Caesar Cipher Program

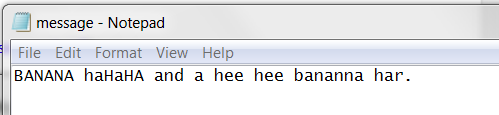
Matt Ritchie

YCP Cyber Security

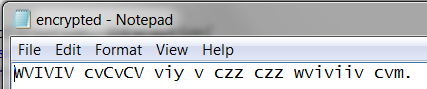
5-7-18

For this project we had to write a program with three functions, one to encode a message and two methods of decoding the message, all using a Caesar Shift cipher. My program at the end of this report.

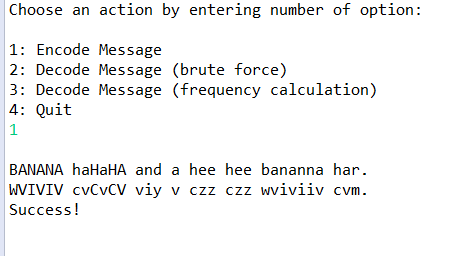
The first part is the encoding of the message, so first I made a text file containing a simple message:



My program was designed with the function to keep uppercase and lowercase letters the same. After running the Caesar shift method on it, the program creates a new text file and saves the encrypted message, giving:

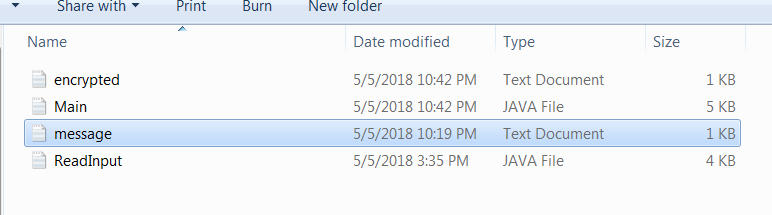


This appears within the Eclipse IDE as:



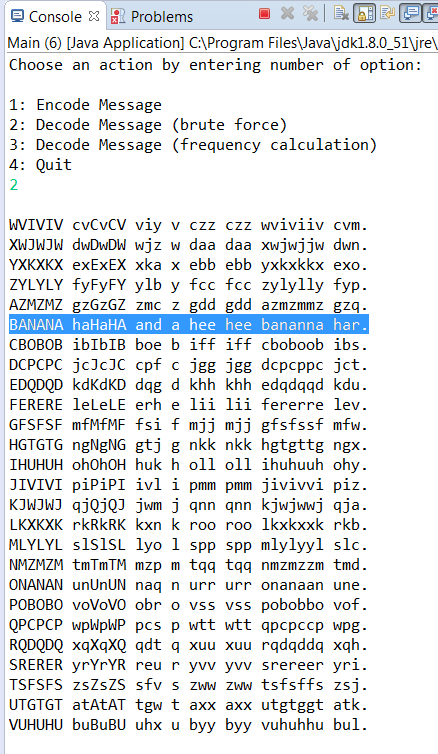
I set it to print out the message before and after altering for easier understanding, and it reports “Success!” upon properly saving to a new file.

Here’s what the program’s folder looks like afterwards:



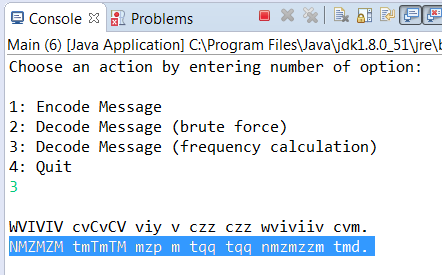
The ReadInput file contains helper methods to enforce the selection of valid menu options, otherwise it has no interference with the main program…which I called “Main” because I was tired and forgot you shouldn’t do that in good code, it should be CaesarDriver or something.

The second part of the program is a brute force attack, printing out the original, encrypted message as well as the 25 other possibilities via Caesar Shift: (I have highlighted the decoded message, showing that the program did come up with the right answer, but it doesn’t possess language abilities to detect the found answer)

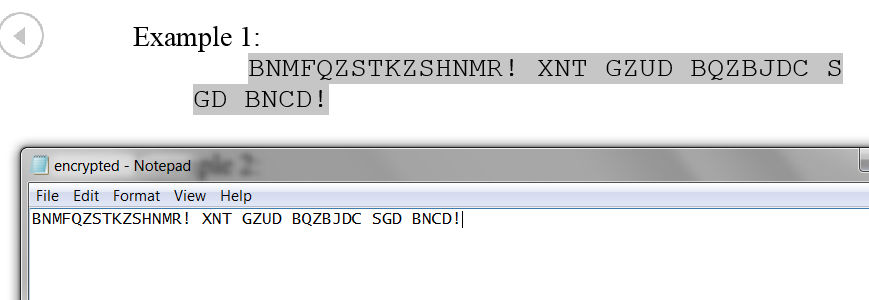


By the way, I discovered it’s a good idea to start from the original encrypted message for each function, because in testing the functions I had some muddling where the results of one method contaminated the inputs of the others, so each of the three runs the input method to draw in the message from the text file. I designed the input method to ignore the plaintext file if an encrypted message file is present.

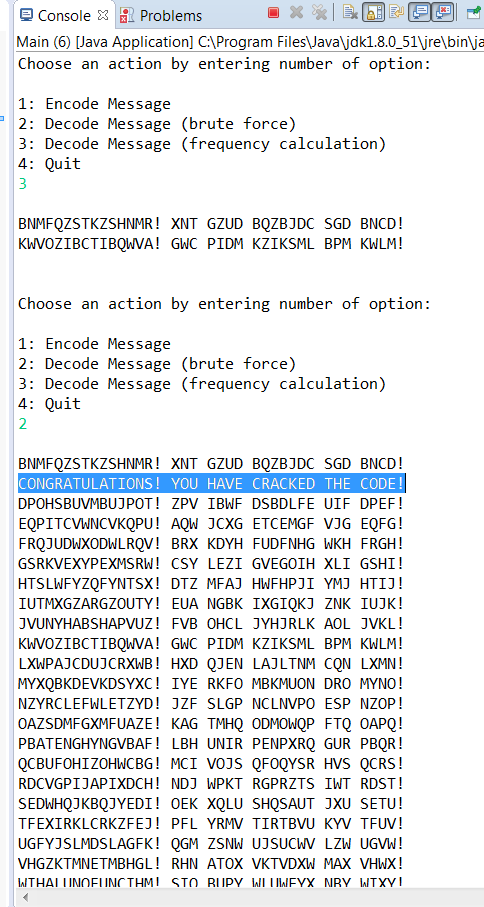
The third part of the program is an analytic attack where the most common letter in the encrypted message is declared to be “e”, the shift distance used in the encryption is calculated, and then that key is reversed and fed back into the actual shifting method to revert the message to plain text. Since I intentionally used an original message where “e” is rare, this attack fails, even though brute force works, as you can see below:



At this point I thought all was well and decided to test the provided examples:

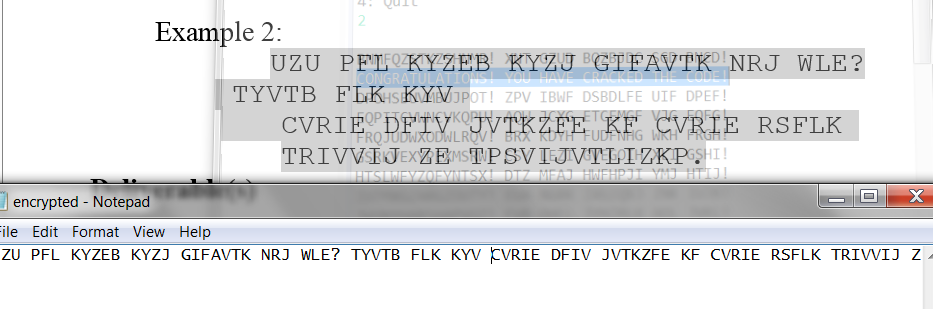


This had a little trouble with the calculated attack too, but the brute force worked:

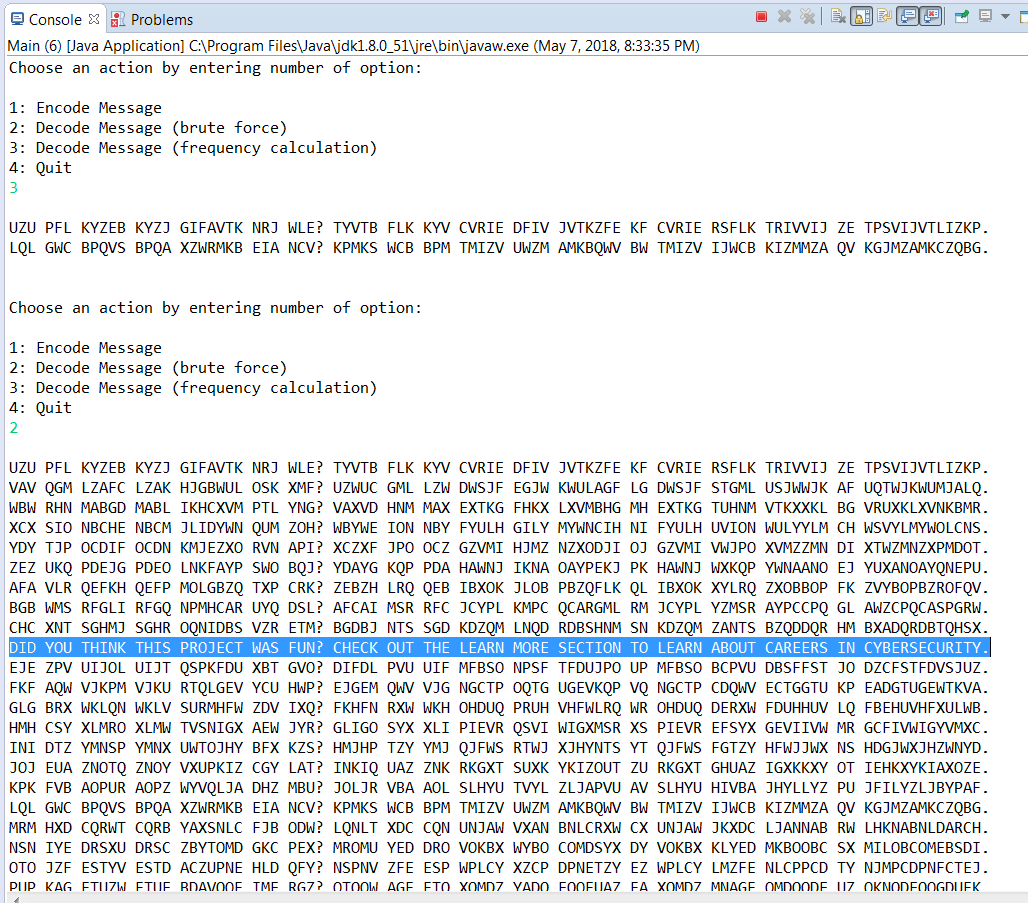


The problem is that my program has no handling for when two letters tie for most popular, and it takes the first one in alphabetical order to be the “e”, which here both “e” and “a” tied, so I blamed the calculation’s failure on that.

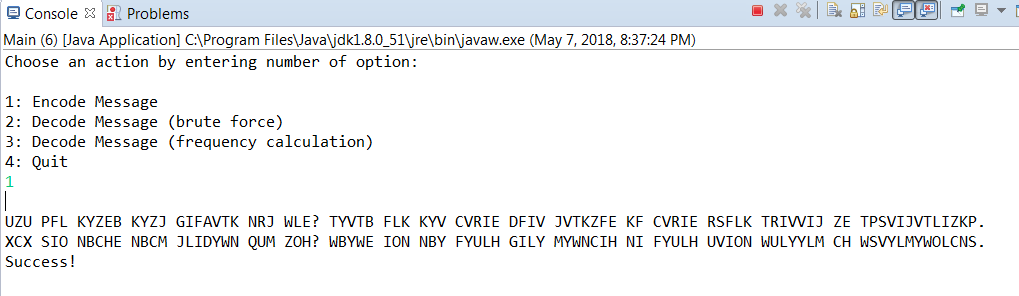
Example two ran similarly:



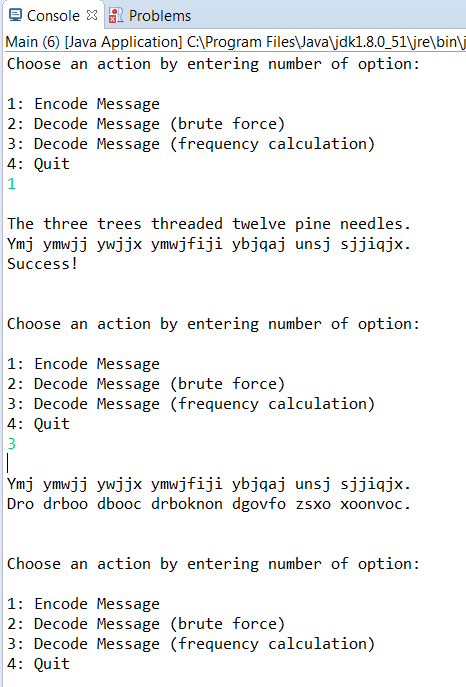
Now, my program demands the entire text be on one line. Yes, it wouldn’t be too hard to put in code to just concatenate lines together into the same string, but it has no bearing on the Caesar shift functionality, so for time I just did it by hand. This too had trouble with the calculated attack:



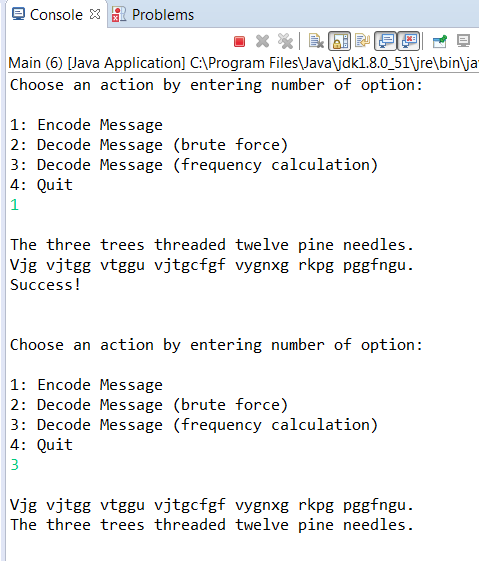
So to directly test the calculation I made a new custom message with lots of “e”s, and learned a new problem when it did this:



The weakness here is that you have to manually delete encrypted.txt for the new message.txt to be recognized. After that quick fix I ended up with:

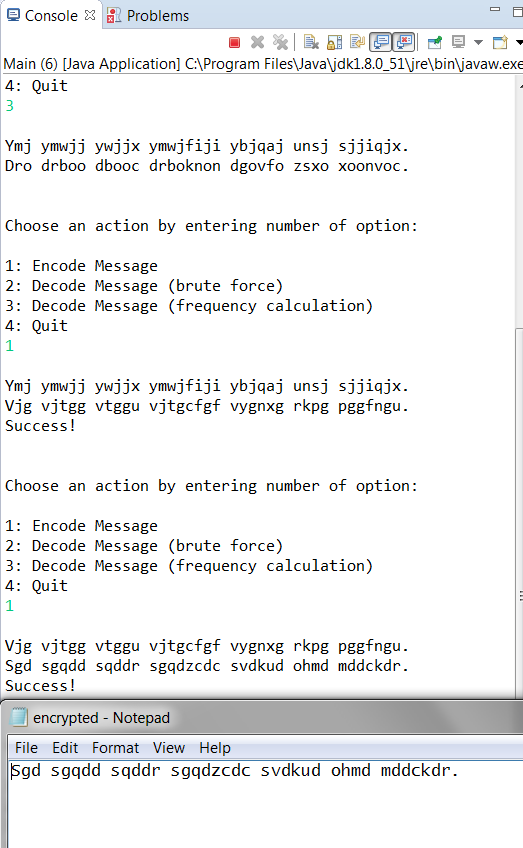


So the glaring mistake here is that the shift distance was calculated correctly, but used in the wrong direction, so I leaned to be careful of what direction the shift is going in, and added a negative, giving:

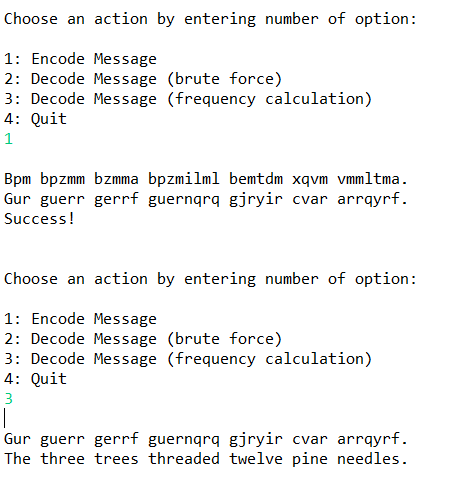


Success!

Oh by the way, some other things I learned with tinkering around; running the encryption method several times in a row shifts the previously encrypted message even further, it also keeps encrypted.txt updated with overwriting:



Additionally, I leaned that the calculated attack doesn’t give a crap about this and runs just fine:



I have to admit though, I took Cryptology class last semester, and this cipher was like chapter 1, so there wasn’t a whole lot left to learn, just a couple nuances with making a program to handle it.

**MY CODE:**

//Matt Ritchie

//CaesarBreaker

//5-7-18

//for reference, use chrome. https://docs.oracle.com/javase/7/docs/api/

**import** java.io.FileNotFoundException;

**import** java.io.IOException;

**import** java.io.\*;

**import** java.util.Random;

/\*\*

\* **@author** Matt Ritchie

\*/

**public** **class** Main {

//global vars

**private** **static** **boolean** *quit*=**false**;

**private** **static** String *text* = "failure";

**private** **static** Random *key* = **new** Random();

**public** **static** **void** main(String[] args) **throws** ClassNotFoundException, IOException {

**while**(!*quit*){

*menu*();

}//end menu loop

}//end main

**public** **static** **void** savenew() **throws** IOException{

**try** {

FileWriter writer = **new** FileWriter("src//encrypted.txt");

BufferedWriter bwriter = **new** BufferedWriter(writer);

PrintWriter pwriter = **new** PrintWriter(bwriter);

pwriter.println(*text*);

pwriter.close();

System.***out***.println("Success!");

} // end try

**catch**(FileNotFoundException | SecurityException e) {

System.***out***.println("Cannot write to file.");

} // end catch

}//end save memory

**public** **static** **void** input() **throws** ClassNotFoundException, IOException{

**try** {

FileReader fReader;

**try**{

fReader = **new** FileReader("src//encrypted.txt");

}

**catch** (FileNotFoundException e){

fReader = **new** FileReader("src//message.txt");

}

BufferedReader bReader = **new** BufferedReader(fReader);

*text* = bReader.readLine();

bReader.close();

}//end try

**catch** (FileNotFoundException e) {

System.***out***.println("File not found.");

}//end catch

}//end

/\*\*

\* Prints the menu, reads in the user's selection, and calls the corresponding method

\* **@throws** IOException

\* **@throws** ClassNotFoundException

\*/

**public** **static** **void** menu() **throws** ClassNotFoundException, IOException{

**int** temp = ReadInput.*range*(0, 1, 4, "Choose an action by entering number of option:" + "\n" + "\n"

+ "1: Encode Message" + "\n" + "2: Decode Message (brute force)" + "\n"

+ "3: Decode Message (frequency calculation)" + "\n" + "4: Quit");

//String menuNum=reader.nextLine();

**if**(temp==1){

System.***out***.println("");

*input*();

//text = "HI hi!";

System.***out***.println(*text*);

*caesarShift*(*key*.nextInt(25));

System.***out***.println(*text*);

*savenew*();

System.***out***.println("\n");

}//end encode message

**else** **if**(temp==2){

System.***out***.println("");

*input*();

String savedtext = **new** String(*text*);

System.***out***.println(savedtext);

**for**(**int** i=0; i<25; i++){

*caesarShift*(1);

System.***out***.println(*text*);

}

//System.out.println("\n" + savedtext);

System.***out***.println("\n");

}//end brute force

**else** **if**(temp==3){

*input*();

System.***out***.println("");

System.***out***.println(*text*);

*caesarShift*( -*counting*() ); //notice the inversion of the calculated encryption shift to decrypt

System.***out***.println(*text*);

System.***out***.println("\n");

}//end frequency attack

**else** **if**(temp==4){

*quit*=**true**;

System.***out***.println("Have a nice day!");

}//end quit

**else**{

System.***out***.println("Something has gone very wrong. Please email Matt Ritchie (the author) at mcritchie97@gmail.com for help.");

}//end error

}//end menu

**private** **static** **int** counting() {

**int**[] lettercounts = **new** **int**[26];

**char** letter;

**int** maxletter=0;

**for**(**int** i=0; i<lettercounts.length; i++){

lettercounts[i] = 0;

}//end initialize counts

**for**(**int** counter = 0; counter<*text*.length(); counter++){

letter = *text*.charAt(counter);

**switch** (letter){

**case** 'a': **case** 'A':

lettercounts[0]++;

**break**;

**case** 'b': **case** 'B':

lettercounts[1]++;

**break**;

**case** 'c': **case** 'C':

lettercounts[2]++;

**break**;

**case** 'd': **case** 'D':

lettercounts[3]++;

**break**;

**case** 'e': **case** 'E':

lettercounts[4]++;

**break**;

**case** 'f': **case** 'F':

lettercounts[5]++;

**break**;

**case** 'g': **case** 'G':

lettercounts[6]++;

**break**;

**case** 'h': **case** 'H':

lettercounts[7]++;

**break**;

**case** 'i': **case** 'I':

lettercounts[8]++;

**break**;

**case** 'j': **case** 'J':

lettercounts[9]++;

**break**;

**case** 'k': **case** 'K':

lettercounts[10]++;

**break**;

**case** 'l': **case** 'L':

lettercounts[11]++;

**break**;

**case** 'm': **case** 'M':

lettercounts[12]++;

**break**;

**case** 'n': **case** 'N':

lettercounts[13]++;

**break**;

**case** 'o': **case** 'O':

lettercounts[14]++;

**break**;

**case** 'p': **case** 'P':

lettercounts[15]++;

**break**;

**case** 'q': **case** 'Q':

lettercounts[16]++;

**break**;

**case** 'r': **case** 'R':

lettercounts[17]++;

**break**;

**case** 's': **case** 'S':

lettercounts[18]++;

**break**;

**case** 't': **case** 'T':

lettercounts[19]++;

**break**;

**case** 'u': **case** 'U':

lettercounts[20]++;

**break**;

**case** 'v': **case** 'V':

lettercounts[21]++;

**break**;

**case** 'w': **case** 'W':

lettercounts[22]++;

**break**;

**case** 'x': **case** 'X':

lettercounts[23]++;

**break**;

**case** 'y': **case** 'Y':

lettercounts[24]++;

**break**;

**case** 'z': **case** 'Z':

lettercounts[25]++;

**break**;

**default**:

**break**;

}//end case statement

}//end counting loop to get counts of each letter

**for**(**int** i=0; i<lettercounts.length; i++){

**if**(lettercounts[i] > maxletter){

maxletter = i;

}//end update max letter

}//end find most used letter

**return** (maxletter+1)-5;

//we need the +1 to convert from array index to alphabet index

//the ()-5 makes the final return value be the shift of where the calculated 'E' is,

// in respect to the real 'E's index is. We can then call caesarShift() on this value to

// directly jump to the (calculated) plaintext, but we still need to invert the sign of the

// shift to decrypt since this method only finds the key used for the encryption.

// \*\*Will fail if 'E' isn't the most common char\*\*

}//end keyfinding analysis

**private** **static** **void** caesarShift(**int** shift) {

String newtext = "";

**char** letter;

**int** numletter = 0;

**for**(**int** i = 0; i<*text*.length(); i++){

letter = *text*.charAt(i);

numletter = (**int**) letter;

**if**(numletter >= 65 && numletter <= 90){

letter = (**char**)((**int**) letter + shift);

numletter = (**int**) letter;

**if**(numletter < 65){

letter = (**char**)((**int**) letter + 26);

}

**else** **if**(numletter > 90){

letter = (**char**)((**int**) letter - 26);

}

newtext = newtext + letter;

}//end if upper case letter

**else** **if**(numletter >= 97 && numletter <= 122){

letter = (**char**)((**int**) letter + shift);

numletter = (**int**) letter;

**if**(numletter < 97){

letter = (**char**)((**int**) letter + 26);

}

**else** **if**(numletter > 122){

letter = (**char**)((**int**) letter - 26);

}

newtext = newtext + letter;

}//end if lower case letter

**else**{

newtext = newtext + letter;

}//end non-letter symbol

}//end shifting loop

*text* = newtext;

}//end ceaser shift

}//end class

//Matt Ritchie

//CaesarBreaker

//10-23-16

//for reference, use chrome. https://docs.oracle.com/javase/7/docs/api/

//This code is the property of Matt Ritchie, any use without permission is stealing and will be perused.

**import** java.util.Scanner;

/\*\*

\* **@author** Matt Ritchie (294021)

\* **@see** ToDoDriver

\*/

**public** **class** ReadInput {

/\*\*Instantiates a scanner to read the user's inputs.\*/

**private** **static** Scanner *reader* = **new** Scanner(System.***in***);

/\*\*

\* Gets passed instructions to print for the user, as well as the limits of the acceptable range of user inputs and loops until the user enters a valid int.

\* **@param** seed

\* **@param** start

\* **@param** end

\* **@param** info

\* **@return**

\*/

**public** **static** **int** range(**int** seed, **int** start, **int** end, String info){

**while** (seed>end || seed<start){

System.***out***.println(info);

**if** (*reader*.hasNextInt()){

seed = *reader*.nextInt();

**if** (!(seed>=start && seed<=end)){

System.***out***.println("That is not a number within the valid range.");

*reader*.nextLine();

seed=-1;

}//end range error

}//end ifGood

**else** {

System.***out***.println("That is not a valid entry.");

*reader*.nextLine();

}//end input error

}//end while

**return** seed;

}//end range

(there’s more code in the ReadInput class, but I didn’t include it since it never gets used)