Port knocking authentication documentation

Quick start

1. Server

1.1. Execute

java -jar Server.jar <[Number of ports to open for authentication]>

or

java -jar Server.jar <[Number of ports to open for authentication]> <[Message to client(optional)]>

1.2. Watch output

Server should print a short notification 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 pass the authentication.

```
$ java -jar Server.jar 3 "My message to client"
-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 -> 62577
1 -> 62578
2 -> 62579
```

2. Client

2.1. Execute

java -jar Client.jar <[Server address]> <[Authentication ports]>

or

java -jar Client.jar <[Server address]> <[Message to server(optional)]> <[Authentication ports]>

2.2. Watch output

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

```
$ java -jar Client.jar 192.168.0.105 "My message to server" 62577 62578 62579
-Client was started
-Such message will be send to server: "My message to server"
-Client will be knocking in such internet socket addresses:
0 -> 192.168.0.105:62577
1 -> 192.168.0.105:62578
2 -> 192.168.0.105:62579
```

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 3 "My message to client"
-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 -> 62577
1 -> 62578
2 -> 62579

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:

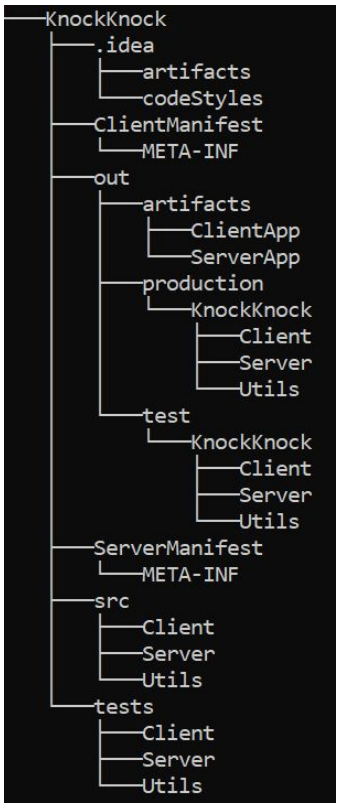
```
$ java -jar Client.jar 192.168.0.105 "My message to server" 62577 62578 62579
-Client was started
-Such message will be send to server: "My message to server"
-Client will be knocking in such internet socket addresses:
0 -> 192.168.0.105:62577
1 -> 192.168.0.105:62578
2 -> 192.168.0.105:62579

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. **artifacts** - folder with compiled jar files
    4.2. **production** - folder with compiled code from **src** folder
    4.3. **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

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 checkArgumentCorrectness(String[] args)`

    Checks if given arguments were correct

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

    Checks if all conditions were satisfied when user gave only one argument for input

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

    Checks if all conditions were satisfied when user gave two arguments for input

- Class **AuthenticationServer**

  Implementation of authentication server

  Constructors:

  `public AuthenticationServer(int numberOfAuthenticationSockets, String messageToClients)`

  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)`

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

  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

  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 instances with given server address where client 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 instances with given server address where client 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 10 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)`
  Mlx 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