Number Systems Tutorial

This brief tutorial is intended to serve as a basic introduction to number systems. As a programmer it is important to have a basic knowledge of different number systems in order to better understand what happens in a computer when certain types of operations are performed on data that is stored in memory.

In this tutorial, we introduce the four key number systems that Java programmers often use: decimal, binary, octal, and hexadecimal. We will first introduce the four number systems, and then discuss how to convert numbers between each of the systems.

The Decimal Number System

The decimal number system uses the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. A decimal number is represented by a sequence of these digits.

As humans, it is very natural for us to use the **decimal** system to express numbers. The decimal system expresses numbers built around the powers of 10…and we say that the *base* of the decimal number system is 10. We all find it very easy to understand the meaning of numbers like 32 and 123.

Let's take slightly more mathematical look at what it means to express numbers in the decimal system. The digits 0 through 9 can be used to express a number in the decimal system. Take the number 123 as an example. The number 123 is really:

$$1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0$$

Recall from basic mathematics that 10 to the second power is 10 times 10 or 100; 10 to the first power is 10; and 10 to the zero power is 1. The position of each digit gives that digit a particular value, which is referred to as its positional value

The Binary Number System

Computers don't use the decimal system when they process numbers and other information. They use zeros and ones instead. Because computers are composed of electronic switches that are either turned on or off, computers use the **binary** system to express numbers and programming instructions and basically any type of information that they need to process.

The binary system is built around powers of 2, and only the digits 0 and 1 can be used in this system. For example, the binary number 1011 is equivalent to:

Number Systems Tutorial

$$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

The binary number 1011 is equivalent to the decimal number 11, since 2 to the third power is 8, 2 to the first power is 2, and 2 to the zero power is 1.

Each digit in the binary system is called a bit. When a computer reserves storage for a numerical data type, it reserves a certain number of bits depending on the specific type of number. In Java, for example, there are four integer data types: byte, short, int, and long. Java stores byte types in 8 bits of memory, short types in 16 bits of memory, int types in 32 bits of memory, and long types in 64 bits of memory.

Let's take Java's byte type and figure out what range of values that type can take on. The same approach can be used for the other integer types, but illustrating the byte type will get us where we need to go. Since byte types are stored in 8 bits of memory, we'll use the following table in our discussion.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	0	1	0	1	1

This table illustrates the binary number 1011 which we have seen is equivalent to the decimal number 11. When computer programmers think of the series of 0's and 1's associated with a binary number, they refer to this series as the **bit pattern** for the number.

If all 8 bits were used to store a byte type then byte variables could take on values between 0 (binary 00000000) and 255 (binary 11111111). But, Java doesn't use all 8 bits. It uses 7 bits to store the value of a number and 1 bit to represent a number's sign. That way, both positive and negative numbers can be allowed. The high bit is used to represent the sign of a number. A value of 0 corresponds to a positive sign and a value of 1 corresponds to a negative sign.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	1	1	1	1	1	1

So, the largest positive number that can be represented in 7 bits of memory is illustrated in the above table. If you work through the math for this bit pattern you would find that it is equivalent to the decimal value 127.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	0	0	0	0	0	0

Number Systems Tutorial

The smallest negative number that can be represented in 7 bits of memory is represented by the above bit pattern. To deal with negative numbers, Java uses something called **two's complement notation**, which we won't get into here. So, looking at the two above bit patterns, byte variables in Java can take on integer values between -128 and 127.

Understanding bit patterns and what happens when they are manipulated will be important when we use Java's bitwise operators and type casting operations.

The Octal Number System

The **octal** number system uses the digits 0, 1, 2, 3, 4, 5, 6, and 7. An octal number is represented by a sequence of these digits. The base of the octal number system is 8, so octal numbers are expressed around powers of 8. For example, the octal number 123 is equivalent to:

$$1 \times 8^2 + 2 \times 8^1 + 3 \times 8^0$$

The octal number 123 is equivalent to the decimal number 83, since 8 to the second power is 64, 8 to the first power is 8, and 8 to the zero power is 1.

In Java, you can define an octal value for an integer variable by preceding the digits associated with the number with a zero. As an example, the following program defines an octal value for the variable *number*.

```
public class OctalNumberDemo
{
      public static void main( String [ ] args )
      {
            int number = 0123;   // assign octal number 123

            System.out.println( number );
      }
}
```

When the program runs, the value displayed will be 83…the decimal equivalent of the octal number 123.

Number Systems Tutorial

The Hexadecimal Number System

The **hexadecimal** number system, also called the **hex** number system, uses 16 digits, so its base is 16. The digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. The digits A-F correspond to the values 10, 11, 12, 13, 14, and 15, respectively. Thus, hexadecimal numbers can be expressed using only the numeric digits 0-9, as in 123, as well as combinations of letters and numeric digits, as in 8ABF, or by using only letters, as in FFC.

Since 16 is the base of the hexadecimal number system, hexadecimal numbers are expressed around powers of 16. As an example, the hexadecimal number 8ABF is equivalent to:

$$8 \times 16^3 + 10 \times 16^2 + 11 \times 16^1 + 15 \times 16^0$$

So, the hexadecimal number 8ABF is equivalent to the decimal number 35,519 since 16 to the third power is 4,096, 16 to the second power is 256, 16 to the first power is 16, and 16 to the zero power is 1.

In Java, you can define a hexadecimal value for an integer variable by preceding the digits associated with the number with a *0X* or *0x*. As an example, the following program defines a hexadecimal value for the integer variable *number*.

```
public class HexNumberDemo
{
      public static void main( String [ ] args )
      {
         int number = 0x8ABF;   // assign hex number 8ABF

         System.out.println( number );
      }
}
```

When the program runs, the value displayed will be 35519…the decimal equivalent of the hex number 8ABF.

Converting Between Binary & Decimal Numbers

Given a binary number $b_n b_{n-1} \ldots b_1 b_o$, the equivalent decimal value is

$$b_n \times 2^n + b_{n-1} \times 2^{n-1} + \ldots + b_1 \times 2^1 + b_0 \times 2^0$$

For example, the binary number 1101 is equivalent to the decimal number

$$1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13$$

Number Systems Tutorial

To convert a decimal number, *d*, to its equivalent binary number, you must find the bits $b_n, b_{n-1}, \ldots, b_1, b_0$ such that

$$d = b_n \times 2^n + b_{n-1} \times 2^{n-1} + \ldots + b_1 \times 2^1 + b_0 \times 2^0$$

This can be done by successively dividing *d* by 2 until the quotient is zero. For example, the binary equivalent of decimal 13 is 1101. This can be determined by the series of successive divisions below.

0	1	3	6
$2\sqrt{1}$	$2\sqrt{3}$	$2\sqrt{6}$	$2\sqrt{13}$
-0	-2	-6	-12
1	1	0	1

Converting Between Octal & Decimal Numbers

Given an octal number $o_n o_{n-1} \ldots o_1 o_o$, the equivalent decimal value is

$$o_n \times 8^n + o_{n-1} \times 8^{n-1} + \ldots + o_1 \times 8^1 + o_0 \times 8^0$$

For example, the decimal equivalent of the octal number 123 is

$$1 \times 8^2 + 2 \times 8^1 + 3 \times 8^0 = 83$$

To convert a decimal number, *d*, to its equivalent octal number, you must find the hexadecimal digits $o_n, o_{n-1}, \ldots, o_1, o_0$ such that

$$d = o_n \times 8^n + o_{n-1} \times 8^{n-1} + \ldots + o_1 \times 8^1 + o_0 \times 8^0$$

This can be done by successively dividing *d* by 8 until the quotient is zero. For example, the octal equivalent of decimal 83 is 123. This can be determined by the series of successive divisions below.

0	1	10
$8\sqrt{1}$	$8\sqrt{10}$	$8\sqrt{83}$
-0	- 8	-80
1	2	3

Number Systems Tutorial

Converting Between Hexadecimal & Decimal Numbers

Given a hexadecimal number $h_n h_{n-1} \ldots h_1 h_o$, the equivalent decimal value is

$$h_n \times 16^n + h_{n-1} \times 16^{n-1} + \ldots + h_1 \times 16^1 + h_0 \times 16^0$$

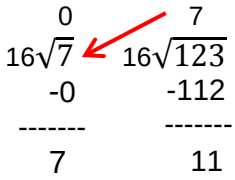
For example, the hexadecimal number 8ABF is equivalent to the decimal number

$$8 \times 16^3 + 10 \times 16^2 + 11 \times 16^1 + 15 \times 16^0 = 35{,}519$$

To convert a decimal number, *d*, to its equivalent hexadecimal number, you must find the digits $h_n, h_{n-1}, \ldots, h_1$, b_0 such that

$$d = h_n \times 16^n + h_{n-1} \times 16^{n-1} + \ldots + h_1 \times 16^1 + h_0 \times 16^0$$

This can be done by successively dividing *d* by 16 until the quotient is zero. For example, the hexadecimal equivalent of decimal 123 is 7B. This can be determined by the series of successive divisions below.



The remainder of 11, expressed as a hexadecimal digit, is B, so the hexadecimal equivalent of decimal 123 is 7B.

Converting Between Binary & Hexadecimal Numbers

To convert a hexadecimal number to its binary equivalent, convert each hexadecimal digit to its four-digit binary equivalent.

For example, the hex number 7B is equivalent to the binary number 1111011. The hex digit 7 has a binary equivalent of 111 ($1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$) and the hex digit B has a binary equivalent of 1011 ($1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$).

To convert a binary number to its hexadecimal equivalent, convert every four digits of the binary number, right-to-left, into its equivalent hexadecimal digit.

Number Systems Tutorial

For example, the binary number 1111011 is equivalent to the hexadecimal number 7B. First, split the binary number, right-to-left, into 4-digit groups, like this: 111|1011. The right-most binary number is decimal 11, which is hexadecimal B. The next binary number is 7. So, 7B is the hex equivalent of 1111011.

Converting Between Binary & Octal Numbers

To convert an octal number to its binary equivalent, convert each octal digit to its three-digit binary equivalent.

For example, the octal number 321 has a binary equivalent of 11010001. The octal digit 3 has a three-digit binary equivalent of 011 ($0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$). The octal digit 2 has a binary equivalent 010 ($0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$), and the octal digit 1 has a binary equivalent 001 ($0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$).

To convert a binary number to its octal equivalent, convert every three digits of the binary number, right-to-left, into its equivalent octal digit.

For example, the binary number 11010001 is equivalent to the octal number 321. First, split the binary number, right-to-left, into 3-digit groups, like this: 11|010|001. The octal equivalent of 11 is 3, the octal equivalent of 010 is 2, and the octal equivalent of 001 is 1. So, 321 is the octal equivalent of 11010001.