Announcements

Easy drop date: Monday at 5
Exams back today..
A “mini-project” up today

For Tuesday: read chapter 8
Recursion

A very general problem-solving technique

Somewhat unusual
The Stamp problem

Suppose you have a sheet of 10 37¢ stamps and 10 4¢ stamps, and you want to put exactly $2.21 on an envelope, using the stamps available. Can you do it?
A seat-of-the-pants way to proceed:
Place a 37¢ stamp on the envelope, and take it from there: given the remaining 9 37¢ stamps, 10 4¢ stamps
Can you solve the new problem:
Make up $1.84 with the remaining stamps

This is a problem transformation:
(10,10,221) \rightarrow (9,10,184)
We'll actually be a little more systematic, and allow two possible transformations:

\[(10,10,221) \rightarrow (9,10,184) \text{ or } (10,9,217)\]

What does this mean??
static boolean stampCheck(int s37, int s4, int t) {
    if ((s37 < 0) || (s4 < 0) || (t < 0))
        return false;
    if (t == 0) return true;
    else
        return ((stampCheck(s37 - 1, s4, t - 37)) ||
                (stampCheck(s37, s4 - 1, t - 4)));
public static boolean stampCheck(int s37, int s4, int t) {  
    if ((s37 < 0) || (s4 < 0) || (t < 0))  
        return false;  
    if (t == 0) return true;  
    else  
        return ((stampCheck(s37 - 1, s4, t - 37)) ||  
                (stampCheck(s37, s4 - 1, t - 4)));  
}
Three significant aspects of a recursive method:

• It calls itself (this could be indirect)
• It has base cases - non-recursive ways out
• All recursive paths lead to a base case
public static int fac(int n){
    if (n <= 0)
        return 1;
    else return n*(fac (n-1));
}

fac(9) transformed into 9 * fac(8), which becomes 9*8*fac(7), which becomes ...
Recall the String method substring:

> String s = "abcde";
> s.substring(1,3)
"bc"
> s.substring(2)
"cde"
> s.substring(1)
"bcdex"

➤ That is: s.substring(a,b) goes from char at position a up to but not including char at b;
➤ s.substring(d) goes from pos d to end
public static void columnString(String s) {
    if (s.length() == 0) return;
    else {
        System.out.println(s.charAt(0));
        columnString(s.substring(1));
    }
}
public static void columnString(String s){
    if(s.length() > 0)
    {
        System.out.println(s.charAt(0));
        columnString(s.substring(1));
    }
}

Here: base case in implicit: if length is 0, do nothing
public static void backString(String s) {
    if (s.length() > 0) {
        backString(s.substring(1));
        System.out.println(s.charAt(0));
    }
}
public static void triString(String s, int level){
    if(s.length() > 0)
    {
        for(int i = 0; i<level;i++)
            System.out.print(s.charAt(0));
        System.out.println();
        triString(s.substring(1),level+1);
    }
}

What does `Methods.triString("abcd",1)` do??
What is the base case??
Why do you know that you’ll reach base??
Methods.palString("abcd");

abcdddcba
public static void palString(String s) {
    if (s.length() > 0) {
        System.out.print(s.charAt(0));
        palString(s.substring(1));
        System.out.print(s.charAt(0));
    }
}
Efficiency - The stamp problem revisited

(3,10,39)
(2,10,2)(3,9,35)
(1,10,-35)(2,9,-2)(3,9,35)
(2,9,-2)(3,8,31)
(2,8,-6)(3,7,27)
...
...
call count: 23
Fibonacci numbers - a catastrophe

1,1,2,3,5,8,13

public static int fib1(int n){
    if (n==0) return 1;
    if (n==1) return 1;
    return(fib1(n-1) + fib1(n-2));
}
\[ \text{fib}(8) = \text{fib}(7) + \text{fib}(6) \]

\[ \text{fib}(7) = \text{fib}(6) + \text{fib}(5) \]

\[ \text{fib}(6) = \text{fib}(5) + \text{fib}(4) \]

\[ \text{fib}(5) = \text{fib}(4) + \text{fib}(3) \]

\[ \text{fib}(5) = \text{fib}(4) + \text{fib}(3) \]
public static int fib2(int b2, int b1, int term, int n) {
    if (n == 0) return 1;
    if (n == 1) return 1;
    if (term == n) return b1;
    else return fib2(b1, b1 + b2, term + 1, n);
}

To get the kth Fibonacci number (starting from the 0th)

fib2(1, 1, 1, k)
Midterm average: 78.66 (last fall: 74.9)