```
Let's play Hangman!

Word:   _ _ _
Misses:
Guess a letter: g

Word:   _ _ G
Misses:
Guess a letter: T

Word:   _ _ G
Misses: T
Guess a letter: g
Already guessed!  Guess a letter: t
Already guessed!  Guess a letter: f

Word:   _ _ G
Misses: T,F
Guess a letter: d

Word:   D _ G
Misses: T,F
Guess a letter: m

Word:   D _ G
Misses: T,F,M
Guess a letter: O
The answer was: DOG
You WIN!
Do you want to play again? Yess!!

Word:   _ _ _ _
Misses:
Guess a letter: u

Word:   _ _ _ _
Misses: U
Guess a letter: I

Word:   _ _ _ _
Misses: U,I
Guess a letter: o

Word:   _ _ _ _
Misses: U,I,O
Guess a letter: a

Word:   _ A _ _
Misses: U,I,O
Guess a letter: R

Word:   _ A _ _
Misses: U,I,O,R
Guess a letter: b

Word:   _ A _ _
Misses: U,I,O,R,B
Guess a letter: p
The answer was: HAND
You LOSE!
Do you want to play again? neIN
You played 2 game(s).  Goodbye!
```

CSE 142, Spring 2008
Programming Assignment #5: Hangman (20 points)
Due: Tuesday, May 6, 2008, 4:00 PM

Program Description:

This assignment focuses on while loops, Boolean methods, and text processing. The program allows the user to play Hangman. Turn in a file named `Hangman.java`. Make sure you have all the required files from the course web site before you start.

Each game of Hangman consists of the program choosing a word and initially displaying each letter of the word with an underscore character (_) and a space. The user has to guess the word by guessing one letter at a time.

- If the user guesses a letter that is *not* in the word, the program adds that letter (in uppercase) to a list of wrong guesses. If the user guesses 6 wrong letters before guessing all the letters of the word, the user loses.
- If the user guesses a letter that is in the word, the program replaces the corresponding underscore(s) with that letter (in uppercase). If the user guesses all the letters of the word before making 6 wrong guesses, the user wins.
- If the user guesses a letter that they have already guessed before, the program notifies them and asks them to enter another letter (see example logs of execution for exact wording).

After each game, the program asks the user if they want to play again. The program should start a new game if the user's response begins with the letter y. That is, answers such as "y", "Y", "YES", "yes", "Yes", or "yeehaw" all indicate that the user wants to play again. Otherwise assume that the user does not want to play again. For example, responses such as "n", "N", "no", "okay", and "hello" would mean that the user doesn't want to play again.

Since you don't know how to read from a file yet, we provide a utility class with a `getWord` method that gets a random word from a file (called `hangman-words.txt`) containing a list of words. On the course web page, there is a list of words you can download or you can create your own list (the name of the file must be `hangman-words.txt`). Words must be on their own line. Only words that consist entirely of letters (no spaces or punctuation) will be used. You can use the following statement to retrieve a random word and store it in the `String` variable called `answer`:

```
String answer = HangmanUtils.getWord();
```

In the utility class, there also is a method to draw a game. The `drawGame` method takes two `String`s as parameters. The first `String` is for the word that the user is supposed to guess (with underscores and letters). The second `String` are all the letters that were wrongly guessed.

You are free to change the drawing mechanism in the `drawGame` method for your own enjoyment. We provide this drawing method, so you can focus on the while loops and other constructs in the assignment. **In fact, you should NOT use `drawGame` until the text version works perfectly as we will be only grading the text output.** (continued on next page)

Implementation Guidelines:

We suggest that you develop this program in stages. One useful stage is to write a program that plays just one game. You can also start by only considering correct letters. If the user types a wrong letter, just ignore it for now. You can even put in code to print out the correct answer for debugging purposes, though you should delete such code before you turn in the assignment. The following might be logs of execution for two in-progress stages of the program, though you do not have to develop the program this way and should not turn in a program that prints the answer:

Log of execution from first hypothetical in-progress version (only consider letters from the correct answer)	*Log of execution from second hypothetical in-progress version (user may guess wrong letters)*

`Let's play Hangman!`
`ANSWER: CAT`
 	
`Word:  _ _ _`
`Guess a letter: T`
 	
`Word:  _ _ T`
`Guess a letter: a`
 	
`Word:  _ A T`
`Guess a letter: C`
`The answer was: CAT`
`You WIN!`
	`Let's play Hangman!`
	`ANSWER: CAT`
	
	`Word:  _ _ _`
	`Misses:`
	`Guess a letter: q`
	
	`Word:  _ _ _`
	`Misses: Q`
	`Guess a letter: B`
	
	`Word:  _ _ _`
	`Misses: Q,B`
	`Guess a letter:`

This program involves a bit of text processing. You should store the guessed letters in a `String`. Whether you store all the guessed letters in one `String` or you separate the correct letters from the wrong letters is up to you. For example, if the correct answer is "dog" and the user has guessed "g", "t", "f", "d", and "m" so far, you can store "gtfdm" as a `String` or separate the correct letters ("gd") from the wrong letters ("tfm"). You'll find the methods `indexOf` and `charAt` from the `String` class to be of general use. See the slides and Section 3.3 of the book for more details.

Assume valid user input. When prompted for a guess, assume that the user will only type one letter. However, the user may enter letters in either uppercase or lowercase and your program must be able to handle that. When the user is prompted to play again, assume the user will type a one-word answer. To deal with the yes/no user response, you may want to use `String` methods described in the slides.

IMPORTANT:

1. While testing your code, you can set the word the user has to guess to anything you want; however, when you submit your program, the word to be guessed must come from a call to `HangmanUtils.getWord()`.
2. Although you may change the method body of the `drawGame` method, do not modify any other existing methods or add new methods in the utility class; do not change any method names, parameters, or return types. For example, the `drawGame` method must take two `String`s and return nothing.

Stylistic Guidelines:

Structure your solution using static methods that accept parameters and return values where appropriate. For full credit, you must have at least **3 non-trivial methods other than main** in your program. Two of these must be the following:

1. a method to play a single game with the user
2. a method to construct the formatted word the user sees

You may define other methods if they are useful for structure or to eliminate redundancy. Unlike in past programs, it is okay to have some `println` statements in `main`, as long as your program has good structure and `main` is still a concise summary of the program. For example, you can place the loop that plays multiple games and the prompt to play again in `main`. As a reference, our solution has 4 methods other than `main` and has 105 lines total.

For this assignment you are limited to the language features in Chapters 1-5 of the textbook. Use whitespace and indentation properly. Limit lines to 100 characters. Use meaningful identifier names and proper capitalization. Include a comment header with basic description information and a comment at the start of each method. Since this program has longer methods than past programs, also put brief comments inside the methods explaining relevant sections of your code.