

# Transmission Control Protocol (TCP) - Client

## Practical 1

### 1. Preparation

- Read this handout before coming to class
- Refresh your understanding of the TCP protocol
- To get the programs for this practical logon to Queen's Online.

### 2. The Client-Server Application

#### 2.1 The Client-Server Model

The two programs in the folder form a Client-Server model of communication. As discussed in the lectures, a server is a program that provides a service and can be reached over a network whereas a client is a program that sends a request to the server for a particular service and receives a response back from the server.

#### 2.2 Client side

When the client makes a connection to the server the server must be running and waiting for the client's connection. The client will be prompted to enter a message to send to the server and will enter an integer between 0 and 4. The client then waits for the server to respond and continues to contact the server until the client decides to quit.

#### 2.3 Server side

On the server side the message is received from the client, appropriate processing is carried out at the server and then a response is sent back to the client. In this practical the server holds a library of 4 documents. The documents are stored as an array of strings indexed 0 to 3. The client requests a document by sending the index to the server. The server responds by sending the client a copy of the document. The client requests documents from the server until the client decides to quit, as outlined in the following algorithm.

#### Client Algorithm

```
Loop
{
    Prompt user to enter index (0 to 3) of document required or 4 to quit
    Send index to server
    if index is 4 then exit loop
    wait for server to send requested document
    display document on screen
}
```

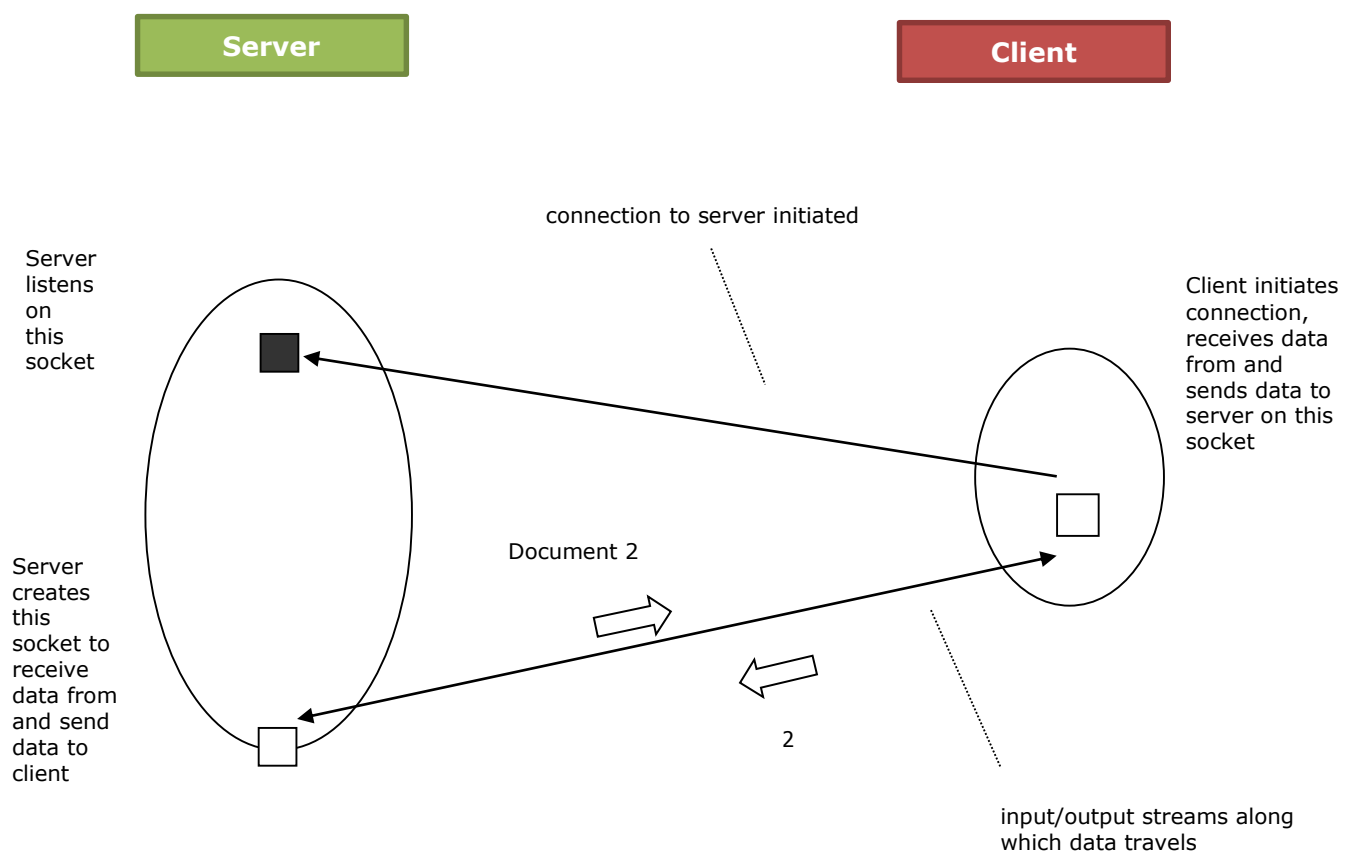
#### Server algorithm

```
{
    Wait for client to send index
    if index is 4 then exit loop
    send requested document to client
}
```

TCPServer.java is complete and just needs to be compiled and executed. Open TCPServer.java to examine the java code before attempting to complete TCPClient.java. Complete TCPClient.java using your programming knowledge; some code has been provided but read the comments in the code and look out for "...". Section 3 contains a summary of some Java constructors and methods that you will use when coding your solution and section 4 contains a pseudo-code summary of the interactions between TCPServer.java and TCPClient.java. You should read both these sections before attempting to complete the skeleton program.

### 3. Transmission Control Protocol (TCP)

TCP is a connection-oriented service which means that a connection between the source and destination must be established prior to transmission and is relinquished only after all transmission is complete. To facilitate communication hosts create end-points called sockets and data transmissions flow in streams between these sockets. A server will contain a server socket that listens at a particular port for connection requests. When it receives such a request it negotiates a connection and allocates another socket to handle all communication between the server and that client. A client contains only one socket from which it will initiate a connection to a server and if this connection is successful will use to send data to and receive data from the server.



### 3.1 Socket & Streams.

Within Java two classes, ServerSocket and Socket provide the means for network communications using TCP. A server socket runs on a server and waits for incoming TCP connection requests to come from a client.

#### Class ServerSocket

This class implements server sockets. A server socket listens on a particular port for connection requests from a client(s) and returns a socket object that is used to communicate with this client.

##### 1. Constructor.

public <b>ServerSocket</b> (int port) throws IOException
--

This constructor creates a server socket on the specified port of the local machine.
--

##### 2. Methods.

public Socket <b>accept</b> () throws IOException
---

This method waits for a connection request and accepts it. When a connection has been accepted a new Socket object is created by which the server will communicate with the client.
---

#### Class Socket

This class implements client sockets.

##### 1. Constructor.

public <b>Socket</b> (InetAddress address, int port) throws IOException
---

This method constructs a socket and connects it to the specified port number at the specified IP address (i.e. the port number at which the server is listening at the specified IP address). Two parameters must be provided:
--

- the InetAddress of the host
- the port number to which the socket will be bound

To get the IP address of the local machine use the getLocalHost method as detailed below – Class InetAddress.

##### 2. Methods.

public void <b>close</b> () throws IOException
--

This method closes a socket.
------------------------------

public OutputStream <b>getOutputStream</b> () throws IOException
--

This method returns an output stream for the socket.
--

public InputStream <b>getInputStream</b> () throws IOException
--

This method returns an input stream for the socket.
---

## Class InetAddress

An IP address is the address of a computer on the Internet; to send a packet to a destination we need to know its IP address. There are no constructors for this class instead instances can be created by using the static method `getLocalHost`.

### 1. Methods.

If the client and server are being executed on the same machine the following method would be used.

<code>public static <a href="#">InetAddress</a> <b>getLocalHost</b>() throws <a href="#">UnknownHostException</a></code>
--

This method returns an <code>InetAddress</code> object corresponding to the IP address of the local machine.
--

e.g. `socket = new Socket(InetAddress.getLocalHost(), 2000);`

## Class ObjectInputStream

To read data from a socket we must firstly get an `InputStream` from the socket using method `getInputStream`. However class `InputStream` offers only methods that permit the reading of bytes, therefore in order to read `Integer` and `String` values from the socket we add an `ObjectInputStream` onto the `InputStream` and use the appropriate method.

### 1. Constructor.

<code>public <b>ObjectInputStream</b>(InputStream in) throws IOException</code>
---

This method creates an <code>ObjectInputStream</code> that uses the specified underlying <code>InputStream</code> . One parameter must be provided:
---

- the specified input stream

### 2. Methods.

<code>public final Object <b>readObject</b>() throws IOException, ClassNotFoundException</code>
---

This method reads an <code>Object</code> from the attached stream as written by the <code>writeObject</code> method of <code>ObjectOutputStream</code> .
--

Method `readObject` returns objects from the input stream and the programmer must then cast these objects to the appropriate type. To do this we place the type within brackets in front of the read statement e.g.

```
int choice = (Integer)serverInputStream.readObject();
```

When there is no more data to be read from a socket the `ObjectInputStream` should be closed using method `close`, this method is inherited from class `FilterInputStream`.

public void <b>close()</b> throws IOException
---

This method closes this input stream and releases any system resources associated with the stream.
--

## **Class ObjectOutputStream**

To write data to a socket we must firstly get an OutputStream from the socket using method `getOutputStream`. However again class OutputStream offers only methods that permit the writing of bytes to a stream, and we must append an ObjectOutputStream onto the OutputStream to enable Integer and String values to be written.

### **1. Constructor.**

public <b>ObjectOutputStream</b> (OutputStream out) throws IOException
--

This method creates an ObjectOutputStream that uses the specified underlying OutputStream. One parameter must be provided:
--

- the specified output stream

### **2. Methods.**

public final void <b>writeObject</b> (Object obj) throws IOException
--

This method writes an int to the underlying output stream. One parameter must be provided:
--

- an int to be written

When there is no more data to be written to a socket the ObjectOutputStream should be closed using method `close`, this method is inherited from class `FilterOutputStream`.

public void <b>close()</b> throws IOException
---

This method closes this output stream and releases any system resources associated with the stream.
---

#### 4. A summary of the interactions between TCPServer.java and TCPClient.java

##### Detailed Algorithm for the Client

```


Create a socket and get a connection to the server
Create an input stream to receive data from the Server
Create an output stream to send data to the Server
Loop
{
    Prompt user to enter index (0 to 3) of document required or 4 to quit
    Send index to server using the output stream
    if index is 4 then exit loop
    wait for server to send a string containing the document (use input stream)
    display the string on screen
}
Close the input stream
Close the output stream
Close the socket

```

TCPClient.java	TCPServer.java
	<ul style="list-style-type: none"> <li>Initialise a server socket and wait for connections.</li> </ul>
<ul style="list-style-type: none"> <li>Initialise a socket and try to get a connection to server</li> </ul>	
	<ul style="list-style-type: none"> <li>Connection request received – create socket to communicate with client and get input and output streams from socket.</li> </ul>
<ul style="list-style-type: none"> <li>Get input and output streams from socket.</li> </ul> loop <ul style="list-style-type: none"> <li>get input from keyboard (0-3 or 4 to quit)</li> <li>Write value to output stream</li> </ul>	
	loop <ul style="list-style-type: none"> <li>Read value from input stream.</li> <li>Display value.</li> <li>if value is 4 exit loop.</li> <li>otherwise retrieve requested document</li> <li>Write document to output stream.</li> </ul>
<ul style="list-style-type: none"> <li>Read String document from input stream.</li> <li>Display document.</li> </ul> Close input and output streams. Close socket.	

## 5. Compiling and executing the programs

To enable testing of TCPClient.java a test server program TCPServer has been provided. Compile and run this file in the normal way. In Eclipse, you can view a separate console for the Client and the Server using the "Display selected console"

icon  - choose from the options in the drop-down list.

If you want to run the Client and Server on different machines you can pair up with another student or you can run the Server on the local machine, then run the Client from your I: drive.

In either case you will need to replace the code in the Client which gets the InetAddress of the LocalHost

```
socket = new Socket(InetAddress.getLocalHost(), 2000);
```

with code which supplies the actual IP address of the Server as in the example below.

```
socket = new Socket(InetAddress.getByName("server IP address"),  
2000);
```

For Example

```
socket = new Socket(InetAddress.getByName("143.117.100.138"),  
2000);
```

## 6. Solution

The solution for this practical will be posted on Queen's Online at 10:00a.m. on Monday of the following week.