CSC115 Assignment 2: Collections

Objectives

Upon completion of this assignment, you need to be able to:

- Practice creating a class that *implements* an interface in Java.
- Use the `Node` class to create a reference-based list.
- Use Exception Handling techniques.
- Read and understand specifications.
- Implement testing of code during development.

Introduction

A local pharmacy would like to keep a small database of its clients and a list of the medications that each patient is currently taking. For now, it is confined to local residents, but the pharmacy is hoping to tap into the larger region and include communication with the hospitals. We are tasked with supporting a basic system that will monitor medication lists to later attach to patient files. Since most people are not taking large numbers or medications, we decide to use the basic List ADT to manage each patient's medication information. We haven't decided whether to use an array-based list or a reference-based list, so we will implement both. That way we can plug in whichever one suits the client's needs, without having to alter the client's handling of the lists. The array-based medication list has already been completed and tested. We need to complete the reference-based list.

In this assignment you will:

- Develop the `MedListRefBased` class that implements the standard `List` interface approved by the client.
- Use the doubly-linked list as the primary data field in this class; it is described in the textbook's Chapter 5, beginning on page 280.

Quick Start

(1) Create a fresh directory to contain this assignment: `CSC115/assn2` is a recommended name. Download this pdf file and the `.java` files to this directory.

(2) Download and extract the `docs.zip` and `javafiles.zip` in the same directory. A `docs` directory will be created to store the specifications for the `public` classes. The following links will work if the documents are stored in the `docs` directory local to this pdf file.

- List specifications
- MedListArrayBased specifications
- Medication specifications

(3) Examine the List specification document. It is similar to the ADT description in the textbook, page 204. The `List.java` file contains all the method headers required of any class that *implements* a List. Note that each method does not contain a body; the implementation is never part of the ADT. Note also that the description uses Java generics, described in the textbook, page 291. This allows the *developer* of a `List` to decide the particular data type of the elements stored in the list.

(4) Examine the completed source code for the `MedListArrayBased` class. Note that it `implements List<Medication>`, and thereby *is-a* `List`. Note also that `MedListArrayBased` implements every method of the `List` interface. The `main` method is used as the test harness for `MedListArrayBased` and tests every method, including the private methods. Use this test harness as a model for the tests that you will create in `MedListRefBased`.

(5) Examine the completed source code for `MedListListTester`. This is an example of an *external* tester. It is the tester that is run once the completed class is ready for its final assessment, before submitting it to the client. Note that it can only test the methods that are listed in the `List` interface. It tests the list for functionality from the perspective of a *user*, the pharmacy, in this case.

(6) Compile all the given Java files. You can run `MedListArrayBased` to examine the internal tester results. You can also run the `MedListTester` to examine the external test results.

(7) Your job is to create a class called `MedListRefBased` that `implements List<Medication>`. It must contain all the required public methods and implement the list using a doubly-linked list, instead of an array. See Chapter 5 for information on the doubly-linked list.

(8) Once the code for `MedListRefBased` is completed and internal testing is done, you can make a very small change to the external tester and perform the external test on your code.

Detailed Instructions

There are several files supplied in this assignment. Examine them many times; their purpose makes sense as you work on the implementation of the linked list version of the medication list. You only need to create one class. Once that is done, you will need to alter the external tester class so that it calls the `MedListRefBased` constructor instead of the `MedListArrayBased` constructor.

You are to create, implement and test the `MedListRefBased` class. Make sure of the following:

(1) The main data structure you use to store the medications is a doubly-linked list, that you create yourself (no classes that extend the `java.util.List` data structure are to be used in your source code).

(2) The class `implements List<Medication>`.

(3) Any private methods are properly commented (follow the `MedListArrayBased` source code as a guide). Note that comments are not required for any method that implements a method required by the `List` interface. Those methods are already commented and the methods you implement will follow those specifications. During the labs, you will be introduced to *javadoc*, a tool to create proper documentation.

(4) The `main` method inside your source code tests each of your methods. Create your own internal tester for `MedListRefBased` to thoroughly check the code you write, including the private helper methods. You do not need to comment out the `main` method and you must not delete it. The marker will be looking for evidence of internal testing.

(5) Once you you are satisfied that all the internal testing works, then you can make a final check, using the external tester that is provided. If the external tester fails, then you must go back to the internal tester and take the necessary steps to debug and fix your code. The external tester will test your program when you alter the one line in `MedListTester.java` that declares the `list` variable by initializing it to a `MedListRefBased` object*. You should see **exactly** the same output as when it tests the `MedListarrayBased` class.

Submission

Submit the following completed file to the Assignment folder on conneX.

- `MedListRefBased.java`

Please make sure you have submitted the required file(s) and conneX has sent you a confirmation email. Do not send [`.class`] (the byte code) files. Also, make sure you *submit* your assignment, not just save a draft. Draft copies are not available to the instructors, so they are not collected with the other submissions. We *can* find your draft submission, but only if we know that it's there.

A note about academic integrity

It is OK to talk about your assignment with your classmates, and you are encouraged to design solutions together, but each student must implement their own solution.

*Note the use of *inheritance* here. The `list` variable is of type `List` but the actual declaration calls the correct constructor of the implementing subclass.

Grading

Marks are allocated for the following:

- Proper programming style is demonstrated, as per the coding conventions on CSC115 conneX Resources.
  - In this assignment, all public methods that implement `List` methods do not need header comments, although internal programmer comments are welcome where needed within the method.
- All methods are implemented exactly to the specifications.
  - Note that the external tester may be used as an indicator of success or failure with respect to the specifications being met. However, external tester success is not the definitive indicator that the program works completely as specified.
- Use of private helper methods where applicable: if similar code is used in more than one method, then a separate private method should be created that is then called by the methods using similar operations. See the array based version of the list as an example.
- Internal testing in the main method that demonstrates clearly that each method was checked.
- Marks will be withheld for code that makes use of any other data type other than a CSC115-student-created-doubly-linked list.

You will receive no marks for any Java files that do not compile.