Chapitre 5 : Les sous-programmes

Juan Carlos QUEZADA

INSA Strasbourg

Méthodes

Définition

- Une méthode décrit une suite d'instructions qui forment un sous-programme
- Ces instructions réalisent un traitement ou élaborent un résultat à partir des paramètres formels

Définition de méthode

- un nom pour le désigner
- un ensemble de paramètres (paramètres formels)
  - en entrée : données utilisées (de quoi j'ai besoin)
  - en sortie : résultats

Type de méthodes

On peut distinguer deux types de méthodes :

Procédures

Qui exécutent une série d'instructions mais qui ne fournissent aucun résultat.

Les fonctions

Qui fournissent un résultat (paramètre de sortie)

Procédures

Définition de procédures

Sous-programme <nom procédure>
  Entrée : <paramètres formels>
  Sortie : vide
  Lexique :
    <déclaration de variables locales>
  Instructions :
    <instructions>
  Fin <nom procédure>

Appel de procédure

Définition

- l'appel d'une procédure est une instruction, et se place donc sur une ligne à part
- pour chaque paramètre formel (indiqué dans la définition), il faut fournir un paramètre réel correspondant
- la correspondance se fait dans l'ordre de définition
- note : comme le type des paramètres est indiquée lors de la définition, on ne le répète pas lors de l'appel

Syntaxe

<nom de la procédure>(<paramètres réels>)

Fonctions

Définition

- une fonction est un sous-programme où un paramètre en sortie à une importance particulière : c'est la valeur de la fonction
- particulièrement adapté aux sous-programmes ayant des paramètres en entrée, et un seul paramètre en sortie
- utilisation plus intuitive que pour une procédure

Fonctions

Définition de fonction

Sous-programme <nom fonction>
  Entrée : <paramètres formels>
  Sortie : <type de la fonction>
  Lexique :
    <déclaration de variables locales>
  Instructions :
    <instructions>
  Retourner <variable>
  Fin <nom fonction>

Fonctions

Définition

- l'appel d'une fonction se place dans une expression.
- après calcul de la fonction, tout se passe comme si la valeur indiquée dans l'instruction retourne en fin de définition prenait la place de l'appel de la fonction dans l'algorithme principal
- la correspondance entre les paramètres formels et réels se fait sinon comme pour une procédure

Appel à une fonction

<variable> ← <nom fonction>(<paramètres réels>)

Utilisation dans l'algorithme principal

Définition de l'algorithme principal

Algorithme principal
  Lexique :
    <déclaration de variables locales>
  Instructions :
    <instructions et appel au sous-programme>
  Fin algorithme principal

Passage de paramètres

Passage par valeur :

La valeur du paramètre réel est copiée dans le paramètre formel lors de l'appel

- modifier le paramètre formel dans le sous-programme ne modifie pas la valeur du paramètre réel dans le programme appelant
- adapté aux paramètres en entrée

Passage par référence :

Le paramètre formel fait référence à la valeur du paramètre réel

- modifier le paramètre formel dans le sous-programme modifie la valeur du paramètre réel dans le programme appelant
- adapté au paramètres en sortie et en entrée/sortie

Les Sous-programmes en JAVA

Classes utilitaires

- Un programme en JAVA est constitué d'un ensemble de classes. Généralement, une classe décrit un nouveau type d'objet. Mais une classe peut aussi servir à regrouper une collection de sous-programmes, sans définir un type d'objet. Une telle classe s'appelle une classe utilitaire.
- Nous pouvons écrire nos propres classes utilitaires à l'intérieur desquelles nous pouvons définir nos propres sous-programmes appelés méthodes. Il existe différents types de méthodes en fonction du type du retour.

Syntaxe

```
public class NomClasse
{
    liste-de-définitions-de-méthodes
}
```

Les Sous-programmes en JAVA

Méthodes

- Une méthode décrit une suite d'instructions qui forment un sous-programme. Ces instructions réalisent un traitement ou élaborent un résultat à partir des paramètres formels.
- En général, le bloc d'instructions d'une méthode est précédé d'une en-tête.
- Une méthode peut ne fournir aucun résultat. Le mot clé void figure alors dans son en-tête à la place du type de la valeur de retour. C'est le cas des procédures.
- Mais une méthode peut aussi fournir un résultat. Nous parlerons alors de méthode fonction.

Les Sous-programmes en JAVA

Exemple d'une procédure

```
/* affichage du double d'un entier*/
public static void affiche (int v){
    int res;
    res = 2*v;
    System.out.println("Le double vaut "+res);
}
```

Exemple d'une fonction

```
/* calcul du carré d'un entier */
public static int carre (int v){
    int res;
    res = v*v;
    return res ;
}
```

Les Sous-programmes en JAVA

Syntaxe générale

```
public static type-résultat nom-methode (déclaration-paramètres-
{
    déclaration variables locales
    liste d'instructions ;
    return expression ;
}
```

- *public* indique que la fonction peut être appelée dans d'autres classes
- *static* indique que cette fonction est utilitaire
- *type-résultat* est le type de la valeur renvoyée en résultat ou *void* si la méthode ne renvoie pas de résultat.
- *nom-methode* est le nom de la méthode, par convention, en minuscule
- *déclaration-paramètres-formels* est une liste de déclarations type identifiant séparées par des virgules. Cette liste peut, éventuellement, être vide si la méthode n'a aucun paramètre.

Les Sous-programmes en JAVA

- Nous pouvons utiliser le mot clé *final* pour nous assurer qu'un paramètre formel ne sera pas modifié par la méthode.

Exemple

```
void f (final int n, double x)
{
    ...
    n = 12; // erreur de compilation
    x = 2.5; // OK
    ...
}
```

Les Sous-programmes en JAVA

Signature d'une méthode

On appelle signature d'une méthode le début de sa description (ou son en-tête). La signature comporte notamment le type du résultat, le nom, ainsi que la liste des paramètres formels.

```
public static int carre (int v)
```

Variables locales

- Des variables locales peuvent être déclarées à l'intérieur d'une méthode. Les déclarations des variables locales se fait généralement au début de la définition de la méthode.
- Une variable locale doit être initialisée avant d'être utilisée, soit par une affectation, soit par une initialisation lors de sa déclaration.
- Une méthode ne peut accéder qu'à ses propres paramètres et à ses propres variables locales. Une variable locale ou un paramètre ne peut être accédé que par les instructions de la fonction où elle est définie.

Les Sous-programmes en JAVA

Instruction return

```
return expression ;
```

- Cette instruction termine l'exécution de la fonction. L'expression est évaluée et la valeur est communiquée à la fonction appelante.
- L'instruction return est en principe la dernière instruction de la fonction. La valeur d'expression doit être le même (ou compatible) avec le type de résultat déclaré de la fonction.
- Dans le cas d'une procédure (type du retour void) il n'y a pas d'instruction return ! !

Les Sous-programmes en JAVA

Appel de méthodes

Pour utiliser une méthode, il faut l'appeler en lui fournissant les valeurs sur lesquelles elle doit opérer. En général, l'appel d'une méthode se fait par l'expression :

```
nom-méthode (liste-expressions)
```

où `liste-expressions` est une suite d'expressions séparées par des virgules, appelées paramètres réels.

Les méthodes peuvent être appelées par d'autres méthodes de la même classe ou par des méthodes d'une classe différente. Pour appeler les méthodes d'une autre classe, par exemple de la classe utilitaire `Math`, il suffit de procéder à une importation :

```
import static java.lang.Math.* ;
double ma = sin (0) ;
```

Les Sous-programmes en JAVA

Sans importation des fonctions de la classe, il faut utiliser la notation « pointée » *nomclasse.nom-méthode* :

```
double ma = Math.sin(0) ;
```

Validité de l'appel d'une méthode

- Lorsqu'une méthode est appelée, il faut lui fournir autant de valeurs que de paramètres formels.
- Ces valeurs remplacent les paramètres formels pour l'élaboration du résultat. Chaque paramètre réel est mis en correspondance un à un avec le paramètre formel de la liste de définition de la méthode.

```
double x1 = Math.pow (2.0, 10.0) ;
```

- Le nombre de paramètres d'appel doit être égal au nombre de paramètres formels de la méthode.

```
double x2 = Math.sqrt (2.0, 10.0) ; // ERREUR
```

Les Sous-programmes en JAVA

Validité de l'appel d'une méthode

- Le type de chaque paramètre d'appel doit être compatible avec le type du paramètre formel correspondant.

```
double 32 = Math.sqrt (“2.0”) ; // ERREUR
```

- La liste d'expressions est vide lorsque la fonction n'admet pas de paramètre formel.

```
double r = Math.random() ;
```

Les Sous-programmes en JAVA

Résultat de l'appel : cas d'une fonction

- L'appel d'une fonction est une expression. La valeur de cette expression est la valeur renvoyée par la fonction appelée. Le type de cette expression est le même que le type du résultat de la fonction appelée.
- L'appel d'une fonction peut être employé chaque fois que la syntaxe du langage autorise l'écriture d'une expression de même type.

```
double x = 0.8 ;
double y ;
y = Math.pow(x, 0.33) + 5 * Math.log(10) ;
```

Exécution d'un appel : cas d'une fonction

L'exécution d'un appel de fonction donne le contrôle à la fonction appelée. Les paramètres formels prennent la valeur des paramètres d'appel. La fonction appelée exécute ses instructions, jusqu'à l'instruction *return* qui renvoie une valeur en résultat.

Les Sous-programmes en JAVA

Le cas particulier des tableaux

- Lorsqu'on transmet un nom de tableau en argument d'une méthode, on transmet en fait une copie de la référence au tableau (nom du tableau). La méthode agit alors directement sur le tableau concerné et non sur sa copie (passage de paramètres par référence et non par valeur).
- Les mêmes réflexions s'appliquent à un tableau fourni en valeur de retour. Par exemple, la méthode suivante fournirait en résultat un tableau formé des n premiers nombres entiers.

```
public static int[] suite (int n){
    int [] res = new int [n] ;
    for (int i=0 ; i<n ; i++){
        res[i] = i+1;
    }
    return res;
```

Les Sous-programmes en JAVA

Le cas particulier des tableaux

- Un appel de suite fournira une référence à un tableau dont on pourra éventuellement modifier les valeurs des éléments.

Exercices

Exercice 1

A la lecture de l'algorithme suivant :

```
public static void main (String[] args) {
    int a,k;
    for(k=-2;k<=2;k++)
    {
        a=calcul(k);
        System.out.println("a = "+calcul(a));
    }
}

public static int calcul(int x){
    int y;
    y = x*x;
    return y;
}
```

Exercices

Exercice 1

1. Quel est le paramètre formel du sous-programme calcul().
2. Quelles sont les valeurs transmises au paramètre du sous-programme calcul() lors de son appel depuis l'algorithme principal.
3. Quels sont les résultats produits par le sous-programme calcul().
4. Quelles sont les valeurs transmises à la variable a ?
5. Décrivez l'affichage réalisé par l'algorithme principal

Exercices

Exercice 2

Soit le sous-programme suivant :

```
Sous-programme calculer
Entree : x en entier
Sortie : un entier
Lexique
   r en entier
Instructions
    r <- -7*x*x+3
    retourner r
fin_calculer
```

- Écrire l'algorithme principal qui affiche le résultat du sous-programme
- Calculer(a) pour a=0.
- Transformer l'algorithme principal de façon à calculer et à afficher le résultat du sous-programme pour a entier variant de –5 à 5.

Exercices

Exercice 3

Écrire l'algorithme principal qui permet de saisir un tableau de 10 nombres entiers et qui fait appel à un sous-programme renvoyant la somme de tous les éléments du tableau. On écrira aussi le sous-programme.

Exercice 4 : Le trinôme version sous-programme

Reprendre l'exercice 5 du Chapitre 2, le structurer en sous-programmes suivants :

- Sous-programme de calcul du discriminant
- Sous-programme de calcul des solutions de l'équation

Exercice 5 : Intégration numérique

Attention

L'exercice suivant est à rendre sur Moodle pour la semaine prochaine

Préambule

Si f est une fonction continue sur un intervalle [a,b], bien souvent on ne sait pas calculer une primitive de f.
Ainsi, si l'on désire obtenir la valeur de $\int_a^b f(x)dx$ il faut parfois se contenter d'obtenir une valeur approchée à l'aide d'une méthode d'intégration numérique.
La plupart des méthodes d'intégration numérique fonctionnent sur le même principe.
On commence par couper l'intervalle [a,b] en N plus petits intervalles $[a_i,a_{i+1}]$ de longueur fixe $h = \frac{b-a}{N}$, avec $a_0 = a$ et $a_N = b$.
Puis, pour chaque intervalle $[a_i,a_{i+1}]$, on essaie d'approcher $\int_{a_i}^{a_{i+1}} f(x)dx$ de différentes manières.

Exercice 5 : Intégration numérique

Objectif

Nous proposons d'étudier les méthodes des rectangles. Selon ces méthodes $\int_{a_i}^{a_{i+1}} f(x)dx$ approchée par l'aire d'un rectangle. Il y a, dans ce cas, trois possibilités :

Exercice 5 : Intégration numérique

Méthode des rectangles à gauche

Le rectangle est construit sur a_i, a_{i+1} et $f(a_i)$. Géométriquement, le calcul de $\int_a^b f(x)dx$ sera approché par la somme de l'aire des rectangles hachurés (cf. Figure 1).

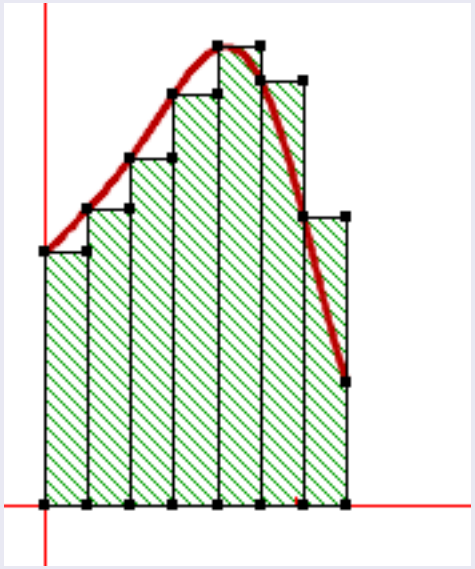

Figure – Intégration numérique par la méthode des rectangles à gauche.

Exercice 5 : Intégration numérique

Méthode des rectangle à droite

Le rectangle est construit sur a_i, a_{i+1} et $f(a_{i+1})$. Géométriquement, le calcul de $\int_a^b f(x)dx$ sera approché par la somme de l'aire des rectangles hachurés.

Méthode du point milieu (ou des tangentes)

Le rectangle est construit sur a_i, a_{i+1} et $f(a_i + a_{i+1})/2$.
Géométriquement, le calcul de $\int_a^b f(x)dx$ sera approché par la somme de l'aire des rectangles hachurés.

Exercice 5 : Intégration numérique

Expression du calcul approximatif de $\int_{a_i}^{a_{i+1}} f(x)dx$

En fonction des explications données ci-dessus donner les expressions permettant d'approcher $\int_{a_i}^{a_{i+1}} f(x)dx$ par les méthodes des rectangles à droite, des rectangles à gauche et du point milieu en fonction de a_i, a_{i+1}, $f(a_i)$, $f(a_{i+1})$ et $f((a_i + a_{i+1})/2)$

Expression du calcul approximatif de $\int_a^b f(x)dx$

A partir de la question précédente, de la relation entre a_i et a_{i+1} ainsi que des valeurs particulières de a_i pour i=0 et i=N, écrire les expressions permettant de calculer, par les différentes méthodes des rectangles (rectangle à gauche, rectangle à droite, point milieu), $\int_a^b f(x)dx$ en fonction de h, et $f(*)$ où * est une expression faisant intervenir a, h et i

Exercice 5 : Intégration numérique

Algorithme et programme de calcul approximatif de $\int_a^b f(x)dx$

- A partir de la question précédente écrire l'algorithme en pseudo-langage permettant de calculer $\int_a^b f(x)dx$ par les différentes méthodes des rectangles (rectangle à gauche, rectangle à droite, point milieu).
- On utilisera pour $f(x)$ la fonction suivante : $f(x) = \frac{1}{x}$. Les bornes inférieure et supérieure de calcul de l'intégrale (a et b) ainsi que le nombre d'itérations (N) seront déclarés en constantes.
- Traduire cet algorithme en Java et le tester avec les valeurs suivantes : a=2, b=4 et N=10 puis 100 puis 1000.

Exercice 5 : Intégration numérique

Décomposition du programme en sous-programme

Décomposer et modifier la méthode main précédente de manière à écrire les méthodes de classe suivantes :

1. Un sous-programme de calcul de $f(x)$ en fonction de x.
2. Un sous-programme de calcul approché de l'intégrale de $f(x)$, par la méthode des rectangles à gauche, sur un intervalle [a,b] avec un nombre d'itération donné.
3. Un sous-programme de calcul approché de l'intégrale de $f(x)$, par la méthode des rectangles à droite, sur un intervalle [a,b] avec un nombre d'itération donné.
4. Un sous-programme de calcul approché de l'intégrale de $f(x)$, par la méthode du point milieu, sur un intervalle [a,b] avec un nombre d'itération donné.

Pour chaque sous-programme préciser l'en-tête du sous-programme (nom de la méthode, les paramètres et type de résultat) et argumenter votre choix.

Exercice 5 : Intégration numérique

Écrire un programme principal avec un menu permettant :

- De calculer la valeur approchée de $\int_a^b \frac{dx}{x}$ par l'une des trois méthodes (rectangles à gauche, à droite, du point milieu) lorsque l'utilisateur tape a, b et N. Le cas échéant, les boucles de contrôle de saisie seront mises en place ;
- De calculer plusieurs approximations de $\int_a^b \frac{dx}{x}$: lorsque l'utilisateur tape a et b, le programme affiche les valeurs approchées, pour chaque sous-programme et pour différents nombres d'itérations (N=10, 100, 1000 et 1000). L'affichage sera de la forme :

N	Rectangle à gauche	Rectangle à droite	Point du milieu
10	$Val_{1,1}$	$Val_{1,2}$	$Val_{1,3}$
100	$Val_{2,1}$	$Val_{2,2}$	$Val_{2,3}$
1000	$Val_{3,1}$	$Val_{3,2}$	$Val_{3,3}$
10000	$Val_{4,1}$	$Val_{4,2}$	$Val_{4,3}$