COP 3540 – Data Structures with OOP
Program #1 – Spring 2011
Due: 31 Jan 2011 (Monday) Start of Class
Drop Dead date: 2 Feb 2011 – Start of Class

Using NetBeans 6.7 or 6.8, you are to write a Java program using OOP principles to accommodate the following functionality

Assignment #1

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, sorting, and comparisons of key searches and sort routines.

Functionality:

Using the external file, Euro-Countries, (Open with WordPad) you are to do the following:

1. **Build Array of Euro-Country Objects.** Using input file, you are to build an array of objects. You are to design and develop a class named **Euro-Country** (in addition to your class named **Main**) and create objects of **type (class) Euro-Country** - one object for each record (line) from the input file, subject to the constraints in 2 below. (You must use Buffered Reader class). **Each Country object will have five properties defined individually as integers or Strings as appropriate for each Country attribute. You should download this file and use this as input to your program to create the array of Country objects. Be certain to drag this file into your project folder.** (Be aware that this file may be modified for follow on programming assignments.)

2. **Exception Processing**. Using Exception Processing Procedures (try…catch), your program is to diagnose input fields (attributes) in error. You need to check for numeric fields in the region number (permissible values are 1 through 4) and the numeric population field that represents the population of the capital of the particular country. (Input attribute must be able to be converted to an integer) For those input 'records' that have problems, you are to **not** build an object for them and they are not to be entered into the array of Euro-Country objects. Rather, as encountered, you are to display (print) a single header (only display this one time): B A D I N P U T (left justified) followed by the entire violating

record (on a single line as read from the external file), followed by the appropriate messages (you may have one or two messages – one per line): "Bad / Missing Input Region Number followed by a single space followed by the number and/or Invalid Population Value followed by a space followed by the offending input value. (Again, you may have one or two violations for each of these bad records).

3. Print Array of Country Objects. From main() you are to access this array of Euro-Country objects and write code to display the array. Use toString() method in the Euro-Country objects you have created. You are to first skip a few lines (blank lines) and then print a column header, followed by a blank line, followed by single spaced (one line of attributes per object (country) as output. All attributes are to be printed per valid Euro-country object.
These are to be single spaced and all aligned nicely. (Attributes are Country Name, Capital, the Capital's Population and the Region Name (such as Central_Europe). All data is to align underneath these headers. Numeric data should be right justified (the population). All other data may be left justified. See examples on my web page for formatting examples.

4. Sort and Display array using an ascending interchange sort using the country name attribute of each object as your sort key.

5. Sequential Search and Print Results. Given the input search file, **Euro-Search** (a text file to be supplied) you are to search the **sorted array** to 'find' or 'not-find' each **search argument** in the search file using a **sequential search**. **Your search key (from the transaction file) is the country name.**

After skipping two lines and displaying an appropriate column header **one** time, you are to display the search key itself, the number of probes to find / not find, and text: 'found' or 'not found' to indicate the success of your search. **(Some of the input search arguments may not match. This is intended.**

6. Binary Search. Similar to (5) above, but use a **binary** search. Your results, as above, are to produce a single column header and to display the search key, number of probes to 'find' or 'not find' the target followed by text 'found' or 'not found' text. Again, line these up and include appropriate spacing between each of these attributes on a line. Be certain to skip two lines between the search results generated from (5) and (6). For each search argument, you need only display the same format as above: search key, number of probes, and found or not found indicators. The same input file is used…

At your discretion, you may read the Euro-Search file into a table and process the table once for item 5 and once for item 6, or, alternatively (much less efficient), you may not build the table in your program from the input euro-search file and rather open the file and read and process for task 5 above; close the file; open and read and process the file again for task 6 above. Choice is yours.

7. **Ragged Array.** Given the four European geographical areas that I have somewhat arbitrarily devised , you are to build four ragged arrays – one for each geographical region. Each array is to have **only** the objects belonging to that region of Asia. (You may create a set of four independent single-dimensional arrays if you wish – not as 'ragged' as I'd like, but acceptable….).

8. **Printing Ragged Arrays.** By sending each of these ragged arrays (in turn) to a special print routine, you are to then display in a professional manner each of the four one-dimensional array of region objects <u>in turn</u> and their respective countries in the following format:

Northern-Europe Countries <note countries below are listed in alphabetical order>

<one blank line>

Country name Capital

<one blank line>

Denmark Copenhagan (entries are to be left justified under the header)
Finland Helsinki
etc.

<two blank lines>

Eastern-Europe Countries.

Country name Capital

Belarus Minsk
Estonia Tallin <note these are single spaced>

etc.

At the end of printing, you are to display:

Total Countries in Northern Europe: xxxxx <line up these columns>
Total Countries in Eastern Europe: xxxxx
…
Total Countries in Europe: xxxxx

All done. Be proud of your work.

UML