Strategy pattern and Visitor pattern design

Currently, Prattle Application does not follow Single Responsibility Principle very well. Based on the discussion with Professor and the advice he gave. The main concern is how to separate data and operation to make maintenance, coding and extending easier. I will follow the advice that using Strategy Pattern and Visitor Pattern. It is good to talk them with some more details.

In **ClientRunnable**, there are mainly two large things to do, the first thing is *handleIncomingMessages* and the second thing is *handleOutgoingMessages*. However, though it looks a little misleading, the same **ClientRunnable** has two different roles, Sender or Receiver. Some of the configuration is decided by sender and some of the configuration is determined by receiver, and it is not true that the sender will know every receiver's configuration. It is possible to just enlarge these two functions with more and more things, let the message take lots of operation and status to reach our goal. Nevertheless, it is definitely not the good practice. Instead, people prefer simpler methods, smaller class with single responsibility that are easy to change and extend.

As a sender, the use cases that I come up with are encryption and expiration. There can be many different behaviors which will enlarge message class so fast. For example, when will the message expire? Maybe sender will choose some deadline, maybe sender will want message "burned" after someone has read it, and maybe sender just wants to retract the message. The strategy pattern just fits the situation. I define **IMessageStrategy** interface which currently contains two method:

String getOutputText(String raw);
boolean isExpired(Message msg);

For a **FlexibleMessage**, a sub class of **Message**, it will act as Context. It has a private field for strategy. Then the task of getting text and checking expiration status delegates to strategy. It is divided from original **Message** class and it is really good.

As a receiver, the use cases now I can come up with are filtering and following (we want to specially mark the message from whom we are following, right?). After spending time understanding the code, I will say surprisingly but reasonably, functionality of receiving message via socket is in **NetworkConnection.** It is also a large class and we will hope to divide some work into another class.

I will prefer to extend **SocketChannel** to have a receiver that implements the visitor pattern. I will also move the job in **NetworkConnection** that send message via socket channel to receiver class. We will have different behaviors about how client receives message (different filter configuration for example). These works should also move outside **Message** class. So, receiver will have *void visit(Message message)* method while there is a interface **IMessageOut** to allow message accept it.

void accept(IReceiver receiver);

Finally, FlexibleMessage extends Message and implements IMessageOut and the patterns are ready to use.

In conclusion, when we want to add new feature that relates with sender, we will think about writing a new Strategy class to handle it and message will construct with strategy (maybe builder pattern can be used here if there are more things). When we want to add new feature that relates with receiver, we will think about writing different visitor without changing message's structure.