Goals
- to practice reading specifications carefully
- to understand the importance of good data representation
- to understand the need for each method to maintain the class invariant

Overview
This program is a solution to problems 2 and 3 of Chapter 2 of the Main textbook, but if there are any conflicts between the book and this handout, rely on the handout.

You should create a Statistician class that maintains and reports information about a series of real numbers (type double in Java). Methods provided include the length of the series, the sum of all numbers, the mean, and the smallest and largest values. There is also a reset feature so that you can start a fresh series. Note that, unlike the book, we are not keeping the most recent value.

Problem 3 has you provide a union method that adds the contents of this statistician and another statistician and returns a new statistician as if all the values from the two had been put in the new one. This is like the + operator. It is good style to implement a += operator whenever you implement +. We'll call that add where the values of a statistician are added to an existing statistician. Doing that first makes the implementation of union easier and more efficient. Implement those after you have the rest working. That portion of the assignment counts for 15 of the 100 points.

We are providing you with a detailed outline for the statistician class, two console user interface classes to test the statistician, and a JUnit test class to test the statistician automatically. We'll discuss JUnit in another handout. The class is in Statistician.java which has extensive comments describing what each method does. You also received that code as a printed handout. The ConsoleMenuUI class lets you add numbers to the statistician and ask about its state. It is described further below.

The Representation
At first it might seem necessary to keep all the numbers in the series. But it is not. You should not keep all the numbers because you do not know how many will be input. Instead, the instance/member variables will hold just enough information to be able to provide the features of the class.
Suppose the input is the series: 5 15 10. You can easily see the length of the series, the sum, the smallest value, and the largest value. The mean is the sum divided by the number of values. What changes when the value 30 is added to the series? You didn't need the earlier values to provide the needed information.

Therefore, four instance variables are enough to hold the state of the statistician class. Near the beginning of the class file, they are listed as: sumOfValues, count, smallestValue, and largestValue. Think about how those instance variables hold the state of the class. We have included a set of rules, the class invariant, that state what should be true of the instance variables outside the method. For example, if one or more numbers are in the series, smallestValue holds the smallest value in the series.

Understand that class invariant. It should be true after you construct the class object. Every method can rely in its being true. Every method should ensure that it is true when the method ends.

The Tester Program
The ConsoleMenuUI class uses the Statistician class. It is an example of a driver program that provides an easy way to test the class's actions. It allows the user to create a statistician and then take the following actions:

q: quit the testing program
?: print this menu
+: add a value to the statistician
p: print a summary of the statistician values
r: reset the statistician
u: runs the union tests which also test clone and equals

You can test your class with this driver. You do not write it. It runs, but doesn't do anything useful, as is.

Steps for Completing the Assignment
Download the handout from AutoLab – you will get a .zip file with the four files below. Create a new project and add these files to your project (either import them from where you downloaded them or copy them into the project directory and refresh the project).

Statistician.java: The class code with some methods completed and some awaiting your implementation.
ConsoleMenuUI.java: The tester program described above.
UnionDriver.java: Runs the union tests including some checks on clone and equal. Called from ConsoleMenuUI.
StatTest.java: JUnit 4 tests that try to break the code with a wide variety of test cases.
Make all of your changes in Statistician.java. Do not change method names, parameters or return types.

Start implementing the methods incrementally. Implement each of the methods starting with the constructor and reset. Think about what happens in the reset. You can call that from the constructor. Insert is probably the hardest one to do. Maybe do the easy ones (the getters) that return simple values.

You shouldn't do all the functions at once. As you implement each function, think about which instance variables it needs to access or change. Think about the state of the statistician and how it changes. Think of the class invariant as a checklist of what must be done to the instance variables to keep the state of the class correct.

When you have enough implemented to do some tests, run it. It will be boring until you implement the insert method.

Carefully test the class focusing on the functions you have implemented most recently and avoiding those you haven't completed yet.

Implement the other methods. Continue building methods until you have all of them except add and union implemented. Refer back to your class invariant to be sure it is true whenever you finish a member function.

Implement add and union. For the add method, you are implementing the actions for the statistician that is being assigned to (the c in c += a which is called in Java as c.add(a)). If you refer to count, that is c's count. To refer to a's count, use addend.count. Note that you have access to a's instance variables. Carefully consider what happens if a is empty, c is empty, or neither is empty. Test and implement the easy ones first. Carefully consider the properties the sum statistician should have. Consider all the instance variables. Getting it right takes some calculation.

When you have the add working, you can implement the union by making a clone of this statistician, using add on the clone with the other statistician, and returning the combined statistician.

Note that union and add can both throw a NullPointerException. However, that will happen automatically. You do not need to test for or throw that.

Using JUnit
To run the Junit tests, you must add the JUnit 4 library to your project. In eclipse, highlight the project, then right click and choose “build path”, then “add library”. Select JUnit and be sure to add JUnit 4. Now, when you run the JUnit test file, you
should run as “JUnit test” rather than “Java Application”. See the JUnit handout on my Resources page for more details.

Test the program and submit. Now that you have implemented all the member functions, check them out more thoroughly using the various options in ConsoleMenuUI. Understand how to run the JUnit tests and do that. We will discuss that separately. Run several detailed test cases. Submit only the Statistician.java file to AutoLab.

Grading: 90 points of this lab will be based on functionality, as tested via AutoLab. 10 pts will be based on commenting, design and style. Most of the design is done for you, since the interface of each of the individual methods is spelled out. Points can be lost for having more than the specified instance variables, inefficient implementations, etc.