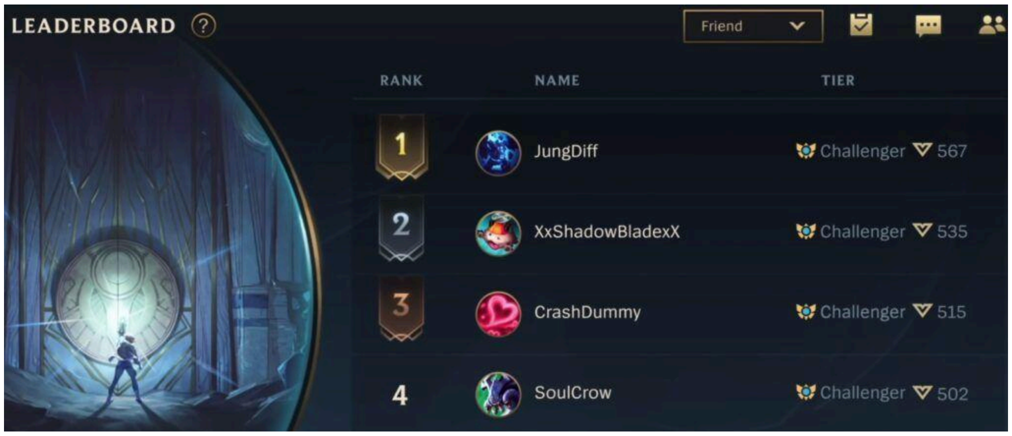
# P09 Leaderboard

## Overview

**STUDENTS BE AWARE: WE ARE ENABLING PvP IN CS 300**

Okay no we're not. But if we were, this is how we'd keep track of who's winning.

Hierarchical data structures allow us to efficiently maintain a sorted data structure, so our first attempt at this will be using a Binary Search Tree to maintain a leaderboard of the players of a game.



| RANK | NAME | TIER |
|---|---|---|
| 1 | JungDiff | Challenger 567 |
| 2 | XxShadowBladexX | Challenger 535 |
| 3 | CrashDummy | Challenger 515 |
| 4 | SoulCrow | Challenger 502 |

## Grading Rubric

| | |
|---|---|
| 5 points | **Pre-assignment Quiz**: accessible through Canvas until 11:59PM on 11/24. |
| +5% | **Bonus Points**: students whose ***final*** submission to Gradescope is before **5:00 PM Central Time** on **WED 11/27** *and* who pass ALL immediate tests will receive an additional 2.5 points toward this assignment, **up to a maximum total of 50 points**. |
| 12 points | **Immediate Automated Tests**: accessible by submission to Gradescope. You will receive feedback from these tests *before* the submission deadline and may make changes to your code in order to pass these tests.<br><br>Passing all immediate automated tests does **not** guarantee full credit for the assignment. |
| 20 points | **Additional Automated Tests**: these will also run on submission to Gradescope, but you will not receive feedback from these tests until after the submission deadline. |
| 13 points | **Manual Grading Feedback**: TAs or graders will manually review your code, focusing on algorithms, use of programming constructs, and style/readability. |
| **50 points** | **MAXIMUM TOTAL SCORE** |

## Learning Objectives

After completing this assignment, you should be able to:

- **Describe** the structure and functionality of a Binary Search Tree.
- **Implement** a Binary Search Tree with the relevant ***recursive*** algorithms for adding, removing, and traversing nodes.
- **Demonstrate** the utility of the Iterator, Iterable, and Comparable interfaces.

## Additional Assignment Requirements and Notes

Keep in mind:

- Pair programming is **NOT ALLOWED** for this assignment. You must complete and submit P09 individually.

- The ONLY external libraries you may use in your program are:

  `java.util.Iterator, java.util.NoSuchElementException`

- Use of *any* other packages (outside of java.lang) is NOT permitted.

- You are allowed to define any **local** variables you may need to implement the methods in this specification (inside methods). You are NOT allowed to define any additional instance or static variables or constants beyond those specified in the write-up.

- You are allowed to define additional **private** helper methods.

- Only **Game** and **LeaderboardTester** may contain a main method.

- All classes and methods must have their own Javadoc-style method header comments in accordance with the CS 300 Course Style Guide.

- Any source code provided in this specification may be included verbatim in your program without attribution.

- All other sources must be cited explicitly in your program comments, in accordance with the Appropriate Academic Conduct guidelines.

- Any use of ChatGPT or other large language models ***must be cited*** AND your submission MUST include **screenshots** of your interactions with the tool ***clearly showing all prompts and responses in full***. Failure to cite or include your logs is considered academic misconduct and will be handled accordingly.

- **Run your program locally before you submit to Gradescope**. If it doesn't work on your computer, *it will not work on Gradescope*.

## Need More Help?

Check out the resources available to CS 300 students here:
https://canvas.wisc.edu/courses/427315/pages/resources

## CS 300 Assignment Requirements

You are responsible for following the requirements listed on both of these pages on all CS 300 assignments, whether you've read them recently or not. Take a moment to review them if it's been a while:

- Appropriate Academic Conduct, which addresses such questions as:
  - How much can you talk to your classmates?
  - How much can you look up on the internet?
  - How do I cite my sources?
  - and more!
- Course Style Guide, which addresses such questions as:
  - What should my source code look like?
  - How much should I comment?
  - and more!

## Getting Started

1. Create a new project in Eclipse, called something like **P09 Leaderboard**.
   a. Ensure this project uses Java 17. Select "JavaSE-17" under "Use an execution environment JRE" in the New Java Project dialog box.
   b. Do **not** create a project-specific package; use the default package.

2. Download two (1) PROVIDED Java source files from the assignment page on Canvas. You will not modify these files at all:
   a. **BSTNode.java**
   b. **Game.java**

3. Download three (3) INCOMPLETE Java source files from the assignment page. **You must complete these files**:
   a. **Player.java** – implements the **Comparable** interface
   b. **Leaderboard.java** – implements the **Iterable** interface
   c. **LeaderboardTester.java**

4. Create one (1) **new** Java source file within that project's src folder:
   a. **LeaderboardIterator.java** – implements the **Iterator** interface

# Implementation Requirements Overview

In this project you will implement an application that maintains a leaderboard of the players of a PvP (player-vs-player) game, ordered by their scores in that game. Players may challenge other players in the game, which may result in changes to the leaderboard.

We are NOT providing additional documentation beyond this writeup and the comments in the provided code.

## Provided Classes

The following classes are provided in their entirety. You do not need to implement anything in them.

- **BSTNode<T>** – a generic Binary Search Tree node.
- **Game** – a class which maintains a given game's Leaderboard across PvP challenges.

## Classes You Will Implement

You will completely or partially implement the following classes:

- **Player** – represents a single player of the game, including their name and numeric score. This class is nearly complete, but you must make these objects **Comparable** to other Players and complete the `compareTo()` method.
- **Leaderboard** – a binary search tree consisting of BSTNode<Player>. We have provided the public interface to this data structure, but the real work is done in the protected recursive helper methods, which you must implement.
- **LeaderboardIterator** – Leaderboard's `toString()` method relies on an enhanced for loop, which in turn requires an iterator. This iterator must begin at the Player with the smallest score in the leaderboard, and iterate through the entirety of the leaderboard in increasing order.
- **LeaderboardTester** – a tester class for your BST implementation.

## Organization of the Leaderboard

The core data structure in this project is the Leaderboard, which is a Binary Search Tree of Player objects. The BST orders the Players by their score (and breaks ties using the Players' names), so the minimum value in the BST is the Player with the smallest score and the maximum value is the Player with the largest score.

We are providing you with a generic **BSTNode<T>** class. Your Leaderboard must be built out of **BSTNode<Player>** (note Player, not T), and you will implement the relevant algorithms in the Leaderboard class. You will also need to implement **Comparable<Player>** for **Player**, which is the comparison method that you will use in Leaderboard.

# Implementation Details and Suggestions

Begin by adding the required interfaces to your classes:

- **Player** must be **Comparable** (to what type?)
- **Leaderboard** must be **Iterable** (over what type?)
- **LeaderboardIterator** must be an **Iterator** (over what type?)

## 1. Implement compareTo() and Tests

Once Player is Comparable, you will need to complete the required `compareTo()` method according to the description in the comments. At this time you should also implement the first three tester methods in LeaderboardTester:

- testCompareToDiffScore
- testCompareToSameScoreDiffName
- testCompareToEqual

The comments in the file are intended to provide some direction, but you are welcome to add additional tests as you see fit.

## 2. Implement Leaderboard and Tests

We recommend implementing the methods in the order below. Testing and debugging the latter methods will be significantly easier if you know the structure of your BST is correct, and implementing the earlier methods will help you get the structure correct. **After implementing *each* of these methods, implement the relevant tester methods in LeaderboardTester, and test your implementation thoroughly!** If you wait until the very end to test, you will have lots of weird and confusing bugs.

See the next subsection for implementation/testing hints.

1. **getMinScoreHelper** and **getMaxScoreHelper**
2. **countHelper**
3. **lookupHelper** and the following test methods:
    a. testLookupRoot
    b. testLookupLeft
    c. testLookupRight
    d. testLookupNotPresent
4. **addPlayerHelper** and the following test methods:
    a. testAddPlayerEmpty
    b. testAddPlayer
    c. testAddPlayerDuplicate
5. **nextHelper** and the following test methods:
    a. testGetNextAfterRoot

   b. testGetNextAfterLeftSubtree
   c. testGetNextAfterRightSubtree
6. **removeHelper** and the remaining test methods

***Note that you may NOT add ANY loops (of any kind) in the Leaderboard class***. You MUST implement the above methods recursively. The loop present in the `toString()` method is the only permitted loop.

### 2.1 Some Hints

- **Draw LOTS of pictures!**
- **Write tests as you go along**. If you save testing for last you will be sad.
- You MAY use the `addPlayer()` method to construct a tree for other testers, but for the lookup tests in particular (since you won't have completed that method yet) you will probably want to check out the provided `getRoot()` method in Leaderboard. You can add ONE player to the tree (we've provided the code for this), and then set up the rest of the tree by using the BSTNode methods `setLeft()` and `setRight()`.
   - **Note that doing this will NOT affect the `size` fiel**d, so if you need to know how many nodes are present, you'll need to use `count()` instead.
- For most of the Leaderboard tests, it is easiest if you use nearly identical Players that only differ by one aspect (e.g. all players are named "A" but have different scores, or all players have the default 1500 score but different names). This makes keeping track of the correct ordering easier.
- **Remember**: in every node, the left and right subtree should also be a valid binary tree. That is, if you were to just yank out any left or right node anywhere in the tree (with its descendants), they could stand alone as a valid binary search tree.
- If you are writing really long methods, you are probably overthinking it. My longest method by far is **removePlayerHelper**, clocking in at about 40 lines *with* comments and whitespace, and it's about twice as long as anything else in the class.

## 3. Implement the Iterator

The **LeaderboardIterator** implements the Iterator interface and iterates through all values in the tree in increasing order. This means that a full run of

```
for (Player p : leaderboard) System.out.println(p);
```

should begin with `leaderboard.getMinScore()` and end with `leaderboard.getMaxScore()`, and include `leaderboard.size()` number of different lines.

YOU MAY CHOOSE the implementation details of this class (constructor, data fields, etc) as long as it conforms to the following requirements:

1. Implements the **Iterator** interface
2. Constructed and initialized by Leaderboard's `iterator()` method
3. First call to `next()` returns `leaderboard.getMinScore()`
4. Can call `next()` without an exception exactly `leaderboard.size()` number of times
5. When `hasNext()` returns false, calling `next()` causes a NoSuchElementException

Your Leaderboard class must also support use of an **enhanced for loop** as shown above.

## 4. [OPTIONAL] Run the Game

Now that your code is completed, you should be able to successfully run the **Game.java** code supported by your Leaderboard. Check out the sample output on the assignment page if you want to verify that everything is working as expected!

## Assignment Submission

Hooray, you've finished this CS 300 programming assignment!

Once you're satisfied with your work, both in terms of adherence to this specification and the academic conduct and style guide requirements, make a final submission of your source code to Gradescope.

For full credit, please submit the following files (**source code**, *not* .class files):

- Player.java
- Leaderboard.java
- LeaderboardIterator.java
- LeaderboardTester.java

Additionally, if you used generative AI at any point during your development, ***you must include screenshots*** showing your FULL interaction with the tool(s).

Your score for this assignment will be based on the submission marked "**active**" prior to the deadline. You may select which submission to mark active at any time, but by default this will be your most recent submission.

## Copyright notice