

A Máquina Virtual do Java: JVM

João M. P. Cardoso
FEUP/Universidade do Porto

Exemplos de Linguagens e modos de execução

Language	IR	Implementation(s)
Java	JVM bytecode	Interpreter, JIT
C#	MSIL	JIT (but may be pre-compiled)
Prolog	WAM code	compiled, interpreted
Forth	bytecode	interpreted
Smalltalk	bytecode	interpreted
Pascal	p-code	interpreted--compiled
C, C++	--	compiled (usually)
Perl 6	PVM	interpreted
Parrot	--	interpreted, JIT
Python	--	interpreted

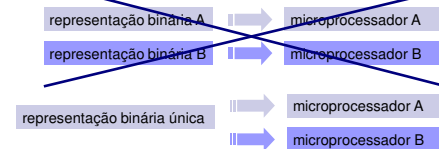
2

Máquinas Virtuais

- Camada de software que permite executar numa máquina real um programa presente num formato não específico a essa máquina real
- Importância cada vez maior nos sistemas embecidos
 - Garantem a portabilidade de aplicações
 - Sem máquinas virtuais, a miríade de sistemas embecidos tornaria um verdadeiro pesadelo o desenvolvimento de aplicações

3

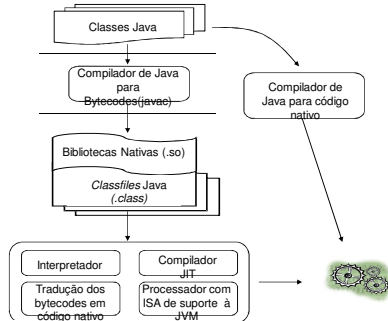
Máquinas Virtuais



- Exemplos
 - JVM, Java Virtual Machine
 - CLR, Common Language Runtime

4

Tecnologia Java



5

Tecnologia Java

- Java
- Classfiles (contêm os bytecodes Java)

```
class Mult {
  static int mult(int a, int b) {
    int r=0;
    for(int i=0; i<b; i++) r+=a;
    return r;
  }
}
```



CAFEBABE00
03002D0020
08...

6

JVM

- A JVM representa uma máquina de computação abstracta
 - Um conjunto de instruções
 - Várias áreas de memória
- Propriedades da JVM:
 - Pilha de operandos e variáveis locais
 - Conjunto de instruções não ortogonal
 - Todas as operações aritméticas usam a pilha de operandos

7

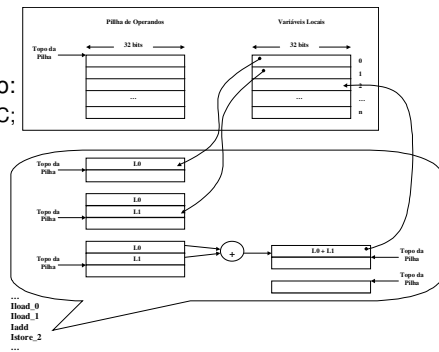
JVM

- A JVM **permite executar qualquer linguagem**, desde que os programas nessa linguagem possam ser descritos sob a forma de *classfiles*
- “class file”
 - Instruções da JVM (chamadas de bytecodes)
 - Uma tabela de símbolos
 - e mais informação

8

JVM

- Exemplo:
 - $A=B+C$;



9

Bytecodes

```
static int mult(int a, int b) {
    int r=0;
    for(int i=0; i<b; i++) r+=a;
    return r;
}
```

```
0: iconst_0
1: istore_2
2: iconst_0
3: istore_3
4: iload_3
5: iload_1
6: if_icmpge 19
9: iload_2
10: iload_0
11: iadd
12: istore_2
13: iinc 3, 1
16: goto 4
19: iload_2
20: ireturn
```

10

Bytecodes

```
static int mult(int a, int b) {
    int r=0;
    for(int i=0; i<b; i++) r+=a;
    return r;
}
```

```
0: iconst_0
1: istore_2
2: iconst_0
3: istore_3
4: iload_3
5: iload_1
6: if_icmpge 19
9: iload_2
10: iload_0
11: iadd
12: istore_2
13: iinc 3, 1
16: goto 4
19: iload_2
20: ireturn
```

11

Bytecodes


```
static int mult(int a, int b) {
    int r=0;
    for(int i=0; i<b; i++) r+=a;
    return r;
}
```

```
0: iconst_0
1: istore_2
2: iconst_0
3: istore_3
4: iload_3
5: iload_1
6: if_icmpge 19
9: iload_2
10: iload_0
11: iadd
12: istore_2
13: iinc 3, 1
16: goto 4
19: iload_2
20: ireturn
```

12

Bytecodes

```
static int mult(int a, int b) {
    int r=0;
    for(int i=0; i<b; i++) r+=a;
    return r;
}
```

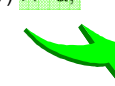


0:	iconst_0
1:	istore_2
2:	iconst_0
3:	istore_3
4:	iload_3
5:	iload_1
6:	if_icmpge 19
9:	iload_2
10:	iload_0
11:	iadd
12:	istore_2
13:	iinc 3, 1
16:	goto 4
19:	iload_2
20:	ireturn

13

Bytecodes

```
static int mult(int a, int b) {
    int r=0;
    for(int i=0; i<b; i++) r+=a;
    return r;
}
```




0:	iconst_0
1:	istore_2
2:	iconst_0
3:	istore_3
4:	iload_3
5:	iload_1
6:	if_icmpge 19
9:	iload_2
10:	iload_0
11:	iadd
12:	istore_2
13:	iinc 3, 1
16:	goto 4
19:	iload_2
20:	ireturn

14

Bytecodes

```
static int mult(int a, int b) {
    int r=0;
    for(int i=0; i<b; i++) r+=a;
    return r;
}
```




0:	iconst_0
1:	istore_2
2:	iconst_0
3:	istore_3
4:	iload_3
5:	iload_1
6:	if_icmpge 19
9:	iload_2
10:	iload_0
11:	iadd
12:	istore_2
13:	iinc 3, 1
16:	goto 4
19:	iload_2
20:	ireturn

15

Bytecodes

```
static int mult(int a, int b) {
    int r=0;
    for(int i=0; i<b; i++) r+=a;
    return r;
}
```

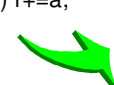


0:	iconst_0
1:	istore_2
2:	iconst_0
3:	istore_3
4:	iload_3
5:	iload_1
6:	if_icmpge 19
9:	iload_2
10:	iload_0
11:	iadd
12:	istore_2
13:	iinc 3, 1
16:	goto 4
19:	iload_2
20:	ireturn

16

Bytecodes

```
static int mult(int a, int b) {
    int r=0;
    for(int i=0; i<b; i++) r+=a;
    return r;
}
```



0:	iconst_0
1:	istore_2
2:	iconst_0
3:	istore_3
4:	iload_3
5:	iload_1
6:	if_icmpge 19
9:	iload_2
10:	iload_0
11:	iadd
12:	istore_2
13:	iinc 3, 1
16:	goto 4
19:	iload_2
20:	ireturn

17


Bytecodes

■ Outro exemplo:

```
...
// Number of array elements N
int N=4;

int L2NORM = 0;

for(int i=0; i<N;i++) {
    short Aux = X[i] - Y[i];
    L2NORM += Aux*Aux;
}
...
```



1:	iconst_0
2:	istore_2
3:	iconst_0
4:	istore_3
5:	goto 28
6:	aload_0
7:	iload_3
8:	saload
9:	aload_1
10:	iload_3
11:	saload
12:	isub
13:	i2s
14:	istore_4
15:	iload_2
16:	iload_4
17:	iload_4
18:	imul
19:	iadd
20:	istore_2
21:	iinc 3 1
22:	iload_3
23:	iconst_4
24:	if_icmplt 7
25:	iload_2
26:	ireturn

18

Bytecodes

■ Outro exemplo:

```
...
// Number of array elements N
int N=4;

int L2NORM = 0;

for(int i=0; i<N;i++) {
    short Aux = X[i] - Y[i];
    L2NORM += Aux*Aux;
}
...
```

```

1  iconst_0
2  istore_2
3  iconst_0
4  istore_3
5  goto 28
-----
6  aload_1
7  iload_2
8  aload_3
9  aload_1
10 iload_3
11 saload
-----
12 iadd
13 i2s
14 istore_4
15 iload_2
16 iload_4
17 iload_4
18 imul
19 iadd
20 istore_2
21 linc 31
-----
22 iload_3
23 iconst_4
24 if_icmplt 7
-----
25 iload_2
26 ireturn

```

19

Bytecodes

■ Tipos de instruções:

- load/store de uma variável local (*iload_1*, *fload 4*, *istore 6*, etc.)
- Manipulação da pilha (*pop*, *dup*, etc.)
- Aritméticas, lógicas, e conversões (*iadd*, *fsub*, *ineg*, *i2s*, *d2f*, etc.)
- Saltos condicionais e incondicionais (*if_icmplt*, *ifne*, *goto*, *jsr*, etc.)
- Comparações (*lcmp*, etc.)
- Criação e manipulação de objectos e arrays (*new*, *newarray*, etc.)
- Acesso aos atributos de um objecto (*putfield*, *getfield*, etc.)
- Invocação de métodos (*invokeinterface*, *invokespecial*, etc.)
- Constantes (*iconst_1*, *ldc* "Hello", etc.)
- Switch (*lookupswitch* e *tableswitch*)
- Definição de regiões de código síncronas (*monitorenter* e *monitorexit*)

20

Classfile

■ Representação binária da classe que lhe deu origem

```
ClassFile {
    signature data
    constant pool
    inheritance information (superclass and interface(s))
    field information
    method information
    attributes
}
```

CAFEBABE00
03002D0020
08...

21

Informação nos ficheiros classe

Constant Pool (tabela de símbolos) + Bytecodes de cada método + ...

<pre> #1 = Method #3:#12; // java/lang/Object.<init>:()V #2 = class #13; // Mult #3 = class #14; // java/lang/Object #4 = Asciz <init>; #5 = Asciz ()V; #6 = Asciz Code; #7 = Asciz LineNumberTable; #8 = Asciz mult; #9 = Asciz (I)I; #10 = Asciz SourceFile; #11 = Asciz Mult.java; #12 = NameAndType #4:#5; // "<init>:()V" #13 = Asciz Mult; #14 = Asciz java/lang/Object; </pre>	<pre> static int mult(int,int); 0: iconst_0 1: istore_2 2: iconst_0 3: istore_3 4: iload_3 5: iload_1 6: if_icmpge 19 9: istore_3 10: iload_0 11: iadd 12: istore_2 13: iinc 3, 1 16: goto 4 19: iload_2 20: ireturn </pre>	<pre> method mult(I)I Stack=2, Locals=4, Args_size=2 code_length = 21 LineNumberTable: line 5: 0 line 6: 2 line 7: 19 1. 2. class Mult { 3. 4. static int mult(int a, int b) { 5. int r=0; 6. for(int i=0; i<b; i++) r+=a; 7. return r; 8. } 9. } </pre>
--	---	---

22

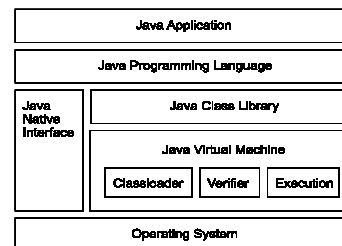
Informação nos ficheiros classe

Constant Pool (tabela de símbolos) + Bytecodes de cada método + ...

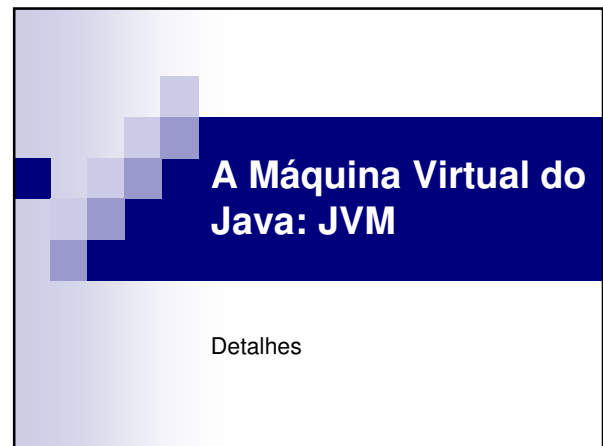
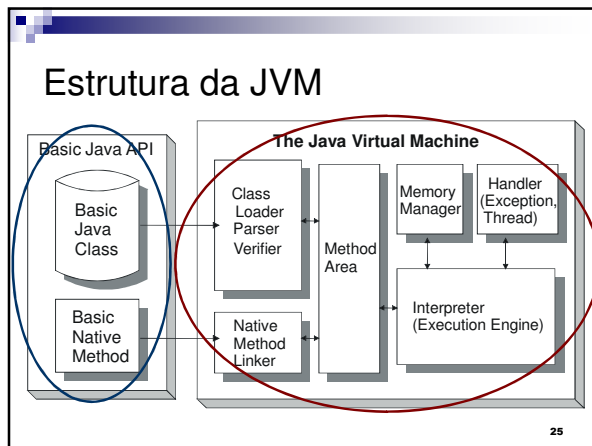
<pre> #1 = Method #3:#12; // java/lang/Object.<init>:()V #2 = class #13; // Mult #3 = class #14; // java/lang/Object #4 = Asciz <init>; #5 = Asciz ()V; #6 = Asciz Code; #7 = Asciz LineNumberTable; #8 = Asciz mult; #9 = Asciz (I)I; #10 = Asciz SourceFile; #11 = Asciz Mult.java; #12 = NameAndType #4:#5; // "<init>:()V" #13 = Asciz Mult; #14 = Asciz java/lang/Object; </pre>	<pre> Mult() 0: aload_0 1: invokespecial #1; //Method java/lang/Object.<init>:()V 4: return </pre>	<pre> method Mult() Stack=1 Locals=1 Args_size=1 code_length = 5 LineNumberTable: line 2: 0 1. 2. class Mult { 3. 4. static int mult(int a, int b) { 5. int r=0; 6. for(int i=0; i<b; i++) r+=a; 7. return r; 8. } 9. } </pre>
--	---	---

23

Sistemas Java



24



Tipos de dados

- **Tipos primitivos** E o boolean?
 - integer
 - signed, two's-complement integers
 - Byte (8-bit); short (16-bit); int (32-bit); long (64-bit)
 - char, 16-bit unsigned integers representing Unicode version 1.1.5 characters
 - IEEE 754 standard floating-point types:
 - float (32-bit); double (64-bit)

Tipos de dados

- **Tipos referência:**
 - class types
 - interface types
 - array types
- Uma referência pode ter como valor a referência especial **null**

Armazenamento de dados

- Espaço afecto utilizando **word** como unidade
 - 32-bits numa máquina de 32 bits
 - 64-bit numa máquina de 64 bits
- Uma palavra deve armazenar um valor do tipo
 - byte, char, short, int, float, reference, returnAddress, ou um apontador nativo
- São utilizadas duas palavras para armazenar valores do tipo
 - long e double

Criação de Objectos e Manipulação

- Criação de uma instância de uma classe:
 - **new**
- Criação de um novo array:
 - **newarray, anewarray, multianewarray**
- Acesso a campos de classes e de instâncias:
 - **getfield, putfield, getstatic, putstatic**

Criação de Objectos e Manipulação

- Load de um elemento de um array para a pilha de operandos:
 - **baload, caload, saload, iaload, laload, faload, daload, aaload**
- Store de um valor da pilha de operandos num elemento de um array:
 - **bastore, castore, sastore, iastore, lastore, fastore, dastore, aastore**
- Tamanho de um array:
 - **arraylength**
- Verifica propriedades de instâncias ou de arrays:
 - **instanceof, checkcast**

31

Invocação de Métodos

- **invokevirtual**
 - Invoca um método de um objecto: resolvido pelo *this* ou pelo despacho dinâmico
- **invokeinterface**
 - Invoca um método implementado por um interface: a procura do método realizada é tendo por base o objecto (*this*) em tempo-real
- **invokespecial**
 - Invoca um método de uma instância que requer tratamento especial: um método de inicialização da instância *<init>*, um método privado, ou um método da super-classe
- **invokestatic**
 - Invoca um método de uma classe (*static*): utiliza o nome da classe

32

Retorno de Métodos

- As instruções de retorno são distinguidas pelo tipo:
 - **ireturn** (valores do tipo: byte, char, short, ou int)
 - **lreturn**
 - **freturn**
 - **dreturn**
 - **areturn**
 - **return** (para métodos declarados como void)

33

Objectos e Herança

```
class A {
    public int y,z;
    public int f() {
        return y+z;
    }
}
```

```
class B extends A {
    public int w,x;
    public int f() {
        return w-x;
    }
}
```

- Objectos consistem de
 - Estado (campos)
 - Comportamento (métodos)
- Herança
 - Aumenta a classe base com
 - novos campos
 - novos métodos
 - Redefine alguns métodos da classe base

34

Objectos e Herança

```
class A {
    public int y,z;
    public int f() {
        return y+z;
    }
}
```

```
class B extends A {
    public int w,x;
    public int f() {
        return w-x;
    }
}
```

- Se B herda a A, então
 - Em qualquer ponto em que o programa declara objectos do tipo A
 - Tem de ser capaz de executar com objectos do tipo B

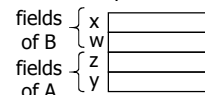
35

Objectos e Herança

```
class A {
    public int y,z;
    public int f() {
        return y+z;
    }
}
```

```
class B extends A {
    public int w,x;
    public int f() {
        return w-x;
    }
}
```

- Guarda campos em espaços de memória contíguos
- Campos da classe que herda guardados depois da classe base



- Propriedade chave: campos da classe base guardados com o mesmo offset da classe que herda

36

Tabelas de Métodos (vtable)

- Cada objecto é constituído por dados e por métodos
 - O objecto tem associado um ponteiro para uma tabela de métodos
 - A tabela armazena os ponteiros para os métodos
- Quando a classe é carregada pela JVM, a tabela de métodos da classe é preenchida com os endereços de entrada de cada método
- Quando um objecto é criado, a sua tabela de métodos apontará para a tabela de métodos da sua classe

37

Chamadas a Métodos Virtuais

```
class A {
    public int y,z;
    public int f() { return y+z; }
}

class B extends A {
    public int w,x;
    public int f() { return w-x; }
}

class C {
    static int p(A) { return a.f(); }
}
```

A a = new A();
int i = C.p(a); // p() calls f() in class A

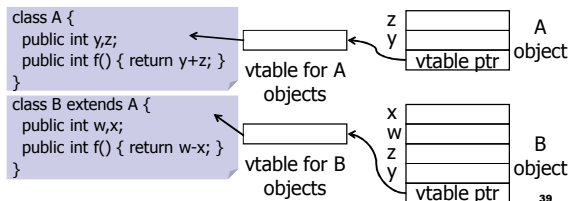
B b = new B(); // A b = new B();
int j = C.p(b); // p() calls f() in class B

Método invocado depende da classe do objecto A

38

Abordagem: vtable

- Cada objecto tem uma referência para uma **vtable**
- Uma **vtable** por classe
- Cada **vtable** contém apontadores para os métodos de cada objecto da sua classe



39

Tabelas de Métodos

```
class A {
    int foo() {...}
    void bar() {...}
};

class B extends A {
    int foo() {...}
    float boo() {...}
};
```

Method table of A

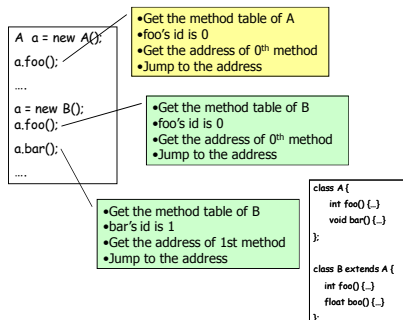
address of foo()	0
address of bar()	1

Method table of B

address of foo()	0
address of bar()	1
address of boo()	2

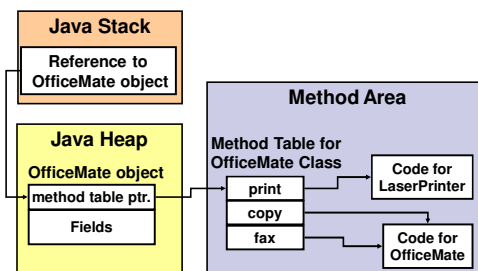
40

Tabelas de Métodos



41

Tabelas de métodos



42

Despacho Dinâmico / *Dynamic Dispatch*

- Propriedade muito utilizada em linguagens orientados por objectos, como o C++ e o Java
- Resolução de qual o código do método a executar durante a execução da aplicação (em tempo-real)

43

Despacho Dinâmico

```
...
void ex(Shape S) {
    ...
    int a = S.area();
    ...
}

...
interface Shape { public int area(); }
class Circle implements Shape {
    int radius;
    public int area() { return (int) (2*3*radius*radius); }
    Circle(int a) { radius = a; }
}
class Rect implements Shape {
    int l1, l2;
    public int area() { return l1*l2; }
    Rect(int a, int b) { l1=a; l2=b; }
}
}
```

44

A Máquina Virtual do Java: JVM

Detalhes das instruções

Sumário do conjunto de instruções

- Uma instrução JVM consiste em:
 - Um opcode de um byte que especifica a operação a realizar
 - Zero ou mais operandos que fornecem os argumentos utilizados pela operação
 - Os argumentos podem identificar variáveis locais, constantes, ou outros argumentos através de referências à *constant pool*

```
do {
    fetch an opcode;
    if (operands) fetch operands;
    execute the action for the opcode;
} while (there is more to do);
```

46

Tipos e a JVM

- Muitas instruções da JVM vêm em grupos com variantes para diferentes tipos de dados
 - `iadd` `ladd` `fadd` `dadd`
(variações da adição)
- Convenção utilizada é que a primeira letra na menmónica da instrução representa o tipo

type	code
int	i
long	l
float	f
double	d
byte	b
char	c
short	s
reference	a

47

Instruções de Load e de Store

- Transfere valores entre as variáveis locais e a pilha de operandos:
 - `iload`, `iload_0`, `iload_1`, ..., `aload`.
 - ..., `lstore`, `fstore`, ..., `astore`,.
- Carregar uma constante para a pilha de operandos:
 - `bipush`, `sipush`, `ldc`, `ldc_w`, `ldc2_w`, `aconst_null`,...

48

JVM: try-catch

```
void catchOne() {
    try {
        tryItOut();
    } catch (TestExc e) {
        handleExc(e);
    }
}
```

Method void catchOne()

0	aload_0	// Beginning of try block
1	invokevirtual #6	// Method Example.tryItOut()V
4	return	// End of try block; normal return
5	astore_1	// Store thrown value in local variable 1
6	aload_0	// Push this
7	aload_1	// Push thrown value
8	invokevirtual #5	// Invoke handler method: // Example.handleExc(LTestExc;)V
11	return	// Return after handling TestExc

Exception table:

From	To	Target Type
0	4	5 Class TestExc

49

Instruções aritméticas

- Combinam tipicamente dois operandos no topo da pilha e colocam o resultado da operação no topo da pilha
 - Add: iadd, ladd, fadd, dadd
 - Subtract: isub, lsub, fsub, dsub
 - Multiply: imul, lmul, fmul, dmul
 - Divide: idiv, lddiv, fdiv, ddiv
 - Remainder: irem, lrem, frem, drem
 - Negate: ineg, lneg, fneg, dneg
 - Shift: ishl, ishr, iushr, lshl, lshr, lushr
 - Bitwise OR, AND, OR exclusivo: ior, lor; iand, land; ixor, lxor
 - Local variable increment: iinc

50

Instruções de conversões de tipos

- Conversões para cima:
 - int to long, float, or double (i2l, i2f, i2d)
 - long to float or double (l2f, l2d)
 - float to double (f2d)
- Conversões para baixo:
 - int to byte, short, or char (i2b, i2c, i2s)
 - long to int (l2i)
 - float to int or long (f2i, f2l)
 - double to int, long, or float (d2i, d2l, and d2f)

51

Acesso a campos

- Exemplo:** `getstatic` 178 (0xb2)
 - Operação: coloca no topo da pilha o valor de um campo estático
- `getstatic`
`indexbyte1`
`indexbyte2`
- `Indexbyte1` e `indexbyte2` referem dois bytes que indexam itens da *constant pool*
 - Nome da classe

52

Criação de objectos

- new**, 187 (0xbb)
 - Operação: cria um novo objecto (o endereço é colocado no topo da pilha)
- `new`
`indexbyte1`
`indexbyte2`
- `Indexbyte1` e `indexbyte2` referem dois bytes que indexam itens da *constant pool*
 - Nome da classe

53

Instruções de Controlo

- Salto condicional:
 - `ifeq`, `iflt`, `ifle`, `ifne`, `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`.
- Salto condicional de múltiplos alvos:
 - `tableswitch`, `lookupswitch`.
- Salto incondicional:
 - `goto`, `goto_w`, `jsr`, `jsr_w`, `ret`.

54

Outras instruções

- Um total de aproximadamente 230 instruções
- Manipulação da pilha (e.g., troca de elementos no topo, duplica elemento)
- Sincronização – monitorenter, ...
- Excepções – athrow
- I/O?
 - Não!
 - Métodos nativos de uma dada classe

55

Instruções da JVM: sumário

Category	No.	Example
arithmetic operation	24	iadd, lsub, frem
logical operation	12	iand, lor, ishl
numeric conversion	15	int2short, f2l, d2l
pushing constant	20	bipush, sipush, ldc, iconst_0, fconst_1
stack manipulation	9	pop, pop2, dup, dup2
flow control instructions	28	goto, ifne, ifge, if_null, jsr, ret
managing local variables	52	astore, istore, aload, iload, aload_0
manipulating arrays	17	aastore, bastore, aaload, baload
creating objects and array	4	new, newarray, anewarray, multianewarray
object manipulation	6	getfield, putfield, getstatic, putstatic
method call and return	10	invokevirtual, invokestatic, areturn
miscellaneous	5	throw, monitorenter, breakpoint, nop

56

Exemplo de *classfile*

```

0000 | CA FE BA BE | 00 03 2D 20 00 20 08 00 1D 07 00 0E | .....*.....
0010 | 07 00 16 07 00 1E 07 00 1C 09 00 05 00 0B 0A 00 | .....
0020 | 03 00 0A 0A 00 02 00 09 0C 00 0C 00 15 0C 00 1A | .....
0030 | 00 1F 0C 00 14 00 18 01 00 07 70 72 69 6E 74 6C | .....printl
0040 | 6E 01 00 0D 43 6F 6E 73 74 61 6E 74 56 61 6C 75 | n...ConstantValu
0050 | 65 01 00 13 6A 61 76 61 2F 69 6F 2F 50 72 69 6E | e...java/io/Prin
0060 | 74 53 74 72 65 61 6D 01 00 0A 45 7B 63 65 70 74 | tStream...Except
0070 | 69 6F 6E 73 01 00 0F 4C 69 6E 65 4E 75 00 62 65 | ions...LineNumbe
0080 | 72 54 61 62 6C 65 01 00 0A 53 6F 75 72 63 65 46 | rtable...SourceF
0090 | 69 6C 65 01 00 0E 4C 6F 63 61 6C 56 61 72 69 61 | ile...LocalVaria
00a0 | 62 6C 65 73 01 00 04 43 6F 64 65 01 00 03 6F 75 | bles...Code...ou
00b0 | 74 01 00 15 28 4C 6A 61 76 61 2F 6C 61 6E 67 2F | t...(Ljava/lang/
00c0 | 53 74 72 69 6E 67 3B 2F 56 01 00 10 6A 61 76 61 | String)V...java
00d0 | 2F 6C 61 6E 67 2F 4F 62 6A 65 63 74 01 00 04 6D | /lang/Object...m
00e0 | 61 69 6E 01 00 0F 48 65 6C 6C 6F 57 6F 72 6C 64 | ain...HelloWorld
00f0 | 2E 6A 61 76 61 01 00 16 28 58 4C 6A 61 76 61 2F | .java...((Ljava/
0100 | 6C 61 6E 67 2F 53 74 72 69 6E 67 3B 2F 56 01 00 | al...LineNumbe
0110 | 06 3C 69 6E 69 74 3F 01 00 15 4C 6A 61 76 61 | class HelloWorld {
0120 | 69 6F 2F 50 72 69 6E 74 53 74 72 65 61 6D |     public static void main(String args[]) {
0130 | 00 10 6A 61 76 61 2F 6C 61 6E 67 2F 53 79 |         System.out.println("Hello world!");
0140 | 65 6D 01 00 0C 48 65 6C 6C 6F 20 57 6F 72 |     }
0150 | 21 01 00 0A 48 65 6C 6C 6F 57 6F 72 6C 64 | }
0160 | 03 28 2F 56 00 01 00 04 00 02 00 00 00 00 00 |
0170 | 00 09 00 17 00 19 00 01 00 13 00 00 25 00 02 | .....$.
0180 | 00 01 00 00 09 B2 00 06 12 01 B6 00 08 B1 00 | .....

```

57

Verificação na JVM

- Examina os bytecodes e verifica se estão conforme regras pré-estabelecidas
 - Magic number 0xCAFEBABE
 - Tamanho do ficheiro
 - Bytecodes válidos
 - Organizado de acordo com regras
 - Utilização de tipos de dados de acordo com as regras
 - Uso de variáveis locais válido
 - Nenhum *overflow* da pilha de operandos
 - Verifica consistência da pilha (conteúdo da pilha tem de ter o mesmo número e tipo de operandos independentemente da instrução que atingiu a instrução actual)
 - Argumentos das funções são de tipos válidos
 - Etc.

58

Verificação na JVM

- Verifica em tempo-real se existe consistência
 - Saltos para fora dos bytecodes do método em causa
 - Acesso a métodos visíveis
 - Indexação de elementos de arrays dentro dos limites
 - etc.

59

Várias Máquinas Virtuais de Java

- JVM (J2EE & J2SE)
 - Máquina virtual para computadores
- CVM, KVM (J2ME)
 - Dispositivos embebidos
 - Reduz algumas das propriedades da JVM para dispositivos com limitações de memória e de desempenho
- JCVM (Java Card)
 - Endereça Smart Cards
 - Inclui o mínimo de propriedades da máquina virtual

60

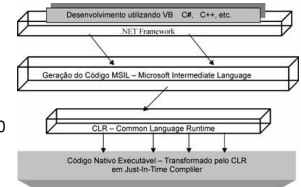
CVM e KVM (K Virtual Machine)

- CVM (*Compact Virtual Machine*) é uma máquina virtual para dispositivos que necessitem do conjunto de facilidades do Java 2, mas com simplificações em alguns aspectos
 - Para dispositivos a executar em microprocessadores/microcontroladores de 32-bits com mais do que 2 MB de memória
- K virtual machine (KVM), máquina virtual optimizada para dispositivos electrónicos de consumo com restrições
 - Para dispositivos com microprocessadores ou microcontroladores RISC/CISC de 16/32-bits e com memória disponível de 160 KB

61

Outras máquinas virtuais

- Common Language Infrastructure (CLI) ou Common Language Runtime (CLR)
 - *Microsoft Intermediate Language* (IL): cerca de 220 instruções
 - Desenvolvida tendo em mente o suporte para uma gama variada de linguagens
 - www.golddotnet.com (contém uma lista de linguagens compiláveis para o CLI)



62