Developper Manual

by Pierre-Jean Besnard & Louis Billaut

Introduction

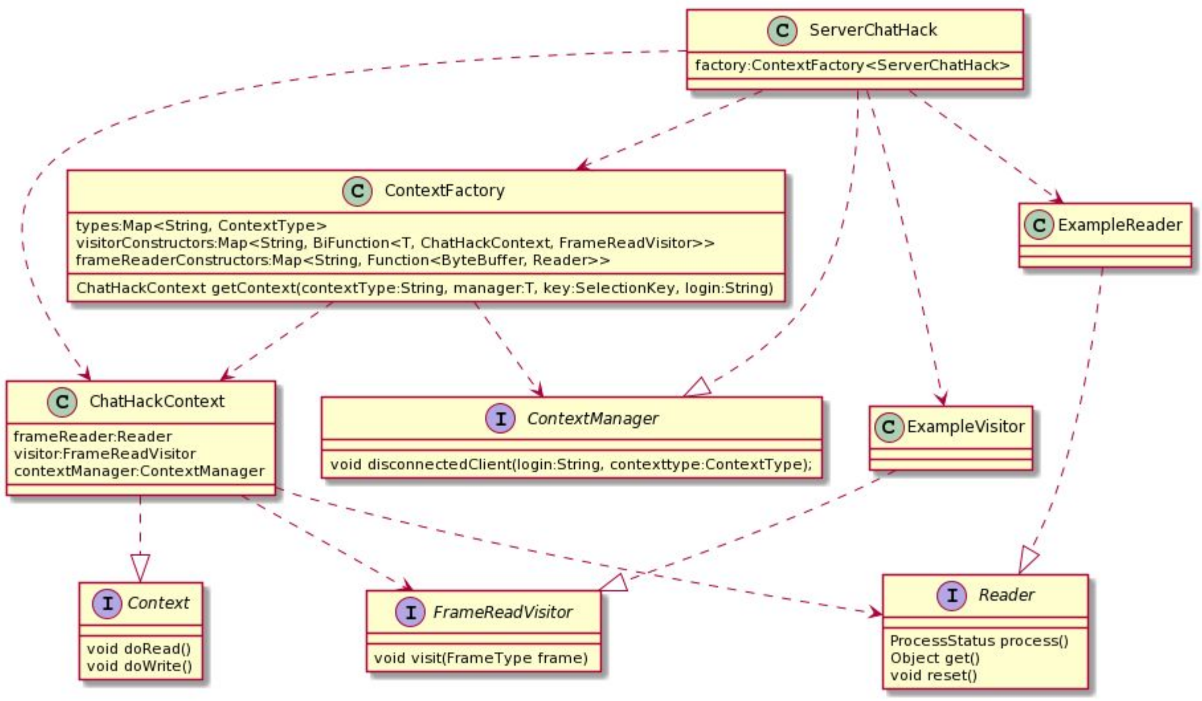
This manual could be useful to some developper which would wants to implement some other features in the ChatHack project. It contains a quick description of our structural choices to develop this and what features it currently implements.
This project has been developed during our first year of Computer Science Master’s degree and has been originally divided in 2 parts by our teacher, so the third section of this manual is about what we have implemented between the first and the second part.

How does it works ?

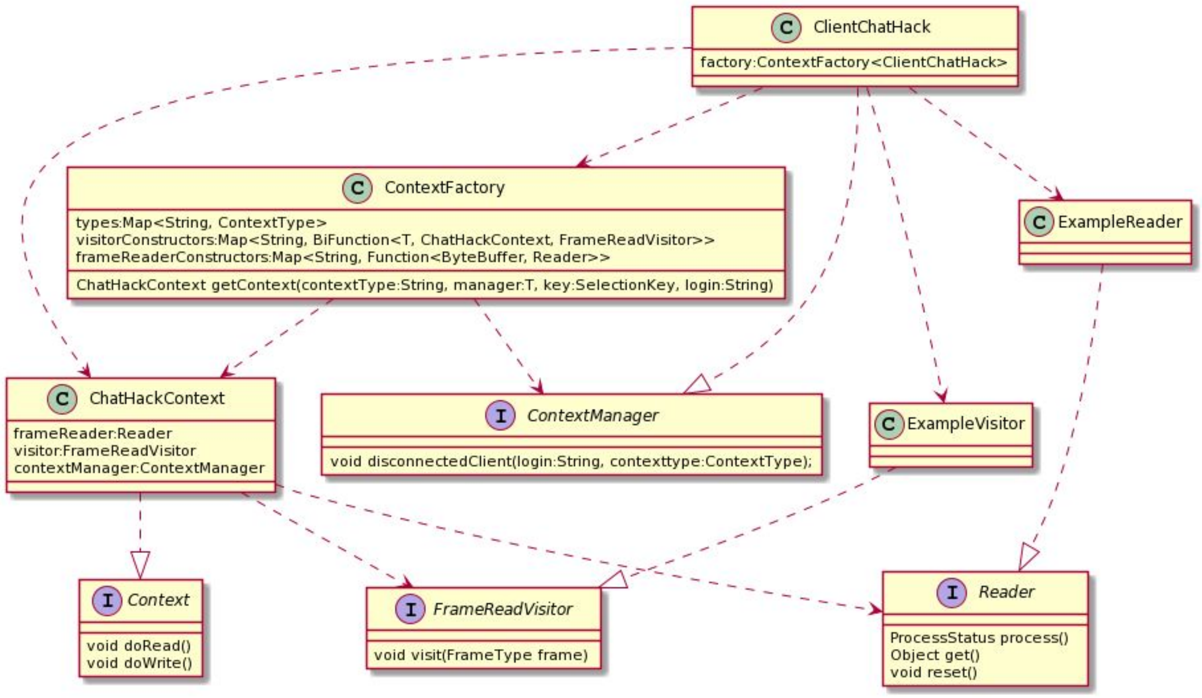
This schemas provides simplified views to allow to understand how does differents elements works with each other in the server and in the client.

Please notice that the “ExampleClass” are not implemented in the server and in the client, that is just to represent the way that you can replace this classes by some other class that implements the same interfaces.

Server structure


ServerChatHack
factory:ContextFactory<ServerChatHack>
ContextFactory
types:Map<String, ContextType>
visitorConstructors:Map<String, BiFunction<T, ChatHackContext, FrameReadVisitor>>
frameReaderConstructors:Map<String, Function<ByteBuffer, Reader>>
ChatHackContext getContext(contextType:String, manager:T, key:SelectionKey, login:String)
ExampleReader
ChatHackContext
frameReader:Reader
visitor:FrameReadVisitor
contextManager:ContextManager
ContextManager
void disconnectedClient(login:String, contexttype:ContextType);
ExampleVisitor
Context
void doRead()
void doWrite()
FrameReadVisitor
void visit(FrameType frame)
Reader
ProcessStatus process()
Object get()
void reset()

Client structure


ClientChatHack
factory:ContextFactory<ClientChatHack>
ContextFactory
types:Map<String, ContextType>
visitorConstructors:Map<String, BiFunction<T, ChatHackContext, FrameReadVisitor>>
frameReaderConstructors:Map<String, Function<ByteBuffer, Reader>>
ChatHackContext getContext(contextType:String, manager:T, key:SelectionKey, login:String)
ExampleReader
ChatHackContext
frameReader:Reader
visitor:FrameReadVisitor
contextManager:ContextManager
ContextManager
void disconnectedClient(login:String, contexttype:ContextType);
ExampleVisitor
Context
void doRead()
void doWrite()
FrameReadVisitor
void visit(FrameType frame)
Reader
ProcessStatus process()
Object get()
void reset()

As you can see, client a server perform in the same way, that is the result of what we have focused on during the development of the project ChatHack.

The ChatHackContext constructor has been put in private visibility to avoid bad constructions.

Final product

Features available

There is a quick list of what possibilities the final program offers.

Server side :

- Launch : Once the server has been launched, it connects itself to the database if it is possible and creates an object with which censure messages sent in the global chat. The censorship is based on the file that has been specified in command line or the default file otherwise. (Please notice that the server will not launch if the database is offline but he will continue to run even if the database has been interrupted, only new connections will be rejected.)

- Client connexion : When a client tries to connect to the server, it has to send a connection frame first. There is two types of server connection, connections with password and without password.
  - Connection with password : If a client tries to connect itself to the server with a login and a password, the server sends a request to the database to verify if the login and the password are corresponding to an registered account.
  - Connection without password : If a client tries to connect itself to the server with a login only, the server sends a request to the database to verify if the login is not corresponding to an registered account.

  During this time, the client is no longer allowed to send frames to the server. Once the server has received the database response, it sends a positive or negative response to the client and update his status in his own data structure.

- Global chat : Once a client has connected itself to the server, it can send messages in global chat.
  - Censorship : When a client sends a message to the global chat, the message could be censored by the server before being broadcasted to other clients connected.
  - Private connection request : Each connected clients can sends private chat requests by the server, the server will transfer the request to the receiver after it verifies if he is connected.

- ○ Private connection approval / deny : When a client A sends a private connection approval or deny frame to a client B, the server verifies if client B has sent a request before transfering the response to client B.

Client side :

- Launch : Once the client has been launched, it connects itself to the server if it is possible. The client will send automatically a connection frame with the informations that has been entered by command line (The frame will be adapted depending on if the user entered a login with a password or just a login. Once the server response received, the client indicates by the console if the connection has been established or not.

- Global chat : Once the client has been identified by the server, he can send global messages to the server.

- Private connection : The client can also : send private connection requests, refuse and accept private connections. (Please refer to “User Manual” to know how to use it).

- Private messages : Once a private connection established, the client is directly connected to the receiver, as an host if he received the connection request, or as a client if he sent the request. The client can then send private messages, files and can interrupt the private connection by some commands entered in the console. The client will be notified if another client has ended his private connection with him or if the server has been interrupted.

Difficulties encountered

No major difficulties have been encountered during the development of this project. We tried to focus on what structural patterns to use and the good organisation of the code that is why it took us more time than expected.

Possible evolutions

As announced it the section “Difficulties encountered” of this section, we tried to make the code as “object” as possible, firstly because that was a good training to us, to show what we can do after have been trained to use some design patterns, and secondly because we wanted to make this project implementable, to allow us or other developper to do add some features and other things easily to this project.
We can imagine this project with a more complex database that save a kind of friend list or some other things like personal informations about the client. We thought about an emoji system implemented in the server to convert some specific strings into ASCII emojis. We reflected to create an angular version to use these programs with a web browser.
There are many possibilities to make this project more complete.

Evolution provided

Requested changes

During the development of the first part of this project, we focused on main functionalities and all required things by our teacher. Consequently we worked on the structure of the code between the first and the final part of this project.
We made this project as “object” as possible and this list is about what we have refactored :

Global :

- Context : The client and the server use “Context” objects to symbolise the link between us and a client or a server. We made it as generic as possible and we developed a unique class which will be useful with any connection, in the server and in the client.
  The unique context takes “frame reader” and “frame visitor” at his construction which allows to read and threat frames received as desired without modify the context class.

- Context factory : We also made a context factory which is configurable with some Maps to avoid bad constructions of Context object. This factory can be used by a client or by a server, it is configurable by his wild card type.

- Some JUnit tests about differents frame reader has been provided.

Server side

- Client connection : A different context with a different visitor has been implemented, it avoid the server do several verifications when a non-identified client tries to send other frames than connection frames. During the time that the server wait the database response, the server cancel the key associated to the client and it registers a new one when the response is finally received.

- Other : The main class ServerChatHack has been refactored, the queue are now composed by ByteBuffer. Each client identified and his requests are stored in Maps which is faster to transfer and check private requests from a client to another.

Client side

- Console : We implemented a message parser to manage messages entered by the user in the console more properly. Each command entered is represented by an object which will be visited by a console visitor to treat it.

- Other : We focused on the way that the program treats exceptions. We fixed some multithreading problems between the thread which treats commands entered by the user and the thread which is in charge to manage the server connection and other private connections. On the same way than the

ServerChatHack, the modifications which concern queue and informations about clients have been implemented in the ClientChatHack.

Optional changes

Server side

- Censorship : Users can launch the server with a censorship file by command line. The server will automatically replace each “bad word” by some *** characters. This censorship only concerns global chatting and if you want insult or threaten your mate, go invite him in a private chat !

Client side

- Console : We made a new console command available with which interrupt a private connection with another client. This command can be used by the character “#”.