Two views of recursive methods

☐ How are calls on recursive methods executed? We saw that in the lecture. Useful for understanding two recursion works.

☐ How do we understand a recursive method — know that it satisfies its specification? How do we write a recursive method? This requires a totally different approach. Thinking about how the method gets executed will confuse you completely! We now introduce this approach.

Understanding a recursive method

Step 1. Have a precise spec!

Step 2. Check that the method works in the base case(s).

Step 3. Look at the recursive case(s). In your mind replace each recursive call by what it does according to the method spec and verify that the correct result is then obtained.

```java
/** = sum of digits of n. * Precondition: n >= 0 */
public static int sum(int n) {
    if (n < 10) return n;
    // n has at least two digits
    return sum(n/10) + n%10 ;
}
```

Step 4. (No infinite recursion) Make sure that arguments to recursive calls are in some sense smaller than the parameters of the method.

n/10 < n
Understanding a recursive method

Step 1. Have a precise spec! Important! Can’t do step 3 without it
Step 2. Check that the method works in the base case(s).
Step 3. Look at the recursive case(s). In your mind replace each recursive call by what it does according to the spec and verify correctness.
Step 4. (No infinite recursion) Make sure that arguments to recursive calls are in some sense smaller than the parameters of the method.

Writing a recursive method

Step 1. Have a precise spec!
Step 2. Write the base case(s): Cases in which no recursive calls are needed Generally for “small” values of the parameters.
Step 3. Look at all other cases. See how to define these cases in terms of smaller problems of the same kind. Then implement those definitions using recursive calls for those smaller problems of the same kind. Done suitably point 4 is automatically satisfied.
Step 4. (No infinite recursion) Make sure that the args of recursive calls are in some sense smaller than the pars of the method.

Example: Palindromes

A palindrome is a String that reads the same backward and forward. A String with at least two characters is a palindrome if:
- (0) its first and last characters are equal and
- (1) chars between first & last form a palindrome:
  e.g. AMANAPLANACANALPANAMA
  A recursive definition!

Example: Counting characters

```java
/**
 * countEm(c,s) = number of times c occurs in s */
/** public static int countEm(char c, String s) */
if (s.length() == 0) return 0;
if (s.charAt(0) != c) return countEm(c, s.substring(1));
return 1 + countEm(c, s.substring(1));
```

Example: Exponentiation

Power computation:
- \(a^0 = 1\)
- \(a^n = a \cdot a^{n-1}\)

```java
/** power(a,n) returns a^n */
static int power(int a, int n) {
  if (n == 0) return 1;
  return a * power(a, n-1)
}
```

Example: Fast Exponentiation

Power computation:
- \(a^0 = 1\)
- \(a^n = a^{n/2} \cdot a^{n/2}\) if \(n\) is even
- \(a^n = a \cdot a^{n-1}\) if \(n\) is odd

```java
/** power(a,n) computes a^n */
static int power(int a, int n) {
  if (n == 0) return 1;
  if (n%2 == 0) return power(a*a, n/2);
  return a * power(a, n-1);
}
```
Example: Searching in a list

```java
/** Returns an index i between fst and lst, inclusive, where a[i] = k. 
* Precondition: such an index exists, 
* and a is sorted in ascending order 
*/
static int search(int[] a, int k, int fst, int lst) {
...
}
```

Example: Sierpinski’s Triangle

```java
/** draws Sierpinski’s triangle bounded by p1, p2, and p3 on g up to order n 
* Precondition: pi not null, n >= 0 */
private static void displayTriangles(Graphics g, int order, Point p1, Point p2, Point p3) {
...
}
```