

CS231 Fall 2013 Lecture 10

Stephanie R Taylor

September 25, 2013

1 Administrative Topics

- Return the quizzes

2 StringLinkedList

Let's return to the linked list of strings we were implementing.

- Make it a class
- Make a constructor with an empty list
- Write the add method, to add a String to the beginning of the list.
- Fun fact: Because StringNode is an inner class of StringLinkedList, it is considered a member of the StringLinkedList. It is part of the definition of the StringLinkedList. So code in the StringLinkedList that is outside of the StringNode can see private fields in the StringNode! That makes writing code nice and simple: e.g. to get an element from part way through the list or to insert part way through the list, we can access the next field of the nodes directly. This makes our code look much cleaner.

3 Iterators

Who cares about them? To see their advantage, let's first create a `toString()` method for our `StringLinkedList` class. It should list the data, separated by spaces.

```
public String toString() {
    String result = "";
    StringListNode temp = head;
    while(temp != null) {
        result += temp.data + " ";
        temp = temp.next;
    }
    return result;
}
```

Note that it uses `temp = temp.next` just like for loops use `i++`. We can actually rewrite this to use a for loop:

```
public String toString() {
    String result = "";
    for(StringListNode temp = head; temp != null; temp = temp.next) {
        result += temp.data + " ";
    }
    return result;
}
```

This code is fine. However, it uses the Nodes directly. What if the user wants to loop through the data in the list? For example, suppose the user wants to print out the airport codes one per line instead in a row. The user should not even know that `ListNode` objects exist. We need some way of letting the user loop through a list. Suggestions? Remember that the user can use only public methods in the `LinkedList` class.

One way is to use `get(i)`:

```
for(int i = 0; i < list.size(); i++) {
    String s = list.get(i);
    // ... now use s for your purposes...
}
```

Problem: What if the list has length 1 million? This will take way too long. We need a way of stepping through the linked list without dealing directly with `StringListNodes`. We can do this with Iterators!

To get this to work, the main thing that the `StringLinkedList` class needs to do is implement a method with header:

```
public Iterator<String> iterator()
```

and also declare to the Java compiler that it is implementing such a method by adding the phrase

```
implements Iterable<String>
```

to its declaration.

Once we have an iterator for the class, we can use it explicitly

```
for (Iterator<String> iterator = list.iterator(); iterator.hasNext(); ) {  
    String item = iterator.next();  
    System.out.println( item );  
}
```

But Java has a wonderful short-cut for us. It is called the `foreach` loop, and it manages the iterator for us:

```
for (String item: list) {  
    System.out.println( item );  
}
```

4 Terms

- **interface:** A group of related methods with empty bodies. e.g. the `Iterator` interface specifies the methods `hasNext`, `next`, and `remove`.
- **implements:** the term you use to indicate that a class provides the code to implement all the methods in a particular interface
- **iterator:** An iterator is an object that makes it easy to iterate through another data structure, such as a linked list.