W. P. CAREY SCHOOL of BUSINESS
ARIZONA STATE UNIVERSITY

CIS 494 – Business Information Systems Development II – Spring 2015

In-class Exercise 1

Due on Blackboard: Wednesday, January 21, by 11:59 PM

Create a new Java application that reads a *variable* number of student test scores and outputs the average and the highest test score.

Skills and Objective: The purpose of this first exercise is to ensure that you are able to recollect how to write methods, call methods, classes, create objects, arrays, arrays of objects, do input using the console, and are able to write basic structured programs that utilize for loops, do loops, if statements, etc.

Apart from console input/output, using and comparing Strings, and providing formatting instructions, you will find that the Java syntax for most statements is the same as in C#.

Style: Use descriptive variable names and utilize *camelCase* for naming your variables and methods., e.g. “firstName”, “calculateAverage()”, “averageScore”, “lookupRecord()”.

Develop your own detailed logic for implementing the various aspects of the program – however, be guided by the following:

- Have a Score class. Each Score should have a private instance variable of type integer to store the score. It should also have get/set ***accessors***/***mutators***. *You wrote mutators in CIS 340!*
    - To ensure you get adequate practice in creating mutators, add one more instance variable, such as “totalOutOfPoints” with mutators.

- Have a ScoreManagement class.
    - This class should have an array of Scores. When you are creating the array, allocate it 10 elements.
    - Have one method each for reading scores, calculating average, and finding the highest score.

Reading Scores:

- The user should be able to enter a ***variable*** number of scores without telling you in advance how many scores he or she would like to enter. Make use of the Scanner class *(Introduction to Java Programming, section 2.3, page 37)* as well as parsing functions built into the Integer/Double classes *(section 4.4, page 137).*

  Therefore, utilize a sentinel controlled loop that will keep asking for new input so long as the user does not enter -1. If a -1 is not entered, treat input as a valid student score. If a -1 is entered, treat that as an indication the user does not want to enter more scores.

Calculating and Displaying Average:

- Use the printf() method to format the average. The printf method can accept formatting instructions just like the placeholders in C# WriteLine methods.

  You can look up the JavaDocs for details on using formatting instructions, however the JavaDocs explanations and examples are not very easy to understand. The book on pages145-147, instructions and examples that are a bit easier to understand..

  Sytem.out.printf("This: %1$.2f, is a formatted decimal", doubleX).

  The string above is "escaped" by %, which denotes that's where the first argument must be placed. It then instructs that the first argument that follows, $1, be formatted with 2 places of decimal precision, .2, and be treated as a floating point, f.

  Sytem.out.printf("This: %1$d, is a formatted integer", intX).

  The string above is "escaped" by %. It then instructs that the first argument that follows, $1, be formatted as an integer, d.

- Call each of the three methods in sequence from the main method of the Score Management class.

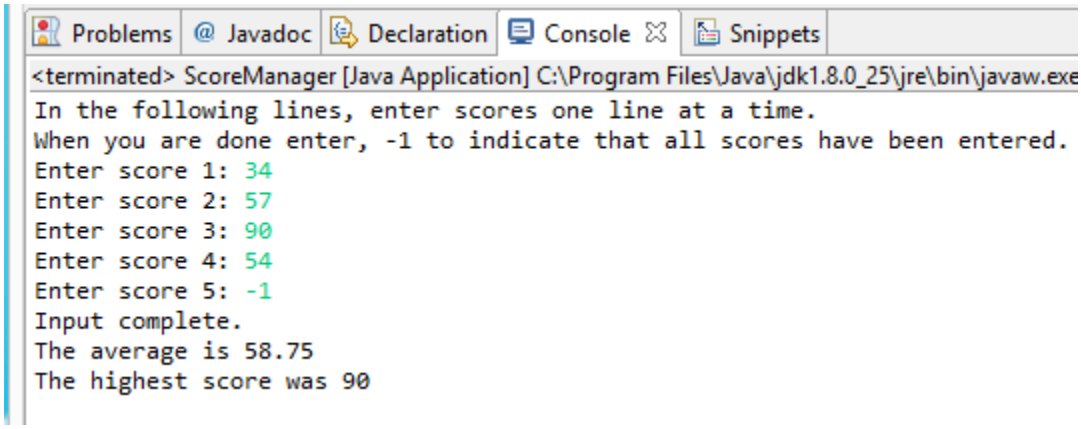
C# vs Java things to try out:

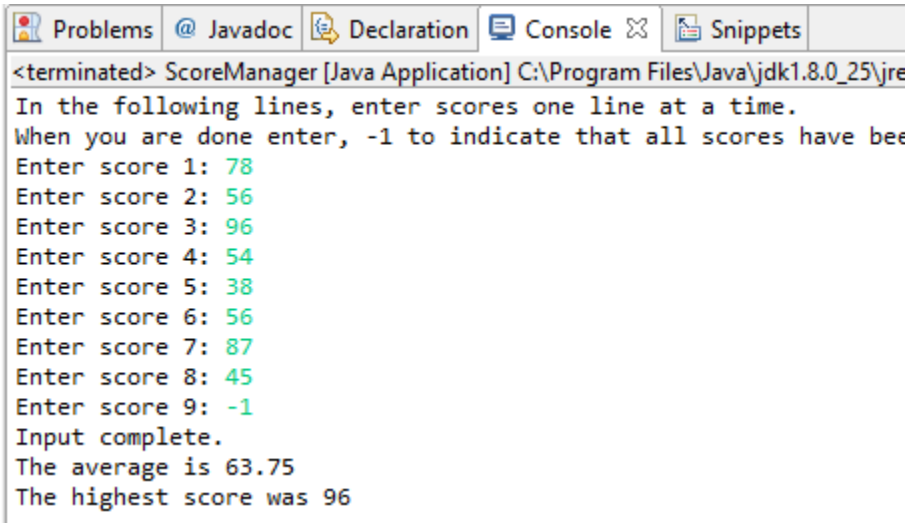
- ☞ Try declaring arrays with the brackets alongside the variable name, not datatype.
- ☞ Strings are classes in Java. Try creating them using the String constructor as well as how you would normally do it.
- ☞ Your String variables are objects. Try comparing them for equality using "==" and the .equals method of the String class.
- ☞ The Scanner class has methods to scan for specific data types, e.g., nextDouble(). Try them out.

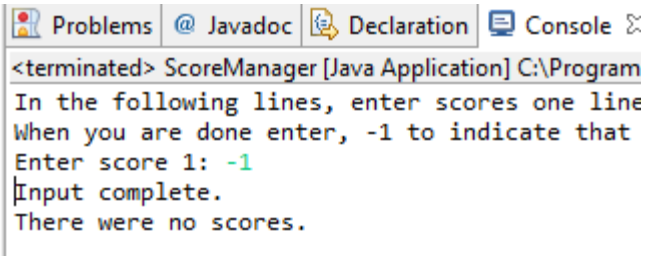
Optionally, develop the program further:

- ☞ Expand the score class to contain names associated with scores. Ask for the scores along with names.
- ☞ Provide functionality to look up scores by names by writing a method that accepts a name. It should loop through the entire array and compare for the name sent to it as a parameter. The method should return the score associated with the name.

Sample Output







Submitting Files

Submission should be made using a zip file that contains the entire Eclipse project folder. You will need to ***zip the entire project folder***. The folder will automatically contain the class source files as well as the compiled .class files.

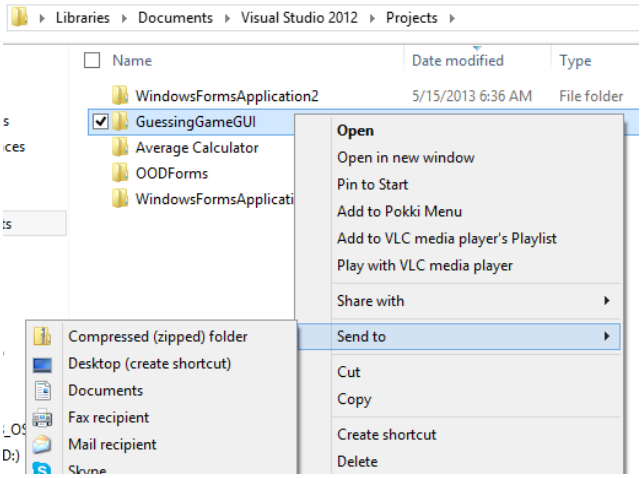
ZIP file should be named: **AX.zip or IC.zip**

where X is the in-class assignment number, e.g., A1.zip is the submission file for Assignment 1.

Note: *The ZIP filename is independent of the Project name. Do not name your project A1.*

How to Properly ZIP and Submit Your Project Files:

- Go to the folder within {Eclipse Workspace}\
- ZIP the entire top-level folder for your project by right-clicking your project folder and selecting Send to | Compressed (zipped) folder.
- Finally, submit the ZIP file using the submission link on Blackboard by the due-date and time listed on the assignment. Upload the ZIP file.



Using built-in windows zip tools: http://windows.microsoft.com/en-us/windows/compress-uncompress-files-zip-files

Verify your files BEFORE and AFTER submission:

- Check for actual class files being present in the folder before you zip it.
- ***Ensure that you are not zipping a short-cut to the folder.***
- After zipping, check file size. A file size under 4K likely does not contain all the files.
- Unzip, extract all files, and verify you see actual files, not a solitary short-cut.
- Uncompress your zip file before submitting and verify that files are present.
- Make sure you have ***submitted*** your file and not just saved a draft on Blackboard. ***A blue clock indicates a submission in progress, i.e. a draft, not a submission. The draft is accessible only to you. You will get a ZERO if you only ever save a draft on Blackboard and never submit your files.***

- Download your zip file after submitting, uncompress, and again verify that your files are present. Test your files in Visual Studio after uncompressing.

This takes an extra couple of minutes. Please do it if your grade is important to you. If you do this, you will not end up submitting a bad file. If you submit an empty file, or one containing only a shortcut, or a bad zip file, or a bad project file, you will receive a score of zero and your only recourse will be to do the makeup assignment at the end of the semester.