School of Information and Physical Sciences
University of Newcastle
SENG4500 –Network & Distributed Computing

Assignment 1

Due using the submission facility of the Canvas Assignment facility: 11:59 p.m. 28/08/22

NOTE: The important information at the end of this assignment specification

Your task in this assignment is to create a server process that uses TCP to implement a service that returns the income tax payable for a client-provided income. A client, whose process you will also create, will specify the tax scale details for system configuration, and the income for which the payable tax is requested.

By way of example, take this income tax scale as follows:

Income Range	**Tax Payable**
$0 - $18,200	Nil
$18,201 - $37,000	19c per dollar over $18,200
$37,001 - $80,000	$3,572 plus 32c per dollar over $37,000
$80,001 - $180,000	$17,547 plus 37c for each dollar over $80,000
$180,001	$54,547 plus 45c for each dollar over $180,000

The server, which will be called `TaxServer`, will operate as follows:

1. Return the income tax payable by a taxpayer after the client connects to the server using the `TAX` instruction,
2. Accepts a `STORE` message used for storing new tax scale entries,
3. Accepts a `QUERY` message used to ask which tax scale entries it knows,
4. Accepts a `BYE` message that results in closing of the current session between the client and the server.
5. Accepts an `END` message that results in closing of the current session and termination of the server.

Operation of the client and the server is as follows.

When the `TaxServer` server starts it implements an internal data structure of size sufficient to store up to ten income ranges, but initially knows nothing about any income ranges or tax payable. The server then waits for a client to connect to it using a server port number specified by you when you start the server. The server is not required to concurrently interact with clients. In other words, if the server is interacting with a client, no other clients are required to be able to interact with the server.

The client (called `TaxClient`) uses `TAX`, `STORE`, `QUERY`, `BYE` and `END` messages to instruct the server. The client program will be capable of sending any of these messages on instruction from the user.

Operation of the client-server pair is as follows:

- `TAX:` This message is sent from the client to the server to make the client's connection to the server, thus initiating a session. It contains the ASCII string "TAX" followed by the newline character "\n". On sending the `TAX` message, the client waits for a return message from the server via the socket that connects them. After receiving and displaying the return message the client loops back so that the user can initiate sending of another message.
  On receipt of a `TAX` message from a client, the server returns the string "`TAX: OK`" followed by a newline character and waits for further instructions from the client.
  The message sequence to start a calculation session is as follows:
  ```
  Client:  TAX
  Server:  TAX: OK
  ```
- `STORE:` This message is sent from the client to the server and causes the following income range and tax rule (i.e. four arguments to the `STORE` instruction) to be stored in the server's memory. The client sending the ACSII string "STORE" followed by the newline character starts the exchange. Following this the client sends the start income for the range to be stored, a newline character, then the end income for the range to be stored, a newline character, then the base tax for the income range, a newline character, and finally the tax (in cents) per dollar

earned above the start income for the range. An unspecified income value (for example in the last above income range, which has open upper end) is indicated by '~'. The server responds with the "STORE: OK" message to confirm that the store operation has been completed.
A typical message sequence would be as follows:

```
Client:  STORE
         18201
         37000
         0
         19
SERVER:  STORE: OK
```

If the server already knows a range including the income specified by the first argument, the previously-known range is either replaced (if the end income matches) or shortened as appropriate. For example, if the server previously knew of income ranges 0 – 10000 and 10001 – 20000, and a new definition was provided for the range 9001 – 18000, the server would shorten the previously know ranges, and create a new entry, ending up with ranges 0 – 9000; 9001 – 18000; and 18001 – 20000. If, following this sequence, a further range 1001 - 19000 was defined, the previous 9001 – 18000 range would be subsumed, and the result would be ranges 0 – 1000; 1001 – 19000; and 19001 – 20000.

- QUERY: This message is sent from the client to the server, and causes the server to return the tax scale data stored in the server's memory. The client starts the exchange by the sending the ACSII string "QUERY" followed by the newline character. The server responds with a sequence of newline-terminated tax scale data, followed by the "QUERY: OK" message to confirm that the query operation has been completed.
  A typical message sequence would be as follows:

```
Client:  QUERY
SERVER:  0 18200 0 0
         18201 37000 0 19
         37001 80000 3572 32
         80001 180000 17547 37
         180001 ~ 54547 45
QUERY: OK
```

- Requesting tax calculations is performed when the client sends a string, representing the income, terminated by a newline character. On receipt of such a string, the server returns the tax payable for the income specified by the client, or indicates that it does not know how to calculate for that income (e.g. if the provided income falls in an undefined range), as shown in the following examples.

```
Client:  10000
Server:  TAX IS 0

Client:  100000
Server:  I DON'T KNOW 100000
```

- BYE: This message is sent from the client to the server, instructing the server that the client no longer needs the current session (that was initially created by the TAX message). The client sends the ASCII string "BYE" followed by a newline character. After reading and displaying the returned message from the server, the client should close its connection.
  The server, on receipt of the BYE message, should return the string "BYE: OK" followed by a newline character, and then should wait for a connection from a new client. Note that any tax scale data stored by the old client are retained in the server's memory and can be requested by any new client.
  A typical message sequence would be as follows:

```
Client:  BYE
SERVER:  BYE: OK
```

- END: This message is sent from the client to the server, instructing the server to shutdown. The client sends the ASCII string "END" followed by a newline character. After reading and displaying the returned message from the server, the client should close its connection.
  The server, on receipt of the END message, should return the string "END: OK" followed by a newline character, then should close all open sockets and terminate. Note that any tax scale data added since the server started are not saved, and if the server is subsequently re-started, it reverts to knowing nothing about any tax scales (but having data structures in which client-provided data could be stored).
  A typical message sequence would be as follows:

```
Client:  END
Server:  END: OK
```

Note that, while proper commercial software would require user input error checking, the purpose of this assignment is to achieve connectivity according to the specified protocol, and you *do not need to test for invalid user input* – you may assume all user input is perfectly compliant.

Before starting you should read the tutorial provided at:
http://download.oracle.com/javase/tutorial/networking/sockets/
Details of a server-side connection-oriented socket are provided in the Java API (http://docs.oracle.com/javase/7/docs/api/index.html) under `ServerSocket`.

The user interface for your server should provide a default port number, and allow the user to over-ride that port number if required. The user interface for your client should provide `localhost` (`127.0.0.1`) as the default server, and the same default port as for the server; once again the user should be able to override the defaults if he/she wishes to do so.
Tips:

- When your submission is marked, we will use a sample client to interact with your server, and a sample server to interact with your client. It is therefore non-optional that you comply with the specified interaction protocol (which is, of course, consistent with the 'real world' of computer networking.
- It is imperative that you use a server port number bigger than 1023 so that you do not interfere with any system-provided ports.
- Start by producing a client and server that can execute the `TAX` message exchange. Then extend your solution to implement the required functionality.
- Initially you can test your program using the loopback IP address on your single node, as discussed in lectures. Make sure that the socket(s) you use are closed when your program terminates. If your program 'crashes' any open socket may remain allocated to that program, and an error may occur when a later program invocation attempts to bind to that socket. If you are running your program on a Unix host, please understand that the port number(s) you use are system-wide, so if you have a problem binding to a port some other concurrent user already having that port open may be the cause.
- To determine your computer's IP address you can
  - Use the `ifconfig` or `ipconfig` commands on modern Unix or Windows-based systems.
  - Examine the properties of your Windows network connection and select the Support tab, or
  - Use the `nslookup <hostname>` command on a Unix system, or
  - Use `winipcfg` on an older Windows system.

Your submission should be made using the Assignments section of the course Blackboard site. **Assignments that do not use the specified class names (if any) will not be further marked. Assignments that do not allow the user to specify the port and/or server identity at run-time will not be further marked.** You should provide all your .java files. Also provide a readme.txt file containing any instructions for the marker. Each program file should have a proper header including your name, course and student number, and your code should be properly documented. Finally, a completed Assignment Cover Sheet should accompany your submission.

This assignment is worth 20 marks of your final result for the course.

Dr. Mark Wallis