ORACLE
Academy

Java Programming

9-1
Java Bytecode

ORACLE
Academy

Objectives

- This lesson covers the following topics:
- Understanding Bytecode
  - How to obtain the bytecode listings
  - How to read the bytecode
  - How the language constructs are mirrored by the compiler: calculation, method calls

ORACLE Academy

Why Understand Bytecode?

- Understanding the Java bytecode is crucial when debugging and doing performance and memory usage tuning
- Knowing the assembler instructions that are generated from source code can help you know how to code differently to achieve your quality goals
- In addition, the knowledge of bytecode can help you to better understand the Java programming language and know the implementation details for your source code

What is the Bytecode?

- It is not usual for programmers to program in Java byte code directly, rather they program in Java and then compile their programs into Java byte code
- Each opcode of the bytecode is represented by a single byte

What is the Bytecode?

- Bytecode is an intermediate representation of a program, halfway between human readable source and machine code
- Bytecode is produced by javac from Java source code files
- Some high-level language features have been removed and do not appear in bytecode
- For example, Java's looping and the if statement are not present, and instead, have become branch instructions in the bytecode

Academy

How to Inspect the Instruction Set

- In order to inspect the output from the Java compiler, we use a program called a disassembler which converts the Java byte code program from a form suitable for efficient interpretation into a textual form suitable for developers to read and study
- Given a compiled Java class in the file SampleClass.class the Java disassembler is invoked with a javap command such as the following:
    - javap -c Sampleclass (The -c option disassembles the code)

JVM Instruction Set

- Java bytecodes are the machine language of the JVM
- In Java source code, the method defines behavior, so after the Classloader loads a class, the stream of bytecodes for each method are stored in the method area of JVM during runtime
- The bytecodes of a method are executed when the method is invoked by the thread during the course of running the application

JVM Instruction Set

- The stream of bytecodes consists of a one-byte opcode followed by zero or more operands
- Instruction Set format:
  - 1 byte opcode
  - 0 or more bytes of operands
- Each opcode has a mnemonic, for example, istore_0.
- JVM instructions are explicitly typed: different codes for instructions for integers, floats, arrays, reference types, etc.

ORACLE
Academy

JVM Instruction Set - Mnemonics

- This is reflected by a naming convention in the first letter of the opcode mnemonics:
    - JVM-Type prefix
        - Byte b
        - Short s
        - Integer i
        - Long l
        - Character c
        - Single float f
        - Double float d
        - References a

        - Example: iload load integer type or fload load float type

ORACLE Academy

JP 9-1
Java Bytecode

JVM Instruction Set - Mnemonics

- Below is a list of common uses of opcode:
  - Shuffling (pop, swap, dup, ...)
  - Calculating (iadd, isub, imul, idiv, ineg,...)
  - Conversion (d2i, i2b, d2f, i2z,...)
  - Local storage operation (iload, istore,...)
  - Array Operation (arraylength, newarray,...)
  - Object management (get/putfield, invokevirtual, new)
  - Push operation (aconst_null, iconst_m1,....)
  - Control flow (nop, goto, jsr, ret, tableswitch,...)
  - Threading (monitorenter, monitorexit,...)

Loops in Java Source Code

- Consider the following Java Methods :
  - void testFor() {
  -     for (int i = 0 ; i < 100 ; i++){;}
  - }

  - int i = 0;
  - void testWhile() {
  - while (i < 100) {
  -     i++;}
  - }

Both methods initialize the loop variable i to zero and then increment until i reaches the 100 limit.

- We consider these testFor() and testWhile() methods equivalent in that we can use each method to perform the same task without the need to change the code

ORACLE
Academy

Loops in Java Byte Code

- The Java byte code language has neither a for loop nor a while loop
- Unlike the source code where the loops are visible, in Java byte code the two forms of loops are identical:
  - Method void testFor() Method void testWhile()
    - 0: iconst_0 0: iconst_0
    - 1: istore_1 1: istore_1
    - 2: iload_1 2: iload_1
    - 3: bipush 100 3: bipush 100
    - 5: if_icmpge 14 5: if_icmpge 14
    - 8: iinc 1, 1 8: iinc 1, 1
    - 11: goto 2 11: goto 2
    - 14: return 14: return

Loops in Java Byte Code

- Below is the byte code generated from the javac compiler for both methods:
  - 0: iconst_0 The integer constant zero is pushed on top of the stack
  - 1: istore_1 The top of the stack is stored into local variable array one (the variable i)
  - 2: iload_1 Load from the local variable
  - 3: bipush 100 Load the integer 100 on top of the stack
  - 5: if_icmpge 14 Compare the top two items on the stack and jump if (i>=100)
  - 8: iinc 1, 1 Increment local variable on by 1 (i++)
  - 11: goto 2
  - 14: return Return void when the end of the method is reached

Local Storage Operation

- The load and store instructions transfer values between the local variables and the operand stack of a Java Virtual Machine frame
  - iload, iload_0,fload_0, ..
  - Istore, Istore, istore_0..
  - ldc, sipush

ORACLE Academy

Arithmetic Instructions Mnemonics

- For a Complete list of the Arithmetic instructions, refer to the JVM Specifiction
- Below is a small subset :
  - Add: iadd, ladd, fadd, dadd
  - Subtract: isub, lsub, fsub, dsub
  - Multiply: imul, lmul, fmul, dmul
  - Divide: idiv, ldiv, fdiv, ddiv

Java Bytecode Example

- Java Souce Code:

```
public class SampleClass {
   public static int test(){
      int x=99999;
      int y=1;
      int z = x + y;
      return z;
   }//end method test
}//end class SampleClass
```

- # javap –c SampleClass
- Java ByteCode:
- public static int test();
- Code:
- 0: ldc #2 // int 99999
- 2: istore_0
- 3: iconst_1
- 4: istore_1
- 5: iload_0
- 6: iload_1
- 7: iadd
- 8: istore_2
- 9: iload_2
- 10: ireturn

Object Management

- Although both class instances and arrays are objects, the Java Virtual Machine creates and manipulates class instances and arrays using distinct sets of instructions:
  - Create a new class instance: new
  - Create a new array: newarray, anewarray, multianewarray
- Access fields of classes (static fields, known as class variables) and fields of class instances (non-static fields, known as instance variables):
  - getstatic, putstatic, getfield, putfield

Instruction-Set: Memory Access

- In the JVM, the contents of different "kinds" of memory can be accessed by different kinds of instructions
  - accessing locals and arguments: load and store instructions
  - accessing fields in objects: getfield, putfield
  - accessing static fields: getstatic, putstatic

Instruction-Set: Memory Access

- Note:
  - Static fields are like global variables
  - They are allocated in the “method area” where code for methods and representations for classes (including method tables) are also stored
  - getfield and putfield access memory in the heap

The new Operator Example

- Java Souce Code:

```
public class SampleClass {
    public static int testID=100;
    public static void test()
    {
        SampleClass sc=new SampleClass();
        testID=200;
    }//end method test

}//end class SampleClass
```

- In the SampleClass java class, we declare a static variable testID.
- And the the test() method, a new Sampleclass object is instantiated

ORACLE Academy

The new Operator Example

- Java ByteCode:
- 0: new #2 // class SampleClass
- 3: dup
- 4: invokespecial #3 // Method "<init>":()V
- 7: astore_0
- 8: sipush 200
- 11: putstatic #4 // Field testID:I
- 14: return

- In the SampleClass test() method bytecode:
  - Line 0: create a new object
  - Line 3: duplicate the top operand stack value
  - Line 4: invoke the SampleClass constructor
  - Line 7: store the newly created object reference into the local variable

The new Operator Example

- Java ByteCode:
- 0: new #2 // class SampleClass
- 3: dup
- 4: invokespecial #3 // Method "<init>":()V
- 7: astore_0
- 8: sipush 200
- 11: putstatic #4 // Field testID:I
- 14: return

- In the SampleClass test() method bytecode:
  - Line 8: push the short 200 into the top of the stack
  - Line 11: set the static field testID to 200
  - Line 14: return the method call

Instruction-Set

- Method invocation:
  - invokevirtual: the usual instruction for calling a method on an object. invokeinterface: same as invokevirtual, but used when the called method is declared in an interface (requires a different kind of method lookup)
  - invokespecial: for calling things such as constructors, which are not dynamically dispatched (this instruction is also known as invokenonvirtual)
  - invokestatic: for calling methods that have the “static” modifier (these methods are sent to a class, not to an object)
- Returning from methods:
  - return, ireturn, lreturn, areturn, freturn, …

Analyze the Bytecode : invokespecial

- Java Source Code:
  - SampleClass sc=new SampleClass();
    - The new operator will be translated into 4 instructions by javac compiler
- Java ByteCode:
  - 0: new #2 // class SampleClass
    - 3: dup
    - Duplicate the top operand stack value
    - 4: invokespecial #3 // Method "<init>":()V
    - Call the constructor of the SampleClass
  - Memory for a new instance of SampleClass class is allocated from the garbage-collected heap, and the instance variables of the new object are initialized to their default initial values

Analyze the Bytecode - Push Short

- Java Souce Code:
  - testID=200;
  - The statement will be translated into two instructions
- Java ByteCode:
  - 8: sipush 200
  - Push short value 200 into stack.
  - 11: putstatic #4 // Field testID:I
  - Set static field in class

ORACLE Academy

Instructions and the “Constant Pool”

- Many JVM instructions have operands which are indexes pointing to an entry in the so-called constant pool
- The constant pool contains all kinds of entries that represent “symbolic” references for “linking”
- This is the way that instructions refer to things such as classes, interfaces, fields, methods, and constants such as string literals and numbers

ORACLE Academy

Instructions and the "Constant Pool"

- Examples of constant pool entries that exist :
  - Class_info
  - Fieldref_info
  - Methodref_info
  - InterfaceMethodref_info
  - String
  - Integer
  - Float
  - Long
  - Double
  - Name_and_Type_info
  - Utf8_info   (Unicode characters)

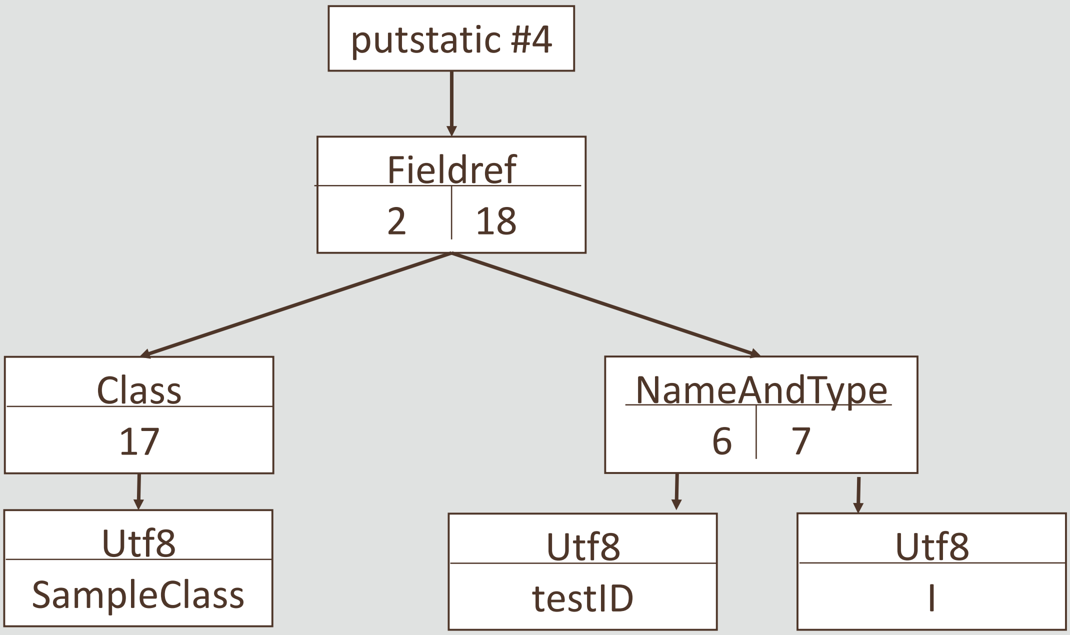
Analyze the Bytecode

- #javap –verbose SampleClass // display the constant pool
- Java ByteCode: 11: putstatic #4 // Field t
- #1 = Methodref #5.#16 // java/lang/Object."<init>":()V
- #2 = Class #17 // SampleClass
- #3 = Methodref #2.#16 // SampleClass."<init>":()V
- #4 = Fieldref #2.#18 // SampleClass.testID:I
- #5 = Class #19 // java/lang/Object
- #6 = Utf8 testID
- #7 = Utf8 I
- #8 = Utf8 <init>
- #9 = Utf8 ()V
- #10 = Utf8 Code
- estID:I

Analyze the Bytecode – Cont.

- #javap –verbose SampleClass // display the constant pool
- Java ByteCode: 11: putstatic #4 // Field testID:I
- #11 = Utf8 LineNumberTable
- #12 = Utf8 test
- #13 = Utf8 <clinit>
- #14 = Utf8 SourceFile
- #15 = Utf8 SampleClass.java
- #16 = NameAndType #8:#9 // "<init>":()V
- #17 = Utf8 SampleClass
- #18 = NameAndType #6:#7 // testID:I
- #19 = Utf8 java/lang/Object

Analyze the Bytecode - Cont



ORACLE
Academy

Summary

- In this lesson, you should have learned:
- Understanding Bytecode
  - How to obtain the bytecode listings
  - How to read the bytecode
  - How the language constructs are mirrored by the compiler: calculation, method calls

ORACLE Academy

ORACLE

Academy