

Chapitre 2

Les classes en Java

2.1 LES MODULES, LES CLASSES, L'ENCAPSULATION

La notion de module existe dans la plupart des langages non-objet. La définition générale d'un module est "une unité faisant partie d'un ensemble".

2.1.1 La notion de module et de type abstrait de données (TAD)

En programmation non-objet (Pascal, C), un module est un ensemble de fonctions traitant des données communes. Les éléments (constantes, variables, types, fonctions) déclarés dans la partie interface sont accessibles de l'extérieur du module, et sont utilisables dans un autre programme (un autre module ou un programme principal). Il suffit de référencer le module pour avoir accès aux éléments de sa partie interface. Celle-ci doit être la plus réduite possible, tout en donnant au futur utilisateur un large éventail de possibilités d'utilisation du module. Les déclarations de variables doivent être évitées au maximum. On peut toujours définir une variable locale au module à laquelle on accède ou que l'on modifie par des appels de fonctions de l'interface.

On parle alors d'*encapsulation* des données qui sont invisibles pour l'utilisateur du module et seulement accessibles à travers un jeu de fonctions. L'utilisateur du module n'a pas besoin de savoir comment sont mémorisées les données ; le module est pour lui un *type abstrait de données (TAD)*. Du reste, cette mémorisation locale peut évoluer, elle n'affectera pas les programmes des utilisateurs du module dès lors que les prototypes des fonctions d'interface restent inchangés.

En langage C, module.h constitue le fichier d'en-tête à inclure dans chaque fichier référençant des fonctions du module. Le corps du module est défini dans module.c.

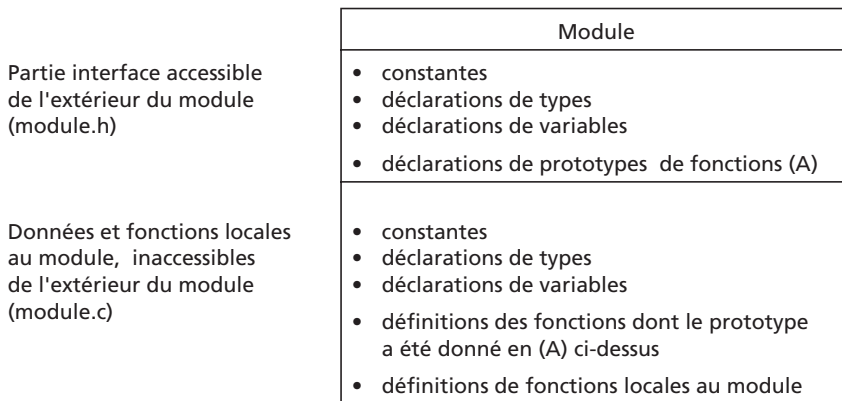


Figure 2.1 — La notion de module séparant ce qui est accessible de l'extérieur du module de ce qui est local au module, aussi bien pour les données que pour les fonctions.

Un exemple classique est celui de la pile. S'il n'y a qu'une pile à gérer, celle-ci peut être encapsulée dans les données locales (dans pile.c) ; la déclaration du fichier d'en-tête pourrait être la suivante en langage C.

```
// pile.h version en langage C gérant une seule pile d'entiers
#ifndef PILE_H
#define PILE_H

void initPile (int max); // initialiser la pile
int pileVide (); // la pile est-elle vide ?
void empiler (int valeur); // empiler une valeur
int depiler (int* valeur); // dépiler à l'adresse de valeur
void viderPile (); // vider la pile
void listerPile (); // lister les éléments de la pile

#endif
```

Si on veut gérer plusieurs piles, les données ne peuvent plus être dans la partie "données locales" du module. Celles-ci doivent être déclarées dans le programme appelant et passées en paramètres des fonctions du module gérant la pile. Le fichier d'en-tête pourrait s'écrire comme indiqué ci-dessous. La déclaration du type Pile doit être faite dans l'interface ; par contre les variables max, sommet et element ne devraient pas être accessibles de l'extérieur du module ce qui n'est pas le cas sur l'exemple suivant où le pointeur de pile donne accès aux composantes de la structure à partir du programme appelant.

```
// pile.h version en langage C gérant plusieurs piles d'entiers
#ifndef PILE_H
#define PILE_H

typedef struct {
    int max; // nombre max d'éléments dans la pile
    int sommet; // repère le dernier occupé de element
```

```

    int* element; // tableau d'entiers alloués dynamiquement
} Pile;

void initPile (Pile* p, int max);
int pileVide (Pile* p);
void empiler (Pile* p, int valeur);
int depiler (Pile* p, int* valeur);
void viderPile (Pile* p);
void listerPile (Pile* p);

#endif

```

2.1.2 La notion de classe

Une classe est une extension de la notion de module. Les données et les fonctions traitant les données sont réunies ensemble dans une classe qui constitue un nouveau type. On peut déclarer des variables du type de cette classe. Cela revient à avoir la possibilité de dupliquer les modules en leur attribuant des noms différents, chaque module ayant ses propres variables locales visibles ou non de l'extérieur.

L'exemple de la pile peut être schématisé comme indiqué sur la figure 2.2 qui suit les conventions de la modélisation objet UML (Unified Modeling Language). On distingue les données de la classe appelées **attributs** en PO (Programmation Objet) et les fonctions appelées **méthodes**. Un signe moins(-) devant un élément indique un élément privé, visible seulement dans la classe ; un signe plus(+) indique un élément public, accessible par tous les utilisateurs de la classe ; l'absence de signe indique une visibilité (voir page 67) seulement pour les classes du paquetage (du répertoire). Les attributs `sommet` et `element`, et la méthode `erreur()` sont privés ; les autres méthodes sont publiques.

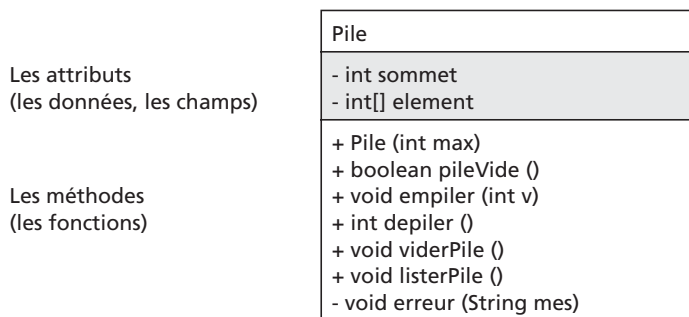


Figure 2.2 — La classe Pile (-indique un attribut ou une méthode privé).

La classe est un modèle qui s'apparente à la déclaration d'un nouveau type de données. On peut déclarer une variable (on parle d'**objet** ou d'**instance de la classe** en PO) du type de cette classe.

Exemple :

```

Pile p1 = new Pile (5); // p1 et p2 sont des instances de la classe Pile
Pile p2 = new Pile (100); // p1 et p2 sont des objets de type Pile

```