Week 03: Let's talk with other programs ?

Challenge 01: YourPrime – Tokenizer for Media (30 mins)

Alright, let's start populating our media library. Of course, we are not going to do this manually. Since our streaming platform might not be generating much income (and we might be operating by quarterly investors cash injections), we must fully optimise our programming skill. Fortunately, you have covered input and output streams topic (as an addition to Java 1), so you should be more than capable of completing this task.

Using the comma-separated-value files from Kaggle, we have 45,000 popular movies (from the MovieLens – a research group from the University of Minnesota). The movies are rated, and the dataset are currently used to develop new experimental tools and interfaces for data exploration and recommendations. Your task is simple, we need to create a parser to capture the movie title, genre and rating from the provided dataset.

TODO#1 – You need to inspect the dataset to familiarise yourself with the data format.

TODO#2 – The data entries are not consistent (this should be your observation from step 1). Therefore, we probably need to parse the file multiple rounds.

TODO#3 – We will use Scanner to split our dataset in the first instance.

TODO#4 – Complete the **`argsParser()`** method to capture the movie title, rating and genre**.

** The current rating is based on the scale of 10, therefore you need to convert the value to based 5, implemented by `YourPrime`. Our movie genre type is limited to `Action`, `Comedy` and `Drama`, therefore, we need a utility function to capture only these values (Check out TODO#5).

TODO#5 – Complete the utility function to capture the movie genre. Assign the other genres as null.

We will stop here – we will look at the instantiation of Movie object using the Map data structure later.

Challenge 02: YourPrime – More money, more problem (15 mins)

The next item in our agenda is to sort out the communication between our subscribers (client) and streaming server. Obviously, we need an enterprise server setup to handle this operation, but we haven't yet covered those topics, so we will have to do it the old fashioned way. We have to write our own socket programming code to facilitate the exchange between the clients and server.

We will implement a simple client – server program. On the client side, you will send the username and password to the server. On the server-side, you will need to authenticate the client credentials and then check for outstanding fees.

TODO#1 – Complete the **`go()`** method in `yourPrimeClient`. You need to provide the username and password to the server.

TODO#2 – In `yourPrimeServer`, complete the **`go()`** method as the main method to send and receive socket.

TODO#3 – Complete the remaining utility methods – **`authenticate()`** as the method to process user access. Use the two Boolean validity methods to assist the `authenticate()` method.

That looks easy – but it does not look secure. We are charging an arm and a leg for this service, so, this kind of security will not fly with our subscribers. Well, don't worry, our next challenge will handle this problem.

Challenge 03: Hmm, this is a bit dodgy (50 mins)

Obviously, we need to provide "some" form of security to our subscribers, so we can continue charging them an arm and a leg. What we need is encryption (remember, you've covered this topic during Year 1 – Intro to cybersecurity). But hang on a minute, we didn't cover it in Java, how do we know how to encrypt our data.

Well, consider this as a bonus material 😊 To be honest, you don't need to implement the security protocol manually. In part 2 of the module, you'll see how this is handle expertly by various enterprise servers. In fact, most of the complexity are hidden from you, with various API supports available to simplify the programming. You are encouraged to check out the Java Cryptography Architecture Reference Guide to learn more about Java security. You should browse the sample programs provided in the reference guide to give you some ideas of the programming idioms normally applied in Java for encryption (e.g., credential authentication between multiple parties). Checkout this link for further guidance on core java security. That should be sufficient for now. Let's get into coding:

TODO#1 – On the client side, you need to complete the `getMessage()` method to encrypt the message that you are sending to the server.

TODO#2 – On the server side, you need to decipher the message you received from the client, and process the data using the `authentication()` method we have defined earlier.

As another bonus, we will try to transmit an encrypted file to the server. Well, this is NOT how we normally do this – you should make use of the SSL protocol when you what to transfer files across a network, however, we are going to do it here for academic reason. In order to protect our data, we are going to encrypt it with the `cipherOutputStream` that we discussed during class.

TODO #3 – On the client side, complete the `writeFile()` method to encrypt a message into a file using `cipherOutputStream`.

TODO#4 – Next, you need to transmit the file in bytes to the server.

TODO#5 – Since the server will be handling multiple requests (it is very busy with subscribers 😊) we need to create a separate thread to handle the transmission of this encrypted file.

TODO#6 – Complete the `readWriteFile()` method that will collect the encrypted data, and then write it to a file on the server side.

TODO#7 – Complete the `decipherFile()` to reveal the hidden message we received from the client side. You should use the `cipherInputStream`, and revere the encryption process that you perform on the client side.