**LAB NO:02**

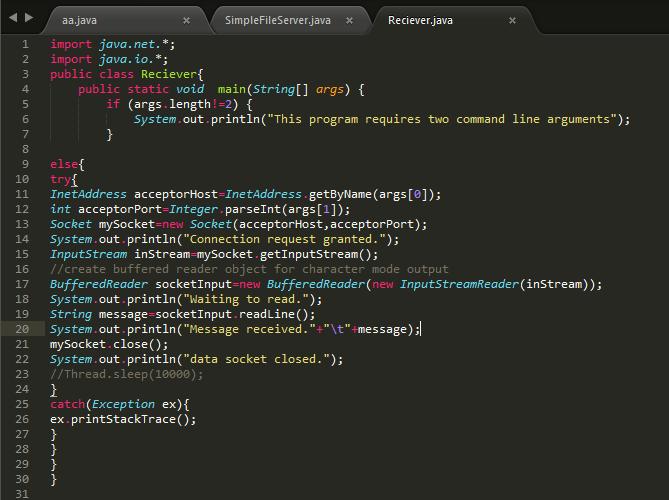
OBJECTIVE:To become familiar with the stream sockets.

**TASKS:**

* Compile and run the above code. Start the acceptor first and then the requestor with appropriate command line arguments. Describe and explain the output.

**CODE:**

**CLIENT**



SERVER:

**SERVER CLASS**

import java.net.\*;

import java.io.\*;

public class ConnectionAcceptor {

//Two command line arguments are needed

// port number of the server socket and second is the message to send

public static void main(String[] args){

if(args.length!=2){

System.out.println("This program requires two command line arguments");

}else{

try{

int portNo=Integer.parseInt(args[0]);

String message=args[1];

ServerSocket connectionSocket=new ServerSocket(portNo);

System.out.println("now ready to accept a connection");

Socket dataSocket=connectionSocket.accept();

System.out.println("Connection Accepted");

OutputStream outStream=dataSocket.getOutputStream();

//create a print writer for character mode output

PrintWriter socketOutput=new PrintWriter(new OutputStreamWriter(outStream));

//write a message into the data stream

**Thread.sleep(3000); //here we make some changes in program**

socketOutput.println(message);

//the ensuing flush method ensures that data is written into the data socket before the socket is closed.

socketOutput.flush();

System.out.println("message sent");

dataSocket.close();

System.out.println("data socket closed.");

connectionSocket.close();

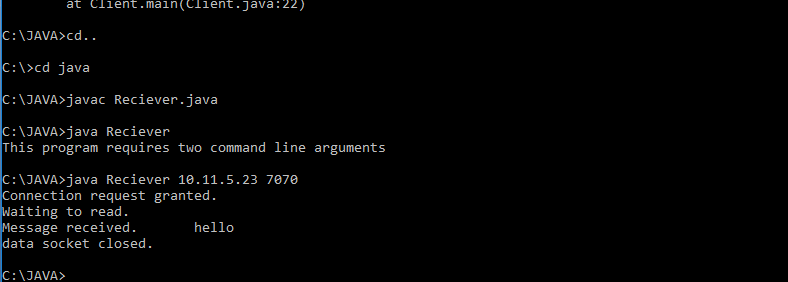
System.out.println("connection socket closed.");

Thread.sleep(10000);

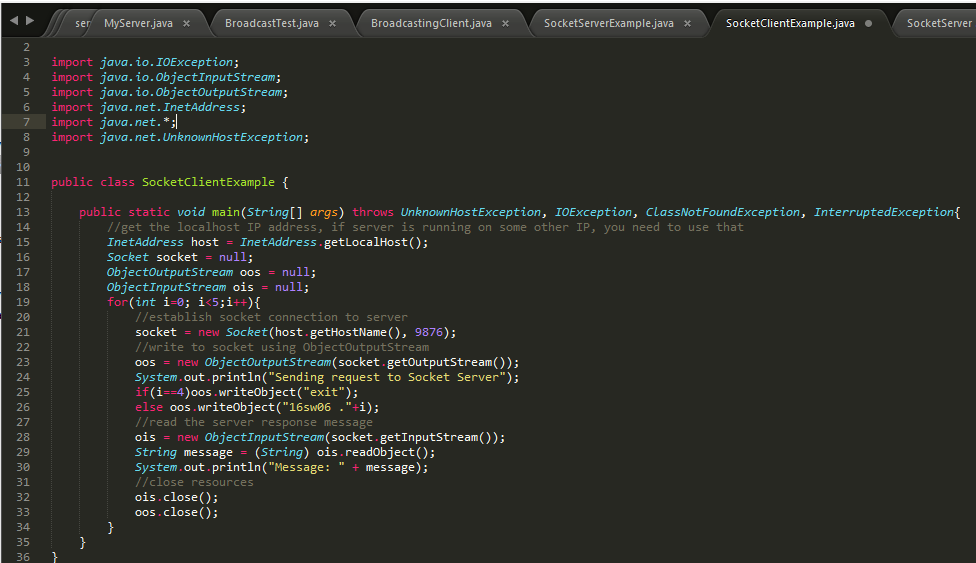
}catch(Exception ex){ ex.printStackTrace();

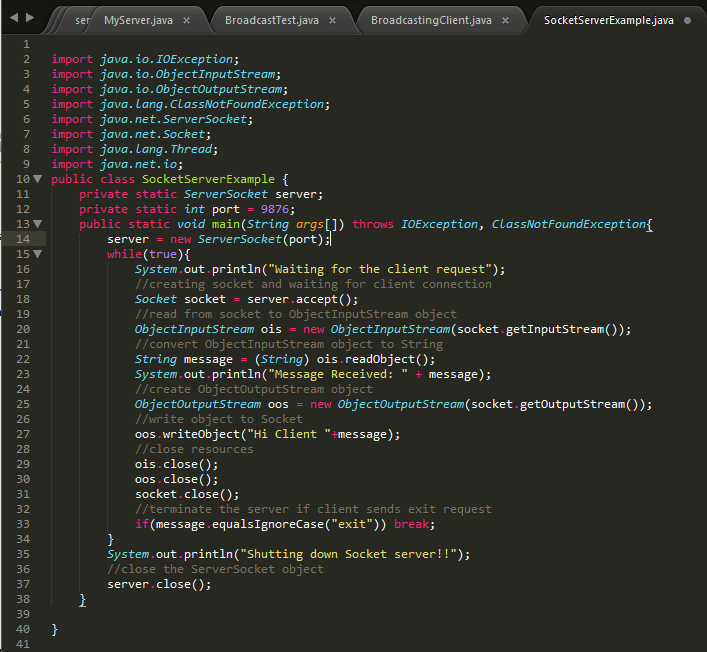
} } } }

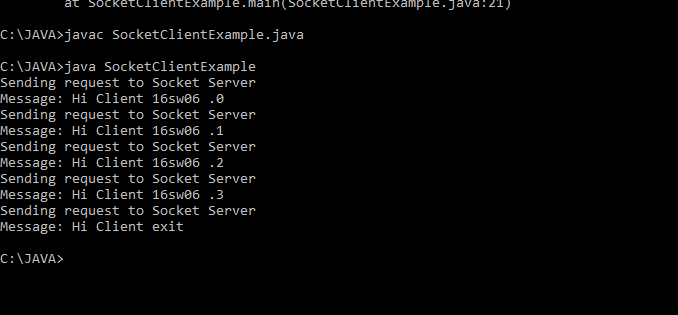
OUTPUT:

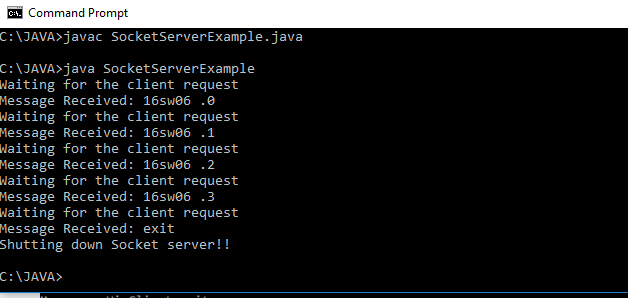


* Now run the code again, but reverse the order of program’s execution. Start the requestor first and then the acceptor. Describe and explain the outcome.

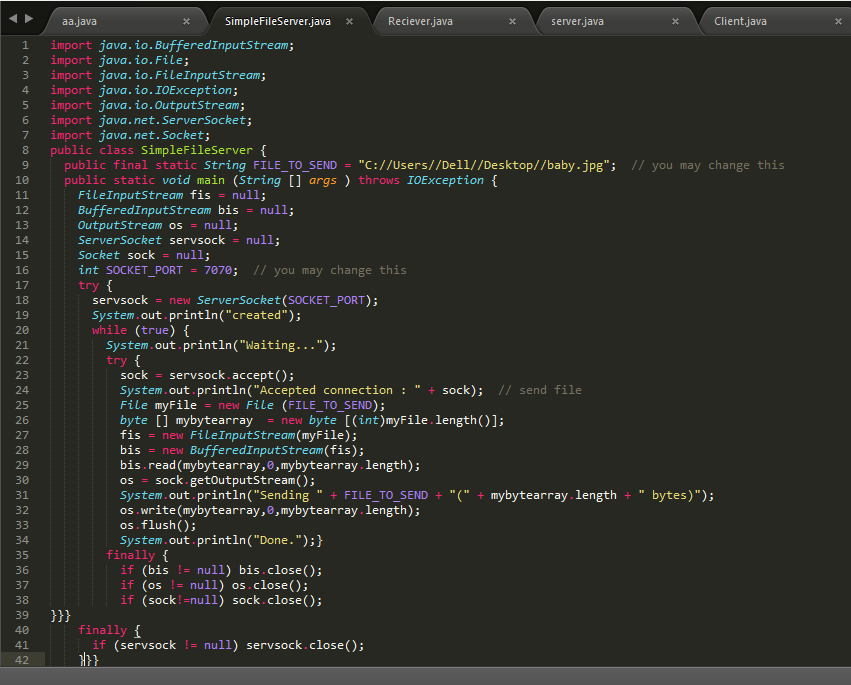
CLIENT: 

SERVER: 

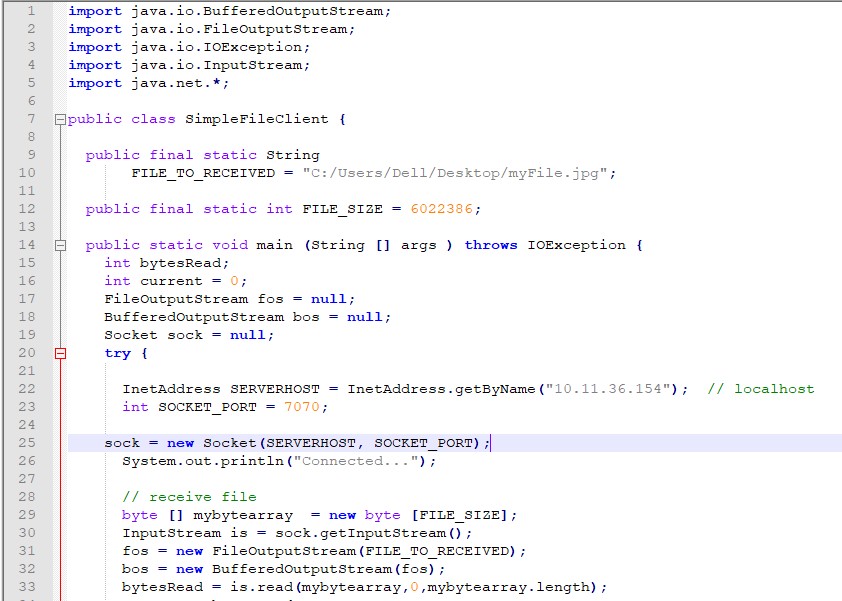


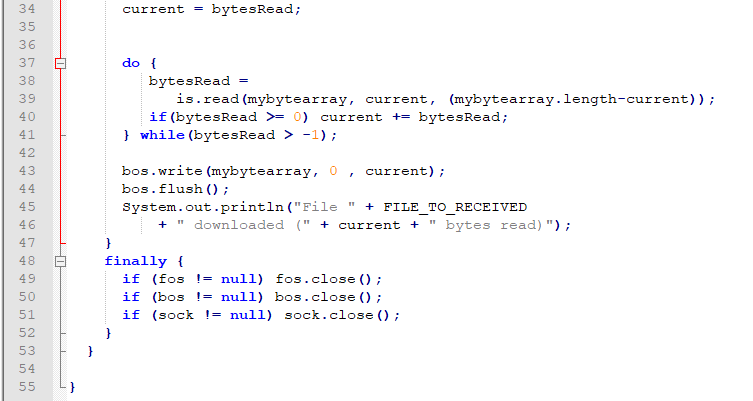


CODE FOR SENDING FILES FROM SERVER:



CODE FOR RECEIVINGFILES FROM SERVER (CLIENT CLASS):

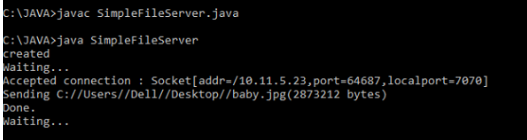




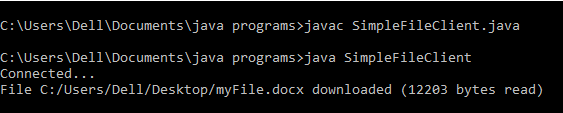
**OUTPUT FOR SERVER:**

**The DOCX File sending to the client**

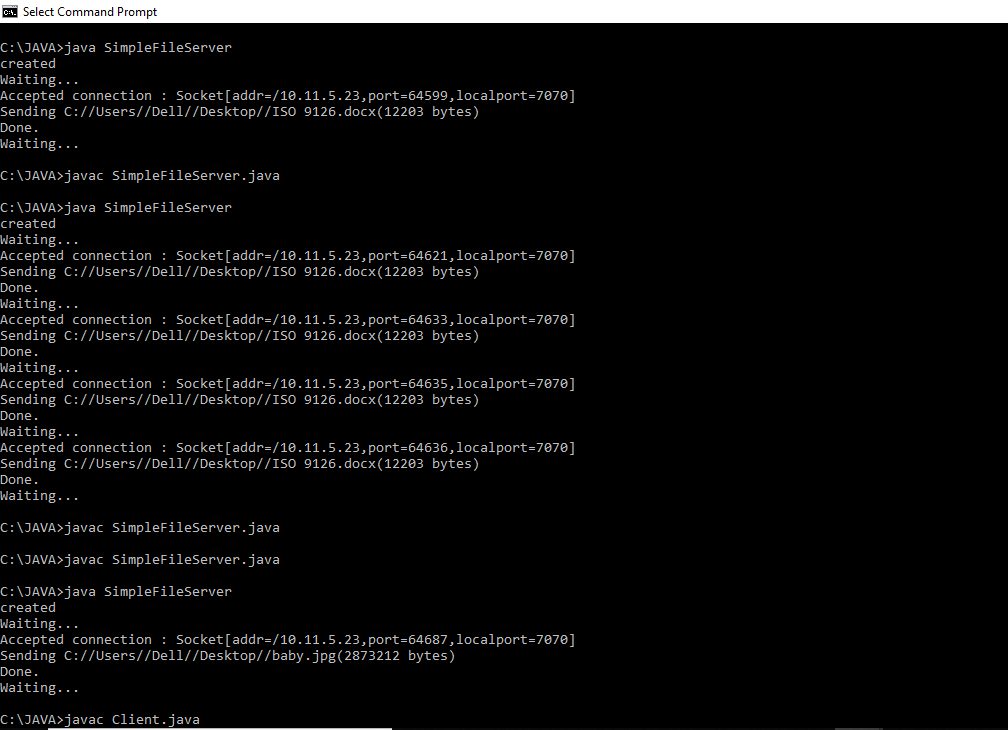
**The JPG File sending to the client**



**OUTPUT FOR CLIENT:**



OUTPUT OF SERVER:



* Add a time delay of 5 seconds in the ConnectionAcceptor process just before the message is written to the socket, then run the program. This will show you the blocking at the receiver. Show a trace of the output of the processes.

CLIENT:

import java.io.IOException;

import java.net.InetAddress;

import java.net.InetSocketAddress;

import java.net.Socket;

import java.net.SocketAddress;

import java.net.UnknownHostException;

public class TestClientSocket {

public static void main(String[] args) {

try {

InetAddress addr = InetAddress.getByName("192.168.1.14");

int port = 9090;

SocketAddress sockaddr = new InetSocketAddress(addr, port);

// Creates an unconnected socket

Socket socket = new Socket();

int timeout = 5000; // 5000 millis = 5 seconds

// Connects this socket to the server with a specified timeout value

// If timeout occurs, SocketTimeoutException is thrown

socket.connect(sockaddr, timeout);

System.out.println("Socket connected..." + socket);

}

catch (UnknownHostException e) {

System.out.println("Host not found: " + e.getMessage());

}

catch (IOException ioe) {

System.out.println("I/O Error " + ioe.getMessage());

} }}

SERVER:

import java.net.\*;

import java.io.\*;

import java.lang.Thread;

public class ConnectionAcceptor {

//Two command line arguments are needed

// port number of the server socket and second is the message to send

public static void main(String[] args){

if(args.length!=2){

System.out.println("This program requires two command line arguments");

}else{

try{

int portNo=Integer.parseInt(args[0]);

String message=args[1];

int timeout=5000;

ServerSocket connectionSocket=new ServerSocket(portNo,timeout);

System.out.println("now ready to accept a connection");

Socket dataSocket=connectionSocket.accept();

System.out.println("Connection Accepted");

OutputStream outStream=dataSocket.getOutputStream();

//create a print writer for character mode output

PrintWriter socketOutput=new PrintWriter(new OutputStreamWriter(outStream));

//write a message into the data stream

socketOutput.println(message);

//the ensuing flush method ensures that data is written into the data socket before the socket is closed.

socketOutput.flush();

System.out.println("message sent");

dataSocket.close();

System.out.println("data socket closed.");

connectionSocket.close();

System.out.println("connection socket closed.");

Thread.sleep(1000);

}catch(Exception ex){ ex.printStackTrace();

} } } }

