

Séance 5

Membre de classe



Ce(tte) œuvre est mise à disposition selon les termes de la Licence Creative Commons
Attribution – Pas d'Utilisation Commerciale – Pas de Modification 4.0 International.

Rappels

- Accès aux membres d'une classe selon **leur visibilité**
 - Modificateur de visibilité private, protected, package et public
 - Encapsulation des données dans une classe
 - Différence entre héritage et composition
- Définition d'**objets immuables**
 - Lien entre immuabilité et encapsulation
 - Garantie et propriétés des objets immuables

Objectifs

- Membre **de classe**

Variable et méthode de classe

- Définition de **classe interne**

- Classe imbriquée et classe interne
- Structure de données de type chainée

- **Introspection** de code

- Réflexivité et introspection
- Modèle orienté objet

Membre de classe



La classe Object

- Représentation d'un **objet générique** avec la classe Object

Racine de toute hiérarchie de classe, classe mère par défaut

- Contient plusieurs **méthodes communes** à tous les objets

Représentation avec ToString, égalité avec Equals...

```
1 Object o = new Object();
2 Console.WriteLine(o);
```

System.Object

Variable de type Object

- Variable de type Object stocke n'importe quelle référence

Puisque ancêtre implicite de toute classe

- Permet notamment d'obtenir des collections hétérogènes

Il suffit que le type statique soit Object

```
1 List<Object> data = new List<Object>();
2 data.Add ("Hello");
3 data.Add (12);
4 data.Add (new DateTime (2016, 10, 18));
5 data.ForEach (Console.WriteLine);
```

```
Hello
12
18-10-16 00:00:00
```

Membre d'une classe

- Une classe peut contenir différents types de **membres**
 - Variable et constante (`readonly`)
 - Constructeur et méthode
 - Et même des classes (imbriquée ou interne)
- Un membre lié à un objet est appelé **membre d'instance**
 - Valeurs/comportements différents selon l'instance
 - Est accédé/appelé à partir d'un objet cible

Membre d'instance (1)

- **Membre d'instance** lié à un objet particulier
 - Variable/constante d'instance spécifique à chaque objet
 - Méthode d'instance appelée sur un objet cible
- Seuls les membres d'instance y ont **accès**

E.g. une méthode d'instance a accès aux variables d'instance
- Mot réservé pour **accéder à l'objet cible** depuis la classe
this (C++, C#, Java), self (Python, Ruby), \$this (PHP)...

Membre d'instance (2)

- Classe représentant un article de magasin

Trois variables d'instance, un constructeur et une méthode

```
1  public class Item
2  {
3      private readonly string barcode;
4      private readonly string name;
5      private readonly double price;
6
7      public Item (string barcode, string name, double price)
8      {
9          this.barcode = barcode;
10         this.name = name;
11         this.price = price;
12     }
13
14     public double GetPrice (int quantity)
15     {
16         return quantity * price;
17     }
18 }
```

Membre d'instance (3)

- Création de **plusieurs instances** de la classe Item

Exécution de la méthode GetPrice pour chaque instance

```
1 Dictionary<Item,int> purchases = new Dictionary<Item,int>()
2 {
3     {new Item ("42X", "Leffe des vignes", 14.50), 10},
4     {new Item ("1101", "MacBook Pro", 1799.99), 1}
5 };
6
7 double totalprice = 0;
8 foreach (KeyValuePair<Item,int> entry in purchases)
9 {
10     totalprice += entry.Key.GetPrice (entry.Value);
11 }
12 Console.WriteLine (totalprice);
```

1944,99

Membre de classe (1)

- **Membres de classe** liés à la classe

Membre partagé entre toutes les instances de la classe

- Un membre de classe n'est **pas associé à une instance**
 - Pas d'accès aux membres d'instance
 - Accès aux autres membres de classe
- **Interdiction** de faire référence à l'objet cible

Sans quoi une erreur de compilation se produit

Membre de classe (2)

- Membre de classe également appelé **membre statique**

Déclaration avec static (C++, C#, Java)

- Définition d'une classe avec des **méthodes utilitaires**

Méthodes pas liées entre elles autour d'un objet

```
1 public class Utils
2 {
3     private static readonly int TAX_RATE = 21;
4
5     public static double GetTaxIncPrice (double price)
6     {
7         return price * (1 + TAX_RATE / 100.0);
8     }
9 }
```

Membre de classe (3)

- Accès aux membres de classe à partir du **nom de la classe**

Pas besoin de créer une instance de la classe

- Même **règles de visibilité** que pour les membres d'instance

Publique, privée, protégée et package

```
1 Item beer = new Item ("42X", "Leffe des vignes", 14.50);  
2 Console.WriteLine (Utils.GetTaxIncPrice (beer.GetPrice (1)));
```

17 ,545

Empêcher l'instanciation

- On peut empêcher la création d'une instance de la classe

En définissant le constructeur par défaut privé

- Éviter la création d'**instances inutiles**

Marque que c'est une classe avec que des méthodes de classe

```
1 public class Utils
2 {
3     private static readonly int TAX_RATE = 21;
4
5     private Utils(){}
6
7     public static double GetTaxIncPrice (double price)
8     {
9         return price * (1 + TAX_RATE / 100.0);
10    }
11 }
```

Variable de classe (1)

- Une variable de classe est **partagée** entre toutes les instances

Une seule copie de la variable par classe

- Un membre d'instance peut **accéder aux membres de classe**

Sans préfixer avec le nom de la classe s'il n'y a pas ambiguïté

```
1 public class InstanceCounter
2 {
3     private static int counter = 0;
4
5     public static int Counter
6     {
7         get { return counter; }
8     }
9
10    public InstanceCounter()
11    {
12        counter++;
13    }
14 }
```

Variable de classe (2)

- Création de **plusieurs instances** de InstanceCounter

Chaque création va incrémenter la variable de classe

- Récupération du nombre d'instances avec la **propriété Counter**

Qui est un membre de classe

```
1 new InstanceCounter();
2 Console.WriteLine (InstanceCounter.Counter);
3
4 new InstanceCounter();
5 Console.WriteLine (InstanceCounter.Counter);
```

1
2

Membre de classe ou d'instance ?

- Choix à faire en fonction du **rôle du membre**
 - Un membre d'instance est lié aux instances
 - Un membre de classe est global à toutes les instances
- Plusieurs **constructions possibles**
 - Uniquement des membres d'instance ou de classe
 - Classe hybride mélangeant les deux types de membres

Singleton

- Classe dont **une seule instance** peut exister

Récupération de l'unique instance grâce à une méthode de classe

- On pourrait **rendre publique** la constante de classe

Mais on casse alors l'encapsulation en révélant la variable

```
1  public class Singleton
2  {
3      private static readonly Singleton singleton = new Singleton();
4
5      public static Singleton GetInstance()
6      {
7          return singleton;
8      }
9
10     private Singleton(){}
11 }
```

Héritage (1)

- Il n'y a pas vraiment d'**héritage des membres de classe**

Mais accès aux membres de classe avec le nom de la sous-classe

```
1 public class SuperInstanceCounter : InstanceCounter
2 {
3     public SuperInstanceCounter()
4     {
5         Console.WriteLine ("Je suis la " + Counter + "e instance.");
6     }
7 }
```

```
1 new SuperInstanceCounter();
2 new SuperInstanceCounter();
3 Console.WriteLine (SuperInstanceCounter.Counter);
```

```
Je suis la 1e instance.
Je suis la 2e instance.
2
```

Héritage (2)

- Une méthode de classe d'une fille peut **cacher** celle de la mère

Modificateur new en C# pour signaler que c'est intentionnel

```
1 public class SuperInstanceCounter : InstanceCounter
2 {
3     public SuperInstanceCounter()
4     {
5         Console.WriteLine ("Je suis la " + Counter + "e instance.");
6     }
7
8     public new static int Counter
9     {
10         get { return 42; }
11     }
12 }
```

Je suis la 42e instance.

Je suis la 42e instance.

42

Méthode *builder* (1)

■ Méthode de classe de **construction d'un objet**

Par exemple pour permettre de parser une chaîne de caractères

```
1 public class Item
2 {
3     // [...]
4
5     public static Item ParseItem (string line)
6     {
7         String[] tokens = line.Split (';');
8         return new Item (tokens[0], tokens[1], Convert.ToDouble (
9             tokens[2]));
10    }
}
```

Méthode *builder* (2)

- Création d'un nouvel objet en **appelant la méthode de classe**
La méthode va elle-même s'occuper d'appeler le constructeur

- **Alternative** à un constructeur standard

Séparer constructions complexes à partir de représentations

```
1 Item beer = Item.ParseItem ("42X;Leffe des vignes;14.50");
2 Console.WriteLine (beer);
```

```
[42X] Leffe des vignes : 1450 euros
```

De C à C# (1)

- En C, création d'une **structure avec fonctions** séparées

Les fonctions prennent la structure en paramètre

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct complex {
5     int real;
6     int imag;
7 } COMPLEX;
8
9 COMPLEX* sum(COMPLEX *a, COMPLEX *b) {
10     COMPLEX *result = malloc(sizeof(COMPLEX));
11     result->real = a->real + b->real;
12     result->imag = a->imag + b->imag;
13     return result;
14 }
15
16 int main() {
17     COMPLEX c1 = {1, 2};
18     COMPLEX c2 = {-1, 1};
19     COMPLEX *s = sum(&c1, &c2);
20     printf("%d + %di\n", s->real, s->imag);
21 }
```

De C à C# (2)

■ En C#, création d'une **classe avec méthodes de classe**

La classe n'agit que comme conteneur de données

```
1  public class Complex
2  {
3      private readonly int real, imag;
4
5      public Complex (int real, int imag)
6      {
7          this.real = real;
8          this.imag = imag;
9      }
10
11     public static Complex sum (Complex a, Complex b)
12     {
13         return new Complex (a.real + b.real, a.imag + b.imag);
14     }
15
16     public static void Main (string[] args)
17     {
18         Complex c1 = new Complex (1, 2);
19         Complex c2 = new Complex (-1, 1);
20         Complex s = Complex.sum (c1, c2);
21         Console.WriteLine (String.Format ("{0} + {1}i", s.real, s.imag));
22     }
23 }
```

De C à C# (3)

- Une classe doit combiner **données et comportement**

Combinaison de variables et de méthodes d'instance

```
1  public class Complex
2  {
3      private readonly int real, imag;
4
5      public Complex (int real, int imag)
6      {
7          this.real = real;
8          this.imag = imag;
9      }
10
11     public Complex sum (Complex other)
12     {
13         return new Complex (this.real + other.real, this.imag + other.imag);
14     }
15
16     public static void Main (string[] args)
17     {
18         Complex c1 = new Complex (1, 2);
19         Complex c2 = new Complex (-1, 1);
20         Complex s = c1.sum (c2);
21         Console.WriteLine (String.Format ("{0} + {1}i", s.real, s.imag));
22     }
23 }
```



Classe imbriquée

Classe comme membre

- Une classe imbriquée est **encapsulée** dans une autre

Une telle classe est un membre d'une autre classe

- Plusieurs **avantages** à une telle encapsulation
 - Modularité et simplification du code
 - Partage d'une référence vers l'objet cible
 - Partage de membres privés avec une autre classe
 - Meilleure encapsulation pour la composition

Classe imbriquée (1)

- Classe imbriquée **sans référence** vers la classe externe (en Java)

Partage uniquement les membres de classe

- **Classe imbriquée** contrôle visibilité et accès

La classe imbriquée est un membre de classe de l'externe

```
1 class Outer
2 {
3     private static String name = "Outer";
4
5     static class Inner
6     {
7         private static String name = "Inner";
8
9         public String toString()
10        {
11             return String.format ("%s\n%s", name, Outer.name);
12         }
13     }
14 }
```

Classe imbriquée (2)

- Référence au nom de la classe imbriquée **à partir de l'externe**

Création instance de la classe imbriquée indépendante externe

- **Exemple de création** depuis le même package

```
1 public class Program
2 {
3     public static void main (String[] args)
4     {
5         Outer.Inner inner = new Outer.Inner();
6         System.out.println (inner);
7     }
8 }
```

Inner
Outer

Classe interne (1)

- Partage de la référence this avec la classe interne (en Java)

Référence vers une instance de la classe externe

- Accès aux membres privés de la classe externe depuis l'interne

La classe interne est un membre d'instance de l'externe

```
1 class Outer
2 {
3     private String name = "Outer";
4
5     class Inner
6     {
7         private String name = "Inner";
8
9         public String toString()
10        {
11             return String.format ("%s\n%s", name, Outer.this.name);
12         }
13     }
14 }
```

Classe interne (2)

- Instanciation classe interne à partir **référence classe externe**

Création instance de la classe imbriquée dépendante externe

- **Exemple de création** depuis le même package

```
1 public class Program
2 {
3     public static void main (String[] args)
4     {
5         Outer outer = new Outer();
6         Outer.Inner inner = outer.new Inner();
7         System.out.println (inner);
8     }
9 }
```

Inner
Outer

Classe interne en C# (1)

- En C#, une classe définie dans une autre est **imbriquée**

Simuler classe interne avec stockage explicite référence externe

```
1  public class Outer
2  {
3      private String name = "Outer";
4
5      private class Inner
6      {
7          private readonly Outer o;
8          private String name = "Inner";
9
10         public Inner (Outer o)
11         {
12             this.o = o;
13         }
14
15         public override string ToString()
16         {
17             return String.Format ("{0}: {1}", this.name, o.name);
18         }
19     }
20 }
```

Classe interne en C# (2)

- Passage explicite **référence classe externe**

Lors de la création d'une instance de la classe imbriquée

- **Exemple de création** depuis le même namespace

```
1 public class Program
2 {
3     public static void Main (string[] args)
4     {
5         Outer outer = new Outer();
6         Outer.Inner inner = new Outer.Inner (outer);
7         Console.WriteLine (inner);
8     }
9 }
```

```
Inner
Outer
```

Structure chainée simple

- Relation de composition par **classe imbriquée**

Exemple avec création d'une liste simplement chainée

```
1 LinkedList<string> data = new LinkedList<string>();
2 Console.WriteLine (data.Size + " : " + data);
3
4 data.addFirst ("un");
5 data.addLast ("deux");
6 Console.WriteLine (data.Size + " : " + data);
7
8 data.addLast ("trois");
9 data.addLast ("quatre");
10 data.addFirst ("zéro");
11 Console.WriteLine (data.Size + " : " + data);
```

```
0 : []
2 : [un, deux]
5 : [zéro, un, deux, trois, quatre]
```

Élément de la liste

- Élément de la liste représenté par la classe ListItem

Classe imbriquée à la classe LinkedList

```
1  public class LinkedList<T>
2  {
3      // [...]
4
5      private class ListItem
6      {
7          public T data;
8          public ListItem next;
9
10         public ListItem (T data, ListItem next)
11         {
12             this.data = data;
13             this.next = next;
14         }
15     }
16 }
```

Initialisation de la liste

- Mémorisation de la **tête de liste** et taille

La taille pourrait être recalculée à chaque fois

```
1 public class LinkedList<T>
2 {
3     private ListItem first;
4     private int size;
5
6     public LinkedList()
7     {
8         first = null;
9         size = 0;
10    }
11
12    public int Size { get { return size; } }
13
14    public bool IsEmpty() { return Size == 0; }
15
16    // [...]
17 }
```

Affichage de la liste

■ Parcours de tous les éléments de la liste

Utilisation d'un « pointeur » de l'élément courant

```
1  public class LinkedList<T>
2  {
3      // [...]
4
5      public override string ToString ()
6      {
7          if (IsEmpty()) { return "[]"; }
8
9          ListItem current = first;
10         string result = "[" + current.data;
11         while (current.next != null)
12         {
13             current = current.next;
14             result += ", " + current.data;
15         }
16         return result + "]";
17     }
18
19     // [...]
20 }
```

Insertion en début de liste

■ Insertion en début de liste immédiate

Le nouvel élément devient la nouvelle tête de liste

```
1 public class LinkedList<T>
2 {
3     // [...]
4
5     public void addFirst (T data)
6     {
7         first = new ListItem (data, first);
8         size++;
9     }
10
11    // [...]
12 }
```

Insertion en fin de liste

■ Insertion en fin de liste plus compliquée

Accrocher le nouveau ListItem au dernier élément

```
1 public class LinkedList<T>
2 {
3     // [...]
4
5     public void addLast (T data)
6     {
7         ListItem item = new ListItem (data, null);
8         if (IsEmpty()) { first = item; }
9
10        ListItem current = first;
11        while (current.next != null)
12        {
13            current = current.next;
14        }
15        current.next = item;
16        size++;
17    }
18
19    // [...]
20 }
```