Criterion A: Planning

Defining the Problem

The **client**, Mr. Anil Yadav, is a math teacher in my school. He teaches math for the IB diploma programme. He currently also teaches the Statistics and Probability option in school.

In class, in order to solve statistics problems, several steps and a lot of processing is required. Data has to be both computed in an advanced manner (complex formulas) and often data ranges have to be kept small in order to both, avoid mistakes in the great amount of processing and also due to the wastage of time created due to finding statistics of large amount of data.

However, in Statistics and Probability, a large amount of data is always encouraged, since they provide reliability and precision, especially in cases of averages and standard deviations. This creates a problem of sorts.

Combined with my own interest in mathematics, with the help of the power of computing, I aim to make such a program, that will calculate statistics (mean, median, mode, variance, population variance, interquartile range) for the three primary forms of data. My computer teacher agreed to be my **advisor**.

Rationale for Proposed Solution

I decided to write a dedicated program using Java as the programming language, because this program, above all else, required a great amount of processing power, as huge amounts of data have to be manipulated. A dedicated program as opposed to a website or spreadsheet software, can take the most amount of advantage of this. Additionally, since this is a math program concerned mainly with data, graphical user interface would not only be unnecessary but also a hindrance, since mathematicians are concerned solely with reliability of data as opposed to its presentation.

Java as an Object-Oriented Programming Language was used specifically because:

1. Code reuse/recycle – In math programs, primarily, complex math computations are often just a build up of smaller principles. By creating methods that can be used universally, a math program can easily deal with much more complex principles, without great complexity or programming time. Additionally, since data can be expressed in three

forms (considered in this program), code reuse between the three will be imperative.
2. It is free to use and has many inbuilt libraries that do not require licensing, but can accomplish great things
3. Can be exported to an executable program that can be run on any machine without additional software.

Stating the Success Criteria

1. Able to accept three forms of statistical data normally used –
   a. Direct data values in a list
   b. Variable values and their corresponding frequencies
   c. Category values and their corresponding frequencies

2. User is allowed to enter as much data as processing power can achieve

3. User does not have to count the amount of data before input (indefinite number of inputs allowed)

4. The program should be able to calculate these values using the information provided for all three forms of data
   a. Mean/Average
   b. Median
   c. Mode
   d. Interquartile Range
   e. Variance
   f. Standard Deviation
   g. Population Variance
   h. Population Standard Deviation

5. Print the statistical measures into file, to be further processed/copied

6. User is allowed to enter data to a number of decimal places (since accuracy is key in this branch of mathematics)

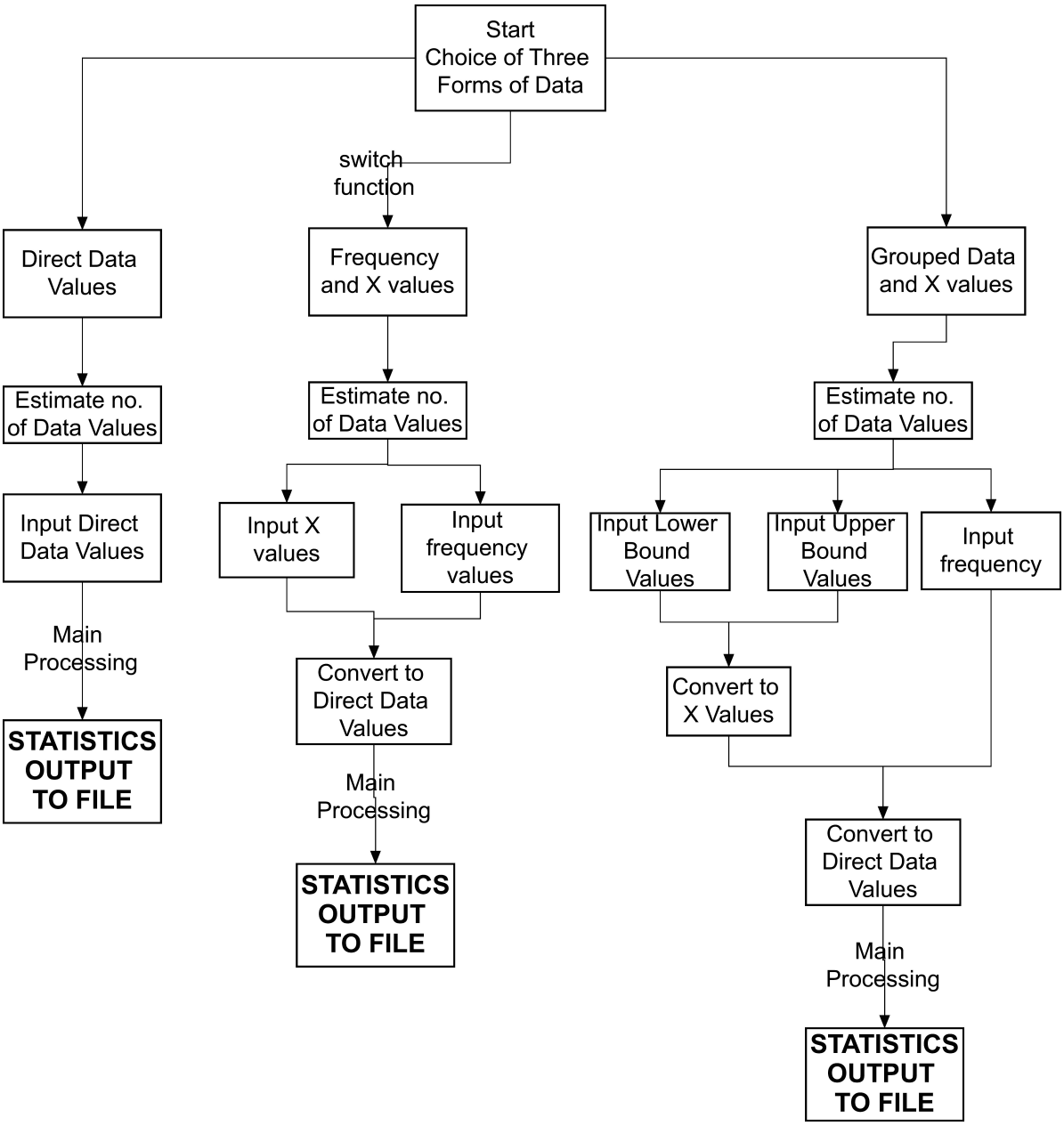
7. On screen instructions as to the input required and the likely output produced, to aid user

8. If mistake in input is made, program should continue running and instead allow the user to input again, instead of crashing or quitting

Word Count: 387

Criterion B: Design

Basic Design of Functionalities with tentative use of programming principles

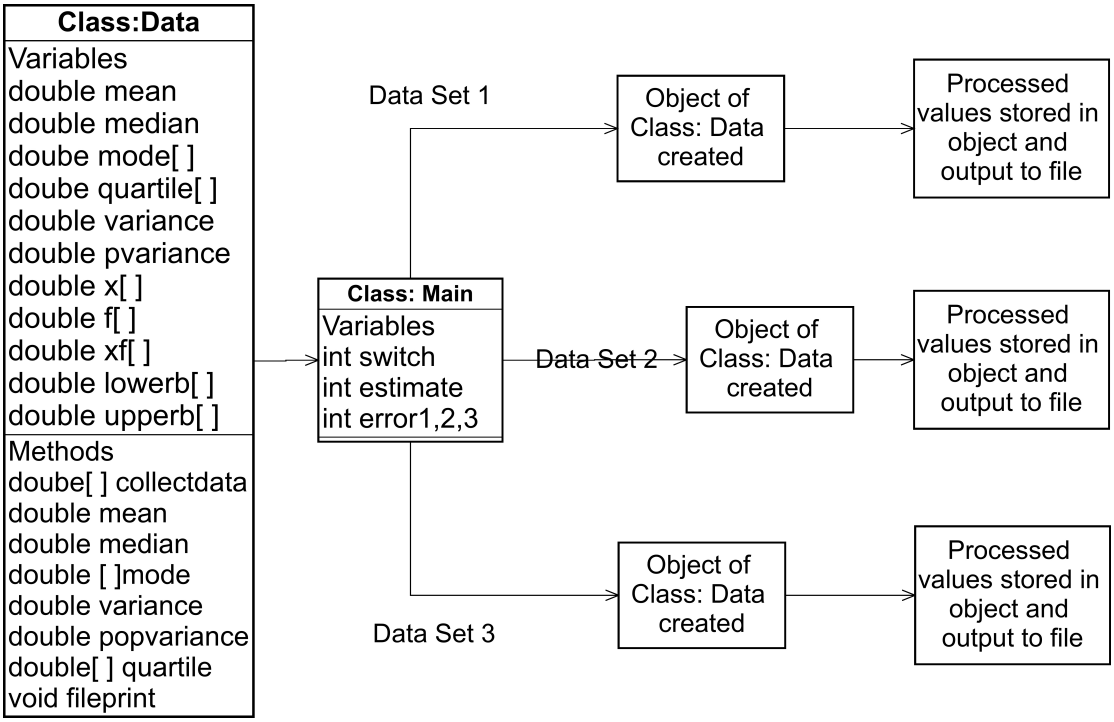


Hence, tasks to be designed can be split into three main factions being:

1. Input of Data
2. Processing of Data
3. Output of Data

Relationships between methods and objects explored in these sub categories.

Diagram showing relationship of classes, variables and methods:



The design and purpose of each method and variable discussed in subcategories below

Input Data

The user will have to input some data and values, and make choices before the processing is carried out and appropriate output is printed.

1. Type of Data to be Input
   This program will accept three different types of data representation. Each type is explained in detail in *Appendix 1.* The processing required to output the statistics of each type of data is different. Hence, a switch case is designed to allow the user to choose the type of data representation required.

2. Estimate number of Data Values
   To initialize array/array list with initial value to save memory. Can be exceeded

3. Actual Statistical Data
   Data to be entered is either direct data values, X values and frequency values or lower bound, upper bound and frequency values. **An object of Class: Data will be created at this stage.** Method of "collectdata()" to be used to take **all**

input of main data. After data entered, array saved in object itself in variables such as: x[ ] , f[ ], xf[ ], lowerb[ ], upperb[ ].

Processing Data

1. All data is converted to direct data values, whichever option the user chooses. The first option does not require further processing in this aspect. In the second option, the two arrays of X values and frequency values converted to direct data values. In the third option, lower bound, upper bound and frequency converted to direct data values.

2. Direct data values passed as a parameter to methods of the object to calculate statistical data. Methods used here are:
    - Mean
    - Median
    - Mode
    - Variance
    - Popvariance
    - Quartile

    Values generated from methods, saved in variables in object itself.

Note: Each object of Class: Data is considered a set of data, capable of calculating and storing all statistics related to the data contained. Several objects can be created for many data sets.

Output Data

Uses the method “fileprint()’

1. Does not have return value. Instead prints the data contained within the object (Entered data and processed statistical data) to file called “Math Data.txt”
2. Can be run separately for different objects, to print data for each object and hence data set.

Data Error

- In the main program, inputs will be checked for error (either IOException or unacceptable data.)
- Varaibles such as error1, error2 will contain either values of 0 or 1. If value is 1, portion of program repeated after error message, so that user can correct mistake
- Will not quit or crash program, but will give user opportunity to correct mistake

Tests Required

Tests will mostly be aimed at varying inputs to get correct output. Hence, the three stages of data input will be tested for accuracy and reliability.

Test Type	Nature of Test	Example
Switch input is acceptable	Should be either 1,2 or 3. If not, error message and allow user to try again	0, -1, 3, 4, “abc”,1.3
Input of ‘estimate’ is acceptable	Should be numerical and greater than 0. If not, ability to try again without running program again	0, -1, 1000, “abc”, 1.5
Input of main data is numerical	Main statistical data has to be numerical (except for flag value) and input with commas in between data	0, 1.34, -1000, 550, “abc”, “end”
Number of Input of X values and frequency are same (2nd Case)	Each X value corresponds with a frequency value. Hence, number of X values and frequency values have to be equal otherwise out of bounds error	Greater number of X values than frequency and vice versa
Number of Input of upper bound, lower bound and frequency are same (3rd Case)	Similar to the 2nd case, each lower bound value corresponds with an upper bound value and frequency. If one has more values than the other, then out of bounds error will result	Different lengths of the 3 arrays. Same length of 3 arrays.
Statistic mathematical calculations are correct	This is most important. The statistics processed using the input value should be numerically/mathematically correct.	Known statistics of data set entered, and output compared with literature values
Add data to file ‘Math Data.txt’	The statistics processed should be appended to the file located in the main Java JDK folder	File created, and program run. Then output compared.
If nonexistent, create file ‘Math Data.txt’	If the file does not exist, instead of ‘file not found’ error, file should be created	Delete created file and run program. After program run, file should be present
Data output in correct format in file	The data written to file should be accurate and precise as processed by the program.	Printed screen values of calculated statistics compared to that written to file

Criterion B: Record of Tasks

Task No.	Planned Action	Planned Outcome	Time Est.	Target Comp. Date	Crit.
1	Initial introduction to project and decision on topic	To decide on a suitable topic that would not only meet the scope of complexity required but also interest me	2 days	29th July 2013	A
2	Final decision on topic	To decide finally the scope and client/advisor to be allocated	10 days	15th August 2013	A
3	Discussion with Math Teacher regarding being the client for this program	Preliminary discussion to understand what is to be expected and what program should achieve	2 days	22nd August 2013	A
4	Interview with Mr. Anil Yadav regarding the program to be made	To document and aid in the success criteria and pinpoint what exactly program should achieve. Most importantly, understanding and defining the problem	1 day	24th August 2013	A
5	Questionnaires from other students regarding program	To note other desirable features that other users of the program would like. Would help in further refining success criteria	4 days	7th September 2013	A
6	Discussion with teacher	To summarize the analysis stage and outline any difficulties that might be met in the development. Software decided. Specify specific Java principles to be used	1 day	17th September 2013	A
7	Basic design of program constructed as instructed by teacher	To have a basic idea of what exactly needs to be designed and developed. Would help in analyzing the problem	2 days	26th September 2013	B
8	Write “Planning” part on report	To benchmark the success criteria that will be referred back to while developing to ensure no deviation	2 days	9th October 2013	
9	Make a chart showing flow of the program	To understand the basic structure that the program would take so that basic methods and classes can be designed and decided	2 days	12th October 2013	B
10	Make a schedule for Development and Testing	This will allow progress and speed to be measured and maintained to ensure enough time to test final product.	1 day	13th October 2013	B

11	Algorithmic design that will allow those features to be completed	This is most important. Hopefully this step will check feasibility of the previously created design; to see if what is designed can indeed be created with current knowledge of Java. If not, alternatives to be found.	10 days	26th October 2013	B
12	Define input/outputs required and testing required	This is the final stage. The algorithmic design will allow evaluating inputs required and outputs generated. With this, testing procedures to be designed	3 days	28th October 2013	B
13	Complete "Design" part of report	With the schedule plan and diagrams created and testing/algorithms designed, report to be updated.	3 days	30th October 2013	
12	Coding the software	This is very important, to not only code a software in Java according to the success criteria, but also to document it. Additionally, algorithmic design has to be converted logically to code. Should not deviate from success criteria. Any additional principles required to be learnt.	15 days	25th November 2013	C
13	Product developed and tested by test criteria defined in "Design"	This will be the alpha testing stage, where I will test the program, note any run-time errors and debug the program. Also refer to success criteria to make sure its covered	10 days	15th December 2013	C,E
14	Solution tested by Client Mr. Anil Yadav	This is the most important as final program reviewed by the client will lead to the main changes needed and whether success criteria is fulfilled as expected	3 days	20th December 2013	C,E
15	Solution tested by students	This will further allow me to make improvements as the programs intended users are also students themselves	5 days	27th December 2013	E
16	Solution improved by feedback	Here, all the feedback compiled from the client, other possible users will be put into affect, changes made, coding done and program debugged and tested again	15 days	15th January 2013	C
17	Submission of program to advisor and client	This will be the final software subject to minor changes, after major changes already made	5 days	22nd January 2013	

	for further testing	based on previous feedback.			
18	Recording Video of Functionality	After the program is finalized, video will be recorded, showing the working of the program, the appropriate inputs and outputs possible and all the features of the program. Will also aid as a guide to using the program	5 days	28th January 2013	D
19	Evaluation of project conducted as further improvements suggested	At this stage, the finalized program will be reviewed against the success criteria, and any weaknesses found. Improvements thought of and suggested based on these weaknesses.	6 days	13th February 2013	E

Criterion C: Development

Techniques used to create this Java Software:

1. Array List and Input
    a. Using Sentinels and Flags
2. Single Dimensional Arrays
3. Flow Control
    a. Switch Statement
    b. Nested Selection Loops (if…else)
4. Error Handling
    a. By using do…while loops
    b. Exception Handling (by using try…catch principle)
5. User Defined Methods
    a. Call by Reference
    b. Call by Value
    c. Create object where data (variables) and methods are stored
6. File Handling
7. Bubble Sorting of Numerical Data
8. Searching

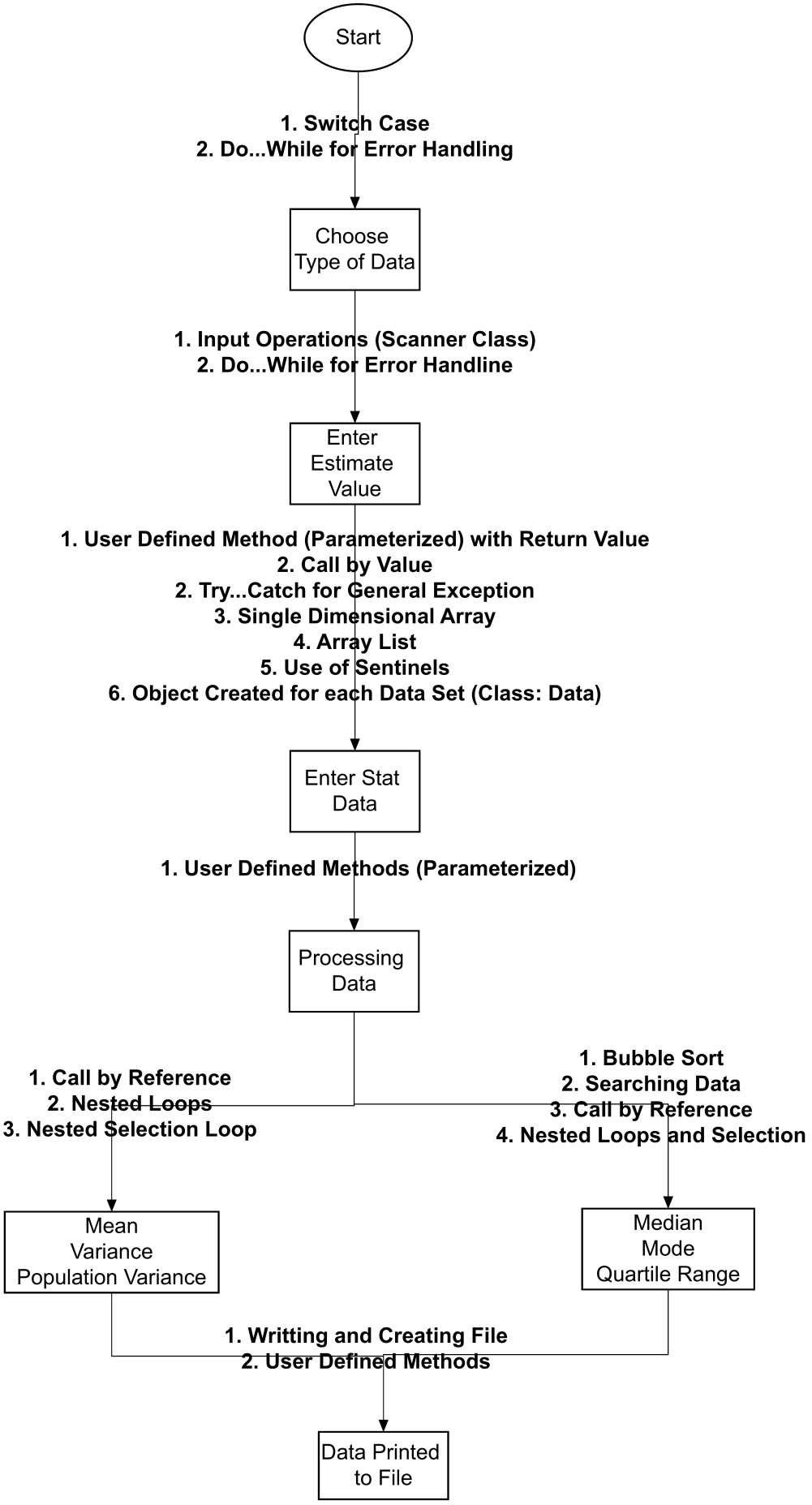
Software Used:

- Java programming language – BlueJ used as Java Editor and Complier
- Word Processing Software for Documentation
- Website Editor

Additional Information in:

1. **Appendix 1** – complete program explained with separate classes. Mathematics used also briefly explained
2. **Appendix 2** – screenshots of program running and output

Diagram of where techniques are used –



1. Array List and Taking Input

Single variable input for example the input for the switch control and the input of the estimate number of data values, was done by the use of the **Scanner** class in Java. The input taken for the 'estimate is shown below. The input value is tested for acceptability.

```
Scanner scan3=new Scanner(System.in);
System.out.println("Please enter an estimate on the number of data values");
System.out.println("NOTE: You can enter as much or as little data as needed.");
System.out.println("This estimate has no affect on that");
estimate=scan3.nextInt();
if (estimate>0)
{
    error3=0;
}
else
{
    System.out.println("Kindly enter a number greater than one");
}
```

After that, the main input has to be received. This is the main data that will be used in all mathematical computation. This data was taken as String, and the user is instructed to append "end" at the end of the string values. This acts as the flag/sentinel.

```
System.out.println("Enter x values, separated by comma. When finished, enter 'end'");
Scanner scan1=new Scanner (System.in);
String main=scan1.nextLine();
```

The principle of array list was used to take input of only statistical data. Hence, to take input of direct data values, X values, lower bound and upper bound values, and frequency values.

```
public double[] collectdata(int est, String s) throws IOException
{
    ArrayList arr = new ArrayList(est);
    StringTokenizer st = new StringTokenizer(s, ",");
    while (st.hasMoreTokens()) {
        String temp=st.nextToken();
        if (temp.equals("end")==false)
        {
            arr.add(temp);
        }

    }
```

The array list is defined above with an initial value of "estimate" collected by the main program. The String of data collected by the main program is used as a parameter for this method. The Array List is defined with an initial value, even

though its not needed, to save memory so that an estimated value of memory is initially used.

The **StringTokenizer** then separates the String, by the commas between data values. And then, the while loop allows each individual data to be added to the Array List. The if loop makes sure, that the last data "end" is not also appended to the Array List.

Hence, Array List created with individual data values only. An Array List was used due to its ability to be flexible as to its length size, unlike an array. The user only has to enter an estimate. If a lot of data is there, then counting the number of data would also be a burden for the user.

2. Single Dimension Array

This is perhaps the most important principle used, since Arrays are the backbone of this program, allowing double type data to be stored and processed, individually and as a group.

The most crucial process after the user enters all the data required into the String which is then converted to Array List, is to further convert this to an Array.

```
double array[]=new double[arr.size()];
for (int i=0; i<arr.size(); i++)
{
    array[i]=Double.parseDouble((String)arr.get(i));
}
return array;
```

A for loop is used until all the elements in the Array List are inserted into the array. The elements in the Array List however, are of type String. Hence, it is parsed to type double.

After this array of data is created, it is return to the main program. Other than arrays of input data (direct values, X and frequency, X and Upper/Lower Bound), these single dimension arrays are processed to calculate each of the statistical measures. For example to calculate mean:

```
public double mean(double a[])
{
    double sum=0;
    for (int i=0; i<a.length; i++)
    {
        sum=sum+a[i];
    }
    double average=sum/a.length;
    return average;
}
```

The single dimension array is processed in that, the sum of the consistent numerical data values is calculated and divided by the array length to get average, which is then returned to the main program.

Apart from these two uses of single dimension arrays, finally, some arrays are also created to represent the statistical data. For example mode:

```
modea=new double[counter];
for (int i=0; i<counter; i++)
{
    if (fa[i]==max)
    {
        modea[i]=xvara[i];
    }
}
return modea;
```

There can be more than one mode (no certain number). Hence, the maximum frequency data already calculated before is compared to the array containing frequencies. If the maximum frequency is found, the corresponding X value (contained in sorted array xvara[]) will be stored into the mode array. All X values with the same maximum frequency are considered modes of the data.

An array is also created to store interquartile range, since there are two values in interquartile range (lower quartile and upper quartile) however, a single method can only return one variable.

3. Flow Control

Flow control, controls the path that the program takes based on user input. There are three representations of data that require separate individual programming to calculate statistics. Hence, in the very beginning, the user will be

```
Scanner scan=new Scanner (System.in);
System.out.println("Input opperator according to requirement");
System.out.println("For direct data values - 1. For frequency table - 2. For Grouped Data - 3");
while (error1==1)
{
    switchvar=scan.nextInt();
    if (switchvar<1 || switchvar>3)
    {
        System.out.println("Incorrect Opperator. Please try again!");
    }
    else
    {
        error1=0;
    }
}

switch(switchvar)
{
    case 1:
```

asked to choose one. This is controlled through a switch.

There are three cases, so the switch variable has to have a value between 1 and 3. The instructions are output, and with the help of **Scanner** class, integer input is recorded. This integer is then checked to be between 1 and 3. If acceptable, then program continues to the switch statement where appropriate case is followed.

Flow control is also affected by if…else conditions used. In the example above, unless acceptable data is entered (checked by if condition) then, while loop will continue running, and user will be asked to enter correct data again and again, without the program progressing further.

Another example is that the entire main program is inside a while loop:

```
while (again==1)
{
```

This will allow the user to run the program again and again, with different data sets, without having to quit the program again and again. The user is asked whether he would like to continue using the program, at the end of one cycle of computations:

```
System.out.println("Do you want to perfrom another calculation? Y/N?");
Scanner scan2=new Scanner(System.in);
String redo=scan2.nextLine();
if (redo.equals("Y")==true||redo.equals("y")==true)
{
    again=1 ;
}
else if (redo.equals("Y")==false||redo.equals("y")==false)
{
    again=0;
}
else
{
    System.out.println("Input was wrong/did not follow rules! Program will automaticaly end.");
    again=0;
}
```

If the user enters “Y”, the value of again will be 1 and the program will start again. If the user enters “N” then the program will quit. There is also a clause if the user enters an unacceptable input, then program will quit, hence effectively influencing the flow of the program.

4. Error Handling

Some error handling techniques are already referenced above in the form of if…else loops and nested selection loops.

All errors are applied to input. If the inputs are valid, then the processes should run and output should be printed. Hence, all inputs are encased in two loops. The first loop is the try…catch loop and the second is the do…while loop. ALL inputs use these two simultaneously. The main data input taken is a prime example:

```
do
{
    System.out.println("Enter x values, separated by comma. When finished, enter 'end'");
    Scanner scan1=new Scanner (System.in);
    String main=scan1.nextLine();
    try
    {
        obj.x=obj.collectdata(estimate,main);
        error2=0;
    }
    catch (Exception e)
    {
        System.out.println("Some data entered other than 'end' was not numerical. Try Again!");
        error2=1;
    }
}
while (error2==1);
```

As seen above, this is the most important test, as the main data entered should be numerical. Hence, first the try…catch loop, will take input and run the method that will process this String, convert it to an Array List and finally an Array (code explained above). However, if the data, other than the end flag value, is not numerical, then, when the method tries to parse the string into double type, an error will be thrown. This will be caught by the loop and the value of error2 is set

to 1. This is where the do…while loop is useful, as for long as the value of error2 is 1, it will repeat the process so that the user can correct his mistake and try again.

Hence, together, the try…catch and do…while loops ensure that correct and valid data is entered and that **there will be no error when processing data**.

Example screenshot of Error Handling:

```
Input opperator according to requirement
For direct data values - 1. For frequency table - 2. For Grouped Data - 3
0
Incorrect Opperator. Please try again!
4
Incorrect Opperator. Please try again!
2
Please enter an estimate on the number of data values
NOTE: You can enter as much or as little data as needed.
This estimate has no affect on that
-1
Kindly enter a number greater than one
Please enter an estimate on the number of data values
NOTE: You can enter as much or as little data as needed.
This estimate has no affect on that
2
Enter x values, separated by comma. When finished, enter 'end'
12,423,abc,12,end
Some data entered other than 'end' was not numerical. Try Again!
Enter x values, separated by comma. When finished, enter 'end'
```

This screenshot shows first unacceptable values are entered for the input required. Then, the program displays an error message but allows the user to try again until he/she gets it right! First, wrong operator value was entered, then an impossible estimate of -1 was entered, then non numerical data “abc” was entered. For each, program responded correctly.

5. User Defined Methods

Each of the statistical measures is in a separate method in the Class: Data. This class also contains variables that can store the result of these calculations. Hence, an object created of this class, can store all the data input, carry out specific methods and finally even store the result of these calculations. The user defined methods are mostly parameterized (both call by value and reference) and have a return type ranging from simple variables to arrays.

Variables in the Object of Class: Data

`double mean;`	store calculated value of mean
`double variance;`	store calculated value of variance
`double popvariance;`	store calculated value of population variance
`double quartile[];`	store calculated array of mode
`double mode[];`	store calculated value of median
`double median;`	store input array of direct values of X
`double x[];`	store calculated array of x when data entered in frequency form
`double xf[];`	store input array of lower bound values
`double lowerb[];`	store input array of upper bound values
`double upperb[];`	store sorted array of x
`double xsort[];`	store input frequency
`double freq[];`	

Hence, the object of this class is extremely powerful, and each object acts as a placeholder for a data set. **Hence, many sets of data can be simultaneously be created and processed and used.**

The methods below range from call by value to call by reference, and have varying parameters.

Methods in Class: Data

```
public double[] collectdata(int est, String s) throws IOException
{
    return array;
}
public double mean(double a[])
{
    return average;
}
public double median(double a[])
{
    return median;
}
public double[] mode(double a[])
{
    return modea;
}
public double variance(double a[])
{
    return variance;
}
public double popvariance(double a[])
{
    return variance;
}
public double[] quartile(double a[])
{
    return quartile;
}
public void fileprint()
{

}
```

Hence, objects of this class is created, when user wants to enter new data:

```
data obj = new data();
```

Then, methods of the object are run, and their results are stored back into variables in the object (using dot operator)

```
obj.mean=obj.mean(obj.x);
obj.median=obj.median(obj.x);
obj.mode=obj.mode(obj.x);
obj.variance=obj.variance(obj.x);
obj.popvariance=obj.popvariance(obj.x);
obj.quartile=obj.quartile(obj.x);

obj.fileprint();
System.out.println("The Data has been appended to file name 'Math Data'");
```

As seen above, the method “mean” is carried out, using a variable of the object “x” and its result is stored back into a variable in the object “mean”. Hence, the object is powerful and self-sustaining. Additionally, since statistics principles are

divided by methods, changes that have to be made can be done separately and independently. Advantages of modular design apply in this case.

6. File Handling

It is important to write the result to a file because math data is often further processed or plotted using spreadsheet software. Output on screen is rarely useful. Hence, FileWriter was used to create and write to file:

```
public void fileprint()
{
    try
    {
        FileWriter file = new FileWriter ("math data.txt",true);
        file.write("\r\n"+"\r\n");
        file.write("Complete Data value for Statistics Below:  ");
        for (int i=0; i<x.length; i++)
        {
            file.write(x[i] + ",   ");
        }
        file.write("\r\n"+"mean  "+mean);
        file.write("\r\n"+"median  "+median);
        file.write("\r\n"+"mode  ");
        for (int i=0; i<mode.length; i++)
        {
            file.write(mode[i] + ",   ");
        }
        file.write("\r\n"+"variance  "+variance);
        file.write("\r\n"+"standard deviation  "+Math.pow(variance,1/2));
        file.write("\r\n"+"population variance  "+popvariance);
        file.write("\r\n"+"population standard deviation  "+Math.pow(popvariance,1/2));
        file.write("\r\n"+"interquartile range  ");
        for (int i=0; i<2; i++)
        {
            file.write(quartile[i]+ ",   ") ;
        }
        file.close();
    }
    catch (FileNotFoundException f)
    {
        try
        {
            FileWriter file = new FileWriter ("math data.txt",true);
```

A read error for the file is caught, when file does not exist, and instead file is created first and process is repeated.

7. Bubble Sort

This sorting technique was used to arrange the data in ascending order. Sorted data in this order is required for median and interquartile range. The bubble sort algorithm was used, where two elements of array are compared and the larger one is displaced to the right.

```
for(int i=0;i<x-1;i++)
{
    for(int j=0;j<(x-1)-i;j++)
    {
        if(a[j]>a[j+1])
        {
            double temp=a[j];
            a[j]=a[j+1];
            a[j+1]=temp;
        }
    }
}
```

8. Searching

Searching was mainly conducted to find the mode of the data. First, frequency array was searched for maximum value. Then it was searched again for where (in which index) did ALL the maximum values fall (hence could be more than one mode). For those index values, the corresponding data in the X value array was retrieved and stored in a array containing the modes.

```
double max=0;
for (int i=0; i<fa.length; i++)
{
    if (fa[i]>max)
    {
        max=fa[i];
    }
}
int counter=0;
for (int i=0; i<fa.length; i++)
{
    if (fa[i]==max)
    {
        counter=counter+1;
    }
}
modea=new double[counter];
for (int i=0; i<counter; i++)
{
    if (fa[i]==max)
    {
        modea[i]=xvara[i];
    }
}
return modea;
```

Criterion E: Evaluation

Meeting the Criteria for Success

1. Able to accept three forms of statistical data normally used –
   a. Direct data values in a list
   b. Variable values and their corresponding frequencies
   c. Category values and their corresponding frequencies

This criterion is met. However, could be made easier by allowing user to input all the values for a single value of x first, rather than having to enter x values first and then all the corresponding frequencies

2. User is allowed to enter as much data as processing power can achieve

This criterion is met, with the help of Array List

3. User does not have to count the amount of data before input (indefinite number of inputs allowed)

This is met. Although an estimate is asked, has no affect on amount of data that can be input.

4. The program should be able to calculate these values using the information provided for all three forms of data
   a. Mean/Average **Calculated**
   b. Median **Calculated**
   c. Mode **Calculated**
   d. Interquartile Range **Calculated**
   e. Variance **Calculated**
   f. Standard Deviation **Calculated**
   g. Population Variance **Calculated**
   h. Population Standard Deviation **Calculated**

Client noted that all values accurate and correct.

5. Print the statistical measures into file, to be further processed/copied

Met. However, after mode, program prints a few 0's, for reasons unknown. Error could not be found.

6. User is allowed to enter data to a number of decimal places (since accuracy is key in this branch of mathematics)

Type double is used for maximum accuracy

7. On screen instructions as to the input required and the likely output produced, to aid user

Clear instructions provided as to the input required

8. If mistake in input is made, program should continue running and instead allow the user to input again, instead of crashing or quitting

Most important factor. Tested to a great extent. (See proof under 'Development')

Recommendations for Future Improvements

There are improvements possible to the program. The most major improvement is to be able to use many data sets simultaneously. Although the facility is provided, as several objects of Class: Data can be created in the main program, and each object will behave independently, the facility is not provided to the user. This will allow a much more advanced program, as several sets of data can not only be processed, but also compared to each other. Hence, the statistics such as the mean can be compared between 2 or more sets of data.

Interquartile data is average or rather estimate and not exact value. When the 25^{th} percentile, 50^{th} percentile and 75^{th} percentile are between two data values, then simple average is taken. Ideally, a graph should be plotted and by connecting points, the specific value of lower quartile, median and upper quartile found. However, making graphs was above the scope of this program.

Finally my choice of Java was good but had some limitations. On one hand, due to the lack of high graphics and other deterrents, most processing power was attributed to the mathematical computations. However, with greater graphic capabilities (something very hard and efficient to do in Java), Graphs, Histograms and other statistics plots like the box and whiskers plot could be created. These graphic representations of statistics would have greatly enhanced the use of the program.

Word Count: 340 words

Appendix 1: Math and Code Explained!

There are three forms of data, and the method to calculate statistical data from each is different.

1. Direct – Data Values

In this type of data, raw data is directly added for example –

547	354	345	134	545	575	174
324	323	976	567	456	245	546
328	453	456	976	567	647	126
213	195	854	857	676	987	464

Hence, this data is taken as input. The method to calculate each of the statistical measures are:

Mean: All the data has to be added and then divided by the total number of data. In the program, this is done by finding sum of all elements in array, and then dividing it by the length of the array.

```
public double mean(double a[])
{
    double sum=0;
    for (int i=0; i<a.length; i++)
    {
        sum=sum+a[i];
    }
    double average=sum/a.length;
    return average;
}
```

For Loop to calculate sum of elements of array

Average found by dividing sum with array length

Median: The median, is the number that appears in the middle of the data, when the data is sorted. Hence, if there are 49 data values, then 25th term will be the middle term. If its an even number like 50, then the two in the middle (25th and 26th term) will be averaged. In the program, this is done by first conducting bubble sort. Then, an if condition differentiates between an even number of data or odd number of data. If even, then middle two are averaged. If odd, middle number is output.

```
public double median(double a[])
{
    int x=a.length-1;
    double median;
    for(int i=0;i<x-1;i++)
    {
        for(int j=0;j<(x-1)-i;j++)
        {
            if(a[j]>a[j+1])
            {
                double temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
        }
    }
    xsort=a;
    if (a.length%2==0)
    {
        median=(a[a.length/2]+a[(a.length/2)+1])/2;
    }
    else
    {
        median=a[(a.length+1)/2];
    }
    return median;
}
```

Bubble sort is used to sort array into ascending order. The first for loop checks the array one time, displacing smaller values to the left. The 2nd for loop does these many times, so that smallest value reaches to the left and so on.

If it is even (modulus operator) then middle two elements averaged. Otherwise, middle element returned

Mode: The mode is the number that appears most frequently. This requires more processing, since first, this data has to be converted into frequency form (see the 2nd data representation below). First the data is sorted (bubble sort) and then, the number of same elements is counted. The highest number *hence greatest frequency) is then output. However, this is done in an array, as there could be more than one mode (If both “5” and “6” appear 3 times)

```
public double[] mode(double a[])
{
    int x=a.length-1;
    double modea[];
    double freq=0;
    for(int i=0;i<x-1;i++)
    {
        for(int j=0;j<(x-1)-i;j++)
        {
            if(a[j]>a[j+1])
            {
                double temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
        }
    }
    ArrayList<Double> xvar = new ArrayList<Double>();
    ArrayList<Double> f = new ArrayList<Double>();
    xvar.add(a[0]);
    for (int i=0;i<a.length-1;i++)
    {
        if (a[i]==a[i+1])
        {
            freq=freq+1;
        }
        else
        {
            f.add(freq);
            xvar.add(a[i+1]);
            freq=0;
        }
    }
    f.add(freq);
    double xvara[]=new double[xvar.size()];
    for (int i=0; i<xvar.size(); i++)
    {
        xvara[i]=xvar.get(i);
    }
    double fa[]=new double[f.size()];
    for (int i=0; i<f.size(); i++)
    {
        fa[i]=f.get(i);
    }
    double max=0;
    for (int i=0; i<fa.length; i++)
    {
        if (fa[i]>max)
        {
            max=fa[i];
        }
    }
    int counter=0;
    for (int i=0; i<fa.length; i++)
    {
        if (fa[i]==max)
        {
            counter=counter+1;
        }
    }
    modea=new double[counter];
    for (int i=0; i<counter; i++)
    {
        if (fa[i]==max)
        {
            modea[i]=xvara[i];
        }
    }
    return modea;
}
```

Bubble sort used to sort data in ascending order (explained in previous ScreenShot.

Converting data to frequency representation. Hence, two new arrays list “xvar” and “f” are created since total number of elements not known

Since sorted array, all same data values are next to each other. Hence, the number of same ‘X’ values is calculated by variable “freq”. When the next element is not the same anymore, then the previous x value and frequency stored, and process continued for next x value.

Array List converted to Array`

Maximum Frequency found in the new frequency array created before.

Array to contain mode created with initialized size of array

‘X’ mode values inserted into mode array (modea)

Variance: Variance is calculated by subtracting each data value, by the mean and then squaring it. This is then added with the result of the other data values and divided by the number of data values. Hence all values in the array are subtracted from mean, squared and then added. Finally it is divided by length of array.

$$\frac{\sum(\boldsymbol{X} - \boldsymbol{mean})^2}{\boldsymbol{n}}$$

```
public double variance(double a[])
{
    double mean=mean(a);
    double sum=0;
    for (int i=0; i<a.length; i++)
    {
        sum=sum+(Math.pow((a[i]-mean),2));
    }
    double variance=sum/a.length;
    return variance;
}
```

X values subtracted by mean and squared

Sum contains sum of the process

Sum divided by number of elements (data values)

Population Variance: Similar to variance except that it is finally divided by 1 less than length of array (1 less than total number of data)

```
public double popvariance(double a[])
{
    double mean=mean(a);
    double sum=0;
    for (int i=0; i<a.length; i++)
    {
        sum=sum+(Math.pow((a[i]-mean),2));
    }
    double variance=sum/(a.length-1);
    return variance;
}
```

Exactly same as Variance, but final sum divided by 1 subtracted from number of elements

Standard Deviation/Population Standard Deviation: This is simply the root of the variance and population variance respectively. Math class used to raise to second power.

Interquartile Range: Similar to Median, however, instead of middle data to be found, the data which stands closest to the $\frac{1}{4}$th and $\frac{3}{4}$th places are output.

```
public double[] quartile(double a[])
{
    int x=a.length-1;
    double lower=0;
    double higher=0;
    for(int i=0;i<x-1;i++)
    {
        for(int j=0;j<(x-1)-i;j++)
        {
            if(a[j]>a[j+1])
            {
                double temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
        }
    }
    if (a.length%4==1)
    {
        lower=a[(a.length-1)/4];
    }
    else if (a.length%4==2)
    {
        lower=(a[(a.length-2)/4]+a[(a.length+2)/4])/2;
    }
    else if (a.length%4==3)
    {
        lower=a[(a.length+1)/4];
    }
    else if (a.length%4==0)
    {
        lower=a[(a.length)/4];
    }
}
```

Array is Bubble Sorted

If the number of elements can be divided by four, then the element at the 1/4th place can be directly output. Otherwise, the index number of element at 1/4th place will be decimal. The index closest to that has to be output.

2. Frequency Table

Example –

X	Frequency
0	2
1	1
2	4
3	3

Though there are ways to calculate each of the statistical measures above for this representation, it is much better instead to convert this representation into the one above, so that methods and codes can be reused. Hence, this data was simple converted to direct data values. The above data means, that the value "0" appears twice, and then the value "1" appears once an so on. SO with the help of two for loops, one which runs the length of the X array, and the second one that runs the number of times of frequency, another array was created which had data in the manner:

0	0	1	2	2
2	2	3	3	3

This array was then processed the same way as the direct data values.

There are two array inputs from the user – Array name: 'x' and 'freq'

```
for (int i=0;i<obj.x.length;i++)
{
    for (int j=0; j<obj.freq[i];j++)
    {
        counter1=counter1+1;
    }
}
obj.xf=new double[counter1];
for (int i=0;i<obj.x.length;i++)
{
    for (int j=0; j<obj.freq[i];j++)
    {
        obj.xf[obj.xf.length-1]=obj.x[i];
    }
}
```

First, counter1 is used with 2 for loops, so that the total elements are calculated. On review, this could have been done by simply finding the sum of elements of frequency.

Each X value appears a few times (hence the frequency array). Therefore, to convert to first type of data representation, the x value is inserted into a new array "xf", but it is entered the actual number of times (frequency). Hence, the first loop does the process for all X values, and the second loop inserts the X value into the new array again and again, according to the frequency.

3. Grouped Data

This data was also converted to the 2nd form first and then finally to the first form.

X Value	Frequency	Mid X Value
0 – 10	5	5
10 – 20	2	15
20 – 30	4	25
30 – 40	3	35

The two arrays contain the lower bound and upper bound values. A third array is created and simple takes the average of the lower and upper bound for each index on the array and hence the "Mid X Value" column is created. This array if X values and the array of frequency, resemble the 2nd form of data. It is then converted to direct data values (first form) in exactly the same way as above.

```
obj.x = new double[obj.upperb.length];

for (int i=0; i<obj.upperb.length; i++)
{
    obj.x[i]=(obj.upperb[i]+obj.lowerb[i])/2;
}
```

A new array is created, where every element is simply the average of lower and upper bound values (also contained in arrays). This becomes the array with X values. Then along with frequency values, it is converted like above (Frequency Table)

Appendix 2: Program in Action!

```
Input opperator according to requirement
For direct data values - 1. For frequency table - 2. For Grouped Data - 3

1
Please enter an estimate on the number of data values
NOTE: You can enter as much or as little data as needed.
This estimate has no affect on that

10
Enter x values, separated by comma. When finished, enter 'end'
12,43,21,432,34,24,54,75,124,end
The Data has been appended to file name 'Math Data'


Do you want to perfrom another calculation? Y/N?


Y
Input opperator according to requirement
For direct data values - 1. For frequency table - 2. For Grouped Data - 3

2
Please enter an estimate on the number of data values
NOTE: You can enter as much or as little data as needed.
This estimate has no affect on that

15
Enter x values, separated by comma. When finished, enter 'end'
1,2,3,4,5,end
Enter frequency values, separated by comma. When finished, enter 'end'
4,6,2,5,3,end
The Data has been appended to file name 'Math Data'


Do you want to perfrom another calculation? Y/N?


N
```

Output text file:

```
Complete Data value for Statistics Below:  12.0,   21.0,   24.0,   34.0,   43.0,   54.0,
75.0,   432.0,   124.0,
mean  91.0
median  54.0
mode  12.0,   21.0,   24.0,   34.0,   43.0,   54.0,   75.0,   432.0,   124.0,
variance  15575.333333333334
standard deviation  1.0
population variance  17522.25
population standard deviation  1.0
interquartile range  24.0,   432.0,
```