Programming Assignment : Hangman Game

In this lab you will implement a "simple" game of Hangman. If you don't know how to play hangman, you should look it up.

For this lab, the entire main method that interacts with user to play the game has been written for you. The game will work if you write a correct implementation of the HangmanGame interface which has also been provided.

The purpose of this lab is to get you very familiar with implementing an interface and with manipulating and using String methods.

What it should look like:

When you run the main method, you should be able to play a game of hangman in the terminal. Here is a transcript of what the game might look like:

```
Welcome to Hangman!
Try to guess the secret word one letter at a time.

Secret Word: _ _ _ _ _ _ _ _ _ _
Letters Guessed:
Guesses remaining: 5

Next guess: R

Yes!

Secret Word: _ _ _ _ R _ _ _ R _
Letters Guessed: R
Guesses remaining: 5

Next guess: S

Nope!

Secret Word: _ _ _ _ R _ _ _ R _
Letters Guessed: RS
Guesses remaining: 4

Next guess: T

Yes!

Secret Word: _ _ _ _ R _ T _ R _
Letters Guessed: RST
Guesses remaining: 4

Next guess: N

Nope!

Secret Word: _ _ _ _ R _ T _ R _
Letters Guessed: RSTN
Guesses remaining: 3

Next guess: E
```

```
Nope!

Secret Word: _ _ _ _ _ R _ T _ R _
Letters Guessed: RSTNE
Guesses remaining: 2

Next guess: A

Yes!

Secret Word: _ A _ _ R A T _ R _
Letters Guessed: RSTNEA
Guesses remaining: 2

Next guess: M

Nope!

Secret Word: _ A _ _ R A T _ R _
Letters Guessed: RSTNEAM
Guesses remaining: 1

Next guess: D

Nope!
Sorry, you lose.  The word was: LABORATORY
```

Implementing the Interface:

Your task is to implement the interface according to the specification. The class you write needs to maintain the state of the game, updating the state each time a letter is guessed. That means your class needs to keep track of the number of guesses remaining and the letters guessed so far, and what the state of the secret word is that should be revealed to the player.

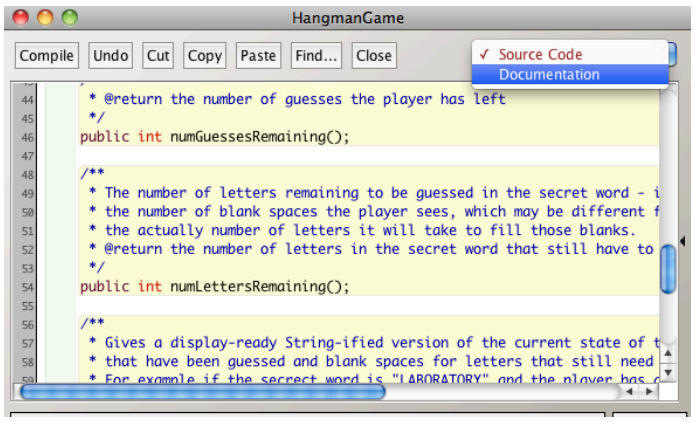
The game keeps altering its state until the game is over and a win/loss condition is met.

A single instance of a HangmanGame should only play one game of Hangman.

Documentation:

Look at the methods defined in the interface and read the comments above each method to learn what the method should do. The comments have been written in "JavaDoc" format. Comments written in this format can be easily converted into HTML documentation – the type you see online. You can read it in that format in blueJ by switching the code window from "Source Code" view into "Documentation" view using the pull down menu in the top right corner.

The only thing not documented in the interface of course is the constructor. A class called MyHangmanGame has



```
     * @return the number of guesses the player has left
     */
    public int numGuessesRemaining();

    /**
     * The number of letters remaining to be guessed in the secret word - i
     * the number of blank spaces the player sees, which may be different f
     * the actually number of letters it will take to fill those blanks.
     * @return the number of letters in the secret word that still have to
     */
    public int numLettersRemaining();

    /**
     * Gives a display-ready String-ified version of the current state of t
     * that have been guessed and blank spaces for letters that still need
     * For example if the secrect word is "LABORATORY" and the player has c
```

been started f or you to show what the constructor should do.

Step 1: getting started

When implementing an interface you can't possibly implement all of the methods at once. You need to do it incrementally, but the class won't compile until you have all the methods present. One technique is to write "dummy" method stubs for all the methods that just return nonsense values just in order to satisfy the compiler.

Once you have method stubs written and the class compiles, you can set about incrementally working on the methods.

Step 2: starting to code

Decide the private variables (at least some of them) that you'll need to maintain the state of the game add them to the class and initialize them in the constructor.

Note: You'll probably want at least three strings: the secret word itself, the state of the secret word with guesses incorporated (i.e. "L _ _ O_ _ TO_Y") and a string that holds the letters guessed so far.

Test the constructor of the object on the BlueJ "object bench", by right-clicking the class and choosing new MyHangmanGame(…). Inspect the object to make sure everything is okay.

Step 3: write the easy methods

Some of the methods are simple "getter" methods that just report values that presumably you're storing as private variables – getSecretWord, numGuessesRemaining, numLettersRemaing, lettersGuessed(). Write them first to get them out of the way.

Step 4: logic methods

Next you should tackle some methods that require logic like isWin() and gameOver(). (i.e. the game is over when/if there are no more guesses remaining or all of the letters in the secret word have been guessed). You can write these, but you won't really know if the logic is totally correct until you've begun to develop the game. You should still write them now the way you think they should be written, but just be prepared to come back to them if necessary.

Test: Time to test on the object bench. Even though your class is very simple, you should still construct an instance on the object bench, and test the methods to see if they're returning the correct values.

Step 5: display methods

There are two methods that are essentially used only for display purposes, one is displayGameState() which should return a String showing blanks for letters that the player hasn't guessed yet and filled in letters for the ones they have. This is a String you should probably maintain as a private variable.

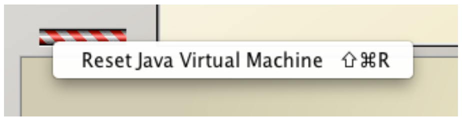
Updating this String correctly is one of the major tasks of the project, but in this method just return a string that presumably you're keeping as state data in the class. (Later you could get fancy and do an ASCII art hangman's noose ☺)

The other is the lettersGuessed() method. This will initially be an empty string, but characters will be appended as you go.

Step 6: the makeGuess method

Don't try to write this method all at once. This method is what triggers all of the game logic to move the game from one state into the next. This method will figure out what's going on in the game and update all the state variables appropriately. Since it's complicated you should start simply.

A PLACE TO START: start by simply appending the char that was guessed to the String keeping track of the letters guessed. And return true (for now). **Then run the provided main method.** As you guess letters you should at least see the list of "Letters Guessed" growing. NOTE: **To KILL a program while it's running** right-click on the red and white "barbershop pole" in the bottom-left of the BlueJ project window and reset the JVM.



As a reminder you can compose strings use the '+' operator. This example below shows more than you need but it does show appending a char to String.

```
String front = "Baker";
String mid = "cool";
char ch = 'a'
String message = front+" is "+ch+" "+mid+" guy.";
//String message is now: "Baker is a cool guy."
```

Now that that's working, try to prevent duplicate characters from being appended to the string (HINT: use indexOf). Then run the main method to make sure that when you guess a letter for the second time it doesn't appear in the string of guessed letters.

Step 7 and beyond…

From this point you've probably got a decent grasp of what lies ahead. Your next step might be to look for the guessed letter in the secret word, and update the display string appropriately. This will involve updating other state variables as well.

You may also want to write yourself some helper methods to put some complicated tasks out of sight and out of mind once you've tackled them.

You may find the following String methods useful to you (which you should look up the documentation for in the Java API):

```
char charAt(int index)
int indexOf(char ch)
ing indexOf(char ch, int fromIndex)
```

```
char[] toCharArray()
String toUpperCase() / toLowerCase()
String substring(int from, int to)
```

What will be challenging is the mixing of Strings and chars, and composing new strings the proper way. You'll need to splice in characters that have been guessed correctly into some other string in the correct way.

Good luck and ask questions on Piazza.