Cours : Programmation orientée objet en C#
Cote du cours : 26978
Session : Hiver 2022

Chapitre 09
Surcharge d'opérateurs

Table des matières

Introduction

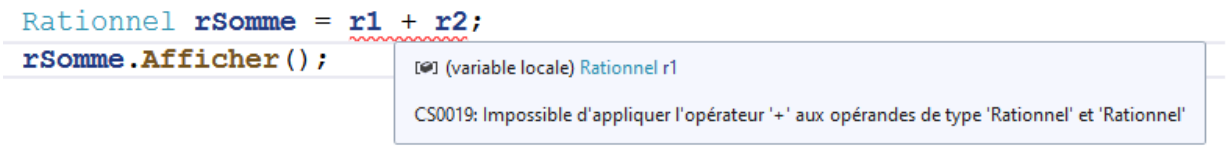
Le but de la surcharge d'opérateurs est de pouvoir utiliser les opérateurs d'addition, soustraction, multiplication et autres sur des objets. Bien sûr sans avoir à créer de méthodes spécifiques.

Si je reprends l'exemple de la classe Rationnel. Pour faire la somme de deux Rationnel, j'ai définit une méthode que j'ai appelé Somme. Ensuite dans ma méthode Main, j'ai fait appel à cette méthode.

```
public Rationnel Somme(Rationnel r)
{
    int numRes = r.Num * this.Den + this.Num * r.Den;
    int denRes = r.Den * this.Den;
    Rationnel res = new Rationnel(numRes, denRes);
    return res;
}
public static void Main()
{
    Rationnel r1 = new Rationnel(5, 2);
    Rationnel r2 = new Rationnel(3, 7);

    Rationnel rSomme = r1.Somme(r2);
    rSomme.Afficher();
}
```

L'objectif de ce chapitre est d'arriver à écrire le code suivant sans avoir d'erreur de compilation:



I. Les opérateurs arithmétiques

La syntaxe de la signature d'une surcharge d'opérateur est la suivante :

public static type operator signe (Arg1 [,Arg2])

- Le type représente le type sur lequel l'opérateur agira et le type de l'objet retourné.
- Le signe représente l'opérateur qui sera surchargé
- Les arguments
  - un seul dans le cas d'une opération unaire. Même type que le type géré par la surcharge.
  - Deux dans le cas d'un opérateur binaire. Un d'entre eux, au moins, devra être identique au type géré par la surcharge.

- L'opérateur surchargé doit **obligatoirement** être **public** et **statique**.

Exemple : surcharge de l'opérateur addition dans la classe Rationnel.

```
public static Rationnel operator +(Rationnel r1, Rationnel r2)
{
    int numRes = r1.Num * r2.Den + r2.Num * r1.Den;
    int denRes = r1.Den * r2.Den;
    Rationnel res = new Rationnel(numRes, denRes);
    return res;
}
```

Maintenant nous pouvons faire la somme des deux rationnels avec l'opérateur « + ».

```
public static void Main()
{
    Rationnel r1 = new Rationnel(5, 2);
    Rationnel r2 = new Rationnel(3, 7);

    Rationnel rSomme = r1 + r2;

    rSomme.Afficher();
}
```

Vous pouvez aussi faire la somme de plusieurs rationnels en même temps.

```
Rationnel rSomme = r1 + r2 + r3;
```

La somme déclarée précédemment fait la somme de deux rationnels. Vous pouvez faire la somme d'un rationnel avec un entier (a / 1) par exemple. L'essentiel est d'avoir une surcharge correcte pour un même opérateur.

```
public static Rationnel operator +(Rationnel r1, int a)
{
    int numRes = r1.Num * 1 + a * r1.Den;
    int denRes = r1.Den * 1;
    Rationnel res = new Rationnel(numRes, denRes);
    return res;
    // ou bien : return r1 + new Rationnel(a, 1);
}
```

On appelle l'opérateur de la façon suivante :

```
Rationnel rSomme = r1 + 4;
```

On peut aussi utiliser l'opérateur d'affectation composée.

```
r1 += 4;      // r1 = r1 + 4;
r1 += r2;     // r1 = r1 + r2;
```

La surcharge d'opérateurs fonctionne sur le même modèle pour tous les opérateurs arithmétiques (+, -, *, /, %).

Exercice 1:
Surcharger les opérateurs suivants pour la classe Rationnel : * et /.

II. Les opérateurs de comparaison

Il existe six opérateurs de comparaison (==, !=, <, >, <=, >=) que vous pouvez surcharger de la même façon que les opérateurs arithmétiques en respectant les contraintes suivantes :

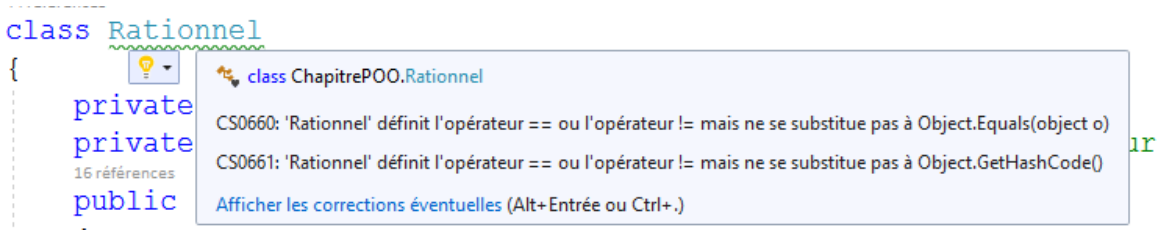
```
public static bool operator signe ( Arg1 ,Arg2] )
```

- La surcharge des opérateurs de comparaison doit être effectuée **par paires**. Si vous surchargez l'opérateur ==, vous devrez aussi surcharger != sinon une erreur sera levée au moment de la compilation. Les paires sont les suivantes :
  - == et !=
  - < et >
  - <= et >=
- Les opérateurs de comparaison doivent **obligatoirement** retourner un type **bool**.
- Si vous surchargez les opérateurs == et !=, vous devez aussi fournir des surcharges pour les méthodes **Equals** et **GetHashCode** héritées du type Object sinon des avertissements lors de la compilation seront levés. La raison principale est que la méthode **Equals** doit utiliser la même implémentation logique que l'opérateur ==.

Exemple : Implémenter la surcharge des opérateurs == et != pour la classe Rationnel.

```
public static bool operator ==(Rationnel r1, Rationnel r2)
{
    return r1.num == r2.num && r1.den == r2.den;
}

public static bool operator !=(Rationnel r1, Rationnel r2)
{
    return r1.num != r2.num || r1.den != r2.den;
}
```



Exemple : Implémentation des méthodes Equals et GetHashCode pour la classe Rationnel :

```
public override int GetHashCode()
{
    return base.GetHashCode();
```

```
}

public override bool Equals(object obj)
{
    if (obj is Rationnel)
    {
        return this == (Rationnel)obj;
    }
    else
    {
        return false;
    }
}
```

La surcharge de la méthode Equals utilise la surcharge de l'opérateur == pour déterminer le résultat retourné. La définition de la méthode Equals et de la méthode GetHashCode seront détaillées dans le chapitre suivant.

Exemple : Appel des opérateurs == et != et de la méthode Equals.

```
public static void Main()
{
    Rationnel r1 = new Rationnel(5, 2);
    Rationnel r2 = new Rationnel(3, 7);

    bool res1 = r1 == r2;
    bool res2 = r1 != r2;
    bool res3 = r1.Equals(r2);

    Console.WriteLine("r1==r2 ==> {0}", res1);
    Console.WriteLine("r1!=r2 ==> {0}", res2);
    Console.WriteLine("r1.Equals(r2) ==> {0}", res1);
}
```

```
r1==r2 ==> False
r1!=r2 ==> True
r1.Equals(r2) ==> False
```

Exercice 2:
Surcharger les opérateurs suivants pour la classe Rationnel : > et <.

Solution des exercices

Exercice 1 :

```
public static Rationnel operator *(Rationnel r1, Rationnel r2)
{
    int numRes = r1.Num * r2.Num;
    int denRes = r1.Den * r2.Den;
    Rationnel res = new Rationnel(numRes, denRes);
    return res;
}
public static Rationnel operator /(Rationnel r1, Rationnel r2)
{
    int numRes = r1.Num * r2.Den;
    int denRes = r1.Den * r2.num;
    Rationnel res = new Rationnel(numRes, denRes);
    return res;
}
```

Exercice 2 :

```
public static bool operator >(Rationnel r1, Rationnel r2)
{
    return ((double)r1.Num / r1.Den) > ((double)r2.Num / r2.Den);
}

public static bool operator <(Rationnel r1, Rationnel r2)
{
    return ((double)r1.Num / r1.Den) < ((double)r2.Num / r2.Den);
}
```