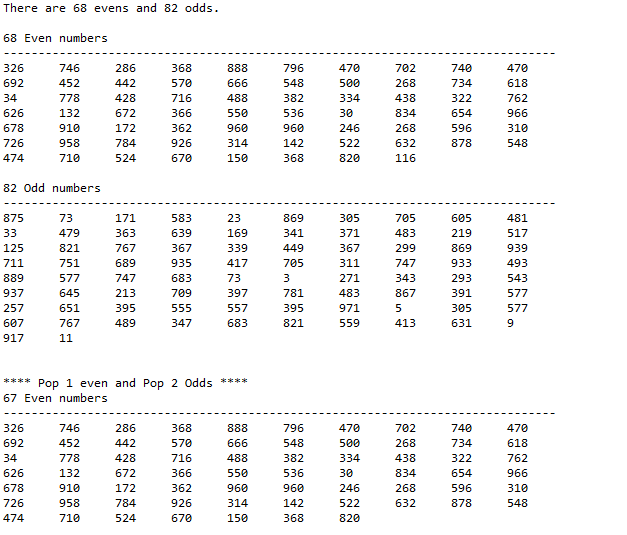
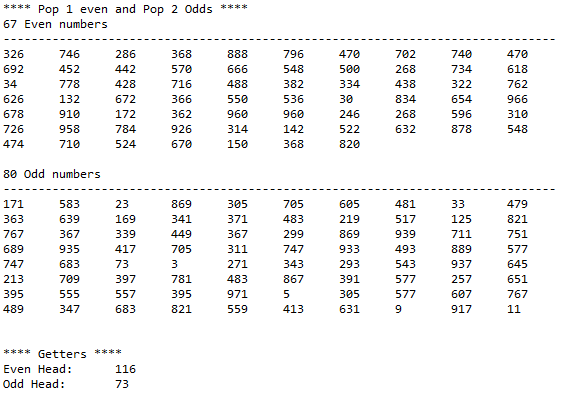
Assignment 5 Reflection:

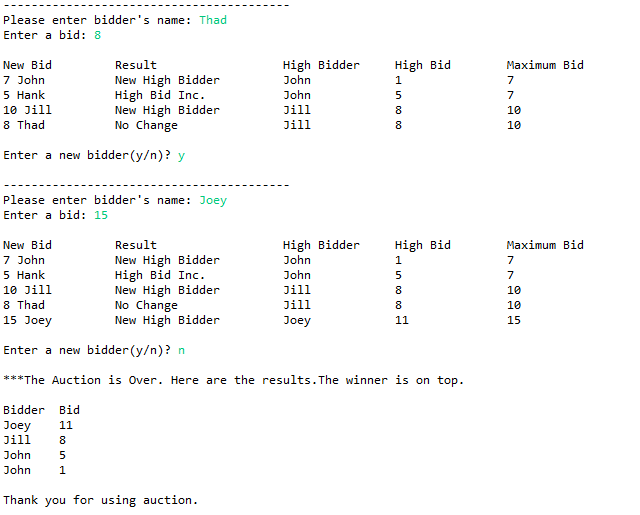
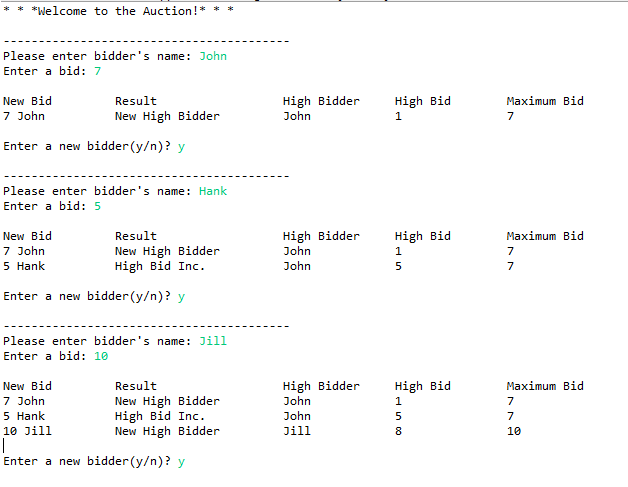
All of the requirements for Assignment 5 are complete. Most of the code for the positive integers program was types from scratch. A majority of the code for the auction program was used from the chapter 3 exercises, and modified to fit the Auction.java.

In the positive integers program, I used the Random function to generate 150 random numbers. I stored both even and odd numbers in a single array, but stored the even numbers starting at the beginning of the array, and the odd numbers starting at the end of the array.

For the auction program, I used the programs that I typed during the chapter 3 explaining how the StackedLists worked. Because of this, I really only had to develop 2 classes, Bidder and Auction. The bidder would store information on a bidder who took position in the auction. If a bidder bid below the max bid of the previous bidder, that bidder was not processed with the bidder class. The Auction class was the demo for the overall program.

**DoubleIntegerStackDemo.java**

**Auction.java**



**Assignment3Project Junit Testing**

|  |  |  |  |
| --- | --- | --- | --- |
| **Methods to Test** | **Test Cases** | **Expected Result** | **Actual Result** |
| removeSecond | |  |  |  | | --- | --- | --- | | **Test Case 1** | **Line 2 is removed** | **Line 2 is removed** | | **Test Case 2** | **None** | **None** | | **Test Case 3** | **Line 2 is removed** | **Line 2 is removed** | | | |
| addLast | |  |  |  | | --- | --- | --- | | **Test Case 1** | **Member added** | **Member added** | | **Test Case 2** | **Member added** | **Member added** | | **Test Case 3** | **n/a** | **n/a** | | | |
| retrieveEveryOtherItem | |  |  |  | | --- | --- | --- | | **Test Case 1** | **Return 1,3,5** | **Return 1,3,5** | | **Test Case 2** | **Return 1** | **Return 1** | | **Test Case 3** | **n/a** | **n/a** | | | |

* The results in removeSecond prove functional because it the second membership gets removed
  + Test Case 1: 4 members in linked list, member 2 was removed
  + Test Case 2: 1 member, no number 2 to remove, therefore none
  + Test Case 3: 2 in linked list, member 2 was removed
* addLast was functional as it added a member to the end of the list
  + Test Case 1: add member to linked list with 5 members
    - Member added to end, link = null
  + Test Case 2: 0 members begin
    - Member added to head/end
* retrieveEveryOtherItem
  + In a list of 6 members, members 1,3,5 were retrieved
  + In a list of 2 members, member 1 was retrieved