Due: **7A – Thursday, October 25, 2012 by 11:59 p.m.** (without correctness tests in Web-CAT)
7B – Monday, October 29 by 11:59 p.m. (with correctness tests in Web-CAT)

Deliverables

The following project files must be submitted to the "graded" assignment by the due date and time specified above (see the Lab Guidelines for information on submitting project files). You should plan to start on the project not later than Wednesday in lab. To ensure that your classes and methods are named correctly, you should make sure that your file is successfully submitted to the "ungraded" assignment in Web-CAT during lab on Wednesday. **Projects sent via e-mail past the deadline at 11:59 PM will not be accepted without a university-approved excuse.**

Files to submit to Web-CAT:

- SportStacker2.java
- SportStacker2Test.java

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

Overview: Sport Stacking consists of an individual or team stacking and un-stacking cups in pre-determined sequences, competing against the clock or another player. In this project, you will create two classes: (1) SportStacker2 which is a class representing a competition sport stacker and (2) SportStacker2Test which is a JUnit class with one or more test methods for each of the methods in SportStacker2. Note that there is no requirement for a class with a main method in this project. **Since you'll be extending the SportStacker class, I strongly recommend that you create a new folder for this project with a copy of your SportStacker class from the previous project. This class should be renamed SportStacker2 and recompiled before you begin making the changes described below (see green highlights).** Here's a video that shows a sport stacker in action: http://www.speedstacks.com/videos/steven_purugganan_cycle_video-4.

- **SportStacker2.java**

  Requirements: Create SportStacker2 class that stores the competitor's name, an array of competition times, the number of times that are stored in the array, and player count (the number of objects of this class that have been created). It also includes methods to get name, times, number of times recorded, and player count, and methods to print a report, add a time, remove the slowest time, find the fastest time, find the slowest time, compute the average time, and compute the median time.

  Design: The SportStacker2 class has fields, a constructor, and methods as outlined below.

  (1) **Fields**: *instance variables* for competitor's name which is a String, an array of competition times of type double, and the number of times that are stored in the array; a *static variable* representing the player count (i.e., the number of SportStacker2 objects that have been created). These variables should be private so that they are not directly accessible from

outside of the class. These are the only fields that this class should have.

(2) **Constructor**: Your SportStacker2 class must contain a constructor that accepts three parameters representing the name, number of times, and an array of times. Note that a variable length parameter list should be used for the array of times parameter. Your constructor should increment the count for the number of SportStacker2 objects that have been created. Below are examples of how the constructor could be used to create a SportStacker objects:

```
SportStacker2 stacker = new SportStacker2 (name, numOfTimes, times);
SportStacker2 stackerTwo = new SportStacker2 (name, 4, 6.5, 7.3, 5.8, 7.0);
```

(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 SportStacker2 are described below.

- `getName`: Accepts no parameters and returns a String representing the name.
- `getTimes`: Accepts no parameters and returns a double array representing times field.
- `getNumTimesRecorded`: Accepts no parameters and returns an int representing the number of times that are stored in the array.
- `getPlayerCount`: Accepts no parameters and returns an int representing the number of SportStacker2 objects that have been created.
- `resetPlayerCount`: Accepts no parameters and resets the player count to zero.
- `toString`: Returns a String containing the information in the SportStacker object as shown below, including newline escape sequences and formatting for the average, median, fastest, and slowest times. Note that the "Player Count" in this example indicates that two other SportStacker2 objects had been created when toString() was called for this object.

```
Sport Stacker Name: Sami
Times: 4.5 6.5 6.0 7.2 6.3
Average Time: 6.10
Median Time: 6.30
Fastest Time: 4.50
Slowest Time: 7.20
Player Count: 3
```

- `addTime`: Accepts a double parameter representing a new time to add to the times 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 times recorded. If this will exceed the capacity of the array, you must call the increaseSize method described below before you add the new time to the times array.
- `increaseSize`: Accepts no parameters and has no return value, increases the capacity of the times 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 size of the array; whereas your method should only increase the size by one.

- `removeSlowestTime`: Accepts no parameters, removes the slowest time (i.e., the largest value) in the times array, and returns a double representing the time that was removed. Note that when an array element is removed, the remaining elements must be shifted to the left as appropriate and the now unoccupied location at the end should be set to zero. [Replaces removeWorstTime() from previous project.]
- `findFastestTime`: Accepts no parameters and returns a double representing the fastest time in the times array.
- `findSlowestTime`: Accepts no parameters and returns a double representing the slowest time in the times array.
- `computeAvgTime`: Accepts no parameters and returns a double representing the average of recorded values in the times array. If there are no times in the array (i.e., the number of recorded times is zero), 0.0 should be returned as the average.
- `computeMedianTime`: Accepts no parameters and returns a double representing the median of recorded values in the times array. If there are no times in the array (i.e., the number of recorded times is zero), 0.0 should be returned as the median. I recommend that you create a copy (e.g., called temp) of the times array using the java.util.Arrays.copyOf method. And then sort the recorded times in temp using java.util.Arrays.sort(temp, 0, n) where n is the number of recorded times. You should then be able to find the median in the sorted temp array depending on whether there is an *odd* or *even* number of recorded times.

Code and Test: As you implement your SportStacker2 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 create an instance of SportStacker in interactions. Remember that when you have an instance on the workbench, you can unfold it to see its values. After you have implemented and compiled one or more of the "getter" methods, you should begin creating test methods in the SportStacker2Test class described below.

- **SportStacker2Test.java**

Requirements: Create a SportStacker2Test class that contains one or more test methods for each method in SportStacker2. These test methods should create SportStacker2 objects and invoke each of the SportStacker2 methods to make sure the methods are working as intended.

Design: Typically, in each test method, you will need to create an instance of SportStacker2, 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. Alternatively, you can think of the test methods as a set of small "main" methods that can be invoked with a single click to test the SportStacker2 methods.

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 SportStacker2 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 SportStacker2 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 SportStacker2.

Web-CAT

Assignment 7A – Although no correctness tests will be included, **7A** will provide feedback on failures as well as how well your SportStacker2Test methods covered your SportStacker2 methods. Web-CAT will use the results of the 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 are intended to test all of your methods. Web-CAT will use the results of the correctness tests as well as the results from your SportStacker2Test class to determine your project grade.