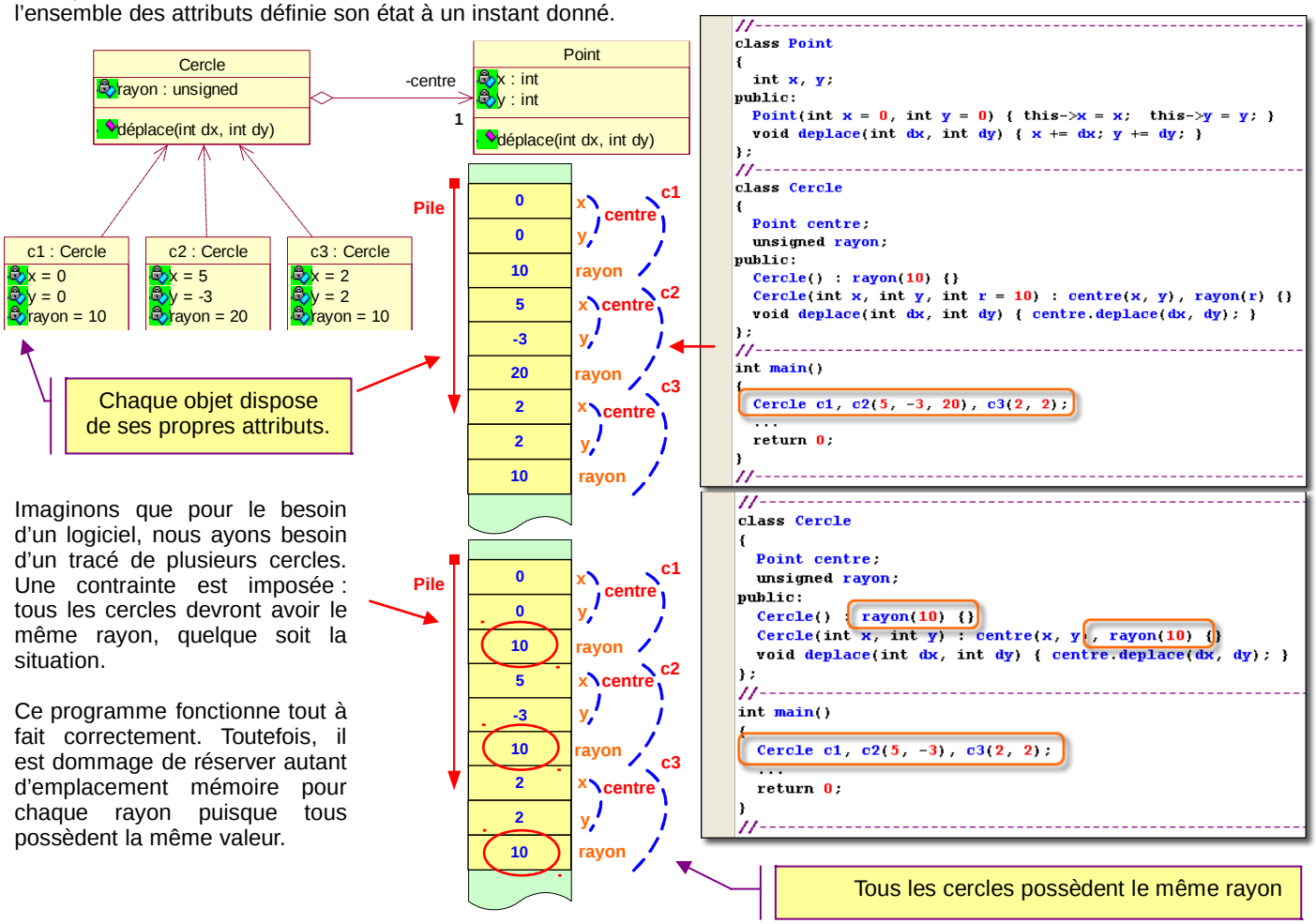
Nous avons établi que lorsque nous créons différents objets d'une même classe, chaque objet possède ses propres attributs. De façon plus précise, ces attributs sont appelés des attributs de l'objet. Effectivement, chaque objet a sa propre identité, et l'ensemble des attributs définie son état à un instant donné.

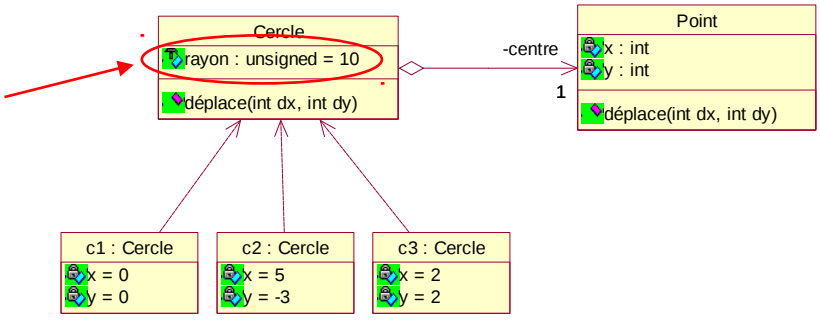


Imaginons que pour le besoin d'un logiciel, nous ayons besoin d'un tracé de plusieurs cercles. Une contrainte est imposée : tous les cercles devront avoir le même rayon, quelque soit la situation.

Ce programme fonctionne tout à fait correctement. Toutefois, il est dommage de réserver autant d'emplacement mémoire pour chaque rayon puisque tous possèdent la même valeur.

Attribut de classe

Il peut donc arriver que tous les objets d'une même classe aient envie de partager une information de telle sorte que lorsqu'un des objets modifie cette information, les autres en soient automatiquement avertis. Cette information mise en commun est également représentée par un attribut, mais qui possède la particularité d'être un attribut de la classe (même valeur pour tous les objets) et non plus un attribut d'un objet (puisque chaque objet possède normalement sa propre valeur).

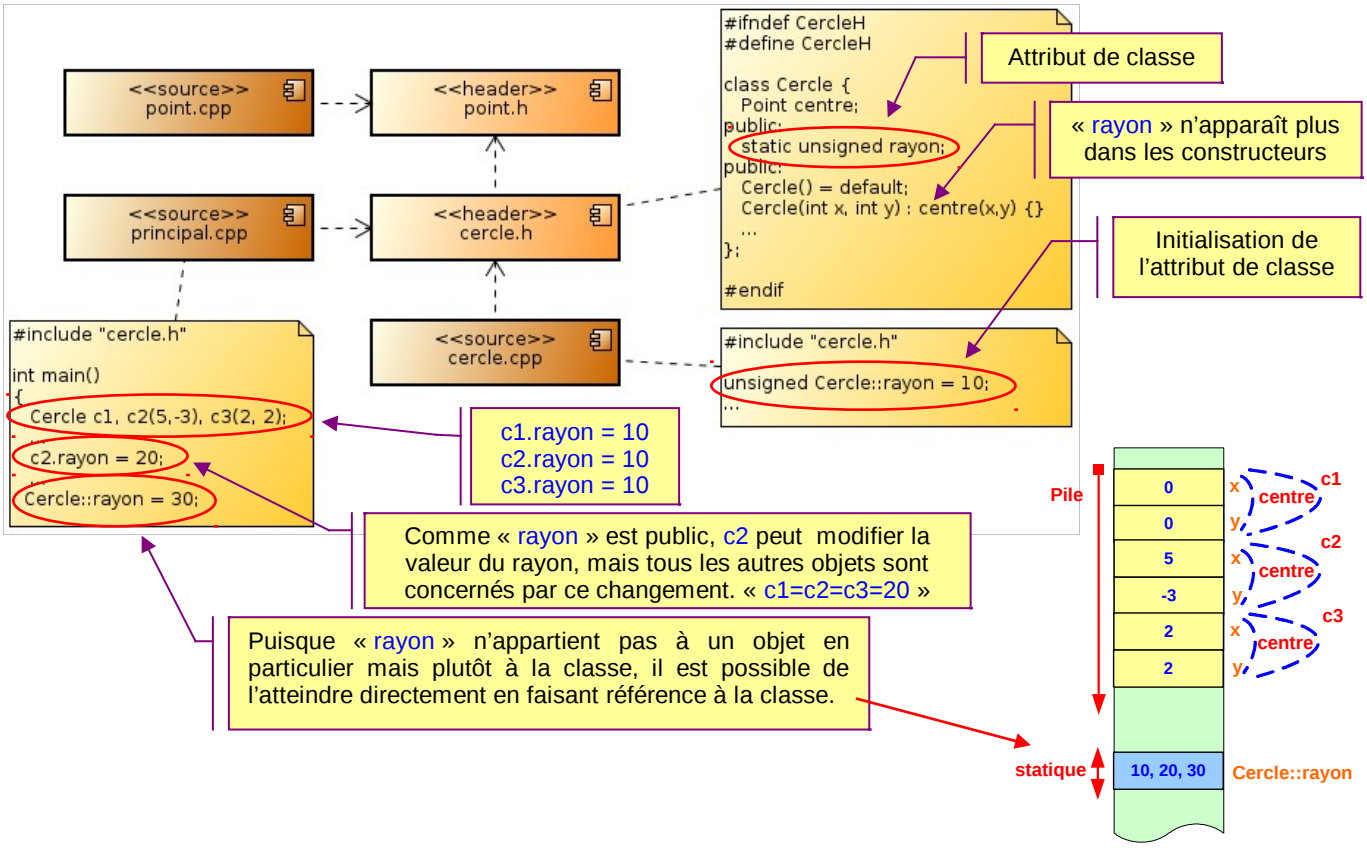


Attribut statique

Dans le langage C++, un attribut de classe correspond à une variable globale dont la portée est limitée à la classe. De ce point de vue, un attribut de classe est un attribut déclaré dans la zone statique et devra donc utiliser ce préfixe, ce qui permettra, d'ailleurs, de l'identifier. Attention, puisqu'il s'agit d'une variable globale (limitée à la portée de la classe), cette variable doit être impérativement initialisée. A ce sujet, il n'existe pas, dans ce cas là, de valeurs par défaut. Par ailleurs, il s'agit d'une définition et non pas d'une déclaration, il ne faut donc pas que cette initialisation se situe dans un fichier en-tête.

Pour finir, il ne s'agit pas d'un attribut d'un objet, en ce sens, ce n'est donc pas au moment de la construction que cette initialisation doit s'opérer puisque un objet peut déjà être créé au moment où nous sollicitons une deuxième création. Il faut donc que cette initialisation soit établie au moment du lancement du programme comme pour les variables globales. Justement, il s'agit d'un attribut déclaré dans la zone statique, cet attribut aura donc une durée de vie correspondant à la durée de vie du programme.

Cette définition s'effectuera donc dans le fichier « cpp » représentant la classe. Pour cette définition, Il faudra qualifié l'attribut par la classe concernée, mais sans spécifier le préfixe « static ».



Dans cette situation, nous utilisons beaucoup moins de mémoire. Par ailleurs, tous les objets sont au courant de la moindre modification d'un attribut statique. Dans cet exemple « rayon » était accessible directement, ce qui est rarement le cas. Il vaut mieux passer par les méthodes.

A ce sujet, les méthodes peuvent également accéder aux attributs statiques comme pour tous les autres attributs. D'ailleurs, le constructeur peut tout à fait intervenir et modifier un attribut de classe en sachant toutefois que l'initialisation a déjà eu lieu au moment du lancement du programme et qu'il s'agit, dans ce cas là, que d'une évolution.

A titre d'exemple, nous allons rajouter un attribut de classe qui permet de comptabiliser le nombre de cercle présent dans l'interface graphique. Par ailleurs tous les rayons devront conserver la valeur de 10 pixels.

```
//-----------------------------------------------------------
class Cercle
{
  Point centre;                    "rayon" n'est plus modifiable
  static const unsigned rayon;
  static unsigned nombre;
public:
  Cercle(){ nombre++; }            Evolution de l'attribut "nombre"
  Cercle(int x, int y) : centre(x, y) { nombre++; }
  ...
};
//-----------------------------------------------------------
const unsigned Cercle::rayon = 10;
unsigned Cercle::nombre = 0;       Initialisation des attributs de classe
//-----------------------------------------------------------
int main()
{
  Cercle c1, c2(5, -3), c3(2, 2);
  ...
  return 0;                        Après création de ces trois objets "Cercle::nombre = 3"
}
//-----------------------------------------------------------
```

Méthodes statiques

Nous venons de voir que chaque attribut de classe existe en un seul exemplaire indépendamment des objets de leur classe. D'une manière analogue, on peut imaginer que certaines méthodes aient un rôle totalement indépendant d'un quelconque objet. On peut certes toujours appeler une telle méthode en la faisant porter artificiellement sur un objet de la classe, et ce, bien que cet objet ne soit pas utile par rapport à la méthode. En fait, il est possible de rendre les choses plus lisibles et plus efficaces en déclarant « static » la méthode concernée.

Dans ce cas là, son appel ne nécessite plus que le nom de la classe correspondante, accompagnée, naturellement, de l'opérateur de résolution de portée. Ceci dit, elle peut être utilisée comme une méthode classique. Attention, vu cette situation, une méthode statique peut être utilisée en dehors de toute création d'objet de la classe. Cela veut dire qu'une méthode statique n'a pas de pointeur « this » et qu'il est donc ***absolument interdit*** d'accéder à un attribut non statique, puisque ces derniers sont connectés implicitement grâce à ce pointeur. La seule possibilité pour ces méthodes, c'est de ne travailler qu'avec des attributs statiques.

```
//-----------------------------------------------------------
class Cercle
{
  Point centre;
  static const unsigned rayon;
  static unsigned nombre;
public:
  Cercle(){ nombre++; }
  Cercle(int x, int y) : centre(x, y) { nombre++; }
  static unsigned getNombre() { return nombre; }
};
//-----------------------------------------------------------
const unsigned Cercle::rayon = 10;
unsigned Cercle::nombre = 0;       Méthode statique qui récupère une valeur d'un attribut statique
//-----------------------------------------------------------
int main()                         On peut passer par un objet ou utiliser directement la classe
{
  Cercle c1, c2(5, -3), c3(2, 2);
  ...
  unsigned nombreCercle = c3.getNombre();
  nombreCercle = Cercle::getNombre();
  ...
```