MEMBRES DE CLASSE ; STATIC (1)

➢ Attribut de classe

- Attribut relatif à la *classe* elle même et non à chaque *objet*
- valeur partagée par toutes les instances/objets de la classe.

Exemples :

- valeur constante
- compteur du nombre d'instances d'une classe donnée.
- etc.

➢ Syntaxe. Instanciation des attributs de classe

```
static <déclaration d'attribut>
```

```
class Complexe {
public:
    static int nbInstances; // DECLARATION
    // nbInstances est un attribut de classe.
    // Il n'est alloué qu'une et une seule fois
    // en mémoire, au lancement du programme.
    // Toutes les intances de la classe partagent
    // cet attribut.
    // Il est attaché à la classe, pas à ses instances
    // Il accessible par la classe elle même
    // ou par chacun des objets de la classe
...
}
```

➢ Initialisation des attributs de classe

- Création au lancement du programme.
- Initialisation explicite, comme les variables globales en C.

En général dans le .cpp de la classe :

```
// Dans le fichier source .cpp
 int Complexe::nbInstance = 0; // INITIALISATION
```

MEMBRES DE CLASSE ; STATIC (2)

➢ Notion de méthode de classe

Méthode qui existe indépendamment des instances de la classe.
Elle n'a accès qu'aux attributs de classe (static), mais pas aux autres attributs.
Déclarée également avec ***static***.

➢ Accès aux méthodes et attributs de classe

Deux syntaxes :

- syntaxe habituelle si on dispose d'une instance de la classe :

```
Complexe c ;
    cout << c.nbInstances;
```

- comme attributs et méthodes de classe sont relatifs à la classe, et pas aux objets, on peut aussi y accéder sans objet avec **<NomDeClasse>::**

```
    cout << Complexe::nbInstances;
```

➢ Exemples d'usages

- Compteur d'instances, comme dans l'exemple précédent.
- Constante de classe (=>valeur est partagé par toutes les instances).

```
class Voiture {
    static const int NB_PORTES=4 ;
    // remarque : un membre static const
    // peut être initialisé dans la déclation
} ;
```

➢ Notion de classe utilitaire

Classe dont tous les membres sont statiques.
Permet de regrouper des algorithmes d'usage courant sous un nom unique.

```
class TimeConvertTool {
    static const int MIN_PER_HOUR = 60 ;
    static double minToHour(int nbMin) {
        return ((double) nbMin) / MIN_PER_HOUR ;
    }
    static int hourToMin(int nbHour) {
        return nbHour * MIN_PER_HOUR ;
    }
} ;
int main() {
    cout << TimeConvertTool::secToHour(3600) ;
    …
}
```

MEMBRES DE CLASSE ; STATIC (3)

➢ Exemple : compter le nombre d'instances d'une classe Etudiant

```
class Etudiant {
private:
     static int s_nbInstances; // attribut de classe
     int age;   string nom ;
public:
     static double s_test ; // un attribut de classe public

     Etudiant() { s_nbInstances++ ; }
     Etudiant(const string & nom, int age) {
          this->nom = nom;          this->age = age;
          s_nbInstances++ ;
     }
     // constructeur de copie
     Etudiant(const Etudiant & other) {
          this->nom = other.nom;    this->age = other.age;
          s_nbInstances++ ;
     }
     ~Etudiant() { s_nbInstances-- ; }  //Destructeur

     void setNom(const string & nom) {this->nom = nom ;}
     const string & getNom() const { return nom ; }
     static int getNbInstances() { return s_nbInstances;}
} ;


// instanciation et initialisation des attributs de classe
// Dans le fichier source .cpp
int Etudiant::s_test = 0;
double Etudiant::s_nbInstances = 0;

int main() {
     Etudiant a("Paul", 21) ;
     Etudiant *pb = new Etudiant("Sylvie", 20) ;
     cout << a.getNbInstances() <<endl;            // affiche 2
     cout << Etudiant::getNbInstances() <<endl;    // Pareil !

     a.s_nbInstances = 9 ;               //INTERDIT: membre privé!
     Etudiant::s_nbInstances = 2 ;       //INTERDIT , idem

     Etudiant::s_test = 8.3 ;            // OK : s_test est public
     cout << a.s_test <<endl;                      // affiche 8.3

     delete pb ;
     cout << a.getNbInstances() <<endl;            // affiche 1
     cout << Etudiant::getNbInstances() <<endl;    // affiche 1
}
```