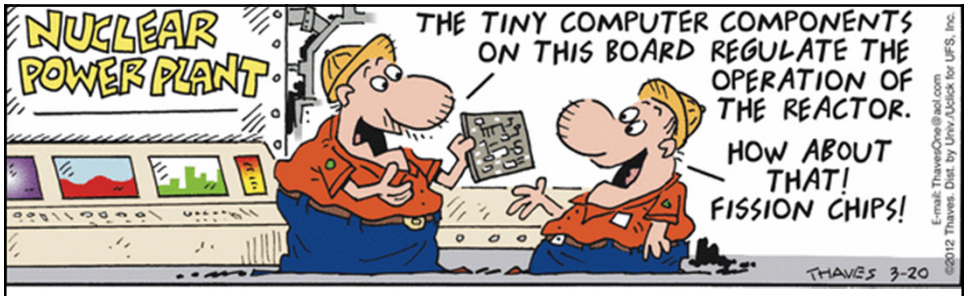


CSSE 230 Day 8

Java Collections Framework
Intro to Trees

Announcements

- Reminder: Pascal partner eval due Wednesday at noon
- ANGEL > Lessons > Dropboxes and Surveys > Surveys

Some Thoughts on Teaming

- Rose is teeming with teams

Two Key Rules

- No prima donnas
  - Work together or divide work equitably
  - Working way ahead, finishing on your own, or changing the team's work without discussion:
    - is selfish
    - harms the education of your teammates
    - may result in a failing grade for you on the project
- No laggards
  - Coasting by on your team's work:
    - is selfish
    - harms your education
    - may result in a failing grade for you on the project

Grading of Team Projects

- I'll assign an overall grade to the project
- Grades of individuals will be adjusted up or down based on team members' assessments
- At the end of the project each of you will:
  - Rate each member of the team, including yourself
  - Write a short Performance Evaluation of each team member

Ratings

Excellent—Consistently went above and beyond: tutored teammates, carried more than his/her fair share of the load

Very good—Consistently did what he/she was supposed to do, very well prepared and cooperative

Satisfactory—Usually did what he/she was supposed to do, acceptably prepared and cooperative

Ordinary—Often did what he/she was supposed to do, minimally prepared and cooperative

Marginal—Sometimes failed to show up or complete tasks, rarely prepared

Deficient—Often failed to show up or complete tasks, rarely prepared

Unsatisfactory—Consistently failed to show up or complete tasks, unprepared

Superficial—Practically no participation

No show—No participation at all

Performance Evaluations

- Ratings must be supported with evidence
- Performance evaluations document:
  - Positives
  - Key negatives
- Performance evaluations are standard procedure in industry
  - Why might that be the case?

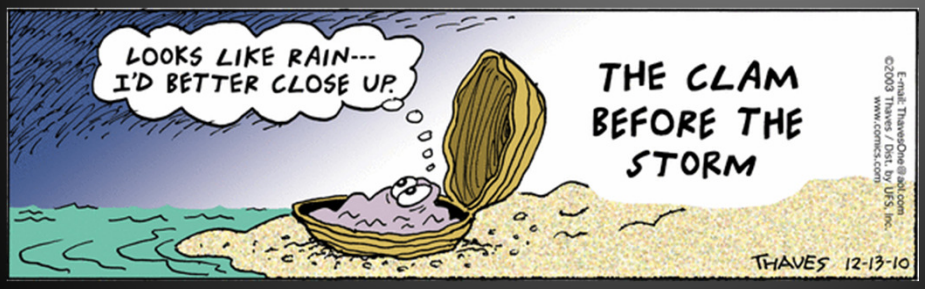
How to Write a Performance Evaluation

- Document dates and actions:
  - Jan. 1, stayed after mtg. to help Bob with hashing
  - Jan. 19, failed to complete UML diagram as agreed
- List positives:
  - The only way to help people improve!
- List **key** negatives:
  - Not **all** negatives
  - Egos are too fragile for long lists, can't fix everything at once anyway

Learning to Write Performance Evaluations

- Keep a journal about the project
- I'll grade your evaluations
- Your evaluations may be shared with the team member being evaluated.
- Don't keep your feelings to yourself
- Get me involved if your team can't work out an issue!

Questions?



Agenda

- Java Collections Framework
- Iterators
- Introduction to trees

Java Collections Framework

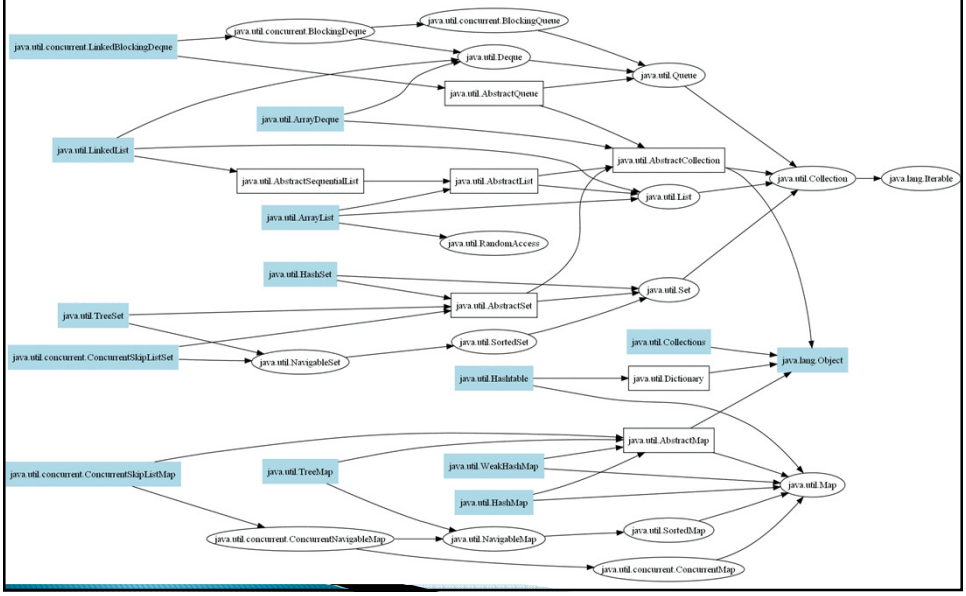
- Available, efficient, bug-free implementations of many key data structures

Most classes are in **java.util**

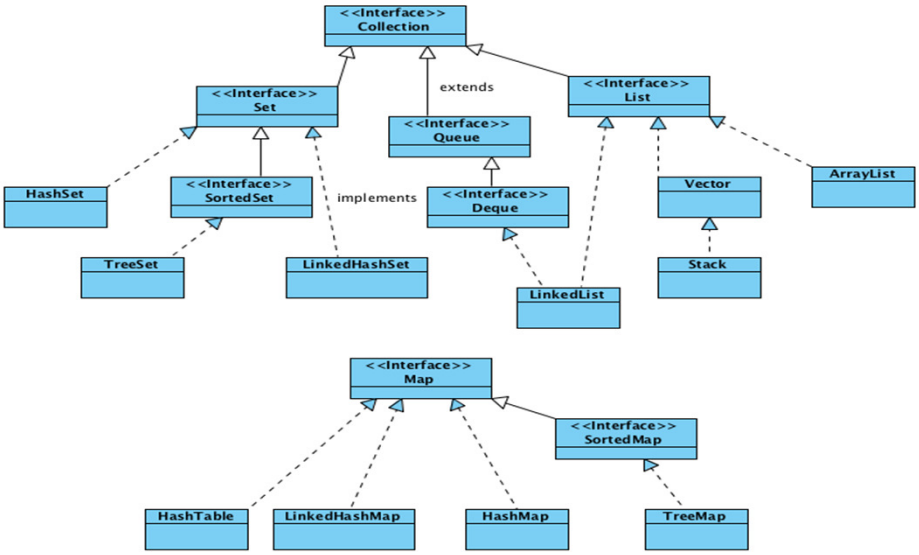
Weiss Chapter 6 has more details about collections

Q1

Java 6 Collections API has many classes



Java Collections: the essentials



Specifying an ADT in Java

- Done with an interface, e.g., java.util.Collection

java.util
Interface Collection<E>

boolean	**add**(E o) Ensures that this collection contains the specified element (optional operation).
boolean	**contains**(Object o) Returns true if this collection contains the specified element.
boolean	**isEmpty**() Returns true if this collection contains no elements.
boolean	**remove**(Object o) Removes a single instance of the specified element from this collection, if it is present (optional operation).
int	**size**() Returns the number of elements in this collection.
Iterator<E>	**iterator**() Returns an iterator over the elements in this collection.

A "factory method"

Q2

What's an iterator?

- In Java, specified by **java.util.Iterator<E>**

boolean	**hasNext**() Returns true if the iteration has more elements.
E	**next**() Returns the next element in the iteration.
void	**remove**() Removes from the underlying collection the last element returned by the iterator (optional operation).

- **ListIterator<E>** adds:

boolean	**hasPrevious**() Returns true if this list iterator has more elements when traversing the list in the reverse direction.
int	**nextIndex**() Returns the index of the element that would be returned by a subsequent call to next.
Object	**previous**() Returns the previous element in the list.
int	**previousIndex**() Returns the index of the element that would be returned by a subsequent call to previous.
void	**set**(Object o) Replaces the last element returned by next or previous with the specified element (optional operation).

Q3

Example: Using an Iterator

ag can be any `Collection` of `Integers`

```
for (Iterator<Integer> itr = ag.iterator(); itr.hasNext(); )
 sum += itr.next();
System.out.println(sum);
```

In Java 1.5 we can simplify it even more.

```
// New approach that uses an implicit iterator:
for (Integer val : ag)
   sum += val;
System.out.println(sum);
```

Note that the Java compiler translates the latter code into the former.

Other `Collection` Methods

- `addAll(Collection other)`
- `containsAll(Collection other)`
- `removeAll(Collection other)`
- `retainAll(Collection other)`
- `toArray()`

Sort and Binary Search

- Handy **java.util.Arrays** utility methods:

static int	**binarySearch**(Object[] a, Object key) Searches the specified array for the specified object using the binary search algorithm.
static int	**binarySearch**(Object[] a, Object key, Comparator c) Searches the specified array for the specified object using the binary search algorithm.
static void	**sort**(Object[] a) Sorts the specified array of objects into ascending order, according to the *natural ordering* of its elements.
	sort(Object[] a, Comparator c) Sorts the specified array of objects according to the order induced by the specified comparator.

See **Collections** for similar methods on **Lists**

The weiss Packages

- **weiss.util**
  - Shows "bare bones" possible implementations of some of the classes in **java.util**
  - Illustrates (just) the essence of what is involved in implementation
- **weiss.nonstandard**
  - Some other data structures, not found in **java.util**
  - Some alternate approaches to some classes that are also in **weiss.util**

Q4

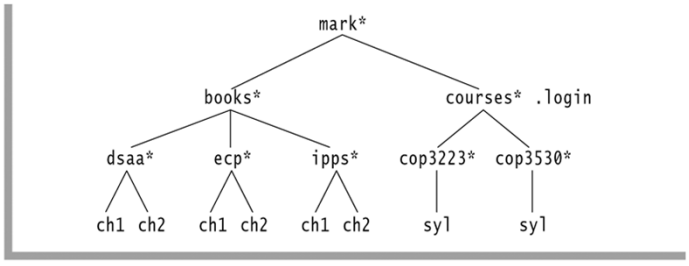
Trees

- Introduction and terminology

Trees in everyday (geek) life

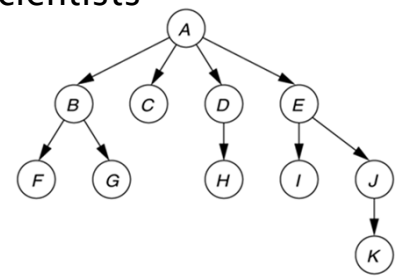
- Class hierarchy tree (single inheritance only)
- Directory tree in a file system

figure 18.4
A Unix directory



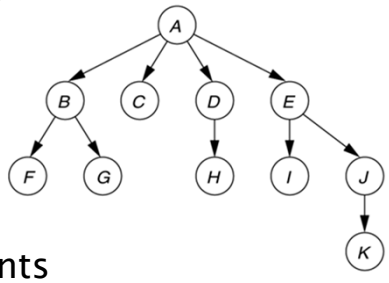
A General Tree—Global View

- 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!



Q5-7

Tree Terminology



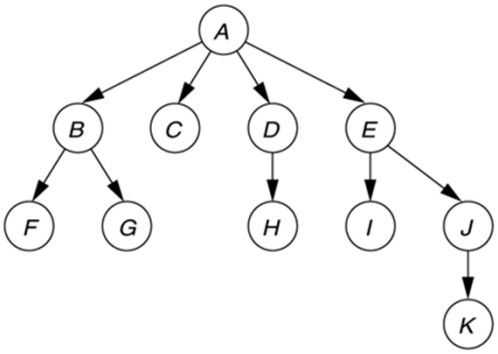
- Parent
- Child
- Grandparent
- Sibling
- Ancestors and descendants
- Proper ancestors, proper descendants
- Subtree
- Leaf, interior node
- Depth and height of a node
- Height of a tree

Q8

Node height and depth examples

figure 18.1

A tree, with height and depth information



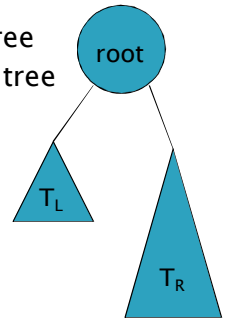
The **height of a tree** is the height of its root node.

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

Q9–12

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



2

Growing Trees

- Let's implement a **BinaryTree<T>** class including methods **size()**, **height()**, **duplicate()**, and **contains(T)**.

Hardy/Colorize Work Time

- Use good pair programming techniques

  Switch drivers frequently

Q5