Com S 227
Summer 2020
Assignment 2
200 points

Due Date: Tuesday, June 30, 11:59 pm (midnight)

Early deadline (5% bonus): Monday, June 29, 11:59 pm

Late deadline (10% penalty): Wednesday, July 1, 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.

Read this specification carefully. It is not quite like reading a story from beginning to end, and you will probably have to reread some parts many times. See the section "If you have questions" if you find something unclear in this document.

Introduction

In this homework you will get some practice implementing a small system that involves several interacting classes. The end result is a simple game called “hangman” that is likely to be familiar from grade school. A secret word or phrase is displayed with the characters hidden. Normally if the word or phrase contains a dash or punctuation, the non-letters are displayed and there is a marker like an underline to indicate where the characters are.

For example, the secret phrase "HELLO, WORLD!" might start out being displayed like this:

```
      ,       !
^^^^^  ^^^^^
```

The player tries to guess the letters. If a letter is guessed correctly, say “o”, it is written in the corresponding spaces.

```
     O,  O   !
^^^^^  ^^^^^
```

The goal is to guess all the letters in the word before running out of guesses. Traditionally, each time a wrong guess is made, A body part is drawn on a stick figure (head, arm, etc.) When the stick figure is complete, the player loses.

Overview of things to do

The parts of the system that you are to implement are the following:

1. The class **`Game`** is responsible for keeping track of the secret word or phrase, the letters that have been guessed, the number of wrong guesses, and whether the game is won or lost. Part of the state of a **`Game`** object is an array of **`HideableChar`**.
2. The class **`HideableChar`** is used to represent one letter in a secret word or phrase. In addition to encapsulating a character, it has a state of *hidden* or *not hidden*. This is useful so that as letters are guessed, they can be un-hidden to make them visible to a client.
3. The class **`PhraseSelector`** is responsible for choosing a random line from a text file.
4. A class **`HideableCharTest`** that is a JUnit test for the class **`HideableChar`**.

Sample user interfaces

Note that none of the classes above should read input or print output. All input or output is done through a user interface. Two user interfaces are provided for you in the sample code:

> **`TextUIMain`** is a very short main class that runs a text-based user interface. *You should read this code* because it will give you a good idea how the other classes are supposed to work and will give you an easy way to try them out. Basically the UI is constructed with a given

PhraseSelector, and it constructs **Game** instances as needed when the user wants to play a round of the game.

UIMain is a main class that starts up an instance of **HangmanUI**, a graphical interface for the game. This does roughly the same thing as the text UI but is more fun and provides a nice way to see how all the pieces come together to make a complete application. *It is **not** necessary for you to read and understand the* **HangmanUI** *code* (though you might find it interesting).

You should not modify any of the user interface code, other than to possibly change the name of the file to be read. The UI code is available in the zip file skeleton provided.

Note: the UI code won't compile until you create the first three required classes!

Detailed specification

See the javadoc online at http://www.cs.iastate.edu/~cs227/assignments/hw2/doc/

Notes

1. Your four classes should be in the package **hw2**.

2. None of the classes that you write should directly read input or print output. All input and output is performed by the user interface classes.

3. The current state of the secret word or phrase is returned to clients via the method **getDisplayedWord**, which returns an array of **HideableChar** in which letters that have not been guessed yet are hidden. Note that non-alphabetic characters are never hidden. It is up to the client to decide how to represent the hidden letters. ~~For example, in the sample text-based UI, each hidden letter is displayed on the console by putting a caret character ("^") beneath the space where it belongs. In the graphical UI, hidden letters are rendered by drawing a partial rectangle below each location.~~

4. The class **Game** must define this constant:

```
public static final int DEFAULT_MAX_WRONG_GUESSES = 7;
```

5. The classes **Game** and **HideableChar** are case sensitive. ~~It is up to the client code to determine whether to provide words and guesses in a case-sensitive way. (Both user interfaces always use upper-case letters.)~~

6. The input file for **PhraseSelector** should contain one word or phrase per line. Each word or phrase may contain arbitrary punctuation. You can create your own word files to try out your

code, and you are welcome to share them on Canvas. (It is common for a word file to have a "theme", e.g., a fruits and vegetables, kitchen utensils, sea animals, automobile parts, etc.) If you run your code from Eclipse, the word file must be located in the project directory. The name of the word file is specified in the class **`TextUIMain`** (for the text-based user interface) and in the class **`UIMain`** (for the graphical interface). Edit the filenames at your convenience.

7. **`PhraseSelector`** must use the given instance of Random as its source of randomness exactly as specified in the Javadoc.

8. Note that in **`PhraseSelector`**, both the constructor and the **`selectWord`** method include the clause "**`throws FileNotFoundException`**". This means that the method/constructor is *allowed* to throw the indicated exception, not that it *must* be thrown. What happens in practice will depend on how you actually implement the class. One option is to have the constructor just record the filename, and then open and read the file whenever **`selectWord`** is called. Another option, if you are comfortable working with **`ArrayList`**, might be to open and read the file in the constructor, storing the lines in an **`ArrayList`** of strings.

JUnit test for `HideableChar`

Each of your test cases should include a message stating specifically what is being tested. Each test case should be focused on checking one simple fact. (Most cases will include just one JUnit assertion.) Be reasonably complete. As in lab 5, one thing we are going to do in checking your JUnit test is to run it on HideableChar implementations with and without bugs.

Most importantly, be sure you are clear about what the specification says. If you have questions or want to double-check anything, please post on Piazza and describe the test cases you have in mind. Since the code is being turned in, you may not post the code, but you *may* share your test cases in words, e.g. "After constructing a HideableChar with character "A", isHidden should return true".

Specchecker

Your class must conform precisely to this specification. The most basic part of the specification includes the class names and package, the required constant, the public method names and return types, and the types of the parameters. We will provide you with a specchecker to verify that your class satisfies all these aspects of the specification and does not attempt to add any public attributes or methods to those specified. If your class structure conforms to the spec, you should see the following message in the console output:

```
4 out of 4 tests pass.
```

(Your instance variables should always be declared **`private`**, and if you want to add any additional "helper" methods, they must be declared **`private`** as well.)

The specchecker will also offer to create a zip file for you. Before you submit it, verify that it contains the four required classes.

The specchecker for this assignment WILL NOT perform any functional tests. In this assignment is up to you to test your code. As described above, you will be turning in a JUnit test for the class **`HideableChar`**, and we strongly encourage you to write additional functional tests, especially for the **`Game`** class. A user interface is provided, but you will quickly find that repeatedly trying out the code in a user interface is a very, very slow and inefficient way to test things.

Your tests for **`Game`** and/or **`PhraseSelector`** should be in separate classes from the required class **`HideableCharTest`**. Since they are not being turned in, you can share them with your friends and post on Piazza if you wish.

Style and documentation

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

- Each **class**, **method**, 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.
    - You can copy and paste from the posted javadoc if you wish.
    - 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 :)
    - **Special note regarding JUnit tests**: Your JUnit test should include an @author tag at the top. Beyond that, you do not have to javadoc the individual test cases, *provided* that each test case includes a string message stating what is being tested.

- All variable names must be meaningful (i.e., named for the value they store).

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

    - As you add instance variables to your class, *immediately* document exactly what each one represents.
    - All instance variables should be **`private`**.

- 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.
- Unless explicitly specified, your code should not be producing console output. Programmers often add **`println`** statements when debugging, but you need to remove them before submitting the code.

If you have questions

It is extremely likely that the specification will not be 100% clear to you. Part of your job as a developer is to determine what still needs to be clarified and to formulate appropriate questions.

For questions, please see the Piazza Q & A pages and click on the folder **`assignment2`**. If you don't find your question answered already, 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

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

Please submit, on Canvas, the zip file that is created by the 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 four files:

```
Game.java
HideableChar.java
HideableCharTest.java
PhraseSelector.java
```

You are responsible for checking that the zip file contains the correct files and for checking that your submission has been correctly uploaded to Canvas.

Submit the zip file to Canvas 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 strongly recommend that you just submit the zip file created by the specchecker. If you mess something up you will lose some points.

We strongly 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 four Java files listed above. You can accomplish this by zipping up the **src** directory of your project. It is ok if your submission includes extra files. 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.