Project 00: Word Guessing

Assigned: Monday, March 18, 2019
Due: Tuesday, April 9 at 11:30pm

1 Objective

One of the most common (and most fun!) uses of indefinite iteration is running a game. Most games end not because a certain number of turns have been taken, but because a player has achieved some goal. We are going to build a game in which one user types a secret word and then another guesses one letter at a time until they have uncovered the secret word. It is similar to the pencil-and-paper game Hangman and the television game show Wheel Of Fortune.

You will write the longest, most complicated program you have worked on so far. Along the way, you will have plenty of opportunities to practice solving indefinite iteration problems, including early exit problems, and will get more experience with processing strings.

2 Pair Programming

If you choose, you may work on this project with one other student, who must be in the same section as you (and must agree that they would like to be your partner). If you choose to work with a partner, you will only have one copy of the program, whch both of you will write together. That file will list both of you as authors, and you will both submit it to AutoLab when you are finished. You must use the pair programming technique, which is used even by experienced professionals when working on difficult problems.

In pair programming, you are always working together, never splitting up the work or working independently. You will only need one computer running Eclipse, where one person will be typing (the driver). While that person is writing the program, their partner (the navigator) is finding relevant examples, drawing diagrams, thinking about the bigger picture, and talking through the problem with the driver. Every 30 minutes or so, the partners should switch roles. You can read more about how to effectively pair program at `https://www.wikihow.com/Pair-Program`.

3 Setup

Continue using the same Eclipse workspace, but create another project named `Project00`. In it, create a class named `GuessingGame`.

Find the AutoLab page for `Project 00: Word Guessing`. As with the labs, it would be a good idea to submit your file after each question so that if there are problems with your work you can fix them before moving on.

4 Overview

While we will be writing one method at a time as usual, starting with the simplest methods and working our way outward toward `main`, it would be helpful to get a sense of what the entire program does before you get started. The next page shows a complete example run of the program.

Our program will need to remember the secret word, but also need to remember how much of it the player has already revealed. To do this, we will start with a string containing only underscore characters – one for each letter in the secret word. When the player guesses a letter that is in the secret word, we will replace the matching underscores with the guessed letter.

5 Guidelines

We have discussed (briefly) that programs that use both the `nextLine` method and the other methods of the `Scanner` class can be difficult to write correctly. In the past, our solution has been to avoid using the `nextLine` method. In this assignment, however, the solution is going to be to *only* use the `nextLine` method.

```
Welcome to my guessing game!

Enter the secret word: book
(... 30 blank lines ...)
Your knowledge so far: ____
Guess a letter: o
Your knowledge so far: _OO_
Guess a letter: k
Your knowledge so far: _OOK
Guess a letter: c
Your knowledge so far: _OOK
Guess a letter: l
Your knowledge so far: _OOK
Guess a letter: b
Congratulations, you discovered the word "BOOK"!
It took you 5 guesses.
Play again?: sure
Please enter 'yes' or 'no'.
Play again?: yes
Enter the secret word: hello world
A secret word may only contain letters.
Enter the secret word: hello
(... 30 blank lines ...)
Your knowledge so far: _____
Guess a letter: a
Your knowledge so far: _____
Guess a letter: e
Your knowledge so far: _E___
Guess a letter: o
Your knowledge so far: _E__O
Guess a letter: i
Your knowledge so far: _E__O
Guess a letter: l
Your knowledge so far: _ELLO
Guess a letter: h
Congratulations, you discovered the word "HELLO"!
It took you 6 guesses.
Play again?: no
Thanks for playing!
```

6 Assignment

Please complete each of the steps below.

1. (5 points) Write a method named `clearConsole` in the `GuessingGame` class. It should accept one parameter, which is a number of lines. It does not return anything. The method should print that many completely empty lines. We will later use this method to hide the answer from the player.

2. (5 points) Write a method named `getGuess` in the `GuessingGame` class. It should accept one parameter, which is a `Scanner` from which it can read input. It should return a character. It should print the prompt `"Guess a letter: "`, allow the user to type something in, and return the uppercase version of the first character typed. You might expect that the `Scanner` class would have a method named `nextChar`, but it does not. Instead, you will need to read an entire line from the user, convert it to uppercase, then return just its first character.

3. (15 points) Write a method named `createBlanksString` in the `GuessingGame` class. It should accept one parameter, which is a length. It should return a `String`. Specifically, the `String` it returns should contain only underscores – and the number of them is determined by the parameter. You can create such a `String` by starting with an empty `String` and repeatedly concatenating an underscore onto the end of it.

4. (15 points) Write a method named `replaceBlanks` in the `GuessingGame` class. It should accept three parameters (in this order):

   - A `String` that contains the secret word.
   - Another `String` that contains underscores for letters that have not yet been guessed, but also contains the letters that have already been revealed.
   - A character that has just been guessed.

   It should return a `String`. Specifically, it should return a version of the revealed guesses that includes the guessed letter wherever it should appear. Here are a few examples to make that clearer:

   - If the parameters were `"MISSISSIPPI"`, `"__________"`, and `'I'`, you would return `"_I__I__I__I"`.
   - If the parameters were `"MISSISSIPPI"`, `"_I__I__I__I"`, and `'X'`, you would return `"_I__I__I__I"`.
   - If the parameters were `"MISSISSIPPI"`, `"_I__I__I__I"`, and `'S'`, you would return `"_ISSISSI__I"`.

   Seeing this, you may say "we are replacing characters, so I'll use the `replace` method!". Unfortunately, it will not work because we do not want to replace all of the underscores – only those that correspond to the guessed letter. Instead, there are two good ways to solve this problem, and you should choose one of them:

- One option is to build the result string similarly to how you did in `createBlanksString`. For each position, you will decide to either copy from the first parameter or from the second.
- The other option is to start with a copy of the second parameter and go through it one position at a time. Whenever you find an underscore that should be replaced, you will replace the copy with all of the characters before this position, the guessed letter, and all of the characters after this position, concatenated together.

5. (10 points) This should not be attempted until after Lecture 12. Write a method named `wordContainsNonLetter` in the `GuessingGame` class. It should have one parameter, which is a `String`. It should return `true` if that `String` contains any characters that are not letters, or `false` if it does not. For example, if the parameter were `"book"` you would return `false`, but if the parameter were `"15th"` or `"some spaces"` then you would return `true`.

   To complete this method, you will need to know about the `Character.isAlphabetic` method, which you can use. This accepts one parameter, which is a `char`, and returns `true` if it is a letter or `false` otherwise.

   This method should use an early exit, and if it does not I will deduct some of the points that AutoLab gives you.

6. (10 points) This should not be attempted until after Lecture 13. Write a method named `getYesNoAnswer` in the `GuessingGame` class. It should accept two parameters (in this order):

   - A prompt that it will print.
   - A `Scanner` from which it can read input.

   It should return a `String`, which will be a response typed by the user. The method should print the prompt to the user and read their response. If the response is exactly `"yes"` or `"no"`, that value can be returned. But if not, the method must ask the user to type their response again, as many times as are necessary until it gets `"yes"` or `"no"`. Between receiving any incorrect responses and printing the prompt again, it should print `"Please enter 'yes' or 'no'."`.

   For example, if the first parameter to the method was `"Example prompt: "`, then the method might produce output like this and then return `"yes"`:

```
Example prompt: something
Please enter 'yes' or 'no'.
Example prompt: make me!
Please enter 'yes' or 'no'.
Example prompt: maybe
Please enter 'yes' or 'no'.
Example prompt: yes
```

   If the user were more cooperative, it would look like this and return `"no"`:

```
Example prompt: no
```

7. (10 points) This should not be attempted until after lecture 13. Write a method named `getSecretWord` in the `GuessingGame` class. It should have one parameter, which is a `Scanner` from which it can read input. It should return a `String` that the user types in as a new secret word.

   The method should print the prompt `"Enter the secret word: "`, read in a line of text, and convert it to uppercase. If this word only contains letters, it can be returned. If not, then we will have to re-prompt the user as many times as necessary for them to type a line containing only letters. Between receiving any incorrect responses and printing the prompt again, it should print `"A secret word may only contain letters."`.

   For example, if the user was bad at following instructions, then the method might produce output like this and then return `"LABROOM"`:

```
Enter the secret word: Caputo 130
A secret word may only contain letters.
Enter the secret word: Caputo One-Thirty
A secret word may only contain letters.
Enter the secret word: Caputo One Thirty
A secret word may only contain letters.
Enter the secret word: Labroom
```

   This method should use the `wordContainsNonLetter` method, and if it does not I will deduct some of the points that AutoLab gives you.

8. (10 points) This should not be attempted until all previous parts are working. Write a method named `play` in the `GuessingGame` class. It should accept one parameter, which is a `Scanner` from which it can read input. It should not return anything. It should do the following:

   - Get a secret word.
   - Create a blank revealed word of the same length.
   - Clear the console with 30 blank lines.
   - Repeat the following steps while the revealed word and secret word do not match:
     - Show the revealed word to the player (exact text below).
     - Get a guess.
     - Replace the revealed word with a version in which the guessed letter has been revealed.
   - Congratulate the player on discovering the word (exact text below).
   - Tell the user how many guesses they made (exact text below).

   Here is what a run of the `play` method might look like:

```
Enter the secret word: book
(... 30 blank lines ...)
Your knowledge so far: ____
```

```
Guess a letter: o
Your knowledge so far: _OO_
Guess a letter: k
Your knowledge so far: _OOK
Guess a letter: c
Your knowledge so far: _OOK
Guess a letter: l
Your knowledge so far: _OOK
Guess a letter: b
Congratulations, you discovered the word "BOOK"!
It took you 5 guesses.
```

This method should use the `getSecretWord`, `createBlanksString`, `clearConsole`, `getGuess`, and `replaceBlanks` methods, and if it does not I will deduct some of the points that AutoLab gives you.

9. (10 points) This should not be attempted until all previous parts are working. Finally, write a `main` method. It should print `"Welcome to my guessing game!"` and a blank line, then play a round of the game. Then it should ask whether or not the user wants to play again (with the prompt `"Play again?: "`) and continue playing the game as many times as they answer with `"yes"`. Finally, it should print `"Thanks for playing!"`. The beginning of these instructions shows a complete session of running the program.

   This method should use the `play` and `getYesNoAnswer` methods, and if it does not I will deduct some of the points that AutoLab gives you.

   Play and enjoy your game!

10. (10 points) AutoLab will now automatically check for Javadoc comments and good style. (It can't find all types of bad style, though, so I will still be checking some things.) Ensure that you are making it happy and that you think you will make me happy as well.

7 Style Guidelines

A well-written program does not simply produce the correct output. It also is written in a way that is easy for other programmers to read, understand, and modify, and that makes efficient use of the computer's resources. Here are some important style guidelines to remember that AutoLab does not automatically check:

- When possible, decisions should be structured to avoid the computer needing to make decisions to which the answer is already known.
- Indefinite iteration problems should never be solved using `for` loops.
- Every loop that could have an early exit should.
- Early exits should never be implemented using `return`, `break`, or `continue`.

8 Submitting Your Work

Make sure that you have submitted your final version to AutoLab, and that (unless you are satisfied with a lower score) you have 100/100 autograded points.

You do not need to submit your work through D2L – I will look at whatever your last AutoLab submission is.