Chapitre 1

Premier programme

1.1 Objectif du cours

Les objectifs du cours sont d'apprendre les bases de la programmation en utilisant le langage Java comme support. Ces bases sont pratiquement communes avec l'écrasante majorité des langages de programmation et nous éviterons d'utiliser ce qui est trop spécifique au langage Java.

Les notions étudiées seront les suivantes :

— variable, type, expression
— l'instruction conditionnelle (if)
— les boucles (for, while)
— les tableaux
— les sous-programmes (méthode, fonction, procédure)
— en conclusion du cours, une ouverture vers la programmation objet avec l'utilisation d'objets de deux classes prédéfinies : String et ArrayList.

Le but du cours est de vous donner non seulement des connaissances, mais également une compétence : celle de réaliser concrètement de petits programmes java utilisant les constructions énumérées ci-dessus et de les exécuter sur un ordinateur.

1.2 Programme

Il existe toutes sortes de programmes qui peuvent s'exécuter sur différents dispositifs pour rendre un service. Nous n'allons pas essayer de définir ce qu'est un programme en général, mais présenter le type de programmes que nous allons écrire dans ce cours.

Il s'agit de programmes applicatifs s'exécutant sur un ordinateur et composés d'une suite d'instructions. Le programme utilise des données dont certaines sont inscrites dans le programme, dans des instructions et d'autres viennent de l'extérieur : elles sont tapées au clavier par l'utilisateur.

Les programmes auront des résultats qui s'afficheront à l'écran.

1.3 Programme source et programme cible

Chaque programme a deux formes différentes : une forme qui est le programme tel qu'il est écrit par le programmeur et l'autre est le programme tel qu'il est exécuté par l'ordinateur. Ces deux formes sont différentes : le programme que l'on écrit a une forme textuelle. On peut le lire : il contient des

mots-clés et des noms qui ont un sens. Mais l'ordinateur ne sait pas exécuter directement ce texte. Le programme qui s'exécute a une forme binaire : la fameuse suite de 0 et de 1 qui est copiée dans la mémoire de l'ordinateur.

1.3.1 Traduction : compilation et interprétation

Pour passer d'une forme à l'autre, il faut une traduction qui est exécutée automatiquement par un programme. Il existe deux sortes de traduction : la traduction réalisée une fois pour toute par un programme appelé un **compilateur** ou la traduction réalisée à chaque exécution par un logiciel appelé **interpréteur**.

Le compilateur est analogue à la traduction d'un livre alors que l'interpréteur est analogue à la traduction simultanée utilisée pour traduire en direct les propos tenus dans une langue étrangère.

Le langage Java que nous utilisons mixe les deux procédés avec utilisation d'un compilateur puis d'un interpréteur.

Dans cette histoire, le langage des programmes écrits par les programmeurs s'appelle le langage source de l'ordinateur. C'est le langage Java qui est unique pour tous les ordinateurs du monde. Le langage des programmes exécutés s'appelle le langage cible et il est différent pour chaque type de processeur, d'architecture et de système d'exploitation employé.

Dans le cas de Java, il y a un troisième langage qui est celui des programmes compilés par le compilateur java. On l'appelle langage intermédiaire, langage de la machine virtuelle Java. Il est unique pour tous les ordinateurs mais les machines ne savent pas l'exécuter directement. C'est pour cela qu'il y a besoin d'un interpréteur qui interprète le code intermédiaire : la machine virtuelle.

1.3.2 Etapes pour la création d'un programme

La création de programmes nécessite l'utilisation de différents outils qui peuvent soit être des outils séparés, soit des outils regroupés dans un unique logiciel appelé environnement de développement intégré (IDE en anglais).

Il y a au minimum trois outils : un éditeur de texte, un compilateur et un interprèteur.

L'éditeur de texte permet de créer le fichier contenant le programme source. Il s'agit d'une sorte de traitement de texte mais qui ne comporte pas d'élément de mise en page car le but n'est pas d'imprimer un texte. Le format de fichier utilisé est le format texte brut, celui des fichiers .txt en windows. Dans le cas des programmes sources java, on n'utilise pas cette extension .txt mais l'extension .java. Cette extension doit être utilisée quel que soit le système d'exploitation (windows, Mac OS, Linux).

Le compilateur prend en entrée le fichier texte contenant le programme source. Si le programme source est correct, le compilateur crée un ou plusieurs fichiers binaires contenant le programme en langage intermédiaire. Ces fichiers ont l'extension .class.

L'interpréteur prend en entrée un fichier contenant du code intermédiaire et exécute le code correspondant. Pour ce faire, il traduit une ligne de code, puis l'exécute immédiatement avant de passer à la suivante.

Pour ce cours, nous vous conseillons d'utiliser un environnement intégré, avec deux possibilités :

— utiliser un environnement pour débutant : drjava. Très facile à utiliser, il vous premettra de démarrer rapidement et sans tracas. Mais il se révèlera limité par la suite. Il suffit amplement pour le cours NFA031.

— utiliser un environnement professionnel : Eclipse. Il est plus difficile à prendre en main. Il a de nombreuses fonctionnalités.

Il existe d'autres outils professionnels comme par exemple Netbeans ou intelliJ. Vous pouvez les utiliser si vous le souhaitez, mais ne vous fournirons pas d'aide pour eux.

1.4 Premier programme

1.4.1 Texte du programme

Le programme a pour but de calculer et afficher la conversion en dollars d'une somme en euros saisie au clavier.

```
public class Conversion {
    public static void main (String[] args) {
        double euros;
        double dollars;
        System.out.println("Somme␣en␣euros?␣");
        euros = Terminal.lireDouble();
        dollars = euros *1.118;
        System.out.println("La␣somme␣en␣dollars:␣");
        System.out.println(dollars);
    }
}
```

Ce texte doit être tapé dans un fichier appelé Conversion.java. On n'a pas le choix du nom : ce nom est celui qui apparaît après le mot-clé **class** sur la première ligne du programme, suivi de l'extension **.java**.

1.4.2 Structure générale d'un programme Java

Ce programme a un squelette identique à tout autre programme Java. Le voici :

Listing 1.1 – (pas de lien)

```
public class ... {
    public static void main (String[] args) {
      ....
    }
}
```

— Tout programme Java est composé au minimum d'une *classe* (mot-clé `class`) dite *principale*, qui elle même, contient une *méthode* de nom `main`. Les notions de *classe* et de *méthode* seront abordées plus tard.

— **public, class, static, void** : sont des mots reservés c'est-à-dire qu'ils ont un sens particuliers pour Java. On ne peut donc pas les utiliser comme nom pour des classes, des variables, etc...

— La **classe principale** : celle qui contient la méthode `main`. Elle est déclarée par

public class *Nom_classe*

C'est vous qui choisissez le nom de la classe.
Le code qui définit une classe est délimité par les caractères `{` et `}`

— **Nom du programme :** Le nom de la classe principale donne son nom au programme tout entier et doit être également celui du fichier contenant le programme, complété de l'extension `.java`
— La **méthode** `main` : obligatoire dans tout programme Java : c'est elle qui "commande" l'exécution. Définie par une suite de déclarations et d'actions délimitées par `{` et `}`. Pour l'instant, et jusqu'à ce que l'on sache définir des sous-programmes, c'est ici que vous écrirez vos programmes.
— *Par convention,* le nom d'une classe commence par une majuscule. Ce n'est pas une obligation absolue, mais il est fortement conseillé de s'y conformer.

Autrement dit, pour écrire un programme que vous voulez appeler `"Truc"`, ouvrez un fichier dans un éditeur de texte, appelez ce fichier `"Truc.java"` ; Ecrivez dans ce fichier le squelette donné plus haut, puis remplissez les ...

1.4.3 comprendre le code

Voyons ligne par ligne le sens des instructions du programme.

`double euros;` déclarer le nom euros comme un nom utilisé pour désigner un nombre à virgule qu'on ne connaît pas au moment où l'on écrit le programme. Lors de l'exécution du programme, viendra un moment où ce nombre sera connu. On appelle **variable** un tel nom. Le mot-clé `double` désigne en Java une approximation des nombres réels codée sur 8 octets.

`double dollars;` déclarer la variable appelée dollars destinée à contenir un nombre à virgule.

`System.out.println("Somme en euros? ");` sert à afficher à l'écran le texte donné entre guillemmets (`Somme en euros?: `). Les guillemets ne sont pas affichés, seulement ce qu'il y a dedans. Cette instruction est un appel d'une méthode prédéfinie appelée System.out.println.

`euros = Terminal.lireDouble();` est une instruction qui sert à deux choses :

— `Terminal.lireDouble()` est un appel de méthode qui permet de lire un nombre à virgule au clavier.
— le reste de l'instruction sert à associer le nom `euros` et le nombre lu au clavier. A partir de l'exécution de cette ligne, le nom euros désigne ce nombre.

`dollars = euros * 1.118;`

— `euros * 1.118` décrit un calcul, à savoir la multiplication du nombre désigné par le nom euros et le nombre 1.118 (un nombre à virgule en notation américaine où le point remplace la virgule). Le signe * désigne la multiplication.
— le reste de l'instruction sert à associer le nom dollars au résultat du calcul. A partir de l'exécution de cette ligne, dollars désigne ce résultat.

`System.out.println("La somme en dollars: ");` affiche le texte donné entre guillemets. Cela ressemble au texte de la ligne `System.out.println("Somme en euros? ");` sauf que c'est la méthode System.out.print qui est appelée au lieu de System.out.println. Ces deux méthodes servent à afficher du texte à l'écran, cependant la seconde passe à la ligne après cet affichage mais pas la première.

`System.out.println(dollars);` affiche à l'écran le nombre à virgule associé au nom dollars. Notez qu'il n'y a pas de guillemets ici. S'il y avait des guillemets, ce serait le mot dollars qui s'afficherait et pas le nombre associé à cette variable.

1.5 Compilation et exécution du programme

1.5.1 Compilation du programme

Lors d'une première étape, le programme a été tapé et sauvé dans un fichier texte appelé Conversion.java. Il faut ensuite compiler ce programme.

Il est possible d'appeler le compilateur en ligne de commande dans un terminal ou une fenêtre de commandes windows. Cette commande sera la suivante :

```
javac Conversion.java
```

Si le programme est correct, la commande se termine sans aucun affichage. Un fichier de code intermédiaire ou byte-code a été créé, contenant le même programme sous forme binaire. Le fichier s'appelle Conversion.class (même nom de fichier mais extension .class au lieu de .java).

Si le programme comporte des erreurs, c'est-à-dire s'il n'est pas écrit en java correct, le compilateur affiche des messages d'erreurs qui expliquent où sont les erreurs détectées et leur nature.

Dans un environnement de programmation intégré (IDE), pour compiler le programme il faut actionner un bouton ou une option de menu. L'IDE appelle le compilateur et s'il y a des erreurs, il les surligne dans l'éditeur et affiche les messages d'erreur.

1.5.2 Erreurs de compilation

Voyons un exemple de programme comportant des erreurs.

```
public classe Bonjour {
    public static void main (String[] args) {
        System.out.println("Bonjour␣tout␣le␣monde␣!"*2);
    }
}
```

Ce code contient deux erreurs :

— *erreur de syntaxe* : le mot-clé `class` qui introduit le nom du programme a été changé en `classe`.

— *erreur de typage (sémantique)* : le message "`Bonjour tout le monde !`" est une chaîne de caractères (type `String`) et en tant que tel ne peut être multiplié par 2.

Voyons ce qui ce passe lorsqu'on compile ce programme en ligne de commande :

```
> javac Bonjour.java
Bonjour.java:1: error: class, interface, or enum expected
public classe Bonjour {
       ^
Bonjour.java:2: error: class, interface, or enum expected
    public static void main (String[] args) {
                  ^
Bonjour.java:4: error: class, interface, or enum expected
    }
    ^
3 errors
```

Il affiche trois erreurs, mais en fait ces trois erreurs n'en sont qu'une : la première.

Le message affiche d'abord le fichier où se situe l'erreur (Bonjour.java). Dans notre exemple, c'est évident, mais les gros programmes comportent plusieurs fichiers. Il est alors important de savoir dans lequel est l'erreur. Ensuite, apparaît le numéro de ligne où l'erreur a été détectée (ici, 1, 2 et 4). En troisième apparaît une courte explication en anglais (ici : class, interface, or enum expected). A la ligne suivante apparaît le code de la ligne en question avec un petit chapeau qui marque l'emplacement probable de l'erreur.

Corrigeons la première erreur et relançons la compilation.

```
> javac Bonjour.java
Bonjour.java:3: error: bad operand types for binary operator '*'
        System.out.println("Bonjour tout le monde !"*2);
                                                    ^
  first type:  String
  second type: int
1 error
```

C'est cette fois la seconde erreur qui est trouvée sur la ligne 3. Voyez que la petite marque est bien sur le signe *.

Il arrive que le message d'erreur soit erroné : le compilateur n'a pas compris quelle était l'erreur. Il arrive également que la position indiquée ne soit pas réellement celle de l'erreur. C'est l'endroit où l'erreur a été détectée par le compilateur, mais l'erreur peut se situer avant : soit plus à gauche sur la même ligne, soit sur une ligne au-dessus de cette position. Ainsi les messages d'erreur sont une aide généralement pertinente pour détecter les erreurs mais cette aide n'est pas infaillible.

Avoir des erreurs de compilation lorsqu'on commence à programmer et même par la suite, est tout-à-fait normal et ne doit pas décourager. Il faut corriger patiemment toutes les erreurs, jusqu'à compiler le programme. Si vous êtes bloqué par une erreur que vous ne comprenez pas, demandez de l'aide à vos enseignants.

1.5.3 Exécution

Pour exécuter le programme, il faut interpéter le code intermédiaire. C'est le rôle de la machine virtuelle Java aussi appelée l'environnement d'exécution java (Java Runtime Environment-JRE). Cet interpréteur peut être appelé en ligne de commande ou par un bouton ou une option de menu dans un IDE.

En ligne de commande, il faut taper le nom de la commande `java` suivi du nom du programme (ici : Conversion). Attention, ce n'est pas le nom d'un fichier qu'il faut donner, seulement le nom du programme (Bonjour et non pas Bonjour.class).

```
> java Bonjour
Bonjour tout le monde !2
```

1.6 Retour sur les instructions

Dans le programme Conversion, on a trois catégories d'instructions :

— des déclarations de variables. Les déclarations servent à donner un nom à une case mémoire, de façon a pouvoir y stocker, le temps de l'exécution du programme, des valeurs. Une fois qu'une variable est déclarée et qu'elle possède une valeur, on peut consulter sa valeur. Cette

valeur n'est pas figée, il est possible de la changer lors de l'exécution du programme par une affectation (cf. ci-dessous).

— des instructions d'entrée-sortie. Un programme a bien souvent (mais pas toujours) besoin d'informations pour lui viennent de l'extérieur. Notre programme calcule la valeur en dollars d'une somme en euros *qui est donnée au moment de l'exécution* par l'utilisateur. On a donc besoin de dire qu'il faut faire `entrer` dans le programme une donnée par le biais du clavier. Il y a en Java, comme dans tout autre langage de programmation, des instructions prédéfinies qui permettent de faire cela (aller chercher une valeur au clavier, dans un fichier, sortir du programme vers l'écran ou un fichier une valeur etc...).

— l'instruction d'affectation (=) qui permet de manipuler les variables déclarées. Elle permet de mettre la valeur de ce qui est a droite de = dans la variable nommée à gauche du =.

1.7 Les variables

Les variables sont utilisées pour stocker les données du programme. A chaque variable correspond un emplacement en *mémoire*, où la donnée est stockée, et un nom qui sert à désigner cette donnée tout au long du programme.
Une variable doit être déclarée dans le programme. On peut alors consulter sa valeur ou modifier sa valeur.

1.7.1 déclaration

Nous avons déjà vu deux déclarations : `double euros;` et `double dollars;` Il est possible de déclarer les deux variables en une seule instruction `double euros, dollars;`. C'est juste une abbréviation pour déclarer les deux variables sur la même ligne.

le nom des variables

`euros` ou `dollars` sont les noms des variables et ont donc été choisis librement par l'auteur du programme. Il y a cependant quelques contraintes dans le choix des symboles constituant les noms de variables. Les voici :

— Les noms de variables sont des *identificateurs* qui peuvent contenir des lettres, des chiffres, les signes souligné (*tiret du 8*) et dollar. Ces noms ne doivent pas commencer par un chiffre.

— Un nom de variable ne peut pas être un mot reservé : (`abstract, boolean, if, public, class, private, static`, etc).

— Certains caractères ne peuvent pas apparaître dans un nom de variable : les espaces, les signes de ponctuation et les caractères spéciaux.

— Dans un nom de variable, les majuscules et les minuscules sont significatifs. Par exemple `texte` et `TEXTE` sont deux noms différents.

Exemples : `a`, `id_a` et `X1` sont des noms de variables valides, alors que `1X` et `X-X` ne le sont pas.

le type de la variable

Dans notre exemple de déclaration `double euros;`, le mot `double` est le nom d'un type prédéfini en Java. Un type est un ensemble de valeurs particulières connues de la machine. Nous

verrons dans le prochain chapitre d'autres types : les nombres entiers, les caractères, les chaînes de caractères, les booléens.

syntaxe des déclarations de variables

Ainsi, pour déclarer une variable, il faut donner un nom de type parmi `int, double, char, boolean` ou `String` suivi d'un nom de variable que vous choisissez.

Pour déclarer plusieurs variables de même type en même temps, il faut donner un nom de type suivi des noms de vos variables (séparés par des virgules).

C'est ce qu'on appelle la *syntaxe* Java des déclarations de variables. Il faut absolument se conformer à cette règle pour déclarer des variables en Java, faute de quoi, votre code ne sera pas compris par le compilateur et produira une erreur. Ainsi, `double x;` est correct alors que `nombre x;` ou encore `x double;` seront rejetés.

Execution des déclarations de variables

L'exécution d'un programme, rappelons-le, consiste en l'exécution successive de chacune de ses instructions.

L'exécution d'une déclaration de variable consiste à réserver (on dit allouer) un espace mémoire de taille suffisante pour contenir la valeur qui sera associée au nom de la variable. Dans le cas du type double, il y aura 8 octets de réservés pour la variable euros.

1.7.2 Affecter une valeur à une variable

Une fois qu'une variable a été declarée, on peut lui donner une valeur, c'est à dire mettre une valeur dans la case mémoire qui lui est associée. Pour cela, il faut utiliser l'instruction d'affectation dont la syntaxe est

nom_variable = *expression* ;

par exemple `x=2.5;`

Attention : le symbole = utilisé dans une affectation ne désigne pas ici une égalité au sens mathématique. Il faut lire cette ligne *x reçoit 2.5* et non pas *x est égal à 2.5*. Ce n'est pas non plus un test d'égalité pour savoir si `x` est égal ou non à `2.5`.

Il est de bonne habitude de donner une valeur initiale à une variable dès qu'elle est déclarée. Java nous permet d'affecter une valeur à une variable au moment de sa déclaration :
par exemple `double x=2.5;`

Des valeurs du bon type

A droite de =, on peut mettre n'importe quelle valeur *du bon type*, y compris des variables. Comme `2.5` est une valeur du type `double`, notre exemple ne sera possible que si plus haut dans notre programme, on a la déclaration `double x;`

Les expressions

Les valeurs ne sont pas forcément simples, elles peuvent être le résultat d'un calcul. On pourrait par exemple écrire `x=(18.3+20.1)*2.5;`. On peut écrire l'expression `(18.3+20.1)*2.5` car + et $\star$ sont des *opérateurs* connus dans le langage Java, désignant comme on s'y attend l'addition et la multiplication et que ces opérateurs, appliqués à des nombres, calculent un nouveau nombre. En effet, $18.3 + 20.1$ donne 38.4 (un entier) et 38.4×2.5 donne 96.0 qui est un nombre à virgule.

Pour construire des expressions arithmétiques, c'est à dire des expressions dont le résultat est de type `int` ou `double` on peut utiliser les opérateurs :`+`, `-`, `*`, `/`

Exécution d'une affectation

L'exécution d'une affectation se fait en deux temps :

1. On calcule le résultat de l'expression **à droite** de = (on dit évaluer l'expression). Si c'est une valeur simple, il n'y a rien a faire, si c'est une variable, on va chercher sa valeur, si c'est une expression avec des opérateurs, on applique les opérateurs à leurs opérandes. Cela nous donne une valeur qui doit être du même type que la variable à gauche de =.
2. On met cette valeur dans l'emplacement mémoire associé à la variable **à gauche** de =.

C'est ce qui explique que l'on peut écrire le programme suivant qui n'a d'autre intérêt que d'illustrer l'affectation) :

```
1 public class essaiVariable {
2     public static void main (String[] args) {
3       double x;
4       double y;
5       y=2.5;
6       x=y+5.1;
7       x=x+2.3;
8     }
9 }
```

Les lignes 3 et 4 déclarent les variables `x` et `y`. Elle sont donc connues dans la suite du programme. Puis (ligne 5) la valeur 2.5 est donnée à `y`. Ensuite est calculé `y+5.1`. La valeur de `y` en ce point de l'exécution est `2.5` donc `y+5.1` vaut `7.6`. A l'issue de l'exécution de la ligne 6, `x` vaut `7.6`. Finalement, (ligne 7), on évalue `x+2.3`. `x` à ce moment vaut `7.6` donc `x+2.3` vaut `9.9`. On donne à `x` la valeur `9.9`.

Cet exemple illustre le fait qu'une variable peut (puisque l'exécution d'un programme est séquentielle) avoir plusieurs valeurs successives (d'où son nom de variable), et que l'on peut faire apparaître une même variable à gauche et à droite d'une affectation : on met à jour son contenu en tenant compte de la valeur qu'elle contient juste avant.

1.8 Les appels de méthodes prédéfinies

1.8.1 La méthode `System.out.println`

La ligne `System.out.println("Somme en euros? ");` donne l'ordre d'afficher à l'écran le message `somme en euros ?:`. Elle le fait en utilisant un programme tout fait qui existe sans qu'on ait à l'écrire. On appelle cela en Java une *méthode*. Ce programme s'appelle `System.out.println`.

Un texte entre guillemets tel que `"Somme en euros? "` s'appelle une chaîne de caractères.

Pour fonctionner, cette méthode a besoin que l'utilisateur, lorsqu'il l'utilise, lui transmette une information : la chaîne de caractères qu'il veut afficher à l'écran. Cette information transmise par l'utilisateur de la méthode est ce qui se trouve entre les parenthèses qui suivent le nom de la méthode. C'est ce qu'on appelle *l'argument* ou le *paramètre* de la méthode.
On a ici utilisé `System.out.println` pour afficher le message `Somme en euros ?` : en faisant l'appel `System.out.println("Somme en euros ? :);`
Pour afficher coucou, il faut faire l'appel `System.out.println("coucou").`

Lors d'un appel, vous devez nécessairement transmettre une et une seule chaîne de caractères de votre choix à `System.out.println` : c'est l'auteur de la méthode qui a fixé le nombre, le type et l'ordre des arguments de cette méthode, lorsqu'il l'a écrite. Ainsi on ne pourra écrire :
`System.out.println("coucou","bidule").` Cette ligne provoquerait une erreur de compilation.

1.8.2 La méthode `Terminal.lireDouble`

La ligne suivante donne l'ordre de récupérer une valeur au clavier.

```
euros=Terminal.lireDouble();
```

Cette ligne est une affectation =. On trouve à droite du = un appel de méthode.

La méthode appelée s'appelle `lireDouble` et se trouve dans la classe `Terminal`.

Pour que cette ligne fonctionne, il faut avoir copié le fichier `Terminal.java` contenant cette classe dans votre répertoire de travail, vous pourrez utiliser autant de fois que vous le désirez cette méthode. Utiliser une méthode existante dans un programme s'appelle *faire un appel* de la méthode.

Le fait qu'il n'y ait rien entre les parenthèses indique que cette méthode n'a pas besoin que l'utilisateur lui fournisse des informations. C'est une méthode sans arguments.

En revanche, le fait que cet appel se trouve à droite d'une affectation indique que le résultat de son exécution produit un résultat (celui qui sera mis dans la variable `euros`). En effet, ce résultat est la valeur provenant du clavier. C'est ce qu'on appelle la *valeur de retour* de la méthode.

Comme pour les arguments de la méthode, le fait que les méthodes retournent ou pas une valeur est fixé par l'auteur de la méthode une fois pour toutes. Il a fixé aussi le type de cette valeur de retour. Une méthode, lorsqu'elle retourne une valeur, retourne une valeur toujours du même type. Pour `lireDouble`, cette valeur est de type `double`.

Lorsqu'ils retournent un résultat, les appels de méthode peuvent figurer dans des expressions.

Exemple : La méthode `Math.min` :prends 2 arguments de type int et retourne une valeur de type `int` : le plus petit de ses deux arguments. Ainsi l'instruction `x = 3 + (Math.min(4,10) + 2);` donne à `x` la valeur `9` car `3+(4+2)` vaut `9`.
L'instruction `x = 3 + (Terminal.lireInt() + 2);` a aussi un sens. Elle s'exécutera de la façon suivante : pour calculer la valeur de droite, l'exécution se mettra en attente qu'une ligne de texte soit entrée (un curseur clignotant à l'écran indiquera cela). Une ligne de texte est une séquence de touche frappées terminée par la touche de passage à la ligne (Enter ou Return ou Entrée). Dès que l'utilisateur a entré une ligns, le calcul de `Terminal.lireInt()` se termine avec pour résultat le nombre entier présent en début de la ligne entrée. Imaginons que l'utilisateur ait pressé 6. 3+6+2 donne 11. La valeur de `x` sera donc 11.

Que se passera-t-il si l'utilisateur tape une ligne qui ne commence pas par un nombre entier ? Une erreur se produira, et l'exécution du programme tout entier sera arrêtée. Nous verrons plus tard qu'il y a un moyen de récupérer cette erreur pour relancer l'exécution.

1.9 Les méthodes prédéfinies d'entrée-sortie

Disons maintenant quelques mots sur les méthodes d'entrée-sortie.

Java est un langage qui privilégie la communication par interfaces graphiques. En particulier, la saisie des données est gérée plus facilement avec des fenêtres graphiques dédiées, des boutons, etc. Mais, pour débuter en programmation, il est beaucoup plus simple de programmer la saisie et l'affichage des données au terminal : la saisie à partir du clavier et l'affichage à l'écran. Nous vous proposons de suivre cette approche en utilisant la bibliothèque `Terminal`.

La bibliothèque `Terminal`

La classe `Terminal` (écrite par nous), regroupe les principales méthodes de saisie et d'affichage au terminal pour les types predéfinis que nous utiliserons dans ce cours : `int`, `double`, `boolean`, `char` et `String`. Le fichier source `Terminal.java`, doit se trouver présent dans le même répertoire que vos programmes. Pour l'employer, il suffit de faire appel à la méthode qui vous intéresse précédé du nom de la classe. Par exemple, `Terminal.lireInt()` renvoie le premier entier saisit au clavier.

Saisie avec `Terminal` : Se fait toujours par un appel de méthode de la forme :

`Terminal.lire`*`Type`*`();`

où *Type* est le nom du type que l'on souhaite saisir au clavier. La saisie se fait jusqu'à validation par un changement de ligne. Voici la saisie d'un `int` dans `x`, d'un `double` dans `y` et d'un `char` dans `c` :

```
int x;
double y;
char c; // Declarations
x = Terminal.lireInt();
y = Terminal.lireDouble();
c = Terminal.lireChar();
```

Méthodes de saisie dans `Terminal` :

— `Terminal.lireInt()`
— `Terminal.lireDouble()`
— `Terminal.lireChar()`
— `Terminal.lireBoolean()`
— `Terminal.lireString()`

Méthodes d'affichage :

Terminal contient également les méthodes d'affichage suivantes qui peuvent remplacer System.out.print et System.out.println.

— `Terminal.ecrireInt(n);`
— `Terminal.ecrireDouble(n);`
— `Terminal.ecrireBoolean(n);`

— `Terminal.ecrireChar(n);`
— `Terminal.ecrireString(n);`
— `Terminal.ecrireIntln(n);`
— `Terminal.ecrireDoubleln(n)` ...

Ces méthodes d'affichage sont équivalentes à `System.out.print` et `System.out.println`.

Affichage avec la bibliothèque `System`

La bibliothèque `System` propose les mêmes fonctionnalités d'affichage au terminal pour les types de base que notre bibliothèque `Terminal`. Elles sont simples d'utilisation et assez fréquentes dans les ouvrages et exemples que vous trouverez ailleurs. Nous les présentons brièvement à titre d'information.

— `System.out.print` : affiche une valeur de base ou un message qui lui est passé en paramètre.

```
System.out.print(5);   ---> affiche 5
System.out.print(bonjour);   ---> affiche le contenu de bonjour
System.out.print("bonjour");   ---> affiche bonjour
```

Lorsque l'argument passé est un message ou *chaîne de caractères*, on peut lui joindre une autre valeur ou message en utilisant l'opérateur de *concaténation* +. **Attention** si les opérandes de + sont exclusivement numériques, c'est leur addition qui est affichée !

```
System.out.print("bonjour " + 5 );   ---> affiche:  bonjour 5
System.out.print(5 + 2);   ---> affiche 7
```

— `System.out.println` : même comportement que `System.out.print` , mais avec passage à la ligne en fin d'affichage.

Passer à la ligne

Il est possible de passer à la ligne sans rien afficher avec la méthode `System.out.println()` sans paramètre ou avec la méthode `Terminal.sautDeLigne()`.