## Com S 228

## Fall 2016

## Exam 2 Sample Solution

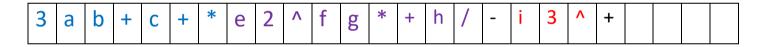
1.

Code snippet	Output	List and iterator state, or exception thrown
<pre>iter = aList.listIterator();</pre>	(none)	A B C D E
<pre>// 2 pts aList.add("X"); aList.remove("F");</pre>	(none)	ABCDEX
<pre>// 4 pts iter = aList.listIterator(2); iter.previous(); iter.remove(); System.out.println(iter.previousIndex());</pre>	0	A B   C D E A   <u>B</u> C D E <b>A   C D E</b> (answer)
<pre>// 5 pts iter = aList.listIterator(aList.size()); while (iter.hasPrevious()) {    iter.previous();    if (iter.hasNext())        System.out.println(iter.next());    iter.previous(); }</pre>	E D C B	A B C D E   (start)  1st iteration:  A B C D   E iter.previous A B C D E   iter.next A B C D   E iter.previous    A B C D E (answer)
<pre>// 4 pts aList.remove("A"); iter = aList.listIterator(); iter.next(); iter.set("X"); iter.remove(); iter.set("Y");</pre>	(none)	IllegalStateException
<pre>// 6 pts iter = aList.listIterator(); iter2 = aList.listIterator(1); while (iter2.hasNext()) {    iter.next();    iter.set(iter2.next());    System.out.println(iter.previous()); }</pre>	B C D	A   B C D E (start)  1st iteration: A   B C D E iter.next A   B   C D E iter2.next B   B   C D E iter.set   B B   C D E iter.previous  after 2nd iteration:   C B C   D E

```
2a) i) O(n); ii) O(n)
b) O(n)
```

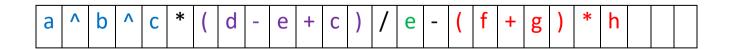
3a) Infix:

Postfix:



b) Postfix:

Infix:



4.

```
import java.util.NoSuchElementException;
```

```
// This class is parametrized with the type E, whose superclass must have // implemented Comparable. Fill a wildcard type in the blank below to make // comparison possible between two objects of type E.
```

```
public class StackReorder<E extends Comparable <? super E>> // 5 pts
{
       * Search for all the elements on a stack that are equal to key. Reorder
       * the stack such that these elements are on the top while preserving their
       * original order. The remaining elements not equal to key must preserve
       * their original order as well.
       * @param stk stack
       * @param key
      public void keyOnTop(PureStack<E> stk, E key)
            // initialize
            //
                  a) a stack as an object of the ArrayBasedStack class;
            //
                  b) a list as an object of the inner SimpleList class.
            // fill in the blanks on the right hand sides of the next two
            // assignments.
            PureStack<E> tempStk = new ArrayBasedStack<E>();
                                                                  // 1 pt
            SimpleList tempList = new SimpleList();
                                                                  // 1 pt
            // store
            //
                  a) elements from the stack stk equal to key on the stack
                     tempStk,
                  b) other elements from the stack in the list tempList.
            //
            // insert code below (6 pts)
            E element:
            while (stk.size() != 0)
            {
                  element = stk.pop();
                  if (element.compareTo(key) == 0)
                        tempStk.push(element);
                  else
                        tempList.add(element);
            }
            // merge elements from tempStk and tempList onto the stack stk.
            // insert code below (6 pts)
            while (!tempList.isEmpty())
                  stk.push(tempList.remove());
            while (tempStk.size() != 0)
                  stk.push(tempStk.pop());
      }
      // singly-linked list for temporary storage
      private class SimpleList
      {
            private Node head;
            private int size;
```

```
/**
* default constructor
public SimpleList()
     // insert code below (3 pts)
     head = new Node(null);
     size = 0;
}
* Create a new node to contain a provide item. Insert the node to the
* front of the list.
* @param item to be added
void add(E item)
   // insert code below (4 pts)
   Node temp = new Node(item);
   temp.next = head.next;
   head.next = temp;
   size++;
}
* Remove the first node.
* @return stored in the removed node if the list is not empty
* # @throws IllegalStateException if the list is empty
E remove() throws IllegalStateException
{
     // check if the list is empty.
      // fill in the blank below (1 pt)
     if (size == 0)
               // insert code below (2 pts)
               throw new IllegalStateException();
      }
     // size != 0
      // insert code below (6 pts)
     Node temp = head.next;
     E val = temp.data;
     head.next = temp.next;
     size--;
      return val;
}
```

```
/**
  * @return true if the list is empty
  */
boolean isEmpty()
{
    // insert code below (2 pts)
    return size == 0;
}

// fully implemented class
public class Node
{
    public E data;
    public Node next;

    Node(E data)
    {
        this.data = data;
    }
}
```

}