Opt-Out Challenge 3

Reading Assignment: How to Program Java, 10th edition

- Chapter 7 – Arrays and ArrayList
- Chapter 8 – Classes and Objects: A Deeper Look
- Chapter 9 – Object-Oriented Programming: Inheritance

Programming Challenges

Create a two-way chat service composed of a server and a client. This should be implemented using stream sockets. The client should be able to access the server over the Internet.

Simple Server Using Stream Sockets

Establishing a simple server in Java requires five steps.

1. Create a `ServerSocket` object:

   ```
   ServerSocket server = new ServerSocket(portNumber, queueLength);
   ```

   The variable `portNumber` is an admissible TCP port number and `queueLength` is the maximum number of clients that can wait to connect to the server.

2. Wait for a connection:

   ```
   Socket connection = server.accept();
   ```

   In this step, the server listen indefinitely for an attempt by a client to connect. The method returns a `Socket` when a connection with a client is established.

3. Manage the I/O streams associated with the socket:

   ```
   connection.getOutputStream();
   connection.getInputStream();
   ```

   These objects can subsequently be employed to send or receive bytes with the `OutputStream` method `write` and the `InputStream` method `read`, respectively. One can also use classes such `ObjectInputStream` and `ObjectOutputStream` to enable entire objects to be read from or written to a stream, a technique called wrapping.

4. Support the live interaction: In the processing phase, the server and the client communicate via the `OutputStream` and `InputStream` objects.

5. Closing the connection: The server closes the connection by invoking the `close` method on the streams and on the `Socket`.

Simple Client Using Stream Sockets

Establishing a simple client in Java necessitates four steps.

1. Create a `Socket` to connect to the server:

   ```
   Socket connection = new Socket( serverAddress, port);
   ```

   When the connection attempt is successful, this returns a `Socket`.

2. Manage the I/O streams.

3. Support the live interaction.

4. Close the connection: The client closes the connection by invoking the `close` method on the streams and on the `Socket`.

Code

1. Implement this task in Java.

2. Using IntelliJ IDEA, Git, and GitHub, commit your code for the server as a project labeled `Challenge3server` under `Students/<GitHubID>/`, where `<GitHubID>` should be replaced by your username on GitHub.

3. In a similar fashion, commit your code for the client as a project labeled `Challenge3client` under `Students/<GitHubID>/`.