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.

Understanding a recursive function

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 }
    return (s[0] = ‘e’ ? 1 : 0) + noe(s.substring(1));
}

Step 2: Check the base case.

When s is the empty string, 0 is returned. So the base case is handled correctly.

Step 3: Recursive calls make progress toward termination.

// = 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 }
    return (s[0] = ‘e’ ? 1 : 0) + noe(s.substring(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[0] == ‘ ’ ? “” : “” + s[1..] with its blanks removed


// = s but with its blanks removed
public static String deblank(String s) {
    if (s.length() == 0) return s;
    // {s is not empty}
    if (s[0] is a blank) return s[1..] with its blanks removed
    // {s is not empty and s[0] is not a blank}
    return s.charAt(0) + deblank(s.substring(1));
}

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..]).

// = s but with its blanks removed
public static String deblank(String s) {
    if (s.isEmpty()) return s;
    // {s is not empty}
    if (s.charAt(0) is a blank) return deblank(s.substring(1));
    // {s is not empty and s[0] is not a blank}
    return s.charAt(0) + deblank(s.substring(1));
}

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

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 & last form a palindrome:
e.g. AMANAPLANACANALPANA
/** = "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));
}

A man, a plan, a caret, a ban, a myriad, a sum, a lac, a liar, a hoop, a piet, a catalpa, a gas, an
oil, a bed, a yell, a rat, a cow, a pan, a wag, a tax, a ray, a ram, a cap, a yam, a gey, a hue, a
wall, a car, a layer, a ward, a bin, a woman, a vassal, a wolf, a luna, a mel, a pull, a fett, a wait,
a boy, a dahl, a fan, a cab, a datum, a gall, a hat, a fog, a zap, a say, a jow, a lay, a wet, a
galleon, a tug, a test, a trap, a tram, a top, a test, a toll, a ball, a fan, a sax, a mimem, a tenor, a
ten, a blass, a passer, a capital, a rapt, an amen, a toil, a cabal, a tang, a sun, an aza,
a move, a sag, a jamb, a daff, a sub, a salt, an atom, a soul, an ad, a woid, a medieval, a room, a
wood, a rip, a tal, a patriach, a revolt, a reel, a pull, a pool, a play, a pin, apeek, a parabolo, a
dog, a pet, a cad, a me, a fan, a pal, a rem, an etu, an el, an el, a batik, a mpg, a met, a
tap, a maxim, a mood, a leek, a ghub, a gld, a stuff, a citalid, a total, a credit, a tap, a
vag, a rat, a manor, a bar, a gal, a cola, a pap, a yow, a tab, a roj, a gah, a nag, a pagan, a bag,
a jar, a hat, a way, a popa, a local, a gar, a bamboo, a mat, a rag, a gap, a tar, a decal, a tot, a
col, a liu, a bandi, a lev, a boy, a bury, a kiel, a dooms, a rem, a map, an atom, a gem, a kil, a
baleen, a gala, a ten, a don, a mural, a pan, a faun, a decal, a pugoda, a lob, a rap, a keep, a
zip, a gulp, a hoop, a steer, a lore, a livin, a hair, a pad, a tap, a door, a mire, an aid, a caed,
a wod, an alias, an ox, an alias, a bus, a madman, a jag, a sow, a mass, an amus, a great, a lab, a
cadet, an em, a natural, a tip, a caros, a pan, a banoset, a minimum, a sani, a fall, a battalion, a
keel, a pot, a rip, a curvel, a matt, a part, a test, a gal, a pull, a gateway, a lion, a joy, a rap, a
rug, a flat, a ball, a gurnat, a dob, a can, a tube, a day, a butt, a waterfall, a punita, a not, a
flow, a lass, a van, a mob, a rib, a dur, a regular, a call, a war, a sky, a jam, a yap, a cam, a
ray, an as, a tag, a wax, a paw, a cut, a valley, a deco, a lion, a sofa, a plat, a catusip, a proob, a
rail, a columnos, a dairyman, a bier, a canul --- Panama!

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:

/** tile a 2^n by 2^n kitchen. */
public static void tile(int n) {
    if (n <= 1) return;
    // { n > 1 }
    tile(n/2);
    // { n/2 squares have been tiled }
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (i%2 == j%2) draw;

    // { n squares have been tiled }
}