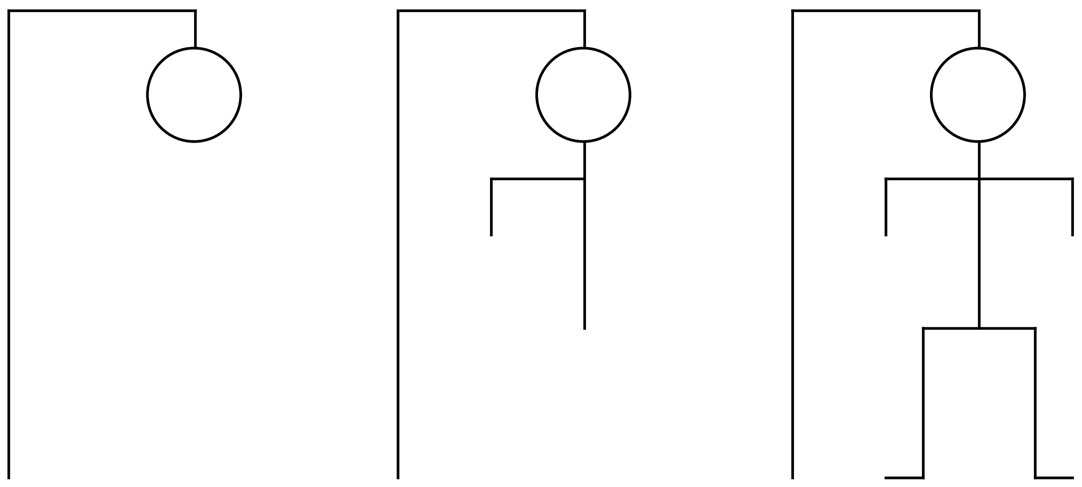
Assignment 4 — Hangman

Assignment #4 is to write a program that plays the game of Hangman. This assignment serves two purposes. First, it is designed to give you some practice writing programs that manipulate strings and files. Second, you will have a chance to work with multiple classes in a single application.

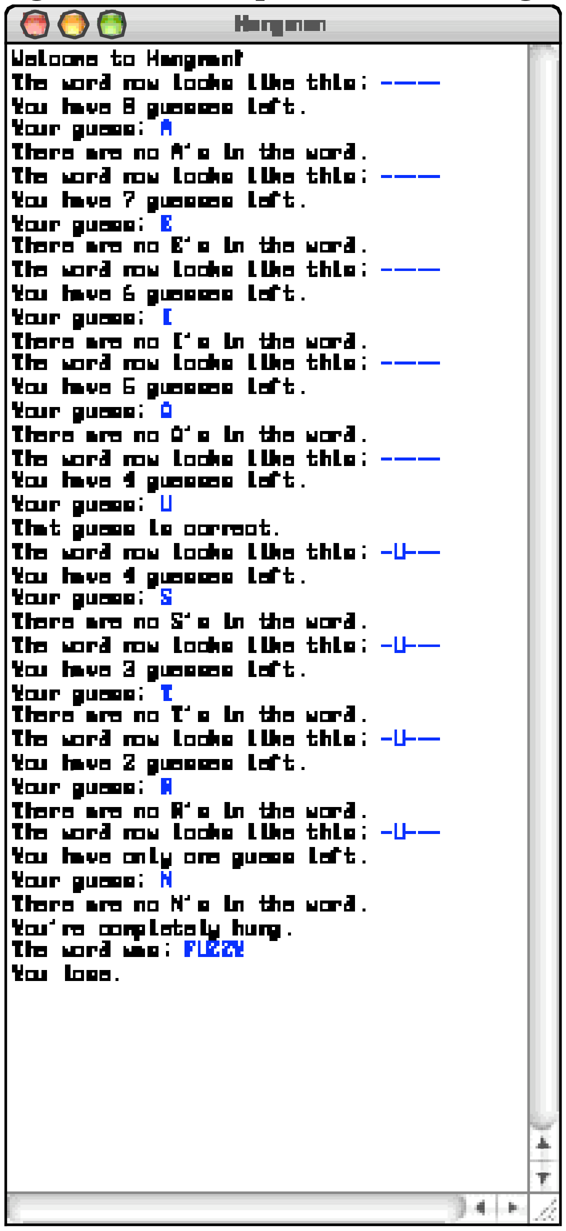
When it plays Hangman, the computer first selects a secret word at random from a list built into the program. The program then prints out a row of dashes—one for each letter in the secret word—and asks the user to guess a letter. If the user guesses a letter that is in the word, the word is redisplayed 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 one) of 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:

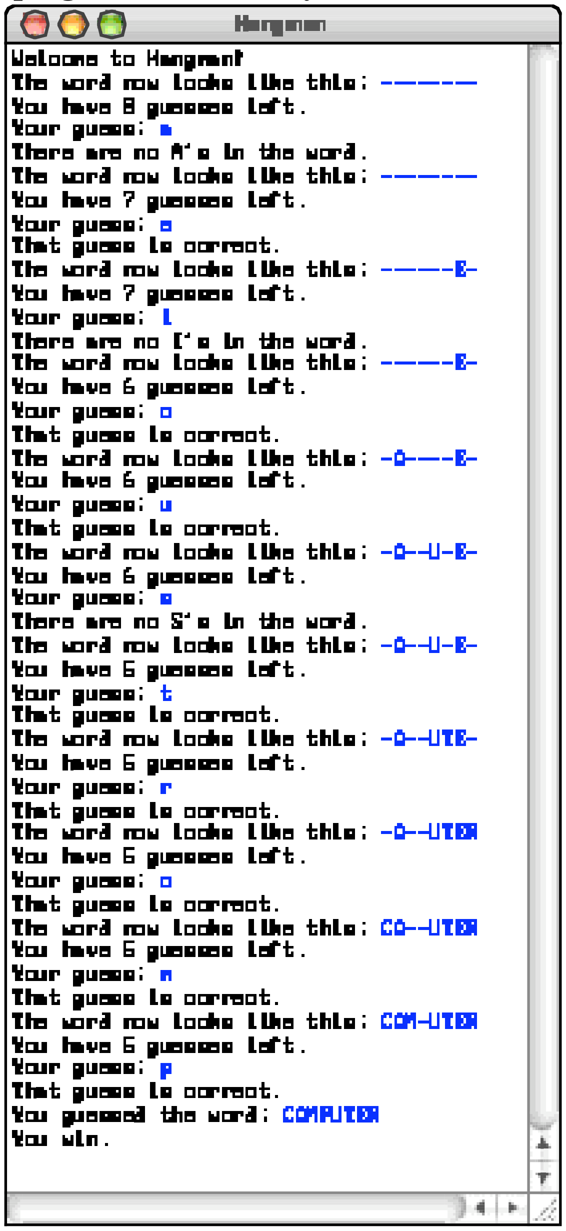


In order to write the program that plays Hangman, you should design and test your program in three parts. The first part consists of getting the interactive part of the game working without any graphics at all and with a fixed set of secret words. The second part consists of building a separate class that maintains the scaffold diagram. The final part requires you to replace the supplied version of the secret word list with one that reads words from a file. The rest of this handout describes these three parts in more detail.

Figure 1. Two sample runs of the Hangman program (console only)



```
Welcome to Hangman!
The word now looks like this: -----
You have 8 guesses left.
Your guess: A
There are no A's in the word.
The word now looks like this: -----
You have 7 guesses left.
Your guess: E
There are no E's in the word.
The word now looks like this: -----
You have 6 guesses left.
Your guess: I
There are no I's in the word.
The word now looks like this: -----
You have 5 guesses left.
Your guess: O
There are no O's in the word.
The word now looks like this: -----
You have 4 guesses left.
Your guess: U
That guess is correct.
The word now looks like this: -U---
You have 4 guesses left.
Your guess: S
There are no S's in the word.
The word now looks like this: -U---
You have 3 guesses left.
Your guess: T
There are no T's in the word.
The word now looks like this: -U---
You have 2 guesses left.
Your guess: R
There are no R's in the word.
The word now looks like this: -U---
You have only one guess left.
Your guess: N
There are no N's in the word.
You're completely hung.
The word was: FUZZY
You lose.
```



```
Welcome to Hangman!
The word now looks like this: --------
You have 8 guesses left.
Your guess: a
There are no A's in the word.
The word now looks like this: --------
You have 7 guesses left.
Your guess: e
That guess is correct.
The word now looks like this: ------E-
You have 7 guesses left.
Your guess: i
There are no I's in the word.
The word now looks like this: ------E-
You have 6 guesses left.
Your guess: o
That guess is correct.
The word now looks like this: -O----E-
You have 6 guesses left.
Your guess: u
That guess is correct.
The word now looks like this: -O--U-E-
You have 6 guesses left.
Your guess: s
There are no S's in the word.
The word now looks like this: -O--U-E-
You have 5 guesses left.
Your guess: t
That guess is correct.
The word now looks like this: -O--UTE-
You have 5 guesses left.
Your guess: r
That guess is correct.
The word now looks like this: -O--UTER
You have 5 guesses left.
Your guess: c
That guess is correct.
The word now looks like this: CO--UTER
You have 5 guesses left.
Your guess: m
That guess is correct.
The word now looks like this: COM-UTER
You have 5 guesses left.
Your guess: p
That guess is correct.
You guessed the word: COMPUTER
You win.
```

Part I—Playing a console-based game

As the first part of this assignment, 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. 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 is then 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 the first two parts of the assignment, you will simply make use of a class that is provided to you, called **`HangmanLexicon`**, that provides a small list of words that will allow you to test your program. (A *lexicon* is like a dictionary but does not necessarily include definitions, making it a more appropriate name for a class that provides a list of words with no associated meanings.) 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. In Part III, you will replace the definition we've provided with one that reads a list of words from a data file.

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, shown 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.

Figure 2. Stub implementation of HangmanLexicon

```
/*
 * File: HangmanLexicon.java
 * ------------------------
 * This file contains a stub implementation of the HangmanLexicon
 * class that you will reimplement for Part III of the assignment.
 */

import acm.util.*;

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: throw new ErrorException("getWord: Illegal index");
      }
   };
}
```

A game that used this implementation of the `HangmanLexicon` class would quickly become uninteresting because there are only ten words available. Even so, it will allow you to develop the rest of the program and then come back and improve this part later.

Part I is a string manipulation problem. 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:

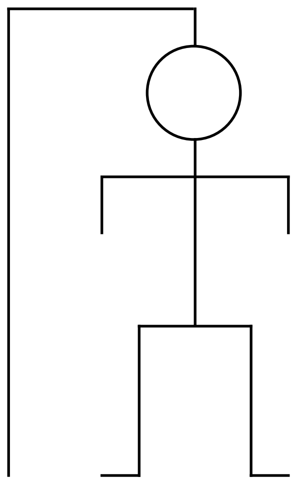
- At the beginning of your `run` method, you need to create a new `HangmanLexicon` and store it in an instance variable. If you extend the program to allow the user to play multiple games, the creation of the `HangmanLexicon` should be performed outside the loop that plays the game repeatedly so that this operation is performed once rather than for every game.
- 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.)

Remember to finish Part I before moving on to Part II. Part II is arguably more fun, but it is essential to develop large programs in manageable stages.

Part II—Adding graphics

For Part II, extend the program you have already written so that it now keeps track of the Hangman graphical display. Although you might want to spice things up in your extensions, the simple version of the final picture for the unfortunate user who has run out of guesses looks like this:



The scaffold and the tiny bit of rope above the head are drawn before the game begins, and then the parts are added in the following order: head, body, left arm, right arm, left leg, right leg, left foot, right foot. Because this picture is simpler than most of the figures you have drawn for section problems, the challenge of this part of the assignment does not lie so much in using the **`acm.graphics`** package but rather in implementing the separation of functions between the class that performs the console-based interaction and the class that manages the display. That class is called **`HangmanCanvas`** and is included in the starter project in the form of the stub implementation shown in Figure 3.

Figure 3. Stub implementation of HangmanCanvas

```
/*
 * File: HangmanCanvas.java
 * ------------------------
 * This file keeps track of the Hangman display.
 */

import acm.graphics.*;

public class HangmanCanvas extends GCanvas {

/* Constants for the simple version of the picture (in pixels) */
   private static final int SCAFFOLD_HEIGHT = 360;
   private static final int BEAM_LENGTH = 144;
   private static final int ROPE_LENGTH = 18;
   private static final int HEAD_RADIUS = 36;
   private static final int BODY_LENGTH = 144;
   private static final int ARM_OFFSET_FROM_HEAD = 28;
   private static final int UPPER_ARM_LENGTH = 72;
   private static final int LOWER_ARM_LENGTH = 44;
   private static final int HIP_WIDTH = 36;
   private static final int LEG_LENGTH = 108;
   private static final int FOOT_LENGTH = 28;

/**
 * Resets the display so that only the scaffold appears.
 */
   public void reset() {
      /* You fill this in */
   }

/**
 * Updates the word on the screen to correspond to the current
 * state of the game.  The argument string shows what letters have
 * been guessed so far; unguessed letters are indicated by hyphens.
 */
   public void displayWord(String word) {
      /* You fill this in */
   }

/**
 * Updates the display to correspond to an incorrect guess by the
 * user.  Calling this method causes the next body part to appear
 * on the scaffold and adds the letter to the list of incorrect
 * guesses that appears at the bottom of the window.
 */
   public void noteIncorrectGuess(char letter) {
      /* You fill this in */
   }

}
```

This stub is somewhat different from the one shown earlier for **HangmanLexicon**. That stub actually did something, even if it was only a part of what the complete implementation of the class will actually do. This stub declares several named constants that define the parameters of the picture, but doesn't actually use them as yet. The three methods in the stub implementation of **HangmanCanvas**—**reset**, **displayWord**, and

noteIncorrectGuess—do absolutely nothing. This strategy, however, is also common in programming. The fact that the class exists and exports methods means that you can call those methods from the console-based **Hangman** class even before you complete their implementation.

The first thing you should do when you begin Part II is to create a new **HangmanCanvas**—in precisely the do-nothing form in which it has been given to you—and install it in the program window next to the console. The **Hangman** class itself is an instance of a **ConsoleProgram**, which means that the startup code in the ACM libraries has installed an **IOConsole** in the window that fills the entire space. Your next task is to add a **HangmanCanvas** to the program window as well. The code you need for this part is extremely simple. First, in the instance variables section of the **Hangman** program, you need to declare an instance variable for the canvas by writing

```
private HangmanCanvas canvas;
```

and then add the following lines to the beginning of your **run** method:

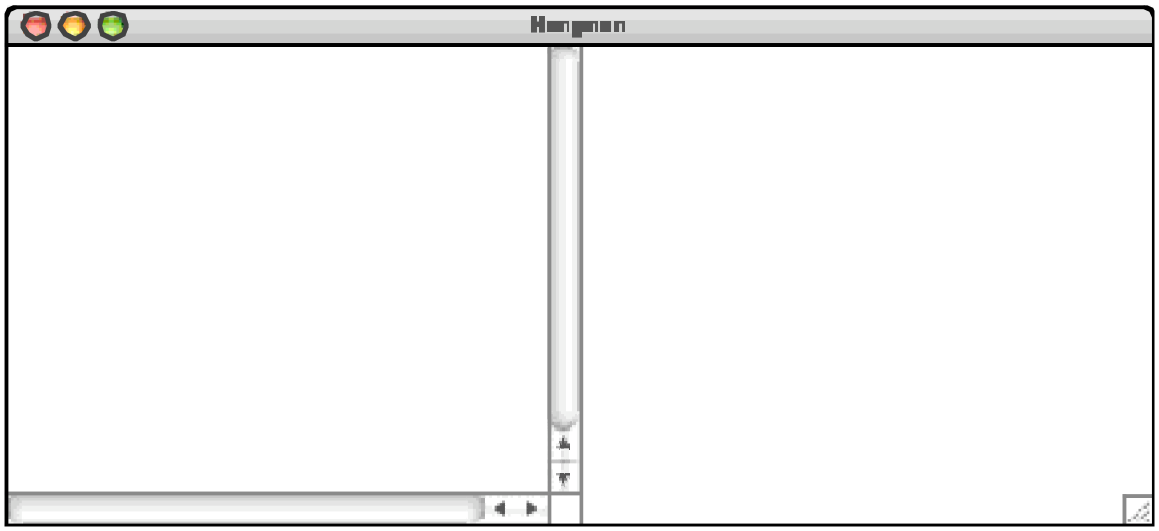
```
canvas = new HangmanCanvas();
setLayout(new GridLayout(1, 2));
add(canvas);
validate();
```

The only confusing lines in this sequence are the **setLayout** call and the call to **validate** at the end. In the Java windowing system, the various displayed objects—which are called **components**—are positioned according to the **layout manager** for the component in which they are contained. Here, the line

```
setLayout(new GridLayout(1, 2));
```

changes that layout to a grid with one row and two equal columns. The console is already installed and will therefore show up in the left column. When you add the **HangmanCanvas** it will occupy the second column, which means that the console and graphics components of the window will share equal space. The **validate** call is necessary to signal the layout manager that it needs to update the layout of the window. Doing so produces the side-by-side display shown in Figure 4. Input and output from the **Hangman** program will continue to appear on the console, and any graphical objects you add to the **HangmanCanvas** will appear in the canvas area on the right.

Figure 4. Screen shot showing side-by-side console and canvas



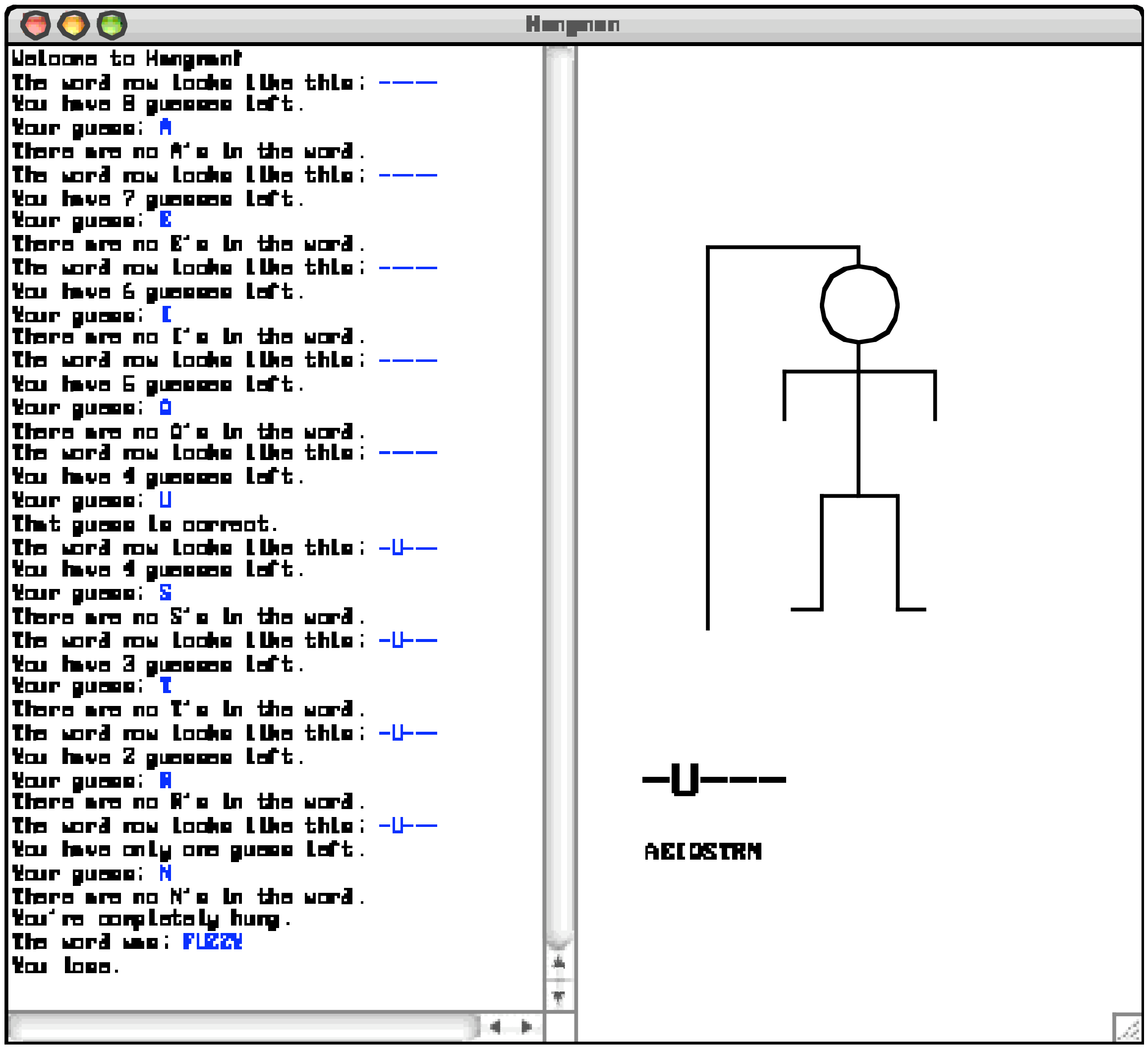
You can now go through and add the calls to the methods in **`HangmanCanvas`**. Every time you start a game, for example, you will need to call

```
canvas.reset();
```

to delete all the body parts from the canvas and redraw the scaffold. Similarly, you will have to call **`displayWord`** and **`noteIncorrectGuess`** at the appropriate points in your code. As of yet, nothing will actually be displayed on the canvas when you make these calls, but your program should run just the same as it did before, freeing you to concentrate on implementing the methods in **`HangmanCanvas`**. Note that you should not add any more **`public`** methods to **`HangmanCanvas`** (**`private`** helper methods are fine).

The implementation of **`HangmanCanvas`** should be reasonably straightforward. Although the sizes of the scaffold and the various body parts are given to you, their positions are not specified, so you will have to do some arithmetic to calculate the coordinates. The center line of the body should be centered horizontally on the screen, and the scaffold should be displayed a bit higher than the center so that there is room underneath for two labels: a label in a large font showing the secret word as it currently stands and a label in a smaller font showing the incorrect guesses. Figure 5 shows how the screen appears at the end of the tragic session in which the user was unable to guess **`FUZZY`**.

Figure 5. The tragic ending of a Hangman game



```
Welcome to Hangman!
The word now looks like this: -----
You have 8 guesses left.
Your guess: A
There are no A's in the word.
The word now looks like this: -----
You have 7 guesses left.
Your guess: E
There are no E's in the word.
The word now looks like this: -----
You have 6 guesses left.
Your guess: I
There are no I's in the word.
The word now looks like this: -----
You have 5 guesses left.
Your guess: O
There are no O's in the word.
The word now looks like this: -----
You have 4 guesses left.
Your guess: U
That guess is correct.
The word now looks like this: -U---
You have 4 guesses left.
Your guess: S
There are no S's in the word.
The word now looks like this: -U---
You have 3 guesses left.
Your guess: T
There are no T's in the word.
The word now looks like this: -U---
You have 2 guesses left.
Your guess: R
There are no R's in the word.
The word now looks like this: -U---
You have only one guess left.
Your guess: N
There are no N's in the word.
You're completely hung.
The word was: FUZZY
You lose.
```

Part III—Reading the lexicon from a data file

Part III of this assignment is by far the easiest and requires considerably less than half a page of code. The only problem is that you won't have seen the Java facilities you need to do it until after the lecture on Friday. So the idea is to start with parts I and II and then fill in this final detail at the end.

Your job in this part of the assignment is simply to reimplement the **`HangmanLexicon`** class so that instead of selecting from a meager list of ten words, it reads a much larger word list from a file. The steps involved in this part of the assignment are as follows:

1. Open the data file **`hangman-lexicon.txt`** using a **`BufferedReader`** that will allow you to read it line by line.
2. Read the lines from the file into an **`ArrayList`**.
3. Reimplement the **`getWordCount`** and **`getWord`** methods in **`HangmanLexicon`** so that they use the **`ArrayList`** from step 2 as the source of the words.

The first two steps should be done in a constructor for **`HangmanLexicon`**, which you will need to add to the file. The last step is simply a matter of changing the implementation of the methods that are already there.

Note that nothing in the main program should have to change in response to this change in the implementation of **`HangmanLexicon`**. Insulating parts of a program from changes in other parts is a fundamental principle of good software design.

Extensions

There are many things you could do with Hangman to make it more fun. Here are some ideas:

- You could spice up the display a little. Stick figures may be fine for elementary school, but they seem a bit tame here.
- You could animate the pictures. Instead of having the body parts and letters merely appear on the screen, you could have them move in from offscreen, as they often do, for example, in PowerPoint slides.
- Once you get the basic structure working, you could expand the program to play something like Wheel of Fortune, in which the single word is replaced by a common phrase and in which you have to buy vowels.
- Use your imagination!