COIT20245

Assignment 2

# Introduction

In this assignment, you are to implement a console application for the management of a premiership table for an Australian Rules football league. The table will have the structure shown in Figure 1.

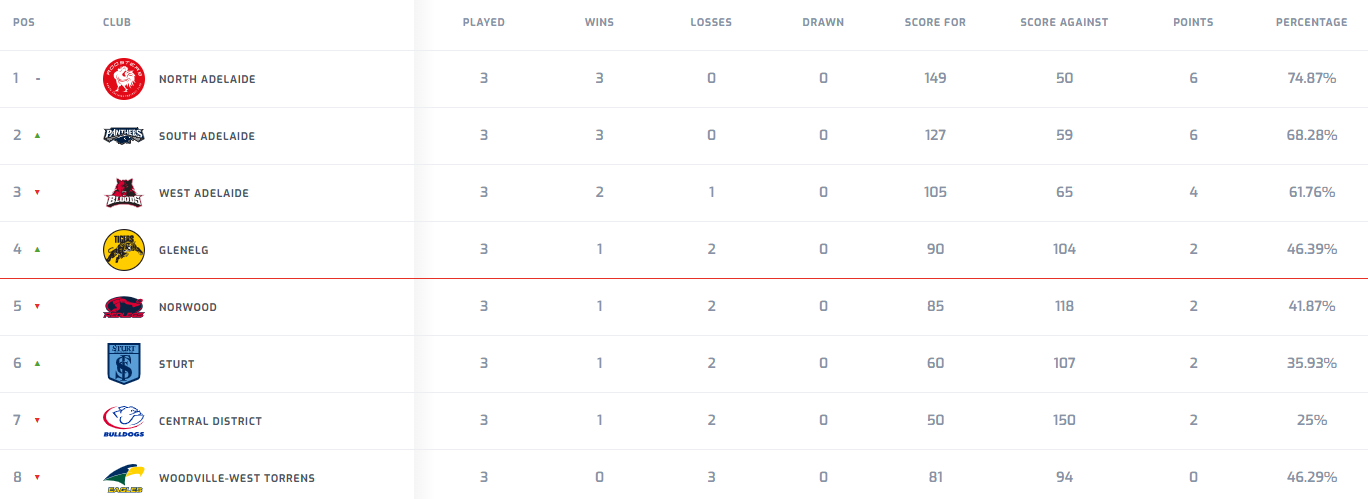


Figure 1. Premiership table for the SANFL Women’s League after Round 3

In this particular league, there are 8 teams, with names of your choosing - I recommend t1-t8. Also, the table displays team names only and not team icon + team name, as in Figure 1. Position in the table is determined on the basis of premiership points (points in Figure 1); teams with the same number of premiership points are ranked according to percentage. Premiership points are awarded for each game – 2 to the winning team, 0 to the losing team and if the game is drawn, 1 to each team. As described in Assignment 1, the outcome of a game is determined using the points scored by each team, where points = goals\*6+behinds. The percentage for each team is calculated as follows:

percentage = score for / (score for + score against)

where

* score for is the total number of points scored by the team in all games played so far and
* score against is the total number of points that the opposing teams have scored in all games played so far

Note that in the AFL, the premiership points available per game and the way in which percentage is calculated are different, but the ranking process is the same. And what happens, if at the end of the season, two teams have the same number of premiership points, score for and score against? I don’t know.

# The Application

The current state of the premiership table is to be stored in 6 arrays, using array initialisers. E.g.

// The data is from a sorted table, so there is no need to sort the table

// once it is generated. The data is from the SANFL women's league, where

// scores are lower than in the AFL. Also in the SANFL, premiership

// points and percentages are calculated differently to the AFL.

String teams[] = {"t1", "t2", "t3", "t4", "t5", "t6", "t7", "t8"};

int wins[] = {3, 3, 2, 1, 1, 1, 1, 0};

int losses[] = {0, 0, 1, 2, 2, 2, 2, 3};

int drawn[] = {0, 0, 0, 0, 0, 0, 0, 0};

int pointsFor[] = {149, 127, 105, 90, 85, 60, 50, 81};

int pointsAgainst[] = {50, 59, 65, 104, 118, 107, 150, 94};

Note that not all columns in Figure 1 are present. Team position is determined from its index in the arrays (add 1 to the index). The remaining columns can be derived from the above data.

The raw data is to be used to create objects of type Entry, which are to be stored in an array of type Entry. The Entry class is to conform to the class shown in the UML Diagram of Figure 2[[1]](#footnote-1). Note that in Figure 2, the visibility of class methods and attributes are designated as + (public) and – (private). Associations are annotated with a label (for readability) and multiplicity. The multiplicity captures the number of object instances that can be involved in the association. In this case, only two values apply – 1 and \*, which means many.

Creation of the Entry array is to occur when the Table object is created, using a private method called load(). This method will iterate through the 6 arrays that define the current state of the table, construct Entry objects and add them to the entry array. As befitting a premiership table, the data array is ordered on position, as in Figure 1. When the table is updated, the table will need to be sorted. As noted in Section 1, this will require sorting on two attributes – the primary attribute is premiership points and the secondary attribute is percentage. The easiest way to do this is to define a compare function:

private int compare(Entry e1, Entry e2) {

// Returns

// -1 if e1 < e2

// 0 if e1 == e2

// +1 if e1 > e2

}

Your (private) sort method **must** implement selection sort.

The application’s Controller class is to execute (using a switch statement) the following command options:

1. Display available commands
2. Display current table
3. Display selected statistics
4. Display the entry for a specified team
5. Display entries for teams with the same points as a specified team
6. Add a new result
7. Exit the application

A screenshot of a cell phone

Description automatically generated

**Figure 2. Class Diagram**

As it is a console application, the user will need to be prompted to enter a command and its arguments (if any). My personal preference is for a minimal interaction scheme, as shown below:

run:

Available commands are:

help 0

display entries 1

display selected statistics 2

lookup a specified team 3 team

find teams with same points as a specified team 4 team

add a new result 5 team1 g1 b1 team2 g2 b2

quit 9

> 3 t4

Pos Team Played Points % Won Lost Drawn PF PA

4 t4 2 2 48.8 1 1 0 59 62

>

Feel free to adopt the above scheme or if you prefer, implement a more verbose interaction scheme.

Note that

1. Each command is designated a number
2. The command options are displayed at the start of the application and whenever a “help” command is entered, rather than after each command.
3. Percentages are to be displayed to 1 decimal point

For the commands that require arguments

1. For commands 3, 4 and 5, an error message must be displayed if a specified team is not in the table. No other data validation is required.
2. The application **must** conform to the class diagram of Figure 2, although additional private members and private methods are permitted.

# Submission

You are to submit two separate files:

1. Your zipped NetBeans project folder. Details of how to do this are available in the NetBeans FAQ on the unit website.
2. Report.docx. This file is to contain the following Sections:
3. Limitations
4. Test plan
5. Test results

Student name, student ID number, unit name and unit code are to be included on the title page.

The limitations section is to specify any limitations that your program has in terms of calculations and data validation.

The test plan is to contain a comprehensive list of program functionality to be tested, the input values to be used to test each item of functionality, the expected output from the test and the actual output from the test.

The test results section is to contain screenshots to demonstrate that the program generates the actual outputs shown in the test plan.

# Marking Criteria

|  |  |  |
| --- | --- | --- |
|  | **Criteria** | **Marks Allocated** |
| **1** | **Functionality** | /8 |
|  | Command loop |  |
|  | Sorting |  |
|  | Searching |  |
|  | Statistics |  |
| **2** | **Language Use** | /8 |
|  | Variable declarations |  |
|  | Constant declarations |  |
|  | Class definition |  |
|  | Object creation |  |
|  | Loop control |  |
|  | Selection statement use |  |
|  | Method definition and use |  |
|  | Reading input |  |
|  | Display of results |  |
|  | Array use |  |
|  | ArrayList use |  |
|  | Unnecessary/very inefficient code |  |
| **3** | **Layout and documentation** |  |
|  | Conventions (Spacing, indentation, naming) | /1 |
|  | Comments (must add value and follow guidelines) | /3 |
| 4 | **Report** |  |
|  | Limitations | /2 |
|  | Test Plan and Test Results (must be comprehensive) | /8 |
|  | **Sub-Total** | /30 |
|  | **Penalties** |  |
|  | Does not compile: up to 30 marks |  |
|  | Partial implementation: up to 30 marks |  |
|  | Late submission : 5% (1.5 mark) / day or part of a day |  |
|  | **Total** | /30 |

For Language Use and Functionality, a checklist is given, but not a detailed marks breakdown. For checklist item non-compliance, you will be penalised 0.5 – 2.5 marks, depending on the degree of non-compliance.

Note that it is **your** responsibility to ensure that source code files are included in your submission as well as report.docx. If either are missing, the assignment will not be marked.

# Appendix 1

TBD

1. Getters and setters can be generated automatically in NetBeans using Refactor > Encapsulate Fields… [↑](#footnote-ref-1)