# **Project Documentation PSS-Part-1**

## **Class: mainSystem**

### **Instance Variables**

Scanner scan = new Scanner(System.in);

* A Scanner object used throughout the class to get user input from the console.

### **Method: createFile**

public File createFile(String name) {

name = fixName(name);

File file = new File(name);

return file;

}

* Creates a new File object with the given name.
* Calls fixName to ensure the file has a .csv extension.
* Returns the created File object without actually creating the physical file.

### **Method: getBounds**

public Double[] getBounds() {

Double[] bounds = new Double[3];

System.out.println("Enter the lower bound of the range");

double low = scan.nextDouble();

System.out.println("Enter the upper bound of the range");

double upper = scan.nextDouble();

System.out.println("Enter the increment rate");

double inc = scan.nextDouble();

bounds[0] = low;

bounds[1] = upper;

bounds[2] = inc;

return bounds;

}

* Creates a Double array to store three values: lower bound, upper bound, and increment.
* Prompts the user for each value and stores them in the array.
* Returns the populated array.

### **Method: writeFile**

public BufferedWriter writeFile(String name) {

try {

File file = createFile(name);

FileWriter fileWriter = new FileWriter(file);

BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);

return bufferedWriter;

} catch (IOException e) {

System.out.println("An error occured during the creation of salted file");

return null;

}

}

* Creates a BufferedWriter for writing to a file with the given name.
* Uses createFile to get a File object, then wraps it with FileWriter and BufferedWriter.
* Handles IOException and returns null if file creation fails.
* The caller is responsible for closing the returned BufferedWriter.

### **Method: readFile**

public BufferedReader readFile(String name) {

try {

name = fixName(name);

BufferedReader br = new BufferedReader(new FileReader(name));

return br;

} catch (IOException e) {

System.out.println("File not found");

return null;

}

}

* Creates a BufferedReader for reading from a file with the given name.
* Ensures the filename has a .csv extension using fixName.
* Wraps a FileReader with BufferedReader.
* Handles IOException and returns null if file not found.
* The caller is responsible for closing the returned BufferedReader.

### **Method: askName**

public String askName() {

String fileName = scan.nextLine();

fileName = fixName(fileName);

return fileName;

}

* Reads a filename from the console input.
* Ensures the filename has a .csv extension using fixName.
* Returns the processed filename.

### **Method: fixName**

public String fixName(String name) {

if (!name.contains(".csv")) {

name += ".csv";

}

return name;

}

* Ensures the filename has a .csv extension.
* Checks if the name already contains ".csv" and appends it if not.
* Returns the corrected filename.

### **Method: salting**

public double salting(double salt, double range) {

Random rng = new Random();

int num = rng.nextInt(2);

switch (num) {

case 0:

salt += range;

break;

case 1:

salt -= range;

break;

default:

break;

}

return salt;

}

* Randomly adds or subtracts a range value from the input salt value.
* Generates a random integer (0 or 1) to determine whether to add or subtract.
* Uses a switch statement to perform the operation.
* Returns the modified salt value.

### **Method: smoothy**

public ArrayList<Double> smoothy(ArrayList<Double> sm) {

int count = 0;

double avg;

ArrayList<Double> average = new ArrayList<>();

for (int i = 0; i < sm.size(); i++) {

count = 1;

avg = sm.get(i);

for (int l = Math.max(0, i - 3); l < i; l++) {

if (l <= (sm.size() - 1) && l >= 0) {

avg += sm.get(l);

count++;

}

}

for (int u = i + 1; u <= Math.min(sm.size() - 1, i + 3); u++) {

if (u <= 3 && u <= sm.size() - 1 || u < (sm.size() - 1)) {

avg += sm.get(u);

count++;

}

}

average.add(format(avg/count));

}

return average;

}

* Implements a smoothing algorithm for an ArrayList of Double values.
* For each point, calculates the average of the point and up to 3 values before and after it.
* The count variable keeps track of how many values are included in the average.
* Uses format method to format the calculated average.
* Returns a new ArrayList with the smoothed values.

### **Method: countLine**

public static int countLine(String name) {

try (BufferedReader br = new BufferedReader(new FileReader(name))) {

int lineCount = 0;

while (br.readLine() != null) {

lineCount++;

}

return lineCount;

} catch (IOException e) {

return -1;

}

}

* Counts the number of lines in a file.
* Uses try-with-resources to ensure the BufferedReader is closed.
* Iterates through each line, incrementing a counter.
* Returns -1 if an IOException occurs.

### **Method: format**

public double format(double in) {

DecimalFormat df = new DecimalFormat("0.###");

String formatted = df.format(in);

return Double.parseDouble(formatted);

}

* Formats a double value to have at most 3 decimal places.
* Uses DecimalFormat with pattern "0.###".
* Converts the formatted string back to a double.
* Returns the formatted double value.

### **Method: getArea**

public double getArea(double cur, double upper, boolean isRad) {

double rad;

if (isRad == false) {

rad = Math.toRadians(cur);

} else rad = cur;

double area = Math.cos(rad);

area = format(area);

return area;

}

* Calculates the cosine of a value.
* Converts from degrees to radians if isRad is false.
* Formats the result using the format method.
* Returns the formatted cosine value.
* Note: Despite its name, this method calculates cosine, not area.

## **Class: plotter**

### **Method: main**

public static void main(String[] args) {

// Initialize variables and parameters as needed

mainSystem use = new mainSystem();

String fileName;

boolean isRad = false;

StringBuilder text = new StringBuilder();

double insurance = 0.000001; // Small extra to include upper bound

try {

System.out.println("Cosine function calculation of integers\n");

Scanner in = new Scanner(System.in);

System.out.println("Enter file name to store data");

fileName = use.askName();

System.out.println("Are the values \n(1)radians \n(2)angle?");

int val = in.nextInt();

if (val == 1) {

isRad = true;

}

System.out.println("Enter the lower bound of the range");

double low = in.nextDouble();

System.out.println("Enter the upper bound of the range");

double upper = in.nextDouble();

System.out.println("Enter the increment rate");

double inc = in.nextDouble();

// Loop through the range and build output

while (low <= upper + insurance) {

text.append(use.format(low)).append(", ").append(use.getArea(low, upper, isRad))

.append("\n");

low += inc;

}

BufferedWriter write = use.writeFile(fileName);

write.write("x, cos(x)\n");

// Write the data to designated file

write.write(text.toString());

System.out.println("Data has been stored in " + fileName);

in.close();

write.close();

}

catch (InputMismatchException e) {

System.out.println("You have entered a non-number variable");

}

catch (IOException e) {

System.out.println("An error occured.");

e.printStackTrace();

}

}

* Main method for the plotter class.
* Calculates cosine values over a user-specified range and writes them to a CSV file.
* Prompts the user for:
  + File name to store data
  + Whether values are in radians or degrees
  + Lower bound, upper bound, and increment rate
* Uses a small "insurance" value to ensure the upper bound is included.
* Loops through the range, calculating cosine values and appending to a StringBuilder.
* Writes the header "x, cos(x)" and the data to the file.
* Handles exceptions for input mismatch and IO errors.

## **Class: salter**

### **Method: main**

public static void main(String[] args) {

// Initialize variables and parameters as needed

mainSystem use = new mainSystem();

String name;

String line;

StringBuilder build = new StringBuilder();

String[] list = new String[1];

Scanner in = new Scanner(System.in);

double x, salt;

String nameSalt = name.replace(".csv", "\_salted.csv");

BufferedWriter write = use.writeFile(nameSalt);

write.write(build.toString());

System.out.println("Sucessfully stored salted data in " + nameSalt);

write.close();

br.close();

}

catch (FileNotFoundException e) {

System.out.println("File not found");

} catch (IOException e) {

System.out.println("An error occured during the creation of salted file");

}

in.close();

}

* Main method for the saltBae class.
* "Salts" the y-values in a CSV file by randomly adding or subtracting a user-specified range.
* Prompts the user for:
  + Input file name
  + Salting range (absolute value is used)
* Reads the input CSV file line by line:
  + Preserves the header (first line)
  + For other lines, parses the x and y values
  + Applies "salting" to the y value
  + Formats both values
  + Appends to a StringBuilder
* Creates a new file with "\_salted" appended to the original name.
* Writes the salted data to the new file.
* Handles exceptions for file not found and IO errors.

## **Class: smoother**

### **Method: main**

public static void main(String[] args) {

// Initialize variables and parameters

mainSystem use = new mainSystem();

String fileName, smoothed;

String line;

StringBuilder build = new StringBuilder();

String[] splitter = new String[2];

ArrayList<Double> x = new ArrayList<>();

//Get ready to write data into a file

for (int num = 0; num < x.size(); num++) {

build.append(x.get(num)).append(", ").append(average.get(num)).append("\n");

}

BufferedWriter write = use.writeFile(smoothed);

write.write(build.toString());

System.out.println("Successfully smooth the data to " + smoothed);

scan.close();

write.close();

} catch (FileNotFoundException e) {

System.out.println("File not found " + fileName);

} catch (IOException e) {

System.out.println("An error occured somewhere.");

}

}

* Main method for the smoother class.
* Smooths the y-values in a CSV file using the smoothy method from mainSystem.
* Prompts the user for the input file name.
* Creates an output file name by replacing ".csv" with "\_smoothed.csv".
* Reads the input CSV file line by line:
  + Preserves the header (first line)
  + For other lines, parses the x and y values into ArrayLists
* Prints the sizes of the x and y ArrayLists.
* Applies the smoothy method to the y values.
* Writes the smoothed data (original x values with smoothed y values) to the output file.
* Handles exceptions for file not found and IO errors.