William Kenji Kiplinger

https://github.com/Kenjum/CS380-EX2/

______________________________________Ex2Client.java____

```
import java.net.Socket;

import java.nio.ByteBuffer;

import java.io.InputStream;

import java.io.OutputStream;

import java.util.zip.CRC32;

import java.util.zip.Checksum;

import javax.xml.bind.DatatypeConverter;


public final class Ex2Client

{

  public static void main(String[] args) throws Exception

  {

			try(Socket socket = new Socket("18.221.102.182", 38102))

			{

				System.out.println("Connected to server.");


				InputStream is = socket.getInputStream();

				OutputStream os = socket.getOutputStream();


				//Normally it would be 100, but since we are essentially getting Hex values

				//of two that need to be combined, we have to double it to 200. This will also
```

```
//make it easier to access the information we want. It will all be in one place.
int[] message = new int[200];


//The getFromServer takes in what is sent from the server.
int index = 0;
int getFromServer = 0;
while(index < message.length)
{
	getFromServer = is.read();
	message[index] = getFromServer;
	index++;
}


//This is where we combine the separate decimal values that we got and turn them into
//a single Hex. Since we're dealing with Hex, we have to start manipulating bits.
//We move the first received part over 4 spaces because Hex values occupy 4 spaces.
//We then add the second received part to the tail so the first 4 bits is the first
//half and the second received part is the second 4 bits. ex. 0xAB
//We also can't forget to cast it to byte (int to byte for byte array).
byte[] messageProper = new byte[100];
index = 0;
for(int i = 0; i < message.length; i = i + 2)
{
	messageProper[index] = (byte) ((message[i] << 4) ^ (message[i+1]));
	index++;
}


//We take the messageProper that has combined 2 parts. This converter turns the combined
//message into Hex.
String messageHex = DatatypeConverter.printHexBinary(messageProper);
System.out.println("Received bytes:");


//This simply prints and presents all the values in the messageHex.
for(int i = 0; i < 200; i = i + 20)
	System.out.println("  " + messageHex.substring(i, i+20));


//update(byte[] b, int off, int len) updates the CRC-32 checksum with the specified array
//of bytes. The error code is generated for the 100 bytes. The ByteBuffer allows for
//fast low-level I/O. crcCheckByte uses ByteBuffer.allocate(4).putInt(crcCheck).array();
//to allow storage of a buffer in a byte array 4 bytes large. It's then converted to
//Hex so it can be properly output.
Checksum checkSum = new CRC32();
checkSum.update(messageProper, 0, messageProper.length);
int crcCheck = (int) checkSum.getValue();
byte[] crcByte = ByteBuffer.allocate(4).putInt(crcCheck).array();


String crcS = DatatypeConverter.printHexBinary(crcByte);
System.out.println("Generated CRC32: "+crcS+".");


//This sends our 100 byte generated error code.
```

```
            os.write(crcByte);


            //This receives from the server whether or not we correctly interpreted the information.
            int serverResponse = is.read();
            if(serverResponse == 1)
                System.out.println("Response good.");
            else
                System.out.println("Response bad.");


            System.out.println("Disconnected from server.");
        }
    }
}
```