M1.23.3 Algorithmique

Jean-François Berdjugin
IUT1, département SRC, L'Isle d'Abeau

Références

- Support de cours, de TD et de TP de Jean-François Remm
- http://fr.wikipedia.org/wiki/Algorithmique

Plan

- Programmation
- Algorithmique
- Notion de variable
- L'instruction d'affectation
- Les instructions d'écriture lecture
- Le choix
- Les boucles
- Factorisation du code

Programmation

- Un microprocesseur ne sait exécuter que du code machine.
- Code peut lisible difficile à maintenir => utiliser des langages de programmation de haut niveaux
- De nombreux langages de différents types (impératifs, fonctionnels, déclaratifs, objets, concurrents, …)
  - Java, C++, ADA
  - Camel, Lisp
  - Prolog, XSLT
  - …
- Mais avec un point commun

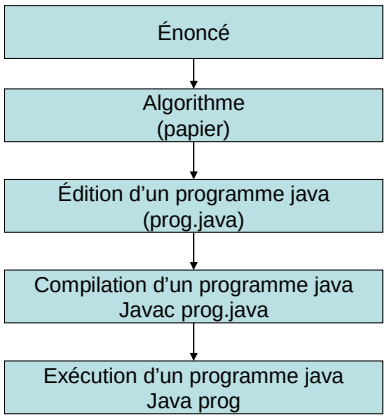
Programmation

Exemple de réalisation d'un programme par analyse descendante :

Énoncé du problème et donnée
⇓
Spécification
⇓
Cahier des charges
⇓
Analyse
⇓
Algorithme
⇓
Traduction en langage et édition
⇓
Programme Source
⇓
Compilation
⇓
Programme exécutable
⇓
Exécution
⇓
Résultats

Pour nous cette année :

Traduction algorithmique d'un problème puis traduction en langage Java, compilation et exécution



Algorithmique

Définition :

Un algorithme est un moyen pour un humain de présenter la résolution par calcul d'un problème à une autre personne physique (un autre humain) ou virtuelle (un calculateur).

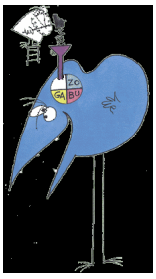
Un algorithme est un énoncé dans un langage bien défini d'une suite d'opérations permettant de résoudre par calcul un problème. Si ces opérations s'exécutent en séquence, on parle d' algorithme séquentiel. Si les opérations s'éxécutent sur plusieurs processeurs en parallèle, on parle d' algorithme parallèle. Si les tâches s'exécutent sur un réseau de processeurs on parle d' algorithme distribué.

Algorithmique

- Utilise une structure logique indépendante du langage
- Deux approches :
  - Organigrammes (lourdeur et manque de structure)
  - Pseudo code ou Pseudo Langage

Ga Bu Zo Meu

- L'instruction d'affectation de variables
- Les instructions d'écriture lecture
- L'instruction de choix
- Les instructions de boucles



Je dis des choses tellement intelligentes
que le plus souvent je ne comprends pas ce que je dis.

Syntaxe Pseudo Langage

En pseudo code

Algo nomAlgo
    déclarations
Début
    instructions
Fin

En java

```
class nomAlgo
{
public static void
   main(String[ ] args)
   {
        déclarations
        instructions
   }
}
```

Syntaxe Pseudo Langage

Bonne Syntaxe

Algo Affect1
    var A : entier
    var B : entier
Début
    A ← 12
    B ← A - 10
    A ← 2
Fin

Mauvaise Syntaxe

Algo Affect1
    var A : entier
    var B : entier

    A ← 5
    B ← A+1
    A ← 2
Fin
Fin

Commentaires

- Tout code doit être commenté

// commentaire

// commentaire

Variable

Les variable peuvent être vue comme des boîtes avec des étiquettes (leurs nom) qui ne peuvent contenir qu'un type de chose.

Il faut avant d'utiliser une variable, créer la boîte et lui coller l'étiquette, c'est une déclaration.

A un instant donné une variable possède une valeur et une seule, elle ne possède pas d'historique.

Une variable qui ne peut changer de valeur est dite constante.

Variables – types -

Types en Pseudo code
- Entier
- Réel
- Booléen
- Caractère
- Chaîne

Types primitifs java :
- byte, short, int, long
- float, double
- boolean
- char

Chaînes :
- String

Les variables – affectation -

identif ← expr

identif = expr;

Une instruction d'affectation ne modifie que la partie gauche de la flèche

Un variable utilisée dans une affectation doit être déclarée

En partie droite d'une affectation ne peut se trouver qu'une variable (pas une autre expression)

Une variable utilisée en partie droite doit avoir été instanciée.

Affectation - expression

- Les expressions sont obtenues en appliquant des opérateurs à des variables, des constantes et des littéraux.
- Une expression est interprétée en une valeur.
- Exemple :
  - 2+3 (5)
  - " il fait " + " beau " ("il fait beau")
  - -4+6 (2)

Affectation

Algo Affect2
  var A : entier
Début
  A ← 12
  A ← 2
Fin

A:2

Algo Affect3
  var A : entier
Début
  A ← 2
  A ← 12
Fin

A:12

Affectation

Algo Affect4
    var A : entier
    var B : entier
Début
    A ← 12
    B ← 2
Fin

A:12
B:2

Algo Affect5
    var A : entier
    var B : entier
Début
    A ← 2
    B ← A * 2
Fin

A:2
B:4

Affectation

Algo Affect6
    var A : entier
    var S : entier
Début
    A ← 12
    S ← 10
    S ← S + A + 1
Fin

A:12
S:23

Algo Affect7
    var A : entier
    var B : entier
    var C : entier
Début
    A ← 3
    B ← 10
    C ← A + B
    B ← A + B
    A ← C
Fin

A:?
B:?
C:?

Affectation

Algo Affect6
    var A : entier
    var S : entier
Début
    A ← 12
    S ← 10
    S ← S + A + 1
Fin

A:12
S:23

Algo Affect7
    var A : entier
    var B : entier
    var C : entier
Début
    A ← 3
    B ← 10
    C ← A + B
    B ← A + B
    A ← C
Fin

A: 13
B: 13
C: 13

Lecture et Ecriture

Comment communiquer avec l'utilisateur ?

=>

En utilisant des instructions de lecture écriture

- Lire(x) attend la frappe de quelque chose au clavier et affecte la saisie à x
- Ecrire(y) écrit la valeur de y à l'écran

Lecture - Ecriture

- Lire (variable)
- Ecrire (variable)

- lecture : compliquée

```
InputStreamReader
  systemReader = new
  InputStreamReader(S
  ystem.in);
...
```

- écriture : System.out.println(var iable)

Test

- Le déroulement normal d'un programme peut être modifié en fonction de certaines conditions (des expressions booléennes)

- Exemple: si j'ai au moins huit œufs alors je fais des crêpes sinon je fais une omelette

Tests

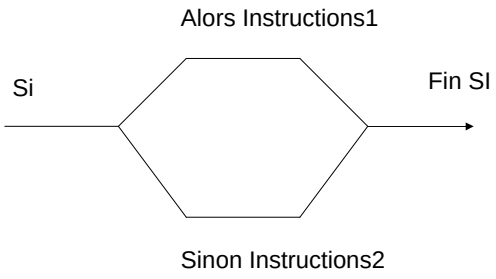
Si (condition) **Alors**
instruction**s**1
Sinon
instruction**s**2
Fin Si

Rem: Le sinon et le else sont facultatifs

```
if (condition)
{
   instructions1
}
else
{
   instructions2
}
```

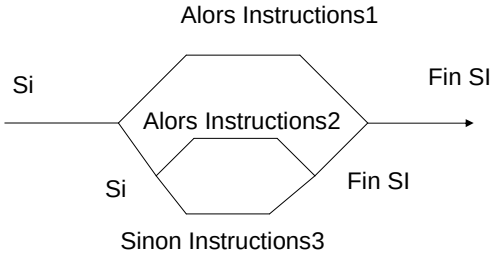
Choix

Penser au aiguillages



Choix

Penser au aiguillages



Choix

Algo Choix1

var oeuf : entier

Début

Lire(œuf)

Si (œuf ≥ 8) alors

Ecrire(“Je fais des crêpes“)

Sinon

Ecrire(“Je fais une omelette“)

Fin Si

Fin

Algo Choix2

var oeuf : entier

Début

Lire(œuf)

Si (œuf < 8) alors

Ecrire(“Je fais une omelette“)

Sinon

Ecrire(“Je fais des crêpes “)

Fin Si

Fin

Choix

Pour faire des crêpes il me faut au moins 8 œuf et un litre de lait et 500 grammes de farine et du beurre en n'importe quelle quantité.

```
Algo Choix3
    var oeuf : entier
    var farine : entier //exprimée en grammes
    var lait : réel //exprimé en lait
    var beurre : booléen //présence de beurre
Début
    Lire(œuf)
    Lire(farine)
    Lire(lait)
    Lire(beurre)
    Si ((œuf ≥ 8) et (farine ≥ 500) et (lait>1) et beurre)
        alors
        Ecrire("Je fais des crêpes")
    Sinon
        Ecrire("Je fais une omelette")
    Fin Si
Fin
```

```
Algo Choix4
    var oeuf : entier
    var farine : entier //exprimée en grammes
    var lait : réel //exprimé en lait
    var beurre : booléen //présence de beurre
Début
    Lire(œuf)
    Lire(farine)
    Lire(lait)
    Lire(beurre)
    Si (œuf ≥ 8) Alors
        Si (farine ≥ 500) Alors
            Si (lait>1) Alors
                Si (beurre) alors
                    Ecrire("Je fais des crêpes")
                Fin Si
            Fin Si
        Fin Si
    Sinon
        Ecrire("Je fais une omelette")
    Fin Si
Fin
```

Bon ?

Choix

```
Algo Choix5
    var oeuf : entier
    var farine : entier //exprimée en grammes
    var lait : réel //exprimé en lait
    var beurre : booléen //présence de beurre
Début
    Lire(œuf)
    Lire(farine)
    Lire(lait)
    Lire(beurre)
    Si (œuf ≥ 8) Alors
        Si (farine ≥ 500) Alors
            Si (lait>1) Alors
                Si (beurre) alors
                    Ecrire("Je fais des crêpes")
                Sinon
                    Ecrire("Je fais une omelette")

                Fin Si
            Sinon
                Ecrire("Je fais une omelette")
            Fin Si
        Sinon
                Ecrire("Je fais une omelette")
        Fin Si
    Sinon
        Ecrire("Je fais une omelette")
    Fin Si
Fin
```

```
Algo Choix6
    var oeuf : entier
    var farine : entier //exprimée en grammes
    var lait : réel //exprimé en lait
    var beurre : booléen //présence de beurre
    Var crepe : booléen //vrai si crêpe possible
Début
    Lire(œuf)
    Lire(farine)
    Lire(lait)
    Lire(beurre)
    crêpe ← faux //a priori je ne peut pas faire de crêpes
    Si (œuf ≥ 8) Alors
        Si (farine ≥ 500) Alors
            Si (lait>1) Alors
                Si (beurre) alors
                    crêpe ← vrai
                Fin Si
            Fin Si
        Fin Si
    Fin Si
    Si crêpes Alors
        Ecrire("Je fais des crêpes")
    Sinon
        Ecrire("Je fais une omelette")
    Fin Si
Fin
```

Choix

Qu'il fasse beau ou qu'il fasse froid je vais à l'IUT mais si il fait froid alors je me presse.

```
Algo Choix5
    var froid :booléen //il fait froid
Début
    Lire(froid)
    Si (froid) Alors
        Ecrire("Je me presse")
    Fin Si
    Ecrire("Je vais à l'IUT")
Fin
```

Choix

- Écrire en fonction de la température, l'état de l'eau (glace, liquide, vapeur)

Choix

```
Algo Choix7
    var temp : entier
Début
    Ecrire "Entrez la température de l'eau :"
    Lire(temp)
    Si (temp ≤ 0) Alors
        Ecrire("C'est de la glace")
    Sinon
        Si (temp < 100) Alors
            Ecrire("C'est du liquide")
        Sinon
            Ecrire("C'est de la vapeur")
        Finsi
    Finsi
Fin
```

Choix

Validation de semestre

A l'issue des contrôles sont calculées : (cf. annexe II)

- une moyenne dans chaque module
- une moyenne par unité d'enseignement (UE)
- une moyenne générale du semestre ou de l'unité pédagogique

La validation d'un semestre ou d'une unité pédagogique est acquise :
de droit lorsque l'étudiant ou le stagiaire, a obtenu à la fois :

- une moyenne générale du semestre ou de l'unité pédagogique

supérieure ou égale à 10/20 et une moyenne supérieure ou égale à 8/20 dans chacune des UE,

- et la validation des semestres ou des unités pédagogiques

précédents lorsqu'ils existent.

Donner l'algorithme permettant de savoir à partir des moyennes des UEs de savoir, si le semestre 1 est validé.

Choix

Validation du S2

Donner l'algorithme permettant de savoir si en fin de S2, ce dernier semestre est validé.

Boucle

- Nous pouvons en fonction de conditions exécuter telle ou telle séquence d'instruction

- Mais il arrive qu'un même traitement se répète plusieurs fois avec des valeurs possiblement différentes pour les variables

=>

- Utiliser des boucles nommée structures répétitives ou itératives

Boucles

Tant que (condition) **Faire**
    instructions
Fin Tant que

Répéter
    instructions
Jusqu'à (condition)

```
while (condition)
{
    instructions
}

do
{
   instructions
}
while ( !condition) ;
```

Boucles

- L'opérateur doit choix de prendre un café ou non

Algo boucle1
  var c : caractère
Début
  Ecrire("Voulez vous un café ? (O/N)")
  Lire(c)
  Tant Que (c ≠ 'O') et (c ≠ 'N') **Faire**
  Lire(c)
  FinTant Que
Fin

Boucles

- Boucle infinie, oublier de modifier la condition de boucle de sorte qu'une fois entré dans une boucle l'on ne puisse jamais en sortir.

```
Algo boucle2
   var i : entier
Début
   i ← 0
Tant que (i<10) Faire
      Ecrire(i)
Fin Tant que
Fin
```

Boucles

- Boucle inutile, il faut vérifier que l'on puisse passer dans la boucle.

=>

Vérifier l'initialisation des variables de la condition

```
Algo boucle3
   var i : entier
Début
   i ← 11
Tant que (i<10) Faire
      Ecrire(i)
Fin Tant que
Fin
```

Boucle

- Écrire un algorithme qui demande successivement 20 nombres à l'utilisateur, et qui lui dise ensuite quel était le plus grand parmi ces 20 nombres.

- Quel type de boucle choisir :
  - Tant que pas de passage obligatoire dans la boucle
  - Répéter jusqu'à au moins un passage dans la boucle

- Que faire dans la boucle
- Qu'elle est la condition de sortie de boucle.

Boucle

```
Algo boucle4
    var i : entier
    var max : entier
     var n : entier
Début
    i ← 1
    lire(n)
    max ← n
      Répéter
          Lire(n)
          Si (n > max) Alors
             max ← n
          Fin Si
          i ← i + 1
      Jusqu'à (i=20)
      Ecrire ("Le maximum est " + max)
Fin

i ?
max ?
n ?
```

```
Algo boucle5
    var i : entier
    var max : entier
     var n : entier
Début
    i ← 1
    lire(n)
    max ← n
Tant que (i < 20) faire
    lire(n)
    Si (n > max) Alors
            max ← n
    Fin Si
    i ← i + 1
Fin Tant que
    Ecrire ("Le maximum est " + max)
Fin

i?
max ?
n ?
```

Boucle

```
Algo boucle4
    var i : entier
    var max : entier
     var n : entier
Début
    i ← 1
    lire(n)
    max ← n
     Répéter
        Lire(n)
        Si (n > max) Alors
           max ← n
        Fin Si
        i ← i + 1
     Jusqu'à (i=20)
     Ecrire ("Le maximum est " + max)
Fin
```

```
Algo boucle5
    var i : entier
    var max : entier
     var n : entier
Début
    i ← 1
    lire(n)
    max ← n
Tant que (i < 20) faire
    lire(n)
    Si (n > max) Alors
          max ← n
    Fin Si
    i ← i + 1
Fin Tant que
    Ecrire ("Le maximum est " + max)
Fin
```

Boucle

```
Algo boucle6
    var i : entier
    var max : entier
     var n : entier
Début
    i ← 0
    Répéter
        Lire(n)
        Si (i = 0) Alors
            max ← n
        Fin Si
        Si (n > max) Alors
           max ← n
        Fin Si
        i ← i + 1
     Jusqu'à (i=20)
     Ecrire ("Le maximum est " + max)
Fin
```

```
Algo boucle7
    var i : entier
    var max : entier
     var n : entier
Début
    i ← 0
    Tant que (i < 20) faire
    lire(n)
    Si (i = 0) Alors
          max ← n
    Fin Si
    Si (n > max) Alors
          max ← n
    Fin Si
    i ← i + 1
Fin Tant que
    Ecrire ("Le maximum est " + max)
Fin
```

Boucle

- Écrire un algorithme qui demande un nombre de départ, et qui calcule la somme des entiers jusqu'à ce nombre. Par exemple, si l'on entre 5, le programme doit calculer :

  1 + 2 + 3 + 4 + 5 = 15

  NB : on souhaite afficher uniquement le résultat, pas la décomposition du calcul.

Boucle

```
Algo boucle7
    var i : entier
    var som : entier
     var n : entier
Début
    i ← 0
    som ← 0
    lire (n)
    Répéter
        som ← som + i
        i ← i + 1
     Jusqu'à (i=n)
     Ecrire ("La somme est  " +som)
Fin

i=?
som=?
n=3
```

```
Algo boucle8
    var i : entier
    var som : entier
     var n : entier
Début
    i ← 0
    som ← 0
    lire (n)
    Tant que (i < n) faire
        som ← som + i
        i ← i + 1
    Fin Tant que
     Ecrire ("La somme est  " +som)
Fin

i=?
som=?
n=3
```

Boucle

```
Algo boucle7
    var i : entier
    var som : entier
     var n : entier
Début
    i ← 0
    som ← 0
    lire (n)
    Répéter
          som ← som + i
          i ← i + 1
     Jusqu'à (i=n)
     Ecrire ("La somme est  " +som)
Fin
```

i=2
som=1+2
n=3

```
Algo boucle8
    var i : entier
    var som : entier
     var n : entier
Début
    i ← 0
    som ← 0
    lire (n)
    Tant que (i < n) faire
          som ← som + i
          i ← i + 1
    Fin Tant que
     Ecrire ("La somme est  " +som)
Fin
```

i=2
som=1+2
n=3

Boucle

```
Algo boucle9
    var som : entier
    var n : entier
Début
    som ← 0
    lire (n)
    Répéter
          som ← som + n
          n ← n - 1
     Jusqu'à (n=0)
     Ecrire ("La somme est  " +som)
Fin
```

som=?
n=3

```
Algo boucle10
    var som : entier
     var n : entier
Début
    som ← 0
    lire (n)
    Tant que (0 < n) faire
          som ← som + n
          n ← n - 1
    Fin Tant que
     Ecrire ("La somme est  " +som)
Fin
```

som=?
n=3

Boucle

Algo boucle9
    var som : entier
    var n : entier
Début
    som ← 0
    lire (n)
    Répéter
        som ← som + n
        n ← n - 1
     Jusqu'à (n=0)
     Ecrire ("La somme est " +som)
Fin

som=3+2+1
n=3

Algo boucle10
    var som : entier
     var n : entier
Début
    som ← 0
    lire (n)
    Tant que (0 < n) **faire**
        som ← som + n
        n ← n - 1
    Fin Tant que
     Ecrire ("La somme est " +som)
Fin

som=3+2+1
n=3

Boucles

Pour var **de** debut **à** n
Faire
    instructions
Fin Pour

for (int var=debut ;var<=n ;var++)
{
    instructions
}

for (int var=debut ;var>=n ;var−−)
{
    instructions
}

Boucle

Pour vari **de** debut **à** n
Faire
    instructions
Fin Pour
⇔
vari ← debut
Tant que (vari < n) **faire**
    Instructions
    vari ← vari + 1
Fin Tant que

Pour i **de** 1 **à** 5
Faire
    Ecrire(i)
Fin Pour
⇔
i ← 1
Tant que (1 < 5) **faire**
    Ecrire(i)
    vari ← vari + 1
Fin Tant que

Boucle

- Ne pas bricoler le compteur dans la boucle pour.

Pour i **de** 1 **à** n
Faire
    Ecrire(i)
    i ← i * 2
Fin Pour

Boucle

- Les boucles peuvent être imbriquées
- Donner {1,2,3,4} x {1,2,3,4,5}

Boucle

- Donner {1,2,3,4} x {1,2,3,4,5}

```
Algo boucle11
    var i: entier
    var j : entier
Début
    Ecrire(" {" );
    pour i de 1 à 4 faire
        pour j de 1 à 5 faire
            Ecrire ("("+ i + ", " + j + ")");
        fin pour
    fin pour
    Ecrire (" }");
Fin
```

Factoriser le code

- Objectifs :
  - Ne pas réécrire l'existant (factoriser)
  - Maintenir le code
- Les fonctions et procédures
- Les classes

Fonctions/Procédures

Prendre un café

```
Algo cafe
    var c : caractere
Début
    Ecrire("Voulez vous un café (o/n)" );
    Lire(c)
    TantQue ((c <> 'o') et (c <> 'n'))
            Ecrire "Tapez o ou n"
            Lire(c)
    FinTantQue
    Si c=o Alors
            Ecrire ("Voulez vous de sucre (o/n)" );
            lire(c)
            TantQue ((c <> 'o') et (c <> 'n'))
                    Ecrire "Tapez o ou n"
                    Lire(c)
            FinTantQue
            Ecrire ("Voulez vous un café long (o/n)" );
             lire(c)
            TantQue ((c <> 'o') et (c <> 'n'))
                    Ecrire "Tapez o ou n"
                    Lire(c)
            FinTantQue
            Ecrire « Veuillez patienter votre café se
    prépare »
    Fin Si
Fin
```

- Quel code peut-il être factorisé ?

Fonctions/Procédures

Prendre un café

```
Algo cafe
    var c : caractere
Début
    Ecrire("Voulez vous un café (o/n)" );
    Lire(c)
    TantQue ((c <> 'o') et (c <> 'n')) faire
            Ecrire "Tapez o ou n"
            Lire(c)
    FinTantQue
    Si c=o Alors
            Ecrire ("Voulez vous de sucre (o/n)" );
            lire(c)
            TantQue ((c <> 'o') et (c <> 'n')) faire
                    Ecrire "Tapez o ou n"
                    Lire(c)
            FinTantQue
            Ecrire ("Voulez vous un café long (o/n)" );
             lire(c)
            TantQue ((c <> 'o') et (c <> 'n')) faire
                    Ecrire "Tapez o ou n"
                    Lire(c)
            FinTantQue
            Ecrire « Veuillez patienter votre café se
    prépare »
    Fin Si
Fin
```

- Quel code peut-il être factorisé ?

```
lire(c)
TantQue ((c <> 'o') et (c <> 'n')) faire
    Ecrire "Tapez o ou n"
    Lire(c)
FinTantQue
```

Fonctions/Procédures

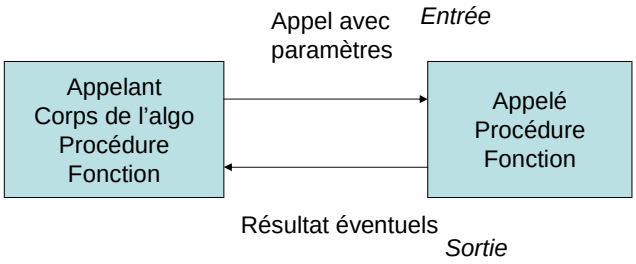
Prendre un café

```
Algo cafe
    var c : caractere
Début
    Ecrire("Voulez vous un café (o/n)" );
    Lire(c)
    TantQue ((c <> 'o') et (c <> 'n'))
            Ecrire "Tapez o ou n"
            Lire(c)
    FinTantQue
    Si c=o Alors
            Ecrire ("Voulez vous de sucre (o/n)" );
            lire(c)
            TantQue ((c <> 'o') et (c <> 'n'))
                    Ecrire "Tapez o ou n"
                    Lire(c)
            FinTantQue
            Ecrire ("Voulez vous un café long (o/n)" );
             lire(c)
            TantQue ((c <> 'o') et (c <> 'n'))
                    Ecrire "Tapez o ou n"
                    Lire(c)
            FinTantQue
            Ecrire « Veuillez patienter votre café se
    prépare »
    Fin Si
Fin
```

```
Fonction oui-non():caractere
    Lire(c)
    TantQue ((c <> 'o') et (c <> 'n')) faire
            Ecrire "Tapez o ou n"
            Lire(c)
    Renvoie c
FinFonction

Algo cafe
    var c : caractere
Début
    Ecrire("Voulez vous un café (o/n)" );
    c ← oui-non():caractere
    Si c=o Alors
            Ecrire ("Voulez vous de sucre (o/n)" );
             c ← oui-non()
            Ecrire ("Voulez vous un café long (o/n)" );
             c ← oui-non()
            Ecrire « Veuillez patienter votre café se
    prépare »
    Fin Si
Fin
```

Fonctions/Procédures



Les procédure sous des sous-programmes qui ne renvoient rien, les fonctions sont des sous-programmes qui renvoient quelque chose.

En java, il n'y a que des fonctions dont le type de retour est vide =>

Fonctions/Procedures

Fonction nomFonc(param1 : type1 [, pa-
ram2 : type2]) : typeRetour
    instructions
    Renvoie (resultat) // si nécessaire
Fin Fonction

Algo Appel
Début
    nomFonc(val1,val2) ou
    X ← nomFonc(val1,val2)
Fin

```
class Appel
{
static typeRetour nomFonc (type1 param1,
type2 param2)
{
    instructions
    return (resultat) ;
}

public static void main(String[ ] args)
{
    nomFonc(val1,val2) ou
    X= nomFonc(val1,val2) ;
}
}
```

Fonctions/paramètres

Nous souhaitons améliorer notre fonction oui-non() pour quelle prenne en paramètre, le texte a afficher.

Fonctions/Procédures

Prendre un café

```
Algo cafe
    var c : caractere
Début
    Ecrire("Voulez vous un café (o/n)" );
    Lire(c)
    TantQue ((c <> 'o') et (c <> 'n'))
            Ecrire "Tapez o ou n"
            Lire(c)
    FinTantQue
    Si c=o Alors
            Ecrire ("Voulez vous de sucre (o/n)" );
            lire(c)
            TantQue ((c <> 'o') et (c <> 'n'))
                    Ecrire "Tapez o ou n"
                    Lire(c)
            FinTantQue
            Ecrire ("Voulez vous un café long (o/n)" );
             lire(c)
            TantQue ((c <> 'o') et (c <> 'n'))
                    Ecrire "Tapez o ou n"
                    Lire(c)
            FinTantQue
            Ecrire « Veuillez patienter votre café se
    prépare »
    Fin Si
Fin
```

```
Fonction oui-non(s:chaine):caractere
    Ecrire(s)
    Lire(c)
    TantQue ((c <> 'o') et (c <> 'n')) faire
            Ecrire "Tapez o ou n"
            Lire(c)
    Renvoie c
FinFonction

Algo cafe
    var c : caractere
Début
            c ← oui-non(« Voulez vous un café (o/n)" )
    Si c=o Alors
             c ← oui-non("Voulez vous de sucre (o/n)" )
             c ← oui-non("Voulez vous un café long (o/n)" )
            Ecrire « Veuillez patienter votre café se
    prépare »
    Fin Si
Fin
```

Fonctions/Passage de paramètres

- le passage par valeur
  - Le paramètre est « copié » dans la classe => les modification du paramètre ne modifient pas la variable passée en paramètre mais sa copie.
- le passage par référence
  - Le paramètre n'est pas « copié » mais utilisé directement dans le sous-programme => la variable passée en paramètre peut être modifiée.

Fonctions/Passage de paramètres

Valeur

```
Fonction plusplus(i: entier):entier
    i ← i + 1
    renvoie(i)
Fin Fonction

Algo testplusplus
    var j,k: entier
Debut
    j ← 2
    k ← plusplus(j)
Fin
    j ?
    k ?
```

Référence

```
Fonction plusplus(i: entier):entier
    i ← i + 1
    renvoie(i)
Fin Fonction

Algo testplusplus
    var j,k: entier
Debut
    j ← 2
    k ← plusplus(j)
Fin
    j ?
    k ?
```

Fonctions/Passage de paramètres

Valeur

Fonction plusplus(i: *entier*):*entier*
    i ← i + 1
    renvoie(i)
Fin Fonction

Algo testplusplus
    var j,k: *entier*
Debut
    j ← 2
    k ← plusplus(j)
Fin
    j 2
    k 3

Référence

Fonction plusplus(i: *entier*):*entier*
    i ← i + 1
    renvoie(i)
Fin Fonction

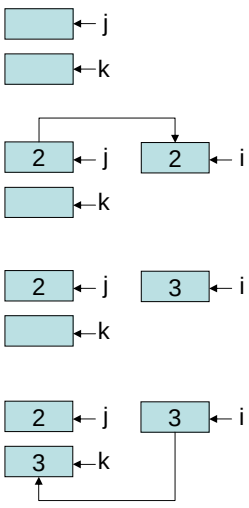
Algo testplusplus
    var j,k: *entier*
Debut
    j ← 2
    k ← plusplus(j)
Fin
    j 3
    k 3

Fonctions/Passage de paramètres

Valeur

Fonction plusplus(i: *entier*):*entier*
    i ← i + 1
    renvoie(i)
Fin Fonction

Algo testplusplus
    var j,k: *entier*
Debut
    j ← 2
    k ← plusplus(j)
Fin
    j 2
    k 3

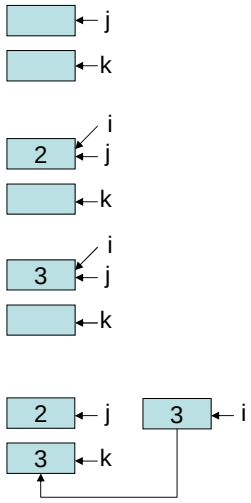


Fonctions/Passage de paramètres

Référence

Fonction plusplus(i: *entier*):*entier*
    i ← i + 1
    renvoie(i)
Fin Fonction

Algo testplusplus
    var j,k: *entier*
Debut
    j ← 2
    k ← plusplus(j)
Fin
    j 2
    k 3



Fonctions/passage de paramètre

- En java c'est plus simple c'est la valeur de la référence qui est passée ????
- En Algorithmique nous utiliserons le même principe

Fonctions/porté variable

```
Fonction plusplus(i: entier):entier
    i ← i + 1
    renvoie(i)
Fin Fonction

Algo testplusplus
    var i,k: entier
Debut
    i ← 2
    k ← plusplus(i)
Fin
```

Les deux i sont totalement disjoints

```
Fonction plusplus(i: entier):entier
    var j : entier
Debut
    i ← i + 1
    renvoie(i)
Fin Fonction

Algo testplusplus
    var j,k: entier
Debut
    j ← 2
    k ← plusplus(j)
Fin
```

Les deux j sont totalement disjoints

Fonctions/récursivité

Les fonctions peuvent s'appeler elle-même => appels récursifs

Itératif

```
Fonction fac(i: entier): entier
    Var res: entier
Debut
    res ← 1
    tant que i > 1faire
        res ← res * i
        i ← i -1
    fin tant que
    renvoi res
Fin
```

Récursif

```
Fonction fac(i: entier): entier
    Var res: entier
Debut
    si i=0 ou i=1 alors
        res ← 1
    Sinon
        res ← i * fac(i-1)
    Fin si
    renvoi res
Fin
```

Fonctions/récursivité

Les fonctions peuvent s'appeler elle-même => appels récursifs

```
Fac(3) <=
    3 *
    fac(2) <=        2*
                     fac(1) <= 1

Fac(3) <=
    3 *
    fac(2) <=        2*
                      1

Fac(3) <=
    3 *
    fac(2) <=        2

Fac(3) <=
    3 *
    2

Fac(3) <= 6
```

Récursif

```
Fonction fac(i: entier): entier
    Var res: entier
Debut
    si i=0 ou i=1 alors
            res ← 1
    Sinon
            res ← i * fac(i-1)
    Fin si
    renvoi res
Fin
```

=>

Utilisation pour les types de données

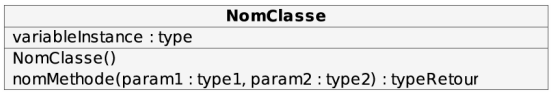
Fonctions

- Les fonctions de sont pas directement liées aux variables, comme rendre le tout cohérent

=>

Classes et les objets

Classes et Objets



Classes et Objets

Classe NomClasse
var variableInstance : type
...
Methode NomClasse()
... // le constructeur qui porte le nom de la classe
Fin Methode
Methode nomMethode(param1 : type1,
param2 : type 2) : typeRetour
Fin Methode
Fin Classe

```
public class NomClasse
{
type variableInstance ;
public NomClasse()
{
}
public typeRetour
    nomMethode(type1 param1
, type2 param2)
{
}
}
```

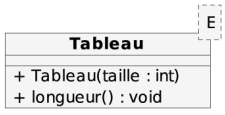
Classes et Objets

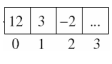
```
Algo Utilisation
    var p1 : Personne
    // DÉCLARATION
Début
    p1 ← nouveau Personne
       ("Martin","
    Martin", nouveau Date(...))
    // CRÉATION  INSTANTIATION
    p1.getNom()
    // UTILISATION DE MÉTHODES
Fin
```

```
public class Utilisation
{
public static void main(String[ ]
    args)
{
Personne p1 ;
p1 = new Personne
    ("Martin","Martin"
,new Date(12/02/85)) ;
System.out.println("le nom est : "
+ p1.getNom()) ;
}
}
```

Tableau

- Un tableau est une structure de donnée permettant de représenter une collection de valeurs. Une valeur est repérée par un nom et un (ou plusieurs) indices(s). Les indices ont des valeurs appartenant à un ensemble fini, en général un sous-ensemble des entiers.





Tableaux

var tabEntier : **Tableau<**entier**>** ; // déclaration
tabEntier ← nouveau Tableau<entier>(10) ; // dimensionnement
tabEntier[2] ← 3 ; //accès en écriture
Ecrire(tabEntier[2]) ; //accès en lecture
Ecrire(tabEntier.longueur()) ; //accès à la longueur