COM S 227
Fall 2019
Homework 2: Anyone for Tennis?

200 points

Due Date: Saturday, October 12, 11:59 pm (midnight)
5% bonus for submitting 1 day early (by 11:59 pm October 11)
10% penalty for submitting 1 day late (by 11:59 pm October 13)
No submissions accepted after October 13, 11:59 pm

General Information

This assignment is to be done on your own. See the Academic Dishonesty policy in the syllabus, http://www.cs.iastate.edu/ cs227/syllabus.html#ad , for details.
You will not be able to submit your work unless you have completed the *Academic Dishonesty policy acknowledgement* on the Homework page on Canvas. Please do this right away.
If you need help, see your instructor or one of the TAs. Lots of help is also available through the Piazza discussions.

Please start the assignment as soon as possible and get your questions answered right away. It is physically impossible for the staff to provide individual help to everyone the night that the assignment is due!

This is a "regular" assignment so we are going to read your code. Your score will be based partly (about a third) on the specchecker's functional tests and partly on the grader's assessment of the quality of your code. See the "More about grading" section.

Tips from the experts: How to waste a lot of time on this assignment

- Start the assignment the day it's due. That way, if you have questions, the TAs will be too busy to help you and you can blame the staff when you get a bad grade.
- Don't bother reading the rest of this document, or even the specification, especially the "Getting started" section. Documentation is for chumps.
- Don't test your code. It's such fun to remain in suspense until you get your score!
- The main guiding principle is: *Try to write all the code before you figure out what it's supposed to do.*

Overview

The goals of this assignment are to give you more experience writing classes from specifications and to provide you with practive working with conditionals.

For this assignment you will implement one class, called `TennisMatch`, that models the serving, end-changing, and scoring of a best-of-three or best-of-five, advantage sets tennis match. Your class will keep track of scores, server, ends, and faults, games, sets, and match.

Additionally, provided for you is a `TennisPlayer` class. Using it reduces some of the complexity of `TennisMatch`. You should use—but should not modify—`TennisPlayer`.

Specification

The specification for this assignment includes this pdf, the `TennisMatch` javadocs, the official rules of Tennis (the 2019 *Friend of the Court*, linked on Piazza) along with any "official" clarifications posted on Piazza.

Where's the main() method??

There isn't one! Like most Java classes, this isn't a complete program and you can't "run" it by itself. It's just a single class, that is, the definition for a type of object that might be part of a larger system. To try out your class, you can write a test class with a main method such the example provided (on Piazza) in `TestTennisMatch.java`.

There will also be a SpecChecker (not ready yet) that will perform a lot of functional tests, but when you are developing and debugging your code at first you'll always want to have some simple test cases of your own as in the main method above.

Note that neither the SpecChecker nor the provided test program nor the combination of the pair provide complete, comprehensive test coverage of your `TennisMatch` implementation. You will have to write your own tests.

Sample usage

A good way to think about the specification is to try to write some simple test cases and think about what behavior you expect to see. The included `TestTennisMatch` class implements a main method which tests much—but not all—of the functionality. It plays a complete first set, then modifies the score to "jump" to the last set and finish the match.

There is also a SpecChecker (see below) that will perform a lot of functional tests, but when you are developing and debugging your code at first you'll always want to have some simple test cases of your own as in the `main` method in `TestTennisMatch.java`.

Suggestions for getting started

Smart developers don't try to write all the code and then try to find dozens of errors all at once; they work *incrementally* and test every new feature as it's written. Here is a rough guide for how an experienced coder might go about creating a class such as this one:

- Create a new, empty project and add a package called `hw2`.
- Create the `TennisMatch` class in the `hw2` package and put in stubs for all the required methods and constructors. For methods that are required to return a value, just put in a "dummy" return statement that returns zero or `false`.
- Download the specchecker, import it into your project as you did in labs 1 and 2, and run it. There will be lots of error messages appearing in the console output, since you haven't actually implemented the methods yet. Always start reading from the top. All you really want to check at this point is whether you have a missing or extra public method, if the method declarations are incorrect, or if something is really wrong like the class having the incorrect name or package. Any such errors will appear first in the output and will usually say "Class does not conform to specification."
- Look at each method. Mentally classify it as either an *accessor* (returns some information without modifying the object) or a *mutator* (modifies the object, usually returning `void`). The accessors will give you a lot of hints about what instance variables you need.
- Before you write code for a method, always write a simple usage example or test case, similar to the `main` method in the provided test code. This will make sure you understand what the code is really supposed to do, and it will give later you a way to check whether you did it correctly. Of course, if you are really not sure what a method is supposed to do, bring up your question for discussion on Piazza!
- It would probably be best to start with the accessors, as they'll help you ascertain what instances variables to expect. The mutators should also be fairly straightforward and provide insights into the necessary instance variables.

- The bulk of the work occurs in `increment*Points`, where `*` is one of `Game`, `Set`, and `Match`. These must not only increment the particular point, but also control other aspects of game start, like switching server, switching ends, and ending the game. Furthermore, `incrementGamePoints` must call `incrementSetPoints` under certain conditions, and that in turn calls `incrementMatchPoints` under other conditions.

Friend of the Court

The rules of tennis that we will be implementing in this simulation are restricted to only a few sections of the Friend of the Court. These sections should be considered part of the specification of this assignment:

- Section 5a for scoring of games
- Section 6a for scoring of sets
- Section 7 for scoring of matches
- Section 10 for changing of ends
- Section 14 for order of service
- Section 23 for description of *the let*

Additionally, it may be benefitial to read other sections of the Friend of the Court for context and terminology; however, we will not be implementing rules described outside of the above sections. For instance, we will not be implementing tie-break sets, described in Section 6b and elsewhere.

Additional notes

- No loops are needed—nor likely helpful—for this assignment. You will, however, need many conditional expressions.
- Do not add any additional `public` methods; you may add your own methods if you feel compelled to do so, but they must be declared `private`. Do not create any additional Java classes.

The SpecChecker

You can find the SpecChecker online; see the Piazza Homework post for the link. Import and run the SpecChecker just as you practiced in Labs 1 and 2. It will run a number of functional tests and then bring up a dialog offering to create a zip file to submit. Remember that error messages will appear in the console output. There are many test cases so there may be an overwhelming number of error messages. *Always start reading the errors at the top and make incremental corrections in the code to fix them*. When you are happy with your results, click "Yes" at the dialog to create the zip file. See the document "SpecChecker HOWTO", which can be found in the Piazza pinned messages

More about grading

This is a "regular" assignment so we are going to read your code. Your score will be based partly (about a third) on the specchecker's functional tests and partly on the grader's assessment of the quality of your code. This means you can get partial credit even if you have errors, and it also means that even if you pass all the specchecker tests you can still lose points. Are you doing things in a simple and direct way that makes sense? Are you defining redundant instance variables? Some specific criteria that are important for this assignment are:

- Use instance variables only for the "permanent" state of the object, use local variables for temporary calculations within methods.
  - You will lose points for having lots of unnecessary instance variables

  - All instance variables should be `private`.
- **Accessor methods should not modify instance variables**.

See the "Style and documentation" section below for additional guidelines.

Style and documentation

Roughly 15% of the points will be for documentation and code style. Here are some general requirements and guidelines:

- Each class, method, constructor and instance variable, whether public or private, must have a meaningful and complete Javadoc comment. Class javadoc must include the `@author` tag, and method javadoc must include `@param` and `@return` tags as appropriate.
  - Try to state what each method does in your own words, but there is no rule against copying and pasting the descriptions from this document or from the posted javadoc.
  - Run the javadoc tool and see what your documentation looks like! You do not have to turn in the generated html, but at least it provides some satisfaction :)
- All variable names must be meaningful (i.e., named for the value they store).
- Your code should not be producing console output. You may add `println` statements when debugging, but you need to remove them before submitting the code.
- Internal (//-style) comments are normally used inside of method bodies to explain how something works, while the Javadoc comments explain what a method does. (A good rule of thumb is: if you had to think for a few minutes to figure out how something works, you should probably include a comment explaining how it works.)
  - Internal comments always precede the code they describe and are indented to the same level. In a simple homework like this one, as long as your code is straightforward and you use meaningful variable names, your code will probably not need many internal comments.
- Use a consistent style for indentation and formatting.
  - Note that you can set up Eclipse with the formatting style you prefer and then use Ctrl-Shift-F to format your code. To play with the formatting preferences, go to Window→Preferences→Java→Code Style→Formatter and click the New button to create your own profile for formatting.

If you have questions

For questions, please see the Piazza Q & A pages and click on the folder `assignment2`. If you don't find your question answered, then create a new post with your question. Try to state the question or topic clearly in the title of your post, and attach the tag `assignment2`. *But remember, do not post any source code for the classes that are to be turned in*. It is fine to post source code for general Java examples that are not being turned in. (In the Piazza editor, use the button labeled "pre" to have Java code formatted the way you typed it.)

If you have a question that absolutely cannot be asked without showing part of your source code, make the post "private" so that only the instructors and TAs can see it. Be sure you have stated a specific question; vague requests of the form "read all my code and tell me what's wrong with it" will generally be ignored.

Of course, the instructors and TAs are always available to help you. See the Office Hours section of the syllabus to find a time that is convenient for you. We do our best to answer every question carefully, short of actually writing your code for you, but it would be unfair for the staff to fully review your assignment in detail before it is turned in.

Any posts from the instructors on Piazza that are labeled "Official Clarification" are considered to be part of the spec, and you may lose points if you ignore them. Such posts will always be placed in the Announcements section of the course page in addition to the Q & A page. (We promise that no official clarifications will be posted within 24 hours of the due date.)

What to turn in

Please submit, on Canvas, the zip file that is created by the SpecChecker. The file, `SUBMIT_THIS_hw2.zip`, will be located in the directory you selected when you ran the SpecChecker. It should contain one directory, `hw2`, which in turn contains one file, TennisMatch.java. Please LOOK at the file you upload and make sure it is the right one!

Submit the zip file to Canvas using the Assignment 2 submission link and verify that your submission was successful. If you are not sure how to do this, see the document "Assignment Submission HOWTO" which can be found in the Piazza pinned messages.

We recommend that you submit the zip file as created by the specchecker. If necessary for some reason, you can create a zip file yourself. The zip file must contain the directory `hw2`, which in turn should contain the file `TennisMatch.java`. You can accomplish this by zipping up the `src` directory of your project. The file must be a zip file, so be sure you are using the Windows or Mac zip utility, and NOT a third-party installation of WinRAR, 7-zip, or Winzip