# Lab 09 – More on Classes, Objects, and Methods

## Introduction

In this lab you will continue to practice defining your own classes, in this case implementing a class called **`Stat`** that is described using a UML[1] class diagram. A class diagram describes the fields (variables) and methods defined for a class. It also specifies the parameters and return types for methods, and it describes whether particular fields and methods are public or private. Collectively, the fields and methods of a class are called the *members* of the class, and the public members define an interface for the class. It is through this interface that interaction with the class and its instances is made possible.

UML is widely used in software engineering to specify the structure and behavior of software systems. We will only deal with a single UML class diagram in this lab, but UML allows one to visually represent large numbers of classes and other structures as well as how they interrelate (for instance, that one class is a specialization of another). For large software projects, formalisms such as UML are essential if one is to cope with the complexity of the software system.

The **`Stat`** class stores an array of **`double`** values called **`data`**. As indicated by the class diagram, you will need to implement public methods to compute the **`min`**, **`max`**, **`mode`**, and **`average`** of these values. You will also implement methods to "get" and "set" the values held by **`data`**. Importantly, **`data`** should be a private instance variable, meaning that each instance of the **`Stat`** class should have its own copy of the **`data`** variable (each object would store different arrays of **`double`** values).

As indicated in the UML diagram, **`data`** has type **`double[]`**. This is a reference type, meaning that **`data`** will store a reference to the memory location where that array is stored. That is, **`data`** will not store the array itself.

We want to ensure that each distinct instance of the **`Stat`** class uses its own array of doubles, and in order to do that, you should define **`getData()`**so that it creates a copy of the **`data`** array and returns a reference to the copy and <u>not</u> a reference to the original. Otherwise, it would be possible to modify the contents of the **`data`** array without going through the methods of the **`Stat`** class (and this is considered bad design). Similarly, **`setData(double[] d)`** should create a copy of the array **`d`** and assign to **`data`** a reference to the copy.

Review relevant sections of the course textbook for more information on objects and references.

## 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;
- methods calling other methods;
- accessor and mutator methods (getters and setters);
- The **`equals()`** method.

[1] Note that many textbooks state that UML stands for "Universal Modeling Language." However, "*Unified* Modeling Language" is another common name.

## What to Submit

The **Stat.java** file should be submitted for grading.

## Instructions

Use the UML diagram and method descriptions below to create your **Stat** class.

| Stat |
|---|
| - data: double[] |
| + Stat()<br>+ Stat(double[] d)<br>+ getData(): double[]<br>+ setData(double[] d): void<br>+ equals(Stat s): boolean<br>+ toString(): String<br>+ min(): double<br>+ max(): double<br>+ average(): double<br>+ mode(): double |

Observe that **data** should be private; you should define your class so that **data** (and its values) can only be altered through **getData** and **setData**.

**Method Descriptions:**

- **Stat()**—The default constructor for **Stat**. It should create a **double** array having a single element **0.0**.
- **Stat(double[] d)**—Constructs a **Stat** object using the values held in **d**. The constructor should create a **double** array of the same length as **d** and holding copies of the values of **d**. A reference to this new array should be assigned to **data**.
- **getData()**—This is an accessor (*get* or *getter*) method used to retrieve the values of **data**. This method should not return a reference to **data**. Instead, it should create a new array containing exactly the values contained in **data**, and then return a reference to this new array.
- **setData(double[] d)** —This is a mutator (*set* or *setter*) method used to set the values of the data array. The method should create a new array containing exactly the elements of **d** and assign to **data** a reference to this new array (that is, the method should not simply assign **d** to **data**).
- **equals(Stat s)** —Returns the **boolean** value **true** if the **data** objects of both the calling Stat object and the passed Stat object **s** have the same values (and in the same order). Otherwise, it returns **false**.
- **toString()**—Returns a **String** representation of the data elements stored in the **Stat** object. Use the samples listed below as guidelines for formatting.
- **min()**—Returns the minimum of the **data** array.
- **max()**—Returns the maximum of the **data** array.
- **average()**—Returns the average of the **data** array. The average is defined to be a double value that returns the mean value of a given array of numbers.

- **`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. Otherwise, **`mode()`** should return **`Double.NaN`**, indicating that no unique value exists. Note: this method is more difficult to code than the other methods, and it should be completed last.

Once you have created your **`Stat`** class, write a main method within the same class to thoroughly test each method in your class. Also, check that your input and output is consistent with all of the examples found on the next few pages (if it isn't consistent with our examples, then you'll need to fix your source code).

## Submission and Grading

After you have completed and thoroughly tested your program, upload **`Stat.java`** to our course webpage in order to receive credit for the lab (if you have any questions about how to submit this assignment, then ask your lab instructor for help at least 24 hours before this assignment is due). Always double check that your submission was successful on our course website.

The lab will be graded according to the following guidelines.

- A score between 0 and 100 will be assigned.
- If the source file(s) are not submitted before the specified deadline, if they do not compile, or if the submitting student has an unexcused absence for a single lab period devoted to the assignment, then a grade of 0 will be assigned. (Note: students who show up to their first lab period and ***s***how their lab TA that they finished and submitted the week's lab assignment may be excused from their second lab period for that week.)
- If the required comment for all labs describing the program and the academic honesty statement is not included at the top of the file, then 10 points will be deducted. Note: this required comment can be found in Lab 02.
- The program will be evaluated using a separate testing file. Multiple instances of the Stat class will be created and their methods invoked.

## Examples

### Example 1

*Example main method:*

```
double[] data = {-5.3, 2.5, 88.9, 0, 0.0, 28, 16.5, 88.9, 109.5, -90, 88.9};
double[] data2 = {100.34, 50.01, 50.01, -8};
Stat stat1 = new Stat();

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

stat1 = new Stat(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() + "\n");

Stat stat2 = new Stat();
stat2.setData(data2);
Stat stat3 = new Stat(stat1.getData());

System.out.println("stat2 data = " +   stat2.toString());
System.out.println("stat3 data = " +   stat3.toString());
System.out.println();
System.out.println("stat1 is equal to stat2 using \"equals()\"? " +
       stat1.equals(stat2));
System.out.println("stat1 is equal to stat3 using \"equals()\"? " +
       stat1.equals(stat3));
System.out.println("stat1 is equal to stat3 using \"==\"? " + (stat1 == stat3));
```

*Example output:*

```
stat1 data = [0.0]
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

stat2 data = [100.34, 50.01, 50.01, -8.0]
stat3 data = [-5.3, 2.5, 88.9, 0.0, 0.0, 28.0, 16.5, 88.9, 109.5, -90.0, 88.9]

stat1 is equal to stat2 using "equals()"? false
stat1 is equal to stat3 using "equals()"? true
stat1 is equal to stat3 using "=="? false
```

**Example 2**

*Example main method:*

```
double[] data = {10.0, 20.0, 30.0};
Stat stat1 = new Stat(data);

data[0] = 100.0;
data[1] = 200.0;
data[2] = 300.0;

Stat stat2 = new Stat(data);

System.out.println("stat1 data = " + stat1.toString());
System.out.println("stat2 data = " + stat2.toString());
System.out.println("The two arrays should be different");
```

*Example output:*

```
stat1 data = [10.0, 20.0, 30.0]
stat2 data = [100.0, 200.0, 300.0]
The two arrays should be different
```

**Example 3**

*Example main method:*

```
double[] data1 = {10.0, 20.0, 30.0};
Stat stat1 = new Stat(data1);

double[] data2 = stat1.getData();

System.out.println("The arrays are identical: " + (data1 == data2));
```

*Example output:*

```
The arrays are identical: false
```

**Example 4**

*Example main method:*

```
double[] data1 = {10.0, 20.0, 30.0};
Stat stat1 = new Stat();
stat1.setData(data1);
Stat stat2 = new Stat(data1);
double[] data2 = stat1.getData();

System.out.println("The arrays are identical: " + (data1 == data2));
System.out.println("stat2 equals stat1: " +
       stat2.equals(stat1));
System.out.println("stat1 equals stat2: " + stat2.equals(stat1));
```

*Example output:*

```
The arrays are identical: false
stat2 equals stat1: true
stat1 equals stat2: true
```

**Example 5**

*Example main method:*

```java
Stat stat1 = new Stat();
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 data = " + stat1.toString());
```

*Example output:*

```
stat1 data = [0.0]
stat1 min = 0.0
stat1 max = 0.0
stat1 average = 0.0
stat1 mode = 0.0
stat1 data = [0.0]
```

**Example 6**

*Example main method:*

```java
double[] data = {1,2,2,3,3,4};
Stat stat1 = new Stat(data);

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 data = " + stat1.toString());
```

*Example output:*

```
stat1 data = [1.0, 2.0, 2.0, 3.0, 3.0, 4.0]
stat1 min = 1.0
stat1 max = 4.0
stat1 average = 2.5
stat1 mode = NaN
stat1 data = [1.0, 2.0, 2.0, 3.0, 3.0, 4.0]
```