COMP 1020
Lab 3

MATERIAL COVERED

- Objects which refer to other objects

Notes:

- Make sure your TA has recorded your mark before leaving.
- The three questions are sequential – each builds on the previous one.
- Only one of the three exercises is required, but try to do two. The Gold exercise is more difficult, as usual.
- The Gold exercise will require more lines of code, and will probably be impractical to complete in a one hour lab. Try to complete it at home as an extra exercise instead. You will learn a lot from it if you do.

Supplied code

This lab builds on a modified version of the **`HockeyTeam`** class from Lab 2. It has been split into two classes (**`HockeyTeam`** and **`TeamList`**), which is the correct way to do it. (It was done in one class in Lab 2 only to give practice with class/static variables.) A few additional methods have also been provided. Download the **`HockeyTeam.java`** and **`TeamList.java`** files which provide two classes with the following methods:

`HockeyTeam` methods:

- **`HockeyTeam(String name, int wins, int losses, int overtimeLosses)`** – constructor
- **`void won()`** – record a win for the team
- **`void lost()`** – record a loss for the team
- **`void lostOvertime()`** – record an overtime loss for the team
- **`int points()`** – find the number of points that the team has in the standings
- **`String toString()`** – obtain a printable String for the team
- **`String getName()`** – find the name of the team
- **`int compareTo(HockeyTeam)`** – compare the points for two teams

`TeamList` methods:

- **`TeamList()`** – constructor (makes an empty list of teams)
- **`void addTeam(HockeyTeam)`** – add a team to the end of the list
- **`String toString()`** – obtain a multi-line printable String for the list of teams
- **`void SortTeams()`** – Sort the teams into descending order by points
- **`HockeyTeam findTeam(String name)`** – find a team by name

Background information on winning, losing, and overtime: For those unfamiliar with NHL games, it works like this. After the third period (the normal end of the game), if the score isn't tied the game is over – one team gets a win (2 points), the other gets a loss (0 points). If the score is tied, then it becomes an overtime game and they keep playing. Eventually one team will be credited with a goal, and get a win (2 points), and the other team gets an "overtime loss" (1 point).

A Game class

---

1. Create a file **`Game.java`** which will implement a class of objects that will store some information about one hockey game. In your file, implement the following variables and methods.

    a. **private** instance variables that will store references to two **`HockeyTeam`** objects representing the teams that are playing in this game (the "home" team and the "away" team).

    b. **private** instance variables that will store the current score (one **`int`** per team).

    c. a constructor that will accept two **`HockeyTeam`** parameters, "home" team first. The score should be initialized to 0-0.

    d. the usual **public** method **`String toString()`** which will return a **`String`** giving the status of the game, in the format **`"Winnipeg(2) at Chicago(1)"`** where the "away" team is listed first and the "home" team is listed second, and the score is given as shown. Note that only the team names are shown, not the records of the teams.

    e. a public method **`void goal(HockeyTeam)`** which will update the score due to a goal being scored by the indicated team. If that team isn't one of the two that are playing in this game, it should use **`System.out.println`** to print the error message **`"That team isn't playing in this game!"`**

2. Run the supplied file **`TestLab3Bronze.java`** to test your **`Game`** class. Look at this file to see how it is using your `Game` objects. If your class is working correctly the output should be:

```
Creating Chicago at Winnipeg game.
Chicago(0) at Winnipeg(0)
Creating Colorado at St. Louis game.
St. Louis(0) at Colorado(0)
Chicago scores
Chicago(1) at Winnipeg(0)
Winnipeg scores
Chicago(1) at Winnipeg(1)
St. Louis scores
St. Louis(1) at Colorado(0)
Colorado scores
St. Louis(1) at Colorado(1)
Chicago scores in the wrong game:
That team isn't playing in this game!
St. Louis scores in the wrong game:
That team isn't playing in this game!
Winnipeg scores
Chicago(1) at Winnipeg(2)
```

Ending games and adjusting standings

---

1. Save your **`Game.java`** file as **`GameV2.java`**. Change the name of the class and the constructor to match. The supplied main program will expect games to be named this way for the Silver exercise.

2. Modify your **`GameV2`** class by adding the following additional instance variables and methods:

    a. **private** **`boolean`** instance variables to keep track of whether a game has ended, and whether it went into overtime (regardless of whether the game is over or not).

    b. a **public** method **`void overtime()`** which will change the game to show that it has gone into overtime. You do not have to do any error checking. (In a real situation, you would check to see that the game was tied, and wasn't already over before this.)

    c. a **public** method **`void ended()`** which will change the game to indicate that the game has ended. (Again, you do not have to do any error checking. Assume that the game hadn't ended before, and isn't tied.) In addition to modifying the object appropriately, this method should send the appropriate **`won()`**, **`lost()`**, or **`lostOvertime()`** messages to the two **`HockeyTeam`** objects, so that they properly adjust their records in the standings.

    d. Make appropriate modifications to the existing methods as follows.

        i. Modify the constructor so that the new game is not over, and not in overtime.

        ii. Modify the **`toString`** method so that it adds **`"OT"`** or **`"Final"`** or both to the end of the **`String`**, to indicate that the game went into overtime, is over, or both. See the sample output below.

        iii. Modify the **`goal`** method so that it automatically ends the game if the goal was scored in overtime.

3. Run the **`TestLab3Silver.java`** file to test your modified class. If your **`GameV2`** class is working properly, the output should be:

```
Standings:
Chicago(28,13,2=58)
St. Louis(27,13,3=57)
Winnipeg(22,14,8=52)
Colorado(18,17,8=44)

Creating Chicago at Winnipeg game.
Creating Colorado at St. Louis game.

After goals are scored:
Chicago(1) at Winnipeg(2)
St. Louis(1) at Colorado(1)

Winnipeg-Chicago game ends
Chicago(1) at Winnipeg(2) Final
St. Louis-Colorado game goes into overtime
St. Louis(1) at Colorado(1) OT
St. Louis scores
St. Louis(2) at Colorado(1) OT Final

Standings:
St. Louis(28,13,3=59)
Chicago(28,14,2=58)
Winnipeg(23,14,8=54)
Colorado(18,17,9=45)
```

Increasing object interactions

In this exercise, you will not create any new classes, but you will modify all of the existing classes (**`HockeyTeam`**, **`GameV2`**, and **`TeamList`**) in order to implement the following method.

1. Add a **`HockeyTeam bestBet()`** method to the **`TeamList`** class. This method is looking for the team that is most likely to win its game. This method should look at all of the teams in the list, and return the one which 1) is currently playing a game, which 2) is not over yet, and 3) is farther ahead of its opponent in the ***standings*** than any other team that is currently playing. The current score in the games doesn't matter, only the points difference in the standings. If no teams at all are playing right now, return **`null`**. If two teams are equally far ahead of their opponent in the standings, return either one of them.

2. This cannot be done without adding additional instance variables and/or methods to the existing classes. Add whatever you need to solve the problem. It's up to you to decide what you need. This is a Gold exercise, after all. There will have to be some additional interactions between the three classes.

3. The rules:
    a. You cannot create any other classes.
    b. You cannot use an additional list or array of games anywhere. The only list should be the existing list of teams.
    c. You cannot add any class/static variables or methods, only instance variables or methods.
    d. Good luck.

4. Run the **`TestLab3Gold.java`** file to test your modified classes. If everything is working properly, the output should be:

```
Standings:
Nashville(29,9,4=62)
Chicago(28,13,2=58)
St. Louis(27,13,3=57)
Winnipeg(22,14,8=52)
Dallas(19,16,7=45)
Colorado(18,17,8=44)
Minnesota(18,19,5=41)

Test 1: Best Bet now should be null - there are no games.
league.bestBet() gives:null

Creating Nashville at Winnipeg game.
Creating Minnesota at St. Louis game.
Creating Dallas at Colorado game.

Test 2: Best Bet now should be St. Louis (16 points better)
league.bestBet() gives:St. Louis(27,13,3=57)

Minnesota(0) at St. Louis(1) Final
Test 3: Best Bet now should be Nashville (10 points better)
league.bestBet() gives:Nashville(29,9,4=62)

Dallas(0) at Colorado(1) Final
Nashville(0) at Winnipeg(1) Final
Test 4: No active games. Best Bet now should be null
league.bestBet() gives:null
```