Exercise 3: Using Wireshark to understand basic HTTP request/response messages (marked, include in your report)

Question 1: What is the status code and phrase returned from the server to the client browser?<o:p></o:p>

status code: 200

response phrase: OK

Question 2: When was the HTML file that the browser is retrieving last modified at the server? Does the response also contain a DATE header? How are these two fields different? <o:p></o:p>

The last modified time at the server is: Last-Modified:

```
Tue, 23 Sep 2003 05:29:00 GMT\r\n
```

It does contain a DATE header: Date:

```
Tue, 23 Sep 2003 05:29:50 GMT\r\n
```

They are different: The Last-Modified time is the time of the object that is modified last time. The Date indicates the packet sent date.

Question 3: Is the connection established between the browser and the server persistent or non-persistent? How can you infer this?<o:p></o:p>

non-persistent.

```
Keep-Alive: timeout=10, max=100\r\n

Connection: Keep-Alive\r\n
```

The timeout is set, hence it will cut down the connection in a certain time. It is non-persistent.

Question 4: How many bytes of content are being returned to the browser?<o:p></o:p>

```
Content-Length: 73\r\n
```

73 bytes

Question 5: What is the data contained inside the HTTP response packet?<o:p></o:p>

```
Content-Type: text/html; charset=ISO-8859-1\r\n
```

it has type of text/html

Exercise 4: Using Wireshark to understand the HTTP CONDITIONAL GET/response interaction (marked, include in your report) <o:p></o:p>

Question 1: Inspect the contents of the first HTTP GET request from the browser to the server. Do you see an “IF-MODIFIED-SINCE” line in the HTTP GET?<o:p></o:p>

No. There is no “IF-MODIFIED-SINCE” line in the first HTTP GET. But it does occur in the second one.

Question 2: Does the response indicate the last time that the requested file was modified? <o:p></o:p>

```
If-Modified-Since: Tue, 23 Sep 2003 05:35:00 GMT\r\n
```

Yes, it does.

Question 3: Now inspect the contents of the second HTTP GET request from the browser to the server. Do you see an “IF-MODIFIED-SINCE:” and “IF-NONE-MATCH” lines in the HTTP GET? If so, what information is contained in these header lines?**<o:p></o:p>**

```
If-Modified-Since: Tue, 23 Sep 2003 05:35:00 GMT\r\n

If-None-Match: "1bfef-173-8f4ae900"\r\n
```

The first line indicated the time of last modified time. If there is modification since then, this object can still be used. The second line indicate that there is a new version of the object.

Question 4: What is the HTTP status code and phrase returned from the server in response to this second HTTP GET? Did the server explicitly return the contents of the file? Explain.<o:p></o:p>

```
HTTP/1.1 304 Not Modified\r\n
```

status code: 304

phrase: Not Modified

The server does not explicitly return the contents of the file because 304 status means the object is old and hence it does not have any content body.

Question 5: What is the value of the Etag field in the 2nd response message and how it is used? Has this value changed since the 1st response message was received?

First response message:

```
If-None-Match: "1bfef-173-8f4ae900"\r\n
```

Second:

```
ETag: "1bfef-173-8f4ae900"\r\n
```

It is used to pass by the message to indicate that it has been modified.

(*) Exercise 5: Ping Client

```
import java.io.BufferedReader;

import java.io.ByteArrayInputStream;

import java.io.IOException;

import java.io.InputStreamReader;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.util.Date;



public class PingClient {

        private static final int TIME_OUT = 1000; // milliseconds
```

```
	public static void main(String args[]) throws Exception {
		if (args.length < 2) {
			System.out.println("Required arguements: hostName/hostAddress,
portNumber");
			return;
		}

		// The port number to listen to.
		int port = Integer.parseInt(args[1]);

		// The server to connect to.
		InetAddress server;
		//pass in localhost or other address
		server = InetAddress.getByName(args[0]);

		// Create a datagram socket for sending and receiving UDP
		// packets through the port specified on the command line.
		DatagramSocket socket = new DatagramSocket();

		int sequence_number;

		long minRtt = 0;
		long maxRtt = 0;
		long averageDelay = 0;

		for (sequence_number = 0; sequence_number < 10; sequence_number++) {
			//timestamp
			Date currentTime = new Date();
			long msSend = currentTime.getTime();

			String str = "PING " + sequence_number + " " + msSend + "\r\n";
```

```
                    byte[] buffer = str.getBytes();

                    // Create a Ping datagram to the specified server
                    DatagramPacket ping = new DatagramPacket(buffer, buffer.length, server,
port);

                    // Send the Ping data packet to the specified server
                    socket.send(ping);

                    // Try to receive the packet
                    // Fail when timeout
                    try {
                            // Set the timeout to 1000ms = 1 second specified
                            socket.setSoTimeout(TIME_OUT);

                            // Create a datagram packet to hold incoming UDP packet.
                            DatagramPacket response = new DatagramPacket(new byte[1024],
1024);

                            // Try to receive the response from the server
                            socket.receive(response);

                            // Timestamp for the receive time
                            currentTime = new Date();
                            long msReceived = currentTime.getTime();

                            long delay = msReceived - msSend;


                            if (sequence_number == 0) {
```

```
                                minRtt = delay;
                                maxRtt = delay;
                        }

                // Calculate minimum delay and maximum delay.
                if (delay < minRtt)
                   minRtt = delay;
                if (delay > maxRtt)
                   maxRtt = delay;

                // Calculate average delay.
                averageDelay += delay / (sequence_number + 1);



                // Print the packet and the delay
                                printData(response, sequence_number, delay);
                   } catch (IOException e) {
                           //This is the case where the packet is lost
                           // Print to indicates which packet is lost
                           System.out.println("Timeout for packet " + sequence_number);
                   }
           }
           System.out.println("min rtt = " + minRtt + " ms" +
                                           ", max rtt = " + maxRtt + " ms" +
                                           ", average rtt = " + averageDelay + " ms");
           socket.close();
   }

   /*
    * Print ping data to the standard output stream.
```

```
     * slightly changed from PingServer
     */
    private static void printData(DatagramPacket request, int sequence, long delayTime)
throws Exception{
        // Obtain references to the packet's array of bytes.
        byte[] buf = request.getData();

        // Wrap the bytes in a byte array input stream,
        // so that you can read the data as a stream of bytes.
        ByteArrayInputStream bais = new ByteArrayInputStream(buf);

        // Wrap the byte array output stream in an input stream reader,
        // so you can read the data as a stream of characters.
        InputStreamReader isr = new InputStreamReader(bais);

        // Wrap the input stream reader in a buffered reader,
        // so you can read the character data a line at a time.
        // (A line is a sequence of chars terminated by any combination of \r and \n.)
        BufferedReader br = new BufferedReader(isr);

        // The message data is contained in a single line, so read this line.
        String line = br.readLine();

        // Print host address and data received from it.
        System.out.println(
        "ping to " +
        request.getAddress().getHostAddress() + ": " +
        new String(line) + "\n" +
        "seq = " + sequence +
        ", rtt = " + delayTime + " ms");
    }
```

```
}
```