Write a text-based program to play a game of Hangman. (If you’re not familiar with the game, check out the Wikipedia page [here](http://en.wikipedia.org/wiki/Hangman_(game)).) Your program will read in a dictionary file (provided) and randomly choose a word. The user will then guess letters until they either guess the word or run out of guesses. There is a sample program you can run posted to WebAccess.

There are two parts to this project. The first is writing the game and getting it working properly. The second is adding exception handling. I strongly recommend you complete Part I before moving on to Part II.

**Part I: The Game (50 points)**

Your game must follow these rules:

* The user gets a pre-defined maximum number of wrong guesses. (My program uses 7.)
* If a user guesses the same wrong letter twice, this letter should only count once towards the maximum wrong guesses.
* The game should ignore the case (upper or lower) of guesses.
* If the user guesses a correct letter, **all** instance of that letter should be revealed.

Here are some notes to help:

* Below is pseudocode for the game. You are **not**required to use this approach. but you might find it helpful.

*read in the list of words in the dictionary file*

*randomly choose a word from this list*

*while the user still has guesses left and they have not guessed the word*

*print the word (displaying guessed letters and blanks for non-guessed letters)*

*read the user’s guess*

*if the user hasn’t already guessed that letter*

*check if the guess is right or wrong*

*if the user didn’t guess the word, update the guesses remaining*

* Break your code up into methods. Do not have the entire game in the main method.
* You will likely need several instance data variables to keep track of things. Below are some recommendations. You are not required to use these.
  + counters: numLetters (size of the selected word), numIncorrectGuesses, numLettersGuessedCorrectly,
  + char[] selectedWordArray- you might find it helpful to keep the characters of the selected word in a char[]. This will allow you to loop through the array and compare each letter to the user’s guess.
  + boolean[] guessedLetter- you might find it helpful to use a boolean[] that represents whether the letter at each position has been guessed. This will be useful when printing the word to the user (as letters and blanks).
  + ArrayList<Character> lettersGuessed- keep track of which letters have been guessed
* Hint for testing: Print out the randomly selected word you are trying to guess. This makes for much easier testing!
* Again, I strongly recommend getting the game working before moving on to Part II.

**Part II: Exception Handling (50 points)**

* Add exception handling to cover three erroneous occurrences.
  + Note: I realize you could write a working game that accounts for these situations without using exception handling. But, for this project, you are required to use exception handling.
* Situation One: The dictionary file does not exist.
  + Use an existing Java exception type to deal with this.
  + Your program can end in this situation. But it should end gracefully with a nice message- not crash with an error.
* Situation Two: The user enters a guess that is not a character (like + or $)
  + Create your own exception type to represent this situation.
  + When the situation occurs, throw an object of the type you just created. Catch the exception and print a message to the user about what went wrong.
  + The user continues on and enters a new guess. The invalid guess does not count against the user.
  + Hint: check out the Character class for help with detecting this situation!
* Situation Three The user enters a guess that is longer than one character (like aa or zb)
  + Create your own exception type to represent this situation.
  + When the situation occurs, throw an object of the type you just created. Catch the exception and print a message to the user about what went wrong.
  + The user continues on and enters a new guess. The invalid guess does not count against the user.
* Your main method should **not** terminate because of any of these thrown exceptions. All thrown exceptions should be caught and handled.

**Extra Credit**

15 points extra credit: Allow the user to play multiple games. Keep track of the number of wins, losses, and the win percentage. Print this information at the end of each game.

Submit a zip file containing your java files and the dictionary file. Include your name in the name of the zip file. If you are working in a group, submit only one assignment. Put the names of all group members in the comments of each java file.