TheServerSide.com

New Java 7 Features: Binary Notation and Literal Variable Initialization

Having Fun with Binary Notation

Given all the fun developers have with *hexadecimal notation*, the visionaries at Sun and Oracle decided to expand the party by introducing the equally cryptic *binary notation* to the Java language. With the addition of *binary notation*, there are now three different ways to initialize a numeric variable, first using the standard base ten number system, secondly using hex notation, and finally using binary. Here's an example that assigns the number twelve to three different variables of type byte.

```
public class BinaryLiterals {
  public static void main(String args[]){
    byte twelve = 12;
    byte sixPlusSix = 0xC;
    byte fourTimesThree = 0b1100;

    System.out.println(twelve);
    System.out.println(sixPlusSix);
    System.out.println(fourTimesThree);

  }
}
```

When the code above runs, it simply prints out the number 12 on three separate lines, demonstrating the veracity of all three notations:

12
12
12

Passing the Java 7 Professional Upgrade Exam

As part of the Oracle Certified Profession, Java 7 certification upgrade exam, currently in beta as of this writing, one of the exam objectives is to "Use binary literals." So, in this article, we're going to take a look at some of the more interesting aspects of binary notation, focussing on what you'll need to know in order to whiz by the related questions on the exam. Even if you're not planning on writing a Java cert, you'll likely find the new news on binary notation quite interesting, so you are implored to keep reading.

Taking a byte out of binary initializations

In the above code snippet, you see an example of a byte being initialized to the value of 12 using binary notation: `byte fourTimesThree = 0b1100;`

Now, everyone knows that a byte represents eight bits of data, so the following

attempted initialization, where you assign a ten bit binary number to a byte, completely blows up with the following error: **Type mismatch: cannot convert from int to byte**

```
byte data = 0b1100110011; // Type mismatch: cannot convert from int to byte
```

Interestingly, though not necessarily surprisingly, if the leading binary digits are zeroes, the JVM recognizes that these numbers are just placeholders, and initialization using ten bits is successful:

```
byte data = 0b0000110011;   //successful
```

So, as was mentioned, a byte represents eight bits of binary data, so you would expect the following to compile, seeing that it initializes eight bits to a byte:

```
byte data = 0b10101010;
```

Sadly, this fails, because the byte data type only uses seven of its eight bits for holding data, while reserving the extra bit to indicate whether a number is positive or negative. That's why the range of the eight bit byte (2^8 =256) is -128 to 127, and not 0 to 255.

By the way, sometimes make the mistake of thinking the leading zero before the 'b' in byte notation (**0**b1100) indicates whether the number is positive or negative. That's not correct. You can toggle the positivity or negativity of a binary notation number the same way as you would with a deicmimal number: with a minus sign:

```
//byte data = 1b1010101;   not valid
  byte data = -0b1010101;  //valid
```

Now speaking of the problem of type mismatches and stuffing too many ones and zeroes into a data type, the short has the same issue as the byte, although with the short, you can initialize it to fifteen binary digits instead of eight:

```
  short number = 0b111111111111111; //valid
//short number = 0b1111111111111111; not valid
```

Inconsistent treatment of ints vs. shorts and bytes

So, the eight bit byte can take seven bits, the sixteen bit short can take fifteen bits, but for some reason, you can specify a full 32 binary bits for a 32-bit int:

```
int overflow = 0b10101010101010101010101010101011;
System.out.println(overflow);
```

Unfortunately, the assignment here does not work as nicely as planned. This number prints out as -1431655765, which is the negative value of the first 31 binary digits. The last digit of the binary number, which is a 1, is used to indicate that the number is negative. If the row of bits was literally translated into a number, without overflowing it into an int, it would actually have the value of 2863311531, not -1431655765.

Oh, and just so you know, there is a way to force the Java compiler to treat a binary number as a long - all you have to do is throw the letter 'L' at the end of the bits. So, take a look at what happens when you print out these two numbers that use binary notation:

```
System.out.println(0b10101010101010101010101010101011);  //-1431655765 using 32 signed bits
System.out.println(0b10101010101010101010101010101011L); //2863311531 using 32 unsigned bits
```

The latter example can also be useful when assigning long binary numbers to a long; it's utility goes far beyond printing out data to the console.

Isn't this int-eresting?

By default, the Java compiler likes to treat numbers as ints, but that can be a problem when you're defining numbers that fall into the exclusive range of a long. Take a look at the following code, which tries to initialize a long to a 33 bit value, a number that is well within the 64 bit range of a long, but far too large to be stuffed into a variable of type int:

```
long bow = 0b10101010101010101010101010101111; //causes a compile error
```

This line of code triggers the following compile error: The literal 0b10101010101010101010101010101111 of type int is out of range.

Of course, we know that this *should* work, so we can force the compiler to treat our binary number as a 64 bit long, again, by appending the letter 'L'.

```
//long bow = 0b10101010101010101010101010101111; fails to compile
  long bow = 0b10101010101010101010101010101111L; //compiles
```

The appended 'L' can be both upper or lowercase, but since the lower case letter can be easily confused with a one, it's recommended to stick with the upper case value.

Now, since it's fine and dandy to place the letter L after a variable of type long is being initialized, you'd probably expect that the same type of notation could be used for float and double types. Well, you'd be wrong. So, while the following four lines of code are all valid:

```
float f1 = 12f;
double d1 = 12d;

float f2 = 0b111;
double d2 = 0b111;
```

The following lines of code will actually blow up:

```
float f3 = 0b111f;
double d3 = 0b111d;
```

Using binary notation with an 'f' or a 'd' to denote that a number is a float or a double generates the following compile time error: **Syntax error on token "f", delete this token Syntax error on token "d", delete this token**

And that's about it. That's everything you need to know about binary notation in order to tackle the corresponding objective in the Oracle Certified Professional for Java 7 exam. Best of luck!

Learning Resources for the Java and Java 7 Certification

OCP Java SE 6 Programmer Practice Exams (Exam 310-065) (Certification Press)
OCP Java SE 7 Programmer Study Guide (Certification Press)
SCJP Sun Certified Programmer for Java 6 Exam 310-065
A Programmer's Guide to Java SCJP Certification: A Comprehensive Primer (3rd Edition)
SCJA Sun Certified Java Associate Study Guide for Test CX-310-019, 2nd Edition

Check out these other tutorials from TheServerSide's Sal Pece and Cameron McKenzie covering the new Java 7 features:

New Java 7 Features: Binary Notation and Literal Variable Initialization
New Java 7 Features: Numeric Underscores with Literals Tutorial
New Java 7 Features: Using String in the Switch Statement Tutorial
New Java 7 Features: The Try-with-resources Language Enhancement Tutorial
New Java 7 Features: Automatic Resource Management (ARM) and the AutoCloseable Interfact Tutorial
New Java 7 Features: Suppressed Exceptions and Try-with-resources Tutorial
Java 7 Mock Certification Exam: A Tricky OCPJP Question about ARM and Try-with-Resources
OCAJP Exam to Debuts in March 2010. OCPJP Released in June?
OCPJP & OCAJP Java 7 Exams: Oracle Drops the Training Requirement
OCAJP and OCPJP Changes for Java 7: New Objectives, a Format Change and a Price Hike

15 Dec 2011