Program #5
File I/O; Arrays of Objects; Iterators, Simple Sorts
COP 2521_706
Fall 2005 – 11/6/05

Note: This program and all remaining programs will be run using the JGrasp facility included as a CD with your textbook. The details of the turning will be discussed ahead.

Objectives:

- Gain experience using multiple classes
- To gain experience with StringTokenizer **(inputting multiple items at one time)**
- To gain experience reading input from an external file and more Java I/O
- To gain optional experience with ArrayList for dynamic array length **(a very nice feature of Java)**
- To gain experience with single and two-dimensional arrays, and arrays of objects
- To gain more experience with the toString methods
- To gain experience writing a larger program and learning how to solve a larger problem.
- To gain experience using an On^2 sort of your choice (interchange or exchange).

Background: Study the programs on Quizzes and Questions and Answers and Complexity. Consider also the UML class diagrams we have had so far including comments on your last UML design from Program 4. Also, you will need to study the StringTokenizer class and its methods. I will present an approach to accessing your external file in JGrasp.

Program 5 is an extension of these ideas plus some good work on arrays, ArrayList (optional), and traditional arrays. Please study this assignment carefully so we can answer any questions you might have. This assignment is a reasonably hefty one and has a number of parts. The key, of course as always, **just do a LITTLE bit at a time**, verifying as you go…I will provide hints during class as it is almost an imperative to develop this programming assignment incrementally.

Objects of the Question class – created by the MiniQuiz class, which contains our static main method, created objects by passing questions and answers to Question as well as setting the complexity for that questions. We will modify this to include arrays…and more. Given this backdrop – and BE SURE to study these examples FIRST – our job is to meet the practical requirements discussed below.

Processing

1. **Introduction**. You need to create an array of objects – one object for each quiz. You are to obtain the data used to create each object from a file called *quiz.txt*. You should call this array of objects that you create from the input file *quizbank*. Each object in the array will have a question, answer, and initial complexity level. Despite the fact that I have eliminated implementing the Complexity interface, we will still alter the complexity attribute of each question, as you shall see ahead. I have provided the initial data in the file *quiz.txt* that you can access to build your array of questions.

2. **Build the array of objects from input file.** Build this array by declaring its initial size to be five. I will give you *several* separate strings of data in *quiz*.txt from which you will build the array of objects. Note that the tokens are separated by a comma. (Refer to StringTokenizer for delimiters on input tokens.)

3. **Display original array.** Having built the array of objects, you will then need to print out the array – very nicely formatted, so that your output includes:

Question	Answer	Complexity
<data> (left just)	<data> (left just)	<data> (centered)

okay? Do not use the iterator form of the for to print out this array list.

4. **Add to the array.** Now, via prompts (use BufferedReader as we have in the past – and please note that this is a different 'kind' of inputstream), you are to solicit from the client (moi) the number of questions that I wish to add to the quiz question repository. Your program should be able to accept any reasonable integer as input, let's say from 1 to 10. You need not bother about editing the range of acceptable values (1 to 10) at this time.

Given this number, you are to loop 'n' times depending on my input. Given your prompt, I will enter two strings and an integer (question, answer, and initial complexity level). You may assume, for simplicity, that I will input these three items at one time separated by a comma (,) before I press Enter. You will again need the StringTokenizer class to assist you here as well as reading the initial file used to build the initial array.

Given each input, you are increase the size of the array from its initial length to the new size by whatever method you choose: increaseSize() or by using ArrayList (bonus points).

5. **Display entire array again.** Afford the client the option to print this new (complete) quiz, for which I will select Y or y. Use the iterator form of the for statement for this displaying.

6. **Administer quiz to three students.**. Now, the quiz must be administered to students. Three students will take the quiz, and their names are Larry, Curly, and Moe (the Three

Stooges of yore). Each of these three students will take the quiz consisting of all of the questions. Cycle through the entire quiz and accept a String answer to each question, keeping score internally for each Student. As each Student takes the Quiz, you are to return a 'correct' or 'incorrect' response, but no option to redo in case of an incorrect answer. Each 'student' gets a single chance to answer the question. You should proceed from question to question for all three Students. You are to keep track of the number of correct answers / incorrect answers for each student. *You need to keep track of these in a array. (took out 'two-dimensional.' Depends on how you implement this.)* You may determine the structure.

7.. For each student, Print Results. Print out each student's name, number correct and incorrect answers – one line per student.

8. Sort the questions / answers. Sort the questions (objects) on decreasing level of complexity. (Use the sort of your choice to sort these objects). This may/may not be a bonus. I will let you know. *Sort can be taken right out of the book on page 501.*

9.. Change Complexities. After all three (*or however many students you have….)* students have finished and you have recorded all answers to each question (number of correct and incorrect) for each student, you are to examine each question and determine the frequency of missed answers to each question. *Remove: (I will ensure that there are no duplicate number of misses at this time). The complexity of questions should be changed as a function of the number of individuals who took the test and the maximum number, therefore, of incorrect answers for each question. Clearly, if four people take the test, then any given question could have four incorrect answers. Change this complexity and any others with that many incorrect answers to 4. For questions having the same number of incorrect answers, the complexity should be the same for those questions. (Eliminate: ➔ Change the complexity of the most frequently missed question to 10, the second to a 9, and so on down the line.)* If a question had no incorrect answers, make the complexity level for that question = 1. *Again, you should use an appropriate integer array – a three (one for each student) by n (the number of answers per student), or perhaps a one dimensional array for each Student object for his/her answers. Bottom line: You may use an array structure of your own choosing.*

10. Print out the Quiz database showing all questions, answers, and complexity in decreasing levels of complexity..

11. Done. .

Inputs

So that we are all dealing with the same data, I will use input that looks something like these in the quiz.dat file. *Notice that the spaces following the commas has been deleted.*

Who_was_the_first_US_President?,George_Washington,1
Who_was_the_second_US_President?,John_Adams,1
Who_was_the_third_US_President?,Thomas_Jefferson,1