

## JVM

### .class, chargeur et instances de Class Jeu d'instructions

Cnam NFP121  
jean-michel Douin, douin au cnam pt fr  
version 13 Décembre de l'an 7

JVM 1

## Bibliographie JVM

- ILY96TT.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/dossier/machine\_virtuelle\_java/H1588

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°xxx xxxxx 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/...

JVM 2

## 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
  - Supervision avec JMX
    - Java Management eXtension

JVM 3


## 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



**Sun**  
% java Test  
ou par l'intermédiaire  
d'un navigateur web



**TINI**



**PC**  
> java Test

JVM 4

## Architecture

- **Java Virtual Machine**
  - Chargeur de classes et l'exécutif

- Extrait de [http://www.techniques-ingenieur.fr/dossier/machine\\_virtuelle\\_java/H1588](http://www.techniques-ingenieur.fr/dossier/machine_virtuelle_java/H1588)

JVM 5

## 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**
- **Les instances de la classe java.lang.Class**
  - « Sont des instances comme les autres »
  - Gérées par le ramasse-miettes

JVM 6

## L'exécutif

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

JVM

7

## Sommaire : Classes et *java.lang.Class*

- Le fichier *.class*
  - format
- Le chargeur de *.class*
  - Les chargeurs ...

JVM

8

## Le fichier généré ".class"

- Prémises
- 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 .*

JVM

9

## Prémises

- 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*

JVM

10

## Prémises

- 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*

JVM

11

## Format du ".class" : description informelle

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

JVM

12

## Entête

- **u4 magic;**
- **u2 minor\_version;**
- **u2 major\_version;**

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

Vérification de la Compatibilité

JVM

13

## 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_InterfaceMethodref	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**

JVM

14

## 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;
                }
            }
        }
    }
}
```

JVM

15

## 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
[ 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
```

JVM

16

## Suite du constant\_pool

- [22] tag: 1 length: 22 ([Ljava/lang/String;)V
- [23] tag: 1 length: 4 trie
- [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 ...**

JVM

17

## 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*

JVM

18

## 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;
```

JVM

19

## 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;
```

JVM

20

## 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

JVM

21

## 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;

JVM

22

## 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;
```

JVM

23

## 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
```

JVM

24

## 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
```

JVM

25

## 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

JVM

26

## 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

JVM

27

## ClassFile.attributes

- typedef struct{
- u2 attribute\_name\_index; – u2 attributes\_count;
- u4 attribute\_length; – attribute\_info \*attributes;
- u2 sourcefile\_index;
- } SourceFile\_attribute;

- 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
```

JVM

28

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

JVM

29

## Avant - Après

```
// Decompiled by Jad v1.5.1g. Copyright 2001. Pavel Kouznetsov.
class bulbe{
    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;
                }
            }
        }
    }
}

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++;
            }
        }
    }
}
}}}}
```

JVM

30

## Sommaire suite

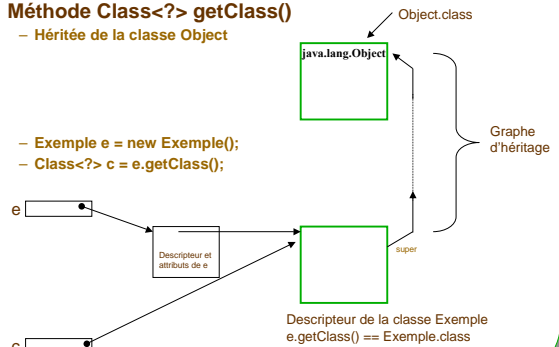
- **Chargeur dynamique de classe**
  - Du « .class » en *Class*
- **Et ensuite**
- **L'exécutif**
  - Machine à pile
  - Registres
  - Jeu d'instructions

JVM

31

## Classe Class

- **Méthode Class<?> getClass()**
  - Héritée de la classe Object



- Exemple `e = new Exemple();`
- `Class<?> c = e.getClass();`

Descripteur de la classe Exemple  
`e.getClass() == Exemple.class`

JVM

32

## java.lang.Class

- **Classe Class et Introspection**
  - `java.lang.Class;`
  - `java.lang.reflect.*;`

### Les méthodes

Constructor[] getConstructors  
Field[] getFields  
Field[] getDeclaredFields  
Method[] getMethods  
Method[] getDeclaredMethods  
.... get ....

- `static Class<?> forName(String name);`
- `static Class<?> forName(String name, boolean init, ClassLoader cl);`
- `ClassLoader getClassLoader()`

JVM

33

## Chargement d'une classe, ClassLoader

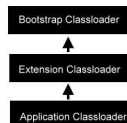
- **Implicite (tardif/paresseux)**
  - Exemple `e;` // pas de chargement
  - Exemple `e = new Exemple();` // chargement (si absente)
  - `Class<Exemple> classe = Exemple.class;` // chargement (si absente)
    - Équivalent à `Class<?> classe = Class.forName("Exemple");`
- **Explicite (immédiat)**
  - `String unNomDeClasse = "XXXXX"`
  - `Class.forName(unNomDeClasse)`
  - `Class.forName(unNomDeClasse, unBooléen, unChargeurDeClasse)`
  - `unChargeurDeClasse.loadClass ( unNomDeClasse )`

JVM

34

## ClassLoader de base

- **ClassLoader**
  - Par défaut celui de la JVM
    - *Bootstrap ClassLoader* en natif (librairies de base, rt.jar)
    - *Extension ClassLoader* en Java ( lib/ext)
    - *Application/System ClassLoader* par défaut
  - *Bootstrap* parent-de *Extension* parent-de *Application*



- *ClassLoader* prédéfinis ?
- Écrire son propre *ClassLoader* ?

JVM

35

## ClassLoader prédéfinis

- **SecureClassLoader**
  - la racine

```

java.net
Class URLClassLoader
  java.lang.Object
    java.lang.ClassLoader
      java.security.SecureClassLoader
        java.net.URLClassLoader
  Direct Known Subclasses:
  MLE
    
```

- **URLClassLoader**

- **RMIClassLoader**

- En distribué
  - Téléchargement des « .class » nécessaires

JVM

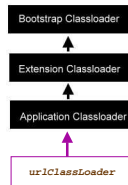
36

## URLClassLoader : un exemple

- **Chargement distant de fichier .class et exécution**

- Depuis cette archive
  - \* <http://jfod.cnam.fr/progAvancee/classes/utiles.jar>
- Ou bien un .class à cette URL
  - \* <http://jfod.cnam.fr/progAvancee/classes/>

1. Création d'une instance de `URLClassLoader`
2. Son parent est le `ClassLoader` par défaut
3. `Class<?> classe = forName(nom,init,urlClassLoader)`
  1. nom le nom de la classe
  2. init : exécution des blocs statiques retardée ou non
4. Recherche de la méthode `main` par introspection



JVM

37

## URLClassLoader : un exemple

```
public class Exemple1{
    URL urlJars = new URL("http://jfod.cnam.fr/progAvancee/classes/utiles.jar");
    URL urlClasses = new URL("http://jfod.cnam.fr/progAvancee/classes/*");

    // par défaut le classloader parent est celui de la JVM
    URLClassLoader classLoader;
    classLoader = URLClassLoader.newInstance(new URL[]{urlJars,urlClasses});

    Class classe = Class.forName(args[0], true, classLoader);

    //introspection ici pour l'exécution de la méthode main
    Method m = classe.getMethod("main", new Class[] {String[].class });
    String[] paramètres = new String[args.length-1];
    System.arraycopy(args,1,paramètres,0,args.length-1);
    m.invoke(null, new Object[]{paramètres});
}

java Exemple1 UneClasse param1 param2 param3
```

JVM

38

## Écrire son propre chargeur de classe

- **ClassLoader est une classe abstraite**

- Il suffit de redéfinir certaines méthodes

```
Class<?> loadClass(String name, boolean resolve)
Recherche si cette classe ne serait pas présente chez le parent
Appel de defineClass qui crée l'instance et l'installe dans l'arborescence
```

```
Class<?> findClass(String name)
appelée par loadClass
```

JVM

39

## Un exemple de Sun

```
class NetworkClassLoader extend ClassLoader {
    String host; int port;
    Map<String,Class<?>> cache = new Hashtable<String,Class<?>>();
    private byte loadClassData(String name)[] {
        // load the class data from the connection
    }
    public synchronized Class loadClass(String name, boolean resolve) {
        Class<?> c = cache.get(name);
        if (c == null) {
            byte data[] = loadClassData(name);
            c = defineClass(data, 0, data.length);
            cache.put(name, c);
        }
        if (resolve)
            resolveClass(c); // édition des liens
        return c;
    }
}
en détail ici http://www.dj.com/mobile/184404484
http://www.koders.com/java/fid182AD13B5471AEF4962D6F58F527AA50E12C3B4C.aspx
```

JVM

40

## Questions ?

- **Plusieurs ClassLoader peuvent-ils co-exister ?**

- Espaces de noms disjoints
  - La même classe peut être chargée dans deux « ClassLoaders » différents

- **Chargement et déchargement de classe**

- Instances de la classe `Class` gérées par le ramasse-miettes
- Déchargement de classe volontaire, par programme ?
- par exemple une mise à jour !
- Impossible ? À suivre...

- **Comment en écrire un ?**

- Les exemples présentés sont en <http://jfod.cnam.fr/progAvancee/classes/>

JVM

41

## Ecrire son propre chargeur : loadClass

```
public class MyClassLoader extends ClassLoader{
    public MyClassLoader(){
        super(MyClassLoader.class.getClassLoader()); // le parent
    }
    protected Class<?> loadClass(String name, boolean resolve)
        throws ClassNotFoundException{
        Class classe = findLoadedClass(name);
        if (classe == null) {
            byte[] classBytes = loadClassBytes(name); // page suivante
            if (classBytes == null){
                return findSystemClass(name);
            }
            classe = defineClass(name, classBytes, 0, classBytes.length);
            if (classe == null)
                throw new ClassNotFoundException(name);
        }
        if (resolve) resolveClass(classe); // recherche de tous les .class
        return classe;
    }
}
```

JVM

42

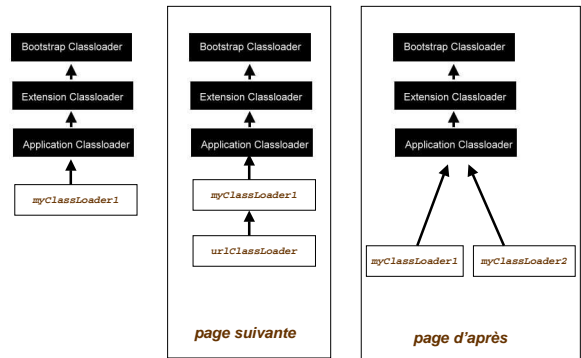
## LoadClassBytes : lecture sur disque du .class

```
private byte[] loadClassBytes(String name){
    // paquetage = répertoire
    String cname = name.replace('.', '/') + ".class";
    FileInputStream in = null;
    try{
        in = new FileInputStream(cname);
        ByteArrayOutputStream buffer = new ByteArrayOutputStream();
        int ch;
        while ((ch = in.read()) != -1){
            byte b = (byte)(ch);
            buffer.write(b);
        }
        in.close();
        return buffer.toByteArray();
    }catch (IOException e){
        if (in != null){
            try {in.close(); } catch (IOException e2) { }
        }
        return null;
    }
}
```

JVM

43

## ClassLoader(s)



JVM

44

## L'exemple revisité

```
public class Exemple1{
    URL urlJars = new
    URL("http://jfod.cnam.fr/progAvancee/classes/utiles.jar");
    URL urlClasses = new URL("http://jfod.cnam.fr/progAvancee/classes/");
    URL[] urls = new URL[]{urlJars, urlClasses};

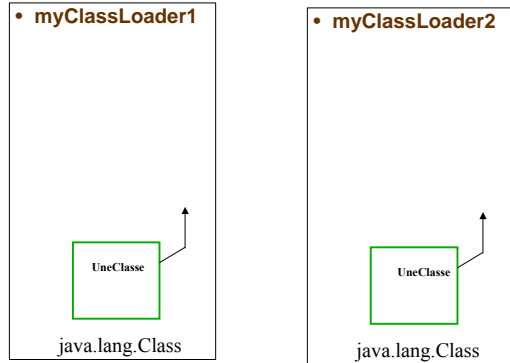
    ClassLoader loader = new MyClassLoader();
    URLClassLoader classLoader;
    classLoader = new URLClassLoader.newInstance(urls, loader);
    Class classe = Class.forName(args[0], true, classLoader);

    Method m = classe.getMethod("main", new Class[] {String[].class });
    String[] paramtres = new String[args.length-1];
    System.arraycopy(args, 1, paramtres, 0, args.length-1);
    m.invoke(null, new Object[]{paramtres});
}
    java Exemple1 UneClasse param1 param2 param3
```

JVM

45

## myClassLoader1 & myClassLoader2



JVM

46

## Exemple

```
ClassLoader loader1 = new MyClassLoader();
ClassLoader loader2 = new MyClassLoader();
Class<?> classe1 = Class.forName(args[0], true, loader1);
System.out.println("classe1 : " + classe1);

Class<?> classe2 = Class.forName(args[0], true, loader2);
System.out.println("classe2 : " + classe2);
System.out.println(" == " + (classe1==classe2));
```

```
Blue:Blue: Terminal - classloader
Options
UneClasse : execution du bloc staticclasse1 : class UneClasse
UneClasse : execution du bloc staticclasse2 : class UneClasse
classe1==classe2 false
```

JVM

47

## Déchargement de classe

- Impossible
  - Bootstrap, Extension et Application
- Possible si
  - Création d'un nouveau chargeur
  - Attention aux références du programme existantes ...

JVM

48



## Conclusion intermédiaire

- **Chargeur dynamique de classe**
  - Instance comme les autres
  - Géré par le ramasse miettes
- **La suite :**
  - Une pause
  - L'exécutif
    - Type de données
    - Les registres de la machine
    - Les instructions

JVM

49

## Pause

- **Combien de classes chargées pour dire « bonjour » toutes les secondes ...**

```
public class Bonjour{
    public static void main(String[] args) throws Exception{
        while(true){
            System.out.println("bonjour");
            Thread.sleep(1000);
        }
    }
}
```

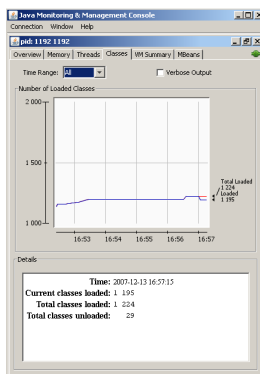
Current classes loaded: 1 200  
Total classes loaded: 1 200  
Total classes unloaded: 0

- Bonjour + jconsole = 1200
- Bonjour seule = 313 classes

JVM

50

## JMX, jconsole



JVM

51

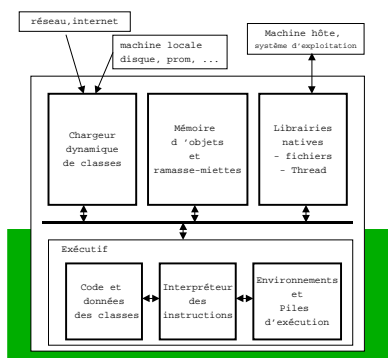
## Pause terminée, sommaire suite

- **JVM c'est aussi**
  - Un exécutif
    - Type de données
    - Les registres
    - RISC ? ou CISC ?
  - Un jeu d'instructions
    - Les grandes familles

JVM

52

## L'exécutif



JVM

53

## 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

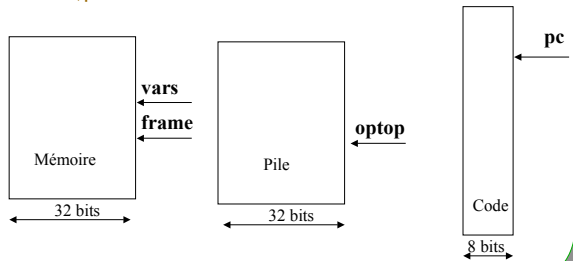
JVM

54

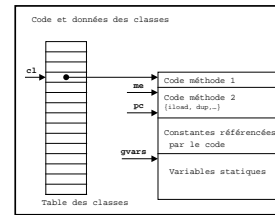
## Les registres

- 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

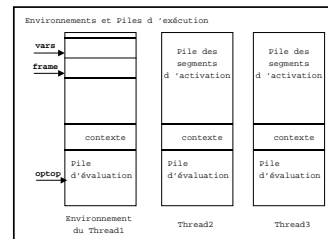
### Aspects dynamiques

- 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 "

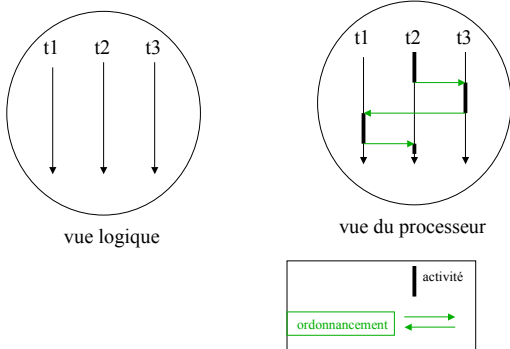
- Une zone mémoire des méthodes de chaque classe (pc)
- Une zone mémoire des variables de classe

## Thread et « JVM virtuelles »



- Une pile par Thread... ou processus léger

## Thread vue logique



## 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"

JVM

61

## 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,
    
```

JVM

62

## 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 : `throw`
- 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`

JVM

63

## 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)

JVM

64

## Format d'une instruction, [LY96] p 275

- **Operation Store int into local variable**

- **Format**

|               |
|---------------|
| <i>istore</i> |
| <i>index</i>  |

- **Forms** `istore = 54(0x36)`

- **Stack** `...,value => ...`

- **Description** : .....

JVM

65

## 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**

JVM

66

## Gestion de la Pile: Constantes et variables

- Lecture d'une variable locale placée sur la pile
  - *iload, iload\_<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>, iconst\_<l>, dconst\_<d>*
- Accès avec déplacement supérieur à 16 bits
  - *wide*                                          *<n>* valeur de l'opérande, 0 à 5

## 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*

## Instructions arithmétiques et logiques

- Addition/soustraction
  - *iadd, ladd, fadd, dadd, isub, lsub, fsub, dsub*
- multiplication, division
  - *imul, fmul, dmul, lmul, idiv, fddiv, ddiv, lddiv, irem, frem, drem, lrem*
- complément
  - *ineg, fneg, dneg, lneg*
- décalage
  - *ishl, ishr, iushr, lshl, lshr, lushr*
- Opérations
  - *ior, lor, iand, land, ixor, lxor*
- 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, ifle, 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);                       |
| 8* if_icmpgt,00,07   |                    | opx2 = depiler();opri = depiler(); |
| 11* iconst_0         |                    | si( opri > opx2) pc = 15;          |
| 12* goto,00,04       |                    | empiler(0);                        |
| 15* iconst_1         |                    | pc = 16;                           |
| 16* iand             |                    | empiler(1);                        |
| 17* istore_0         |                    | empiler(depiler() & depiler());    |
| 18* iload_0          | b = b && (i > 10); | frame[vars+0] = depiler();         |
| 19* ifeq,00,09       |                    | empiler(frame[vars+0]);            |
| 22* iload_1          |                    | si(depiler()==depiler()) pc=28;    |
| 23* bipush,10        |                    | empiler(frame[vars+1]);            |
| 25* if_icmpgt,00,07, |                    | empiler(10);                       |
| 28* iconst_0         |                    | opx2 = depiler();opri = depiler(); |
| 29* goto,00,04       |                    | si( opri > opx2) pc = 32;          |
| 32* iconst_1         |                    | empiler(0);                        |
| 33* istore_0         |                    | pc = 33;                           |
|                      |                    | empiler(1);                        |
|                      |                    | 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      |                        | default : 34               |
| 00,00,00,34      |                        | low : 0                    |
| 00,00,00,00      | case 0 : i = 0; break; | high : 1                   |
| 00,00,00,01      | case 1 : i = 1; break; | <0/24>                     |
| 00,00,00,24      |                        | <1/29>                     |
| 00,00,00,29      |                        |                            |
| 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 implémentée 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*<sub>(void)</sub>

## Exemple invoke static

| Les instructions   | Le source Java                 | Les opérations engendrées       |
|--------------------|--------------------------------|---------------------------------|
|                    | public class MethodesDeClasse{ |                                 |
| static void m(){   |                                | MethodesDeClasse.m1(); // V     |
| invokestatic,00,05 | m1();                          | empiler(5);                     |
| iconst_5           | int i = 5;                     | frame[vars+0] = depiler();      |
| istore_0           |                                | empiler(frame[vars+0]);         |
| iload_0            |                                | MethodesDeClasse.m2(); // (I)V  |
| invokestatic,00,06 | m2(i);                         | empiler(frame[vars+0]);         |
| iload_0            | int j = f(i);                  | MethodesDeClasse.f(); // (I)I   |
| invokestatic,00,04 |                                | frame[vars+1] = depiler();      |
| istore_1           |                                | restaurer;                      |
| return             | }                              |                                 |
| return             | static void m1(){              | restaurer;                      |
| return             | static void m2(int k){         | restaurer;                      |
| return             | public static int f(int x){    | empiler(frame[vars+0]);         |
| iload_0            | return x + 1;                  | empiler(1);                     |
| iconst_1           |                                | empiler(depiler() + depiler()); |
| ladd               |                                | restaurer;                      |
| ireturn            | }                              |                                 |

## Exemple invoke virtual

| Les instructions    | Le source Java                                    | Les opérations engendrées |
|---------------------|---------------------------------------------------|---------------------------|
|                     | 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              | }                                                 |                           |

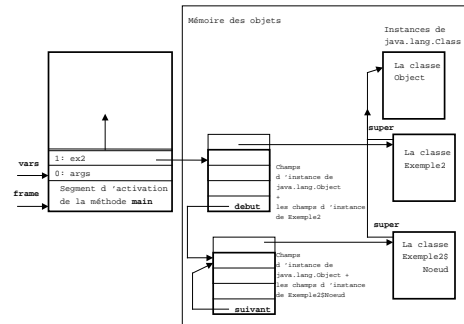
## Instructions sur les instances

- **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`

JVM

79

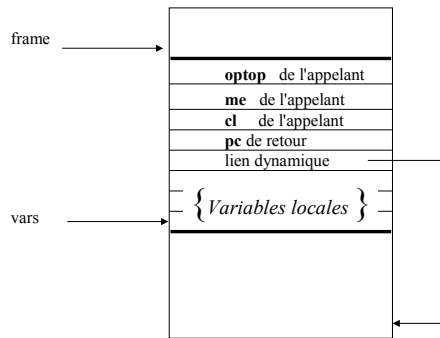
## Pile et mémoire des objets



JVM

80

## Zoom : Segment d'activation dans sa pile



JVM

81

## Exemple putfield, getfield

| Les instructions                 | Le source Java                                                                 | Les opérations engendrées                           |
|----------------------------------|--------------------------------------------------------------------------------|-----------------------------------------------------|
|                                  | <code>public class Point extends java.lang.Object{<br/>private int x,y;</code> |                                                     |
|                                  | <code>// le constructeur par défaut Point.&lt;init&gt;()V</code>               |                                                     |
| <code>aload_0</code>             |                                                                                | <code>empiler(this);</code>                         |
| <code>invokespecial,00,04</code> |                                                                                | <code>java/lang/Object.&lt;init&gt;(); //()V</code> |
| <code>return</code>              |                                                                                |                                                     |
|                                  | <code>public void moveTo(int nx, int ny){</code>                               |                                                     |
| <code>aload_0</code>             | <code>this.x = nx;</code>                                                      | <code>empiler(this);</code>                         |
| <code>iload_1</code>             |                                                                                | <code>empiler(frame[vars+1]);</code>                |
|                                  |                                                                                | <code>ref = depiler();</code>                       |
| <code>putfield,00,07</code>      |                                                                                | <code>((Point)ref).x = depiler();</code>            |
| <code>aload_0</code>             | <code>y = ny;</code>                                                           | <code>empiler(this);</code>                         |
| <code>iload_2</code>             |                                                                                | <code>empiler(frame[vars+2]);</code>                |
|                                  |                                                                                | <code>ref = depiler();</code>                       |
| <code>putfield,00,08</code>      |                                                                                | <code>((Point)ref).y = depiler();</code>            |
| <code>return</code>              |                                                                                |                                                     |
|                                  | <code>public int getX(){</code>                                                |                                                     |
| <code>aload_0</code>             | <code>return x;</code>                                                         | <code>empiler(this);</code>                         |
|                                  |                                                                                | <code>ref = depiler();</code>                       |
| <code>getfield,00,07</code>      |                                                                                | <code>empiler((Point)ref).x;</code>                 |
| <code>ireturn</code>             |                                                                                |                                                     |

JVM

82

## Exceptions et Synchronisation

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

JVM

83

## Exemple

| Les instructions                 | Le source Java                                         | Les opérations engendrées                            |
|----------------------------------|--------------------------------------------------------|------------------------------------------------------|
|                                  | <code>public int distOrigin() throws Exception{</code> |                                                      |
| <code>aload_0</code>             | <code>long x1 = x;</code>                              | <code>empiler(this);</code>                          |
| <code>getfield,00,06</code>      |                                                        | <code>empiler(depiler().x);</code>                   |
| <code>i2l</code>                 |                                                        | <code>empiler((long)depiler());</code>               |
| <code>lstore_1</code>            |                                                        | <code>frame[vars+1]=depiler();</code>                |
| <code>lload_1</code>             | <code>if(x1*x1 &gt;</code>                             | <code>empiler(frame[vars+1]);</code>                 |
| <code>lload_1</code>             | <code>Integer.MAX_VALUE)</code>                        | <code>empiler(frame[vars+1]);</code>                 |
| <code>lmul</code>                |                                                        | <code>empiler(depiler()*depiler());</code>           |
| <code>ldc2_w,00,14</code>        |                                                        | <code>empiler(Integer.MAX_VALUE);</code>             |
| <code>lcmp</code>                |                                                        | <code>empiler(comp(depiler(),depiler()));</code>     |
| <code>ifle,00,11</code>          |                                                        | <code>if (depiler()=-1) pc =24</code>                |
| <code>new ,00,03</code>          | <code>throw new Exception();</code>                    | <code>empiler(new java.lang.Exception());</code>     |
| <code>dup</code>                 |                                                        | <code>empiler(sommet());</code>                      |
| <code>invokespecial,00,08</code> |                                                        | <code>java/lang/Exception.&lt;init&gt;(); //V</code> |
| <code>athrow</code>              |                                                        | <code>propager;</code>                               |
| <code>24*</code>                 |                                                        |                                                      |
|                                  | <code>}</code>                                         |                                                      |

JVM

84

## 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>()JV
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]);
return          }                          propager;

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

JVM

85

## Décompilation

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

JVM

86

## Pause, un décompilateur désassembleur

```
le.class bulbe extrait
class bulbe
{
    bulbe()
    {
        // 0 0:aload_0
        // 1 1:invokespecial #1 <Method void Object()>
        // 2 4:return
    }

    public static void main(String args[])
    {
        int ai[] = new int[6];
        // 0 0:bipush 6
        // 1 2:newarray int[]
        // 2 4:astore_1
        ai[0] = 0;
        // 3 5:aload_1
        // 4 6:iconst_0
        // 5 7:iconst_0
        // 6 8:astore
        ai[1] = 2;
    }
}
```

JVM

87

## Suite, et il en manque...

```
boolean flag = false;
// 27 29:iconst_0
// 28 30:istore_2
while(!flag)
/** 29 31:iload_2
/** 30 32:ifne 86
{
    flag = true;
// 31 35:iconst_1
// 32 36:istore_2
    int i = 0;
// 33 37:iconst_0
// 34 38:istore_3
    while(i < 5)
/** 35 39:iload_3
/** 36 40:iconst_5
/** 37 41:icmpge 83
    {
        if(ai[i] > ai[i + 1])
/** 38 44:aload_1
/** 39 45:iload_3
/** 67 77:inc 3 1
        }
    }
/** 68 80:goto 39
/** 69 83:goto 31
/** 70 86:return
}
```

JVM

88

## Instructions "\_quick" & JIT

- Une optimisation en JIT (Just in Time)
- Réécriture (à la volée) de certaines instructions,
  - Les accès au "constant pool" sont résolus une seule fois
- JIT (Just in Time) en code natif : machine Kaffe de Tim Wilkinson
  - KAFFE v0.5.6 - A JIT and interpreting virtual machine to run Java(tm) code
  - JIT en code natif (i386)

JVM

89

## Les\_quick

- case ldc\_quick, case ldc\_w\_quick, case ldc2\_w\_quick
- case anewarray\_quick, case multianewarray\_quick
- case putfield\_quick, case putfield2\_quick
- case getfield\_quick, case getfield2\_quick
- case putstatic\_quick, case putstatic2\_quick
- case getstatic\_quick, case getstatic2\_quick : /\* 212 \*/
- case getfield\_quick\_w : /\* 227 \*/
- case invokevirtual\_quick : /\* 214 \*/
- case invokevirtual\_quick : /\* 215 \*/
- case invokesuper\_quick : /\* 216 \*/
- case invokestatic\_quick : /\* 217 \*/
- case invokeinterface\_quick : /\* 218 \*/
- case invokevirtualobject\_quick : /\* 219 \*/
- case invokevirtual\_quick\_w : /\* 226 \*/
- 
- case new\_quick : /\* 221 \*/
- case checkcast\_quick : /\* 224 \*/
- case instanceof\_quick : /\* 225 \*/

JVM

90

## Un exemple approprié\* ! Une boucle ...

- un automate d'états déclenché pendant 2\*10 secondes
  - 6 états, (dont 2 automates de 2 états en //)
- 2 styles de génération de code en Java de l'automate
  - 1) à l'aide de *if else...*
  - 2) à l'aide d'un *switch-case*
  - *trace d'exécution sur PowerPC* :
  - on déclenche l'automate pendant env. 10 sec
  - if\_style :
  - L'automate est déclenché : 28109 fois en if\_style
  - on déclenche l'automate pendant env. 10 sec
  - case\_style :
  - L'automate est déclenché : 29663 fois en case\_style

\* Donc non significatif

JVM

91

## \_quick occurrence des instructions

- iconst\_0; 3(0x3);460648
- iconst\_1; 4(0x4);576952
- iconst\_2; 5(0x5);115548
- iconst\_3; 6(0x6);231106
- bipush; 16(0x10);226446
- ldc; 18(0x12); 8
- ldc2\_w; 20(0x14); 57772
- iload; 21(0x15); 57774
- dload; 24(0x18); 115544
- iload\_1; 27(0x1b); 57772
- iload\_2; 28(0x1c); 57772
- iload\_3; 29(0x1d); 57772
- **aload\_0; 42(0x2a); 1759797**
- aload\_1; 43(0x2b); 28109
- aload\_2; 44(0x2c); 29663
- iaload; 46(0x2e); 662047
- istore; 54(0x36); 2
- dstore; 57(0x39); 57774
- istore\_3; 62(0x3e); 57772
- iastore; 79(0x4f); 173340
- ...
- iflt; 155(0x9b); 57772
- ifle; 158(0x9e); 57772
- if\_icmpne; 160(0xa0); 255311
- if\_icmplt; 161(0xa1); 231096
- goto; 167(0xa7); 87439
- tableswitch; 170(0xaa); 29663
- lookupswitch; 171(0xab); 117098
- ireturn; 172(0xac); 57772
- return; 177(0xb1); 8
- getstatic; 178(0xb2); 10
- putstatic; 179(0xb3); 3
- getfield; 180(0xb4); 69
- putfield; 181(0xb5); 34
- invokevirtual; 182(0xb6); 57782
- invokespecial; 183(0xb7); 9
- invokestatic; 184(0xb8); 57774
- new; 187(0xbb); 5
- newarray; 188(0xbc); 8
- **getfield\_quick; 206(0xce); 1644152**
- putfield\_quick; 207(0xcf); 288858
- invokesuper\_quick; 216(0xd8); 2

aload\_0 et getfield\_quick représentent 20 % chacune des occurrences

JVM

92

Table 3.1 page 73 [LY96], types et instructions

| opcode    | byte    | short     | int     | long    | float   | double    | char    | reference |
|-----------|---------|-----------|---------|---------|---------|-----------|---------|-----------|
| Tipush    | bipush  | sipush    |         |         |         |           |         |           |
| Ticonst   |         |           | iconst  | lconst  | fconst  | dconst    |         | aconst    |
| Tload     |         |           | iload   | lload   | float   | dload     |         | aload     |
| Tstore    |         |           | istore  | lstore  |         | dstore    |         | astore    |
| Tinc      |         |           | inc     |         |         |           |         |           |
| Taload    | baload  | saload    | iaload  | laload  | float   | daload    | caload  | aaload    |
| Tastore   | bastore | sastore   | istore  | lstore  | fastore | dastore   | castore | aastore   |
| Tadd      |         |           | iadd    | ladd    | fadd    | dadd      |         |           |
| Tsub      |         |           | isub    | lsub    | fsub    | dsub      |         |           |
| Tmul      |         |           | imul    | lmul    | fmul    | dml       |         |           |
| Tdiv      |         |           | idiv    | ldiv    | fdiv    | ddiv      |         |           |
| Trem      |         |           | irem    | lrem    | frem    | drem      |         |           |
| Tneg      |         |           | ineg    | lneg    | fneg    | dneg      |         |           |
| Tshl      |         |           | shl     | lshl    |         |           |         |           |
| Tshr      |         |           | shr     | lshr    |         |           |         |           |
| Tushr     |         |           | ushr    | lushr   |         |           |         |           |
| Tand      |         |           | iand    | land    |         |           |         |           |
| Tor       |         |           | ior     | lor     |         |           |         |           |
| Txor      |         |           | ixor    | lxor    |         |           |         |           |
| i2T       | i2b     | i2s       |         | i2l     | i2f     | i2d       |         |           |
| I2T       |         |           | i2i     | i2l     | I2f     | I2d       |         |           |
| E2T       |         |           | e2i     | e2l     |         | e2d       |         |           |
| d2T       |         |           | d2i     | d2l     | d2f     |           |         |           |
| Tcmp      |         |           |         | lcmp    |         |           |         |           |
| Tcmpl     |         |           |         |         | fcmpl   | dcmpl     |         |           |
| Tcmpg     |         |           |         |         | fcmpg   | dcmpg     |         |           |
| if_TcmpOP |         | if_icmpOP |         |         |         | if_acmpOP |         |           |
| Treturn   |         |           | ireturn | lreturn | freturn | dreturn   |         | areturn   |

JVM

93

## Outils

- Outil prédéfini javap du J2SE
  - `javap -c un_fichier`

- <http://jasmin.sourceforge.net/>



Jasmin

- <http://jakarta.apache.org/bcel/>
  - Byte Code Engineering Library

- Utilitaire utilisé sur ce support
  - <http://fod.cnam.fr/progAvancees/classes/>

JVM

94

## 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

JVM

95

## factoriel : enfin

```
.class public factoriel
.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
    imul
    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
}
```

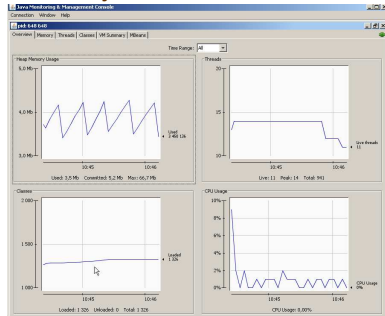
JVM

96



## Pause en fin : JMX

- Outil de supervision
- jconsole PID ou jconsole



## VM summary

The screenshot shows the VM Summary window with the following details:

- Connection name:** pid: 648-648
- Uptime:** 29 days 18 hours 42 minutes
- Virtual Machine:** Java HotSpot(TM) Client VM version 1.6.0\_02-806
- Process CPU time:** 14,437 seconds
- Vendor:** Sun Microsystems Inc.
- JIT compiler:** HotSpot Client Compiler
- Name:** 648@nvaldi
- Total compile time:** 2,387 seconds
- Live threads:** 12
- Current classes loaded:** 1 326
- Peak:** 14
- Total classes loaded:** 1 326
- Daemon threads:** 10
- Total classes unloaded:** 0
- Total threads started:** 242
- Current heap size:** 2 452 Kbytes
- Committed memory:** 6 056 Kbytes
- Maximum heap size:** 65 000 Kbytes
- Pending finalization:** 0 objects
- Garbage collector:** Name = Copy, Collection = 178, Total time spent = 0,335 seconds
- Garbage collector:** Name = MarkSweepCompact, Collection = 0, Total time spent = 0,000 seconds
- Operating System:** Windows 2000 5.0
- Total physical memory:** 523 280 Kbytes
- Architecture:** i86
- Free physical memory:** 51 868 Kbytes
- Number of processors:** 1
- Total swap space:** 1 277 552 Kbytes
- Committed virtual memory:** 23 596 Kbytes
- Free swap space:** 543 612 Kbytes

## Depuis une application

Le nombre de classes

```

ClassLoadingMXBean classLoading =
ManagementFactory.getClassLoadingMXBean();

System.out.println(classLoading.getLoadedClassCount());
System.out.println(classLoading.getTotalLoadedClassCount());
System.out.println(classLoading.getUnloadedClassCount());
    
```

313 classes pour Bonjour  
La connexion avec jconsole engendre le reste ...

À essayer `java -verbose -cp . Exemple`  
Et ensuite jconsole

## Conclusion

## Annexes

## Un interpréteur en C ...

### • Un interpréteur de bytecode

- `classes.c` /\* gestion des classes de l'application \*/
- `classfie.h,c` /\* gestion du .class \*/
- `constpoo.h,c` /\* gestion du contant\_pool \*/
- `exceptio.h,c` /\* exception, uniquement issue de la machine \*/
- `int64.h,c` /\* le type long en Java, les entiers sur 64 bits \*/
- `interpre.h,c` /\* l'interpreteur de bytecode \*/
- `jvm.c` /\* le module "main" \*/
- `memalloc.h,c` /\* en attendant le ramasse miettes \*/
- `object.h,c` /\* gestion des objets, allocation ... \*/
- `signatur.h,c` /\* la signature "L<>DL..." en nombre d'octets \*/
- `statisti.h,c` /\* statistiques \*/
- `types.h,c` /\* définition avec vérification des types de la machine \*/
- `opcodes.h, constant.h`
- soit 6500 lignes de C, sur PPC 47Ko de Code, et 67 KO de données

### • Voir la KVM de SUN / J2ME

## Un interpréteur en C

- **Sommaire**

- Algorithme général
- Structures de données
  - Registres
  - Implantations des classes et méthodes (aspects statiques)
  - Environnement et pile d'exécution (aspects dynamiques)
- Objets et instances
  - Accès aux champs
  - Appels de méthodes virtuelles (par défaut en Java)
  - Le ramasse miettes

## Un interpréteur en C

- **Algorithme principal**

```
– do{  
    ir = next(); /* lecture de l'instruction */  
    switch (ir) {  
        case(...  
        case(iadd) : ipush( ipop() + ipop()); break;  
        case(...  
        case(iand) : ipush( ipop() & ipop()); break;  
        case(...  
        case(...  
    } while( true);
```