CMSC 132 Homework #2 Fall 2012

Date due: Wednesday, October 24, 10:00:00 p.m.

1 Introduction and purpose

For this homework you will write four recursive static methods that will perform simple tasks on generic lists. The homework is designed to help you practice implementing recursive algorithms, in particular ones that operate upon data structures. The solutions to these problems should be quite short.

Check out the homework starting code in the project `hw2` as for the earlier assignments. The project contains a package `recursiveListMethods`, as well as a package `tests`. Note, however, that to encourage you to write your own tests of your methods (actually to require that you do so) there will not be any public tests for this homework, only secret tests. The `tests` package is for your convenience in writing your own tests, which you will need to do to test the correctness of your methods. The posted project grading policy explains how homeworks are graded vs. projects, namely that homeworks are not graded for style. However, if you don't follow the requirements in Section you will lose most or all of the credit for this homework. (If you have any questions about whether you're doing things right, ask in office hours before the homework due date.)

2 Homework description

The methods all use Java's `List` interface, part of Java's Collections framework, which is a generic list. The `ArrayList` class implements the `List` interface, as does Java's `LinkedList` class, which as its name implies is a linked list. Since all the methods will have parameters that are of type `List<E>`, we could call them passing in an `ArrayList` (instantated with some type of objects, a `LinkedList`, or even other classes that implement the `List` interface. A `List` can obviouly contain zero or more elements.

In writing the methods you may find it helpful to be able to generate a sublist from a `List`. The `List` interface includes a method with the signature:

```
List<E> subList(int fromIndex, int toIndex)
```

Note that the parameter `toIndex` is **not** inclusive, although `fromIndex` is. For example, suppose a list `x` contains the elements (`Character`s in this case) `a`, `b`, `c`, `d`, and `e`, then `x.subList(2, 4)` would return a list containing just `c` and `d`, not a list with `c`, `d`, and `e` as you might have expected. (This is analogous to the `substring()` method of the `String` class.) Check out the other methods of the `List` interface in the Java API, since they may be useful.

Note lastly that `<T>` appears between the word `static` and the return type of the first three methods to be written (with `<T extends Comparable<T>>` in the fourth one). This indicates that they are generic methods. Notice that the class `RecursiveListMethods` that the methods are in is not a generic class, so (although this wasn't mentioned before in class) Java allows having generic methods in a non–generic class. These type parameters will be instantiated with a specific type when the methods are invoked.

3 Description of methods

All methods that have any objects as parameters should throw a `NullPointerException` if any object or array parameter is `null`. It doesn't matter what string or message any exceptions are constructed with, or even if they have a message string at all. (Note that you may not have to do anything special to get this behavior.) The first two methods should simply do nothing if their list parameter is empty, since there is no first or last element to be duplicated.

3.1 `public static <T> void duplicateFirst(List<T> list)`

This method will replace all of the elements in its parameter `list` with copies of the list's **first** element. For example, if the original list contains the elements (`Character`s in this example) `a`, `b`, `c`, `d`, and `e`, then after the method is over the list will contain `a`, `a`, `a`, `a`, and `a`.

3.2 `public static <T> void duplicateLast(List<T> list)`

This method will replace all of the elements in its parameter `list` with copies of the list's **last** element. For example, if the original list contains the elements (`Character`s in this example) `a`, `b`, `c`, `d`, and `e`, then after the method is over the list will contain `e`, `e`, `e`, `e`, and `e`.

3.3 `public static <T> boolean anyDuplicates(List<T> list)`

This method will return false if all of the elements of its parameter `list` are distinct, or if the list is empty, and true if there are any duplicates. The `equals()` method should be used in comparing elements of lists (although it may be called indirectly, depending upon how you write the method).

3.4 `public static <T extends Comparable<T>> T findLargest(List<T> list) throws NoSuchElementException`

This method will return the largest element of the list. Note that the elements in the list must be `Comparable` to one another, and this indicates how they may be compared. If the list is empty the method should throw a `NoSuchElementException`.

4 Homework requirements

1. This homework will be graded entirely on the results of secret tests. You should write student tests in order to be assured that your methods work right, but they will not be graded. However, although your programming style will not be graded, your source code will be inspected to ensure that it satisfies the requirements below. Violations of the following two requirements will result in loss of **significant** credit on the homework, up to receiving no credit at all for it.
2. You **may not** use any loops anywhere in your code.
3. You **may not** add any static variables or fields (instance variables) to the `RecursiveListMethods` class, or any other classes or packages to the homework.
4. The project grading policy on ELMS describes the late policy for projects and homeworks. **Do not wait** until the last minute to finish and submit your homework. It is strongly suggested that you finish and submit the homework **at least** one day before the deadline, to allow time to reread the assignment and ensure you have not missed anything that could cause you to lose credit. If you have, this will give you time to correct it. This will also give you time to get any last–minute unexpected problems fixed.
5. After you submit you must **log into the submit server** and verify that your submission compiled and worked correctly (at least on your student tests) there!

5 Academic integrity

Please **carefully read** the academic honesty section of the course syllabus. **Any evidence** of impermissible cooperation on projects, use of disallowed materials or resources, or unauthorized use of computer accounts, **will be submitted** to the Student Honor Council, which could result in an XF for the course, or suspension or expulsion from the University. Be sure you understand what you are and what you are not permitted to do in regards to academic integrity when it comes to project assignments. These policies apply to all students, and the Student Honor Council does not consider lack of knowledge of the policies to be a defense for violating them. Full information is found in the course syllabus– please review it at this time.

The academic integrity requirements also apply to any student tests for programming assignments, which must be **your own original work**. Copying the public tests and turning them in as your student tests would be plagiarism, and sharing student tests in any way is prohibited.