PLAGIARISM STATEMENT

Name Shreyas Sesham

Signature [illegible]

Date 25/11/21

Name Sarthak Agarwal

Signature [illegible]

Date 25/11/21

Description of Classes

Server

Class Server contains four variables ipAddress, which includes the IP address of the server, socket, socket, the port the server is connected to and inputStream, which is used to take in an ordered system of bytes as an input from the client and send it to the server.

The constructor for the server takes in the ipAddress and port as input and assigns them to the class variables. It then checks whether the IP address is not null and not empty. If the IP address is not null and not empty, the constructor creates an object for a new server socket with the given port and an InetAddress with a reference to the IP address entered, if the IP address is null or empty, the InetAddress of the Server Socket is accessed through the localhost.

Method listen, that may throw an Exception contains String type variable data that stores the data sent by the client, a client of type socket that accepts a connection from the server socket and a String clientAddress that returns the host address from the Inet address of the client socket. It takes an input stream from the client and using a BufferedReader and InputStreamReader assigns the received data to the data variable. It then prints the address of the client and the message received. If the data received is not empty, it sends a prompt message to the client port. The method then takes in an authentication string and if this is sting is “yes”, it sends the response to the client and prints “Response Sent” and closes the input stream. If the authentication string is “no” it prints “Response not sent!” and closes the input stream.

getSocket of type InetAddress returns the Inet address of the socket.

getPort of type int returns the local port of the socket.

sendPromptMessage, which might throw an IOException takes in the client socket as an input. It uses PrintWriter to print the output stream. It asks the server whether it has sent the request packet and prints the message from the client and asks whether the client has sent the request packet.

sendResponse, which might throw an IOException takes in the client socket as an input. Variable data of type string stores the data being passed on through the packet and is hardcoded as an example for simplicity’s sake. Variable response of type ResponsePacket with parameters as the host address of the socket, the host address of the client, the protocol (currently hardcoded as TCP/IP) and the data of the packet, and sends “Response Sent!” as response message in the packet. It then prints the output message on the client.

The main method creates a server with address 127.0.0.1 and port 8000, prints the address and port of the server. It tries to execute the listen method and prints and Socket Exception if caught.

Client

Class Client contains two variables socket, which is an instance of the socket class that opens a socket connection to the server and ip_adress, which holds the IP address of the client.

The Constructor for Client takes the server address and server port as inputs and may throw an Exception. It then assigns the client socket to a socket corresponding to the server address and port.

Method start may throw an exception and it calls method sendrequest.

Method listen may throw an exception. It contains a variable data of type string that stores the data sent by the server. It uses a Buffered Reader to take data in from the server and prints the message along with the server address.

Method sendRequest, which may throw an Exception contains a variable currentIP to store the IP address of the client. It creates a request of type RequestPacket with protocol TCP/IP, and sends a hardcoded message as “Request!” It prints the message from the client to the server along with the IP address on the console.

Method getResponse may throw an IOException, it contains a response string, which is read through a response reader of type BufferedReader with input from the server. The response reads this message and checks whether it is null. If the response is not null, it prints a Success message and reveals the packet information, else it prints an authentication failure error message. The response reader is closed after this operation.

Method authenticate takes a parameter of a string validate and may throw an exception, If validate equals “yes” or “no”, the message contained in validate is printed on the server.

Method getIPAddress returns the IP Address of the client.

Method main may throw a NumberFormatException, UnknownHostException or Exception. It tries to execute taking a user input for the IP Address and port of the Server for the CLient to connect to and check the authentication. If the server is valid it creates a new client connected to the server IP and Port. It starts the client and calls the listen function of the client and then the input for the authenticate function in the form of “yes” or “no” and calls the validate and getResponse functions of the client. If a connect exception is generated during the process, it is printed out.

Packet

Interface Packet contains 2 methods displayMessage of type Sting and actionComplete of type boolean.

ActionPacket

Class ActionPacket implements the interface Packet and contains 6 variables; source of type string, which contains the source of the packet, destination of type string that holds the destination of the packet, protocol of type string that contains the protocol through which the packet is transferred, message of type string containing the message that needs to be transmitted, actionComplete of type boolean that describes whether the desired action was fulfilled and packetType of type string that stores whether the packet is a request or response packet.

There are 3 constructors for this class one which takes inputs of only packetType, another which takes in source and destination in addition and the third which takes in the protocol as well. The default values of the source and destination are “127.0.0.1” and the default for the protocol is “TCP/IP”.

Method setMessage assigns the parameter message to the class variable message.

Method displayMessage, which overrides the method from the interface packet of return type String returns the stored message.

Method actionComplete of type boolean overrides the method from the interface packet returns the value of the actionComplete variable.

Method getSource of type string returns the value of the variable source.

Method getDestination of type string returns the value of the variable destination.

Method getProtocol of type string returns the value of the variable protocol.

RequestPacket

Class RequestPacket extends the class ActionPacket.

The constructors for this class call the constructors for the superclass(Action Packet) with packet type as “Request”

Method setMessage takes the message as a parameter and calls the setMessage of the superclass.

Method getRequestMessage of return type string calls the displayMessage of the superclass.

Method toString overrides the system method and calls the getRequestMessage method.

ResponsePacket

Class RequestPacket extends the class ActionPacket. It contains a variable data of type string that contains the data to be sent through the response.

The constructors for this class call the constructors for the superclass(Action Packet) with packet type as “Response” with the addition of data of type String that assigns the parameter to the data variable.

Method getResponseMessage of type string returns the displayMessage method of the superclass.

Method setResponseMessage takes the message as a parameter and calls the setMessage of the superclass.

Method toString of type string overrides the system method stores the Source IP, Destination IP, Protocol and Data in a variable output of type string and returns the output variable.

Inputs and Outputs

The server needs to be run first followed by the client. Upon running the server the console displays “Running Server” along with the IP address of the host and port the server is connected to.

Once the client is activated, it displays the message “Enter the Server IP Address: “, where the user gives an input of the IP and “ENter server port to connect:”, where the user enters the port of the server that the user desires to connect to. The console then displays “Press enter to authenticate!” and the user is supposed to press enter. If the IP address and port entered don’t match the IP Address and port of the server that is run, the client console displays java.net.ConnectException: Connection refused: connect. If the IP and port match the server, the client console displays “Connected to Server: /” with the IP address. And the server console displays “New connection from:” with the IP address of the client.

The client then displays a request message in its console and sends it to the server which displays the same. The server then asks the client whether it has sent a request packet, which is displayed on both consoles. If the user inputs yes on the client’s console then the server receives and displays the same and then send the response and displays “Response sent!” on the console. The client receives this response and prints “SUCCESS: Response received!!!” on its console followed by the packet information. If the user inputs “no” on being asked whether they sent the request packet, the “no” is sent to the server and displayed on the server console followed by a “Response not sent!” message and “ERROR: Authentication failure!!!” is displayed on the client console.

Known Limitations

The authentication needs to be manually done and detection of spoofed IPs has to be manually done by the user operating the client, and they need to check the validity of the IP of the packet in the same way. This also does not allow for the detection of packets with spoofed IPs. The system remains permeable to hackers and viruses due to the same and hence the system is not secure. Another limitation of the project is that the data stored inside the packet and the message is hardcoded, purely as an example to show how the overall system functions.

Future work Possible

Prospects of this project include a provision for IP spoofing and detection of spoofed IP addresses along with a higher degree of automation in detecting the authenticity of the packet received by the server rather than the client manually validating whether they have sent the packet or not. Future work can also be done to store more meaningful data in the packet and provisions to encode this through the user and the client, instead of it being an exemplar hardcoded data.

Analysis of the Design Principles

1. Encapsulate What Varies:

Encapsulation is the process of wrapping code and data together in a single unit. This can be done by making all the data members of the class private and using setter and getter methods to set and get the data from it. It has several advantages such as more control over the data, helping achieve data hiding and being easy to test.

The principle “Encapsulate what varies” means encapsulating the code that varies a lot. To account for this the Requeest packet and and Response packet classes had a similar basis, however had minor differences in the functioning of all the methods that were present in both the classes. Due to this an Action Packet class was created that contains a brief abstraction of the common methods and can be utilized by both the classes with provisions for required variation.

2. Favouring Composition over Inheritance:

“Favoring composition over inheritance” is an OOP design principle that suggests that classes should achieve polymorphism by composition over inheritance.

Since Java doesn’t support multiple inheritances we favour composition over inheritance. Since you can only extend a single class in Java, if you need multiple features, you can make them as private members and achieve composition.

Even though composition and inheritance both allows us to reuse code, the disadvantage of using inheritance is that it breaks encapsulation so if child class depends on the action of the parent class for its function, if the parent class behaviour changes, then the child class functionality can be broken.

The ActionPacket class is the parent class that inherits the Packet interface and has two child classes namely Request and Response packet. Although this design largely follows the inheritance principle, it can be refactored to follow composition.

The Request and Response packet classes can have a data member that is an object of ActionPacket class in addition to the additional data members present in their respective classes.

3. Program to an interface not implementation

Coding to interfaces is a technique to write classes based on an interface; an interface that defines what the behaviour of the object should be. It involves creating an interface first, defining its methods and then creating the actual class with the implementation.
The client code(not to be confused with Client class) always holds an Interface object which is supplied by a factory.
Any instance returned by the factory would be of type Interface which any factory candidate class must have implemented. This way the client program is not worried about implementation and the interface signature determines what all operations can be done. This can be used to change the behaviour of a program at run-time. It also helps you to write far better programs from the maintenance point of view.

However, the current code does not implement this principle. It can be implemented by making a interface for Packet that the Request and Response packet classes implement and while creating objects of the packet call it by using
Packet requestPacket = new Request();

4. Strive for loose coupling between objects that interact

In loose coupling, a method or class is almost independent, and they have less dependence on each other.In this project, while the Request and Response classes are both tightly coupled with the ActionPacket classes, the Client and Server classes are loosely coupled.

The Request and Response classes inherit from the ActionPacket class so any change in the ActionPacket class can majorly impact these two classes.

Client and Server classes are largely independent of each other as they do not inherit from a common parent class.

5. Classes should be open for extension and closed for modification

“Classes should be open for extension and closed for modification” means we should code such that we’ll be able to add new functionality without changing the existing code. This can be done using inheritance or using interfaces.

However, usage of inheritance will lead to tight coupling hence it is suggested to use interfaces.

In this project, inheritance has been used where ActionPacket is the parent class and Request and Response packets.We can add new types of packets by just creating new classes that inherit from the ActionPacket class for which we do not need to modify the existing code.

6. Depend on abstraction, not on concrete classes

This means that it is better to code to interfaces or abstract classes rather than inheriting a non abstract class.In this project the ActionPacket implements the Packet interface however the RequestPacket and ResponsePacket just inherits a concrete class.

Hence, in order to implement this principle, we can create an abstract class that can be inherited by both RequestPacket and ResponsePacket.

Analysis of Design Patterns:

1. Factory Design Pattern:

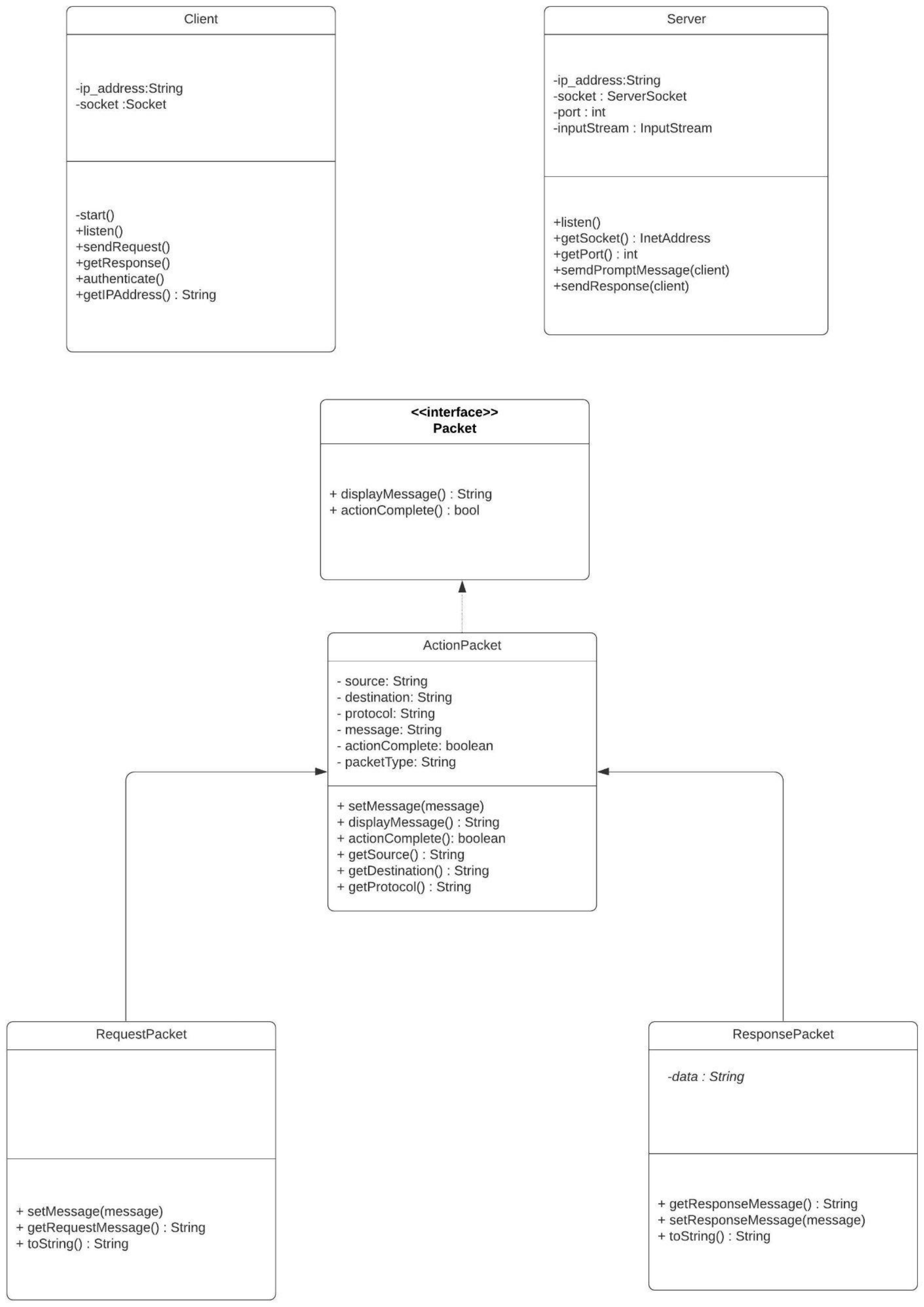
In a factory design pattern, an object is created without exposing the creation logic to the client and refer to a newly created object using a common interface.

This design pattern is partially followed in the code of the project in the implementation of the Packet, ActionPacket, RequestPacket and ResponsePacket. The interface Packet is defined and ActionPacket implements the interface and contains most of the necessary methods. RequestPacket and ResponsePacket inherit this class and have objects made from them, however, there is no factory class implemented in the project.

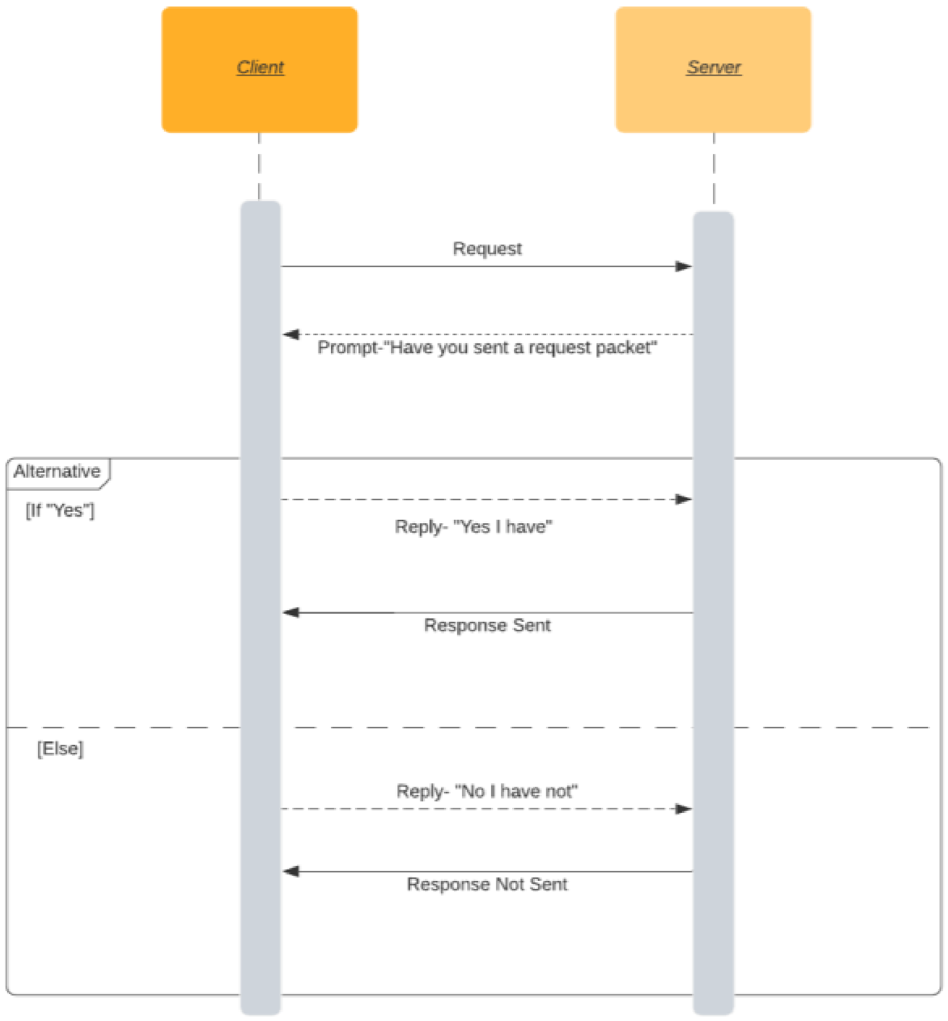
A way to implement this principle could be to define a PacketFactort class through which the request and response packets are generated.

UML Diagrams

1)Class diagram



2)Sequence diagram



GitHub link to the project: https://github.com/shreyas1209/Client-Server-Authentication
Drive link to the video:
https://drive.google.com/drive/folders/13Dl-AVtm5UzkO5P1gK4bguaLsrg7K9Cl