Joshua Mckensie

Project 2

The first program that I will describe is the plotter, salter and smoother program. This program consists of 3 different classes that all perform different operations with comma separated values. The plotter class is responsible for creating a comma separated list that represents a function of my choice. For my test I decided to create a comma separated list that represents the square root function. To achieve this I added a method that takes a value and performs the square root function, both the input and the output of this function will be added to the comma separated value file. My inputs were zero through fifty so I had fifty-one total inputs and outputs. To successfully create and write to a comma separated value file I used FileWriter and BufferedWriter. FileWriter allows for data to be written to a designated file, the file that I wrote to is called “SqrtFunction.csv”. Additionally, the BufferedWriter is used for efficient data writing to a file. PlotterTester contains the main method that will use the method singleLine() to write to our file.

After the plotter has successfully written to “SqrtFunction.csv”, the salter will be used. This salter will both read from the file “SqrtFunction.csv” and also output a new comma separated file called “Salter.csv”. The purpose for this program is to randomize the y value in the comma separated value file. Using FileWriter and BufferedReader we can read from “SqrtFunction.csv” and grab each line in the file. However, I was only concerned with getting the y value from “SqrtFunction.csv” and each line had both an x and y value. To get only the y value, I was able to split the two up by finding the comma in each line, then I added the x and y values into a temporary array. The y value would always be the second element in the array so I could easily access it now. Once I could access the y value I could add a random number, with a set bound, to a new array called yVals. Finally with FileWriter and BufferedWriter I could write the x and newly generated y values located in the yVals array, to the file “Salter.csv”. This new comma separated value file will slightly resemble the original file “SqrtFunction.csv”.

Once “Salter.csv” is generated, the smoother can be used. The purpose of this program is to attempt to unscramble the randomness from the “Salter.csv” file and generate a third comma separated value file that better resembles the original file “SqrtFunction.csv”. In order to do this I used FileReader and BufferedReader to get the data from “Salter.csv”. Again I needed to get the y values from the comma separated value file, so I used the same method that I used in the salter. Once I had all the y values stored in an array, I was able to begin smoothing out the graph. To do this I would take the average of a range of y values. The range is a variable that can be changed in the program's constructor. For instance, if the range is five then I would take the average of every five y values. These averages were used as the y values for the final comma separated file “Smooth.csv”.

The second program that I will discuss is the poker hand evaluator. For this program I created four classes, Card, Deck, HandEvaluator, and HandEvaluatorTester. The Card class is responsible for creating a Card object, each Card object has both a card number and suite.

The Deck class is responsible for generating a set of 52 cards, these Card objects are held in an ArrayList called deckOfCards, this list symbolizes a deck of playing cards. Additionally, the Deck class is also responsible for shuffling the deck, to do this a random card from deckOfCards is added to a new ArrayList called shuffledDeck. Lastly, Deck has a method that will draw a card from the shuffled deck, this is done by removing the first card from shuffledDeck and returning it.

The HandEvaluator class is responsible for generating a poker hand and evaluating what type of hand occurs. In order to achieve this the HandEvaluator will use methods from Deck to get the deck, shuffle it, and draw cards from the shuffled deck. HandEvaluator has an ArrayList called hand that will store the cards that were drawn. Seven cards in total will be stored in the hand ArrayList. Once a hand is generated, an array called handNumbers will store the frequency that each number occurs in a given hand. Another array called handSuites will work similarly to handNumbers, it will store the frequency of a suite occurring in a hand. Once hand, handNumbers and handSuites are set, the program can now start evaluating the hand. The program will check for the following hands: a pair, two pairs, three of a kind, straight, flush, full house, four of a kind, and a royal flush. For every hand there are 10,000 runs to track its probability of occurring. The HandEvaluatorTester contains the main method and is used to run this simulation. The following are the probabilities that showed to be very consistent: pair occurring 49.22%, two pairs occurring 22.32%, three of a kind occurring 7.39%, straight occurring 4.12%, flush occuring 2.98%, full house occurring 2.58%, four of a kind occurring 0.19%, royal flush occurring 0.0%. I found that for almost all runs of 10,000, a royal flush did not occur, however if the number of runs is increased to something such as 100,000, then I did see that there were occurrences.

In addition to these programs I also have created programs that can compute all the distributions that are covered in chapter 3. These include binomial distribution, geometric distribution, negative binomial distribution, hypergeometric distribution and poisson distribution. For each of these distribution solvers, you can input the values that you want to calculate in the program's constructor. Additionally, for distributions where applicable, you can enter operators such as “=”, “<”, “>”, “<=”, and “>=” in the constructor. These operators will change how the distributions are calculated.

When deciding which distributions should be used to solve a problem there are certain factors that you must consider. For instance, you should use binomial distribution when there are exactly two outcomes in a trial, such as pass or fail. You should use geometric distribution when you are trying to find the number of failures that occur before you experience the first success. For example, if drawing an ace is a success, every card that is drawed and is not an ace will be stored as a failure. Negative binomial distribution is similar to geometric distribution, however you should use negative binomial distribution to get the number of failures until a certain amount of consecutive successes are observed. An example of this could be the amount of times it takes until you flip four heads in a row. You should use hypergeometric distribution when you are trying to find the probability that a success occurs. In addition, the sample size that you are running trials on will not be replaced. The poker hand evaluator seems to be a good example of how hypergeometric distribution can be used. Lastly, you should use the Poisson distribution when an interval of time is given. If you know how often a certain event takes place, you can use this distribution to find the probability of it occurring.