Eric Olsen

Assumptions

The task for this assignment is to design a linked list data structure containing String elements in each node. This list is circular and doubly linked. We will add to this data structure through two primary methods: addMenuItem(String) and addMenuItemBA(String, String, boolean). The method addMenuItem will add a specified String to the tail end of the structure and addMenuItemBA will add a menu item before or after a specified node. We will make a few assumptions to narrow the scope of this program. First, there will be no overall data structure holding all the values of our menu nodes. Every node that we delete will merely have its pointers reassigned; the java garbage collector will consequently handle the deletion of the node's String element.

Additionally, the user will type in all searched and inserted menu items. The program will not accept multiple words as an element, it will accept only the first word, and the program will make no efforts to change the content of the user's typed input. Additionally, the calling method will not trim inputted Strings and will observe case sensitivities. This program will verify user input by using an endless loop and a combination of selection statements and break key words.

Main Design

From a very general standpoint, three classes will constitute this program: Menu, MenuNode, and TestMenu. Menu is the overall class that holds the addMenuItem, insertMenuItemBA, deleteMenuItem, and toString methods. MenuNode is an inner class of Menu and is the primary data structure constituting this program's linked list. It is composed of three elements, a String value, a nextItem pointer, and a prevItem pointer. The TestMenu contains a file reader which reads and inserts the first set of menu items from a local text document. It then takes user input and allows the user to add or delete as many items as they wish.

I will first describe the broad concepts of the Menu Class. The first design decision I made was to designate a root node as this class's only field. The root node is necessary so there is always a reference point for every update to the linked list. I also did not utilize a dummy node. In retrospect this may have been a poor choice. Consequently the Menu class requires some very careful checks before inserting or deleting anything. I will briefly describe the necessary checks that each class method makes when executing the program.

The addMenuItem always creates a new tail. This consequently creates three checks that we must perform. First, we must check to see if the root node is empty. By empty I mean that it exists but has been assigned a blank element. If this is the case, the inserted item becomes the root node and the root node has null values for its neighbors. If the root node is not blank, we check to see if the root node has neighbors. If it does not, the inserted items become the next and previous nodes to the root. The final check for this method is if the root node already has a tail. If this is the case we assign the inserted item into the tail position and update the required pointers. This method returns a true Boolean if successful and a false if an error occurs.

The insertMenuItemBA method is a little more complex, but follows the same general idea. We first check to see if the root value is null. Our test program should ensure that in this situation, this method never executes, but in the event that this method occurs with a null root node, the method returns a false Boolean and terminates before encountering the null pointer exception. When the root node is not null we check to see if the root node has neighbors. If it does not, we simply create a new tail using the addMenuItem method. After confirming that there is a root node and a next value, we begin to traverse the data structure until we locate the searched item. We do this by creating a temporary node, assigning it the root node's element, and then continually assign it its next neighbor's value until it reaches the searched item. If it arrives at the root node, the method returns a false Boolean and an error message displays saying that the searched item could not be located. If it finds the searched method, it inserts the new element before searched item when ba is true and after when false.

The deleteMenuItem method is fairly straight forward, but requires some care when deleting the root node. To delete a menu node we must utilize the same data structure traversal used in the insertMenuItemBA method. When the searched item is located, a deletion simply reassigns pointer values so the deleted node is skipped over during the next traversal. When deleting the root node, we need to conduct a few extra checks to see if the root node has neighbors. If the deleted root node has only one neighbor, we must to promote that neighbor to the root and assign the next and prev value for the

root to null. If the deleted root node has no neighbors, we must assign the root node a blank element since we cannot and should not delete the root node.

I will only speak briefly on the test driver class. This class immediately loads a local text document from file and populates a menu item list before the user gets a chance to add any inputs. The user is first directed to add one menu item to the list and then delete one. Following this, another series of while loops and if statements add or remove as many nodes as the user wishes.

Error Handling

This program does not catch any exceptions, but it does keep the user locked in an inner and outer while loop that only breaks when proper input has been provided. As one example, if the user types in a new menu item, but they previously deleted the root node, the main method will call insertMenuItem() instead of insertMenuItemBA() since there is no item to search for. The test plan below should do a better job at showing all the errors I attempted to create than my narrative description of them. There is one exception that I could not properly handle. If the user enters the forward slash '\' and the enter key simultaneously, a PatterSyntaxException will result. My attempts to detect this issue were unsuccessful and it remains a bug in the programming. Forward slashes at the beginning and end of Strings creates some issues for the JVM.

Test Plan:

Test Case #	Input	Description	Expected Output	Actual Output	Pass / Fail
1	Add: "jump" "hop" Delete "File" "Save."	Add 2 new items and delete 2 old items.	success	success	Pass
2	Search for "skydive" when it is not a menu item.	Test the error message for searched items.	User stays in input loop	User stayed in input loop	Pass
3	Type "g" for boolean value.	Mis-type boolean value.	User stays in input loop	User stayed in input loop	Pass
4	Delete all nodes, insert "jump," "hop," and "squat."	Delete all items on list, insert 3.	success	success	Pass
5	Jump, but hit "\" and enter key at same time	Throw in an unusual character with text input. Check for PatternSyntaxException.	Exception	Exception	Fail

Test Case1:

Test Case 4:

…

Lessons Learned

I learned that a dummy node would have drastically simplified my code. I'll ensure that I include one if I ever need to create my own linked list data structure again. I also learned that passing Boolean values as calling methods is a very efficient way to provide a means of error control. Finally, I'll need to study the PatternSyntacException and formulate how I can check and trim the forward slash String method inputs.