CSC207.01 2014S, Class 32: Pause for Breath

Overview

- Preliminaries.
  - Admin.
  - Questions.
- More notes on exam 2.
- Remaining DLL topics.
- Generics, revisited.
- Inner classes, revisited.

Preliminaries

Admin

- No writeup for today.
- Happy Pi day!
- Exam 2 code is ready. Unit tests for problems 3 and 4 forthcoming.
  - I'll be writing the unit tests and solving all of the problems as soon as I'm healthy.
  - Due date moved to class time Friday after break.
- Yesterday's extra has an extended riff on the various inner classes.
- Extra credit:
  - CS table today: Is Codecademy worth it and other casual conversations.

More notes on exam 2

Problem 3: remove() is complicated in singly linked lists if the iterator is on the node that we want to remove

- Traditionally, iterators in linked lists refer to the node immediately before the node containing the value that we return when the client calls next.
- In almost every case, it's *on* the thing that next just returned.
- The iterator needs to back up to the previous node, but we can't do that in constant time.
- So, what alternatives do we have?
  - NO Change the meaning of remove: Remove the next element
    - remove is specified in java.lang.Iterator. Changing the meaning will make things rocky.
    - removing things you haven't seen yet is weird
  - A Keep two pointers! One to the previous element and one to the current element.
    - Needs care in how we implement next
  - B Keep the cursor two back, rather than one back

        - Seems to be hard to maintain that invariant
    - C Keep the cursor one back and add a boolean flag to indicate whether or not we'd returned the next value
    - D Copy the data, rather than eliminating the node
        - Dangerous if other folks refer to the node
- What should we do for skip lists?
    - One option: Keep track of the previous at every level

Remaining DLL topics

- Remember: DLLs exist because remove is such a pain for singly linked lists.
- Three (or more) methods to think about
    - add
    - remove
    - set
    - previous
    - ...
- Design decision: The iterator points to the node before the node that contains the value that next returns

Implement add

- create a new node, n
- fun with pointers
    - n.prev = this.current;
    - n.next = this.current.next;
    - this.current.next.prev = n
    - this.current.next = n
- The last two lines could be written
    - n.prev.next = n
    - n.next.prev = n
    - And those can be in either order
- move iterator forward; semantics of add are "add before the iterator"
    - this.current = n

Implement remove

- splice out the node
    - current.prev.next = current.next
    - current.next.prev = current.prev
- move the iterator backwards
    - current = current.prev;
- If possible, set the next/prev of the now deleted node to null for safety

Generics, revisited

Inner classes, revisited