# Homework 03 – UEFA Tournaments

Authors: Faris, Chelsea, Aqdas, David, Allan
Topics: abstract classes, overriding, equals method, toString

## Problem Description

Please make sure to read all parts of this document carefully.

The Union of European Football Associations (UEFA) has commissioned you as a freelance developer to help refine its database of tournaments and add its two new tournaments to that database. Having recently learnt about **abstract classes**, **overriding**, and some **components of a good class**, you accept the job.

## Solution Description

Create files Tournament.java, SummerShowdown.java, and ConferenceLeague.java.
You will be creating several fields and methods for each file. Based on the description given for each variable and method, you will have to decide whether the variables/method should be static, and what visibility modifier applies. 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.

## *Tournament.java*

This file represents any Tournament that UEFA runs. Tournament.java is an abstract class.

**Variables:**

All variables should not be allowed to be directly modified outside the class in which they are declared, unless stated otherwise:

- `String name` – the name of the tournament
- `int numTeams` – the number of teams in a tournament (this must be an even number, otherwise set it to 16)
- `boolean knockout` – if it is a knockout tournament, this is true (default value is true). A knockout tournament is a tournament where the loser is immediately eliminated.
- `String finalReferee` – the name of the referee for the final game of the tournament
- `int maxCapacity` – the maximum capacity for all stadiums, which must be watched due to newly implemented COVID restrictions in Europe (this variable must be accessible only by all of Tournament's subclasses, regardless of if they are in the same package or not)
- `int FINAL_CAPACITY` – the maximum number of people allowed for the final game; UEFA wants this to be a smaller number, to make the final an intimate game between players and fans (this value must not be changeable after it is declared -- this will be set to 30,000)

Constructor(s):

- A constructor that takes in `name`, `numTeams`, `knockout`, `finalReferee`, and `maxCapacity`
    - First, if the number of teams is not even, set it to 16.
    - Then, if number of teams in the tournament is greater than 64, and it is not a knockout tournament, make it a knockout tournament (ie: set `knockout` to `true`)

**Methods:**

All methods in this class should be public, unless stated otherwise:

- `toString()`
    - This method should properly override `Object`'s `toString()` method
    - If it is a knockout tournament, the `toString()` method should say "this is a KO tournament"; if not, it should say "this isn't a KO tournament"
    - The method should return:
      "`Name: {name}, Number of Teams: {numTeams}, KO: this is/isn't a KO tournament, Final Referee: {finalReferee}, Maximum Capacity: {maxCapacity}`"
- `equals()`
    - This method should properly override `Object`'s `equals()` method
    - If two Tournaments have the same `name`, `numTeams`, and `knockout` values, they are equal
- `refereeHire()`
    - This method is abstract and will take in a `String`
    - It will not return anything
- Any **necessary** getter and setter methods (emphasis on **necessary** – don't write any that have no use)

## *SummerShowdown.java*

This class describes one of UEFA's new tournaments, Summer Showdown. This class will inherit from the Tournaments class.

**Variables:**

All variables must not be allowed to be directly modified outside the class in which they are declared, unless otherwise stated in the description of the variable.

- `String backupReferee` – every referee can only referee one tournament's final, so we have a backup referee just in case

**Constructor(s):**

- A constructor that takes in `numTeams`, `knockout`, `finalReferee`, `maxCapacity`, and `backupReferee`
    - Reuse any code if you can
    - name will be "Summer Showdown"

- A constructor that takes in `finalReferee` and `backupReferee`
    - Use constructor chaining if possible
    - `name` will be "Summer Showdown", `numTeams` will be 64, `knockout` will be false, and `maxCapacity` will be 20,000

**Methods:**

All methods must have the proper visibility to be used where it is specified they are used.

- `toString()`
    - This method should properly override Object's `toString()` method
    - It should return the following:
      "`Name: {name}, Number of Teams: {numTeams}, Final Referee: {finalReferee}, Backup Referee: {backupReferee}, Maximum Capacity: {maxCapacity}`"
- `equals()`
    - This method should properly override Object's `equals()` method
    - If two `SummerShowdown` objects have the same `finalReferee` and `maxCapacity`, they are equal
- `refereeHire()`
    - This method should properly override the `Tournament` class' `refereeHire()` method
    - If the inputted referee and `SummerShowdown`'s `finalReferee` are the same, print, "`We need a new referee! {inputReferee} is already refereeing the final!`" and change `backupReferee` to "`Undecided`"
    - If `backupReferee` and the input are the same, print, "`Be ready for some conflicts!`" and divide `maxCapacity` by 2 – this will reduce the pressure on the referees with a smaller crowd
    - Otherwise, print "`Ready to play!`"

## *ConferenceLeague.java*

This class describes the second of UEFA's new tournaments, Conference League. This class will also inherit from the Tournaments class.

**Variables:**

All variables must not be allowed to be directly modified outside the class in which they are declared, unless otherwise stated in the description of the variable.

- `int teamsPerLeague` – the number of participating teams from each league
- `String currentHolder` – the current holder of the Conference League trophy

**Constructor(s):**

- A constructor that takes in `finalReferee, teamsPerLeague` and `currentHolder`
    - Use constructor chaining if possible
    - `name` will be "Conference League", `numTeams` will be 32, `knockout` will be true, and `maxCapacity` will be 15,000

**Methods:**

All methods must have the proper visibility to be used where it is specified they are used.

- `toString()`
  - This method should properly override `Object`'s `toString()` method
  - It should return the following:
    "`Name: {name}, Number of Teams: {numTeams}, Final Referee: {finalReferee}, Teams per League: {teamsPerLeague}, Current Holder: {currentHolder}`"
- `equals()`
  - This method should properly override `Object`'s `equals()` method
  - If two `ConferenceLeague` objects have the same `currentHolder` and `finalReferee`, they are equal
- `nextRound()`
  - This method does not return anything
  - It first checks if the tournament is a knockout tournament or not. If so, it checks whether there are more than 2 teams remaining or not:
    - If there are more than 2 teams remaining, the number of teams is halved
    - If not, it prints, "We have reached the final!" and checks if `maxCapacity` is greater than `FINAL_CAPACITY`; if so, it sets `maxCapacity` to the value of `FINAL_CAPACITY`
- `refereeHire()`
  - This method should properly override `Tournament`'s `refereeHire()` method
  - If the referees (the inputted one and `ConferenceLeague`'s one) are the same, print, "`We need a new referee!`" to a new line, and change the value of the `ConferenceLeague`'s referee to "Undecided"
  - Otherwise, print "`We're ready to go!`" to a new line and set `maxCapacity` to double what it is – these referees can handle pressure from bigger crowds

## Testing

Reuse your code when possible. Certain methods can be reused using certain keywords. These tests and the ones on Gradescope are by no means comprehensive, so be sure to create your own!

## Checkstyle

**You may ignore the hashcode checkstyle error. We will not take points off for this error.**

You must run checkstyle on your submission (To learn more about Checkstyle, check out cs1331-style-guide.pdf under CheckStyle Resources in the Modules section of Canvas.) **The Checkstyle cap for this assignment is 20 points.** This means there is a maximum point deduction of 20. 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 points we would take off (limited by the checkstyle cap mentioned in the Rubric section). 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.

## Turn-In Procedure

### *Submission*

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

- `Tournament.java`
- `SummerShowdown.java`
- `ConferenceLeague.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 test 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 Piazza 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 homework. 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.

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.

## Allowed Imports

To prevent trivialization of the assignment, you are not allowed to import any classes or packages.

## 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`
- `System.arraycopy`

## Collaboration

Only discussion of the Homework (HW) at a conceptual high level is allowed. You can discuss course concepts and HW 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.

## 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.