„Die Java – Klassendateien“

Seminararbeit von Studierach Beatrix, 0030213

LVA: Seminar aus Softwareentwicklung: Inside Java und .NET
WS 03/04

Inhaltsverzeichnis

Einleitung – Was eine Java – Klassendatei?

Eine Java-Klassendatei ist eine binäre Datei für Java Programme. Jede Klasse oder jedes Interface wird durch genau eine Klassendatei definiert. Durch das genau festgelegte Format kann die Klassendatei von jeder Java virtuellen Maschine geladen und interpretiert werden. Dadurch erreicht man eine gewisse Unabhängigkeit, da es einerlei ist, auf welchem System das Classfile erzeugt wurde.

Die EBNF der Klassendatei

Class File

<ClassFile> := <magic> <minor_version> <major_version> <constant_pool_count> <constant_pool> <c_access_flags> <this_class> <super_class> <interfaces_count> <interfaces> <fields_count> <fields> <methods_count> <methods> <attributes_count> <attributes>

<magic> : = 0xCAFEBABE

<major_version> := <u2>

<minor_version> := <u2>

<constant_pool_count> := 0x0001 | 0x0002 | 0x0003 |……| 0xFFFF

<constant_pool> := *empty* {cp_info}

<c_access_flags> := 0x0000 | 0x0001 | 0x0010 | 0x0011 | 0x0020 | 0x0021 | 0x0400 | 0x0401 | 0x0420 | 0x0421 | 0x0200 | 0x0201 | 0x0220 | 0x0221

<this_class> := 0x0001 | 0x0002 | 0x0003 |……| 0xFFFF

<super_class> := <u2>

<interfaces_count> := <u2>

<interfaces> := {<u2>}

<fields_count> := <u2>

<fields> := {field_info}

<methods_count> := <u2>

<methods> := {method_info}

<attributes_count> := <u2>

<attributes> := {attribute_info}

Constant Pool

<cp_info> := <CONSTANT_Class_info> | <CONSTANT_Fieldref_info> |
<CONSTANT_Methodref_info> |
<CONSTANT_InterfaceMethodref_info> | <CONSTANT_String_info> |
<CONSTANT_Integer_info> | <CONSTANT_Float_info> |
<CONSTANT_Long_info> | <CONSTANT_Double_info> |
<CONSTANT_NameAndType_info> | <CONSTANT_Utf8_info>

<CONSTANT_Class_info> := <CONSTANT_Class > <name_index>

<CONSTANT_Fieldref_info> := <CONSTANT_Fieldref> <class_index>
<name_and_type_index>

<CONSTANT_Methodref_info> := <CONSTANT_Methodref> <class_index>
<name_and_type_index>

<CONSTANT_InterfaceMethodref_info> := <CONSTANT_InterfaceMethodref>
<class_index>
<name_and_type_index>

<CONSTANT_String_info> := <CONSTANT_String> <string_index>

<CONSTANT_Integer_info> := <CONSTANT_Integer> <bytes>

<CONSTANT_Float_info> := <CONSTANT_Float> <bytes>

<CONSTANT_Long_info> := <CONSTANT_Long> <high_bytes> <low_bytes>

<CONSTANT_Double_info> := <CONSTANT_Double> <high_bytes> <low_bytes>

<CONSTANT_NameAndType_info> := <CONSTANT_NameAndType>
<name_index> <descriptor_index>

<CONSTANT_Utf8_info> := <CONSTANT_Utf8> <length> {<bytes>}

<CONSTANT_Class> := 0x0007
<CONSTANT_Fieldref> := 0x0009
<CONSTANT_Methodref> := 0x000A
<CONSTANT_InterfaceMethodref> := 0x000B
<CONSTANT_String> := 0x0008
<CONSTANT_Integer> := 0x0003
<CONSTANT_Float> := 0x0004
<CONSTANT_Long> := 0x0005
<CONSTANT_Double> := 0x0006
<CONSTANT_NameAndType> := 0x000C

<CONSTANT_Utf8> := 0x0001

Fields

<field_info> := <f_access_flags> <name_index> <descriptor_index>
<attributes_count> <attributes>

<f_access_flags> := 0x0000 | 0x0001 | 0x0002 | 0x0004 | 0x0008 | 0x0009 | 0x000A | 0x000C | 0x0010 | 0x0011 | 0x0012 | 0x0014 | 0x0018 | 0x0019 | 0x001A | 0x001C | 0x0040 | 0x0041 | 0x0042 | 0x0044 | 0x0048 |0x0049 | 0x004A | 0x004C | 0x0080 | 0x0081 | 0x0082 | 0x0084 | 0x0088 | 0x0089 | 0x008A | 0x008C | 0x0090 | 0x0091 | 0x0092 | 0x0094 | 0x0098 | 0x0099 | 0x009A | 0x009C | 0x00C0 | 0x00C1 | 0x00C2 | 0x00C4 | 0x00C8 | 0x00C9 | 0x00CA | 0x00CC

Methods

<method_info> := <m_access_flags> <name_index> <descriptor_index>
<attributes_count> <attributes>

<m_access_flags> := 0x0000 | 0x0001 | 0x0002 | 0x0004 | 0x0008 | 0x0009 | 0x000C | 0x0010 | 0x0011 | 0x0012 | 0x0014 | 0x0018 | 0x0019 | 0x001C | 0x0020 | 0x0021 | 0x0022 | 0x0024 | 0x0028| 0x0029 | 0x002C | 0x0030 | 0x0031 | 0x0032 | 0x0034 | 0x0038 | 0x0039 | 0x003C | 0x0100 | 0x0101 | 0x0102 | 0x0104 | 0x0108 | 0x0109 | 0x010C | 0x0110 | 0x0111 | 0x0112 | 0x0114 | 0x0118 | 0x0119 | 0x011C | 0x0120 | 0x0121 | 0x0122 | 0x0124 | 0x0128 | 0x0129 | 0x012C | 0x0130 | 0x0131 | 0x0132 | 0x0134 | 0x0138 | 0x0139 | 0x013C | 0x0400 | 0x0401 | 0x0404

Attributes

<attribute_info> := <SourceFile_attribute> | <ConstantValue_attribute> |
<Code_attribute> | <Exceptions_attribute> |
<LineNumberTable_Attribute> | <LocalVariableTable_Attribute>

<SourceFile_attribute> := <attribute_name_index> <attribute_length>
<sourcefile_index>

<ConstantValue_attribute>:= <attribute_name_index> <attribute_length>
<constantvalue_index>

<Code_attribute> := <attribute_name_index> <attribute_length> <max_stack>
<max_locals> <code_length> {code} <exception_table_length>
{<exception_table>} <attributes_count> {<attributes>}

<Exceptions_attribute> := <attribute_name_index> <attribute_length>
<number_of_exceptions> {<exception_index_table>}

<LineNumberTable_Attribute> := <attribute_name_index> <attribute_length>
<line_number_table_length> {<line_number_table>}

<LocalVariableTable_Attribute> := <attribute_name_index> <attribute_length>
<local_variable_table_length> {<local_variable_table>}

<attribute_length> := <u4>

<max_stack> := <u2>
<max_locals> := <u2>
<code_length> := <u2>
<code> := <u1>
<exception_table_length> := <u2>
<exception_table> := <pc_start> <end_pc> <handler_pc> <catch_type>
<start_pc> := <u2>
<end_pc> := <u2>
<handler_pc> := <u2>
<catch_type> := <u2>

<number_of_exceptions> := <u2>
<exception_index_table> := <u2>

<line_number_table_length> := <u2>
<line_number_table> := <start_pc> <line_number>
<line_number> := <u2>

<local_variable_table_lenght> := <u2>
<local_variable_table> := <start_pc> <length> <name_index> <descriptor_index>
<index>
<length> := <u2>
<index> := <u2>

Indices

(Verweisen auf gültige Einträge im Constant Pool)

<class_index> := <u2>
<name_and_type_index> := <u2>
<string_index> := <u2>
<name_index> := <u2>
<descriptor_index> := <u2>
<attribute_name_index> := <u2>
<constantvalue_index> := <u2>
<sourcefile_index> := <u2>

Bytes

<bytes> := <u4>
<high_bytes> := <u4>
<low_bytes> := <u4>

<u1> := 0x00 | 0x01 | 0x02 |….| 0xFF
<u2> := 0x0000 | 0x0001 | 0x0002 |….| 0xFFFF
<u4> := 0x0000 0000 | 0x0000 0001 | 0x0000 0002 |….| 0xFFFF FFFF
<u8> := 0x0000 0000 0000 0000 | 0x0000 0000 0000 0001 |
0x0000 0000 0000 0002 |….| 0xFFFF FFFF FFFF FFFF

Strings

Fully Qualified Names

Im Klassendatei – Format treten Klassennamen immer als “Fully Qualified Names” auf. Klassennamen werden immer als CONSTANT_Utf8_info Strukturen dargestellt, die von einer CONSTANT_Class_info Struktur referenziert werden. Auch von einer CONSTANT_NameAndType_info Struktur kann es eine Referenz darauf geben, wenn der Klassenname im Deskriptor **(siehe)** vorkommt.

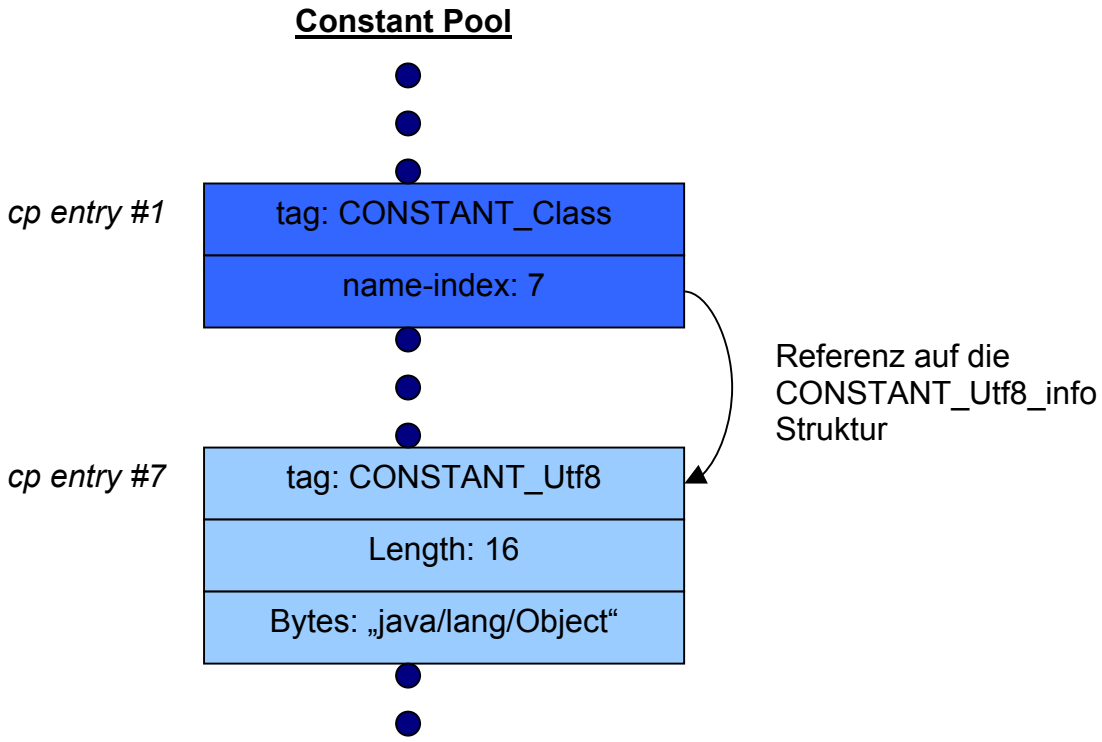


Abb.
Normalerweise ist man gewöhnt „Fully Qualified Names“ mit Trrennnung durch Punkte („java.lang.Object“) anzuschreiben. Die interne Darstellung in der Klassen-datei erfolgt jedoch mit Schrägstrichen („java/lang/Object“).

Einfache Namen (Simple Names)

Die Namen von Feldern und Methoden werden als einfache Namen dargestellt.

Beispiel:

- Methode: String toString() → „toString“
- Feld: int number → „number”

Deskriptoren (Descriptors)

Ein Deskriptor beschreibt die Parameter einer Methode oder den Typ eines Feldes. Die Deskriptoren werden nach folgender Grammatik gebildet.

Felddeskriptoren (Field Descriptors)

<FieldDescriptor> := <FieldType>
<ComponentType> := <FieldType>
<FieldType> := <BaseType> | <ObjectType> | <ArrayType>
<BaseType> := B | C | D | F | I | J | S | Z
<ObjectType> := L <classname> ;
<ArrayType > := [ComponentType

- **<classname>** repräsentiert einen Klassennamen in Form eines “fully Qualified Names“.
- **L** steht für eine Instanz einer Klasse.
- **[** steht für eine Dimension des Arrays.

Die Bedeutung der Basistypen

- B byte
- C char
- D double
- F float
- I integer
- J long
- S short
- Z boolean

Methodendeskriptoren (Method Descriptors)

<MethodDesciptor> := ({<ParameterDescriptor> }) ReturnDescriptor
<ParameterDescriptor> := <FieldType>
<ReturnDescriptor> := <FieldType> | V

- **<ParameterDescriptor>** beschreibt die Typen der Eingangsparameter
- **<ReturnDescriptor>** beschreibt den Typ des Rückgabenwerts
- **V** steht für den Rückgabewert void

Beispiele für Deskriptoren

Felder:

- int number → I
- double [ ][ ] matrix → [[D
- ArrayList myList → L java/util/ArrayList;
- Date [] myDates → [L java/util/Date;

Methoden:

- int getNumber() → () I
- String toString() → () L java/lang/String;
- Date getDates() → () [L java/util/Date;
- void main (String[] args) → ([L java/lang/String;) V
- boolean saveData (boolean male, int number, String name, int age, float height)
  → (Z I L java/lang/String; I F) Z

Das Klassendateiformat

ClassFile {

u4	magic;	**DATEIKOPF**
u2	minor_version;	
u2	major_version;	
u2	constant_pool_count;	**KONSTANTENPOOL**
cp_info	constant_pool[constant_pool_count-1];	
u2	access_flags	**KLASSENBESCHREIBUNG**
u2	this_class;	
u2	super_class;	
u2	interfaces_count;	
u2	interfaces [interfaces_count];	
u2	fields_count;	**ARRAY VON DATENFELDERN**
field_info	fields [fields_count];	
u2	methods_count;	**ARRAY VON METHODEN**
method_info	methods [methods_count];	
u2	attributes_count;	**KLASSENATTRIBUTE**
attribute_info	attributes [attributes_count];	

}

Der Dateikopf

Im Dateikopf befindet sich die so genannte “magic number”. Diese kennzeichnet das Klassendateiformat. Beginnt die Datei nicht mit 0xCAFEBABE, weiß man sicher, dass es sich dabei nicht um eine Java – Klassendatei handelt.

Weiters findet man dort auch die Nummern minor_version und die major_version. Da sich die Java- Technologie immer weiter entwickelt, verändert sich auch das Klassendateiformat. Die Versionsnummern geben an, zu welchem Format eine Klassendatei gehört. JVM können Klassendateien mit einer bestimmten major-Versionsnummer und einer Reihen von minor- Versionsnummern laden. Andere Klassendateien werden zurückgewiesen.

Der Konstantpool

Einer der wichtigsten Bestandteile ist der Constant Pool. Der Konstatenpool ist ein heterogenes Array, das elf verschiedene Elemente enthalten kann, die durch einen Kenner („tags“) voneinander unterschieden werden können.
Der Wert von constant_pool_count gibt an, wie viele Einträge es im Konstantenpool gibt und muss größer als 0 sein.
Der erste Eintrag im Konstantenpool (constant_pool[0]) ist für den internen Gebrauch der JVM reserviert. Der „richtige“ erste Eintrag im Konstantpool startet beim Index 1.

ConstantPool – Tags

Typ des Konstantenpooleintrags	Wert
CONSTANT_Utf8	1
CONSTANT_Integer	3
CONSTANT_Float	4
CONSTANT_Long	5
CONSTANT_Double	6
CONSTANT_Class	7
CONSTANT_String	8
CONSTANT_Fieldref	9
CONSTANT_Methodref	10
CONSTANT_InterfaceMethodref	11
CONSTANT_NameAndType	12

CONSTANT_Class

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

Der ‚tag’ ist vom Typ CONSTANT_Class und hat den Wert 7.
Der Wert von name_index muss ein gültiger Eintrag vom Typ CONSTANT_Utf8_info im Konstantpool sein, der einen gültigen Java-Klassenamen als fully qualified name repräsentiert. Ein Beispiel dafür ist in Abb. dargestellt.

CONSTANT_Fieldref, CONSTANT_Methodref, CONSTANT_InterfaceMethodref

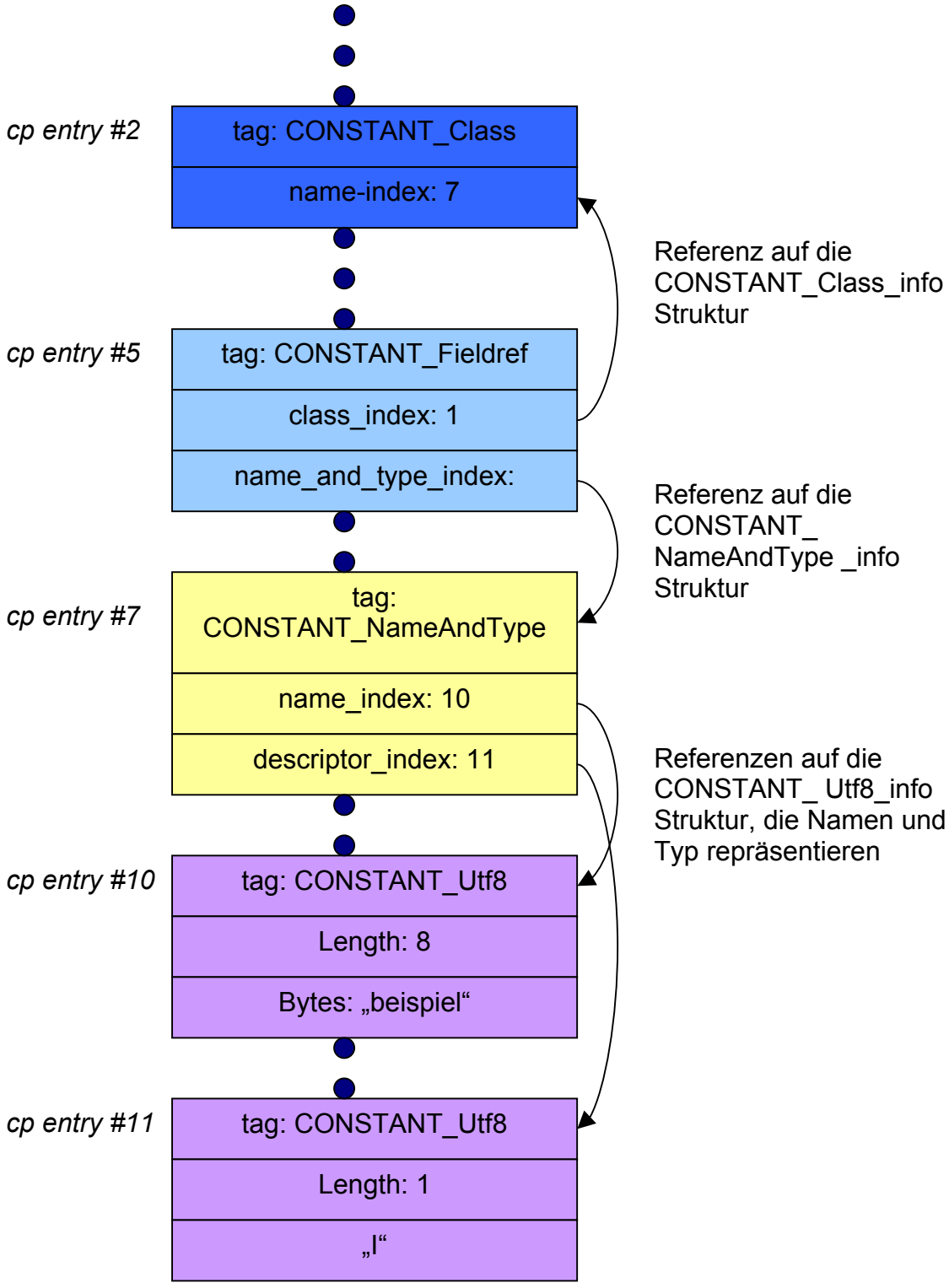
```
CONSTANT_Fieldref_info {
        u1      tag;
        u2      class_index;
        u2      name_and_type_index;
}

CONSTANT_Methodref_info {
        u1      tag;
        u2      class_index;
        u2      name_and_type_index;
}

CONSTANT_InterfaceMethodref_info {
        u1      tag;
        u2      class_index;
        u2      name_and_type_index;
}
```

<table>
<tr><td></td><td>CONSTANT_Fieldref</td><td>CONSTANT_Methodref</td><td>CONSTANT_
InterfaceMethodref</td></tr>
<tr><td>tag</td><td>9</td><td>10</td><td>11</td></tr>
<tr><td rowspan="2">class_index</td><td colspan="3">Der Wert von class_index muss ein gültiger Eintrag vom Typ CONSTANT_Class_info im Konstantenpool sein, der</td></tr>
<tr><td colspan="2">eine Klasse repräsentiert.</td><td>ein Interface repräsentiert</td></tr>
<tr><td rowspan="2">name_and_type_index</td><td colspan="3">Der Wert von name_and_type_index muss ein gültiger Eintrag vom Typ CONSTANT_NameAndType_info im Konstantenpool sein, der Namen und</td></tr>
<tr><td>den Deskriptor des Feldes repräsentiert.</td><td colspan="2">den Deskriptor der Methode repräsentiert.</td></tr>
</table>

Constant Pool – Feld „int beispiel;“



CONSTANT_String

```
CONSTANT_String_info{
        U1      tag;
        U2      string_index;
}
```

Der ‚tag' ist vom Typ CONSTANT_String und hat den Wert 8.
Der Wert von string_index muss ein gültiger Eintrag vom Typ CONSTANT_Utf8_info im Konstantpool sein, der eine Zeichenkette repräsentiert.

CONSTANT_Integer, CONSTANT_Float

	CONSTANT_Integer	CONSTANT_Float

Struktur	CONSTANT_Integer_info {
	u1 tag;
	u4 bytes;
	}
		CONSTANT_Float_info {
		u1 tag;
		u4 bytes;
		}

tag	3	4
bytes	enthält den Integer Wert gespeichert in big-endian Darstellung.	enthält den Float Wert dargestellt im IEEE 754 floating point format.

Ergänzungen zum IEEE754- Format:

Spezialfälle:

0x7F80 0000 → Double- Wert ist ∞
0xFF80 0000 → Double- Wert ist -∞
0x7F80 0001 – 0x7FFF FFFF → Double- Wert ist NaN
0xFF80 0001 – 0xFFFF FFFF → Double-Wert ist NaN

Berechnung der Variablen des Floating Point Werts $s*m*2^{e-150}$:

```
int s = ((bytes >> 31) == 0) ? 1 : -1;
int e = (int) ((bytes >> 23) & 0xFF);
long m = (e== 0) ?
        (bytes & 0x7FFF FFFF) <<1 :
        (bytes & 0x7FFF FFFF) | 0x8000 0000;
```

CONSTANT_Long, CONSTANT_Double

	CONSTANT_Long	CONSTANT_Double
Struktur	CONSTANT_Long_info { u1 tag;	CONSTANT_Double_info { u1 tag;

	u4 high_bytes; u4 low_bytes; }	u4 high_bytes; u4 low_bytes; }
tag	5	6
high_bytes, low_bytes	enthalten zusammen den Long Wert jeweils gespeichert in big-endian Dartstellung. ((long) hight_bytes << 32) +low_bytes	Diese Struktur enthält den Double Wert im IEEE 754 floating point format. Die high_bytes und low_bytes werden zuerst in einen Long-Wert konvertiert.

Ergänzungen zum IEEE754- Format:

Spezialfälle:

0x7F80 0000 0000 0000L → Double- Wert ist ∞
0xFF80 0000 0000 0000L → Double- Wert ist -∞
0x7FF0 0000 0000 0001L – 0x7FFF FFFF FFFF FFFFL → Double- Wert ist NaN
0xFFF0 0000 0000 0001L – 0xFFFF FFFF FFFF FFFFL → Double-Wert ist NaN

*Berechnung der Variablen des Floating Point Werts s*m**2^{e-1075}:*

```
int s = ((bits >> 63) == 0) ? 1 : -1;
int e = (int) ((bits >> 52) & 0x7FFL);
long m = (e== 0) ?
        (bits & 0xFFFF FFFF FFFF FFFFL) <<1 :
        (bits & 0xFFFF FFFF FFFF FFFFL) | 0x1000 0000 0000 0000L;
```

CONSTANT_NameAndType

```
CONSTANT_NameAndType_info {
    u1    tag;
    u2    name_index;
    u2    desciptor_index;
}
```

Der ‚tag' ist vom Typ CONSTANT_NameAndType und hat den Wert 12.
Der Werte von name_index und descriptor_index müssen gültige Einträge vom Typ CONSTANT_Utf8_info im Konstantpool sein, wobei die Referenz von name_index einen Namen eines Feldes oder einer Methode repräsentiert und die Referenz von descriptor_index einen Felddeskriptor oder einen Methodendeskriptor.

CONSTANT_Utf8

```
CONSTANT_Utf8_info{
    u1    tag;
    u2    length;
    u1    bytes[lenght];
```

}

Der ‚tag' ist vom Typ CONSTANT_Utf8 und hat den Wert 1.
Die Anzahl der bytes im bytes-Array, welches die bytes des Strings enthält, ist gegeben durch length. Folgende Werte sind im byte- Array nicht erlaubt: (byte) 0 und (byte) (0xf0) bis (byte) 0xFF

Klassenbeschreibung

Die Merkmale von Klassen sind in den so genannten access_flags gespeichert. In der nachfolgenden Tabelle sehen sich zusammengefasst alle access_flags, ihren Wert, ihre Bedeutung und wo sie zu finden sind, da einige beispielsweise nur bei Methoden oder Variablen Sinn machen. Diese werden in Feld- und Methodenstrukturen verwendet, die etwas in den nächsten Kapiteln besprochen werden. Einige access_flags schließen einander aus oder können nur gemeinsam verwendet werden. Beispielsweise kann eine Methode nicht gleichzeitig abstrakt und final sein, da dies keinen Sinn ergeben würden, das heisst es kann nicht gleichzeitig das Flag „ACC_ABSTRACT" und „ ACC_FINAL" gesetzt sein. Andererseits ist jedes Feld eines Interfaces implizit statisch, final und public, das heisst es müssen alle Flags „ACC_STATIC", „ACC_FINAL" und „ACC_PUBLIC" gesetzt sein.

Name	Wert	Bedeutung	Benutzt vo
CLASSFILE			
ACC_PUBLIC	0x0001	sichtbar für alle	Klasse, Interface
ACC_FINAL	0x0010	keine Vererbung erlaubt	Klasse
ACC_SUPER	0x0020	behandelt Methoden von Superklassen speziell in „invokespecial"	Klasse, Interface
ACC_INTERFACE	0x0200	kennzeichnet ein Interface	Interface
ACC_ABSTRACT	0x0400	kann nich instantiiert werden	Klasse, Interface
FELDER			
ACC_PUBLIC	0x0001	sichtbar für alle	Jedes Feld
ACC_PRIVATE	0x0002	nur für eigene Klasse sichtbar	Klassenfeld
ACC_PROTECTED	0x0004	nur für Subklassen sichtbar	Klassenfeld
ACC_STATIC	0x0008	Klassenfeld	Jedes Feld
ACC_FINAL	0x0010	Kein Überschreiben oder Zuweisen nach Initialisierung	Jedes Feld
ACC_VOLATILE	0x0040	Kann nicht ge-„cached" werden	Klassenfeld
ACC_TRANSIENT	0x0080	Kann von einem persistente Objekmanager nicht geschrieben oder gelesen werden	Klassenfeld
METHODEN			
ACC_PUBLIC	0x0001	sichtbar für alle	Jede Methode

ACC_PRIVATE	0x0002	nur für eigene Klasse sichtbar	Klassen-/Instanz-methode
ACC_PROTECTED	0x0004	nur für Subklassen sichtbar	Klassen-/Instanz-methode
ACC_STATIC	0x0008	Klassenmethode	Klassen-/Instanz-methode
ACC_FINAL	0x0010	Kein Überschreiben	Klassen-/Instanz-methode
ACC_SYNCHRONIZED	0x0020	Synchronisierte Methode	Klassen-/Instanz-methode
ACC_NATIVE	0x0100	Nicht in Java implementiert	Klassen-/Instanz-methode
ACC_ABSTRACT	0x0400	Keine Implementation der Methode	Jede Methode

Die beiden Einträge this_class und super_class bilden jeweils einen Index auf einen CONSTANT_Class Eintrag im Konstantenpool und bestimmen so die eigene und die Superklasse. Ist die Superklasse Null, so handelt es sich dabei um die Klasse „java.lang.Object“ die Oberklasse aller Klassen.
Wie bei allen Arrays im Klassendatei- Format wird auch dem Interface-Array die Anzahl der Elemente im Array –in diesem Fall die Anzahl der Interfaces - vorangestellt. Das Interface-Array enhält Indizes, die Einträgen im Konstantenpool entsprechen. An den angegebenen Stellen befinden sich CONSTANT_Class Einträge, die alle Interfaces repräsentieren.

Array von Datenfeldern und Array von Methoden

„fields_count“ gibt die Anzahl der Felder an, die in der Klasse vorkommen. Das darauf folgende Array enthält Informationen über die Felder. Diese Informationen werden folgender Weise repräsentiert.

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

„methods_count“ gibt die Anzahl der Methoden an, die in der Klasse auftreten. Das darauf folgende Array enthält Informationen über diese Methoden. Diese Informationen werden folgender Weise repräsentiert.

```
method_info {
        u2              access_flags;
        u2              name_index;
        u2              desciptor_index;
        u2              attributes_count;
```

```
    attribute_info    attributes [attributes_count];
}
```

Die access_flags wurden bereits im vorigen Kapitel besprochen. Die Indices name_index und descriptor_index zeigen beide jeweils auf einen CONSTANT_Utf8 Eintrag in Konstantenpool, wobei der eine den Feldnamen und der andere den Feldtyp spezifiziert. Bei den Methoden wird der Methodenname und die Eingangs- beziehungsweise die Rückgabeparameter gespeichert.
Attributes_count gibt die Anzahl der Attribute an. In attribute-Array werden die Attribute gespeichert. Das einzige Attribut das in dem Zusammenhang mit Feldern erlaubt ist, ist das ConstantValue- Attribute. Bei den Methoden sind nur das Code-Attribut und das Exceptions- Attribut möglich.Auf Attribute wird im nächsten Kapitel noch genauer eingegangen.

Attribute

Mit Attributen werden Klassen, Methoden und Felder näher beschrieben. Die Attribute werden durch einen Index in den Konstantenpool und dem dort vorliegenden CONSTANT_Utf8 Eintrag identifiziert. Dieser muss dazu den Namen des Attributs selbst enthaltne. Das heißt es muss die Zeichenkette "ConstantValue" für das ConstantValue-Attribut gefunden werden. Für Klassen, Methoden und Felder gibt es eine unterschiedliche Anzahl von möglichen Attributen. Es gibt auch Attribute die wiederum selbst nur in Attributen auftreten dürfen.

Klassenattribute

Für Klassen gibt es nur ein Attribut das SourceFile-Attribut. Es gibt den Namen der class-Datei an.

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

Der attribut_name_index zeigt auf einen CONSTANT_Utf8 Eintrag im Constant Pool, der die Zeichenkette „SourcFile“ repräsentiert. Attribut_length gibt –wie der Name bereits sagt – die Länge des Attributes an und ist in diesem Fall 2. Auch der sourcefile_index zeigt auf einen CONSTANT_Utf8 Eintrag im Constant Pool, der den Namen der Quelldatei, von der die Klassendatei stammt enthält.

Feldattribute

Auch für Felder gibt es nur ein Attribut, das ConstantValue-Attribut. Es gibt einen Initialisierungswert für das Feld an.

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

Der attribut_name_index zeigt auf einen CONSTANT_Utf8 Eintrag im Constant Pool, der die Zeichenkette „ConstantValue“ repräsentiert. Die Attributlänge ist hier 2. Der constantvalue_index zeigt auf den Konstantenpool, je nach Feldtyp auf einen CONSTANT_Integer, CONSTANT_Long, CONSTANT_Float, CONSTANT_Double oder CONSTANT_String.

Methodenattribute

Für Methoden gibt es zwei Attribute. Das Code-Attribut enthält z.B. den eigentlichen Bytecode einer Methode und Informationen für die Ausnahmebehandlung. Das zweite Attribut ist das Exceptions- Attribut, das ein Liste alles Ausnahmen enthält die von der entsprechenden Methode geworfen werden können.

```
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]
}
```

Der attribut_name_index zeigt auf einen CONSTANT_Utf8 Eintrag im Constant Pool, der die Zeichenkette „Code“ repräsentiert. Die Attributlänge gibt die Länge des Attributes an ohne die sechs byte am Anfang zu berücksichtigen.
max_stack gibt die maximale Anzahl der Wörter am Operandenstack an, die für diese Methode benötigt werden.
max_locals gibt die Anzahl der lokalen Variablen an. Dabei werden auch Eingangsparameter inkludiert.
Der Eintrag code_length gibt die Anzahl der Bytes des Code Arrays an. Und muss größer als 0 sein. Das Code-Array enthält den Code für eine Methode in Bytesdarstellung. Das Code darf nicht leer sein.
Exception_table_length gibt die Anzahl der Einträge in der exception_table an. Die Einträge in der exception_table enthalten Informationen über jede in dieser Methode behandelte Ausnahme.

start_pc und end_pc kennzeichnen den Bereich wo der ExceptionHandler aktiv ist.
handler_pc kennzeichnet die Stelle im Code, ab welcher der Code des Handlers selbst steht.
attribute_count gibt die Anzahl der Attribute an, die im nachfolgenden Array enthalten sind. Das Attribute- Array kann zwei Arten von Attributen enthalten, das LineNumberTable- und das LocalTable-Attribute.

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

Der attribut_name_index zeigt auf einen CONSTANT_Utf8 Eintrag im Constant Pool, der die Zeichenkette „Exceptions“ repräsentiert. Die Attributlänge gibt die Länge des Attributes an ohne die sechs byte am Anfang zu berücksichtigen. number_of_exceptions gibt die Anzahl der Exceptions an, die im nachfolgenden Array gespeichert sind. In der exception_index-table muss jeder Wert der ungleich null ist ein gültiger Eintrag im Constant Pool sein, der den Typ CONSTANT_Class hat und die Exception- Klasse repräsentiert.

Sonstige Attribute

Das LineNumberTable- Attribut und das LocalVariableTable- Attribut sind optionale Attribut mit variabler Länge, die nur in der Attributliste des Codeattributes auftreten dürfen. Das LineNumberTable- Attribut dient dem Debugger, um zu wissen, welcher Teil des Codearrays der JVM zu welcher Zeile in der Quelldatei gehört. Das LocalVariableTable- Attribut wird auch vom Debugger benutzt, um herauszufinden, welchen Wert eine lokale Variable während der Ausführung einer Methode hat.

```
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_langth];
}
```

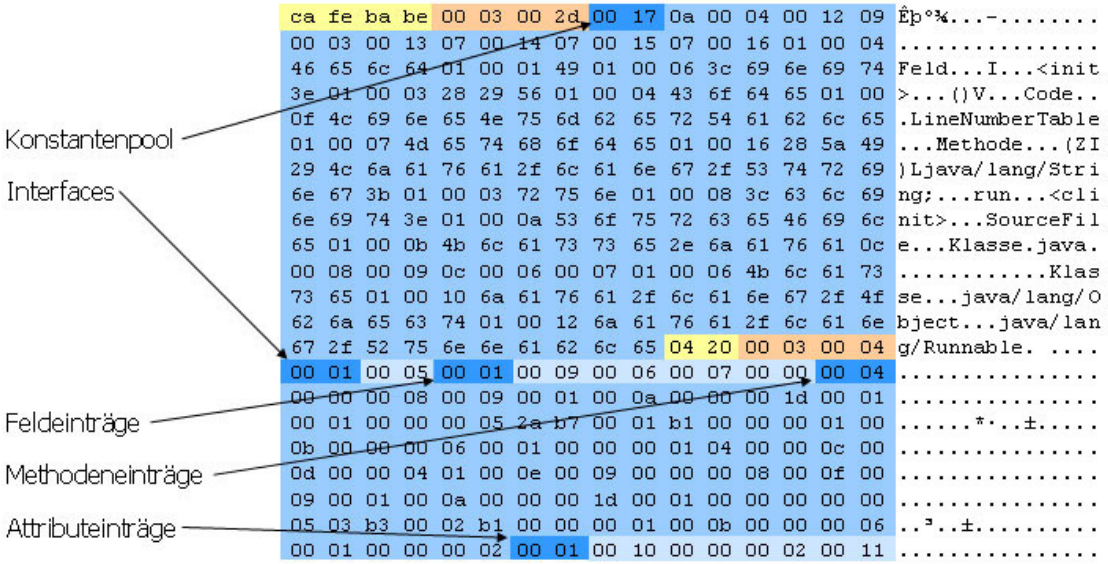
```
LocalVariableTable_attribute{
        u2      attribute_name_index;
        u4      attribute_length;
        u2      local_variable_table_length;
        {
                u2      start_pc;
                u2      length;
```

```
        u2      name_index;
        u2      descriptor_index;
        u2      index;
    }local_variable_tabel[local_variable_table_length];
}
```

Zur genaueren Erklärung der Struktur der beiden Attribute wird auf die angegebene Literatur verwiesen.

Ein Beispiel für eine Klassendatei

```
abstract class Klasse implements Runnable {
    public static int Feld = 0;
    abstract String Methode(boolean b, int i);
}
```




```
ca fe ba be 00 03 00 2d 00 17 0a 00 04 00 12 09  Êþº¾...-........
00 03 00 13 07 00 14 07 00 15 07 00 16 01 00 04  ................
46 65 6c 64 01 00 01 49 01 00 06 3c 69 6e 69 74  Feld...I...<init
3e 01 00 03 28 29 56 01 00 04 43 6f 64 65 01 00  >...()V...Code..
0f 4c 69 6e 65 4e 75 6d 62 65 72 54 61 62 6c 65  .LineNumberTable
01 00 07 4d 65 74 68 6f 64 65 01 00 16 28 5a 49  ...Methode...(ZI
29 4c 6a 61 76 61 2f 6c 61 6e 67 2f 53 74 72 69  )Ljava/lang/Stri
6e 67 3b 01 00 03 72 75 6e 01 00 08 3c 63 6c 69  ng;...run...<cli
6e 69 74 3e 01 00 0a 53 6f 75 72 63 65 46 69 6c  nit>...SourceFil
65 01 00 0b 4b 6c 61 73 73 65 2e 6a 61 76 61 0c  e...Klasse.java.
00 08 00 09 0c 00 06 00 07 01 00 06 4b 6c 61 73  ............Klas
73 65 01 00 10 6a 61 76 61 2f 6c 61 6e 67 2f 4f  se...java/lang/O
62 6a 65 63 74 01 00 12 6a 61 76 61 2f 6c 61 6e  bject...java/lan
67 2f 52 75 6e 6e 61 62 6c 65 04 20 00 03 00 04  g/Runnable. ....
00 01 00 05 00 01 00 09 00 06 00 07 00 00 00 04  ................
00 00 00 08 00 09 00 01 00 0a 00 00 00 1d 00 01  ................
00 01 00 00 00 05 2a b7 00 01 b1 00 00 00 01 00  ......*·..±.....
0b 00 00 00 06 00 01 00 00 00 01 04 00 00 0c 00  ................
0d 00 00 04 01 00 0e 00 09 00 00 00 08 00 0f 00  ................
09 00 01 00 0a 00 00 00 1d 00 01 00 00 00 00 00  ................
05 03 b3 00 02 b1 00 00 00 01 00 0b 00 00 00 06  ..³..±..........
00 01 00 00 00 02 00 01 00 10 00 00 00 02 00 11  ................
```

Eine Beispielklasse mit ihrer kompletten class-Datei in hexadezimaler und ASCII Darstellung.Die farbliche Markierung zeigt bereits die Grobstruktur des class-Dateiformats

Eine komplette Auflösung aller Felder und ihrer Bedeutung liefert folgende Tabelle. Die farbliche Kodierung stimmt mit der im Bild oben überein. Die erste Spalte enthält die groben Abschnitte innerhalb der class-Datei wie sie auch durch die Pfeile im Bild angegeben sind. Die zweite und dritte Spalte enthalten die Datenbytes und ihre grundsätzliche Bedeutung. In der letzten Spalte werden alle hexadezimalen Zahlen in dezimale umgerechnet und die Verweise etwaiger Konstantenpooleinträge aufgelöst. D.h. ist in der Zeile ein Index in den Konstantenpool wird die Zeichenkette auf die gezeigt wird bereits in der letzten Spalte angegeben.

Abschnitt	Bytes	Bedeutung	Weiterführende Bedeutung
	ca fe ba be	Erkennungszahl	
	00 03	Subversionsnummer	(3)

	00 2d	Versionsnummer	(45)
	00 17	Anzahl Konstanten	(23)
Konstantenpool			
1.	0a 00 04 00 12	CONSTANT_Methodref	(4,18)
2.	09 00 03 00 13	CONSTANT Fieldref	(3,19)
3.	07 00 14	CONSTANT_Class	(20)
4.	07 00 15	CONSTANT_Class	(21)
5.	07 00 16	CONSTANT_Class	(22)
6.	01 00 04 46 65 6c 64	CONSTANT_Utf8	(4) "Feld"
7.	01 00 01 49	CONSTANT_Utf8	(1) "I"
8.	01 00 06 3c 69 6e 69 74 3e	CONSTANT_Utf8	(6) "<init>"
9.	01 00 03 28 29 56	CONSTANT_Utf8	(3) "()V"
10.	01 00 04 43 6f 64 65	CONSTANT_Utf8	(4) "Code"
11.	01 00 0f 4c 69 6e 65 4e 75 6d 62 65 72 54 61 62 6c 65	CONSTANT_Utf8	(15) "LineNumberTable"
12.	01 00 07 4d 65 74 68 6f 64 65	CONSTANT_Utf8	(7) "Methode"
13.	01 00 16 28 5a 49 29 4c 6a 61 76 61 2f 6c 61 6e 67 2f 53 74 72 69 6e 67 3b	CONSTANT_Utf8	(22) "(ZI)Ljava/lang/String;"
14.	01 00 03 72 75 6e	CONSTANT_Utf8	(3) "run"
15.	01 00 08 3c 63 6c 69 6e 69 74 3e	CONSTANT_Utf8	(8) "<clinit>"
16.	01 00 0a 53 6f 75 72 63 65 46 69 6c 65	CONSTANT_Utf8	(10) "SourceFile"
17.	01 00 0b 4b 6c 61 73 73 65 2e 6a 61 76 61	CONSTANT_Utf8	(11) "Klasse.class"
18.	0c 00 08 00 09	CONSTANT_NameAndType	(8,9)
19.	0c 00 06 00 07	CONSTANT_NameAndType	(6,7)
20.	01 00 06 4b 6c 61 73 73 65	CONSTANT_Utf8	"Klasse"
21.	01 00 10 6a 61 76 61 2f 6c 61 6e 67 2f 4f 62 6a 65 63 74	CONSTANT_Utf8	"java/lang/Object"
22.	01 00 12 6a 61 76 61 2f 6c 61 6e 67 2f 52 75 6e 6e 61 62 6c 65	CONSTANT_Utf8	"java/lang/Runnable"

	04 20 (ACC_ABSTRACT, ACC_SYNCHRONIZED)	Merkmal	
	00 03	Klasse	(3)
	00 04	Superklasse	(4)
	00 01	Anzahl_Interfaces	(1)
Interfaces			
	00 05	Interfaceindex	(5)

	00 01	Anzahl_Felder	(1)
Feldeinträge			
1.	00 09 (ACC_PUBLIC, ACC_STATIC)	Merkmale	
	00 06	Namensindex	(6)
	00 07	Signatur-Index	(7)
	00 00	Anzahl_Attribute	(0)

	00 04	Anzahl_Methoden	(4)
Methodeneinträge			
1.	00 00	Merkmale	
	00 08	Namensindex	(8) "<init>"
	00 09	Signatur-Index	(9) "()V"
	00 01	Anzahl_Attribute	(1)

	00 0a	Attributname-Index	(10) "Code"
	00 00 00 1d	Attributlänge	(29)
	00 01	Stackgröße	(1)

	00 01	Anzahl_lokale_Variablen	(1)
	00 00 00 05	Code-Länge	(5)
	2a b7 00 01 b1	Code	
	00 00	Ausnahmetabelle_Größe	(0)
	00 01	Anzahl_Attribute	(1)
	00 0b	Attributname-Index	(11) "LineNumberTable"
	00 00 00 06	Attributlänge	(6)
	00 01	Zeilennummertabelle_Größe	(1)
	00 00	Anfang_pc	
	00 01	Zeilennummer;	
2.	04 00 (ACC_ABSTRACT)	Merkmale	
	00 0c	NamensIndex	(12) "Methode"
	00 0d	Signatur-Index	(13) "(ZI)Ljava/lang/String;"
	00 00	Anzahl_Attribute	(0)
3.	04 01 (ACC_ABSTRACT, ACC_PUBLIC)	Merkmale	
	00 0e	NamensIndex	(14) "<clinit>"
	00 09	Signatur-Index	(9) "()V"
	00 00	Anzahl_Attribute	(0)
4.	00 08 (ACC_STATIC)	Merkmale	
	00 0f	NamensIndex	(15) "run"
	00 09	Signatur-Index	(9) "()V"
	00 01	Anzahl_Attribute	(1)
	00 0a	Attributname-Index	(10) "Code"
	00 00 00 1d	Attributlänge	(29)
	00 01	Stackgröße	(1)
	00 00	Anzahl_lokale_Variablen	(0)
	00 00 00 05	Code-Länge	(5)
	03 b3 00 02 b1	Code	
	00 00	Ausnahmetabelle_Größe	(0)
	00 01	Anzahl_Attribute	(1)
	00 0b	Attributname-Index	(11) "LineNumberTable"
	00 00 00 06	Attributlänge	(6)
	00 01	Zeilennummertabelle_Größe	(1)
	00 00	Anfang_pc	(0)
	00 02	Zeilennummer	(2)
	00 01	Anzahl_Attribute	(1)
Attributeinträge			
	00 10	Attributname-Index	(16) "SourceFile"
	00 00 00 02	Attributlänge	(2)
	00 11	Quelldatei-Index	(17) "Klasse.class"

Literatur- und Quellverzeichnis

[YeL97] Frank Yellin, Tim Lindholm: The Java Virtual Machine Specification, Addison-Wesley 1997
[YeL_Online] http://java.sun.com/docs/books/vmspec
[Ven99] Bill Venners: Inside the Java2 Virtual Machine, McGraw Hill, 1999
[Internet] http://homepages.fh-giessen.de/~hg10013/Lehre/MMS/SS01_WS0102/JavaVM/classDateiformat.htm