Project 2: Due March 15 by 3:00pm

COP 3502 Spring 2017

Post-Apocalyptic Poetry Interpreter

The Scene.

The year is 2217. World War IV has recently ended. It engulfed planet earth in tragedy, and in its aftermath, governments everywhere have cracked down on artistic expression. By stifling the strong emotions that are often expressed through art, the governments of the world hope to avert future wars.

But the human spirit is resilient. People all over the globe have joined the Poetry Movement in a commitment to continue creating and sharing poetry of all kinds. They are determined to prevent this post-apocalyptic world from becoming a post-artistic world.

However, they must be stealthy. If they are caught transmitting their poetry, the punishment is prison for an indefinite length of time.

The poets have devised a method of communication that embeds their poems into seemingly random strings of hexadecimal digits. Because of your reputation as a trusted member of the Poetry Movement, you have been approached by a young man in need of your help. He chose to forego all the computer science classes offered in his elementary, middle, and high school, so he doesn't have the skills to encode or decode poetry. But his beloved was stranded on the other side of the world when war broke out, and she has been sending him encrypted poems.

Your mission is to assist this young man to decode the poems from his beloved. When you have done this, your program will also be able to decode poems of every kind! You will be supporting love, family, and freedom for poets the world over!

You should know: The poems that are created in this post-apocalyptic expressionism are different than the poems of old. They are less sentential in form, and more like abstract word art. This structure allows them to much more easily evade detection. But as you will soon see, they can be just as meaningful.

The Task.

DO NOT start your own code from scratch. Paste the skeleton code provided (Canvas -> files -> Project 2 -> PoetryMovementSkeletonCode.java) to start your own .java file. Read all of these instructions before doing so, because there are requirements for naming your file and for importing the necessary libraries that you will be using.

The encoded poems that you will take as input appear to be a randomly formatted string of hex digits. For example, here is one (hex digits are NOT case sensitive):

6C6F7665

In order to decode this hex string into the words it contains, you must pair up the hex digits – in this case, it will be 6C, 6F, 76, 65. You must transform those hex digit pairs into their corresponding ASCII character. (http://www.ascii-code.com/)

6C = l

6F = o

76 = v

65 = e

This hex string, 6C6F7665, actually spells "love". **Note: you do NOT need a long if-else or switch statement to translate these hex digits into letters. You can do one-line character arithmetic that translates the hex code into a valid English letter.

After you have decoded your hex string into a string of English letters, the real computational task begins.

You must search through the string of English letters to identify all words that are in the English language. We are providing you with a toolkit to check if a word is in the dictionary, so you don't have to worry about that part (you will just call the method isWord() which is provided in the skeleton code).

You must identify words of length 4, 5, and 6. Traverse your string checking for words of all of those lengths. The same characters may be members of more than one word.

You must download the skeleton code provided by the teaching staff and use it as the starting point for your own code.

Here are some examples of how some of your methods should work:

An example of hex character translation: (for an odd number of digits)

0f is mapped to p

6f is o in the ascii table

77 is w in the asci table

Ex: String letter := “fourgooddogsswam”, int wordSize := 4

This method would check these four-letter substrings:

four

ourg

urgo

rgoo

good

oodd

oddo

ddog

dogs

ogss

gssw

sswa

swam

The string array returned from your method would be ["four", "good", “dogs", “swam”, null, null, null, null, null, null, null, null, null]

It is possible that the string of hex digits is too short to support looking for some lengths of words. For example, if only six hex digits are provided, that would translate into three English letters, which cannot produce a word of lengths 4, 5, or 6. In that case your program should not crash, it should simply not give any output. You will have to handle this case within your methods. If the input is long enough to find words of size 4 but not larger, or up to size 5 but not larger, you must find those words and output a poem as normal. The program will simply not check for larger words if the input is not long enough to support that length of word.

Formatting the poem for output.

You must format the poem as follows for output. Note that your method will be returning a formatted String and the ONLY println will happen in the main method which is fully provided for you. To insert tabs and line breaks into your String, you will insert \t and \n characters, respectively. For example, the string “theirs earth mine\n\tyours ours\n\t\tours\n\t\t\tours” will print as follows. Note the line break and the tabs. The \n and \t are treated as one single special character in Java.

```
theirs earth mine
        yours ours
                ours
                        hers
```

All poems are formatted with at most one *word of each length* per line. First, the word of longest length (6) appears if there is such a word that hasn’t already been placed in the poem. Then a space, then a word of the medium length (5) if available. Then, finally a word of the shortest length (4) if available. Each successive line should be indented one additional tab more than the previous line. See the example output poems at the end of this document for details on how it works with different sets of words.

Instructions for setting up your project workspace:

First, create your project. In Eclipse, click File -> New -> Java Project. **Name it PoetryMovement.** Make sure that you check the option to create a **separate src folder**.

Then, in the src folder that was created, make a new Java Class (a new .java file will result). This file will be your main source code, where you will create your solution to the project. Name this Java file **PoetryDecoder.**

Now, in your PoetryDecoder.java file is where you will **copy-paste the skeleton code** provided for you below. This is also the file you will turn in for grading. You will only submit PoetryDecoder.java.

Finally, you must add an external set of utilities to your project space.

First, download the jar file with utilities. Add this .jar (Java Archive, a kind of zip file) to your project so that your code will be able to call the packages in it to determine what is a word. After you have downloaded the .jar to your computer, right-click on your project name in Eclipse and choose Properties. On the left bar click Java Build Path. On the top tabs, click Libraries. Then click the Add External Jar button on the right. A file chooser will appear. Navigate to wherever you saved the .jar file we provided. (It may be in your Downloads folder by default depending on how you have your computer set up.) After you do this, all the import statements in the skeleton code will compile! Before you import the jar, the skeleton code won’t be able to find the packages we import (for example, org.jsoup.Jsoup, etc). They are in that jar that you add to your project.

Instructions for submitting:

Turn in only the PoetryDecoder.java file on Canvas. If you do not follow the naming requirements above or if you rename any methods (or change their signatures) in the skeleton code provided, your code will lose all points associated with automatic testing. The teaching staff will automatically test your code using JUnit testing, which we will cover in class later. JUnit testing requires that our code be able to call your code by the precise method names specified, which is why you must not change them. There is a second reason that this structure benefits you: it helps you learn how to organize a program into methods that make sense and communicate with each other.

Documentation:

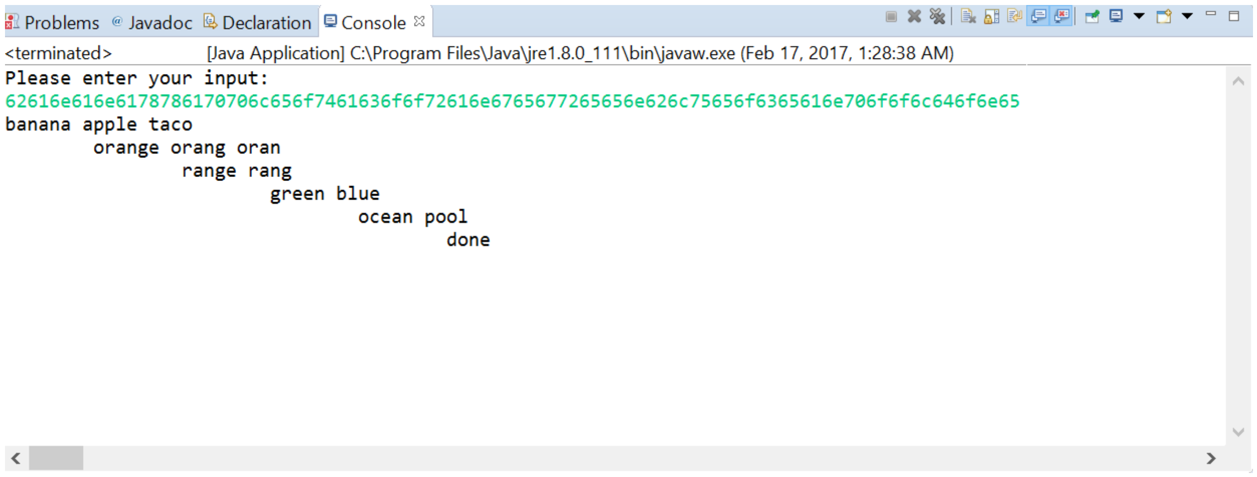
The skeleton code has a lot of instructions for you. Before you turn in your code, replace the instructional documentation with appropriate comments of your own. Some of the existing comments will remain, but you should comment each method and the class as you normally would.

Test cases:

```
Please enter your input:
6c6f766564
loved love

Please enter your input:
736b69657368667368696e6568666c6f76657070666c6f77626277777468656d
skies kies
        shine shfs
                shin
                        love
                                flow

Please enter your input:
7368696e657362626c6f76657071707175706f6e72776f726c64
shines shine shin
        world love

Please enter your input:
6d696e65707068656172746e6d75706f6e7161686561727473676e657665726170617274
hearts heart mine
        heart hear
                never hear
                        apart arts
                                neve
                                        ever
                                                apar
                                                        part
```

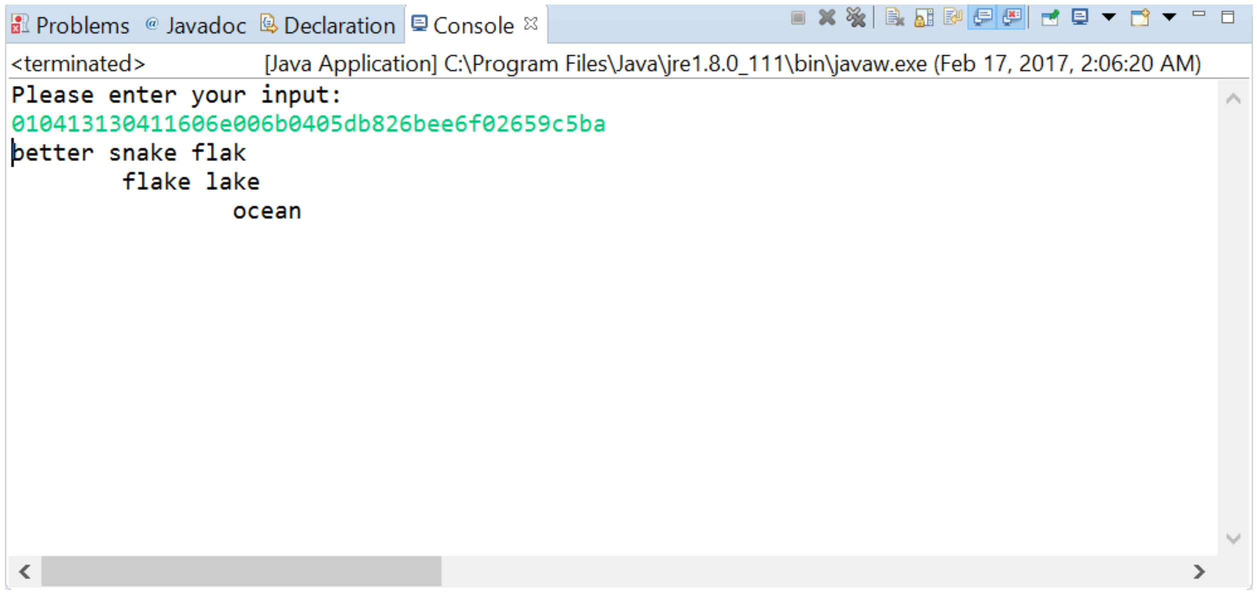
(Note this took 1 min and 10 seconds to display, be patient!!)

```
Please enter your input:
62616e616e6178786170706c656f7461636f6f72616e6765677265656e626c75656f6365616e706f6f6c6
46f6e65
banana apple taco
        orange orang oran
                range rang
                        green blue
                                ocean pool
                                        done
```


Problems @ Javadoc Declaration Console

```
<terminated>        [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (Feb 17, 2017, 1:28:38 AM)
Please enter your input:
62616e616e6178786170706c656f7461636f6f72616e6765677265656e626c75656f6365616e706f6f6c646f6e65
banana apple taco
        orange orang oran
                range rang
                        green blue
                                ocean pool
                                        done
```

```
Please enter your input:
010413130411606e006b0405db826bee6f02659c5ba
better snake flak
        flake lake
                ocean
```


Problems @ Javadoc Declaration Console

```
<terminated>        [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (Feb 17, 2017, 2:06:20 AM)
Please enter your input:
010413130411606e006b0405db826bee6f02659c5ba
better snake flak
        flake lake
                ocean
```