Project 8: Weighted Grades App

Due: Skeleton Code (ungraded) – Friday, October 18, 2013 by 11:59 PM (checks class, method names, etc.)
Completed Code (100%) – Monday, October 21, 2013 by 11:59 PM

Deliverables

Your project files should be submitted to Web-CAT by the due date and time specified above (see the Lab Guidelines for information on submitting project files). You may submit your skeleton code files until Fri 18 Oct 2013 11:59 PM. You should do your best to get this done in lab on Wed or by the end of the Help session on Fri to take advantage of this prior to the Fri night deadline. You must submit your completed code files to Web-CAT no later than 11:59 PM on the due date for the completed code to avoid a late penalty for the project. You may submit your completed code up to 24 hours after the due date, but there is a late penalty of 15 points. No projects will be accepted after the one day late period. If you are unable to submit via Web-CAT, you should e-mail your project Java files in a zip file to your lab instructor before the deadline.

Files to submit to Web-CAT:

WeightedGrades.java
WeightedGradesApp.java

Specifications - Use arrays in this project; ArrayLists are not allowed!

Overview: This project will create two classes: (1) WeightedGrades is a class representing a set of weighted grades for a student and (2) WeightedGradesApp is a driver class with a main method that reads input from a file specified as a command line argument, creates a WeightedGrades object, then provides options for printing the grade report, printing grades for a category, adding a new grade, deleting a grade and quitting the application. **I strongly recommend that you create a new folder for this project.**

- **WeightedGrades.java**

  Requirements: Create WeightedGrades class that stores the students's name, the number of grades, and an array of grades (where each grade has a category code as the first character). This class should also include five double constants representing the "weights" for each grade category. It also includes methods to get name, get grades, get numGrades, and methods to print a report, print a category of grades, add a grade, delete a grade, calculate weighted average of the grades, and others as described below. This class should handle the following are five case sensitive grade categories: activity ('a'), quiz ('q'), project ('p'), exam ('e'), and final exam ('f').

  Design: The WeightedGrades class has fields, a constructor, and methods as outlined below.

  (1) **Fields**: instance variables for student's name which is a String, number of grades, and a String array that holds the grades. These are the only instance variables this class should have and they should be private so that they are not directly accessible from outside of the class. This class should also include five public double constants representing the "weights" for each of the categories of grades: `ACTV_WT = 0.05, QUIZ_WT = 0.10, PROJ_WT = 0.25, EXAM_WT = 0.30, and FINAL_EXAM_WT = 0.30;`

(2) **Constructor**: Your WeightedGrades class must contain a constructor that accepts three parameters representing the name as a String, number of grades as an int, and a String array of grades. Below is an example of how the constructor could be used in main to create a WeightedGrades object:

```
WeightedGrades student = new WeightedGrades(name, numGrades, grades);
```

(3) **Methods**: Usually a class provides methods to access (or read) and modify each of its instance variables (known as get and set methods) along with any other required methods. The methods for the Player class are described below.

- `getName`: Accepts no parameters and returns a String representing the student's name.
- `getNumGrades:` Accepts no parameters and returns an int representing the number of grades stored in the grades array.
- `getGrades`: Accepts no parameters and returns a String array representing the grades array field.
- `gradesByCategory`: Accepts a char representing the category and returns a double array representing the grades in that category field. The length of array should be the same as the number of grades in the category.
- `toString`: Returns a String representing the processed information in the WeightedGrades object as shown below, including newline and tab escape sequences and formatting for the course average. <u>Note that no lines are skipped</u>.

```
      Student Name: Pat Smith
      Activities: 100.0 90.0 95.0
      Quizzes: 90.0 80.0 80.0 70.0
      Projects: 95.0 100.0 85.0
      Exams: 87.5 85.0
      Final Exam: 90.0
      Course Average: 88.96
```

- `addGrade`: Accepts a String parameter representing a new grade (including category prefix) to add to the grades array; and returns nothing. Note that when an array element is added, it should be placed at the next available index indicated by the field for number of grades recorded. If this will exceed the capacity of the array, you must call the increaseGradesCapacity method described below before you add the new grade to the grades array.
- `deleteGrade`: Accepts a String parameter representing a grade (including category prefix) to delete from the grades array; deletes the grade if found in the array (two grades are equal if the categories match and double values match); if the grade is deleted, the number of grades should be reduced by 1 and true is returned; if the grade is not deleted false is returned. Note that when an array element is removed, the remaining elements must be shifted as appropriate to utilize the vacated position. After the elements are shifted, there should be an additional unoccupied location at the end of the array and it should be set to " ". This method deletes at most one grade each time it is invoked.
- `increaseGradesCapacity`: Accepts no parameters and has no return value, increases the capacity (or length) of the grades array by one. You must create a new temp array that is one element larger, copy the old array to the temp array, and then

assign the temp array to the old array. See the example on pages 398-401 in your text. Specifically, see the increaseSize() method on page 399. Note that this method doubles the length of the array; whereas your method should only increase the length by one.

- `average`: Accepts a double array and returns a double representing the average for the values in the array. If there are no values in the array, 0.0 should be returned as the average. This method and gradesByCategory should be called by the courseAvg method below to get the average for each grade category.
- `courseAvg`: Accepts no parameters and returns a double representing the weighted average for the grades in the grades array. For example using a formula such as:
  ```
  avg = actvAvg * ACTV_WT + quizAvg * QUIZ_WT + projAvg * PROJ_WT + examAvg
  * EXAM_WT + finalAvg * FINAL_EXAM_WT;
  ```
  If there are no grades in the array (i.e., the number of grades is zero), 0.0 should be returned as the average.

Code and Test: You may want to develop the WeightedGradesApp class in parallel with the WeightedGrades class. Alternatively, as you implement your WeightedGrades class, you should compile it and then test it using interactions. For example, as soon as you have implemented and successfully compiled the constructor, you should create an instance of WeightedGrades in interactions.

```
String[] myGrades = {"a90", "a100", "q90", "q80", "p100", "e87.5", "f90"};
WeightedGrades wg = new WeightedGrades("your name", 7, myGrades);
```

Remember that when you have an instance on the workbench, you can unfold it to see its values or you can open a canvas window and drag items from the Workbench tab (or debug tab) onto the canvas. After you have implemented and compiled one or more methods, create a WeightedGrades object and invoke each of your methods on the object to make sure the method is working as intended. For example assuming the object above:

```
java.util.Arrays.toString(wg.gradesByCategory('q'))
[90.0, 80.0]
```

- **WeightedGradesApp.java**

Requirements: Create a WeightedGradesApp class with a main method that gets a file name as a run argument. To use run arguments in jGRASP, click the Build menu, then check "Run Arguments". The Run Arguments text box should open above your code. The main method reads in the file and creates a WeightedGrades object. The program then presents a menu to the user with five options, each of which is implemented: (1) print report, (2) print category, (3) add a new grade, (4) delete a grade, or (5) quit the program.

Design: If no file name is provided as a run argument, an appropriate message should be printed and the program should end (e.g., you can use a *return* statement with no value to exit the main method).

Line #	Program output

1
2
	`File name was expected as a run argument.`
	`Program ending.`

One example input file is provided (*student1.dat*), but you may want to create additional input files perhaps with a lesser or greater number of grades for testing. After reading in the input file, the main method should print an appropriate success message followed by a list of options with the action code and a short description followed by a line with just the action codes prompting the user to select an action. After the user enters an action code, the action is performed, including output if any. Then the line with just the action codes prompting the user to select an action is printed again to accept the next code. Since the input is read in and the WeightedGrades object is created prior to the menu, the first action a user should normally perform is 'P' to print the report and verify that the input file was read and processed correctly. The other actions can then be used as appropriate. This continues until the user enters 'Q' to quit. Note that your program should accept both uppercase and lowercase action codes.

Below is the output produced after successfully reading the file and creating a WeightedGrades object. It includes the action codes and short descriptions followed by the prompt with the action codes waiting for the user to make a selection.

Line #	Program output
1	`File read in and WeightedGrades object created.`
2	
3	`Weighted Grades App Menu`
4	`P - Print Report`
5	`C - Print Category`
6	`A - Add Grade`
7	`D - Delete Grade`
8	`Q - Quit`
9	
10	`Enter Code [P, C, A, D, or Q]:`

The result of the user selecting 'p' to print a report is shown below. Note that this is produced by printing the WeightedGrades object (which implicitly calls its toString method). This is followed by the prompt with the action codes waiting for the user to make the next selection.

Line #	Program output
	`Enter Code [P, C, A, D, or Q]: p`
	`      Student Name: Pat Smith`
	`      Activities: 100.0 90.0 95.0`
	`      Quizzes: 90.0 80.0 80.0 70.0`
	`      Projects: 95.0 100.0 85.0`
	`      Exams: 87.5 85.0`
	`      Final Exam: 90.0`
	`      Course Average: 88.96`
	`Enter Code [P, C, A, D, or Q]:`

The result of the user selecting ‘a’ to add a new grade “a100” followed by a ‘c’ to print the category ‘a’ is shown below. Note that after ‘a’ was entered, the user was prompted for the grade to add which was entered as a String which began with the category. This is followed by the prompt for the next action where ‘c’ was entered, then category ‘a’ to print the activity grades, which now include 100.0, the graded added. Finally, the prompt for the next action is displayed again.

Line #	Program output

```
Enter Code [P, C, A, D, or Q]: a
      Grade to add: a100

Enter Code [P, C, A, D, or Q]: c
      Category: a
      100.0 90.0 95.0 100.0

Enter Code [P, C, A, D, or Q]:
```

Here is an example of deleting a grade by entering ‘d’ followed by the grade “a90” and then entering ‘c’ to print category followed by ‘a’, the category to print. We see that the activity grade 90 has been removed.

Line #	Program output

```
Enter Code [P, C, A, D, or Q]: d
      Grade to delete: a90
      Grade removed.

Enter Code [P, C, A, D, or Q]: c
      Category: a
      100.0 95.0 100.0

Enter Code [P, C, A, D, or Q]:
```

Here is an example of an attempt to delete a final exam grade “f100’ which was not found.

Line #	Program output

```
Enter Code [P, C, A, D, or Q]: d
      Grade to delete: f100
      Grade not found.

Enter Code [P, C, A, D, or Q]:
```

Code and Test: After reading in the file and printing the menu of actions and descriptions, you should have a **do-while loop** that prints the prompt with just the action codes followed by a **switch statement** that performs the indicated action. The do-while loop ends when the user enters ‘q’ to quit. You should be able to test your program by exercising each of the action codes. You may also want to run in debug mode with a breakpoint set at the switch statement. A good strategy would be to open a canvas window then drag your instance of WeightedGrades onto the canvas. You may want switch the viewer to “Basic”. You may also want to open a second viewer on the array inside the WeightedGrades object. You can do this by unfolding the WeightedGrades object in the debug tab so that you can see the grades array and drag the array onto the canvas.

Although your program may not use all of the methods in WeightedGrades and WeightedGradesApp, you should ensure that all of your methods work according to the specification. You can either user interactions in jGRASP or you can write another class and main method to exercise the methods. Web-CAT will test all methods to determine your project grade.