CMPSCI 187 / Fall 2018
Recursive Linked List

Mark Corner and Joe Chiu

Contents

Overview **3**

Learning Goals 3

General Information [Common] 3

Problem 1 **3**

Import Project into Eclipse [Common] 3

Implement the ListInterface 4

Export and Submit [Common] 5

Overview

In this project, you will write a linked-based implementation for the List abstract data type, as specified in `structures.ListInterface`. The requirement is all methods you implement must use *recursion* and you are NOT allowed to use any loop (such as **`for`**, **`while`**, **`do-while`**). Finally, you'll need to write your own tests for the code; we will not be providing a comprehensive test suite.

Learning Goals

- Show ability to write a generic class that implements a given interface, fulfilling the contract it specifies (including $O()$ behavior and an `Iterable` implementation).
- Continue to practice recursive methods and List ADT.
- Continue to practice writing appropriate unit tests.

General Information [Common]

Reminder: Copying partial or whole solutions, obtained from other students or elsewhere, is academic dishonesty. Do not share your code with your classmates, and do not use your classmates' code. If you are confused about what constitutes academic dishonesty you should re-read the course policies. We assume you have read the course policies in detail and by submitting this project you have provided your virtual signature in agreement with these policies.

- For some projects, it may be useful for you to write additional java files. Any java file you write MUST be placed in the provided `src` directory of your Eclipse project.
- When submitting your project, **be sure to remove all compilation errors**. Any compilation errors will cause the autograder to fail to run, and consequently you will receive a zero for your submission. **No Exceptions!**

In the `test` directory, we provide several JUnit tests that we refer to as the **public tests**. We recommend you run the tests often and use them as starting points to test your project. In addition, some of the java files come with their own `main` functions. You can run them as Java applications to interact with the project.

Be aware that the autograder will use a more comprehensive set of tests (referred to as **private tests**) to grade your project. These tests are hidden from you. We recommend that you think about possible test cases and add new `@Test` cases to your public test files as part of your programming discipline. In particular, you should consider:

- Do your methods handle edge cases such as integer arguments that may be positive, negative, or zero. Many methods only accept arguments that are in a particular range.
- Does your code handle cases such as when an array or list is empty, or has reached full capacity?

More complex tests will be project-specific. To build good test cases, think about ways to exercise methods. Work out the correct result for a call of a method with a given set of parameters by hand, then add it as a test case.

Problem 1

Import Project into Eclipse [Common]

Begin by downloading the starter project and importing it into your workspace. It is very important that you **do not rename** this project as its name is used during the autograding process. If the project is renamed, your assignment will not be graded, and you will receive a zero.

The imported project may have some compilation errors, but these should not prevent you from getting started. Specifically, we may provide JUnit tests for classes that do not yet exist in your code. You can still run the other JUnit tests. After completing your code, the compilation errors should be gone.

Problem 1 continued on next page...

The project should normally contain the following root items:

`src` This is the source folder where all code you are submitting must go. You can change anything you want in this folder (unless otherwise specified), 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 **must not** change or add anything in this folder. To help ensure that, we suggest that you set the support folder to be read-only. You can do this by right-clicking on it in the package explorer, choosing Properties from the menu, choosing Resource from the list on the left of the pop-up Properties window, unchecking the Permissions check-box for Owner-Write, and clicking the OK button. A dialog box will show with the title "Confirm recursive changes", and you should click on the "Yes" button.

`test` The test folder where all of the public unit tests are available. You can feel free to add your own tests here.

`JUnit 4` A library that is used to run the test programs.

`JRE System Library` This is what allows Java to run; it is the location of the Java System Libraries.

If you are missing any of the above or if errors are present in the project (other than as specifically described below), seek help immediately so you can get started on the project right away.

Implement the ListInterface

In the `src/structures` directory, create a new class called `RecursiveList` which implements the `ListInterface` described in the `structures` package. Again, please make sure to place your Java file under the `src/` directory, and NOT in the `support/` or `test/` directories.

In your implementation, you may **NOT** use any loop (i.e. **`for`**, **`while`**, or **`do while`** loops). Also, you may **NOT** use any Java class that implements the `Collection` interface (e.g. ArrayList). **If you submit an implementation using explicit iteration, you will receive a zero for this project.**

Be sure that your implementation complies with the required big-O runtime bounds in the comments of each method. You will not pass all of the autograder tests if your methods are not within these bounds.

Hints: Recursion is all about reducing some problem slightly to make it easier to solve. For some methods, to implement them using recursion, you will find it necessary to create private, "helper" methods that are recursive, and then have your public methods call these helper methods to jump start the recursive call. You have seen such an example in the `revPrint` method (i.e. printing a linked list backwards). You have to decide what arguments these helper methods need in order to implement them recursively.

Let's look at a specific example for this project: take a look at the `indexOf` method – this public method is written in such a way that does not lend itself to recursion on a linked list node. So you should create a private helper method to use internally. Perhaps something like:

```
private int indexOf(T elem, int currIndex, LLNode<T> currNode)
```

We can then simply ask, does the current node contain the element you are looking for? If it does, we can return the current index. Otherwise, we recurse on the next node in the list, at the next index, with the same element to search for. Think about what the base cases are.

You should create a `LLNode<T>` class yourself. You can write it directly inside `RecursiveList.java` as long as you don't make it public (i.e. one Java file may contain only one public class, but multiple non-public classes). You also need to create an iterator class for your list. As before, you do NOT need to implement the iterator's `remove` method, so just have it throw an `UnsupportedOperationException`.

Problem 1 continued on next page...

We have provided you with an absolutely minimal set of tests. You will need to come up with your own test cases to help you debug. Try to think of all of the edge cases that could occur, and write tests to check for each of them. You will not be graded on your tests, but writing them and passing them is the only way that you can be reasonably sure that your code works and passes private tests.

Export and Submit [Common]

When you have completed this project, you should export an archive file containing the entire Java project. To do this, click on the `recursive-list-student` project in the package explorer. Then choose "File → Export" from the menu. In the window that appears, under "General" choose "Archive File". Then choose "Next" and enter a destination for the output file. Be sure that the project is named **`recursive-list-student`** (otherwise you may be exporting the wrong project!). Save the exported file with the `zip` extension.

Once you have the zip file ready, you can submit it to the online autograder (Gradescope). Please see the course website and/or documentation explaining how to use the online autograder. If you have any questions please be prompt in asking the course staff before it is too late to submit your solution. If you encounter any errors that look like the autograder failed and not your project, please let the instructors know.