Informatique 1

6. Fonctions

Dr Pierre-André Mudry
pierre-andre.mudry@hevs.ch

Rev 1.38

Motivation initiale

```
int a = 34;
int b = -23;
int c = -2;

int abs_a = a > 0 ? a : -a;
int abs_b = b > 0 ? b : -b;
int abs_c = c > 0 ? c : -c;

if ((abs_a + abs_b + abs_c) < 100) {
    ...
}
```

Pas très élégant…

Introduction

- Le code jusqu'à maintenant: monolithique + haut vers le bas
  - réutiliser = recopier le code partout
  - pas très élégant !
- *Solution* :
  1. Définir une fonction
  2. Utiliser cette fonction

Solution avec fonction

```
public static int abs(int x) {
    if (x > 0)
        return x;
    else
        return -x;
}

public static void main(String args[]) {
    int a = 34;
    int b = -23;
    int c = -2;

    if ((abs(a) + abs(b) + abs(c) < 100) {
    ...
```

Caractéristiques des fonctions

1. Peuvent être appelées
2. Peuvent recevoir des paramètres
3. L'évaluation d'une fonction retourne une valeur

```
…
double x = Math.cos(3.1415926);
…
```

En d'autres termes…

Autres caractéristiques des fonctions

1. Le **langage en fournit** certaines
   - *Java* en fournit beaucoup → http://docs.oracle.com/javase/8/docs/
   - Mathématiques, traiter du texte, images, …
2. On **peut en définir** de nouvelles
   - pour éviter de réécrire du code plusieurs fois par ex.

Quelques fonctions mathématiques existantes

Prototypes

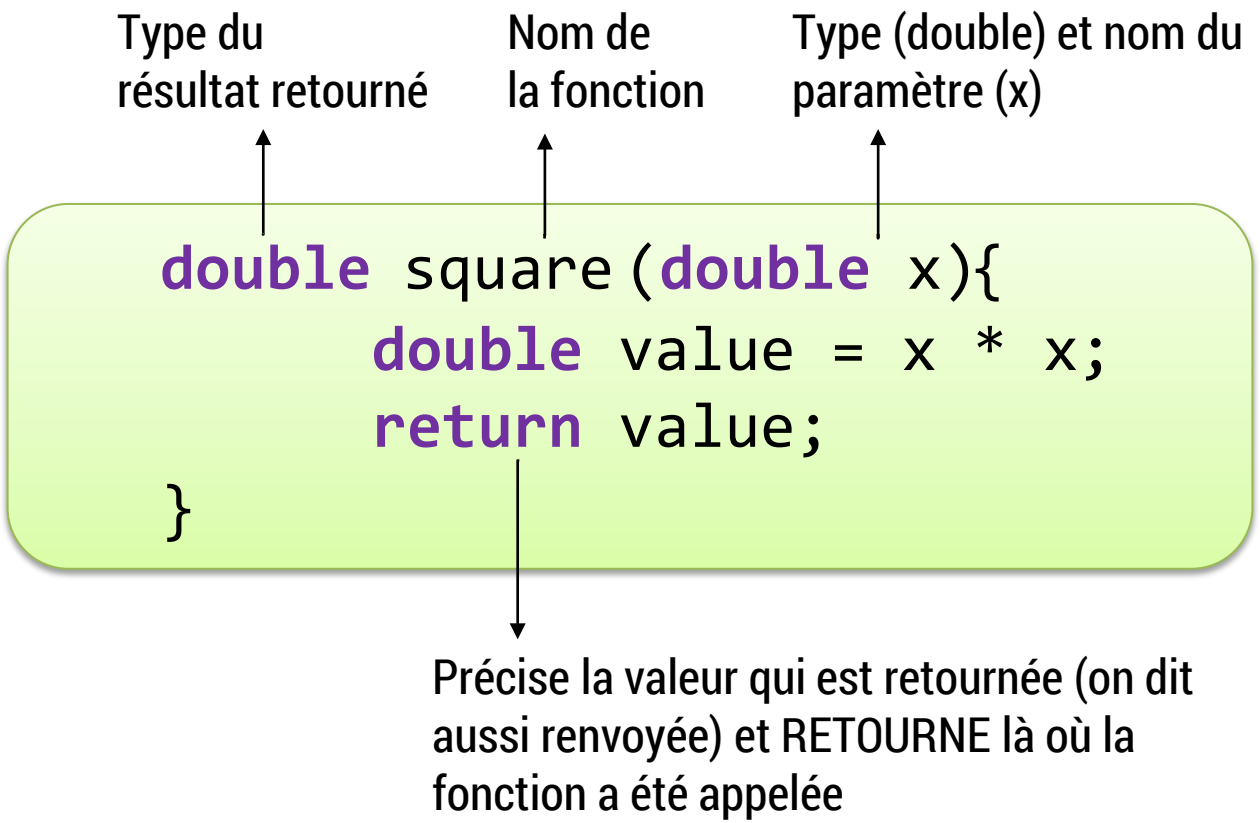
6.1 DÉFINIR DES FONCTIONS

Définir des fonctions

- Pour définir une fonction, on donne son prototype
- Le prototype contient
  1. Le *nom* de la fonction (un identificateur)
  2. Le *nom* et le *type* du ou des arguments. Si aucun argument, mettre des parenthèses vides `( )`
  3. Le *type* de la valeur de retour (max une). Si la fonction ne retourne rien, le type de retour est **`void`**

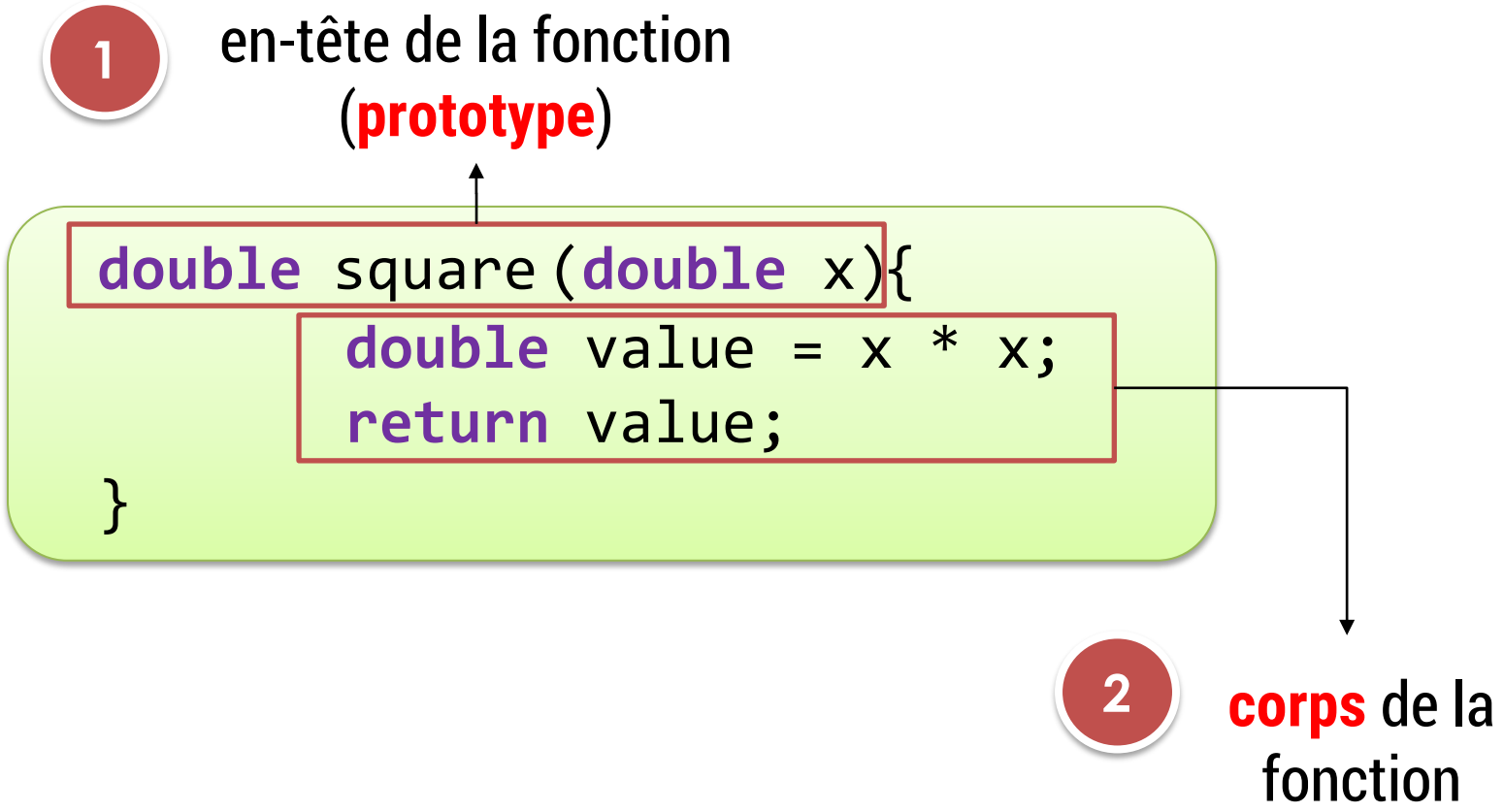
Définition de fonction

- *Exemple* : mise au carré



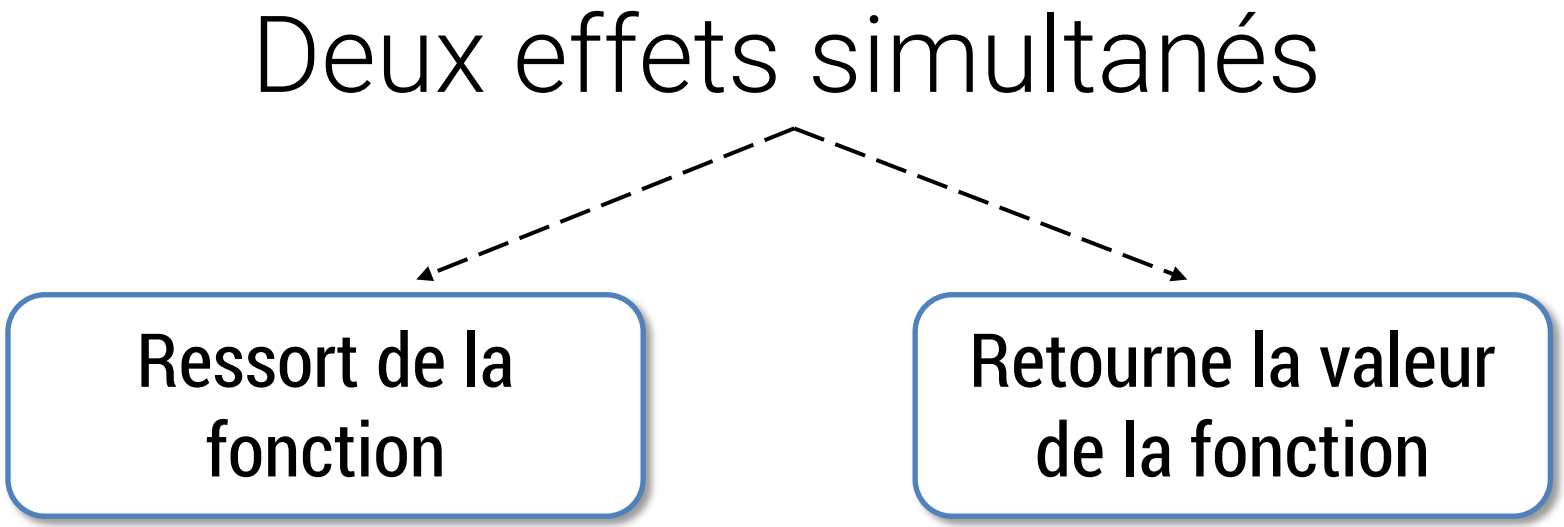
Définition de fonction, vocabulaire

- *Exemple* : mise au carré



Arguments multiples

Instruction `return`



- Note : il est possible d'avoir plusieurs `return` dans une fonction

Le type `void`

- Certaines fonctions ne retournent pas de valeur utiles, utilisation d'un `void`
- Type particulier, usage unique
- Effet de bord

Instruction **return**, exemple

```
boolean isZero(int i) {
    if (i == 0) {
        return true;
    }

    return false;
}

void so(String s) {
    System.out.println(s);
    return; // Here, optional
}
```

Remarques

- Les fonctions **peuvent** être différentes des fonctions mathématiques
  1. Produisent pas toujours de valeurs → on parle dans ce cas de procédure
  2. Parfois modifient la valeur des arguments qu'elles reçoivent...

Déclaration des fonctions

- On peut déclarer une fonction n'importe où dans la classe (avant ou après **`main`** par exemple)
- Appel de fonction = saut aller-retour
  - Dans une fonction, haut vers le bas
- Le corps des fonctions est exécuté uniquement lors de leur appel

Exemple appel de fonction

```
public class Returns {
    static boolean isZero(int i) {
        if (i == 0) {
            return true;
        }
        return false;
    }

    static int abs(int x) {
        if (x > 0)
            return x;
        else
            return -x;
    }

    static void so(String s) {
        System.out.println("[You are the best] " + s);
        return; // Optional
    }

    static void main(String[] args) {
        int foo = abs(-4) - abs(4);
        if (isZero(foo))
            so("Value is zero");
    }
}
```

Implémenter les fonctions suivantes :

6.1. Une fonction retournant la somme de trois nombres entiers.

6.2. Une fonction qui retourne le plus petit de deux nombres réels

Giving arguments

6.2. PARAMÈTRES DES FONCTIONS

Paramètres formels et effectifs

```
int max(int a, int b)
```

- Les paramètres dans l'en-tête d'une fonction se nomment les paramètres formels
- Leur portée est limitée à la fonction

```
int toto = max(3, 5);
```

- Les paramètres fournis lors de l'appel se nomment les paramètres effectifs
- On peut utiliser n'importe quelle expression comme paramètre effectif, c'est la valeur qui est importante

Paramètres formels et effectifs (2)

x = paramètre **formel**

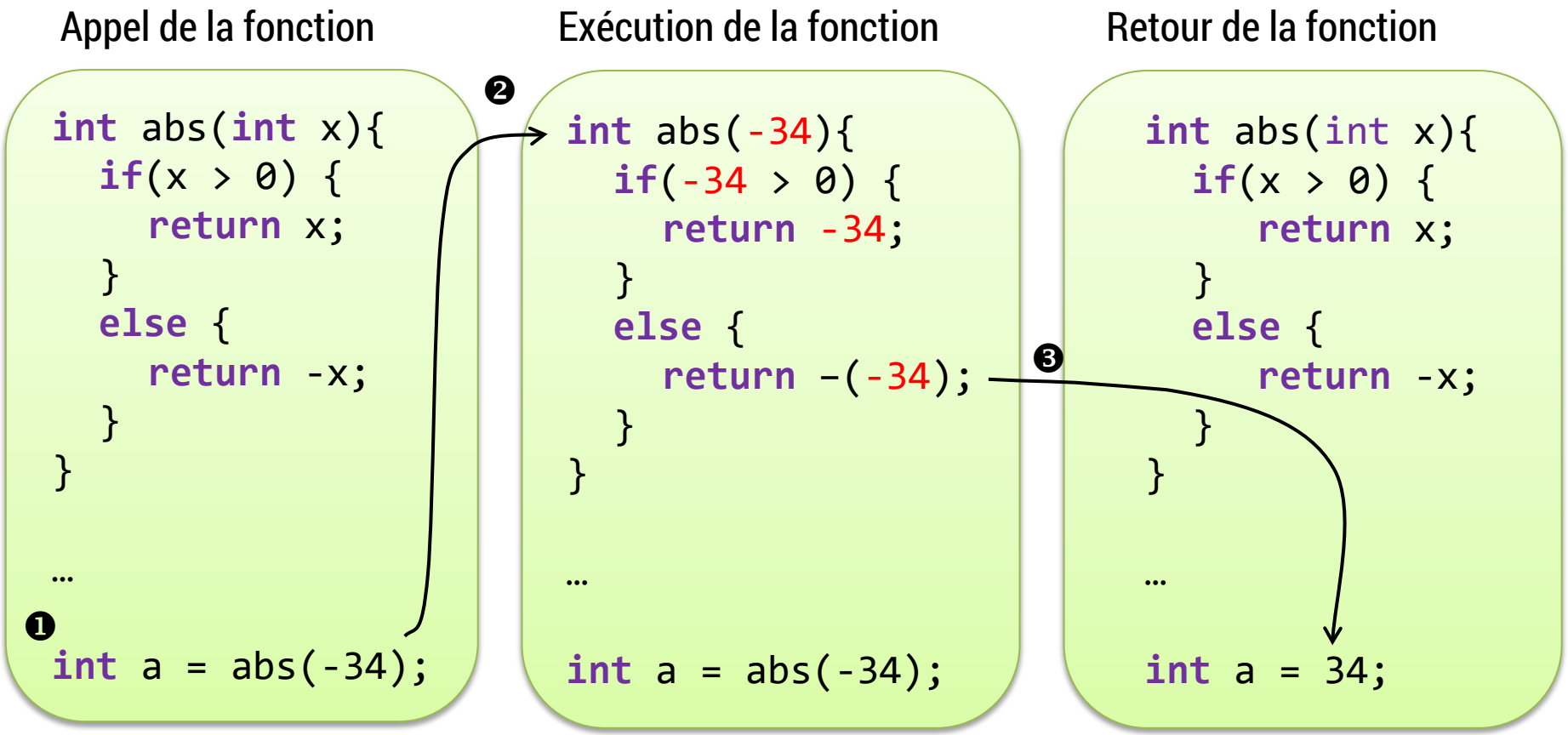
```
int abs(int x){
    if(x > 0) {
        return x;
    }
    else {
        return -x;
    }
}
```

-34 = paramètre **effectif**

```
// Simple use
int a = abs(-34);

// Use on another variable
int x = abs(-23);
int y = abs(x);

// Nested calls
int z =  abs(abs(abs(-x)));

// As an argument
System.out.println(abs(-2));
```

- x est déclaré localement
- x est remplacé à l'appel par la valeur effective !

Exécution pas-à-pas



Surcharge de fonction

- Il est possible de définir, dans la même portée, des fonctions de même nom, mais acceptant des paramètres différents (en *type* ou *nombre*).

  Exemple:

```
int max(int a, int b);
int max(int a, int b, int c);
int max(int a, int b, int c, int d);
```

Surcharge (2)

- Prototype = signature de fonction
- Impossible de discriminer deux fonctions uniquement sur leur valeur de retour !

Les fonctions d'un programme Java

- Les fonctions appartiennent toujours à une classe en *Java*, car il est orienté objet (OO)
- On parle normalement de **méthodes** plutôt que de fonctions en OO, mais ce sont des synonymes
- Pour nous, le point d'entrée d'un programme Java = la méthode **`main`**

Une méthode particulière: **`main`**

Prototype (on ne peut pas le changer):

```
public static void main(String args[])
```

- Plusieurs nouveaux éléments :
  - *public* : change la portée de la fonction
  - *static* : indique comment la fonction peut être appelée
  - *args[ ]* : un tableau d'éléments de type **`String`**

Modificateurs `public` et `static`

- Liés programmation objets
  - Plus informations plus tard
  - Pour l'instant, vous devez déclarer toutes vos méthodes comme

```
public static
```

Erreurs typiques liées aux fonctions

1. Si le type de la fonction n'est pas **`void`**, il doit avoir au moins un **`return`** dans le corps de la fonction
2. Si type de retour **`void`**
   1. pas de **`return`** dans le corps de la fonction
   2. OU le **`return`** doit être sans paramètres
3. Les instructions après un **`return`** ne sont jamais exécutées

Conclusions

- Fonctions
  - Cadre d'utilisation
  - Définition
  - Paramètres et valeur de retour
- Appel de fonctions
  - Paramètres formels et effectifs
- Prototypes multiples