Homework 3 Design Documentation

1. Problem Analysis
   1. Design a circular double linked list.
      1. The first part of the problem is to design a universal object to store whatever data each element consists of.
      2. This Node must be designed so that it knows what is in forward and backward of itself
      3. The List object must have a concept of start and finish, in order to correctly handle circular functionality.
2. Design Decisions
   1. CircularDoublyLinkedList
      1. class Node
         1. The private class Node is an object used to store an element of the list
         2. It has public Nodes for forward and backward. These are used to store the positon of the node before and after the current node.
         3. Lastly, E data is a universal object identifier. It initializes that object as whatever the CircularDoublyLinkedList is cast as.
      2. Node prev, next, head, tail
         1. These Node objects will be used as place holders during the addition of Nodes to the list
      3. Node current, lastAccessed
         1. helper Node objects for directional functionality
      4. int size
         1. This is used to define how many elements exist in the list
      5. int index
         1. used to traverse the list in properly
      6. Constructor
         1. Initialize previous and next, set the state of head and tail, define the direction of previous and next for proper function.
      7. boolean isEmpty
         1. check whether there are any elements in the list
      8. void add
         1. add a new element to the list
         2. correctly assign the before and after positions for this node
         3. check whether there are already existing elements in the list
         4. if so, set this element to tail
         5. if not, set this element to head and tail
      9. Node current
         1. sets the placement of the current element
      10. Node lastAccessed
          1. place holder to be set once the first element is accessed
      11. int index
          1. An index used for proper traversal of the list
      12. void previousIndex, nextIndex
          1. default public methods of ListIterator
      13. E next
          1. default public method of ListIterator, modified to check whether the current element is the tail, and properly adjust the index
      14. E previous
          1. default public method of ListIterator, modified to check whether the current element is the head, and properly adjust the index
      15. void remove
          1. removes an element from the list based on an argument
      16. void add
          1. adds an element to the list
      17. void set
          1. set the data of the current element to whatever data is passed in as an argument
   2. TestCircularDoublyLinkedList
      1. testWithString
         1. Initialize a CircularDoublyLinkedList and cast it with Strings
         2. add 5 string elements to the list
         3. Cast a new ListIterator with MyDLLIterator
         4. iterate over next 6 elements using this new iterator to verify head is set properly and circular functionality works
         5. iterate over previous 4 elements to verify tail is set properly and circular functionality works
         6. remove an element to exercise requirement for assignment
         7. iterate to next element to validate functionality
      2. testWithInt
         1. This performs a similar functionality to testWithString
         2. It is intended to exercise the ability to cast the object with different types
3. User Interface
   1. I chose not to implement a user interface, and instead used the console output to validate my functionality.
4. Testing and Test Cases
   1. testWithString
      1. Initialize a CircularDoublyLinkedList and cast it with Strings
      2. add 5 string elements to the list
      3. iterate over next 6 elements using this new iterator to verify head is set properly and circular functionality works
      4. iterate over previous 4 elements to verify tail is set properly and circular functionality works
      5. remove an element to exercise requirement for assignment
      6. iterate to next element to validate functionality
   2. testWithInt
      1. This performs a similar functionality to testWithString
      2. It is intended to exercise the ability to cast the object with different types
5. Error Handling
   1. When performing a set or remove I validate that next or previous has first been accessed, an IllegalStateException is thrown if the error state is present
   2. When performing hasNext or hasPrevious, I validate that there is at least an element in the array, otherwise throw an IllegalStateException
6. Lessons Learned
   1. I learned a bit more about the benefits of inheritance over composition, as using inheritance allowed me to create a ListIterator while also modifying the public methods of it.
   2. I also learned a good deal more about the internal mechanisms of a linked list, as well as how to properly handle circular functionality.