Recursive Linked List

Overview

For this assignment, you will be implementing yet another Linked List data structure. However, this time all of the methods must be recursive.
Due: February 28th 2014 @ 5PM

Files to complete

You are expected write an implementation for each of the interfaces listed in **config.Configuration**. As with the last assignment, you must specify which implementation you would like us to grade in this file.

In addition to this, you **MUST** include a String field in the **config.Configuration** file with your UMass student ID number. This is the 8 digit value found on your student ID card and is used to identify you in moodle. This will allow us to get your grade and feedback to you quickly.

It will almost certainly be useful for you to write additional class files. Any class file you write that is used by your solution **MUST** be in the provided **src** folder. When we test your assignment, all files not included in the **src** folder will be ignored.

Note: You may not use any of the provided Collection classes provided by the java standard API. A list of them can be seen in the "All Known Subinterfaces" and "All Known Implementing Classes" of the following URL: http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html

Test files

In the test folder, you are provided with several JUnit test cases that will help you keep on track while completing this assignment. We recommend you run the tests often and use them as a checklist of things to do next. You are not allowed to modify these files. If you have errors in these files, it means the structure of the files found in the **src** folder have been altered in a way that will cause your submission to lose points.

Support Code API

The Support Code's comments have been generated into a nicely formatted API that can be found here: http://people.cs.umass.edu/~jcollard/cs187/s14/assignments/5/doc/

It is highly recommended that you spend a day simply reading over the comments in each of the interfaces and classes provided:

structure.ListInterface - An interface for a List

Part One: Importing Project into Eclipse

Begin by downloading the provided project from Moodle and importing it into your workspace.

You should now have a project called **recursive-list-student** it is very important that you do not rename this project as it is used during the grading process. If the project is renamed, your assignment may not be graded.

By default, your project should have no errors and contain the following root items:
src - The source folder where all code you are submitting must go. You can change anything you want in this folder, you can add new files, etc...
support - This folder contains support code that we encourage you to use (and must be used to pass certain tests). You are not allowed to change or add anything in this folder
test - The test folder where all of the public unit tests are available
JUnit 4 - A library that is used to run the test programs
JRE System Library - This is what allows java to run

If you are missing any of the above or errors are present in the project, seek help immediately so you can get started on the project right away.

Part Two: Implement the ListInterface

Start off by implementing the ListInterface provided. Unlike previous lists you've created, in this one you may not use loops of any kind. Be sure to pay close attention to the big-O running times. You will not pass all of the tests if you do not adhere to these running times.

Hints: Recursion is all about reducing some problem slightly to make it easier to solve. Ask yourself, "What method would make this easier?" For example, the contains method signature is written in a way that it is not going to be possible to do recursion on a Node. However, there is nothing stopping us from writing a private method that we can use internally. Wouldn't it be great if we had a method that had this signature:

```
private int contains(T toFind, Node<T> toCheck, int currentIndex);
```

We can then simply ask, does toCheck hold toFind? If it does, we can return currentIndex. Otherwise, we recurse on the next node in the list, at the next index, with the same toFind value. What should we do if toCheck is null?

Try and come up with good "helper" methods that lend themselves to recursion. You can't modify the interface, but that doesn't stop you from adding new methods.

Part Three: Export Project and Submit

When you have finished your solution and are ready to submit, export the entire project. Be sure that the project is named **recursive-list-student**. Save the exported file with the zip extension (any name is fine).

Log into Moodle and submit the exported zip file.