Northeastern Illinois University
CS 207 Programming II
Research Lab: Hangman

Due: Sunday November 11, 11:59 PM

Goal:
The goal of this research lab is to modify the given Java files to investigate how to use and manipulate strings and various classes in JavaFX in order to create a playable Hangman game. As this is a "research" lab, you will need to analyze the code that has been provided for you in order to be able to modify it using the materials presented in class.

Project:
Hangman is a simple guessing game in which the player has the guess a mystery word letter by letter. If the player guesses a letter correctly, that letter is revealed in the word. If they guess incorrectly, a body part is added to the "hangman character" – first the head, then body, arms, and legs. The game repeats until either the word is fully revealed (won) or all body parts are drawn (lost).

Instructions:
- You should work in groups of 2-3 individuals. Groups of more than 3 are not permitted.
- Each group should submit ONE lab write-up. It is the responsibility of each group member to ensure that their name is on the write-up.
- The lab write-up should be typed! Type each question (and the question number) followed by your group's answer. **Convert your lab write-up to a .pdf.**
- You should use complete sentences and proper grammar in your write-up. Use spell-check! This counts as part of your grade.
- Do not copy/paste directly from your sources for your answers (this is called plagiarism). Instead, you should re-word the information in your own words.
- Submit the PDF and Java files in a ZIP file to D2L by the specified due date.
- Each member of the group must turn in a readable digital copy of the peer assessment to an individual Dropbox by the assigned due date and time. The peer assessment counts as a significant part of your grade and you will receive a zero for that portion of the research lab grade if you do not turn it in.

Part A: Getting Started

Question #A.1

Download the project files from D2L. There are two Java class files – `Dictionary` and `Hangman` – and a text file called `words.txt`. Open the `Dictionary` class, which has already been created for you. We use this class to hold a list of words (read in from the `words.txt` file – you will learn how to read files later this semester!). `Dictionary` also has a constant called `ALPHABET`, which contains all 26 letters, in alphabetical order.

List the names and visibility modifiers for each variable in `Dictionary`. How many constructors are there? For all methods in the class, list: the name of the method, its visibility modifier, its parameters (if any), and return type.

Question #A.2

Examine the instance variable – what type of object is it? Name 1-2 major ways in which this object is different than an array. What type of objects does this object hold?

Compile the Dictionary class.

Question #A.3 + coding

Now, open up `Hangman.java`. Look at the import statements, and notice that they all come from the JavaFX package. JavaFX is a set of packages used to create applications and event-driven programs. Applications are laid out a bit differently from the types of programs we have been writing so far. (To learn how to write applications, take CS-319!)

Notice the class header for `Hangman`. How is our class related to the `Application` class? (i.e. What keyword is used, and what does this keyword mean?)

Look up `Application` in the JavaFX 8 documentation. Read the section called "Life-cycle" – what five steps happen whenever a JavaFX Application is launched? The "Life-cycle" mentions three methods: are we overriding any of these methods in the Hangman class? If so, add the `@Override` keyword just before the method header.

Question #A.4

As you start to write longer programs, it's important to have a good understanding of the purpose of the program and to create an outline BEFORE you start writing any code! Start by creating a summary of the class – list the method header for each method in `Hangman`. Be sure to include the visibility modifier, return type, name of the method, and any arguments.

Question #A.5

Now, let's understand how the game works before we try to code it. The game player "guesses" a letter by pressing that letter on the keyboard. To consider this guess, the computer asks several questions and executes several actions. These questions and actions are listed below, but you need to put the steps in the right order! Each question has only two answers: yes or no. Each question is asked only once, but actions may be used several times.

Questions (each used exactly once)

- Is this still an active game?
- Has this letter already been guessed?
- Is the guessed letter correct?
- Did the player just win the game?
- Did the player just lose the game?

Actions (each can be used multiple times)

- Nothing happens, wait for next guess
- Turn the letter green, reveal the letter where it appears in the secret word, and update the status to read, "Letter is correct!"
- Turn the letter red, add a body part to the hangman, and update the status to read, "Letter is incorrect."

Part B: Instance variables and the start method

Question #B.1

Notice that the `Hangman` class has seven instance variables. The first variable is a Dictionary object called `dict`. The solution word (which the player tries to guess) is stored as a String in the variable `secretWord`. The value of `secretWord` is randomly chosen for each game; find the line of code where `secretWord` is instantiated – what method is used to generate the random word?

Question #B.2 + coding

The next three instance variables are arrays which hold objects. What type of object can each array hold?

`letterGuesses` will hold 26 objects, one for each letter of the alphabet. Scroll down to the `setUpLayout` method, and find the comment `//create letterGuesses`. Below this comment, you can see that two lines of code are not finished – complete these lines to instantiate the array, and fill each index with a different letter of the alphabet. String `temp` should contain one letter of the alphabet at a time. Hint: Use your String methods to take advantage of the constant from the `Dictionary` class!

Question #B.3

The next instance variable, `lettersRevealedSoFar`, is an array that holds one object for each letter in `secretWord`. (If the `secretWord` is "`dogs`", `lettersRevealedSoFar.length = 4`.) At the beginning of the game, each Text object contains a "_"; when a letters is guessed correctly, the "_" changes to the letter.

The final array, `hangman`, contains six objects. Look up `Shape` in the JavaFX documentation – name four classes which inherit from `Shape`. A typical "hangman" is comprised of a few parts: head, body, arms, and legs. Which child class(es) of `Shape` could you use to represent each body part?

Question #B.4

In the setUpLayout method, find the section headed with the comment `//set up the hangman outline`. This is where the hangman figure is created – were your `Shape` guesses correct? In the for-loop just below, notice that we call the same methods on each object of `hangman`, even though they are objects of different classes! Which concept of object-oriented programming does this embody?

Question #B.5 + coding

The final two instance variables are an integer called `currentHangmanPart`, and a boolean called `activeGame`. `currentHangmanPart` starts at 0 and increments with each incorrect guess. If it reaches 6, you lose the game. `activeGame` is set to true when a game begins, and is set to false once the game is won or lost.

Go to the `start` method. As part of a Java application, `start` is executed once, and is used to set up the appearance and functionality of the application. Find the line `resetButton.setOnAction( e -> {`. Object `e` is an EventHandler, and the syntax `e -> { /* code here */ }` is called a Lambda

expression – all of the code between `{` and `}` will be executed whenever `resetButton` is activated (i.e. by clicking with a mouse, or pressing the "Enter" key). These three lines of code will reset the game board and start a new game.

Go to the next Lambda expression, which is called inside `scene.setOnKeyPressed`. The first line of this method detects what key was pressed, and creates a String object called `guess` which represents that letter/key. Use a String method to make sure `guess` is lowercase. Finally, fill in the missing parameter for `checkGuess` – how can you tell what kind of object should be used there?

Part C: The findLetter and foundInSecretWord methods

Question #C.1

In order to write the `findLetter` method, we will need to do some investigating to figure out what it should do! Where is `findLetter` called – specifically, in which method of Hangman? Read through this method, and describe what it does in 1-2 sentences.

What are the parameter(s) of `findLetter`? What does it return? Where is this return value stored in the calling method? (i.e. What is the name of the instance variable?)

Question #C.2 + coding

The writer of this code was very kind, and left some comments in the `findLetter` method, as a hint! Use all of this information that you've gathered to plan and execute your code for the `findLetter` method.

Remember: `letterGuesses` is not an array of Strings! What type of objects does `letterGuesses` contain? Use the JavaFX documentation to find a method which will return a String that you can use. What method did you pick?

Question #C.3 + coding

We can repeat the same process to write the `foundInSecretWord` method. Where in Hangman is this method called? What comments did the previous developer leave for you?

Using the information you gathered, write the `foundInSecretWord` method. You'll need to use the same method you use for `letterGuesses` to access the text of `lettersRevealedSoFar`.

Part D: Checking Guesses

Question #D.1 + coding

Your hangman game is almost complete! The `checkIfLost` method has been completed for you, and you just need to complete part of `checkIfWon`. Create a boolean called `won`, and set it to `true` if all letters have been guessed. Set it to `false` if there are any letters that still need to be guessed (denoted by the "_" string).

Question #D.2 + coding only

Compile the `Hangman` class, and fix any errors as needed.

Question #D.3

Time to play hangman – a.k.a. software testing! You should play the game several times to confirm it works. First, try playing the game "correctly," by entering in single letters to guess, and clicking the

reset button (or hitting "enter") to start a new game. Describe what happens when you win a game, and when you lose.

Try playing the game "incorrectly," e.g. by typing numbers, or trying the same letter multiple times. What happens? What other things could you try to "break" your game?

Once you're satisfied your game works, click the "X" in the upper right corner to exit the game.

Question #D.4

Reference the instructions (list of questions and actions) you created for Question #A.5 – does the order of your instructions match the `Hangman` code? (Hint: Trace the code starting from `scene.setOnKeyPressed` in the `start` method.) If not, which steps did you put in a different order?

Sometimes, it can be okay to swap the order of instructions, like the commutative rule in math (e.g. 2*3 = 3*2). Other times, changing the order of instructions can make the code execute differently, like how 10-3 is not the same as 3-10. Find one example of instructions (questions) which could be swapped, and one example of instructions (questions) for which the order is important.

Part E: Final Summary

Whenever you complete a project, it is important to assess what you think went well and what you need to improve on.

Question #E.1

What was the most challenging part of this research lab for your group?

Question #E.2

What did your group learn/find the most useful by doing this research lab?

Question #E.3

What topics covered in your Programming II class (either in lectures or textbook reading) did you use in this lab?

Question #E.4

What was the most fun aspect of doing this research lab?