

R1.01

INITIATION AU DÉVELOPPEMENT

Cours 2 : fonctions et procédures

(résumé dans Mémento 2)





Hervé Blanchon & Anne Lejeune

Université Grenoble Alpes

IUT 2 – Département Informatique

Sommaire

Introduction

-  Rappel : procédure `main`
-  Distribution du travail
-  Définition de procédure ou de fonction
-  Notion de paramètre (formel vs. effectif)


Fonctions

Procédures




Appel de fonctions ou procédures dans une procédure `main`

INTRODUCTION





Rappel : procédure `main(...)`




-  Un programme exécutable contient une **procédure `main`** qui est exécutée en premier


```
public static void main(String[] args) {  
    // séquence d'instructions  
    ...  
}
```

-  Les instructions de cette **procédure** peuvent utiliser
 -  d'autres **procédures** (non `main`)
 -  des **fonctions**

Distribution du travail

-  Une procédure **main** peut utiliser des **fonctions** ou des **procédures**...
 -  définies dans des bibliothèques importées dans sa classe
 -  définies dans sa classe
 -  définies dans d'autres classes du même projet java (vu plus tard)

-  Une **fonction** ou une **procédure** (non **main**) est un sous-programme permettant de :
 -  rendre le programme principal plus lisible
 -  réutiliser des traitements

-  Pour utiliser une fonction ou une procédure (non **main**), il faut l'**appeler** en lui fournissant les informations dont elle a besoin

Distribution du travail - Exemple

```
import java.util.Scanner;

public class Exemple_Main {

    private static void salutations(String nomPers){
        //{} => {Un message d'accueil est affiché}
        System.out.println("Bonjour " + nomPers);
    }

    private static int ageDe(int anNaiss, int anCour){
        //{}=> {résultat = anCour-AnNaiss}
        return anCour-anNaiss;
    }

    public static void main(String[] args) {
        Scanner lecteur = new Scanner(System.in);
        String nom;
        int anneeNaissance, age;
        final int anneeCourante = 2021;
        System.out.print("Quel est votre nom ? ");
        nom = lecteur.nextLine();
        salutations(nom); // appel de la procédure saluation
        System.out.println("Quelle est votre année de naissance ?");
        anneeNaissance = lecteur.nextInt();
        lecteur.nextLine();
        // appel de la fonction ageDe, son résultat initialise age
        age = ageDe(anneeNaissance, anneeCourante);
        System.out.println("Cette avez ou aurez " + age + " ans");
    }
}
```

procédure utilisée par
la procédure principale

fonction utilisée par la
procédure principale

Définition de procédure ou de fonction

Une fonction ou une procédure est définie par...

Son **entête**

 sa visibilité :

 **private** ou **public** (le mot-clé choisi dépend des besoins)

 **static** (optionnel, dépend des besoins - ce sera vu plus tard)


 sa nature (fonction vs. procédure) :

 **type_résultat** (**fonction**) / **void** (**procédure**)

 son nom

 ses **paramètres formels**

Sa **spécification** (le service rendu)

 précondition : conditions à respecter pour son utilisation

 postcondition : résultat produit (**fonction**) / ce qui a été fait (**procédure**)

Son **corps** (son fonctionnement)

 bloc d'instructions

Définition de procédure ou de fonction

Notation de la spécification (pour R1.01)

```
// {précondition} => {postcondition}
```

Modèle d'une fonction


```
{private|public} [static] type_rés Nom([type_param nom_param,...] {  
  // {précondition} => {postcondition}  
  bloc d'instructions  
}
```

[] : optionnel
{ ... | ... } : choix
[, ...] ou [...] : répétition
de l'élément précédent

Modèle d'une procédure

```
{private|public} [static] void Nom([type_param nom_param,...] {  
  // {précondition} => {postcondition}  
  bloc d'instructions  
}
```


Notion de paramètre formel (1/2)

 Un paramètre d'une fonction ou d'une procédure définit dans son entête s'appelle :


un « **paramètre formel** »

 Quand on écrit le corps d'une fonction ou d'une procédure :

 on ignore la **valeur** d'un paramètre

 on connaît uniquement son **type**

Notion de paramètre formel (2/2)

 Un paramètre **formel** d'une fonction ou d'une procédure est considéré comme une « **variable locale** »


 Cette variable locale :

 est **visible** (utilisable) seulement à l'intérieur de la fonction ou de la procédure






 a un comportement qui dépend de son type

 Un paramètre formel de **type primitif** ou **String** :

 peut être **consulté** et/ou **modifié**

 comme il est considéré comme une variable locale, sa modification n'est visible que dans le corps de la fonction ou de la procédure

Notion de paramètre effectif

-  Lors de l'appel (utilisation) d'une **fonction** ou d'une **procédure**, les paramètres **formels** doivent être remplacés par des paramètres **effectifs**
 -  Il y a autant de paramètres effectifs que de paramètres formels
 -  L'ordre et le type des paramètres effectifs doit respecter l'ordre et le type des paramètres formels
 -  Si un paramètre formel est de **type primitif** ou **String**, alors le paramètre effectif peut être une *variable* ou une *valeur* (du bon type)
-
-  Exemples d'appels (vus plus loin)

FONCTIONS

Fonction

 Une **fonction** retourne un résultat

cf. Fonctions

 Analogie avec une fonction mathématique

 $f(x) \rightarrow y ; g(x, y) \rightarrow z$

 f appliquée au paramètre x retourne le résultat y

 g appliquée aux paramètres x et y retourne le résultat z

Fonction - Exemples d'entêtes

```
private static boolean estMajeur(int age) {  
    // {} => {résultat = true si age >= 18, false sinon}  
}
```



définition d'une fonction de nom `estMajeur` qui :



a un paramètre **entier** de nom `age`



retourne un **booléen** dont la valeur est égale à :



`true` si `age` a une valeur supérieure ou égale à 18



`false` si `age` a une valeur inférieure à 18

```
private static int minimum(int x, int y) {  
    // {} => {résultat = le plus petit parmi x et y}  
}
```



définition d'une fonction de nom `minimum` qui :



a deux paramètres de type **entier** nommés `x` et `y`



retourne un **entier** dont la valeur est égale au à :




la valeur de `x` ou `y` en cas d'égalité



la plus petite des valeurs de `x` ou `y` en cas de différence

Fonction - Retour du résultat

 On dispose de l'instruction **return**

 Cette instruction permet de retourner une valeur de type compatible avec celui défini par l'en-tête de la fonction


Exemples

 `return 3;` // si la fonction doit renvoyer un entier

 `return "Monsieur";` // si la fonction doit renvoyer une chaîne

 `return res;` // si la variable res est du bon type

 `return 3 + x;` // si x est un entier et que la fonction doit
// retourner un entier




 `return (a < 5) | (a > 15);` // si a est un entier et que la fonction doit
// retourner un booléen

 Cette instruction interrompt l'exécution de la fonction

 Aucune des instructions qui suivent le **return** ne sera exécutée


Fonction - Utilisation du résultat

 Le résultat d'une fonction peut être utilisé

-  pour l'afficher
-  pour initialiser ou changer la valeur une variable
-  dans une expression

 Exemples

 `if (estMajeur(age) == true) {...`
`// le résultat de estMajeur est utilisé comme condition du if //`
`(expression booléenne)`

 `System.out.println("a = " + a + ", b = " + b);`
`System.out.println(" -> le plus petit est " + minimum(a, b));`

`// affichage de la valeur de a et de celle de b, puis`
`// affichage de la valeur retournée par la fonction minimum`

 `int c = 10 + minimum(a, b);`
`// utilisation du résultat de la fonction minimum dans une`
`// somme puis affectation`


```
private static boolean estMajeur(int age) {  
    // spécification {} => {résultat = true si age >= 18,  
    //                                     false sinon  
    return age >= 18;  
}
```

```
private static int minimum(int x, int y) {  
    // spécification {} => {résultat = le plus petit  
    //                                     parmi x et y}  
    if (x < y) {  
        return x;  
    } else {  
        return y;  
    }  
}
```

```
import java.util.Scanner;


public class Fonctions_Main {
    private static boolean estMajeur(int age) {
        ...
    }

    private static int minimum(int x, int y) {
        ...
    }

    public static void main(String[] args) {
        Scanner lecteur = new Scanner(System.in);          int age;
        System.out.print("Saisir un âge (entier positif) : ");
        age = lecteur.nextInt(); // initialisation par lecture
        lecteur.nextLine();
        if (estMajeur(age) == true) { // test du résultat de la fonction estMajeur
            System.out.println("c'est un âge de majeur");
        } else {
            System.out.println("c'est un âge de mineur");
        }
        // affichage du résultat de la fonction minimum
        int a = 17, b = 5;
        System.out.println("a = " + a + ", b = " + b);
        System.out.println(" -> le plus petit est " + minimum(a, b));
        // utilisation du résultat de la fonction minimum pour un calcul
        int c = 10 + minimum(a, b);
        System.out.println("c = " + c);
        System.out.println("minimum de 23 et 12 : " + minimum(23, 12));
    }
}
```

PROCÉDURES

 Une **procédure** ne retourne pas de résultat *au sens d'une fonction*

 $p(x) ; q(x, y)$

 p appliquée à x fait des traitements sans retourner de résultat

 q appliquée à x et y fait des traitements sans retourner de résultat

Procédure - Exemples d'entêtes

```
private static void afficheTableDe7() {  
    // {}  
    //=> {la table de multiplication de 7 a été affichée}  
}
```



définition d'une procédure de nom afficheTableDe7 qui :



n'a pas de paramètre



affiche la table de multiplication de 7

```
private static void afficheTableDe (int x) {  
    // {}  
    //=> {la table de multiplication de x a été affichée}  
}
```



définition d'une procédure de nom afficheTableDe qui :







a un paramètre **entier** nommé x



affiche la table de multiplication de x

Procédure - Utilisation

-  Une procédure peut être utilisée
 -  pour simplifier la lecture du code
 -  pour réutiliser des traitements
 -  L'appel d'une procédure est une **instruction**

Exemples

-  appel avec une valeur immédiate

```
afficheTableDe(9);  
// la table de multiplication de 12 est affichée
```

-  appel avec une variable initialisée

```
int valeur = 17;  
afficheTableDe(valeur);  
// la table de multiplication de valeur (17) est affichée
```

LA FONCTION *minimum*

Fonction minimum dans une classe `_Main`

```
private static int minimum( int x , int y ) {  
    // {} => {résultat = le plus petit parmi x et y}  
    if ( x < y ) {  
        return x ;  
    } else {  
        return y ;  
    }  
    // on retourne bien un résultat dans tous les cas  
}
```

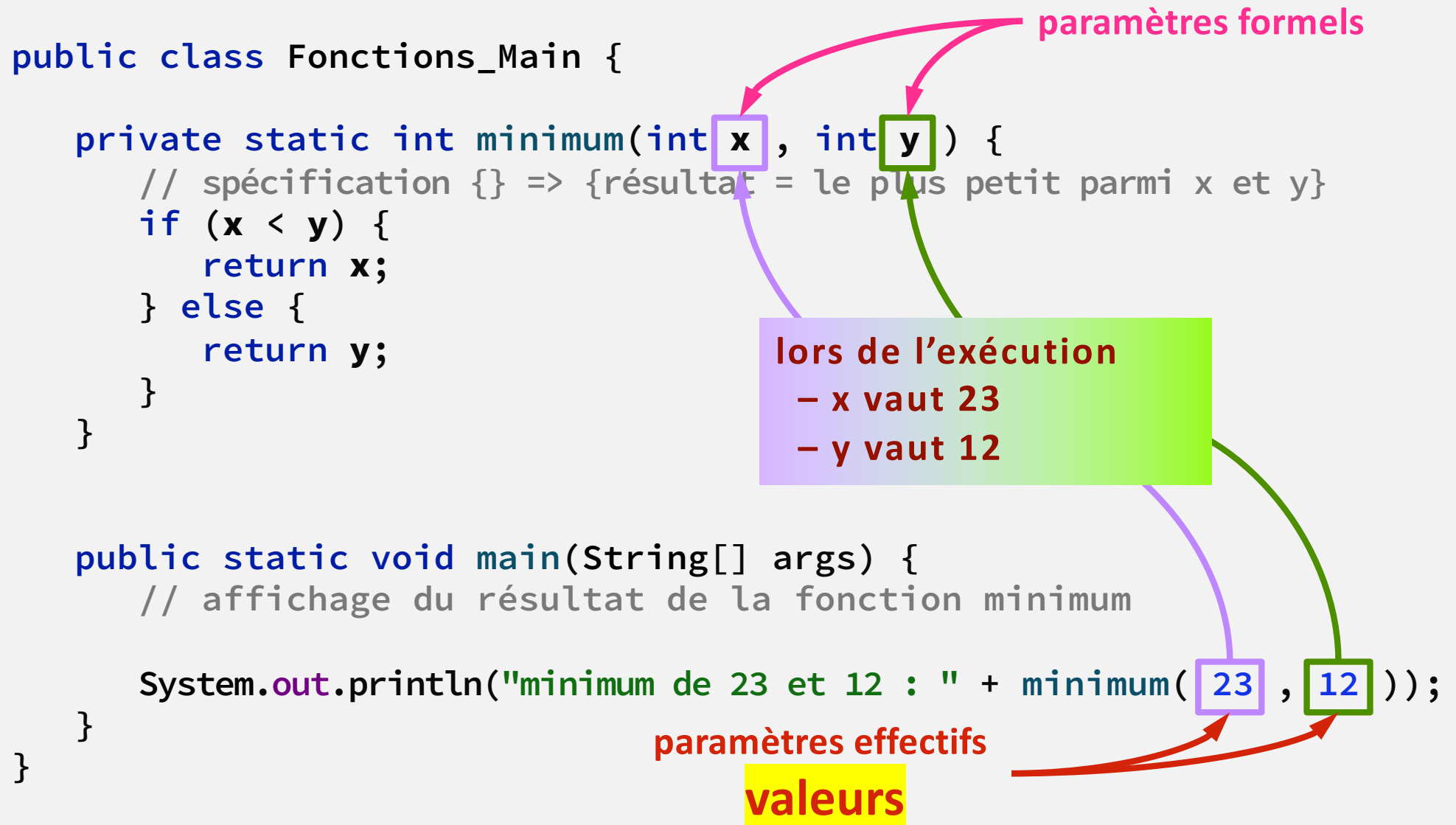
type du résultat

paramètres formels

utilisation de la valeur des paramètres formels lors de l'appel

le type annoncé (entête) et le type retourné (return) doivent être les mêmes (ou compatibles)

Classe _Main contenant minimum



Classe _Main contenant minimum

```
public class Fonctions_Main_Cours {  
  
    private static int minimum(int x, int y) {  
        // spécification {} => {résultat = le plus petit parmi x et y}  
        if (x < y) {  
            return x;  
        } else {  
            return y;  
        }  
    }  
  
    public static void main(String[] args) {  
        // affichage du résultat de la fonction minimum  
        int a = 17, b = 5;  
        System.out.println("a = " + a + ", b = " + b);  
  
        System.out.println(" -> le plus petit est " + minimum(a, b));  
    }  
}
```

paramètres formels

lors de l'exécution
- x vaut 17
- y vaut 5

paramètres effectifs
variables initialisées

The diagram illustrates the parameter passing mechanism. In the `minimum` method, `x` and `y` are formal parameters. In the `main` method, `a` and `b` are actual parameters. Arrows show that `a` is passed to `x` and `b` is passed to `y`. A central box states that during execution, `x` holds the value 17 and `y` holds the value 5, which are the values of `a` and `b` at that point.

Classe _Main contenant minimum

```
public class Fonctions_Main {
```

```
    private static int minimum(int x, int y) {  
        // spécification {} => {résultat = le plus petit parmi x et y}  
        if (x < y) {  
            return x;  
        } else {  
            return y;  
        }  
    }
```

lors de l'exécution
– x vaut 17
– y vaut 5

```
    public static void main(String[] args) {  
        // affichage du résultat de la fonction minimum  
        int a = 17, b = 5;  
  
        int c = 10 + minimum(a, b);  
  
        System.out.println("c = " + c);  
    }  
}
```

paramètres formels

paramètres effectifs
variables initialisées



Code complet

Code de la classe

```

1  public class Fonctions_Main_Cours {
2
3      private static int minimum(int x, int y) {
4          // spécification {} => {résultat = le plus petit parmi x et y}
5          if (x < y) {
6              return x;
7          } else {
8              return y;
9          }
10     }
11
12     public static void main(String[] args) {
13         // affichage du résultat de la fonction minimum
14         System.out.println("minimum de 23 et 12 : " + minimum(23, 12));
15         int a = 17, b = 5;
16         System.out.println("a = " + a + ", b = " + b);
17         System.out.println("-> le plus petit est " + minimum(a, b));
18         int c = 10 + minimum(a, b);
19         System.out.println("c = " + c);
20     }
21 }

```

Trace

```

14  minimum de 23 et 12 : 12
16  a = 17, b = 5
17  -> le plus petit est 5
19  c = 15

```

PROCÉDURE AfficheTableDe

Classe _Main contenant afficheTableDe7

```
public class Procedures_Main {  
  
    private static void afficheTableDe7() {  
        // spécification {} => {La table de multiplication  
        //                               de 7 a été affichée}  
        System.out.println("Table de 7 :");  
        for (int i = 1; i <= 10; i++) {  
            // i va parcourir l'intervalle [1; 10]  
            // écrire la ligne courante  
            System.out.println(i + " x 7 = " + (i * 7));  
        }  
    }  
  
    public static void main(String[] args) {  
        afficheTableDe7();  
    }  
}
```

appel de la procédure **sans paramètre**
c'est une instruction

Classe _Main contenant afficheTableDe

```
public class Procedures_Main_Cours {  
  
    private static void afficheTableDe(int tableDe) {  
  
        // spécification {} => {La table de multiplication  
        //                               de tableDe a été affichée}  
        System.out.println("Table de " + tableDe + " :");  
        for (int i = 1; i <= 10; i++) {  
            // i va parcourir l'intervalle [1; 10]  
            // écrire la ligne courante  
            System.out.println(i + " x " + tableDe + " = " + (i * tableDe));  
        }  
    }  
  
    public static void main(String[] args) {  
  
        afficheTableDe(12);  
  
    }  
}
```

paramètre formel

lors de l'exécution
– tableDe vaut 12

paramètre effectif
valeur

Classe _Main contenant afficheTableDe

```
public class Procedures_Main_Cours {  
  
    private static void afficheTableDe(int tableDe) {  
        // spécification {} => {La table de multiplication  
        //                               de tableDe a été affichée}  
        System.out.println("Table de " + tableDe + " :");  
        for (int i = 1; i <= 10; i++) {  
            // i va parcourir l'intervalle [1; 10]  
            // écrire la ligne courante  
            System.out.println(i + " x " + tableDe + " = " + (i * tableDe));  
        }  
    }  
  
    public static void main(String[] args) {  
        int valeur = 17;  
  
        afficheTableDe(valeur);  
    }  
}
```

paramètre formel

lors de l'exécution
– tableDe vaut 17

paramètre effectif
variable initialisée



Code complet

Code de la classe

```

1  public class Procedures_Main_Cours {
2      private static void afficheTableDe(int  tableDe ) {
3          // ...
4          System.out.println("Table de " + tableDe + " :");
5          for (int i = 1; i <= 10; i++) {
6              // i va parcourir l'intervalle [1; 10]
7              // écrire la ligne courante
8              System.out.println(i + " x " + tableDe + " = " + (i * tableDe));
9          }
10     }
11     public static void main(String[] args) {
12         afficheTableDe(12);
13         int valeur = 17;
14         afficheTableDe(  valeur  );
15     }
16 }

```

Trace

```

12  Table de 12 :
    1 x 12 = 12
    2 x 12 = 24
    3 x 12 = 36
    4 x 12 = 48
    5 x 12 = 60
    6 x 12 = 72
    7 x 12 = 84
    8 x 12 = 96
    9 x 12 = 108
   10 x 12 = 120

```

```

14  Table de 17 :
    1 x 17 = 17
    2 x 17 = 34
    3 x 17 = 51
    4 x 17 = 68
    5 x 17 = 85
    6 x 17 = 102
    7 x 17 = 119
    8 x 17 = 136
    9 x 17 = 153
   10 x 17 = 170

```