CS21120 Assignment 1

Competition Management System

Release date 3rd November 2014

Hand in date 24th November 2014 (23:59 via Blackboard)

Feedback date 12th January 2015 (via Blackboard)

Aims and Objectives

The aim of this assignment is to give you experience in writing and using some basic data structures.

The objective is to write a system for managing competition matches with a variety of possible rules.

Overview

Many competitions involve matches between pairs of players e.g. Wimbledon tennis championships, the football world cup, world chess championships, etc. Competitions such as Wimbledon use a single elimination style of competition, where pairs of players play each other and the loser leaves the competition and the winner goes on to the next round until a winner is decided. Assuming each player always plays to their ability, such a competition is guaranteed to find the correct winner. The single elimination style of competition doesn't provide much information about relative ability of other players. For example the second best person in the competition could go out to the eventual winner in the first round, or the other finalist could be half way down the ranking if they had a particularly lucky draw. Other systems have been developed to overcome these problems, such as a league or round robin, where every team plays every other team, or various forms of *repechage*, where losers are allowed to carry on in some way with a second chance to win places.

In this assignment you are asked to develop a system for managing such competitions, with the precise style of competition implemented by a class that manages matches between competitors. The system should read a list of players (e.g. individuals or teams) from a text file (one per line). It should then print out the players for each match and request user input to indicate the winner. The system should then print out the winner and any runner up places the system has determined. The result for a match is in the form of an integer numerical score and there should always be a winner (no draws). The system should warn the user if some incorrect input (e.g. not a pair of numbers or a draw) is entered and request input again. You have been provided with a basic command line program for this. For simplicity the competition manager doesn't need to worry about matches happening at the same time, or publishing the full draw before the start of the competition.

Requirements

You need to implement 3 versions of the provided *IManager* interface. Each must be defined in the package *uk.ac.aber.dcs.<userid>.cs21120.assignment1* where <userid> should be replaced with your user ID (email). You must use the class names specified i.e. *SingleElimination*, *DoubleElimination* and *BubbleElimination*.

1) *SingleElimination* (20%): This should be implemented using a queue. All players are placed in the queue and pairs of players are taken from the front of the queue for each match. The

winner is placed on the back of the queue and the loser is discarded. The *getWinner* method only needs to return the winner if asked for place 0 after the competition.

2) *DoubleElimination* (20%): In this style of competition players are divided into two groups, a winner's bracket (which acts like a single elimination competition) and a losers' bracket, which losers from the winners' bracket are gradually put into throughout the competition. This should be implemented using two queues. All players are placed on the "winners' bracket" queue at the start. For each match, if the winners' queue is longer then take the match from the winners' queue, otherwise take it from the losers' queue. If the Match was taken from the winners' queue, the winner is placed on the back of the queue and the loser goes onto the back of the losers' queue. If the match is taken from the losers' queue the winner goes on to the back of the queue and the loser is discarded. This goes on until there is only one player in each queue and they play for the final. Both the winner (winner 0) and first runner up (winner 1) should be reported in the *getWinner* method following the competition.
3) *BubbleElimination* (20%): This competition should behave like a binary heap-based priority queue, which can be implemented in an array. Each player enters the queue at the end of the queue array and plays their "parent" in the competition tree. If the new player loses they stay in their current slot, if they win they swap places with the loser. This is repeated until either the player loses or reaches the front of the queue. Once all competitors have been through this process the competition's overall winner is at the front of the queue. To select other places the winner is removed and replaced with the last competitor in the heap array. This player should play its new left child, and the winner of that match should play the right child. If the winner of the second match is the inserted player then stop. Otherwise the winner of the second match is swapped with the inserted player and the process is repeated with its two new children. This is continued until the bottom of the tree, or until the inserted player beats both its new children in the heap. This process can be repeated to get the other positions (third place, fourth place and so on), which should be saved in an array for reporting if requested by the *getWinner* method. In the heap the "children" of array element *i* are elements *2i+1* and *2i+2*, hence the parent of element *i* is *(i-1)/2* (using integer divide). In the first stages of the competition the competitors use the heap *bubble up* operation. After all the competitors have been entered and the first place has been decided, the heap *bubble down* operation is used to select the other places and reorder the heap.

You are also asked to provide the following:

1) Testing (20%): You should provide *JUnit* tests for each of the implementations described above. These should be in classes called *SingleEliminationTest*, *DoubleEliminationTest* and *BubbleEliminationTest* defined in the package *uk.ac.aber.dcs.<userid>.cs21120.assignment1.* These *must* only test each of the public methods of the *IManager* interface. It is important that you use the provided factory class to generate all instances of your classes and each class should have a no argument constructor.
2) Report (20%): 2 sides of A4. First page should include the title, your student ID number and module code and a brief description of what you have done, focusing on what you have managed to achieve and what hasn't worked or not been attempted / completed. The second page should show the output of your unit testing e.g. as screenshots.

Marking

This assignment will use an element of automated marking. I will apply my own *JUnit* tests to your code, I will also apply your *JUnit* tests to both your own and other classes, so it is important that you follow the specification exactly. In particular the naming and packaging of your classes and the use of the provided factory class to generate instances of your class (or other class) for testing are essential. Also, do not modify any of the supplied code, including package name. If you find any bugs please report them.

Resources

For this assignment you are not permitted to use classes from the java collections classes (java.util.* package) *except* for *ArrayList* and *Scanner*. For the rest you must implement your own classes. You should not use Threads in your code. You may use material from the lectures / course, but it is your responsibility to check it and correct if needed (but please report any bugs you find). Any resources you use (Including these) must be acknowledged in your report. You are provided with a simple test program, the *Match* class, the *CompetitionManager* class, the *IManagerFactory* class, some example data and the *IManager* interface. The factory method provided should be supplied with a String giving the full name of the class (i.e. including package), which suggests the class to use. This behaviour may be modified when your code is tested.

Academic conduct

As with all such assignments, the work must be your own. Do not share code with your colleagues and ensure that your own code is securely stored and not accessible by your classmates. Any sources you use should be properly credited with a citation, and any help you get (e.g. from demonstrators) should be acknowledged. Your report must accurately reflect what you have achieved with your code, any discrepancies between your code and report could be treated as academic fraud.

Bernie Tiddeman

31st October 2014