csci 211 Assignment 1

Linear Data Structures
Due: Thursday, September 30 2010[1]

This assignment provides experience with linear data structures, generics, inheritance, packages, and testing.

Overview
This assignment asks you to implement several important linear data structures, and also to use generics and inheritance. In addition, you are asked to verify your implementations by writing test cases in a separate main() method for each class.

What to do
First, create a fresh project called a1, and add a new file called a1.java. Next, create a file called `List.java`, and specify its package as "edu.olemiss.cs211.a1". Use this same package for other kinds of list, but `a1.java` should be in the default package.

Several files are provided on blackboard to get you started. You could either add them directly to your project, or simply create your own files and copy and paste the contents of the provided versions.

The List Class

The List class maintains a list of elements in no particular order. It does not check for duplicate elements. This class is declared as

```
public class List<T> implements Iterable<T> {
```

An existing List.java file available on Blackboard supplies the iterator() method, which creates an iterator for the list, and satisfies the requirements for the Iterable interface. Paste the contents of this file into your own List.java. You'll want to use iterators in this assignment. Here are some examples:

```
for(T n: this){ // "this" must be a List instance

    foo.append(n); // append element n to foo
}
```

[1] Assignments are not considered late until 8AM the following morning. After that, a late penalty of 5% per day will be applied up to five days, after which the assignment cannot be accepted.

```
for(City neighbor: neighbors) // same for neighbors
     System.out.println(neighbor)
```

Inside the provided List class, there is also an inner class called ListNode, which uses Generics. You can make a new node with something like:

```
head = new ListNode<T>(o, null);
```

Here's the methods you need to implement:

```
public List() // construct an empty list

public void add(T o) // add the object to the list
public int size()// return the number of elements
public String toString() // return a string representing the list
```

The Stack Class

You'll have to write your own Stack.java from scratch. It should be a child of List, and should provide the following methods, with the obvious meanings:

```
public Stack()
public void push(T e)
public T pop()
boolean isEmpty()
```

If the user attempts to pop() from an empty stack, you should do the following:

```
throw new java.util.NoSuchElementException();
```

The OrderedList Class

This class maintains an ordered list of *unique* elements. If `add()` is called with an argument `a` that is `equal()` to an existing element, `a` is not added. Remember not to use == to test for equality.

```
public class OrderedList<T extends Comparable<? super T>> extends List<T> {
```

This confusing declaration says that OrderedList is a child class of List, and also uses Generics to insist that the OrderedList contain only elements of type T, where T is determined at compile time. The type T must implement the *Comparable* interface, meaning that a `compareTo()` method is provided which determines an ordering of all T objects.

That is, if you say:

```
OrderedList  myList = new OrderedList<City>();
```

then any attempt to insert objects that are not Cities will be caught by the compiler. Also, the compiler will ensure that the City class implements the *Comparable* interface.

For this class you need only implement the following method:

```
public void add(T o){
```

This method inserts an object of type T into the list, but maintains the order of the list by inserting it into the proper position, as determined by the compareTo() method of class T.

Be sure to handle the case where an element is inserted at the head of the list. This is somewhat different from inserting in the middle.

Testing

For all three classes, you should test your code carefully by adding a `main()` method directly to each class. This main method should create one or more instances of the class and test the behavior of each method. As described in the book, take care to identify the boundary cases, and explicitly test them. The a1.java class will also exercise your code, but perhaps not exhaustively. To run each of your main() methods individually, you can right click (control click for 1 button people) on the text window of your class and select the "Run As..." option. Select "Java Application", and your test code should run.

Submission

It is important to put your name on all of your files. It also important to make a single zip archive (not 7zip, or any other weird formats) containing your project (You can do this from inside Eclipse). Next, submit your archive file using the blackboard dropbox mechanism. Be sure to complete the blackboard submission process entirely.