



## TD Modules

### Résumé

Ce TD a pour but d'aborder pourquoi et comment découper un algorithme en modules (morceaux d'algorithmes) et en méthodes (morceaux de code).

<b>1</b>	<b>Découper du code</b>	<b>2</b>
1.1	Découper du code : modules ou méthodes . . . . .	2
<b>2</b>	<b>Passage de paramètres</b>	<b>2</b>
2.1	Module . . . . .	2
2.2	Appel de module . . . . .	3
<b>3</b>	<b>Module renvoyant une valeur</b>	<b>4</b>
3.1	En algo . . . . .	4
<b>4</b>	<b>Méthodes et appel de méthodes</b>	<b>6</b>
4.1	Méthode . . . . .	6
4.2	Appel de méthode . . . . .	6
4.3	Exemples . . . . .	7
4.4	Commenter une méthode . . . . .	8
4.5	Un exemple complet . . . . .	12
<b>5</b>	<b>Exercices</b>	<b>12</b>
5.1	Compréhension d'algorithme . . . . .	13
5.2	Compréhension de codes Java . . . . .	14
5.3	À vous de jouer... . . . .	15



# 1 Découper du code

## 1.1 Découper du code : modules ou méthodes

Pourquoi ?

- Pour le réutiliser
- Pour scinder la difficulté
- Pour faciliter le déverminage
- Pour accroître la lisibilité
- Pour diviser le travail

Comment ?

- Il existe un nom qui décrit tout ce qu'il fait
- Il résout un sous-problème bien précis
- Il est fortement documenté
- Il est le plus général possible
- Il tient sur une page

En algo, nous parlerons de **module**. En Java nous parlerons de **méthode**.

# 2 Passage de paramètres

## 2.1 Module

Pour pouvoir faire communiquer les modules entre eux, il faut les équiper d'une «interface» de transmission des variables appelée l'**en-tête** du module et qui contient une déclaration de variables qu'on appellera ici **paramètres** du module.

Les variables accompagnées d'une flèche vers le bas ( $\downarrow$ ) sont des **paramètres d'entrée** qui reçoivent des **données** au début de l'exécution du module (données qui ne doivent donc plus être lues) tandis que celles accompagnées d'une flèche vers le haut ( $\uparrow$ ) sont des **paramètres de sortie** qui permettent de renvoyer des résultats à l'**issue** de l'exécution du module (résultats qui ne doivent donc plus être affichés).

Par exemple, ceci donnerait pour le module max2 :

```
module max2 (a $\downarrow$ , b $\downarrow$  : réels, max $\uparrow$  : réel)
  si a > b alors
    max  $\leftarrow$  a
  sinon
    max  $\leftarrow$  b
```

```
    fin si
fin module
```

Sous cette forme, ce module est devenu «inactif », **il ne lit rien ni n’affiche rien**, mais il est cependant prêt à être utilisé. Il suffit pour cela de lui fournir les valeurs de **a** et **b** (paramètres d’entrée) pour qu’il entre en action. Une fois le maximum calculé, celui-ci est affecté à la variable **max** qui - en tant que paramètre de sortie - contiendra et renverra le résultat voulu.

## 2.2 Appel de module

Pour faire appel aux services de ce module, il suffit à présent d’écrire son nom suivi d’un nombre de variables (ou, en entrée, d’expressions) en accord avec son en-tête.

Montrons par un exemple comment le module **max3** peut faire appel au module **max2** :

```
module max3 ()
    a, b, c, temp, max : réels
    lire a, b, c
    max2( a, b, temp )
    max2( temp, c, max )
    afficher max
fin module
```

L’instruction **max2(a, b, temp)** se déroule comme suit :

- le contenu des variables **a** et **b** est affecté aux paramètres d’entrée **a** et **b** du module **max2**, puis ce module peut entrer en action ;
- à l’issue du module **max2**, la valeur du paramètre de sortie **max** est communiquée à la variable **temp**, qui contient à présent le maximum de **a** et **b**

L’instruction **max2(temp, c, max)** se déroule comme suit :

- le contenu des variables **temp** et **c** est affecté aux paramètres d’entrée **a** et **b** du module **max2**, puis ce module peut entrer en action ;
- à l’issue du module **max2**, la valeur du paramètre de sortie **max** est communiquée à la variable **max** qui contient à présent le maximum des 3 nombres de départ.

Implicitement, on a introduit ici un nouveau type de primitive, l’appel à un module qui consiste en la simple écriture de son nom dans le code. Cette

instruction est reconnue comme telle car elle ne s'apparente pas aux autres types d'instruction vus précédemment (primitives lire, afficher,  $\leftarrow$ , structures de contrôle si, selon que, tant que, pour. . .). Lorsque l'ordinateur rencontre cette instruction, il va rechercher si un module portant ce nom existe. Une fois ce module trouvé, les paramètres ad hoc lui sont communiqués et toutes les instructions le composant sont exécutées séquentiellement. Arrivé à l'issue du module, les paramètres de sortie sont communiqués au module appelant et le déroulement de celui-ci se poursuit là où il avait été interrompu.

Remarques :

- un paramètre peut être à la fois paramètre d'entrée et de sortie ; on le fait suivre alors de la double flèche  $\downarrow\uparrow$  ;
- si tous les paramètres sont en entrée, on peut omettre les flèches ;
- la présence de ces paramètres n'est pas obligatoire, on peut envisager un module sans paramètre de sortie (par ex. un module qui reçoit en entrée un nombre et dont la seule fonction est de l'afficher à l'écran), ou inversement sans paramètre d'entrée (un module qui simule un lancer de dé, et renvoie en paramètre une valeur aléatoire entre 1 et 6) ;
- pour appeler correctement un module, il faut fournir des noms de variables, des expressions ou des constantes en **même nombre** et en **même ordre** que les paramètres du module ;
- en outre, les types des variables doivent correspondre entre l'appel et l'en-tête du module ;
- ne peut être affectée à un paramètre d'entrée du module qu'une constante, une expression ou une variable préalablement affectée ;
- à un paramètre de sortie d'un module doit toujours correspondre une autre variable, jamais une constante ou une expression ;
- il faut s'assurer qu'à l'issue du module, tous les paramètres de sortie aient été affectés lors de l'exécution de ce module.

### 3 Module renvoyant une valeur

Un deuxième type de module est le module renvoyant une valeur. (On désigne aussi ce type de modules par le terme fonction).

#### 3.1 En algo

Son en-tête est du type suivant :

```
module exemple (var1 $\downarrow$  : type1, var2 $\downarrow$  : type2, . . . , varN $\downarrow$  : typeN) -> typeRes
```

Il se distingue du module précédemment étudié par la flèche de renvoi à droite, et possède les particularités suivantes :

- ce module renvoie une valeur qui n'est pas affectée à une variable de sortie ;
- à droite de la flèche de renvoi ne se trouve donc pas le nom d'un paramètre de sortie, mais le **type** de la valeur renvoyée ;
- typeRes est le type de la valeur renvoyée ; en théorie, ce peut être un type simple (entier, réel, booléen, chaîne, caractère), un type structuré ou même un tableau (ces types seront vus dans les chapitres suivants). En pratique il conviendra de s'en tenir aux limitations du langage utilisé ;
- var1, . . . , varN sont les paramètres du module (aussi appelés **arguments**) ; ce sont, le plus souvent, des paramètres d'entrée, les paramètres de sortie étant plus rares dans ce type de module ;
- ces arguments deviennent automatiquement des variables locales du module ; déclarées dans son en-tête, elles ne doivent plus l'être dans la partie déclarative ;
- la valeur renvoyée est définie à la fin du module via la primitive retourner résultat, où résultat est une variable (ou plus généralement une expression) de type typeRes ;
- pour appeler ce type de module, on utilise son nom comme celui d'une variable ou dans une expression apparaissant dans le module appelant, mais jamais à gauche du signe d'affectation.

Comme précédemment, il faut veiller, lors de l'appel de ce type de module, à ce que le nombre d'arguments ainsi que leur type correspondent à ceux spécifiés dans l'en-tête.

Par exemple, ceci donnerait pour le module max2 :

```
module max2 (a↓, b↓ : réels)-> réel
  si a > b alors
    max ← a
  sinon
    max ← b
  fin si
  retourner max
fin module
```

Pour faire appel aux services de ce module, il suffit à présent d'écrire son nom suivi d'un nombre de variables (ou, en entrée, d'expressions) en accord avec son en-tête.

Montrons par un exemple comment le module max3 peut faire appel au module max2 :

```
module max3 ()  
    a, b, c, temp, max : réels  
    lire a, b, c  
    temp ← max2( a, b)  
    max ← max2( temp, c)  
    afficher max  
fin module
```

## 4 Méthodes et appel de méthodes

### 4.1 Méthode

Syntaxe :

```
public static <typeRetour> nomMéthode ([paramètre, paramètre, ...]) {  
    // instructions  
    <return résultat>;  
}
```

En algorithmique, il y a 3 passages de paramètres : en entrée ↓, en sortie ↑, en entrée-sortie ↓↑

En Java, le passage de paramètre se fait uniquement par valeur ; c-à-d que la valeur est copiée dans le paramètre. Pour les types primitifs, c'est équivalent d'un paramètre en entrée.

### 4.2 Appel de méthode

Une méthode est une **boîte noire** :



FIGURE 1 – methode.jpg

Pour l'utiliser, il faut connaître :

- son nom ;
- ce dont elle a besoin ;
- ce qu'elle retourne ;
- mais **pas comment** elle fait ;

On peut appeler une méthode

- à partir du code d'une autre classe : `NomClasse.nomMéthode(...)`
- au sein de la même classe : `nomMéthode(...)`

### 4.3 Exemples

Par exemple :

```
public static double moyenne ( double nb1, double nb2 ) {
    double moyenne = (nb1 + nb2) / 2.0;
    return moyenne;
}
```

Appel possible (si dans la même classe) :

```
double cote = moyenne(12.5, 17.5);
```

Par exemple :

```
public static int absolu ( int nb ) {
    int abs;
    if (nb<0) {
        abs = -nb;
    } else {
        abs = nb;
    }
    return abs;
}
```

Appels possibles (si dans la même classe) :

```
int resultat = absolu(4);
int ecart = -10;
int ecartAbsolu = absolu(ecart );
```

Par exemple :

```
public static void presenter (String nomPgm) {
    System.out. println ( "Programme_ "+nomPgm);
}
```

Appel possible (si dans la même classe) :

```
presenter ("moyenne_de_2_nombres");
```

Par exemple :

```
public static int lireEntier () {  
    Scanner clavier = new Scanner(System.in);  
    int nb;  
    System.out. println ("Entrez_un_nombre_entier!");  
    nb = clavier . nextInt ();  
    return nb;  
}
```

Appel possible (si dans la même classe) :

```
int nb = lireEntier ();
```

Montrons par un exemple comment écrire la méthode max2 :

```
import java.util.Scanner;  
public class Test{  
    public static double max2(double nb1, double nb2){  
        double max = nb1;  
        if (nb1 < nb2) {  
            max = nb2;  
        }  
        return max;  
    }  
}
```

## 4.4 Commenter une méthode

- Pour qui ?
  - Le programmeur qui va utiliser le code
  - Le programmeur qui va maintenir le code (peut-être vous)
- Quel type de documentation ?
  - Ce que fait la méthode/classe
  - Comment elle le fait (peut être réduit au minimum si code lisible)
- Qui est intéressé par quoi ?
  - Le programmeur-utilisateur : intéressé uniquement par le quoi
  - Le programmeur-mainteneur : intéressé par le quoi et le comment
- Où mettre la documentation ?
  - Avec le code
    - Plus facile pour le maintenir
    - Plus de chance de garder la synchronisation avec le code
  - Mais le programmeur-utilisateur n'a pas à voir le code pour l'utiliser



## literate programming

- la documentation accompagne le code
- un outil extrait cette documentation pour en faire un document facile à lire
- toute la documentation suit la même structure, le même style -> plus facile à lire

Regardons ce qui existe. Par exemple, la méthode `abs` de la classe `Math` :

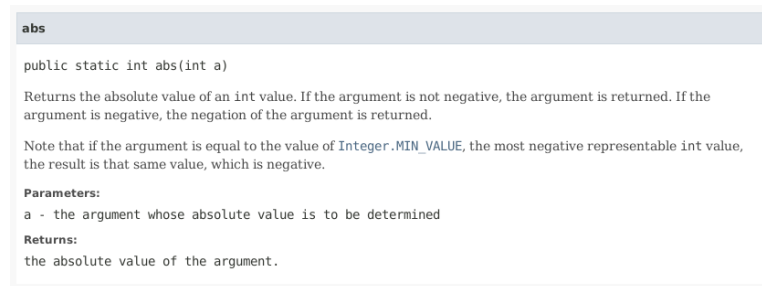


FIGURE 2 – Mathabs.png

Il est essentiel de commenter chaque méthode. C'est ce qui permet de pouvoir les utiliser facilement.

Comment écrire la nôtre :

## javadoc

- Le commentaire javadoc est identifié par `/**... */` ;
- la documentation est produite au format HTML ;
- On commente essentiellement
  - la classe : rôle et fonctionnement
  - les méthodes publiques : ce que ça fait, paramètres et résultats
- Le commentaire se met **juste au dessus** de ce qui est commenté.

## Tag

On utilise des **tags** pour identifier certains éléments. Les plus courants :

- `@param` : décrit les paramètres
- `@return` : décrit ce qui est retourné
- `@throws` : spécifie les exceptions lancées
- `@author` : note sur l'auteur

```
/**  
 * Donne la racine carree d un nombre.
```

```

* @param nb le nombre dont on veut la racine carree .
* @return la racine carree du nombre.
* @throws IllegalArgumentException si le nombre est negatif .
*/
public static double sqrt( double nb ) {
    if(nb<0) {
        throw new IllegalArgumentException("Nombre_negatif");

    return Math.sqrt(nb);
}

```

Exemple : la valeur absolue

```

/**
* Calcul de la valeur absolue .
* @param nb le nombre dont on veut la valeur absolue .
* @return la valeur absolue de <code>nb</code>
*/
public static int absolu ( int nb ) {
    int abs;
    if (nb<0) {
        abs = -nb;
    } else {
        abs = nb;
    }
    return abs;
}

```

- Les types sont déduits de la signature et ajoutés à la documentation.
- La première phrase (terminée par un .) sert de résumé.

## html

La documentation peut contenir des balises HTML.

```

/**
* Indique si l annee est bissextile . Pour rappel :
* <ul>
* <li>Une annee qui n est pas divisible par 4 n est pas bissextile (ex: 2009)</li>
* <li>Une annee qui est divisible par 4</li>
* </ul>
* <li>est en general bissextile (ex: 2008)</li>
* <li>sauf si c est un multiple de 100 mais pas de 400 (ex: 1900, 2100)</li>
* <li>les multiples de 400 sont donc bien bissextiles (ex: 2000, 2400)</li>
* </ul>
* Plus formellement, <code>a</code> est bissextile si et seulement si <br/>
* <code>a MOD 400 = 0 OU (a MOD 4 = 0 ET a MOD 100 != 0)</code>
* @param annee l annee dont on se demande si elle est bissextile
* @return vrai si l annee est bissextile
*/

```

Ce qui donne :

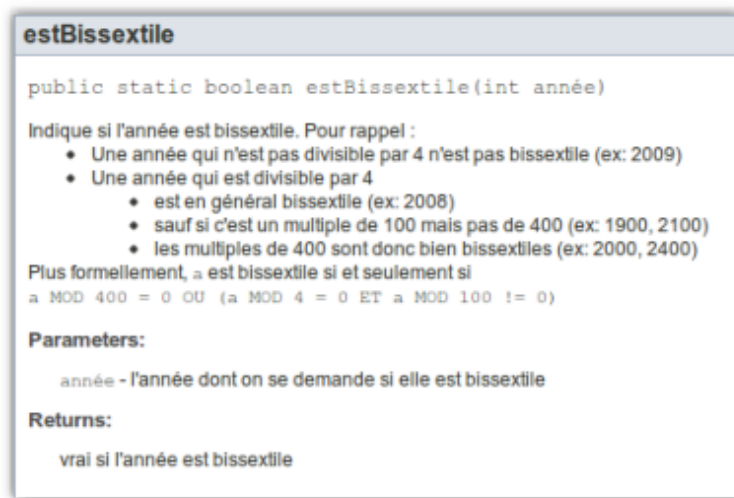


FIGURE 3 – javadocBissextile.png

## Produire la documentation

La commande javadoc :

- javadoc Temps.java
- javadoc \*.java
- javadoc -d doc \*.java
- javadoc -charset utf-8 \*.java
- . . . et beaucoup d'autres options (cf. documentation de javadoc)

## Une bonne documentation

Une bonne javadoc décrit le quoi mais jamais le comment -> ne jamais parler de ce qui est privé.

Mauvais exemples :

- On utilise un for pour parcourir le tableau.
- Pour aller plus vite, on stocke le **prix hors tva** dans une variable temporaire.

Ne pas écrire ce que javadoc écrit lui-même :

Mauvais exemples :

- nb - un entier qui ...
- La méthode sqrt ...
- Cette méthode ne retourne rien.

Pour en savoir plus : <http://www.oracle.com/technetwork/articles/java/index-137868.html> ([www.oracle.com/technetwork/articles/java/index-137868.html](http://www.oracle.com/technetwork/articles/java/index-137868.html))

## 4.5 Un exemple complet

```
import java. util .Scanner;
public class MaxEntiers {
    /**
     * Donne le maximum de 2 nombres.
     * @param nb1 le premier nombre.
     * @param nb2 le deuxieme nombre.
     * @return la valeur la plus grande entre <code>nb1</code> et <code>nb2</code>
     */
    public static int max ( int nb1, int nb2 ) {
        int max=0;
        if (nb1 > nb2) {
            max = nb1;
        } else {
            max = nb2;
        }
        return max;
    }

    /**
     * Lit un nombre entier .
     * Le nombre est lu sur l entree standard ( le clavier ).
     * @return le nombre entier lu .
     */
    public static int lireEntier () {
        Scanner clavier = new Scanner(System.in);
        System.out. println ( "Entrez_un_nombre_entier!");
        return clavier . nextInt ();
    }

    /**
     * Affiche le maximum de 2 nombres entres au clavier.
     * @param args pas utilise.
     */
    public static void main ( String [] args ) {
        int max; // Le max des nombres lus
        int nb1, nb2; // Chacun des nombres lus
        nb1 = lireEntier ();
        nb2 = lireEntier ();
        max = max(nb1,nb2);
        System.out. println ( "max_=_ " + max);
    }
}
```

## 5 Exercices

Maintenant, mettons tout ça en pratique.

## 5.1 Compréhension d'algorithme

Pour ces exercices, nous vous demandons de comprendre des algorithmes donnés.

### Compréhension

Que vont-ils afficher ?

```
— module ex1 ()
    x, y : entiers
    addition(3, 4, x)
    afficher x
    x ← 3
    y ← 5
    addition(x, y, y)
    afficher y
fin module

module addition(a↓, b↓, c↑ : entiers)
    somme : entier
    somme ← a + b
    c ← somme
fin module

— —
— module ex2 ()
    a, b : entiers
    addition(3, 4, a)
    afficher a
    a ← 3
    b ← 5
    addition(b, a, b)
    afficher b
fin module

module addition(a↓, b↓, c↑ : entiers)
    somme : entier
    somme ← a + b
    c ← somme
fin module

— —
— module ex3 ()
    a, b, c : entiers
    calcul(3, 4, c)
```

```

    afficher c
    a ← 3
    b ← 4
    c ← 5
    calcul(b, c, a)
    afficher a, b, c
fin module

module calcul(a↓, b↓, c↑ : entiers)
    a ← 2*a
    b ← 3*b
    c ← a+b
fin module

-----
— module ex4 ()
    a, b, c : entiers
    a ← 3
    b ← 4
    c ← f(b)
    afficher c
    calcul2(a, b, c)
    afficher a, b, c
fin module

module calcul2 (a↓, b↓, c↑ : entiers)
    a ← f(a)
    c ← 3*b
    c ← a+c
fin module

module f (a↓ : entier) → entier
    b : entier
    b ← 2*a+1
    retourner b
fin module

-----

```

## 5.2 Compréhension de codes Java

Pour ces exercices, nous vous demandons de comprendre des codes Java donnés.

## Compréhension

Que vont-ils afficher si à chaque fois les deux nombres lus au départ sont successivement 2, 3 et 4 ?

```
import java.util.Scanner;
public class Exercice1 {
    public static void main(String [] args) {
        Scanner clavier = new Scanner(System.in);
        int nb1 = clavier.nextInt();
        int nb2 = clavier.nextInt();
        int nb3 = clavier.nextInt();
        if (nb1 < nb2){
            System.out.println(nb1);
        } else {
            System.out.println(nb2);
        }
    }
}
```

```
import java.util.Scanner;
public class Exercice2 {
    public static void main(String [] args) {
        Scanner clavier = new Scanner(System.in);
        int nb1 = clavier.nextInt();
        int nb2 = clavier.nextInt();
        int nb3 = clavier.nextInt();
        if (nb1 > nb2 && nb1 > nb3){
            System.out.println(nb1);
        } else {
            if (nb2 > nb3){
                System.out.println(nb2);
            } else {
                System.out.println(nb3);
            }
        }
    }
}
```

```
import java.util.Scanner;
public class Exercice3 {
    public static void main(String [] args) {
        Scanner clavier = new Scanner(System.in);
        int nb1 = clavier.nextInt();
        int nb2 = clavier.nextInt();
        int nb3 = clavier.nextInt();
        switch (nb1){
            case 1 : System.out.println("premier"); break;
            case 2 : System.out.println("deuxieme"); break;
            case 3 : System.out.println("troisieme"); break;
            default : System.out.println("pas dans le trio");
        }
    }
}
```

```

import java.util.Scanner;
public class Exercice3 {
    public static void main(String [] args) {
        Scanner clavier = new Scanner(System.in);
        int nb1 = clavier.nextInt();
        int nb2 = clavier.nextInt();
        int nb3 = clavier.nextInt();
        switch (nb1){
            case 1 : System.out.println("premier");
            case 2 : System.out.println("deuxieme");
            case 3 : System.out.println("troisieme");
            default : System.out.println("pas_dans_le_trio");
        }
    }
}

```

### 5.3 À vous de jouer...

Il est temps de se lancer et d'écrire vos premiers modules et programmes Java correspondant. Voici quelques conseils pour vous guider dans la résolution de tels problèmes :

- il convient d'abord de bien comprendre le problème posé ; assurez-vous qu'il est parfaitement spécifié ;
- déclarez ensuite les variables (et leur type) qui interviennent dans l'algorithme ; les noms des variables risquant de ne pas être suffisamment explicites ;
- mettez en évidence les variables «données », les variables «résultats »et les variables de travail ;
- n'hésitez pas à faire une ébauche de résolution en français avant d'élaborer l'algorithme définitif pseudo-codé.
- Écrivez la partie algorithmique **AVANT** de vous lancer dans la programmation en Java.

Écrivez les algorithmes et codez les programmes Java correspondant qui

1. étant donné deux nombres quelconques, recherche et affiche le plus grand des deux. Attention ! On ne veut pas savoir si c'est le premier ou le deuxième qui est le plus grand mais bien quelle est cette plus grande valeur. Le problème est donc bien défini même si les deux nombres sont identiques.
2. étant donné trois nombres quelconques, recherche et affiche le plus grand des trois.
3. affiche un message indiquant si un entier est strictement négatif, nul ou strictement positif.



4. étant donné trois nombres, recherche et affiche si le premier des trois appartient à l'intervalle donné par le plus petit et le plus grand des deux autres (bornes exclues). Qu'est-ce qui change si on inclut les bornes ?
5. étant donné une équation du deuxième degré, déterminée par le coefficient de  $x^2$ , le coefficient de  $x$  et le terme indépendant, recherche et affiche la (ou les) racine(s) de l'équation (ou un message adéquat s'il n'existe pas de racine réelle).
6. à partir d'un moment exprimé par 2 entiers, heure et minute, affiche le moment qu'il sera une minute plus tard.
7. vérifie si une année est bissextile. Pour rappel, les années bissextiles sont les années multiples de 4. Font exception, les multiples de 100 (sauf les multiples de 400 qui sont bien bissextiles). Ainsi 2012 et 2400 sont bissextile mais pas 2010 ni 2100.
8. Dans une rue où se pratique le stationnement alternatif, du 1 au 15 du mois, on se gare du côté des maisons ayant un numéro impair, et le reste du mois, on se gare de l'autre côté. Écrivez un algorithme et le code java correspondant qui, sur base de la date du jour et du numéro de maison devant laquelle vous vous êtes arrêté, indique si vous êtes bien stationné ou non.