Cup Size -- a story about variables

The Coffee Corral coffeehouse at the Ranch is famous around these parts for its unique cup collection. If you really need to understand the way values and objects are used in Java, you gotta start here.

I like cups. Big cups, little cups. Painted tea cups, huge cappuccino bowls, cups with logos from the UCLA coffeehouse. Cups I painted myself at the "All Fired Up" pottery shop. Cups with curvy, sexy handles. Metallic cups that I now know must never, ever go in the microwave.

So when I think of variables, I naturally think of... cups.

A variable is just a cup. It has a size, and it can hold something.

In Java, cups come in two main styles: primitive and reference.

Primitive cups hold primitive values.

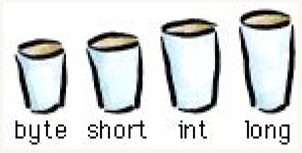
Reference cups hold remote controls to objects.

We'll start with primitives. Primitive cups are like the cups they have at the coffeehouse. If you're familiar with Starbucks' you know what I mean. They come in different sizes, and each size has a name like short, tall, grande. As in, "I'd like a grande mocha java with extra whipped cream. Oh, and use non-fat milk please")

Our coffeehouse has a picture of the cups on the counter, so customers know what to order. It looks like this:

short
single
grande
tall

In Java, integer primitives comes in different sizes, and those sizes have names. They look like this:



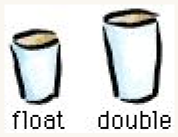
These cups hold a value.

So instead of saying, "I'd like a tall French Roast", you say to the compiler, "I'd like an int with the number 90 please." And that's what you get. (you also have to give your cup a name, but we'll get to that later.)

The number 90 is dropped into your int-sized cup.

But what about floating point numbers? (the things with decimal points)

They get their own cups too.



And there's another cup for booleans, that can store values of true or false. And a cup for chars, that store single characters like the letter 'c' or 'z'.

In Java, **each of these cups (float, char, long, etc.) is a specific size**. Byte is the smallest, double and long are the largest. Rather than measure in milliliters (or ounces as we do in the US) Java variables have a size measured in bits:

byte - 8 bits

short - 16 bits

int - 32 bits

long - 64 bits

All of these integer types are SIGNED. The leftmost bit represents the sign (positive or negative) and is NOT part of the value. So with a byte, for instance, you don't get the whole 8 bits to represent your value. You get 7. This gives you a range, for bytes, of :

(-2 to the 7th) through (2 to the 7th) -1. Why that little -1 on the end? Because zero is in there, and zero counts as negative. Works the same way with the others.

float - 32 bits

double - 64 bits

Floating point numbers are in the IEEE 754 standard. If that means anything to you, great. If it doesn't, well, then, you'll just have to struggle through the long technical dissertation on floating point numbers which I feel compelled to insert here. What the heck, I'll skip it.

We rejoin our primitive variables, already in progress.

char - 16 bits, UNSIGNED

(Unicode format -- for English, it maps perfectly to ASCII with the high 8-bits just hanging out as a bunch of zeros)

boolean - hmmmm... you're not supposed to ask. It holds a value of true or false, but it's really stored as a numeric value, probably in a byte-sized cup. Try not to think about the size; all you know is that it holds a boolean.

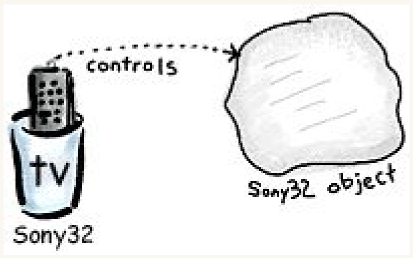
Let's get to the Really Interesting Cups... **REFERENCES**

In Java, if you want to stick an object in a variable, remember that the object is created out on the garbage-collectible heap. Always. So it's not IN the variable. There aren't giant, expandable cups which can be made big enough to hold any object. And unlike C/C++, there aren't cups which hold the exact memory location of the object.

In Java, **objects are created on the heap**, and a REFERENCE to the object is stored in the cup. Think of it as a remote control to a specific type of object.

At the CoffeeCorral, a customer can ask for a remote control to a TV They ask for it like this:

"I'd like a reference to a new Sony32 television please, and name it TV." which in Java looks like:



But notice the NAME written on the cup. **The cups have unique names**. Imagine if you had a pile of remote controls, and nobody knew which one controlled which TV. So we make people put names on their remote control cups. But the name is not on the object -- the object has a unique ID, like a serial number, but that isn't the same as what you name the reference! The reference name (the name on the cup) is YOUR choice.

It's the same for primitives -- you name the cups like this:

"I'd like a byte with the number 7, and name it x please".

For object references, you can also ask for a self-serve remote control... a remote control where you don't have a television object picked out yet. You still get the remote in the cup, but you "program" that remote for a specific television later.

In Java you say:

`Sony32 tv;` // declare but don't initialize with an actual Sony32 object.

value is null
(not controlling
any tv/object)
tv
Sony32

It's like saying, "I'd like a reference to a Sony32 television, and name the remote 'tv', but I'll pick out the actual television later.". You get the cup, you get the remote, but there's no object and the remote isn't controlling ANYTHING.

In Java, a remote which refers to nothing is a reference with a value of null.

So what's REALLY inside that cup? What's a remote control?

In Java, remote controls are called references. They store a value which the Java Virtual Machine (JVM) uses to get to your object. It sure looks and feels a lot like a pointer, and it might very well be a pointer to a pointer, or...

You Can't Know. It's an implementation detail that you, as a programmer, can't access. Don't even think about it. There's no way to use that value other than to access the methods and variables of the actual object the reference refers to. That's part of what makes Java safer than C/C++. You can not go directly to any arbitrary memory location. The JVM allocates memory on your behalf, for your object, and stores an address-like thing in the reference cup (which is most likely a 32-bit cup, but not guaranteed to be).

If this depresses you, take a deep breath, have a nice big cup of hot tea and get over it. When you start doing CORBA you'll be grateful beyond belief that you don't have to take care of the memory.