COP 3540 – Data Structures with OOP

Project 3 – Fall 2011

Due: Monday, 24 October 2011; Drop dead date: Wednesday, 26 October 2011.
Dates / times are as of Start of Class.

Doubly-Linked Lists

Using NetBeans 7.1, you are to write a Java program using OOP principles to accommodate the following functionality

Assignment #3

Objectives:

Provide student with additional experiences with file input output.
Provide student with exercises in learning UML
Provide student with exercises in Javadoc and its various formats
Provide student with exercises in building a doubly-linked list of State objects.
Provide student with experience in inserting, deleting, and searching the doubly-linked list
Provide student with exercises in updating a linear linked list

Functionality:

Given a sequential file, States.Fall2011 on my web page, you are to build a doubly-linked list. You are familiar with the formats of the inputs and they are also described below.

Task 1: Build an array of State objects from the input data set.

Read the input data set one record (line) at a time; create a State object from each line and insert this State object into an **array** of State objects.

Task 2: Sort the array of States using an any O(n2) sort of your choice. Your sort key is state population.

Task 3: Display the array of State objects based on their position in the array

So, include a header line that states **Bubble** (or whatever sort) **Sorted Array of State Objects**. Skip a line and then display in fifty consecutive lines each state object. You are to display the state name and the population only here.

Task 5: Build a doubly-linked list of State objects. Using the array of sorted State objects, you are to build a doubly-linked list, such that all links (State objects) will appear in the same order as they are in the sorted array. (ascending by state population)

Task 6: **Display Doubly Linked List – Forward Pointers**. After building the doubly-linked list and after skipping a couple of lines, display the doubly-linked list with the header: **Doubly-Linked List of States Using Forward Pointers – Ascending**

Population. Skip a line after this header, and display the links one link per line. Display the State, its abbreviation, its population, and the region number. (If you built objects with 'all' attributes, you may display them all; otherwise, just display the state name and its population.)

Task 7: Display Double Linked List – Rear Pointers. Using the same header, you are to display the same list using the rear pointers. Start at the end of the list and using the rear pointers, display the list (same format) advancing to the front of the list. Each line is formatted as above. There is to be spaces between each output attribute for each state. Header is to be: **Doubly-Linked List of States Using Rear Pointers – Ascending Population.**

Task 8: Update the doubly-linked list and Display changes. Using the inputs provided to you in Link.States.Trans, you will need to update the linked list. Using the stateNames and populations from the input file and forward pointers in the linked list, you are to search the linked list for a hit based on stateName. If you encounter a hit on stateName, you are to replace the existing statePopulation in the node with the new state population read in from the input file. (I am aware this may now render the linked list no longer sorted by population.)

You are to use **String Tokenizer** to gain values for the two attributes on each input record. Then, after displaying a single header line that says: (left to right) Old State Population New State Population, Number of Searched Links, you are to skip one blank line and print the stateName, the new statePopulation and the number of probes it took to find the node you just updated. Print **only those nodes that were changed.**

In the case where no hit found on stateName, you are to display the input transaction formatted the same way with stateName, statePopulation, and the text: No Match under Number of Search Links. Space this out so it looks nice.

One more time, **you have PLENTY of time if you start right away and work slowly and methodically.**

Procedure:

I urge you to tackle this problem incrementally. Using a small sample of the input file to test your procedure. Then build the entire linked list. Verify as you go. Use the toString method or other display method to your advantage and VERIFY that you are in fact building the doubly-linked list correctly!

Deliverable: Your zipped folder to me MUST include copies of your data files. I will need these to run your program and to test it. Do not provide me with output your program generates. I will get your program to generate your outputs. As noted, your outputs will be displayed on the screen.

You are to zip all files in your P3 folder as expected and Send (do not Add) them to me via Assignment tabs in Blackboard using the same naming conventions as in

previous deliverables. Your zip file is NOT to include your N-number. Rather, it must be project3.youruserid, as project3broggio NOT project3n00010109.

Grade Sheet for Program 3

Name: _____

Grade: _____

Source Code – 20 points. Points earned: _____

Indentation
Internal comments
Scope terminators
Overall program documentation.

Program Design – 20 points. Points earned: _____

Your design must reflect good object-oriented design as we have discussed over and over. Objects are to contain the methods that operate on the data they contain as much as possible. Ensure your UML design reflects these classes and their methods.

UML – (10 points) – no excuse to not have this wonderful at this time too.

Correctness, associations, completeness. This means that the classes you identify are correct, that associations are indicated, and that the attributes and methods are documented within the classes.

Pseudo-Code – (10 points).

There are plenty of examples that you have had access to. Provide the pseudo-code for your Linked List class – or whatever class you define *first* and *last* in as instance variables that point to the linked list.

Javadoc – 10 points - no excuse to not have this wonderful this time!

Appropriateness and completeness of comments
ALL methods must have Javadoc comments up front that are meaningful, please.

Outputs – 40 points

Accuracy and **Format. All functionality present!**
Skip lines in between displayed numbers for **readability**.
Include headers / descriptors as you may feel appropriate, but they need to be respectable by my standards..

Program must run correctly to receive a passing grade and to receive at least partial credits above. If the program does not