Langages de programmation
Notes de cours

Maria-Iuliana Dascalu, PhD

mariaiulianadascalu@gmail.com

www.mariaiulianadascalu.com

Université Polytechnique de Bucharest
Département d'Ingénierie en Langues étrangères

Rappel

- Tableaux

Tableaux

- Un tableau est une structure de données qui peut contenir plusieurs éléments du même type.
- Les éléments d'un tableau peuvent être de n'importe quel type: type primitif, type composite ou classe définie par l'utilisateur.
- Exemples:

```
int idEtudiant [];

char[] grades;

float coordonnées[][];

String tableau[] = new String[50];
```

Déclaration, construction, initialisation

- La déclaration d'une variable de type tableau de nom tab dont les éléments sont de type typ:

typ[] tab;

- L'opération de construction s'effectue en utilisant un new:

tab = new typ[taille];

- Taille est une constante entière ou une variable de type entier dont l'évaluation doit pouvoir être effectuée à l'exécution. Une fois qu'un tableau est créé avec une certaine taille, celle-ci ne peut plus être modifiée.
- On peut aussi regrouper la déclaration et la construction en une seule ligne par:

typ[] tab = new typ[taille];

- Exemple:

```
int[] tab = new int[10];
for(int i = 0; i < 10; i++)
        tab [i] = i;  // initialisation
```

Exemples

- L'erreur la plus fréquente est celle-ci :

```
int[] tab;
tab[0] = 1;
```

qui provoque la réponse :java.lang.NullPointerException at Bug1.main(Bug1.java:5)

- L'erreur vient de tenter d'utiliser un tableau qui n'a pas été alloué.
- Initialisation statique:

```
int[] tab = {1,2,4,8,16,32,64,128,256,512,1024};
for(int i = 0; i < tab.length; i++)
    System.out.println(tab[i]);
```

- la taille d'un tableau tab peut s'obtenir grâce à l'instruction tab.length
- un tableau en Java commence nécessairement à l'indice 0
- on peut considérer tab[i] comme une variable et effectuer sur celle-ci toutes les opérations admissibles concernant le type typ, bien entendu l'indice i doit être inférieur à la taille du tableau donnée lors de sa construction
- Si vous rencontrez l'erreur ArrayIndexOutofBoundException….c'est que vous avez demandé une case du tableau qui n'existe pas!
  - Exemple : la case d'indice –1 ou la case d'indice 5 pour un tableau de 5 cases !

- Parmi les opérations simples sur les tableaux, il y a la recherche du plus petit élément qui se réalise par la fonction suivante:

```
static int plusPetit (int[] x)
{
    int k = 0, n = x.length;
    for(int i = 1; i < n; i++)
        // invariant : k est l'indice du plus petit
        if(x[i] < x[k])
            k = i;
    return k;
}
```

- Une autre façon d'utiliser un tableau consiste à effectuer une affectation, par exemple:

```
int[] tabnouv = tab;
```

Noter aussi que l'opération de comparaison de deux tableaux x == y est évaluée à true dans le cas où x et y référencent le même tableau (par exemple si on a effctué l'affectation y = x). Si on souhaite vérifier l'égalité des contenus, il faut une écrire une fonction particulière :

```
static boolean estEgal(int[] x, int[] y)
{
    if(x.length != y.length)
        return false;
    for(int i = 0; i < x.length; i++)
        if(x[i] != y[i])
            return false;
    return true;
}
```

Dans cette fonction, on compare les éléments terme à terme et on s'arrête dès que deux éléments sont distincts, en sortant de la boucle et de la fonction dans le même mouvement.

Tableaux à plusieurs dimensions, matrices

- Un tableau à plusieurs dimensions est considéré en Java comme un tableau de tableaux.
- Par exemple, les matrices qui sont des tableaux à deux dimensions. Leur déclaration peut se faire par:

typ[][] tab;

- On doit aussi le construire à l'aide de *new:*

tab = new typ[N][M]; //un tableau de *N* lignes à *M* colonnes.

- L'instruction *tab.length* retourne le nombre de lignes, alors que tab[i].length retourne la longueur du tableau tab[i], c'est-à-dire le nombre de colonnes.
- On peut aussi, comme pour les tableaux à une dimension, faire une affectation de valeurs en une seule fois:

int[][] tab = {{1,2,3},{4,5,6},{7,8,9}};

Pour le parcourir ….. Des boucles imbriquées feront l'affaire !

```
int[][] matrice=new int[longueur][largeur];
for (int i=0; i<longueur; i++){
        for (int j=0; j<largeur; j++){
                System.out.println(matrice[i][j]);
}}
```

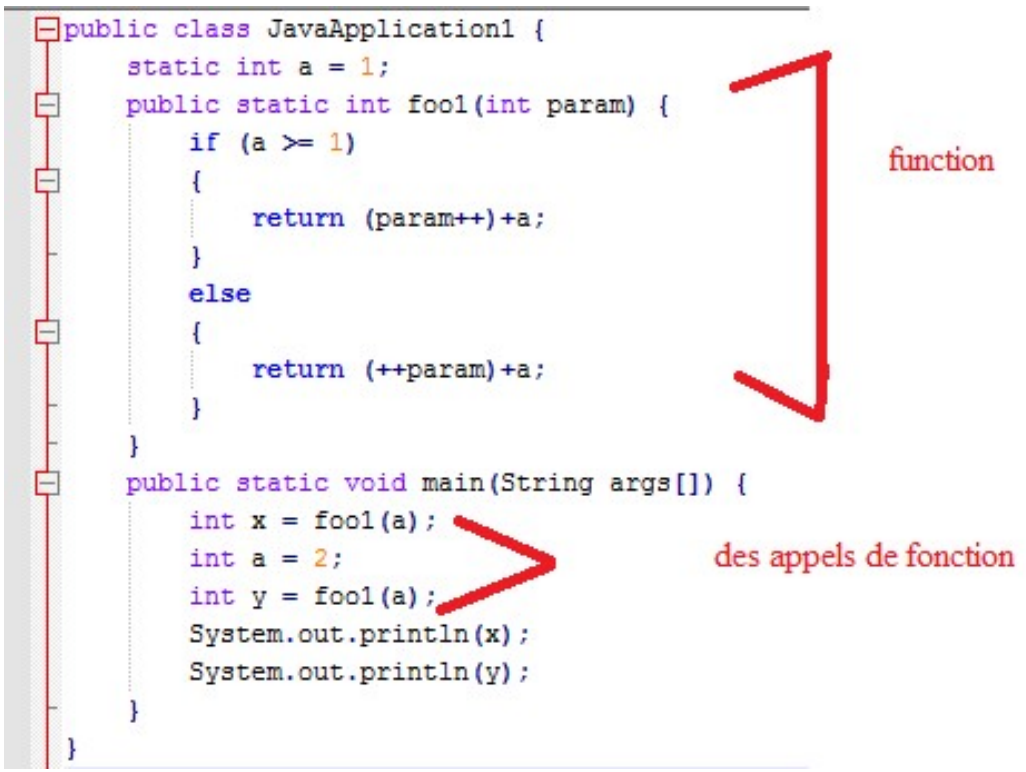
Objectifs pour aujourd'hui

- Fonctions

Quelle est une fonction?

Un bloc de code qui doit être exécutée plusieurs fois.

Exemple:



Observations

- Si nous avons besoin de faire quelque chose pour plusieurs valeurs de *a* dans le même temps, nous allons devoir recopier le programme à chaque fois. Le plus simple est donc d'écrire une fonction à part.

- Remarquons également que nous avons séparé le calcul proprement dit de l'affichage du résultat.

Rôles de functions

- on donne la possibilité de réutilisation des blocs de code à différents endroits du programme

- on clarifie la structure du programme, pour le rendre lisible et compréhensible par d'autres personnes que le programmeur original

Comment écrire des fonctions

- Une function a:
  - un nom
  - une signature (un ensemble de paramètres)
  - elle retourne toujours certains résultat (nous avons le "return" mot)

public static typeRes nomFonction(type1 nom1, type2 nom2, ..., typek nomk)

  - nomFonction est le nom
  - nom1, nom2,…nomk sont des paramètres
  - typeRes est le type du résultat

La signature d'une fonction

- est constituée de la suite ordonnée des types des paramètres
- est unique dans un programme: on peut définir plusieurs fonctions qui ont le même nom à condition que leurs signatures soient différentes (**surcharge = overloading**)
- le compilateur doit être capable à déterminer la fonction à partir du type des paramètres d'appel

Exemple de surchage: l'opérateur + est surchargé : non seulement il permet de faire des additions, mais il permet de concaténer des chaînes de caractères

Le type du résultat d'une function

- doit être indiqué après un *return*
- est obligatoire de prévoir une telle instruction dans toutes les branches d'une fonction
- l'exécution d'un return a pour effet d'interrompre le calcul de la fonction en rendant le résultat à l'appelant

```
public static typeRes nomFonction(type1 nom1, type2 nom2, ..., typek nomk)
{
   typeRes r;
   r = ...;
   return r;  // r est le résultat
}
```

Appel d’une fonction

- On fait appel à une fonction par:

nomFonction(var1, var2, ... , vark)

- En général cet appel se situe dans une affectation.

```
public static void main(String[] args)
{
    type1 n1;
    type2 n2;
    ...
    typek nk;
    typeRes s;
    s = nomFonction(n1, n2, ..., nk); //appel
}
```

Le type spécial *void*

- Le type du résultat peut être void, dans ce cas la fonction ne rend pas de résultat.
- Notons que le return n'est pas obligatoire dans une fonction de type *void*, à moins qu'elle ne permette de sortir de la fonction dans un branchement.
- Nous la mettrons souvent pour marquer l'endroit où on sort de la fonction, et par souci d'homogénéité de l'écriture.
- La procédure principale (dans tous les programmes Java) :

```
class Premier
{
   public static void main(String[] args)
   {
            System.out.println("Bonjour !");
            return; //il peut manquer
   }
 }
```

Visibilité des variables: locale vs globales

- variables d'une fonction sont appelés locales: elles sont visibles uniquement en fonction
- les autres sont globales: elles sont visibles à travers le programme
- les arguments d'une fonction sont passés **par valeurs**, c'est à dire que leur valeurs sont recopiées lors de l'appel: après la fin du travail de la fonction les nouvelles valeurs, qui peuvent avoir été attribuées à ces variables, ne sont plus accessibles

Exemples

```
// Calcul de circonférence
public class Cercle{
    static float pi = (float) Math.PI;

    public static float circonference (float r) {
        return 2. * pi * r;
    }

    public static void main (String[] args){
        float c = circonference (1.5);

        System.out.print("Circonférence:  ");
        System.out.println(c);
        return;
    }
}
```

variable de classe, ce qui veut dire qu'elle est connue par toutes les fonctions présentes dans la classe

La variable r présente dans la définition de circonference est instanciée au moment de l'appel de la fonction par la fonction main. Tout se passe comme si le programme réalisait l'affectation r = 1.5 au moment d'entrer dans f.

Les tableaux comme arguments de fonction

- Considérons le programme suivant:

```
class Test
{
    static void f(int[] t)
    {
            t[0] = -10; return;
    }
    public static void main(String[] args)
    {
        int[] t = {1, 2, 3};
        f(t);
        System.out.println("t[0]="+t[0]);
        return;
    }
}
```

- Que s'affiche-t-il? Pas 1 comme on pourrait le croire, mais -10.
- En effet, nous voyons là un exemple de ***passage par référence***: le tableau t n'est pas recopié à l'entrée de la fonction f, mais on a donné à la fonction f la référence de t, c'est-à-dire le moyen de savoir où t est gardé en mémoire par le programme. On travaille donc sur le tableau t lui-même. Cela permet d'éviter des recopies fastidieuses de tableaux, qui sont souvent très gros.

```
class Essai{
    static int f(int n){
        int m = n+1;

        return 2*m;
    }

    public static void main(String[] args){
        int m = 3;
        System.out.print("résultat=");
        System.out.print(f(4));
        System.out.print(" m=");
        System.out.println(m);
        return;
    }
}
```

La variable m n'est connue (on dit vue) que par la fonction f. En particulier, on ne peut l'utiliser dans la fonction main ou toute autre fonction qui serait dans la classe.

La variable m d'ici est différente de la variable m de la function ci-dessus