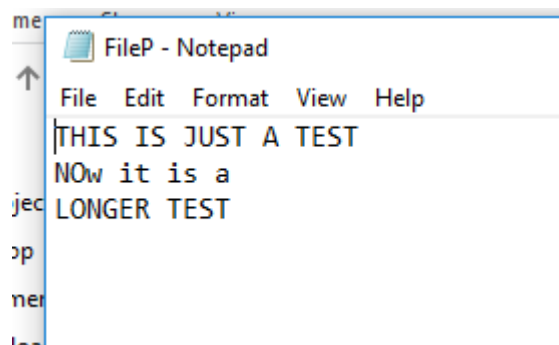


## Programming Assessed Exercise 2

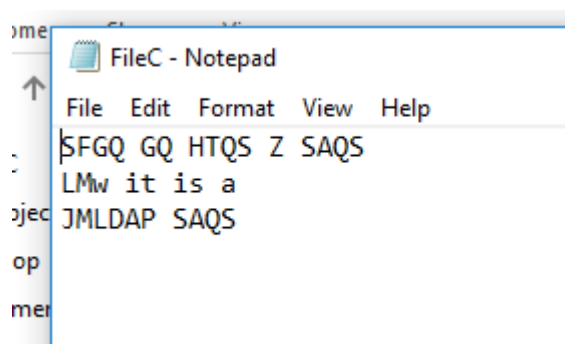
Stuart Rawlinson – 2376095

The program meets all the requirements specified in the brief. Technically, the file is read before the program asks for the keyword, but that was to ensure the user encountered a 'File Not Found' exception immediately after entering its name, rather than later in the program.

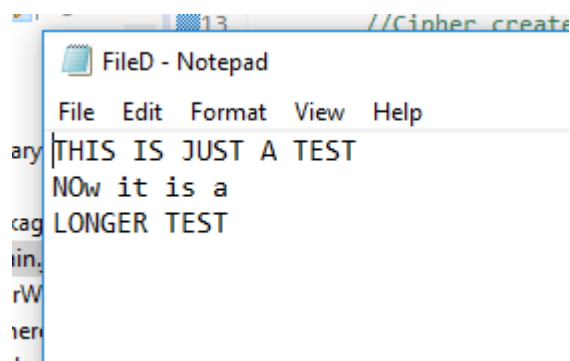
For testing I used a file called FileP to demonstrate how my program covers messages using a mixture of upper and lower case letters across multiple lines. The contents of FileP are shown below:



Using the Monoalphabet cipher with the keyword 'ZEBRA' transforms the text into 'FileC', as shown below. 'FileC' demonstrates that the cipher only operates over the capital letters while ignoring other characters such as the new line character, which it replicates.



Using the same keyword on the Monoalphabet cipher ensures that 'FileD' is translated to the same text as within 'FileP':



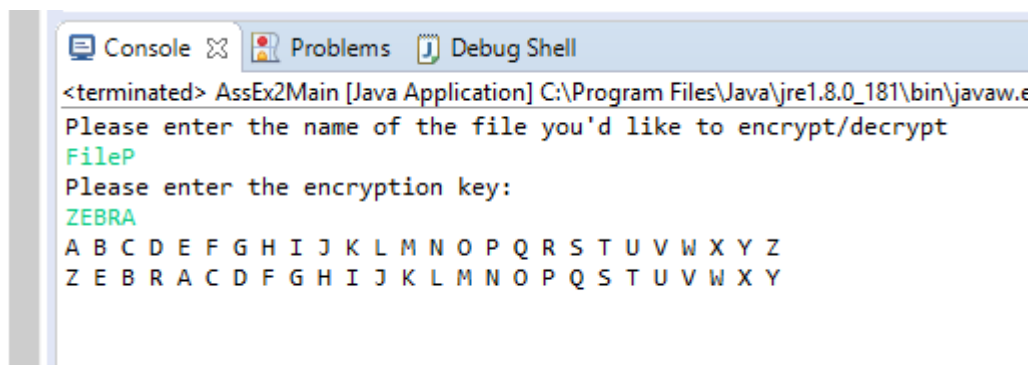
The 'FileF' output shown below is the one generated with 'FileD'. I used methods within the 'LetterAnalyser' class in order to format this output. While the output is as expected, I'm not certain this section of the program is as efficient as it could be, and using a mixture of tabs and string

formatting means the columns format may not be completely reliable as the numbers involved get larger.

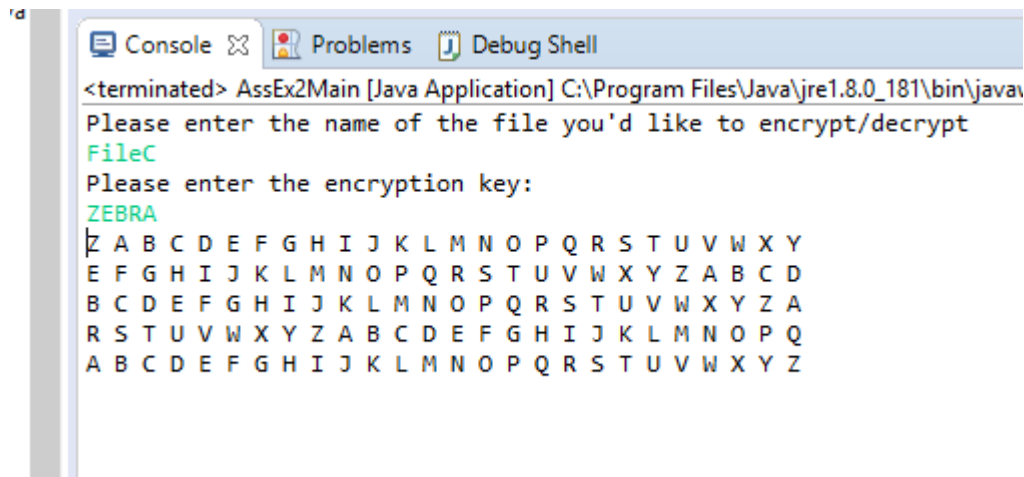
```
LETTER ANALYSIS:
Letter      Freq      Freq%      AvFreq%     Diff
A           1         3.7         8.2         -4.5
B           0         0.0         1.5         -1.5
C           0         0.0         2.8         -2.8
D           0         0.0         4.3         -4.3
E           3        11.1        12.7         -1.6
F           0         0.0         2.2         -2.2
G           1         3.7         2.0          1.7
H           1         3.7         6.1         -2.4
I           2         7.4         7.0          0.4
J           1         3.7         0.2          3.5
K           0         0.0         0.8         -0.8
L           1         3.7         4.0         -0.3
M           0         0.0         2.4         -2.4
N           2         7.4         6.7          0.7
O           2         7.4         7.5         -0.1
P           0         0.0         1.9         -1.9
Q           0         0.0         0.1         -0.1
R           1         3.7         6.0         -2.3
S           5        18.5         6.3        12.2
T           6        22.2         9.1        13.1
U           1         3.7         2.8          0.9
V           0         0.0         1.0         -1.0
W           0         0.0         2.4         -2.4
X           0         0.0         0.2         -0.2
Y           0         0.0         2.0         -2.0
Z           0         0.0         0.1         -0.1
The most frequent letter is T at 22.2
```

Aside from the possible inefficiency the output does what is required.

The following images show that the Monoalphabet cipher and the Vignere cipher are both generated as expected. I used a multi-dimensional array for the Monoalphabet cipher, incorporating the regular alphabet, as I thought this would make the encryption easier to perform. The Vignere does not have a built-in array with the alphabet as the multiple rows involved with encryption would make the encryption process more difficult.



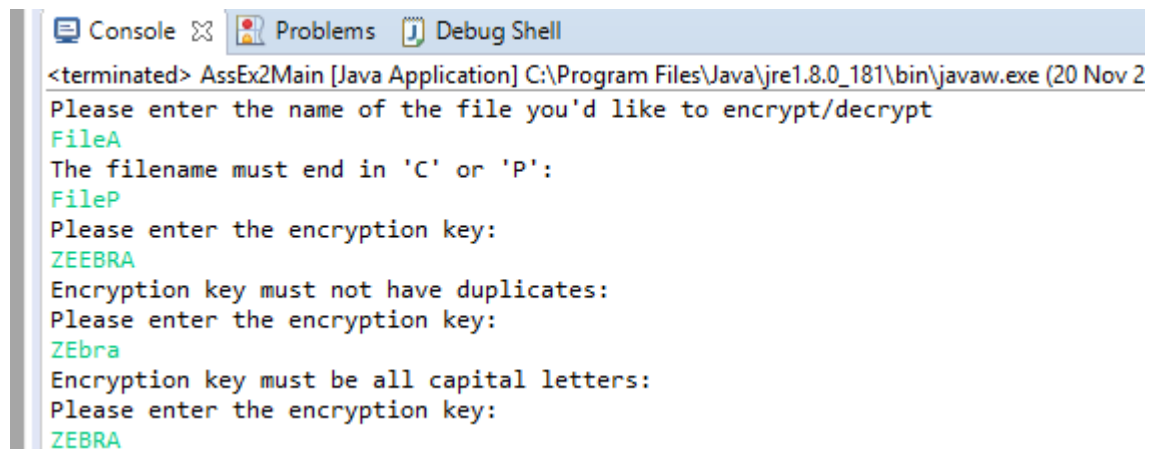
```
Console Problems Debug Shell
<terminated> AssEx2Main [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.e
Please enter the name of the file you'd like to encrypt/decrypt
FileP
Please enter the encryption key:
ZEBRA
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Z E B R A C D F G H I J K L M N O P Q S T U V W X Y
```



```
<terminated> AssEx2Main [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\java.exe
Please enter the name of the file you'd like to encrypt/decrypt
FileC
Please enter the encryption key:
ZEBRA
Z A B C D E F G H I J K L M N O P Q R S T U V W X Y
E F G H I J K L M N O P Q R S T U V W X Y Z A B C D
B C D E F G H I J K L M N O P Q R S T U V W X Y Z A
R S T U V W X Y Z A B C D E F G H I J K L M N O P Q
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

While I am confident that ciphers are both successful, the use of an alphabet array and dividing the encryption process into Strings and Chars (in order to be more confident in the results from testing and building the ciphers) led me to believe I would not be able to make use of polymorphism or an interface. Any feedback as to how those tools might have been used in this instance would be greatly appreciated.

The last image shows my program successfully identifying inappropriate encryption keys.



```
<terminated> AssEx2Main [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (20 Nov 2
Please enter the name of the file you'd like to encrypt/decrypt
FileA
The filename must end in 'C' or 'P':
FileP
Please enter the encryption key:
ZEEBRA
Encryption key must not have duplicates:
Please enter the encryption key:
ZEbra
Encryption key must be all capital letters:
Please enter the encryption key:
ZEBRA
```

I managed to use a loop in order to keep the program running in the event of a mistaken user entry. Assuming that an inappropriate key was the result of a user mistake was also what prompted me to not accept lower case letters – rather than format them to upper case versions. The program also successfully tests for an appropriate file name without terminating.