Announcements

Programming assignment Five due next Friday
Ch 8 Embedded Problems due next Wednesday, trapdoor problems (recursion) included!
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) -> (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)));}

recursive calls

base cases
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
• Conceptually it’s about subproblems: to solve a problem instance, 1) check if it’s a base case; and 2) if it isn’t, solve by combining a little work with a (smaller) subproblem.
How would you compute factorial recursively?
This is n! -- 4! = 4*3*2*1, also 0! = 1 (by definition)

What's a subproblem of (calculating) 4!
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:

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

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 boolean isPrefix(String s, String t){
    if (isEmpty(s)) return true;
    else if (isEmpty(t)) return false;
    else if (s.charAt(0) != t.charAt(0)) return false;
    else return isPrefix(s.substring(1), t.substring(1));
}

Note: 3 base cases. Order they’re checked matters
The subproblem is: knock off lead character of both strings, and continue..
public static boolean isPrefix2(String s, String t) {
    if (isEmpty(s)) return true;
    else if (isEmpty(t)) return false;
    else if (head(s) != head(t)) return false;
    else return isPrefix(tail(s), tail(t));
}
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",3) do??
What is the base case??
Why do you know that you'll reach base??
Methods.palString("abcd");

abcddcda
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));
    }
}

What’s the base case? What subproblem does the recursive call address?
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)
...

(3,10,39)
(2,10,2)
(1,10,-35)
(2,9,-2)
(3,9,35)
(3,9,35)
(2,9,-2)
(3,8,31)
(2,8,-6)
(3,7,27)
(3,7,27)
(2,7,-10)
(2,7,-10)
(3,6,23)
(3,6,23)
(2,6,-14)
(2,6,-14)
(3,5,19)
(2,5,-18)
(3,4,15)
(2,4,-22)
(3,3,11)
(2,3,-26)
(3,2,7)
(2,2,-30)
(3,1,3)
(2,1,-34)
(3,0,-1)
no way
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)