# Programmation Orientée Objet

# TD n°4

## Exercice 1

On dispose des déclarations et définitions suivantes :

```
#include <iostream.h>

class A {
private :
  int x;
public:
  A(int);
  ~A();
};

class B {
private :
  A * pA;
  int y;
public:
  B(int, int);
  ~B();
};

class C: public B {
private :
  int z;
public:
  C(int, int, int);
  ~C();
};

C::C(int a, int b, int c) : B(a, a+b)
{
  cout << "++ C debut" << endl;
  z = c;
  cout << "++ C z = " << z << endl;
  cout << "++ C fin" << endl;
}

C::~C()
{
  cout << "-- C debut z = " << z << endl;
  cout << "-- C fin" << endl;
```

```
}

B::B(int a, int b)
{
  cout << "++ B debut" << endl;
  y = b;
  cout << "++ B y = " << y << endl;
  pA = new A(a);
  cout << "++ B fin" << endl;
}

B::~B()
{
  cout << "-- B debut y = " << y << endl;
  delete pA;
  cout << "-- B fin" << endl;
}

A::A(int a)
{
  cout << "++ A debut" << endl;
  x = a;
  cout << "++ A x = " << x << endl;
  cout << "++ A fin" << endl;
}

A::~A()
{
  cout << "-- A debut x = " << x << endl;
  cout << "-- A fin" << endl;
}

main()
{
  B to(1, 2);
  C ti(3, 4, 5);
}
```

1. Donner la sortie (trace) du programme.
2. On ajoute la fonction membre `imprimer()` dans la classe `B` et on modifie le `main()` :

```
class B {
...
public:
...
  void imprimer();
};
void B::imprimer() {
  cout << "B::imprimer: " << pA->x << " " << y << endl;
}
```

```
main()
{
  B to(1, 2);
  C ti(3, 4, 5);
  to.imprimer();  //instruction (1)
  ti.imprimer();  //instruction (2)
}
```

Quel problème va se poser ? Comment résoudre le problème ?
Donner la sortie des instructions (1) et (2) une fois le problème résolu.

3. On ajoute la fonction membre `imprimer()` dans la classe `C` :

```
class C: public B {
...
public:
...
  void imprimer();
};
void C::imprimer() {
  cout << "C::imprimer: " << pA->x << " " << y << " " << z << endl;
}
```

Quels problèmes vont se poser ? Comment les résoudre ?
Donner la sortie des instructions (1) et (2) de la question précédente une fois tous les problèmes résolus.

4. On modifie le programme principal comme suit :

```
main()
{
  B to(44, 55);
  C ti(11, 22, 33);
  B * pto;
  C * pti;
  pto = &to;
  pti = &ti;
  pto->imprimer(); // instruction (3)
  pti->imprimer(); // instruction (4)
  pto = pti;
  pto->imprimer(); // instruction (5)
}
```

Donner la sortie des instructions (3), (4) et (5).

## Exercice 2

On souhaite définir plusieurs objets graphiques tels que des cercles ou des rectangles. Ces objets seront utilisés pour représenter un "dessin", qui peut être vu comme un ensemble d'objets graphiques (Exercice 3).

1. Écrire une classe Point contenant deux coordonnées x et y (avec les getters/setters associés). Ajouter une méthode afficher pour afficher les deux coordonnées.
2. Écrivez une classe ObjetGraphique comprenant un objet Point, correspondant à l'origine de l'objet graphique, et les méthodes suivantes dont vous déterminerez le prototype :
   — afficher : affiche les coordonnées du point origine
   — perimetre

3. Écrire les classes Rectangle et Cercle héritant d'0bjetGraphique sachant qu'un rectangle est défini par son point d'origine (coin inférieur gauche) et d'un deuxième point (coin supérieur droit) et qu'un Cercle est défini par son point d'origine et un rayon.
4. Dotez les classes Rectangle et Cercle de constructeur permettant de les définir complètement.
5. Implantez les méthodes afficher et perimetre sur ces classes

**Exercice 3** On souhaite utiliser les classes créées dans l'exercice 2 pour représenter un dessin comme un ensemble d'objets graphiques. Un dessin peut être vu comme un tableau regroupant plusieurs objets graphiques. On suppose que chaque dessin regroupe au plus 5 objets graphiques.

1. Écrivez une classe Dessin permettant de stocker jusqu'à 5 objets graphiques dans un tableau.
2. Ajouter la méthode Affichage affichant l'information des objets graphiques contenus dans le dessin et la méthode perimetre retournant le périmètre total de tous les objets graphiques.
3. Ajouter un constructeur par copie.
4. Ajouter l'opérateur d'affectation.