Lab 2 — Hangman

1. The goal of this lab is to write a Java program to play the game of Hangman. Go to the class web page under *Labs* and download the file **lab02_files.zip**. Unzipping this file will create a folder called lab02_files containing all of the supporting files you will need, which include the following:

   - **WordSet.java** is the WordSet definition that we developed together in class.
   - **TestWordSet.java** demonstrates the use of the WordSet class.
   - **HangmanGallows.class** is a compiled class for drawing the Hangman gallows.
   - **TestHangmanGallows.java** is a program that demonstrates the use of HangmanGallows.
   - **shortlist.txt** is a short text file of words you can use for developing and debugging your game.
   - **longlist.txt** is a much larger file you can use for playing the game once it is fully debugged.
   - **Hangman.java** contains starter code for your game.

2. The program **TextFileInputDemo.java** demonstrates how to read information from text files. We looked at this example in class on Tuesday. As a reminder, open it in DrJava, examine the code, and then run it on the provided text file **haunting.txt**.

3. Next, open **WordSet.java** and **TestWordSet.java** (which we also developed in class). This version of WordSet stores a hard-coded array of word strings. Instead, we would like WordSet to read in the strings from a text file specified by the user (such as **shortlist.txt**). Using the TextFileInputDemo program as a guide, rewrite WordSet so that its constructor takes the name of a text file as an input parameter, opens the file, and reads in each word from the file, storing them in the `words` array. Because opening a file can in principle generate a FileNotFoundException, the WordSet constructor must be declared as shown below:

   ```
   public WordSet(String filename) throws FileNotFoundException {
       ...
   }
   ```

   Don't forget to include the appropriate `import` statements at the top of the file. To test your new version of WordSet, you will also need to modify TestWordSet so that it calls the WordSet constructor with a filename string. For example: `WordSet wordset = new WordSet("shortlist.txt");` Furthermore, since a WordSet may now throw a FileNotFoundException, you will need to add this declaration to the `main` method of TestWordSet, and import java.io.FileNotFoundException.

4. Now you are ready to develop your game. Open **Hangman.java** and examine the code. Several variables are already declared for you, which you will use to keep track of the state of the game. `partialWord` will be initialized to all underscore characters `'_'` at the start of the game, and will get filled in each time the user guesses a correct letter. The string `lettersGuessed` will keep track of all of the user's guesses so far. This way, the program can avoid penalizing the user if they enter a letter that they've already guessed before. In addition, you will implement and test a couple of static helper methods, called `contains` and `showWord`.

5. As a first step, implement the helper method `contains`, which should take a string and a character as input, and return true if the character appears somewhere in the string, or false otherwise. You should use a for-loop. Test your method by temporarily adding the test code shown below to the `main` method of **Hangman.java**. IMPORTANT: make sure not to change the `main` methods in the other files (TestHangmanGallows, TestWordSet, or TextFileInputDemo). From this point on, all new code should go in **Hangman.java**.

   ```
   String s = "apple";
   System.out.println(contains(s, 'a'));
   System.out.println(contains(s, 'e'));
   System.out.println(contains(s, 'x'));
   ```

6. Next, implement and test the `showWord` method, which should take an array of characters representing the mystery word as input, and print it out in a nice, readable way, as shown below:

   ```
   Mystery word: _ P P _ E
   ```

   To test your method, try adding some temporary test code like this to `main`:

   ```
   char[] letters = { '_', 'P', 'P', '_', 'E' };
   showWord(letters);
   ```

7. Next, we need some graphics. The HangmanGallows class can be used to draw the gallows at each stage of the game. (For this to work, your computer must be connected to the Internet.) This class is provided as a pre-compiled .class file, without the source code. You can create a HangmanGallows object in your game, and use it to indicate the number of wrong guesses made so far by the user. Try out **TestHangmanGallows.java** for an example of how to use this class, which provides the following constructor and methods:

   - `new HangmanGallows()` creates an object that displays the gallows in a graphics window.
   - `void drawNext()` draws the next body part in sequence.
   - `void clear()` resets the display back to the beginning.
   - `void close()` closes the display.

8. Your Hangman `main` program should ask the user for the name of a file of words (such as **shortlist.txt**), using a Scanner to read input from the keyboard. It should then create a WordSet object based on the file, and a HangmanGallows object for the graphics. It should also use an array of characters to keep track of the current state of the mystery word. Your game should implement the following features:

   - The mystery word should be printed out using underscores separated by spaces.

   - A maximum of 7 wrong guesses are allowed. After the 8th wrong guess, you lose.

   - The program should report what the mystery word was upon winning or losing.

   - If you accidentally guess a letter that you've already guessed before, the program should tell you, but it should not count this as a new guess.

   - If you guess a letter that is in the mystery word, the program should report how many times the letter occurs in the word using correct grammar, and should fill in all of the appropriate blanks.

   - If you guess all the letters in the mystery word, the program should print an appropriate congratulatory message.

   A possible sample run is shown below:

   ```
   Name of word file to use: shortlist.txt

   Mystery word:  _ _ _ _ _
   Your guess? e
   There is one E in the word.
   Mystery word:  _ _ _ _ E
   Your guess? p
   There are 2 P's in the word.
   Mystery word:  _ P P _ E
   Your guess? t
   Nope, T is not in the word.  (The program draws the first body part on the gallows.)
   ```

```
Mystery word:  _ P P _ E
Your guess? x
Nope, X is not in the word.   (The program draws the next body part.)
Mystery word:  _ P P _ E
Your guess? p
You already guessed P. Try again.
Mystery word:  _ P P _ E
Your guess? x
You already guessed X. Try again.
Mystery word:  _ P P _ E
Your guess? a
There is one A in the word.
Mystery word:  A P P _ E
Your guess? l
There is one L in the word.
Congratulations, you got it! The word was APPLE.
```

A pseudocode outline of the program is provided below. Once you are confident that your game is working correctly, try it out with **longlist.txt**, which contains several thousand words. Have fun!

```
ask the user for the name of the word file to use

initialize the game and generate a random mystery word

repeat as long as the user has guesses left:

    print out the current partial word

    ask the user for their next guess

    if they entered a letter they have already guessed:
        print a warning message and ignore the input

    if they guessed a correct letter:
        print an appropriate message
        fill in the appropriate blanks
        if the user has won:
            print an appropriate message and stop

    if they guessed a wrong letter:
        print an appropriate message
        update the gallows
        if the user has lost:
            reveal the mystery word and stop
```

EXTRA CREDIT (optional)

Modify your program to allow the user to play multiple games. That is, after winning or losing a game, the program should ask if the user would like to play again. If the user types in "yes", the gallows is cleared, a new mystery word is chosen (from the same WordSet object as before), and a new game commences. Otherwise, the program prints a short farewell message and terminates. This process should continue for as long as the user wishes to keep playing.