);

    // Instancier le visiteur #2.
    Visiteur visiteur2 = new Visiteur("Savard", "Line");
    MessageBox.Show("Nombre de visiteurs: " +
                    Visiteur.NombreVisiteurs);

    // Instancier le visiteur #3.
    Visiteur visiteur3 = new Visiteur("Roy", "Roger");
    MessageBox.Show("Nombre de visiteurs: " +
                    Visiteur.NombreVisiteurs);
}
```



Exercice 2. : Ajouter la classe **Visiteur** et ajouter le code de l'exemple précédent. Dans un bouton, ajouter le code de l'événement Click précédent.

Exercice 3. : Dans le même projet, ajouter **une liste** globale dans Visiteur qui va contenir tous les commentaires des visiteurs. Cette liste est indépendante de l'instanciation d'un objet Visiteur. Par la suite, ajouter la méthode **AjouterCommentaire** qui prend le commentaire en paramètre d'un visiteur et qui va l'ajouter à la liste. Voici des exemples de commentaires :

- **Visiteur #1** : « Super! »
- **Visiteur #2** : « Je vais y revenir plusieurs fois »
- **Visiteur #3** : « Trop de monde! »

Afficher les commentaires dans txtRésultat à partir de cette liste globale.

Dans le framework .NET, il existe beaucoup de classes qui comportent des membres statiques. Voici un exemple avec la classe **String**. Vous pouvez concaténer deux chaines de caractères sans instancier un objet de type **String**. Exemple : `string sNomComplet = string.Concat("Marc", "Tremblay");`

Les classes statiques

Nous pouvons non seulement appliquer le mot-clé *static* à des membres d'une classe mais à une classe elle-même. Lorsqu'une classe est déclarée avec le modificateur *static*, **tous ses membres doivent être déclarés avec le modificateur *static* également**. Lorsqu'une classe est déclarée statique, elle **n'est pas instanciable du tout**. Ce sont des classes dites « utilitaires ».

À savoir :

- L'instance statique d'une classe est une instance spéciale qui n'existe **qu'en une, et une seule copie**.

- Chaque classe peut posséder au maximum une instance statique d'elle-même!
- **L'instance statique** d'une classe est **créée automatiquement** par C# lors de l'exécution. Les **instances (tout court)** sont **créées manuellement avec l'opérateur *new***.
- Le moment particulier est celui où l'exécution tente d'accéder à un membre statique (lazy instanciation). L'opérateur new ne peut être utilisé pour créer l'instance statique manuellement.
- Le nom de la classe est utilisé comme référence pour accéder aux membres statiques. À l'intérieur de la classe, cette référence est implicite (mécanisme similaire au *this*). Le mot clé *this* ne s'applique pas à l'instance statique. Seulement le nom de la classe s'y applique.

Exemple #1 : Je souhaite créer ma propre classe statique (STHMath) qui permettra d'additionner, soustraire, multiplier et diviser deux nombres.

```
static public class STHMath
{
    static public int Additionner(int iNombre1, int iNombre2)
    {
        return iNombre1 + iNombre2;
    }

    static public int Soustraire(int iNombre1, int iNombre2)
    {
        return iNombre1 - iNombre2;
    }

    static public int Multiplier(int iNombre1, int iNombre2)
    {
        return iNombre1 * iNombre2;
    }

    static public double Diviser(int iNombre1, int iNombre2)
    {
        return iNombre1 / (double)iNombre2;
    }
}
```

Exemples d'appel (Click):

```
int iNb1 = STHMath.Additionner(20,30);
int iNb2 = STHMath.Soustraire(21, 18);
int iNb3 = STHMath.Multiplier(4, 7);
double dNb4 = STHMath.Diviser(11, 5);
```

Exercice 4. : Ajouter une classe **STHMath** et ajouter le code de l'exemple précédent. Dans un bouton, ajouter le code de l'événement Click précédent.

Exercice 5. : Ajouter une classe statique nommée **Température** qui permet de convertir des **Celsius** en **Farenheit** et vice versa. Créer deux méthodes à cet effet qui prend le degré en paramètre. Voici les formules :

Celsius = (Fahrenheit - 32) X 5 ÷ 9
Fahrenheit = (Celsius X 9 ÷ 5) + 32;

Demander à l'usager de spécifier quel type de dégré qu'il va saisir, ensuite le degré et afficher le conversion.

En conclusion :

- Aucun membre *static* ne peut interagir avec un membre d'instance de sa classe.
- Tous les membres *static* d'une même classe peuvent interagir entre eux.
- Tous les membres d'instance d'une classe peuvent interagir avec les membres *static*!

Les méthodes d'extension

À partir de la version 3.5 du *framework .NET*, nous pouvons utiliser les méthodes d'extension. Les méthodes d'extension permettent **d'ajouter des méthodes à des classes déjà existantes** :

- Sans recompiler ou modifier la classe d'origine
- Même sans posséder le code de la classe

On peut considérer ces méthodes comme des méthodes « **plug-ins** » pour classes. Les méthodes d'extension les plus connues proviennent de LINQ et elles s'appliquent aux collections en général. Elles sont un type particulier de méthodes statiques qui seront appelées comme des méthodes d'instance statique.

Conditions spéciales :

- Elles sont déclarées dans **une classe statique**
- Elles sont associées à des classes d'instance
- Le **premier paramètre** sert à **arrimer la méthode au type** (à la classe d'instance). Ce paramètre est précédé par le mot clé ***this***
- La méthode est appelée **sans mentionner le premier paramètre**, donc du **2e au dernier paramètre**.

Exemple #1 : Ajouter une méthode nommée **Tripler** au type de base *int* qui prend le chiffre contenu dans la variable et retourne sa valeur en la triplant.

```
// Classe statique qui sert à héberger
// une ou plusieurs méthodes d'extensions
static public class MesExtensions
{
    // Méthode d'ext. arrimée au type int
    static public int Tripler(this int iNombre)
    {
        return iNombre * 3;
    }
}
```

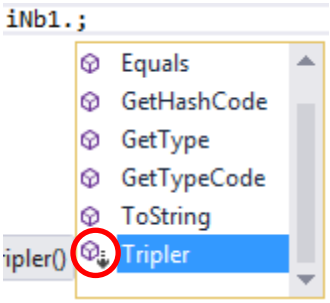
Le type (la classe) avec lequel la méthode est arrimée.

Exemple d'appel :

```
private void btnExercice01_Click(
     object sender,
     RoutedEventArgs e)
{
    int iNb1 = 5;
    int iNb2 = iNb1.Tripler();
    Console.WriteLine(iNb2); // 15
}
```

Appel

Exemple lors de la saisie :



Exercice 6. : Ajouter une classe **MesExtensions** et ajouter le code de l'exemple précédent. Dans un bouton, ajouter le code de l'événement Click précédent.

Exercice 7. : Dans la même classe, ajouter une méthode d'extension nommée « **Inverser** » qui va s'amarrer au type **string** pour retourner la chaine de caractères à l'envers. Exemple : « mot » devient « tom ».

À savoir sur l'implémentation de méthodes d'extension:

- Les méthodes d'extension n'ont aucun privilège d'accès spécial. Les membres ***private*** et ***protected*** d'une classe « extensionnée » demeureront inaccessibles à la méthode qui « extensionne ».
- On ne peut pas créer des méthodes d'extension pour les classes statiques car l'instance statique **n'a pas de *this***.
- On **ne peut pas créer d'autres membres d'extension que les méthodes** (pas de champs, propriétés, indexeurs)
- Les autres paramètres d'une méthode d'extension (à partir du 2e) n'ont **pas de contraintes**. Il peuvent être passés : par valeur, par référence (ref ou out), et utiliser le *params*.
- Une méthode d'extension **peut être surchargée** grâce à ces paramètres.
- Une méthode d'extension **sera ignorée** si elle entre en conflit (de signature) avec une méthode de la classe « extentionnée ».
- Les méthodes d'extension distribuées dans une ou plusieurs classes statiques, doivent être visibles par le projet où l'on désire les utiliser :
    - Soit en ajoutant **une référence de projet** et/ou en ajoutant les ***using* des espaces des noms** où elles ont été déclarées;
    - Soit en ajoutant les classes statiques au même espace de noms des projets qui les utilisent.
- Il est possible de déclarer toutes les méthodes d'extension dans une même classe statique. C'est à vous de déterminer quelle est la meilleure organisation de vos méthodes d'extension. Cependant, si elles deviennent nombreuses, il est préférable de les séparer dans des classes statiques distinctes, selon le type de la classe « extensionnée », afin de les retracer plus facilement.

Exercice 8. : Dans la même classe, ajouter une méthode d'extension nommée « **Répéter** » qui va s'amarrer au type **string** pour répéter le nombre de fois spécifié la chaîne de caractères. Exemple : **"mot".Répéter(4)** devient « motmotmotmot ».

Exercice 9. : Dans la même classe, ajouter une méthode d'extension nommée « **Choisir** » qui va s'amarrer à un tableau de string pour choisir au hasard un mot se trouvant dans le tableau. Exemple :

```
string[] asMots = { "Avion", "Auto", "Bateau", "Fusée", "Train" };
MessageBox.Show("Choix: " + asMots.Choisir());
```

Exercice 10. : Dans la même classe, ajouter une méthode d'extension nommée « **Mélanger** » qui va s'amarrer au type **List<T>** pour mélanger le contenu de la liste par permutation selon le nombre de fois spécifié en paramètre. (Plus difficile)

Bibliographie

Mastriani, R. (2013, 05 12). PowerPoint - Statique et extension. Saint-Hyacinthe, QC, Canada.

Tourreau, G. (2010). *C#: L'essentiel du code et des classes.* Paris: Pearson Education France.