# Design / Implement a new Class [program] that meets requirements of a “Numerical Analyzer” defined below.

|  |
| --- |
| ASSIGNMENT GOALS:   1. Get more experienced with the Java Language and Eclipse operation. 2. Develop a Class ‘from scratch’ driven by:    1. functional requirements    2. desired output format as described below. 3. Begin to leverage “Core Java” Classes for our needs. 4. Provide input data as command-line parameters to a Java Application 5. Detect and deal with error conditions programmatically 6. Use an “instance” of a Class rather than rely on calls to static methods. |

(20 Points)

1. Read JavaTextbook Chapter 3 – pp 111-175 – and complete the following exercises (starts on P177):

**Textbook Questions**

Question 13 at top of Page 177 (Review Questions and Exercises)

13) (5 pts)

|  |
| --- |
| Multipule choice:  C.three |

Question 4 middle of Page 178 (Find the Error)

4) (5 pts)

|  |
| --- |
| switch statement cannot have comparisons, and case must be lower case it the highest score need to be 100 |

Question 3 middle of Page 179 (Algorithm Workbench)

3) (10 pts)

|  |
| --- |
| if(sales <=10000)  commission = sales\* 0.1;  else if(sales >10000 && sales <=15000)  commission = sales \* 0.15;  else if(sales > 15000)  commission = sales \* 0.2; |

(80 Points)

NOTE package up programming assignment Class[es] under: **edu.cuny.csi.csc330.lab2**

1. Programming assignment:

***The Lab 1 development assignment was largely an exercise in completing an already started implementation. The Lab 2 development assignment will call on you to implement a program from scratch.*** This is an exercise in learning more about Java basics, core Java Classes and Class/ method-level **modularity**.

**Implement a ‘runnable’ Class called “NumericAnalyzer”.** Here’s the functional behavior that must be implemented.

* NumericAnalyzer will accept ***a list of 1 or more numbers as command line arguments***.

***NOTE: Don’t prompt the user for input – this is an exercise passing values to your program via the command line! We’re forcing the user to “push” data values to the application rather than “pull” it from the user!***

* **Error checking**:
  + if the user fails to pass in parameters, the program will display an error message (of your choice) and exit early.
  + EXTRA CREDIT (+5) Data Validation – flag non-numeric parameters as “invalid data” and exit early.
* The main() method’s String [] args argument values must be converted and assigned to a numeric/integer array and passed to an instance of NumericAnalyzer. (SEE EXAMPLE main() IMPLEMENTATION BELOW).   
  main() should also be responsible for error checking logic.
* **Your program will display the following information about the numbers provided by the user:**
  1. **The set of numbers provided by the user sorted ascendingly.   
     NOTE: DO NOT implement your own sort algorithm!   
     Part of this assignment is to begin discovering functionality provided by Core Java.   
     Here’s a good place to start:** <https://www.geeksforgeeks.org/arrays-sort-in-java-with-examples/>
  2. **The size of number list (the number of numbers!)**
  3. **The average or mean value.**
  4. **The median - the middle ‘value’ of the set of numbers sorted. Specific Definition:** 
     + **Case 1 – odd set of numbers:**
       - **Median is the middle value.**
     + **Case 2 – even set of numbers:**
       - **Median is the average of the middle 2 values**

|  |  |
| --- | --- |
| **Odd Sized Example** | 2 3 4 8 12 16 32 = **8**   **^** |
| **Even Sized Example** | 1 3 4 12 16 32 = **4 + 12 = 16 / 2 = 8  ^ ^** |

* 1. **The min value.**
  2. **The max value.**
  3. **The sum**
  4. **The range: the difference between the max and min**
  5. **Variance:** Subtract the mean from each value in the list. This gives you a measure of the distance of each value from the mean. Square each of these distances (and they’ll all be positive values), add all of the squares together, and divide that sum by the number of values (that is, take the average of these squared values) .
  6. **Standard Deviation:** is simply the square root of the variance.

**Development / solution guidelines:**

* The output should be neat, formatted and well organized – should be easy on the eyes (see sample output below).  
  Just as with lab 1, follow through the link below to reference “printf()” method usage in Java. <https://www.cs.colostate.edu/~cs160/.Summer16/resources/Java_printf_method_quick_reference.pdf>
* Your code should adhere to naming conventions as discussed in class.
* Your main() method’s actions should be limited to:
  + Gathering command line arguments values …
  + Error handling
  + creating an instance of NumericAnalyzer and invoking its top level method(s) (e.g., “calculate()”, “analyze()”, “display()”… )
* The “real work” should be performed by instance methods. That is, your implementation should embrace modularity:
  + Each mathematical calculation should probably be implemented by a separate method.
  + Yet another method should be responsible for displaying a sorted list of the numbers provided, and displaying all derived values above.

***NOTE: Deriving calculations and displaying output to a Console are separate threads of responsibility, and should therefore be implemented independently of each other.***

* Your implementation should embrace using core Java modules:
  + Use the java.lang.Math Class methods to calculate square roots and perform power-to values.

**So your main() method will include a sequence of instructions similar to this. Use your own ingenuity to complete:**

|  |
| --- |
| **// main() method code fragment example  if(args.length == 0 ) {**  **// Display some error message … (System.err. )**  **System.exit(1);**  **}**  **// create an int array**  **int [] numbers = new int[args.length];**  **for(int i = 0 ; i < args.length ; ++i ) {**  **///////////////////////////**  **///// EXTRA CREDIT FEATURE**  **if(!NumericAnalyzer.isNumeric(args[i])) {**  **System.err.println("Expecting Numeric Data: " + args[i]);**  **System.exit(2); // exit code for invalid data**  **}**  **//////////////////////////**  **numbers[i] = Integer.parseInt(args[i]);**  **}**  **NumericAnalyzer analyzer = new NumericAnalyzer(numbers);**  **analyzer.analyze();**  **analyzer.display();**  **System.exit(0);** |

**Your ‘test cases’ should include testing for (see sample output below –PASTE YOUR Console OUTPUT ON PAGE 5 below):**

1. **No input parameters – error condition.**
2. **1 value**
3. **Multiple values – at least 6+ …**
4. **EXTRA CREDIT: invalid data detection – non numeric parameters**

**SAMPLE OUTPUT FOR TEST CASES 1, 2, 3, 4**

|  |
| --- |
| 1. Pass in 1 or more positive integer number values. |
| **(2)**  256  Count: 1  Min: 256  Max: 256  Range: 0  Sum: 256  Mean: 256  Median: 256  Variance: 0  Standard Deviation: 0 |
| **(3)**  2 4 8 16 32 64 128 256 512  Count: 9  Min: 2  Max: 512  Range: 510  Sum: 1,022  Mean: 113  Median: 32  Variance: 25,941  Standard Deviation: 161 |
| (4) Expecting Numeric Data: abc |

**PASTE YOUR OUTPUT HERE FOR ALL 3 (or 4) TEST CASES**

|  |
| --- |
| **0 args**  system err- there needs to be a positive integer - exiting early. |
| **1 arg**  300  Count: 1  Min: 300  Max: 300  Range: 0  Sum: 300  Mean: 300  Median: 300  Variance: 0  Standard Deviation: 0 |
| **multiple args (at least 6)**  2 4 8 16 32 64 128 256 512  Count: 9  Min: 2  Max: 512  Range: 510  Sum: 1,022  Mean: 113  Median: 32  Variance: 25,941  Standard Deviation: 161 |
| **EXTRA CREDIT – DETECTION OF 1 OR MORE NON-NUMERIC VALUES**  Exception in thread "main" java.lang.NumberFormatException: For input string: "apple"  at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:67)  at java.base/java.lang.Integer.parseInt(Integer.java:668)  at java.base/java.lang.Integer.parseInt(Integer.java:786) |