The wine samples data

You can download the two original datasets for the red and white Portuguese "Vinho Verde" varieties from BlackBoard. Both datasets were collected as part of a research project and made publicly available by the creators in the UCI Machine Learning Repository[1]. The filenames of these two datasets will be provided to your main class (`WineAssignment.java`). The first argument will refer to the file containing the red wine samples, and the second argument to the one containing the white wine samples. There will be a third argument provided to this class, and it will be discussed in the section *'The queries'*.

Both files are text and use the Comma Separated Values file format (CSV). In this case, the first row provides the twelve categories listed below and the remaining rows list the numerical values for each of the wine samples in the datasets. You are advised to use spreadsheet software to verify that your program is working as desired (e.g. Excel, Google Sheet, Numbers…). In Excel this means using its ability to load CSV files and Data-Filter capabilities. There are many online tutorials showing how to use this last feature (e.g. https://youtu.be/_OdsZR_rL1U).

Below you can find a brief description of the wine properties available for each wine sample (these are the twelve values listed in the first row of the CSV file):

Wine Properties - Objective physicochemical measurements:

1 - fixed acidity
2 - volatile acidity
3 - citric acid
4 - residual sugar
5 - chlorides
6 - free sulfur dioxide
7 - total sulfur dioxide
8 - density
9 - pH
10 - sulphates
11 – alcohol

Wine Properties - Subjective wine quality assessment (based on sensory data):

12 - quality (score between 0 and 10)

You do not need to know much about these and other terms to do this assignment, but if you wish to find out more, check the paper referenced in the UCI Machine Learning repository.

Note that the datasets do not provide a wine sample ID. You do not need to worry about that as the classes that you have been provided already assign an ID to each of the wine samples. Please, study the class `AbstractWineSampleCellar` and its `readWineFile` method.

The questions

Your programme (`WineAssignment.java`) will be printing to the console some basic details about the wine dataset including:

- The total number of wine samples (red and white) in the dataset.
- The total number of red wine samples in the dataset.
- The total number of white wine samples in the dataset.
- The highest alcohol content of the red wine samples.
- The ~~highest~~ citric acid content of the white wine samples.
- The average alcohol content of the samples of the white wine samples.

Lowest

The class already provides you with the template methods that will allow you answering these questions, check the methods `printNumberWines()`, `printQuestionAnswers()`, and `printFirstFiveWines()`. You will need to work on the class `WineSampleCellar` to make them work.

[1] http://archive.ics.uci.edu/ml/datasets/Wine+Quality

The queries

Your main class (`WineAssignment.java`) will receive as third argument another filename. That file will contain a list of individual queries of the form:

```
select red where qual > 6
select white or red where qual < 6 and qual > 4 and pH > 2.8
```

In the example above, the first query should produce all red wine samples whose quality is greater than 6.
The second should find white or red wine samples whose quality is between 4 and 6, with a pH above 2.8.

In general, queries will be built as follows:

```
select [red | white | red or white | white or red] where [subqueries]
```

where the `[subqueries]` can have as many `and` clauses as specified by the user. Note that the `or` clause can only be used to select the type of wine samples (i.e. in the first part of the `select` statement when indicating "red `or` white", but not in the list of subqueries)[2].
For example, to select all red wine samples with fixed acidity above 8, quality below 5, and pH between 2.7 and 2.79, one would use a slightly more complex query such as:

```
select red where f_acid > 8 and qual < 5 and pH > 2.7 and pH < 2.79
```

The `[subqueries]` in your queries will be your subqueries and they should support the following comparison operators:

Operator	Meaning
>	Larger than
>=	Larger than or equal to
=	Equal to
!=	Not equal to
<	Less than
<=	Less than or equal to

The notation used to represent the different wine properties in the queries is described in the following table:

Wine Property	Query keyword
Fixed acidity	`f_acid`
Volatile acidity	`v_acid`
Citric acid	`c_acid`
Residual sugar	`r_sugar`
Chlorides	`chlorid`
Free sulfur dioxide	`f_sulf`
Total sulfur dioxide	`t_sulf`
Density	`dens`
pH	`pH`
Sulphates	`sulph`
Alcohol	`alc`
Quality	`qual`

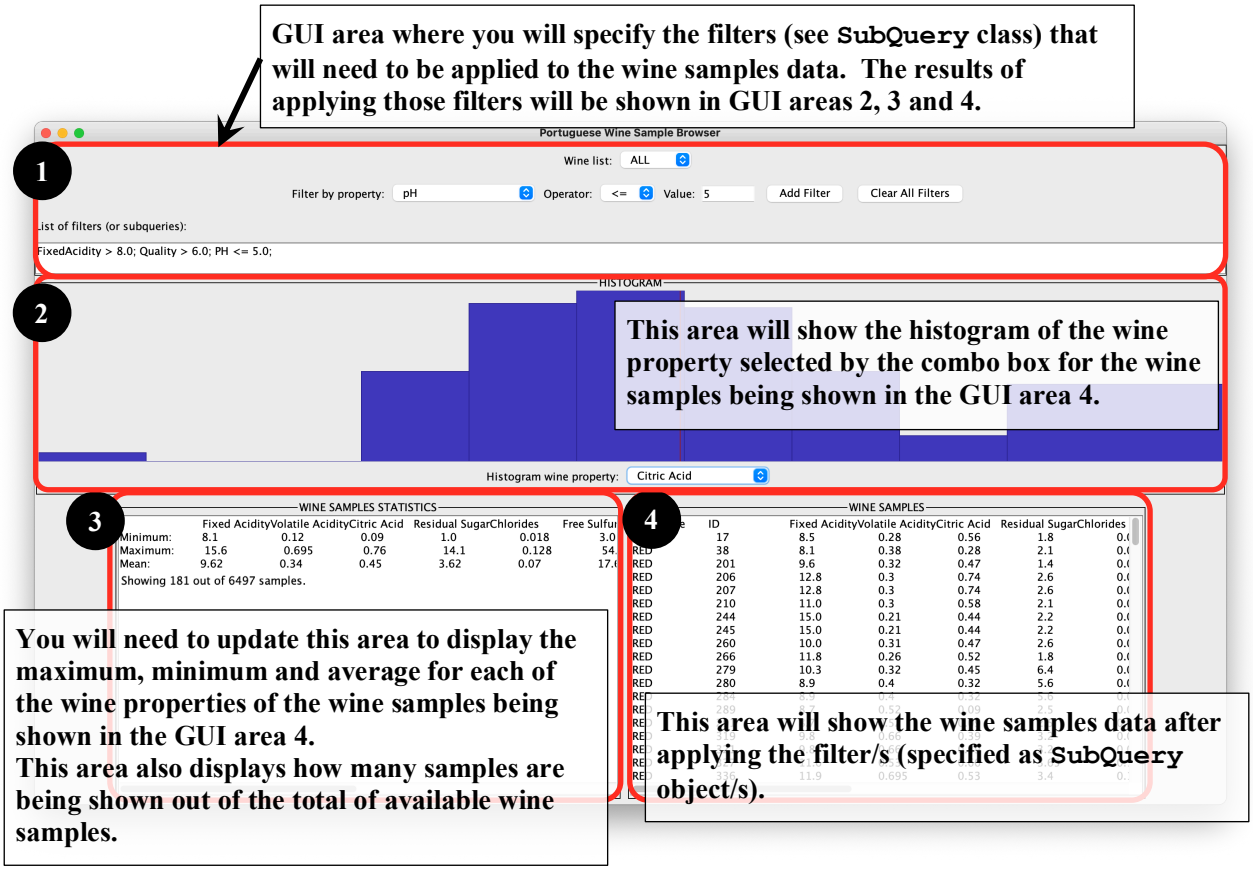
In most cases your queries will return multiple wine samples, and your programme will provide **self-informative and easy-to-read outputs to the console** to each of the queries. The methods `printNumberQueries()` and `executeQueries()` will report via console the results after running all the queries in the provided queries file. To make them work, you will need to work on the classes `QueryParser`, `Query`, and `SubQuery`.

[2] This is meant to simplify your programming task.

The GUI

The figure below shows how the GUI could look like. You have been provided with a basic implementation of the main panel (class `AbstractWineSampleBrowserPanel`) showing the four areas highlighted below.

- *Area 1 (top):* Allows the user to specify the filters that will need to be applied to the wine samples data.
- *Area 2 (center):* Shows the histogram for the wine property selected after applying the filters listed in Area 1 (List of subqueries). The class `HistogramPanel` will show the histogram bins, and the mean value for that property. You should also add meaningful axes and labels to the histogram[3]. More details about this are shown in the section *'The Histogram'*.
- *Area 3 (bottom-left):* Shows basic statistics describing the resulting dataset after applying the filters listed in Area 1 (List of subqueries).
- *Area 4 (bottom-right):* Shows the subset of wine samples and their wine properties after applying the filters listed in Area 1 (List of subqueries).



Areas 2, 3 and 4 will be updated whenever the user changes either the Wine list (red, white or all), or clicks the buttons Add filter or Clear All Filters.

In the example shown in the figure above, the user specified three filters (or subqueries), which were translated into three `SubQuery` objects representing the subqueries "Fixed Acidity > 8.0", "Quality > 6.0" and "pH <= 5.0", respectively. Every time the user clicked the button Add Filter, a new filter was added to the list of subqueries being shown in the GUI area 1. If the user clicks the button Clear All Filters, the List of subqueries needs to be cleared and no filters should be applied to the dataset being shown in areas 2, 3 and 4. This means that the areas 2, 3 and 4 would need to be updated to display the complete (unfiltered) original dataset for the wine type selected with the combobox named "Wine list".

[3] The figure does not show neither the axes nor their labels.

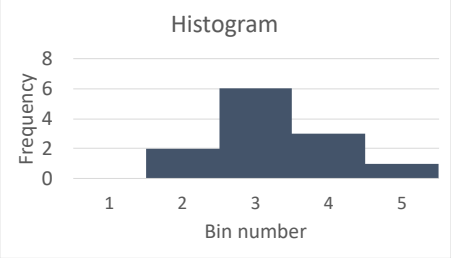
The Histogram

To understand the wine dataset, the GUI will offer the user the possibility of plotting a histogram for the different wine properties available in the dataset.

Histograms are used in statistics to plot as a bar chart the frequency of values of a quantity against a variable number of bins which contain the values in that class range. The height of the bins is proportional to the frequency, and their width is proportional to the class range, i.e. the range of values included in that bin.

For example, for an arbitrary list of values: { 2.1, 4.1, 6.3, 7.0, 6.9, 9.1, 3.0, 4.3, 5.1, 5.95, 4.3, 4.1 }, the table of frequencies for a number of bins n=5 and the corresponding histogram are:

Bin number	Class interval	Frequency	Values in the bin
1	0-2	**0**	{}
2	2-4	**2**	{2.1, 3.0}
3	4-6	**6**	{4.1, 4.3, 4.3, 4.1, 5.1, 5.95}
4	6-8	**3**	{6.3, 7.0, 6.9}
5	8-10	**1**	{9.1}



The histogram plots that bin 1 contains 0 values, bin 2 contains the 2 values in {2.1, 3.0}, bin 3 contains the 6 values in {4.1, 4.3, 4.3, 4.1, 5.1, 5.95}, bin 4 contains the 3 values in {6.3, 7.0, 6.9}, and bin 5 contains the single value {9.1}.

Please, study carefully the classes `HistogramBin` and `AbstractHistogram`, as they are well documented and provide you detailed guidance about what you need to do. Your class `Histogram` will extend `AbstractHistogram` and will be in charge of building the `Histogram` object that the `HistogramPanel` will plot in the final GUI. You have also been provided with the abstract class `AbstractHistogramPanel` that your `HistogramPanel` class will need to extend. Your class `HistogramPanel` will use Java2D for plotting. You are expected to plot not only the bins but also the axes labels, tick marks and even the class interval ranges (e.g. in the example above, bin 3 contains values in the range 4-6[4]).

You are not meant to use any external packages to plot the histogram.
You are meant to use exclusively Java2D for that.

[4] The `HistogramBin` class defines if the lower and upper boundary of an interval are included or not in that bin.

Programming task

The overall aim of the assignment is to produce a Java program (`WineAssignment.java`). The program will print in the console the answers to some questions about the dataset and the answers to a number of queries provided in a text file. The program will also open a GUI that will allow users to browse the wine dataset and exploring the wine properties of each wine sample. The user will be able to interact with the GUI to indicate which wine samples need to be shown. You will need to write several new classes that operate together with a set of classes that are provided (see below for details). Your program should work as follows.

There must be a `WineAssignment` class (i.e. a file called `WineAssignment.java`) with a `main` method. This will be the main class, and when invoked, the main method will do the following:

- Read the command line arguments to obtain the names of the three input files required by **`WineAssignment`** (pay attention to the order in which the three filenames need to be provided: 1) red wines CSV file, 2) white wines CSV file, and 3) queries text file). You have already been provided with the following piece of code in your class `WineAssignment`. Please, DO NOT change it:

```
public static void main(String[] args) {
    if (args.length == 0) {
        args = new String[] {
            "resources/winequality-red.csv",
            "resources/winequality-white.csv",
            "resources/queries.txt"};
    }
    String redWineFile = args[0];
    String whiteWineFile = args[1];
    String queriesFile = args[2];

    WineAssignment wineAssignment =
            new WineAssignment(redWineFile, whiteWineFile, queriesFile);

    wineAssignment.startCLI();
    wineAssignment.startGUI();
}
```

- Pay attention to the constructor of the class `WineAssignment`, as it creates from the input text files the data structures where the wine samples data will be stored (using the class `WineSampleCellar`), and the list of queries that will be executed (using the classes `AbstractQueryParser` and `QueryParser`). You will be using these in your program.
- The main will call the `startCLI()` method, which will display some basic information about the dataset, and will also display the answers to the queries provided in the file containing the list of queries. You should start with this part, and once it is completed, you should move to the GUI. While you work on this part, you are recommended to just comment the call to the `startGUI()` method.
- The main will call the `startGUI()` method, which will start the Graphical User Interface that will allow the user to interact with the dataset. Once you have completed the client part of this assignment, you should move to implementing the GUI. While you work on the GUI, you are recommened to just comment the call to the `startCLI()` method. Do not forget to verify that it still works while working on your GUI and remember to uncomment it before submission.

Classes provided

Several classes are provided on BlackBoard. They are organised in several subpackages. All of them are part of the package `uk.ac.sheffield.assignment2021.codeprovided` package. ***You must use these classes and you must <u>not</u> modify them***. Please, study very carefully these classes and how they are interrelated. The classes provided are:

- To represent the data stored in the CSV files:
  - `WineProperty.java` is a helper enum that provides a list of helpful constants describing the wine properties of a wine sample.
  - `WinePropertyMap.java` is a simplified `Map` wrapper to be used by a `WineSample`, storing the values for each of its properties.
  - `WineType.java` is a helper enum that provides a list of helpful constants describing a wine type.
  - `WineSample.java` contains all the necessary member variables and methods to store a wine sample. This means that one row in the original CSV files provided will become a `WineSample` object in this project.
  - `AbstractWineSampleCellar.java` is an abstract class that provides implementations of file reading utilities for the input datasets containing the wine samples and the query text file. The class stores in a `Map` all the wine samples read from the CSV files (variable `wineSampleRacks`). Please study well how to use the `Map` collection and what each of the abstract methods provided are meant to do. These abstract methods are the ones that your **`WineSampleCellar`** will need to override to provide the functionalities described in this handout.
- To represent the queries and read the queries from the input queries text file:
  - `Query.java` is a class that represents one query. There will be one `Query` object for each of the queries contained in the text file with a list of queries provided to you. Please, study well this class and how it relates to the `SubQuery` class described next.
  - `SubQuery.java` is a class that specifies one of the subqueries that are part of a `Query` object. The GUI (in particular the class `WineSampleBrowserPanel`) will create objects of this type to represent the user-specified filters or subqueries (using the GUI controls in GUI Area 1).
  - `AbstractQueryParser.java` is an abstract class that you will use to parse the queries stored in the queries text file. This class provides some functionality to read from the queries text file, extracting a list of strings that will need to be converted into a list of `Query` objects by the method `readQueries`. Your **`QueryParser`** class will extend `AbstractQueryParser` and will need to program the `readQueries` method.
- To help representing a histogram:
  - `AbstractHistogram.java` is an abstract class that will help you build a histogram. Your **`Histogram`** class will extend `AbstractHistogram`.
  - `HistogramBin.java` is a class that represent one bin in a histogram. You will be using this class in your `Histogram` class.
- To help creating the GUI:
  - `AbstractWineSampleBrowserPanel.java` is an abstract class that provides the basic code you will need to build a GUI similar to the one shown in the figure below. It defines a list of methods that your **`WineSampleBrowserPanel`** class will override to provide the functionalities described in this handout.
  - `WineSampleBrowser.java`. This class will create the GUI. You have been provided with this class and you will not need to modify it.
  - `AbstractHistogramPanel.java` is an abstract class that will plot a `Histogram` object in the GUI. Your class **`HistogramPanel`** will extend this class.

- To help you get started, you have been provided with basic templates of the classes that need to be implemented: `WineAssignment.java`, `WineSampleCellar.java`, `QueryParser.java`, `Histogram.java`, `HistogramPanel.java`, and `WineSampleBrowserPanel.java`.

Classes to implement

In this project, you must fully provide an implementation of the following classes. They will need to operate with the classes that have been provided to you. Each class should have the name specified in bold, although the way that these classes interact is up to you. You have been provided with a template implementation of each of them:

- **`WineAssignment.java`** (your main class)
- **`WineSampleCellar.java`** (will extend `AbstractWineSampleCellar`)
- **`QueryParser.java`** (will extend `AbstractQueryParser`)
- **`Histogram.java`** (will extend `AbstractHistogram`)
- **`HistogramPanel.java`** (will extend `AbstractHistogramPanel`)
- **`WineSampleBrowserPanel.java`** (will extend `AbstractWineSampleBrowserPanel`)

You might need to create additional helper classes. You will also need to submit those.
Your classes should not be using any external APIs.

Coding style

For coding style, readability is of great importance. Read through the Java coding guidelines used by Google, https://google.github.io/styleguide/javaguide.html. Take care with line breaks and whitespace, use comments only when required (i.e. not excessively, but you should have a JavaDoc comment block at the head of each class), use indentation consistently with 4 whitespaces per indent. Before you hand in your code, ask yourself whether the code is understandable by someone marking it. In your design you should aim to have classes that balance cohesion and coupling, so that each class has clear and natural responsibilities within the program.

Your code should adhere to the following style – note that in Eclipse, you can format your code automatically, and a quick internet search will tell you how to do this:

- Indentation 4 spaces (not tabs) per block of text.
- As a minimum you should provide a JavaDoc block and comment block at the head of each class.
- Other comments indented to the same level as surrounding code, with correctly formatted JavaDoc.
- Empty lines and whitespace used to maximize code readability.
- One variable declared per line of code.
- Sensible and descriptive names for methods and variables.
- No empty code blocks or unreachable code, no empty catch blocks.
- No big chunks of commented (unused) code.
- All class names start with Capital letter, i.e. in CamelCase, all method, member, and parameter names in lowerCamelCase.
- Sensible balance of coupling and cohesion in each class (look up cohesion and coupling if you don't know what these terms mean).

How to proceed

This may seem like a complex task, this handout is quite long and you have been provided with numerous classes. The first thing you should do is to examine the classes that are provided. Do not start coding straight away. Think carefully about what each of these classes knows (what are the instance variables?), and what they do (what are the methods?). Then consider the associations between the classes.

Next, think through the logical series of operations that the `WineAssignment` class should do to work as described in the handout. Which methods and objects are involved at each stage? Try to break each stage down into small chunks. Again, don't do any coding. Write your ideas down on a piece of paper. Design your new classes.

What does each class need to know (what are the instance variables?), and what does it need to do (what are the methods?)? Go back over your design and see if there are any problems.

Now you can start coding. Make sure you write your code incrementally. You are recommended to start with the client part of the programme (`startCLI()` method in `WineAssignment`). Once completed that part, move to the GUI (`startGUI()` method in `WineAssignment` class).

You are not requested to provide any test classes. You have been provided with some test classes for you to verify that your solution is working as expected. The provided tests are not complete, so it might be a good idea to write small test classes to create objects from the code provided and verify that they work correctly. Remember that you can use a spreadsheet software to verify your answers. Add small sections of code to your new classes and then test to see if they will compile and run. Write test classes and go back to refine your design if you need to.

Testing your classes with provided basic tests

You have been provided with two packages: `uk.ac.sheffield.assignment2021.test.common` and `uk.ac.sheffield.assignment2021.test.forstudents`. The provided tests are meant to help you assess that your solution is 'correct' and that you are working in the right direction. Your submitted code will need to successfully pass all the tests provided. Note that the battery of tests provided doesn't fully test your solution, so even if your solution passes these tests, you need to be sure that your solution is actually fully correct, as during marking, other tests will be run on your code.

Submission and deadline

There is an upload link on BlackBoard**.** You should upload your java source code as a compressed zip archive to BlackBoard before the submission deadline. ***You must upload a .zip file and not any other form of compressed file. For example, .tgz, and .rar files will not be accepted.***

The zip file should be named according to your CICS username, e.g. if your user name is aca21xxx then your file should be named aca21xxx.zip. The zip file should uncompress into a folder with your CICS username 'aca21xxx', containing the subfolder 'uk', which will contain 'ac', which will contain 'sheffield' which will contain 'assignment2021' containing your source code, with a sub-folder called 'codeprovided' containing the provided classes. You can use Eclipse to export your code as shown for Problem sheet 2.

Submission checklist

It is imperative that you adhere precisely to the required program structure outlined in this assignment. If you do not, your code may not be compatible with the marking system and will be returned to you.

- Ensure that your classes are part of the `uk.ac.sheffield.assignment2021` package, and import the classes provided for you from the `uk.ac.sheffield.assignment2021.codeprovided` package (and subpackages).
- Ensure that your submission has the following classes in the `uk.ac.sheffield.assignment2021` package, with exact matches to filenames:
    - `WineAssignment.java`
    - `WineSampleCellar.java`
    - `QueryParser.java`
    - `Histogram.java`
    - `HistogramPanel.java`
    - `WineSampleBrowserPanel.java`
- If your submission uses any **additional helper classes**, please do include them as well as part of the package `uk.ac.sheffield.assignment2021`. You are expected to need some additional classes to your assignment.
- Also ensure that your submission has the **unmodified** classes in the package (and subpackages) of `uk.ac.sheffield.assignment2021.codeprovided`.

Your code must compile from the command line from the `src` folder with the command `javac uk/ac/sheffield/assignment2021/WineAssignment.java` (on MacOS/Linux operating systems) or `javac uk\ac\sheffield\assignment2021\WineAssignment.java` (on Windows). Test this before you submit your code and make sure that you can run your code from command line as well.