COP 3540 – Data Structures with OOP

Project 2 Alt – Spring 2011

Due: Monday, 21 February 2pm

Drop dead date/time: 21 Feb, 2pm (unchanged)

Circular Queues

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

Assignment #2

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 an array of objects to support priority queuing and circular queuing.
Provide student with experience in inserting, deleting, and searching priority queue of objects.

Functionality:

Task 1: Build ArrayLists: Given a sequential file, ***Euro-Countries.2.txt*** on my web page, you are to build a series of circular priority queues discussed ahead, and one single input stack.

Here in Step 1, you are to build four single dimensional arrays **using ArrayList().** Thus you will not declare an input size restriction on the array. The input data is not sorted in any way. As you read an input record from this file, create an object and write the object to the appropriate ArrayList. There should be one array for each region as we have done. You are only to build ArrayLists for those records whose region-code is 1-4. When you have fully processed the input file, you will have four arrays of objects of differing lengths- something you are aware of.

Do **not** sort the original array of strings that you read as inputs. The order / contents is changed from program 1.

Task 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 any array of Euro-Country objects. Rather, these bad records are to be written to an output sequential file entitled, ***ErrorFile.txt.*** As you encounter a bad input, your catch code is to simply write the entire record unchanged to the external errorfile. The next 'write' is to include: BAD INPUT (left justified) 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). At the end of building the circular queues, you must close() this file. Be certain to drag it into your project folder so I can look at it later.

Task 3: Build Circular Queues – Initial Build. Using each array in turn, you are to build a **circular queue** from each array. Each circular queue is to be of size 10. There is no priority in inserting elements in the queues from the array lists. Build each of the regional circular queues with items as they appear in the arrayLists subject to constraints below.

As you build each circular queue from its respective unsorted sorted array, you will note that there are more entries in a couple of arrays than there is room in the circular queue for that array.
Thus you have a problem. 'No Room in the Inn."

Task 4: Insufficient Room in the Queues. Continuation of Step 3 As you attempt to add to the circular queues and you discover that the queue is full, you are to display a message citing the country name of the object you wish to add to the queue, display on the next line the country name of the object you are removing (remove()) from the queue to make room for the new object (queue[front++] mod queueSize), add this new object into the queue (queue[++rear] (mod queueSize) = new object), and then print "success" on the third line indicating that the queue now has the new item in it and has removed() one item. (Lots of fun!!!) Display these actions as:

Wish to Add: <country name>
Removed: <country name>
Success in adding <country name>

Task 5: Display Circular Queues. Upon completion of building the circular queues, you are to print out **nicely formatted** each queue in column order such as:

Circular Queue for Northern Europe
XX <country names> front to rear
XX
XX
…
XX

Circular Queue for Central Europe
XX
XX
XX
…

All of your answers should be the same. So check with your classmates!! Also, take advantage of the discussion board on Blackboard.

Task 6: Remove Elements from Queues. Remove four objects from Eastern Europe and from Western Europe.
Print out what you have removed using a header like:

Removal of Four Objects from Eastern Europe
<list country names in order of removal of the four objects>
Skip a line (one blank line)

Removal of Four Objects from Western Europe
<list country names in order of removal of the four objects>
Skip three lines

Task 7: Display Resulting Circular Queues. Clearly label and print out all resulting circular queues.
Use a header and detail lines as follows:
Final Circular Queue for Eastern Europe
<list country names in order>
Skip a line

Final Circular Queue for Northern Europe
<list country names in order>
Skip a line
etc.

(the order of the queues (that is, Eastern before Northern, etc. does not matter)

DONE!!!!

Note: in this alternative, your front and rear pointers will wrap the queue boundary. So the notion of "moding" the queue size is important. Front and rear pointers will move throughout the queue structures.