Informatique 1
L1 Portail IE
L1 Portail MA

Responsables:

pierre-alain.fouque@univ-rennes1.fr

patrick.derbez@univ-rennes1.fr

Déclaration et portée des variables

Chaque fonction peut déclarer des variables dites « locales ».

- ces variables servent généralement à stocker des résultats intermédiaires de calcul.
- ces variables ont une portée locale à la fonction dans laquelle elles sont déclarées (et ne sont pas accessibles dans les autres fonctions)

```
static double determinant(int a, int b, int c) {
    double d = b*b - 4*a*c;    // déclaration de la variable locale d
    return d;
}

public static void main(String[] args) {
    double valeur = determinant(3, -1, 4);
    if (d < 0)  // erreur! la variable d n'est pas déclarée!
        System.out.println("Aucune racine réelle");
    // Sa portée est locale à la fonction determinant.
}
```

Déclaration et portée des variables

Chaque fonction peut déclarer des variables dites « locales ».

- ces variables servent généralement à stocker des résultats intermédiaires de calcul.
- ces variables ont une portée locale à la fonction dans laquelle elles sont déclarées (et ne sont pas accessibles dans les autres fonctions)

```
// autre exemple
static void determinant(int a, int b, int c) {
    d = b*b - 4*a*c;    // erreur, la variable d n'est pas déclarée
}

public static void main(String[] args) {
    double d;
    determinant(3, -1, 4);
    if (d < 0) // la variable d est déclarée
        System.out.println("Aucune racine réelle");
    // mais elle ne peut pas être modifiée par la fonction déterminant
    // (la variable d est ici locale à la fonction main)
}
```

Déclaration et portée des variables

De même, des variables locales peuvent être déclarées dans n'importe quel bloc d'instruction délimité par les accolades ouvrantes et fermantes.

- **en JAVA, il est interdit d'avoir plusieurs variables locales avec le même nom!**

```
for (int i = 0; i < 10; ++i) { // i locale à la boucle
    {
        int x = i*i; // x locale au bloc
        {
            int x = 3; // Erreur: x déjà déclarée
        }
    }
    int x = 3; // ici on peut déclarer une nouvelle variable x
}
System.out.println(x + i); //Erreur: x et i non définies
```

Variables globales

Les langages impératifs autorisent généralement la déclaration de variables dites « globales ».

- les variables « globales » sont accessibles et modifiables dans l'ensemble des fonctions (on parle de variables à portée globale)

Le langage JAVA autorise la déclaration de variables globales au sein des classes:

- ces variables sont accessibles et modifiables dans l'ensemble des fonctions déclarées dans la classe. Exemple de déclaration de variables globales:

```
public class Calcul {

  static int noteCC;
  static int noteExam;

}
```

Portées locales et globales

Les langages impératifs autorisent généralement la déclaration de variables « globales » et de variables « locales » avec des noms identiques!

- ce sont pourtant des variables différentes!
- la variable de portée la plus locale est considérée d'abord

Exemple avec JAVA :

```
public class LocalGlobal {

    static int x;      // variable globale x
    static int y;      // variable globale y

    static void traitement1( int x ) {
        y = x;         // ici on modifie la variable globale
    }                  //   avec la valeur locale donnée à x

    static void traitement2(int a) {
        int y;          // ici on déclare une variables locale y
        y = 3*a + x + 2;   // et on modifie la variable locale y
    }                   // à partir de la variable globale x et du
                        // paramètre effectif a
}
```

Portées locales et globales

En cas de conflit de nom, la déclaration de portée la plus locale est retenue. Il est cependant possible de spécifier l'accès à une variable de portée globale en préfixant la variable avec le nom de classe:

Exemple avec JAVA :

```
public class LocalGlobal {

    static int x;          // variable globale x
    static int y;          // variable globale y


    static void traitement3( int x ) {
        y = x;             // ici on modifie la variable globale
        LocalGlobal.x = x; // ici aussi
    }

}
```

Tableaux

Tableaux à une dimension

Qu'est-ce qu'un tableau ? Une structure de données qui permet d'utiliser un grand nombre de variables de même type. C'est l'un des objets qui permettent le traitement massif des données.

À quoi servent les tableaux ?

- Stocker des données de même type (dictionnaires, etc.);
- modéliser vecteurs, matrices, polynômes; images, etc.;
- implanter de manière efficace de multiples structures de données (piles, files, arbres, graphes).

Le premier exemple

```
int x = 1;
```

x
1

```
int[] t = new int[5];
for(int i = 0; i < 5; i++)
  t[i] = i*i;
```

t.length	t[0]	t[1]	t[2]	t[3]	t[4]
5	0	1	4	9	16

```
for(int i = 0; i < t.length; i++)
  System.out.println(t[i]);
```

Syntaxe

Déclaration:

```
typ[] t;
```

Par défaut, un tableau est initialisé à la valeur **null**. Il faut donc le construire:

```
t = new typ[taille];
```

new demande de la mémoire au système pour créer un tableau de **taille** éléments de type **typ**.

Les éléments du tableau sont initialisés à la valeur par défaut du type (zéro pour les **int**) à la création.

Syntaxe

Propriétés:

- Les cases du tableau sont numérotées à partir de 0: elles sont désignées par `t[0]`, `t[1]`, ..., `t[taille-1]`.
- On peut récupérer la taille de `t` à l'aide de `t.length`.
- `t[0]`, `t[1]`, ... se manipulent comme des variables ordinaires.
- Java teste les débordements de tableau : interdit `t[-1]`, `t[t.length]`, etc.

Déclarer, construire et initialiser en une seule instruction:

```
int[] t = {6, -4, 7};
```

Java déduit et fixe la taille de `t`; `t[0]` contient 6, `t.length` contient 3, etc.

Erreurs fréquentes

```
int[] t;
t[0] = 2;
```

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
	The local variable t may not have been initialized
```

```
int[] t = {0,1,2};
t[3] = 3;
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
```

Organisation de la mémoire

La mémoire d'un ordinateur peut être vue comme un tableau, dont l'organisation exacte n'a pas besoin d'être connue du programmeur. Tous les acteurs présents dans un programme (variables, tableaux, objets) sont stockés dans ce tableau, et repérés par un indice qu'on appelle référence (ou adresse).

```
int x = 3, y = 4, z;
```

@100	...	@200	...	@300
x	...	y	...	z
3	...	4	...	?

```
z = x + y;
```

@100	...	@200	...	@300
x	...	y	...	z
3	...	4	...	7

Que se passe-t-il pour un tableau?

tableau = référence

```
int[] t;                    // bon de commande
                            // pour la commode
t = new int[3];             // on construit la commode
t[0] = 2;                   // on remplit des tiroirs
t[1] = 1;
t[2] = t[0]+t[1];// on les utilise
```

Mémoire

```
int[] t;
```

@10
t
@0 (null)

```
t = new int[3];
```

@10
t
@500

@500	@504	@508	@512
t.length	t[0]	t[1]	t[2]
3	0	0	0

```
t[0] = 2;
```

@10
t
@500

@500	@504	@508	@512
t.length	t[0]	t[1]	t[2]
3	2	0	0

Affichage d'un tableau

```
public static void main(String[] args) {
    int[] t = {3,2,1,0};
    System.out.println(t);
}
```

Résultat: `[I@7852e922`

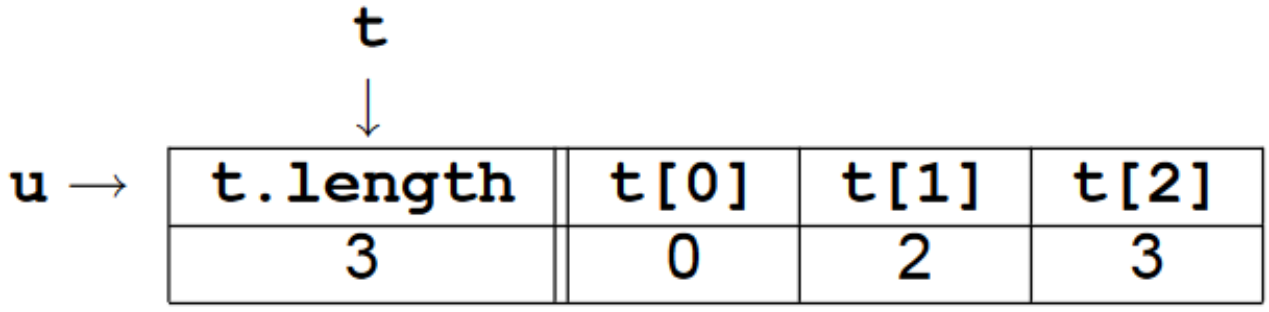
Afficher un tableau

```
public static void afficheTab(int[] tab) {
    for (int i = 0; i < tab.length; ++i) {
        System.out.print(tab[i] + " ");
    }
}

public static void main(String[] args) {
    int[] t = {0,1,2,3};
    afficheTab(t);
}
```

Une conséquence non triviale

```
int[] t = {1, 2, 3}; // t = @500

int[] u = t; // u = @500
u[0] = 0;
System.out.println(t[0]);
```

t et u font référence à la même suite d'emplacements:



Copie d'un tableau

Création d'une copie différente de l'original:

```
int[] t = {1, 2, 3}; // t = @500

int[] u = new int[t.length]; // u = @600
for(int i = 0; i < t.length; i++)
  u[i] = t[i];
```

@500	@504	@508	@512
t.length	t[0]	t[1]	t[2]
3	1	2	3

@600	@604	@608	@612
u.length	u[0]	u[1]	u[2]
3	1	2	3

Tableaux comme argument de fonctions

Rappel: Java passe toujours les arguments des fonctions par valeur (recopie).

```
public static void f(int n){
  n = -10;
}
public static void main(String[] args){
  int n = 1;

  f(n);
  System.out.println(n);
}
```

Le programme affiche 1.

Passage par référence

On passe par valeur la référence du tableau.

```
public static void f(int[] w){ // w = @100
  w[0] = -10;
}
public static void main(String[] args){
  int[] t = new int[5]; // t = @100

  t[0] = 1;
  f(t);
  System.out.println(t[0]);
}
```

Le programme affiche -10, car on a modifié la mémoire globale (du processus).

Passage par copie de la référence

```
public static void swapTabs(int[] t1, int[] t2) {
    int[] t3 = t1;
    t1 = t2;
    t2 = t3;
    // t1 et t2 ont été échangés
}

public static void main(String[] args) {
    int[] tab1 = {0,1,2};
    int[] tab2 = {1,1,1};
    swapTabs(tab1,tab2);
    // tab1 et tab2 n'ont pas été échangés!
    System.out.println(tab1[0]);
}
```