# Programming IT 2018-2019

## Assessed Exercise 3

**Handed out: 20th Nov 2018**
**Deadline: 1730, 18th December 2018**

# Introduction

In science, business and other domains, data is often analysed using charts and graphs. In this exercise, you will implement an interactive application using Swing, that reads an input file containing financial data, and presents the data in that file as an interactive dot plot or 'scatterplot'.

# The data to display

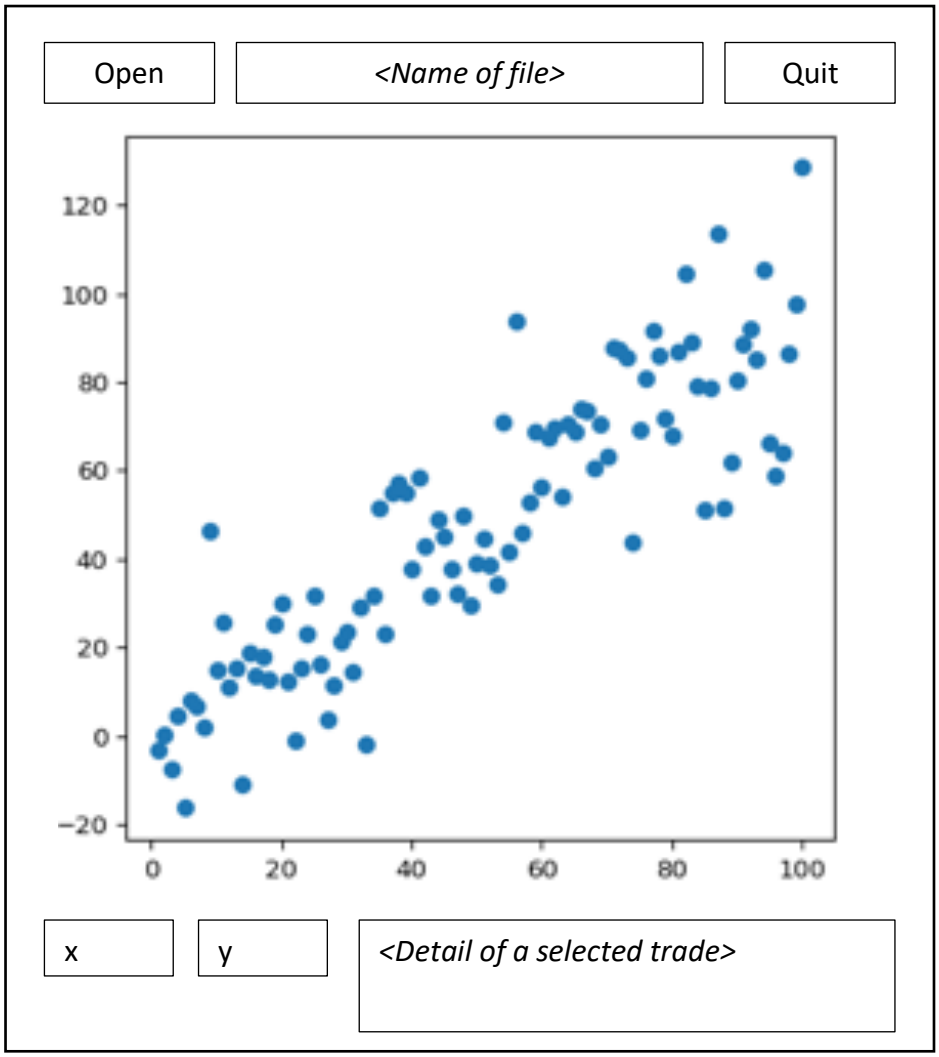
The data is in the widely-used CSV format: comma separated values that you can easily read into a spreadsheet tool (like Excel) or a text editor. The data describes 100 foreign exchange bond trades for a department of the Union Bank of Switzerland, over a period of a few weeks in 1996. The meanings of these columns are as follows:

- YIELD: The percentage return to be made on the trade. High yield bonds are riskier than low yields.
- DAYS_TO_MATURITY: Time until the bond matures - when the owner get their money back.
- AMOUNT_CHF(000): The amount of the trade in Swiss Francs x 1000. (Note that YIELD x AMOUNT_CHF(000) gives the amount of profit that will be made on the trade.)

In the example CSV file on Moodle, the first line of the file contains each of these headers (or 'field names') separated by commas. The lines thereafter contain the trade data, with each line representing one bond trade. Please download this file, store it in the directory for your code for this AE, and have a look at it (e.g. in Excel) to become familiar with its contents.

# Requirements

Your code should create a Swing *JFrame*, and then set up components within it roughly as sketched below, and explained in later points:




1. The *Open* button should trigger a Swing *JFileChooser*, used to obtain the name of an input file. The file to be selected will be something of the form xxxxx.csv. Begin the search for that file from the same directory as the code. The filename should be shown in the text field to the right of the *Open* button. Your code will open the file and read in the first line, to get the field names. Then it should read in and store the bond trades.
2. The *Quit* button should cause the program to exit.
3. The *drawing area* is a Swing component, on which you should use Swing's paint libraries to draw a scatterplot based on the CSV file. Initially, the x-axis should be based on the first column in the file, and the y-axis the second column. Draw and label axis lines, with the lines having regular tick marks and numbers to help the user read the scatterplot. Draw a small circular dot for each bond trade, based on the columns used for x and y axes. Calculate the positions of the dots by scaling (i.e. multiplying) and shifting (i.e. adding to) the 'raw' values in the column, so that the dots range across the area defined by your axis lines.

4. Allow the user to click on a dot in the drawing area, and have the three column values for the corresponding trade then be written out in the text field labelled above as *Detail of a selected trade.* Each time the user clicks on a dot, this field should be updated.
5. Allow the user to change which columns are used for the axes, by using two *JComboBoxes*: one for the x axis, and one for the y. These are labelled as *x* and *y* in the diagram above, but each should normally have its currently selected field name. In this way, for example, a user might then change from a graph of YIELD against DAYS_TO_MATURITY, to one showing DAYS_TO_MATURITY against AMOUNT_CHF(000).

# Some suggestions

1. Make a class for a bond trade.
2. Since you won't know exactly how many trades will be in a CSV file, i.e. how many lines there will be in it, you should choose a way to handle this situation. If using an array of trades, then use the array length doubling trick used in *Scores.java* in the examples of week 9. You might prefer, though, to use a Java Collection such as *ArrayList* instead—as it will extend itself as you add items to it.
3. Use the Model-View-Controller (MVC) design pattern (example code on Moodle), so that the collection of bond trades is stored and managed separately to the code used to handle Swing-based interaction.
4. Break the problem down into manageable chunks. Test as you go along. My advice would be to first make the overall *JFrame* and its three components. Add the components to the content pane of the *JFrame* as in this code example. I would make the top component be a *JPanel,* (containing the *Open* and *Quit* buttons), a *JComponent* for the drawing area, and finally another *JPanel* for the *x* and *y* combo boxes and the *detail of a selected trade* text field.
5. It can be tricky to calculate good labels for the 'tick marks' — the small lines sticking out from the axis, with labels such as *-20, 0, 20, 40…120* in the image above. Calculate the minimum and maximum of the values for that axis, and then several options open up. You could look at more complex options, but one simple approach is to just put a tick mark for every gap of *(maximum-minimum)/k,* where *k* is the number of tick marks.

# Report

Write a short summary (no more than two pages, excluding screen dumps) of the final state of your program in a Word file. It should indicate how closely your program matches the specification. There should be a statement detailing how you tested that the program is working correctly before submitting the final version. This statement should include a description of the results you would expect when executing the program with the given CSV file (or another similar one). The report should include:

- Your ***name and student number*** at the top.
- Any assumptions you have made that have affected your implementation.
- Any known deficiencies, e.g., missing functionality or discrepancies between your expected and actual results and how you might correct them. If your program does meet the specification, all that is needed is ONE sentence stating this.

- Details of the tests used (and the results obtained) should be provided, using screenshots as appropriate. Testing could include working through a set of actions in the interface, or even unit testing of elements of the interface.

.

# Submission

You should submit a single zip file containing your .java files and your report (preferable .pdf, .doc also ok) through Moodle. Normal penalties for late submission apply.

# Marking Scheme

The assignment will be marked in bands on the scale A1,A2,.....F3,G1,G2,H. Marks will be awarded according to the quality of these separate elements:

Functionality (50% weighting)

- Proportion of the specified functionality implemented, and program correctness

Design (20% weighting)

- Choice of methods and instance variables within classes, design of methods and use of design pattern.

Documentation and Testing (20% weighting)

- Readability of code (comments, layout, indentation), quality of status report, testing procedure.

Combo Boxes (10% weighting)

- Functionality, design and documentation in respect of the combo boxes used to change what field names are used for which axes.

*Notes:*

- *We are expecting you to stick rigidly to the specification. Don't spend time. implementing functionality that was not asked for – it will not receive additional credit, and your time would be better spent on other tasks!*
- I*f major aspects of the functionality are not even attempted then the credit given for the Design, Documentation and Testing will be reduced. So, for example, if only half of the functionality is attempted then it will only be possible to obtain half marks for Design, Documentation and Testing*.