



# Das class-Dateiformat

Das class-Dateiformat ist die elementare Datenstruktur aus der sich die Java Virtual Machine alle zur Laufzeit einer Klasse benötigten Informationen besorgen kann. Für jede in einem Programm deklarierte Klasse wird eine eigene class-Datei angelegt. Die Gesamtstruktur wird hier in C-Pseudocode angegeben. Dabei entspricht ein Feld vom Typ "int" einem 32 Bit und ein "short" einem 16 Bit langen Datenwort.

Klassendatei

```
{
  int      Erkennungszahl;
  short    Subversionsnummer;
  short    Versionsnummer;
  short    Anzahl\_Konstanten;
  Konstantenpooleintrag Konstantenpool\[Anzahl\_Konstanten - 1\];
  short    Merkmale;
  short    Klasse;
  short    Superklasse;
  short    Anzahl\_Interfaces;
  short    Interfaces\[Anzahl\_Interfaces\];
  short    Anzahl\_Felder;
  Feldeintrag Felder\[Anzahl\_Felder\];
  short    Anzahl\_Methoden;
  Methodeneintrag Methoden\[Anzahl\_Methoden\];
  short    Anzahl\_Attribute;
```

```
Attributeintrag      Attribute\[Anzahl\_Attribute\];  
}
```

Im folgenden werden alle Felder des Dateiformats besprochen. Da nicht alle Strukturen des Formats dargestellt werden, wird auf den Abschnitt "[The class File Format](#)" der Originalspezifikation verwiesen.

## Erkennungszahl, Subversionsnummer und Versionsnummer

### Erkennungszahl

Die ersten 32 Bit bilden die Kennung einer class-Datei. Sie wird verwendet um es der Java Virtual Machine zu ermöglichen eine Datei als class-Datei zu identifizieren und wurde auf den hexadezimalen Wert CAFEBAFEh festgelegt.

### Subversionsnummer, Versionsnummer

Auf die Erkennungszahl folgen jeweils zwei 16 Bit Werte die die Versionsnummern des Compilers, mit dem die class-Datei erzeugt wurde angeben. Damit kann der Javainterpreter herausfinden, ob es zu etwaigen Versionkonflikten kommen und er die Klasse nicht ausführen kann. Bedingung für eine kompatible class-Datei ist, daß die Versionsnummer mit der des Interpreters übereinstimmt. Bei korrekter Implementierung des Interpreter sollte er mit allen Subversionsnummer zurecht kommen.

## Konstantenpool

Ein elementarer Bestandteil der Klassendatei ist der Konstantenpool. Fast alle weiteren Einträge beziehen sich auf ihn. Alle in einer Klasse enthaltenen Konstanten werden in ihm abgelegt. Dazu stehen 12 unterschiedliche Konstantentypen zu Verfügung. Sie werden in der folgenden Tabelle aufgeführt und exemplarisch erklärt. Für eine vollständige Übersicht wird auf den Abschnitt "[Constant Pool](#)" der Originalspezifikation verwiesen. Der Konstantenpool ist ein Array mit "Anzahl\_Konstanten - 1" Elementen. Der nullte Eintrag wird grundsätzlich nicht genutzt und ist zur freien Verwendung jeder Interpreter-Implementierungen verfügbar. Anders wie in einem normalen Array mit identischen Elementen, sind die Einträge im Konstantenpool unterschiedlich groß. Man kann also nicht wahlfrei auf das i-te Element zugreifen, sondern muß sequentiell alle vorangegangenen Einträge lesen.

Typenbezeichnung	Wert
CONSTANT_Utf8	1
CONSTANT_Unicode	2
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\_Utf8**

Der CONSTANT\_Utf8 Typ enthält konstante Zeichenketten die nach dem Utf8 Standard kodiert sind. Dieses Format erlaubt es ASCII Zeichen mit 1 Byte abzulegen, unterstützt aber trotzdem Unicode. Dieser wird dann in einer erweiterten Darstellung mit 2 oder 3 Byte pro Zeichen gespeichert. Die Struktur eines Eintrags wird wieder in C-Pseudocode dargestellt.

```
CONSTANT_Utf8_Eintrag
{
    byte        Typ;
    short       Länge;
    byte        Zeichen[Länge];
}
```

Typ            hat den Wert CONSTANT\_Utf8, also 1  
Länge          gibt die Anzahl der Byte im folgenden Feld an  
Zeichen[Länge] enthält die eigentliche Zeichenkette in Utf8 Darstellung

**CONSTANT\_InterfaceMethodref**

Diese Element enthält Einträge um Methoden die durch ein Interface implementiert wurden zu

spezifizieren.

```
CONSTANT_InterfaceMethodref
{
    byte          Typ;
    short         Klassenindex;
    short         Name-/Typ-Index;
}
```

Typ hat den Wert CONSTANT\_InterfaceMethodref, also 11  
Klassenindex enthält einen Index in den Konstantenpool vom Typ  
CONSTANT\_Class um die Klasse anzugeben aus der die  
Methode stammt  
Name-/Typ-Index enthält einen Index in den Konstantenpool vom Typ  
CONSTANT\_NameAndType um die Signatur der Methode  
anzugeben

Für die spezielle Struktur weitere Konstanteneinträge wird auf den Abschnitt "[Constant Pool](#)" der Originalspezifikation verwiesen.

## Merkmale, Klasse und Superklasse

### Merkmale

In diesem Feld sind Merkmale von Klassen als Bitflags gespeichert. Die nachfolgende Tabelle

führt alle möglichen Merkmale. Es gibt allerdings Merkmale die nur auf Felder oder Methoden einer Klasse angewendet werden können und für eine Klasse keinen Sinn machen. Dies gilt natürlich auch umgekehrt. Diese werden in den Feld- und Methodeneinträgen verwendet, die etwas weiter unten besprochen werden.

<b>Merkmalsname</b>	<b>Wert</b>	<b>Bedeutung</b>	<b>Benutzt von</b>
ACC_PUBLIC	0x0001	sichtbar für alle	Klassen, Methoden, Variablen
ACC_PRIVATE	0x0002	nur für die eigene Klasse sichtbar	Methoden, Variablen
ACC_PROTECTED	0x0004	nur für abgeleitete Klassen sichtbar	Methoden, Variablen
ACC_STATIC	0x0008	Variable oder Methode ist statisch	Methoden, Variablen
ACC_FINAL	0x0010	keine Ableitung erlaubt kein Überschreiben erlaubt kein Zuweisen nach der Initialisierung erlaubt	Klassen Methoden Variablen
ACC_SYNCHRONIZED	0x0020	Benutzung muß durch Monitor geschützt werden	Methoden
ACC_VOLATILE	0x0040	darf nicht im Cache abgelegt werden	Variablen
ACC_TRANSIENT	0x0080	darf nicht für persistente Objekte verwendet werden	Variablen
ACC_NATIVE	0x0100	ist nicht in Java, sondern in einer anderen Sprache implementiert	Methoden
ACC_INTERFACE	0x0200	ist eine Schnittstelle	Klassen
ACC_ABSTRACT	0x0400	es gibt keinen Code	Klassen, Methoden

## **Klasse, Superklasse**

Diese beiden 16 Bit breiten Einträge bilden jeweils einen Index auf CONSTANT\_Class Konstanten im Konstantenpool und bestimmen so die eigene Klasse und Superklasse. Ist Superklasse Null, so handelt es sich um die Klasse "java.lang.Object" die an der Wurzel der Vererbungshierarchie von Java steht.

## **Anzahl\_Interfaces, Interfaces**

Wie bei allen Arrays im class-Dateiformat wird auch dem Interfacearray die Anzahl der Elemente im "Anzahl\_Interfaces"-Feld vorangestellt. Das Interfacearray enthält Indizes in den Konstantenpool. An den angegebenen Stellen befinden sich CONSTANT\_Class Einträge die alle durch ein Interface hinzugekommenen Klassen angeben.

## **Anzahl\_Felder, Felder**

"Anzahl\_Felder" gibt die Anzahl der Instanzvariablen der Klassendeklaration im darauf folgenden Feldarray an. Das Feldarray enthält für jede Variable der Klasse eine Beschreibungsstruktur. Diese hat folgenden Aufbau.

```
Feldeintrag
{
    short           Merkmale;
```



short	Namensindex;
short	Signatur-Index;
short	Anzahl_Attribute;
Attributeintrag	Attribute[Anzahl_Attribute];
}	
Merkmale	enthält eine Reihe von <a href="#">Merkmalen</a> die für Felder gültig sind
Namensindex	Index in den Konstantenpool auf einen CONSTANT_Utf8 Eintrag. Gibt den Namen der Variablen an
Signatur-Index	Index in den Konstantenpool auf einen CONSTANT_NameAndType Eintrag. Liefert den Typ der Variablen
Anzahl_Attribute	Anzahl der Attributeinträge im angrenzenden Array
Attribute[Anzahl_Attribute]	Attributeintrag für Variablen (siehe <a href="#">Attribute von Feldern</a> )

## Anzahl\_Methoden, Methoden

"Anzahl\_Methoden" gibt die Anzahl der Methoden der Klassendeklaration im darauf folgenden Methodenarray an. Das Methodenarray enthält für jede Methode der Klasse eine Beschreibungsstruktur. Diese hat folgenden Aufbau.

```
Methodeneintrag
{
  short      Merkmale;
  short      Namensindex;
  short      Signatur-Index;
  short      Anzahl_Attribute;
  Attributeintrag Attribute[Anzahl_Attribute];
}
```

Merkmale enthält eine Reihe von [Merkmalen](#) die für Methoden gültig sind

Namensindex Index in den Konstantenpool auf einen CONSTANT\_Utf8 Eintrag. Gibt den Namen der Methoden an

Signatur-Index Index in den Konstantenpool auf einen CONSTANT\_NameAndType Eintrag. Liefert die Signatur der Methode

Anzahl\_Attribute Anzahl der Attributeinträge im angrenzenden Array

Attribute[Anzahl\_Attribute] Attributeintrag für Methoden (siehe [Attribute von Methoden](#))

## **Anzahl\_Attribute, Attribute**

Mit Attributen werden Klassen, Methoden und Felder näher beschrieben. Die Attribute werden

durch einen Index in den Konstantenpool und dem dort vorliegenden Utf8 Eintrag identifiziert. Dieser muß dazu der Name des Attributs selbst sein. Also z.B. muß die Zeichenkette "SourceFile" für das SourceFile-Attribut gefunden werden. Falls neue Attribute in einer neuen Java Version eingeführt werden, so muß ein älterer Interpreter diese ignorieren können. Dazu ist der zweite Eintrag der Attributstrukturen immer die Länge des Attributs, so daß er das unbekannte Attribut überlesen kann. Für Klassen, Methoden und Felder gibt es eine unterschiedliche Anzahl von möglichen Attributen. Diese werden im folgenden beschrieben.

### Attribute für Klassen

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

SourceFile-Attribut

```
{
  short      Attributname-Index;
  int        Attributlänge;
  short      Quelldatei-Index;
}
```

Attributname-Index	zeigt auf einen Utf8 Eintrag mit der Zeichenkette "SourceFile"
Attributlänge	gibt die Länge des Attributs an
Quelldatei-Index	zeigt auf einen Utf8 Eintrag mit dem Namen der class-Datei

### Attribute für Felder

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

ConstantValue-Attribut

```
{  
    short      Attributname-Index;  
    int        Attributlänge;  
    short      Konstantenindex;  
}
```

Attributname-Index	zeigt auf einen Utf8 Eintrag mit der Zeichenkette "ConstantValue"
Attributlänge	gibt die Länge des Attributs an
Konstantenindex	zeigt auf einen numerischen Eintrag im Konstantenpool mit dem Initialisierungswert für das Feld

### Attribute für Methoden

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 eine Liste aller Ausnahmen enthält, die von der entsprechenden Methode geworfen werden können.

## Code-Attribut

```
{
  short      Attributname-Index;
  int        Attributlänge;
  short      Stackgröße;
  short      Anzahl_lokale_Variablen;
  int        Code-Länge;
  byte       Code[Code-Länge];
  short      Ausnahmetabelle_Größe;
  {
    short     Anfang_pc;
    short     Ende_pc;
    short     Handler_pc;
    short     Ausnahmetyp;
  }
  short      Ausnahmetabelle[Ausnahmetabelle_Größe];
  short      Anzahl_Attribute;
  Attributeintrag Attribute[Anzahl_Attribute];
}
```

Attributname-Index	zeigt auf einen Utf8 Eintrag mit der Zeichenkette "SourceFile"
Attributlänge	gibt die Länge des Attributs an
Stackgröße	gibt die maximal benötigte Größe des Stacks für diese

Anzahl_lokale_Variablen	Methode an gibt an wieviel lokale Variablen in der Methoden verwendet werden
Code-Länge	gibt an, aus wieviel Byte der Code für diese Methode besteht
Code[Code-Länge]	enthält den eigentlichen Bytecode
Ausnahmetabelle_Größe	gibt an wieviel Einträge die Ausnahmetabelle hat
Ausnahmetabelle[Ausnahmetabelle_Größe]	enthält Informationen für jede in diese Methode behandelte Ausnahme
Anfang_pc	bezeichnet die Stelle im Bytecode, ab der der Exception Handler, der in diesem Eintrag beschrieben wird, gültig ist. Der Wert bezeichnet einen Offset ab dem Anfang des Codes dieser Methode.
Ende_pc	bezeichnet die Stelle im Code, ab der der Exception Handler nicht mehr gültig ist
Handler_pc	gibt die Stelle im Code an,

Ausnahmetyp	ab der der Code des Handlers selbst steht zeigt auf einen CONSTANT_Class Eintrag im Konstantenpool, der die Klasse des Exception Handlers angibt
Anzahl_Attribute	Anzahl der Attribute im Code-Attribut
Attribute	enthält die Attribute des Code-Attributes. Es sind zwei Attribute definiert, das <a href="#">LineNumberTable</a> - und <a href="#">LocalVariableTable</a> -Attribut. Auch deren Struktur sind der Originalspezifikation zu entnehmen.

```
Exceptions-Attribut
{
    short  Attributname-Index;
    int    Attributlänge;
    short  Anzahl_Ausnahmen;
    short  Ausnahmeindextabelle[Anzahl_Ausnahmen];
}
```

Attributname-Index	zeigt auf einen Utf8 Eintrag mit der Zeichenkette "ConstantValue"
Attributlänge	gibt die Länge des Attributs an
Anzahl_Ausnahmen	gibt die Anzahl der Feldeinträge in folgender Tabelle an
Ausnahmeindextabelle[Anzahl_Ausnahmen]	Diese Tabelle enthält ein Reihe von Indizes in den Konstantenpool, die jeweils auf Klasseneinträge zeigen, die anzeigen, welche Klassen von Ausnahmen diese Methode auslösen kann.

## Signaturen

Signaturen dienen dazu die Parametertypen und den Typ des Rückgabewerts einer Methode zu bestimmen. Man bedient sich dazu einer speziellen Codierung in einer Zeichenkette. Diese Zeichenkette wird als Utf8 Eintrag im Konstantenpool abgelegt. In folgender Tabelle werden die Buchstabencodes der einzelnen Datentypen und deren Beschreibung angegeben.

Buchstaben	Java-Datentyp	Beschreibung
Z	boolean	Wahrheitswert (true oder false)



B	byte	8 Bit Wert mit Vorzeichen
C	char	16 Bit Zeichen unicode-kodiert
S	short	ganze Zahl, 16 Bit
I	int	ganze Zahl, 32 Bit
J	long	ganze Zahl, 64 Bit
F	float	Fließkommazahl, 32 Bit
D	double	Fließkommazahl, 64 Bit

Falls kein elementarer Datentyp sondern eine Objektreferenz einer bestimmten Klasse als Parameter oder Rückgabewert übergeben wird, so wird dies mit dem vollständigen Klassenpfad und Klassennamen und einem vorangestellten "L" und abschließendem ";" ausgedrückt. Der "void" Rückgabewert wird durch ein "V" dargestellt. Arrays werden durch eine eckige Klammer "[", gefolgt von der Anzahl der Arrayelemente und dem Typ der Elemente kodiert. Ist das Array von unbestimmter Größe so wird direkt der Datentyp hinter die eckigen Klammer geschrieben. Um nun die Signatur einer Methode anzugeben, werden einfach die Parameter in runden Klammern hintereinander gestellt und hinter der schließenden runden Klammer der Rückgabewert angegeben. Um die Sache etwas klarer zu machen wird ein Beispiel angegeben.

**Beispiel**

```
StringBuffer append(char[] str, int offset, int len);
```

Die Methode "append" liefert eine Objektreferenz vom Typ "StringBuffer", dies wird durch "Ljava/lang/StringBuffer;" gekennzeichnet. Das char-Array durch "[C" und die Integerparameter jeweils durch ein "I". Es ergibt sich also folgende Signatur.

```
([CII)Ljava/lang/StringBuffer;
```

Wie man sieht, ist der Methodenname selbst nicht in der Signatur enthalten. Das steht im Gegensatz zu der Bedeutung der Signatur, wie man sie gewöhnlich in der Informatik verwendet. Für die komplette Signatur (Parameter, Rückgabewert und Methodenname) steht aber der [Konstantenpooleintrag](#) `CONSTANT_NameAndType` zu Verfügung.

## Dateibeispiel

Um das class-Dateiformat etwas anschaulicher zu machen, gibt es ein abschließendes Beispiel. Es wurde eine "abstract" Klasse "Klasse" erstellt. Sie besitzt eine "abstract" Methode "Methode" und ein "static" Feld "Feld". Das Interface Runnable wurde implementiert welche die Methoden "clinit()" und "run()" der Klasse hinzufügen. Diese kann man im Methodenfeld wiederfinden. Ebenfalls ist ein Eintrag im Interfacefeld zu finden der auf das fünfte Element im Konstantenpool verweist. Dieser ist ein `CONSTANT_Class` Eintrag der auf die Utf8 Konstante „java/lang/Runnable“ zeigt. Die Methode "Methode" ist an zweiter Stelle nach dem Konstruktor "init()" der Klasse im Methodenfeld eingetragen. Das Variablenfeld enthält einen Eintrag für die Variable "Feld". In dieser Art und Weise kann man nun die komplette Systematik des class-Dateiformat nachvollziehen.

```

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

```

Konstantenpool

Interfaces

Feldeinträge

Methodeneinträge

Attributeinträge

ca	fe	ba	be	00	03	00	2d	00	17	0a	00	04	00	12	09	Ëp%{...-.....
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
Konstantenpool	ca fe ba be	Erkennungszahl	
	00 03	Subversionsnummer	(3)
	00 2d	Versionsnummer	(45)
	00 17	Anzahl Konstanten	(23)
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) "(Z)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	Merkmal	
	(ACC_ABSTRACT, ACC_SYNCHRONIZED)		
	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	Merkmale	
	(ACC_ABSTRACT)		
	00 0c	NamensIndex	(12) "Methode"
	00 0d	Signatur-Index	(13) "(Z)Ljava/lang /String;"
	00 00	Anzahl_Attribute	(0)
3.	04 01	Merkmale	
	(ACC_ABSTRACT, ACC_PUBLIC)		
	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)
Attributeinträge	00 01	Anzahl_Attribute	(1)
	00 10	Attributname-Index	(16) "SourceFile"
	00 00 00 02	Attributlänge	(2)
	00 11	Quelldatei-Index	(17) "Klasse.class"