University of Victoria
Faculty of Engineering: Computer Science

CSC 115: Fundamentals of Programming II

Spring 2016

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.

Resources:

- CSC 115 Java Coding Conventions.pdf (found on conneX, under Resources)
- Textbook Chapters 4 and 5.
- List.java
- MedicationNode.java
- ListIndexOutOfBoundsException.java
- MedListArrayBased.java
- MedListTester.java

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 on huge numbers of mediation, 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) If you haven't done so already, download all the necessary documents for this assignment from conneX into a specific folder for CSC115 assignment two, `assn2` is a recommended name.
2) Double-click on the `List.html` to see the specifications for a standard List interface. It is similar to the ADT description on the Text page 204. The `List.java` file contains all the method headers required of any class that *implements* a list. Note that none of the methods contain a body; the implementation is never part of an ADT. Note also that the description uses Java generics, described on page 291. This allows the *developer* of a List to make the decision of the type of elements E that will be stored in the list.
3) Examine the completed source code for the `MedListArrayBased` class. Note that it `implements List<Medication>`, allowing for any object of the `MedicationArrayBased` to also call itself a `List` object. Note also that `MedListArrayBased` implements every method of the `List` interface. Similarly to the file in Assignment one, there is a `main` method where internal testing of each method is carried out. Internal testing allows a programmer to test all methods, including the private methods during the development phase.
4) Examine the file called `MedListTester.java`. This is an example of an *external* tester. It is the tester that is used 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` ADT. It tests the list for functionality from the perspective of a *user,* the pharmacy in this case.
5) Compile all the given java files. You can run `MedListArrayBased` to examine the internal tester result. You can also run the `MedListTester` to examine the external test results.
6) Your job is to create a class called `MedListRefBased` that `extends 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.
7) Once the code for `MedListRefBased` is completed and internal testing is done, you can use the external tester after changing a portion of a single statement in the tester.

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 `java.util` classes 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. Use this extensively to check the progress, including the private helper methods. You do not need to comment out the `main` method or delete it. The marker will be looking for evidence of internal testing.
5. Once you have thoroughly tested all the methods internally, then you can move onto the external tester. 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 list` variable and initialize it with a call to the `MedListRefBased` constructor. Then re-compile and run `MedListTester`. You should see **exactly** the same output as when it tested the `MedListArrayBased` class.

Submission

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

- `MedListRefBased.java`

Please make sure that conneX has sent you a confirmation email. Do not send [`.class`] (the byte code) file, or any another file for this assignment. Also make sure you *submit* your assignment, not just save a draft. All draft copies are not available to the instructors, so we cannot mark them.

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 (code) their own solution.
We will be using plagiarism detection software on your assignment submissions.

Grading

You will receive no marks for java files that do not compile.
The marker will be looking for:

- Proper programming style as per the code conventions on CSC115 conneX Resources.
  - In this case, 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 in List.html.
  - Note that the external tester may be used as an indicator of success or failure with respect to the specifications being met.
- Evidence of private helper methods where applicable.
- Marks will be withheld for code that makes use of any other data type other than a CSC115-student-created-doubly-linked-list. In other words, there should be no array or any existing API class from outside the standard java.lang package used as a data field.