TP POO C++ : La classe Point

TP POO C++ : La classe Point

Les TP d'acquisition des fondamentaux visent à construire un socle de connaissances de base, à appréhender un concept, des notions et des modèles qui sont fondamentaux. Ce sont des étapes indispensables pour aborder d'autres apprentissages. Les TP sont conduits de manière fortement guidée pour vous placer le plus souvent dans une situation de découverte et d'apprentissage.

Les objectifs de ce TP sont de découvrir la programmation orientée objet en C++.

Initiation à la programmation orientée objet

Nous allons successivement découvrir les notions suivantes :

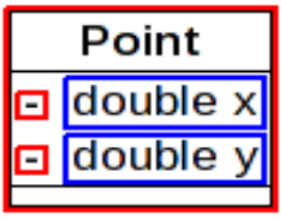
- la modélisation et la déclaration d'une classe
- la construction d'objets d'une classe et la définition de ses fonctions membres
- l'accès contrôlé aux membres d'une classe
- les services rendus par une classe

Une classe Point

On veut manipuler des **points**. Un **point** est défini par **son abscisse** (`x`) et **son ordonnée** (`y`). L'abscisse et l'ordonnée d'un point sont des **réels** (`double`).

On en sait suffisamment pour modéliser une **classe `Point`** :

UML	C++

	`class Point`
	`{`
	`  private:`
	`  double x;`
	`  double y;`
	`};`

Lorsqu'on modélise une nouvelle classe, on commence toujours par identifier ses caractéristiques afin de déterminer le nom et le type des ses attributs. C'est seulement ensuite que l'on s'attache à son comportement.

La classe `Point` devra disposer de **deux constructeurs** : un constructeur par défaut qui initialisera `x` et `y` à zéro et un autre constructeur qui recevra en paramètres l'abscisse et l'ordonnée du point à initialiser.

Tous les attributs de la classe `Point` seront **privés par respect du principe d'encapsulation**. On veut néanmoins pouvoir connaître son abscisse et son ordonnée, et éventuellement les modifier. Il faudra donc créer les **accesseurs** et **mutateurs** correspondants.

Un point pourra s'**afficher**. On aura donc une **méthode `affiche()`** qui produira un affichage de ce type : `<x,y>`

On doit aussi pouvoir calculer la **distance** entre 2 points et le **milieu** de 2 points.

Une méthode publique est un service rendu à l'utilisateur de l'objet.

Travail demandé

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

```
#include <iostream>
#include <iomanip>

using namespace std;

#include "Point.h"

int main()
{
   /* Question 1 */
   /*
   cout << "Question 1 : constructeur par défaut" << endl;
   Point p0;
   */

   /* Question 2 */
   /*
   cout << "Question 2 : constructeur" << endl;
   Point p1(1., 2.);
   Point p2(4., 0.);
   */

   /* Question 3 */
   /*
   cout << "Question 3 : accesseurs et mutateurs" << endl;
   cout.precision(2);

   cout << "L'abscisse de p0 est " << p0.getX() << endl;
   cout << "L'abscisse de p1 est " << p1.getX() << endl;
   cout << "L'abscisse de p1 est " << p2.getX() << endl;
   cout << endl;

   cout << "L'ordonnée de p0 est " << p0.getY() << endl;
   cout << "L'ordonnée de p1 est " << p1.getY() << endl;
   cout << "L'ordonnée de p1 est " << p2.getY() << endl;
   cout << endl;

   p0.setX(5.5);
   cout << "L'abscisse de p0 est maintenant " << p0.getX() << endl;
   p0.setY(6.5);
   cout << "L'ordonnée de p0 est maintenant " << p0.getY() << endl;
   cout << endl;
   */
```

```
/* Question 4 */
/*
cout << "Question 4 : allocation dynamique" << endl;
Point *p3; // je suis pointeur non initialisé sur un objet de type Point
p3 = new Point(2. , 2.); // j'alloue dynamiquement un objet de type Point

// Comme p3 est une adresse, je dois utiliser l'opérateur -> pour accéder aux membres de
   cet objet
cout << "L'abscisse de p3 est " << p3->getX() << endl;
p3->setY(0); // je modifie la valeur de l'attribut y de p3
(*p3).setY(1); // cette écriture est possible : je pointe l'objet puis j'appelle sa
   méthode
cout << "L'ordonnée de p3 est " << p3->getY() << endl;
cout << "L'adresse de p3 est " << p3 << endl;
delete p3; // ne pas oublier de libérer la mémoire allouée pour cet objet
p3 = NULL;
cout << endl;
*/

/* Question 5 */
/*
cout << "Question 5 : tableau d'objets" << endl;
Point tableauDe10Points[10]; // typiquement : les cases d'un tableau de Point

cout << "Un tableau de 10 Point : " << endl;
for(int i = 0; i < 10; i++)
{
   cout << "P" << i << " = <" << tableauDe10Points[i].getX() << "," << tableauDe10Points[
      i].getY() << ">\n";
}
cout << endl;
*/

/* Question 6 */
/*
cout << "Question 6 : objet constant" << endl;
const Point p4(7., 8.);
cout << "L'abscisse de p4 est " << p4.getX() << endl;
//p4.setX(5.); // essayez quand même !
cout << "L'ordonnée de p4 est " << p4.getY() << endl;
cout << endl;
*/

/* Question 7 */
/*
cout << "Question 7 : rendre des services" << endl;
cout << "p0 = ";
p0.affiche();
cout << "p1 = ";
p1.affiche();
cout << "p2 = ";
p2.affiche();
```

```
   //cout << "p3 = ";
   //p3->affiche(); // essayez quand même !!
   cout << "p4 = ";
   p4.affiche();
   cout << endl;
   */

   /* Question 8 */
   /*
   cout << "Question 8 : calcul de la distance entre 2 points" << endl;
   double distance = p1.distance(p2);
   cout << "La distance entre p1 et p2 est de " << distance << endl;
   cout << endl;
   */

   /* Question 9 */
   /*
   cout << "Question 9 : calcul le point milieu entre 2 points" << endl;
   Point pointMilieu = p1.milieu(p2);
   cout << "Le point milieu entre p1 et p2 est ";
   pointMilieu.affiche();
   cout << endl;
   */

   return 0;
}
```

testPoint.cpp

À la fin du TP, vous devez obtenir l'exécution suivante :

```
$ ./testPoint
Question 1 : constructeur par défaut
Question 2 : constructeur
Question 3 : accesseurs et mutateurs
L'abscisse de p0 est 0
L'abscisse de p1 est 1
L'abscisse de p1 est 4

L'ordonnée de p0 est 0
L'ordonnée de p1 est 2
L'ordonnée de p1 est 0

L'abscisse de p0 est maintenant 5.5
L'ordonnée de p0 est maintenant 6.5

Question 4 : allocation dynamique
L'abscisse de p3 est 2
L'ordonnée de p3 est 1
L'adresse de p3 est 0x696010

Question 5 : tableau d'objets
Un tableau de 10 Point :
P0 = <0,0>
P1 = <0,0>
```

```
P2 = <0,0>
P3 = <0,0>
P4 = <0,0>
P5 = <0,0>
P6 = <0,0>
P7 = <0,0>
P8 = <0,0>
P9 = <0,0>

Question 6 : objet constant
L'abscisse de p4 est 7
L'ordonnée de p4 est 8

Question 7 : rendre des services
p0 = <5.5,6.5>
p1 = <1,2>
p2 = <4,0>
p4 = <7,8>

Question 8 : calcul de la distance entre 2 points
La distance entre p1 et p2 est de 3.6

Question 9 : calcul le point milieu entre 2 points
Le point milieu entre p1 et p2 est <2.5,1>
```

Constructeur par défaut

Si vous essayer de créer un objet sans lui fournir une abscisse `x` et une ordonnée `y`, vous obtiendrez le message d'erreur suivant :

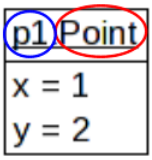
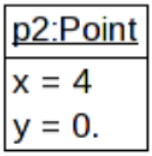
```
ligne x: erreur: no matching function for call to Point::Point()'
```

Ce type de constructeur se nomme un **constructeur par défaut**. Le constructeur par défaut de la classe `Point` doit initialiser les attributs `x` et `y` à zéro.

Question 1. Compléter les fichiers `Point.cpp` et `Point.h` fournis afin d'implémenter le constructeur par défaut de la classe `Point`.

Constructeur

On doit maintenant surcharger un nouveau constructeur pour initialiser des objets `Point` en fonction de leur abscisse et leur ordonnée :

UML	C++
	`Point p1(1, 2);`
	`Point p2(4, 0.);`

Question 2. Compléter les fichiers `Point.cpp` et `Point.h` fournis afin d'implémenter un constructeur qui recevra en paramètres l'abscisse et l'ordonnée d'un point.

Accesseurs et mutateurs d'un point

Question 3. Compléter les fichiers `Point.cpp` et `Point.h` fournis afin d'implémenter les accesseurs et mutateurs des attributs `x` et `y`.

Allocation dynamique d'objet

Pour allouer dynamiquement un objet en C++, on utilisera l'opérateur `new`. Celui-ci renvoyant une adresse où est créé l'objet en question, il faudra un pointeur pour la conserver. Manipuler ce pointeur, reviendra à manipuler l'objet alloué dynamiquement.

Pour libérer la mémoire allouée dynamiquement en C++, on utilisera l'opérateur `delete`.

Question 4. Vérifier le fonctionnement de l'allocation dynamique de l'objet `Point p3` réalisée dans le programme `testPoint.cpp`.

Un tableau d'objets

Il est possible de conserver et de manipuler des objets `Point` dans un tableau.

Question 5. Tester le fonctionnement de la déclaration d'un tableau de 10 objets `Point` réalisée dans le programme `testPoint.cpp`. Quel constructeur est appelé pour la création des objets `Point` ? Combien de fois ?

Un objet Point constant

Les règles suivantes s'appliquent aux objets constants :
- On déclare un objet constant avec le modificateur `const`
- On ne peut appliquer que des méthodes constantes sur un objet constant
- On déclare une méthode constante en ajoutant le modificateur `const`
- Un objet passé en paramètre sous forme de référence constante est considéré comme constant

Question 6. Vérifier l'utilisation de l'objet constant `p4` réalisée dans le programme `testPoint.cpp`. Si vous obtenez une erreur à la compilation lorsque vous appellez la méthode `getX()` ou `getY()` sur l'objet constant `p4`, proposez une correction de cette erreur dans la classe `Point`.

Rendre des services

Un objet point pourra s'**afficher**. On aura donc une méthode `affiche()` qui produira un affichage de ce type : `<x,y>`

On utilisera cette méthode dès que l'on voudra **afficher** les coordonnées d'un point :

```
cout << "P1 = ";
p1.affiche();

cout << "P2 = ";
p2.affiche();
```

Ce qui doit donner à l'exécution :

```
P1 = <1,2>
P2 = <4,0>
```

A l'intérieur de la méthode `affiche()` de la classe `Point`, on peut accéder directement à l'abscisse du point en utilisant la donnée membre `x`. De la même manière, on pourra accéder directement à l'ordonnée du point en utilisant la donnée membre `y`. Le type d'accès aux membres (`private`, `public`, ...) est toujours défini au niveau d'une classe et non d'un objet.

Question 7. Compléter les fichiers `Point.cpp` et `Point.h` fournis afin d'implémenter la méthode `affiche()`.

On doit aussi pouvoir calculer la **distance** entre 2 points et le point **milieu** de 2 points.

Si un objet (ici `Point`) doit être passé en **argument** d'une fonction ou d'une méthode, il est recommandé de faire un **passage par référence** car cela assure de bonnes performances et évite des recopies inutiles. Si la fonction ou la méthode ne doit pas modifier l'objet reçu en argument, il faut alors faire un passage par référence **constante**.

Question 8. Compléter les fichiers `Point.cpp` et `Point.h` fournis afin d'implémenter la méthode `distance()`.

Question 9. Compléter les fichiers `Point.cpp` et `Point.h` fournis afin d'implémenter la méthode `milieu()`.

Destructeur

Le destructeur est une méthode membre appelée automatiquement lorsqu'un objet est détruit.

Question 10. Compléter les fichiers `Point.cpp` et `Point.h` fournis afin d'implémenter le destructeur de la classe `Point` qui affichera le message "Je suis le destructeur de la classe Point !" et l'adresse mémoire de l'objet en hexadécimal.

Question 11. Combien d'objets ont été détruits au total ? Vérifier.