Understanding and developing recursive methods

Base cases and recursive cases

Consider again the definition of the nonnegative powers of 2:

$2^0 = 1$	base case
$2^k = 2*2^{k-1}$ for $k > 0$	recursive case

This definition has a *base case*: a case in which the answer is given without recursion. $2^0 = 1$.

This definition has a *recursive case*: a case in which recursion is used. $2^k = 2*2^{k-1}$. Further, note that 2^k is defined in terms of 2^{k-1}; The exponent `k-1` in the recursive definition is smaller than the exponent `k` in the thing being defined. This is what allows us to conclude that the definition is well-defined.

In the same way, a recursive method will have one or more base cases and one or more recursive cases. And, in each recursive case, the arguments of the recursive call are in some sense smaller than the parameters of the recursive method; this is what allows us to conclude that the recursion terminates.

The recursive function to the right has the base case `k = 0`, the recursive case `k > 0`, and in the recursive case the argument `k-1` of the recursive call is smaller than parameter `k`.

```
/** = 2^k.
  * Precondition k >= 0. */
public static int pow(int k) {
   if (k == 0) return 1;
   return 2 * pow(k-1);
}
```

In the

Two recursive functions

Above, we gave a definition of the nonnegative powers of 2. Such a mathematical definition can be transformed easily, almost automatically, into a Java function, as shown to the right. You can write many recursive mathematical definitions as Java functions in this fashion.

To the right below is another recursive function, which returns the number of digits in the decimal representation of a nonnegative integer `n`.

If `n < 10`, the answer is 1 (even if `n` is 0).

The comments in the function tell you that for `n` $\geq$ 10, the answer is 1 plus the number of digits in `n /10`. That's the value that the return statement returns. We can calculate the call numDigits(352):

```
  numDigits(352)
=    <with n = 352, the value of numDigits(n/10) + 1 is returned>
  numDigits(35) + 1
=   <with n = 35, the value of numDigits(n/10) + 1 is returned>
  numDigits(3) + 1 + 1
=   <with n = 3, the value 1 is returned>
  1 + 1 + 1
=   <arithmetic>
  3
```

Two ways to think about recursive methods

With a recursive method, we have two different questions, with two totally different answers:

1. How is a call on a recursive method *executed* —how could it possible work if it calls itself?
2. How do we *understand* a recursive method, and how do we *write/develop* a recursive method?

Our first task is to show you how recursive calls are executed, so that you can then know that they work. After that, we can study how to understand a recursive method and how to go about developing one.

For the first task, we ask that you visit the JavaHyperText and click the link

`Explain constructs / 3. Method calls`

in the top horizontal navigation bar. This tutorial covers these topics:

1. The frame for a method call.

2. Two data structures: the queue and the stack.
3. The call stack and the algorithm for executing *any* method call.
4. Verification that the process for executing a method call works for recursive calls.

For the second task, look at the appropriate item in entry "recursion" in the JavaHyperText.