NFP121, Cnam/Paris

Java Virtual Machine
.class, chargeur et instances de Class
Jeu d 'instructions
Génération de code et patron Visiteur

Cnam Paris
jean-michel Douin, douin au cnam point fr
version du 10 Décembre 2018

Bibliographie JVM

- **[LY96]T.Lindholm,F.Yellin. The Java Virtual machine Specification. The Java Series Addison Wesley. 1996.**
- **The VM specification.http://java.sun.com:81/docs/books/vmspec/html**
- **Présentation PowerPoint de Axel Kramer http://www.well.com/user/axel**
- **www.gamelan.com, recherche de: "Java Virtual Machine"**
- **La machine Kaffe de Tim Wilkinson, http://www.sarc.city.ac.uk/~tim/kaffe**
- **http://www.techniques-ingenieur.fr**

Interpréteurs et machine à pile

- **N. Wirth.Algorithms+Data Structures=Programs,Chap 5 pp 280-347. Prentice Hall. 1976.(La machine P-code).**
- **N.Wirth. LILITH Modula workstation.Rapport ETH n° xxxxxxxx 1982. (La machine M-code).**

Processeurs Java

- **PicoJava: The Java Virtual Machine in Hardware. M.Tremblay Sun Microelectronics. support de l'exposé effectué à JavaOne (voir également microJava et ultraJava)**
- **Java Based Devices from Mitsubishi,M32R/D. E. Nguyen. exposé JavaOne**
- **voir Digital StrongARM,...**
- **Ajile, zucotto,…**

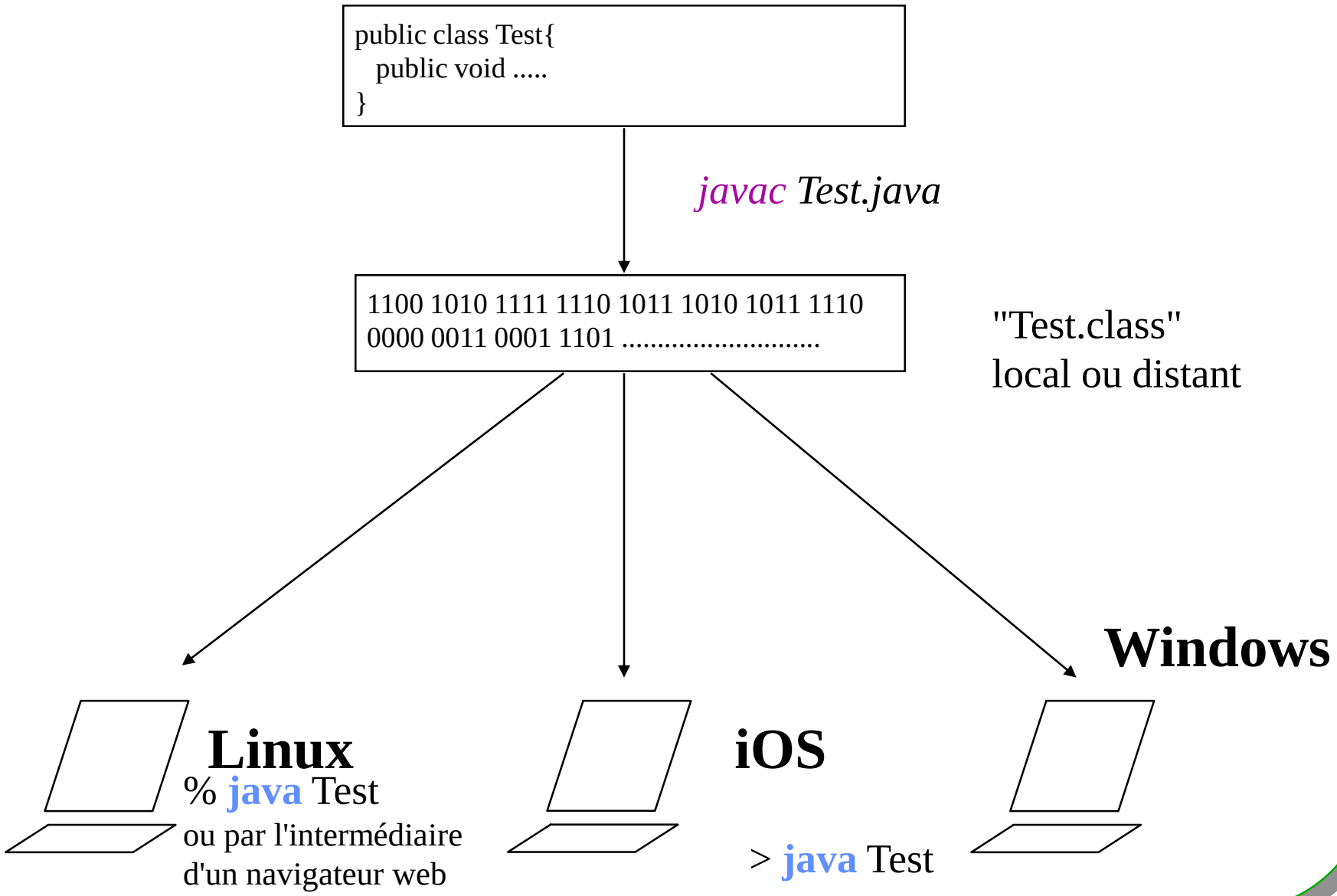
Processeurs basés sur une machine à pile

- **D.A.P.Mitchell,J.A.Thomson,G.A.Manson,G.R.Brookes.Inside the Transputer.BlackWell Scientific Publications. 1990**
- **ST20450, 32 bit microprocessor. Doc SGS-Thomson, May 1995. http://www.st.com/....**

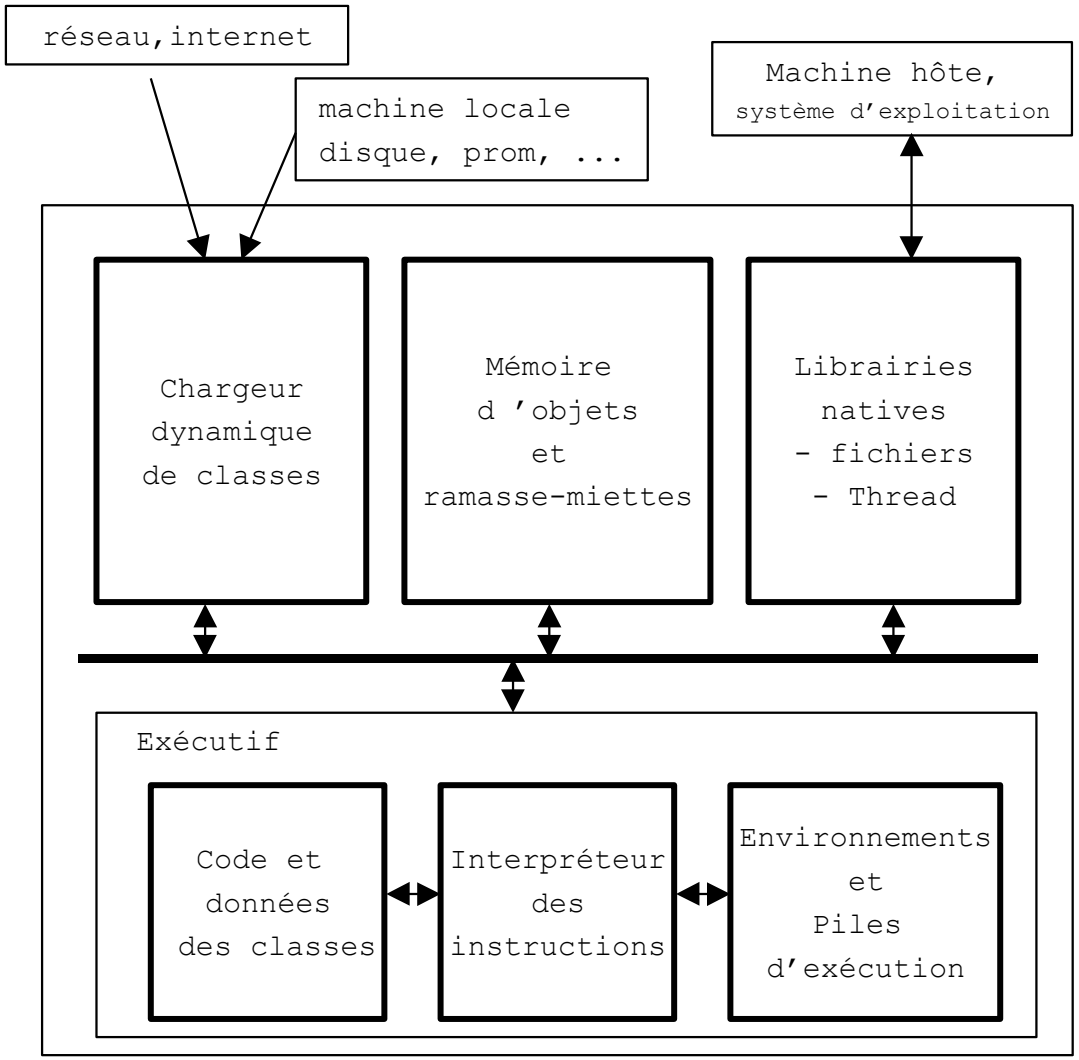
Sommaire

- **Présentation de la machine virtuelle Java (JVM)**
  - **Objectifs et architecture de la JVM**
  - **Le fichier généré ".class"**
  - **Le chargeur de ".class"**
    - **Instances de java.lang.Class**
  - **Le jeu d'instructions**
  - **WhileL, son composite, cf. le cours 8, TP 6**
    - **Un visiteur de génération de code**

Objectifs


public class Test{
public void .....
}
javac Test.java
1100 1010 1111 1110 1011 1010 1011 1110
0000 0011 0001 1101 .....
"Test.class"
local ou distant
Windows
Linux
% java Test
ou par l'intermédiaire
d'un navigateur web
iOS
> java Test

Architecture



- **Java Virtual Machine**
  - **Chargeur de classes et l'exécutif**
  - **Extrait de http://www.techniques-ingenieur.fr**

Chargeurs de classe

- **Chargement dynamique des *.class***
  - **Au fur et à mesure en fonction des besoins**
    - **Chargement paresseux, tardif, *lazy***
- **Le chargeur**
  - **Engendre des instances de *java.lang.Class***
  - **Maintient l'arbre d'héritage**
- **Plusieurs chargeurs de classes peuvent co-exister**

L'exécutif

- **Types de données**
- **Les registres**
- **La pile d'exécution et la mémoire**
  - ***constant pool***
  - ***Interface,field,methods***
- **L'interpréteur de *bytecode***

Sommaire : Classes et java.lang.Class

- **Le fichier *.class***
  - **format**

- **Le chargeur de *.class***
  - **Les chargeurs …**

Le fichier généré ".class"

- **Prémisses**

- **Format du fichier généré**
- **Le *constant pool***
- **Informations sur l'interface(s) utilisée(s) par cette classe**
- **Description des champs des variables de classe ou d'instances**
- **Description des méthodes**
- **Description de ce fichier**

→ ***Table des symboles, sans choix d'implantation .***

Prémisses

- **Compatibilité binaire chapitre 13 *Java Specification***
- **Gestion de projets et internet**
- **Quelles sont les contraintes de compatibilité binaire ?**

- **? Peut-on ajouter de nouvelles méthodes ou variables d'instance d'une classe tout en garantissant l'exécution d'une application antérieure ?**

- ***→ Forme symbolique du .class***

Prémisses

- **ajout de nouveaux champs**
- **ajout de nouvelles classes dans un package**
- **modification du type de déclaration**
- **modification du graphe d'héritage**
- **...**

- ***→ Forme symbolique du .class***

Format du ".class" : description informelle

- ClassFile {

– u4 magic; – u2 minor_version; – u2 major_version;	*Entête du fichier*
– u2 constant_pool_count; – cp_info *constant_pool;	*Symboles et signatures*
– u2 access_flags; – u2 this_class; – u2 super_class;	*"type" de la classe, son nom, le nom de la super-classe*
– u2 interfaces_count; – u2 *interfaces;	*les interfaces*
– u2 fields_count; – field_info *fields;	*Variables de la classe ou d'instances*
– u2 method_count; – method_info *methods;	*Les méthodes de classe ou d'instances*
– u2 attributes_count; – attribute_info *attributes; }	*Description de ce fichier*

Entête

- u4 magic;
- u2 minor_version;
- u2 major_version;

- **0xCAFE 0xBABE**
- **3**
- **45**

Discussions intenses sur le web à l'époque sur valeur du champ magic….

Vérification de la Compatibilité

Le constant_pool

- u2 constant_pool_count;
- cp_info *constant_pool;

```
#define CONSTANT_Class                  7
#define CONSTANT_Fieldref               9
#define CONSTANT_Methodref              10
#define CONSTANT_String                 8
#define CONSTANT_Integer                3
#define CONSTANT_Float                  4
#define CONSTANT_Long                   5
#define CONSTANT_Double                 6
#define CONSTANT_InterfaceMethoderef    11
#define CONSTANT_NameAndType            12
#define CONSTANT_Asciz                  1
#define CONSTANT_Utf8                   1
```

- ***cp_info *constant_pool;***
- **typedef struct {**
- **u1 tag;**
- **u1 *info;**
- **}**cp_info;

```
typedef struct{
        u1 tag;
        u4 bytes;
  }CONSTANT_Integer_info;
```

- **Exemple :**

  si **pool_constant[i] est un entier**

  alors **pool_constant[i].tag == 3);**

  pool_constant[i]->info == valeur de cet entier

u1 *: un octet,* ***u4*** *: 4 octets*

Un exemple « primitif »

```
class bulbe{
  public static void main( String args[]){
    int [] n = new int[6];
    n[0]=0;n[1]=2;n[2]=1;n[3]=3;n[4]=4;n[5]=1;



    boolean sorted = false;
    while(!sorted){
      sorted = true;
      for(int i = 0; i < 5; i++){
        if (n[i] > n[i + 1]){
          int temp = n[i];
          n[i] = n[i + 1];
          n[i + 1] = temp;
          sorted = false;
        }
      }
    }}}
```

Un exemple de constant_pool

```
pool_count : 31
[ 1]    tag:  7  name_index:  9
[ 2]    tag:  7  name_index: 20
[ 3]    tag: 10  class_index:  2  name_and_type_index:  4
[ 4]    tag: 12  class_index: 24  descriptor_index: 28
[ 5]    tag:  1  length:  4  this
[ 6]    tag:  1  length:  1  Z
[ 7]    tag:  1  length: 13  ConstantValue        // type
[ 8]    tag:  1  length:  7  Lbulbe;
[ 9]    tag:  1  length:  5  bulbe
[10]    tag:  1  length: 18  LocalVariableTable
[11]    tag:  1  length:  4  temp
[12]    tag:  1  length: 10  Exceptions
[13]    tag:  1  length: 10  bulbe.java
[14]    tag:  1  length: 15  LineNumberTable
[15]    tag:  1  length:  1  I
[16]    tag:  1  length: 10  SourceFile
[17]    tag:  1  length: 14  LocalVariables
[18]    tag:  1  length:  4  Code
[19]    tag:  1  length:  4  args
[20]    tag:  1  length: 16  java/lang/Object
[21]    tag:  1  length:  4  main
```

Suite du constant_pool

```
• [22]   tag:  1  length: 22  ([Ljava/lang/String;)V
• [23]   tag:  1  length:  4  temp
• [24]   tag:  1  length:  6  <init>
• [25]   tag:  1  length:  6  sorted
• [26]   tag:  1  length:  1  n
• [27]   tag:  1  length:  2  [I
• [28]   tag:  1  length:  3  ()V
• [29]   tag:  1  length:  1  i
• [30]   tag:  1  length: 19  [Ljava/lang/String;
```

- `pool_constant[0] est réservé`

- → *Forme symbolique du .class …*

Pause …

- → *Forme symbolique du .class*

- **Usage d'un décompilateur du « .class » en « .java »**
  - **Par exemple**
    - **http://www.kpdus.com/jad.html**
    - **http://members.fortunecity.com/neshkov/dj.html**
  - **Démonstration**
    - **jad Exemple.class**

- **Obfuscator**
  - **Par exemple**
    - **http://proguard.sourceforge.net/**

Avant - Après

```
class bulbe{
  public static void main(String args[]){
    int [] n = new int[6];
    n[0]=0;n[1]=2;n[2]=1;
    n[3]=3;n[4]=4;n[5]=1;

    boolean sorted = false;
    while(!sorted){
      sorted = true;
      for(int i = 0; i < 5; i++){
        if (n[i] > n[i + 1]){
          int temp = n[i];
          n[i] = n[i + 1];
          n[i + 1] = temp;
          sorted = false;
        }
      }
    }}}
```

```
// Decompiled by Jad v1.5.8g. Copyright
2001   Pavel Kouznetsov.

class bulbe{

 bulbe(){
 }

 public static void main(String args[])
 {
     int ai[] = new int[6];
     ai[0] = 0;ai[1] = 2;ai[2] = 1;
     ai[3] = 3;ai[4] = 4;ai[5] = 1;
     boolean flag = false;
     while(!flag){
         flag = true;
         int i = 0;
         while(i < 5)
         {
             if(ai[i] > ai[i + 1])
             {
                 int j = ai[i];
                 ai[i] = ai[i + 1];
                 ai[i + 1] = j;
                 flag = false;
             }
             i++;
         }}}}
```

access_flag, this_class, super_class

- u2 access_flags;
- u2 this_class;
- u2 super_class;

```
#define ACC_PUBLIC        0x0001
#define ACC_FINAL         0x0010
#define ACC_SUPER         0x0020    /* obsolète */
#define ACC_INTERFACE     0x0200
#define ACC_ABSTRACT      0x0400
```

- **this_class**
  - **Indice dans le constant_pool, (nom de la classe)**
    Indice 1 pour l'exemple (tag 7)
- **super_class**
  - **Indice dans le constant_pool, (nom de la super classe),**
    Indice 2 pour l'exemple (tag 7)
    soit java/lang/Object

field_info

- u2 interfaces_count;
- u2 *interfaces;
- u2 fields_count;
- field_info *fields;

```
typedef struct{
        u2 access_flags;
        u2 name_index;                          /* indices */
        u2 descriptor_index;        /* dans le constant_pool    */
        u2 attributes_count;
        ConstantValue_attribute *attributes;
      }field_info;
```

```
typedef struct{
        u2 attribute_name_index;
        u4 attribute_length;
        u2 constantvalue_index;
      } ConstantValue_attribute;
```

Lecture des descripteurs de "Field"

- ***FieldType ::= BaseType | ObjectType | ArrayType***
- ***BaseType***
  - B **byte**
  - C **char**
  - D **double**
  - F **float**
  - I **int**
  - J **long**
  - S **short**
  - Z **boolean**
- ***ObjectType***
  - L**<classname>**;
- ***ArrayType***
  - [**table**

Exemples :
double m[] [] --> **[[D**

Strings args[] --> **[Ljava/lang/String;**

Field

- **Fields**
  - recense tous les champs d'une classe
  - Statiques
    - fields[i].access_flag & ACC_STATIC == ACC_STATIC
  - ou locaux à chaque instance

  - Note d'implantation :
    - Les types B,C,F,I,L et [occupent un mot machine (32 bits)
    - Les types D et J occupent 2 mots

method_info

- u2 method_count;
- method_info *methods;

- typedef struct{
- u2 access_flags;
- u2 name_index;
- u2 descriptor_index;
- u2 attributes_count;
- Code_attribute *attributes;
- }method_info;

method_info.Code_attribute

```
typedef struct{
        u2 start_pc;
        u2 end_pc;
        u2 handler_pc;
        u2 catch_type;
      } exception_info;
```

```
typedef struct{
        u2 attribute_name_index;
        u4 attribute_length;
        u2 max_stack;
        u2 max_locals;
        u4 code_length;
        u1 *code;
        u2 exception_table_length;
        exception_info *exception_table;
        u2 attributes_count;
        attribute_info *attributes;
      } Code_attribute;
```

Sur l'exemple

- method_count: 2
- method.access_flags: 0x9 /* c'est la méthode main */
- method.name_index: 21
- method.descriptor_index: 22
- method.attributes_count: 1
- attribute_name_index: 18
- attribute_length: 297
- code : 10,6,bc,a,.....3e,b1, /* le byte code 297 octets */

- Soit dans le constant_pool

```
[18]   tag:  1  length:  4  Code
[21]   tag:  1  length:  4  main
[22]   tag:  1  length: 22 ([Ljava/lang/String;)V
```

-

Lecture des descripteurs de "method"

- ***MethodDescriptor ::= (FieldType *) ReturnDescriptor***
- ***ReturnDescriptor ::= FieldType | V***
  - **V si le type retourné est void**

Exemples :
Object m(int i, double d, Thread T)
--> **(IDLjava/lang/Thread;)Ljava/lang/Object;**

void main(String args[]) --> **([Ljava/lang/String;)V**

méthodes d'initialisation

- **<init>V**
  - **Constructeur par défaut de chaque instance**
  - *Sur l'exemple "bulbe.<init>V" est bien présent*

- **<clinit>V**
  - **méthode d'initialisation d'une classe (bloc static)**
  - **exécutée une seule fois au chargement de celle-ci**

method_info.Code_attribute.attributes

- **typedef struct{**
- **u2 attribute_name_index;**
- **u4 attribute_length;**
- **u2 line_number_table_length;**
- **line_number_info *line_number_table;**
- **}**LineNumberTable_attribute;

- *--> informations destinées au débogueur symbolique*

ClassFile.attributes

- typedef struct{
- u2 attribute_name_index;
- u4 attribute_length;
- u2 sourcefile_index;
- } SourceFile_attribute;

- u2 attributes_count;
- attribute_info *attributes;

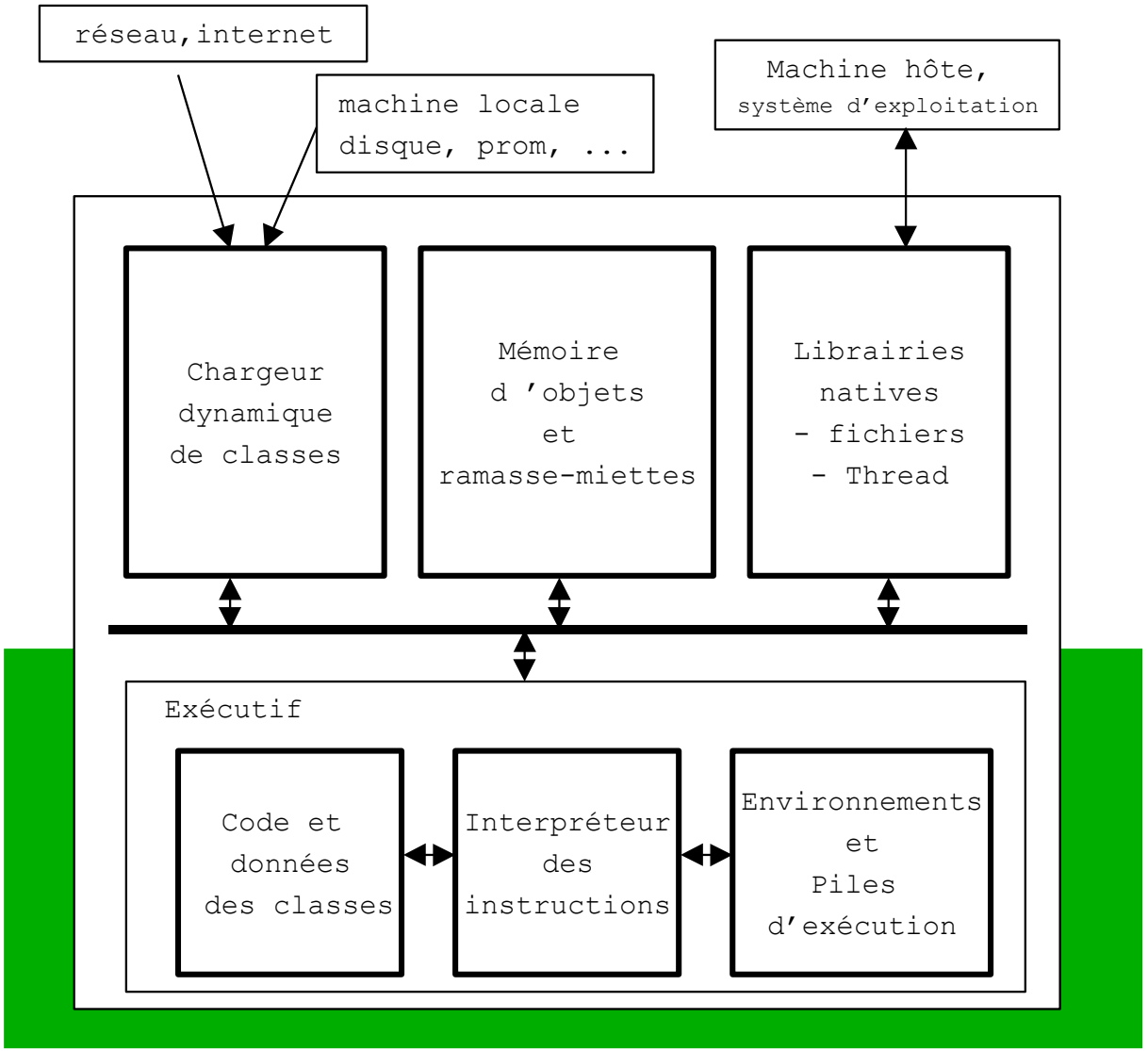
- Sur l'exemple
- analyse de 'attributes'
- attributes_count: 1
- source_attribute.name_index : 16
- source_attribute.length : 2
- source_attribute.sourcefile_index : 13

constant_pool
[13] tag: 1 length: 10 bulbe.java
[16] tag: 1 length: 10 SourceFile

Sommaire suite

- **L'exécutif**
  - **Machine à pile**
  - **Registres**
  - **Un exécutif**
    - **Type de données**
    - **RISC ? ou CISC ?**
  - **Un jeu d'instructions**
    - **Les grandes familles**
- Le chargeur de classes est en annexe

L'exécutif


réseau,internet
machine locale
disque, prom, ...
Machine hôte,
système d'exploitation
Chargeur
dynamique
de classes
Mémoire
d 'objets
et
ramasse-miettes
Librairies
natives
- fichiers
- Thread
Exécutif
Code et
données
des classes
Interpréteur
des
instructions
Environnements
et
Piles
d'exécution

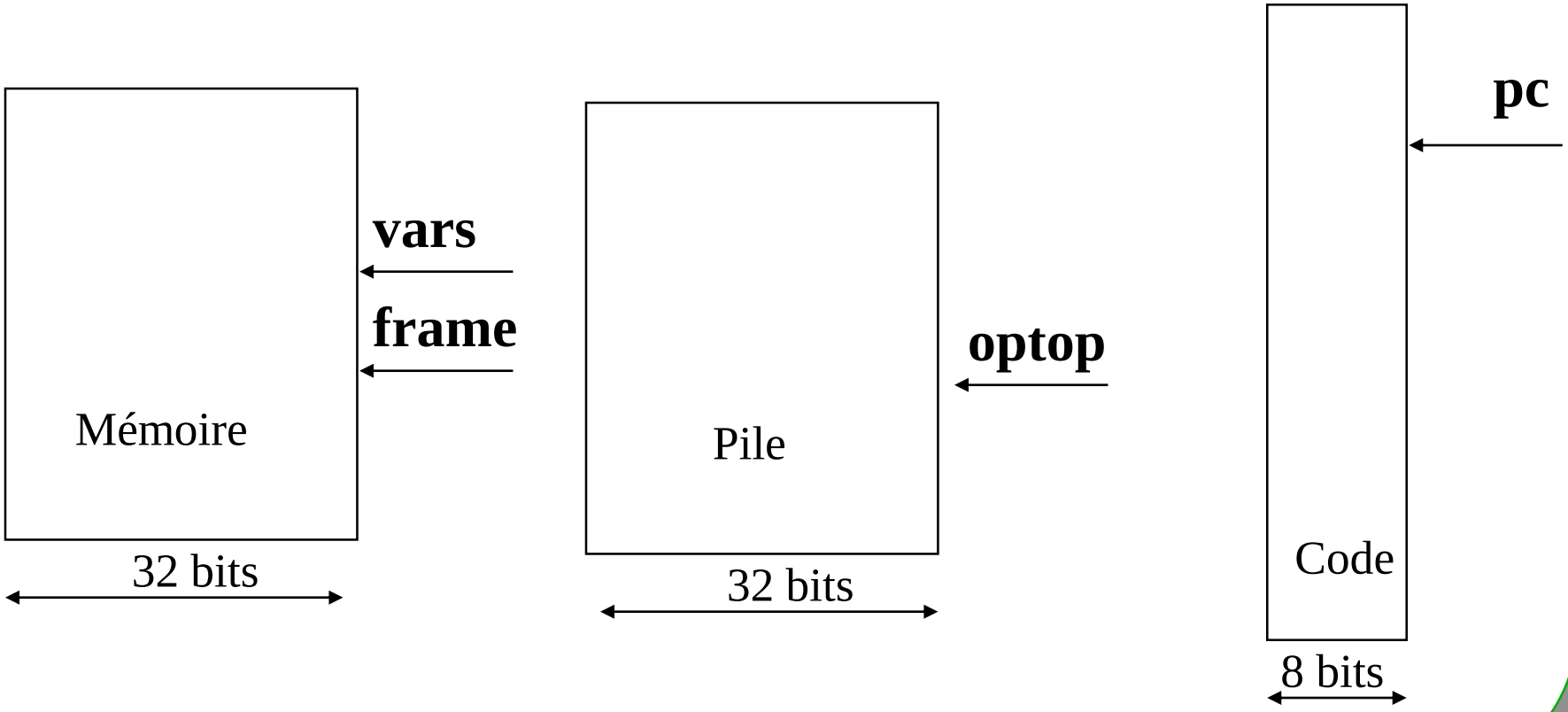
Types de données

- byte : **8 bits en complément à 2**
- short : **16 bits en complément à 2**
- int : **32 bits en complément à 2**
- long : **64 bits en complément à 2**
- char : **16 bits format Unicode**
- float : **32 bits IEEE 754**
- double : **64 bits IEEE 754**
- "adresse" : **32 bits**

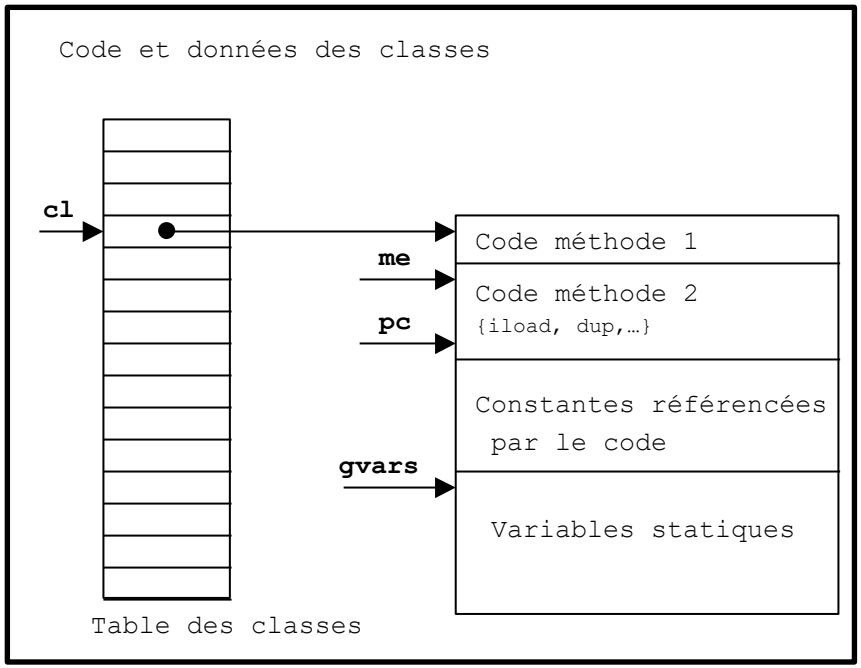
- **« En général » la taille des mots d'une machine virtuelle Java est de 32 bits voire 64 …**

Les registres

- **4 registres**
  - pc, **compteur ordinal**
  - optop, **pointeur de pile**
  - frame, **pointeur de segment d'activation**
  - vars, **pointeur de données locales**



Une vue de la table des classes



- **Autres registres possibles**

La pile d'exécution et la mémoire

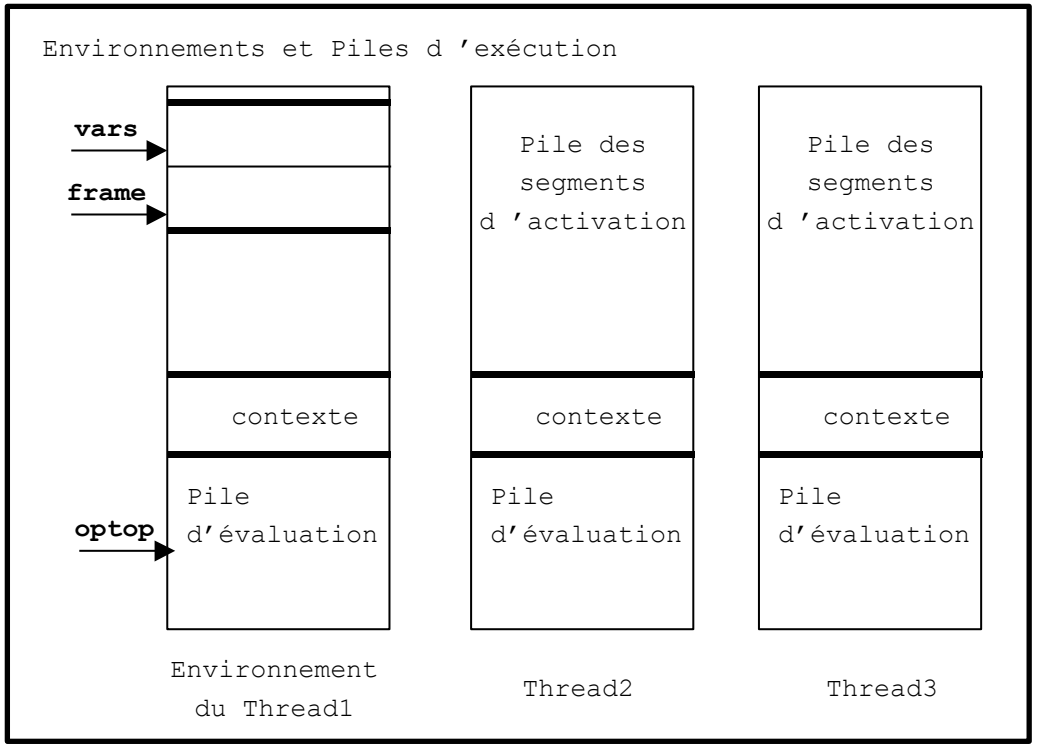
Aspects dynamiques

- **Une pile d'exécution en mots de 32 bits,**
  - **Exemple : iadd == > push(pop() + pop());** *(optop)*
- **Une pile de segments d'activation**
  - **Appels de méthodes** *(frame & vars)*
- **Une zone mémoire est réservée pour les instances**
  - **Pas de déallocation programmée, gérée par un ramasse-miettes**

Aspects "statiques "

- **Une zone mémoire des méthodes de chaque classe** *(pc)*
- **Une zone mémoire des variables de classe**

Thread et « JVM virtuelles »



- **Une pile par Thread… ou processus léger**

Java le langage vers la machine

- **Classes, Variables de classes**
  - **"Constructeurs" de classe**
- **Instances, Variables d'instances**
- **Invocation de méthodes, héritage**
  - **Méthodes de classes**
  - **Méthodes virtuelles et finales**
  - **Constructeurs, (destructeur et ramasse-miettes)**
- **Interfaces**
- **Exceptions**
- **Thread**
- **Appel de code natif**

Le jeu d'instructions

- **Machine à pile**

- **Instructions**
  - **Gestion de la pile constantes et variables**
  - **Gestion des tableaux**
  - **Opérations sur la pile**
  - **Instructions arithmétiques et logiques**
  - **Opérations de conversions**
  - **Instructions de contrôles**

  - **Appels et retours de méthodes**
  - **Instructions sur les instances**
  - **Exceptions et Instructions "_quick"**

Machine à pile

```
class Exemple {
 public static void main( String args[]) {
   int compte=1, resultat=1;
   int valeur = 1;

      compte = compte + 1;
      resultat = resultat * compte;
 }
}
                /* Exemple.main.([Ljava/lang/String;)V */

/*  0*/ iconst_1,                          empiler ( 1)
/*  1*/ istore_1,        compte=1          compte = depiler()
/*  2*/ iconst_1,                          empiler ( 1)
/*  3*/ istore_2,        resultat=1        resultat = depiler()
/*  4*/ iload_1,                           empiler ( compte )
/*  5*/ iconst_1,                          empiler ( 1)
/*  6*/ iadd,                              empiler ( depiler() + depiler())
/*  7*/ istore_1,        compte=compte + 1 compte = depiler()
/*  8*/ iload_2,                           empiler ( resultat )
/*  9*/ iload_1,                           empiler ( compte )
/* 10*/ imul,                              empiler ( depiler() * depiler())
/* 11*/ istore_2,       resultat=resultat*compte  resultat = depiler()
 /*12*/ returnn,
```

Les familles

Le groupe des familles d'instructions non liées aux objets :

- Gestion des variables locales et des constantes : **aload, iload, istore, iconst,…**
- Manipulation de la pile : **dup, pop, …**
- Arithmétiques et logiques : **iadd, imul, …**
- Gestion des conversions entre types primitifs : **i2l,…**
- Contrôle, branchement : **if_icmplt, …goto, jsr, … return**
- Gestion des exceptions : **athrow**
- Accès aux variables de classe : **getstatic, putstatic**
- Appels de méthodes de classe : **invokestatic**

Le groupe des familles d'instructions liées à la gestion des objets :

- Allocation de l'espace mémoire : **new, newarray , …**
- Accès aux champs d'instance : **getfield, putfield, …**
- Accès aux éléments de tableau : **aaload, aastore, …**
- Appels de méthodes d'instance : **invoke_virtual, invoke_super, invokeinterface**
- Gestion de la concurrence : **monitorenter, monitorexit**
- Gestion des conversions entre classes : **checkcast, instanceof**

Format d'une instruction

- **Le code opération des instructions est sur 8 bits**
  - exemple : istore,

0x36					

- **Une instruction possède de 0 à 4 opérandes***
  - exemple : istore

0x36	0x01				

  - [iconst0, istore,0,dup,istore,1]
  -

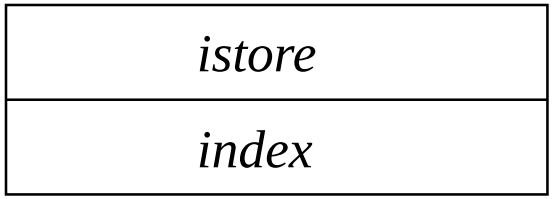
0x03	0x36	0	0x59	0x36	0x01

- **Manipulation implicite de la pile**
  - --> un code plus compact / machine à registre

* il y a des quelques exceptions (3) à cette règle ! (switch, wide)

Format d'une instruction,[LY96] p 275

- **Operation** **Store int into local variable**
- **Format**



- **Forms** ***istore*** **= 54(0x36)**
- **Stack** **…,value => …**
- **Description : .....**

Syntaxe des instructions

- **Txxx**
  - **T indique le type des opérandes**

- **istore**
  - **Rangement d'un entier**
- l **long**
- d **double**
- f **float**
- c **char**
- a **object**

Gestion de la Pile:Constantes et variables

- **Lecture d'une variable locale placée sur la pile**
  - *iload,ilaod_<n>, lload,lload_<n>, fload, fload_<n>, dload,dload_<n>, aload,aload_<n>*
- **Ecriture d'une variable locale depuis la pile**
  - *istore,istore_<n>,lstore,lstore_<n>,fstore, fstore_<n>, dstore,dstore_<n>, astore,astore_<n>*
- **Lecture d'une constante placée sur la pile**
  - *bipush,sipush,ldc,ldc_w,ldc2_w,aconst_null,iconst_m1, iconst_<i>,lconst_<l>,dconst_<d>*
- **Accès avec déplacement supérieur à 16 bits**
  - *wide*

<n> valeur de l'opérande, 0 à 5

Pause : déassembleurs de bytecode

- **jad –a XXXX.class**
- **javap –classpath . –c XXXX**
- **…**

Gestion des tableaux

- **Création de tableaux**
  - *newarray, anewarray,multianewarray*
- **Accès en lecture**
  - *baload,caload,saload,iaload,laload,faload,daload,aaload*
- **Accès en écriture**
  - *bastore,castore,sastore,iastore,lastore,fastore,dastore,aastore*
- **longueur du tableau**
  - *arraylength*

Opérations sur la pile

- **retrait du sommet**
  - ***pop, pop2***
- **duplication du sommet de pile**
  - ***dup,dup,dup_x1,dup2_x1,dup2_x2***
- **échange du sommet de pile et du sous-sommet**
  - ***swap***

Instuctions arithmétiques et logiques

- **Addition/soustraction**
  - *iadd,ladd,fadd,dadd/isub,lsub,fsub,dsub*
- **multiplication,division**
  - *imul,fmul,dmul,lmul,idiv,fdiv,ddiv,ldiv,irem,frem,drem,lrem*
- **complément**
  - *ineg,fneg,dneg,lneg,*
- **décalage**
  - *ishl,ishr,iushr,lshl,lshr,lushr*
- **Opérations**
  - *ior,lor,iand,land,ixor,lxor*
    - *monseignor… il est lor…*
- **Incrément de variable locale**
  - *iinc*

Opérations de conversions

- **int en long, float ou double**
  - ***i2l,i2f,i2d***
- **long en float ou double**
  - ***l2f,l2d***
- **float en double**
  - ***f2d***

Instructions de contrôles

- **Sauts conditionnels**
  - *ifeq,iflt,ilfe,ifgt,ifge,ifnull,ifnonnull,if_icmpeq,if_icmpne, if_icmplt,if_icmpgt,if_icmple,if_icmpge,if_acmpeq,if_acmpne, lcmp,fcmpl,fcmpg,dcmpl,dcmpg*
- **Sauts inconditionnels**
  - *goto,goto_w,jsr,jsr_w,ret*
- **Accès indirect**
  - *tableswitch,lookupswitch*

Exemple : instructions de contrôle

Les instructions	Le source Java	*Les opérations engendrées*
0* iconst_0		*empiler(0);*
1* istore_0	boolean b = false;	*frame[vars+0] = depiler();*
2* iconst_0		*empiler(0);*
3* istore_1	int i=0;	*frame[vars+1] = depiler();*
4* iload_0	b = b & (i > 10);	*empiler(frame[vars+0]);*
5* iload_1		*empiler(frame[vars+1]);*
6* bipush,10		*empiler(10);*
		opr2 = depiler();opr1 = depiler();
8* if_icmpgt,00,07		*si(opr1 > opr2) pc = 15;*
11*iconst_0		*empiler(0);*
12*goto,00,04		*pc = 16;*
15*iconst_1		*empiler(1);*
16*iand		*empiler(depiler() & depiler());*
17*istore_0		*frame[vars+0] = depiler();*
18*iload_0	b = b && (i >10);	*empiler(frame[vars+0]);*
19*ifeq,00,09		*si(depiler()==depiler()) pc=28;*
22*iload_1		*empiler(frame[vars+1]);*
23*bipush,10		*empiler(10);*
		opr2 = depiler();opr1 = depiler();
25*if_icmpgt,00,07,		*si(opr1 > opr2) pc = 32;*
28*iconst_0		*empiler(0);*
29*goto,00,04		*pc = 33;*
32*iconst_1		*empiler(1);*
33*istore_0		*frame[vars+0] = depiler();*

Exemple de tableswitch

Les instructions	Le source Java	*Les opérations engendrées*
`iconst_0`	`int i=0;`	*empiler(0);*
`istore_0`		
`iload_0`	`switch(i){`	*empiler(frame[vars+0]);*
`tableswitch`		
`00,00,00,34`		*default : 34*
`00,00,00,00`	`case 0 : i = 0; break;`	*low : 0*
`00,00,00,01`	`case 1 : i = 1; break;`	*high : 1*
`00,00,00,24`		*<0/24>*
`00,00,00,29`		*<1/29>*
`24* iconst_0`		*empiler(0);*
`istore_0`		*frame[vars+0] = depiler();*
`goto,00,08`		*pc = 34;*
`29* iconst_1`		*empiler(1);*
`istore_0`		*frame[vars+0] = depiler();*
`goto,00,03`	`}`	*pc = 34;*

Exemple lookupswitch

```
Les instructions   Le source Java      Les opérations engendrées

34*   iconst_0     int j=0;           empiler(0);
35*   istore_1                        frame[vars+0] = depiler();
36*   iload_1      switch(j){         empiler(frame[vars+1]);
37*   lookupswitch
00,00           // 2 octets pour un alignement en mots de 32 bits
00,00,00,74        case 1000 : j = 2;break;  default : 74
00,00,00,02        case 0    : j = 0;break;  npairs  : 2
48*   00,00,00,00,00,00,00,32,               <0/<0-69>>
56*   00,00,03,232,00,00,00,27,               <1/<1000-64>>
64*   iconst_2                          empiler(2);
istore_1                                frame[vars+1] = depiler();
goto ,00,08                          pc = 74;
69*   iconst_0                          empiler(0);
istore_1                           frame[vars+1] = depiler();
goto,00,03                            pc = 74;
74*   return           }
```

Appels de méthodes & valeurs retournées

- **Appel d'une méthode (par défaut virtuelle en java)**
  - ***invokevirtual***
- **Appel d'une méthode implantée par une interface**
  - ***invokeinterface***
- **Appel d'une méthode de la classe super**
  - ***invokespecial***
- **Appel d'une méthode de classe**
  - ***invokestatic***
- **Valeurs retournées**
  - ***ireturn,lreturn,dreturn,areturn, return*** **(void)**

Exemple invoke static

```
Les instructions     Le source Java          Les opérations engendrées

                public class MethodesDeClasse{

static void m(){
invokestatic,00,05   m1();                   MethodesDeClasse.m1(); // V
iconst_5             int i = 5;              empiler(5);
istore_0                                     frame[vars+0] = depiler();
iload_0                                      empiler(frame[vars+0]);
invokestatic,00,06   m2(i);                  MethodesDeClasse.m2(); // (I)V
iload_0              int j = f(i);               empiler(frame[vars+0]);
invokestatic,00,04                           MethodesDeClasse.f(); // (I)I
istore_1                                     frame[vars+1] = depiler();
return           }                           restaurer;

                static void m1(){}
return                                       restaurer;

            static void m2(int k){}
return                                       restaurer;

                public static int f(int x){
                  return x +1;
iload_0                                      empiler(frame[vars+0]);
iconst_1                                     empiler(1);
iadd                                         empiler(depiler() + depiler());
ireturn             }                        restaurer;
```

Exemple invoke virtual

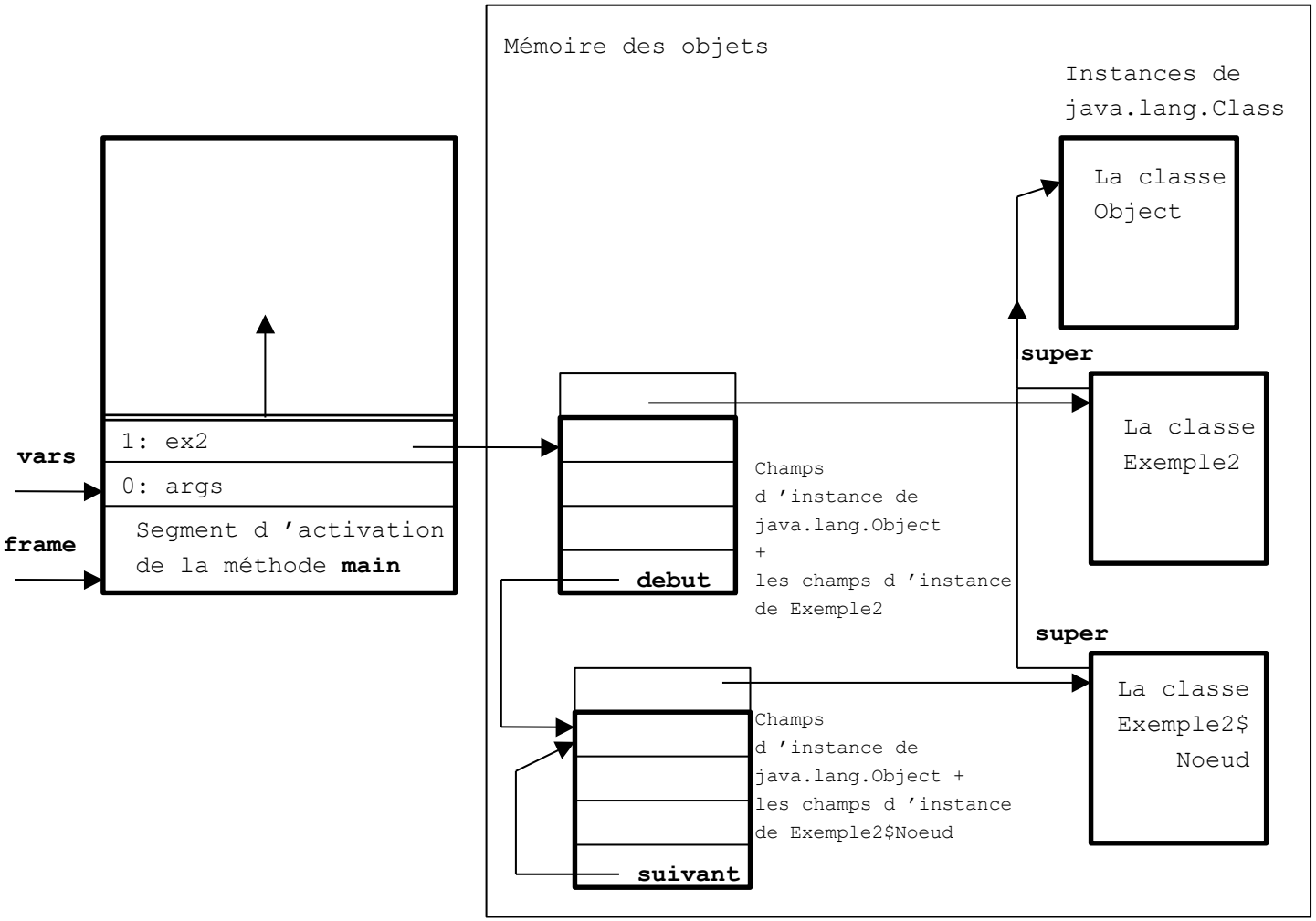
Les instructions engendrées	Le source Java	Les opérations

```
import java.awt.Color;
public class DrawablePoint extends Point{
              private java.awt.Color color;
            public DrawablePoint(Color ic){super();color=ic;}
public static void m(){
          Point p = new DrawablePoint(Color.blue);
new,00,01
dup
getstatic,00,06     // soit java/awt/Color.blue Ljava/awt/Color;
invokespecial,00,05   //soit DrawablePoint.<init>(Ljava/awt/Color;)V
astore_0
aload_0          p.moveTo(10,10);              empiler(p);
bipush,10                                      empiler(10);
bipush,10,                                     empiler(10);
invokevirtual,00,08                            Point.moveTo();//II V
return        }
```

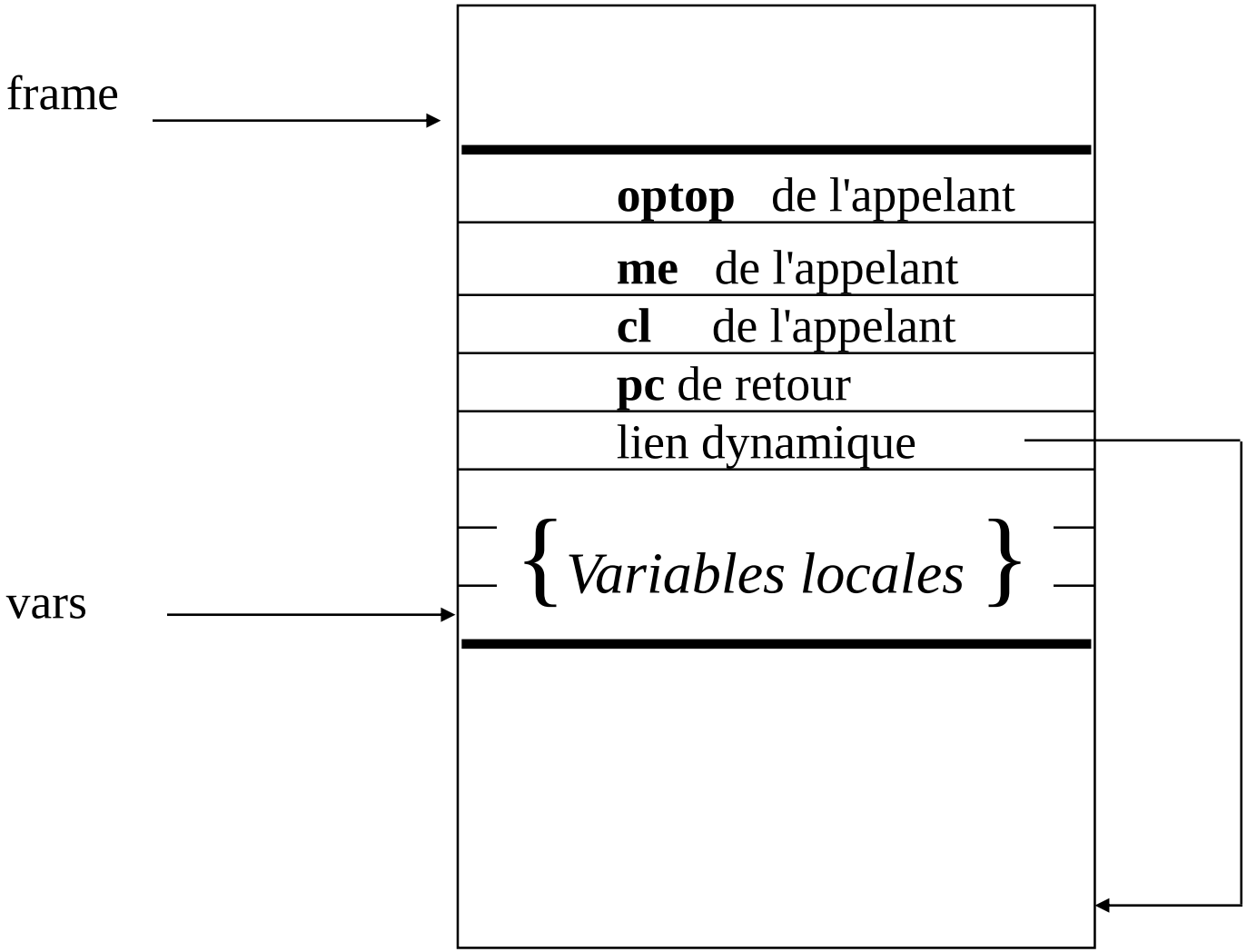
Instructions sur les instances

- **Création d'un objet**
  - *new*
- **Accès aux champs d'instances**
  - *getfield,putfield,getstatic,putstatic*
- **Accès aux champs des variables de classe**
  - *getstatic,putstatic*
- **Contrôles de propriétés (changement de type)**
  - *instanceof, checkcast*

Pile et mémoire des objets



Zoom : Segment d'activation dans sa pile



Exemple putfield, getfield

```
Les instructions      Le source Java        Les opérations engendrées

public class Point extends java.lang.Object{
                private int x,y;

// le constructeur par défaut Point.<init>()V
aload_0                                     empiler(this);
invokespecial,00,04                         java/lang/Object.<init>(); //()V
return

             public void moveTo(int nx, int ny){
aload_0            this.x = nx;             empiler(this);
iload_1                                     empiler(frame[vars+1]);
                                            ref = depiler();
putfield,00,07                              ((Point)ref).x = depiler();
aload_0            y = ny;                  empiler(this);
iload_2                                     empiler(frame[vars+2]);
                                            ref = depiler();
putfield,00,08                              ((Point)ref).y = depiler();
return        }

             public int getX(){
aload_0            return x;                empiler(this);
                                            ref = depiler();
getfield,00,07    }                         empiler(((Point)ref).x);
ireturn
```

Exceptions et Synchronisation

- **Levée d'une exception**
  - ***athrow***
- **Synchronisation de blocs d'instruction**
  - ***monitorexit,monitorenter***

Filtrage

```
Les instructions   Le source Java      Les opérations engendrées

public static void m() throws Exception{
                  try{
0*  new,00,01       Point p = new Point();     empiler(new Point());
3*  dup                                        empiler(sommet());
4*  invokespecial,00,06                        Point.<init>()V
7*  astore_0                                   frame[vars+0] = depiler();
8*  aload_0        int i=p.distOrigin();       empiler(frame[vars+0]);
9*  invokevirtual,00,09  Point.distOrigin();   //I
12* istore_1        }catch(Exceptione){        frame[vars+1]=depiler();
13* goto,00,06                                 pc = 19;
16* astore_0             throw e;              frame[vars+0]=depiler();
17* aload_0                                    empiler(frame[vars+0]);
athrow                                         propager;
return           }

// exception   : start_pc, end_pc, handler_pc, catch_type
//   table[    0]              0  ,       13  ,              16  ,       2
   (java.lang.Exception)
```

Décompilation

- **Le .class et ses instructions**
- **javap –c**
- **Ou autres outils**

Outils

- **Outil prédéfini javap du J2SE**
  - >javap –c *un_fichier*
- http://jasmin.sourceforge.net/

- http://jakarta.apache.org/bcel/
  - Byte Code Engineering Library
- Utilitaire utilisé sur ce support
  - http://jfod.cnam.fr/progAvancee/classes/

jasmin enfin

```
.class public Hello

  .super java/lang/Object
   ;
   ; main() - prints out bonjour!
   ;
   .method public static main([Ljava/lang/String;)V
      .limit stack 2   ; up to two items can be pushed

      ; push System.out onto the stack
      getstatic java/lang/System/out Ljava/io/PrintStream;

      ; push a string onto the stack
      ldc "bonjour!"

      ; call the PrintStream.println() method.
      invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V

      ; done
      return
   .end method
```

un assembleur et un éditeur de texte : retour aux sources

factoriel itératif : enfin lisible

```
.class public factorial
.super java/lang/Object
.method public static main([Ljava/lang/String;)V
   .limit stack 2
   .limit locals 4
      bipush 12
      istore 1
      bipush 1
      istore 2
      bipush 1
      istore 3
   etiq1 :
      iload 2
      iload 1
      isub
      ifge etiq2
      iload 2
      bipush 1
      iadd
      istore 2
      iload 3
      iload 2
      imul
      istore 3
      goto etiq1
   etiq2 :
   getstatic java/lang/System/out Ljava/io/PrintStream;
      iload 3
   invokevirtual java/io/PrintStream/println(I)V
return
 .end method
```

Génération de bytecode, démonstration

- **Un exemple simplifié : WhileL**
  - **tp6, tp9 et tp11 en NFP121/Cnam/Paris**
    - WhileL dont la grammaire est décrite par un un Composite
    - Plusieurs visiteurs, associés à ce Composite
      - Un visiteur d'évaluation
      - Un visiteur de génération de code source en Java
  - Ajout d'un nouveau visiteur : générateur de code
    - **Générateur de code assembleur jasmin**

Objectifs

```
public void test_CompilationDeFactoriel(){
  Contexte m = new Memoire();
  Variable x = new Variable(m,"x",5);
  Variable fact = new Variable(m,"fact",1);

Instruction inst =
  new TantQue(
      new Sup(x,new Constante(1)),
      new Sequence(
        new Affectation(fact,new Multiplication(fact,x)),
        new Affectation(x,new Soustraction(x,new Constante(1))))
      );

  Code code = new Code("TestsFactoriel", m);
   VisiteurExprJasmin vej = new VisiteurExprJasmin(m,code);
   VisiteurBoolJasmin vbj = new VisiteurBoolJasmin(vej);
   VisiteurInstJasmin vij = new VisiteurInstJasmin(vej,vbj);

 // de l'assembleur Jasmin
Jasmin.assemble(code);
 // puis exécution en utilisant l'introspection
Jasmin.execute(code);
}
```

Prémisses

- **Retour sur la compilation**
  - **D'une expression arithmétiques**
  - **D'une expression booléenne**
  - **D'une instruction**

Pour une expression arithmétique

- Générer du *bytecode* pour une expression arithmétique :

- **GenererCode(N) = iconst_N *si N [0..5]***
  GenererCode(N) = iconst_m1 *si N == -1*
  GenererCode(N) = bipush N *si N* [-128..-2] et [6..+127]
  GenererCode(N) = sipush N *si N [-32768..-129] et [+128..+32767]*
  GenererCode(N) = ldc N *en dehors des valeurs ci-dessus*

- **GenererCode(V) = iload_N *avec N = IDX si IDX [0..3]***
  GenererCode(V) = iload N *avec N = IDX si IDX [4..65535]*

- **GenererCode(E1 + E2) = GenererCode(E1),GenererCode(E2),iadd**
  GenererCode(E1 - E2) = GenererCode(E1),GenererCode(E2),isub
  GenererCode(E1 * E2) = GenererCode(E1),GenererCode(E2),imul
  GenererCode(E1 / E2) = GenererCode(E1),GenererCode(E2),idiv

Exemples 3+2…

expression arithmétique	L'instance du composite paquetage whileL	en pseudo-code pour machine à pile	en assembleur Jasmin, syntaxe ici
3 + 2	new Addition(new Constante(3), new Constante(22))	*empiler(3)* *empiler(22)* *additionner*	iconst_3 bipush 22 iadd
(3 - x) / y	Memoire m=new Memoire(); Variable x = new Variable(m,"x",2); Variable x = new Variable(m,"y",3); new Division(new Soustraction(3,x), y);	*empiler(3)* *empiler(x)* *soustraire* *empiler(y)* *diviser*	iconst_3 iload_1 ; x en mémoire à l'adresse vars +1 isub iload_2 ; x en mémoire à l'adresse vars +2 idiv

Pour une expression booléenne

```
GenererCode(Vrai) = iconst_1
GenererCode(Faux) = iconst_0
GenererCode(Non(Bexp)) = GenererCode(Bexp);
                         ifne Etiq1
                         iconst_1
                         goto Fin
                  Etiq1: iconst_0
                    Fin:

GenererCode(BE1 et BE2)) = GenererCode(BE1);
                           ifeq Etiq1
                           GenererCode(BE2);
                           ifeq Etiq1
                           iconst_1
                           goto Fin
                    Etiq1: iconst_0
                      Fin:

GenererCode(E1 > E2 ) = GenererCode(E1);
                        GenererCode(E2);
                        if_cmple Etiq1
                        iconst_1
                        goto Fin
                  Etiq1: iconst_0
                    Fin:
```

Un exemple

expression booléennes	L'instance du composite	en pseudo-code pour machine à pile	en assembleur Jasmin
i > 10	Memoire m= ... new Sup(i, new Constante(10))	*empiler(i)* *empiler(10)* *si(dépiler()<=dépiler()) saut_en échec* *empiler(vrai);* *saut_en fin* *échec :* *empiler(faux);* *fin :*	iload_1 bipush 10 if_icmple #_21 iconst_1 goto #_23 #_21: iconst_0 #_23:

Pour les instructions

```
GenererCode(V := E1) = GenererCode(E);istore_N si IDX(V) appartient à [0..3]
GenererCode(V := E1) = GenererCode(E1);istore N si IDX(V) appartient à [4..32767]

GenererCode(I1 ; I2) = GenererCode(I1);GenererCode(I2)

GenererCode(si(BE) alors I1) = GenererCode(BE);
                               ifeq Fin
                               GenererCode(I1);
                          Fin:
GenererCode(si(BE) alors I1 sinon I2) = GenererCode(BE);
                                        ifeq Etiq1
                                        GenererCode(I1);
                                        goto Fin
                                  Etiq1: GenererCode(I2);
                                    Fin:

GenererCode(TantQue BE1 faire I) =
                                  Debut: GenererCode(BE1);
                                         ifeq Fin
                                         GenererCode(I);
                                         goto Debut
                                    Fin:
```

Un exemple

```
while(i<10)
  i:=i+1;

assert i==10: "i != 10 ???"
```

```
Instruction i1 =
  new Sequence(
    new TantQue(
      new Inf(i,new Constante(10)),
      new Affectation(i, new Addition(i,new Constante(1)))
    ),
  new Assertion( new Egal(i, new Constante(10))," i != 10 ???")
);
```

début:
empiler(i)
empiler(10)
si(dépiler()>=dépiler())
alors saut_en fin
empiler(i)
empiler(1)
iadd
i := dépiler();
saut_en début
fin:
assert i==10:"i != 10";

- **Le code généré**

```
while(i<10)
  i:=i+1;

assert i==10: "i != 10 ???"
```

```
#_13:
iload_1
bipush 10
if_icmpge #_22
iconst_1
goto #_24
#_22:
iconst_0
#_24:
ifeq #_33
iload_1
iconst_1
iadd
istore_1
goto #_13
#_33:
iload_1
bipush 10
if_icmpne #_42
iconst_1
goto #_44
#_42:
iconst_0
#_44:
ifne #_55
new java/lang/AssertionError
dup
ldc " i != 10 ???"
invokespecial java/lang
/AssertionError/<init>(Ljava
/lang/Object;)V
athrow
#_55:
```

Démonstration

- !

WhileL, un visiteur
qui génère du bytecode Java

```
public void test_CompilationDeFactoriel(){
  Contexte m = new Memoire();
  Variable x = new Variable(m,"x",5);
  Variable fact = new Variable(m,"fact",1);

Instruction inst = // idem transparent précédent
  new TantQue(
      new Sup(x,new Constante(1)),
      new Sequence(
        new Affectation(fact,new Multiplication(fact,x)),
        new Affectation(x,new Soustraction(x,new Constante(1))))
      );

   Code code = new Code("TestsFactoriel", m);
   VisiteurExprJasmin vej = new VisiteurExprJasmin(m,code);
   VisiteurBoolJasmin vbj = new VisiteurBoolJasmin(vej);
   VisiteurInstJasmin vij = new VisiteurInstJasmin(vej,vbj);

  // vérification par l'exécution de l'assembleur Jasmin
}
```

Le bytecode généré

compatible jasmin *jasmin.sourceforge.net/*

```
.class public TestsFactoriel
.super java/lang/Object
.method public static main([Ljava/lang/String;)V
.limit stack 3
.limit locals 3
   ldc 1
   istore 1
   ldc 5
   istore 2
#_14:
   iload 2
   iconst_1
   if_icmple #_23
   iconst_1
   goto #_25
#_23:
   iconst_0
#_25:
        ifeq #_43
        iload 1
        iload 2
        imul
        istore 1
        iload 2
        iconst_1
        isub
        istore 2
        goto #_14
#_43:
        return
.end method
```

Un détail : la visite du TantQue

```
public Integer visite(TantQue tq){
      int start = code.currentPosition();
      code.addLabel(start);
      int hc = tq.cond().accepter(this.vbj);
      code.add("ifeq");
      int jumpIfAddr = code.currentPosition();
      code.add("labelxxxxx");
      int h = tq.i1().accepter(this);
      code.add("goto");
      int jumpAddr = code.currentPosition();
      code.add("labelxxxxx");
      code.setLabel(jumpAddr,start);
      code.setLabel(jumpIfAddr,code.currentPosition());
      code.addLabel(code.currentPosition());
      return hc + h;
    }
```

Conclusion

- **Le tp associé …**
  - **Un visiteur de WhileL qui génère du bytecode**
    - **http://jfod.cnam.fr/progAvancee/**
  - **Rare exercice de génération de code …**