# Assignment 08 – Volleyball

You must work in **alone** on this assignment. Do not use any Java language features we have not cover so far in this course.

## Assignment Objectives

After completing this assignment the student should be able to:

- Implement a class according to given specifications
- Implement private instance variables
- Implement constructor methods
- Implement getter and setter methods
- Read input from files

## Assignment Requirements

For this assignment you are given the following files:

| | |
|---|---|
| **Assignment08.java** | (make no changes to this file) |
| **Roster1.dat** | (make no changes to this file) |
| **Roster2.dat** | (make no changes to this file) |
| **Player.java** | (you must complete this file) |
| **Roster.java** | (you must complete this file) |

## Problem Description and Given Info

The volleyball coach at Verde Valley High School would like some help managing her team. She would like a program to help her identify the best players. She has team rosters stored in text files that contain the names of her players (first name, then last name separated by a space), and their stats as attacks per set (a double) followed by blocks per set (a double). Higher stats are better. Each data field is separated by a space. For example, one line in a roster file would look like:

```
Gabrielle Reece 4.57 1.79
```

The coach would like the program to do the following:

Present a menu with the following options:

```
1. Open a roster file
2. List all players
3. List top attackers
4. List top blockers
5. Add a player
6. Change a player's stats
7. Count players
8. Quit program
```

When the user chooses 1 to open a roster file, then the program will ask for the filename of a roster file, then open and read the data from that file into an **ArrayList**.

When the user chooses 2 to list all players, then the program will list the names and stats of all player players.

When the user chooses 3 to list top attackers, then the program will determine and list the names and stats of the players with the top 2 attack stats.

When the user chooses 4 to list top blockers, then the program will determine and list the names and stats of the players with the top 2 stats for blocks.

When the user chooses 5 to add a player, then the program will prompt the user to enter the new player's name (first and last), attack stat (**double**), and block stat (**double**). The program should collect this information from the user, and then instantiate a new **Player** object with the given name and stats, and add that Player to the roster.

When the user chooses 6 to change a player's stats, then the program will prompt the user to enter the player's name (first and last). If there is a player on the roster with the given name, then the program will collect the new attack stat (**double**), and block stat (**double**), and will update the stat values for this player.

When the user chooses 7 to count players, then the program will display the number of players on the current roster.

When the user chooses 8 the program will end.

The main class has already been designed and written for this program (given in **Assignment08.java**). Carefully review the code in the **Assignment08.java** file and be sure that you understand how it works.

Your task is to implement the **Player** and **Roster** classes. The **Player** class will allow us to instantiate **Player** objects that will store the important information (name, attack stat, block stat) for a player. The **Roster** class will allow us to create and manage a roster of players - we will use an **ArrayList<Player>** to store the **Player** objects.

## Part 1 - Implement the *Player* Class

In a file named **Player.java**, implement the class described below.

The **Player** class must have the following private instance variables:

- a variable named **name** that will store a **String**
- a variable named **attackStat** that will store a **double**
- a variable named **blockStat** that will store a **double**

The **Player** class must have the following public constructor method:

- an overloaded constructor that takes three arguments. The first argument will be a **String** (player's name). The second (attack stat) and third (block stat) arguments will be **double**.

The **Player** class must have the following public methods:

- a method named **getName**. This accessor method will take no arguments. This method will return a **String**.
- a method named **getAttackStat**. This accessor method will take no arguments. This method will return a **double**.
- a method named **setAttackStat**. This mutator method will take one **double** argument. This method will not return anything.
- a method named **getBlockStat**. This accessor method will take no arguments. This method will return a **double**.

- a method named **setBlockStat**. This mutator method will take one **double** argument. This method will not return anything.
- a method named **printPlayerInfo**. This method will take no arguments. This method will not return anything.

Other Details

- The overloaded constructor should initialize the object's **name**, **attackStat** and **blockStat** variables with the values passed in to the parameter variables.
- The **getName** accessor method should simply return the value stored in the object's **name** variable.
- The **getAttackStat** accessor method should simply return the value currently stored in the object's **attackStat** variable.
- The **setAttackStat** mutator method should store the value passed in as an argument in the object's **attackStat** variable.
- The **getBlockStat** accessor method should simply return the value currently stored in the object's **blockStat** variable.
- The **setBlockStat** mutator method should store the value passed in as an argument in the object's **blockStat** variable.
- The **printPlayerInfo** method should print out to the console, the name and stats for this player object. The printout should look like this:

```
Rachael Adams (attack = 3.36, block = 1.93)
```

### Part 2 - Implement the *Roster* Class

In a file named **Roster.java**, implement the class described below.

The **Roster** class must have the following private instance variables:

- a variable named **players** that will store a reference to an **ArrayList<Player>**

The **Roster** class must have the following public constructor methods:

- a default constructor
- an overloaded constructor that takes one argument. This argument will be a **String**.

The **Roster** class must have the following public methods:

- a method named **addPlayer**. This method will take three arguments. The first argument will be a **String** (player's name). The second (attack stat) and third (block stat) arguments will be **double**.
- a method named **countPlayers**. This method will take no arguments. This method will return an **int**.
- a method named **getPlayerByName**. This method will take one **String** argument. This method will return a **Player** reference.
- a method named **printTopAttackers**. This method will take no arguments. This method will not return anything.
- a method named **printTopBlockers**. This method will take no arguments. This method will not return anything.
- a method named **printAllPlayers**. This method will take no arguments. This method will not return anything.

Other Details

The default constructor should instantiate a new **ArrayList** object, and store a reference to this object in the **Roster**'s **players** instance variable.
The overloaded constructor should instantiate a new **ArrayList** object, and store a reference to this object in the **Roster**'s **players** instance variable. It should then open the open the roster file named in the parameter variable. It should then read in the data from this roster file and, for each line in the file, it should create a new **Player** object with the player name and stat data on that line, and add this **Player** to the **ArrayList players**.
The **addPlayer** method should instantiate a new **Player** object with the name and stats provided in the argument values, and then add this **Player** to the roster's **ArrayList<Player>** object.
The **countPlayers** method should return the number of players currently stored in the roster's **ArrayList<Player>** object.
The **getPlayerByName** method should iterate through the roster's **ArrayList<Player>** object and search for a player with a name that is equal to the argument value. If such a **Player** is found, then this method should return a reference to that **Player** object, otherwise this method should return **null**.
The **printTopAttackers** method should determine the two **Player** objects with the best attack stats (in descending order). It should then call the **printPlayerInfo** method on these Player objects.
The **printTopBlockers** method should determine the two **Player** objects with the best block stats (in descending order). It should then call the **printPlayerInfo** method on these **Player** objects.
The **printAllPlayers** method should iterate through the roster's **ArrayList<Player>** object and call the **printPlayerInfo** method on each of these **Player** objects.

## What to turn in

For this assignment you must upload the following files by the due date.

**Assignment08.java**
**Player.java**
**Roster.java**

Any assignment submitted **less than 24 hours after the posted due date** will have **10 points deducted**.

Any assignment submitted **more than 24 hour after the posted due date** will receive **a zero in the grade book**.

## Grading Rubric

| Criteria | Points |
|---|---|
| ***All required files are submitted*** | 10 |
| Each file includes a comment header with the following information:<br>• `CSE 110 : <Class #> / <meeting days and times>`<br>• `Assignment : <assignment #>`<br>• `Author : <name> & <studentID>`<br>• `Description : <of the file contents>`<br>• Partial credit can be awarded | |
| ***Code is neat and well organized*** | 10 |
| • Good naming conventions for all identifiers<br>• Good use of whitespace<br>• Descriptive comments<br>• Methods are small and flat<br>• Partial credit can be awarded | |
| ***Code compiles with no syntax errors*** | 20 |
| • No Partial credit can be awarded<br>• **No credit will be awarded for structure or logic if your code does not compile** | |
| ***Code passes structure tests*** | 30 |
| • Partial credit can be awarded based on percentage of tests passed<br>• Link to Structure Tests – https://repl.it/@MuhilanRamamoor/CSE110NVolleyballStructureTests-2 | |
| ***Code passes logic tests*** | 30 |
| • **No credit will be awarded for logic if your code does not pass all structure tests**<br>• Partial credit can be awarded based on percentage of tests passed<br>• Link to Logic Tests – https://repl.it/@MuhilanRamamoor/CSE110NVolleyballLogicTests-2 | |
| **TOTAL** | **100** |