

ESPECIFICAÇÃO DO FORMATO .CLASS

BASEADO NO LIVRO "THE JAVA VIRTUAL MACHINE SPECIFICATION", JAVA SE 8 EDITION,
TIM LINDHOLM, FRANK YELLIN, GILAD BRACHA & ALEX BUCKLEY

INTRODUÇÃO

Arquivo .class

- Definição de uma única classe ou interface
- Stream de bytes de 8 bits
 - Itens multi-bytes armazenados em big-endian ordem
- Notação com sintaxe similar a linguagem C
- Tipos de dados

u1	typedef unsigned char;
u2	typedef unsigned short;
u4	typedef unsigned int;
u8	typedef unsigned long;

stdint.h é do C99

INTRODUÇÃO

Arquivo .class

- Itens são sucessivos
 - Armazenados sequencialmente sem caracteres de preenchimento ou alinhamento
- Tabelas
 - Consistem de zero ou mais itens de tamanho variável
 - Índice não pode ser traduzido diretamente em um offset (deslocamento)
- Arrays
 - Consistem de zero ou mais itens de tamanho fixo
 - Podem ser indexados como arrays em C

INTRODUÇÃO

Nomes de classes e interfaces

- Representação completa de nomes qualificados
 - Por exemplo, o nome da classe Thread é
`java.lang.Thread`
- São representados como constantes em UTF-8
 - O ponto qualificador é substituído pela barra de divisão
 - Por exemplo, o nome da classe Thread passa a ser
`java/lang/Thread`

INTRODUÇÃO

Descritores

- Strings representando o tipo de field ou método
 - Representado em UTF-8
- Gramática
 - Tipo de classe, instância ou variável local
 - FieldDescriptor:
 - FieldType
 - ComponentType:
 - FieldType
 - FieldType:
 - BaseType
 - ObjectType
 - ArrayType
- BaseType:
 - B
 - C
 - D
 - F
 - I
 - J
 - S
 - Z
- ObjectType:
 - L <classname> ;
- ArrayType:
 - [ComponentType

INTRODUÇÃO

DESCRITORES DE CAMPO (*FIELD*)

Caracter Tipo Base	Tipo	Interpretação
B	byte	Byte com sinal
C	char	Caracter Unicode codificado em UTF-16
D	double	Ponto flutuante (dupla precisão)
F	float	Ponto flutuante (precisão simples)
I	int	Inteiro
J	long	Inteiro longo
L<nome_classe>;	referência	Instância da classe <nome_classe>
S	short	Inteiro curto com sinal
Z	boolean	<i>true</i> ou <i>false</i>
[referência	Uma dimensão de array

I, LJAVA/LANG/OBJECT;; [[[D <-> VARIÁVEL DE INSTÂNCIA DO TIPO INT, VARIÁVEL DE INSTÂNCIA DO TIPO OBJECT, ARRAY MULTIDIMENSIONAL DE DOUBLES

INTRODUÇÃO

DESCRITORES DE MÉTODOS

Representam tipos dos parâmetros passados para o método e do valor que ele retorna:

MethodDescriptor:

(ParameterDescriptor*) ReturnDescriptor

ParameterDescriptor:

FieldType

ReturnDescriptor:

FieldType

V

- Por exemplo, para o método cujo protótipo é:

Object mymethod(int i, double d, Thread t)

o descritor é dado por:

(IDLjava/lang/Thread;)Ljava/lang/Object;

ESTRUTURA INTERNA DE ARQUIVO .CLASS

ESTRUTURA CLASSFILE

```
ClassFile {  
    u4                magic;  
    u2                minor_version;  
    u2                major_version;  
    u2                constant_pool_count;  
    cp_info           constant_pool [constant_pool_count-1];  
    u2                access_flags;  
    u2                this_class;  
    u2                super_class;  
    u2                interfaces_count;  
    u2                interfaces [interfaces_count];  
    u2                fields_count;  
    field_info        fields [fields_count];  
    u2                methods_count;  
    method_info        methods [methods_count];  
    u2                attributes_count;  
    attribute_info      attributes [attributes_count];  
}
```


ESTRUTURA INTERNA DE ARQUIVO .CLASS

MAGIC E VERSÃO

u4 magic

- Assinatura do arquivo .class
 - valor 0xCAFEBAE

u2 minor_version (m)

u2 major_version (M)

- Indicam a versão do formato na forma M.m
 - Versões podem ser ordenadas lexicograficamente
 - Por exemplo, $1.5 < 2.0 < 2.1$
 - São definidas pela Oracle

ESTRUTURA INTERNA DE ARQUIVO .CLASS

POOL DE CONSTANTES

u2 constant_pool_count

- Número de entradas na tabela constant_pool + 1.
 - $1 \leq \text{índice constant_pool} < \text{constant_pool_count}$
 - Se n é um índice válido para uma constante do tipo long ou double, então o índice n+1 é inválido!

cp_info constant_pool []

- Tabela de estruturas representando
 - string, nomes de classes ou interfaces, nomes de campos, ...
- referidos dentro de estruturas do ClassFile e suas subestruturas.
 - O formato de cada entrada é indicado pelo byte de "tag".

ESTRUTURA INTERNA DE ARQUIVO .CLASS

CONTROLE DE ACESSO

u2 access_flags

- Máscara de bits que especifica permissões de acesso e propriedades da classe ou interface

Nome do Flag	Valor	Interpretação
ACC_PUBLIC	0x0001	Declarada pública: pode ser acessada de fora do pacote.
ACC_FINAL	0x0010	Declarada final: não pode ter subclasses .
ACC_SUPER	0x0020	Chama métodos de superclasse via a instrução <i>invokespecial</i> .
ACC_INTERFACE	0x0200	É interface, não uma classe.
ACC_ABSTRACT	0x0400	Declarada abstrata: não pode ser instanciada.
ACC_SYNTHETIC	0x1000	Declarado <i>synthetic</i> (não presente no fonte)
ACC_ANNOTATION	0x2000	Declarado como tipo <i>annotation</i>
ACC_ENUM	0x4000	Declarado como tipo <i>enum</i>

- Todos os demais bits são reservados para uso futuro
 - Devem ser definidos como zero.

ESTRUTURA INTERNA DE ARQUIVO .CLASS

DEFINIÇÃO DA CLASSE

u2 this_class

- O valor desse item deve ser um índice válido da tabela constant_pool
 - Aponta para uma estrutura CONSTANT_Class_info
 - Representa a classe ou interface **definida por esse arquivo**:

```
CONSTANT_Class_info {  
    u1 tag;  
    u2 name_index;  
}
```

ESTRUTURA INTERNA DE ARQUIVO .CLASS

DEFINIÇÃO DA SUPER CLASSE

u2 super_class

- O valor desse item deve ser um índice válido da tabela constant_pool
 - Aponta para uma estrutura CONSTANT_Class_info
 - Representa a super classe direta (classe mãe) da classe definida nesse arquivo
- 0 (zero) se a classe não for derivada.
 - Se for 0 essa classe **estende a classe Object**

ESTRUTURA INTERNA DE ARQUIVO .CLASS

INTERFACES

u2 interfaces_count

- Número de entradas no array interfaces[].
 - $0 \leq \text{índice interfaces} < \text{interfaces_count}$
- Número de superinterfaces diretas dessa classe ou interface.

u2 interfaces []

- Cada valor no array interfaces deve ser um índice válido na tabela constant_pool
 - A entrada nesse índice deve ser uma estrutura do tipo CONSTANT_Class_info
 - Representa uma interface que é uma superinterface direta da classe ou interface representada nesse arquivo.

ESTRUTURA INTERNA DE ARQUIVO .CLASS

CAMPOS

u2 fields_count

- Número de variáveis de classe ou variáveis de instâncias declaradas nesta classe ou interface.
- Número de estruturas field_info na tabela fields []

field_info fields []

- Cada entrada na tabela fields deve ser uma estrutura field_info
 - Descrição completa de um campo dessa classe ou interface.
- A tabela inclui apenas os campos declarados na classe ou interface.
 - Não inclui campos herdados das superclasses ou superinterfaces.

ESTRUTURA INTERNA DE ARQUIVO .CLASS

MÉTODOS

u2 methods_count

- Número de estruturas method_info na tabela methods[]

method_info methods []

- Cada entrada na tabela methods deve ser uma estrutura method_info
 - Descrição completa de um método nessa classe ou interface.
 - Se o método é **não nativo ou não abstrato**, as instruções da JVM que o implementam também são fornecidas.
 - Todos os tipos de métodos:
 - declarados pela classe ou interface, de instância, estáticos, iniciação de instância, e qualquer método de iniciação de classe ou interface.
- Essa tabela não inclui itens representando métodos herdados de superclasses ou superinterfaces.

ESTRUTURA INTERNA DE ARQUIVO .CLASS

ATRIBUTOS

u2 attributes_count

- Número de estruturas attributes_info na tabela attributes[]

attribute_info attributes []

- Cada entrada na tabela attributes deve ser uma estrutura attribute_info
- **A implementação da JVM deve ignorar em silêncio qualquer atributo que não reconheça.**

POOL DE CONSTANTES

Informações simbólicas estão armazenadas na tabela `constant_pool`

Cada entrada em `constant_pool` tem a forma

```
cp_info {  
    u1 tag;  
    u1 info[ ];  
}
```

O byte de **tag** define o tipo da informação em `cp_info`

POOL DE CONSTANTES

TIPOS VÁLIDOS DE TAGS

Tipo de Constante	Valor
CONSTANT_Class	7
CONSTANT_Fieldref	9
CONSTANT_Methodref	10
CONSTANT_InterfaceMethodref	11
CONSTANT_String	8
CONSTANT_Integer	3
CONSTANT_Float	4
CONSTANT_Long	5
CONSTANT_Double	6
CONSTANT_NameAndType	12
CONSTANT_Utf8	1
CONSTANT_MethodHandle	15
CONSTANT_MethodType	16
CONSTANT_InvokeDynamic	18

POOL DE CONSTANTES

ESTRUTURA CONSTANT_Class_info

Representa uma Classe ou Interface

```
CONSTANT_Class_info {
```

```
    u1 tag;                                // valor 7
```

```
    u2 name_index;
```

```
}
```

- **name_index** é índice válido para a tabela constant_pool
 - a entrada deve ser uma estrutura **CONSTANT_Utf8_info** representando um nome **completo qualificado** da classe ou interface nesse arquivo.
 - Exemplos
 - Ljava/lang/Thread;

POOL DE CONSTANTES

ESTRUTURA CONSTANT_Fieldref_info

Representa um field

```
CONSTANT_Fieldref_info {  
    u1 tag;                                // valor 9  
    u2 class_index;  
    u2 name_and_type_index;  
}
```

- **class_index** é índice válido para a tabela constant_pool
 - **CONSTANT_Utf8_info** representando **nome completo da classe ou interface** que contem a **declaração desse field**
- **name_and_type_index** é um índice válido para constant_pool
 - **CONSTANT_NameAndType_info** indicando o **nome e o descritor do field**.

POOL DE CONSTANTES

ESTRUTURA CONSTANT_Methodref_info

Representa um método

```
CONSTANT_Methodref_info {  
    u1 tag;                                // valor 10  
    u2 class_index;  
    u2 name_and_type_index;  
}
```

- **class_index** é índice válido para a tabela constant_pool
 - **CONSTANT_Utf8_info** representando **nome completo da classe** que contem a **declaração desse método**.
- **name_and_type_index** é um índice válido para constant_pool
 - **CONSTANT_NameAndType_info** indicando o **nome e o descritor do método**.
 - Se iniciar com '<' então o nome deve ser <init> representando um método de iniciação (construtor) de instância.

POOL DE CONSTANTES

ESTRUTURA CONSTANT_InterfaceMethodref_info

Representa um

```
CONSTANT_InterfaceMethodref_info {  
    u1 tag;                                // valor 11  
    u2 class_index;  
    u2 name_and_type_index;  
}
```

- **class_index** é índice válido para a tabela constant_pool
 - **CONSTANT_Utf8_info** representando **nome completo da interface** que contem a declaração desse método.
- **name_and_type_index** é um índice válido para constant_pool
 - **CONSTANT_NameAndType_info** indicando o **nome e o descritor do método**.

POOL DE CONSTANTES

ESTRUTURA CONSTANT_NameAndType_info

Representa um field ou método sem indicar classe ou interface a que pertence

```
CONSTANT_NameAndType_info {  
    u1 tag; // valor 12  
    u2 name_index;  
    u2 descriptor_index;  
}
```

- **name_index** é índice válido para a tabela constant_pool
 - **CONSTANT_Utf8_info** representando um nome simples de **field** ou **método** ou ainda o nome do **método especial <init>**
- **descriptor_index** é um índice válido para constant_pool
 - **CONSTANT_Utf8_info** representando um **descriptor válido** de field ou de método.

POOL DE CONSTANTES

ESTRUTURA CONSTANT_Utf8_info

Representa valores strings constantes, inclusive Unicode

```
CONSTANT_Utf8_info {
```

```
    u1 tag;                                // valor 1
```

```
    u2 length;
```

```
    u1 bytes [length];
```

```
}
```

- **length** indica o **número de bytes no array bytes**
 - **não indica** o número de bytes da string
 - essa não finaliza com o carater nulo.
- **bytes** contêm os bytes da string
 - nenhum byte pode ter valor zero ou estar no intervalo 0xf0 a 0xff, isto é [240, 255].

POOL DE CONSTANTES

ESTRUTURA CONSTANT_Utf8_info

Utiliza códigos de 1, 2 ou 3 bytes

- Não utiliza códigos de 4 bytes do formato UTF-8

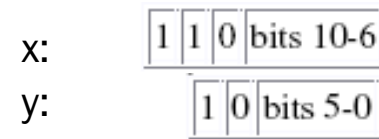
- Um byte

Caracteres no intervalo '\u0001' a '\u007F', isto é [1,127] são representados por

0	bits 6-0
---	----------

- Dois bytes

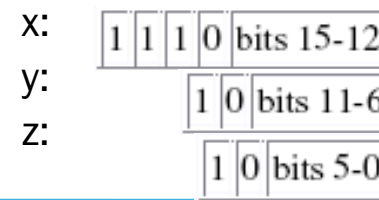
- Zero (null) e caracteres no intervalo '\u0080' a '\u07FF', isto é [128, 2047] são representados por um par de bytes x e y:



O caracter é formado por $((x \& 0x1f) \ll 6) + (y \& 0x3f)$.

- Três bytes

- Caracteres no intervalo '\u0800' to '\uFFFF', isto é, [2048, 65535] são formados por três bytes x, y e z:



O caracter é formado por $((x \& 0xf) \ll 12) + ((y \& 0x3f) \ll 6) + (z \& 0x3f)$

POOL DE CONSTANTES

CONSTANT_String_info

Representa objetos constantes do tipo String

```
CONSTANT_String_info {
```

```
    u1 tag;                                // valor 8
```

```
    u2 string_index; }
```

- string_index é índice válido para a tabela constant_pool
 - CONSTANT_Utf8_info representando a seqüência de caracteres com a qual o objeto String será iniciado.

POOL DE CONSTANTES

CONSTANT_Integer_info

Representa uma constante inteira de 4 bytes

```
CONSTANT_Integer_info {
```

```
    u1 tag;                                // valor 3
```

```
    u4 bytes;
```

```
}
```

- bytes

- Representa o valor da constante int, em big-endian.

POOL DE CONSTANTES

CONSTANT_Float_info

Representa uma constante de ponto flutuante de 4 bytes

```
CONSTANT_Float_info {
```

```
    u1 tag;                                // valor 4
```

```
    u4 bytes;
```

```
}
```

- bytes
 - Representa o valor da constante float, em big-endian, no formato de precisão simples de ponto flutuante padrão IEEE 754.

POOL DE CONSTANTES

CONSTANT_Float_info

O valor representado é inicialmente convertido em uma constante inteira (bits):

- se bits é 0x7f800000, o valor float value é infinito positivo.
- se bits é 0xff800000, o valor float é infinito negativo.
- se bits está na faixa de 0x7f800001 até 0x7fffffff ou na faixa de 0xff800001 até 0xffffffff, o valor float é NaN.

- senão, sejam s, e, m três valores computados de bits como:

```
int s = ((bits >> 31) == 0) ? 1 : -1;
```

```
int e = ((bits >> 23) & 0xff);
```

```
int m = (e == 0) ?
```

```
(bits & 0x7fffff) << 1 :
```

```
(bits & 0x7fffff) | 0x800000;
```

O valor float é o resultado da expressão $s \cdot m \cdot 2^{e-150}$.

POOL DE CONSTANTES

CONSTANT_Long_info

Representa uma constante inteira de 8 bytes, armazenados em big-endian ordem. Ocupa dois índices na tabela `constant_pool`.

```
CONSTANT_Long_info {  
    u1 tag;                                // valor 5  
    u4 high_bytes;                          // unsigned  
    u4 low_bytes;                           // unsigned  
}
```

- O valor long armazenado é pela expressão:
 $((\text{long}) \text{high_bytes} \ll 32) + \text{low_bytes}$

POOL DE CONSTANTES

CONSTANT_Double_info

Representa constante de ponto flutuante de 8 bytes, armazenados em big-endian ordem, no formato de ponto flutuante de dupla precisão IEEE 754. Ocupa dois índices na tabela constant_pool.

```
CONSTANT_Double_info {  
    u1 tag;                                // valor 6  
    u4 high_bytes;                         // unsigned  
    u4 low_bytes;                          // unsigned  
}
```

- O valor armazenado é inicialmente convertido em uma constante inteira long pela expressão:
$$(\text{long bits}) = ((\text{long}) \text{high_bytes} \ll 32) + \text{low_bytes};$$

POOL DE CONSTANTES

CONSTANT_Double_info

- se bits é 0x7ff0000000000000L, o valor float value é infinito positivo.
- se bits é 0xfff0000000000000L, o valor float é infinito negativo.
- se bits está na faixa de 0x7ff0000000000001L até 0x7ffffffffffffL ou na faixa de 0xfff0000000000001L até 0xfffffffffffffL, o valor float é NaN.
- senão, sejam s, e, m três valores computados de bits como:
int s = ((bits >> 63) == 0) ? 1 : -1;
int e = ((bits >> 52) & 0x7ffL);
long m = (e == 0) ?
 (bits & 0xfffffffffffffL) << 1 :
 (bits & 0xfffffffffffffL) | 0x100000000000000L;
- O valor double é o resultado da expressão $s \cdot m \cdot 2^{e-1075}$.

POOL DE CONSTANTES

EXEMPLOS

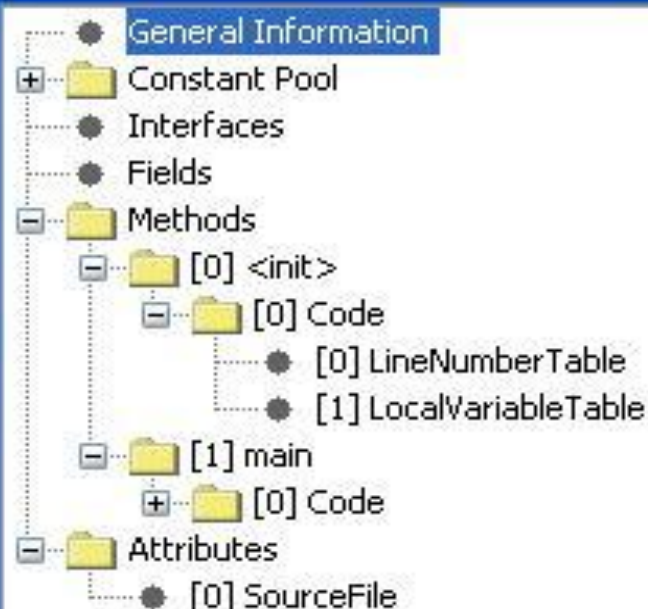
```
/*
The HelloWorld class is an application that
displays
"Hello World!" to the standard output.
*/
public class HelloWorld {

    // Display "Hello World!"
    public static void main (String args[ ]) {
        System.out.println ("Hello World!");
    }
}
```

```
class Teste{
    public static int soma (int a, int b) {
        return a + b;
    }
    public static void main (String[ ] s)
    {
        int i;
        float x = 100.f;
        double y = 1000.;
        int j = 20;
        for (i = 0; i < 10; i++){
            j = soma(j, 10);
        }
        System.out.print (j);
    }
}
```



C:\aulas\06-2\sb\teoria\HelloWorld.class



Minor version: 0
Major version: 46
Constant pool count: 34
Access flags: 0x0021 [public]
This class: cp_info #2 <HelloWorld>
Super class: cp_info #4 <java/lang/Object>
Interfaces count: 0
Fields count: 0
Methods count: 2
Attributes count: 1

FIELDS

Cada field é descrito por uma field_info. Dois campos na mesma classe não podem ter o mesmo nome.

```
field_info {  
    u2 access_flags;  
    u2 name_index;  
    u2 descriptor_index;  
    u2 attributes_count;  
    attribute_info attributes[attributes_count];  
}
```

Fields

Controle de Acesso

u2 access_flags

- Máscara de hexadecimais especificando permissões de acesso e propriedades do campo.

Nome do Flag	Valor	Interpretação
ACC_PUBLIC	0x0001	Declarado público: pode ser acessado de fora do pacote.
ACC_PRIVATE	0x0002	Declarado privado: contexto restrito à definição da classe.
ACC_PROTECTED	0x0004	Declarado protegido: pode ser usado na classe e nas subclasses.
ACC_STATIC	0x0008	Declarada estático: variável de classe e não de instância.
ACC_FINAL	0x0010	Declarado final: não pode ter seu valor alterado após a iniciação.
ACC_VOLATILE	0x0040	Declarado volátil: não pode ser colocado em cache. A Thread que a usa deve conciliar sua cópia dessa variável com a mestra toda. vez
ACC_TRANSIENT	0x0080	Declarado transiente: não pode ser lido ou gravado por um gerente de objetos persistente.
ACC_SYNTHETIC	0x1000	Declarado sintético: não é presente no código fonte. Código nativo
ACC_ENUM	0x4000	Declarado como um elemento de uma enum

- Todos os demais hexadecimais são reservados para uso futuro
 - Devem ser definidos como zero.

Fields

Nome e Descritor do Campo

u2 name_index

- **name_index** é índice válido para a tabela constant_pool
 - **CONSTANT_Utf8_info** representando um **nome simples de field** (isto é, um identificador Java)

u2 descriptor_index

- **descriptor_index** é um índice válido para constant_pool
 - **CONSTANT_Utf8_info** representando um **descritor de campo válido**.

FIELDS

ATRIBUTOS DO CAMPO

u2 attributes_count

- Número de atributos do campo

attribute_info attributes [attributes_count]

```
attribute_info {  
    u2 attribute_name_index;  
    u4 attribute_length;  
    u1 info [attribute_length];  
}
```

- A implementação da JVM **deve ignorar em silêncio qualquer atributo que não reconheça.**
 - Toda implementação deve reconhecer e ler corretamente o atributo **ConstantValue**.

MÉTODOS

Cada método, inclusive método de iniciação de instância, classe ou interface, é descrito por uma estrutura `method_info`.

- métodos na mesma classe não podem ter mesmo nome e descritor

`u2 methods_count;`

- Número de estruturas `method_info` na tabela `methods`

`method_info methods [methods_count];`

```
method_info {  
    u2 access_flags;  
    u2 name_index;  
    u2 descriptor_index;  
    u2 attributes_count;  
    attribute_info attributes[attributes_count];  
}
```


Métodos

Controle de Acesso

u2 access_flags

- Máscara de hexadecimais especificando permissões de acesso e propriedades do método.

Nome do Flag	Valor	Interpretação
ACC_PUBLIC	0x0001	Público: pode ser acessado de fora do pacote.
ACC_PRIVATE	0x0002	Privado: acesso restrito à definição da classe.
ACC_PROTECTED	0x0004	Protegido: pode ser chamado na classe e subclasses.
ACC_STATIC	0x0008	Estático: método de classe (chamado sem referir objeto)
ACC_FINAL	0x0010	Final: não pode ser sobre-escrito em subclasses.
ACC_SYNCHRONIZED	0x0020	Sincronizado: requer um <i>monitor</i> antes de ser executado (Thread)
ACC_BRIDGE	0x0040	Método Bridge, gerado por um compilador
ACC_VARARGS	0x0080	Declarado com um número variável de argumentos
ACC_NATIVE	0x0100	Nativo: implementado em linguagem não Java (C, C++, Assembly)
ACC_ABSTRACT	0x0400	Abstrato: sem definição, deve ser sobreescrito em uma subclasse.
ACC_STRICT	0x0800	Strictfp: utiliza modo de ponto flutuante FP-strict (não normalizado)
ACC_SYNTHETIC	0x1000	Sintético: não presente no código fonte

- Todos os demais hexadecimais são reservados para uso futuro
 - Devem ser definidos como zero.

Métodos

Nome e Descritor do Método

u2 name_index

- **name_index** é índice válido para a tabela constant_pool
 - **CONSTANT_Utf8_info** representando um **nome** especial de método (<init> ou <clinit>) ou um nome simples, válido como nome de método.

u2 descriptor_index

- **descriptor_index** é um índice válido para constant_pool
 - **CONSTANT_Utf8_info** representando um **descritor de método válido**.

MÉTODOS

ATRIBUTOS DO MÉTODO

u2 attributes_count

- Número de atributos do Método

attribute_info attributes [attributes_count]

attribute_info {

u2 attribute_name_index;

u4 attribute_length;

u1 info [attribute_length];

}

- A implementação da JVM deve ignorar em silêncio qualquer atributo que não reconheça.

Toda implementação deve reconhecer e ler corretamente os atributos Code, Exceptions e Sourcefile.

Usados nas estruturas ClassFile, field_info, method_info e code_attribute.

- Podem ser pré-definidos na especificação JVM ou definidos por compilador Java*

ATRIBUTOS

u2 attribute_name_index

- attribute_name_index é índice válido para a tabela constant_pool
 - **CONSTANT_Utf8_info** representando o nome do atributo.

u4 attribute_length

- attribute_length **indica o tamanho, em bytes**, do restante do atributo
 - não **inclui os 6 bytes que contêm o índice do nome e o comprimento do atributo.**

ATRIBUTOS

PREDEFINIDOS NA ESPECIFICAÇÃO .CLASS

ConstantValue

Code

Deprecated (op)

Exceptions

InnerClasses

SourceFile (op)

Synthetic

LineNumberTable (op)

LocalVariableTable (op)

JVM deve implementar

- Code
- ConstantValue
- Exceptions

JVM Java 2 e superior

- InnerClasses
- Synthetic

Opcionais

- Os demais; a JVM pode ler esses atributos ou ignorá-los.

ATRIBUTOS

ATRIBUTO **ConstantValue**

Em estrutura **field_info** inicializa variáveis (implícitas ou explícitas) **estáticas**

- Somente um valor é possível por variável
 - Utilizado para iniciar a variável **antes da chamada do método de iniciação de classe ou interface** que contém a variável
 - Essa estrutura aponta para estrutura **CONSTANT_<tipo>_info** que contém a constante a ser utilizada na iniciação.
- Se **field_info** contém um atributo **ConstantValue** associado a uma variável não estática, a JVM deve ignorá-lo em silêncio.
- Toda JVM deve reconhecer atributos **ConstantValue**

ATRIBUTOS

ATRIBUTO ConstantValue

Estrutura field_info de variável estática

```
field_info {  
    u2 access_flags;  
    u2 name_index;  
    u2 descriptor_index;  
    u2 attributes_count;  
    attribute_info attributes  
        [attributes_count];  
    Tipo constantValue  
}
```

u2 attribute_name_index

- attribute_name_index é índice para constant_pool
 - CONSTANT_Utf8_info representando a string “**ConstantValue**”.

u4 attribute_length

- assume sempre o valor 2

u2 constantvalue_index

- constantvalue_index é índice para constant_pool com uma estrutura **CONSTANT_<tipo>_info**
 - representando o valor constante associado ao atributo.

ATRIBUTOS

ATRIBUTO ConstantValue

u2 constantvalue_index

- índice de uma estrutura **CONSTANT_<tipo>_info** do tipo apropriado para a variável estática associada.

Tipo campo	Tipo da estrutura
long	CONSTANT_Long
float	CONSTANT_Float
double	CONSTANT_Double
int, short, char, byte, boolean	CONSTANT_Integer
String	CONSTANT_String

ATRIBUTOS

ATRIBUTO CODE

Utilizado em estrutura `method_info`

- Atributo de tamanho variável
- Somente **um** atributo code é possível por método
 - **Método não nativo**
 - contém o código JVM e informações auxiliares para o método, método de iniciação de instância, classe ou interface.
 - **Método nativo ou abstrato**
 - Não possui esse atributo de código
- Toda JVM deve reconhecer o atributo Code

ATRIBUTOS

ATRIBUTO CODE

```
Code_attribute {  
  u2 attribute_name_index;  
  u4 attribute_length;  
  u2 max_stack;  
  u2 max_locals;  
  u4 code_length;  
  u1 code [code_length];  
  u2 exception_table_length;  
  {  
    u2 start_pc;  
    u2 end_pc;  
    u2 handler_pc;  
    u2 catch_type; } exception_table [exception_table_length];  
  u2 attributes_count;  
  attribute_info attributes[attributes_count];  
}
```

ATRIBUTOS

ATRIBUTO CODE

u2 attribute_name_index

- attribute_name_index é índice válido para a tabela constant_pool
 - **CONSTANT_Utf8_info** representando a string **"Code"**.

u4 attribute_length

- número de bytes desse atributo, exceto esses 6 bytes iniciais.

u2 max_stack

- **profundidade máxima da pilha de operandos durante a execução desse método**

u2 max_locals

- **número de variáveis locais (incluindo os parâmetros) do seu vetor de variáveis locais**

u4 code_length

- **número de bytes no seu array code** (deve ser maior que zero)

u1 code []

- **bytecodes da JVM que implementam o código desse método.**

ATRIBUTOS

ATRIBUTO CODE

u2 exception_table_length

- número de entradas na tabela exception_table

exception_table []

- cada entrada descreve um manipulador das exceções que podem ocorrer no código JVM contido no array code

u2 start_pc

u2 end_pc

- manipulador ativo para os índices [start_pc, end_pc) para code

u2 handler_pc

- índice para code indicando o bytecode inicial do manipulador

u2 catch_type

- Se **não nulo** (cláusula catch de um comando try)
 - índice válido para a tabela constant_pool
 - **CONSTANT_Class_info** representando a classe de exceções a ser capturada pelo manipulador (Throwable ou uma de suas subclasses)

Se **nulo** (cláusula **finally** de um comando try)

manipulador é chamado para todo tipo de exceção que pode ser lançado no comando try em questão.

ATRIBUTOS

ATRIBUTO CODE

u2 attributes_count

- Número de atributos associados ao atributo code de um método

attribute_info attributes [attributes_count]

- **Pode existir um número qualquer de atributos opcionais (debug) associados ao atributo code**
 - A JVM pode ignorá-los em silêncio
- Atributo lineNumberTable
 - **associa posições no array code com linhas do arquivo fonte**
- Atributo LocalVariableTable
 - **Utilizado por debuggers para determinar o valor de uma variável local durante a execução**
- **Atributos proprietários**
 - **Não podem afetar a semântica do arquivo .class**
 - Fornecem informação adicional de descrição do arquivo

ATRIBUTOS

ATRIBUTO CODE

Localização do manipulador de exceções pela JVM

- busca no método corrente
 - Se localizado, desvia a execução para handler_pc
- senão método corrente é terminado abruptamente
 - pilha de operandos e vetor de variáveis locais são descartados
 - frame é desempilhado e o controle é passado para o método chamador do método corrente
- exceção é relançada para o método chamador que se torna o corrente
 - esse processo continua até encontrar um manipulador ou o final da cadeia dos métodos chamadores
 - Se nenhum manipulador adequado for encontrado, a execução da thread onde a exceção foi lançada é terminada

ATRIBUTOS

ATRIBUTO DEPRECATED

Opcional, tamanho fixo: tabela attributes de estrutura

ClassFile, field_info ou method_info

- informa ao usuário que a classe, interface, campo ou método está superado
 - não altera a semântica da classe ou interface

```
Deprecated_attribute {  
    u2 attribute_name_index;  
    u4 attribute_length;  
}
```

- attribute_name_index é índice válido para a tabela constant_pool
 - CONSTANT_Utf8_info representando a string “Deprecated”.
- attribute_length valor fixo em zero.

ATRIBUTOS

ATRIBUTO EXCEPTIONS

Utilizado em estrutura `method_info`

- Atributo de tamanho variável
- No máximo, um atributo Exceptions por método
- Indica quais exceções verificadas o método pode lançar
- Formato

```
Exceptions_attribute {  
    u2 attribute_name_index;  
    u4 attribute_length;  
    u2 number_of_exceptions;  
    u2 exception_index_table[number_of_exceptions];  
}
```


ATRIBUTOS

ATRIBUTO EXCEPTIONS

u2 attribute_name_index

- attribute_name_index é índice válido para a tabela constant_pool
 - CONSTANT_Utf8_info representando a string “Exceptions”.

u4 attribute_length

- número de bytes desse atributo, exceto esses 6 bytes iniciais.

u2 number_of_exceptions

- Número de entradas na tabela exception_index_table

u2 exception_index_table []

- índice válido para a tabela constant_pool
 - CONSTANT_Class_info representando um tipo de classe de exceção que o método pode lançar
 - Por exemplo, classe ou subclasse de RuntimeException, Error ou Throwable

ATRIBUTOS

ATRIBUTO INNERCLASSES

Atributo de tamanho variável utilizado na tabela attributes de estrutura ClassFile

- Pool de constantes de uma classe ou interface refere a classe ou interface C que não é membro de um pacote
- sua estrutura ClassFile deve conter exatamente um atributo InnerClasses na sua tabela attributes
- Formato

```
InnerClasses_attribute {  
    u2 attribute_name_index;  
    u4 attribute_length;  
    u2 number_of_classes;  
    { u2 inner_class_info_index;  
      u2 outer_class_info_index;  
      u2 inner_name_index;  
      u2 inner_class_access_flags;  
    } classes [number_of_classes];  
}
```

ATRIBUTOS

ATRIBUTO INNERCLASSES

u2 attribute_name_index

- attribute_name_index é índice válido para a tabela constant_pool
 - CONSTANT_Utf8_info representando a string “InnerClasses”.

u4 attribute_length

- número de bytes desse atributo, exceto esses 6 bytes iniciais.

u2 number_of_classes

- Número de entradas na tabela classes

classes []

- um membro de uma classe ou interface aninhada terá o atributo InnerClasses para toda classe envolvente e para cada membro imediato.

ATRIBUTOS

ATRIBUTO INNERCLASSES

- Os itens do array classes são
 - u2 inner_class_info_index
 - índice válido para a tabela constant_pool
 - CONSTANT_Class_info representando a classe C. Os outros itens dão informação sobre a classe ou interface C.
 - u2 outer_class_info_index
 - Zero (se C não é um membro) ou índice válido da tabela constant_pool
 - CONSTANT_Class_info representando a classe ou interface da qual C é um membro.
 - u2 inner_name_index
 - Zero (se C is anônima) ou índice válido da tabela constant_pool
 - CONSTANT_Utf8_info representando nome simples original de C, como no fonte do qual esse arquivo foi compilado.
- u2 inner_class_access_flags
 - Máscara de bits especificando permissões de acesso e propriedades da classe ou interface C declaradas no fonte.

Atributos

Atributo InnerClasses

Nome do Flag	Valor	Interpretação
ACC_PUBLIC	0x0001	Marcada ou implicitamente public no fonte.
ACC_PRIVATE	0x0002	Marcada private no fonte.
ACC_PROTECTED	0x0004	Marcada protected no fonte.
ACC_STATIC	0x0008	Marcada ou implicitamente static no fonte.
ACC_FINAL	0x0010	Marcada final no fonte. Não pode ser estendida.
ACC_INTERFACE	0x0200	É uma interface no fonte.
ACC_ABSTRACT	0x0400	Marcada abstract no fonte. Possui apenas métodos abstratos. Não pode ser instanciada.

- Todos os demais bits são reservados para uso futuro

Devem ser definidos como zero.

ATRIBUTOS

ATRIBUTO LINENUMBERTABLE

Opcional, tamanho variável: tabela attributes do atributo Code

- permite a debugger determinar que posições no array code correspondem a uma dada linha do arquivo fonte
 - relação 1-1 entre um atributo LineNumberTable e um linha no fonte original não é requerida.
 - múltiplos atributos LineNumberTable juntos podem representar uma linha
- Formato

```
LineNumberTable_attribute {  
    u2 attribute_name_index;  
    u4 attribute_length;  
    u2 line_number_table_length;  
    {  
        u2 start_pc;  
        u2 line_number;  
    } line_number_table [line_number_table_length];  
}
```

ATRIBUTOS

ATRIBUTO LINENUMBERTABLE

u2 attribute_name_index

- attribute_name_index é índice válido para a tabela constant_pool
 - CONSTANT_Utf8_info representando a string “LineNumberTable”.

u4 attribute_length

- número de bytes desse atributo, exceto esses 6 bytes iniciais.

u2 line_number_table_length

- número de entradas no array line_number_table

line_number_table []

- u2 start_pc
 - índice para o array code correspondendo ao código que inicia uma nova linha no arquivo fonte original
- u2 line_number
 - número dessa linha no arquivo fonte

ATRIBUTOS

ATRIBUTO LOCALVARIABLETABLE

Opcional, tamanho variável: tabela attributes do atributo Code

- permite a debugger determinar o valor de uma dada variável local, durante a execução de um método
 - No máximo, um atributo LocalVariableTable por variável local em Code
- Formato

```
LocalVariableTable_attribute {  
    u2 attribute_name_index;  
    u4 attribute_length;  
    u2 local_variable_table_length;  
    {  
        u2 start_pc;  
        u2 length;  
        u2 name_index;  
        u2 descriptor_index;  
        u2 index;  
    } local_variable_table [local_variable_table_length];  
}
```


ATRIBUTOS

ATRIBUTO LOCALVARIABLETABLE

u2 attribute_name_index

- attribute_name_index é índice válido para a tabela constant_pool
 - CONSTANT_Utf8_info representando a string “LocalVariableTable”.

u4 attribute_length

- número de bytes desse atributo, exceto esses 6 bytes iniciais.

u2 local_variable_table_length

- número de entradas no array local_variable_table

local_variable_table []

- cada entrada em local_variable_table indica:
 - uma faixa de índices no array code na qual uma dada variável local mantém o mesmo valor.
 - o índice dessa variável local no array de variáveis locais do frame corrente no qual ela pode ser encontrada.

ATRIBUTOS

ATRIBUTO LOCALVARIABLETABLE

os itens do array `local_variable_table` são:

- u2 `start_pc` e u2 `length`
 - a variável local possui o mesmo valor no intervalo `[start_pc, start_pc+length]` de offsets de code
 - esses offsets correspondem a índices válidos em code que apontam para opcode da JVM
- u2 `name_index`
 - `name_index` é índice válido para a tabela `constant_pool`
 - `CONSTANT_Utf8_info` representando um nome válido de variável local armazenado como um `nome simples`
- u2 `descriptor_index`
 - `descriptor_index` é índice válido para a tabela `constant_pool`
 - `CONSTANT_Utf8_info` representando um `descriptor` de campo válido com o tipo da variável local no programa fonte
- u2 `index`

índice no array de variáveis locais do frame corrente correspondendo à variável local em questão

se do tipo `double` ou `long` a variável ocupa as posições `index` e `index+1`.

ATRIBUTOS

ATRIBUTO SOURCEFILE

Opcional, tamanho fixo: tabela attributes de estrutura ClassFile

- nome (relativo) do fonte a partir do qual a classe foi compilada.
 - apenas um atributo SourceFile pode aparecer por ClassFile.

Formato

```
SourceFile_attribute {  
    u2 attribute_name_index;  
    u4 attribute_length;  
    u2 sourcefile_index;  
}
```

- attribute_name_index é índice válido para a tabela constant_pool
 - CONSTANT_Utf8_info representando a string “SourceFile”.
- attribute_length assume sempre o valor 2
- sourcefile_index é índice válido para a tabela constant_pool
 - CONSTANT_Utf8_info representando uma string com o nome do fonte.

FOPEN

FILE* fopen(const char* filename, const char* mode);

"r" text reading

"w" text writing

"a" text append

"r+" text update (reading and writing)

"w+" text update, discarding previous content (if any)

"a+" text append, reading, and writing at end

"b" after the first character for binary files.

LEND0 .CLASS

Formato big-endian

```
static u2 u2Read(FILE *fd) {  
    u2 toReturn = getc(fd);  
    toReturn = (toReturn << 8) | (getc(fd));  
    return toReturn;  
}
```

ESTRUTURA CLASSFILE

```
ClassFile {  
    u4          magic;  
    u2          minor_version;  
    u2          major_version;  
    u2          constant_pool_count;  
    cp_info     constant_pool [constant_pool_count-1];  
    u2          access_flags;  
    u2          this_class;  
    u2          super_class;  
    u2          interfaces_count;  
    u2          interfaces [interfaces_count];  
    u2          fields_count;  
    field_info  fields [fields_count];  
    u2          methods_count;  
    method_info methods [methods_count];  
    u2          attributes_count;  
    attribute_info attributes [attributes_count];  
}
```

LEND0 .CLASS

```
ClassFile *cf = (ClassFile *) malloc(sizeof(Class));  
cf->magic = u4Read(fd);  
cf->minor_version = u2Read(fd);  
cf->major_version = u2Read(fd);  
cf->constant_pool_count = u2Read(fd);
```

POOL DE CONSTANTES

Cada entrada em `constant_pool` tem a forma

```
cp_info {  
    u1 tag;  
    u1 info[ ];  
}
```

O byte de tag define o tipo da informação em `cp_info`

POOL DE CONSTANTES

TIPOS VÁLIDOS DE TAGS

Tipo de Constante	Valor
CONSTANT_Class	7
CONSTANT_Fieldref	9
CONSTANT_Methodref	10
CONSTANT_InterfaceMethodref	11
CONSTANT_String	8
CONSTANT_Integer	3
CONSTANT_Float	4
CONSTANT_Long	5
CONSTANT_Double	6
CONSTANT_NameAndType	12
CONSTANT_Utf8	1

POOL DE CONSTANTES

EXEMPLOS DE ESTRUTURAS CONSTANT

```
CONSTANT_Class_info {  
    u1 tag;                                // valor 7  
    u2 name_index;  
}  
CONSTANT_Fieldref_info {  
    u1 tag;                                // valor 9  
    u2 class_index;  
    u2 name_and_type_index;  
}  
CONSTANT_NameAndType_info {  
    u1 tag;                                // valor 12  
    u2 name_index;  
    u2 descriptor_index;  
}
```

LEND0 CONSTANT_POOL

```
typedef struct {  
    u1 tag;  
    union {  
        struct {  
            u2 name_index;  
        } Class;  
        struct {  
            u2 class_index;  
            u2 name_and_type_index;  
        } Fieldref;  
        ...  
    };  
};
```

LEND0 CONSTANT_POOL

```
Constant *constantPool = (Constant*) malloc(...  
Constant *cp;  
for (cp = constantPool; cp < constantPool + count - 1; cp++) {  
    cp->tag = u1Read(cf);  
    switch (cp->tag) {  
        case CONSTANT_Class:  
            cp->u.Class.name_index = u2Read(fd);  
            break;  
        case CONSTANT_Fieldref:  
            cp->u.Fieldref.class_index = u2Read(fd);  
            cp->u.Fieldref.name_and_type_index = u2Read(fd);  
            break;
```