Homework 05

Authors: Aqdas, Ariel, Connor, Julia, Tejas

Problem Description

The Association of Tennis Professionals is responsible for organizing the major tennis tournaments every year. Each year, two categories of tournaments take place: The Masters Tournament and the Grand Slam tournaments. Unfortunately, this year they have misplaced their leaderboard which contains the rankings of all the tennis players. Fortunately, they hire you as a developer to create the leaderboard utilizing your **asymptotics, searching, and sorting skills**.

Solution Description

You will be creating the files below with the following names:

- Player.java
- ProfessionalPlayer.java
- AmateurPlayer.java
- Masters.java
- GrandSlam.java

`Player.java`

This class represents a tennis `Player` object. This class must be abstract.

Interfaces:

- The `Player` class must implement the `Comparable` interface parameterized for `Player` objects.

Instance Variables:

- All instance variables in this class should be private unless otherwise stated.
- `String name`
    - This represents the name of the tennis player.
- `double points`
    - This represents the number of points accumulated over the year by a player
- `int trophies`
    - This represents the number of trophies won by the player in the previous year

Constructors:

- Include a constructor that takes in `name`, `points`, and `trophies` in this exact order.
- Make sure the constructor parameters have the same type as their corresponding fields.

Methods:

- All methods in this class should be public unless otherwise stated.
- `calculateSkillLevel()`
    - This method should be abstract and return a double.
- `compareTo()`
    - Since the `Player` class implements the `Comparable` interface, this class must properly override the `compareTo()` method.

    - This method should take in a `Player` object as a parameter.
    - The current `Player` is greater than the passed-in `Player` if the current Player's skill level is greater than the other Player's skill level.
- `equals()`
    - This method must properly override the `equals()` method from the `Object` class.
    - Two Players are equal if they have the same `name`, `points`, and `trophies`.
- `toString()`
    - This method returns a String of the following format:
      `"{name}: {points rounded to 1 decimal place} points - {trophies} trophies"`
        - The curly braces {} denote values that should be replaced with the corresponding variable(s), do not include the curly braces in your output.
- Necessary getters and setters.

ProfessionalPlayer.java

This file represents a tennis `ProfessionalPlayer` object. You will create this file and make sure that it **inherits from the `Player` class.** This class must be concrete.

Instance Variables:

- All instance variables in this class should be private unless otherwise stated.
- `boolean isSeeded`
    - This boolean represents whether the `ProfessionalPlayer` was seeded (ranked in the top ten Players) in the past year.

Constructors:

- Include a constructor that takes in `name`, `points`, `trophies and isSeeded` in this exact order.
- Make sure the constructor parameters have the same type as their corresponding fields.

Methods:

- All methods in this class should be public unless otherwise stated.
- `calculateSkillLevel()`
    - This method does not take in any parameters and overrides the abstract method in the `Player` class.
    - Since the trophies won by professionals are valuable, the skill level of the player will be calculated using the following formula:
      `{points} + 1.5 * {trophies}`
    - If the professional player is seeded, then the result of the above formula gets doubled.
    - The total value calculated by the above operations should be returned.
- `equals()`
    - This method must properly override the `equals()` method from the `Player` class and utilize it to maximize code reuse.
    - Two `ProfessionalPlayer` objects are equal if they have the same `name`, `points`, `trophies`, and `isSeeded`.

`AmateurPlayer.java`

This file represents a tennis `AmateurPlayer` object. You will create this file and make sure that it **inherits from the `Player` class.** This class must be concrete.

Constructors:

- Include a constructor that takes in `name`, `points`, and `trophies` in this exact order.
- Make sure the constructor parameters have the same type as their corresponding fields.

Methods:

- All methods in this class should be public unless otherwise stated.
- `calculateSkillLevel()`
  - This method does not take in any parameters and overrides the abstract method in the `Player` class.
  - Since the trophies won by amateurs are not as valuable as those of a `ProfessionalPlayer`, the skill level of this Player will be calculated using the following formula:
    `{points} + 0.5 * {trophies}`
    - Note: This formula is different than the one used in the `ProfessionalPlayer` class above.
  - The total value calculated by the above formula should be returned.

`Masters.java`

This file represents a `Masters` tournament object. This class should be concrete. You will create this file and fill in various methods that implement sorting and searching functionality for the Players in the tournament. The Masters tournament is open for all Players regardless of their level.

Instance Variables:

- All instance variables in this class should be private unless otherwise stated.
- `String name`
  - This represents the name of the specific Masters tournament object
- `ArrayList leaderboard`
  - This represents the rankings of all the `Player` objects registered for the tournament
  - This `ArrayList` should only be able to hold `Player` objects

Constructors:

- Include a two-argument constructor that takes in the `name` of the tournament and the `leaderboard` of `Player` objects in this exact order.
  - Because the tournament's previous management was very disorganized, the passed-in leaderboard may not be in the correct order. However, because this tournament involves lots of players, we do not have time to sort it ahead of time. Therefore, **do not** sort `leaderboard`, simply assign the passed-in variables accordingly.
    - `leaderboard` should be a shallow copy of the passed-in `ArrayList`
- Make sure the constructor parameters have the same type as their corresponding fields.

Methods:

- All methods in this class should be public unless otherwise stated.

- `createLeaderboard()`
  - This method should not take any parameters.
  - Given the unsorted `leaderboard` of `Player` objects the management must sort them according to their natural ordering.
    - Since the management of the Masters tournament is not experienced, they can only look at the Player objects one at a time and transfer them to the correct relative position. Therefore, this method should employ the **Insertion Sort Algorithm.**
      - ◊ The algorithm should start at the beginning of the `ArrayList` and Players with the highest skill level will be shuffled to the front.
      - ◊ If two Players have the same skill level, the `Player` object that appears first in the unsorted leaderboard should appear first in the sorted leaderboard.
    - The `compareTo()` method should be used to determine the natural ordering of the objects.
    - You should not invoke the `ArrayList` `remove()` method at any point in the implementation of this method. Instead, you should use a combination of the `get()` and `set()` methods to swap elements when necessary.
  - This method should not return anything.
- `findPlayer()`
  - This method takes in a `Player` object.
  - This method should employ a search algorithm to find a `Player` object in the `leaderboard` equal to the passed-in `Player`.
    - Since the management of the `Masters` tournament is not efficient, the only way they can find a Player is to check the list one by one. Therefore, this method should employ a search algorithm which has a **time complexity of `O(n)`**.
  - If there is no `Player` object in the `leaderboard` equal to the passed-in `Player` object, then return -1. Otherwise, return the index of the `Player` object in the `leaderboard`.
  - To prevent trivialization of this method, you are not allowed to use the `ArrayList` `indexOf` or `lastIndexOf` methods.
- `toString()`
  - This method returns a String of the following format:
    `"Welcome to the {name} tournament! The current leaderboard is: {leaderboard.toString()}"`
    - The curly braces {} denote values that should be replaced with the corresponding variable(s), do not include the curly braces in your output.

GrandSlam.java

This file represents a `GrandSlam` tournament. This class should be concrete. GrandSlams are the most competitive tennis tournaments and therefore have an experienced management. Due to the competitive nature of Grand Slam tournaments, only ProfessionalPlayers can participate in them.

Instance Variables:

- All instance variables in this class should be private unless otherwise stated.
- `String name`
  - This represents the name of the GrandSlam tournament.

- `ArrayList leaderboard`
    - This represents the rankings of all the Professional Players registered for the tournament.
    - This `ArrayList` should only be able to hold `ProfessionalPlayer` objects.

Constructors:

- Include a two-argument constructor that takes in the `name` of the tournament and the `leaderboard` of `ProfessionalPlayer` objects in this exact order.
    - Because the tournament's previous management was very disorganized, the passed-in leaderboard may not be in the correct order. Unlike the `Masters` tournament, the `GrandSlam` tournament only consists of a select group of players, so we can afford to take the time to sort the `leaderboard` ahead of time. You will need to sort the `leaderboard` yourself using the `createLeaderboard()` method.
        - `leaderboard` should be a shallow copy of the passed-in `ArrayList`
- Make sure the constructor parameters have the same types as their corresponding fields.

Methods:

- All methods in this class should be public unless otherwise stated.
- `createLeaderboard()`
    - This method should not take any parameters.
    - Given the unsorted `leaderboard` of `ProfessionalPlayer` objects the management must sort them according to their natural ordering.
        - Since the management of the `GrandSlam` tournament is very experienced, they can pull Players from anywhere in the list. Therefore, the method should employ the **Selection Sort Algorithm.**
            - ◊ The algorithm should start at the beginning of the `ArrayList` and Players with the highest skill level will be shuffled to the front.
            - ◊ If two ProfessionalPlayers have the same skill level, the `ProfessionalPlayer` object that appears first in the unsorted leaderboard should appear first in the sorted leaderboard.
        - The `compareTo()` method should be used to determine the natural ordering of the objects.
        - You should not invoke the `ArrayList remove()` method at any point in the implementation of this method. Instead, you should use a combination of the `get()` and `set()` methods to swap elements when necessary.
    - This method should not return anything.
- `findPlayer()`
    - This method takes in a `ProfessionalPlayer` object.
    - This method should employ a search algorithm to find a `ProfessionalPlayer` object in the `leaderboard` equal to the passed-in `ProfessionalPlayer`.
        - Since the management of the GrandSlam tournament is very efficient, they can find a player by looking anywhere in the list. Additionally, the list was sorted ahead of time (in the constructor) due to the elite nature of this tournament. Therefore, this method should employ a search algorithm which has a **time complexity of O(log n).**
            - ◊ At each step, if there are two middle indices, your search algorithm should choose the **lower** of the two to check.

    - If there is no `ProfessionalPlayer` object in the `leaderboard` equal to the passed-in `ProfessionalPlayer` object, then return -1. Otherwise, return the index of the `ProfessionalPlayer` object in the `leaderboard`. Note that `a.compareTo(b)` being 0 does not necessarily imply `a.equals(b)` is true.
    - **Note:** We will not test this method with a `leaderboard` that contains multiple `ProfessionalPlayer` objects with the same skill level.
- `toString()`
    - This method returns a String of the following format:
      `"Welcome to the {name} tournament! The current leaderboard is: {leaderboard.toString()}"`
        - The curly braces {} denote values that should be replaced with the corresponding variable(s), do not include the curly braces in your output.

Clarifications and Example Output

Please refer to the pinned Piazza thread for HW05 Clarifications for any necessary clarifications about the requirements outlined in this pdf. We may also post some example output that you can use to check your code against on the same Piazza post.

Allowed Imports

You may only import `java.util.ArrayList`.

Feature Restrictions

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

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

Checkstyle and Javadocs

For this homework you may ignore the "Definition of 'equals()' without corresponding definition of 'hashCode()'." Checkstyle error.

You must run Checkstyle on your submission. The Checkstyle cap for this assignment is **15 points**. If you don't have Checkstyle yet, download it from Canvas -> Modules -> Checkstyle Resources (all sections). Place it in the same folder as the files you want Checkstyled. 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 points we would take off (limited by the Checkstyle cap mentioned above). The Java source files we provide contain no Checkstyle errors. In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

Additionally, you must Javadoc your code.

Run the following to only check your Javadocs:

```
$ java -jar checkstyle-8.28.jar -j yourFileName.java
```

Run the following to check both Javadocs and Checkstyle:

```
$ java -jar checkstyle-8.28.jar -a yourFileName.java
```

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

Collaboration

No collaboration is allowed on this assignment. See syllabus for more details.

In addition, note that it is not allowed to upload your code to any sort of public repository or website. This could be considered an Honor Code violation, even if it is after the homework is due. Accessing a posted solution is also a violation of the Honor Code.

Turn-In Procedure

Submission

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

- Player.java
- ProfessionalPlayer.java
- AmateurPlayer.java
- Masters.java
- GrandSlam.java

Please only submit the individual files—do not put them into a package or you will risk getting a zero on the assignment. Make sure you see the message stating "HW05 submitted successfully". From this point, Gradescope will run a basic autograder on your submission as discussed in the next section.

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

Gradescope Autograder

For each submission, you will be able to see the results of a few basic test cases on your code. Each test typically corresponds 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.

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.

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
- Run Checkstyle on your code to avoid losing points
- Submit every file each time you resubmit
- Read the "Allowed Imports" and "Feature Restrictions" to avoid losing points
- Check on Piazza for a note containing all official clarifications