ASSIGNMENT 1: Introduction to Object-Oriented Programming
COMP 1020 Winter 2022

Due Date: March 4, 2022 at 11:59pm

Instructions

- You must complete the "**Blanket Honesty Declaration**" checklist on the course website before you can submit any assignment. You will not be able to see the Assignment submission link if you have not accepted the Honesty Declaration.
- Only submit the **java files**. Do ***not*** submit any other files, unless otherwise instructed.
- To submit the assignment, upload the specified files to the **Assignment 1** folder on the course website.
- Assignments must follow the **programming standards** document published on UMLearn.
- After the due date and time, assignments may be submitted but will be subject to a late penalty. Please see the ROASS document published on UMLearn for the course policy for late submissions.
- If you make multiple submissions, only the **most recent version** will be marked.
- These assignments are your chance to learn the material for the exams. Code your assignments independently. We use software to compare all submitted assignments to each other, and **pursue academic dishonesty vigorously**.
- Your Java programs must compile and run upon download, without requiring any modifications.

Assignment overview

In this assignment, you will implement a set of related classes of objects:

- **Player**: A Laser Tag player
- **Team**: A team of Laser Tag players
- **Game**: A game of Laser Tag between two teams
- **Location**: A Laser Tag venue, which Players can be members of

Keep all of your methods short and simple. In a properly-written object-oriented program, the work is distributed among many small methods, which call each other. There are no methods in this entire assignment that should require more than 10 lines of code, not counting comments, blank lines, or {} lines (and many of them need no more than 3).

Unless specified otherwise, all instance variables must be **`private`**, and all methods should be **`public`**. Also, unless specified otherwise, there should be ***no*** **println** statements in your classes. Objects usually do not print anything, they only return **String** values from **toString** methods. **println** in your objects will result in a lost of marks.

Your code should be **DRY – don't repeat yourself**. If you have a method that already accomplishes a task, or has some logic, that logic should be used. You may add any **`private`** methods you see fit to help accomplish this task.

The description of the assignment has methods that are required. There are suggested names for the parameters, which you may change. **Do not change the signatures of the methods**. **Your classes and objects should work with the provided test files.**

You may not use an ArrayList in this assignment. You *may* write your own collection class, but that is not the expectation for this assignment.

Testing Your Coding

We have provided sample test files for each phase to help you see if your code is working according to the assignment specifications. These files are starting points for testing your code. Part of developing your skills as a programmer is to think through additional important test cases and to write your own code to test these cases. Different, more vigorous tests will be used to test your program.

ASSIGNMENT 1: Introduction to Object-Oriented Programming
COMP 1020 Winter 2022

Sample output is provided in some cases, for you to match as closely as possible. Other output is left out, for you to practice reading code, understand a program, and practice seeing if output is correct or not based on inputs.

Phase 1: Class Player

This class represents a Laser Tag player in our system. The class tracks the user's name, how many games they have played, and can calculate their average score. You will need to make instance variables to keep this information in your objects.

Class player requires the following methods:

- A constructor public `Player(String name)`. Initialize other instance variables as required.
- Method `public void addGameResult(int newScore)`. This method is called after a game is played, the value passed is added to the user's average that is being tracked by this object.
- `public boolean equals(Player other)`, which checks equality of two Player objects. Two Player Objects are equal if the names of the two Players are the same.
- A `toString` method that prints out information of the object. If the player has not played any games, print that the player has not played any games. If they player has played some games, print out the player's game average score. Use integer division to calculate the average. See the sample output, and match it as closely as possible.
- Accessor method `getName()`, which returns the Player's name.

Sample output:

```
Should print "Player: Jim Beam has played no games"
Player: Jim Beam has played no games
Should print "Player: Johnny Walker has played no games"
Player: Johnny Walker has played no games
Should print "Player: Jim Beam has played 1 game with an average of
50"
Player: Jim Beam has played 1 game with an average of 50
Should print "Player: Jim Beam has played 2 games with an average of
51"
Player: Jim Beam has played 2 games with an average of 51
Should print "Player: Johnny Walker has played 50 games with an
average of 25"
Player: Johnny Walker has played 50 games with an average of 25
true
false
Hello, my name is Jim Beam
```

Phase 2: Class Team

Create class Team, which is a collection of Players who play Laser Tag games. Teams have maximum sizes, and players that come and go from the team. Use a partially-filled array to keep track of the players on the team.

- Create a constant to track the maximum size of the team. Set the constant to 5.
- Add a constructor, that has no parameters, if you need one.
- Add method `public boolean addMember(Player newOne)`, which adds a player to the team. A player can be added to a team if: the player is not already on the team and there is space left in the team. Players can be a part of more than one team. Return true if the player was added to the team, false if the player was not added to the team.
- Add method `public String toString()`, which prints out the team roster. See sample output for the format required.

- Add method `public boolean hasCommonPlayers(Team other)`, which returns true if there is a player that is on the roster of both teams.
- Method `public boolean teamFull()`, which returns true if the team has a full complement of members. Use the constant you used for maximum size of the team to determine this.
- Method `public boolean removePlayer(Player toRemove)`, which removes a player from the team. Return true if the player was removed, false if the player was not removed (the player was not on the team).
- Method public Player[] getRoster() which returns a **deep copy** of the team. The returned array should be the same size as the number of players. For instance, if there is only 1 member on the team, the returned array should return an array of size 1. You do not have to do a **recursive deep copy**, that is, you do not ~~have to~~ deep copy the Player objects – just the array.

Sample output:

```
Team members:
1: Ashley Johnson

Team members:
1: Ashley Johnson
2: Laura Bailey

Team members:
1: Ashley Johnson
2: Laura Bailey

Team members:
1: Ashley Johnson
2: Laura Bailey
3: Liam O'Brien
4: Marisha Ray

false
Team members:
1: Ashley Johnson
2: Laura Bailey
3: Liam O'Brien
4: Marisha Ray
5: Sam Riegel

true
true
Team members:
1: Ashley Johnson
2: Laura Bailey
3: Liam O'Brien
4: Marisha Ray
5: Sam Riegel

true
Is there common members?
false
Is there common members now?
true
false
```

Phase 3: Class Game

Create class Game, which is a Laser Tag game between two teams. To accomplish this:

- Add a constructor that has 2 Team parameters, one for each team that is participating in this game.
- Add method `public boolean validGame()` that checks to see if these two teams can participate in a game. Teams can play each other if: There are not members that are participants in both teams, and both teams are full (no more members can be added).
- `public void awardWinner(Team winner, int points)`, which gives points to the team members of the winning team. Points are divided equally between the team members. Round down if the points do not divide evenly to the number of players. To do this, you may add methods to the Game class. Consider where the code for this task should exist for maximum cohesion in your objects. Teams that did not win get 0 points awarded. You may assume that '`winner`' will be passed a team that is part of this game.
- A `toString` method that prints the team rosters side-by-side. Print "Not a valid game" if the game is not valid (as defined above). Otherwise, print the team rosters side-by-side. The right column must be aligned properly. Example:

```
Team rosters:
Ashley Johnson      Chris Straub
Laura Bailey        Amy Falcone
Liam O'Brien        Kate Welch
Sam Riegel          Ryan Hartman
Taliesin Jaffe      Jerry Holkins
```

Phase 4: Class Location

Games happen at Laser Tag locations. Create a Location class, that represents a location where Laser Tag games happen. Locations have games, and members.

In your Location class tracks a list of members (use a partially-filled array). Add the following methods:

- Constructor with two parameters: String name, String address, which are stored for use in the toString method.
- `public void addMember(Player newbie)` which adds a member to the membership list that is tracked by this object. Players can only hold one membership per Location, that is, the same player should not show up multiple times in the toString method.
- `public String toString()` which prints the name and address of the location, then prints out all the names, number of games played, and average score of all the players that belong to this Laser Tag location. Consider any methods you have already written to help you do this.
    - Sample output:

```
Membership list for LaserTopia, Waverley Ave location
Player: Player: Jerry Holkins has played 1 game with an average
of 20
Player: Travis Willingham has played no games
```

Also, add a second constructor to class Player, which accepts 2 parameters: String name, Location place. When given a Location, the constructor the Player becomes a member of the given Location.

Finally, add static data and methods to Location:

- A partially filled array, that tracks all of the locations
- public static String allLocationNames() which returns a String of all the location names

- public static String whichLocations(Player who) which returns a string of all the Locations that the Player 'who' is a member of.
    - Sample output:

```
Orion Acaba is a member at:
Darkzone
LaserTopia
```

Hand in

Submit your four Java files (**`Player.java`**, **`Game.java`**, **`Location.java`**, **`Team.java`**). ***Do not submit .class or .java~ files!*** You do ***not*** need to submit the **`TestPhaseN.java`** files that were given to you. If you did not complete all four phases of the assignment, **use the Comments field when you hand in the assignment to tell the marker which phases were completed**, so that only the appropriate tests can be run. For example, if you say that you completed Phases 1-2, then the marker will compile your files and also TestPhase2.java, and then run the main method in TestPhase2. If it fails to compile and run, you will lose ***all*** of the marks for the test runs. The marker will ***not*** try to run anything else, and will ***not*** edit your files in any way. ***Make sure none of your files specify a package at the top (edit this out before submission if necessary)!***