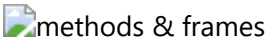
We've looked at how local variables are stored in frames but there is more to it than that. Frames also store a number of addresses to ensure the code runs properly.

The actual code is stored in the **bytecode** area of memory. This is the compiled code, from start to finish. Its structured like your code is structured, with different sections for each method. When a method is called the code jumps from where it is in the bytecode to the location where the method is and then continues running from there. Once the method is completed, it returns to the position (in the bytecode) that it came from. How does it know where to go to? Simple, we store the address we came from in the frame. Similarly when we create a new frame we store the address of the previous frame so we know where in the stack to go to when we complete the method and delete its frame.

 methods & frames

The image above shows a simplified example of this. We have a program with a main() and a function(). At some point in main() we call function(). We create the frame for function() as normal and we store the parameter, the local variable, the address for the main() frame and the address in the bytecode we jumped from.

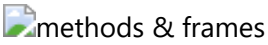
Note also that we store the address on the stack of the variable in main() we want to return a value to.

The Stack Grows Downwards, The Frame Is Generated Downwards but is filled upwards

The frame is placed on the stack as high as possible. An area of the stack is set aside that is big enough to store all the required data. The required data is stored, one by one, from the top of the frame but the data itself is still stored normally, going from low to high.

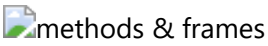
Think of memory as a long flat line of cells, going from 0 on the left to the highest memory address on the right. If we want to store a four byte integer in the frame we start at the last filled spot, move down 32 cells and store the integer starting at that point and filling in those 32 cells so it ends just before the previously lowest used cell.

We cal think of the frame as looking something like this.

 methods & frames

An array example

Lets imagine our method has a local array of integers, what does the frame look like.

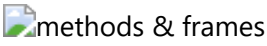
 methods & frames

Here we can clearly see that space has been set aside for the array by moving down from the last item in the from (where we are storing the address to return the value to) but the array itself "grows up" from its starting point.

A potential problem

There is a potential problem though. What happens if we try to store too much data in a variable? Java does its best to prevent us doing this but its not impossible. C is perfectly happy for us to store more items in an

array than will fit in it. C simply puts the data in the spaces after the array. This *overrun* can and will crash programmes.


methods & frames

In the example above, the address the method is supposed to return a value to has been changed to address 42. Whatever is in that location will be over-written. The effects of this are, well, unpredictable but seldom good.

Done accidentally this leads to unstable software. Done maliciously the "resume code from" address can be deliberately changed so the programme executes code it was never supposed to.

There are protections in place to prevent this happening but no protection is foolproof.