The exam is over:

- Generics,
- Linked lists,
- List<E> Abstract Data Type,
- ArrayList<E>,
- LinkedList<E>,
- Stack<E> Abstract Data Type.

You should also review all the code examples that we covered in class.

1. The following two lines both generate compiler warnings. What is wrong with them?

```
Stack<String> stack1 = new Stack();
Stack stack2 = new Stack<String>();
```

2. The following line generatea a compiler error. What is wrong?

```
List<String> list = new List<String>();
```

3. While `String` is a sub-type of `Object` (that is, every `String` object "is a" `Object` object), it is **not** true that `List<String>` is a sub-type of `List<Object>`. Explain why. (**Hint:** If `B` "is a" `A`, then everything you can do with `A` you can also do with `B`. What can you do with a `List<Object>` that you cannot do with a `List<String>`?)

4. Here is part of the definition for a `LinkedList` class.

```
public class LinkedList
{
   private Node head;
   private int size;

   public LinkedList()
   {
      this.head = null;
      this.size = 0;
   }

   // LinkedList methods...
}


class Node
{
   public int item;    // An item in the list.
   public Node next;   // Reference to next item in the list.
}
```

(a) Write the method

```
public void add(int element)
```

that adds a new node at the head of the linked list. (Notice that the class `Node` only has a default constructor.)

(b) Write the method

```
public int remove( )
```

that removes from the linked list the node at the head of the list and returns the `int` that was stored in that node. Throw an exception if the linked list is empty.

(c) Explain how you would modify the `add` method so that the following two lines of code will compile and run correctly.

```
LinkedList list = new LinkedList();
list.add(3).add(2).add(5, 6, 7).add(0).add(8);
```

5. On the last page of these review problems is an implementation of a `Node` class. Use that `Node` class to implement the following static methods.

   (a) Write an implementation of the static method

   ```
   public static int countZeros( Node<Integer> head )
   ```

   that will count the number of zeros that occur in the given linked list of ints.

   (b) Write an implementation of the static method

   ```
   public static String list2String( Node<Integer> head )
   ```

   that returns a `String` representation of the linked list referred to by the parameter `head`. If the linked list is empty, the `String` representation should be `"[]"` (two square brackets next to each other). If the linked list is not empty, the `String` representation should look like this, `"[ 3 52 0 2 -4 16 ]"`, with a space before and after each entry of the list.

   (c) Write an implementation of the static method

   ```
   public static <E> Node<E> getThirdNode( Node<E> head )
   ```

   that returns a reference to the second node after the node referred to by the parameter `head` (you can assume that node does exist).

   (d) Write an implementation of the static method

   ```
   public static <E> void duplicateNode( Node<E> head )
   ```

   that inserts into the linked list a copy of the node referred to by `head` right after `head`.

   (e) Write an implementation of the static method

   ```
   public static <E> void set( E element, int i, Node<E> head )
   ```

   that modifies the linked list referred to by the parameter `head` so that the `i`'th node in the linked list has its data changed to `element` (the `0`'th node is the node referred to by `head`). If there is no `i`'th node in the linked list, then the list is not modified.

6. Once again using the `Node` class from the last page, consider the following three lines of code.

```
Node<Integer> hd = new Node<>(4,new Node<>(7,new Node<>(5,new Node<>(3,null))));
Node<Integer> ptr = hd.getLink().getLink();
hd.getLink().setLink( new Node<>(22, null) );
```

   (a) Draw a picture of Java's memory after the first line above has been executed. Be sure to include what data is in each node.

   (b) Draw a picture of Java's memory after the first and second lines above have been executed.

   (c) Draw a picture of Java's memory after all three lines above have been executed.

   (d) What would be a `String` representation for the linked list referred to by `hd`?

   (e) What would be a `String` representation for the linked list referred to by `ptr`?

   (f) What would be a `String` representation for the linked list referred to by `ptr` after executing the following line (which should be executed after the above three lines)?

   ```
   ptr.getLink().setLink( hd.getLink() );
   ```

7. Suppose we implement a stack using a partially-filled array. What is wrong with storing the top-of-stack at location `[0]` and the bottom of the stack at the last used position of the array?

8. If we use a linked list with a head reference to implement a stack, which of the following is better and why? Having the top-of-stack at the head of the linked list, or having the top-of-stack at the end of the linked list?

9. Suppose we perform the following series of stack operations on a single, initially empty stack:

   $push(5), push(3), pop(), push(2), push(8), pop(), pop(), push(9), push(1), pop(), push(7), push(6), pop(), pop(), push(4), pop(), pop()$.

   Draw a picture of the stack at the point where it contains the maximum number of elements (be sure to indicate the top and bottom of the stack). How many of the above operations had been performed at that point?

10. Suppose you have three stacks `s1`, `s2`, and `s3`. Suppose that `s1` contains `(1 2 3)`, with `1` being the top-of-stack, and `s2`, `s3` are both empty. Using no other variables and/or constants, and only the `push()` and `pop()` stack operations on the `s1`, `s2`, and `s3` variables, write a sequence of operations that leave the three stacks with each of the following contents.
    (a) Leave the stack `s2` with the contents `(1 2 3)`, with `1` as top-of-stack, and `s1`, `s3` both empty.
    (b) Leave the stack `s1` with the contents `(3 2 1)`, with `3` as top-of-stack, and `s2`, `s3` both empty.
    (c) Leave the stack `s2` with the contents `(1)`, leave the stack `s3` with the contents `(2 3)`, with `2` as top-of-stack, and leave `s1` empty.

11. Here is an incorrect pseudo code for an algorithm which is supposed to determine whether a String of parentheses is balanced: Give an example of an input string that is made up of only the characters `'('` and `')'`, is unbalanced, but for which this algorithm will return true. Explain what is wrong with the algorithm. Can this algorithm ever incorrectly return false when its input string is a balanced string?

```
boolean isBalanced( String input )
{
   declare a character stack
   while ( input has more characters )
   {
      read a character from input
      if ( the character is a '(' )
         push it on the stack
      else if ( the stack is not empty )
         pop a character off the stack
      else
         return false
    }
    return true
}
```

12. Using the two stack algorithm for evaluating fully parenthesized infix expressions, draw the contents of the two stacks just after the `'4'` token has been read from the following input string and processed by the algorithm.

```
"( ( ( 2 * 3 ) * ( 9 + ( ( 3 - 1 ) + 4 ) ) ) * ( 5 - 1 ) )"
```

```
class Node<E>
{
   private E data;
   private Node<E> link;

   public Node(E data, Node<E> link)
   {
      this.data = data;
      this.link = link;
   }

   public E       getData( )            { return data; }
   public Node<E> getLink( )            { return link; }
   public void    setData(E       data) { this.data = data; }
   public void    setLink(Node<E> link) { this.link = link; }
}//Node
```