

Homework 2 - Abstract Classes and Programming with Lists

Due Tuesday by 11:59pm **Points** 0

Available Nov 1 at 11am - Nov 12 at 11:59pm 12 days

Before you begin, please make sure you are familiar with the guidelines discussed on the [Expectations on Homework](#) page, the "Homework Assignments" section of the [CS 2102 FAQ Page](#), and the FAQ at the bottom of this page.

Assignment Goals

- To be able to share data/code among classes using abstract classes
- To use the Java LinkedList class
- To become familiar with element-based for loops

Reminders

- **Please use the default package in your Java project.** There will be a penalty for using a named package.
- Starting with this assignment, please include a Javadoc statement above each method. There will be a penalty for forgetting your Javadoc statements.

Helpful Videos

This assignment uses LinkedLists, for-each loops, and Javadocs, which are concepts we have not yet covered in class as of 11/1. We will go over these on 11/4 and 11/5, but if you want a head start, you can watch the following recordings from similar lectures given two years ago. These recordings are listed in the recommended viewing order:

1. [Overloading](https://video.wpi.edu/Watch/CS2102_D20_L06-01) [_ \(https://video.wpi.edu/Watch/CS2102_D20_L06-01\)](https://video.wpi.edu/Watch/CS2102_D20_L06-01)
2. [Mutators](https://video.wpi.edu/Watch/CS2102_D20_L06-02) [_ \(https://video.wpi.edu/Watch/CS2102_D20_L06-02\)](https://video.wpi.edu/Watch/CS2102_D20_L06-02)
3. [Lists and Chains](https://video.wpi.edu/Watch/CS2102_D20_L06-03) [_ \(https://video.wpi.edu/Watch/CS2102_D20_L06-03\)](https://video.wpi.edu/Watch/CS2102_D20_L06-03)
4. [LinkedLists](https://video.wpi.edu/Watch/CS2102_D20_L06-04) [_ \(https://video.wpi.edu/Watch/CS2102_D20_L06-04\)](https://video.wpi.edu/Watch/CS2102_D20_L06-04)
5. [For-Each Loops](https://video.wpi.edu/Watch/CS2102_D20_L06-05) [_ \(https://video.wpi.edu/Watch/CS2102_D20_L06-05\)](https://video.wpi.edu/Watch/CS2102_D20_L06-05)
6. [Accumulators](https://video.wpi.edu/Watch/CS2102_D20_L06-06) [_ \(https://video.wpi.edu/Watch/CS2102_D20_L06-06\)](https://video.wpi.edu/Watch/CS2102_D20_L06-06)
7. [Javadocs](https://video.wpi.edu/Watch/CS2102_D20_L08-01) [_ \(https://video.wpi.edu/Watch/CS2102_D20_L08-01\)](https://video.wpi.edu/Watch/CS2102_D20_L08-01)
8. [LinkedList equality](https://video.wpi.edu/Watch/CS2102_D20_L08-02) [_ \(https://video.wpi.edu/Watch/CS2102_D20_L08-02\)](https://video.wpi.edu/Watch/CS2102_D20_L08-02)

LinkedList API: <https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html>
[\(https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html\)](https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html)

Note to those with prior Java experience: One goal of 2102 is to help everyone learn when different iteration constructs (for, while, etc) are needed for a particular problem. Style grading will check whether you are using appropriate constructs. This week, we will cover the per-element style for loop, not the for loop that uses a variable to index into elements. For full points, do not use index-based for loops on this assignment. Use a per-element style loop instead.

Problem Description and Context

For last week's homework, you developed classes for contestants and matches. Now we want to build off of those to create *tournaments*. A tournament is a set of *rounds*, each of which in turn consists of a set of *matches*.

There are two kinds of rounds: *initial* rounds occur at the beginning of the tournament (like how every team plays at least three games in the FIFA World Cup); *advanced* rounds occur later in the tournament and involve contestants who advanced from earlier matches in the tournament (such as the quarterfinals, semi-finals, and finals in the World Cup).

We are going to capture tournaments in Java in a way that reuses the classes and interfaces you created for homework 1 (you are welcome to update or fix your work from homework 1 as needed).

Programming Problems

1. Starting with your Homework 1 submission, develop an additional set of classes and interfaces that capture the concepts of rounds and tournaments. In order to let us run tests against your code, everyone needs to use standard names for interfaces and classes. Use the following:
 - `InitialRound`, `AdvancedRound`, and `AbsRound` for rounds
 - `Tournament` for the tournament class
 - Also create an interface `IWinner` for reasons that will be explained farther down.


The class constructor signatures should look like the following (parameter names may vary, of course):

```
public InitialRound(LinkedList<Match> matches) {  
    ...  
}  
  
public AdvancedRound(LinkedList<Match> matches, LinkedList<IContestant> contestants) {  
    ...  
}  
  
public AbsRound(LinkedList<Match> matches) {  
    ...  
}  
  
public Tournament(LinkedList<IWinner> rounds) {  
    ...  
}
```

2. Both `InitialRound` and `AdvancedRound` have a `LinkedList` of *matches* that contains all of the matches played in that round. `AdvancedRound` also has a list of winners from the previous round.
3. Both `InitialRound` and `AdvancedRound` have a method `getMatchWinners()` that returns a `LinkedList` of all of the *contestants* that won each match in each round.
4. Both `InitialRound` and `AdvancedRound` have a method `getNumWinners()` that returns the number of winners in the round. Think about how you could use the a call to the `getMatchWinners()` method in this method.
5. `AdvancedRound` has a method `isWinner()` that takes in an `IContestant` and returns whether that contestant was one of the winners from the previous round. This makes sure that there is nobody advancing who shouldn't be advancing.
6. `InitialRound` also has a method `isWinner()` that takes in an `IContestant` and determines whether that contestant is a winner by checking to see that the contestant was a winner in at least one of the matches that makes up that round.
7. `AbsRound` is an abstract class that holds fields and method implementations common to both `InitialRound` and `AdvancedRound`. Move any common fields and method implementations from `InitRound` and `AdvancedRound` into `AbsRound`.
8. `Tournament` has a single `LinkedList` of all of the rounds (both initial and advanced) that make up the tournament.
9. `Tournament` has a method `finalWinnerIsValid()` that takes in a contestant representing the tournament winner and checks whether that contestant is a valid winner. A contestant is a valid winner if he or she has won more than half of the rounds in the tournament. Use the `isWinner()` methods in `InitialRound` and `AdvancedRound` and the `IWinner` interface to solve this problem.
10. Create a test suite for your work. Put all of your tests and test data in a class called `Examples`. **Your class must be called this so that the auto-grader can find it!**

Just because your program passes all of your JUnit tests does not guarantee that it will pass all of ours. However, you can minimize the risk that your methods will fail our JUnit tests by sticking with the proper naming conventions and writing as many edge cases as possible. The more testing you do, the less the likelihood of failure!

Support Files

Here is a [CompileCheck.java](https://canvas.wpi.edu/courses/27993/files/4197680/download?download_frd=1)  (https://canvas.wpi.edu/courses/27993/files/4197680/download?download_frd=1) file that attempts to create objects from the expected classes and call the expected methods within those classes. Including this file when you compile will check that you have the class and method names that our grading tools expect, which saves you from losing points.

If you get a compilation error involving this file on a class or method that you defined, **do not edit this file. Edit your files instead!** As you are working, you may wish to comment out sections of the file that check methods you haven't written yet (that's fine). The final work you turn in should, however, compile against the entire contents of this file.

Just because your programs compiles does not mean that your program will pass all of the auto-grader tests! A program can compile but still be full of errors. The CompileCheck file is there only to make sure your naming conventions are correct, not that your classes and methods work correctly.

You are welcome to leave this file in the directory when you submit your work. It can serve as the main method for your program.

Grading

[Homework 2 Grading Rubric](https://canvas.wpi.edu/courses/27993/files/4197679/download?download_frd=1)  (https://canvas.wpi.edu/courses/27993/files/4197679/download?download_frd=1)

Here are some details on what we will look for in grading this assignment:

- Did you create classes with the fields required in the problem?
- Do your methods produce the answers expected based on the problem statements?
- Is your code neatly indented and presented in a clean, readable manner?
- Are your test cases correct relative to the problem statement?
- Are your test cases thorough, covering different situations (based on input data) and exercising all of your code?

Programs must compile in order to receive credit. If you submit a program that doesn't compile, the grader will notify you and give you one chance to resubmit within 24 hours; a penalty (25% of the total points for the assignment) will be applied as a resubmission penalty. Code that is commented out will not be graded.

What to Turn In

Submit (via [InstructAssist](https://ia.wpi.edu/cs2102/) [\(https://ia.wpi.edu/cs2102/\)](https://ia.wpi.edu/cs2102/)) a single zip file (not tar, rar, 7z, etc) containing all of your .java files that contain your classes, interfaces, and examples for this assignment. Do not submit the .class files. You may put all of your other classes and interfaces either into a single file or into separate ones (as you prefer). If you have separate src and test subdirectories, you may retain that structure in your zip file.

Make sure all of your tests are in separate files from your code, as explained in the [Expectations on Homework](#) page.

FAQs

Q: Should we override the equals() method for this assignment?

A: DO NOT write your own equals() methods for Homework 2. We will discuss how to do this in class. However, this will be just to show that we can create our own definition of equality if we wanted. You should still use Java's definition of equality for LinkedLists when writing your unit tests in Homework 2.

Q: How do I write a unit test for a method that returns a LinkedList?

A: Check out the lectures posted at the top of this assignment for examples of this.

Q: Do we need to write Javadocs for all of our methods and classes from Homework 1?

A: No. You only need Javadocs for methods and classes you write for Homework 2.

Q: Should we keep our unit tests from Homework 1?

A: It's a good idea to do so as a check that you didn't accidentally modify any of your Homework 1 methods. However, we will only be evaluating your unit tests specific to Homework 2.

Q: Should we correct any errors in our Homework 1 code?

A: Yes. Although we won't be evaluating your Homework 1 code as part of your Homework 2 grade, your Homework 2 methods rely on your Homework 1 methods.

Q: Is it okay to override the Homework 1 CompileCheck file with the Homework 2 CompileCheck file?

A: Absolutely! The Homework 2 file contains all of the stuff from Homework 1 as well as additional instantiations and method calls for Homework 2. (You ARE using the CompileCheck file in your homework assignments, right?)

Q: I'm not sure why I'm getting this error in my code.

A: You are welcome to use office hours/Slack/the IA forum for help, but before you do, please try using the debugging tools that I demonstrated in class. In addition, here are some helpful tutorials:

[Debugging in Eclipse](https://www.vogella.com/tutorials/EclipseDebugging/article.html) [_\(https://www.vogella.com/tutorials/EclipseDebugging/article.html\)](https://www.vogella.com/tutorials/EclipseDebugging/article.html)

[Debugging in Eclipse \(Video\)](https://www.youtube.com/watch?v=jybuG6QrLQg) [_\(https://www.youtube.com/watch?v=jybuG6QrLQg\)](https://www.youtube.com/watch?v=jybuG6QrLQg)

[Debugging in IntelliJ](https://www.jetbrains.com/help/idea/debugging-your-first-java-application.html) [_\(https://www.jetbrains.com/help/idea/debugging-your-first-java-application.html\)](https://www.jetbrains.com/help/idea/debugging-your-first-java-application.html)

[Debugging in IntelliJ \(Video\)](https://www.youtube.com/watch?v=IAWnIP1S6UA) [_\(https://www.youtube.com/watch?v=IAWnIP1S6UA\)](https://www.youtube.com/watch?v=IAWnIP1S6UA)

Q: Why do I get errors when I try to add an object to a LinkedList?

A: Make sure your calls to add() happen inside of a method body. If this is in the context of writing your unit tests, the calls to add() should be inside your Examples constructor or an @Before method (if you want the object to be added to your list for all tests) or inside your JUnit test method (if you want the object to be added to your list for just one test).

Q: Does the list of winners in `AdvancedRound` need to reflect the actual winners from `InitialRound`?

A: No. You can and should treat `InitialRound` and `AdvancedRound` independently for this assignment. Don't worry about making the data try to line up. Similarly, for your list of rounds in `Tournament`, the individual rounds don't actually need to be related to one another.

Q: Do I need to look at the list of previous winners in `AdvancedRound` as part of my `getMatchWinners()` implementation?

A: No. The `getMatchWinners()` method should only care about the winner of each match in the list of matches.

Q: I'm getting a `NullPointerException` error.

A: There are a couple of common causes for this. First, for this assignment, you do not need to worry about whether `winner()` in the `Match` class is returning a null value, so don't create test objects such that a null value is returned. Second, make sure that any fields in `AbsRound` aren't also in `InitialRound` or `AdvancedRound`. If there are, it may mean you have undefined fields in the latter two classes overriding a common field in `AbsRound`.