# Project 2: Hangman Game

The goal of this project is to become more familiar with creating and using classes and objects in Java, to further introduce you to the concepts of abstraction and modularization, and to introduce you to collections of items. In this project you will build on your Person & Hangman classes from Project 1. You will complete your Hangman so it contains everything we need to display and control your game.

# Instructions

1. Create an AlphabetPanel class that extends JPanel which uses the Text class posted on the course web site. **DO NOT MODIFY THE TEXT CLASS IN ANY WAY.**
2. Add an AlphabetPanel class and implement the following:
   1. Add a main method that creates an AlphabetPanel object and puts it in a JFrame.
   2. Shows all of the letters of the alphabet (using a Text object for each letter)
   3. reset – a method that resets the alphabet panel letters all to the default color
   4. setLetterColor – a method that sets the color of a character letter to a specified color
   5. getLetterColor – a method that gets the color of a specified letter as a character parameter. If the letter is not valid (i.e., not in the panel), return null.
   6. ERROR HANDLING: make sure the previous two methods work the same for upper and lower case letters; and that it correctly handles (ignores) characters that are not ‘A’ through ‘Z’
   7. hasLetterBeenSeen – a method that returns true if the letter has been pressed (i.e., changed color) since the last reset
   8. isVowel – returns whether or not the character is a vowel
3. Create a GuessPhrasePanel class and implement the following (NOTE: Make sure your methods work the same for upper and lower case letters):
   1. Has a constructor taking a String guess phrase
   2. Shows all the letters in the phrase, initially just showing the underline for all alphabet letters (not the letter as it needs to be guessed). Spaces and other punctuation should be shown with no underline.
   3. setPhrase – sets a new phrase to this GuessPhrase object, gets rid of old phrase parts and creates new ones
   4. hasLetter – a method that returns whether or not a specified letter is in the phrase
   5. revealLetter – a method that reveals or shows a specified letter (if there are more than one of that letter, all should be revealed)
   6. isFullPhraseRevealed – returns whether or not the full phrase has been revealed
   7. revealFullPhrase – reveals the full phrase
4. Create a HangmanGame class to test GuessPhrasePanel
5. Create fields for RandomString and GuessPhrasePanel objects
6. Create a constructor that initializes the fields and adds a GuessPhrase panel to a JFrame. The constructor should also add a KeyListener to the frame that does the following:
   * 1. reveals the letter typed if it is in the guess phrase
     2. reveals the whole phrase if the enter key is pressed
     3. gets a random string and sets it as the text for the GuessPhrase if the space bar is pressed
     4. does nothing if any other key is pressed
7. Create a main that calls the HangmanGame constructor
8. You now have all of the pieces needed to complete the game, now your job is to make them work together.
   1. Coordinate the user input to appropriately change the alphabet panel, the person, and the guess phrase based on each letter pressed. Your key listener code in HangmanGame should turn incorrect letters a different color (like red) after they have been guessed. Similarly, correct letter guesses should also be turned a different color (like green) after being entered by the user.
      1. *Hint*: you may need to make your HangmanGame a JFrame, and put the code that normally goes in main for a JPanel into Hangman’s constructor. Then the HangmanGame main simply becomes: **new** HangmanGame();
   2. Ask the user if they want to play again. If they do, reset the guess phrase to a new phrase and reset the rest of the game. Otherwise, exit the game. To ask the user if they want to play again, you can either:
      1. Use a message dialog box ***OR***
      2. Add another Text panel and show/hide it
   3. NOTES:
      1. Clicking in the window should not advance the person (so delete the mouse event handling you previously had in person)
      2. Only incorrect letters should make a part of the person show up and **ONLY** the first time the incorrect letter is pressed (pressing the same incorrect letter over and over should not cause any more body parts to show).
9. Be sure to comment everything
10. Each of your methods (including their parameters).
11. Anything else that needs explanation.
12. Use proper indentation
13. Use appropriate variable/field names
14. Group like things together (items that are part of the same objects)
15. Use appropriate comments – including comments for each method
16. Check your curly braces
17. For this project you will submit the following (see “Saving and Submitting your Work” for details):
18. A general archive file (e.g. a zip)
19. A runnable JAR file
20. At least one JPG showing your application
21. EXTRA CREDIT: Add comments for correct/incorrect answers. NOTE: You can use two files for this and reuse your RandomString class from Project 5. Display a corresponding comment from the person being hung, negative if the user/player guessed wrong, positive or encouraging if the user/player got it right.

# Hints / tips

* Internally, characters are stored as numbers, specifically, as integers. This means we can work with AlphabetPanel letters as if they were numbers. For example, ‘A’ + 1 = ‘B’. Similarly, to determine that ‘Y’ is the 25th letter of the alphabet, we could calculate ‘Y’-‘A’+1 = 25.
* You can add Text objects directly to your AlphabetPanel, without explicitly creating an ArrayList by calling this.add. You can use JPanel methods like this.getComponent or this.getComponents to retrieve the text objects later. Google the JPanel javadoc for more information about these methods.

For example, to get all the text objects you’ve added to a panel:

**for** (Component c : **this**.getComponents()) {

Text t = (Text) c;

// what you want to do to every Text panel

}

* In order to support key events you will need to add the following code (which you need to complete) in the **constructor** of your class:

**this**.setFocusable(**true**); // enables panel to listen to key events

**this**.addKeyListener(**new** KeyAdapter(){

**public** **void** keyTyped(KeyEvent e)

{

// logic to handle key events

}});

* Google the javadoc for KeyEvent for more information about how to access keys the user types.
* When initializing your GuessPhrasePanel and creating the setPhrase method, you may find it useful to create a private initializeLetters() method to be used by both the constructor and setPhrase. It would look like:  
  **private** **void** initializeLetters() {

**this**.removeAll(); // throw away existing letters in phrase

**this**.updateUI(); // reset entire UI

// add code statements to

// initialize letters in phrase to be guessed   
 // & add them to the GuessPhrasePanel

**this**.revalidate();// push changes through  
}

* You can match special keys on the keyboard like enter or space using virtual key constants in the KeyEvent class, like KeyEvent.VK\_ESCAPE.
* For the comments positive and negative you can reuse your RandomString class
* When incorporating multiple panels in a single frame, you will need to set a layout scheme, such as BoxLayout.
* As part of project 1 you need to make the logic of the game work properly as well as ask the user if they would like to play again. You may find it useful to use the following logic:  
   **if** person or phrase is revealed  
   process the user's Y/N answer  
   **else**  
   process the input as usual for the game, updating the person, guess phrase panel,   
   & alphabet panel as appropriate  
   **if** person or phrase is revealed  
   ask the user if they would like to play again
* To create a runnable JAR file:
  + Follow the saving and submitting instructions to create a runnable JAR and make sure you select the HangmanGame class as your launch configuration.
  + Upload the runnable JAR file as an attachment to your project submission, so your project submission should have at least 3 files: an archive file, a runnable JAR file, and an image of your game (perhaps part way through a game).
  + The JAR file is basically an executable that you will be able to double click to launch your program. Note: you will still need to have your "guess\_phrases.txt" file in the same directory as your JAR file. **You must name the file with your guess phrases: “guess\_phrases.txt”**

# Saving and Submitting Your Work

* An archive file (often called a zip file) can be created directly from Eclipse by going to File  Export and then selecting General  Archive.
* Upload as a secondary file a JPG or PNG picture/image of your HangmanGame class when it is drawn. To do so run your program, then:
  + For Windows: Select the window that has the drawing in it then press the ALT and Print Screen buttons at the same time. This puts a copy of the active window in memory. You can open Microsoft Paint or any image editor and paste the image in and save it as a JPG or PNG file in the directory of your project. Upload that saved file.
  + For Mac: Press Command – Shift – 4, followed by the spacebar, and then select the desired application window. This puts an image file on your desktop, which you can then upload.
* For Java application projects you will also create a runnable JAR. To do this you will again select File  Export, and then select Java  Runnable JAR file.
* When submitting online be sure all files are uploaded. These files include: an archive file [e.g. zip], an image file [e.g. PNG], and for Java applications a runnable JAR. Once these files are uploaded to the Blackboard project submission page, press the “Submit” button. You may submit as many times as necessary, but your **last** submission is the only one that will be evaluated – so it must have **all** the necessary file attachments.