COP 2551 – Introduction to OOP
Program #4
Due: Wednesday, August 3rd, 2011 – midnight
Drop dead: Thursday, August 4th, 2011 - midnight

Using NetBeans 6.9 or later version you are to write a Java program using OOP principles to accommodate the following functionality

Assignment #4

Objectives:

Provide student with experience building arrays of objects
Provide student with opportunity in doing file input and output.
Provide student with exercises in learning UML
Provide student with exercises in Javadoc and its various formats
Provide student with exercises in searching and traversing the array of objects.

Functionality:

Using the external file States.txt, you are to do the following:

1. Build an array of State objects.

You are to develop a class named State and create as many objects of this type – one object for each record (line) from the input file. (Take a look at the file. The first several lines describe the relative positions of the data underneath. You may eliminate those lines and start reading from Washington, etc. When you Save Target As to download this state file, you may eliminate those first lines at that time. But you need the information in your program in order to parse. Thus, only process the data lines.

You are **not** to alter the data, however, in any way.) Each **state** object will have properties as shown and defined individually as Strings, ints, or whathaveyou, as appropriate for each state object. Hint: you may use substring method in class String to parse as expected. Please note that the first two lines provide the layout of the data below.

2. **Display files Requirement**. From main() you are to write code to display the array of objects to the screen – recommend using toString(). This is to include a nice looking header spanning the display line followed by nicely spaced columns of state attributes aligned under their respective header. Text data is to be left justified; numeric data, always right justified with commas as appropriate.

3. **Copy the array of objects to an external file** too, and name the output file, **outStates.txt**. See slides for example. You will need FileWriter and similar classes / objects.

For the file output, only write the records. No header line. So the individual lines should look like what you display in #2 above – nicely spaced out; no header.

4. Scan and Total the Populations Requirement. Using the array of State objects that you have built, you are to examine each state object in turn and accumulate data. You are to access each state object in turn and total up the state populations **per region**. (Each state belongs to a region, as you can see from the data) At the end of scanning and accumulating this data from your array of states, you are to print out a header that says State Population By Region (left justified) (see ahead)and underneath this, you are to print the region name, the number of states and the average population of states in that region. Your format for these detail lines should appear as:

State Population By Region

New England 6 321,123
(or whatever average population is)

Middle Atlantic 6 x,xxx,xxx

etc.

(note the spacing; use the formatting links provided on my web page) Note also that I only have three regions.

At the **end** of displaying these lines for each of the regions, you are to skip a couple of lines and display:

Total State Population of the Indicated Regions is: (this will be a single line). This will look like:

Total State Population for the West is 21,233,222

Total State Population for the Middle_Atlantic is 212,222,222

etc.

At the very end, print an overall total in the format:

Overall State Count nn 331,222,333
(or whatever you add up to)

nn is the total number of states processed (number of objects) and the population is the total population of all those states processed.

6. Search Requirement. Bonus. 20 points to be applied to your Programming Grade. Please note: This will <u>ONLY</u> apply if everything else meets the specifications. This cannot make up for missing UML, Javadoc, unmet specifications, etc., okay? All else must be wonderful for this bonus to apply.

You are to search through the entire array of State objects that are in region 6 and region 3. If any of those state populations exceed 5,000,000, you are to display: (for example)

Region: South
State: Georgia
State population: 8,345,111 (include the commas!)
<blank line>
Region: South
State: Florida
State capital: Tallahassee
State population: 14,345,444
<blank line>
Region: West
State: California
State capital: Sacramento
State population: 32,444,555
Etc.

You are to process region 3 before processing region 6.

Prior to displaying these detail lines, you are to display the header:

State Populations Exceeding 5,000,000 for Regions 3 and 6.
The detail lines above are to appear underneath, as displayed – a single blank line separates individual sets of three lines as shown above.

At the end of searching, you are to display the number of successful searches according to the format:

<blank line>
Summary Statistics:
Number of State Populations Exceeding 5,000,000: <an integer>