

**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**

- [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](http://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](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

---

---

---

---

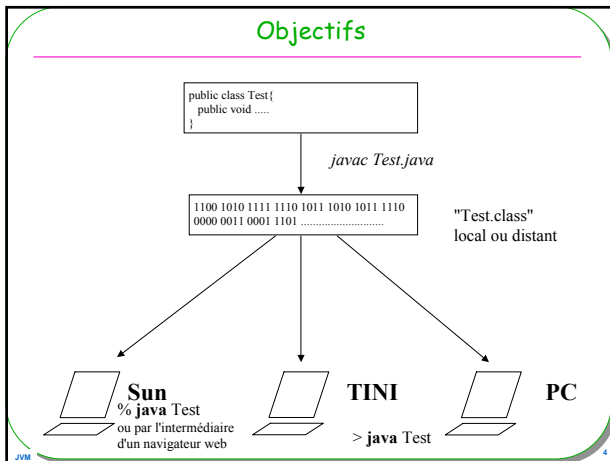
---

---

---

---

## Objectifs



---

---

---

---

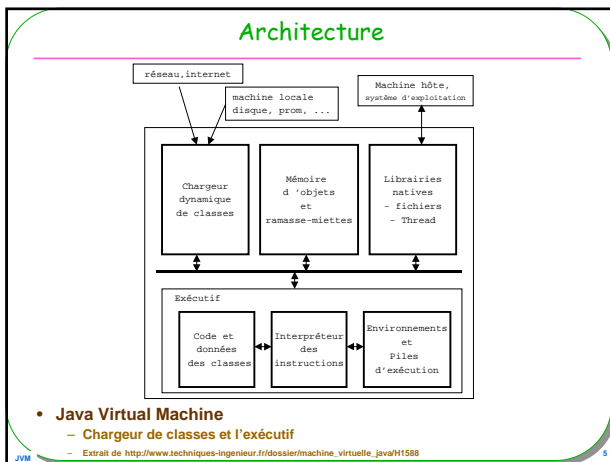
---

---

---

---

## Architecture



---

---

---

---

---

---

---

---

## 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

---

---

---

---

---

---

---

---

## 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*

---

---

---

---

---

---

---

---

## 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;	<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;	
– u2 this_class;	<i>"type" de la classe, son nom,</i>
– u2 super_class;	<i>le nom de la super-classe</i>
– u2 interfaces_count;	
– u2 *interfaces;	<i>les interfaces</i>
– u2 fields_count;	
– field_info *fields;	<i>Variables de la classe ou d'instances</i>
– u2 method_count;	
– method_info *methods;	<i>Les méthodes de classe ou d'instances</i>
– u2 attributes_count;	
– attribute_info *attributes;	<i>Description de ce fichier</i>
}	

---

---

---

---

---

---

---

---

## 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;**

- **cp\_info \*constant\_pool;**
- **typedef struct {**
- **u1 tag;**
- **u1 \*info;**
- **}cp\_info;**

#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

- **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

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

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[] [] --> [ID  
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
```

---

---

---

---

---

---

---

---

## 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

---

---

---

---

---

---

---

---



## Sommaire suite

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

---

---

---

---

---

---

---

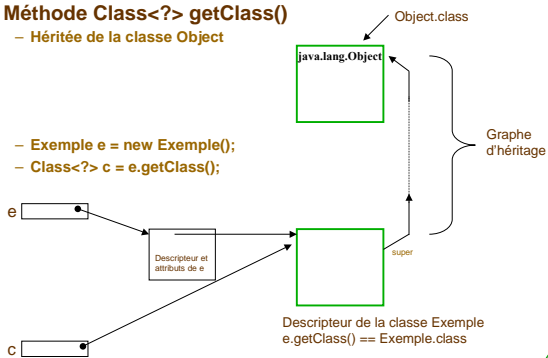
---

## Classe Class

### • Méthode `Class<?> getClass()`

- Héritée de la classe `Object`

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



---

---

---

---

---

---

---

---

## 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()`

---

---

---

---

---

---

---

---

## 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

---

---

---

---

---

---

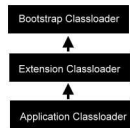
---

---

## 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:  
[ML et](#)

- **URLClassLoader**

- **RMIClassLoader**

- En distribué
  - Téléchargement des « .class » nécessaires

JVM

36

---

---

---

---

---

---

---

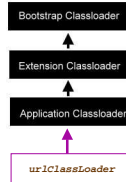
---

## URLClassLoader : un exemple

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

- Depuis cette archive
  - \* <http://jfod.cnam.fr/progAvancees/classes/utiles.jar>
- Ou bien un .class à cette URL
  - \* <http://jfod.cnam.fr/progAvancees/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/progAvancees/classes/utiles.jar");
    URL urlClasses = new URL("http://jfod.cnam.fr/progAvancees/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 extends 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.ddj.com/mobile/184404484
http://www.koders.com/java/182AD13B5471AEF4962D6F58F527AA50E12C3B4C.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://jfoad.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

---

---

---

---

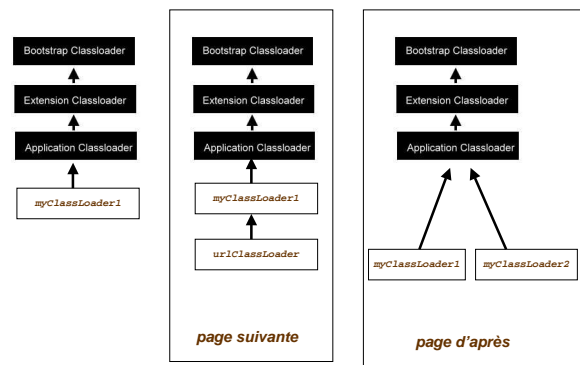
---

---

---

---

## 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

---

---

---

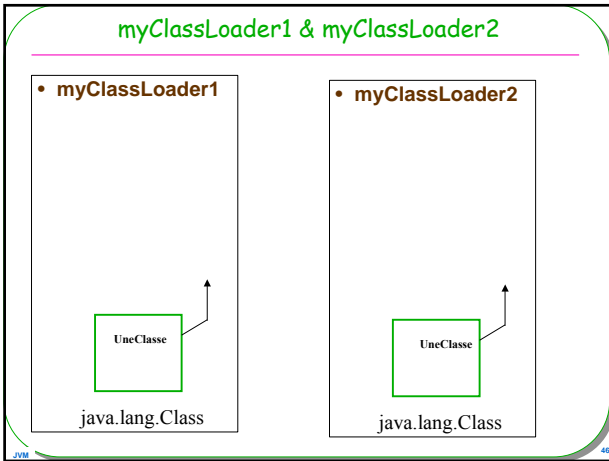
---

---

---

---

---




---

---

---

---

---

---

---

---

### 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));

```

BlueJ: BlueJ: Terminal - classloader

Options

UneClasse : execution du bloc staticclasse1 : class UneClasse  
 UneClasse : execution du bloc staticclasse2 : class UneClasse  
 classe1==classe2 false

---

---

---

---

---

---

---

---

### Déchargement de classe

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

---

---

---

---

---

---

---

---



## Conclusion intermédiaire

- **Chargeur dynamique de classe**

- Instance comme les autres
- Géré par le ramasse miettes

- **La suite :**

- Une pause
  - Type de données
  - Les registres de la machine
  - Les instructions

---

---

---

---

---

---

---

---

## 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

---

---

---

---

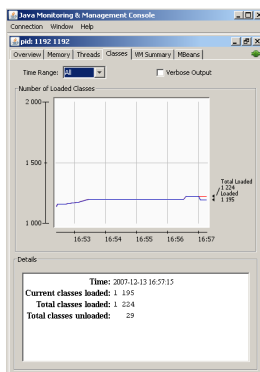
---

---

---

---

## JMX, jconsole



---

---

---

---

---

---

---

---

## 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

---

---

---

---

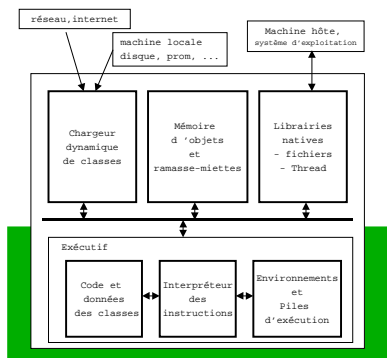
---

---

---

---

## 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

---

---

---

---

---

---

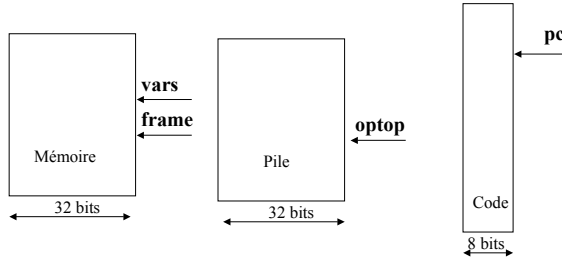
---

---

## Les registres

- 4 registres

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




---

---

---

---

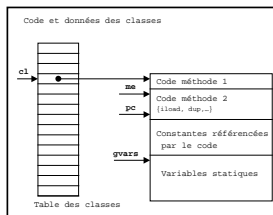
---

---

---

---

## 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

---

---

---

---

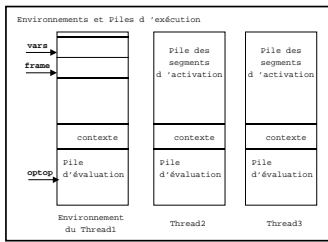
---

---

---

---

## Thread et « JVM virtuelles »



- Une pile par Thread... ou processus léger

---

---

---

---

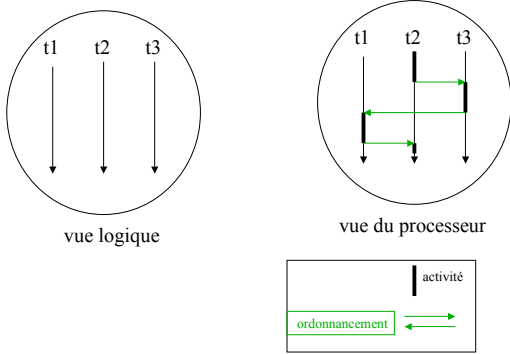
---

---

---

---

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

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 : **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**

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

---

---

---

---

---

---

---

---

---

---



## Instructions arithmétiques et logiques

- **Addition/soustraction**
  - *iadd,ladd,fadd,dadd/isub,lsub,fsub,dsub*
- **multiplication,division**
  - *imul,fmul,dmul,lmul,ldiv,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*
- **Incrément de variable locale**
  - *inc*

JVM

70

---

---

---

---

---

---

---

---

## Opérations de conversions

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

JVM

71

---

---

---

---

---

---

---

---

## Instructions de contrôles

- **Sauts conditionnels**
  - *ifeq,iflt,ifge,ifgt,ifne,ifnull,ifnonnull,if\_icmpeq,if\_icmpne,if\_icmplt,if\_icmpgt,if\_icmple,if\_icmpge,if\_acmpeq,if\_acmpne,icmp,fcmpl,fcmpg,dcmpl,dcmpg*
- **Sauts inconditionnels**
  - *goto,goto\_w,jsr,jsr\_w,ret*
- **Accès indirect**
  - *tableswitch,lookupswitch*

JVM

72

---

---

---

---

---

---

---

---





## 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*<sub>(void)</sub>

---

---

---

---

---

---

---

---

---

---

## Exemple invoke static

| Les instructions                | Le source Java                              | Les opérations engendrées                    |
|---------------------------------|---------------------------------------------|----------------------------------------------|
|                                 | <code>public class MethodesDeClasse{</code> |                                              |
|                                 | <code>static void m(){</code>               |                                              |
| <code>invokestatic,00,05</code> | <code>m1();</code>                          | <code>MethodesDeClasse.m1(); // V</code>     |
| <code>iconst_5</code>           | <code>int i = 5;</code>                     | <code>empiler(5);</code>                     |
| <code>istore_0</code>           |                                             | <code>frame[vars+0] = depiler();</code>      |
| <code>load_0</code>             |                                             | <code>empiler(frame[vars+0]);</code>         |
| <code>invokestatic,00,06</code> | <code>m2(i);</code>                         | <code>MethodesDeClasse.m2(); // (I)V</code>  |
| <code>load_0</code>             | <code>int j = f(i);</code>                  | <code>empiler(frame[vars+0]);</code>         |
| <code>invokestatic,00,04</code> |                                             | <code>MethodesDeClasse.f(); // (I)I</code>   |
| <code>istore_1</code>           |                                             | <code>frame[vars+1] = depiler();</code>      |
| <code>return</code>             | <code>}</code>                              | <code>restaurer;</code>                      |
|                                 | <code>static void m1(){}</code>             | <code>restaurer;</code>                      |
|                                 | <code>static void m2(int k){}</code>        | <code>restaurer;</code>                      |
|                                 | <code>public static int f(int x){</code>    |                                              |
|                                 | <code>return x + 1;</code>                  |                                              |
| <code>iload_0</code>            |                                             | <code>empiler(frame[vars+0]);</code>         |
| <code>iconst_1</code>           |                                             | <code>empiler(1);</code>                     |
| <code>iadd</code>               |                                             | <code>empiler(depiler() + depiler());</code> |
| <code>ireturn</code>            | <code>}</code>                              | <code>restaurer;</code>                      |

---

---

---

---

---

---

---

---

---

---

## Exemple invoke virtual

| Les instructions engendrées      | Le source Java                                                    | Les opérations                     |
|----------------------------------|-------------------------------------------------------------------|------------------------------------|
|                                  | <code>import java.awt.Color;</code>                               |                                    |
|                                  | <code>public class DrawablePoint extends Point{</code>            |                                    |
|                                  | <code>private java.awt.Color color;</code>                        |                                    |
|                                  | <code>public DrawablePoint(Color ic){super();color=ic;}</code>    |                                    |
|                                  | <code>public static void m(){</code>                              |                                    |
|                                  | <code>Point p = new DrawablePoint(Color.blue);</code>             |                                    |
| <code>new,00,01</code>           |                                                                   |                                    |
| <code>dup</code>                 |                                                                   |                                    |
| <code>getstatic,00,06</code>     | <code>// soit java/awt/Color.blue Ljava/awt/Color;</code>         |                                    |
| <code>invokespecial,00,05</code> | <code>//soit DrawablePoint.&lt;init&gt;(Ljava/awt/Color;)V</code> |                                    |
| <code>astore_0</code>            |                                                                   |                                    |
| <code>aload_0</code>             | <code>p.moveTo(10,10);</code>                                     | <code>empiler(p);</code>           |
| <code>bipush,10</code>           |                                                                   | <code>empiler(10);</code>          |
| <code>bipush,10</code>           |                                                                   | <code>empiler(10);</code>          |
| <code>invokevirtual,00,08</code> |                                                                   | <code>Point.moveTo();//II V</code> |
| <code>return</code>              | <code>}</code>                                                    |                                    |

---

---

---

---

---

---

---

---

---

---

## 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

---

---

---

---

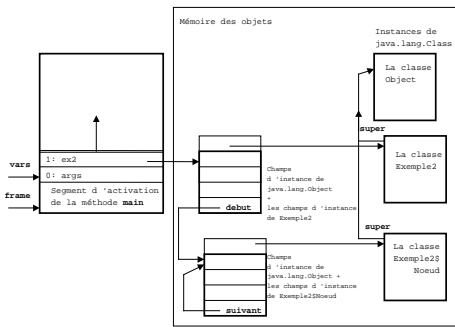
---

---

---

---

## Pile et mémoire des objets



JVM

80

---

---

---

---

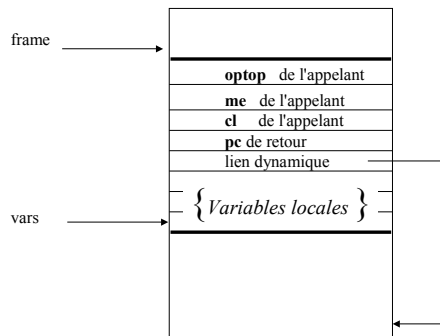
---

---

---

---

## Zoom : Segment d'activation dans sa pile



JVM

81

---

---

---

---

---

---

---

---

## Exemple putfield, getfield

| Les instructions | Le source Java                                                                                    | Les opérations engendrées                                        |
|------------------|---------------------------------------------------------------------------------------------------|------------------------------------------------------------------|
|                  | <pre>public class Point extends java.lang.Object{     private int x,y;</pre>                      |                                                                  |
|                  | <pre>// le constructeur par défaut Point.&lt;init&gt;()V aload_0 invokespecial,00,04 return</pre> | <pre>empiler(this); java/lang/Object.&lt;init&gt;(); //()V</pre> |
|                  | <pre>    public void moveTo(int nx, int ny){</pre>                                                |                                                                  |
| 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           | }                                                                                                 |                                                                  |
|                  | <pre>    public int getX(){</pre>                                                                 |                                                                  |
| aload_0          | return x;                                                                                         | empiler(this);                                                   |
|                  |                                                                                                   | ref = depiler();                                                 |
| getfield,00,07   | }                                                                                                 | empiler(((Point)ref).x);                                         |
| ireturn          |                                                                                                   |                                                                  |

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

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
4* invokespecial,00,06      empiler(sommet());
                          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)
```

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:iastore
        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:load_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:load_3
/** 36 40:iconst_5
/** 37 41:icmpge    83
    {
        if(a[i] > a[i + 1])
/** 38 44:load_1
/** 39 45:load_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)<sup>®</sup> 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



## 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://jod.cnam.fr/progAvancee/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 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
    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

---

---

---

---

---

---

---

---





## 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);
```

---

---

---

---

---

---

---

---