Independent Java Project Narrative:

During the course of thinking of this project and actually coding it I came up against quite a bit of problems. One of the first problems that I encountered happened before I even started to code. I was going to do a black jack/gambling game, but I thought that would be too easy and I would get bored with it. I ended up choosing Evil Hangman from nifty Stanford. The thing that attracted me to this idea was the concept of a cheating Hangman and also that I had no idea how to do it, which in hindsight may have led to some problems later on.

When I first started to write the code the first thing that I needed to solve was how to import a file into dr.java. After about two days of research and trial and error John and I figured out how to use the scanner to go through the document. My code used a while statement, hasNext(), and nextLine() to add all the words in the document to an arraylist. When I look at it now the solution was relatively simple, but I blew how difficult it was out of proportion.

Another problem that I had was the families and how to sort them. The reason why this was so difficult was because if there were too many for loops the program would lag. Another difficulty about the families is that I didn't know how to actually do it. I knew that for me to be able to find out where the letters in each string I would need to use substrings, but I wasn't sure of how to keep track of it. I decided that the best plan of action would be to split of the making of families into different sections to make it easier. The first thing that I did to narrow down the amount of words and different cases I would need to go through was to narrow down the list of words to only those that matched the inputted word length. The word length actually played a key part in figuring out how to loop through the different words. The reason being is that if you set the length in for loop wrong you would get an error, and by using a variable, such as word length, you make the loop universal for all types of word lengths. The reason using word length was so vital was to make the families work I needed to use two for loops; one so that it went through the whole word length and the second so it would go through the word. But before I did I used another method to help narrow down the amount of words again. I used a method that would take in a word and a letter, the guessed letter, and tell you how many times that letter appears in the word. Using this I figured out one part of the puzzle of families. By doing this I was able to separate the dictionary into separate groups depending on how many times the letter showed up in the word. I would put each word into an array of arraylist and depending on how many times the letter showed up they would be put into the corresponding arraylist at that index (ie the letter shows up twice the word is put in the arraylist at the 2^{nd} index of the array). I then would pick one of those at random, but first making sure that it wasn't empty. That was how I solved the first part of families but I still needed to figure out how to make sure the word that was randomly chosen and the rest of the words in the word bank had the same placement of the guessed letter.

For this second part of the problem I worked backwards. Instead of making all the words in the word bank have the same placement from the beginning I would take what is being displayed to the player and then used a backwards for loop to take out the words that didn't share the same placement of the letter. I did this by using substring and .equals(). By doing this I made sure that not only did the words all have the correct placement but also made the chances of the program messing up because the words didn't match up less likely.

(forgot to mention early) I also used a third method that would randomly choose whether or not the person's guess was in correct (don't know if that's the right word). The one stipulation being that if taking out the words that didn't fit the condition meant that there were no more

words then the condition would flip. This method worked by creating a clone of the arraylist so that it wouldn't erase all the words and for loops. The for loop in this method just looked to see if the letter appeared, so pretty much a simpler loop that was used to see how many times a letter appeared.

Another difficulty that I faced was how to get the display working. The reason why it was difficult to get the display to work was because I need the computer to display the correct amount of –'s and then for the computer to display the letter in the correct place, while also keeping the dashes. I tried a few different ways but I kept on over thinking things and the solution, which Mark helped me come up with, was something that I had done before and is relatively easy to do. All the solution did was print out what was being displayed up to where the new letter appears then skip one of the display things and print out the rest. But there was one problem that I couldn't solve and it seems to randomly happen. The problem was that the display would randomly show letters in place of other letters but then fix itself later on in the game.

Another thing that I messed up on while creating my game was the design and how the different methods act with each other. The way I had it originally made it so that my program would glitch out because of bad communication between the different classes. Luckily I was able to catch this before the problem became too big. To solve this problem I made more getter and setter methods that I thought I didn't need in the beginning.

The funny thing about this project was that on the day that we beta tested there were a bunch of super little things that were wrong, but still allowed the game to work. Once I fixed these small problems the game worked fine but still once in a while if the correct scenario comes about there will be bugs. I think that if I had slightly more time and was able to troubleshoot more I could polish this game and make it work almost perfectly.

During this project I hoped that I would learn how to code better and get a better understanding of the different things in java. The reason being is that when you have to do a big project there are many things that could come into play and to make all those things work you need to understand it. I also kniw that from last year when we had to do the bigger project I came out with a better understanding of the code and how to code. I think that same happened with this project too and I became a better coder.

Next time I think I would try to not over think things. In this project I think that I did that a little too much and in doing so made my code more complicated and buggy than it needed to be. I think I would also focus on the little things because most of the bugs in my code came about because of same errors in my coding that led to bugs.