Problem Solving

- A big focus of this course is problem solving
- Key idea: problems are easier to solve if we break them down into manageable pieces
- Wherever possible, we try to exploit the nature of the problem to simplify the computation
  - e.g., if problem involves repetition, we can use iteration to make the problem much easier to solve

Recursive thinking

- Sometimes, problems have an interesting characteristic: their solution requires solving a simpler version of the same problem
- This is known as recursion
- Example: factorials on board

Recursion in Java

- It's possible to have a method call itself!
- How does this work? (on board, also see page 663)
- How can we code a recursive solution to the factorial problem using Java?

Solution to factorial

```java
//if n isn't positive, return 1
public static int factorial(int n){
    if (n <= 1) return 1;
    return n * factorial (n-1);
}
```

Recursive Solutions

- In general, a recursive method has two parts:
  - Base case: a trivial version of the problem that can be solved immediately
  - Recursion step: a solution that involves a recursive call to the method with a simpler version of the problem
- The base case is responsible for stopping the recursion
Why recursion?

• Some problems are naturally recursive; it may be easier to think about a recursive solution
• Recursive methods are sometimes simpler than iterative solutions
  – Minor example: in factorial program, we didn't have to explicitly keep a variable for the current total
• Any task that can be solved with iteration can be solved with recursion and vice-versa
• The choice of which is more appropriate depends on the problem

Efficiency of recursion

• Calling a method adds some overhead to the program execution (why?)
• For this reason, an iterative solution may be slightly faster
• In extreme cases where the recursion is really "deep" (i.e. a lot of method calls are required), your program could run out of memory
• Typically, though, recursive solutions may be nearly as fast as iterative solutions, and may be easier to understand

Helper methods

• Sometimes, when making a recursive call, it is helpful to pass along some additional information besides the original parameters
• In these cases, we can define a helper method that does the actual recursion (see section 18.3)
• Example: DivDigits problem from homework #3