Design of Information Structures - Lab 2

January 25, 2019

1 Linked Lists

In last week's lab we looked at how to create an `Array List` structure. In particular , we saw that ArrayLists dynamically grow to store as many items as needed. In this week's session we will begin by looking at another dynamic list structure, the `Linked List`.

1.1 A Reminder about References in Java

In a simplistic view, when we create an object in Java we are creating a reference to a space in memory where the object is stored. If we represent this graphically we could imagine the memory might look something like this:

```java
String myString = new String("hello");
String myNullRef = null;
```

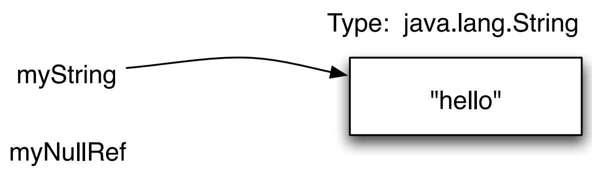

Figure 1: **References to Java Objects**

Notice that when we declare something to be `null` only a reference is actually created. It is only when we initialize an object, typically by calling a constructor, that space is allocated for the object type itself.

1.2 An Introduction to Linked Lists

Imagine taking a piece of paper to write a shopping list. You start by writing the first item on the first line, followed by the second, etc.. The size of the list is determined by the number of items written on the piece of paper. Conceptually, we can think of a list as being defined recursively — a list is one item followed by zero or more items (if there are more than zero items we call this a sublist). Show graphically, a list might look like:

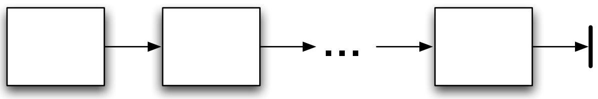

Figure 2: **Linked List**

1.3 Representing Linked Lists in Java

The question is now how can we represent a linked list structure in Java? The answer perhaps unsurprisingly is to use a combination of classes and references. Using our definition of a list above, we can define a Java list item to be a piece of data and a reference to the next list item. The code for a linked list element containing a String is shown below:

```
public class MyLinkedListElement
{
    private String value;
    private MyLinkedListElement next_elem;

    public MyLinkedListElement(String val) {
         this.value = val;
         this.next_elem = null;
    }

    public void setNext(MyLinkedListElement next_element) {
        next_elem = next_element;
    }
}
```

The `value` field represents the value being stored at that position in the list — in our shopping example this is the item we want to buy. The `next_elem` field represents the link (arrow in the picture) to the next element in the list (i.e. the start of the sublist).

Now we have designed a basic element in our list we can chain these together to build a list of any size we need to store data. The next step in defining our list structure is to produce a class which can construct the list when we want to add data elements. The code this is given below:

```java
public class MyLinkedList {
        MyLinkedListElement head;

        public MyLinkedList() {
                // The list is empty (there is no list)
                head = null;
        }

        // Add element to head of the list.
        public void add(String s) {
                MyLinkedListElement temp =
                        new MyLinkedListElement(s);

                // If list is not empty, point new link to head
                if (head != null) {
                        temp.setNext(head);
                }
                // Update the head
                head = temp;
        }
}
```

1.4 Developing Singly Linked Lists

Copy the files for this laboratory from the cs126 web site (download file `lab2.zip`).

Again, the folder contains the IList interface, which is the same file as we used in the last lab. In this lab, rather than implement the list interface using arrays, we are going to use our LinkedListElements. As in the previous lab, some of the methods are incomplete (those preceeded by an `//INCOMPLETE.` comment) — and you will now finish them.

Exercise 1 *Edit `MyLinkedList.java` and complete the `isEmpty()` method:*

```java
// INCOMPLETE.
public boolean isEmpty() {
        // returns whether the list is empty
        return false;
}
```

Exercise 2 *Modify the `addToHead()` method to use your new `isEmpty()` method.*

Milestone 1) You have now completed **Milestone 1**. You should contact your lab tutor and get this signed off.

---

2 Iterating over the List

Now that we have built our list structure we want to develop a set of methods to add data to the list and process all of the items. One of the most important concepts is **iteration**. This is essentially visiting and processing each element in the list in order — we are said to 'iterate' over the list.

We can iterate over our list structure by taking each item, carrying out the desired operation and then following the link to the next item. We repeat this process until the next item link is `null`, meaning we have no more items to visit and have therefore reached the end of the list.

The basic code to complete the iteration is given below:

```
MyLinkedListElement<E> ptr = head; // begin at the beginning

while (ptr != null) {
    // ptr points the next element
}
```

There are other examples of iteration in the `MyLinkedList.java` file that you should understand to complete the next exercises.

Exercise 3 *You should use this iteration code to complete the `size` method which returns the number of elements found in the list. Test out your method using the `ListTest` program. You should create three lists — an empty list, a list of one element (containing your name) and a list containing more than one element.*

Exercise 4 *Rewrite the `isEmpty()` method to use `size()` instead of checking whether the head is null.*

Exercise 5 *Add an integer field to the `MylinkedList` class called `count`. You should add code to keep track of the size of the list every time an element is added. You will also need to update your `size()` method to use the new count variable.*

Exercise 6 *Add a toString method to your* `MyLinkedList` *class based on the following template:*

```
public String toString() {
    // convert the contents of the list
    // to a printable string
}
```

Exercise 7 *Modify the* `ListTest` *program to print out the list before and after having added your name.*

Milestone 2) You have now completed **Milestone 2**. You should contact your lab tutor and get this signed off.

3 Adding and Removing Elements

For the final milestone of this lab you are required to add extra functionality to your list class to allow the user to add items to the tail end of the list and remove elements.

Exercise 8 *Write a new method in your* `MyLinkedList` *class which adds elements to the end of the list.*

```
public void addToTail(E s) {
        // Adds an element to the tail of the list
}
```

Exercise 9 *Write a new method to remove an element from the end of the list. You should base your method on the code below:*

```
public String removeFromHead() {
    MyLinkedListElement temp = head;

    head = head.getNext();
    count--;

    return temp.getValue();
}
```

Milestone 3) You have now completed **Milestone 3**. You should contact your lab tutor and get this signed off.