Full name : Pham Van Phuc

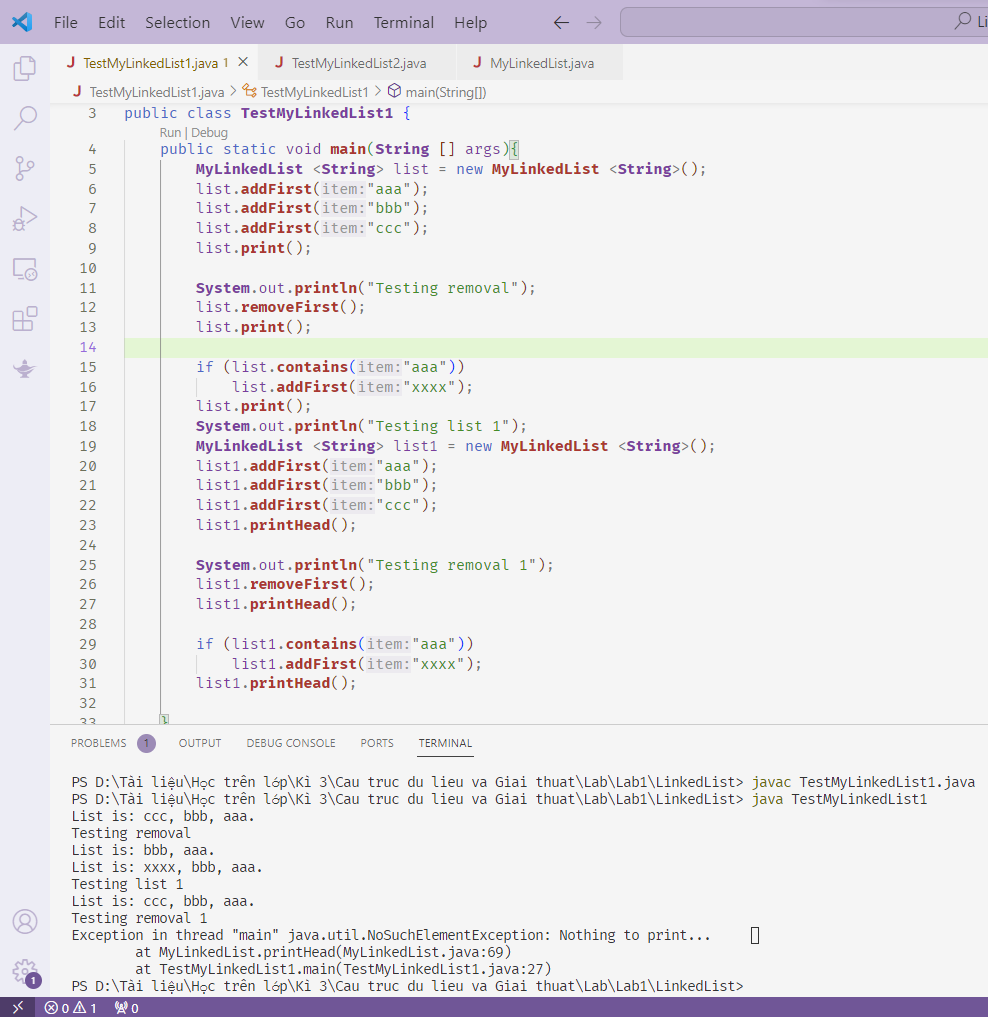
Student ID : 522H0068

LAB1

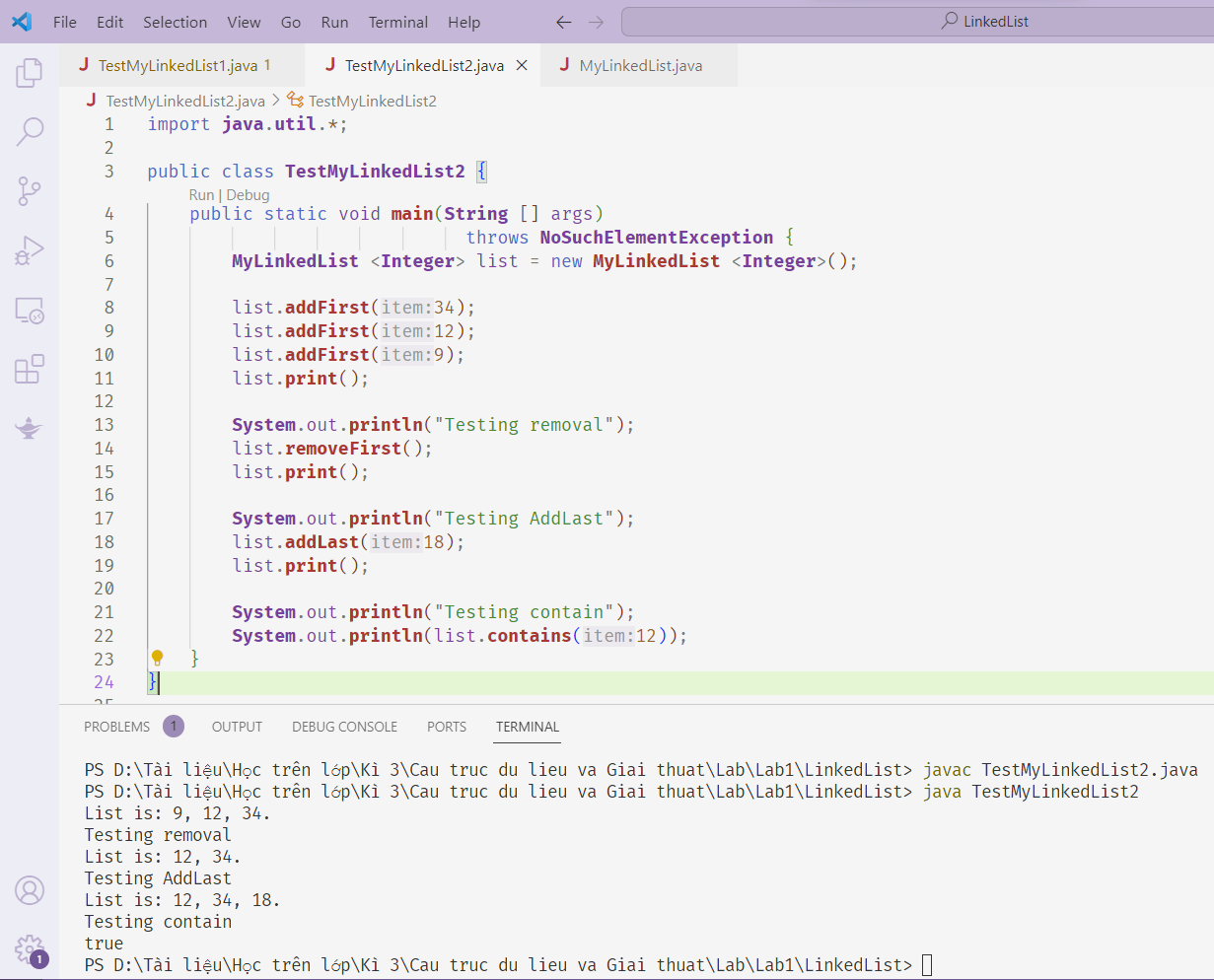
I. Classwork

1. Here is the run screen after I edited the source code provided as requested

Run file TestMyLinkedList1 with strings

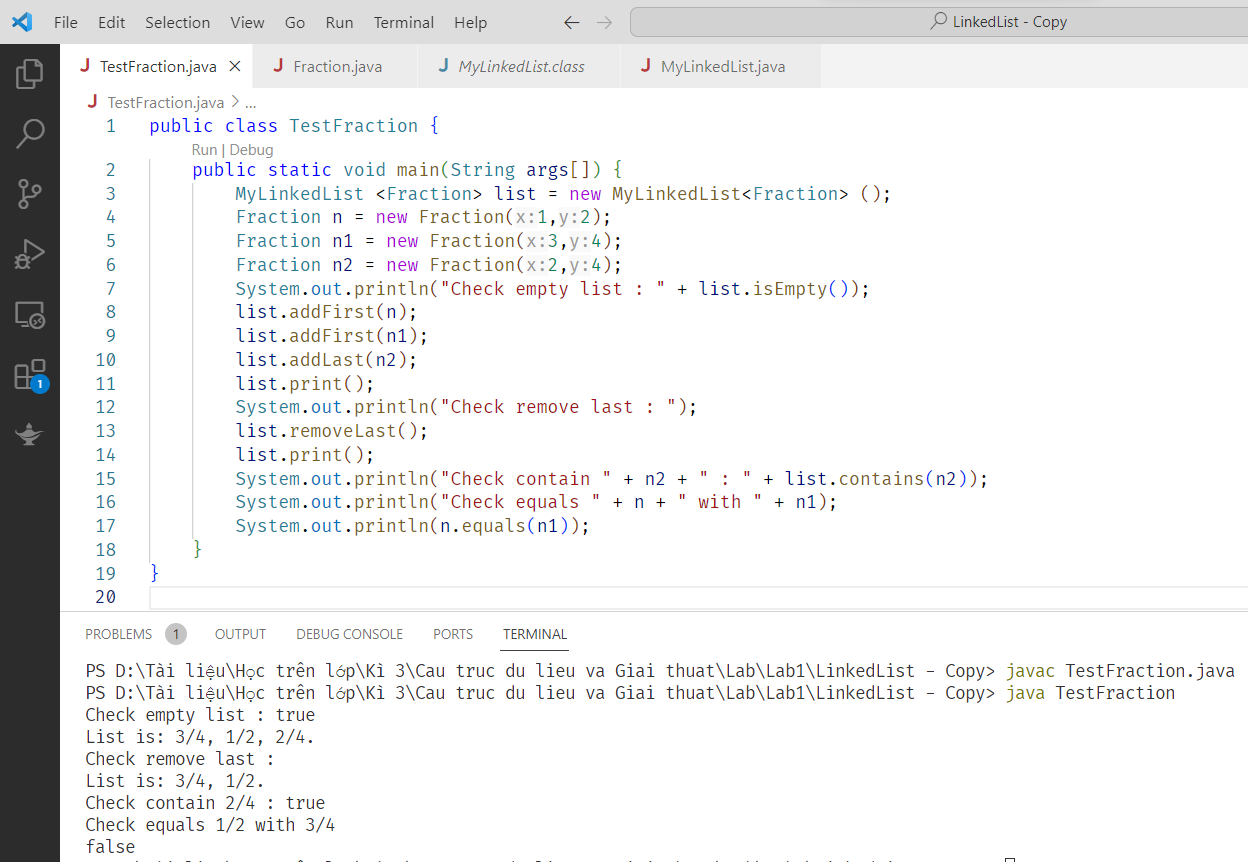


2.Run file TestMyLinkedList2 with Integer



II.Homework

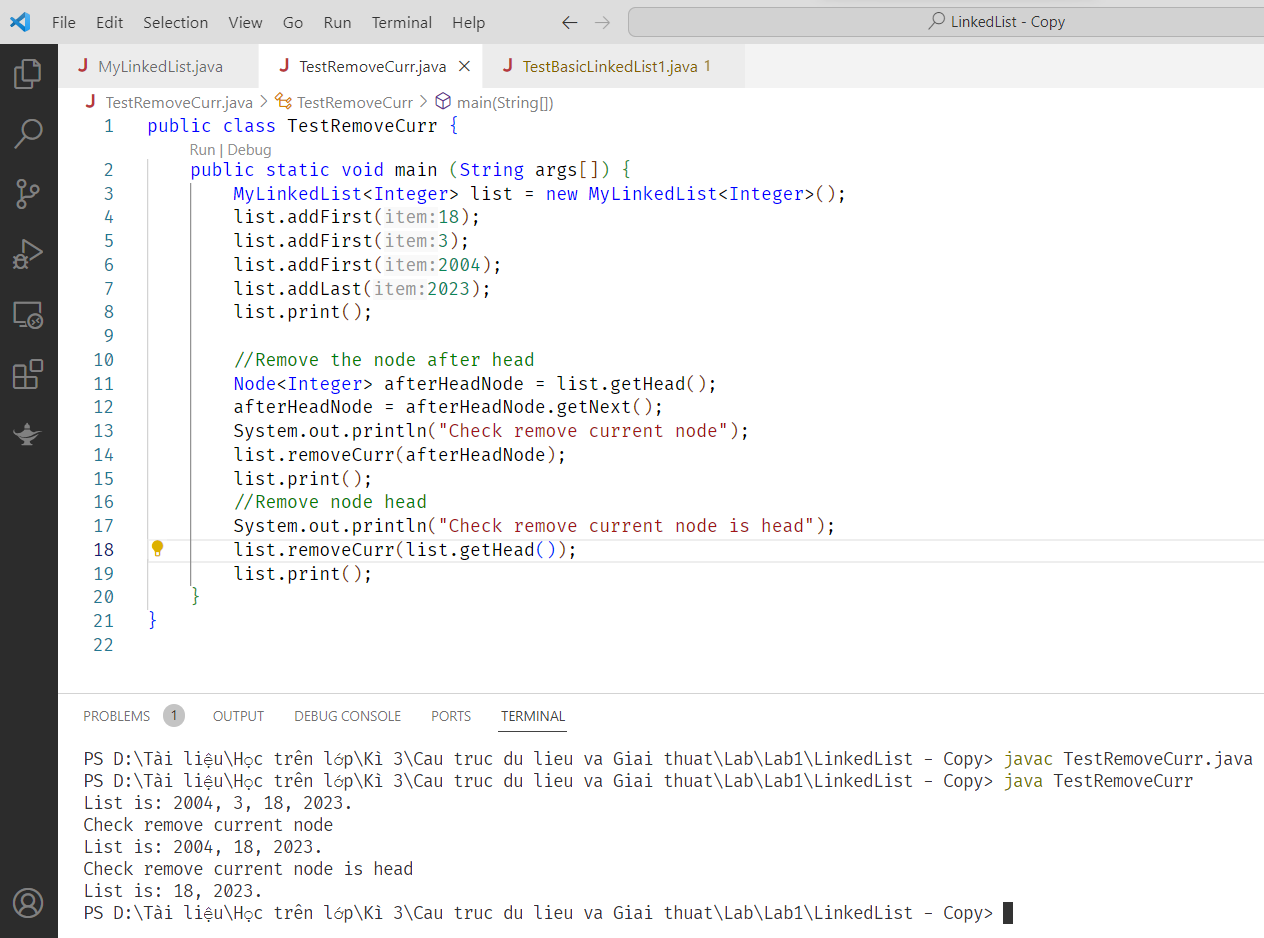
Exercise 1 : Implements the Fraction class that manipulates fractions



Exercise 2 : Remove node at position curr

First I check if the node I want to delete is head then call the removeFirst method. If not, use a while loop to find the position before the node you want to delete. The previous node's next assignment points to the next node.

Run results screen:



Exercise 3

Suppose we are having a list of integer numbers, do the following requirements:

1. Count the number of even item in the list.   
   I created a countEven function of type int. In that function, I went through each node of the list. If any node has an even number of data, then increase the counter variable by 1. Then return the value of the counter variable.

(b) Count the number of prime item in the list.   
First I create a sub function called checkPrime with an int parameter to check whether the integer is prime or not. Then in the countPrime method, I go through each node and take the value of that node and put it in the checkPrime function to check. If it is a prime number, increase the counter variable by 1. Then return the value of the counter variable.

(c) Add item X before the first even element in the list.

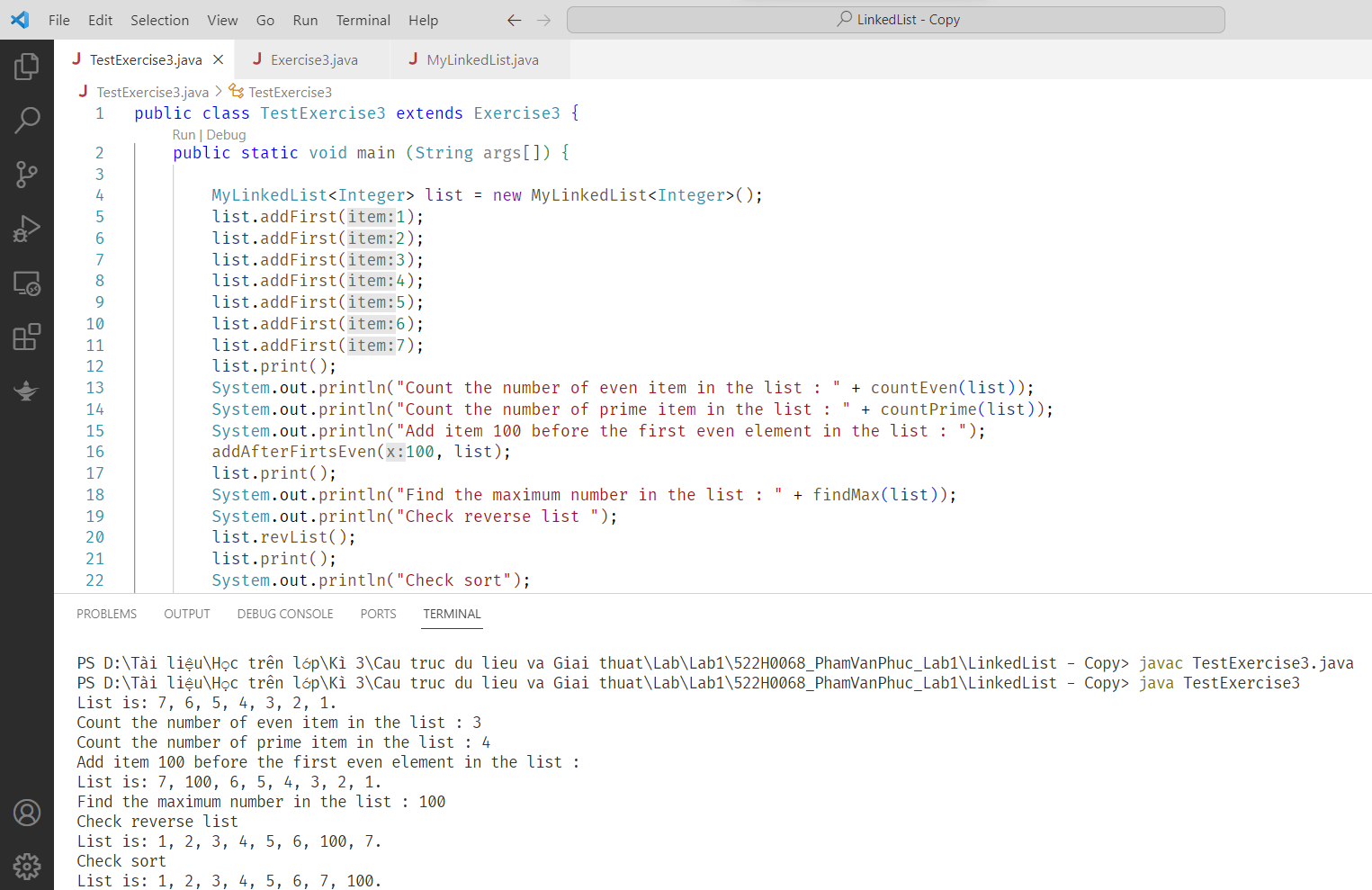
I use node curr to find the first even element in the list. If curr is a node with an even number of data, the addAfter function will be called with node tmp (Because node tmp is the node that comes after curr, it must be added after node tmp to be able to add it before the first even number). (d) Find the maximum number in the list.

I go through each node of the list. Assign the max variable to the first node's data, then browse from the second element and compare that node's data with max to find the largest value.

(e) (\*) Reverse the list without using temporary list  
Start the while loop, looping through each element in the linked list (starting at the beginning of the list and working your way through the last element). Store the next pointer of currNode in tmp. Then, reverse the association of currNode so that it points to prevNode. Then, update prevNode to point to currNode. Update currNode to point to the next element. After the loop finishes executing, assign head to point to the last element

f) (\*) Sort the list in ascending order.  
I create a new list called sortedList. Then use the curr pointer to browse through each element of the original array. Get values ​​from the original list in order then compare them with the nodes inside the sortedList to place the node in the appropriate position.

Run results screen:



Exercise 2 : Double Linked List

Deployed with two nodes head and tail. The operations are similar to BasicLinkedList

Run results screen:

