Part C: Development

Data Structures

The CustomerData and BenchmarkData objects use Strings to store names and doubles in order to store any numeric values. An instance of the CustomerData class is created with a name, mean, 25th percentile, and 75th percentile (shown below). An instance of the BenchmarkData class is created with a name, 25th percentile, 75th percentile and mean. (shown below).

```
public class CustomerData
{
    String name;
    double count;
    double min;
    double median;
    double max;
    double mean;
    public CustomerData(String nm, double c, double mi, double ma, double me, double m)
    {
        name = nm;
        count = c;
        min = mi;
        median = m;
        max = ma;
        mean = me;

    }
```

```
public class BenchmarkData
{
    String name;
    double oneMedian;
    double median;
    double twoMedian;
    public BenchmarkData(String nm, double om, double m, double tm)
    {
        name = nm;
        oneMedian = om;
        median = m;
        twoMedian = tm;
    }
```

When initialized, these objects are placed into an ArrayList of their respective types. An ArrayList was appropriate for use here due to the fact that the number of CustomerData and BenchmarkData objects is not known until the process of initialization is complete, so objects should be able to be added at any point.

Algorithms

Reading data

The reader methods use the Apache POI's available tools in order to read in data from the given excel file. Apache POI is a java library that allows users to easily work with various Microsoft documents. It is optimized to read and write excel files using its various classes and methods. This library was chosen for the solution because it is much more efficient at reading excel files than writing a program from scratch. The ability to read excel files easily and efficiently makes Apache POI a great choice for this project.

There are two separated reader methods: one for creating instances of the CustomerData class, and one for creating instances of the BenchmarkData. The reader methods implement for and while loops in order to check if there is a next row of data to be read. Research on how to efficiently implement the Apache POI was conducted before developing the program; research consisted of practice examples and viewing other examples of how to use the library.

Taking the CustomerData reader method as an example, the method begins by creating an ArrayList of CustomerData objects that will later be returned. The method then opens a file input stream with the given excel file and gets the number of sheets that will be read in. The method then enters a for loop that runs while the index is less than the number of sheets. At this point, the method creates a row iterator object. This row iterator object enables the methods to both read in data and check if there is more data to be read in. After the creation of the iterator, the method enters a while loop that runs while the row iterator has a next row using the .hasNext() method which returns a boolean value. If the value of .hasNext() is true, the loop will enter a second loop that will run while there is a next cell in the file; the second loop reads the next cell in the file after creating a cell iterator. The loop will first change the cell type to String for the name of the CustomerData instance and create and initialize the name variable. After doing so, it will change the cell type to numeric, so the various values can be created and initialized as doubles. Once the cell iterator does not have a next cell to read, a new CustomerData object is created and added to the ArrayList. Once there are no more rows to read, the method closes the input stream and returns the ArrayList.

```
    ArrayList<CustomerData> newLine = new ArrayList<CustomerData>();
    try
    {
        File file = new File(fileName);
        FileInputStream fis = new FileInputStream(fileName);
        XSSFWorkbook wb = new XSSFWorkbook(fis);
        int numSheets = wb.getNumberOfSheets();
        for(int i = 0; i < numSheets; i++)
        {
            Sheet sheet = wb.getSheetAt(i);
            Iterator<Row> rowIterator = sheet.iterator();
            while(rowIterator.hasNext())
            {
                String name = "";
                double count = 0;
                double min = 0.0;
                double max = 0.0;
                double mean = 0.0;
                double median = 0.0;
                Row row = rowIterator.next();
                Iterator<Cell> cellIterator = row.cellIterator();
                while(cellIterator.hasNext())
                {
                    Cell cell = cellIterator.next();
                    switch(cell.getCellType())
                    {
                    case Cell.CELL_TYPE_STRING:
                        if(name.equals(""))
                            name = cell.getStringCellValue();
                            break;
                    case Cell.CELL_TYPE_NUMERIC:
                        if(count == 0.0)
                            count = cell.getNumericCellValue();
                        else if(min == 0.0)
                            min = cell.getNumericCellValue();
                        else if(max == 0.0)
                            max = cell.getNumericCellValue();
                        else if(mean == 0.0)
                            mean = cell.getNumericCellValue();
                        else if(median == 0.0)
                            median = cell.getNumericCellValue();
                        break;
                    }


                    }
                }
                CustomerData newCD = new CustomerData(name, count, min, max, mean, median);
                newLine.add(newCD);
            }
        }
        fis.close();
    }
    catch(IOException e)
    {
        e.printStackTrace();
    }
    return newLine;
}
```

Graphing Data

The graphing method implements the JFreeChart library of graphing utilities in order to graph data. The JFreeChart library provides many classes and methods that allow users to easily graph functions. The main appeal of the JFreeChart in regards to the solution is the existence of a sample normal distribution function. This makes the process of graphing the BenchmarkData values much simpler because it does not require an equation; it only requires the median/mean and the value of one distribution. These values are either provided or easily calculated which makes JFreeChart a valuable tool for the solution. JFreeChart also has built in markers; markers are useful because they provide a way to clearly graph one value on the graph. The change of color and thickness compared to the graph makes it very easy to see where and what the marker represents.

In order to graph the provided data, the user must first select which CustomerData and Benchmark data objects should be graphed. If the user has selected two excel files and valid objects to be graphed, the JFreeChart library is used to then graph the data. The graph implements the SampleFunction2D class in order to create a normal distribution function dataset with the given data points according to the BenchmarkData object that was selected. After the creation of the dataset, three ValueMarker objects were created in order to graph the 25th percentile and 75th percentile of the BenchmarkData object and the median of the CustomerData object. The two percentile markers are drawn in black; the color of the median of the CustomerData object is dependent on the value relative to the 75th percentile. If the 75th percentile is less than the median of the CustomerData object, the marker will be red. If the

median of the CustomerData object is less than the 75th percentile object, the marker will be blue. This color difference is essential because it makes the data much more understandable for the user.

Tools and Libraries

The following imports were used to graph the data:

```
import org.jfree.chart.ChartPanel;

import java.awt.BasicStroke;
import java.awt.Color;

import javax.swing.JFrame;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.plot.ValueMarker;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.xy.XYDifferenceRenderer;
import org.jfree.data.function.Function2D;
import org.jfree.data.function.NormalDistributionFunction2D;
import org.jfree.data.general.DatasetUtilities;
import org.jfree.data.xy.XYDataset;
```

The following imports were used to create the GUI and read in data:

```
import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.filechooser.FileNameExtensionFilter;

import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
import org.jfree.ui.RefineryUtilities;
```