handout #11

CSE143—Computer Programming II
Programming Assignment #4
due: Thursday, 2/6/14, 9 pm

thanks to Keith Schwarz for this "evil hangman" assignment

This programming assignment will give you practice with the Java collections classes and interfaces. You are going to write a class that keeps track of the state of a game of hangman. But this won't be any ordinary game of hangman. Our hangman program is going to cheat.

If you aren't familiar with the general rules of hangman, you should review the wikipedia entry for it:

http://en.wikipedia.org/wiki/Hangman_%28game%29

In a normal game of hangman, the computer picks a word that the user is supposed to guess. In our game of hangman, the computer is going to delay picking a word until it is forced to. As a result, at any given point in time there will be a set of words that are currently being used by the computer. Each of those words will have the same pattern to be displayed to the user. A client program called HangmanMain has been written for you. It handles the user interaction. You are to write a class called HangmanManager that keeps track of the state of the game. You won't be writing any code that prints information or reads information from the user because all of that code is in HangmanMain.

Your class should have the following public methods.

	Description
anager(Collection<String> nt length, int max)	Your constructor is passed a dictionary of words, a target word length, and the maximum number of wrong guesses the player is allowed to make. It should use these values to initialize the state of the game. The set of words should initially contain all words from the dictionary of the given length, eliminating any duplicates. It should throw an IllegalArgumentException if the collection size is less than 1 or if the max is less than 0.
words()	The client calls this method to get access to the current set of words being considered by the hangman manager.
,eft()	The client calls this method to find out how many guesses the player has left.
:haracter> guesses()	The client calls this method to find out the current set of letters that have been guessed by the user.
rn()	This should return the current pattern to be displayed for the hangman game taking into account guesses that have been made. Letters that have not yet been guessed should be displayed as a dash and there should be spaces separating the letters. There should be no leading or trailing spaces. This method should throw an

	IllegalStateException if the set of words is empty.
ıar guess)	This is the method that does most of the work by recording the next guess made by the user. Using this guess, it should decide what set of words to use going forward. It should return the number of occurrences of the guessed letter in the new pattern and it should appropriately update the number of guesses left. This method should throw an IllegalStateException if the number of guesses left is not at least 1 or if the list of words is empty. It should throw an IllegalArgumentException if the list of words is nonempty and the character being guessed was guessed previously.

The constructor uses the Collection<E> interface for its parameter so that a client can pass either a set or a list. The interface has the usual iterator() method for getting an iterator and allows access with foreach loops.

There are two unusual aspects of the guesses method. It returns a SortedSet rather than a Set. SortedSet is a variation of the Set interface that requires that the values in the set appear in increasing order. We want this property for the set of guesses so that we see guesses in alphabetical order. The TreeSet class implements this interface, just as TreeSet implements the Set interface. Also notice that it is a set of Character values. Character is the wrapper class for char values. You can generally manipulate the set as if it were a set of simple char values (e.g., calling add or contains with a simple char value).

Notice that the pattern and record methods throw an exception when the list of words is empty. The only way this can happen is if the client requests a word length for which there are no matches in the dictionary. For example, the dictionary might not have any words of length 25.

As noted earlier, this version of hangman cheats. It doesn't actually pick a word until it needs to. Suppose that the user has asked for a 5-letter word. Instead of picking a specific 5-letter word, it picks all 5-letter words from the dictionary. But then the user makes various guesses, and the program can't completely lie. It has to somehow fool the user into thinking that it isn't cheating. In other words, it has to cover its tracks. Your HangmanManager object should do this in a very particular way every time the record method is called. Let's look at a small example.

Suppose that the dictionary contains just the following 9 words:

[ally, beta, cool, deal, else, flew, good, hope, ibex]

Now, suppose that the user guesses the letter 'E'. You now need to indicate which letters in the word you've "picked" are E's. Of course, you haven't picked a word, and so you have multiple options about where you reveal the E's. Every word in the set falls into one of five "word families:"

- "- - - -": which is the pattern for [ally, cool, good]
- "- e - -": which is the pattern for [beta, deal]
- "- - e -": which is the pattern for [flew, ibex]
- "e - - e": which is the pattern for [else]
- "- - - e": which is the pattern for [hope]

Since the letters you reveal have to correspond to some word in your word list, you can choose to reveal any one of the above five families. There are many ways to pick which family to reveal – perhaps you want to steer your opponent toward a smaller family with more obscure words, or toward a larger family in the hopes of keeping your options open. In this assignment, in the interests of simplicity, we'll adopt the latter approach and always choose the largest of the remaining word families. In this case, it means that you should pick the family "- - - -". This reduces your set of words to:

[ally, cool, good]

Since you didn't reveal any letters, you would count this as a wrong guess.

Let's see a few more examples of this strategy. Given this three-word set, if the user guesses the letter O, then you would break your word list down into two families:

- "- o o -": containing [cool, good]
- "- - - -": containing [ally]

The first of these families is larger than the second, and so you choose it, revealing two O's in the word and reducing your set of words to

[cool, good]

In this case, you would count this as a correct guess because there are two occurrences of O in the new pattern.

But what happens if your opponent guesses a letter that doesn't appear anywhere in your word list? For example, what happens if your opponent now guesses 'T'? This isn't a problem. If you try splitting these words apart into word families, you'll find that there's only one family – the family "- o o -" in which T appears nowhere and which contains both "cool" and "good". Since there is only one word family here, it's trivially the largest family, and by picking it you'd maintain the word list you already had and you would count this as an incorrect answer.

To implement this strategy, you should use a map. The keys will be the different patterns for each word family. Those keys should map to a set of words that have that pattern. For each call on record, you will find all of the word families and pick the one that has the most elements. This will become the new set of words for the next round of the game. If there is a tie (two of the word families are of equal size), you should pick the one that occurs earlier in the map (i.e., the one whose key comes up first when you iterate over the key set).

You are expected to do some error checking, as outlined in the descriptions of the public methods. But you aren't checking for all possible errors. You may assume that the list of words passed to your constructor is legal in that it will be a nonempty list of nonempty strings composed entirely of lowercase letters. You may assume that all guesses passed to your record method are lowercase letters.

You should use the TreeSet and TreeMap implementations for all of your sets and maps. You should use interfaces for all variables, fields and parameters. You should avoid making a value a field when it can instead be a local variable. You should implement these operations in a reasonably efficient manner. You should thoroughly document all methods of your class and include a general description of the class in the class header. And you should introduce private methods to avoid redundancy and to break up large methods into smaller methods. In particular, **you should not have any methods that have more than 20 lines of code in their body (not counting blank lines and lines that have just comments or curly braces)**. If you have a method that requires more than 20 lines of code, then you should break it up into smaller methods.

This program comes with several resource files including HangmanMain.java and dictionary.txt. The file dictionary.txt contains a huge dictionary of over 127 thousand words that is the official English Scrabble dictionary. It has some unusual entries, but you can go to http://www.hasbro.com/scrabble/en_US/search.cfm to look up the definitions to see that these are words that are considered legal in Scrabble. The resources will all be included in a zip file called ass2.zip. You will find that HangmanMain has two constants that you might want to change. The first is for the name of the dictionary file. By default, it will read from dictionary.txt. You might want to change this constant to dictionary2.txt which is a short dictionary of 9 words also included in the zip file. These are the 9 words used in the short example earlier in the write-up. You might also want to change the setting for SHOW_COUNT. By default it is set to false. By setting it to true, you will be shown how many words there are in the current set of words as you play the game.

Keep in mind that in playing hangman, you have a certain number of wrong guesses that you are allowed to make, but this isn't the same as the total number of guesses made. Sometimes you guess a letter that is in the word (a correct guess) and sometimes you guess a letter that isn't in the word (a wrong guess). The user has a certain number of wrong guesses allowed. The user can make an indefinite number of correct guesses because they don't count against them.

It is likely that you will make a mistake somewhere in specifying your generic structures. When you do so, the Java compiler will warn you that you have "unchecked or unsafe operations" in your program. You will lose style points if you don't fix these warnings. You can have jGRASP show you the exact line by going to: Settings/Compiler Settings/Workspace/Flags / Args and then uncheck the box next to "Compile" and type in:

```
    -Xlint:unchecked
```

You should name your file HangmanManager.java and you should turn it in electronically from the "Homework" tab on the class web page.