Colissa Pollard December 16th, 2018

This final project is a simple chat program where users can communicate to one another through a network on different chat "channels". Users publish messages to a channel and the subscribers then receive these messages. Users are able to subscribe and unsubscribe from specific channels. The Admin extends the User class and is given additional capabilities to add and remove any user from a channel. The User class has a private ArrayList of channel objects that "this" User is subscribed to.

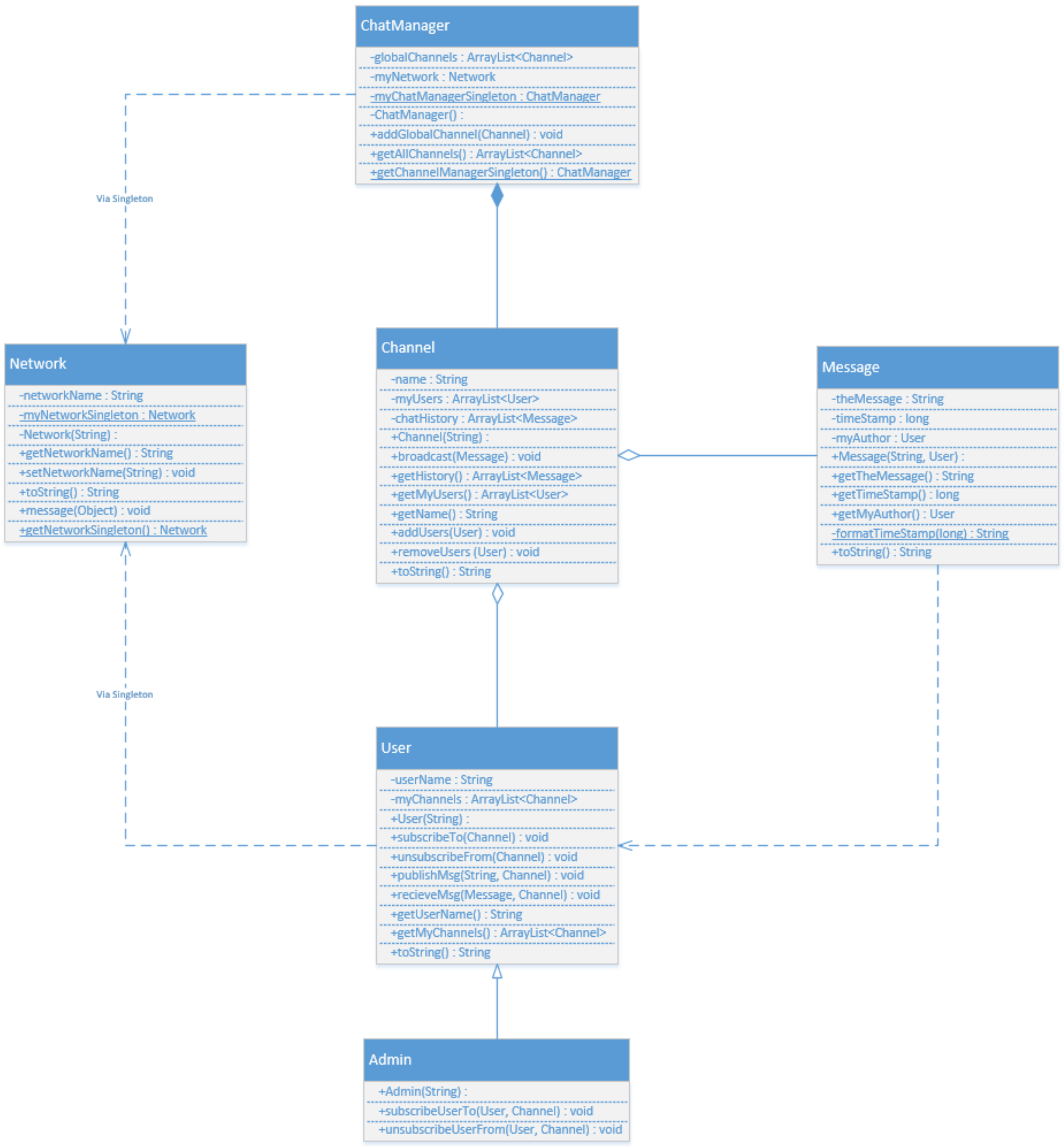
The ChatManager class is a singleton object that manages the "global" list of channels. It has a private static singleton, a private ArrayList of Channels, and a private Network. I have used the Singleton design pattern to ensure that only one ChatManager object is used to manage the global list of channels. This is accomplished by having a private constructor and a public getter method which checks to see if the ChatManager has already been instantiated before returning the singleton instance.

The Network class is a simple "fake println" network, which simulates routing chat Messages through a network. To simplify the simulation, the activity is merely printed onto the screen. The Network class also uses the Singleton design pattern to ensure one Network is used.

The Channel class utilizes the Publisher-Subscriber design - the subscribers of each Channel will receive messages that are published by a User. The Channel class has a name, an ArrayList of Messages (which represents its chat history) and an ArrayList of subscribed Users. This allows the publisher to send their message without knowing who the subscribers are. Also, the subscribers don't need to worry about who the publishers are since they only care about the topic of the channel.

The Message class has a timestamp, author, and the message content. The timestamp is obtained from the system at the time of message creation and is formatted by using the SimpleDateFormat utility class. The Message toString() method takes care of automatically displaying the timestamp, author, and message content whenever a message is sent to the Network.

Updated UML diagram:


ChatManager
-globalChannels : ArrayList<Channel>
-myNetwork : Network
-myChatManagerSingleton : ChatManager
-ChatManager() :
+addGlobalChannel(Channel) : void
+getAllChannels() : ArrayList<Channel>
+getChannelManagerSingleton() : ChatManager
Via Singleton
Network
-networkName : String
-myNetworkSingleton : Network
-Network(String) :
+getNetworkName() : String
+setNetworkName(String) : void
+toString() : String
+message(Object) : void
+getNetworkSingleton() : Network
Channel
-name : String
-myUsers : ArrayList<User>
-chatHistory : ArrayList<Message>
+Channel(String) :
+broadcast(Message) : void
+getHistory() : ArrayList<Message>
+getMyUsers() : ArrayList<User>
+getName() : String
+addUsers(User) : void
+removeUsers (User) : void
+toString() : String
Message
-theMessage : String
-timeStamp : long
-myAuthor : User
+Message(String, User) :
+getTheMessage() : String
+getTimeStamp() : long
+getMyAuthor() : User
-formatTimeStamp(long) : String
+toString() : String
Via Singleton
User
-userName : String
-myChannels : ArrayList<Channel>
+User(String) :
+subscribeTo(Channel) : void
+unsubscribeFrom(Channel) : void
+publishMsg(String, Channel) : void
+recieveMsg(Message, Channel) : void
+getUserName() : String
+getMyChannels() : ArrayList<Channel>
+toString() : String
Admin
+Admin(String) :
+subscribeUserTo(User, Channel) : void
+unsubscribeUserFrom(User, Channel) : void