COMP125 Semester 2 2016
Assignment 2 Description

September 29, 2016

1 Overview

This assignment asks you to complete a Java project which provides functionality to simulate a group of people playing in a golf tournament. The assignment gives you practice with a number of topics in COMP125: objects and classes, problem solving, sorting and searching, the ArrayList class, recursion, and more. The assignment is worth 10% of your final mark for this unit. A starting project **tournament.zip** is provided to you, and two electronic submissions are required from you, as described below. Some but not all of your work will be auto-tested. All your work will be looked over by a tutor. Your mark for the assignment will be given on the basis of both the auto-testing of certain methods (done by machine), and the quality of your work (as determined by visual inspection). Submit two completed files separately: **Player.java** and **Tournament.java**. Submit your work by 11.45 pm on Sunday 6 November.

2 Motivation, background and starting point

The motivation for developing software to simulate a golf tournament comes from the desire to offer the pleasure, excitement and element of luck of playing golf without the need to invest time and money joining a club, purchasing equipment, trudging around a golf course in all weathers, etc. The software to be developed could also be useful in managing a real golf tournament.

A golf course usually has 18 holes (or 19, if you include the bar back at the club house). A tournament often consists of four rounds through the golf course. In this assignment, we will simplify things a bit, and assume that the tournament consists of just one round, that is, 18 holes in total. We will allow any number of players to enter the tournament. (In practice, in a real tournament, the number of players will usually be capped.)

As most people know, even those who do not play golf, a player takes a certain number of strokes from the tee (starting point) to the hole. This number of strokes is known as the *score* for the hole. Of course, the aim is to minimise the score for each hole, and hence the total score for the whole course. Each hole has a *par* value

– typically 3, 4 or 5 – which represents the expected number of strokes which a good player will require for the hole. The par value is assessed based on the difficulty (length, terrain, etc.) of the hole. A serious golf player will usually have a *handicap*. This is the number of strokes which the player is likely to make *above par* for any golf course. For example, suppose that your handicap is 10. This means that you typically require 10 strokes above the total par to complete one round of a course. So, the lower your handicap, the better player you are.

The starting project supplied to you consists of two primary classes, namely `Player` and `Tournament`. (A sample Junit test class is also supplied for each one.) The `Player` class models the attributes of a player, namely, the player's name (family name only is stored, for simplicity), handicap and scores. This class also provides methods appropriate for a player, such as `totalScore` (returns player's total score) and `play` (simulate playing a round). Some methods are already complete, but others need work. The `Tournament` class models the key features of a tournament, namely, the list of par values for the holes of the course, and the list of players who are taking part in the tournament. Appropriate methods are also provided; again, some are already complete, but others need to be coded.

You need to complete the project step by step, as described below. The starting point for your work, and the requirements wanted from you, are described in more detail in the next section.

3 More detail about starting project and requirements

3.1 The code template provided

You are given a code template to start with. This is a zipped archive project folder `tournament.zip` available from iLearn. Download and import this file. This project (`tournament`) contains two primary Java source code files: `Player.java` and `Tournament.java`. There are two sample Junit test files as well, one for each class. It may be helpful to picture the classes defined therein as follows:

```
TournamentTest
    \
     \
   Tournament  PlayerTest
        \     /
         \   /
         Player
```

The above diagram is meant to suggest that the class `Player` is the simplest (most basic) of the four classes, that the classes `Tournament` and `PlayerTest` use (that is, are clients of) `Player` and that, in turn, the class `TournamentTest` uses (that is, is a client of) `Tournament`. (Note also that each class `Player` and `Tournament` contains a `main` method, which is a client of the class it is in. The `main` method provides a simple demonstration of the methods of the class.)

Both classes `Player` and `Tournament` are incomplete, and need to be completed by you. The methods which require completion are carefully specified in the comments supplied in the starting project.

The files `PlayerTest.java` and `TournamentTest.java` are simple JUnit test files which you could use to fine tune your methods and prepare for machine testing. You might want to add your own test routines to the test files.

3.2 Specific methods required and marks allocated

The assignment overall will be marked out of 100.

Please look first at the class `Player`. This class specifies for you the methods which need to be written by you. Auto-marking of certain methods of this class will be worth 20 marks altogether. In particular the following methods of this class will be auto-tested: `equalArrays(...)` (5 marks), `equals(...)` (5 marks), `comment(...)` (5 marks), `totalScore(...)` (5 marks). Your method `comment` should return an appropriate comment for the given score and par value. In particular, it should return `"condor!!!!"` for a score 4 below par, `"albatross!!!"` for a score 3 below par, `"eagle!!"` for a score 2 below par, `"birdie!"` for a score 1 below par, `"par"` for a score equal to par, `"bogey"` for a score 1 above par, `"double bogey"` for a score 2 above par, and `"triple+ bogey"` for a score more than 2 above par.

Your completed class `Player` will also be visually marked. A visual mark out of 15 will be awarded for this class. This will comprise 3 marks for method `play(...)`, 3 marks for method `totalScoreRec(...)`, 2 marks for completing method `main` as requested, 5 marks for overall code quality in the class, and 2 marks for your answer to Question P1.

Please look next at the class `Tournament`. This class specifies for you the methods which need to be written by you. Auto-marking of certain methods of this class will be worth 45 marks altogether. In particular the following methods of this class will be auto-tested: `enter(3 parameter version)` (5 marks), `winningScore(...)` (5 marks), `winners(...)` (5 marks), `alphabeticSort(...)` (10 marks), `findScore(...)` (10 marks), `ranking(...)` (10 marks).

Your completed class `Tournament` will also be visually marked. A visual mark out of 20 will be awarded for this class. This will comprise 3 marks for each of the three Questions T1, T2 and T3, 5 marks for overall code quality in the class, 2 marks for completing method `main` as requested, and 4 marks for method `alphabeticSortRec`. Type your answer to each question, comprising 6 to 10 lines, say, as a block comment following each question.

4 Hints for approaching the assignment

Develop your software gradually. Try to complete class `Player` (or most of it) first. Then move on to class `Tournament`. The basic principle continues to be: after a method is written, test it. You could carry out some testing using the supplied main method, which you could suitably add to (or modify) for each test. You could also use the sample JUnit test file provided to you to help you fine tune your methods. Feel free to write more JUnit tests for your methods to ensure they work correctly with a range of test data.

For class `Tournament`, you may wish to tackle the simpler methods `enter(...)`, `playRound(...)`, `winningScore(...)`, and `winners` first. Then address the remaining ones. For `alphabeticSort(...)` you are requested to adapt the **insertion sort** algorithm for this task. Please note:

- For coding this method you may find it helpful to first implement the suggested "helping" method `insertIntoSortedRegion(...)`.
- For comparing two names (strings) do not use the simple `<=` operator. Instead, look up the Java String methods and choose a suitable one to use for the comparision. *Hint.* Case (upper or lower) is not really relevant in determining the correct alphabetic order.

Hint for answering Questions T1, T2, T3. In your answers you may wish to refer to certain observations made in lectures concerning the time complexities (operation counts) of the basic searching and sorting methods which you studied earlier in the semester.

5 Summary of requirements

Complete the classes `Player` and `Tournament`. Include answers as block comments to the questions asked in each class. (In particular, answer Question P1 in class `Player` and Questions T1, T2, T3 in class `Tournament`.) Submit the files `Player.java` and `Tournament.java` using the separate submission boxes provided on iLearn. Do **not** submit a zip file, do **not** submit a class file (that is, no `name.class`). Ensure that your class names, method names and parameters (for auto-testing), and your package name are exactly as in the template classes provided. Submit your work by the deadline which is 11.45 pm on Sunday 6 November.