Lab Exercise 4
Linked Lists

In this lab, you will be introduced to what is necessary to implement linked lists. We will take a program using linked lists, examine how it works and implement a small piece of functionality.

1 Introduction

In the lab we will be working with a collector. A collector maintains a list of distinct items and their cardinality (how many times they appear in the collection). You can prompt the collector to grab a thing, and if the collector already has that item, the cardinality of that item increases by one. If the collector does not have that item, it adds it to the collection and sets its cardinality to one. You can also prompt the collector to leave the items, after which the collection is empty.

This lab also introduces you to working with linked lists. A linked list is a series of objects, each of which points to the next object. For this lab, each object is called a CollectorNode, and has three data members: an enumeration saying what sort of thing is being collected, an integer representing the count (cardinality) of the item type, and a reference to the next node. For example:

```
<table>
<thead>
<tr>
<th>coin</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
</tr>
</tbody>
</table>
```

This cell means that we’ve collected three coins. The pointer to the next cell is a null pointer, as it is not linked. If we link it with two other cells, we have a non-trivial linked list:

```
  head
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>coin</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>-----</td>
</tr>
</tbody>
</table>

  pre
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>coin</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>-----</td>
</tr>
</tbody>
</table>

  cur
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ring</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>-----</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>stone</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>
```

The head pointer points to the first node and is where we enter the list. The picture also shows the possible position of two local CollectorNode variables, pre and cur.
# Starting up

Using Eclipse, import “Lab 4”:

/afs/cs.uwm.edu/users/classes/cs351/401/pantherid/git/lab4.git

Run “Main” as a Java Application found in the default package and grab the following things in the exact order:

1. A gem
2. A ring
3. A key
4. A gem
5. A coin

Then write down the contents of the collector:

Collector has

Now what if we grab another key? Write down what you expect the contents of the collector to be and draw what you would expect the linked list to look like. (Drawings for linked list should be in a similar format as the above pictures).

Collector has

Expected:

Now actually grab another key. Then write the contents of the collector and circle the unexpected result due to the bug in `grabSome`. Print the linked list to show what it actually looks like.

As you can see there is an extra “key” node in the list; there should instead just be one “key” node with the number collected being “2.” (As it happens, the first node is the extraneous one.)

Collector has

Actual:
3 Understanding the Code

Take a look at the code for `grabSome` and concentrate on the section where a bug of omission is marked. This section executes if the current item being grabbed is not the head item. Note that `pre` always points to the node before the node which `cur` is pointing. Using the example in the last section, during the last step of grabbing another key. Draw pointers for `pre` and `cur` for both before and after the insert of the last item “key”.

Before the Bug Section (i.e., after the `while` loop):

After the Bug Section:

Looking at the section again, are you able to point out what is missing for the case of grabbing another key?
4 Fixing the Code

So sometimes we want to insert a new node and sometimes not. We don’t want to insert a node if we are pointing to a node that is already collecting this kind of item. Add the following (buggy!) code before the code which creates a new node:

```java
if (cur.thing == new_thing) {
    cur.count += count;
    return;
}
```

Run “Main” again, but this time first grab a gem and then grab a ring. The program will give an error. Draw what the collection looks like at the time the program crashed by evaluating the operation of the `grabSome` method with the provided input. Put in the small pointers for `pre` and `cur`:

The (buggy) code you inserted didn’t handle the case that `cur` was null. What should happen in this case?

Fix the bug by making a change to the code you inserted earlier. Make sure your program is correct and then see your TA to receive credit for this lab.