Chapitre 3

Instancier et utiliser des objets

1 Programmer avec des objets

On a vu qu'une classe permet de définir des objets. L'esprit de la programmation par objets est le suivant :

- on fait la liste des objets utiles à la résolution du problème pour lequel on veut programmer une stratégie de résolution ;
- si certains de ces objets ont déjà été définis (par exemple par la bibliothèque du langage de programmation) $\Rightarrow$ on les réutilise ;
- on définit au moyen d'une classe les objets qui n'existent pas déjà ;
- on écrit un programme principal qui résout le problème en faisant interagir ces objets.

Une classe n'est que la description générique des objets que l'on souhaite utiliser. C'est en quelque sorte le moule à partir duquel on fabrique les objets. Le processus de fabrication d'un objet à partir de ce moule s'appelle *l'instanciation*, ou encore la création de l'objet. Par exemple, à partir de la classe NombreComplexe, on peut créer deux nombres complexes distincts $c1 = i$ et $c2 = 1 - i$. $c1$ et $c2$ sont deux *instances* de la classe NombreComplexe. On dit aussi que ce sont deux *objets* NombreComplexe.

2 Type primitif vs. type Objet

En définissant une classe, on définit un *type*, qui peut être utilisé pour typer des variables. Un tel type est appelé *type objet*, et n'est pas de même nature que les types de base d'un langage de programmation, que l'on appelle *types primitifs* dans les langages à Objets.

Les types primitifs en langage algorithmique sont :

- entier
- réel
- booléen

– caractère

Les langages de programmation déclinent en général les types entiers et réels en plusieurs versions, qui varient par la taille des nombres qu'ils peuvent contenir, ou par leur précision.

Les types primitifs du langage Java sont :
– byte, short, int, long *(types entiers)*
– float, double *(types réels)*
– char *(type caractère)*
– boolean *(type booléen)*

Tous les autres types, qu'ils soient fournis par le langage de programmation (comme String en java) ou définis au moyen d'une nouvelle classe, sont des types objet.

2.1 Syntaxe de la déclaration d'une variable

On déclare les variables d'un type primitif et celles d'un type objet de la même manière : en associant le nom de la variable à son type.

Langage algorithmique. nom_variable : nom_type
exemple :
entier n (variable n de type primitif)
NombreComplexe c1 (variable $c1$ de type objet)

Langage Java. nom_type nom_variable exemple :
int n
NombreComplexe c1

2.2 Effet en mémoire des déclarations

Type primitif Quand on déclare une variable de type primitif, de la place dans la mémoire de l'ordinateur est automatiquement réservée pour y stocker des valeurs du type de la variable.

Exemple. La déclaration
entier n
réserve une « case »mémoire nommée n et pouvant recevoir des valeurs entières :

Type objet Les types objets sont d'une nature différente : ils peuvent contenir plusieurs valeurs simultanément (une par attribut), et ils possèdent des méthodes qui permettent de les manipuler.

Quand on déclare une variable de type Objet, aucune place n'est réservée en mémoire pour mémoriser le contenu de cet objet.

Exemple.
NombreComplexe c1
ne réserve pas de case mémoire nommée $c1$. $c1$ devient juste un nom réservé pour un objet de type NombreComplexe qui n'existe pas encore physiquement en mémoire.

Pour que de la place mémoire soit allouée à un objet, il est nécessaire de *l'instancier*.

2.3 Instanciation d'un objet

L'instanciation d'un objet correspond à la création physique (i.e. en mémoire) de l'objet. Autrement dit, cela réserve un espace en mémoire pour chaque attribut de type primitif.

Langage algorithmique. L'instanciation d'un objet s'obtient au moyen de l'opérateur créer appliqué au nom du type avec des parenthèses :

nom_variable ← créer nom_type()

Exemple.

c1 ← créer NombreComplexe()

Langage Java. L'opérateur d'instanciation est new en Java, appliqué au nom du type avec des parenthèses :

nom_variable = new nom_type()

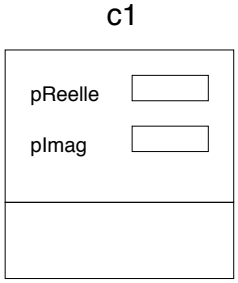
Exemple.

c1 = new NombreComplexe()

N.B. on peut réaliser en Java la déclaration et l'instanciation en une seule ligne :

NombreComplexe c1 = new NombreComplexe()

L'objet n'existe physiquement en mémoire qu'après cette opération.



Remarque. Par convention, un objet déclaré mais non instancié aura pour valeur nul (null en java).

3 Utilisation d'un Objet

Une fois l'objet créé, il peut être utilisé. L'utilisation peut se faire en
- affectant des valeurs à ses attributs (autorisé mais déconseillé, comme nous le verrons dans un autre cours),
- *invoquant* des méthodes de l'objet (utilisation conseillée).

Remarque. Toute tentative d'utilisation d'un objet *nul*, c'est à dire non créé, se traduit par une erreur à l'exécution du programme.

Attributs Un attribut d'un objet créé est désigné par la syntaxe : nom_objet.nom_attribut.

Exemple. L'attribut plmag de l'objet de type NombreComplexe nommé $c1$ est désigné par :

```
c1.plmag
```

et on peut lui affecter une valeur (mais c'est en principe déconseillé) :

```
c1.plmag ← 3 ;
```

ou encore l'afficher (idem)

```
afficher(c1.plmag) ;
```

Méthodes Lorsqu'on applique une méthode à un objet, on dit qu'on *invoque* une méthode sur l'objet. La syntaxe est : nom_objet.nom_methode(valeur_param_1, . . . , valeur_param_n).

Exemple. Invocation de la méthode module() sur l'objet $c1$ (et affectation du résultat dans une variable réelle nommée m) :

```
m ← c1.module() ;
```

Un exemple complet.

```
Programme Principal()
debut
NombreComplexe c1 ;
c1 ← créer NombreComplexe() ;
c1.pReelle ← 5 ;
c1.plmag ← 5 ;
afficher("Le module de c1 est : ", c1.module()) ;
NombreComplexe c2
c2 ← créer NombreComplexe() ;
c2.pReelle ← c1.plmag ;
c2.plmag ← c1.pReelle ;
c2.conjugue() ;
...
fin
```

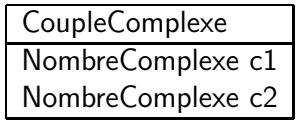
N.B. On verra plus tard que si l'objet a été prévu pour, il est possible de fournir des valeurs en paramètre directement à la création de l'objet pour indiquer dans quel état construire l'objet (voir le cours sur les constructeurs).

4 Compléments sur l'instanciation

4.1 Et si les attributs ne sont pas d'un type primitif ?

Le mécanisme d'instanciation doit alors être appliqué aussi aux attributs d'un type non primitif, pour que l'espace dédié à l'attribut existe en mémoire.

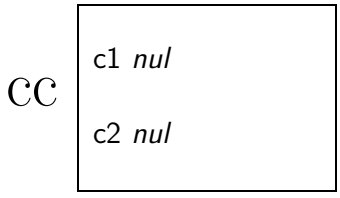
Exemple. On considère un objet CoupleComplexe qui représente un couple de nombres complexes, pour un problème mathématique.



Ses attributs sont $c1$ et $c2$, tous deux de type NombreComplexe. On ne se préoccupe pas des méthodes de cet objet pour l'exemple. Les instructions

CoupleComplexe cc
cc ← creer CoupleComplexe()

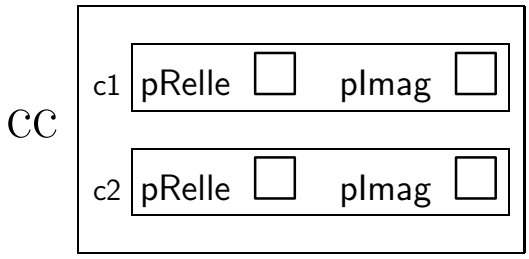
créent en mémoire :



Les attributs n'ont pas été alloués, ils ont pour valeur nul. Il faut faire en plus

cc.c1 ← creer NombreComplexe()
cc.c2 ← creer NombreComplexe()

pour avoir



En théorie, la procédure est donc :
– instancier l'objet
– instancier chaque attribut de l'objet qui est d'un type objet
Mais comme les attributs de type objet peuvent eux mêmes avoir des attributs non primitifs, on peut devoir pousser plus loin la procédure !

En pratique, la question se pose heureusement très rarement dans ces termes. En effet, on verra au chapitre suivant qu'un mécanisme de *constructeur* permet à chaque objet de s'assurer lui-même de sa bonne construction.

4.2 Cas particulier des chaînes et des tableaux

Chaînes. En java, le type String (chaîne de caractères) n'est pas un type primitif. Il devrait donc théoriquement être nécessaire d'instancier une variable de type String avant de l'utiliser. En pratique, ce n'est pas nécessaire. Pour la commodité des programmeurs, les objets de type String sont automatiquement instanciés par Java. On adoptera la même souplesse en langage algorithmique.

Tableaux. Pour instancier un tableau, il faut préciser le nombre d'éléments qu'il pourra contenir. En langage algorithmique, la syntaxe d'instanciation d'un tableau est la suivante :

nom_tableau ← creer type_tableau[nombre_elements]

En java, c'est la même chose en remplaçant "←" par "=" et "creer" par "new".

Exemple. Un tableau tab de 20 entiers

algo

tableau d'entier tab ; *{déclaration}*
tab ← creer entier[20] ; *{instanciation}*

Java

int[] tab ; *// déclaration*
tab = new int[20] ; *//instanciation*