Due: 7A Completed Code **– Thursday, October 31, 2013 by 11:59 p.m.** (graded with your test file)
7B Completed Code – **Monday, November 4, 2013 by 11:59 p.m.** (graded with Web-CAT tests)

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). Note that there is no Skeleton Code assignment this week. 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 grace 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. The grade for the **7A Completed Code** submission (WeightedGrades2.java, WeightedGrades2Test.java) will be determined by the tests that you pass or fail in WeightedGrades2Test and by the level of coverage attained in WeightedGrades2. The **7B Completed Code** will be tested against the test methods in your WeightedGrades2Test.java file as well as usual correctness tests in Web-CAT. The 7B Completed Code assignment will not be posted until after 7A Completed Code assignment is closed at 11:59 p.m., Friday, November 1, 2013 (i.e., after the late submission day).

Files to submit to Web-CAT:

WeightedGrades2.java
WeightedGrades2Test.java

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

Overview: This project will create two classes: (1) WeightedGrades2 is a class representing a weighted grade object and (2) WeightedGrades2Test class which contains one or more test methods for each method in the WeightedGrades2 class. Note that there is no requirement for a class with a main method in this project. **Since you'll be modifying the WeightedGrades class from the previous project, I strongly recommend that you create a new folder for this project with a copy of your WeightedGrades class from the previous project. This class should be renamed WeightedGrades2 in the source code and all references to WeightedGrades should be changed to WeightedGrades2. You should either manually change the file name to WeightedGrades2.java or do a File > Save As WeightedGrades2.java (i.e., the class name in the source must match the file name). You should compile WeightedGrades2.java before you begin making the changes described below.**

New requirements and design specifications are underlined in the descriptions below to help you identify them.

- **WeightedGrades2.java** (a modification of the WeightedGrades class in the previous project)

  Requirements: Create WeightedGrades class that stores the students's name, the number of grades, an array of grades (where each grade has a category code as the first character), and count, which represents the number of WeightedGrades objects that have been created. 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, get count, reset count, and methods to print a report, print a category of grades, add a grade, delete a grade, delete lowest grade, calculate weighted average of the grades, compute the median grade, 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; a *static variable* representing the WeightedGrades count (i.e., the number of WeightedGrades objects that have been created). These variables 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;`
These are the only nine fields that this class should have.

(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. Note that a variable length parameter list should be used for the String array of grades parameter. Your constructor should increment the count for the number of WeightedGrades objects that have been created. Below are examples of how the constructor could be used to create WeightedGrades2 objects. Note that the second one makes use of the variable length parameter list.

```
WeightedGrades2 stu1 = new WeightedGrades2(name, numGrades, grades);

WeightedGrades2 stu2 = new WeightedGrades2(name, numGrades, "a90", "a100",
"q90", "q80", "p100", "e87.5", "f90");
```

(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 WeightedGrades 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.
- `getWeightedGradesCount`: Accepts no parameters and returns an int representing the number of WeightedGrades objects that have been created. This method should be *static*.
- `resetWeightedGradesCount`: Accepts no parameters and resets the WeightedGrades count to zero and returns nothing. This method should be *static*.
- `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. Note that no lines are skipped.

```
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.
- `deleteLowestGrade`: Accepts a char representing the category, attempts to find the lowest grade in the category, then deletes the lowest grade in the that category if found and returns true, otherwise returns false. *Hint: one approach would be to get the double[] for grades in the category, find the lowest grade in the double[]and then call your deleteGrade method with that grade (remember that the parameter for deleteGrade is a String that includes the category as the first character)*.
- `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.

- `computeMedianGrade`: Accepts a char representing the category and returns a double representing the median of recorded values in that category. If there are no grades in the category, 0.0 should be returned as the median. *Hint: Call gradesByCategory to get a double[] of grades in the category. And then sort the double[] using java.util.Arrays.sort(whatever you called the double[]). You should then be able to find the median in the sorted array depending on whether there is an odd or even number of grades. When n is odd, the median will be the middle value; when n is even, the median will be the average of the two "middle" values.*

Code and Test: You may want to develop the WeightedGrades2Test class (described below) in parallel with the WeightedGrades2 class. That is, as you implement your WeightedGrades2 class, you should compile and test it as methods are created by using a combination of interactions and JUnit test methods. For example, as soon you have implemented and successfully compiled the constructor, you should be able to create an instance of WeightedGrades in interactions and/or a JUnit test method using code similar to the following. It may be useful to modify the first line so that it includes all of the grades that were in the student1.dat file.

```
String[] myGrades = {"a90", "a100", "q90", "q80", "p100", "e87.5", "f90"};
WeightedGrades2 wg = new WeightedGrades2("your name", 7, myGrades);
```

Remember that you can set a breakpoint in a JUnit test method and run the test file in Debug mode. Then, when you have an instance in the Debug tab, you can unfold it to see its values or you can open a canvas window and drag items from the Debug tab onto the canvas.

- **WeightedGrades2Test.java**

Requirements: Create a WeightedGrades2Test class that contains a set of *test* methods to test each of the methods in WeightedGrades2.

Design: Typically, in each test method, you will need to create an instance of WeightedGrades2, call the method you are testing, and then make an assertion about the expected results and the actual results. You can think of a test method as simply formalizing or codifying what you have been doing in interactions to make sure a method is working correctly. That is, the sequence of statements that you would enter in interactions to test a method should be entered into a single test method. You should have at least one test method for each method in WeightedGrades2. However, if a method contains code that results in more than one distinct outcome, you need a test method for each outcome. For example, if the method returns boolean, you should have one test method where the expected return is false and another test method that expects the return to be true. Collectively, these test methods are a set of test cases that can be invoked with a single click to test all of the methods in your WeightedGrades2 class.

Code and Test: Since this is the first project requiring you to write JUnit test methods, a good strategy would be to begin by writing test methods for those methods in WeightedGrades2 that you "know" are correct. By doing this, you will be able to concentrate on the getting the test methods correct. That is, if the test method *fails*, it is most likely due to a defect in the test

method itself rather the WeightedGrades2 method being testing. As you become more familiar with the process of writing test methods, you will be better prepared to write the test methods for the new methods WeightedGrades2. Be sure to call the WeightedGrades2 toString method in one of your test cases so that Web-CAT will consider the toString method to be "covered" in its coverage analysis.

Web-CAT

You will submit WeightedGrades2.java and WeightedGrades2Test.java to **7A** and **7B**.

Assignment 7A – No correctness tests will be included in Web-CAT for assignment 7A; the correctness tests are simply the JUnit test methods that we use grade your program (as we have done on previous projects). When you submit to **7A,** Web-CAT will provide feedback on failures of your test methods as well as how well your WeightedGrades2Test methods covered your WeightedGrades2 methods. Web-CAT will use the results of your test methods and their level of coverage to determine your grade.

Assignment 7B – As with previous projects, **7B** will include the correctness tests which we use to test all of your methods. Web-CAT will use the results of these correctness tests as well as the results from your WeightedGrades2Test class to determine your project grade.