CmpSci 187: Programming with Data Structures
Spring 2015

Lecture #6

John Ridgway

February 12, 2015

Administrivia

- Collect papers for discussions 2 and 3.
- First exam — Feb 19 at 7:00pm in Marcus 131. If you can't do it then (documented excuse) let me know TODAY.
- Registering clickers on Moodle

Deleting nodes from a linked list

- Like insertion, depends upon location.
- Deleting the head node is easy. Set the head pointer to the current head's link. The old head is now inaccessible.
- Deleting the tail's node means we need to set the tail pointer to its predecessor, and its predecessor's link node to null. But we need to know its predecessor to do this, which requires a traversal.
- Deleting a node in the middle similarly requires knowing a node's predecessor. Set its predecessor's link to its successor.

Linked List Manipulation Clicker Question

Suppose `x` is a node in the middle a large linked list. What will be the effect of `x.setLink(x.getLink().getLink())`?

A. nothing, this command won't compile

B. delete `x` from the list

C. delete the node following `x` from the list

D. make `x` the last node of the list

E. a `NullPointerException` will be thrown

1 LinkedList StringLog ADT Implementation

`LinkedStringLog`

- Attributes: `log`, `name`
- Constructors: `LinkedStringLog(String name)`
- Mutators (Transformers): `insert(String)`, and `clear()`
- Observers: `isFull()`, `size()`, `contains(String)`, `getName()`, and `toString()`

Fiddling with `size()` Clicker Question

What happens if we swap the lines inside the loop?

```
int size() {
  int count = 0;
  LLStringNode node = log;
  while (node != null) {
    count++;
    node = node.getLink();
  }
  return count;
}
```

A. a `NullPointerException` could be thrown

B. `count` would be too small by 1

C. the method would not compile

D. nothing would change

2 Stacks

Stacks

- A *stack* is a data structure that holds a collection of objects.
- Objects can be added using a *push* operation, and removed (and retrieved) using a *pop* operation, which returns the objects in the reverse order that they were pushed, removing them from the stack.
- We can also *peek* at the top item, returning it without removing it.

Stack Underflow and Overflow

- Popping or peeking an empty stack causes *stack underflow*, an error usually signaled by an exception in Java. We can avoid this by checking if the stack is empty before doing a peek or pop.
- Some stacks are *bounded*, which means they have a fixed maximum capacity. Pushing onto a full stack causes *overflow*. Like underflow, we can check for this in advance.

Stack Clicker Exercise

Suppose that I create a bounded stack `stack` of integers, with size 3, then run the following:

```
stack.push(1);
stack.push(2);
stack.push(3);
int x = stack.pop();
int y = stack.peek();
int z = stack.pop();
```

What are the values of `x`, `y` and `z`?

A. `x` = 3, `y` = 2, and `z` = 1

B. `x` = 2, `y` = 1, and `z` = 1

C. `x` = 3, `y` = 2, and `z` = 2

D. `x` = 1, `y` = 1, and `z` = 1

A Few Applications of Stacks

- Stacks were originally named after the receptacles for plates in a cafeteria, where only the top plate can be seen.
- They are widely useful in computing, being the underlying execution mechanism for a lot of code (value stacks and call stacks).
- In a future project we'll show how by using stacks to evaluate expressions.
- In a future discussion we will sort items using two stacks.
- How about parenthesis and bracket matching?