OOP Summative - Hang Man Summative

- A game that allows the player to guess the letters of a secret word, similar to hangman
- As the user guesses the correct letters, dashes are replaced with the letters
- Throughout your game you will be responsible for testing your code to make sure it works. I have added a Google Doc to this post so that you can keep track of how successful your program is. Make sure to test all possible scenarios during each phase.
- Once your code is working and you are finished, record a short video (5-10 mins) of your screen as you explain your algorithm. This will be a part of your Knowledge mark.

Phase 1: Getting the basic functionality of the game to work

Specifications for WordGuess

- A single player starts the game
- Secret word is BRAIN, game starts with 5 dashes -----
- When a matching letter is guessed, it replaces the corresponding dash
- Uppercase or lowercase can be entered, only uppercase displayed
- If the player enters an exclamation point (!), they can guess a word and WIN or LOSE
- Or the player can continue to guess letters until word is revealed
- At the end, the player is shown the total number of guesses

Output Sketch for WordGuess

```
WordGuess game.
-----
Enter a letter (! to guess entire word): a
--A--
Enter a letter (! to guess entire word):v
--A--
Enter a letter (! to guess entire word):!
--A--


What is your guess?
brain
You won!
The secret word is
BRAIN You made 3
guesses
```

The WordGuess Algorithm

1. Display a row of dashes to represent the word
2. Prompt the user for a letter guess
3. If the letter guessed is a part of the word, then display that letter in place of the corresponding dash
4. Repeat steps 2 and 3 until all the letters have been guessed or an exclamation point has been entered by the user
5. If an exclamation point has been entered , prompt the user to guess the entire word
6. If the player correctly guesses the entire word or all the letters have been guessed, then display a message indicating that the player has won, otherwise the message should indicate that the player has lost.
7. Display the secret word and the number of guesses

OOP Summative - Hang Man Summative

Phase 2: Making it better

1. Add a title screen using ASCII art. Have an option to start the game or read the instructions. Make sure it is clear how to start the game. Use the Java class file attached to use with your ASCII art. Store your ASCII art in a separate text file. You may use this site to get started: https://textfancy.com/ascii-art/ Use a separate Java class file to handle File I/O operations. There is an attachment that explains how to read ASCII art from a text file. You should clear the screen when the game begins so that you don't see the instructions or the main title screen. Do some research to learn how to do this.
2. When the game is over, give the user the option to play again or stop playing the game.
3. Use a text file as a database to store all of the words you would like to use in the game. Everytime the game is played, choose a new word at random. Include functionality in your game to not allow the same word to be chosen twice.
4. Make it so that the user cannot choose the same letter twice in the same game. Let them know that they made a mistake.
5. A lowercase letter should be treated the same as an uppercase letter. Leading and trailing spaces should not be counted as an error.

Phase 3: Make the game multiplayer

1) Give the user an option at the beginning to play in one player mode or in multiplayer mode.
2) Allow two players to play the game against each other. Keep track of their guesses and decide how a player wins. Is there a number a total tries each player gets? Make it so that there is a definitive way to win and lose. Announce the winner and allow the players to decide if they would like to play again. Keep track of points (1 for winning).

3) Introduce methods into your game to decide things like the winner/loser, and other important key decisions in your game. If there is reoccurring code in your game that executes you should put it in a method and call it when needed.

Phase 4: Make your game object oriented

1) Now that your game works for single and multiplayer mode, you will now design classes so that the same functionality exists but you will use objects instead of putting all of your code into one file. You need a player class, game class (that takes care of game features and a client class to test your code. Think about this when designing your class:
   - Encapsulation
   - Constructors
   - equals and toString() methods that override the Object class' methods
   - Instance vs class variables
   - Instance vs class methods
2) Next, make it so that you can play against the computer who will randomize their choices. You can use the game class to instantiate the computer object and also use the class to have functions such as declare the winner and run basic game tasks.

OOP Summative - Hang Man Summative

3) Your client code should create the objects needed for the game and allow for user input.
4) Think of an additional feature to increase the user's overall level of enjoyment to the game.

Categories	**Level 1** **(50 – 59%)**	**Level 2** **(60 – 69%)**	**Level 3** **(70 – 79%)**	**Level 4** **(80 – 100%)**
Communication Code Design Program Header Comments Variable names Method names Indentation / 10	Variable names and method names along with a description of methods and any required parameters are missing crucial elements and do not communicate a clear plan	Variable names and method names along with a description of methods and any required parameters is communicated with limited clarity	Variable names and method names along with a description of methods and any required parameters is communicated with only minor omissions or errors	Variable names and method names along with a description of methods and any required parameters is communicated clearly
Thinking Program Testing /10	Program testing was missing crucial elements.	Program testing was insufficient to conclude that the program runs properly	Program testing missed covered a considerable number of possible cases and was summarized in a logical way	Program testing was thorough, and summarized in a logical and succinct way during each phase.
Application Program Execution Error Checking / 25	Program doesn't run properly, or the output has serious errors.	Program runs properly. Output has errors. The provided client code will not run properly, but student provided client does run properly	Program runs properly. Output is correct with only minor errors. The provided client code will run properly	Program runs properly as is. The output is identical or superior to the sample provided. The provided client code will run properly with an additional feature.
Knowledge Programming Concepts <u>Focus:</u> Video File I/O Methods OOP Concepts / 25	Few of the programming concepts from the unit are used properly	Some of the programming concepts from the unit are used properly	Most of the programming concepts from the unit are used properly	Many or all of the programming concepts from the unit are used properly to maximize the efficiency of the code