**The Datasets**

Two datasets are included in the src/main/resources directory in your GitHub repository: epl-2223.csv and laliga-2223.csv. These datasets have been adapted from an open-source Kaggle dataset. The filenames of these two datasets will be provided to your main class (FootieDashboard.java) in a fixed order: EPL first and LIGA second. There will be a third argument provided to this main class, which will be discussed in the section *‘The Queries’*.

Both data files are plain text and use the Comma Separated Values file format (CSV).  The first row provides the column **header** (i.e., property name) listed below and the remaining rows contain the actual values for each of the properties in the datasets (in the same order as the headers). Each of these lines are considered a **player entry**.

The following table provides a brief description of each properties available in the datasets for each player entry (i.e., these are the twelve values listed in the first row of the CSV file):

| **Index** | **Header** | **Description** | **Category** |
| --- | --- | --- | --- |
| 1 | Player | Player's name | Details |
| 2 | Nation | Player's nation | Details |
| 3 | Position | Position in the pitch | Details |
| 4 | Team | Team’s name | Details |
| 5 | Age | Player's age | General |
| 6 | Matches | Matches played | General |
| 7 | Minutes | Minutes played | General |
| 8 | YellowCards | Yellow cards per 90 minutes | General |
| 9 | RedCards | Red cards per 90 minutes | General |
| 10 | Goals | Total number of goals scored | Goals |
| 11 | PKGoals | Penalty kick goals per 90 minutes | Goals |
| 12 | PKAttempts | Penalty kicks attempted per 90 minutes | Goals |
| 13 | Assists | Assists per 90 minutes | Goals |
| 14 | OwnGoals | Own goals per 90 minutes | Goals |
| 15 | PassAttempted | Passes attempted per 90 minutes | Play |
| 16 | PassCompleted | Passes completed per 90 minutes | Play |
| 17 | AerialsWon | Aerial challenges won per 90 minutes | Play |
| 18 | AerialsLost | Aerial challenges lost per 90 minutes | Play |
| 19 | AerialsWonPerc | Percentage of aerial challenges won per 90 minutes | Play |
| 20 | Tackles | Tackles made per 90 minutes | Defence |
| 21 | TacklesWon | Tackles that regained ball possession per 90 minutes | Defence |
| 22 | Clearances | Clearances made per 90 minutes | Defence |
| 23 | FoulsCommitted | Fouls committed per 90 minutes | Defence |
| 24 | PKConceded | Penalty kicks conceded per 90 minutes | Defence |
| 25 | Shots | Shots total per 90 minutes | Attack |
| 26 | ShotsTarget | Shots on target per 90 minutes | Attack |
| 27 | FoulsDrawn | Fouls drawn per 90 minutes | Attack |
| 28 | Crosses | Crosses made per 90 minutes | Attack |
| 29 | PKWon | Penalty kicks won per 90 minutes | Attack |

*Table 1. Description of the properties for each player entry in the dataset, and how they are grouped by category to be shown in the radar chart (see ‘The GUI’ and ‘The Radar Chart’).*

Note that the datasets do not provide a player 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 players (be aware that player names may be duplicated in multiple **player entries** when a player transfers from one team to another mid-season, e.g., *João Félix* from *Atlético Madrid* to *Chelsea*). Please, study the class AbstractPlayerCatalog and its readDataFile method.

You are strongly advised to use **spreadsheet software** to verify that your program is working as desired (e.g., Excel or Google Sheet). In Excel this means using its ability to load CSV files and data-filter feature.

***The questions***

Your program (FootieDashboard) will be printing to the console some basic details about the dataset, including:

* The total number of *unique* players in the combined dataset (EPL *and* LIGA).
* The total number of player entries in the EPL dataset.
* The total number of player entries in the LIGA dataset.
* The highest number of assists in the EPL.
* The highest number of fouls committed in the LIGA.
* The average number of shots on target across the EPL and the LIGA.

The class already provides you with the template methods that will allow you to answer these questions, check the methods printNumberUniquePlayers(), printQuestionAnswers(), and printFirstFivePlayerEntries(). You will need to work on the class PlayerCatalog to make them work.

***The queries***

Your main class (FootieDashboard) will receive another filename as a third argument which will contain a list of individual queries (one per line) of the form:

select epl where Goals > 2

select epl or liga where Goals > 2 and Goals < 5 and Assists > 0.5

In the example above, the first query should produce all player entries in the EPL where the number of goals is greater than 2.  The second query should find player entries in either EPL or LIGA who scored between 2 and 5 goals and made more than 0.5 assists per 90 minutes.

In general, queries will be built as follows:

select *[epl | liga | epl or liga | liga or epl]* where *[subqueries]*

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 league (i.e., in the first part of the select statement when indicating “epl or liga” (or vice versa), but not in the list of subqueries after where).

For example, to select all LIGA player entries with more than 0.5 yellow cards per 90 minutes, fewer than 2 tackles made per 90 minutes, and between 100 and 200 minutes played (inclusive), the user would formulate a slightly more complex query:

select liga where YellowCards > 0.5 and Tackles < 2 and Minutes >= 100 and Minutes <= 200

The [subqueries] in your queries will be your subqueries and they should support the following comparison operators **only** for player properties (see class PlayerProperty) and **not** player details (see class PlayerDetail):

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

The headers in the datasets indicate the *exact* syntax to be used in the queries (**not** case sensitive). You will need to control syntactic errors in the provided queries, if there is anything that falls out of the allowed syntax, an IllegalArgumentException should be raised and handled appropriately. You can assume that there will be one query per line in this file, but malformed queries are possible and should be handled appropriately using exceptions to ignore them.

In most cases your queries will return multiple player entries, and your program will provide **informative** **and** **easy-to-read** **outputs to the console** for each of the queries.  The selection can range from one player to all the players in the dataset (an empty selection should be handled gracefully, too).

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.

***The GUI***

The figure below shows an example of how the dashboard GUI **could** look like.  You have been provided with a basic implementation of the main panel containing the four areas highlighted below (see class AbstractPlayerDashboardPanel).

* *Area 1 (top):* Allows the user to specify the filters that will need to be applied to the football players performance dataset (see SubQuery class). The results of applying these filters will have to be shown in GUI areas 2, 3 and 4.
* *Area 2 (left):* Shows the radar chart for the player attributes selected after applying the filters listed in Area 1 (List of subqueries). The class RadarChartPanel will show an axis for each of the selected attributes and values will be plotted to form a polygon across the axes. Plots for the min, max and average for each selected property should be overlapped to the selection plot using different colours. You should also add meaningful labels to the radar chart. The set of properties to be drawn in the radar chart is determined by the *category drop-down menu* (see Table 1 and bottom of area 2). More details are provided in section *‘The Radar Chart’*.
* *Area 3 (right-top):* Shows basic statistics (maximum, minimum and average) for the resulting dataset after applying the filters listed in Area 1 (List of subqueries). Only numeric attributes are included in this area (all of them). This area could also display how many entries are being shown out of the total of entries in the full dataset.
* *Area 4 (right-bottom):* Shows all the details and properties of the selection of player entries resulting from applying the filters listed in Area 1 (List of subqueries, i.e., SubQuery object(s)).

图形用户界面, 图表, 应用程序

描述已自动生成

Areas 2, 3 and 4 must refresh whenever the user changes either the league dataset (EPL, LIGA or ALL); the player’s name, nation, position or team; or clicks the buttons *Add filter* or *Clear All Filters*.

In the example shown in the figure above, the user specified two filters (or subqueries), which were translated into two SubQuery objects representing the subqueries “yellowcards > 3”, “tackles >= 10”, and “minutes <= 90”, 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 league selected with the combo boxes labelled “League”, “Player’s name”, “Nation”, “Position”, and “Team”.

As mentioned before, the screenshot above is just an example. You have flexibility to implement your GUI to look as you wish within the given constraints. For example, a JTextArea is used in areas 3 and 4. Since the formatting options for the JTextArea component are limited, the assessment will consider what is achievable with a JTextArea.

**The Radar Chart**

To understand the football players performance dataset, the GUI will allow the user to plot a radar chart for a selection of the different numeric performance attributes available in the dataset.

Radar charts, also known as spider charts, are used in statistics to visualise multivariate data (multiple variables – at least three – at once). An axis comes out from a central point for each variable (in this assignment called performance attributes). The angles of the different axes depend on the number of attributes included and should evenly span 360 degrees. The length of each axis should be fixed, but the values plotted on it should be **normalised**. The value for each property should be plotted on top of the corresponding axis, and all the values should be connected around the axes.图表, 雷达图

描述已自动生成

For example, given the data in the table below, a corresponding radar chart could look like the one on the right. Notice that George is the best at solos, Ringo has the best rhythm, and both John and Paul are excellent composers, with John being arguably the funniest one.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Composition** | **Voice** | **Rhythm** | **Solos** | **Humour** |
| **John** | **10** | **8** | **6** | **5** | **9** |
| **Paul** | **10** | **7** | **6** | **7** | **6** |
| **George** | **8** | **7** | **6** | **10** | **5** |
| **Ringo** | **2** | **2** | **10** | **5** | **5** |

Please, study carefully the classes RadarAxisValues and AbstractRadarChart, as they are well documented and provide you detailed guidance on what you need to do. Your class RadarChart will extend AbstractRadarChart and the RadarChartPanel class will create an object of type RadarChart and will plot it in the final GUI. You have also been provided with the abstract class AbstractRadarChartPanel that your RadarChartPanel class will need to extend. Your class RadarChartPanel will use Java2D for plotting.  You are expected to plot not only the radar chart but also the axes’ labels (e.g., in the example above, “Composition”, “Voice”, “Rhythm”, “Solos”, and “Humour” are the labels) and the axes’ values (e.g., in the example above, all axes have the same scale, from 0 to 10, so values are plotted along one single axis, but in general, each axis should have its own value labels).

The radar charts in this assignment will always show five player entry properties (only numeric columns in the datasets). Table 1 provides the mapping between player entry properties and the five categories to be displayed (*General, Goals, Play, Defence, and Attack*). The class Category will be helpful.

**Scaling of player properties when drawing a radar chart:**  
By the time you start working on drawing a radar chart, you will have already seen that different player properties come in significantly different value ranges. This means that when drawing your radar chart, the values in each axis will go from 0 to the maximum value for the player property that the axis displays for the complete dataset (ALL).

For the dataset you have been provided (ALL), if we pick three properties (e.g. Player’s age, Minutes played, and Penalty kick goals), you will need to use their maximum value to scale the values shown in the chart.  For these three properties, the maximum values to be used for the scaling are:

* Player’s age: 41,
* Minutes played: 2067,
* And Penalty kick goals: 0.74.

This means that to have meaningful radar charts, you will need to scale the Player’s age axis to go from 0 to maximum value in the whole dataset (ALL), which in this case is 41, similarly for Minutes played, from 0 to the maximum value of that property for the whole dataset, in this case 2067, and similarly for the Penalty kick goals axis to go for 0 to 0.74.

**You are not allowed to use any external packages to plot the radar chart.   
You are meant to use Java2D exclusively for that.  
Those submissions using an external package for this part will be marked as 0.**

**The Programming Task**

The overall aim of the assignment is to produce a Java program (FootieDashboard.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 players performance dataset and explore performance properties for a selection of player entries. The user will be able to interact with the GUI to indicate which player entries need to be shown. You will need to write several new classes that will integrate with a set of classes that are provided for you (see below for details). Your program should work as follows.

There must be a FootieDashboard class (implemented as a public class in a Java source file called FootieDashboard.java) with a main method. This will be the main class, and its main method will do the following:

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

public static void main(String[] args) {

   if (args.length == 0) {

      args = new String[] {

        "src/main/resources/epl-2223.csv",

        "src/main/resources/laliga-2223.csv",

        "src/main/resources/queries.txt"};

   }

   String eplFile = args[0];

   String ligaFile = args[1];

   String queriesFile = args[2];

   FootieDashboard footieDashboard =

            new FootieDashboard(eplFile, ligaFile, queriesFile);

   footieDashboard.startCLI();

   footieDashboard.startGUI();

}

* Pay attention to the constructor of the class FootieDashboard, as it uses the input text files to create the data structures where the player performance entries will be stored (using the class PlayerCatalog), and the list of queries that will be executed (using the classes AbstractQueryParser and QueryParser). You will be using these in your program.
* FootieDashboard.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 on to the GUI. While you work on this part, you are recommended to just comment the call to the startGUI() method.
* FootieDashboard.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 on to implementing this GUI.

***Classes provided***

Your GitHub assignment repository includes several classes, organised in several subpackages, under package uk.ac.sheffield.com1003.assignment.codeprovided. ***You must use these classes and you must not modify them***. You are expected to familiarise yourself with these classes and the relationships between them. The classes provided are:

* To represent the data stored in the CSV files:
  + PlayerProperty.java is a helper enum that provides a list of helpful constants describing the player performance properties in a player entry.
  + PlayerPropertyMap.java is a simplified Map wrapper to be used by a PlayerEntry, storing the values for each of its properties.
  + League.java is a helper enum that provides a list of helpful constants for the leagues being considered.
  + Category.java is a helper enum that provides a list of helpful constants and methods representing which player entry properties map to each category. These will be used when plotting the radar chart.
  + PlayerEntry.java contains all the necessary member variables and methods to store a player performance entry. This means that one row in the original CSV files provided will become a PlayerEntry object in the system.
  + AbstractPlayerCatalog.java is an abstract class that provides implementations of file reading utilities for the input datasets containing the player entries and the query text file. The class stores in a Map all the player entries read from the CSV files (variable playerEntriesMap). 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 PlayerCatalog 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 this class well 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 PlayerDashboardPanel) 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 implement the readQueries method.
* To help representing a radar chart:
  + AbstractRadarChart.java is an abstract class that will help you build a radar chart.  Your RadarChart class will extend AbstractRadarChart.
  + RadarAxisValues.java is a class that represents one axis in the radar chart. You will be using this class in your RadarChart class.
* To help creating the GUI:
  + AbstractPlayerDashboardPanel.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 PlayerDashboardPanel class will override to provide the functionalities described in this handout.
  + PlayerEntryDashboard.java. This class will create the GUI. You have been provided with this class and you will not need to modify it.
  + AbstractRadarChartPanel.java is an abstract class that will plot a RadarChart object in the GUI. Your class RadarChartPanel will extend this class.
* To help you get started, you have been provided with basic templates of the classes that need to be implemented: FootieDashboard.java, PlayerCatalog.java, QueryParser.java, RadarChart.java, RadarChartPanel.java, and PlayerDashboardPanel.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:

* FootieDashboard.java (your main class)
* PlayerCatalog.java (will extend AbstractPlayerCatalog)
* QueryParser.java (will extend AbstractQueryParser)
* RadarChart.java (will extend AbstractRadarChart)
* RadarChartPanel.java (will extend AbstractRadarChartPanel)
* PlayerDashboardPanel.java (will extend AbstractPlayerDashboardPanel)

You might need to create additional helper classes and commit them to your repository. Your classes should **not** use any external APIs.

***Coding style***

For coding style, readability is of great importance. Read through the [Google Java Coding Guidelines](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 any IDE, 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 maximise 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, but this handout is thorough and you have been provided with numerous classes to help you get started. You should start by inspecting the classes you are provided with. 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 FootieDashboard 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 just yet. 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 that you understand the problem at hand and have an outline of a solution, you can start coding. Make sure you write your code incrementally and make frequent commits to your repository. You are recommended to start with the client part of the program (FootieDashboard.startCLI() method). Once you complete that part, move on to the GUI (FootieDashboard.startGUI() method).

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 your own test classes to create objects from the code provided and verify that they work correctly. *You are not required to implement any test classes but you are encouraged to include your tests in your repository, although they will not be assessed*.

Remember that you can also use 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 a JUnit test class in the src/main/test directory. The provided JUnit 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.