Lab 11 – More on Classes, Objects, and Methods

Introduction

This lab focuses on method overloading and also gives you further practice working with some subtleties of one-dimensional arrays. In it, you will modify the **`Stat`** class created in the previous lab, expanding its functionality. If you recall, that class stored an array of **`double`** values and computed the minimum, maximum, mode, and average of these values. There were also methods for getting and setting the array of values. In particular, there was a method called **`setData`** which used a **`double`** array as its single parameter, and it copied the values from that array to the underlying data array of the **`Stat`** object.

In this lab, you will overload the **`setData`** method (and the class constructors as well), creating versions that use arrays of **`int`**, **`long`**, and **`float`** values as parameters and handles **`null`** parameters properly (i.e. runtime errors should not occur when a parameter is **`null`**). A **`double`** array will still be used internally by the **`Stat`** class to store the values. You will also define an additional set of methods, all called **`append`**, to add new values to the underlying data array.

Important note about this lab

If you completed the previous **`Stat`** lab, then you may reuse your own source code from that lab for this lab. However, if you did ***NOT*** complete the previous **`Stat`** lab, then you will need to do additional work to complete all of the methods in this lab. Source code from the previous **`Stat`** lab will ***NOT*** be provided, and you are ***NOT*** permitted to ask other students for any code associated with this lab or the previous **`Stat`** lab. Also, you are ***NOT*** permitted to give source code to other students for this lab or the previous **`Stat`** lab.

It is important to note that the automatic type conversion of primitive data types that can occur during method calls does not apply to arrays of primitive data types. And so, while it is perfectly possible to define a method having formal parameters that are **`double`** values and then invoke that method using **`int`** values, it is not possible to automatically convert, for instance, an **`int`** array to a **`double`** array. Automatic type conversion can be performed on individual elements of arrays but not on the arrays themselves.

In the lab, you will also modify the code of the **`Stat`** class to allow a data array of 0 elements. It is perfectly possible to create an array of length 0 in Java, and having a variable hold a reference to an array of length 0 is in many ways preferable to simply assigning **`null`** to the variable (it avoids so-called null pointer exceptions, for instance).

Using zero-length arrays requires altering the methods of the **`Stat`** class that were defined in the previous lab. E.g., an empty array has no minimum or maximum value, and so we modify the methods **`min`** and **`max`** to return **`Double.NaN`** (which represents "Not a Number") in those cases. Other adjustments in the same vein are needed throughout the modified program.

The class will also be modified in this lab to handle methods invoked with **`null`** as a parameter. Specifically, you will need to modify several of the methods to check that the value passed to them is not **`null`**. In part, this is done to ensure that calculations are never performed on null values (this makes your program more robust).

As part of the lab, you will also implement methods to compute the variance and standard deviation of the stored data values.

It is important to note that if done somewhat naively, your finished program might contain a significant amount of redundant code. Such redundancy should in general be avoided, as it is ultimately more difficult to maintain, and it increases the chances of an error occurring in your program. Because of this, you should attempt to identify tasks in your program that need to be performed often and then defining a method to perform that task. Once done, other methods can be implemented to make use of it.

Lab Objectives

By the end of the lab, you should be able to create classes utilizing: constructors; access modifiers; instance variables; void methods and methods which return values; accessor and mutator methods (getters and setters) methods; methods calling other methods; method overloading.

You should have also gained further experience working with one-dimensional arrays (including empty arrays) of various data types.

Prerequisites

The lab deals with material from Chapter 5, 6, and 7.

What to Submit

The modified **`Stat.java`** file should be submitted to eLC for grading (you should keep a copy of the original).

Instructions

Use the UML diagram and method descriptions below to create your modified **`Stat`** class. In the diagram, methods not defined in the previous lab are shown in red. Observe that in many cases, previously existing methods require alteration.

Stat
- data: double[]
1. + Stat() **2. + Stat(double[] d)** **3. + Stat(float[] f)** **4. + Stat(int[] i)** **5. + Stat(long[] lo)** **6. + setData(float[] f): void** **7. + setData(double[] d): void** **8. + setData(int[] i): void** **9. + setData(long[] lo): void** **10. + getData(): double[]** **11. + equals(Stat s): boolean** **12. + reset(): void** **13. + append(int[] i): void** **14. + append(float[] f): void** **15. + append(long[] lo): void**

```
16.   + append(double[] d): void
17.   + isEmpty(): boolean
18.   + toString(): String
19.   + min(): double
20.   + max(): double
21.   + average(): double
22.   + mode(): double
23.   - occursNumberOfTimes(double value): int
24.   + variance(): double
25.   + standardDeviation(): double
```

In your code, you are not allowed to use any java.util.Arrays methods or the java stream API (if you are familiar with either of these). Using the java.util.Arrays class or Java stream in any way will result in a grade of zero on this assignment.

Method Descriptions:

- **(1) `Stat()`**—The default constructor for **`Stat`**. It should create a **`double`** array having length 0.
- **(2,3,4,5) `Stat(double[] d)`, `Stat(int[] i)`, `Stat(long[] lo)`, `Stat(float[] f)`** — Constructs a **`Stat`** object using the values held in the parameter array. Invoking the constructor should create a **`double`** array of the same length as the parameter array and holding copies of its values. A reference to this new array should be assigned to **`data`**.
  Note that if the parameter is **`null`**, then an empty array should instead be assigned to **`data`**.
- **(6,7,8,9) `setData(double[] d)`, `setData(int[] i)`, `setData(long[] lo)`, `setData(float[] f)`** — As in the previous lab, these methods are used to set the values of the **`data`** array. Here, if the array used as parameter is not **`null`**, then each of these methods should create a new **`double`** array containing exactly the elements of the parameter array. A reference to this new array is assigned to **`data`**. If the parameter is **`null`**, however, then an empty array should instead be assigned to **`data`**.
- **(10) `getData()`**—This method is left unchanged from the previous lab. It should create a new array containing exactly the values contained in **`data`** and return a reference to this new array. This should happen even if **`data`** is an empty array (has length 0).
- **(11) `equals(Stat s)`** — Unchanged from the previous lab. The method returns **`true`** if the **`data`** arrays of both objects, the calling **`Stat`** object and the passed **`Stat`** object **`s`**, have the same values (and in the same order). Otherwise, it returns **`false`**. If the parameter **`s`** is null, the method returns **`false`**.
- **(12) `reset()`**: This clears the data array. A new empty **`double`** array is created and assigned to **`data`**.
- **(13,14,15,16) `append(double[] d)`, `append(int[] i)`, `append(long[] lo)`, `append(float[] f)`** —These methods should create a new double array containing exactly those elements of **`data`** followed by those of the array passed as parameter. A reference to this array should be assigned to **`data`**. If the parameter is **`null`**, then the method should do nothing (no new array created).
- **(17) `isEmpty()`**— returns the **`boolean`** value **`true`** if the **`data`** object is empty (has length 0). Otherwise, it returns **`false`**.
- **(18) `toString()`**—As in the previous lab, this method returns a **`String`** representation of the data elements, if any, stored in the **`Stat`** object. See the examples below for the correct format.

- **(19) `min()`**—Returns the minimum of the **`data`** array. If the array is empty, then it should return **`Double.NaN`**.
- **(20) `max()`**—Returns the maximum of the **`data`** array. If the array is empty, then it should return **`Double.NaN`**.
- **(21) `average()`**—Returns the average or mean of the values in the **`data`** array. If **`data`** is an empty array, then the method should return **`Double.NaN`**.
- **(22) `mode()`**— The mode is the value that occurs most frequently in a collection of values. In the **`Stat`** class, if one value occurs more frequently in **`data`** than all others, then **`mode()`** should return this value. If there is no such unique value, or if the data array is empty, **`mode()`** should return **`Double.NaN`**.
- **(23) `occursNumberOfTimes(double value)`** — Returns the number of times the **`value`** occurs in the **`data`** array. This is a private helper method for the **`mode()`** method, and its implementation is optional, but excellent practice decomposing the **`mode()`** method.
- **(24) `variance()`**— Returns the variance of the data in the **`data`** array. To compute this, find the difference between the value of each element of the data array and the mean, square this distance, and then sum these squared values. The variance is this sum divided by the number of elements in **`data`**.
  Note that if the **`data`** array is empty, then **`Double.NaN`** should be returned.
- **(25) `standardDeviation():`** Returns the square root of the variance. If the **`data`** array is empty, then **`Double.NaN`** should be returned.

Additional Requirements

These are things that make the graders lives easier, and ultimately, you will see in the real world as well. Remember the teaching staff does not want to touch your code after they gave you requirements; they want to see the perfect results they asked for! Here is a checklist of things you can **lose points** for:

- (-1 point) If the source file(s)/class(es) are named incorrectly (case matters!)
- (-1 point) If your source file(s) have a package declaration at the top
- (-1 point) If any source file you submit is missing your Statement of Academic Honesty at the top of the source file. All submitted source code files must contain your Statement of Academic Honesty at the top of each file.
- (-1 point) If you have more than one instance of Scanner in your program.
- (-1 point) Inconsistent I/O (input/output) that does not match our instructions or examples exactly (unless otherwise stated in an assignment's instructions). Your program's I/O (order, wording, formatting, etc.) must match our examples and instructions.
- (-2 points) If your submission is late
- (-7 points) If the source file(s) are not submitted before the specified deadline's late period ends (48 hours after the deadline).
- (-7 points) If the source file(s) do not compile. All classes must be declared as public and all method signatures must match exactly what is in these instructions. Your program must compile on the command line using the `javac` command and using the required Java compiler for this course. Students are responsible for checking that their source code file(s) compile using the `javac` command for all methods in these instructions.
- If your (-1 point) comments or (-1 point) variables are "lacking"

  - Here, "lacking" means that you or a TA can find **any** lines of code or variables that take more than 10 seconds to understand, and there is no comment, or the variable name does not make sense (variable names like **b**, **bb**, **bbb**, etc. **will almost never be acceptable)**
- (-1 point) Indentation is not consistent throughout your source code
  - Refresh your memory of indentation patterns in chapter 2 in the course textbook
  - Be careful of a combination of tabs and spaces in your files (use one or the other)
- (-7 points) If you use a non-standard Java library or class (StringBuilder class, Arrays class, any class in java.util.stream) or other code specifically disallowed by the lab/project.

If any of the above do not make sense to you, then talk to a TA or your lab instructor.

eLC Submission and Grading

After you have completed and thoroughly tested your program, upload **`Stat.java`** to ***eLC*** to receive credit. Always double check that your submission was successful on ***eLC***!

- Students may earn 3 points for lab attendance and 7 points if their program passes various test cases that we use for grading. **Some test cases may use values taken from the examples in this document and some test cases may not**. You should come up with additional test cases to check that your program is bug free.
- All source code must adhere to good Java programming style standards as discussed in lecture class and in the course textbook readings.
- All instructions and requirements must be followed; otherwise, points maybe deducted.

Examples

Example 1

Example main method:

```
double[] data1 = {};
Stat stat1 = new Stat(data1);
System.out.println("stat1 data = " + stat1.toString());
System.out.println("stat1 min = " + stat1.min());
System.out.println("stat1 max = " + stat1.max());
System.out.println("stat1 average = " + stat1.average());
System.out.println("stat1 mode = " + stat1.mode());
System.out.println("stat1 variance = " + stat1.variance());
System.out.println("stat1 standard deviation = " + stat1.standardDeviation());
System.out.println("stat1 is empty = " + stat1.isEmpty() + "\n");
```

Example output:

```
stat1 data = []
stat1 min = NaN
```

```
stat1 max = NaN
stat1 average = NaN
stat1 mode = NaN
stat1 variance = NaN
stat1 standard deviation = NaN
stat1 is empty = true
```

Example 2

Example main method:

```
double[] data1 = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
Stat stat1 = new Stat(data1);
System.out.println("stat1 data = " + stat1.toString());
System.out.println("stat1 min = " + stat1.min());
System.out.println("stat1 max = " + stat1.max());
System.out.println("stat1 average = " + stat1.average());
System.out.println("stat1 mode = " + stat1.mode());
System.out.println("stat1 variance = " + stat1.variance());
System.out.println("stat1 standard deviation = " + stat1.standardDeviation());
System.out.println("stat1 is empty = " + stat1.isEmpty() + "\n");

stat1.reset();
System.out.println("stat1 data = " + stat1.toString());
System.out.println("stat1 min = " + stat1.min());
System.out.println("stat1 max = " + stat1.max());
System.out.println("stat1 average = " + stat1.average());
System.out.println("stat1 mode = " + stat1.mode());
System.out.println("stat1 variance = " + stat1.variance());
System.out.println("stat1 standard deviation = " + stat1.standardDeviation());
System.out.println("stat1 is empty = " + stat1.isEmpty() + "\n");
```

Example output:

```
stat1 data = [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]
stat1 min = 1.0
stat1 max = 9.0
stat1 average = 5.0
stat1 mode = NaN
stat1 variance = 6.666666666666667
stat1 standard deviation = 2.581988897471611
stat1 is empty = false

stat1 data = []
stat1 min = NaN
stat1 max = NaN
stat1 average = NaN
stat1 mode = NaN
stat1 variance = NaN
stat1 standard deviation = NaN
stat1 is empty = true
```

Example 3

Example main method:

```
float[] data1 = {10.0F,10.0F};
Stat stat1 = new Stat(data1);
System.out.println("stat1 data = " + stat1.toString());
System.out.println("stat1 min = " + stat1.min());
System.out.println("stat1 max = " + stat1.max());
System.out.println("stat1 average = " + stat1.average());
System.out.println("stat1 mode = " + stat1.mode());
System.out.println("stat1 variance = " + stat1.variance());
System.out.println("stat1 standard deviation = " + stat1.standardDeviation() + "\n");

long[] data2 = {80L, 60L};

stat1.append(data2);

System.out.println("stat1 data = " + stat1.toString());
System.out.println("stat1 min = " + stat1.min());
System.out.println("stat1 max = " + stat1.max());
System.out.println("stat1 average = " + stat1.average());
System.out.println("stat1 mode = " + stat1.mode());
System.out.println("stat1 variance = " + stat1.variance());
System.out.println("stat1 standard deviation = " + stat1.standardDeviation() + "\n");
```

Example output:

```
stat1 data = [10.0, 10.0]
stat1 min = 10.0
stat1 max = 10.0
stat1 average = 10.0
stat1 mode = 10.0
stat1 variance = 0.0
stat1 standard deviation = 0.0

stat1 data = [10.0, 10.0, 80.0, 60.0]
stat1 min = 10.0
stat1 max = 80.0
stat1 average = 40.0
stat1 mode = 10.0
stat1 variance = 950.0
stat1 standard deviation = 30.822070014844883
```

Example 4

Example main method:

```
double[] data = {-5.3, 2.5, 88.9, 0, 0.0, 28, 16.5, 88.9, 109.5, -90, 88.9};
Stat stat1 = new Stat();

System.out.println("stat1 data = " +    stat1.toString());
```

```
stat1.append(data);

System.out.println("stat1 has been altered.");

System.out.println("stat1 data = " +   stat1.toString());
System.out.println("stat1 min = " + stat1.min());
System.out.println("stat1 max = " + stat1.max());
System.out.println("stat1 average = " + stat1.average());
System.out.println("stat1 mode = " + stat1.mode());
System.out.println("stat1 variance = " + stat1.variance());
System.out.println("stat1 standard deviation = " + stat1.standardDeviation() + "\n");
```

Example output:

```
stat1 data = []
stat1 has been altered.
stat1 data = [-5.3, 2.5, 88.9, 0.0, 0.0, 28.0, 16.5, 88.9, 109.5, -90.0, 88.9]
stat1 min = -90.0
stat1 max = 109.5
stat1 average = 29.80909090909091
stat1 mode = 88.9
stat1 variance = 3192.369917355372
stat1 standard deviation = 56.50106120556827
```

Example 5

Example main method:

```
double[] data1 = {50.0, 60.0};
float[]  data2 = {70.0F, 80.0F};
int[]    data3 = {90, 100};
long[]    data4 = {100L, 110L};

Stat stat1 = new Stat();
System.out.println("stat1 data = " +   stat1.toString());
stat1.setData(data1);
System.out.println("stat1 data = " +   stat1.toString());
stat1.setData(data2);
System.out.println("stat1 data = " +   stat1.toString());
stat1.setData(data3);
System.out.println("stat1 data = " +   stat1.toString());
stat1.setData(data4);
System.out.println("stat1 data = " +   stat1.toString());
data1 = null;
stat1.setData(data1);
System.out.println("stat1 data = " +   stat1.toString());
```

Example output:

```
stat1 data = []
stat1 data = [50.0, 60.0]
stat1 data = [70.0, 80.0]
stat1 data = [90.0, 100.0]
```

```
stat1 data = [100.0, 110.0]
stat1 data = []
```

Example 6

Example main method:

```
double[] data1 = {50.0, 60.0};
float[]  data2 = {70.0F, 80.0F};
int[]    data3 = {90, 100};
long[]    data4 = {100L, 110L};
Stat stat1 = new Stat();
System.out.println("stat1 data = " +   stat1.toString());
stat1.append(data1);
System.out.println("stat1 data = " +   stat1.toString());
stat1.append(data2);
System.out.println("stat1 data = " +   stat1.toString());
stat1.append(data3);
System.out.println("stat1 data = " +   stat1.toString());
stat1.append(data4);
System.out.println("stat1 data = " +   stat1.toString());
data1 = null;
stat1.append(data1);

System.out.println("stat1 data = " +   stat1.toString());
System.out.println("stat1 min = " + stat1.min());
System.out.println("stat1 max = " + stat1.max());
System.out.println("stat1 average = " + stat1.average());
System.out.println("stat1 mode = " + stat1.mode());
System.out.println("stat1 variance = " + stat1.variance());
System.out.println("stat1 standard deviation = " + stat1.standardDeviation() + "\n");
```

Example output:

```
stat1 data = []
stat1 data = [50.0, 60.0]
stat1 data = [50.0, 60.0, 70.0, 80.0]
stat1 data = [50.0, 60.0, 70.0, 80.0, 90.0, 100.0]
stat1 data = [50.0, 60.0, 70.0, 80.0, 90.0, 100.0, 100.0, 110.0]
stat1 data = [50.0, 60.0, 70.0, 80.0, 90.0, 100.0, 100.0, 110.0]
stat1 min = 50.0
stat1 max = 110.0
stat1 average = 82.5
stat1 mode = 100.0
stat1 variance = 393.75
stat1 standard deviation = 19.84313483298443
```

Example 7

Example main method:

```
double[] data1 = {10,10};
int[] data2 = {10,10};
```

```
Stat stat1 = new Stat(data1);
Stat stat2 = new Stat(data2);
Stat stat3 = new Stat();
Stat stat4 = null;
System.out.println("stat1 data = " +   stat1.toString());
System.out.println("stat2 data = " +   stat2.toString());
System.out.println("stat2 data = " +   stat2.toString());
System.out.println("stat1 equals stat2 = " +  stat1.equals(stat2));
System.out.println("stat1 equals stat3 = " +  stat1.equals(stat3));
System.out.println("stat1 equals stat4 = " +  stat1.equals(stat4));
```

Example output:

```
stat1 data = [10.0, 10.0]
stat2 data = [10.0, 10.0]
stat2 data = [10.0, 10.0]
stat1 equals stat2 = true
stat1 equals stat3 = false
stat1 equals stat4 = false
```

Example 8

Example main method:

```
double[] data1 = {};
double[] data2 = { 25 };
float[] data3 = {};
float[] data4 = { 25 };
int[] data5 = {};
int[] data6 = { 50 };
long[] data7 = {};
long[] data8 = { 12 };

Stat stat1 = new Stat();
stat1.append(data1);
stat1.append(data2);
stat1.append(data3);
stat1.append(data4);
stat1.append(data5);
stat1.append(data6);
stat1.append(data7);
stat1.append(data8);
data1 = null;
stat1.append(data1);

System.out.println("stat1 data = " + stat1.toString());
System.out.println("stat1 min = " + stat1.min());
System.out.println("stat1 max = " + stat1.max());
System.out.println("stat1 average = " + stat1.average());
System.out.println("stat1 mode = " + stat1.mode());
System.out.println("stat1 variance = " + stat1.variance());
System.out.println("stat1 standard deviation = " + stat1.standardDeviation() + "\n");
```

Lab 11 – More on Classes, Objects, and Methods

Example output:

```
stat1 data = [25.0, 25.0, 50.0, 12.0]
stat1 min = 12.0
stat1 max = 50.0
stat1 average = 28.0
stat1 mode = 25.0
stat1 variance = 189.5
stat1 standard deviation = 13.765899897936205
```