Recursion: If you get the point, stop; otherwise, see Recursion.

Infinite recursion: See Infinite recursion.

Read: pp. 403-408 but SKIP sect. 15.1.2

ProgramLive CD, page 15-3, has interesting recursive methods.
Download presented algorithms from the website

Recursive definition: A definition that is defined in terms of itself.

Recursive method: a method that calls itself (directly or indirectly).
Recursion is often a good alternative to iteration (loops), which we cover later. Recursion is an important programming tool. Purely functional languages have no loops — only recursion.

Two issues in coming to grips with recursion

1. How are recursive calls executed?

2. How do we understand a recursive method and how do we create one?

We discussed the first issue earlier. If you execute a call on a recursive method carefully, using our model of execution, you will see that it works. Briefly, a new frame is created for each recursive call. We do this in the next lecture.

DON'T try to understand a recursive method by executing its recursive calls! Use execution only to understand how it works.

Step 1: HAVE A PRECISE SPECIFICATION

// = number of 'e's in s
public static int noe(String s) {
    if (s.length() == 0) {
        return 0; // base case
    } // {s has at least one character}
    int result = (s[0] == 'e' ? 1 : 0) + noe(s.substring(1));
    return result; // recursive case (has a recursive call)
} // {s has at least one character}

Step 2: Check the base case.
When s is the empty string, 0 is returned. So the base case is handled correctly.

Understanding a recursive method

Notation:
- s[i..j] shorthand for s.substring(i, j).
- s[i] shorthand for s.charAt(i).

/** = the number of 'e's in s */
public String noe(String s) {
    if (s.length() == 0) {
        return ""; // the empty string
    } // {s has at least one char}
    // return (s[0] = 'e' ? 1 : 0) + number of 'e's in s[1..];
    String result = (s[0] == 'e' ? 1 : 0) + noe(s.substring(1));
    return result; // recursive case (has a recursive call)
} // {s has at least one character}

Express the answer with the same terminology as the specification, but on a smaller scale:

number of 'e's in s = (if s[0] = 'e' then 1 else 0) + number of 'e's in s[1..]
Step 4: Recursive case is correct.

Creating a recursive method

Task: Write a method that removes blanks from a String.

0. Specification:
/** = s but with its blanks removed */
public static String deblank(String s) {

1. Base case: the smallest String s is "."
   if (s.length() == 0)
     return s;

2. Other cases: String s has at least 1 character.
   return s.substring(1); // with its blanks removed

The tasks given by the two English, blue expressions are similar to the
task fulfilled by this function, but on a smaller String! Rewrite each as
deblank(s[1..]).

Check the four points:
0. Precise specification?
1. Base case: correct?
2. Recursive case: progress toward termination?
3. Recursive case: correct?

A man, a plan, a canal, a maniac.

Check palindrome-hood

A String with at least two characters is a palindrome if
(0) its first and last characters are equal and
(1) chars between first and last form a palindrome:

A M A N A P L A N A C A N A L P A N A M A

/** = "s is a palindrome " */
public static boolean isPal(String s) {
  if (s.length() <= 1)
    return true;
  // ( s has at least two characters )
  return s.charAt(0) == s.charAt(s.length()-1) &&
  isPal(s.substring(1, s.length()-1));
}
Tiling Elaine’s kitchen

Elaine has a 2^n by 2^n kitchen. One square of it is covered by a 1 by 1 refrigerator. Tile the kitchen with these kinds of tiles:

```java
/** tile a 2^n by 2^n kitchen with 1 square filled. */
public static void tile(int n) {
    if (                             )
        }
```