Port knocking authentication documentation

Contents

Quick star……2

Libraries……3

Project Structure……3

Input arguments……4

Code Documentation……5

Quick start

1. Server

1.1. Execute

java -jar Server.jar -m <[message to clients(optional)]> -p <[your port 1]> <[your port 2]> <[your port n]>

or

java -jar Server.jar -m <[message to clients(optional)]> -n <[number of ports that will be opened automatically]>

or

java -jar Server.jar --help

1.2. Watch output

Server should print a short notification, telling user that it was started, show you what message will be sent to authenticated clients and give order of authentication
ports to which client should knock to in order to pass the authentication.

```
$ java -jar Server.jar -m "My message to client" -p 12345 12346 12347
-Server was started
-Such message will be send to all clients that passed authentication: "My message
to client"
-Server will be waiting client to knock in such order of authentication ports:
0 -> 12345
1 -> 12346
2 -> 12347
```

2. Client

2.1. Execute

java -jar Client.jar -s <[server address]> -p <[port #1]> <[port #2]> <[port #n]> -m <[message to server(optional)]>

or

java -jar Client.jar --help

2.2. Watch output

Client should print a short notification, telling that it was started, show you what message will be sent to server after passing authentication and show order of authentication
ports to which client will knock to in order to pass the authentication.

```
$ java -jar Client.jar -m "My message to server" -s "192.168.0.105" -p 12345 12346 12347
-Client was started
-Such message will be send to server: "My message to server"
-Client will be knocking in such internet socket addresses:
0 -> 192.168.0.105:12345
1 -> 192.168.0.105:12346
2 -> 192.168.0.105:12347
```

3. Back to server

3.1. If some of the clients passed the authentication, server should display its address and message that was sent:

```
$ java -jar Server.jar -m "My message to client" -p 12345 12346 12347
-Server was started
-Such message will be send to all clients that passed authentication: "My message
to client"
-Server will be waiting client to knock in such order of authentication ports:
0 -> 12345
1 -> 12346
2 -> 12347

Client 192.168.0.105 responded with "My message to server"
```

4. Back to client

4.1. If client passed authentication, it will display the received message from server:

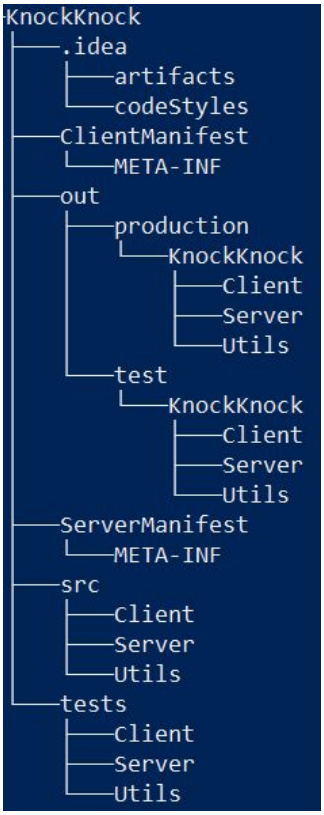
```
$ java -jar Client.jar -m "My message to server" -s "192.168.0.105" -p 12345 12346 12347
-Client was started
-Such message will be send to server: "My message to server"
-Client will be knocking in such internet socket addresses:
0 -> 192.168.0.105:12345
1 -> 192.168.0.105:12346
2 -> 192.168.0.105:12347

Server responded with "My message to client"
```

Libraries

1. openjdk 8
2. hamcrest-core 1.3
3. Junit 4.12

Project Structure



1. .**idea** - folder that stores information about project for **IntelliJ IDEA ide**
2. **ClientManifest** - folder that contains metadata for client
3. **ServerManifest** - folder that contains metadata for server
4. **out** - folder where compiled code is stored
   - 4.1. **production** - folder with compiled code from **src** folder
   - 4.2. **tests** - folder with compiled tests
5. **src** - folder that contains source code for **client** and **server**
   - 5.1. **Client -** package that contains source code for client
   - 5.2. **Server** - package that contains source code for server
   - 5.3. **Utils** - package that contains source code for utils
6. **tests** - folder that contains source code for test
   - 6.1. **Client -** package that contains source code for client tests
   - 6.2. **Server** - package that contains source code for server tests
   - 6.3. **Utils** - package that contains source code for utils tests

Input arguments

- Server
  - **-p** obligatory flag which tells the program that after this argument sequence of ports for opening authentication sockets will be given.
    Example: java -jar Server.jar **-p 12345 12346 12347 12348**
    This flag can not be specified alongside with flag **“-n”**
    If a server can not be created with some port, the server will show you an error message and will be terminated.
  - **-n** obligatory flag which tells the program that after this argument there will be a number that defines the quantity of sockets that will be opened by the server automatically.
    Example: java -jar Server.jar **-n 12**
    This flag can not be specified alongside with flag **“-p”**
  - **-m** optional flag which tells the program that after this argument there will be a custom message that the server will send to all of the clients that passed authentication. If a message is not specified, the server will send “Hello client!” message by default
    Example: java -jar Server.jar -n 12 **-m “My message to clients”**
  - **--help** optional flag that shows a little hint. Can not be used alongside with any flags.
- Client
  - **-s** obligatory flag which tells the program that after this argument there will be an address of the server to which the client will try to “knock”.
    Example: java -jar Client.jar **-s “123.345.678.90”** -p 12345 12346 12347 12348
  - **-p** obligatory flag which tells the program that after this argument a sequence of ports will be given to which client will try to knock. Ports should be given in the same order as the user wants the client to knock.
    Example: java -jar Client.jar -s “123.345.678.90” **-p 12345 12346 12347 12348**
  - **-m** optional flag which tells the program that after this argument there will be a custom message that the client will send to the server after passing authentication. If a message is not specified, the client will send “Hello server!” message by default
  - Example: java -jar Client.jar -s “123.345.678.90” -p 12345 12346 12347 12348 **-m ”My message to clients”**
  - **--help** optional flag that shows a little hint. Can not be used alongside with any flags.

Code Documentation

Package **Server:**

Class **ServerApp**

Class that is responsible for proper server starting, and handling input arguments from user

Constructors:

NONE

Methods:

`public static void main(String[] args)`

Methods that process user input arguments and launches server with specified properties

`private static void checkArgumentsCorrectness(String[] args)`

Checks if given arguments were correct

Class **AuthenticationServer**

Implementation of authentication server

Constructors:

`public AuthenticationServer(String messageToClients, int numberOfAuthenticationSockets)`

Creates a server with a given number of sockets and specifies custom messages that will be sent to clients that pass authentication.

`public AuthenticationServer(int numberOfAuthenticationSockets)`

Creates a server with a given number of sockets and specifies default message (`"Hello client!"`) that will be sent to clients that passed authentication.

`public AuthenticationServer(int... customUserPorts)`

Creates server with custom user ports and specifies default message (`"Hello client!"`)

`public AuthenticationServer(String messageToClients, int... customUserPorts)`

Creates server with custom user ports and specifies users message

Methods:

`public void startServer()`

Method that initializes and starts a specified number of AutenticationSockets and other additional data-structures required for the server.

`public void stopServer()`

Method that stops all of AutenticationSockets, deletes all data-structures that was used for authentication and preppers server for next possible start.

`public void openSocketForSuchAddress(String remoteAddress, int remotePort)`

Method that opens TCP socket for given address and sends information about opened port to the client's UDP socket.

`public synchronized boolean checkAuthentication(String addressOfRequester, int authenticationSocketNumber)`

Checks if the given address has been knocking to previous ports in the right order.

`public synchronized void addAddressToAuthenticationList(String address, int socketAuthenticationNumber)`

Adds address to structure that holds information about order of knocking of given address

`public synchronized void removeAddressFromAuthenticationList(String address, int socketAuthenticationNumber)`

Removes all information about this address and its authentication from the server

`public boolean isWorking()`

Indicates if server is working

`public int[] getAuthenticationPorts()`

Returns all authentication ports in right order

`public void addMessage(String address, String message)`

Adds message to the Map data-structure that holds information about received messages from clients.

`public Map<String, String> getMessagesFromAuthorisedClients()`

Returns Map data-structure that holds information about received messages from clients.

`public String getMessageToClients()`

Returns message that will be send to authenticated clients

`public Set<String>[] getAuthenticationAddresses()`

Returns Array of sets that holds information about authentication (Mostly used for tests)

Class **AuthenticationSocket** extends Thread

Implementation of authentication socket that defines proper sequence of knocking for authorization.

Constructors:

```
public AuthenticationSocket(AuthenticationServer server, int authenticationNumber) throws SocketException
```

Constructor that creates AuthenticationSocket with given number that identifies the order in which the client should knock to pass the authentication

```
public AuthenticationSocket(AuthenticationServer server, int customUserPort, int authenticationNumber)
```

Constructor that creates AuthenticationSocket with a custom port. If a socket can not be opened with such port, the process will be terminated.

Methods:

```
public void stopListening()
```

Stops socket thread

```
public void run()
```

Method that specifies logic of the socket that will be executed in separate thread. Method listens for income datagrams and notifies the server about it. If the address was knocking in the right order to the previous sockets, it will be added to the structure that holds information about the order of knocking or if the socket is the very last in the order and income address passed authentication in previous sockets, it will ask server to open TCP socket for such address. Otherwise it deletes all information about this address and its authentication from the server.

```
public int getPort()
```

Returns port of given AuthenticationSocket

Class **Server Processing** extends Thread

Class that processes client that passed authentication in separate thread via TCP socket.

Constructors:

```
public ServerProcessing(Socket socket, AuthenticationServer server)
```

Creates instance with specified connected socket and reference to the server

Methods:

```
private void processClient() throws IOException
```

Method that establishes communication with client

```
public String getMessageFromClient()
```

Returns message that was received from the client

Package **Client:**

Class **ClientApp**

Class that is responsible for proper client starting, and handling input arguments from user

Constructors:

NONE

Methods:

```
public static void main(String[] args)
```

Methods that process user input arguments and launches client with specified properties

```
private static void checkArgumentCorrectness(String[] args)
```

Method that checks correctness of given arguments for input

Class **AuthenticationClient**

Implementation of the client that should work with port-knocking authentication

Constructors:

```
public AuthenticationClient(String authenticationServerAddress, String messageToServer, int... authenticationServerPorts)
```

Constructor that creates instance with given server address where clients will knock to. Specifies the message that client will send to the server after authentication. Specifies the ports and its order to which client will knock in order to pass authentication.

```
public AuthenticationClient(String authenticationServerAddress, int... authenticationServerPorts)
```

Constructor that creates instance with given server address where clients will knock to. Specifies the ports and its order to which client will knock in order to pass authentication. Specifies default message (`"Hello server!"`) which client will

send to the server after authentication.

Methods:

```
public void startClient()
```

Method that starts client.

```
public void startKnocking()
```

Method that asks the client to start knocking in ports specified in the constructor before.

```
public void listenToServer() throws IOException
```

*Method that asks client to start listening to income datagrams from the server to discover on which port TCP connection was opened if client passed authentication. If there is no response from the server for **5 seconds**, it terminates the client.*

```
public void connectToServerSocket(String address, int port) throws IOException
```

Method that establishes communication with opened TCP socket on the server.

```
public String getMessageFromServer()
```

Returns message that server sent to the client during communication

```
public String getMessageToServer()
```

Returns message that client will send to the server during communication

Package **Utils:**

Class **Constants**

Class that holds common variables for different classes, constant values, and debug properties for tests

Constructors:

NONE

Methods:

NONE

Class **KnockUtils**

Implementation of tools that is commonly used through the project and tests.

Constructors:

NONE

Methods:

```
public static<T> T[] shuffleArray(T[] arrayToMix)
```

MIx order of given array

```
public static void sendDatagramMessage(String message, String destinationAddress, int destinationPort)
```

Sends message via UDP to the given IP address

```
public static void sendDatagramMessageFromBoundedSocket(String message, String destinationAddress, int destinationPort, DatagramSocket socket)
```

Sends message via UDP from specific socket to the given IP address

```
public static boolean checkIfSuchNetworkInterfaceExists(String address)
```

Checks if given address is available

```
public static List<String> getAllAvailableInetInetInterfaces()
```

Returns the list of all available addresses

```
public static<T> int indexOf(T[] array, T find)
```

Return index of the first element that was found in the given array. If the element was not found, it returns -1