Hangman

In this project, the code will simulate a classic game of Hangman. The basic premise of the game is that the player will be given a set of blank spaces (represented by underscores) that will spell out a word. The player will guess letters; if the word contains the guessed letters, the blank spaces will slowly fill up, and the player will win once all the blank spaces are filled up. However, with every incorrectly guessed letter, a man will start to form on a displayed “gallow”. With every letter, a body part will be drawn and the player will lose when a man is fully constructed. The code will execute each of these behaviors and have a display that shows the gallows, letters, and a man.

1. Structural Design

|  |  |
| --- | --- |
| Data | *Interface* => class |
| Dictionary of words | Array of strings |
| Guessed letters | Set => HashSet<Integer> |
| Slots for correct guesses | Array of characters |

* 1. The base of words from which we select each “game” is in an array of strings, as we have a set quantity of words and it is easy to generate random indices for each game.
  2. We store the guessed letters in a HashSet of integers because we simply want quick “add” and “contains” calls (both of which are O(1) for HashSets).
  3. We store the correct guesses in an array of characters, because there is a predetermined quantity of elements and we simply want quick access to each index (constant time for primitive arrays).

1. Object-Oriented Design

The Hangman class contains a Type object and generates each “round” by selecting a random word for play.

The Type object generates the GUI for the text field, receives individual character guesses and determines whether or not each guess is correct. It contains a Slot object (which is activated when the guess is correct) and a WrongGuesses object (which is activated when the guess is incorrect).

When the guess is incorrect, the WrongGuesses class contributes to the GUI by updating the list of incorrect guesses by printing a formatted string. It also advances the state of the Man object so that we see the progression of the Hangman.

The Man class creates the GUI for the actual Hangman and updates the stick figure whenever a wrong guess is incurred.

When the guess is correct, the Slot class updates the GUI by adding the correctly guessed character to the appropriate “slot” in the Hangman answer field.

1. Testing
   1. To test this code, we will be writing a JUnit testing class. This class will test each class and their respective methods to ensure that they all work correctly.
   2. The JUnit test will test the GUI, each of the class’s functions individually, as well as the entire game running as a whole.