Brandon Tenorio

CMSC204

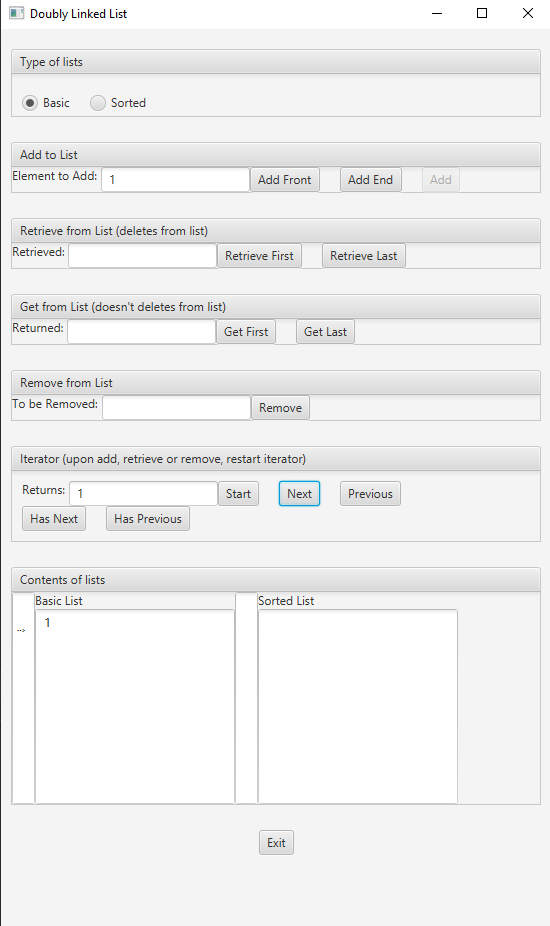
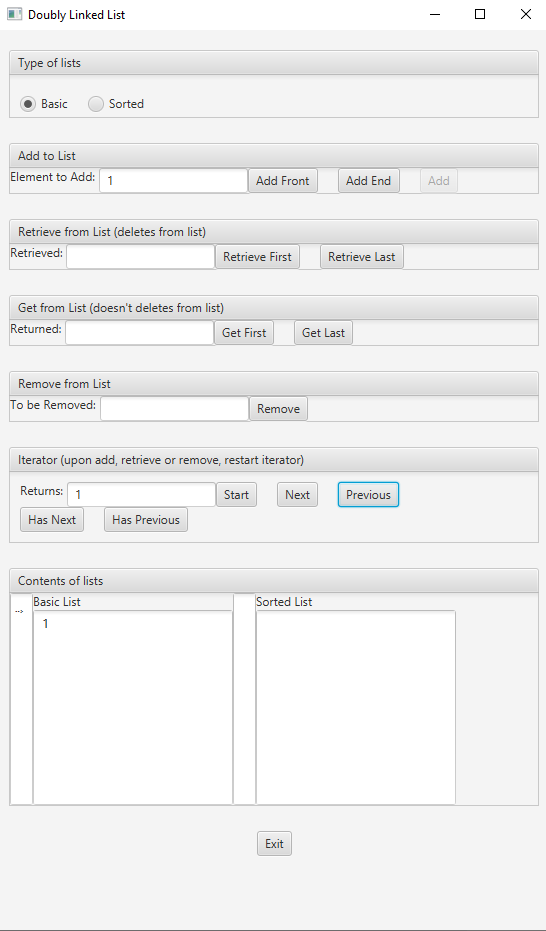
Gary C. Thai

Fall 2021

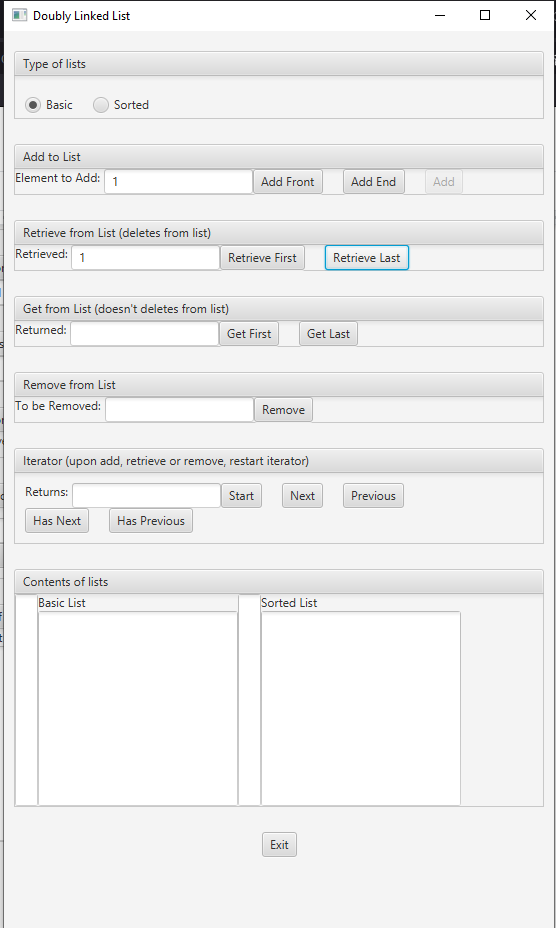
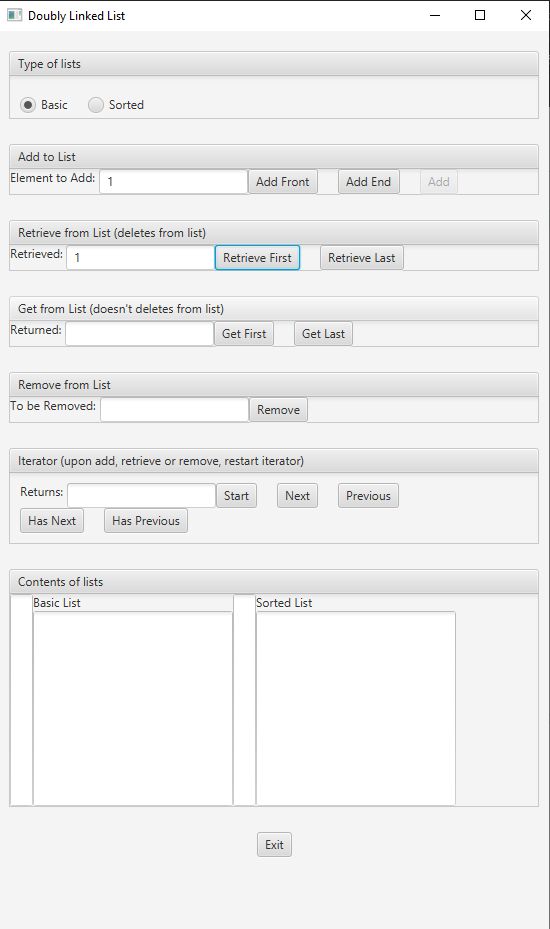
Project 3

Tests

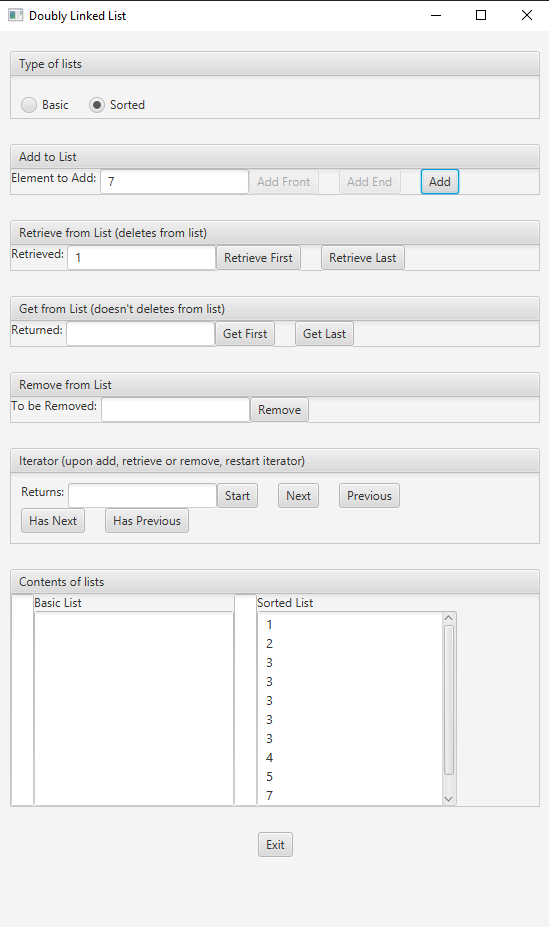
Test 1: Consecutive calls to next() then previous() returns the same number; Passes



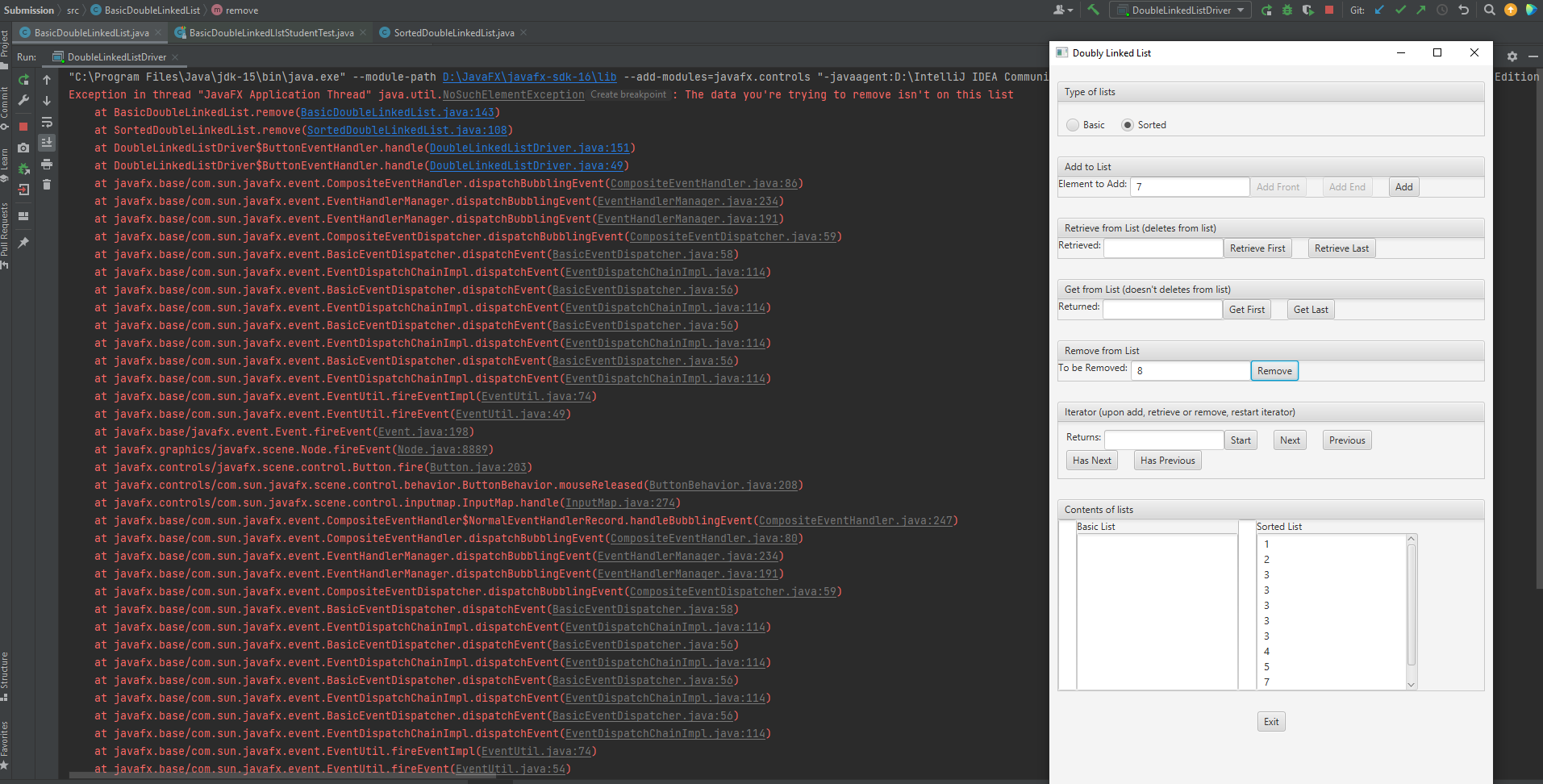
Test 2: Adding one to the front of a basic list returns one when retrieving both first and last element; Passes

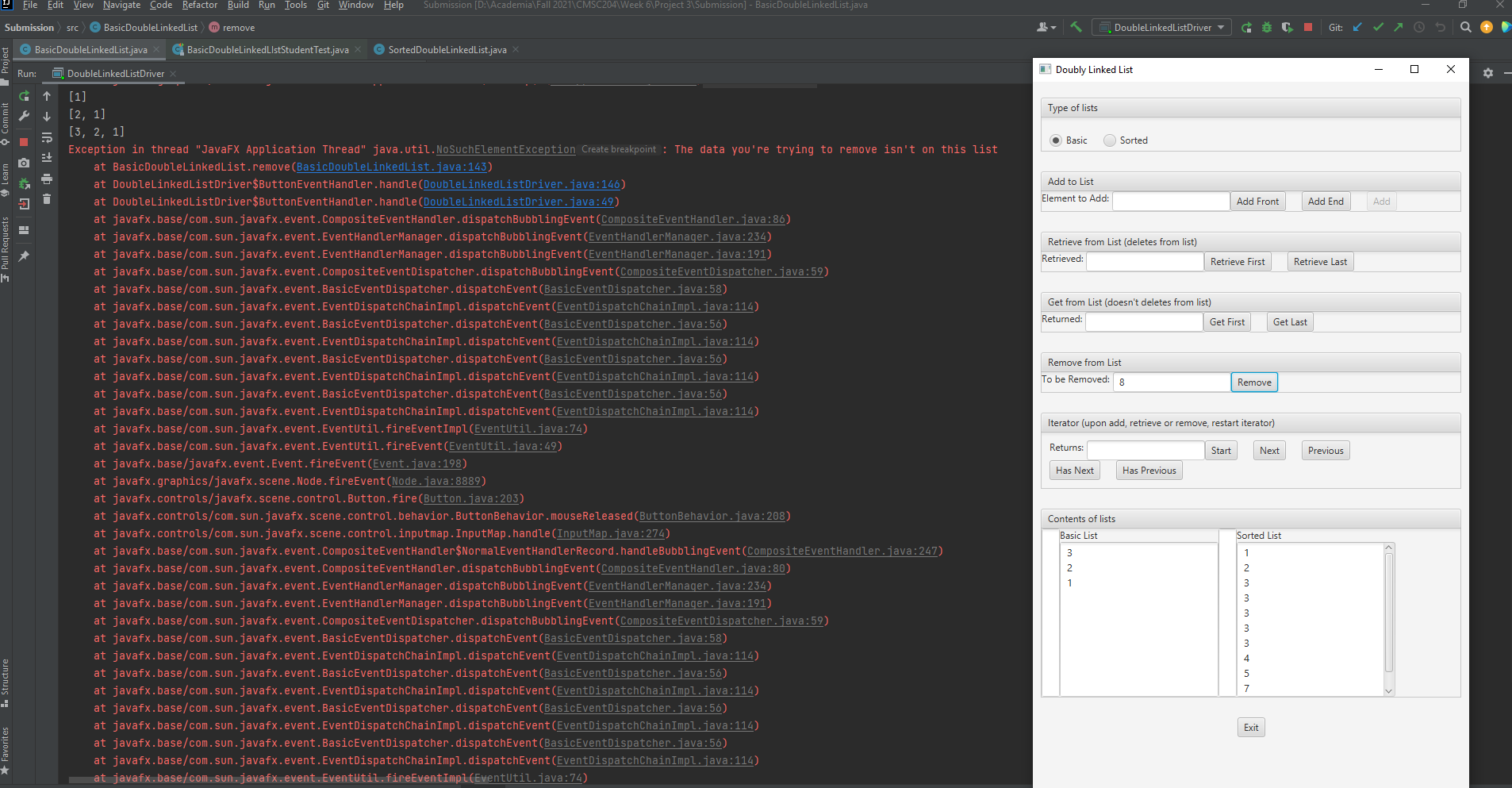


Test 3: Adding the same element to a sorted list doesn’t affect the sorting, and all elements go on the correct spot; Passes

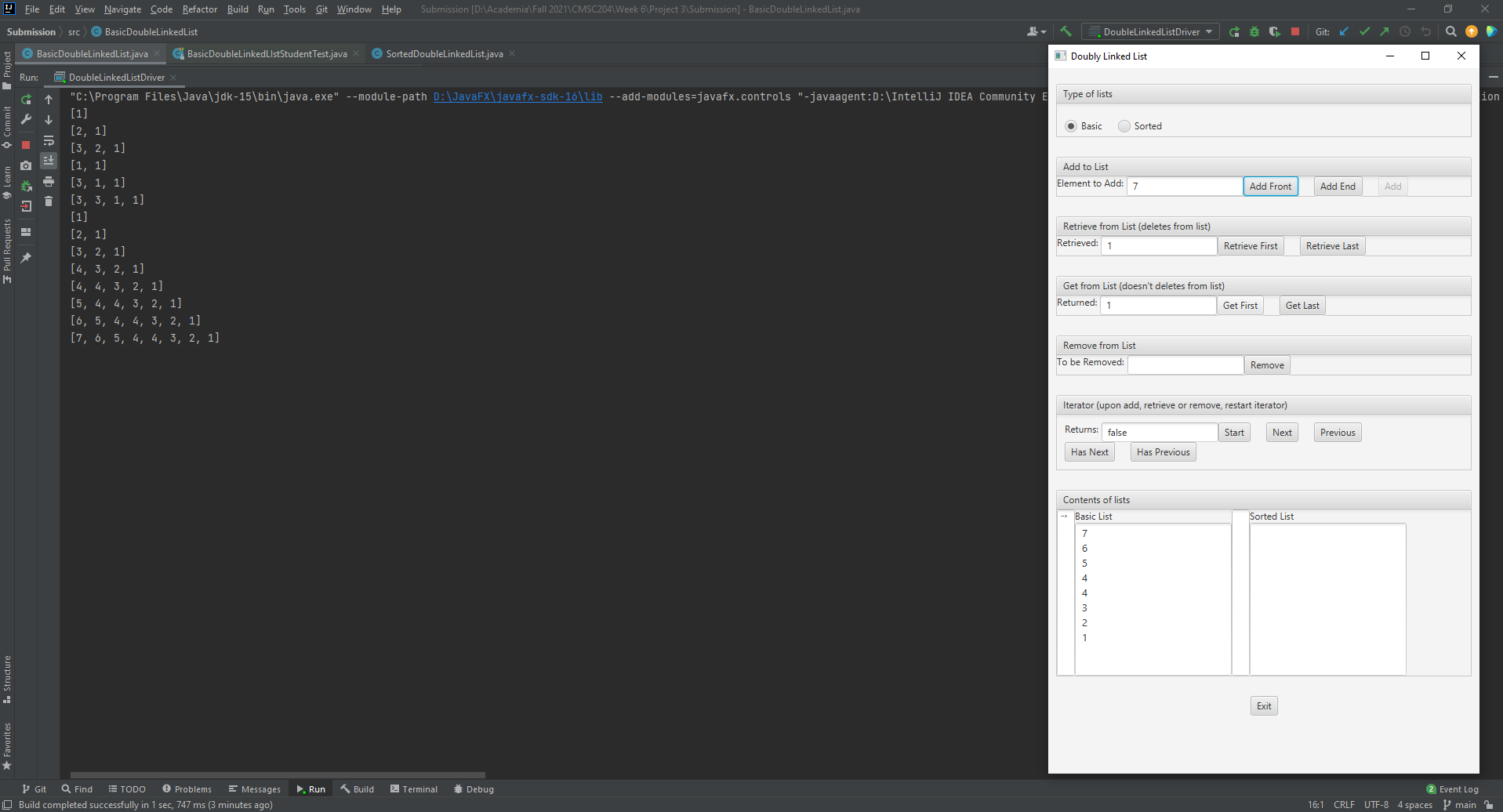


Test 4: Attempting to remove an object that is not on the list throws a NoSuchElementException; passes





Test 4: retrievingFirstElement and then adding and then retrieving first element works with no problem; passes



Discussion:

This project was one of the easiest we’ve had to do so far. One of the hardest parts I encountered while working on this project is implementing the ListIterator methods hasNext(), next(), hasPrevious(), and previous(). The difficult part was coding the boolean methods of this class in such a way that they would never fail. It was hard to implement the logic of this method because at first I was only keeping track of one node in the list, which worked perfectly fine for next() and hasNext(), but it wouldn’t allow previous() and hasPrevious() to work as intended. After a long day of thinking about it, I realized that I can have two pointers -- one to look at the current node, and another one to look at the node before the current node. This simplified the implementation by ten-fold. After this revelation, it was finally possible to use previous() after having called next() n times, where n is the size of the linked list; this wasn’t possible before this discovery, and it’s a feature that I’m very proud of. While thinking about possible private cases, and more specifically special cases, to take into account and test my code, I came up with the realization that perhaps my code wouldn’t work if there were only one element on the list. Thus I created a boolean method called thereIsOnlyOneElementInTheList() to check for this special condition. The importance of this special; case cannot be overlooked. Certain methods, like all the addition and retrieval methods as well as the iterator methods, were created by me with the preconceived notion that this list would hold more than one element; however, having one element on the list and only having methods that account for there being more than one, means that my code is prone to throwing NullPointerExceptions at any moment. Having a boolean method to see if there is only one element in the list increases readability and minimizes room for unexpected exceptions. A feature that I added in many methods of this project is exception handling. Methods like next(), previous(), remove(), and both retrieval methods throw a NoSuchElementException if they are unable to find the element that the user would like to retrieve. As a closing statement I’d like to say that I am very proud of this project. I tried my best to break my own code -- as you can probably see from the many tests I created --, and in doing so I was able to find certain gaps in logic that I would’ve otherwise never have found. I believe the readability of my code is quite high as well, which is a quality that I’ve come to understand is of utmost importance in the field of software development. The most important thing I learned while doing this project is coming up with an extensive testing plan to break my own code and find what is wrong with it.