Avoid magic numbers

We saw a method with this in it:

if (c > 64 **&&** c < 91) …

Good heavens! What do those magic numbers 64 and 91 mean? Don't *ever* write a program with such mysterious looking, magic numbers. Readers will wonder what they mean —and you will too, three weeks after you write them.

It turns out that c was a **char** variable, that 65 is the Unicode representation of character 'A', and that 90 is the Unicode representation of character 'Z'. Therefore, the programmer should have written it this way:

if ('A' <= c **&&** c <= 'Z') …

What a difference! Now it's clear to everyone that the condition is true if c is a conventional capital letter.

What is a magic number?

One definition of *magic numbers* on Wikipedia is:

Unique values with unexplained meaning or multiple occurrences
which could (preferably) be replaced with named constants

and that's just what the numbers 64 and 91 were. We replaced by constants.

Other magic numbers

Suppose the program you are writing requires a field that is one of the main operating systems:

int os; // 1 is IOS, 2 is Unix, 3 is Android, 4 is Windows

That will result in code with the magic number number 2, for example:

if (os == 2) …

You could declare names along with the declaration of os:

public static final int IOS= 1;
public static final int Unix= 2;
public static final int Android= 3;
public static final int Windows= 4;

But that doesn't *force* you to use them, and you might still have magic numbers in your program.

Instead of that, declare an **enum** like this:

enum OS {IOS, Unix, Android, Windows}

and you can then write

OS os;

…

if (os == OS.Unix) …

Enums are preferred because there is then *no possibility* of having magic numbers.

Discussion

Don't think you can get by with having magic numbers in your program but always commenting what they mean. You will forget to put in comments, and it's easy to make mistakes. Instead, keep magic numbers out of your program entirely.