Chapitre 4

Les constructeurs

1 Notion de constructeur

Rappel.
- Une classe possède des *attributs* et des *méthodes*.
- Quand on instancie un objet au moyen de l'opérateur `creer`, un espace mémoire est attribué à l'objet.

Une question se pose : « À quelles valeurs sont initialisés les attributs des objets, et surtout, par quel mécanisme ? »

Réponse. À chaque fois que l'on fait un `creer` pour instancier un objet, la construction de l'objet en mémoire est assurée par un *constructeur*.

Définition 4.1 (Constructeur)**.** *Un constructeur est une méthode particulière d'une classe, chargée de construire un objet en mémoire lors de son instanciation.*

Un constructeur définit notamment les valeurs initiales des attributs. Si des attributs ne sont pas de type primitif, son rôle est également d'instancier ces attributs.

Ainsi, le fait de réaliser une instanciation d'un objet avec `creer` a pour effet d'invoquer sur lui cette méthode particulière de l'objet qu'est son constructeur.

Exemple.

```
NombreComplexe c ;
c ← creer NombreComplexe() ;
```

La deuxième ligne invoque le constructeur sur l'objet c.

Dans une classe, un constructeur peut être *implicite* ou *explicite*.

2 Constructeur implicite

Un constructeur est *implicite* lorsque sa définition n'apparaît pas textuellement dans la définition de la classe (= il n'a pas été défini par la personne qui a écrit la classe).

Exemple. Nous n'avons pas encore défini de constructeur pour la classe `NombreComplexe`. `NombreComplexe` possède donc un constructeur implicite.

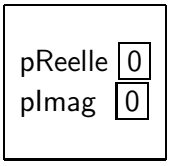
Ainsi, même quand aucun constructeur n'a été explicitement rédigé dans une classe, la classe en possède un malgré tout qui est appelé chaque fois qu'un objet est instancié au moyen de `creer`.

En java, un constructeur implicite a pour effet d'initialiser chaque attribut à une « valeur nulle ». Une « valeur nulle » a le sens suivant selon le type de l'attribut :
- entier : 0,
- réel : 0.0,
- booléen : constante `faux` (`false` en Java),
- caractère : caractère nul (constante '0000'),
- type objet : constante `nul` (`null` en Java).

Ainsi, le résultat de

```
c ← creer NombreComplexe() ;
```

est que c existe en mémoire, avec une partie réelle et une partie imaginaire nulles :



3 Doter une classe de constructeurs explicites

Si on veut qu'un objet soit créé en mémoire avec des valeurs non nulles pour ses attributs, il est nécessaire de rédiger *explicitement* un constructeur dans la classe.

Un constructeur explicite permet de :
- décrire le code chargé d'initialiser (voire d'instancier) les attributs,
- recevoir sous forme de paramètres des valeurs à affecter aux attributs à la création de l'objet.

Autrement dit, un constructeur explicite permet à l'utilisateur d'un objet de paramétrer la façon dont il souhaite voir l'objet se construire au moment de l'appel à `creer`.

Exemple. Pour créer un nombre complexe c de partie réelle 5 et de partie imaginaire 8, on écrit actuellement :

```
NombreComplexe c ;
c ← creer NombreComplexe() ;
c.pReelle ← 5 ;
c.pImag ← 8 ;
```

Il serait plus simple d'écrire :

```
NombreComplexe c ;
```

```
c ← creer NombreComplexe(5,8) ;
```

Cela est réalisable en dotant la classe `NombreComplexe` d'un constructeur explicite qui accepte en paramètres la partie réelle et la partie imaginaire du nombre complexe à construire.

3.1 Syntaxe

Définition 4.2 (Constructeur explicite)**.** *Un* constructeur explicite *est une méthode particulière d'une classe qui ne possède pas de type de retour (pas même* `rien`*, ou* `void` *en Java).*

C'est cette méthode qui sera invoquée lors de la construction de l'objet avec `creer`.

N.B. En notation PDL++, on utilisera le nom Constructeur(...) comme nom de méthode pour un constructeur. **Attention.** En Java, un constructeur ne se nomme pas Constructor(...), mais il possède *exactement le même nom* que la classe.

UML. On rajoute une case entre les attributs et les méthodes au modèle UML pour représenter un constructeur explicite.

Par exemple, avec la classe `NombreComplexe` :

NombreComplexe
réel pReelle réel pImag
Constructeur(réel pReelleInit, réel pImagInit)
réel module() rien conjugue()

Ce modèle indique que la classe NombreComplexe est dotée d'un constructeur (la méthode Constructeur(...)) explicite à deux paramètres réels.

PDL++. La classe est définie ainsi en PDL++ :

```
classe NombreComplexe {
    reel pReelle ;
    reel pImag ;

    Constructeur(réel pReelleInit, réel pImagInit)
        pReelle ← pReelleInit ;
        pImag ← pImagInit ;
    fin Constructeur

    réel module()
    ... code ...

    rien conjugue()
    ... code ...
}
```

Maintenant, on peut créer le nombre complexe `c` au moyen de

```
c ← creer NombreComplexe(5,8) ;
```

> **Remarque. TRÈS IMPORTANT.** Le fait d'ajouter un constructeur explicite à une classe a pour conséquence que le constructeur implicite n'existe plus.

Dans notre exemple, on ne peut maintenant plus construire un nombre complexe c avec :

```
c ← creer NombreComplexe() ;
```

Il est nécessaire de donner les valeurs des deux paramètres.

3.2 Instanciation des attributs objets

En plus de définir des valeurs initiales pour les attributs de type primitif, le second rôle des constructeurs (explicites) est d'instancier les attributs objets.

Exemple. Pour la classe CoupleComplexe vue au chapitre précédent, le constructeur pourrait être :

```
Constructeur()
debut
c1 ← créer NombreComplexe(1,1) ;
c2 ← créer NombreComplexe(1,1) ;
fin
```

4 Doter une classe de plusieurs constructeurs

4.1 Surcharger les constructeurs

On aura souvent envie d'avoir le choix entre plusieurs manières de construire un objet : soit en précisant des paramètres, soit en adoptant une construction par défaut fournie par l'objet.

Un objet doit alors être doté de plusieurs constructeurs pour pouvoir répondre à cette demande. Tout comme avec les méthodes, il est possible de définir *plusieurs* constructeurs pour une seule classe : il suffit pour cela de *surcharger* les constructeurs.

Par exemple, on fixe arbitrairement qu'un nombre complexe a par défaut une partie réelle de 1 et une partie imaginaire de 1. Un utilisateur d'un objet NombreComplexe choisira au moment de la création si le nombre complexe par défaut lui suffit, ou s'il veut un autre nombre complexe dont il indiquera par des paramètres la partie réelle et la partie imaginaire. Il faut alors doter la classe `NombreComplexe` de deux constructeurs : l'un sans paramètre pour construire le nombre complexe par défaut, et l'autre avec deux paramètres (partie réelle et partie imaginaire).

Le modèle de cette classe devient donc :

NombreComplexe
réel pReelle réel pImag
Constructeur() *Constructeur(réel pReelleInit, réel pImagInit)*
réel module() rien conjugue()

La règle pour surcharger les constructeurs est la même qu'avec les autres méthodes : chaque constructeur doit avoir une signature distincte de celles des autres.

C'est bien le cas des deux constructeurs de `NombreComplexe` où le premier constructeur a pour signature Constructeur() et le second Constructeur(réel, réel).

On n'a pas le droit de rajouter un autre constructeur où on définirait d'abord la partie imaginaire puis la partie réelle comme dans

~~Constructeur(réel pImagInit, réel pReelleInit)~~

car la classe posséderait deux constructeurs de même signature (`Constructeur (réel, réel)`).

Remarque. Le fait que l'ajout d'un constructeur explicite « invalide » le constructeur implicite doit inciter à systématiquement doter une classe d'un constructeur sans paramètre, sauf si cela n'a pas de sens, afin qu'une création sans préciser de paramètre soit toujours possible.

4.2 Invoquer un constructeur depuis un autre constructeur

Dans son code, un constructeur peut profiter du travail qu'effectue un autre constructeur en l'invoquant. En PDL++, on invoque un autre constructeur par le mot Constructeur(...) en précisant les valeurs des paramètres.

Exemple. Le code des deux constructeurs de la classe NombreComplexe est actuellement le suivant :

```
Constructeur(réel pReelleInit, réel pImagInit)
  pReelle ← pReelleInit ;
  pImag ← pImagInit ;
  fin
```

```
Constructeur()
  pReelle ← 1 ;
  pImag ← 1 ;
  fin
```

Le constructeur sans paramètre pourrait fournir les valeurs 1 et 1 en paramètres au constructeur avec deux paramètres :

```
Constructeur()
  Constructeur(1,1) ;
fin
```

Ce mécanisme est soumis aux restrictions suivantes :
- l'invocation d'un constructeur ne peut se faire que depuis un autre constructeur, c'est à dire qu'une méthode (non constructeur) n'a pas le droit d'invoquer un constructeur ;
- un autre constructeur ne peut être invoqué au maximum qu'une seule fois par constructeur ;
- cet appel est nécessairement la première des instructions du constructeur.

Attention : la syntaxe Java est différente. En Java, on invoque un autre constructeur en le nommant this(). Le code Java des constructeurs de notre exemple s'écrit donc :

```
NombreComplexe(double pReelleInit, double pImagInit) {
  pReelle = pReelleInit ;
  pImag = pImagInit ;
}

NombreComplexe() {
  this(1,1) ;
}
```