The University of Saskatchewan

Saskatoon, Canada

Department of Computer Science

CMPT 280– Intermediate Data Structures and Algoirthms

Assignment 1

Date Due: January 23, 2018, 10:00pm

Total Marks: 39

1 Submission Instructions

- Assignments must be submitted using Moodle.
- Programs must be written in Java.
- No late assignments will be accepted. See the course syllabus for the full late assignment policy for this class.

2 Background

For this assignment you'll be working with linked list classes from the data structure library `lib280`. `lib280` is a library of data structures that we will build up over the duration of the course. We will start with a version of `lib280` that has very few data structures in it and add more with each assignment. Each assignment will come with a new version of `lib280` which contains the correct implementations of ADTs that were the subject of the previous assignment.

Obtaining and Setting Up `lib280`

For this assignment the first thing you'll need to do is to obtain a copy of `lib280-asn1`. It is provided along with this assignment description on the class webpage. Download the lib280-asn1.zip file and expand its contents somewhere in your filesystem.

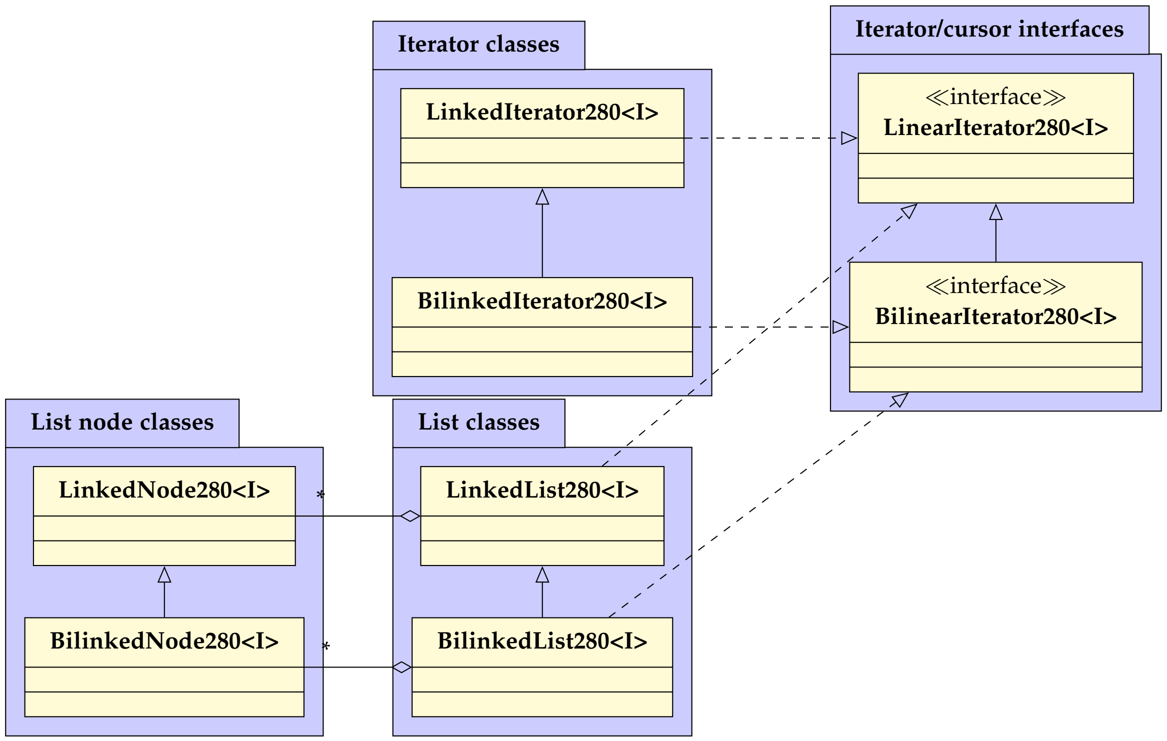
The class website provides a self-guided tutorial that explains how to import lib280 into an IntelliJ project once you have downloaded it; it is located under the "Laboratory/Tutorial Resources" heading. First complete part 1 of the tutorial to create an empty IntelliJ project. Then complete part 2 of the tutorial to import `lib280-asn1` into your project. For question 1 complete part 3 of the tutorial to create a module for your program that can use the classes from `lib280-asn1`. For questions 2 and 3 you don't need to complete part 3 again because you'll just be working within the `lib280-asn1` module.

Navigating `lib280-asn1` Within IntelliJ

The `lib280-asn1` module contains several packages. The classes of interest to use for this assignment are in the `lib280.list` package. Find the `lib280-asn1` module in your "project" tab, normally located on the left side of the IntelliJ window. Expand it by clicking the little triangle beside it. This should reveal a folder called "src". Expand that as well. Now you will see a list of java packages that contain the various classes in the `lib280-asn1` library. For this assignment, the classes we are interested in are in the `lib280.list` package, so click the triangle to expand it. You should now see classes like `LinkedList280` and `BilinkedList280`.

Singly- and Doubly- Linked List Classes in `lib280`

The UML diagram below shows the class hierarchy you'll be working with in this assignment. It may look a bit daunting at first, but you'll soon see it's not that complicated. There are four pairs of classes/interfaces (surrounded by light blue boxes[1]). In each pair, there is one class for a singly-linked list and one for a doubly-linked list. The class/interface of each pair that pertains to doubly linked lists extends the class/interface related to singly linked lists.



`LinkedNode280<I>`: The node class used for a singly-linked list.

`BilinkedNode280<I>`: An extension of `LinkedNode280<I>` that adds the "previous node" reference required for nodes in a doubly linked list.

`LinearIterator280<I>`: An interface that defines the methods that must be supported by cursors and iterators that can step forwards over a linear structure, such as `goFirst()`, `goForth()`, `after()`, etc.

`BilinearIterator280<I>`: An interface that extends `LinearIterator280<I>` by adding methods that allow stepping backwards, such as `goBack()` and `goLast()`.

`LinkedIterator<I>`: An implementation of `LinearIterator280<I>` which is an iterator object for a singly-linked list. It is used by the `LinkedList280<I>` class to provide iterators.

`BiinkedIterator<I>`: An implementation of `BilinearIterator280<I>`, and an extension of `LinkedIterator280<I>`, which is an iterator object for a doubly-linked list. It is used by the `BilinkedList280<I>` class to provide iterators.

`LinkedList280<I>`: A singly-linked list class. It provides a cursor by implementing the LinearIterator280<I> interface. The Nodes of the list are `LinkedNode280<I>` objects, and it can provide iterators of of type `LinkedIterator<I>`.

[1]The light blue boxes in the UML diagram are only to show the pairs of classes that serve the same roles for singly-/doubly-linked lists and do not represent any actual grouping within `lib280`. All of the pictured classes are in the same package within `lib280`.

`BiLinkedList280<I>`: A doubly-linked list class. It provides a cursor that can move both forwards and backwards by implementing the BilinearIterator280<I> interface. The nodes of the list are `BilinkedNode280<I>` objects, and it can provide iterators of of type `BilinkedIterator<I>`.

Take a moment to familiarize yourself with these classes and their methods, particularly the `LinkedList280<I>` and `LinkedIterator280<I>` classes as you will be working on coding extensions of these classes.

Iterators

This section describes a bit more about how iterators work. Iterators provide the same functionality as a container ADT that has a cursor, but they are separate objects from the container. This allows us to record a cursor position that is different and independent from the position recorded by the container's internal cursor.

The list objects, `LinkedList280<I>` and `BilinkedList280<I>` both have methods called `iterator`. The `iterator` method in the `LinkedList280<I>` class returns a new cursor position encapsulated in an instance of the `LinkedIterator280<I>` class. This instance will have references directly to the nodes of the `LinkedList280<I>` instance that created it. In essence, the `LinkedIterator280<I>` contains its own copies of the `position` and `prevPosition` fields that appear in `LinkedList280<I>` – i.e. another cursor that is external to the list! This cursor can be manipulated in exactly the same was as the internal cursor of the list. If you compare the methods in `LinkedIterator280<I>` to the methods of the same name in `LinkedList280<I>`, you'll see that they are almost identical.

Thus, each time we want a new cursor that is independent of the list's internal cursor, we can call the `iterator` method and get a new one. This adds additional flexibility. If we can get away with just using the lists internal cursor for our purposes, then we can do so, but we have the option to create more cursors in the form of iterators should we so desire.

3 Your Tasks

Question 1 (9 points):

Tractor Jack is a notorious pirate captain who sails the Saskatchewan River plundering farms for wheat, barley, and all the other grains. You may remember him from his exploits in CMPT 141.[2]

Jack employs the members of his pirate crew at ten different ranks (numbered 0 through 9). A crew member's rank is generally related to the number of sacks of grain they are able to plunder annually. Although there is some variability, Jack is certain that pirates of higher rank generally plunder more sacks of grain than those he appoints at lower ranks. As such, jack generally pays pirates of higher rank more than than pirates of lower rank (though each pirate's pay is negotiated separately, so every crew member has a different annual salary).

Jack has always negotiated crew member pay based on his gut feeling, which is perhaps a poor way of doing "business". He wants to analyze his payroll to see if the annual salaries of his crew members are equitable in the sense that he'd like his crew members at each rank to be paid about the same **per sack of grain** that they plunder.

You will write a program to determine the total amount Jack paid his crew **per sack of grain** for each crew rank separately. If the amount paid per sack of grain plundered varies a lot over the different ranks, Jack may want to re-think his pay scale and his negotiation tactics.

In the provided file `PerformanceByRank.java` you are given code that reads a data file `piratecrew.txt` (provided) containing the pay data for all of Jack's crew. The data will be returned as a linked list of `Crew` objects of type `LinkedList280<Crew>`. A `Crew` object contains the rank, annual pay (in golden guineas), and the number of sacks of grain plundered in the last year by a crew member. It has getters and setters for each of these three pieces of data. `Crew.java` defining this class is provided.

In the `main()` function of `PerformanceByRank.java`, write a program that does the following:

1. Create an array `piratesByRank` of ten `LinkedList280<Crew>` objects, one for each rank (you saw how to do this in Tutorial 1).
2. Initialize the elements of the array of linked lists `piratesByRank`.
3. Iterate through the list of Crew objects returned by `readCrewData()` and add each `Crew` object to the list at offset `r` of `piratesByRank`, where `r` is the crew member's rank.
4. For each list, determine the amount paid to all of the crew members of that rank **per sack of grain plundered by all of the crew members of that rank** and print it to the console (see the sample output section, below). If a list is empty, instead print to the console a message that Jack has no pirates employed at that rank.

You are not permitted to modify any of the provided code. Exception: you are allowed to modify the pathname to the input file `piratecrew.txt` in the event that you get a "file not found" error.

Sample Output

```
Jack's rank 0 crew members were paid of 2000.0 guineas per sack of grain plundered.
Jack's rank 1 crew members were paid of 258.78087720909235 guineas per sack of grain plundered.
Jack's rank 2 crew members were paid of 252.1541797737082 guineas per sack of grain plundered.
Jack's rank 3 crew members were paid of 274.9021704266409 guineas per sack of grain plundered.
Jack's rank 4 crew members were paid of 317.725137629488 guineas per sack of grain plundered.
Jack's rank 5 crew members were paid of 339.7282378647562 guineas per sack of grain plundered.
Jack's rank 6 crew members were paid of 360.94124515527716 guineas per sack of grain plundered.
Jack's rank 7 crew members were paid of 386.91136146494415 guineas per sack of grain plundered.
Jack's rank 8 crew members were paid of 398.8308541852052 guineas per sack of grain plundered.
Jack's rank 9 crew members were paid of 436.2818307435587 guineas per sack of grain plundered.
```

[2]This question was inspired by this song (click to link). Arrrr!

Question 2 (20 points):

The `BilinkedList280<I>` and `BilinkedIterator280<I>` classes in `lib280-asn1` are incomplete. There are missing method bodies in each class. Each missing method body is tagged with a `// TODO` comment. The javadoc headers for each method explain what each method is supposed to do[3]. Many of the methods you must implement override methods of the `LinkedList280<I>` superclass. Add your code right into the existing files within the `lib280-asn1` module.

When implementing the methods, consider carefully any special cases that might cause need to update the cursor position, or ensure that it remains in a valid state.

You are not permitted to modify any existing code in the .java files given. You may only fill in the missing method bodies.

Question 3 (10 points):

Write a regression tests for the `BilinkedList280<I>` class. You only need to test the methods that *you* had to write. You may generate test cases using white-box, black-box, or a combination of both methods. Again, write this code in the existing `BiLinkedList280.java` within the `lib280-asn1` project. A function header for the regression test (`main()` function) has already been provided.

Marks for this question will be earned for generating and coding good tests, not whether or not the methods being tested actually work. This means that you can still get full marks on this question even if the methods you were supposed to code in Question 2 don't work.

4 Files Provided

lib280-asn1: A copy of lib280.

Crew.java: A class that holds data about one crew member's pay and plunder.

PerformanceByRank.java: Starter code for question 1.

piratecrew.txt : Data file for question 1.

5 What to Hand In

PerformanceByRank.java: Your program for question 1. (You do not need to submit `Crew.java`).

BilinkedList280.java: Your completed doubly linked list class from question 2 and its regression test that you wrote for question 3.

BilinkedIterator280.java: Your completed iterator class from question 2.

6 Grading Rubric

The grading rubric can be found on Moodle.

[3]The javadoc comments in these files are also good examples of how we will expect you to document methods that you write yourself in future assignments.