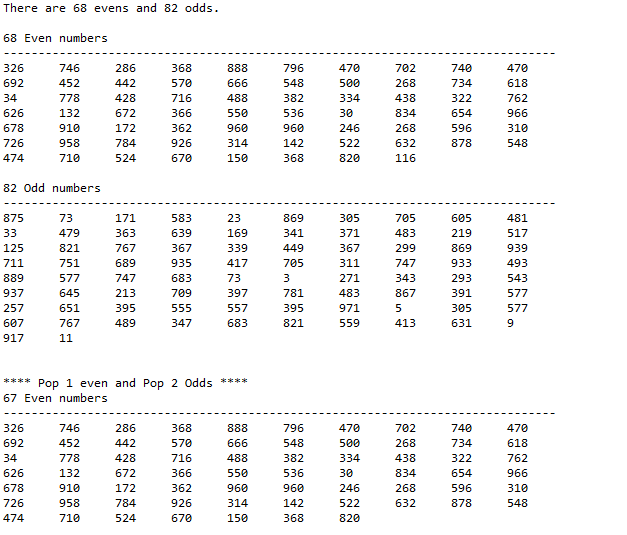
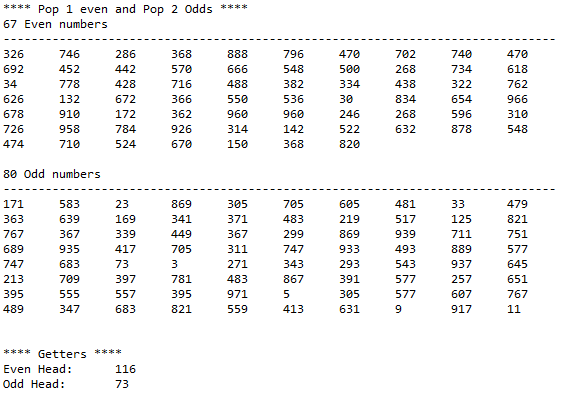
Assignment 5 Reflection:

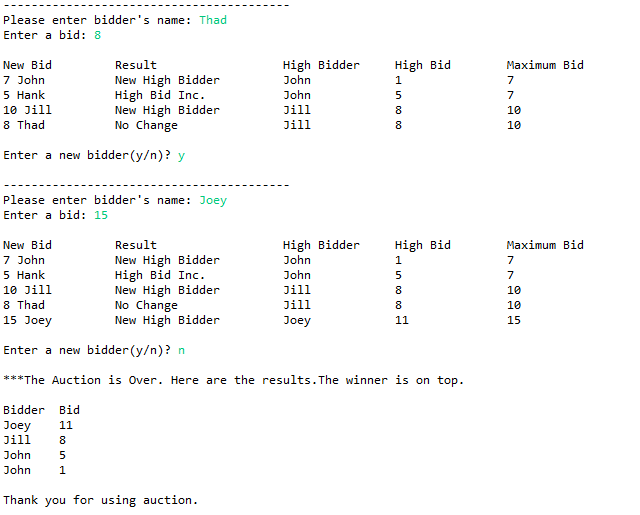
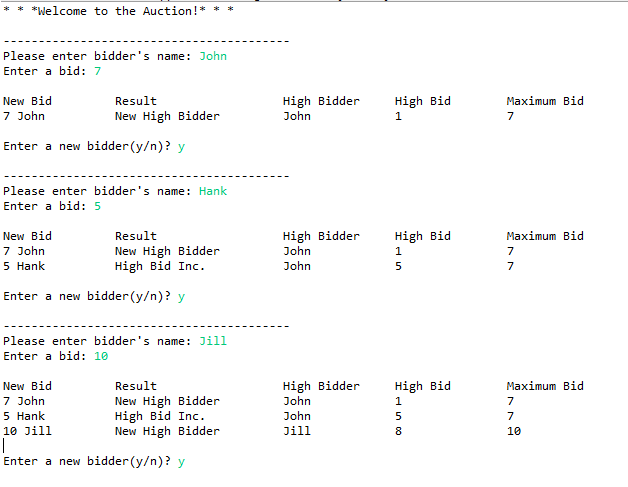
All of the requirements for Assignment 5 are complete. Most of the code for the positive integers program was types from scratch. A majority of the code for the auction program was used from the chapter 3 exercises, and modified to fit the Auction.java.

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** |
| Do{}While | |  |  |  | | --- | --- | --- | | **Test Case 1** | **Add member with bid below current bid and max** | **Member is not added to history** | | **Test Case 2** | **Add member with bid below max but above current bid** | **Current bid is raised for current max bidder** | | **Test Case 3** | **Add member with bid above max bid** | **Member is set to highest bidder, current bid is raised** | | | |

* The easiest way to check the Positive Integer program was to generate random numbers and check to make sure that the even numbers and odd numbers were sorted properly.

* The Auction program was tested as follows:
  + The main part of the program was a do{ }while that processed one bid and then additional bids as the user chose
    - The first test was to add a member that had a bid below the max bid of the previous user, and also below the current bid
      * The result: The member was printed to the summary of bids, but was not printed in the history, as they did not affect the actual bidding process
    - The second test was to add a member that bid above the current bid, but not above the max bid
      * The result: The member was printed to summary of bids, and the current high bidder had their current bid upped to the new the bid
    - The last test was to add a member that bid above the max
      * The result: The member was added to the history of bids, and was also updated to the highest bidder