CS 101 - Algorithms & Programming I

Fall 2021 - Lab 7

Due: Week of November 29, 2021

*Remember the **honor code** for your programming assignments.*
*For all labs, your solutions must conform to the CS101 style **guidelines**!*
All data and results should be stored in variables (or constants where appropriate) with meaningful names.

The objective of this lab is to learn how to define custom classes and use them. Remember that analyzing your problems and designing them on a piece of paper *before* starting implementation/coding is always a best practice. Specifically for this lab, you are to both organize your data and methods working on them.

0. Setup Workspace

Start VSC and open the previously created workspace named `labs_ws`. Now, under the `labs` folder, create a new folder named `lab7`.

In this lab, you are to create Java classes/files (under `labs/lab7` folder) as described below. We expect you to submit a total of 3 files including: the `Player` class, a revised version of the `Player` class and a new class required by the revision (only 2 for Wednesday's section as they do not need to change the `Player` class). Separate files containing the main method for the original assignment and the revisions are already provided for testing purposes. Do *not* upload the provided files or other/previous lab solutions in your submission. Outputs of sample runs are shown as brown.

1. NBA Players

This year, ESPN president James Pitaro decided that they need a well-defined and informative representation for NBA players, for the sake of presenting statistics about players during the game. For this task, the commissioner of NBA, Adam Silver, selected you as an ambitious programmer. You will complete this task as a part of this lab assignment.

Create a new/empty file to define a class named `Player`. Your implementation will provide a simplified definition for an NBA player, which has the following data members:

Static Data Members:

- `count`: Counts the number of players actively playing in the league.
- `allCount`: Number of players registered to the league from the beginning. (**Hint:** You can use this to assign a unique id to a player)

Instance Data Members:

- `id`: An identification number that is unique for each player.
- `name`: Stores the name of the player.
- `position`: Stores the current position of the player in a starting five.
- `team`: The team that the player plays for. In this simplified definition, you may assume that teams are defined with a sequence of three letters (e.g., "LAL" for Los Angeles Lakers and "NYC" for New York Knicks).
- `rosterNumber`: The jersey number that the player currently wears.
- `isActive`: Data that holds whether the player is still actively playing basketball or not.

Methods:

- Constructor:
    - Registers the player to the system.
    - Takes `name`, `position`, `team` and `rosterNumber.`
    - Initializes the data members `name`, `position`, `team` and `rosterNumber.`
    - Assigns an id as `id`.
    - Modifies `count` accordingly (using `register` method).
- Accessor methods:
    - You need to implement methods to access the data variables `name`, `id`, `position`, `team` that the player plays for, `isActive` as the activity status of the player and the `rosterNumber`.
- Setter methods:
    - `setName`: Sets the name of the player to the parameter passed.
    - `setPosition`: Changes the position that the player plays to parameter passed.
    - `setRosterNumber`: Changes the jersey number that the player wears during the game.
    - `register`: Registers a player to the league and modifies `count, allCount` accordingly.
    - `remove`: Removes a player from the players registered.
    - `retire`: Ends the active career of a player. You will need to modify `isActive` and `count` to simulate this.
    - `transfer`: Changes the team that the player plays for. This requires a check on whether the team change is redundant (i.e., already in that team). If the function call does not change the team that the player plays, output an error message. A sample message is provided in sample runs.
- Service Methods:
    - `isBeforeThan`: To be able to report the player names in alphabetical order, a comparison method is required. This method takes another instance of a `Player` class and returns true if the name of the current player (`this.name`) is "smaller" (i.e., lexicographically before) than the name of the player inputted. Otherwise, this method returns false as a boolean. (**Hint:** You can use the `String.compareTo` method for this purpose). A sample run for this method is provided below.

Note: While changing the value of `position`, you need to check whether the parameter passed is valid. Valid player positions are: SF (small forward), PF (power forward), SG (shooting guard), PG (point guard) and C (center). You may also implement a helper method for this purpose.

Sample run:

```
name = "LeBron James";
position = "PF";
team = "CLE";
rosterNumber = 23;
Player player1 = new Player(name, position, team, rosterNumber);
player1.setRosterNumber(6);
player1.transfer("LAL");
player1.setPosition("SF");

name = "Ron Artest";
position = "SF";
team = "LAL";
rosterNumber = 37;
Player player2 = new Player(name, position, team, rosterNumber);
player2.setName("Metta World Peace");
player2.retire();

name = "Stephen Curry";
position = "SG";
team = "GSW";
rosterNumber = 30;
Player player3 = new Player(name, position, team, rosterNumber);

System.out.println(player1.isBeforeThan(player3))

true // Since L < S lexicographically
```

2. Player Statistics

Now that the basic functionality of the `Player` class is complete, you are going to add game statistics to your implementation. During basketball games, broadcasting services usually provide game averages for players (like average points scored). To implement this, you will include some data members to your class and provide service methods accordingly. In your class definition you need to add the following members for this functionality:

Instance Data Members:

- `gamesPlayed`: Number of games that the player participated by playing at least one minute.
- `pointsPerGame`: Points averaged per game for the player.
- `assistsPerGame`: Assists averaged per game for the player.
- `reboundsPerGame`: Rebounds averaged per game for the player.
- `turnoversPerGame`: Turnovers averaged per game for the player.

Service Methods:

- `recordGame`: Takes number of points, assists, rebounds and turnovers for the game to be recorded and updates per game statistics defined above accordingly.

- `toString`: Returns a string representation that contains all the information stored about the player. A sample run is provided below.

Sample run:

```
player1.recordGame(30, 10, 4, 4);
player1.recordGame(15, 8, 7, 2);
player1.recordGame(26, 8, 3, 7);
player1.recordGame(19, 6, 6, 3);
player1.recordGame(25, 5, 2, 5);
player1.recordGame(34, 5, 11, 4);

System.out.println(player1.toString());

**************************************************
Player ID: 1
Player Name: LeBron James
Player Position: SF
Current Team: LAL
Jersey Number: 6
--------------------------------------------------
Stats:
PPG: 24.83
APG: 7.00
RPG: 5.50
TPG: 5.50
--------------------------------------------------
Retirement Status: Active among 2 active player(s)
**************************************************
```

3. The MEP Ladder

With the decision of the commissioner of NBA, Adam Silver, this year a trophy based on player efficiency will be awarded to one of the players (Most Efficient Player award). For this, the league defined an efficiency score to be able to compare players quantitatively. The details of the metric is as follows:

$$E = 1.0 \times PPG + 0.7 \times APG + 0.7 \times RPG - 0.9 \times TPG$$

PPG: Points per game

APG: Assists per game

RPG: Rebounds per game

TPG: Turnovers per game

Using this metric, now you will modify the `toString` method that you previously defined so that it now includes the efficiency score E also. In addition, you will implement another service method `isMoreEfficient` that compares two players with respect to their efficiency scores. This method takes only one argument which is a player to compare. As this is going to be an instance method for `Player` class, this method will compare the current player (`this`) and the `Player` instance that `isMoreEfficient` method takes. (**Tip:** You can implement a helper method to calculate a player's efficiency)

In the sample run, keep in mind that we used the call `player2.retirePlayer()` to modify the number of active players, which is a desired functionality.

Sample usage/output:

```
name = "Kevin Durant";
position = "PF";
team = "BKN";
rosterNumber = 7;
Player comparedPlayer = new Player(name, position, team, rosterNumber);
comparedPlayer.recordGame(32, 4, 11, 1);
comparedPlayer.recordGame(29, 12, 15, 2);
comparedPlayer.recordGame(38, 3, 5, 2);
comparedPlayer.recordGame(25, 4, 8, 1);
comparedPlayer.recordGame(25, 2, 11, 5);
System.out.println(player1.toString())
**************************************************
Player ID: 1
Player Name: LeBron James
Player Position: SF
Current Team: LAL
Jersey Number: 6
--------------------------------------------------
Stats:
PPG: 24.83
APG: 7.00
RPG: 5.50
TPG: 5.50
Efficiency: 28.63
--------------------------------------------------
Retirement Status: Active among 3 active player(s)
**************************************************
System.out.println(comparedPlayer.toString())
**************************************************
Player ID: 4
Player Name: Kevin Durant
Player Position: PF
Current Team: BKN
Jersey Number: 7
--------------------------------------------------
Stats:
PPG: 29.80
APG: 5.00
RPG: 10.00
TPG: 10.00
Efficiency: 31.30
--------------------------------------------------
```

```
Retirement Status: Active among 3 active player(s)
**************************************************

On System.out.println(player1.isMoreEfficient(player3))
false
```

To test your player class, we provided you with a file named `Lab7_Main.java`. This file consists of three static methods where each will test one part (`part{1,2,3}Test()`). Keep in mind that your implementation should be compatible with the provided implementation. For the desired outputs, you need to stay consistent with the details described by this document.