Homework 2 - Abstract Classes and Programming with Lists

Due Nov 5, 2019 by 6pm **Points** 0

Before you begin, please make sure you are familiar with the guidelines discussed on the Expectations on Homework page and the "Homework Assignments" section of the CS 2102 FAQ 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 statemnt above each method. There will be a penalty for forgetting your Javadoc statements.

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)

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.

We will have covered the material for this assignment during the second week of classes.

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:

    - `InitRound`, `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 InitRound(LinkedList<Match> matches) {
         ...
    }

    public AdvancedRound(LinkedList<Match> matches, LinkedList<IContestant> contestants) {
         ...
    }

    public AbsRound(LinkedList<Match> matches) {
         ...
    }
    public Tournament(LinkedList<IWinner> rounds) {
         ...
    }
    ```

2. Both `InitRound` 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 `InitRound` and `AdvancedRound` have a method `getMatchWinners()` that returns a LinkedList of all of the *contestants* that won each match in each round.

4. Both `InitRound` 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. `InitRound` 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 `InitRound` and `AdvancedRound`

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 at least half of the rounds in the tournament. Use the `isWinner()` methods in `InitRound` 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 are a couple of files that may be helpful.

- a skeletal **Examples.java** **(https://canvas.wpi.edu/courses/16466/files/2217271/download?download_frd=1)** file showing you the shape of a test case. You should create your own Examples.java file by right-clicking your project in Eclipse and going to New -> JUnit Test Case. **Make sure JUnit 4 is selected!**

- a **CompileCheck.java** **(https://canvas.wpi.edu/courses/16466/files/2261339/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/16466/files/2260980/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/)**) a single zip file (not tar, rar, 7zip, 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.