# Lab 5:

*Due April 2 at midnight*

*(65 pts)*

You may work with a partner if you want. Turn in one version with 2 names on it. Do not forget your partner's name. Or work alone. Your choice.

**PROBLEM 1 (Interfaces): (10 pts)**

The methods belonging to an ArrayList are as follows:

- add(o) appends the new object o to the end of the list
- add(i, o) adds the object o at the index i (an int)
- addAll( c) adds everythin from collection c to the ArrayList (at the end)
- clear() removes all elements from the list
- contains(o) returns true if the list contains the object o
- get(i) returns the element located at index i in the list
- indexOf(o) returns the index of the object o in the list
- isEmpty() returns true if the list contains no elements
- lastIndexOf(o) returns the index of the last matching element in the list
- remove(index) removes the element at index and returns it from the list
- remove(object) removes the object from the list and returns true if successful, false otherwise
- removeAll(c) remove from this list all the elements in the collection list
- removeRange(fromindex, toindex) removes everything in the list from the fromindex to the toindex (excluding the toindex)
- retainAll(c) keeps in the list only objects in the collection
- set(index, o) sets the element at the index to the object o
- size() returns the number of elements in the list
- subList(fromindex, toindex) returns a list from the from index to the toindex.

The methods belonging to a LinkedList are as follows:

- add(o) appends the new object o to the end of the list
- add(i, o) adds the object o at the index i (an int)
- addAll(c) adds all of the elements in c to the end of the linked list
- clear() removes all elements from the list
- contains(o) returns true if the list contains the object o
- get(i) returns the element located at index i in the list
- indexOf(o) returns the index of the object o in the list
- isEmpty() returns true if the list contains no elements
- lastIndexOf(o) returns the index of the last matching element in the list
- remove(index) removes the element at index and returns it from the list
- remove(object) removes the object from the list and returns true if successful, false otherwise
- removeAll(c) remove from this list all the elements in the collection list

- removeRange(fromindex, toindex) removes everything in the list from the fromindex to the toindex (excluding the toindex)
- retainAll(c) keeps in the list only objects in the collection
- set(index, o) sets the element at the index to the object o
- size() returns the number of elements in the list
- subList(fromindex, toindex) returns a list from the from index to the toindex.

Both inherit from the same interface (the List interface).

Based on the above methods, write the List interface. NOTE: NAME THE INTERFACE SOMETHING OTHER THAN LIST!!!) Note 2: in Eclipse, File->new about 5 down is interface.

A couple of things to note:

1. When you see o as the input parameter, that indicates an object. The generic object class is Object. So for instance, a function with an Object as an input parameter type would look like this:

   ```
   public void func(Object o) {

           …

   }
   ```

2. When you see c as the input **parameter**, that indicates a collection of some sort. For that type, you'd have Collection c. You may need to import Collection from `import java.util.Collection;`
3. If there's an index or something of type i, that's an int (the primitive type!)

**Problem 2: (20 pts)**

**This problem involves an abstract class, 2 derived classes, and one class used in the derived class (composition):**

**Student Class:**

**Start with the simplest – this class doesn't inherit and doesn't use composition. It is a Student class, with 6 fields. The first is the first name, the second is the last name (both strings), the third, fourth, and fifth are lab, project, and exam grades (ints) in that order. The sixth field is an int for total grade – the sum of the lab, project, and exam grade (just a sum).**

**The constructor takes 5 parameters (2 strings, 3 ints) and sets the first 5 fields, and then either calls a function or just sums the last three fields to get the total.**

**There is also a toString method that creates a string out of the last name, the first name, and the total.**

**ReadAndWrite Abstract Class:**

**This class has 2 fields: a string representing the name of the file being read from, a string representing the file being written to.**

**The constructor takes two Strings (representing the name of the file being read from, and the file being created) as input parameters and sets the first field to the first string, and the second field to the second string.**

**It has 2 abstract methods: a ReadFile method that takes as an input parameter a Scanner object and returns nothing; and a WriteString method that takes nothing as an input parameter and returns a string.**

**Now you must write the interesting methods:**

**OpenForReading: this method takes no input parameters and returns nothing. It creates a file object out of the first field, then creates a Scanner object out of the file object. It then reads in the file USING THE ABSTRACT METHOD ReadFile(). It then closes the scanner object (note: you'll have to catch the FileNotFoundException).**

**OpenForWriting(): this method also takes no input parameters and returns nothing. It creates a new File object out of the second field. It also creates a string. It then uses f.createNewFile() to create a new file for writing, and a PrintStream object. It then gets the string that will be written to the new file USING THE ABSTRACT WriteString() METHOD. That string is printed to the PrintStream object and the PrintStream object is closed. Note: You will have to try and catch the IOException error for both the createNewFile and the PrintStream.**

**Note: createNewFile() creates a new file for writing. But if you don't want to have to keep deleting the file you created as you test your code, you can check to see if the file exists, and if it doesn't, create a new file, e.g.,**

```
If (!f.exists()) {
        f.createNewFile();
}
```

**Derived Class 1: ReadStudentClass (also composed class – has as a field an ArrayList of Student objects):**

**This class must be derived from the abstract class ReadandWrite.**

**The first field for this class is an ArrayList of Student objects.**

**The constructor calls the constructor for its parent abstract class with the name "ClassData.txt" as the input file, and whatever you want as a name for the output file.**

**It then implements the ReadFile method by using the scanner object to read in the data in the ClassData.txt file, making a student object out of each row, and placing it in the Student ArrayList.**

**It has a SortStudent method that sorts the Array of Student objects (the class field) based on their total grade.**

**It must also implement the WriteFile() method, which should first sort the Student Array field, and then should create a big long string of each student (using the student's toString method), formatted appropriately (meaning, there should be some formatting included so you don't**

**end up with one big long string all running together, but I'm not terribly picky about how you go about formatting this string for printing). It should then return this string.**

**Derived Class 2: ReadDataClass:**

**This class must be derived from the abstract class ReadandWrite.**

**It has a field that is an ArrayList of Doubles (the object kind of doubles).**

**It does everything the ReadStudent class did (i.e., implements the ReadFile method, Implements the WriteFile method, sorts the data), but it does it all for a file containing doubles.**

**Create objects of both the ReadDataClass and the ReadStudentClass, and test to make sure it is reading in, sorting, and writing out data appropriately.**

## Problem 3: Creating a web page (20 pts)

Download JackBeanstalk.txt from my web site. You are going to read in this text file and convert it to a web page. To do this, you must both read in the text file and write to an html file (You must create two new File objects, one for the file you're reading from, and one for the file you're writing to).

To the html file, you must add at the top the following html tags:

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My Web Page</title>
  </head>
  <body>
```

Then for each line of the file you read in, you want to convert it to a paragraph by adding the … and  tags around the line. So, for instance, if the line you read in is:

ONCE upon a time there was a poor widow who lived in a little cottage with her only son Jack.

You want to write it to the html file as:

 ONCE upon a time there was a poor widow who lived in a little cottage with her only son Jack.

There are, however, certain lines in the file that should not be paragraphs, but instead should be headers.

The first line, for instance, should have header 1 tags ( <h1> </h1> ) surrounding it.

Also, when an entire line consists of all capital letters, it should have header 2 tags ( <h2> </h2> ) surrounding it., e.g.,

<h2> WONDERFUL GROWTH OF THE BEANSTALK </h2>

When you're done reading in the beanstalk text file and writing it out as an html file, you'll need to add two tags to the bottom of the html file to finish it. Those two tags are:

</body>

</html>

Finally, you'll want to launch your newly created web page in the browser. So you'll have to close the printstream. Once it's closed, you then want to launch the file object (not the printstream object) in the browser using :

```
try {
      Desktop.getDesktop().browse(f2.toURI());
}
catch(IOException e) {
      System.out.println(e.getMessage());
}
```

Where f2 is the File object used to create the html file. You'll have to import at the top of your class definition:

import java.awt.Desktop;

When you run your program, you should have the web page automatically launched in a browser. You've just converted a text file to a web page! (Note: I didn't use any arrays reading in or writing the web page).

## Problem 4: TreeSets Modified: (15 pts)

***Note: this should not involve a lot of coding…***

TreeSets are type of se that infer from the collection interface. As you know, a TreeSet is ordered in the sense that the nodes are placed in the tree in order of least to greatest (as opposed to ArrayList ordering, which is based on indices in the array). TreeSets allow us to add, remove, and check for inclusion (contains) relatively quickly.

TreeSets work by incorporating nodes into a tree-like structure. The node class has 3 fields: the data field (this can be any type of object or simple type), the left child, which is of type node, and the right child, which is also of type node. In each tree, there is a head node. New nodes are inserted into the tree as follows: the new node is compared to the head node. If the new node's data is less than the head node's data, the new node is compared to the head node's left child. If the new node's data is greater than the head node's data, the new node is then compared to the head node's right child. The new node is then compared to the child node's left node or right node, depending on whether the new node is greater than or less than the child node. This continues until a child node is null. The new node is inserted there in the tree. (We discussed this in class).

I have included code for a Node class and a simple TreeSet class, with add, find, and print already defined for you. Try running the code as it stands now and see how it works.

Your job is to change both classes so that we now allow for duplicates. We will allow for duplicates by adding a clone field to the node class, which will link to a Node with duplicate data as the original Node. You will then need to modify the addNode method so that if the node is already in the tree, we add the new node as a clone child of the node in the tree (Note: there can be more than one duplicate). You will also need to modify the findNode method so that if a node with matching data is found in the Tree, you will print out all of the duplicate nodes.

So given this main:

```
public static void main(String[] args) {
	Tree theTree = new Tree();
	theTree.addNode(50);
	theTree.addNode(30);
	theTree.addNode(25);
	theTree.addNode(15);
	theTree.addNode(30);
	theTree.addNode(75);
	theTree.addNode(85);
	theTree.addNode(30);
	theTree.addNode(50);
	theTree.addNode(25);
	theTree.addNode(25);
	theTree.PrintTree(theTree.root);
	// Find the Node2 with key 75
	System.out.println("\nNode with the key 75");
	System.out.println(theTree.findNode(75));
	System.out.println("\nNode with the key 30");
	System.out.println(theTree.findNode(30));
	System.out.println("\nNode with the key 15");
	System.out.println(theTree.findNode(15));
	System.out.println("\nNode with the key 12");
	System.out.println(theTree.findNode(12));
	System.out.println("\nNode with the key 82");
	System.out.println(theTree.findNode(82));

}
```

***You should get this output from the original code I've included below:***

```
30 is already in the tree.
30 is already in the tree.
50 is already in the tree.
25 is already in the tree.
25 is already in the tree.

15  25  30  50  75  85

Node with the key 75
Found! 75
75

Node with the key 30
Found! 30
30
```

```
Node with the key 15
Found! 15
15

Node with the key 12
12 not in tree
null

Node with the key 82
82 not in tree
Null
```

***And you should get this from your modified code:***

```
30 is already in the Tree2.
30 is already in the Tree2.
50 is already in the Tree2.
25 is already in the Tree2.
25 is already in the Tree2.

15  25  25  25  30  30  30  50  50  75  85

Node2 with the key 75
Found! 75
75

Node2 with the key 30
Found! 30
Found! 30
Found! 30
30

Node2 with the key 15
Found! 15
15

Node2 with the key 12
12 not in Tree2
null

Node2 with the key 82
82 not in Tree2
Null
```

**Note:** In order to include duplicates, the total number of lines of code you need to add to the original code is 10 lines (and you could probably get away with less).

**Note2:** I am including the PrintTree2 method for your new code. It requires that you name the new field in your new node class clone. Use the PrintTree2 method in place of the PrintTree method after you have modified the TreeSet code to include duplicates.

Here is the code:

```
public class Tree {
```

```
	Node root;

	public void addNode(int key) {
		// Create a new Node and initialize it
		Node newNode = new Node(key);
		// If there is no root this becomes root
		if (root == null) {
			root = newNode;
		} else {
			// Set root as the Node we will start with as we traverse
the tree
			Node focusNode = root;
			// Future parent for our new Node
			Node parent;
			while (true) {
				// root is the top parent so we start there
				parent = focusNode;
				// Check if the new node should go on the left side
of the parent node
				if (key < focusNode.key) {
					// Switch focus to the left child
					focusNode = focusNode.leftChild;
					// If the left child has no children
					if (focusNode == null) {
						// then place the new node on the left of
it
						parent.leftChild = newNode;
						return; // All Done
					}
				}
				else if (key > focusNode.key){
				// If we get here put the node on the right
					focusNode = focusNode.rightChild;
					// If the right child has no children
					if (focusNode == null) {
					// then place the new node on the right of it
						parent.rightChild = newNode;
						return; // All Done
					}
				}
				else {  //already in tree
					System.out.println(key + " is already in the
tree.");
					return;
				}
			}
		}
	}

	//All nodes are visited in ascending order. Recursion is used to go to
	// one node and then go to its child nodes and so forth
	public void PrintTree(Node focusNode) {
		if (focusNode != null) {
			// Traverse the left node
			PrintTree(focusNode.leftChild);
			// Visit the currently focused on node
			System.out.print(focusNode + " ");
```

```
                // Traverse the right node
                PrintTree(focusNode.rightChild);

            }
        }
        public Node findNode(int key) {
            // Start at the top of the tree
            Node focusNode = root;
            // While we haven't found the Node keep looking
            while (focusNode.key != key) {
                // If we should search to the left
                if (key < focusNode.key) {
                    // Shift the focus Node to the left child
                    focusNode = focusNode.leftChild;
                } else {
                    // Shift the focus Node to the right child
                    focusNode = focusNode.rightChild;
                }
                // The node wasn't found
                if (focusNode == null) {
                    System.out.println(key + " not in tree");
                    return null;
                }
            }
            System.out.println("Found! "+ focusNode);
            return focusNode;
        }
        public static void main(String[] args) {
            Tree theTree = new Tree();
            theTree.addNode(50);
            theTree.addNode(30);
            theTree.addNode(25);
            theTree.addNode(15);
            theTree.addNode(30);
            theTree.addNode(75);
            theTree.addNode(85);
            theTree.addNode(30);
            theTree.addNode(50);
            theTree.addNode(25);
            theTree.addNode(25);
            theTree.PrintTree(theTree.root);
            // Find the Node2 with key 75
            System.out.println("\nNode with the key 75");
            System.out.println(theTree.findNode(75));
            System.out.println("\nNode with the key 30");
            System.out.println(theTree.findNode(30));
            System.out.println("\nNode with the key 15");
            System.out.println(theTree.findNode(15));
            System.out.println("\nNode with the key 12");
            System.out.println(theTree.findNode(12));
            System.out.println("\nNode with the key 82");
            System.out.println(theTree.findNode(82));
        }
}
```

Here's the node class:

```
public class Node {
        int key;
```

```
	Node leftChild;
	Node rightChild;

	public Node(int key) {
		this.key = key;
	}

	public String toString() {
		String str = "";
		str += key + " ";
		return str;
	}
}
```

Here is the new PrintTree2 method:

```
	public void PrintTree2(Node focusNode) {
		if (focusNode != null) {
			// Traverse the left Node
			PrintTree2(focusNode.leftChild);
			// Visit the currently focused on Node2
			System.out.print(focusNode + " ");
			Node tempnode = focusNode;
			while (tempnode.clone != null) {
				System.out.print(tempnode.clone + " ");
				tempnode = tempnode.clone;
			}
			// Traverse the right Node2
			PrintTree2(focusNode.rightChild);
		}
	}
```

***When Finished, study for Celebration of Knowledge II***
***Start Final Project***