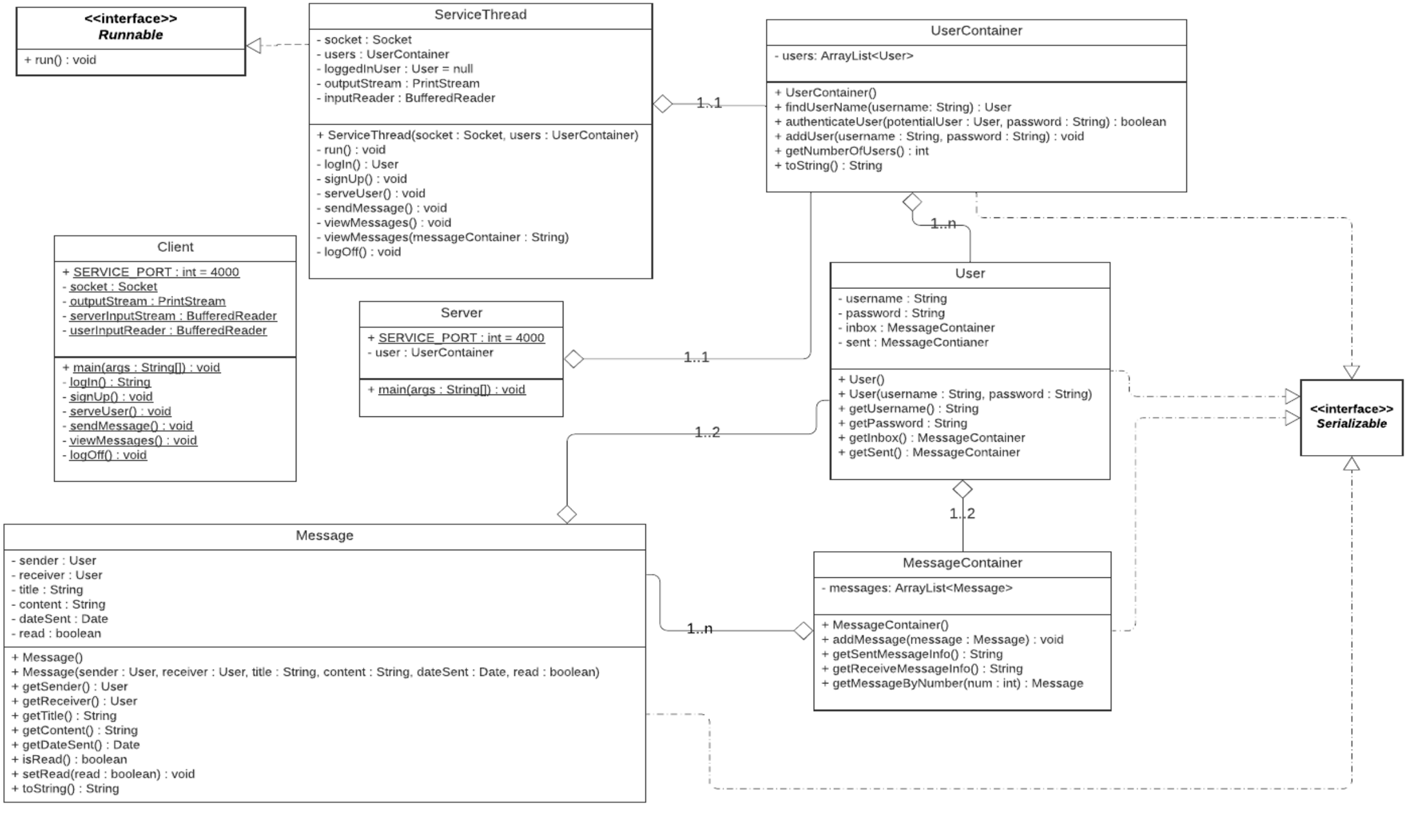
Project Report

Name : Ahmad Abu Salem

Student ID : 0186425


<<interface>>
Runnable
+ run() : void
ServiceThread
- socket : Socket
- users : UserContainer
- loggedInUser : User = null
- outputStream : PrintStream
- inputReader : BufferedReader
+ ServiceThread(socket : Socket, users : UserContainer)
- run() : void
- logIn() : User
- signUp() : void
- serveUser() : void
- sendMessage() : void
- viewMessages() : void
- viewMessages(messageContainer : String)
- logOff() : void
1..1
UserContainer
- users: ArrayList<User>
+ UserContainer()
+ findUserName(username: String) : User
+ authenticateUser(potentialUser : User, password : String) : boolean
+ addUser(username : String, password : String) : void
+ getNumberOfUsers() : int
+ toString() : String
1..n
Client
+ SERVICE_PORT : int = 4000
- socket : Socket
- outputStream : PrintStream
- serverInputStream : BufferedReader
- userInputReader : BufferedReader
+ main(args : String[]) : void
- logIn() : String
- signUp() : void
- serveUser() : void
- sendMessage() : void
- viewMessages() : void
- logOff() : void
Server
+ SERVICE_PORT : int = 4000
- user : UserContainer
+ main(args : String[]) : void
1..1
User
- username : String
- password : String
- inbox : MessageContainer
- sent : MessageContainer
+ User()
+ User(username : String, password : String)
+ getUsername() : String
+ getPassword : String
+ getInbox() : MessageContainer
+ getSent() : MessageContainer
<<interface>>
Serializable
1..2
1..2
Message
- sender : User
- receiver : User
- title : String
- content : String
- dateSent : Date
- read : boolean
+ Message()
+ Message(sender : User, receiver : User, title : String, content : String, dateSent : Date, read : boolean)
+ getSender() : User
+ getReceiver() : User
+ getTitle() : String
+ getContent() : String
+ getDateSent() : Date
+ isRead() : boolean
+ setRead(read : boolean) : void
+ toString() : String
1..n
MessageContainer
- messages: ArrayList<Message>
+ MessageContainer()
+ addMessage(message : Message) : void
+ getSentMessageInfo() : String
+ getReceiveMessageInfo() : String
+ getMessageByNumber(num : int) : Message

1. Message Class :

- The class has a no-arg constructor which is only available as a requirement for serialization, it also has another constructor which takes all values to be assigned to the data fields and assigns them, respectively.
- The class has public getters for every data field, and a setter for the *read* data field.
- The class overrides the *toString* method of the *Object* class and returns a string containing the values for every data field of the message except for *read*, each on a new line with a fitting label for each field.

2. MessageContainer Class :

- This class serves as an abstraction for a collection of messages, with useful methods to operate on all messages.
- The no-arg constructor initializes the *messages* ArrayList to an empty list.
- The *addMessage* method takes a *Message* object as a parameter and adds it to the *messages* list.
- The *getSentMessageInfo* and *getReceiveMessageInfo* methods operate similarly in that they return a string containing meta-information (i.e., without the message content) about every message in the messages list; each message info on a new line with fitting labels and distinguishing between every data field with a slash (/). The difference between the two methods is that the former includes the name of the recipient and not the sender, while the second one includes the name of the sender and not the recipient.
- The *getMessageByNumber* method takes an integer *num* as input, and returns the num^{th} number, which has an index of (*num* - 1) because of zero-indexing.

3. User Class :

- The class has a no-arg constructor which is only available as a requirement for serialization, it also has another constructor which takes a username and a password and assigns them to the respective data fields and initializes the *inbox* and *sent* message containers using the no-arg constructor of the *MessageContainer* class.
- The class has public getters for every data field.

4. UserContainer Class :

- This class serves as an abstraction for a collection of users, with useful methods to operate on all users.
- The no-arg constructor initializes the *users* ArrayList to an empty list.
- The *findUserName* method takes a username string as input, and searches the *users* ArrayList for a user with a matching name. It returns the user if found and null if not found.

- The *authenticateUser* method takes a *User* object and a password string as input, and returns true if the passed user is authentic and false otherwise. The user is valid if his password stored in the *users* list is the same as the password parameter.
- The *addUser* method takes a username and a password as parameters, it creates a *User* object using the fitting constructor of the User class, then adds the user to the *users* list.
- The *getNumberOfUsers* method returns the number of entries in the *users* list.
- The class overrides the *toString* method of the *Object* class and returns a string containing the usernames of all the entries in the *users* list, each name on a new line.

5. Server Class :

- The *main* method of this class loads the stored *users* user container from the local system if one is available, else it initializes the container using the no-arg constructor of the *UserConstructor* class. Then it binds to port 4000 to infinitely listen for connection requests; when a new connection arrives, the server accepts it and creates a new *ServiceThread* passing the corresponding socket and the same *users* data field for all threads. It then starts the thread and returns to listening for other connection requests.

6. ServiceThread Class :

- The class has a constructor which takes a socket object and a *UserContainer* object, it assigns them to the corresponding data fields. It also initializes the input/output streams for communicating with the client.
- The class implements the *Runnable* interface to gain thread functionality. The implemented method *run* first starts with handling the log-in/sign-up menu of the application. It reads the choice from the client side and then calls either *logIn* or *signUp* methods to implement the functionality. If log-in is successful the *loggedInUser* data field is updated with the *User* object. If sign-up is successful then log-in must be subsequently chosen to enter the system correctly. If either choices is not successful, the method keeps reading new choices until successful log-in is achieved. Once logged in, the method calls the *serveUser* method, after that method returns the run methods ends the thread terminates.
- If any of the methods make any changes to the *users* object or any object inside it, the updated *users* object is immediately written to a file on the local system to stay updated even if an abrupt termination of the connection happens.
- The *logIn* method reads the username and password attempt from the client then calls *findUserName* and *authenticateUser* of the *users* object to check if the attempt is valid. If it is found to be valid, the corresponding *User* object is returned and a success code is written to the client, otherwise, null is returned to the calling method and different error codes are written to the client based on the cause of log-in failure.
- The *signUp* method reads the potential new username and password from the client then calls the *findUserName* of the *users* object to check if the username already exists; if it does exist, the method

writes an error code to the client and returns without doing anything. if the username does not exist, the method writes a success code to the client then creates a *User* object with the read credentials and adds it to the *users* container object.

- The *serveUser* method represents serving the main menu of the application. The method repeatedly reads a choice from the client and either calls *sendMessage*, *viewMessage* or *logOff* based on the choice. If *logOff* is called and finished, this method also returns.
- The *sendMessage* method sends to the client the string returned by the *toString* method of the *users* object, excluding the name of the *loggedInUser*, then reads the name of the user to send the message to from the client. It writes a success code to the client if the name exists and is not the same as the username of the *loggedInUser*, otherwise it writes an appropriate error code and goes back to reading another name. When the read name is valid, the methods reads the title and content of the message then creates a new *Message* object with the entered info and the appropriate default info for other data fields in the Message class. The message is then added to the *inbox* of the recipient and the *sent* of the sender.
- The no-arg *viewMessages* method reads a choice of either viewing the *inbox* or *sent* of the *loggedInUser*. Based on the choice, the *viewMessages(messageContainer : String)* method is called passing a string of "i" in case of *inbox* and "s" in case of *sent*.
- The *viewMessages(messageContainer : String)* method writes the meta-data info returned by either *getSentMessageInfo* or *getRecieveMessageInfo* based on the parameter to the client. The method then reads the number of the message to display its details (i.e. adding the content). When the number is read, the message is displayed by getting the message using the *getMessageByNumber* of either the *inbox* or *sent* of the user object, then calling the *toString* method of that message. If the message is in the *loggedInUser*'s inbox and is viewed then it is immediately marked as seen.
- The *logOff* method closed the socket between the server and client and returns.

7. Client Class :

- The main method of the class connects to the server on port 4000 and initializes the input/output streams to read and write to both the server and the user console. Then provides a user interface for the log-in/sign-up menu; reading the user console input and passing it to the server. Then calling either *logIn* or *signUp* methods. The method remains in this menu until a successful log-in is encountered, where a *serveUser* method will be called then the client will be terminated.
- The *logIn*/*signUp* methods prompt the user for his password and username, send the input to the server, then read the server response code and write a user-friendly notification of the state of the log-in/sign-up, either successful or the reason of the error.
- The *serveUser* method displays the main menu of the application to the user console, then reads the user input of the operation choice and passes it to the server, then calls either *sendMessage*, *viewMessages* or *logOff* based on the choice. The method remains in this menu until *logOff* is called and finished, then the method also returns.

- The *sendMessage* method reads the names of the registered users from the server and displays them to the user. Then reads the user input of the name of the wanted recipient and sends it to the server. After that it reads the server response code, if the code is a success the method proceeds to prompt the user for a title and a content for the message and sends the info to the server, otherwise, it displays a notification to the user of the reason of the error.
- The *viewMessages* method displays a menu for the user to choose between viewing his inbox or his sent messages, then sends the choice to the server. The server sends the meta-data info of all messages in the selected message box, this method displays that info to the user. The user enters a number representing one of the messages, the client passes this number to the server then receives the message itself from the server and displays it to the user. If an error code is returned from the server; because an invalid number is entered by the user, the client displays a message explaining the cause of the error and prompting for a valid input.
- The logOff method closes the socket connection to the server and returns.