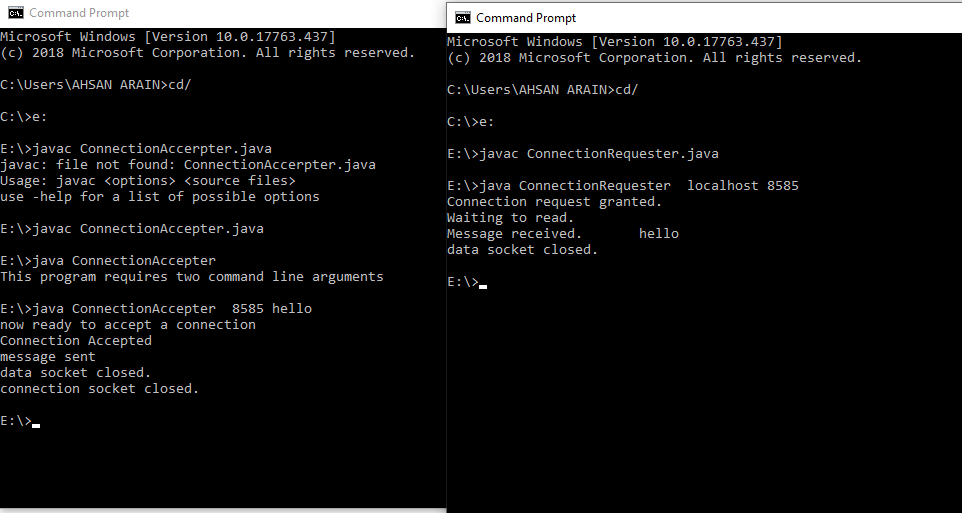
**Lab#02**

**Object: To learn creation of stream sockets.**

**Task#01: Compile and run the above code. Start the acceptor first and then the requestor with appropriate command line arguments. Describe and explain the output.**

****

**Task#02: 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.**

**ConnectionAcceptor.java**

**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**

**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();**

**} } } }**

**ConnectionRequestor.java**

**import java.net.\*;**

**import java.io.\*;**

**//this application requests a connection and sends a message**

**// using the stream mode socket.**

**public class ConnectionRequestor {**

**public static void main(String[] args){**

**if(args.length!=2){**

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

**// the arguments are**

**//host name of connection acceptor and port number of connection acceptor**

**}**

**else{**

**try{**

**InetAddress acceptorHost=InetAddress.getByName(args[0]);**

**int acceptorPort=Integer.parseInt(args[1]);**

**Socket mySocket=new Socket(acceptorHost,acceptorPort);**

**System.out.println("Connection request granted.");**

**InputStream inStream=mySocket.getInputStream();**

**//create buffered reader object for character mode output**

**BufferedReader socketInput=new BufferedReader(new InputStreamReader(inStream));**

**System.out.println("Waiting to read.");**

**String message=socketInput.readLine();**

**System.out.println("Message received."+"\t"+message);**

**mySocket.close();**

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

**Thread.sleep(10000);**

**}catch(Exception ex){**

**ex.printStackTrace();}**

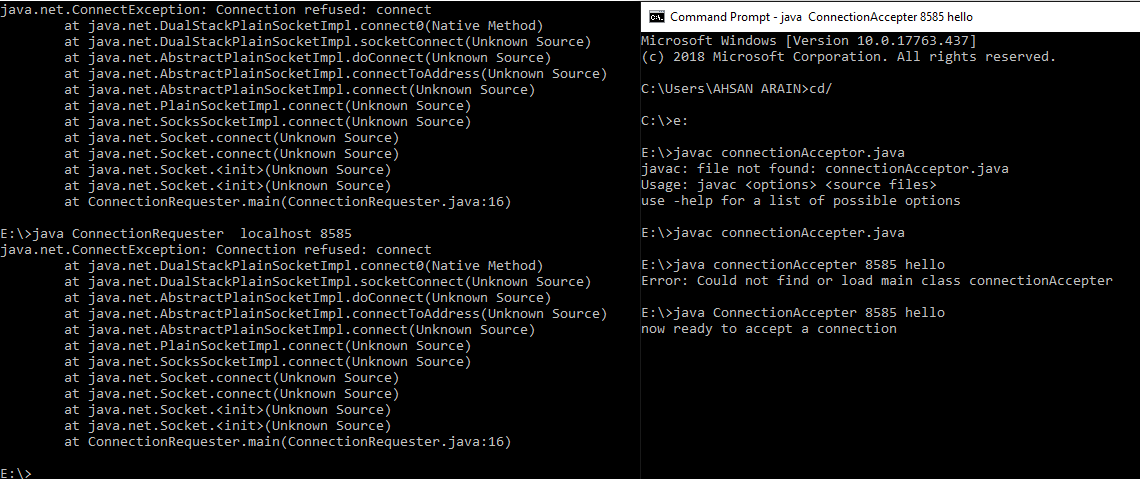
**}**

**}**

**}**

**OUTPUT :**

**In order make connection, Connection Request needs a port no that is not pre occupied. Connection Acceptor allocates the port no to Connection Acceptor. Hence Connection Acceptor is not running so Connection Request can’t make connection and “Connection refused” exception has occurred.**



**Task#03: 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.**

***TASK2Acceptor.java***

***import java.net.\*;***

***import java.io.\*;***

***public class TASK2Acceptor {***

***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(5000);***

***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(5000);***

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

***} } } }***

***TASK2 Requestor.java***

***import java.net.\*;***

***import java.io.\*;***

***//this application requests a connection and sends a message***

***// using the stream mode socket.***

***public class TASK2Requestor {***

***public static void main(String[] args){***

***if(args.length!=2){***

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

***// the arguments are***

***//host name of connection acceptor and port number of connection acceptor***

***}***

***else{***

***try{***

***InetAddress acceptorHost=InetAddress.getByName(args[0]);***

***int acceptorPort=Integer.parseInt(args[1]);***

***Socket mySocket=new Socket(acceptorHost,acceptorPort);***

***System.out.println("Connection request granted.");***

***InputStream inStream=mySocket.getInputStream();***

***//create buffered reader object for character mode output***

***BufferedReader socketInput=new BufferedReader(new InputStreamReader(inStream));***

***System.out.println("Waiting to read.");***

***String message=socketInput.readLine();***

***System.out.println("Message received."+"\t"+message);***

***mySocket.close();***

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

***Thread.sleep(5000);***

***}catch(Exception ex){***

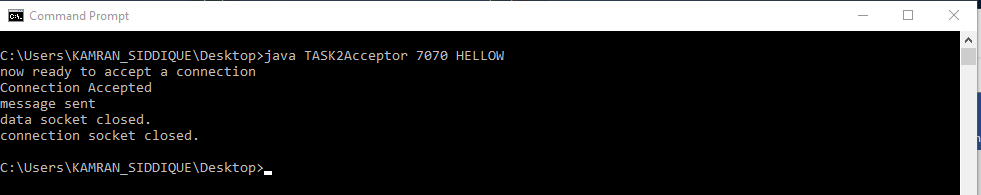
***ex.printStackTrace();}***

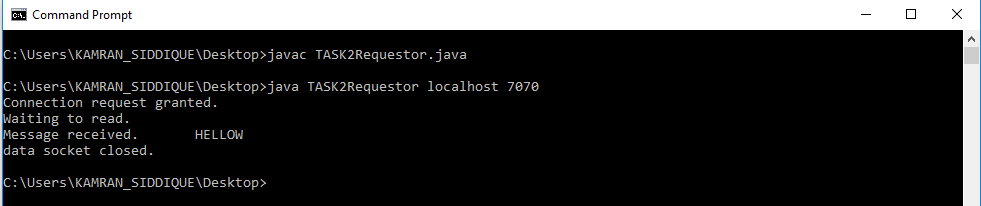
***}***

***}***

***}***

***OUTPUT :***

**

**

**Task#04: Modify the sample code to include two way communication between the client and the server.**

***TASK3Acceptor.java***

***import java.net.\*;***

***import java.io.\*;***

***import java.util.Scanner;***

***public class TASK3Acceptor {***

***public static InputStream inStream = null;***

***public static Scanner input = new Scanner(System.in);***

***public static PrintWriter socketOutput = null;***

***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]);***

***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();***

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

***InetAddress acceptorHost=InetAddress.getByName("localhost");***

***int acceptorPort=Integer.parseInt("7071");***

***Socket mySocket=new Socket(acceptorHost,acceptorPort);***

***inStream=mySocket.getInputStream();***

***while(true)***

***{***

***TASK3Acceptor.sendMsg();***

***TASK3Acceptor.receiveMsg();***

***}***

***//dataSocket.close();***

***//connectionSocket.close();***

***//mySocket.close();***

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

***}***

***}***

***public static void sendMsg()***

***{***

***try{***

***System.out.print(">>");***

***String message=input.nextLine();***

***socketOutput.println(message);***

***socketOutput.flush();***

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

***}***

***public static void receiveMsg()***

***{***

***try{***

***BufferedReader socketInput=new BufferedReader(new InputStreamReader(inStream));***

***String message=socketInput.readLine();***

***System.out.println("%%"+message);***

***}catch(Exception ex){***

***ex.printStackTrace();}***

***}***

***}***

***TASK3 Requestor.java***

***Import java.net.\*;***

***import java.io.\*;***

***import java.util.Scanner;***

***public class TASK3Requestor {***

***public static PrintWriter socketOutput = null;***

***public static InputStream inStream = null;***

***public static Scanner input = new Scanner(System.in);***

***public static void main(String[] args){***

***if(args.length!=2){***

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

***}***

***else{***

***try{***

***InetAddress acceptorHost=InetAddress.getByName(args[0]);***

***int acceptorPort=Integer.parseInt(args[1]);***

***Socket mySocket=new Socket(acceptorHost,acceptorPort);***

***System.out.println("Connection request granted.");***

***inStream=mySocket.getInputStream();***

***//create buffered reader object for character mode output***

***int portNo=Integer.parseInt("7071");***

***ServerSocket connectionSocket=new ServerSocket(portNo);***

***Socket dataSocket=connectionSocket.accept();***

***OutputStream outStream=dataSocket.getOutputStream();***

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

***while(true)***

***{***

***TASK3Requestor.receiveMsg();***

***TASK3Requestor.sendMsg();***

***}***

***}catch(Exception ex){***

***ex.printStackTrace();}***

***//mySocket.close();***

***//dataSocket.close();***

***//connectionSocket.close();***

***}***

***}***

***public static void receiveMsg()***

***{***

***try{***

***BufferedReader socketInput=new BufferedReader(new InputStreamReader(inStream));***

***String message=socketInput.readLine();***

***System.out.println("%%"+message);***

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

***}catch(Exception ex){***

***ex.printStackTrace();}***

***}***

***public static void sendMsg()***

***{***

***try{***

***System.out.print(">>");***

***String message=input.nextLine();***

***socketOutput.println(message);***

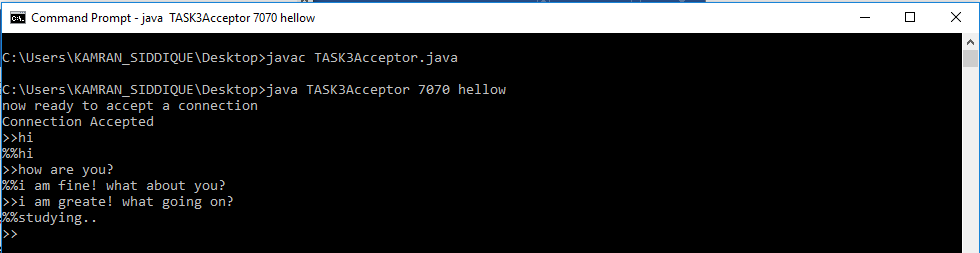
***socketOutput.flush();***

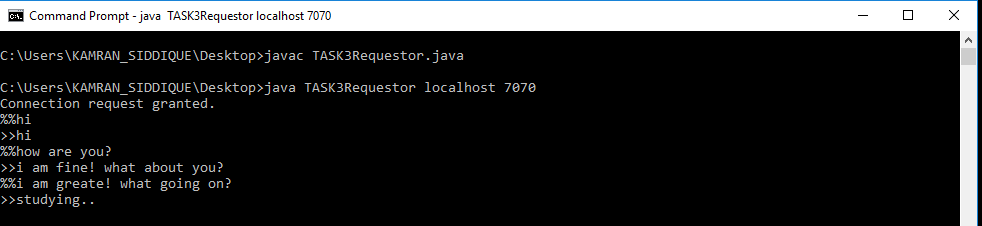
***}catch(Exception ex){***

***ex.printStackTrace();}***

***}***

***}***

**

**

**Task#05: Modify the sample code to send complete files between the client to the server.**

**SimpleFileClient.java**

**import java.io.BufferedOutputStream;**

**import java.io.FileOutputStream;**

**import java.io.IOException;**

**import java.io.InputStream;**

**import java.net.Socket;**

**public class SimpleFileClient {**

**public final static int SOCKET\_PORT = 13267; // you may change this**

**public final static String SERVER = "127.0.0.1"; // localhost**

**public final static String**

**FILE\_TO\_RECEIVED = "EdxA.docx"; // you may change this, I give a**

**// different name because i don't want to**

**// overwrite the one used by server...**

**public final static int FILE\_SIZE = 6022386; // file size temporary hard coded**

**// should bigger than the file to be downloaded**

**public static void main (String [] args ) throws IOException {**

**int bytesRead;**

**int current = 0;**

**FileOutputStream fos = null;**

**BufferedOutputStream bos = null;**

**Socket sock = null;**

**try {**

**sock = new Socket(SERVER, SOCKET\_PORT);**

**System.out.println("Connecting...");**

**// receive file**

**byte [] mybytearray = new byte [FILE\_SIZE];**

**InputStream is = sock.getInputStream();**

**fos = new FileOutputStream(FILE\_TO\_RECEIVED);**

**bos = new BufferedOutputStream(fos);**

**bytesRead = is.read(mybytearray,0,mybytearray.length);**

**current = bytesRead;**

**do {**

**bytesRead =**

**is.read(mybytearray, current, (mybytearray.length-current));**

**if(bytesRead >= 0) current += bytesRead;**

**} while(bytesRead > -1);**

**bos.write(mybytearray, 0 , current);**

**bos.flush();**

**System.out.println("File " + FILE\_TO\_RECEIVED**

**+ " downloaded (" + current + " bytes read)");**

**}**

**finally {**

**if (fos != null) fos.close();**

**if (bos != null) bos.close();**

**if (sock != null) sock.close();**

**}**

**}**

**}**

**SimpleFileServer.java**

**import java.io.BufferedInputStream;**

**import java.io.File;**

**import java.io.FileInputStream;**

**import java.io.IOException;**

**import java.io.OutputStream;**

**import java.net.ServerSocket;**

**import java.net.Socket;**

**public class SimpleFileServer {**

**public final static int SOCKET\_PORT = 13267; // you may change this**

**public final static String FILE\_TO\_SEND = "Edx.docx"; // you may change this**

**public static void main (String [] args ) throws IOException {**

**FileInputStream fis = null;**

**BufferedInputStream bis = null;**

**OutputStream os = null;**

**ServerSocket servsock = null;**

**Socket sock = null;**

**try {**

**servsock = new ServerSocket(SOCKET\_PORT);**

**while (true) {**

**System.out.println("Waiting...");**

**try {**

**sock = servsock.accept();**

**System.out.println("Accepted connection : " + sock);**

**// send file**

**File myFile = new File (FILE\_TO\_SEND);**

**byte [] mybytearray = new byte [(int)myFile.length()];**

**fis = new FileInputStream(myFile);**

**bis = new BufferedInputStream(fis);**

**bis.read(mybytearray,0,mybytearray.length);**

**os = sock.getOutputStream();**

**System.out.println("Sending " + FILE\_TO\_SEND + "(" + mybytearray.length + " bytes)");**

**os.write(mybytearray,0,mybytearray.length);**

**os.flush();**

**System.out.println("Done.");**

**}**

**finally {**

**if (bis != null) bis.close();**

**if (os != null) os.close();**

**if (sock!=null) sock.close();**

**}**

**}**

**}**

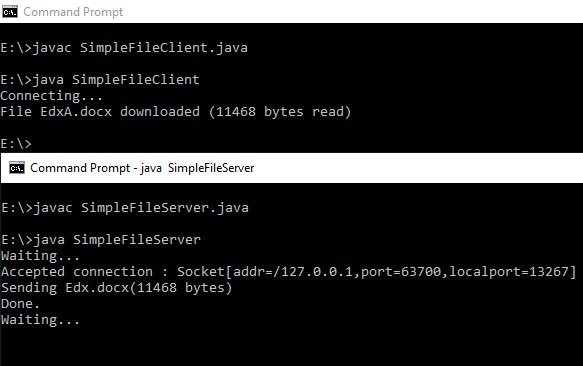
**finally {**

**if (servsock != null) servsock.close();**

**}**

**}**

**}**



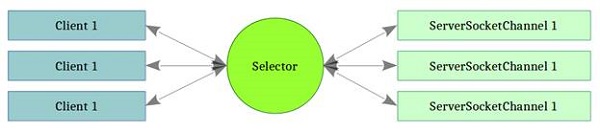
**Task#06: Explore the non-blocking java socket API in the nio package and implement a sample program.**

**Starting**[**JDK 1.4**](https://crunchify.com/where-is-java-installed-on-my-mac-osx-system/)**, NIO was created to allow all**[**Java programmers**](https://crunchify.com/category/java-tutorials/)**to implement very high-speed input/output without having to deal with custom native code. NIO uses java.nio.buffer library compare to**[**simple I/O**](https://crunchify.com/java-file-copy-example-simple-way-to-copy-file-in-java/)**which drains and fills back buffer internally any operating system.**

**In this tutorial we will go over java.nio.channels and java.nio.channels.Selectorlibraries.**

* **channels represent connections to entities that are capable of performing I/O operations, such as files and sockets; defines selectors, for multiplexed, non-blocking I/O operations.**
* **selector may be created by invoking the open method of this class, which will use the system’s default selector provider to create a new selector.**

**How it works:**



**Code:**

**SocketServerExample.java**

**import java.io.IOException;**

**import java.net.InetSocketAddress;**

**import java.net.Socket;**

**import java.net.SocketAddress;**

**import java.nio.ByteBuffer;**

**import java.nio.channels.SelectionKey;**

**import java.nio.channels.Selector;**

**import java.nio.channels.ServerSocketChannel;**

**import java.nio.channels.SocketChannel;**

**import java.util.\*;**

**public class SocketServerExample {**

**private Selector selector;**

**private Map<SocketChannel,List<byte[]>> dataMapper;**

**private InetSocketAddress listenAddress;**

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

**Runnable server = new Runnable() {**

**@Override**

**public void run() {**

**try {**

**new SocketServerExample("localhost", 8090).startServer();**

**} catch (IOException e) {**

**e.printStackTrace();**

**}**

**}**

**};**

**Runnable client = new Runnable() {**

**@Override**

**public void run() {**

**try {**

**new SocketClientExample().startClient();**

**} catch (IOException e) {**

**e.printStackTrace();**

**} catch (InterruptedException e) {**

**e.printStackTrace();**

**}**

**}**

**};**

**new Thread(server).start();**

**new Thread(client, "client-A").start();**

**new Thread(client, "client-B").start();**

**}**

**public SocketServerExample(String address, int port) throws IOException {**

**listenAddress = new InetSocketAddress(address, port);**

**dataMapper = new HashMap<SocketChannel,List<byte[]>>();**

**}**

**// create server channel**

**private void startServer() throws IOException {**

**this.selector = Selector.open();**

**ServerSocketChannel serverChannel = ServerSocketChannel.open();**

**serverChannel.configureBlocking(false);**

**// retrieve server socket and bind to port**

**serverChannel.socket().bind(listenAddress);**

**serverChannel.register(this.selector, SelectionKey.OP\_ACCEPT);**

**System.out.println("Server started...");**

**while (true) {**

**// wait for events**

**this.selector.select();**

**//work on selected keys**

**Iterator<SelectionKey> keys = this.selector.selectedKeys().iterator();**

**while (keys.hasNext()) {**

**SelectionKey key = (SelectionKey) keys.next();**

**// this is necessary to prevent the same key from coming up**

**// again the next time around.**

**keys.remove();**

**if (!key.isValid()) {**

**continue;**

**}**

**if (key.isAcceptable()) {**

**this.accept(key);**

**}**

**else if (key.isReadable()) {**

**this.read(key);**

**}**

**}**

**}**

**}**

**//accept a connection made to this channel's socket**

**private void accept(SelectionKey key) throws IOException {**

**ServerSocketChannel serverChannel = (ServerSocketChannel) key.channel();**

**SocketChannel channel = serverChannel.accept();**

**channel.configureBlocking(false);**

**Socket socket = channel.socket();**

**SocketAddress remoteAddr = socket.getRemoteSocketAddress();**

**System.out.println("Connected to: " + remoteAddr);**

**// register channel with selector for further IO**

**dataMapper.put(channel, new ArrayList<byte[]>());**

**channel.register(this.selector, SelectionKey.OP\_READ);**

**}**

**//read from the socket channel**

**private void read(SelectionKey key) throws IOException {**

**SocketChannel channel = (SocketChannel) key.channel();**

**ByteBuffer buffer = ByteBuffer.allocate(1024);**

**int numRead = -1;**

**numRead = channel.read(buffer);**

**if (numRead == -1) {**

**this.dataMapper.remove(channel);**

**Socket socket = channel.socket();**

**SocketAddress remoteAddr = socket.getRemoteSocketAddress();**

**System.out.println("Connection closed by client: " + remoteAddr);**

**channel.close();**

**key.cancel();**

**return;**

**}**

**byte[] data = new byte[numRead];**

**System.arraycopy(buffer.array(), 0, data, 0, numRead);**

**System.out.println("Got: " + new String(data));**

**}**

**}**

**SocketClientExample.java**

**import java.io.IOException;**

**import java.net.InetSocketAddress;**

**import java.nio.ByteBuffer;**

**import java.nio.channels.SocketChannel;**

**public class SocketClientExample {**

**public void startClient()**

**throws IOException, InterruptedException {**

**InetSocketAddress hostAddress = new InetSocketAddress("localhost", 8090);**

**SocketChannel client = SocketChannel.open(hostAddress);**

**System.out.println("Client... started");**

**String threadName = Thread.currentThread().getName();**

**// Send messages to server**

**String [] messages = new String []**

**{threadName + ": test1",threadName + ": test2",threadName + ": test3"};**

**for (int i = 0; i < messages.length; i++) {**

**byte [] message = new String(messages [i]).getBytes();**

**ByteBuffer buffer = ByteBuffer.wrap(message);**

**client.write(buffer);**

**System.out.println(messages [i]);**

**buffer.clear();**

**Thread.sleep(5000);**

**}**

**client.close();**

**}**

**}**

