

# CSSE 230 Day 6

## Intro to Trees

After today, you should be able to...

- ...use tree terminology
- ...write recursive tree functions

# Announcements

- ▶ Review Day 5's quizzes on Java Collections and Data Structures
- ▶ Preview of HW3: includes an essay

# Observation about Stacks and Queues Infix $\rightarrow$ Postfix problem

- ▶ It must be  $O(n)$ , so you can't grow your strings character-by-character:
  - Strings are immutable, so characters must be copied. `s += "*"`  is as slow as growing an array using the `+1` scheme
- ▶ Solution? Use a `StringBuilder`!
  - They have internal capacity, which doubles when full!
- ▶ See the example at the end of Warmup and Stretching's `ShapeTest.java` for an example.

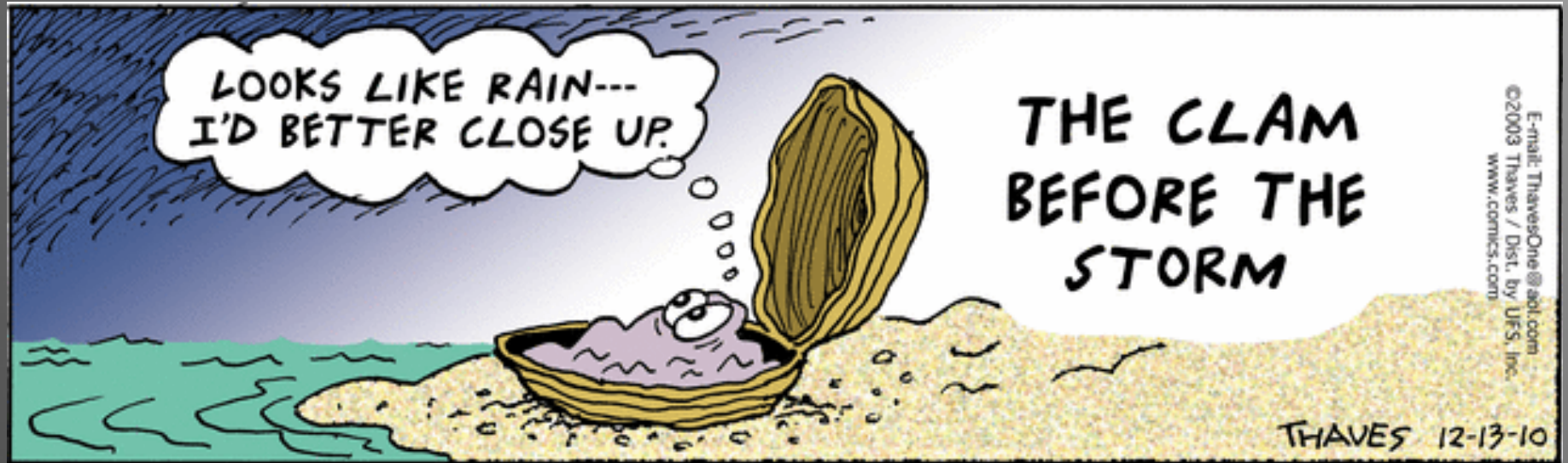
# Exam 1

- ▶ Exam 1 – Thu next week: 7–9 pm
  - Coverage:
    - Everything from reading and lectures, Sessions 1–5
    - Programs: Warmup, Stacks and Queues
    - Homeworks 1–2
  - Allowed resources:
    - Written part:  $\frac{1}{2}$  of one side of 8.5 x 11 paper
      - Goal: to let you use formulas but force you to summarize.
    - Programming part:
      - Textbook
      - Eclipse (including programs you have written for CSSE230)
      - CSSE230 web pages and materials on Moodle
      - Java API documentation – bookmark these in your browser
    - Two previous 230 Exam 1's are available in Moodle

# Exam 1 Possible Topics

- Written (50–70%):
  - Growable Arrays
  - MCSS
  - big  $O/\theta/\Omega$ : true/false, using definitions, code analysis
  - Binary search
  - ADT/Collections
  - Choosing an ADT to solve a given problem
- Programming (30–50%):
  - Implementing an ADT using an array, nodes, or another ADT
  - Writing an efficient algorithm to solve a simple array-based problem

# Questions?



# Next:

- ▶ an implementation that offers interesting benefits, but is more complex to code than arrays or lists...
- ▶ ... Trees!

# Trees

Introduction and terminology  
for three types



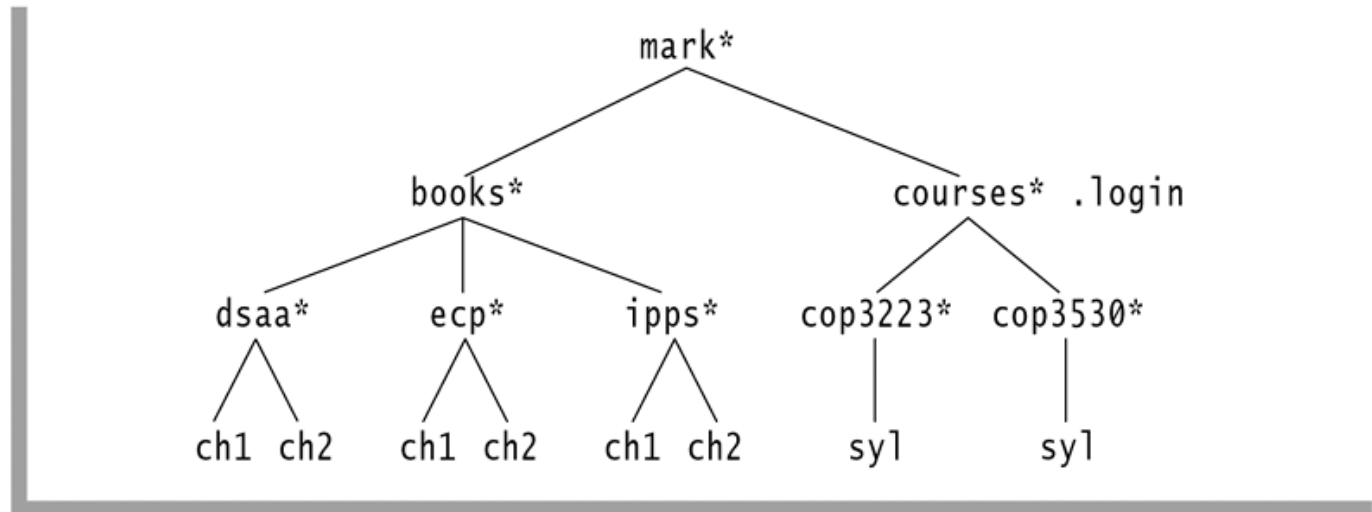


# Trees in everyday life

- ▶ Class hierarchy tree (single inheritance only)
- ▶ Directory tree in a file system

**figure 18.4**

A Unix directory



# Traverse a Directory Tree

```
import java.io.File;

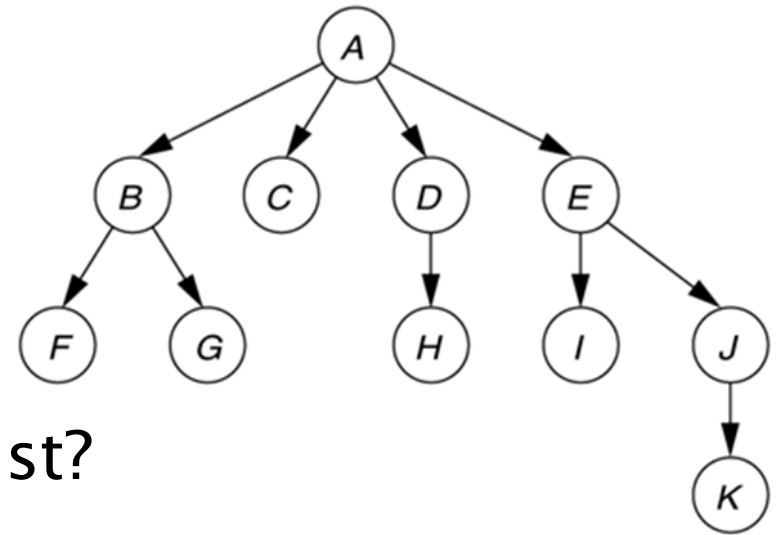
public class TraverseFiles {

    public static void main(String... args) {
        File[] files =
            new File("C:/EclipseWorkspaces/csse230-2014/BST2").listFiles();
        showFiles(files, 0);
    }

    public static void showFiles(File[] files, int indent) {
        for (File file : files) {
            if (file.isDirectory()) {
                System.out.println("        ".substring(0,indent) +
                    "Directory: " + file.getName());
                showFiles(file.listFiles(), indent+1); // Calls method again.
            } else {
                System.out.println("        ".substring(0,indent) +
                    "File: " + file.getName());
            }
        }
    }
}
```

# A General Tree—Global View

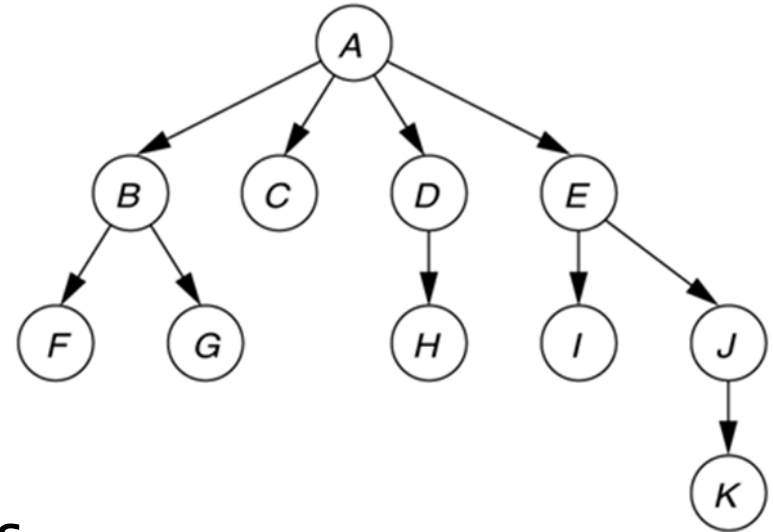
- ▶ A collection of **nodes**
- ▶ Nodes are connected by **directed** edges.
  - One special **root node** has no incoming edges
  - All other nodes have exactly one incoming edge
- ▶ One way that Computer Scientists are odd is that our trees usually have their root at the top!



- ▶ How are trees like a linked list?
- ▶ How are they different?

# Tree Terminology

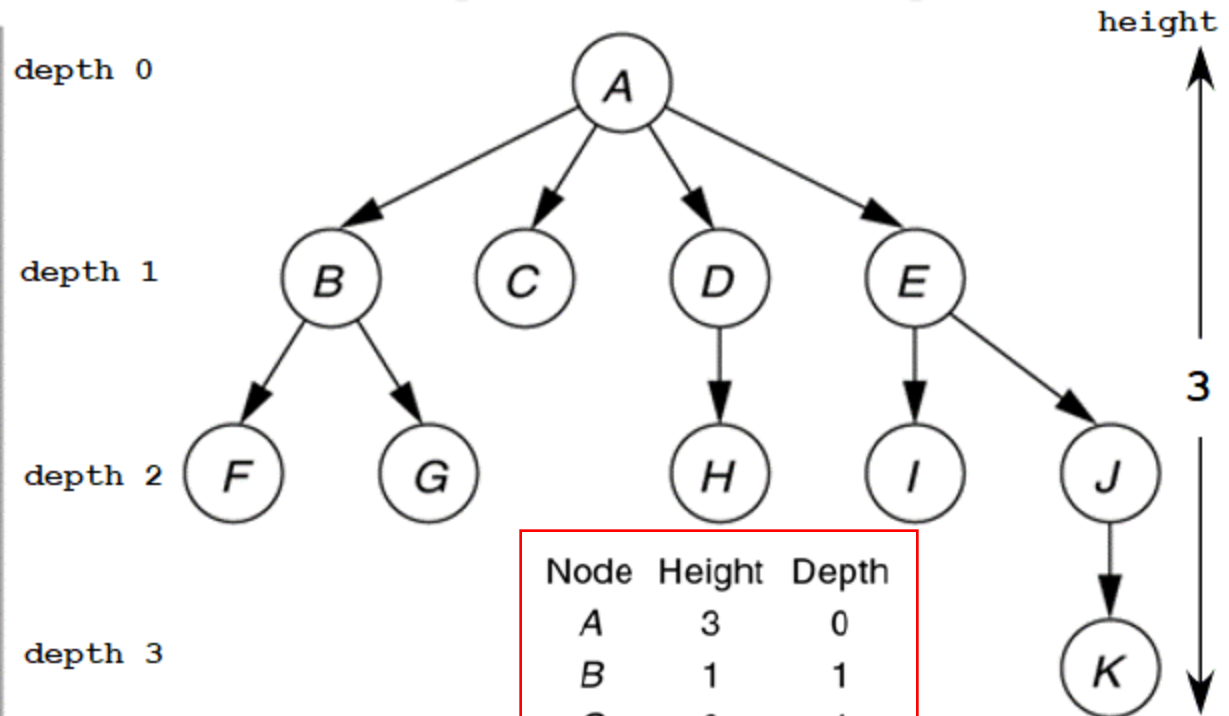
- ▶ Parent
- ▶ Child
- ▶ Grandparent
- ▶ Sibling
- ▶ Ancestors and descendants
- ▶ Proper ancestors, proper descendants
- ▶ Subtree
- ▶ Leaf, interior node
- ▶ Depth and height of a node
- ▶ Height of a tree



# Node height and depth examples

**figure 18.1**

A tree, with height and depth information



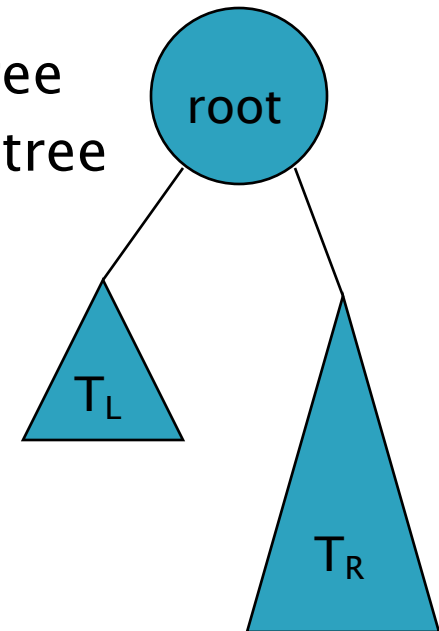
The height of a tree is the height of its root node.

Which is larger, the sum of the heights or the sum of the depths of all nodes in a tree?

| Node | Height | Depth |
|------|--------|-------|
| A    | 3      | 0     |
| B    | 1      | 1     |
| C    | 0      | 1     |
| D    | 1      | 1     |
| E    | 2      | 1     |
| F    | 0      | 2     |
| G    | 0      | 2     |
| H    | 0      | 2     |
| I    | 0      | 2     |
| J    | 1      | 2     |
| K    | 0      | 3     |

# Binary Tree: Recursive definition

- ▶ A **Binary** Tree is either
  - **empty**, or
  - **consists** of:
    - a distinguished node called the root, which contains an element, and
    - A left subtree  $T_L$ , which is a binary tree
    - A right subtree  $T_R$ , which is a binary tree
- ▶ A **Binary** tree node has at most 2 children



# Binary Search Tree (BST)

- ▶ A binary tree with the Search Property:
  - Every element in the left subtree is smaller than the root, and every element in the right subtree is larger than the root. And this is true at **every node**, not just the root.
  
- ▶ Compare: search on
  - Binary tree?
  - Binary Search Tree?

# Recursion in Binary Trees

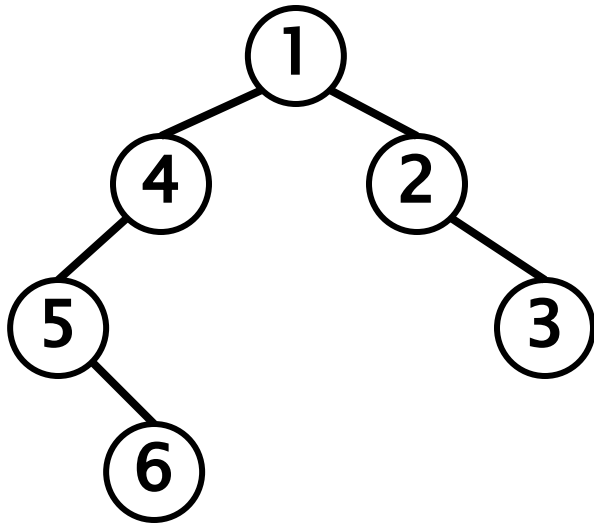
- ▶ (Review) Write `size()` for linked list
  - Non-recursively
  - Recursively
- ▶ Write `size()` for a tree
  - Recursively
  - Non-recursively (later)
- ▶ What are characteristics of correct, efficient recursive code?



# Growing Trees

- ▶ Let's start the BinarySearchTrees assignment: implement a **BinaryTree<T>** class

Test tree:



A single tiny recursive method for size will touch **every node in the tree**. Let's write, then watch in debugger.