COP 2551 – Intro to OOP
Program #3
Due: 27 October 2010 (Wednesday)

Using NetBeans 6.7.1 or later version, you are to write a Java program using OOP principles to accommodate the following functionality

Assignment #3

Objectives:

Provide student with opportunity in doing file input output.
Provide student with exercises in learning UML
Provide student with exercises in Javadoc and its various formats
Provide student with opportunity to create objects and develop a number of methods within their own objects
Provide student with opportunity to use String methods for accessing inputs
Provide student with exposure to static attributes and static methods.

Functionality:

Using States.Fall2010.txt, you are to do the following: You are to read data from an input file using the Buffered Reader class and String operations (see the String class), create objects from each input line and display each of these objects to the screen. You will gain essential practical knowledge of Javadoc, as well as undertake architectural design (UML). Assignment does **not** include use of arrays (next assignment). You will not be writing to an output sequential file.

Step 1. Access the External File. You are to access a file named States.Fall2010.txt of **seven records** (you may look at the file).

Step 2. Create seven state objects. For each line (record) read, you are to create an object of type State. (This will entail creating a new class called State. This is in addition to your accustomed class, Main.) Since this assignment precedes our learning of arrays, you may call the objects state1, state2, etc. You will need to do file I/O to read the file. (See link entitled, Very Brief Introduction to Java I/O on my web page for example code). You will also likely need to use the substring method in the class String in order to access properties in each input line from the file. Each time you create an object, you are to increment a static counter in the State class. (We will discuss, but they are found in the next chapter)

Step 3. Display the objects. Once the objects are created, you are to access the objects and display the state name, its capital and its population (use commas for the population). Use the toString method for displaying your output. This method must be within your State class.

Step 4. Search and Display by State Abbreviation. Once this is done, you are to first print out a header line (shown below) and then prompt the user (me) for a two-character state abbreviation. For simplicity, I will only submit valid state codes. You are to loop through the state objects to see if the state code that was inputted equals the state code for a particular state object. When a match occurs, you are to display the message (left justified): Match found: Input: <my input code> followed by the state abbreviation, the state name, its capital, and its population each spaced out nicely on a single line. I may want to search for more than one state, so allow me multiple opportunities. A 'no' will tell you that I am done searching. Your prompt should indicate that I am to enter a two-character state code or a 'no' to terminate the search.

Example output:

User	Input	State	Capital	Population
<skip a line>				
Input	CT	Connecticut	Hartford	3,274,069
Input	MA	Massachusetts	Boston	6,147,132

Step 5. Print Totals. At the very end, you are to skip a couple of lines and print out the number of State objects you created in the format: (left justified) Number of State Objects created is: <an integer> The number '7' below must come from your static counter using a static method in main().

Example output:

Number of State Objects is: 7

Step 6 UML. You must include a UML design (your architectural design) showing object dependencies and object contents. Your book does a good job here. We will also discuss. Follow the book carefully.

Step 7 Javadoc You are to include Javadoc documentation in your program. Be careful. If you do not format the Javadoc code correctly within your source code, it will not be displayed in HTML format. **ALL** methods are to include Javadoc documentation and pay close attention to any parameters that might be needed. All methods are to have an @params and a @returns even if **nothing is returned.**

Note: You will need to import java.io and java.util. BufferedReader, FileReader, and IOException are in java.io We will discuss as needed.

UML

You are to include a UML class diagram. You may use Word or Power Point. No other technology may be used!

Use the examples in your 2551 text. Each class listed in your UML diagram must have attributes listed (name, type). Methods must be shown with visibility indicator, and number /type of arguments plus the return type. Connect all classes (label the associations).

'Drag' your UML design file into your P3 subfolder within your COP2551 desktop folder. It will be included in the zip file to me.

Javadoc

All programming is to be accompanied by appropriate Javadoc.

Each class and each method in each class (object) must have appropriate documentation associated with it and appear just prior to the method header.

Appropriate documentation for methods consists of a short description (single sentence) plus any parameters and any return types specified via @params and @return.

Appropriate documentation for classes includes several sentences describing the purpose of the class. Include @author and any other documentation that assists in documenting that particular class. Of course, objects related to this class should be described via accompanying UML diagrams.

When you generate your Javadoc, all these comments plus a great deal more will be automatically generated for you by NetBeans.

Generation of Javadoc: (Please note that there are other ways within NetBeans to generate Javadoc. But they generate lesser Javadoc than the process below. Use this approach:

From the Project window pane, right click on the projectname at the top of the pane.
Select Properties (last choice in the drop down).
Select Documenting
Select Document Additional Tags – Author
Click OK
Then go to the Build Menu (drop down) and select Generate Javadoc
All Javadoc will be generated and moved to your program folder. (What a deal!)

You are to zip all files in your P3 as expected and Send them to me via the Assignment portion of Blackboard using the same naming conventions as in the past. **If you have any questions, please ask early!!**