Lab 9 Concepts: Cryptography (Encryption/Decryption), Program Design using *Model-View-Controller* Architecture and basic *Exception Handling* (20 pts.)

Use the notes from this week's lecture and the examples posted in Canvas (and your textbook, if purchased) to help you complete the lab

- For this lab, you may use whatever method you wish to edit, compile and run the program.

Reminder: Save all your work to your *student network drive* today (NOT the *desktop* or *local C:* drive). At the completion of lab, put all files into a single folder, zip (i.e. compress) the folder, attach it to an email and send it to yourself. All labs are a good point of reference for projects and exams. This also serves as a possible verification of completing the lab if a discrepancy arises.

Before you begin, DOWNLOAD the additional files for this lab (*Cipher.java* and *View.java*) using the "**Lab9 Data Files.zip**" link within the assignment directions in Canvas. You will be modifying these two classes in addition to creating a third (Lab9.java) that will be the *controller* portion of the program.

In this lab you will be creating an application that *encrypts/decrypts* a message using a *rotation cipher*. A rotation cipher has an integer key, such that when you *encrypt* a message, each letter of the message is shifted to the right *key*-letters in the alphabet. Similarly, when you *decrypt* an encrypted message, each letter of the encrypted message is shifted to the left *key*-letters in the alphabet. If shifting would go past the end of the alphabet, you must wrap back around to the *front* (or *end*, if decrypting) of the alphabet, as necessary. For example encrypting "*tree*" with a key of 10 yields "dboo". If this lab, **assume that the message will be all *lowercase* words** with no special characters and assume the key will be from 0-25.

Hint/Reminder: 'a' maps to 97 in ASCII and 'z' maps to 122. So, if the key was 1, 'a' + 1 = 'b'.
(-4) % 26 = -4 (-32) % 26 = -6 32 % 26 = 6

Your program will have 3 classes that meet the specification below. *Cipher* and *View* have been started for you.

Cipher (*Model* class) **No I/O statements in this class**…

- int key instance variable
- constructor initializes key
- ***encrypt*** method (takes message param and returns the encrypted result)
- ***decrypt*** method (takes message param, returns the decrypted result)

**Again, assume that all message characters are *lowercase*

View

- Scanner instance variable, constructor
- ***getMessage*** method - ask user for message, return
- ***getKey*** method - ask user for key, return.
- ***displayResult*** method - has message param, prints result

(-*cont*-)

Lab9 (*Controller* class - you need to create this from scratch)

- contains the *main* method
- creates *View* object
- gets key value from View
- creates *Cipher* object
- get message from View
- encrypt the message and display the result (call *encrypt* in *Cipher* and *displayResult* in *View*)
- decrypt the encrypted message and display the result (call *decrypt* in *Cipher* and *displayResult* in *View*)

Sample Run

```
Enter a number from 0 to 25: 13
Enter message: this is fun
Encrypted: guvf-vf-sha
Decrypted: this is fun
```

**Once complete, test your program and make sure that it both properly *encrypts* AND *decrypts* your message. If it works OK, continue on to Part 2…

Part 2: ***Required*** *(FYI – this modification should only take 5-10 minutes!)*

Add the following functionality to your application – to avoid crashing if a double or characters are entered for the *key* (and accept integers only), add a *try/catch* block within a loop to repeatedly ask for the integer key value until an integer is entered. Since we have divided this application into *Model-View-Controller*, it should be an easy fix. Just modify the method with the class controlling input. *Everything else should remain the same.* This illustrates the power of *MVC* architecture. You can modify the *view* without changing the *model*.

Sample Run

```
Enter a number from 0 to 25: 3.4
Error: Input not an integer.
Enter a number from 0 to 25: hello
Error: Input not an integer.
Enter a number from 0 to 25: 5
Enter message: i am done
Encrypted: n%fr%itsj
Decrypted: i am done
```

**Once complete, test that your modification works. If it works OK, demo for your GTA.

Extra Credit (10% / +2): Must be completed within 2-hour lab period – *Extra Credit not accepted late*.
(Partial Credit available but only if submitted before the end of lab)

Extra Credit not accepted after lab ends, but you are welcome to work on it on your own, if you wish

(FYI – this modification should only take 5-10 minutes!)

-Save the ***View.java*** class as ***View_GUI.java***

-Modify ***View _GUI.java*** to use **GUI** I/0 instead of **Console** (text-based) I/O

Here is some refresher code…

```
import javax.swing.JOptionPane;      // import for GUI interface
```

For **input**: `JOptionPane.showInputDialog("Prompt Message");` … returns a String, like **s.nextLine()**

For **output**: `JOptionPane. showMessageDialog (null, "String Msg");` … displays a String, like **println**

- In your controller class, create an **IO_GUI** object instead of an **IO** object. *Everything else should remain the same*! This illustrates the power of *MVC* architecture. You can change the *view* without changing the *model*.

After all parts are working correctly, **demo your working program for your GTA**. Then, put **ALL (3) .java files** into a folder called **Lab9**. Zip the lab folder and upload the **Lab9.zip** file to Canvas.

Don't leave until the GTA has seen your program run, checked you off the list and you have uploaded your .**java** files to Canvas. **Verify that the files have been properly submitted before leaving**! It is recommended that you attach your file(s) to an email and send it to yourself. All labs are a good point of reference for projects and exams. This also serves as a possible verification of completing the lab if a discrepancy arises.