Com S 227
Spring 2017
Assignment 2
220 points

Due Date: Friday, February 17, 11:59 pm (midnight)

NO LATE SUBMISSIONS

(Remember that Exam 1 is MONDAY, February 20.)

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 **, 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 Blackboard. 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.

Note: Our first exam is Monday, February 20, which is just three days after the due date for this assignment. ***It will not be possible to have the assignments graded and returned to you prior to the exam****. We can post a sample solution on February 18.*

Please start the assignment as soon as possible and get your questions answered right away!

Introduction

The purpose of this assignment is to give you some practice working with conditional statements. In addition, it will serve as an example of how a Java class might interact with other Java classes to form a complete application.

For this assignment you will implement one class, called `FootballGame`, that encapsulates the state of a highly simplified form of the charming and popular game known as "American

football". If you know nothing about football, never fear, neither do we! We just looked it up on Wikipedia! Although the real-life game includes many complex and byzantine rules, we are going to ignore most of them. For a precise explanation of the simplified rules that we are using for this assignment, and the exact behavior of the `FootballGame` methods, see the online javadoc:

http://web.cs.iastate.edu/~cs227/homework/hw2/doc/

What's the rest of that stuff described in the Javadoc??

For this assignment, your job is to implement only the `FootballGame` class. The Javadoc also describes three other classes in a package called `example`: `Team`, `OutcomeGenerator`, and `UI`. These are provided just to illustrate how a class like `FootballGame` might form part of a larger system, and to give you a way to play with it after you get it implemented. The source code for these three classes is provided for you. **You are not required to read them or use them** to complete this assignment, but you might find them interesting.

The sample code also includes an incomplete skeleton for `FootballGame` that has just the required constant declarations. Note that the *UI code will not compile* until you have at least written stubs for the required constructor and methods of `FootballGame`. Once your `FootballGame` class is implemented and tested, you can try out the UI just by running it (the `UI` class has a `main` method).

Importing the sample code

The sample code includes a partial skeleton of the `FootballGame` class. It is distributed as a complete Eclipse project that you can import. However, **the UI code will not compile** until you have added stubs for the required methods of `FootballGame`. See the "getting started" section. General instructions for importing an Eclipse project:

1. Download the zip file to a location outside your workspace. You don't need to unzip it.
2. In Eclipse, go to File -> Import -> General -> Existing Projects into Workspace, click Next.
3. Click the radio button for "Select archive file".
4. Browse to the zip file you downloaded and click Finish.

Alternate procedure: If you have an older version of Java (below 8) or if for some reason you have problems with this process, or if the project does not build correctly, you can construct the project manually as follows:

1. Unzip the zip file containing the sample code (you have to actually extract the files, not just double click on the zip file to see them)
2. In Windows File Explorer, or OS X Finder, browse to the src directory of the zip file contents
3. Create a new empty project in Eclipse
4. In the Package Explorer, navigate to the src folder of the new project.
5. Drag the **`hw2`** and **`example`** folders from Explorer/Finder into the **`src`** folder in Eclipse.

Testing and the SpecCheckers

As always, you should try to work incrementally and write tests for your code as you develop it. One of the best ways to start implementing a method is to first write a simple test case or usage example, so that you have absolutely no doubt as to what the code should do. There are several examples shown in the "getting started" section.

More often than not, when you start to write simple tests like the ones shown, you'll have questions about what a method is really supposed to do. Since test code that you write is not a required part of this assignment and does not need to be turned in, **you are welcome to post your test code on Piazza for others to check, use and discuss.**

The specchecker, like the one from Assignment 1, will run a number of functional tests. You will nonetheless find it extremely worthwhile to write simple test cases like the ones above as you develop your code. **Additional test cases may be added when the assignments are graded.**

More about grading

Be sure you have checked out the feedback from the grader on your Assignment 1.

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.
    - 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.

- Do not embed numeric literals in your code (except 0 and 1 for the team numbers). Use the defined constants wherever appropriate.
- 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.
- 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, and **for this assignment you are welcome to post and discuss test code**. (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.)

Suggestions for getting started

1. Create a new Eclipse project and within it create a package **`hw2`**.

2. Go ahead and create the **`FootballGame`** class in the **`hw2`** package, or download the skeleton from the sample code. Add the constant declarations and put in stubs for all the methods and constructors described in the Javadoc. For methods that need to return a value, just return a "dummy" value as a placeholder. At this point there should be no compile errors in the project.

3. Import and run the first specchecker. *Start reading the output at the top*. Make sure you don't have any errors of the form "MISSING CLASS" or "Class does not conform to specification".

4. Javadoc the classes and methods. This is a required part of the assignment, and doing it now will help clarify for you what each method is supposed to do before you begin the actual implementation.

- Copying method descriptions from the online Javadoc is perfectly acceptable, though **you are not required to be as detailed** as the online documentation. You'll need to fill in the @param and @return tags for completeness.
- Don't forget to add a brief Javadoc to the class itself, with an @author tag with your name

5. Now you'll need to start thinking about instance variables. Looking at the accessor methods often gives you clues about what information needs to be stored in the object. For example, how will you implement **`getYardsToGoalLine`** ? The current location of the ball will have to be stored somehow in an instance variable. Make sure you initialize it correctly in the constructor. This is easy to illustrate and check with a simple test case:

```
FootballGame game = new FootballGame();
System.out.println(game.getYardsToGoalLine());  // expected 70
```

Other obvious cases include **`getScore`**, **`getDown`**, and **`getOffense`**. Go ahead and add relevant instance variables and implement these accessors.

6. The most complex method is **`runOrPass`**, since you can tell from the Javadoc that there are a lot of cases to consider. So instead, it might make sense to think about a simpler method first and make sure you get it working correctly. The method **`punt`** is pretty simple. The ball moves forward the given amount, and then the other team becomes the offense. State by writing a simple test case such as this,

"After a punt, the other team becomes the offense."

```
FootballGame game = new FootballGame();
System.out.println(game.getOffense());          // expected 0
game.punt(50);
System.out.println(game.getOffense());          // expected 1
```

"After a 50 yard punt from the 70 yard line, the other team becomes the offense with 80 yards to the goal."

```
FootballGame game = new FootballGame();
System.out.println(game.getOffense());          // expected 0
System.out.println(game.getYardsToGoalLine());  // expected 70
game.punt(50);
System.out.println(game.getOffense());          // expected 1
System.out.println(game.getYardsToGoalLine());  // expected 80
```

(Aside: if you use an integer 0 or 1 to represent which team is the offense, you can switch between them easily with a line of code such as

```
theOffense = 1 - the Offense;
```

If you're using a boolean variable, you could do it like this with the "not" operator,

```
offenseIsTeam0 = !offenseIsTeam0;
```
)

7. Maybe next try **`fieldGoal`**, which is similar to **`punt`**, but also may update the score if the "success" parameter is true. Again, try a couple of scenarios in a simple test case:

"After a failed field goal, the other team becomes the offense with the ball in its current location."

```
FootballGame game = new FootballGame();
game.fieldGoal(false);
System.out.println(game.getOffense());          // expected 1
System.out.println(game.getYardsToGoalLine()); // expected 30
```

"After a successful field goal, the other team becomes the offense with the ball 70 yards from the goal line."

```
game = new FootballGame();
game.fieldGoal(true);
System.out.println(game.getOffense());          // expected 1
System.out.println(game.getYardsToGoalLine()); // expected 70
System.out.println(game.getScore(0));           // expected 3
```

The method **`extraPoint`** is similar too. (Although it is expected that **`extraPoint`** is only called by clients after a touchdown, there is no mechanism to enforce this, and the method should still work as expected.)

8. Now you'd better start thinking about **`runOrPass`**. The first thing to think about is updating the ball's location. At first, don't worry about counting "downs", just check that the location is updating. For example, if the offense runs or passes 5 yards, what should you observe about the ball's location?

"After running or passing 5 yards, the ball should be 5 yards closer to the goal".

```
FootballGame game = new FootballGame();
System.out.println(game.getYardsToGoalLine());  // expected 70
game.runOrPass(5); // advance the ball 5 yards
System.out.println(game.getYardsToGoalLine());  // expected 65
```

9. Next, check for a touchdown (distance to goal is negative), and adjust the score accordingly. (You don't switch the offense or adjust the ball location after a touchdown, since the client is supposed to call **`extraPoint`** to handle that.)

10. Finally, add a mechanism for counting downs. At this point you'll need to figure out how to detect whether the ball has advanced 10 yards since the first down. If so, the offense gets to start another first down. If not, the other team gets the ball. Essentially, you are implementing **`getYardsToFirstDown`**. Depending on how you have set things up so far, you may need another

instance variable or two. (Remember that every time the other team becomes the offense, it starts with a first down.) For example, try a scenario like this:

```
FootballGame game = new FootballGame();
System.out.println(game.getDown());              // expected 1
System.out.println(game.getYardsToFirstDown()); // expected 10
game.runOrPass(-4);
System.out.println(game.getDown());              // expected 2
System.out.println(game.getYardsToFirstDown()); // expected 14
game.runOrPass(10);
System.out.println(game.getDown());              // expected 3
System.out.println(game.getYardsToFirstDown()); // expected 4
game.runOrPass(20);
System.out.println(game.getDown());              // expected 1
System.out.println(game.getYardsToFirstDown()); // expected 10
```

What to turn in

Note: You will need to complete the "Academic Dishonesty policy questionnaire," found on the Homework page on Blackboard, before the submission link will be visible to you.

Please submit, on Blackboard, the zip file that is created by the second SpecChecker. The file will be named **`SUBMIT_THIS_hw2.zip`**. and it 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, **`FootballGame.java`**.

Please LOOK at the zip file you upload and make sure it is the right one!

Submit the zip file to Blackboard using the Assignment 2 submission link and **verify that your submission was successful by checking your submission history page**. If you are not sure how to do this, see the document "Assignment Submission HOWTO" which can be found in the Piazza pinned messages under “Syllabus, office hours, useful links.”

> 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 **`FootballGame.java`**. Make sure all files have the extension **`.java`**, NOT **`.class`**. You can accomplish this easily by zipping up just the **src** directory of your project (NOT the entire 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.

Appendix - test scenarios used in the last few SpecChecker test cases

Scenario 1:

Team 0 runs 4 yards at a time to -2, and makes extra point
Team 1 runs 4 yards at a time to -2, and makes extra point
Team 0 runs to 20 yard line and then makes field goal
Team 1 runs 4 yards at a time to -2 and makes extra point

Scenario 2:

Team 0 runs 4 yards at a time to -2 and makes extra point
Team 1 runs to 20 yard line and makes field goal
Team 0 runs 1 yard at a time x 4, first down for team 1
Team 1 runs to 20 yard line and misses field goal attempt
Team 0 runs 4 yards at a time to -4, misses extra point

Scenario 3:

Team 0 runs 4 yards at a time to -2, and makes extra point
Team 1 runs to 20 yard line and makes field goal
Team 0 punts 20 yards
Team 1 runs to 20 yard line, misses field goal attempt
Team 0 runs 4 yards at a time to -4, makes extra point
Team 1 punts 20 yards
Team 0 runs to 20 yard line, makes field goal
Team 1 runs to 20 yard line, misses field goal attempt
Team 0 runs 4 yards at a time to -4, makes extra point
Team 1 runs to 20 yard line, makes field goal
Team 0 punts 20 yards
Team 1 runs to 20 yard line, misses field goal attempt