COMP3522 Lab #08: Chain of Responsibility

Rahul Kukreja

rkukreja1@bcit.ca

BCIT CST - November 13, 2019

Welcome!

Welcome to the 8th COMP 3522 lab.

In today's lab, you will:

1. Implement the Chain of Responsibility Pattern
2. Use the DES (Data Encryption Standard) module to encrypt and decrypt data

Grading

This lab and all future labs will be marked out of 5.

Figure 1: This lab is graded out of 5

For full marks this week, you must:

1. (3 points) Implement the Chain of Responsibility Pattern correctly and have the right output
2. (2 points) Follow PEP-8 standards, best practices, OO Design principles, and write good code.
3. (1 point) **BONUS:** Draw a UML class diagram to represent your system.

💡 **IMPORTANT:** This lab is due **in-class.**

Requirements

It's time to implement a behavioral pattern. I usually find these to be a lot simpler and easier to understand. In this week's lab we will implementing the Chain of Responsibility pattern to create a series of handlers that validate and parse user input followed by either encrypting or decrypting the data according to the options specified.

First, download `crypto.py` from D2L / Learning Hub.

Secondly, install the DES (Data Encryption Standard) package. You will need to install this package via pip. If you are using a virtual environment in your PyCharm project install it using the terminal window in PyCharm. To do so enter the following:

```
pip3 install des
```

💡 This is very **IMPORTANT**. Read through the whole document before writing code. Then take a piece of paper and:
- Identify the different classes and responsibilities
(don't make this too complex now, keep it simple)
- Identify the different handlers needed
- Start building your code up from the essential classes to the ones that add in extra functionality. That is, get 2 handlers working in a chain and then add more.

User Input:

This application is meant to be run from the command line. To learn how to do this, open up terminal in MacOS/Linux or cmd in windows. Navigate to the folder with `crypto.py` and type in the following: (depending on your machine you will either need to type in `python` or `python3`)

```
python3 crypto.py -h
```

This will bring up a list of positional and optional arguments that can be provided. Many of these arguments will have a short form (e.g `-o`) and a long form (e.g. `--output`). Either one can be used.

To correctly encrypt or decrypt data you will need:

- data to encrypt or decrypt
- the encryption or decryption key. This is a relatively short string that is used by the encryption or decryption algorithm to encode the data.

Here are some examples of possible ways to call the program. Some of the handlers in your program might want to validate the request.

- `python3 crypto abcd1234 -s "Test Data to be encrypted"`
  This provides a string to be encrypted with the key abcd1234
- `python3 crypto abcd1234 -f "inputFile.txt"`
  This provides a text file with the required data that needs to be encrypted with key abcd1234
- `python3 crypto -f "inputFile.txt" abcd1234`
  This is the exact same command as the previous one
- `python3 crypto abcd1234 --file "inputFile.txt"`
  This is also the exact same command as the previous one. This uses the long form instead
- `python3 crypto abcd1234 -f "inputFile.txt" -m en`
  This is also the exact same command, but it explicitly sets the mode to encryption
- `python3 crypto abcd1234 -f "inputFile.txt" -m de`
  This provides a text file with data that needs to be decrypted with the given

key.

- `python3 crypto abcd1234 -f "inputFile.txt" -m de --output print`
  This is the exact same command as the previous one explicitly stating that the output should be printed to the console.
- `python3 crypto abcd1234 -s "Some Test Data that is encrypted" -m de --output "outputFile.txt"`
  This provides a input string that needs to be decrypted with the key abcd1234. The output is saved to a file instead of being printed to the console

Open the code and take a look at it. Right now it just prints the request. We use a module called `argparse` to accept input via the command line. This is a very useful tool to know. As a developer you may be asked to write command line tools for other developers. For this lab, you do not need to modify this. I have already set up argparse for you.

Encryption and Decryption

Right, lets get to it.

We will be using the `des` package to encrypt and decrypt a string. DES stands for Data Encryption Standard. It takes a `byte string` as the key and the data to be encrypted/decrypted.

A byte string here just means that the string is a sequence of bytes. This can easily be achieved by adding the letter `b` as the prefix to a string

```
byte_string = b"Some string"
```

To convert this back to a string we can use the built in encode method found in the `str` library.

```
original_string =  byte_string.encode('utf-8')
```

Now that you know what byte strings are, go check out a quick tutorial on how to encrypt or decrypt data using DES. You can find this at: https://pypi.org/project/des/

Ok, so what do I have to do exactly?

Implement the `Chain of Responsibility` Pattern to create handlers that would process the input request. In `crypto.py`

I have included the code to take the command line arguments and assign them to an object of type `Request`. You will need to:

- Identify all the handlers needed to validate and process the request
- Implement these according to the `Chain of Responsibility` Pattern.

An Enum called `CryptoMode` defines 2 modes that the application can run in. An `EN` (Encryption) mode and a `DE` (Decryption Mode).

I have also included the stub for the class `Crypto` which has the following stub methods:

- `__init__(self)`
  This initialization method should set up the two chains of handlers. One for encryption and the other for decryption. It should store the reference to the first handler of each chain in `self.encryption_start_handler` and `self.decryption_start_handler` respectively.
- `execute_request(self, request: Request)`
  This method accepts a request and starts executing the first handler in the appropriate chain.