The code for this assignment is available through Snarf as well as on the course webpage.

Ambient: [https://www.cs.duke.edu/csed/ambient/](https://www.cs.duke.edu/csed/ambient/)

**INTRODUCTION:**

In this assignment you will be doing several things:

1. Reading and understanding a basic code specification
2. Writing a clean and correct implementation of the game Hangman
3. Doing basic analysis using the code given as well as writing some of your own.

**Java Help**

You should expect to take some time getting used to the Java language. There is a cheat sheet available off of the course webpage under the "Help" tab to relate basic Java to Python and MATLAB syntax that you may be more familiar with.

Additionally the Java API docs ([http://docs.oracle.com/javase/7/docs/api/](http://docs.oracle.com/javase/7/docs/api/)) can be of use when looking up how specific method or classes work.

Finally if you have a specific question about a method or a certain class, it is highly recommended to Google the specific thing you are looking for (i.e. "Java String substring method"). This will often turn up exactly what you are looking for.

Finally your classmates and TAs are available on Piazza to answer any questions you have. **DO NOT post your own code on Piazza publically. This will be removed and is in violation of course policy (sharing code with classmates is not allowed).** You can post your code in a private Piazza question though, although it is often more helpful to bring specific code questions to office hours.
Provided Code/Classes

You are provided with four classes:

1. The class **HangmanStats** is the class you will use/modify in part 1. It has a main method `public static void main(String[] args)` that executes the code in **HangmanStats**.

2. The class **HangmanFileLoader** loads a text file and has a method to select and return a random word. Look in the sample code in **HangmanStats** and **HangmanGame** to see how to use this class. **You shouldn’t need to modify this code at all.**

3. The class **HangmanExecuter** has the `public static void main(String[] args)` method that should be the starting point of the Hangman game that you will write. It creates a **HangmanGame** object and invokes the `play()` method. You can write the game **without** modifying this class. **If you do modify it, you must include an explanation at the start* of your analysis.**

4. The class **HangmanGame** is the game-playing code, you’ll modify the `play()` method and you may add new methods of state-instance variables to this class.

Part 1 - Analysis

1a - Statistical/Forensic Analysis

You’re given a class **HangmanFileLoader** that you use in the game to get a word the user will guess. For this first part of the assignment you’ll add to code started for you in the method `numberOfWords(HangmanFileLoader loader)` in **HangmanStats.java** to answer the question below.

We want you to estimate the number of different words there are of 4, 5, 6, and so on up to 20 letters long. For example, to estimate the number of 6-letter words you can repeatedly call `HangmanFileLoader.getRandomWord(6)` and add the returned word to a set. The size of the set is an estimate of the number of words of length 6.

When we modified this code to call `HangmanLoader.getRandomWord` 10,000 times for word-lengths of 4-10 we obtained the below results from one run.
If run again, the numbers above will be similar, but different. You'll need to decide on a strategy and write code to get estimates of the number of different words for each word-length from 4-20 (inclusive).

1b - Your Own Question

Pose your own question about the words and the code that generates the random words of a specific length in HangmanFileLoader. Write code to answer your question in statisticalQuestion(HangmanFileLoader loader) in HangmanStats.java.

Remember, to calculate an average you will need to run your code multiple times to collect data. Any question you write is fine, but you must write code to answer the question. You’re welcome to come up with ‘extra’ questions (up to three) that you don’t write code for, but which you’d like to be able to answer/write code to answer. Include all questions in your analytic write-up.

Analysis.pdf

In your Analysis.pdf you'll explain your methodology for Part 1a and 1b. Include your questions and data to support your conclusions. You're welcome to create and use charts, tables, etc.

You should also include the code you wrote to create your statistical analysis and explain how you ran the code to get the data you used in your analysis.
To get full credit you must be thorough when writing up your assignment.

**Part 2 - Hangman Game**

Write a program to play a console-based, word-oriented game of hangman. The user should be allowed to specify the number of letters in the word and the number of misses until the game is lost (see the sample runs below for details). The program should be reasonably robust in the face of faulty input from the user, though don’t go overboard in writing code to protect against bad input (i.e. characters other than letters, more than one character). One detail you should consider is duplicate letters. If you have already guessed ‘e’ and then you guess ‘e’ again, that should not cost you a guess.

You will modify the `play()` method and you may add new methods of state-instance variables in the `HangmanGame` class.

Basic coding guidelines are included at the end of this write up.

**Sample Program Output**

Here’s a sample run of Hangman. You do not need to follow the format exactly, but you should include with each turn the player takes the following information:

- Secret word, showing blanks and correct guesses.
- Number of misses left until hanging occurs.
- Letters that have been guessed incorrectly.

In the run below the user input is in italics, the program prints the other text.

```
# How many letters in guess word: 8
# How many guesses to hanging: 7
misses left: 7
guesses so far: 
# What letter do you want to guess: e
no e

misses left: 6
guesses so far: e
# What letter do you want to guess: a
no a

misses left: 5
guesses so far: a e
# What letter do you want to guess: o
no o

```
Extra Credit

Words are kind of old - offer the user the choice of playing hangman in more than one category, reading “words” from files you provide, e.g., instead of words use actors, books, and so on.

To obtain extra credit, your extra credit submissions should be 100% functional. There is no partial extra credit.

Additionally you should not overwrite the base assignment in doing the extra credit. This will hurt your grade.

Make the extra credit potions of the assignment easily accessible for the TA, or you will not get credit. We should not have to dig to figure out what you have done. For this assignment extra credit should be accessible when the program is run (i.e. the
code should ask what category to play before starting the game). You should also have a section entitled Extra Credit in your Analysis.pdf explaining what you did and how it works.

Coding Guidelines

Well-written code is easy to read and makes graders happy. The sample output should be generically followed in that the user should see a representation of the secret word with guesses filled in, the letters used so far, and so on.

Additionally:

- Methods should be brief/short, ideally 10-20 lines, but that’s a guideline, not a requirement.
- Methods that do more than one thing are candidates to be divided into more than one method so that each method does only one thing. A prime candidate is the play() method, it will consist of picking letters, updating word-displays, determining if the game should continue, validating input, and so on. In principle, each of these should be a different method, though in practice that’s not always possible.
- A method should minimize its side-effects. Changing one parameter and returning a value are OK. Changing multiple parameters and returning a value should be avoided, though sometimes this is ok too — have a justification for the code you write.
- Use good method and parameter names and follow the Java naming guidelines. Good names make it evident as to what a method does and what a parameter represents.
  - Examples of good names: carPrices, usedLetters, currentLocation
  - Examples of bad names: floatArray, hvs, uLets, currentPlayer
- Brief comments should accompany any complex code.

Submitting

Submit your code, README.txt, and Analysis.pdf using either the Ambient plugin, or Ambient web submit (https://www.cs.duke.edu/csed/websubmit/app/). You should submit to the Hangman project.

Incorrect submissions will incur a penalty.

Grading

- Part 1 - 4 points total
  - 1a code - 1 point
  - 1a write up - 1 point
- 1b code - 1 point
- 1b write up - 1 point

- Hangman Game - 6 points total
  - Guess count tracking - 1 point
  - Guessed letters - 2 points
  - Saving/displaying secret word - 2 points
  - Correct end of game - 1 point

- Extra Credit - 1 point possible