**CS 210**

**Lab 10: Observer Pattern**

**Lab Date:** Wednesday, April 8.

**Due Date**: Sunday, April 19, 11:59pm

**Submission**

* This is an **individual work** lab.
* Submit all source code in a single zip file to the Lab 10 dropbox on Kodiak.
* Be sure to **comment your code using Javadoc style comments**.

**Goal:** Apply the Observer Pattern to small applications

**Summary**

Consider a competition or tournament. At any given time, people may want to see the overall rankings in the competition. Or perhaps, rather than searching through a set of rankings, they want notifications only after their favorite team has played a match in the competition. You will apply the Observer pattern to create these two applications.

**The Competition**

The competition that we will be working with is made up of several matches. Each match features an alliance of 3 teams facing off with another alliance of 3 teams. We will call these the Red and Blue alliances. Alliances change every match. Each match ends in a score for each alliance. The alliance with the higher score wins the match, and it is possible for alliances to play to a draw.

In addition to the scoring, there are two achievements that are available to each alliance: “Light the Lamp” and “Return to Base”. Note that a team can win without completing either achievement and/or a team can lose having completed both.

In the competition, teams are ranked by average ranking points per match. Ranking points (RP) are earned as follows:

* A team earns 2 RP if its alliance wins a match, 1 RP if its alliance draws a match, or 0 RP if its alliance lost a match.
* A team earns 1 RP if its alliance completed the “Light the Lamp” achievement, regardless of the match result (win/loss/draw).
* A team earns 1 RP if its alliance completed the “Return to Base” achievement, regardless of the match result (win/loss/draw).

If teams are tied based on average ranking points, then ties are broken based on average match score.

You have been given two data files. The first is teams.txt, which is a list of teams involved in the competition. Each team is simply an integer in the range [1,9999]. The second is competition.csv where each row represents a match, and each row is formatted as follows:

MatchNum, RedTeam1, RedTeam2, RedTeam3, BlueTeam1, BlueTeam2, BlueTeam3, RedScore, BlueScore, RedLamp, BlueLamp, RedBase, BlueBase

* MatchNum is a unique positive integer representing a particular match
* RedTeam1, RedTeam2, RedTeam3, BlueTeam1, BlueTeam2, and BlueTeam3 are the six teams involved in the match. The first three teams are in the Red alliance, and the next three are in the Blue alliance. Each entry is one of the integers that is in the teams.txt data file. You can also assume that all six teams are unique. (Teams aren’t on an alliance twice, nor are they on the Red and Blue alliances in the same match.)
* RedScore and BlueScore are the match scores for the Red alliance and Blue alliance, respectively. They are each integers greater than or equal to zero.
* RedLamp and BlueLamp indicate whether the Red alliance and Blue alliance, respectively, completed the “Light the Lamp” achievement. If they did, the entry contains True. If not, it contains False.
* RedBase and BlueBase indicate whether the Red alliance and Blue alliance, respectively, completed the “Return to Base” achievement. If they did, the entry contains True. If not, it contains False.

1,11,22,33,44,55,66,71,42,True,True,False,False  
2,11,77,88,99,101,202,80,90,False,False,False,False

In the example above, Teams 11, 22, and 33 earned 3 RP from the first match (2 from winning, one from the Lamp achievement) while Teams 44, 55, and 66 earned 1 RP (Lamp achievement). In the second match, Team 11 joined up with Teams 77 and 88 and lost while completing no achievements. The alliance of Teams 99, 101, and 202 won the match, but also failed to complete any achievements.

After those 2 matches, teams 22 and 33 lead the rankings (3 RP/1 match = 3), followed by teams 99, 101, and 202 (2 RP/1 match = 2). Team 11 is ranked after them (3 RP/2 matches = 1.50). Teams 44, 55, and 66 have (1 RP/1 match = 1). Teams 77 and 88 have (0 RP/1 match = 0).

Create Competition.java, a class that can read in these data files and updates team rankings after each match. The constructor should be given the names of these two data files as arguments. You have also been given skeleton code for Team and Match classes, which you should add to.

**The Applications**

Our first application – RankingsDisplay – displays the rankings (as text) and should print updated rankings after each match via the update method. When constructed, the application should be provided an integer that represents the number of teams to show. For example, if given 8, the application will show the Top 8 teams in the rankings after each match.

Using our example above, and using Top 3, we would expect to see:

AFTER MATCH 1:  
Team 11 3.00 RP  
Team 22 3.00 RP  
Team 33 3.00 RP  
AFTER MATCH 2:  
Team 22 3.00 RP  
Team 33 3.00 RP  
Team 99 2.00 RP

The second application – TeamDisplay – is given a team number and displays the results of that team’s matches and its ranking at the conclusion of those matches via the update method.

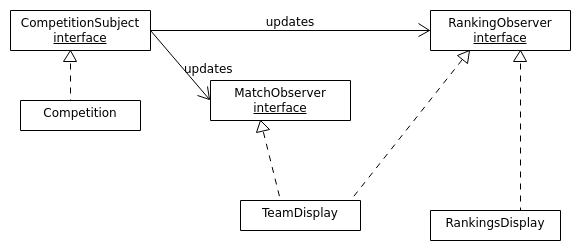
Using our example above for Team 99, we would expect to see:

Team 99 won Match 2 90-80. Current rank: 3

You have been given three interfaces to help you: CompetitionSubject, RankingsObserver, and MatchObserver. RankingsObservers get updated rankings, while MatchObservers get told about every match. Note that a class could implement either or both observer interfaces.

Update the Competition class to be a subject. Create RankingsDisplay and TeamDisplay as observer classes. Note that this means that each should implement the appropriate interfaces. Create Lab10Driver which should create one instance of RankingsDisplay and two instances of TeamDisplay. The driver class should allow a user to input the number of teams to show in the RankingDisplay and the teams to monitor for the two TeamDisplays.

Below is a partial UML diagram for the lab:



**What to Submit:**

* A zip file containing the final versions of your source files (.java files), as well as any necessary source files given to you.