Homework 01 – Java FC

Author: Daniel, Emily, Nicolas, Ricky

Topics: classes and objects, encapsulation, constructors, visibility modifiers, getters, setters

Problem Description

You have recently just been hired as Java Football Club's new manager and are already aiming your sights to win the league cup this season. You didn't expect that coaching a soccer team would be so difficult, so you decided to use your knowledge of object-oriented programming to write a program to help you keep track of your players. To do this, you will create four classes: `Position.java`, `Player.java`, and `Roster.java` to put together your team.

Solution Description

Create `Position.java` to define the various positions on the field that a player can play, `Player.java` to represent the players you coach, and `Roster.java` to represent the roster of players on Java FC. You will be creating several fields and methods for each class. Based on the description given for each field and method, you will have to decide whether they should be static or non-static, and whether it should be private or public. To make these decisions, you should carefully follow the guidelines on these keywords as taught in lecture. In some cases, your program will still function with an incorrect keyword.

REMEMBER: Homeworks are different from PEs in that they state REQUIREMENTS. These requirements may NOT always be stated in the ORDER in which you should implement them. For example, we may ask you to return a value and report an error, but these two requirements must be handled in the correct order.

Notes:

1. All variables must **not** be allowed to be **directly modified** outside the class in which they are declared and require an instance to be accessed through, unless otherwise stated in the description of the variable.
2. All methods should be accessible from everywhere and require an instance to be accessed through, unless otherwise specified.
3. Use constructor and method chaining whenever possible! Ensure that your constructor chaining helps you reduce code repetition and maximize code reuse.

Position.java

This file defines the `Position` enumeration, used to represent the positions a player specializes in. Define the possible positions in this order: `GOALKEEPER`, `DEFENDER`, `MIDFIELDER`, and `FORWARD`.

Player.java

This class defines the states and behaviors of any instance of `Player`.

Variables:

- `String playerName` – the name of this player (e.g., "Lionel Messi")
  - Once this field is initialized, this field should be immutable.
    - Note that there is a difference between the field and the object being immutable.
- `int stamina` – the stamina of this player
  - This must ***always*** be a value in the range [0, 100]. If the initial `stamina` is not within the appropriate range, initialize it to a value of 75.
- `Position[] positions` – the positions on the field that this player specializes in
  - This is represented by an array because a player may be able to play multiple positions. The first element in the array is this player's preferred position.
- `int skillRating` – the skill rating of this player
  - Players on Java FC have a skill rating in the range [40, 100]. Ensure that `skillRating` is ***always*** within the appropriate range.
  - When `skillRating` is initialized, one of the following statements should be printed:
    - "`Skill rating: Excellent`" if the rating is in [90, 100].
    - "`Skill rating: Great`" if the rating is in [80, 89].
    - "`Skill rating: Very Good`" if the rating is in [70, 79].
    - "`Skill rating: Good`" if the rating is in [60, 69].
    - "`Skill rating: Fine`" if the rating is in [50, 59].
    - "`Skill rating: Bad`" if the rating is in [40, 49].
    - **Note:** This should only be printed when `skillRating` is first initialized (i.e., on construction).
  - If the initial `skillRating` is not within the appropriate range, set it to 80 and print the appropriate statement.

Constructors:

- A constructor that takes in `playerName`, `stamina`, `positions`, and `skillRating`.
- A constructor that takes in `playerName` and `positions` and defaults `stamina` to 75 and `skillRating` to 80.
- A constructor that takes in no arguments and defaults `playerName` to "Lionel Messi", `stamina` to 75, `positions` to an array of length 1 containing `FORWARD`, and `skillRating` to 100.
- **Note:**
  - Constructor parameters should be written in the order described above.
  - You may assume that the `playerName` input will not be null.
  - You may assume that the `positions` input will not be null nor contain null or duplicate elements. If the input array is empty, initialize `positions` to an array of length 1 containing `MIDFIELDER`.

Methods:

- `isTrainable`
  - This method returns true if this player's skill rating is in [50, 89], and false otherwise.
- `preferredPosition`

  - This method returns this player's preferred position (i.e., the first element in the `positions` array).
- `canPlayAs`
  - This method takes in a `Position` and returns true if the player can play in this position, and false otherwise.
- `toString`
  - This method returns a String representation of this `Player`.
  - The string must be formatted as follows (**with angle brackets**, without curly braces, without spaces between commas, and without trailing whitespace):

    ```
    <{value1},{value2},{value3},{value4},{value5}>
    ```

    - The `{valueN}` should be displayed in the following order:
      - `playerName`, `stamina`, `preferredPosition()`, `skillRating`, `isTrainable()`
  - You must use the `String.format` method (do not use string concatenation).
- Getters and setters ***only as necessary***.
- Any helper methods that you may need; ensure that these methods are not accessible outside of this class.

Roster.java

This class defines a roster of players.

Variables:

- `Player[] players` – the players on the roster
  - **Note:** Treat this array like a binder of sleeves that can hold cards. If you take a card out of the sleeve, there is no need to move all the other card forward one sleeve to ensure a space is not "empty." You should also think about how we represent the **absence** of an object when we have reference types. There **must** be a value in the reference, but what value for reference types is suitable to represent "empty"?
- `int size` – the size of this roster
  - The value of this field should be updated accordingly to represent the number of players on the roster.

Constructors:

- A constructor that takes in `players`.
  - Don't forget to initialize the value of the `size` field correctly based on the number of players in the `players` array.
    - **Hint:** The `size` may not necessarily be the length of the array.
- A constructor that takes no parameters and initializes `players` to an empty array of length 4.
- **Note:**
  - Constructor parameters should be written in the order described above.
  - You may assume that the `players` input will not be null. However, the `players` array may contain null elements.

Methods:

- `signPlayer`

- Given an `index` represented by an `int` and a `player` represented by a `Player`, add that player to the specified index in the `players` array.
- Print the `Player` that was previously at that index in the following format on its own line: "`Replaced: {oldPlayer}`", and return the player that was previously in that slot. If there is not already a player in the slot, print the player being inserted in the following format on its own line: "`Signed: {player}`", and return `null`.
  - Note that you can print out a `Player` directly since we implemented the `toString` method above. In the following weeks, we will go more in-depth as to why this is the case!
- If the index is invalid or the player passed in is `null`, return `null` and print out "`Cannot add a player to this spot on the roster.`"
- Update `size` if a player was added to an unoccupied slot.

- `transferPlayer`
  - Given an `index` represented by an `int`, sell the player in that slot to the transfer market and print "`Transferred: {player}`" on its own line. Remove the player from the `players` array and return the player that was transferred.
  - If the index is invalid or there is not a player at that index, print "`There was no player to transfer!`" on its own line and return `null`.
  - Update `size` if a player was removed.
- `showBestPlayers`
  - Given an `int` representing a player's skill rating, display all players with a `skillRating` **<u>greater than</u>** the value passed in by printing each player on its own line.
  - **Hint:** As you're writing this method, you will likely run into an exception. Consider what that exception is and how you can check for it.
- `trainAllPlayers`
  - Train all the players that are trainable.
  - To train a player, add a random integer in the range [1, 10] to their current `skillRating`.
  - Print each `Player` that was trained in the following format on its own line: "`Trained to {newSkillRating}: {player}`".
    - **Note:** `player` refers to the player's information **<u>before</u>** they are trained (i.e., the player should be printed with the old skill rating).
  - If no players were trained, print "`There were no players to train.`" on its own line.
  - If training a player makes their skill rating exceed 100, set the player's skill rating to 100.
- `play`
  - Given an `index` and a `position`, play the player found at the specified index in the `players` array in a league match in the given position on the pitch.
    - You may assume that the `position` passed in will not be null.
  - If the given `position` is the player's preferred position, subtract a random integer in the range [1, 5] from the player's current `stamina`. If the given `position` is not the player's preferred position but is a position this player can play, subtract a random integer in the range [5, 10] from the player's current `stamina`.
    - **Note:** Remember that `stamina` has a minimum value of 0, so if subtracting the random integer makes it fall below this number, `stamina` should be set to 0.

- If the position passed is not a position the player can play, print out "`This player cannot be played in position {position}.`", and return from the method.
- If the player at the specified index is played in the league match, print out the player with the updated stamina in the following format on its own line: "`Played: {player}`".
- If the index is invalid, or there is not a player at the specified index, print "`Cannot play the player in this spot.`" on its own line.

- `toString`
  - This method returns a String representation of this `Roster`.
  - If there are no players on the roster, return "`The team has no players!`".
  - Otherwise, return a string that is formatted as follows without leading nor trailing whitespace:

    ```
    There are {size} players on Java FC.
    {player1}
    {player2}
    {player3}
    ...
    ```

    - **Note:** You must print out **all** players in the `roster` in the specified format. The above example only displays up to a hypothetical third player.
- Getters and setters ***only as necessary***
- Any helper methods that you may need; ensure that these methods are not accessible outside of this class.

Checkstyle

You must run Checkstyle on your submission (to learn more about Checkstyle, check out cs1331-style-guide.pdf under the Checkstyle Resources module on Canvas). **The Checkstyle cap for this assignment is 10 points.** This means there is a maximum point deduction of 10. If you don't have Checkstyle yet, download it from Canvas → Modules → Checkstyle Resources → checkstyle-8.28.jar. Place it in the same folder as the files you want to run Checkstyle on. Run Checkstyle on your code like so:

```
$ java -jar checkstyle-8.28.jar YourFileName.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the number of points we would take off (limited by the Checkstyle cap). In future assignments we will be increasing this cap, so get into the habit of fixing these style errors early!

For additional help with Checkstyle see the CS 1331 Style Guide.

Turn-In Procedure

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `Position.java`
- `Player.java`
- `Roster.java`

Make sure you see the message stating the assignment was submitted successfully. From this point, Gradescope will run a basic autograder on your submission as discussed in the next section. **Any autograder tests are provided as a courtesy to help "sanity check" your work and you may not see all the test cases used to grade your work.** You are responsible for thoroughly testing your submission on your own to ensure you have fulfilled the requirements of this assignment. If you have questions about the requirements given, reach out to a TA or Professor via the class forum for clarification.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the assignment. We will only grade your latest submission. **Be sure to submit every file each time you resubmit**.

Gradescope Autograder

If an autograder is enabled for this assignment, you may be able to see the results of a few basic test cases on your code. Typically, tests will correspond to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue. **We reserve the right to hide any or all test cases, so you should make sure to test your code thoroughly against the assignment's requirements.**

The Gradescope tests serve two main purposes:

- Prevent upload mistakes (e.g., non-compiling code)
- Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or Checkstyle your code; you can do that locally on your machine.

Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.

Burden of Testing

You are responsible for thoroughly testing your submission against the written requirements to ensure you have fulfilled the requirements of this assignment.

Be **very careful** to note the way in which text output is formatted and spelled. Minor discrepancies could result in failed autograder cases.

If you have questions about the requirements given, reach out to a TA or Professor via the class forum for clarification.

Allowed Imports

- `java.util.Random`

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our autograder. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- `System.exit`
- `System.arraycopy`

Collaboration

Only discussion of the assignment at a conceptual high level is allowed. You can discuss course concepts and assignments broadly; that is, at a conceptual level to increase your understanding. If you find yourself dropping to a level where specific Java code is being discussed, that is going too far. Those discussions should be reserved for the instructor and TAs. To be clear, you should never exchange code related to an assignment with anyone other than the instructor and TAs.

The only code you may share are test cases written in a Driver class. If you choose to share your Driver class, they should be posted to the assignment discussion thread on the course discussion forum. We encourage you to write test cases and share them with your classmates, but we will not verify their correctness (i.e., use them at your own risk).

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items.
- Do not submit `.class` files.
- Test your code in addition to the basic checks on Gradescope.
- Submit every file each time you resubmit.
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points.
- **Check on Ed Discussion for a note containing all official clarifications and sample outputs.**

It is expected that everyone will follow the Student-Faculty Expectations document and the Student Code of Conduct. The professor expects a **positive, respectful, and engaged academic environment** inside the classroom, outside the classroom, in all electronic communications, on all file submissions, and on any document submitted throughout the duration of the course. **No inappropriate language is to be used, and any assignment deemed by the professor to contain inappropriate offensive language or threats will get a zero.** You are to use professionalism in your work. Violations of this conduct policy will be turned over to the Office of Student Integrity for misconduct.