Hangman Game Implementation

GOAL:

You've got your dream job in a gaming company. Specifically, you work in the "Little kids" team. You have been assigned to write their new "Hangman game". Specifically, you will write:

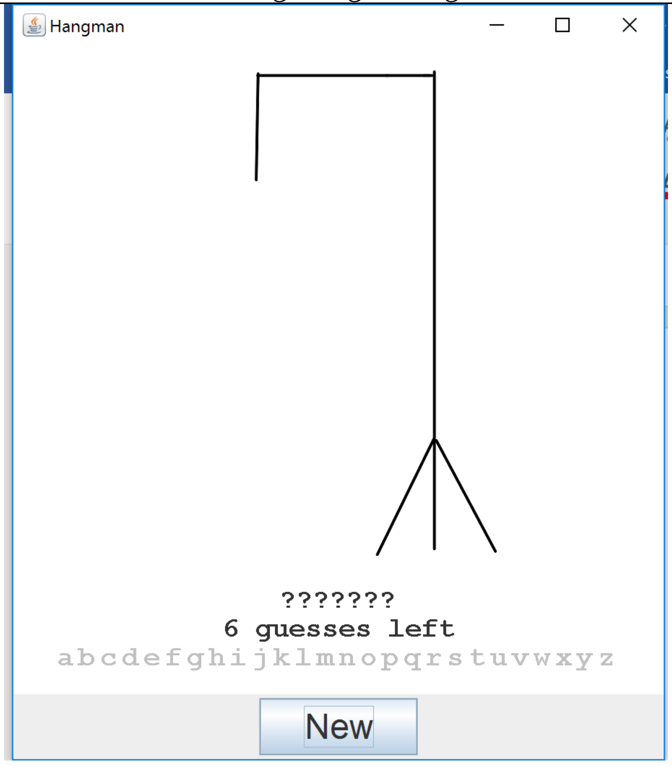
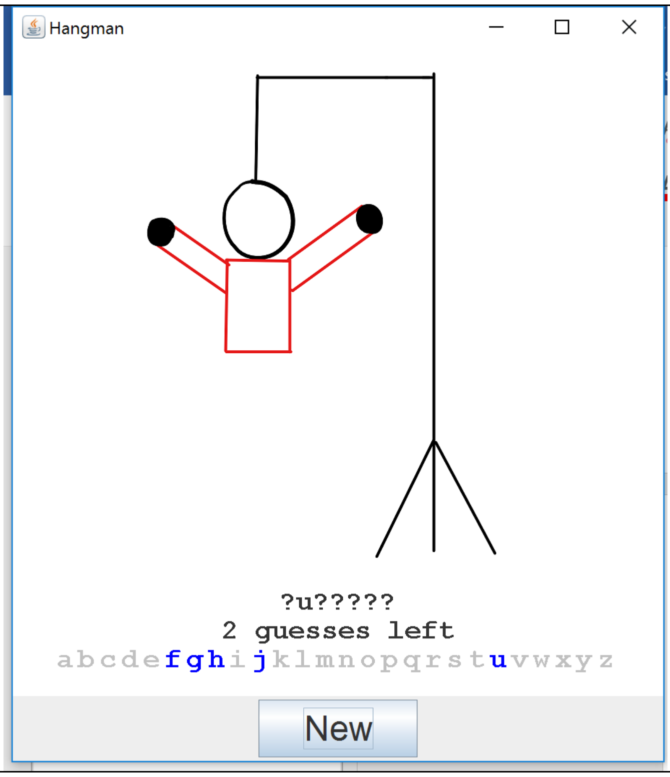
1. The model of the game (the logic that will maintain the game's state)
2. And the graphical user interface for the game (the view the will present to the user the game's state)

By programming this game, you will practice the design patterns discussed in the class. For example: Callbacks, Observer & MVC patterns.

Game Behavior:

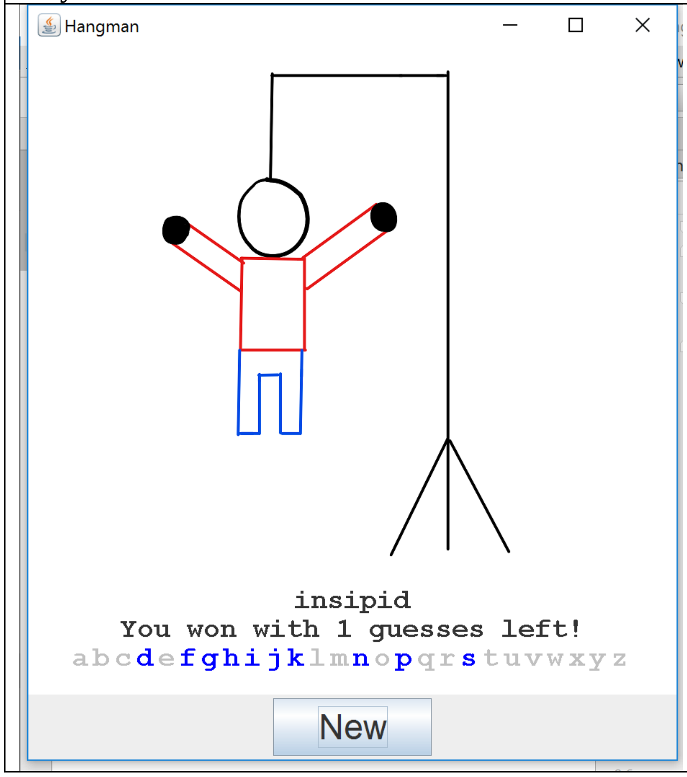
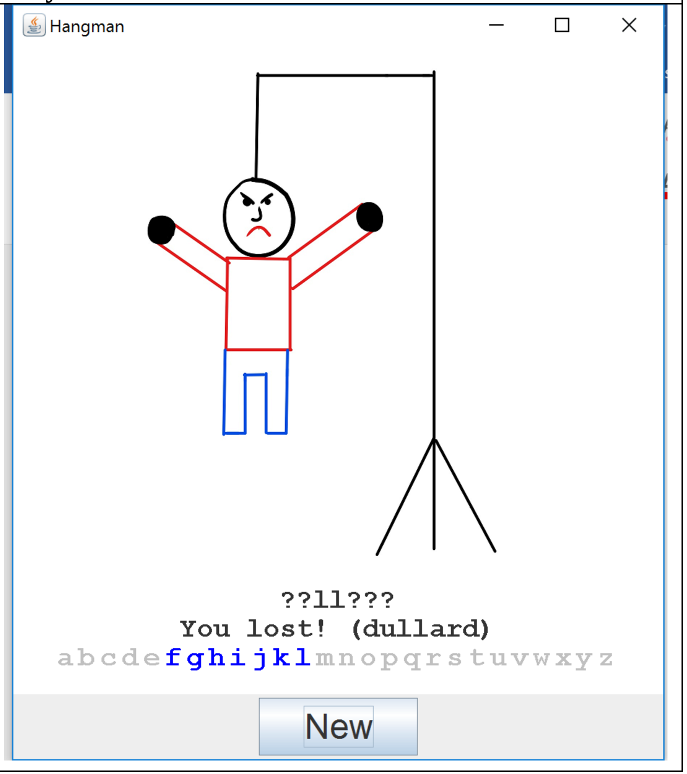
Hangman is a simple game where a player attempts to guess a secret word by picking letters from the alphabet. Next, we outline the steps of the game, along with the GUI specifications:

1. Each time the game begins, a random word is chosen from the provided words.txt file
2. The chosen/secret word appears as a series of ? (question marks), one for each letter in the secret word.
3. A player should try to guess a letter by simply pressing (i.e. typing) the respective key on the keyboard.
4. The letter that the player has pressed becomes blue (initially all the letters are light gray).
5. <u>One **letter** key = one guess</u>, unless a player had hit the same key before.
6. If the player guesses a letter that occurs in the secret word, **all** occurrences of that letter will appear, changing from question marks into the correct letter.
7. If the player guesses a letter that does not occur in the secret word, the player uses up one guess (the number of remained guesses is updated), and a portion of the hangman's body appears.

At the beginning of the game	In the middle
 Hangman ??????? 6 guesses left a b c d e f g h i j k l m n o p q r s t u v w x y z New	 Hangman ?u????? 2 guesses left a b c d e f g h i j k l m n o p q r s t u v w x y z New

8. If the player correctly guesses all the letters in the secret word before running out of guesses, he/she wins the game.
9. If all guesses are used up, the hangman's entire body appears and the player loses the game.
10. The player may start playing again by pressing the New button.

Possible screenshots of the game widow are presented above and below. These are NOT all the possibilities – only some examples.

Player Wins	**Player Looses**
 Hangman insipid You won with 1 guesses left! abcdefghijklmnopqrstuvwxyz New	 Hangman ??ll??? You lost! (dullard) abcdefghijklmnopqrstuvwxyz New

On your game window the following information should be visible during the ENTIRE game:

- The state of the hangman via one of the provided images (NO NEED to draw the hangman, just show the relevant image)
- Word being guessed (? stands for letters not yet guessed or the letter appears if it was correctly guessed)
- Number of guesses left (maximum 6)
- All possible letters (letters that were used are colored by blue).

Model:

You are responsible to implement the logic that maintains the game state.
The model should maintain the following:

- Secret word
- Guessed word
- Number of guesses left
- Possible words (from supplied words)
- Game state

The model should respond to user guesses.
The model should also follow the **Observer pattern**, (you might want to extend `Observable` to allow you to attach observers to it). Observers are notified upon any change to the model. For full credit on the assignment, you should create a class that implements `Observer` and uses it to repaint your drawing panel; do all repainting in response to model notifications, rather than repainting directly in the panel. See additional GUI restrictions on the next page.

GUI and Drawing Panel:

Here are restrictions of your GUI properties that you should follow in your design and coding:

- The frame should contain a drawing panel in its center that occupies most the window area.
- The bottom of the window should contain a panel with a **"New"** button. When the New button is clicked, the game should be restarted and a new game should begin (**without** need to resize the window or run again the program)
- The frame should grab keyboard focus when it is started, so that key presses will correctly go to the frame instead of other GUI components. Do this by calling the frame's `requestFocus()` method when it is started.
- When the user types a letter key (lowercase `'a'` through `'z'`), the game should guess that letter. The guess may change the state of the model, and if so, the drawing panel should be repainted. If the user presses a key other than a lowercase letter, or if the key has been pressed before, no action should be taken.

The game's drawing panel should have the following properties:

- Its background color should be white.
- An **image** should appear on the panel always, to indicate the current hang man's status. You are provided with the images (if you do not like those, you may create your own). **You should NOT draw hangman in GUI. You should use externally supplied images to present current hangman state.** The image to use should reflect how many guesses the player has used up. That is, if the player has made 0 guesses, the file `hangman0.png` is the image to use; if the player has used 4 guesses, `hangman4.png` is the image to use; and so on.
- Below are specific instructions regarding the images:
  - The image should be located 10 pixels from the top of the panel, centered horizontally in the panel regardless of the panel's size. Do NOT hard-code the image dimensions or position into your code. Document any assumptions you make regarding the images.
  - The image should always be up-to-date with the game's current state (in other words, the image should NOT lag behind the game). Remember your GUI should be always ready for `repaint()` calls.
  - It should NOT be necessary to resize or obscure the window to refresh the current image.
- Three **strings** of text should always also appear on the panel. Below are restrictions and specifications of these strings:
  - The text should be centered and appear at the bottom (below Hangman image)
  - All text on the panel should be drawn in **Monospaced font, bold** (size is up to you)
  - Unless otherwise specified, text appears in black.
  - One string is the current word being guessed at, showing the letters that the player has tried and ? (question marks) for all letters that have not been correctly guessed.
  - The second string of text is a display of the current game status. If the game is in progress, the string will show the number of guesses the player has remaining. For example, if the player has 2 guesses left, the string will be **"2 guesses left"**. If the game has been won, the string should display this information and the number of guesses left. For example, if the player wins the game with 3 guesses remaining, the string will be **"You won with 3 guesses left!"**. If the game has been lost, the string should display this information as well as what the word really was. For example, if the player runs out of guesses while trying to guess the word `banana`, the string will be **"You lost! (banana)"**.
  - The third string of text (more accurately, many individually drawn characters of text) is a list of all the letters of the alphabet, in lower case, each colored to indicate whether the player has guessed it or not. If a letter has been guessed, it should be colored in blue; if not, it should be colored in light gray.
    Hint: since each letter is potentially a different color, each letter must be drawn individually, rather than drawing all the letters as a combined string.

During the entire game, ONLY ONE WINDOW (described and illustrated above) should be shown.

Resources:

The following is a partial list of Java packages and classes that may help you in writing this project:

- `package java.awt: Color, Container, FlowLayout, Font, FontMetrics, Graphics, Graphics2D, Image, MediaTracker, Toolkit`
- `package java.awt.event: ActionEvent, ActionListener, KeyAdapter, KeyEvent, KeyListener`
- `package java.util: Observable, Observer`
- `package javax.swing: JFrame, JButton, JPanel`

Use the `stringWidth(String)` method from a `FontMetrics` object to discover how wide a string will be when drawn on screen. To retrieve the font metrics for a panel, call the `getFontMetric()` method on its Graphics context.

Style and Design:

You should demonstrate good **object-oriented design** in your program. One class one responsibility - separate event listeners and observers from GUI components by putting them in separate classes (they can be inner classes). Your fields should encapsulate a game/GUI state (do not set unnecessary variables as fields).
You should follow **style, JavaDocs** and other guidelines as described in your previous projects.
The logic/model module of your game should be tested as usual using JUnit testing.
No need to test the GUI part, it should just run properly.