|  |  |
| --- | --- |
| Student Name:  Student ID:  Unit:  Teacher: | Daniel Field  3107593512  ICTPRG123  Mark O’Reilly |

May 26, 2016

Ian’s Industrial Installation

Documentation

Contents

[Technical Documentation 1](#_Toc452012316)

[IPO Chart 2](#_Toc452012317)

[TOE Chart 3](#_Toc452012318)

[Pseudo-code 3](#_Toc452012319)

[Drawing the Table 3](#_Toc452012320)

[Checking the colour of a single reading 5](#_Toc452012321)

[Painting the components 5](#_Toc452012322)

[Stylised Representation of the Flow of Data 7](#_Toc452012323)

# Technical Documentation

## IPO Chart

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Process** | **Output** | **Storage** |
| Nitrogen Dioxide button | - Calls method “loadData()” and passes in the file string as a parameter  - Calls the method “resizeFrame()” | - Displays a table on the frame with the “Table” class.  - Changes the labels to display the top 3 lines of the file. | - Stores the data in a HashMap<Point, Integer> in RAM. |
| Sulphur Dioxide button | - Calls method “loadData()” and passes in the file string as a parameter  - Calls the method “resizeFrame()” | - Displays a table on the frame with the “Table” class.  - Changes the labels to display the top 3 lines of the file. | - Stores the data in a HashMap<Point, Integer> in RAM. |
| Carbon Monoxide button | - Calls method “loadData()” and passes in the file string as a parameter  - Calls the method “resizeFrame()” | - Displays a table on the frame with the “Table” class.  - Changes the labels to display the top 3 lines of the file. | - Stores the data in a HashMap<Point, Integer> in RAM. |
| Obstructions button | - Calls method “loadData()” and passes in the file string as a parameter  - Calls the method “resizeFrame()” | - Displays a table on the frame with the “Table” class.  - Changes the labels to display the top 3 lines of the file. | - Stores the data in a HashMap<Point, Integer> in RAM. |
| Import button | - Instantiates a FileDialog  - sets the file type to csv.  - waits for user to select file  - checks if the user has selected a file  - if so, call “LoadData”, pass in the file name and call “resizeFrame” | - displays file dialog  - displays table with data from selected file. | - Stores the data in a HashMap<Point, Integer> in RAM. |
| Export button | - Creates an export dialog that can export the file being displayed. | - Displays export dialog (see “ExportDialog” in the Algorithms section) with a textfield to choose the file, and allows you to export to specific file types. | - Stores the export dialog object in RAM. |
| Table cell click | - checks each cell to see if you clicked on it, and then it returns the x and y coordinates and the reading value. | - Displays the selected cell x and y  - Displays the value | - It stores the coordinates and the reading in RAM  - The strings for the labels are stored in RAM |
| Close button | - Calls the method “exit(0)” from the System class. | - Closes the JFrame | N/A |

See “Pseudo-code” for details on how the methods work.

## TOE Chart

|  |  |  |
| --- | --- | --- |
| **Task** | **Object** | **Event** |
| Call “LoadData()” and pass in the NO2 csv file as a parameter. | Form | Open |
| Call “LoadData()” and pass in the NO2 csv file as a parameter. | Nitrogen Dioxide button | Click |
| Call “LoadData()” and pass in the SO2 csv file as a parameter. | Sulphur Dioxide button | Click |
| Call “LoadData()” and pass in the CO csv file as a parameter. | Carbon Monoxide button | Click |
| Call “LoadData()” and pass in the Obstruction csv file as a parameter. | Obstructions button | Click |
| Creates a file dialog to load data into the table | Import button | Click |
| Creates an export dialog to export the data to specified file types | Export button | Click |
| Closes the programme | Close button | Click |
| Works out where on the table you clicked, and then displays the cell x,y, and reading value. | Table | Click |

## Pseudo-code

### Drawing the Table

To draw the table on the JFrame, I used the Graphics class to draw the grid and the data within the grid. Here is the pseudo-code for drawing the table:

Void drawTable(Graphics g, lineColour, backgroundColour, location, numberOfColumns, numberOfRows, cellWidth, cellHeight) {

Bounds = new rectangle(location.x, location.y, (numberOfColumns\*cellWidth), (rows\*cellHeight))

SetColor(backgroundColour)

FillRectangle (bounds.x,bounds.y,bounds.width,bounds.height)

setColor(lineColour)

drawRectangle (bounds.x,bounds.y,bounds.width,bounds.height)

int x = cellWidth

int y = 0

for(I < numberOfColumns) {

drawLine(x+bounds.x, y+bounds.y, x+bounds.x, bounds.height+bounds.y)

x = x+cellwidth

}

Y = cellHeight

for(I < numberOfRows) {

drawLine (bounds.x, y+bounds.y, bounds.width+bounds.x, y+bounds.y)

y = y+cellheight

}

}

To draw the data on the table, I call the “drawReading” method within a nested loop. Here is the pseudo-code for drawing a reading on the table:

Void drawReading(Graphics g, point cell, reading, acceptable, concerning, danger) {

SetColour(checkColour(reading, acceptable, concerning, danger))

fillRect(tablePosition.x+(cell.x\*32)+1, tablePosition.y+(cell.y\*16)+1, cellWidth-1, cellHeight-1)

}

To use this method I create a loop and use the x and y coordinates for the cell.

### Checking the colour of a single reading

To export data and draw readings, there needs to be a way to check whether the reading is within the acceptable, concerning, or dangerous level. Here is the pseudo-code used to return the colour which represents the hazard level:

Colour checkColour(value, acceptable, concerning, danger) {

if (value >= danger) {

return RED;

} else if (value >= concerning) {

return YELLOW;

} else if (value >= acceptable) {

Return GREEN;

} else {

return WHITE;

}

}

### Painting the components

To paint the table and readings, I used the paintComponent method. Here is the pseudo-code:

@Override

Protected void paintComponent(Graphics g) {

super.paintComponent(g)

repaint()

table = new Table(g, GRAY, WHITE, tablePosition, columnCount, rowCount - 3, 32, 16)

for (y < rowCount - 3) {

for (x < columnCount) {

Point p = new Point(x, y);

table.drawReading(g, p, tbl.get(p), acceptable, concerning, danger)

}

}

lblLegend.setBounds(tablePosition.x + (cellWidth \* colCount) + 10, tablePosition.y / 2 - 10, lblLegend.getWidth(), lblLegend.getHeight())

lblRecordedLvl.setBounds(tablePosition.x, tablePosition.y + (cellHeight \* rowCount - 3) - 40, 500, 30)

}

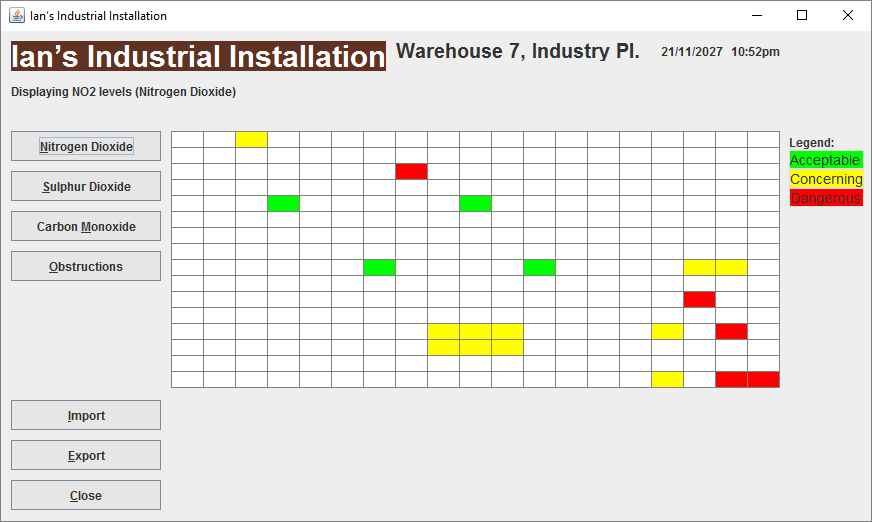
## Stylised Representation of the Flow of Data

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | HashMap<Point, Integer> in RAM: | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| CSV file: |  | 0,0 | 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0,34,0,0,0,0,0,0 |  | 1,0 | 34 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0,7,0,0,0,0,0,0 |  | 2,0 | 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 10,11,4,0,0,0,10,23 |  | 3,0 | 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0,3,9,7,0,1,0,0 |  | 4,0 | 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0,0,0,0,0,0,0,1 |  | 5,0 | 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0,7,0,0,0,0,0,1 |  | 6,0 | 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 10,11,4,0,0,0,10,24 |  | 7,0 | 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | 0,1 | 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | 1,1 | 7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | … | … |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | \*Data in screenshot is different from the data shown in the diagram | | | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | (Export button) | | | | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | 2d Integer array in RAM: | | | | | | | |  |  |  |  |
|  |  |  |  |  |  | 0 | 34 | 0 | 0 | 0 | 0 | 0 | 0 |  |  |  |  |
|  |  |  |  |  |  | 0 | 7 | 0 | 0 | 0 | 0 | 0 | 0 |  |  |  |  |
|  |  |  |  |  |  | 10 | 11 | 4 | 0 | 0 | 0 | 10 | 23 |  |  |  |  |
|  |  |  |  |  |  | 0 | 3 | 9 | 7 | 0 | 1 | 0 | 0 |  |  |  |  |
|  |  |  |  |  |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |  |  |  |  |
|  |  |  |  |  |  | 0 | 7 | 0 | 0 | 0 | 0 | 0 | 1 |  |  |  |  |
|  |  |  |  |  |  | 10 | 11 | 4 | 0 | 0 | 0 | 10 | 24 |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

## Class Diagram



# User Manual



*This is the main frame of the programme.*

**How do I see nitrogen dioxide levels?** To see the nitrogen dioxide levels, simply run the programme and it will display nitrogen dioxide by default. You can also click on the button on the left of the frame labelled ‘Nitrogen Dioxide’. (Hotkey: Alt+N)

**How do I see Sulphur Dioxide levels?** To display the sulphur dioxide levels, click ‘Sulphur Dioxide’. (Hotkey: Alt+S)

**How do I see Carbon Monoxide levels?** To display the carbon monoxide levels, click ‘Carbon Monoxide’. (Hotkey: Alt+M)

**How do I see Obstructions?** To see the obstructions, click on ‘Obstructions’. (Hotkey: Alt+O)

**How do I see the reading values?** To view a reading in more detail, click on the cell of the table, and you will see text appear just below the table displaying the details of that reading.

**How do I import a CSV file?** To import a CSV file, click ‘import’ (Hotkey: Alt+I), and when the dialog pops up, navigate to the file and click ‘Open’. You will see the readings on the table. It will also display the details of when and where the reading took place, and the hazard type.

**How do I export the table?** To export the data from the table, click ‘Export’ (Hotkey: Alt+E), then select the file type(s) to which you would like to export the data. Then type in the directory path into the text field, and click ‘Export’.

**How do I close the programme?** To close the programme, click ‘Close’, or click on the ‘X’ in the top right corner of the frame. (Hotkeys: Alt+C, Alt+F4)