Лекция 2. Структура байт кода

- Компиляция java в class
- ByteCode виртуальной машины Java
- Структура class файла
- Средства работы с class файлами и примеры.

Компиляция программы

Компилятор создает файл , содержащий версию кода виртуальной машины программы. Код виртуальной машины Java - это промежуточное представление программы, содержащее инструкции, которые будет выполнять виртуальная машина Java. Результат работы компилятора не является непосредственно исполняемым кодом.

В процессе компиляции исходного кода каждый отдельный класс помещается в собственный выходной файл, названный по имени класса и получающие расширение .class.

ByteCode виртуальной машины Java

Байт-код Java — набор инструкций, исполняемых виртуальной машиной Java. Каждый код операции байт-кода имеет размер один байт.

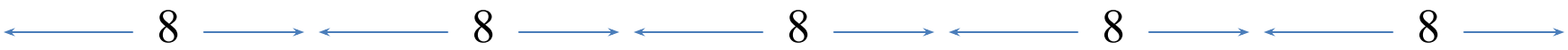
Виды инструкций:

- Загрузка и сохранение;
- Арифметические и логические операции;
- Преобразование типов;
- Создание и преобразование объекта;
- Управление стеком;
- Операторы перехода;
- Вызовы методов и возврат;

ByteCode виртуальной машины Java

Если рассматривать более детально, то формат инструкций наглядно выглядит так:

Биты



Формат

1. Операция				
2.Операция	Байт			
3. Операция	SHORT			
4. Операция	Индекс	Константа		
5. Операция	Индекс		Размерность	
6. Операция	Индекс		Параметры	0
7. Операция	Индекс		Константа	
8. Операция	32- битное смещение перехода			

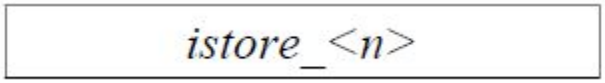
ByteCode виртуальной машины Java

Загрузка из локальной переменной:

- **i**load — для типа **int**
- **l**load — для типа **long**
- **f**load — для типа **float**
- **d**load — для типа **double**

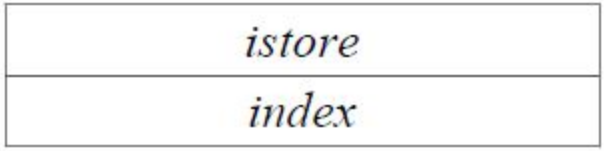
Сохранение локальной переменной типа int:

- istore_0 = 59 (0x3b)
- istore_1 = 60 (0x3c)
- istore_2 = 61 (0x3d)
- istore_3 = 62 (0x3e)



Не имеют аргументов

istore = 54 (0x36)



Имеет один аргумент

ByteCode виртуальной машины Java

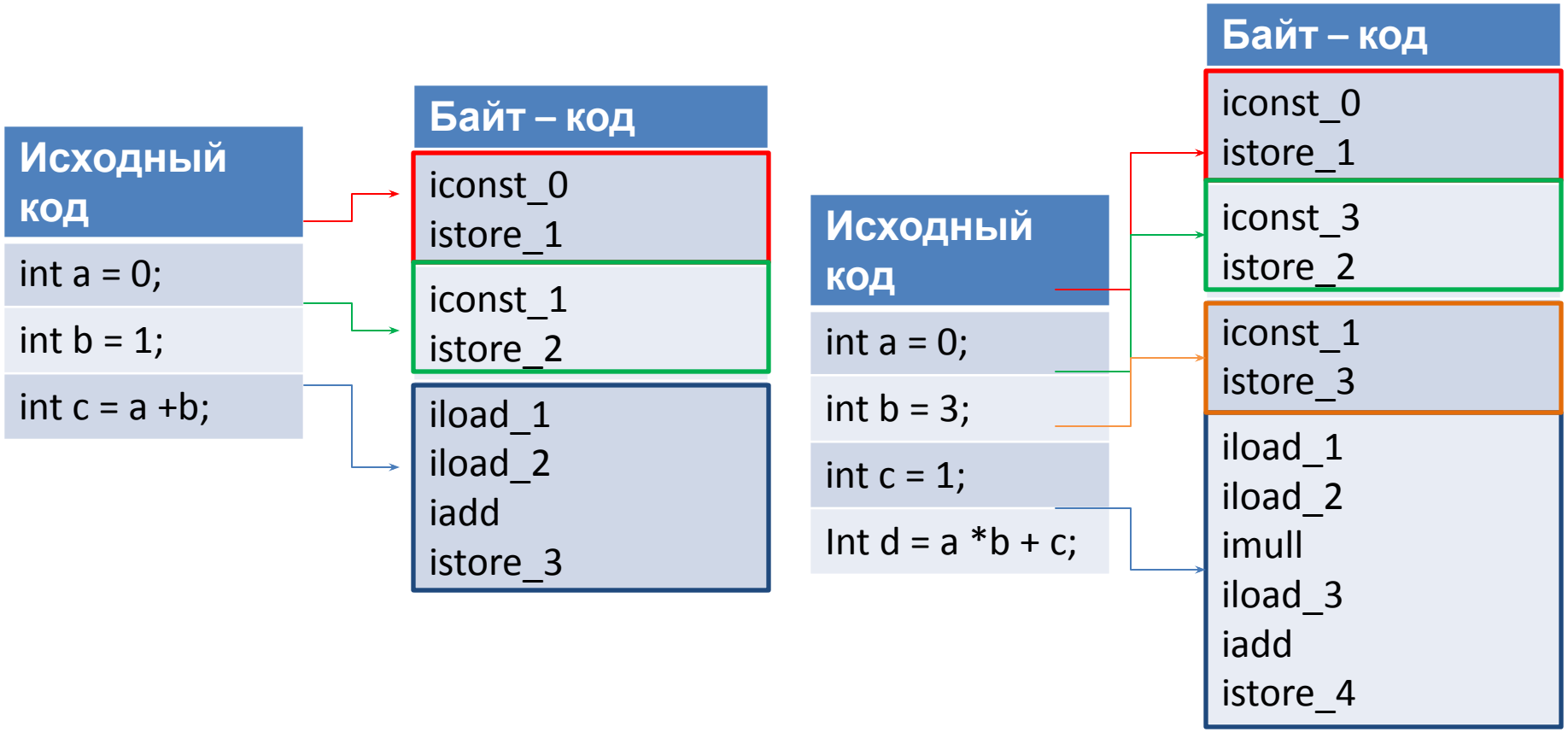
Математические операции:

- iadd = 96 (0x60) – сложение типа int
- ladd = 97 (0x61) – сложение типа long
- fadd = 98 (0x62) – сложение типа float
- dadd = 99 (0x63) – сложение типа double
- imul = 104 (0x68) – умножение типа int

```
public class Example {
      public int plus(int a) {
     int b = 1;
    return a + b;
       }
}
```

```
public int plus(int);
Code: Stack=2, Locals=3,
Args_size=2
0: iconst_1
1: istore_2
2: iload_1
3: iload_2
4: iadd
5: ireturn
LineNumberTable:
line 5: 0
line 6: 2
```

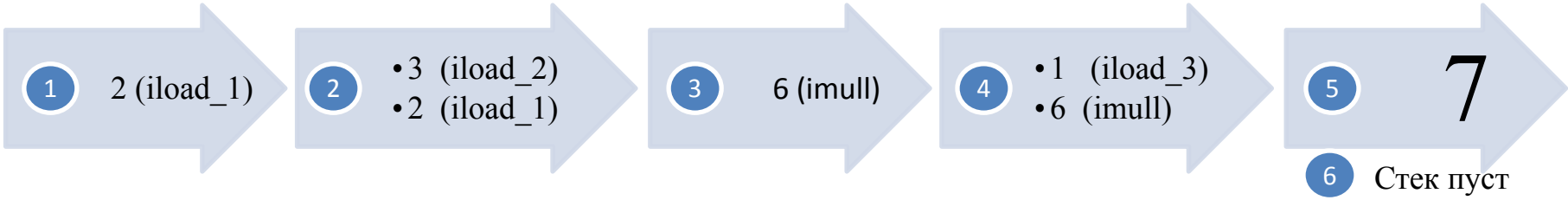
ByteCode виртуальной машины Java



ByteCode виртуальной машины Java

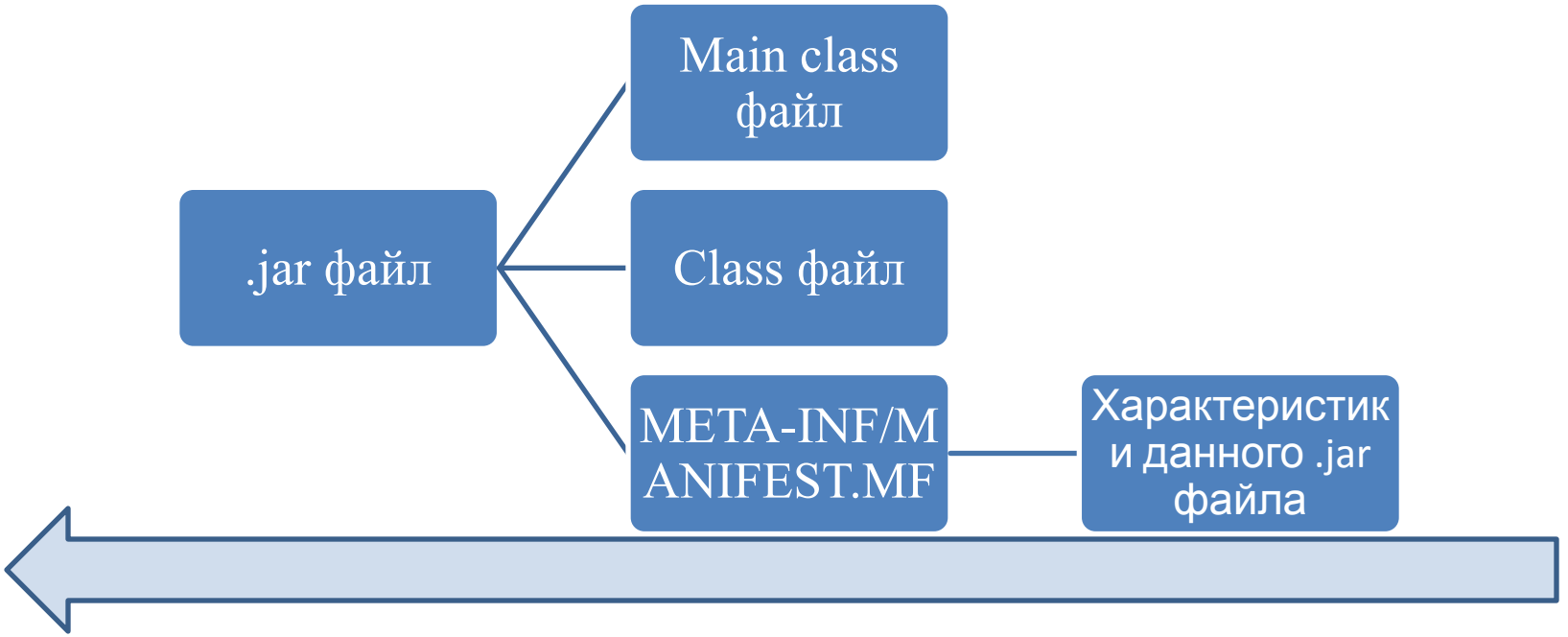
Пример вычисления:

Исходный код	ByteCode
int a =2;	1. iload_1
int b =3;	2. iload_2
int c = 1;	3. imull
int d =a*b + c;	4. iload_3
	5. iadd
	6. istore_4



Class файл

- Содержит байт-код, который выполняется на виртуальной машине.
- Содержит информацию о классе.
- Генерируется компилятором из исходного кода (.java)
- Jar файл – представляет собой zip архив class файлов.



Структура class файла

class файл состоит из 1,2,4 байтовых значений:

- u1 – byte (1 байт)
- u2 – long (2 байта)
- u4 – int (4 байта)

```
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];
}
```

Структура class файла

Разберемся на самом тривиальном примере:

```
package hello;

public class App {
    public static void main(String[] args) {
        System.out.println("Hello world!");
}
}
```

Скомпилировав данный код, мы получим class файл. С помощью любого hex-редактора откроем его.

Структура class файла

```
0000000 ca fe ba be 00 00 00 34 00 1d 0a 00 06 00 0f 09
0000010 00 10 00 11 08 00 12 0a 00 13 00 14 07 00 15 07
0000020 00 16 01 00 06 3c 69 6e 69 74 3e 01 00 03 28 29
0000030 56 01 00 04 43 6f 64 65 01 00 0f 4c 69 6e 65 4e
0000040 75 6d 62 65 72 54 61 62 6c 65 01 00 04 6d 61 69
0000050 6e 01 00 16 28 5b 4c 6a 61 76 61 2f 6c 61 6e 67
0000060 2f 53 74 72 69 6e 67 3b 29 56 01 00 0a 53 6f 75
0000070 72 63 65 46 69 6c 65 01 00 08 41 70 70 2e 6a 61
0000080 76 61 0c 00 07 00 08 07 00 17 0c 00 18 00 19 01
0000090 00 0c 48 65 6c 6c 6f 20 77 6f 72 6c 64 21 07 00
00000a0 1a 0c 00 1b 00 1c 01 00 09 68 65 6c 6c 6f 2f 41
00000b0 70 70 01 00 10 6a 61 76 61 2f 6c 61 6e 67 2f 4f
00000c0 62 6a 65 63 74 01 00 10 6a 61 76 61 2f 6c 61 6e
00000d0 67 2f 53 79 73 74 65 6d 01 00 03 6f 75 74 01 00
00000e0 15 4c 6a 61 76 61 2f 69 6f 2f 50 72 69 6e 74 53
00000f0 74 72 65 61 6d 3b 01 00 13 6a 61 76 61 2f 69 6f
0000100 2f 50 72 69 6e 74 53 74 72 65 61 6d 01 00 07 70
0000110 72 69 6e 74 6c 6e 01 00 15 28 4c 6a 61 76 61 2f
00001a0 00 0e 00001a2
```


Структура class файла

Так выглядит байт код нашей простой программы. Есть два вида байт кода. Собственно, сам **байт код** представленный на предыдущем слайде и его **мнемоническое представление для виртуальной машины Java**, которое будет показано позже. Рассмотрим байт код подробнее.

u4 magic

Это 4 байта для magic, который определяет формат class файла (ca fe ba be).
Позволяет идентифицировать .class файл.
Всегда принимает значение: 0xCAFEBABE.

u2 minor_version (00 00)
u2 major_version (00 34)

Вспомогательная и основная версии class файла.

u2 constant_pool_count **(** 00 1d)

Размер массива констант. Эти два байта представляют **constant_pool_count** и отвечают за размер **constant_pool.**

Структура class файла

cp_info constant_pool [constant_pool_count-1]
Пул констант представлен в виде массива структур представляющих различные строковые константы - имена классов и интерфейсов, полей, методов и другие константы, на которые есть ссылки в файле класса.

Далее в байт коде идут элементы вида:

```
cp_info {
u1 tag; // 1 байт на тег
u1 info[]; // массив с описанием
}
```

Формат каждого элемента пула констант определяется первым байтом (**tag**).

Структура class файла

Constant Type	Value
CONSTANT_Class	7
CONSTANT_Long	5
CONSTANT_Methodref	10
CONSTANT_String	8
CONSTANT_Float	4
CONSTANT_Fieldref	9
CONSTANT_Utf8	1
CONSTANT_MethodType	16

```
Constant_Long_info {
    u1 tag;
    u4 high_bytes;
    u4 low_bytes;
}
```

```
Constant_utf8_info {
    u1 tag;
    u2 length;
    u1 bytes[length];
}
```

Структура class файла

u2 access_flags

Флаг доступа, отображающий модификаторы, заданные в определении класса (**public, final, abstract** и т.д.), а также признак класса или интерфейса.

Значение элемента является маской флагов, используемых таким образом, чтобы обозначить права доступа и свойства этого класса.

Flag Name	Value	Interpretation
ACC_PUBLIC	0x0001	Объявлен открытым. Может быть доступен извне пакета.
ACC_FINAL	0x0010	Объявлен final. Классы наследники не допускаются.
ACC_SUPER	0x0020	Обращение к методам суперкласса. Когда вызывается инструкция invokespecial.
ACC_INTERFACE	0x0200	Интерфейс. Не класс.
ACC_ABSTRACT	0x0400	Объявлен абстрактным. Не может иметь экземпляров.
ACC_SYNTHETIC	0x1000	Объявлен synthetic; Не представлен в исходном коде.
ACC_ANNOTATION	0x2000	Объявлен как тип аннтоция.
ACC_ENUM	0x4000	Объявлен в качестве типа перечисления.

Структура class файла

u2 this_class
u2 super_class
Ссылки на константу с названием класса и его суперкласса.

u2 interfaces_count
Размер массива интерфейсов.

u2 interfaces[interfaces_count]
Массив интерфейсов. Каждый элемент массива является индексом таблицы пула констант, где указывается имя интерфейса.

Структура class файла

field_info fields[fields_count]
Массив полей.

u2 fields_count
Размер массива полей.

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

Структура class файла

method_info methods[methods_count]
Массив методов.

u2 methods_count
Размер массива методов.

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

Структура class файла

attribute_info attributes[attributes_count]
Массив атрибутов.

u2 attributes_count
Размер массива атрибутов.

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

Нас интересует атрибут Code, так как именно в нем содержится необходимая нам информация. Если у вас нет исходных кодов программы, есть только class файл и вам необходимо внести некоторые изменения в работе программы, то начинать следует именно с этого атрибута.

Структура class файла

Список атрибутов (attribute)	
ConstantValue	LineNumberTable
Code	LocalVariabletable
StackMapTable	LocalVariableTypeTable
Exceptions	Deprecated
InnerClasses	RuntimeVisibleAnnotations
EnclosingMethod	RuntimeInvisibleAnnotations
Synthetic	RuntaimeVisibleParameterAnnotations
Signature	RuntaimeInvisibleParameterAnnotations
SourceFile	AnnotationDefault
SourceDebufExtension	BootstrapMethods

Структура class файла

```
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];
}
```

Средства работы с class файлами

JBE - Java Bytecode Editor – программа, позволяющая просматривать и редактировать class файлы.

javap – декомпилятор class файлов.

javap команда дизассемблирует один или более файлов класса. Его вывод зависит от используемых опций. Если никакие опции не используются, **javap** распечатывает пакет, защищенные, и общедоступные поля и методы классов, которые передают к этому. **javap** печатает свой вывод к stdout.

Средства работы с class файлами

Некоторые опции javap:

-? Распечатывает сообщение справки для javap.

-p Показать все классы и элементы.

-s Печатает внутренние подписи типа.-sysinfo

-c Распечатывает дизассемблированный код, то есть, инструкции, которые включают Байт-коды Java для каждого из методов в классе. Они документируются в Спецификацию виртуальной машины Java.

Средства работы с class файлами

Файл Main.java

```
public class Main {
    public static void main (String [] args ){
                int a = 4;
        int b = 5;
        int c = a+b;
    }
}
```

Скомпилируем данный код с помощью команды **javac Main.java**
Затем в командной строке или если в используемой вами IDE есть своя командная строка, то в ней выполним команду:

javap -c Main.class

Средства работы с class файлами

Команда вывела на экран инструкции, которые включают Байт-коды Java для каждого из методов в классе.

```
Compiled from "Main.java"
public class Main {
  public Main();
    Code:
       0: aload_0
       1: invokespecial #1                  // Method java/lang/Object."<init>":
()V
       4: return

  public static void main(java.lang.String[]);
    Code:
       0: iconst_4
       1: istore_1
       2: iconst_5
       3: istore_2
       4: iload_1
       5: iload_2
       6: iadd
       7: istore_3
       8: return
}
```

Средства работы с class файлами

JBE - Java Bytecode Editor – программа, позволяющая просматривать и редактировать class файлы.

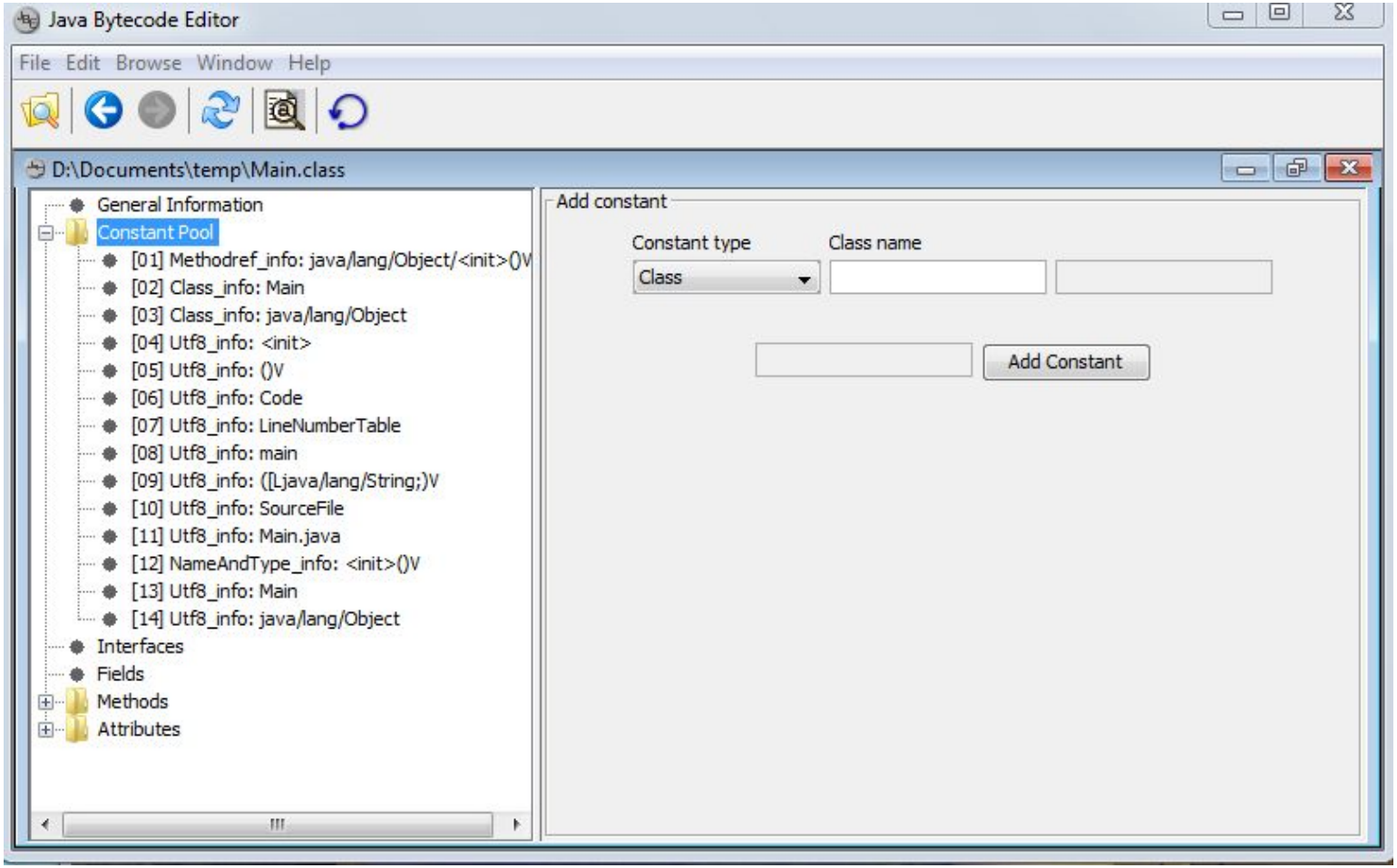
Файл Main.java

```
public class Main {
    public static void main (String [] args ){
            int a = 4;
       int b = 5;
       int c = a+b;
    }
}
```

Скомпилируем данный код с помощью команды **javac Main.java**. Полученный class файл загрузим в ByteCode Editor.

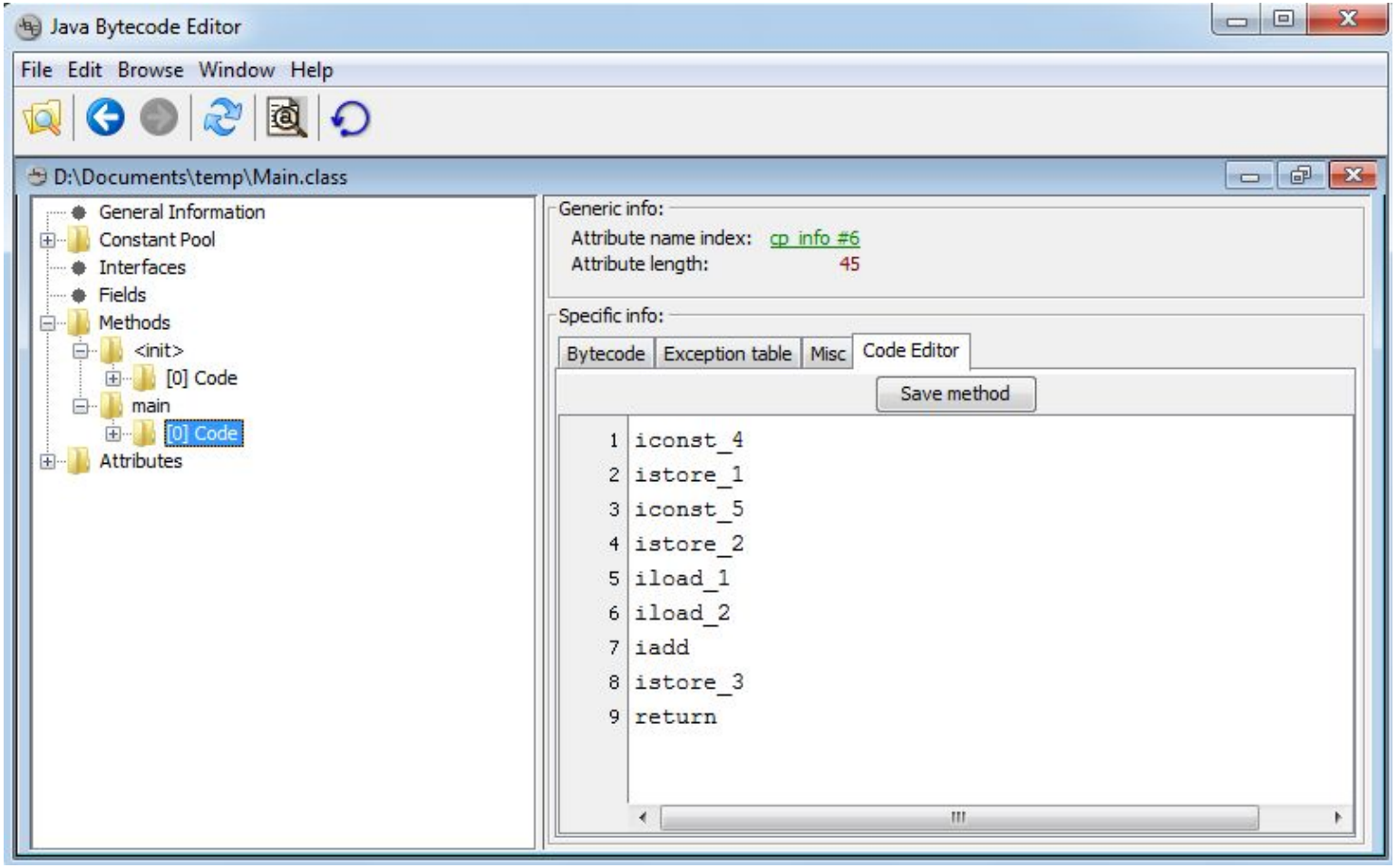
Средства работы с class файлами

Как видно на данном скриншоте, программа позволяет посмотреть пул констант



Средства работы с class файлами

А также, возможен просмотр списка инструкций, в атрибуте Code



Результат идентичен выполнению команды javap -c

Java