

## Michael Bushman

### Design and Algorithm

I first began with the BasicDoubleLinkedList class. I declared the class variables, head, tail and size along with the constructor as well. I implemented the getSize method and then got stuck with the addToEnd method. I shifted my focus to the Node class where I declared the variables data, prev and next. I also added the constructor as well. Next, I worked on the DoubleLinkedListIterator class and implemented the variables nextElement and previousElement. I used the constructor to set the variables to head and null respectively. The hasNext method was straight forward since the book gave us the pseudocode. I set the return to true if nextElement doesn't equal null. For the next method, I create a T object to hold my next element. I checked if the list had anymore element in the forward direction using hasNext. I then assign the next variable to the nextElement's data. I also assigned the previousElement to the nextElement, advanced the nextElement's cursor forward and returned the reference variable. For the hasPrevious method, I did the opposite of hasNext and used the previousElement instead. For the previous method, I created a T object to hold my previous element. Using an if statement along with the hasPrevious method, I set the reference variable equal to the previousElement's data. I then set the nextElement equal to the previousElement, moved the previousElement's cursor backwards and returned the reference. The rest of the methods were straight forward since all we had to do was throw a new UnsupportedOperationException.

I then moved back to the BasicDoubleLinkedList class and began working on the addToEnd method. I first created a new node for the given element. I first checked if the head was null since if it is, we can then assign the head and tail to that new element since both would point to the same location. If the head wasn't null, I then updated the new element's previous to be the tail, the tail's next to be the new element and the tail to be the new element. After everything is finished, I then increased the list size by 1. For the addToFront method, I did the same thing as the addToEnd method but opposite. I created a new node to hold the new element. I first checked if head was null since if it is, we can then assign the head and tail to that new element since both would point to the same location. If the head wasn't null, I set the new element's next to the head of the list, the head's previous to the new element, and the head to the new element. I then increased the list size by 1. For the getFirst and getLast method, both were straight forward. I first checked if the list size was 0. If so, then to return null. If not, then to return the head's data or the tail's data based on which method. For the iterator method, I just return a new object of the DoubleLinkedListIterator method. For the remove method, I first created a node called startingElement to start at the head of the list. I used a while the startingElement was not null, to then check using the comparator of the targetData vs the startingElement's data. If the both objects were equal, I checked if the starting element was equal to the head. If so, to then point the head to the next position on the list. I also checked is the element was equal to the tail and if so, to then point the tail to the previous element. I then decreased the list size by 1. Outside of the comparator if statement, I made sure to point the startingElement to the next element. For the retrieveFirstElement method, I first created a referenced variable to hold the head's data. Based on the given

pseudocode, I first checked if the list size was zero and if so, to return null. I then checked if the list size was 1 and if so, to set head and tail to null since we are removing the element as well. I used an else to then make sure to point the head to the next position and the head's previous to null. I then decreased the list size and returned the first element from the list. For the retrieveLastlement method, I created a reference variable to hold the tail data. I then checked to see if the list size was zero and if so, to return null. I checked if the list size was equal to 1 and if so, to set head and tail equal to null. I used an else to then point the tail to the previous element as well as point the tail's next to null. I then decreased the list size by 1 and returned that last element. For the toArrayList method, I created an arrayList along with a currentElement to start at the head. Using a while loop to check if the currentElement isn't null, I then used the add method to add each element to the arraylist. I then pointed the currentElement to the next element. After everything finished, I returned the arraylist.

For the SortedDoubleLinkedList class, I first created the Comparator variable. I then created the constructor which assigned the given Comparator to my Comparator variable. For the add method, I first created a newElement node to hold the given element data and a currentElement to hold the head. I first checked if head was null and if so, to set the head and tail equal to the newly added element, increase the list size and to return that reference. I used an else if to use the given comparator to check is the head is greater than the given element. If so, the newElement's next is now the head, the head's previous is now the newElement, the head is now the new element and the list size increases by 1. I then used an else to check whether to add the element after the tail or between the head and tail. Using a while loop and the given comparator to see if the currentElement's data is less than the list's data. If so, I then checked if the currentElement's next was equal to null. This means we have reached the end of the list so now this currentElement would be the new tail. I then assigned the currentElement's next to the newElement, set the newElement's previous to the currentElement, the tail equal to the newElement and increased the list size by 1. Using an else, I then advanced the currentElement's cursor forward. Once the while loop has finished, I updated the currentElement's previous next (itself) to be the newElement. I also set the newElement's previous to the currentElement's previous, set the currentElement's previous to the newElement and the newElement's next to the currentElement. Finally, I increased the size of the list by 1 and returned that reference. For the addToEnd and addToFront method, both were straight forward and just needed to throw a new UnsupportedOperationException. For the iterator method, I called the super class iterator method. For the remove method, I called the super class remove method, implementing both given data and comparator.

## Test Runs:

### Basic Test

Add Names: Michael, Dan, Steve, Jack, Ben, Ryan

Contents of lists	
	Basic List
	Michael
	Dan
	Steve
	Jack
	Ben
	Ryan

Remove Dan

Contents of lists	
	Basic List
	Michael
	Steve
	Jack
	Ben
	Ryan

Get First Expected: Michael

Reality: Michael

Retrieve from List (deletes from list)	
Retrieved:	<div>Michael</div> <div>Retrieve First</div>

Get from List (doesn't deletes from list)	
Returned:	<div></div> <div>Get First</div> <div>G</div>

Remove from List	
To be Removed:	<div>Dan</div> <div>Remove</div>

Iterator (upon add, retrieve or remove, restart iterator)	
Returns:	<div></div> <div>Start</div> <div>Next</div>
<div>Has Next</div> <div>Has Previous</div>	

Contents of lists	
Basic List	Sorted List
<div>Steve</div> <div>Jack</div> <div>Ben</div> <div>Ryan</div>	

Get Last Expected: Ryan      Reality: Ryan

Get from List (doesn't deletes from list)		
Returned:	<input type="text" value="Ryan"/>	<input type="button" value="Get First"/> <input type="button" value="Get Last"/>

Retrieve Last Expected: Ryan      Reality: Ryan

Retrieve from List (deletes from list)	
Retrieved:	<input type="text" value="Ryan"/> <input type="button" value="Retrieve First"/>

Get from List (doesn't deletes from list)	
Returned:	<input type="text" value="Ryan"/> <input type="button" value="Get First"/> <input type="button" value="G"/>

Remove from List	
To be Removed:	<input type="text" value="Dan"/> <input type="button" value="Remove"/>

Iterator (upon add, retrieve or remove, restart iterator)		
Returns:	<input type="text"/>	<input type="button" value="Start"/> <input type="button" value="Next"/>
<input type="button" value="Has Next"/>	<input type="button" value="Has Previous"/>	

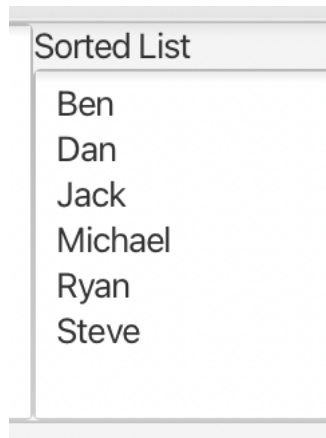
Contents of lists	
Basic List	Sorted List
Steve Jack Ben	

## Sorted

Add Names: Michael, Dan, Steve, Jack, Ben, Ryan

Expected: Sorted: Ben, Dan, Jack, Michael, Ryan, Steve

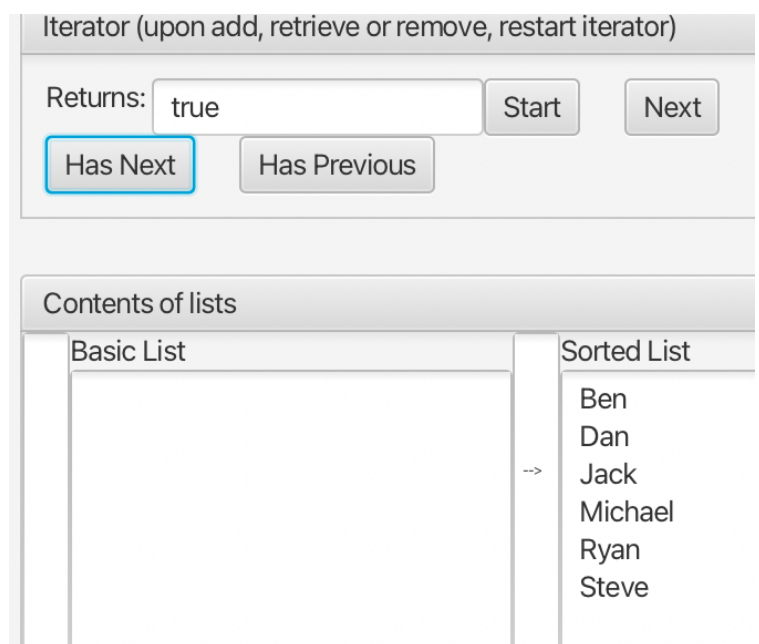
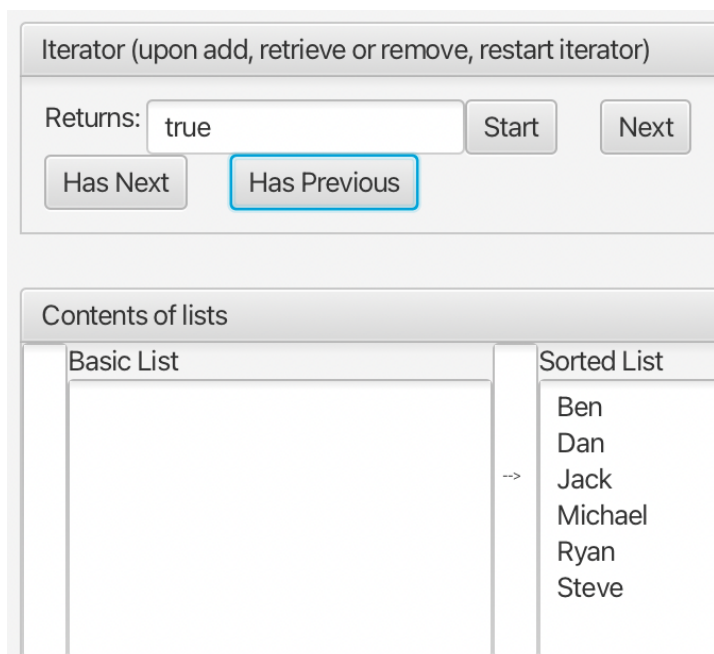
Reality: Sorted: Ben, Dan, Jack, Michael, Ryan, Steve,



Start: Jack



HasPrevious Expected: True Reality: True

HasNext Expected: True Reality: True




## Junits

Finished after 0.039 seconds

Runs: 14/14     Errors: 0     Failures: 0

- BasicDoubleLinkedListTest [Runner: JUnit 5] (0.000 s)
  - testGetFirst (0.000 s)
  - testRetrieveFirstElement (0.000 s)
  - testIteratorNoSuchElementExceptionPrevious (0.000 s)
  - testIteratorUnsupportedOperationException (0.000 s)
  - testIteratorSuccessfulPrevious (0.000 s)
  - testAddToEnd (0.000 s)
  - testGetLast (0.000 s)
  - testGetSize (0.000 s)
  - testAddToFront (0.000 s)
  - testIteratorNoSuchElementExceptionNext (0.000 s)
  - testIteratorSuccessfulNext (0.000 s)
  - testRetrieveLastElement (0.000 s)
  - testToArrayList (0.000 s)
  - testRemove (0.000 s)



Runs: 13/13

 Errors: 0

 Failures: 0

- SortedDoubleLinkedListTest [Runner: JUnit 5] (0.000 s)
  - testIteratorSuccessfulDoublePrevious (0.000 s)
  - testRemoveMiddleCar (0.000 s)
  - testIteratorNoSuchElementException (0.000 s)
  - testIteratorUnsupportedOperationExceptionString (0.000 s)
  - testIteratorSuccessfulCarPrevious (0.000 s)
  - testAddToEnd (0.000 s)
  - testRemoveEndCar (0.000 s)
  - testAddToFront (0.000 s)
  - testIteratorSuccessfulNext (0.000 s)
  - testRemoveFirstCar (0.000 s)
  - testAddCar (0.000 s)
  - testIteratorSuccessfulDoubleNext (0.000 s)
  - testIteratorSuccessfulStringPrevious (0.000 s)


Finished after 0.041 seconds


Runs: 13/13     Errors: 0     Failures: 0

- SortedDoubleLinkedList\_STUDENT\_Test [Runner: JUnit 5] (0.001 s)
  - testIteratorSuccessfulDoublePrevious (0.000 s)
  - testRemoveMiddleCar (0.000 s)
  - testIteratorNoSuchElementException (0.000 s)
  - testIteratorUnsupportedOperationExceptionString (0.000 s)
  - testIteratorSuccessfulCarPrevious (0.000 s)
  - testAddToEnd (0.000 s)
  - testRemoveEndCar (0.000 s)
  - testAddToFront (0.000 s)
  - testIteratorSuccessfulNext (0.000 s)
  - testRemoveFirstCar (0.000 s)
  - testAddCar (0.000 s)
  - testIteratorSuccessfulDoubleNext (0.000 s)
  - testIteratorSuccessfulStringPrevious (0.000 s)

Finished after 0.038 seconds


Runs: 14/14

 Errors: 0

 Failures: 0

- BasicDoubleLinkedList\_STUDENT\_Test [Runner: JUnit 5] (0.000 s)
  - testGetFirst (0.000 s)
  - testRetrieveFirstElement (0.000 s)
  - testIteratorNoSuchElementExceptionPrevious (0.000 s)
  - testIteratorUnsupportedOperationException (0.000 s)
  - testIteratorSuccessfulPrevious (0.000 s)
  - testAddToEnd (0.000 s)
  - testGetLast (0.000 s)
  - testGetSize (0.000 s)
  - testAddToFront (0.000 s)
  - testIteratorNoSuchElementExceptionNext (0.000 s)
  - testIteratorSuccessfulNext (0.000 s)
  - testRetrieveLastElement (0.000 s)
  - testToArrayList (0.000 s)
  - testRemove (0.000 s)


Finished after 0.049 seconds


Runs: 1/1     Errors: 0     Failures: 0

- BasicDoubleLinkedList\_GFA\_Test [Runner: JUnit 5] (0.000 s)
  - testGetSize (0.000 s)

Finished after 0.029 seconds

Runs: 1/1

 Errors: 0

 Failures: 0

- SortedDoubleLinkedList\_GFA\_Test [Runner: JUnit 5] (0.000 s)
  - testAddToEnd (0.000 s)

### **Learning Experience:**

I have learned a lot with this project. I have learned a lot about using previous and next with nodes along with using Comparator's. I got very comfortable creating new nodes to hold new elements, using those nodes with a comparator to find out if the two elements are greater than, less than or equal to each other. One big takeaway is using prev.next for the add method in the SortedDoubleLinkedList.. I managed to get majority of the test cases right except for about 4 of them. I would say this was where majority of my headache occurred. Everything made sense with added an element before a head or after a tail but adding it between a head and tail was the tricky part of the assignment. Realizing prev.next calls the node itself and knowing when to appropriately place these statements was hardest part I felt. I realized that my list was not deleting the actual typed in name and narrowed it down to my remove method. I realized I needed to use prev.next to assign the next position Overall, I felt pretty good about this assignment and it was good to learn the different techniques of prev.next and when and how to use them. I also learned that it is the best practice to do all your next or prev statement prior to declare the end change (ex. Changing a new element to be the tail).

### **Assumptions:**

In the SortedDoubleLinkedList class, for the method add, I had to change the return type to the SortedDoubleLinkedList<T> instead of void. For some reason, I kept failing three of the Junit tests. I kept failing where the tests would try and grab the current size of the list. I realized the only way to update the size of the current list was to change the return type of the add method. In the test RemoveMiddleCar, I realized the add method was used here which would allow it to report the current list size if I changed the return type from void to SortedDoubleLinkedList<T>. Not sure if this what we were meant to do this but that's the only way I got the code to work with these few test cases.

### **Observations:**

SLOW DOWN AND THINK AHEAD. I realized the importance of order in this assignment. Even the slightest change of prev or next in your statements can completely alter the code. I need to methodically and slowly think my way through each if else if statement, while loop etc.