

# TP n°7

## Introduction JVM

Le but de ce TP est de comprendre le code-octet Java produit à partir d'un programme Java. N'hésitez pas à consulter la description du jeu d'instructions disponible à :

[docs.oracle.com/javase/specs/index.html](https://docs.oracle.com/javase/specs/index.html)

Résumé : [en.wikipedia.org/wiki/Java\\_bytecode\\_instruction\\_listings](https://en.wikipedia.org/wiki/Java_bytecode_instruction_listings)

**Exercice 1.** On considère une série de suite d'instructions JVM d'une méthode dans une classe `Exo1`. La classe est sous la forme (sauf questions 2, 5 et 6)

```
class Exo1 {  
    int f(int x, float y) {...}  
}
```

Que fait chacun des codes assembleurs suivants? Quel code Java le génère? Attention à la différence entre avec `static` et sans `static`.

Pour chaque question, utilisez votre éditeur de texte favori pour produire un fichier `Exo1.java` contenant la classe `Exo1`, puis compilez avec `javac Exo1.java` et affichez le contenu de la classe avec `javap -c -verbose Exo1.class`. On peut ignorer `LineNumberTable` et `StackMapTable`.

1.	<code>int f(int, float);</code> descriptor: (IF)I flags: Code: stack=2, locals=4, args_size=3 0: iload_1 1: iconst_2 2: iadd 3: istore_3 4: iload_3 5: ireturn	7: istore_3 8: iload_3 9: iload_1 10: iadd 11: ireturn	
2.	<code>static int f(int, float);</code> descriptor: (IF)I flags: (0x0008) ACC_STATIC Code: stack=2, locals=3, args_size=2 0: iload_0 1: iconst_2 2: iadd 3: istore_2 4: iload_2 5: ireturn	4.	<code>int f(int, float);</code> descriptor: (IF)I flags: Code: stack=2, locals=4, args_size=3 0: iload_1 1: i2f 2: fload_2 3: fadd 4: fstore_3 5: bipush 42 7: ireturn
3.	<code>int f(int, float);</code> descriptor: (IF)I flags: Code: stack=2, locals=5, args_size=3 0: iconst_3 1: istore 4 3: iload 4 5: iconst_2 6: iadd	5.	<code>static int f5(int[], int);</code> descriptor: ([II)I flags: (0x0008) ACC_STATIC Code: stack=3, locals=3, args_size=2 0: aload_0 1: iconst_0 2: iaload 3: aload_0 4: iload_1 5: iaload 6: imul 7: istore_2 8: iload_2 9: ireturn

```

6.  static int f(int, float);
    descriptor: (IF)I
    flags: (0x0008) ACC_STATIC
    Code:
      stack=2, locals=5, args_size=2
      0: iconst_0
      1: istore_3
      2: iconst_2
      3: istore      4
      5: bipush      33
      7: istore_2
      8: iload_2
      9: iload      4
     11: if_icmpne   16
     14: iload_3
     15: ireturn
     16: iinc        3, 1
     19: iload_3
     20: ireturn

7.  int f(int, float);
    descriptor: (IF)I
    flags:
    Code:
      stack=2, locals=5, args_size=3
      0: iconst_3
      1: istore_3
      2: iconst_0
      3: istore      4
      5: iload      4
      7: iload_1
      8: if_icmpge   20
     11: iinc        3, 11
     14: iinc        4, 1
     17: goto        5
     20: iload_3
     21: ireturn

8.  ...
    Constant pool:
    #1 = Methodref #4.#14 // java/lang/Object."<init>":()V
    #2 = Methodref #3.#15 // Exo1.f:(IF)I
    ...

    int f(int, float);
    descriptor: (IF)I
    flags:
    Code:
      stack=5, locals=3, args_size=3
      0: fload_2
      1: fconst_0
      2: fcmpl
      3: ifne      8
      6: iconst_1
      7: ireturn
      8: iload_1
      9: aload_0
     10: iload_1
     11: fload_2
     12: fconst_1
     13: fsub
     14: invokevirtual #2 // Method f:(IF)I
     17: imul
     18: ireturn

9.  Note : un float en java se termine par la lettre
    f. Donc
    — 5.6 est de type double;
    — 5.6f est de type float.

    ...
    Constant pool:
    #1 = Methodref #5.#14 //java/lang/Object."<init>":()V
    #2 = Float      3.0f
    #3 = Float      5.6f
    #4 = Class      #15 // Exo1
    #5 = Class      #16 // java/lang/Object
    ...

    int f(int, float);
    descriptor: (IF)I
    flags:
    Code:
      stack=2, locals=4, args_size=3
      0: ldc        #2 // float 3.0f
      2: fstore_3
      3: fload_3
      4: ldc        #3 // float 5.6f
      6: fadd
      7: fstore_3
      8: fload_3
      9: f2i
     10: iload_1
     11: iadd
     12: ireturn

```

**Exercice 2.** Pour chaque code octet du Exercice 1 (sauf 5), décrire le fonctionnement de la JVM (notamment l'évolution des variables et de la pile de calcul des blocs d'activation) lorsque la méthode est appelée avec les paramètres 2 et 1.f.

**Exercice 3.** Pour chacune des portions de code Java suivantes, déterminez une traduction en bytecode, en supposant que le code définit le corps de la méthode f de la classe Exo du Exercice 1. Comparez vos propositions avec la sortie de `javap -c -verbose`

<pre>(1) x = 1-2+x; return x;</pre>	<pre>(2) for (int i = 0; i &lt;= 10; i++){     y = y * x; } return (int) y;</pre>	<pre>(3) float [] tab = {y, y+1}; return (int) tab[x];</pre>
-------------------------------------	---	--