Cours R1.01 : Introduction Algo et Prog Cours 4

M.Adam, N.Delomez, JF.Kamp, L.Naert

IUT de Vannes

8 août 2022

Sommaire

```
/**
* Saisie des notes en informatique et calcul de la moyenne
* @author M.Adam
**/
class Moyenne {
    void principal () {
        double m1101;
        double m1102;
        double m1103;
        double moyenne;

        m1101 = SimpleInput.getDouble
          ("M1101 : Introduction aux systèmes informatiques");
        while (m1101 < 0 || m1101 > 20) {
            m1101 = SimpleInput.getDouble
              ("M1101 : Introduction aux systèmes informatiques");
        }

        m1102 = SimpleInput.getDouble
          ("M1102 : Introduction à l'algorithmique et à la programmation");
        while (m1102 < 0 || m1102 > 20) {
            m1102 = SimpleInput.getDouble
              ("M1102 : Introduction à l'algorithmique et à la programmation");
        }

        m1103 = SimpleInput.getDouble
          ("M1103 : Structures de données et algorithmes fondamentaux");
        while (m1103 < 0 || m1103 > 20) {
            m1103 = SimpleInput.getDouble
              ("M1103 : Structures de données et algorithmes fondamentaux");
        }

        moyenne = (3.5 * m1101 + 3.5 * m1102 + 2.5 * m1103) / 9.5;
        System.out.println ("Moyenne = " + moyenne);
    }
}
```

Remarques

- le même code est écrit plusieurs fois,
- même avec une version simplifiée le code est long à lire,
- le risque d'erreurs est multiplié.

Sommaire

Définition

- Une méthode est une suite d'instructions réalisant une action et pouvant être utilisée plusieurs fois dans un programme.
- Une méthode peut rendre un résultat d'un certain type (`int`, `char`, `long`, `float`, `double`, etc.) ou aucun (`void`).
- Une manière de factoriser le code est d'utiliser les méthodes.
- Une méthode doit être **déclarée** pour pouvoir être **utilisée** ou **appelée**.

Méthodes déjà connues

Méthodes qui ne rendent pas de résultat :

- `System.out.print()`
- `System.out.println()`

Méthodes qui rendent un résultat :

- `SimpleInput.getInt(...)` :

```
int v = SimpleInput.getInt("v = ");
```

- `SimpleInput.getChar(...)`
- `SimpleInput.getString(...)`

Les paramètres

Un paramètre est une valeur ou une variable passée en argument d'une méthode à son appel et déclarée à la définition de la méthode.

- `Math.random()` n'a aucun paramètre :
  ```
  x = Math.random();
  ```
- `Math.sqrt()` et `System.out.print()` ont un paramètre :
  ```
  x = Math.sqrt(4);
  ```
- `Math.min()` a deux paramètres `min = Math.min(x, y);`

Reprise de l'exemple

La bonne manière d'écrire le programme de calcul de la moyenne Info serait :

```
/**
* Saisie des notes en informatique et calcul de la moyenne
* @author M.Adam
**/
class MoyenneMethode {

    void principal () {
        double m1101;
        double m1102;
        double m1103;
        double moyenne;

        m1101 = saisirNote ("M1101 : Introduction aux systèmes informatiques");
        m1102 = saisirNote ("M1102 : Introduction à l'algorithmique et à la programmation");
        m1103 = saisirNote ("M1103 : Structures de données et algorithmes fondamentaux");

        moyenne = (3.5 * m1101 + 3.5 * m1102 + 2.5 * m1103) / 9.5;
        System.out.println ("Moyenne = " + moyenne);
    }
...
```

- le code est plus lisible,
- il est simple d'ajouter une note,
- il est possible de tester de **manière unitaire** `saisirNote()`

Déclaration

La déclaration d'une méthode consiste à définir son nom, ses paramètres et ses instructions.

```
typeRenvoyé nomMéthode (typeParam1 nomParam1, typeParam2 nomParam2, ...) {

    // déclaration des variables locales>;
    typeRenvoyé resultat = ..;
    ...
    //instructions;
    ...
    return resultat;
}
```

Remarques

- Les variables, y compris les paramètres, définies dans une méthode ne sont visibles et utilisables que dans la méthode (portée).
- Il est fortement conseillé de déclarer et d'initialiser une variable du type du résultat dès le début de la méthode et de placer un `return` à la fin de la méthode.
- L'instruction `return` est la dernière exécutée dans une méthode.
- Il est fortement conseillé (obligatoire en R1.01) de n'avoir qu'un seul `return` dans une méthode.

Retour sur l'exemple

```
/**
 * saisie d'une note comprise entre 0 et 20
 * @param titre intitulé du module
 * @return note saisie
 **/
double saisirNote (String titre) {
    double note;

    note = SimpleInput.getDouble (titre);
    while (note < 0 || note > 20) {
        note = SimpleInput.getDouble (titre);
    }

    return note;
}
```

Exemple complet I

```
/**
* Saisie des notes en informatique et calcul de la moyenne
* @author M.Adam
**/
class MoyenneMethode {

    void principal () {
        double m1101;
        double m1102;
        double m1103;
        double moyenne;

        m1101 = saisirNote ("M1101 : Introduction aux systèmes informatiques");
        m1102 = saisirNote ("M1102 : Introduction à l'algorithmique et à la programmation");
        m1103 = saisirNote ("M1103 : Structures de données et algorithmes fondamentaux");

        moyenne = (3.5 * m1101 + 3.5 * m1102 + 2.5 * m1103) / 9.5;
        System.out.println ("Moyenne = " + moyenne);
    }
```

```
    /**
     * saisie d'une note comprise entre 0 et 20
     * @param titre intitulé du module
     * @param note saisie
     **/
    double saisirNote (String titre) {
        double note;

        note = SimpleInput.getDouble (titre);
        while (note < 0 || note > 20) {
            note = SimpleInput.getDouble (titre);
        }

        return note;
    }
}
```

Sommaire

Déclaration

```
void nomMéthode (typeParam1 nomParam1, typeParam2 nomParam2, ...) {

    // déclaration des variables locales>;
    ...
    //instructions;
    ...
}
```

- le type "retourné" est `void`,
- il n'y a pas de `return`.

Exemple

```
/**
* Affiche le contenu d'un tableau
* @author M.Adam
**/
class AfficherTableau {

    void principal () {
        int[] tab = {10, 20, 1, 15, 0, 100};

        afficherTab (tab);
    }

    /**
     * affiche le contenu d'un tableau d'entiers
     * @param t tableau d'entiers
     **/
    void afficherTab (int[] t) {
        int i;

        i = 0;
        System.out.print("{");
        while (i < t.length) {
            if (i != 0) {
                System.out.print(", ");
            }
            System.out.print(t[i]);
            i = i + 1;
        }
        System.out.println("}");
    }
}
```

L'exécution donne le résultat suivant :

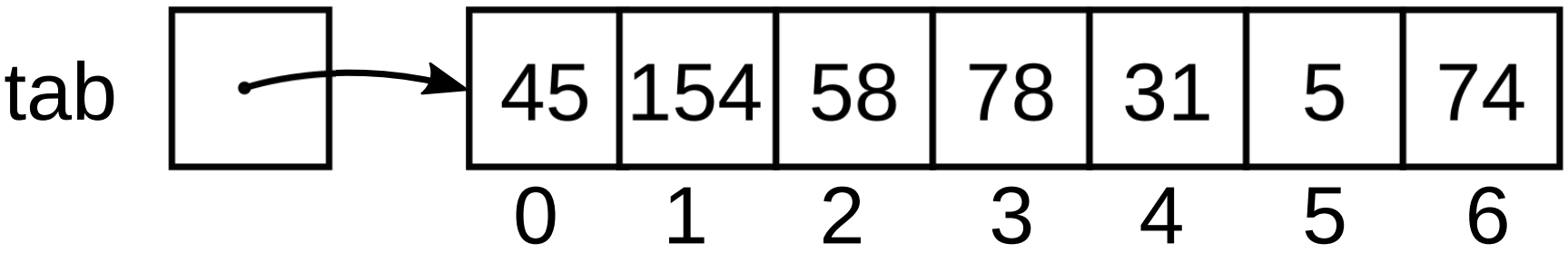
```
$
{10, 20, 1, 15, 0, 100}


------------------
(program exited with code: 0)
$
```

Sommaire

Rappel

Un tableau n'est pas un type primitif. Il est manipulé à travers une référence sur le tableau :

```
int[] tab = {45, 154, 58, 78, 31, 5, 74}
```



Cas d'un tableau en paramètre

Un exemple

```
/**
* Ajouter 1 à chaque élément d'un tableau
* @author M.Adam
**/
class Plus1 {

    void principal () {
        int[] tab = {45, 154, 58, 78, 31, 5, 74};
        afficherTab (tab);
        plus1 (tab);
        afficherTab (tab);
    }

    /**
     * ajoute 1 à chaque élément du tableau
     * @param t tableau d'entiers
     **/
    void plus1 (int[] t) {
        int i;

        i = 0;
        while (i < t.length) {
            t[i] = t[i] + 1;
            i = i + 1;
        }
    }
    ...
```

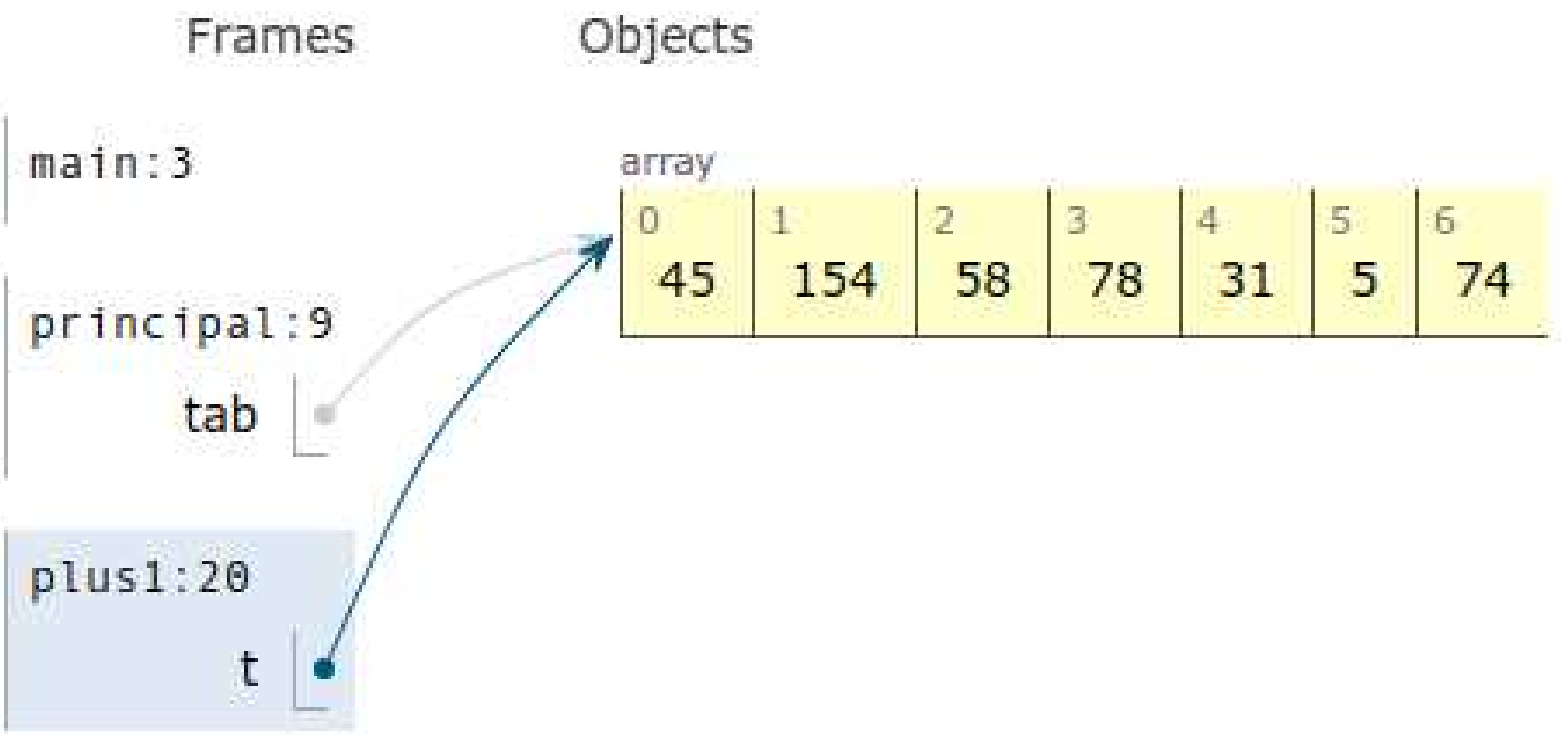
L'exécution donne le résultat :

```
$
{45, 154, 58, 78, 31, 5, 74}
{46, 155, 59, 79, 32, 6, 75}


------------------
(program exited with code: 0)
$
```

Les valeurs du tableau sont modifiées. Comme c'est la référence du tableau qui est passé en paramètre `tab` et `t` désigne le **même** tableau.

Print output (drag lower right corner to resize)

```
{45, 154, 58, 78, 31, 5, 74}
```



Lien JavaTutor : https://tinyurl.com/C04JT01

Cas d'un tableau en retour

```
/**
* Copie un tableau d'entiers
* @author M.Adam
**/
class CopierTab {
    void principal () {
        int[] tab = {45, 154, 58, 78, 31, 5, 74};
        int[] copie;
        afficherTab (tab);
        copie = copier(tab);
        afficherTab (copie);
    }

    /**
     * copie le tableau passé en paramètre
     * @param t tableau d'entiers
     * @return une copie du tableau passé en paramètre
     **/
    int[] copier (int[] t) {
        int[] c = new int[t.length];
        int i;

        i = 0;
        while (i < t.length) {
            c[i] = t[i];
            i = i + 1;
        }
        return c;
    }
```

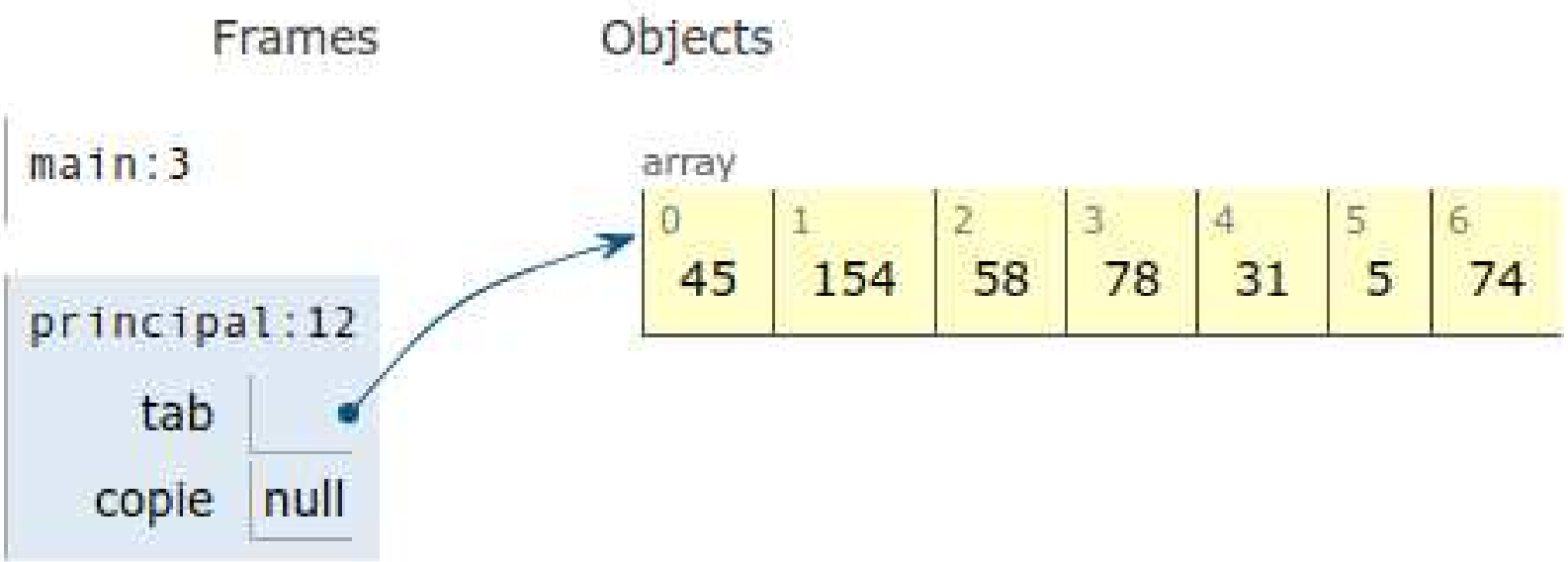
L'exécution donne le résultat :

```
$
{45, 154, 58, 78, 31, 5, 74}
{45, 154, 58, 78, 31, 5, 74}


------------------
(program exited with code: 0)
$
```

Avant la copie

Print output (drag lower right corner to resize)

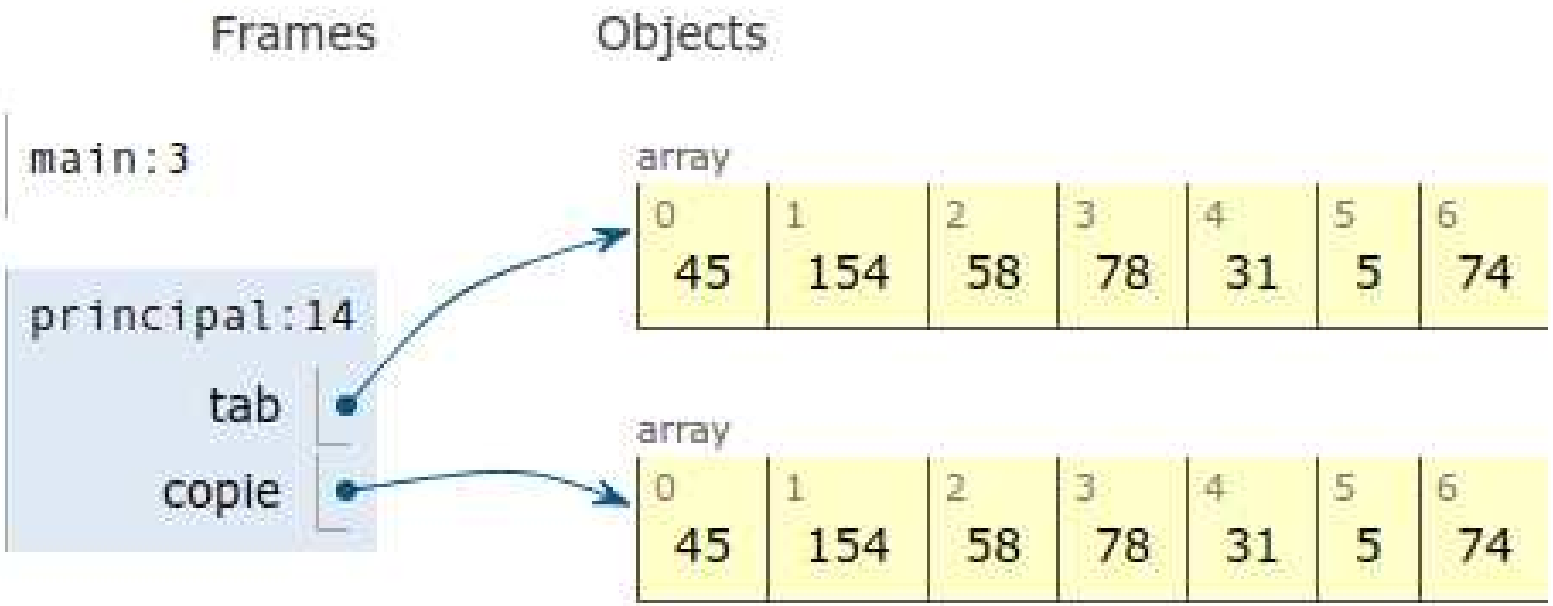
```
{45, 154, 58, 78, 31, 5, 74}
```



Après la copie

Print output (drag lower right corner to resize)

```
{45, 154, 58, 78, 31, 5, 74}
{45, 154, 58, 78, 31, 5, 74}
```



Lien JavaTutor : https://tinyurl.com/C04JT02

Attention

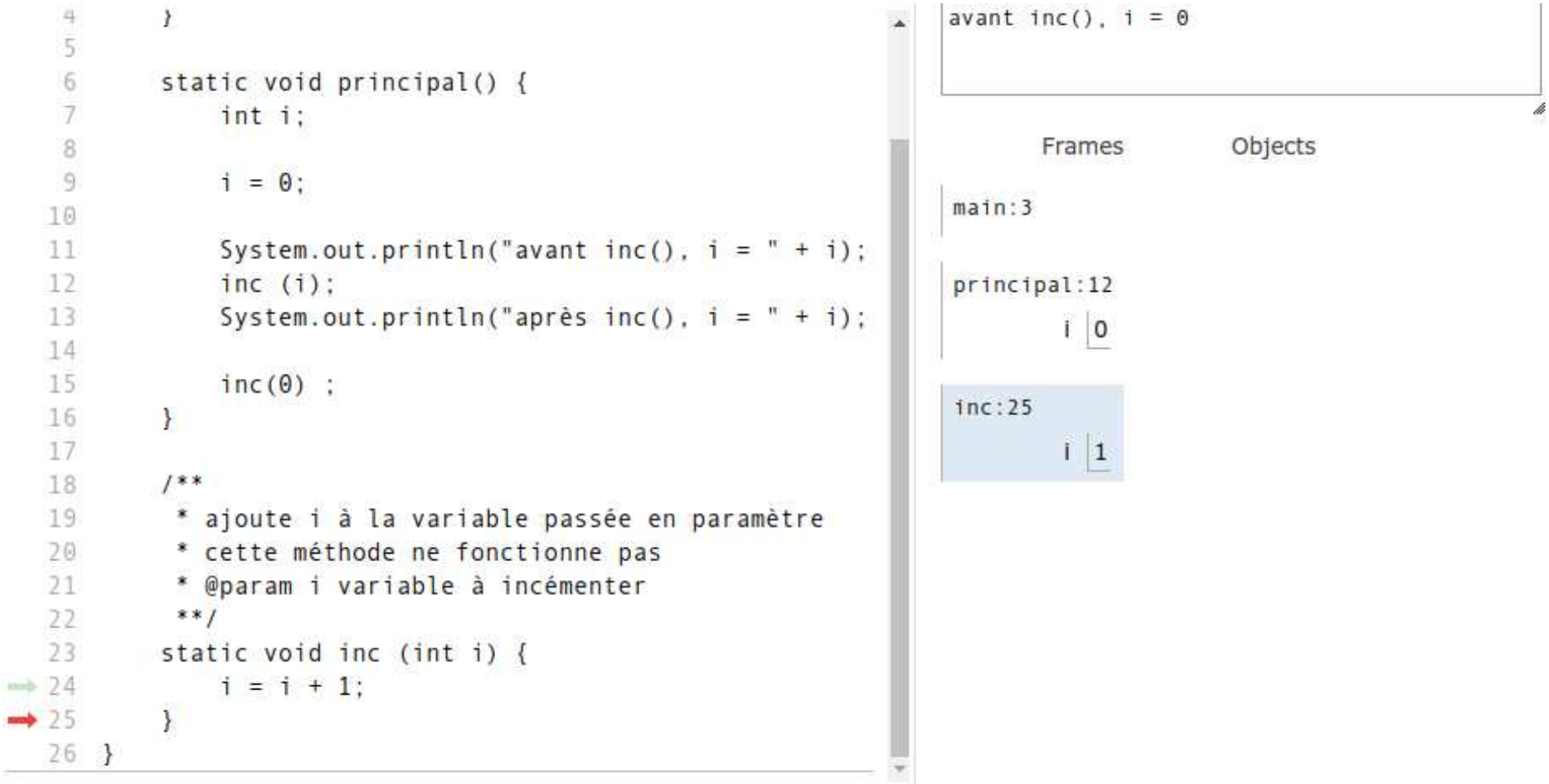
Quand une variable de type primitif est passée en paramètre d'une méthode, la méthode travaille sur une **copie** de la variable.

Exemple

```
class Inc {

    void principal() {
        int i;

        i = 0;

        System.out.println("avant inc(), i = " + i);
        inc (i);
        System.out.println("après inc(), i = " + i);

        inc(0) ;
    }

    /**
     * ajoute 1 à la variable passée en paramètre
     * cette méthode ne fonctionne pas
     * @param i variable à incémenter
     **/
    void inc (int i) {
        i = i + 1;
    }
}
```

Résultat :

```
$
avant inc(), i = 0
après inc(), i = 0

------------------
(program exited with code: 0)
```



Lien JavaTutor : https://tinyurl.com/C04JT03

Version correcte

```
class Add1 {

    void principal() {
        int i;

        i = 0;

        System.out.println("avant inc(), i = " + i);
        i = inc (i);
        System.out.println("après inc(), i = " + i);

        i = inc(0) ;
    }

    /**
     * ajoute 1 à la variable passée en paramètre
     * @param i variable à incémenter
     * @return i + 1
     **/
    int inc (int i) {
        return i + 1;
    }
}
```

Résultat :

```
$
avant inc(), i = 0
après inc(), i = 1

------------------
(program exited with code: 0)
```

Lien JavaTutor : https://tinyurl.com/C04JT04

Sommaire

Signature

La signature d'une méthode, est définie de manière unique par :

- son nom,
- la déclararation de ses paramètres,
- le type du retour.

Par exemple :

```
double exposant (double x, int n)
```

La signature suffit au programmeur pour appeler la méthode, ainsi que pour récupérer les résultats.

Mais pour que le programmeur puisse utiliser une méthode il lui faut également son rôle :

```
/**
* Calcul de x^n
* @param x valeur à monter à l'exposant
* @param n valeur de l'exposant
* @return la valeur de x exposant n
**/
double exposant (double x, int n)
```

Portée

Les paramètres déclarées dans la signature d'une méthode ou dans un bloc, entre { et }, ne sont utilisables que dans la méthode ou le bloc.

```
/**
* Calcul de x^n
* @param x valeur à monter à l'exposant
* @param n valeur de l'exposant
* @return la valeur de x exposant n
**/
double exposant (double x, int n) {
    double res;
    int i;

    res = 1;
    i = 1;
    while (i <= n) {
        res = res * x;
        i = i + 1;
    }
    return res;
}
```

Les variables `x`, `n`, `i`, `res` ne sont utilisables que dans la fonction `exposant`.
Cette notion est appelée la **portée** des variables.

De même, les variables déclarées dans `principal()` ne sont visibles que dans celui-ci.

Un exemple JavaTutor : https://tinyurl.com/C04JT05

Cas général

```
class Portee {
    // déclaration des constantes
    // déclaration des variables globales
    final int LG_TAB = 10 ;

    void principal() {
        // déclaration des variables de principal()
        int[] t;
        int i;


        if (i < LG_TAB) {
            int j;

        }
    }

    int max (int a, int b) {
        //déclaration des variables max()
        int ret = a;

        return ret;
    }
}
```

Sommaire

Tests Fonctionnels ou unitaires

- **fonctionnels** : se mettre à la place de l'utilisateur du programme et tester que toutes les fonctionnalités sont correctes
- **unitaires** : tester une partie du code, des fonctions ou des méthodes pour vérifier qu'elles donnent les bons résultats.

Sommaire

Utilisation des méthodes

- Quand une séquence d'instructions se répète
- Pour décomposer un gros problème en sous problèmes plus simples
- Pour enrichir le langage avec de nouveaux "types" ou "objets" (voir au semestre 2)
- Pour pouvoir tester un programme plus simplement
- Pour rendre le code plus simple à lire et à modifier

A retenir

- Notion de méthodes avec ou sans retour
- Notion de paramètres
- Cas particulier des tableaux
- Notion de portées
- Tests fonctionnels et unitaires