Recursion

"To understand recursion, you must first understand recursion"
$\sim$ Every CS prof ever

Recursion

- Consider the following:
  - GNU's Not Unix
  - The TTP Project
  - WINE Is Not An Emulator
  - PIP Installs Packages
- Each acronym's definition contains the acronym
- Recursion: Repeating elements in a self similar way
  - Like Russian Stacking Dolls
- Most definitions for Recursion use the words recursion, reursive,... so we will work wih examples

Recursion Example

Problem

Calculate a^b, where b is a positive integer

We know that:

- $a^0 = 1$
  - Stopping Condition
- $a^b = a * a^{b-1}$
  - Note that we define the power function with a call to the power function

myPow code

```
1  public static int myPow(int a, int b)
2  {
3    if (b == 0) // BASE CASE
4      return 1;
5    return a * myPow(a, b-1);
6    // RECURSIVE CALL
7  }
8
9  public static void main(String [] args)
10 {
11   myPow(2, 4);
12 }
```

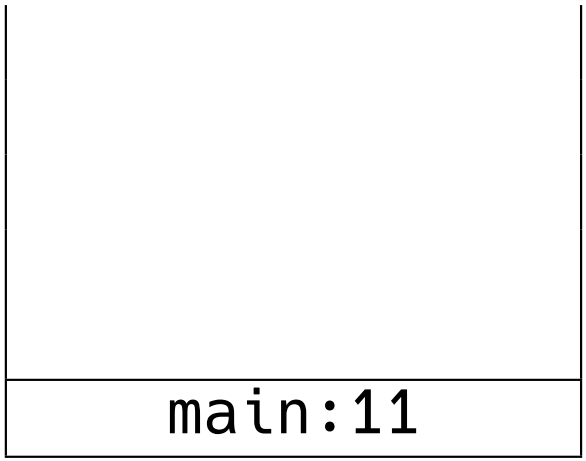
Base cases are important to ensuring you do not attempt to infinitely recurse!

How Does It Work?

- Whenever you enter a method, Java saves where you are by pushing an Activation Record to the Call Stack
- Every time you hit a `return` statement, Java pops the top element off the call stack and sends the resulting value back to the next location on the call stack
- When the call stack is empty, the program has finished executing

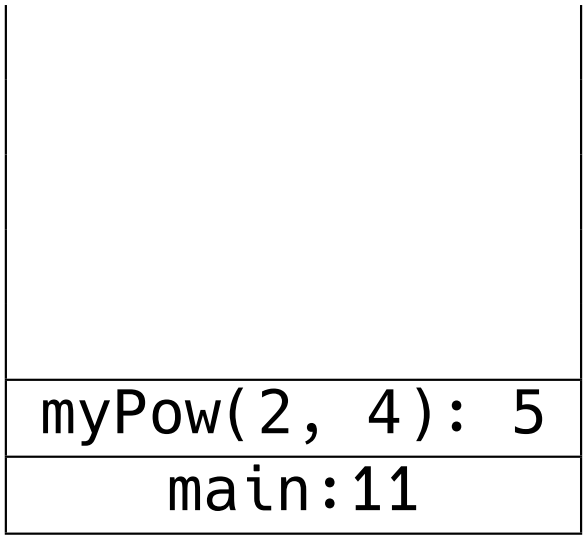
Call Stack Example

When the program starts, Java pushes an activation record for `main` and keeps track of the location (which will be represented by line numbers)
At line 11, we hit a method call



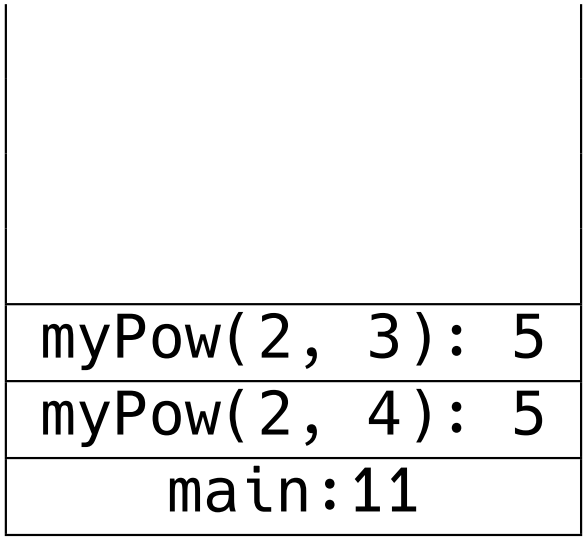
Call Stack Example

`myPow(2, 4)` runs until it reaches another method call on line 5 (and calls `myPow(2, 3)`)



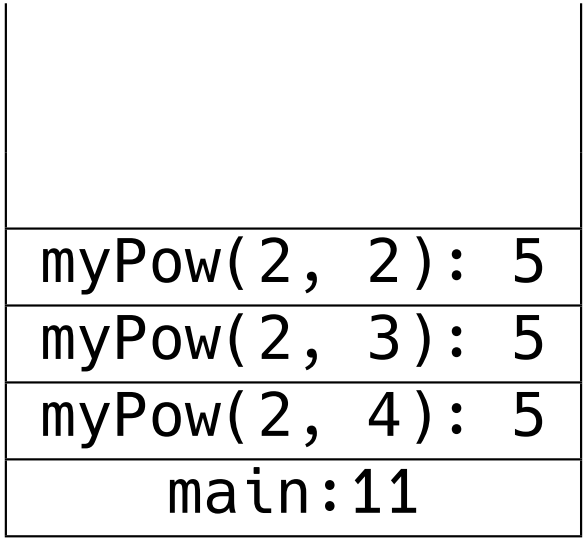
Call Stack Example

`myPow(2, 3)` runs until it reaches another method call on line 5 (and calls `myPow(2, 2)`)



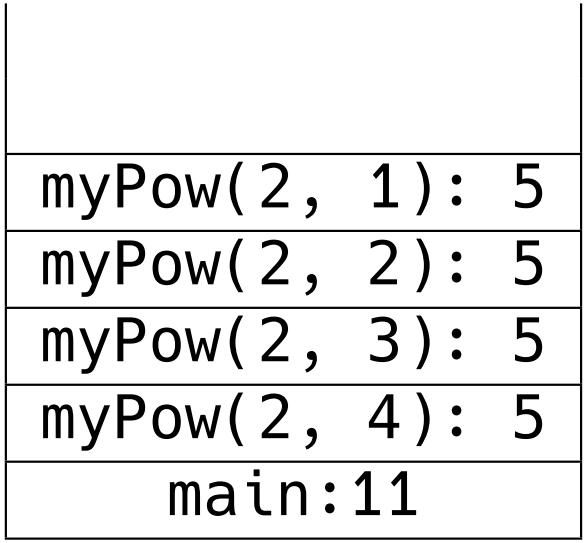
Call Stack Example

`myPow(2, 2)` runs until it reaches another method call on line 5 (and calls `myPow(2, 1)`)



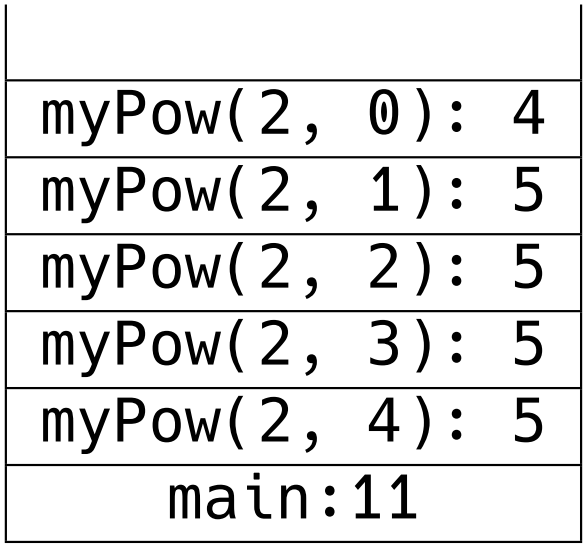
Call Stack Example

`myPow(2, 1)` runs until it reaches another method call on line 5 (and calls `myPow(2, 0)`)



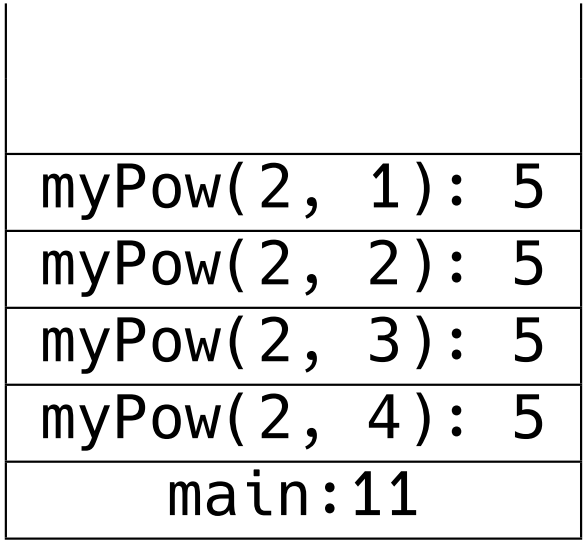
Call Stack Example

`myPow(2, 0)` satisfies the condition on line 3, so no longer recurses. It sends the value `1` back to `myPow(2, 1)` on line 5



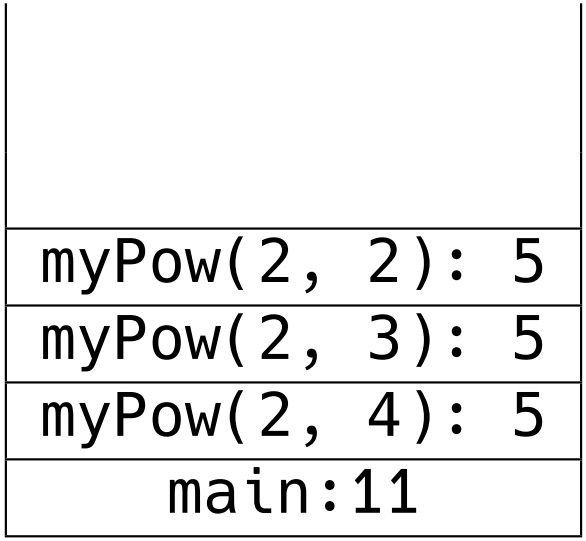
Call Stack Example

There are no more recursive calls, so `myPow(2,1)` sends the value `2 * 1 = 2` back to `myPow(2, 2)` on line 5



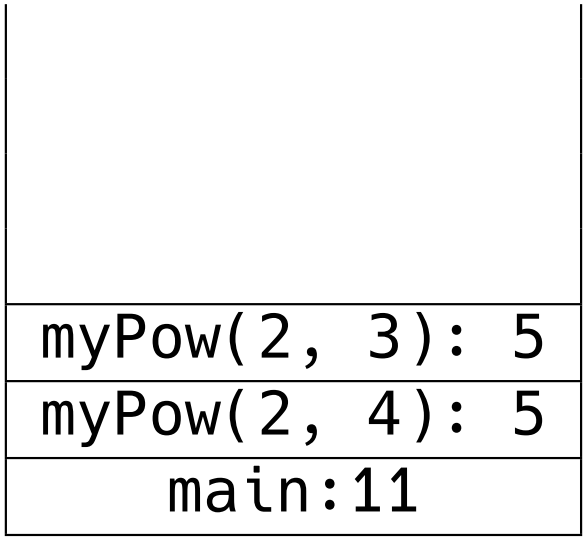
Call Stack Example

There are no more recursive calls, so `myPow(2,2)` sends the value `2 * 2 = 4` back to `myPow(2, 3)` on line 5



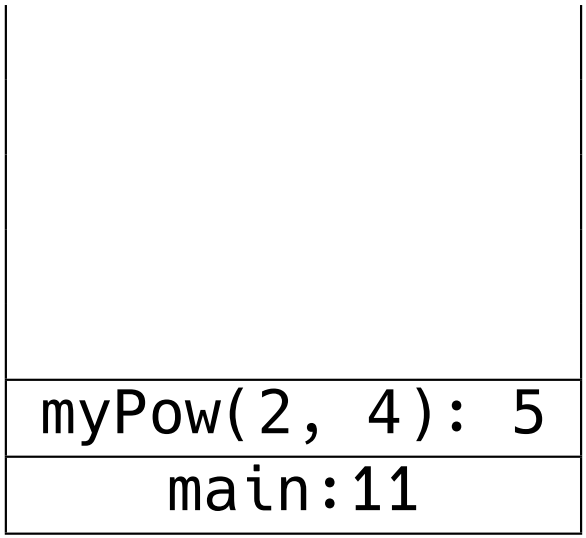
Call Stack Example

There are no more recursive calls, so `myPow(2,3)` sends the value `2 * 4 = 8` back to `myPow(2, 4)` on line 5



Call Stack Example

There are no more recursive calls, so `myPow(2,4)` sends the value `2 * 8 = 16` back to `main` on line 5



Call Stack Example

As the main method completes, it will pop the last element off the call stack; program execution is completed!