TP POO C++ : Les membres statiques

TP POO C++ : Les membres statiques

Les objectifs de ce TP sont de découvrir l'utilisation des membres statiques (attributs et méthodes) en C++.

Travail demandé

On vous fournit un programme `testPoint.cpp` où vous devez décommenter progressivement les parties de code source correspondant aux questions posées.

Membres données statiques

Un membre « donnée » (attribut) déclaré avec le mot clé `static` est **partagé par tous les objets de la même classe**. Il existe même lorsque aucun objet de cette classe n'a été créé.

Un membre « donnée » (attribut) statique doit être initialisé explicitement, à l'extérieur de la classe (même s'il est privé), en utilisant l'opérateur de résolution de portée (::) pour spécifier sa classe.

En général, son initialisation se fait dans le fichier `.cpp` de définition de la classe.

On va utiliser un attribut statique `nbPoints` pour **compter le nombre d'objets `Point` créés à un instant donné**.

On **déclare** un membre « donnée » statique de la manière suivante :

```
class Point
{
   private:
      double x, y; // nous sommes des attributs de la classe Point
      static int nbPoints; // je suis un membre statique

   ...
};
```

Point.h

Il faut maintenant `initialiser` ce membre statique et `compter/décompter` le nombre d'objets créés et détruits :

```
int Point::nbPoints = 0; // initialisation d'un membre statique

Point::Point() // Constructeur
{
   x = 0.;
   y = 0.;
   nbPoints++; // un objet Point de plus
}

...

Point::~Point() // Destructeur
{
   nbPoints--; // un objet Point de moins
}
```

Point.cpp

Tous les constructeurs de la classe `Point` doivent incrémenter le membre statique `nbPoints` !

Fonctions membres statiques

Lorsqu'une fonction membre a **une action indépendante d'un quelconque objet de sa classe**, on peut la déclarer avec le mot clé `static`.

Dans ce cas, une telle fonction peut être appelée, sans mentionner d'objet particulier, en préfixant simplement son nom du nom de la classe concernée, suivi de l'opérateur de résolution de portée (::).

Les fonctions membre statiques (méthodes) ont les particularités suivantes :

- elles ne peuvent pas accéder aux attributs de la classe car il est possible qu'aucun objet de cette classe n'ait été créé.
- elles peuvent accéder aux membres données statiques car ceux-ci existent même lorsque aucun objet de cette classe n'a été créé.

On va utiliser une fonction membre statique `compte()` pour **connaître le nombre d'objets `Point` existants à un instant donné**.

On **déclare** une fonction membre statique de la manière suivante :

```
class Point
{
   private:
      double x, y; // nous sommes des attributs de la classe Point
      static int nbPoints; // je suis un attribut statique

   public:
      ...
      static int compte(); // je suis une méthode statique

};
```

Point.h

Il faut maintenant `définir` cette méthode statique :

```
// je retourne le nombre d'objets Point existants à un instant donné
int Point::compte()
{
   return nbPoints;
}
```

Point.cpp

Question 1. Compléter les fichiers `Point.cpp` et `Point.h` fournis afin d'implémenter les membres statiques `nbPoints` et `compte()`. Décommenter les parties de code de cette question dans le fichier `testPoint.cpp` et tester. Pensez à bien compter tous les objets créés !

```
Point p0, p1(4, 0.0), p2(2.5, 2.5);
Point *pp = new Point(1,1);

cout << "Il y a " << Point::compte() << " points\n"; // Affiche : Il y a 4 points

delete pp;

cout << "Il y a " << Point::compte() << " points\n"; // Affiche : Il y a 3 points
```

Les services, qui permettent de calculer la **distance** entre 2 points et le **milieu** de 2 points, ont une action indépendante et peuvent être déclarés comme statique.

Question 2. Compléter les fichiers `Point.cpp` et `Point.h` fournis afin d'implémenter les méthodes statiques `distance()` et `milieu()`. Décommenter les parties de code de cette question dans le fichier `testPoint.cpp` et tester. Vérifier que vous obtenez ceci.

```
Point p1(4, 0.0), p2(2.5, 2.5);

double distance = Point::distance(p1, p2);
cout << "La distance entre p1 et p2 est de " << distance << endl;

Point pointMilieu = Point::milieu(p1, p2);
cout << "Le point milieu entre p1 et p2 est "; pointMilieu.affiche();
```

Ce qui donne :

```
La distance entre p1 et p2 est de 2.91548
Le point milieu entre p1 et p2 est <3.25,1.25>
```