**Assignment 1 Due Date: 18-09-2018**

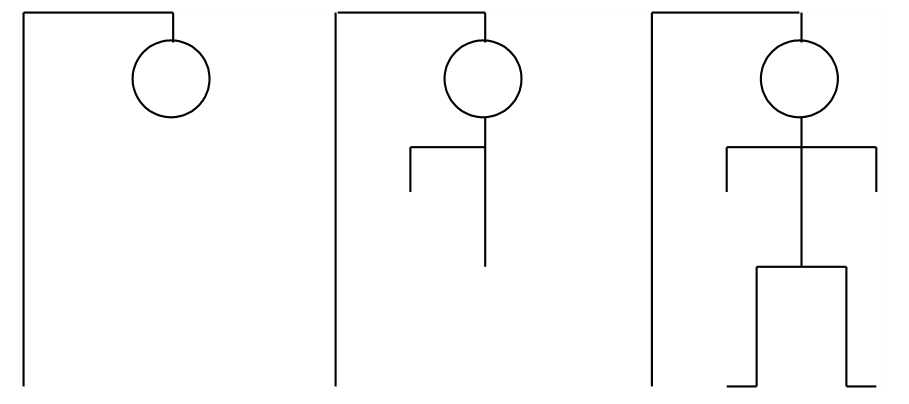
**This assignment should be done in a group of two students.**

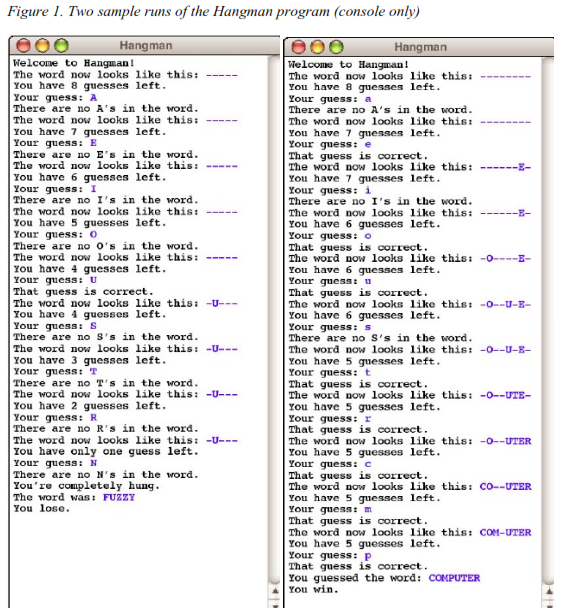
**Task 1: Hangman**

For this assignment, your mission is to write a program that plays the game of Hangman.

When the user plays *Hangman,* the computer first selects a secret word at random from a list built into the program. The program then prints out a row of dashes—one for each letter in the secret word—and asks the user to guess a letter. If the user guesses a letter that is in the word, the word is redisplayed with all instances of that letter shown in the correct positions, along with any letters correctly guessed on previous turns. If the letter does not appear in the word, the user is charged with an incorrect guess. The user keeps guessing letters until either (1) the user has correctly guessed all the letters in the word or (2) the user has made eight incorrect guesses. Two sample runs that illustrate the play of the game are shown in Figure 1 on the next page.

When it is played by children, the real fascination (a somewhat morbid fascination, I suppose) from *Hangman* comes from the fact that incorrect guesses are recorded by drawing an evolving picture of the user being hanged at a scaffold. For each incorrect guess, a new part of a stick-figure body—first the head, then the body, then each arm, each leg, and finally each foot—is added to the scaffold until the hanging is complete. For example, the three diagrams below show the drawing after the first incorrect guess (just the head), the third (the head, body, and left arm), and the diagram at the tragic end of a losing game: **(Don’t worry you don’t need to implement the graphical part at this stage)**.





**Implementation Notes: Playing a console-based game**

In this assignment, your job is to write a program that handles the user interaction component of the game—everything except the graphical display. To solve the problem, your program must be able to:

• Choose a random word to use as the secret word (Hint use Random class to pick random index). That word is chosen from a word list, as described in the following paragraph.

• Keep track of the user’s partially guessed word, which begins as a series of dashes and then gets updated as correct letters are guessed.

• Implement the basic control structure and manage the details (ask the user to guess a letter, keep track of the number of guesses remaining, print out the various messages, detect the end of the game, and so forth).

The only operation that is beyond your current knowledge is that of representing the list of words from which you can choose a word at random. For this assignment, you will simply

make use of a class that we’ve given you called HangmanLexicon that provides a small list of words that will allow you to test your program. The implementation of the class you’ve been given is only a temporary expedient to make it possible to code the rest of the assignment. The strategy of creating a temporary implementation that provides enough functionality to implement the rest of the program is a common technique in programming. Such temporary implementations are usually called **stubs.** In this assignment, the starter project comes with a stub implementation of the HangmanLexicon class, which appears in Figure 2. The class contains two public methods: getWordCount(), which returns the number of words in the lexicon, and getWord(i), which returns the word at index i. Like all indices in Java, the value i runs from 0 to one less than the number of words.

/\*

\* File: HangmanLexicon.java

\* ------------------------

\* This file contains a stub implementation of the HangmanLexicon class

\*/

**public** **class** HangmanLexicon {

/\*\* Returns the number of words in the lexicon. \*/

**public** **int** getWordCount() {

**return** 10;

}

/\*\* Returns the word at the specified index. \*/

**public** String getWord(**int** index) {

**switch** (index) {

**case** 0:

**return** "BUOY";

**case** 1:

**return** "COMPUTER";

**case** 2:

**return** "CONNOISSEUR";

**case** 3:

**return** "DEHYDRATE";

**case** 4:

**return** "FUZZY";

**case** 5:

**return** "HUBBUB";

**case** 6:

**return** "KEYHOLE";

**case** 7:

**return** "QUAGMIRE";

**case** 8:

**return** "SLITHER";

**case** 9:

**return** "ZIRCON";

**default**:

**return** **new** String("Illegal index");

}

}

}

The sample runs in Figure 1 should be sufficient to illustrate the basic operation of the game, but the following points may help to clarify a few issues:

• Create a new class Hangman and implement the method. **(For this problem, use String, StringBuffer instead of Arrays).**

• At the beginning of your main method in Hangman class, you need to create a new HangmanLexicon and store it in an instance variable.

• You should accept the user’s guesses in either lower or upper case, even though all letters in the secret words are written in upper case.

• If the user guesses something other than a single letter, your program should tell the user that the guess is illegal and accept a new guess.

• If the user guesses a correct letter more than once, your program should simply do nothing. Guessing an incorrect letter a second time should be counted as another wrong guess. (In each case, these interpretations are the easiest way to handle the situation, and your program will probably do the right thing even if you don’t think about these cases in detail.)

**Assignment Evaluation: (Total marks: 10)**

Each group member will be evaluated individually. I will do some changes in the code and ask each student to fix the issue. Don’t copy anything from internet otherwise you will get straight zero.

**Note: Don’t use Arrays. Instead Use String and StringBuffer class.**

**Task 2: Find Sentences**

Write a program to find all the sentences, or consecutive sequence of sentences, in the provided input string, where:  min <= length <= max.  
Assume that a sentence ends in a period, question mark, or exclamation point.  
  
Count all blanks and punctuation, but assume only one blank between sentences.  
(All EOL characters should be converted to blanks).  
Precondition: Min and Max will be positive integers less than 1000, and Min <= Max.

Your program should handle both cases:

1. It should check if Min, Max and the string (text content) are provided as command line argument, then there is no need to take it from the Standard Input.
2. If these 3 are not provided as command line, then take the input from the user.

Precondition: Min and Max will be positive integers less than 1000, and Min <= Max.

For example, given this text:  
Black is white.  Day is night.  Understanding is ignorance.  
Truth is fiction.  Safety is danger.  
  
If min = 15, and max = 16 then the output is   
    "Black is white."  
If min = 17, and max = 18 then the output is   
    "Truth is fiction."  
  "Safety is danger."  
If min = 30 and max = 37 then the output is   
    "Truth is fiction. Safety is danger."  
because the two sentences are consecutive sentences with the desired length.