Programmation et Java

introduction – variables, expressions

P. Bakowski

bako@ieee.org

Programmation et Java

Nos objectifs pédagogiques sont :

1. Etudier les concepts de base de la programmation dans les langages de haut-niveau et les appliquer en Java,

2. S’initier a l’analyse et resolution de problèmes, via la programmation,

3. Acquerir certaines méthodes de resolution des problèmes classiques en informatique

Pourqoi Java

1. Orienté objet, exceptions, polymorphisme, gestion de la mémoire, transparence des pointeurs, ..

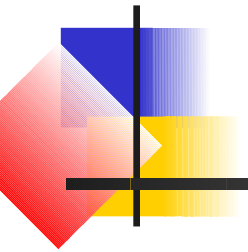
2. Fortement typé

3. Les programmes sont portables. indépendante de la plateforme d'architecture

1. Internet et le WEB

2. Programmation distribuée

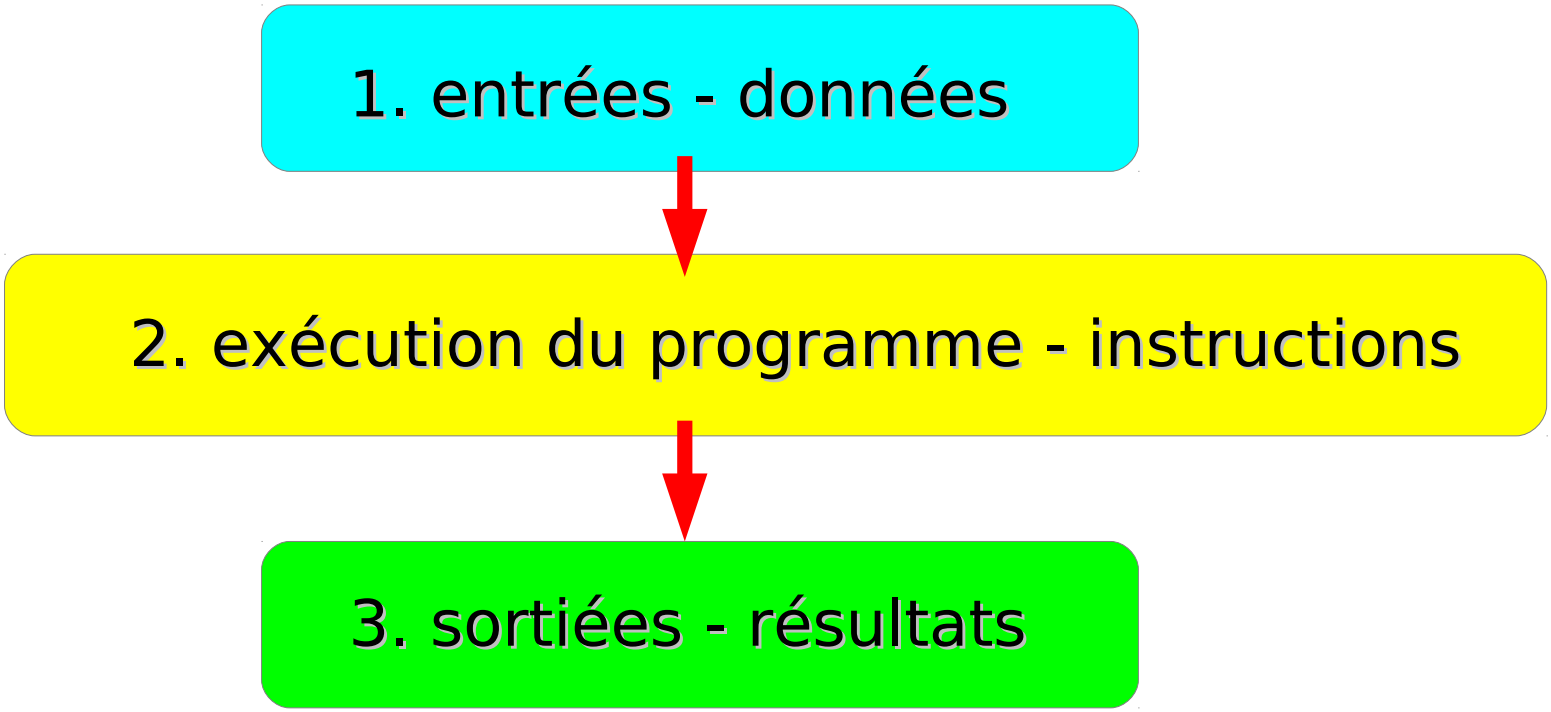
3. Programmation embarquée



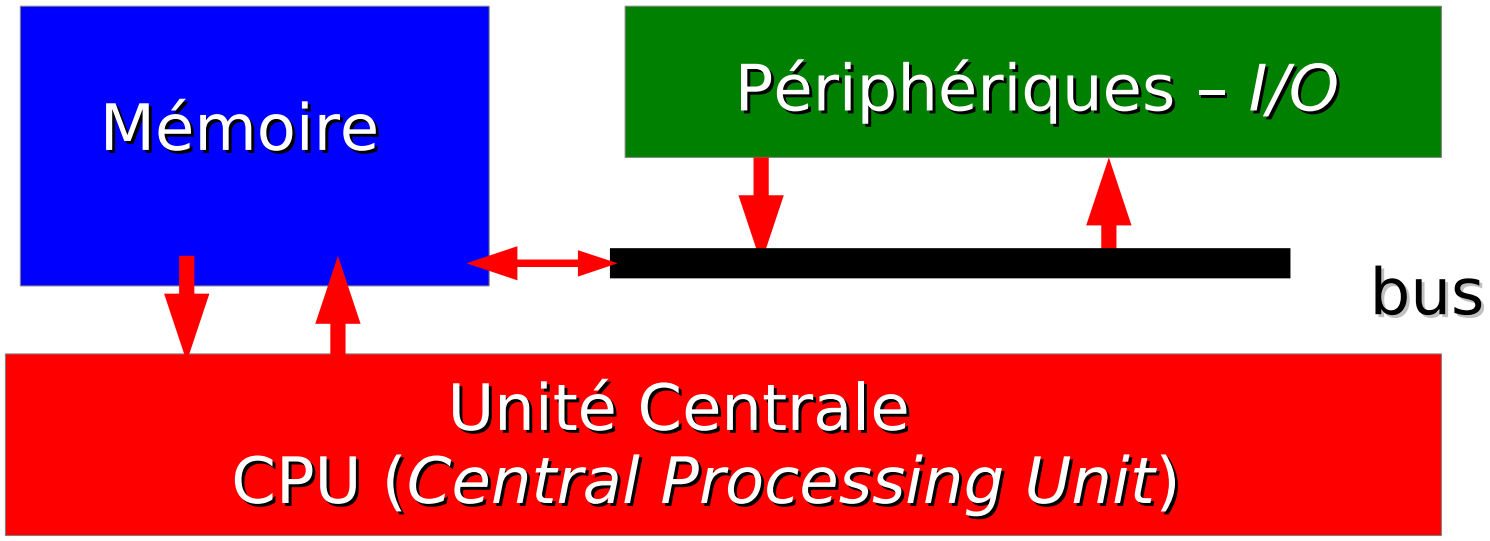
Java et domaines d'application

1. Internet et le WEB
2. Programmation distribuée
3. Programmation embarquée

Structure d'un programme



Les Ordinateurs



Les langages de programmation

Il en existe deux groupes principaux:

– Langages de haut niveau.

Exemples: C, C++, Java, Python).

– Langages de bas niveau.

Assembleurs :

code machine : ARM, Intel/PC, etc

Langages de haut niveau

– Les types des données :

```
int, char, double, ..
```

– La syntaxe :

```
1 <= x && x<= 7.
```

– La sémantique :

`"bonjour"*2` (problème?)

La production des programmes

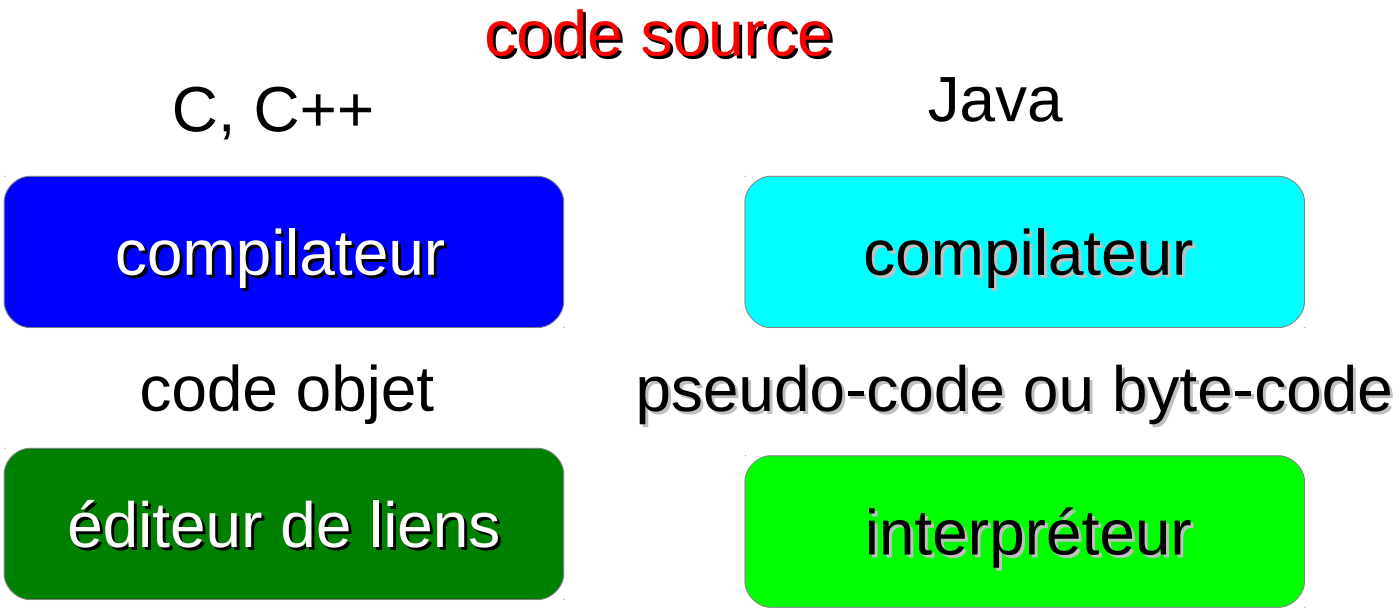
1. Analyse et Conception

2. Codage

3. Compilation
4. Tests
5. Correction d’erreurs

Traducteurs de programmes

– code source
– code cible
– code objet
– pseudo-code ou byte-code



Le programme «bonjour»

```
/ * Premier programme
- a enregistrer dans le fichier Bonjour.java
- a compiler par :  javac Bonjour.java
- a executer par :  java Bonjour
*/
public class Bonjour {
public static void main (String[] args) {
Terminal.ecrireStringln(”Bonjour tout le monde !”);
}
}
```

Compilation et exécution du «bonjour»

Compilation

```
Java/Essais> javac Bonjour.java
Java/Essais>
```

Interprétation du byte-code dans `Bonjour.class`

```
Java/Essais> java Bonjour
Bonjour tout le monde !
```

Programme (class) « Conversion »

```
public class Conversion {
public static void main(String[] args) {
// variables du programme
double euros, renminbis ;
// message pour la saisie
Terminal.ecrireStringln("Entrez la somme en euros:");
// lecture dans la variable euros
euros = Terminal.lireDouble( );
// calcul de la conversion
renminbis = euros * 8.21;
// affichage du resultat
Terminal.ecrireStringln("La somme convertie en
renminbis:" + renminbis);
}
}
```

Programme (class) « Conversion »

La compilation et l'exécution (interprétation) du programme Conversion.

```
Java/Essais> javac Conversion.java
Java/Essais> java Conversion
Entrez la somme en euros:
-3
La somme convertie en renminbis: ????
Java/Essais> java Conversion
Entrez la somme en euros:
15.5
La somme convertie en renminbis: ????
```

Programme en Java

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

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

Un programme comporte des mots reservés comme :

`public, class, static, void, main`

Il comporte egalement des noms qui sont donnés par le programmeur avec une assez grande liberté.
C'est la cas du nom du programme : `Conversion` et des noms des données `euros` et `renminbis`.

Programme en Java

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

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

Un programme est consitué d'au moins d'une classe et d'une méthode qui s'appelle main.
Il ne reste qu'à donner un nom au programme entre le mot réservé `class` et l'accolade et une suite d'instructions entre les accolades de la méthode `main`.

Eléments du programme

– des déclarations de variables. Les déclarations servent a donner un nom a 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.

– des instructions d'entrée-sortie. Notre programme calcule la valeur en renminbis 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 ordres prédéfinis qui permettent de faire cela (aller chercher une valeur au clavier, dans un fichier, sortir une valeur du programme vers l'écran, etc...).

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

Le nom des variables

`euros` ou `renminbis` sont les noms des variables, ils sont 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 c'est a dire commencent nécessairement par une lettre, majuscule ou minuscule, qui peut etre ou non suivie d'autant de caractères que l'on veut parmi l'ensemble `a . . .z, A . . .Z, 0..9, _, $`

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

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

Un type est un ensemble de valeurs particulières connues de la machine. Nous allons travailler avec les types java suivants :

– Le type **`int`** désigne l'ensemble de tous les nombres entiers représentables en machine sur 32 bits, avec les valeurs dans l'intervalle $\{-2^{31}, \ldots, 2^{31}-1\}$.

– le type **`double`** désigne les nombres a virgule (à précision double 64 bits). Les éléments de cet ensemble sont (exemples) -12.98,0.0, 0.1, . . .18.58 . . .

– Le type **`boolean`** modélise les deux valeurs de vérité dans la logique propositionnelle. Ses éléments sont **`true`** (vrai) et **`false`** (faux).

– Le type **`char`** modélise l'ensemble des caractères Unicode (sur 16 bits). Ses éléments sont des caractères entourés de guillemets simples, par exemple : 'a', '2', '@' , 'à', 'ç'

– le type **`String`** modélise les chaînes de caractères. Ses éléments sont les suites de caractères entourées de guillemets doubles : "coucou", "entrer une somme en euros : ?", "a" . . .

Syntaxe des déclarations de variables

Pour déclarer une variable, il faut donner un nom de type parmi int, double, char,boolean, String suivi d'un nom de variable que vous inventez.

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

C'est ce qu'on appelle la syntaxe Java des déclarations de variables. Il faut absolument se conformer a 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,

```
int x ; String coucou ;
```
sont corrects

alors que

```
entier x ;
```
ou encore
```
x int ;
```
seront rejetés

Affectation d'une valeur à une variable

Une fois qu'une variable a été declarée, on peut lui donner une valeur. Pour cela, il faut utiliser l'instruction d'affectation dont la syntaxe est :

nom variable = expression ;

par exemple : `x=2 ;`

Il est de bonne habitude de donner une valeur initiale a une variable dès qu'elle est déclarée. Java nous permet d'affecter une valeur a une variable au moment de sa déclaration :

par exemple : `int x=2 ;`

Si on déclare deux variables sur la même ligne, l'initialisation ressemble à cela :
`int x=0, y=0 ;`

A droite du signe =, on peut mettre n'importe quelle valeur du bon type. Comme 2 est une valeur du type int, notre exemple ne sera possible que si plus haut dans notre programme, on a la déclaration `int x;`

Expressions

Les expressions sont des calculs permettant d'obtenir une valeur d'un certain type. Elles servent a donner une valeur a une variable dans une affectation : cette valeur est d'abord calculée puis mise dans l'espace mémoire réservée à la variable. Un exemple d'expression arithmétique est :

(18+20)/2

Cette expression est un calcul permettant de calculer un nombre entier. C'est donc une expression permettant de donner une valeur a une variable déclarée de type int.

On peut ecrire l'affectation :

```
X=(18+20)/2;
```

On est habitué à utiliser des expressions arithmétiques dans la vie de tous les jours. Dans les programmes, les expressions peuvent concerner les autres types de données tels que les caractères et les booléens.

Constituants d'une expressions

– une valeur de base ou littéral, comme par exemple 18, 20 et 2. Chaque type java a ses propres valeurs de base.

– un opérateur qui calcule un résultat en utilisant un ou deux opérandes. Par exemple +, l'addition, calcule un nombre en additionnant deux nombres qui sont les opérandes. Chaque type java a ses propres opérateurs. Les opérandes ne sont pas nécessairement des valeurs de base : ce sont des expressions qui permettent de calculer la valeur a utiliser dans le calcul. Dans notre exemple, l'opérateur de division/ a deux opérandes (18+20) et 2. Le premier opérande est une expression comportant l'opérateur +.

– une variable. Lorsqu'une variable apparait dans une expression, le calcul a effectuer est d'aller chercher le contenu de la variable, c'est à dire la valeur stockée dans la case mémoire associée à la variable.

– un appel de méthode. Nous présenterons ce type d'expressions un peu plus loin dans ce chapitre.

Opérateurs

– Pour les nombres : les opérateurs arithmétiques sont l'addition (+), la soustraction (-), la multiplication (*), la division (/). Il y a egalement l'opérateur modulo (%) qui permet d'obtenir le reste de la division entière. Par exemple 7%3 permet d'obtenir le reste de la division de 7 par 3, c'est a dire 1.

– Pour les booléens : il existe des opérateurs comme le et logique noté &&, le ou logique noté || ou la négation noté !.

– Pour les chaînes de caractères : il existe un opérateur très utile appelé la concaténation, qui permet de coller bout à bout deux chaînes. Cet opérateur est noté +. Par exemple "abcd"+"xyz" est une expression, un calcul, dont le résultat est "abcdxyz".

Opérateurs

– Opérateurs de comparaison : ils permettent de comparer deux valeurs du même type. Le résultat de la comparaison est une valeur de type boolean. Ces opérateurs sont :

– l'égalité noté ==. Renvoie **`true`** si les deux valeurs comparées sont égales, **`false`** sinon. Exemple : 1==4 renvoie **`false`**, 'a'=='a' renvoie **`true`**
– est plus petit (<). Renvoie **`true`** si la première valeur est strictement plus petite que la deuxième, **`false`** sinon. Exemple : 1<4 renvoie **`true`**, 'a' < 'b' renvoie **`true`** (ordre entre caractères).
– est plus grand (>). Renvoie **`true`** si la première valeur est strictement plus grande que la deuxième, **`false`** sinon. Exemple : 1>4 renvoie **`false`**.
– est plus grand ou egal (>=). Renvoie **`true`** si la première valeur est plus grande ou egale que la deuxième, **`false`** sinon.
– est plus petit ou egal (<=). Renvoie **`true`** si la première valeur est plus petite ou egale que la deuxième, **`false`** sinon.
– différent (!=). Renvoie **`true`** si les deux valeurs sont différentes et **`false`** si les deux valeurs sont égales.

Expressions booléennes

Les expressions booléennes posent parfois des problèmes aux programmeurs débutants. Ce sont des expressions comme les autres, permettant de calculer une valeur et pouvant etre utilisée dans une affectation a une variable de type **`boolean`**. Voici quelques exemples d'expressions booléennes parfaitement correctes utilisées dans des affectations.

```
Boolean varbool ;
varbool = true ;
varbool = true||false ;
varbool = 14<=5 ;
varbool = (14>5)&&('a'!='b');
```

Deux expressions booléennes sont equivalentes si les calculs qu'elles dénotent ont le même résultat. Par exemple, si `vb` est une variable booléenne, les deux expressions `vb==true` et `vb` sont equivalentes. Si `vb` contient la valeur true, alors `vb==true` vaut true aussi. Si `vb` contient false, alors `vb==true` vaut false aussi. Pour cette raison, on n'écrit jamais une expression telle que `vb==true` dans un programme, car elle contient un calcul inutile.

La méthode Terminal.ecrireStringln()

Le programme initial donne l'ordre d'afficher a l'écran le message *Somme en euros ?*. Il le fait en utilisant une fonction écrite pour vous. Ce type de fonction on appelle en Java une méthode. Cette méthode - `ecrireStringln()` fait partie de la classe `Terminal`. Vous pouvez l'utiliser autant de fois que vous le désirez.

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 a 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 `Terminal.ecrireStringln` pour afficher le message *Somme en euros ?* en faisant l'appel :

```
Terminal.ecrireStringln("Somme en euros ? :);
```

Pour afficher coucou, il faut faire l'appel :
`Terminal.ecrireStringln("coucou").`

La méthode Terminal.lireDouble()

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

Cette ligne est une affectation. On trouve a droite du signe = un appel de méthode. La méthode appelée s'appelle `lireDouble()` et se trouve dans la classe `Terminal`.

Le fait qu'il n'y ait rien entre les parenthèses indique que cette méthode n'a pas besoin que le programme lui fournisse des informations. C'est une méthode sans paramètre. En revanche, le fait que cet appel se trouve a 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 ou résultat de la méthode.

Comme pour les paramètres 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`**.

Méthodes et expressions

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

Exemple : La méthode `Math.min()` : prends 2 paramètres de type `int` et retourne une valeur de type `int` : le plus petit de ses deux paramètres. 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 touche du clavier soit pressée (un curseur clignotant a l'écran indiquera cela). Dès que l'utilisateur presse une touche, le calcul de `Terminal.lireInt()` se termine avec pour résultat cette valeur.

Imaginons que l'utilisateur a pressé 6. 3+6+2 donne 11.
La valeur de `x` sera donc 11.

Les appels de méthodes en général

De nombreuses classes contenant des méthodes existent en Java, soit dans la bibliothèque commune au langage, soit dans des répertoires particulier qu'il faut alors indiquer. Toute méthode existante posède un nom, appartient à une classe, possède des paramètres dont le nombre le type et l'ordre sont fixés. Une méthode peut renvoyer une valeur ; le type de la valeur de retour est alors fixé. Pour appeler une méthode, il faut connaître toutes ces informations. L'appel de méthode sera conforme à la syntaxe suivante :

```
NomClasse.NomMethode(par1,...,parn);
```

Ainsi, on peut voir un méthode comme une boîte noire capable de réaliser des traitements sur les paraètres ou entrées de la méthode. Si l'on désire effectuer les traitements, on invoque ou appelle la méthode en lui passant les données à traiter, qui peuvent varier pour chaque appel. En sortie on obtient : ou bien une valeur, résultat des calculs effectués, ou bien un changement dans l'état de ou bien un changement dans l'état de la machine appelé aussi effet ; parfois, les deux. En exemple d'effet est l'affichage d'un message à l'écran. Les méthodes d'affichage ont un effet visible, mais elles ne renvoient pas de résultat.

La bibliothèque `Terminal`

La classe `Terminal` est un programme ecrit pour faciliter les saisies faites au clavier. Nous allons l'utiliser tout au long du cours dans nos exemples. Elle regroupe des méthodes de saisie et d'affichage au terminal pour les cinq types prédéfinis que nous utiliserons :

`int`, **`double`**, **`boolean`**, **`char`** et **`String`**.

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.

Méthodes de saisie dans `Terminal` :

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

Affichage avec `Terminal`

Pour chaque type, il existe deux méthodes d'affichage : une qui affiche une valeur et passe à la ligne suivante et l'autre qui affiche une valeur mais ne saute pas de ligne. Les méthodes avec saut de ligne ont un nom qui se termine par `ln`.

`Terminal.ecrireIntln` et `Terminal.ecrireInt`
`Terminal.ecrireDoubleln` et `Terminal.ecrireDouble`
`Terminal.ecrireCharln` et `Terminal.ecrireChar`
`Terminal.ecrireBooleanln` et `Terminal.ecrireBoolean`
`Terminal.ecrireStringln` et `Terminal.ecrireString`

L'affichage de messages, ou chaînes de caractères (type `String`), autorise l'adjonction d'autres messages ou valeurs en utilisant l'opérateur de concaténation `+`.

```
Terminal.ecrireString("bonjour " + 5 + 2);
---> affiche: bonjour 52
Terminal.ecrireString("bonjour " + (5 + 2));
---> affiche: bonjour 7
Terminal.ecrireString(5 + 2);
---> Erreur de Typage!
```

Résumé

Eléments du programme :

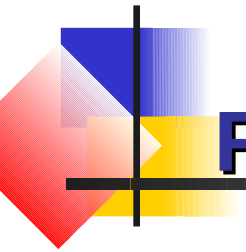
- déclarations - variables

- instructions E/S

- expressions

Méthodes et l'appel des méthodes

Entrées/Sorties par les méthodes da la classe `Terminal`



Résumé

Un ordinateurs :
- processeur (CPU), mémoire centrale, périphériques

Un programme :
- instructions, données, entrées/sorties

Compilation, interprétation :
- lexique, syntaxe, sémantique