CS W3134: Data Structures in Java

Lecture #4: Lists
9/16/04
Janak J Parekh

Administrivia

- Homework 1 out today
- Webboard up
- We'll post some of the recitation notes for those of you who couldn't make it

Agenda

- Couple last Java points…
- Start list basics

Java refresher

- Static and main( ), revisited
  - Avoid overuse of static (in fact, you won't need it much at all right now…)
- Default constructor
- Any other questions for now?

More complex example

- We're not going to spend too much time on OO concepts right now
  - Will introduce them as they come up
- Let's start building an Employee database
  - What classes?
  - What methods/variables?
  - What kinds of operations?
- How do we store many Employees?

We use arrays

- Chapter 2
- Arrays are the simplest way to store lists (but not the only way)
- Creating and using arrays
  - New: new type[];
    - Initialization of arrays in Java – default and custom ({})
  - Access an element by index

Array-backed lists

- First sample book program starts with these primitives and works with them manually
  - Similar to 1004/1007 strategy
  - Works, but… kind of awkward – we must always worry about the array throughout the program
  - Wouldn't it be nice if we could separate all of the array "stuff" into a separate class and let it worry about it?

Smarter lists!

- We want to create an *interface* for a list: what the user has to deal with
  - Next refinement: setElem(i) and getElem(i)
  - Still too much work!
  - Who thinks of arrays or indices when making a shopping list?
- Higher-level interface definitions: *abstraction*
  - What operations can you think of?

"Unordered" lists

- How do we do…
  - Insert( )?
  - Delete( )?
  - Find( )?
  - Display( )?
  - Sort( )? (We wait)
- Play with the sample applet
  - Operations include New, Fill, Insert, Find, Delete

Ordered lists

- What's an ordered list?
- How do we do…
  - Insert( )? Book page 60 has a clever technique
  - Find( )? Book page 57
    - lowerBound, upperBound

Costs

- How much do each of the previous operations cost in the *worst case*?
  - Most are linear, some are unit
- Binary search is special – it's better than linear time
  - Divide the range by half until too small to divide further == # of comparisons needed
  - Reverse: what's the range that can be covered with *n* steps? (Book page 63)
  - i.e., $r = 2^s$
  - What's this expressed as in terms of s?
    - $s = \log_2 r$
  - Algorithm grows *logarithmically*

Formalizing costs

- Terminology differs based on details; we'll go light
- Time to insert one element is some constant *K*
  - e.g., T(N) = K
- Time to search for an element is T(N) = K * N
- "Big-Oh Notation": upper-bound on worst-case time
  - We drop the constant K – for *sufficiently large N*, the constant is unimportant
  - The idea of doubling your computer's speed is embedded in K
  - T(N) = O(N), for example

Examples of costs

- For lists using arrays?
  - Linear search: O(N)
  - Etc.
  - Draw a graph of the comparative costs, page 72
- What are bad about arrays?
  - Slow search in unordered, slow insert in ordered – can we speed both? Yes
  - Fixed size: can we change that?

Next Time

- Big-Oh notation, cont'd
- Sorting lists