[CI2] Concepts informatiques
3. Fonctions : polymorphisme, transmission des paramètres

Daniela Petrişan
Université de Paris, IRIF

Fonctions

- aussi appelés méthodes, routines, procédures, etc.
- le même principe : une partie de code qui peut être utilisée à volonté, pour répéter des instances du même type de calculs
- utiles pour réaliser des programmes modulaires : le code est décomposé en parties plus petites et plus simples, qui peuvent être réutilisées si nécessaire.

Fonctions

- aussi appelés méthodes, routines, procédures, etc.
- le même principe : une partie de code qui peut être utilisée à volonté, pour répéter des instances du même type de calculs
- utiles pour réaliser des programmes modulaires : le code est décomposé en parties plus petites et plus simples, qui peuvent être réutilisées si nécessaire.

Dans ce cours :

- paramètres
- surcharge (une forme de polymorphisme)
- transmission des paramètres
- la manipulation de la mémoire

Signature et prototype d'une fonction Java

Dans de nombreux langages, y compris en Java, la description d'une fonction doit être suffisamment précise pour que le compilateur puisse vérifier qu'elle est utilisée correctement.

La déclaration d'une fonction est constituée de son prototype qui précise

- son nom
- une liste ordonnée de paramètres typés
- le type de sa valeur de retour

Signature et prototype d'une fonction Java

Dans de nombreux langages, y compris en Java, la description d'une fonction doit être suffisamment précise pour que le compilateur puisse vérifier qu'elle est utilisée correctement.

La déclaration d'une fonction est constituée de son prototype qui précise

- son nom
- une liste ordonnée de paramètres typés
- le type de sa valeur de retour

La signature de la fonction est la partie du prototype qui permet de déterminer de quelle fonction on parle. En Java, la signature est le prototype moins le type de retour.

```
1   public static void f(byte b, int a)
```

Correction d’un appel

Les paramètres spécifiés dans la déclaration de la fonction sont appelés paramètres formels.

Les valeurs utilisées lors de l’appel de la fonction sont appelées paramètres d’appel.

Pour qu’un appel de fonction soit considéré comme correct par le compilateur, il est nécessaire que les paramètres d’appel correspondent en nombre et en type aux paramètres formels.

Correction d’un appel

Les paramètres spécifiés dans la déclaration de la fonction sont appelés paramètres formels.

Les valeurs utilisées lors de l’appel de la fonction sont appelées paramètres d’appel.

Pour qu’un appel de fonction soit considéré comme correct par le compilateur, il est nécessaire que les paramètres d’appel correspondent en nombre et en type aux paramètres formels.

Notez cependant qu’il existe une certaine souplesse qui est permise pour la correspondance des types. Si deux types sont liés par une relation de sous-typage, alors une valeur du sous-type peut être utilisée là où une valeur du type est attendue.

Correction d'un appel

Les paramètres spécifiés dans la déclaration de la fonction sont appelés paramètres formels.

Les valeurs utilisées lors de l'appel de la fonction sont appelées paramètres d'appel.

Pour qu'un appel de fonction soit considéré comme correct par le compilateur, il est nécessaire que les paramètres d'appel correspondent en nombre et en type aux paramètres formels.

Notez cependant qu'il existe une certaine souplesse qui est permise pour la correspondance des types. Si deux types sont liés par une relation de sous-typage, alors une valeur du sous-type peut être utilisée là où une valeur du type est attendue.

En Java, la surcharge est autorisée : le même nom peut être utilisé pour des fonctions différentes, mais elles doivent avoir des signatures différentes pour éviter toute ambiguïté.

Correction d'un appel

```
1   public static void f(byte b) {
2        System.out.print("avec byte  b = " + b);
3    }
4
5    public static void main(String[] args) {
6        int n = 5;
7        f(n);
8      }
```

Correction d'un appel

```java
public static void f(byte b) {
     System.out.print("avec byte  b = " + b);
 }

 public static void main(String[] args) {
     int n = 5;
     f(n);
   }
```

```
Exception in thread "main" java.lang.Error:  Unresolved compilation problem:
    The method f(byte) in the type Correction is not applicable for the arguments (int)
```

Correction d'un appel

```
1   public static void f(byte b) {
2        System.out.print("avec byte  b = " + b);
3    }
4
5    public static void main(String[] args) {
6        int n = 5;
7        f((byte)n);
8      }
```

Correction d'un appel

```
1   public static void f(byte b) {
2        System.out.print("avec byte  b = " + b);
3    }
4
5    public static void main(String[] args) {
6        int n = 5;
7        f((byte)n);
8      }
```

avec int b = 5

Correction d'un appel

```
1   public static void f(int b) {
2        System.out.print(" Avec int  b = " + b + ".");
3    }
4
5    public static void main(String[] args) {
6        byte n = 5;
7        f(n);
8      }
```

Correction d'un appel

```
1   public static void f(int b) {
2        System.out.print(" Avec int  b = " + b + ".");
3    }
4
5    public static void main(String[] args) {
6        byte n = 5;
7        f(n);
8      }
```

```
Avec int b = 5.
```

Correction d'un appel

```java
public static void f(int b) {
    System.out.print(" Avec int  b = " + b + ".");
 }

 public static void main(String[] args) {
    byte n = 5;
    f(n);
  }
```

```
Avec int b = 5.
```

En Java, `byte` est considéré comme un sous-type d'`int`, de sorte que lorsqu'un `int` est attendu, une valeur `byte` peut être utilisée. Mais pas l'inverse.

Les types primitifs

- le type booléen boolean, qui n'est pas un type entier , et qui peut prendre les valeurs false et true
- les types entiers byte (8-bit), short (16-bit), int (32-bit) et long(64-bit) et aussi char (16-bit)
- les types nombres flottants float (32-bit) et double (64-bit)

Surcharge

Le même identificateur peut être utilisé pour désigner des fonctions ayant des signatures différentes.

```java
public static void f(byte b) {
  System.out.print(" Avec byte  b = " + b + ".");
}

public static void f(int b) {
  System.out.print(" Avec int  b = " + b + ".");
}

public static void main(String[] args) {
  int n = 5;
  f(n);
  f((byte)n);
}
```

Quiz : Que se passe-t-il lors de l'exécution de ce code ?

Surcharge

Le même identificateur peut être utilisé pour désigner des fonctions ayant des signatures différentes.

```java
public static void f(byte b) {
  System.out.print(" Avec byte  b = " + b + ".");
}

public static void f(int b) {
  System.out.print(" Avec int  b = " + b + ".");
}

public static void main(String[] args) {
  int n = 5;
  f(n);
  f((byte)n);
}
```

Quiz : Que se passe-t-il lors de l'exécution de ce code ?

```
> Avec int b = 5.  Avec byte b = 5.
```

Surcharge

```java
public static void g(int a, byte b) {
  System.out.println("int a = " + a + ", byte b = " + b);
}

public static void g(byte b, int a) {
  System.out.println("int a = " + a + ", byte b = " + b);
}

public static void main(String[] args) {
  int n = 5;
  byte m = 7;
  g(n,m);
  g(m,n);
}
```

Quiz : Que se passe-t-il lors de l'exécution de ce code ?

Surcharge

```java
public static void g(int a, byte b) {
  System.out.println("int a = " + a + ", byte b = " + b);
}

public static void g(byte b, int a) {
  System.out.println("int a = " + a + ", byte b = " + b);
}

public static void main(String[] args) {
  int n = 5;
  byte m = 7;
  g(n,m);
  g(m,n);
}
```

Quiz : Que se passe-t-il lors de l'exécution de ce code ?

```
> int a = 5, byte b = 7
  int a = 5, byte b = 7
```

Surcharge

```java
public static void g(int a, byte b) {
  System.out.println("int a = " + a + ", byte b = " + b);
}

public static void g(byte b, int a) {
  System.out.println("int a = " + a + ", byte b = " + b);
}

public static void main(String[] args) {
  byte n = 5;
  byte m = 7;
  g(n,m);
}
```

Surcharge

```java
public static void g(int a, byte b) {
  System.out.println("int a = " + a + ", byte b = " + b);
}

public static void g(byte b, int a) {
  System.out.println("int a = " + a + ", byte b = " + b);
}

public static void main(String[] args) {
  byte n = 5;
  byte m = 7;
  g(n,m);
}
```

Quiz : Que se passe-t-il lors de l'exécution de ce code ?

```
Exception in thread "main"
java.lang.Error:
Unresolved compilation problems:
The method g(int, byte)
is ambiguous
for the type Correction
```

Modes de transmission des paramètres

Il existe plusieurs façons de transmettre des paramètres :

- appel par valeur
- appel par référence
- appel par nom

En Java, le premier mode, l'appel par valeur est utilisé exclusivement.

Appel par valeur

Dans le mode de passage d'un paramètre, ce qui est transmis est la valeur du paramètre d'appel ou de l'expression.

Appel par valeur

Dans le mode de passage d'un paramètre, ce qui est transmis est la valeur du paramètre d'appel ou de l'expression.

Le paramètre formel est vu comme une variable locale de la fonction, et est donc créé à l'entrée du point de contrôle dans le corps de la fonction et est détruite à la sortie de cette fonction.

Son initialisation se fait immédiatement après la création et la valeur initiale est la valeur du paramètre d'appel.

Appel par valeur

Dans le mode de passage d'un paramètre, ce qui est transmis est la valeur du paramètre d'appel ou de l'expression.

Le paramètre formel est vu comme une variable locale de la fonction, et est donc créé à l'entrée du point de contrôle dans le corps de la fonction et est détruite à la sortie de cette fonction.

Son initialisation se fait immédiatement après la création et la valeur initiale est la valeur du paramètre d'appel.

Dans un appel de fonction, nous utilisons une variable **différente** de celle qui est utilisée au moment de l'appel de fonction. Nous faisons une **copie** du paramètre d'appel ! Ainsi, toute modification du paramètre formel ne change que cette copie.

Un paramètre d'appel n'est jamais affecté par une fonction qui utilise simplement sa valeur.

Appel par valeur

```java
public class AppelParValeurEx1 {
    static int a = 10;

    public static void foo(int a) {
      a++;
      System.out.println("dans foo, a = " + a);
    }

    public static void main(String[] args) {
      System.out.println("dans main, a = " + a);
      foo(a);
      System.out.println("dans main, apres foo, a = " + a);
    }
}
```

Appel par valeur

```java
public class AppelParValeurEx1 {
    static int a = 10;

    public static void foo(int a) {
      a++;
      System.out.println("dans foo, a = " + a);
    }

    public static void main(String[] args) {
      System.out.println("dans main, a = " + a);
      foo(a);
      System.out.println("dans main, apres foo, a = " + a);
    }
}
```

Quiz : Que se passe-t-il lors de l'exécution de ce code ?

Appel par valeur

Attention: La règle du passage des arguments par valeur s'applique aussi aux objets et aux tableaux !!

Dans ce cas c'est une référence vers l'objet/tableau qui est copiée.

Appel par valeur : tableaux

```java
public class AppelParValeurEx2 {

    public static void foo(int [] t) {
      for(int i = 0; i < t.length; i++) {
        t[i] *= 2;
      }
      System.out.println(t);
    }


    public static void main(String[] args) {
      int [] v = {1,2,3};
      foo(v);
      System.out.println(v);
      for(int i = 0; i< v.length; i++){
        System.out.print(v[i] + " ");
      }}}
```

Appel par valeur : tableaux

```java
public class AppelParValeurEx2 {

    public static void foo(int [] t) {
      for(int i = 0; i < t.length; i++) {
        t[i] *= 2;
      }
      System.out.println(t);
    }


    public static void main(String[] args) {
      int [] v = {1,2,3};
      foo(v);
      System.out.println(v);
      for(int i = 0; i< v.length; i++){
        System.out.print(v[i] + " ");
      }}}
```

Quiz : Que se passe-t-il lors de l'exécution de ce code ?

Appel par valeur : objets

```
1public class AppelParValeurEx3 {
2    public static void foo(Chat c){
3        if (c.nom.equals("Neko")) {
4          System.out.println("C'est Neko!!!");
5        }
6        c.nom = "Simba";
7        System.out.println("dans foo, le nom de c est mis à jour: " + c.nom);
8        c = new Chat("Pilou");
9        System.out.println("dans foo, le nom de c est mis à jour: " + c.nom);
10     }
11   public static void main(String[] args) {
12     Chat n = new Chat("Neko");
13     Chat m = n;
14     System.out.println("dans main, le nom de n est " + n.nom);
15     foo(n);
16     System.out.println("dans main, apres foo, le nom de n est " + n.nom);
17     System.out.println(m==n); }}
```