//

Bloc2 Bases de la POO	CHAPITRE 5 Attributs et méthodes à portée classe, constructeur statique classe statique

Objectifs : comprendre l'utilité des membres (attributs et méthodes) à portée classe
comprendre l'utilité d'un constructeur statique
comprendre l'utilité d'une classe statique

SOMMAIRE

I. 1

II. 2

1) 3

2) 5

3) 6

4) **Erreur ! Signet non défini.**

5) 7

III. 7

1) Présentation des classes statiques 7

2) 8

I.PROBLÉMATIQUE

La SII SynapsInfo souhaite développer une application lui permettant de gérer ses employés. Pour cela une classe Employe a été réalisée. Elle vous est fournie partiellement ci-dessous :

```
class Employe
{
        private int numero;
        private string nom ;
        private int age;
        public Employe(int unNum, string unNom, int unAge) {…}
        … // accesseurs //
        … // autres méthodes//
}
```

Demande d'évolution : on souhaite que notre classe contienne l'âge minimum (16 ans) que doit avoir un employé pour travailler dans l'entreprise

Proposition de solution : ajout d'un attribut privé pour gérer cette information dans la classe

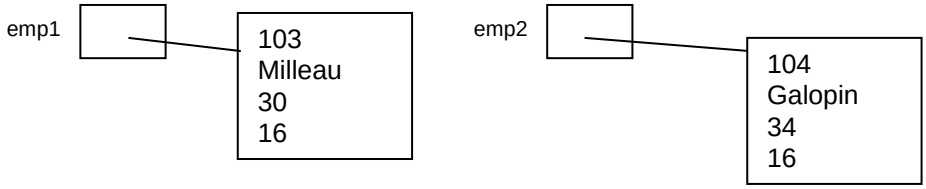
```
class Employe
{
        private int numero;
        private string nom ;
        private int age;
        private int ageMin;

        public Employe(int unNum, string unNom, int unAge) {…}
        … // accesseurs //
        … // autres méthodes//
}
```

Créons 2 objets ayant pour référence emp1 et emp2, instances de la classe Employe :

```
Employe emp1 = new Employe(103,"Milleau", 30, 16)
Employe emp2 = new Employe(104,"Galopin", 34, 16)
```

En mémoire, chacun de ces objets dispose de ses propres variables :



Problème de cette proposition :

- Redondance d'information : chaque objet possède une variable contenant l'âge minimal

Solution retenue :

⇨ **Nous allons utiliser un attribut <u>à portée classe</u> pour gérer cette information.**

⇨

⇨

⇨ **Cet attribut à portée classe permettra d'enregistrer l'âge minimim <u>une seule fois</u>.**

II. ATTRIBUTS ET MÉTHODES À PORTÉE CLASSE, CONSTRUCTEUR STATIQUE

1) Les attributs à portée classe

➢ **Une classe peut contenir deux catégories d'attributs :**

- des attributs d'instance (ceux que vous utilisez déjà)
- des attributs à portée classe (appelés également attributs de classe ou attributs statiques, ou encore variable de classe ou variable à portée classe)

➢ **Déclaration d'un attribut est à portée classe**

Voici comment déclarer l'âge minimum dans la classe Employe :

```
private  static  int ageMin = 16 ;
```

ATTENTION : il est possible d'initialiser l'attribut ageMin lors de sa déclaration mais il est préférable de réaliser l'initialisation de l'attribut **dans un constructeur statique** (voir paragraphe suivant).

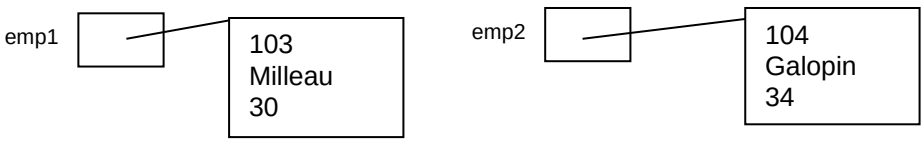
➢ **Différence entre un attribut à portée classe et un attribut d'instance**

- Un **attribut d'instance** a une valeur pour **chaque objet créé**
- Un **attribut à portée classe** se rapporte à la classe, sa valeur existe **en un seul exemplaire**. **Les objets créés ne contiendront aucune valeur pour cet attribut à portée classe**

Créons 2 objets ayant pour référence emp1 et emp2, instances de la classe Employe contenant à présent l'attribut à portée classe ageMin :

```
Employe emp1 = new Employe(103,"Milleau",30)
Employe emp2 = new Employe(104,"Galopin",34)
```

En mémoire, chacun de ces objets dispose d'une valeur pour chaque attribut d'instance mais il ne possède aucune valeur pour l'attribut à portée classe :



➢ **Quand utiliser un attribut à portée classe ?**

Quand tous les objets d'une classe ont besoin de gérer une valeur identique à tous les objets.

Exemples : les employés ne peuvent pas être âgés de moins de 16 ans
les employés ne peuvent pas percevoir un salaire annuel > 150 000 €

➢ **Pour utiliser un attribut à portée classe, vous n'avez pas besoin de créer un objet**

➢ **Mode de protection des attributs à portée classe**

private pour respecter le principe d'encapsulation

⇨ **il faudra donc définir des accesseurs pour y accéder**

ATTENTION : les accesseurs permettant d'obtenir les valeurs des attributs de classe devront **obligatoirement être des méthodes à portée classe**

2) Les méthodes à portée classe

➢ **Une classe peut contenir deux catégories de méthodes :**

- des méthodes d'instance qui seront appelées **par un objet** (= une instance)

exemple : emp1.AfficherCaract()

- des méthodes à portée classe (ou méthodes statiques)

  une méthode à portée classe n'a pas besoin d'objet pour l'appeler

  Il y en de nombreuses méthodes à portée classe dans le framework dotnet :

  Console.WriteLine(…) ;
  int nbjour = DateTime.DaysInMonth(2012,12)

➢ **Comment indiquer dans une classe qu'une méthode est à portée classe ?**

Exemple : **public static int GetAgeMin()**

```
{
   …
}

public static void SetAgeMin(int nouvAge)
{
```

```
    …
}
```

➢ **Comment appeler une méthode à portée classe ?**

nomDeLaClasse.nomDeLaMéthodeAPortéeClassse(….)

exemple :
int ageMin = **Employe.**GetAgeMin() ;

➢ **Quand a-t-on besoin d'une méthode à portée classe ?**

- C'est obligatoire pour accéder à un attribut statique (getter + setter)
- Lorsque l'on veut utiliser une méthode sans être obligé de créer un objet.

Par exemple, dans la classe Employe, on pourrait ajouter une méthode à portée classe permettant de créer une collection contenant des employés dont les caractéristiques sont enregistrées dans un fichier public static List<Employe> RemplirListeAPartirFichier(string nomfic)

➢ **Quel est le mode de protection des méthodes à portée classe ?**

Elles peuvent être privées ou publiques

3) Le constructeur statique

➢ un constructeur statique est utilisé pour **initialiser n'importe quel attribut à portée classe**, **OU** pour effectuer un traitement **devant être effectué une seule fois**

➢ il est appelé **automatiquement avant** que la première instance soit créée, il **ne peut pas être appelé dans un programme via une instruction**

➢ un constructeur statique **n'accepte pas** de mode de protection (private et public interdits) et **n'a pas de paramètres**.

➢ Faisons le nécessaire pour que l'attribut contenant l'âge minimum soit initialisé dans un constructeur statique :

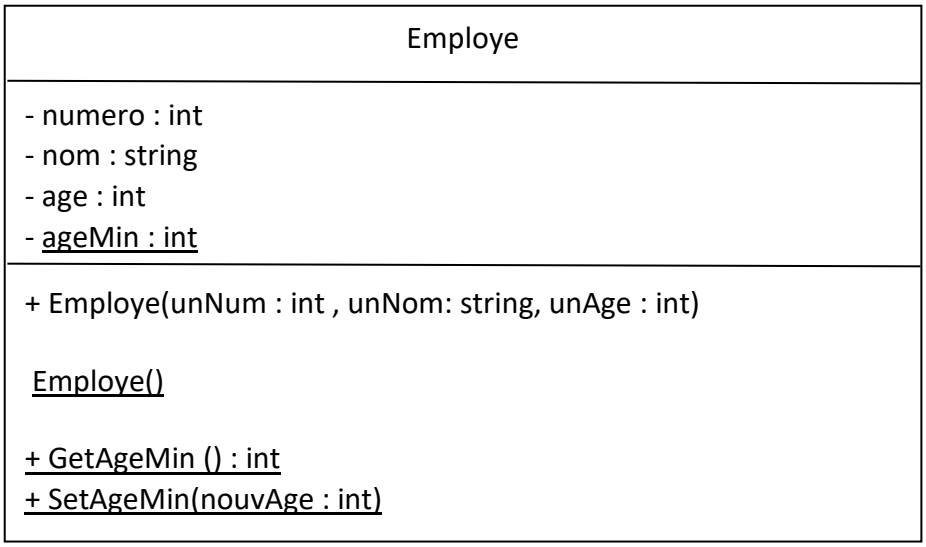
```
class Employe
{
        private int numero;
        private string nom ;
        private int age;
        private static int ageMin;

        public Employe(int unNum, string unNom, int unAge)
        {…}
        static Employe()
{
    ageMin = 16;
 }
```

```
        … // accesseurs et autres méthodes
}
```

4) Diagramme de classes UML

int



5) Application

En utilisant Visual Studio, créez la classe Employe avec l'attribut static **ageMin** et ajoutez un **nouvel attribut static contenant le salaire annuel maximum** pouvant être perçu. L'attribut contenant le salaire maximum autorisé sera initialisé dans le constructeur statique avec la valeur 150000 et il ne sera accessible qu'en lecture.

Vous ajouterez un attribut d'instance à votre classe qui contiendra le salaire annuel de l'employé.

Puis, vous créerez le programme principal qui réalisera le traitement suivant :

- Affichez l'âge minimum de travail
- Demandez à l'utilisateur s'il souhaite modifier l'âge minimum. S'il répond oui, demandez à l'utilisateur de saisir la nouvelle valeur et modifiez la valeur dans la classe.
- Faites saisir les caractéristiques des employés à l'utilisateur et stockez-les au fur et à mesure dans une collection. Faites les contrôles de saisie suivants au cours de la saisie des caractéristiques d'un employé :
  - l'âge de l'employé doit être supérieur ou égal à l'âge min,
  - le salaire de l'employé doit être inférieur au salaire max
- Suite à la saisie, vous afficherez les caractéristiques des tous les employés

III. LES CLASSES STATIQUES

1) Présentation des classes statiques

➢ **Définition**

Une classe est dite statique lorsque tous les attributs et toutes les méthodes qu'elle contient sont statiques.

➢ **Comment indiquer qu'une classe est statique ?**

En indiquant le mot clé static devant le mot clé class

Exemple :

```
static class TVA
{

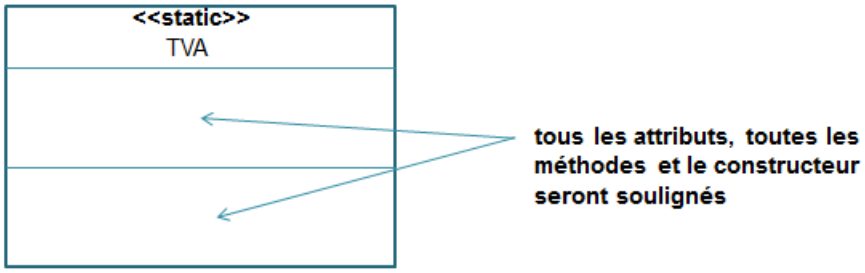
}
```

➢ **Caractéristiques d'une classe statique**

- Il n'est pas possible de créer des instances à partir d'une classe statique (new interdit !!)
- Elle ne peut contenir que des méthodes statiques et le constructeur doit être statique
- On utilise ce type de classe lorsqu'on souhaite ne pas avoir à créer d'objet pour appeler des méthodes
- Une classe statique est **« sealed »** , ce qui signifie que l'on ne peut pas en hériter –C.F. prochain cours–

2) UML et les classes statiques

Au-dessus du nom de la classe on indique <<static>>



3) Exemple de classe statique

```
static class TVA
{
    static float tauxNormal;

    public static float TauxNormal
    {
        get { return TVA.tauxNormal; }
        set { TVA.tauxNormal = value; }
    }
    static float tauxReduit;

    public static float TauxReduit
    {
        get { return TVA.tauxReduit; }
        set { TVA.tauxReduit = value; }
    }
static TVA()
{
    string [ ]tabEnreg; // tableau retourné lors de l'appel de la méthode split
    string enreg;       // contient un enregistrement (une ligne) du fichier
    StreamReader monFichier=null ;
    try
    {
        // création du flux associé au fichier
        monFichier = new StreamReader(@"c:\SI6\tva.txt");

        // lecture d'un enregistrement
        enreg = monFichier.ReadLine();

        // test pour savoir si le fichier est vide
        if (enreg == null)
        {
            tauxNormal = 0;
            tauxReduit = 0;
        }
        else
            while (enreg != null)
            {
                // la méthode split permet d'obtenir les valeurs contenues entre les ;
                // tabEnreg[0] va contenir le code tva,
                // tabEnreg[1] va contenir le taux de tva
                tabEnreg = enreg.Split(';');
                if (tabEnreg[0] == "N")
                    float.TryParse(tabEnreg[1], out tauxNormal);
                if (tabEnreg[0] == "R")
                    float.TryParse(tabEnreg[1], out tauxReduit);
                enreg = monFichier.ReadLine();
            }
        // fermeture du fichier
        monFichier.Close();
    }
        catch(System.IO.DirectoryNotFoundException)
        {
            Console.WriteLine("Répertoire non trouvé");
        }
        catch (Exception err)
        {
            Console.WriteLine("Problème de lecture des taux de TVA "+ err.Message);
        }
        finally
        {
            // le bloc finally est exécuté dans tous les cas ( traitement OK ou si exception)
            if (monFichier != null) monFichier.Close();
        }
    }
    public static float CalculMontantTva(float ht,char typeTaux)
    {
        if (typeTaux=='N')
            return (1 + tauxNormal/100) * ht;
        else
            return (1+ tauxReduit/100) * ht;
    }
}
```