Administrivia

- Reminder: Quiz 5 next Tuesday. Likeliest topic is something GUI-related.
- Reminder: Homework 6 design due today, code next Tuesday.
- I added a few more things to the sampler program from last time. On the Web ("sample programs" page).

Recursion — Overview

- Basic approach:
  - Identify "base case" — something you can solve directly.
  - Figure out how to decompose non-base cases into "smaller" problems, and apply algorithm to smaller problems.
- How to think about "does it work?"
  - Does it work for base case(s)?
  - Assuming recursive calls work, does it work for other cases?
  - Does every recursive call get you at least one step closer to a base case?
- Implementation — conceptually (and usually in fact) involves a stack of calls-in-progress.
- Can be slower than iteration (though sometimes not), but can also be much easier to understand.
Recursion — Simple Examples

- Factorial function.
- Function to compute Fibonacci numbers (very slow!).

Recursion — Parsing an Arithmetic Expression

- “Fully parenthesized arithmetic expression” is one of two things:
  - A number $n$.
  - Something of the form $(e \, op \, f)$

  where $e$ and $f$ are expressions and $op$ is one of the four arithmetic operators.
- How to evaluate one of these?
- Let’s write code for that …
Recursion — More Examples

• Quicksort — pick “pivot” element, split array into elements less than pivot and elements greater than pivot, and sort recursively. Why does this work?

• Mergesort — split array (or list) into two pieces of equal size, sort recursively, merge. Why does this work?

• Filling the area inside a border.

Minute Essay

• Consider the following recursive function.

```java
public static int mystery(int m, int n) {
    if (n == 0)
        return m;
    else
        return 1 + mystery(m, n-1);
}
```

• What does `mystery(5, 3)` return?

• Give a short description in general of what `mystery` accomplishes (not how it accomplishes it — e.g., we don’t really care whether `Math.min(a, b)` uses `if` or something else, so long as it returns the smaller of `a` and `b`). Assume input `n` is non-negative, or also say what happens if `n` is negative.
• \texttt{mystery(5, 3)} returns the value 8.
• In general, \texttt{mystery(m, n)} adds \texttt{m} and \texttt{n} — assuming \texttt{n} is non-negative. If \texttt{n} is negative, you get “infinite” recursion (the quotes are because usually the recursion stops, with a crash, when you run out of stack space).