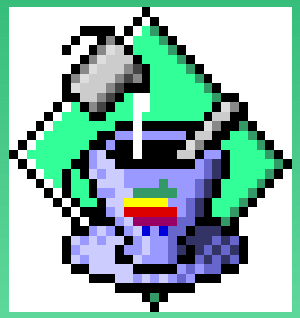
Programmation Java

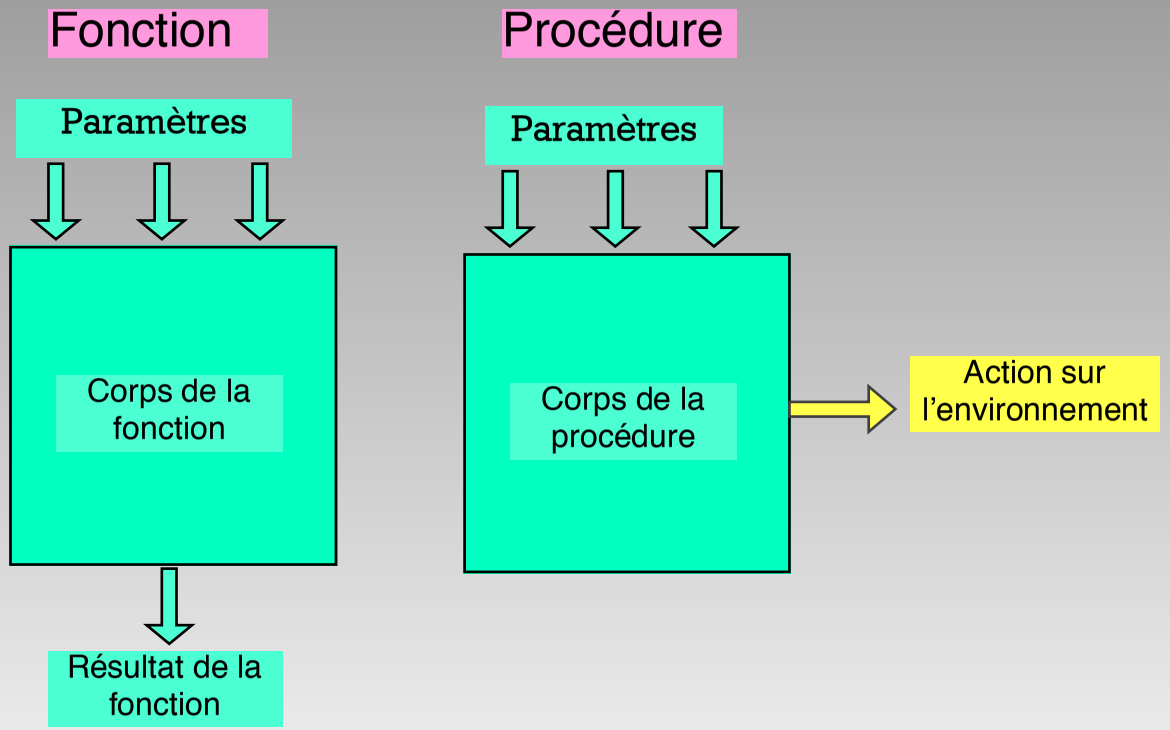


par Denis Monasse

Lycée Louis le Grand, Paris

• **Utilisation libre pour usage pédagogique**

Fonctions et procédures



Structure générale d'une fonction ou procédure

type_retour nom_fonction(*paramètres formels*)

{ *variables locales* ;

instruction;

instruction;

.....

instruction

}

type de retour : void pour une procédure, sinon le type des objets renvoyés par la fonction

paramètres formels : type1 nom1,type2 nom2, ...

variables locales :
type1 nom11,nom12,...
type2 nom21,nom22,...
....

Passage de parametres

$f: x \mapsto \sin(2x+3)$ x est paramètre formel de f

$f(\frac{3\pi}{2}+t)$ $\frac{3\pi}{2}+t$ est paramètre effectif de f

Règle de passage des parametres

- Avant de procéder au calcul, Java évalue les paramètres effectifs puis crée des variables locales ayant les noms des paramètres formels initialisées par les valeurs des paramètres effectifs correspondants
- les tableaux ne sont pas recopiés, le paramètre formel désigne le même tableau que le paramètre effectif

Passage de parametres

Exemple:

```
float bidule(int i, float x, boolean[] tab){
    ....
}

bidule(10, t + 3.14, a);
```

A l'entrée dans l'évaluation de `bidule(10, t + 3.14, a)`

- Java crée une variable locale `i` de type entier et l'initialise à 10
- Java crée une variable locale `x` de type réel et l'initialise à la valeur de `t` augmentée de 3.14
- Java crée une variable locale de nom `tab` et lui fait désigner le tableau `a` : les tableaux ne sont pas recopiés, toute modification de `tab` entraîne la modification correspondante de `a`

Sortie et retour d'un resultat

L'évaluation de l'expression Java

`return` *expression*

à l'intérieur du corps d'une fonction provoque l'évaluation de *expression*, la sortie immédiate du corps de la fonction avec comme résultat la **valeur** de *expression*.

Variables locales

Les variables locales sont utilisées pour stocker temporairement à l'intérieur de la fonction des résultats intermédiaires; elles sont en général typées, nommées et éventuellement initialisées au début de la fonction, avant toute instruction. Exemple: int i; float pi=3.14;

Les noms de ces variables locales n'interfèrent pas avec ceux des variables globales: ils se contentent de masquer **temporairement**, pendant l'exécution du corps de la fonction, des variables globales de même nom.

Le contenu de ces variables est **détruit** à la sortie de la fonction

Instructions elementaires

Affectation

- Permet de stocker un résultat dans une variable globale ou locale
- Sous la forme *nom* = *expression*. Lors de l'évaluation de cette affectation, *expression* est tout d'abord évaluée et sa valeur est affectée à *nom*. A partir de cet instant, toute évaluation de *nom* renverra cette valeur comme résultat. Les types doient être identiques (ou compatibles)

Effet de bord

- Appel de procédure avec des paramètres effectifs pour effectuer une action sur l'environnement
- Affichage par la fonction `System.out.println`

Séquence d'instructions

Suite d'instructions élémentaires (terminées par des points virgules) ou composées; les résultats eventuels d'appels de fonctions sont ignorés

Exemple :

```
x = sin(3*t);
y = 2*x + Pi/4;
System.out.println(x+1);
sin(y);  # inutile, ce résultat est ignoré
z = x+y
```

Instruction composee

Séquence d'instructions comprise entre deux accolades; considérée comme une seule instruction.

INSTRUCTION CONDITIONNELLE

Si une condition est vérifiée, on fait quelque chose, sinon on fait autre chose.

```
if( condition ) then instruction
[else autre instruction]
```

La partie entre crochets est facultative.

L'évaluation de ***condition*** doit nécessairement renvoyer un résultat de type booléen, vrai ou faux (`true` ou `false`).

Très souvent les instructions à exécuter seront des instructions composées ce qui donne la structure générale

```
if(condition) then
   {   instruction 1;
       instruction 2;
       ...
   } else {
       instruction 3;
       instruction 4;
       ...
   }
```

INSTRUCTION REPETITIVE

On répète une même instruction tant qu'une condition est vérifiée.

```
while( condition ) instruction
```

L'évaluation de ***condition*** doit nécessairement renvoyer un résultat de type booléen, vrai ou faux (`true` ou `false`).

Le test de la condition est effectué avant l'exécution de l'instruction (celle-ci peut donc très bien ne jamais être effectuée).

Très souvent l'instruction à exécuter est une instruction composée ce qui donne la structure générale

```
while(condition)
    {   instruction 1;
        instruction 2;
        ...
    }
```

INSTRUCTION ITERATIVE

On répète une même instruction en faisant varier un ou des compteurs jusqu'à ce que ces compteurs dépassent certaines valeurs ou qu'une condition ne soit plus vérifiée.

`for(` *initialisations* **`;`** *tests* **`;`** *incrémentations* **`)`** *instruction*

- les initialisations sont effectuées
- l'instruction est exécutée de manière répétitive
  - en effectuant entre deux exécutions les incrémentations demandées
  - tant que tous les tests renvoient vrai

Les tests sont effectués avant l'exécution de l'instruction (celle-ci peut donc très bien ne jamais être effectuée).

Equivaut à peu près à

```
initialisations;
while( tests ) {
    instruction;
    incrémentations }
```

Types de données

- nombre entier: `int, longint : 123456`
- nombre réel: `float, double : 2.71828 , 1.3e10`
- chaine de caractères: `"x" , "bonjour monsieur"`
- booléen : `true, false`
- tableaux à une ou plusieurs dimensions d'objets d'un même type

Les tableaux

- le type tableau : *type_élément* `[ ]` : `int[], float[][]`
- accès à un élément par son indice : *nom_tableau* `[` *indice* `]`: `a[5],b[0][3]`
  (attention: les indices démarrent à 0)
- initialisation d'un tableau : *nom_tableau* = `new` *type_élément* `[` *longueur* `]`
- les tableaux ne sont pas recopiés par affectation ou passage en tant que paramètre d'une fonction mais au contraire partagés
- un tableau à deux dimensions est simplement un tableau de tableaux, etc.

STRUCTURE DU PROGRAMME

`class` *nom_du_programme* {

- déclaration et éventuellement initialisation des variables globales
- définition des fonctions auxiliaires
- définition de la fonction principale:

  `public static void main(String[] args)`

  - son nom est nécessairement `main`
  - le type renvoyé est nécessairement `void`
  - elle doit être déclarée avec les qualificatifs `public static`
  - elle doit recevoir en paramètres un tableau de chaînes de caractères `String[]`

}

LES COMMENTAIRES

Indispensables pour faire comprendre la programmation au lecteur

- **Tout ce qui est compris entre /* et */ est négligé par le comilateur**
- **Toute la fin de ligne à partir de // est négligée par le compilateur**

Exemple de programme

```
class toto {
    int i=10;
    float[] a;

    void initialise(int dim){// initialise le tableau
        int i;
        a = new float[dim];
        for(i=0; i<dim; i=i+1)
            a[i]=1.0;
    }

    public static void main(String[] args){
        initialise(i);
        for(int k=0; k<i ; k=k+1)
                System.out.println(a[k]);
    }
 }
```