Chapitre 1

Introduction au langage Java

Le langage Java est un langage généraliste de programmation synthétisant les principaux langages existants lors de sa création en 1995 par *Sun Microsystems*. Il permet une programmation orientée-objet (à l'instar de SmallTalk et, dans une moindre mesure, C++), modulaire (langage ADA) et reprend une syntaxe très proche de celle du langage C.

Outre son orientation objet, le langage Java a l'avantage d'être **modulaire** (on peut écrire des portions de code génériques, c-à-d utilisables par plusieurs applications), **rigoureux** (la plupart des erreurs se produisent à la compilation et non à l'exécution) et **portable** (un même programme compilé peut s'exécuter sur différents environnements). En contre-partie, les applications Java ont le défaut d'être plus lentes à l'exécution que des applications programmées en C par exemple.

1.1 Environnement Java

Java est un langage interprété, ce qui signifie qu'un programme compilé n'est pas directement exécutable par le système d'exploitation mais il doit être interprété par un autre programme, qu'on appelle interpréteur. La figure 1.1 illustre ce fonctionnement.

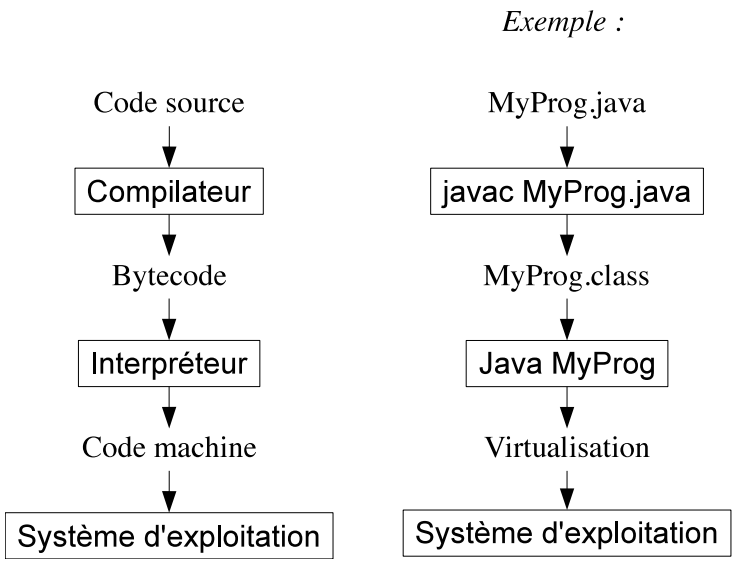

FIGURE 1.1 – Interprétation du langage

Un programmeur Java écrit son code source, sous la forme de classes, dans des fichiers dont l'extension est `.java`. Ce code source est alors compilé par le compilateur `javac` en un langage appelé *bytecode* et enregistre le résultat dans un fichier dont l'extension est `.class`. Le bytecode

ainsi obtenu n'est pas directement utilisable. Il doit être interprété par la *machine virtuelle* de Java qui transforme alors le code compilé en code machine compréhensible par le système d'exploitation. C'est la raison pour laquelle Java est un langage portable : le bytecode reste le même quelque soit l'environnement d'exécution.

En 2009, Sun Microsystems est racheté par Oracle Corporation qui fournit dorénavant les outils de développement Java SE (*Standard Edition*) contenus dans le *Java Development Kit* (JDK). Au moment où est écrit ce livret, la dernière version stable est le JDK 1.7.

1.1.1 Compilation

La compilation s'effectue par la commande `javac` suivie d'un ou plusieurs nom de fichiers contenant le code source de classes Java. Par exemple, `javac MyProg.java` compile la classe `MyProg` dont le code source est situé dans le fichier `MyProg.java`. La compilation nécessite souvent la précision de certains paramètres pour s'effectuer correctement, notamment lorsque le code source fait référence à certaines classes situées dans d'autres répertoires que celui du code compilé. Il faut alors ajouter l'option `-classpath` suivie des répertoires (séparés par un ; sous Windows et : sous Unix) des classes référencées. Par exemple :

```
javac -classpath /prog/exos1:/cours MyProg.java
```

compilera le fichier `MyProg.java` si celui-ci fait référence à d'autres classes situées dans les répertoires `/prog/exos1` et `/cours`. Le résultat de cette compilation est un fichier nommé `MyProg.class` contenant le bytecode correspondant au source compilé. Ce fichier est créé par défaut dans le répertoire où la compilation s'est produite. Il est cependant fortement souhaitable de ne pas mélanger les fichiers contenant le code source et ceux contenant le bytecode. Un répertoire de destination où sera créé le fichier `MyProg.class` peut être précisé par l'option *-d*, par exemple :

```
javac -d /prog/exos1 -classpath /cours MyProg.java
```

1.1.2 Interprétation

Le bytecode obtenu par compilation ne peut être exécuté qu'à l'aide de l'interpréteur. L'exécution s'effectue par la commande `java` suivie du nom de la classe à exécuter (sans l'extension `.class`). Comme lors de la compilation, il se peut que des classes d'autres répertoires soient nécessaires. Il faut alors utiliser l'option `-classpath` comme dans l'exemple qui suit :

```
java -classpath /prog/exos1:/cours MyProg
```

1.2 Programmation orientée-objet

Chaque langage de programmation appartient à une "famille" de langages définissant une approche ou une méthodologie générale de programmation. Par exemple, le langage C est un langage de programmation *procédurale* car il suppose que le programmeur s'intéresse en priorité aux *traitements* que son programme devra effectuer. Un programmeur C commencera par identifier ces traitements pour écrire les fonctions qui les réalisent sur des données prises comme paramètres d'entrée.

La programmation *orientée-objet* (introduite par le langage SmallTalk) propose une méthodologie centrée sur les données. Le programmeur Java va d'abord identifier un ensemble d'**objets**, tel que chaque objet représente un élément qui doit être utilisé ou manipulé par le programme, sous la forme d'ensembles de données. Ce n'est que dans un deuxième temps, que le programmeur va écrire les traitements, **en associant chaque traitement à un objet donné**. Un objet peut

être vu comme une entité regroupant un ensemble de données et de **méthodes** (l'équivalent d'une fonction en C) de traitement.

1.2.1 Classe

Un objet est une variable (presque) comme les autres. Il faut notamment qu'il soit déclaré avec son type. Le type d'un objet est un type complexe (par opposition aux types primitifs entier, caractère, …) qu'on appelle une **classe**.

Une classe regroupe un ensemble de données (qui peuvent être des variables primitives ou des objets) et un ensemble de méthodes de traitement de ces données et/ou de données extérieures à la classe. On parle d'**encapsulation** pour désigner le regroupement de données dans une classe.

Par exemple, une classe *Rectangle* utilisée pour instancier des objets représentant des rectangles, encapsule 4 entiers : la longueur et la largeur du rectangle ainsi que la position en abscisse et en ordonnée de l'origine du rectangle (par exemple, le coin en haut à gauche). On peut alors imaginer que la classe *Rectangle* implémente une méthode permettant de déplacer le rectangle qui nécessite en entrée deux entiers indiquant la distance de déplacement en abscisse et en ordonnée. L'accès aux positions de l'origine du rectangle se fait directement (i.e. sans passage de paramètre) lorsque les données sont encapsulées dans la classe où est définie la méthode. Un exemple, écrit en Java, de la classe *Rectangle* est donné ci-dessous :

```
class Rectangle {

    int longueur;
    int largeur;
    int origine_x;
    int origine_y;

    void deplace(int x, int y) {
        this.origine_x = this.origine_x + x;
        this.origine_y = this.origine_y + y;
    }

    int surface() {
        return this.longueur * this.largeur;
    }
}
```

Pour écrire un programme avec un langage orienté-objet, le programmeur écrit uniquement des classes correspondant aux objets de son système. Les traitements à effectuer sont programmés dans les méthodes de ces classes qui peuvent faire appel à des méthodes d'autres classes. En général, on définit une classe, dite "exécutable", dont une méthode peut être appelée pour exécuter le programme.

Encapsulation

Lors de la conception d'un programme orienté-objet, le programmeur doit identifier les objets et les données appartenant à chaque objet mais aussi des droits d'accès qu'ont les autres objets sur ces données. L'encapsulation de données dans un objet permet de cacher ou non leur existence aux autres objets du programme. Une donnée peut être déclarée en accès :

- **public** : les autres objets peuvent accéder à la valeur de cette donnée ainsi que la modifier ;

- **privé** : les autres objets n'ont pas le droit d'accéder directement à la valeur de cette donnée (ni de la modifier). En revanche, ils peuvent le faire indirectement par des méthodes de l'objet concerné (si celles-ci existent en accès public).

Les différents droits d'accès utilisables en Java sont détaillés dans la section 3.2.1.

Méthode constructeur

Chaque classe doit définir une ou plusieurs méthodes particulières appelées des **constructeurs**. Un **constructeur** est une méthode invoquée lors de la création d'un objet. Cette méthode, qui peut être vide, effectue les opérations nécessaires à l'initialisation d'un objet. Chaque constructeur doit avoir le même nom que la classe où il est défini et n'a aucune valeur de retour (c'est l'objet créé qui est renvoyé). Dans l'exemple précédent de la classe rectangle, le constructeur initialise la valeur des données encapsulées :

```
class Rectangle {
    ...
    Rectangle(int lon, int lar) {
        this.longueur = lon;
        this.largeur = lar;
        this.origine_x = 0;
        this.origine_y = 0;
    }
    ...
}
```

Plusieurs constructeurs peuvent être définis s'ils acceptent des paramètres d'entrée différents.

1.2.2 Objet

Instanciation

Un objet est une instance (anglicisme signifiant « cas » ou « exemple ») d'une classe et est référencé par une variable ayant un état (ou valeur). Pour créer un objet, il est nécessaire de déclarer une variable dont le type est la classe à instancier, puis de faire appel à un constructeur de cette classe. L'exemple ci-dessous illustre la création d'un objet de classe `Cercle` en Java :

```
Cercle mon_rond;
mon_rond = new Cercle();
```

L'usage de parenthèses à l'initialisation du vecteur, montre qu'une méthode est appelée pour l'instanciation. Cette méthode est un constructeur de la classe `Cercle`. Si le constructeur appelé nécessite des paramètres d'entrée, ceux-ci doivent être précisés entre ces parenthèses (comme lors d'un appel classique de méthode). L'instanciation d'un objet de la classe `Rectangle` faisant appel au constructeur donné en exemple ci-dessous pourra s'écrire :

```
Rectangle mon_rectangle = new Rectangle(15,5);
```

Accès aux variables et aux méthodes

Pour accéder à une variable associée à un objet, il faut préciser l'objet qui la contient. Le symbole '`.`' sert à séparer l'identificateur de l'objet de l'identificateur de la variable. Une copie de la longueur d'un rectangle dans un entier `temp` s'écrit :

```
int temp = mon_rectangle.longueur;
```

La même syntaxe est utilisée pour appeler une méthode d'un objet. Par exemple :

```
mon_rectangle.deplace(10,-3);
```

Pour qu'un tel appel soit possible, il faut que trois conditions soient remplies :

1. La variable ou la méthode appelée existe !
2. Une variable désignant l'objet visé existe et soit instanciée.
3. L'objet, au sein duquel est fait cet appel, ait le droit d'accéder à la méthode ou à la variable (cf. section 3.2.1).

Pour référencer l'objet "courant" (celui dans lequel se situe la ligne de code), le langage Java fournit le mot-clé `this`. Celui-ci n'a pas besoin d'être instancié et s'utilise comme une variable désignant l'objet courant. Le mot-clé `this` est également utilisé pour faire appel à un constructeur de l'objet courant. Ces deux utilisations possibles de `this` sont illustrées dans l'exemple suivant :

```
class Carre {

    int cote;
    int origine_x;
    int origine_y;

    Carre(int cote, int x, int y) {
        this.cote = cote;
        this.origine_x = x;
        this.origine_y = y;
    }

    Carre(int cote) {
        this(cote, 0, 0);
    }
}
```

Chapitre 2

Syntaxe du langage

Le langage C a servi de base pour la syntaxe du langage Java :

- le caractère de fin d'une instruction est " `;` "

```
a = c + c;
```

- les commentaires (non traités par le compilateur) se situent entre les symboles "`/*`" et "`*/`" ou commencent par le symbole "`//`" en se terminant à la fin de la ligne

```
int a; // ce commentaire tient sur une ligne
int b;
```

ou

```
/*Ce commentaire nécessite
 2 lignes*/
int a;
```

- les identificateurs de variables ou de méthodes acceptent les caractères {a..z}, {A..Z}, `$`, _ ainsi que les caractères {0..9} s'ils ne sont pas le premier caractère de l'identificateur. Il faut évidemment que l'identificateur ne soit pas un mot réservé du langage (comme `int` ou `for`).
  Ex : `mon_entier` et `ok4all` sont des identificateurs valides mais
  `mon-entier` et `4all` ne sont pas valides pour des identificateurs.

2.1 Types de données

2.1.1 Types primitifs

En plus de ces types primitifs, le terme `void` est utilisé pour spécifier le retour vide ou une absence de paramètres d'une méthode. On peut remarquer que chaque type primitif possède une classe qui encapsule un attribut du type primitif. Par exemple, la classe `Integer` encapsule un attribut de type `int` et permet ainsi d'effectuer des opérations de traitement et des manipulations qui seraient impossibles sur une simple variable de type `int`.

A l'inverse du langage C, Java est un langage très rigoureux sur le typage des données. Il est interdit d'affecter à une variable la valeur d'une variable d'un type différent si cette seconde variable n'est pas explicitement transformée. Par exemple :

Type primitifs de données en Java

Type	Classe éq.	Valeurs	Portée	Défaut
`boolean`	`Boolean`	`true` ou `false`	N/A	`false`
`byte`	`Byte`	entier signé	{-128..128}	o
`char`	`Character`	caractère	{\uoooo..\uFFFF}	\uoooo
`short`	`Short`	entier signé	{-32768..32767}	0
`int`	`Integer`	entier signé	{-2147483648..2147483647}	0
`long`	`Long`	entier signé	$\{-2^{31}..2^{31}-1\}$	0
`float`	`Float`	réel signé	$\{-3, 4028234^{38}..3, 4028234^{38}\}$ $\{-1, 40239846^{-45}..1, 40239846^{-45}\}$	0.0
`double`	`Double`	réel signé	$\{-1, 797693134^{308}..1, 797693134^{308}\}$ $\{-4, 94065645^{-324}..4, 94065645^{-324}\}$	0.0

```
int a;
double b = 5.0;
a = b;
```

est interdit et doit être écrit de la manière suivante :

```
int a ;
double b = 5.0 ;
a = (int)b ;
```

2.1.2 Tableaux et matrices

Une variable est déclarée comme un tableau dès lors que des crochets sont présents soit après son type, soit après son identificateur. Les deux syntaxes suivantes sont acceptées pour déclarer un tableau d'entiers (même si la première, non autorisée en C, est plus intuitive) :

```
int[] mon_tableau;
int mon_tableau2[];
```

Un tableau a toujours une taille fixe qui doit être précisée avant l'affectation de valeurs à ses indices, de la manière suivante :

```
int[] mon_tableau = new int[20];
```

De plus, la taille de ce tableau est disponible dans une variable `length` appartenant au tableau et accessible par `mon_tableau.length`. On peut également créer des matrices ou des tableaux à plusieurs dimensions en multipliant les crochets (ex : `int[][] ma_matrice;`). À l'instar du C, on accède aux éléments d'un tableau en précisant un indice entre crochets (`mon_tableau[3]` est le quatrième entier du tableau) et un tableau de taille `n` stocke ses éléments à des indices allant de `0` à `n-1`.

2.1.3 Chaînes de caractères

Les chaînes de caractères ne sont pas considérées en Java comme un type primitif ou comme un tableau. On utilise une classe particulière, nommée `String`, fournie dans le package `java.lang`. Les variables de type String ont les caractéristiques suivantes :

- leur valeur ne peut pas être modifiée
- on peut utiliser l'opérateur + pour concaténer deux chaînes de caractères :

```
String s1 = ”hello”;
String s2 = ”world”;
String s3 = s1 + ” ” + s2;
//Après ces instructions s3 vaut ”hello world”
```

- l'initialisation d'une chaîne de caractères s'écrit :

```
String s = new String(); //pour une chaine vide
String s2 = new String(”hello world”);
// pour une chaîne de valeur ”hello world”
```

- un ensemble de méthodes de la classe `java.lang.String` permettent d'effectuer des opérations ou des tests sur une chaîne de caractères .