# **Part 3 Java Classes Documentation**

## **Class: main\_system**

### **Package**

package final\_project\_part\_3;

* Defines the package where this class belongs.

### **Imports**

import java.awt.Dimension;

import javax.swing.JFrame;

import org.jfree.chart.ChartFactory;

import org.jfree.chart.ChartPanel;

import org.jfree.chart.JFreeChart;

import org.jfree.data.xy.XYSeries;

import org.jfree.data.xy.XYSeriesCollection;

* Imports necessary libraries for creating graphical user interfaces (GUI) and charts.
* The JFreeChart library is used for creating and displaying charts.

### **Class Declaration**

public class main\_system extends JFrame {

* Defines a class named main\_system that extends JFrame.
* JFrame is a top-level window container that provides a frame for GUI components.

### **Constructor**

public main\_system(String title, XYSeries points) {

super(title);

XYSeriesCollection data\_set = new XYSeriesCollection(points);

JFreeChart chart = ChartFactory.createXYLineChart("Cos(x) graph", "X", "Cos(x)", data\_set);

ChartPanel chartPanel = new ChartPanel(chart);

chartPanel.setPreferredSize(new Dimension(500, 300));

setContentPane(chartPanel);

}

* Constructor that takes a title string and an XYSeries of data points.
* Calls the superclass (JFrame) constructor with the provided title.
* Creates an XYSeriesCollection data set from the provided points.
* Creates a line chart using ChartFactory with title "Cos(x) graph" and axis labels "X" and "Cos(x)".
* Creates a ChartPanel to hold the chart.
* Sets the preferred size of the chart panel to 500x300 pixels.
* Sets the content pane of the JFrame to the chart panel.

## **Class: plot\_graph**

### **Package**

package final\_project\_part\_3;

* Defines the package where this class belongs.

### **Imports**

import java.io.BufferedWriter;

import java.io.IOException;

import java.util.InputMismatchException;

import java.util.Scanner;

import Project\_part1.mainSystem;

import javax.swing.\*;

import org.jfree.data.xy.XYSeries;

* Imports necessary libraries for file I/O, exception handling, user input, and GUI components.
* Imports the mainSystem class from package Project\_part1.
* Imports XYSeries from JFreeChart for storing data points.

### **Class Declaration**

public class plot\_graph {

* Defines a class named plot\_graph.

### **Main Method**

public static void main(String[] args) {

* The main entry point of the program.

### **Variable Initialization**

mainSystem use = new mainSystem();

XYSeries points = new XYSeries("Cos(x) graph");

StringBuilder text = new StringBuilder();

String fileName;

boolean isRad = false;

double insurance = 0.000001; *// To cover the entire range*

* Creates a mainSystem object for utility functions.
* Creates an XYSeries named "Cos(x) graph" to store data points.
* Creates a StringBuilder to accumulate text for file output.
* Declares variables for the file name, a flag for radian/degree mode, and a small insurance value.

### **User Input and Data Generation**

try (Scanner in = new Scanner(System.in)) {

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

*// Ask for file name and create BufferedWriter*

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

fileName = in.nextLine();

BufferedWriter write = use.writeFile(fileName);

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

*// Ask for radians or angle*

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

int val = in.nextInt();

if (val == 1) {

isRad = true;

}

*// Get bounds*

Double[] bounds = use.getBounds();

double low = bounds[0];

double upper = bounds[1];

double inc = bounds[2];

*// Loop through the range and calculate cos(x)*

while (low <= upper + insurance) {

double x = use.format(low);

double y = use.getArea(low, upper, isRad);

text.append(x).append(", ").append(y).append("\n");

points.add(x, y);

low += inc;

}

write.write(text.toString());

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

write.close();

in.close();

*// Write the data to designated file*

new main\_system("Line Graph Example", points);

* Uses a try-with-resources block with a Scanner for user input.
* Prompts the user for a file name and creates a BufferedWriter.
* Writes a header line "x, cos(x)" to the file.
* Asks if values should be in radians (1) or degrees (2).
* Gets bounds (lower, upper, increment) using the getBounds method from mainSystem.
* Loops through the range, calculating cosine values using the getArea method.
* For each value:
  + Formats the x value.
  + Calculates the cosine of x.
  + Appends the values to the text buffer.
  + Adds the (x, y) point to the XYSeries.
* Writes the accumulated data to the file.
* Closes the writer and scanner.
* Creates a new main\_system graph with the points (but doesn't display it).

### **Exception Handling**

} catch (InputMismatchException e) {

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

} catch (IOException e) {

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

e.printStackTrace();

}

* Catches and handles exceptions:
  + InputMismatchException for invalid input.
  + IOException for file operation errors.

### **Creating and Displaying the Graph**

main\_system graph = new main\_system("Cos(x) graph", points);

graph.setSize(800, 600);

graph.setLocationRelativeTo(null);

graph.setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

graph.setVisible(true);

* Creates a new main\_system graph with the title "Cos(x) graph" and the data points.
* Sets the window size to 800x600 pixels.
* Centers the window on the screen (setLocationRelativeTo(null)).
* Sets the default close operation to exit the program when the window is closed.
* Makes the graph visible.

## **Class: salt\_graph**

### **Package and Imports**

package final\_project\_part\_3;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.util.Scanner;

import Project\_part1.mainSystem;

import javax.swing.\*;

import org.jfree.data.xy.XYSeries;

* Similar to plot\_graph, with additional imports for file reading.

### **Main Method and Variable Initialization**

public static void main(String[] args) {

*// Initialize variables and parameters as needed*

mainSystem use = new mainSystem();

XYSeries points = new XYSeries("Cos(x) graph");

String name;

String line;

StringBuilder build = new StringBuilder();

String[] list = new String[1];

Scanner in = new Scanner(System.in);

double x, salt;

* Creates necessary objects and variables.
* points will store the data points for the graph.
* build accumulates text for file output.
* list will hold split CSV values.

### **User Input for File Names and Salting Range**

*// Enter name of file and correct the format*

System.out.println("Enter file name");

name = use.askName();

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

*// Asking for the range of salting*

System.out.println("Enter a number for the salting range:");

double range = Math.abs(in.nextDouble());

* Prompts for and reads the input file name using askName from mainSystem.
* Creates an output file name by replacing ".csv" with "\_salted.csv".
* Prompts for and reads the salting range, ensuring it's positive with Math.abs().

### **Reading Input and Creating Salted Data**

try {

BufferedReader br = use.readFile(name);

BufferedWriter write = use.writeFile(nameSalt);

boolean firstLine = true;

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

list = line.split(",");

if (firstLine) {

build.append(line).append("\n");

firstLine = false;

continue;

}

salt = Double.parseDouble(list[1].trim());

*// Salting the result (y values)*

salt = use.salting(salt, range);

x = Double.parseDouble(list[0].trim());

x = use.format(x);

salt = use.format(salt);

build.append(x).append(", ").append(salt).append("\n");

points.add(x, salt);

}

write.write(build.toString());

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

write.close();

br.close();

* Opens the input file for reading and the output file for writing.
* Reads the input file line by line:
  + Preserves the header (first line).
  + For other lines, parses the x and y values.
  + Applies "salting" to the y value using the salting method from mainSystem.
  + Formats both values using the format method.
  + Appends to the StringBuilder and adds to the XYSeries for graphing.
* Writes the salted data to the output file.
* Closes the reader and writer.

### **Exception Handling and Graph Display**

} 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\_system graph = new main\_system("Cos(x) graph", points);

graph.setSize(800, 600);

graph.setLocationRelativeTo(null);

graph.setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

graph.setVisible(true);

* Handles exceptions for file not found and IO errors.
* Closes the Scanner.
* Creates and displays a graph of the salted data.

## **Class: smooth\_graph**

### **Package and Imports**

package final\_project\_part\_3;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Scanner;

import Project\_part1.mainSystem;

import javax.swing.\*;

import org.jfree.data.xy.XYSeries;

* Similar to the previous classes, with an additional import for ArrayList.

### **Main Method and Variable Initialization**

public static void main(String[] args) {

*// Initialize variables and parameters*

mainSystem use = new mainSystem();

XYSeries points = new XYSeries("Smoothed Cos(x) graph");

String fileName, smoothed;

String line;

int count;

int limit = 3;

double avg;

StringBuilder build = new StringBuilder();

String[] splitter = new String[2];

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

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

Scanner scan = new Scanner(System.in);

* Creates objects and variables needed for the smoothing operation.
* Sets limit to 3, which determines how many neighboring points to consider for smoothing.
* Creates ArrayLists to store the x and y values from the input file.

### **User Input for File Names**

System.out.println("Enter file name: ");

fileName = use.askName();

smoothed = fileName.replace(".csv", "\_smoothed.csv");

* Prompts for and reads the input file name.
* Creates an output file name by replacing ".csv" with "\_smoothed.csv".

### **Reading Input Data**

try {

BufferedReader br = use.readFile(fileName);

BufferedWriter write = use.writeFile(smoothed);

boolean firstLine = true;

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

splitter = line.split(",");

if (firstLine) {

build.append(line).append("\n");

firstLine = false;

continue;

}

x.add(Double.parseDouble(splitter[0].trim()));

y.add(Double.parseDouble(splitter[1].trim()));

}

* Opens the input file for reading and the output file for writing.
* Reads the input file line by line:
  + Preserves the header (first line).
  + Parses x and y values from other lines and adds them to the respective ArrayLists.

### **Smoothing Algorithm**

double[] average = new double[y.size()];

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

count = 1;

avg = y.get(i);

*//This add the values of the left*

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

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

avg += y.get(l);

count++;

}

}

*//this add the values of the right*

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

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

avg += y.get(u);

count++;

}

}

*//sotre the averages*

average[i] = use.format(avg / count);

}

* Creates an array to store the smoothed y values.
* For each point:
  + Starts with the current point's value (avg = y.get(i)).
  + Adds values from up to 3 points to the left (before) the current point.
  + Adds values from up to 3 points to the right (after) the current point.
  + Calculates the average by dividing by the count of points considered.
  + Formats the result using the format method from mainSystem.
* The conditional u <= 3 && u <= y.size() - 1 || u < (y.size() - 1) has potential issues:
  + It may not work as intended for arrays larger than 3 elements.
  + It seems to be checking if the index is within the first 3 elements OR not the last element.

### **Writing Smoothed Data and Creating Graph**

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

double x\_val = x.get(num);

double y\_val = average[num];

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

points.add(x\_val, y\_val);

}

write.write(build.toString());

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

scan.close();

write.close();

* Pairs each x value with its corresponding smoothed y value.
* Appends each pair to the StringBuilder and adds it to the XYSeries for graphing.
* Writes the smoothed data to the output file.
* Closes the scanner and writer.

### **Exception Handling and Graph Display**

} catch (FileNotFoundException e) {

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

} catch (IOException e) {

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

}

main\_system graph = new main\_system("Cos(x) graph", points);

graph.setSize(800, 600);

graph.setLocationRelativeTo(null);

graph.setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

graph.setVisible(true);

* Handles exceptions for file not found and IO errors.
* Creates and displays a graph of the smoothed data.