Chapitre 6

Méthodes et attributs de classe (statiques) ou d'instances

1 Attributs et méthodes d'instances

Définition 6.1 (instance)**.** *Une instance d'une classe est un objet de cette classe ayant été créé en mémoire.*

Autrement dit, une instance est un exemplaire en particulier fabriqué selon le modèle de la classe. Une instance est identifiée par une référence. Elle possède son propre état par rapport aux autres instances de la même classe.

Par défaut, les attributs et les méthodes apparaissant dans la définition d'une classe sont dit *d'instance*. Cela signifie que pour affecter une valeur à un tel attribut, ou pour invoquer une telle méthode, il est nécessaire de disposer d'une instance de la classe, et que ces manipulations s'appliquent à cette instance et à elle seule.

Autrement dit, si je dispose par exemple de deux instances distinctes $c1$ et $c2$ de NombreComplexe, les manipulations effectuées sur $c1$ n'ont pas d'effet sur $c2$ et *vice-versa*. Bien que $c1$ et $c2$ soient du même type, et possèdent donc par définition des attributs et méthodes identiques, leur état est bien distinct.

Vocabulaire. On dit aussi que ces attributs et méthodes sont *attachés* à leurs instances.

2 Attributs et méthodes de classe

Il est possible pourtant de définir des attributs et des méthodes qui ne sont pas attachés aux instances d'une classe mais qui sont attachés à la classe elle même. Ces attributs et méthodes sont alors partagés par toutes les instances de la classe, c'est à dire que ce sont *physiquement* les mêmes pour toutes les instances de la classe.

Un attribut ou une méthode est dit *de classe* quand il (ou elle) n'est pas attaché(e) aux instances mais à la classe elle-même.

Vocabulaire. Les attributs et méthodes de classe sont aussi appelés *statiques*.

Contrairement aux attributs d'instance, les valeurs des attributs de classe sont les mêmes pour toutes les instances.

Il n'est pas nécessaire (ni interdit) d'instancier une classe pour pouvoir affecter des valeurs à ses attributs statiques. Il n'est pas non plus nécessaire (ni interdit) d'instancier une classe pour pouvoir invoquer ses méthodes statiques. Par contre, ces manipulations auront des effets sur *toutes les instances* de la classe s'il en existe.

On peut désigner un attribut statique par la syntaxe

NomClasse.nomAttribut

(au lieu de

NomInstance.nomAttribut

qui reste pourtant autorisée), et invoquer une méthode statique par la syntaxe

NomClasse.nomMethode()

(au lieu de

NomInstance.nomMethode()

également autorisée).

2.1 Exemples d'attributs et de méthodes de classe

Nous avons déjà rencontré des attributs ou des méthodes que l'on pouvait utiliser sans avoir créé auparavant une instance de leur classe.

- Math.PI est un attribut de la classe Java Math qui contient la valeur de π. On peut l'utiliser sans faire de new Math().
- De même, on calcule la racine carrée de x par Math.sqrt(x) (sqrt() est une méthode statique de Math) sans avoir à instancier Math.
- Ecran.afficher() est une méthode de la classe Ecran utilisée pour afficher du texte sur la console. On l'invoque sans faire de new Ecran().

2.2 Notation

Par défaut, les attributs et méthodes définis dans une classe sont d'instance. Pour dire qu'un attribut ou une méthode doit être de classe et pas d'instance en PDL++, on fera précéder sa définition du mot clé statique. C'est la même chose en Java, mais avec le mot static.

Remarque. En vraie notation UML, les attributs ou les méthodes statiques sont soulignés. On préférera préciser statique entre parenthèses après avoir déclaré ces éléments.

3 Zone d'allocation dynamique et zone d'allocation statique

On peut considérer que les attributs et les méthodes de classe sont placés dans une zone de la mémoire différente de celle où sont enregistrées les instances elles mêmes. Les attributs et méthodes de classe sont enregistrés « une bonne fois pour toutes » dans une zone d'allocation statique, tandis que les attributs et les méthodes d'instance sont répliqués sur chaque instance, dans une zone d'allocation dynamique.

Exemple. On considère une classe A dont le modèle est le suivant

A
entier statA {attribut statique} *entier* dynN1 {attribut d'instance} *entier* dynN2 {attribut d'instance}
. . .

et un code qui instancie trois instances $a1$, $a2$ et $a3$ de la classe A. On peut représenter cela par le schéma de la figure 3.

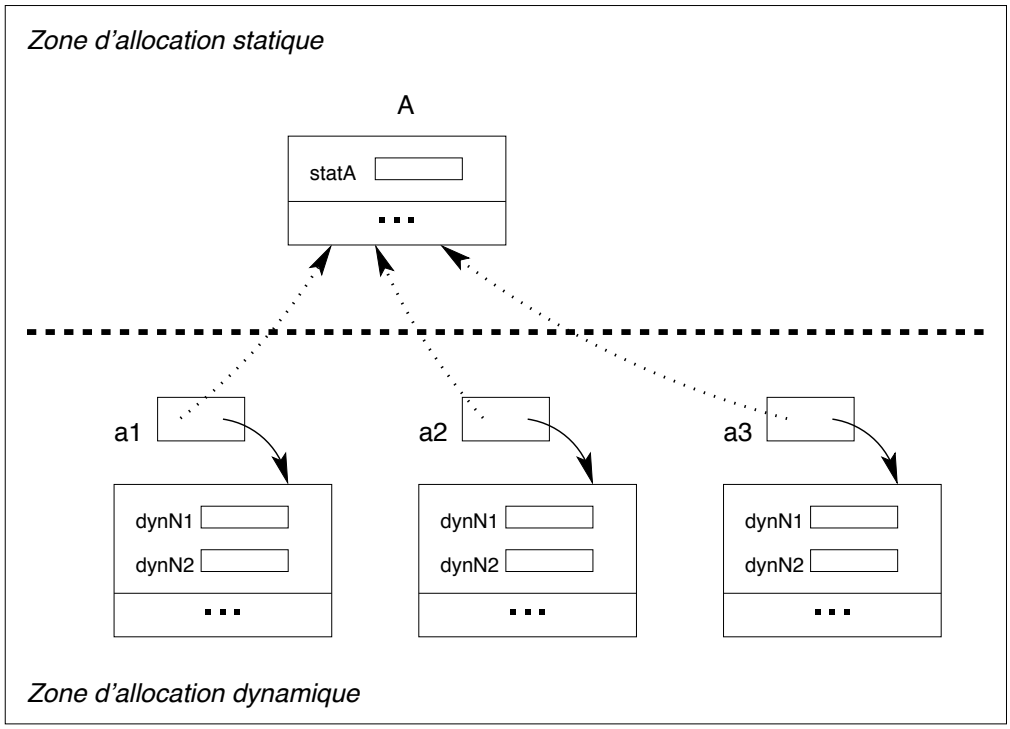

FIG. 6.1 – Emplacement des attributs statiques et dynamiques

L'attribut statA est bien unique et partagé par toutes les instances de A. Les attributs dynN1 et dynN2 sont dupliqués en autant d'exemplaires qu'il y'a d'instances de la classe A.

N.B. Ceci est une représentation simplifiée. En réalité, $a1$, $a2$ et $a3$ ne contiennent pas *deux* références, mais désignent indirectement la partie partagée statique par leur typage : elles sont de la classe A.

Cloisonnement entre zone statique et zone dynamique. Cette représentation de la mémoire montre clairement que la zone dynamique n'est pas accessible depuis la zone statique. Par exemple, l'expression A.dynN1 = 5 n'a pas de sens car on ne sait pas de quelle instance est l'attribut dynN1 dont on parle.

En pratique, cela se traduit par l'interdiction de faire référence à un attribut ou une méthode d'instance dans un contexte statique. Le message d'erreur java correspondant est par exemple

non-static variable xxx cannot be referenced from a static context

où xxx serait une variable d'instance que l'on tenterait d'utiliser dans une méthode statique (par exemple main()).

L'accès en sens inverse ne pose pas de problèmes.

4 À quoi servent les attributs et méthodes statiques ?

4.1 Attributs statiques, constantes

Les attributs statiques servent en général à définir :
- des constantes
- des variables partagées par l'ensemble des instances d'une classe

Exemple.

```
classe A {
    statique entier nbInstancesDeA ;

    Constructeur()
    debut
        nbInstancesDeA ← nbInstancesDeA + 1 ;
    fin
}
```

Dans cet exemple, nbInstancesDeA est un attribut statique de la classe A. Toutes les instances de A ont pour cet attribut la même valeur. Comme cet attribut est incrémenté (par le constructeur) à chaque instanciation de la classe A, il contient, de manière partagée par toutes les instances de A, le nombre d'objets de la classe A qui ont été instanciés.

Constantes. Les constantes dans une classe sont précédées en Java du mot clé final. En PDL++, on les repère par le mot constante. Une variable final ne peut être affectée qu'une seule fois.

Lorsque la valeur d'une constante ne dépend pas d'une instance particulière d'une classe (c'est le cas par exemple pour la constante π), elle est en plus déclarée statique.

Exemple. Dans la classe Java Math, la constante PI est déclarée par

```
public static final double PI ;
```

Si une constante peut avoir des valeurs différentes selon les instances, elle n'est alors pas *statique* (mais elle reste *final* pour chacune des instances). Exemple : la capacité (attribut Java .length) d'un tableau.

Initialisation des constantes. Comme une constante ne peut être affectée qu'une seule fois, il n'y a que deux endroits où cette affectation (qui est aussi l'initialisation de la variable) peut être réalisée :

- directement « en dur » dans le code au moment de sa déclaration,
- ou dans les constructeurs.

En pratique, les constantes d'instance sont initialisées dans les constructeurs et les constantes statiques sont initialisées en dur dans le code.

Notons que si une constante a été initialisée en dur, les constructeurs n'ont plus le droit de lui affecter une valeur.

4.2 Méthodes de classe

Elles définissent des méthodes d'usage général liées au thème de la classe, mais ne dépendant pas d'une valeur en particulier de ses attributs. Il s'agit souvent de méthodes utilitaires. L'avantage est qu'on peut les invoquer sans instancier la classe.

Exemple. La méthode sin(...) de la classe Java Math permet de calculer le sinus d'un nombre fourni en paramètre. Le calcul ne dépend que de la valeur du paramètre, et ne dépend de la valeur d'aucun attribut éventuel de la classe Math.

```
public static double sin(double a)
```

Remarque. Quand une méthode d'une classe ne fait pas référence à ses attributs, il faut :

1. se demander si elle est une méthode utile, et bien positionnée dans cette classe
2. si oui, la déclarer statique.

4.3 Pourquoi le main() est-il statique ?

Parce que c'est par là que l'interpréteur Java commence l'exécution d'un programme. Si le main() était une méthode de classe, il faudrait d'abord instancier cette classe avant de pouvoir l'utiliser. Or où cela pourrait-il être fait puisque rien n'est exécuté avant le main() ?