COP 3540 – Data Structures with OOP

Project 3 – Spring 2007

Due: Wednesday, March 7th, 2007

Doubly-Linked Lists

Using NetBeans 5.0, 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.Spring2007 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.

Note: Tasks 1 through 4 are optional. If done, and if done correctly, you may earn an additional 15 points added to your overall project total. Please note: all four tasks must execute perfectly to get the fifteen points.

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 in **main**().

Task 2: Display the array of State objects based on their position in the array.

So, include a header line that states Unsorted Array of State Objects. Skip a line and then display in fifty consecutive lines each state object.

Task 3: Sort the array of States using an Insertion Sort.

Task 4: Display the array of sorted State objects.

As in Task 2 above, skip a few lines and display the sorted list of State objects. The header should read: Array of State Objects – Sorted via Insertion Sort.

Alternatively, you may start here by creating State objects from each input record and inserting them into the doubly-linked list. If you do not elect to undertake tasks 1 through 4 (and this is fine….), then the States will appear in the order that they appear in from the input file. Otherwise, they will be in sorted order from Task 3 above.)

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

Task 6: 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. 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.

Task 7: Use 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.

Task 8: Update the doubly-linked list and Display changes. Using the inputs provided to you in Link.States.Trans, you are to update the linked list. Using the forward pointers only, you are to search the linked list for a hit. If you encounter a hit on stateName, you are to replace the existing statePopulation with the state population indicated on the input file. (You may use String Tokenizer to pick up the two fields 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 those items from the old node and from the updated node. Print **only those that were changed.** In the case where no match is found, you are to display the input transaction formatted as: state name, population, and the text: No Match. Keep track of the number of links searched to find or not find the target. 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!

Do this slowly and methodically. Do not rush, but start right away.

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 Digital Dropbox 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 project2n00010109.

Grading

Source Code – 20 points

Indentation
Internal comments
Scope terminators
Overall program documentation.

Program Design – 20 points

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.

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.

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

You were presented examples of 'the right stuff.'

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.

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..

Bonus – all or nothing. – 15 points. (Described above)