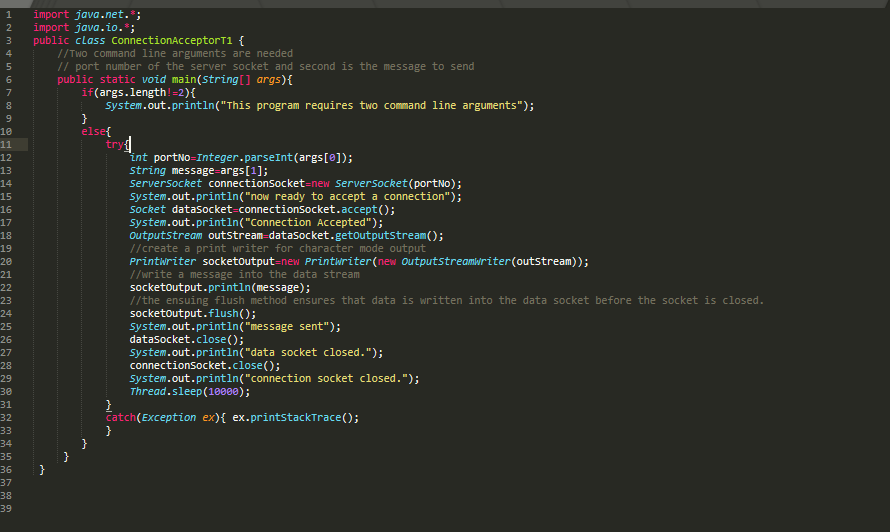
Lab 2

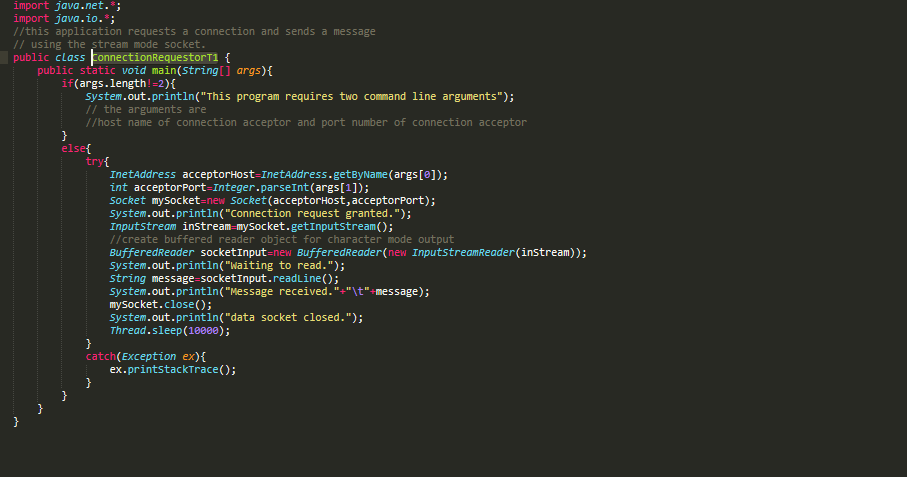
Objective :To become familiar with stream sockect API

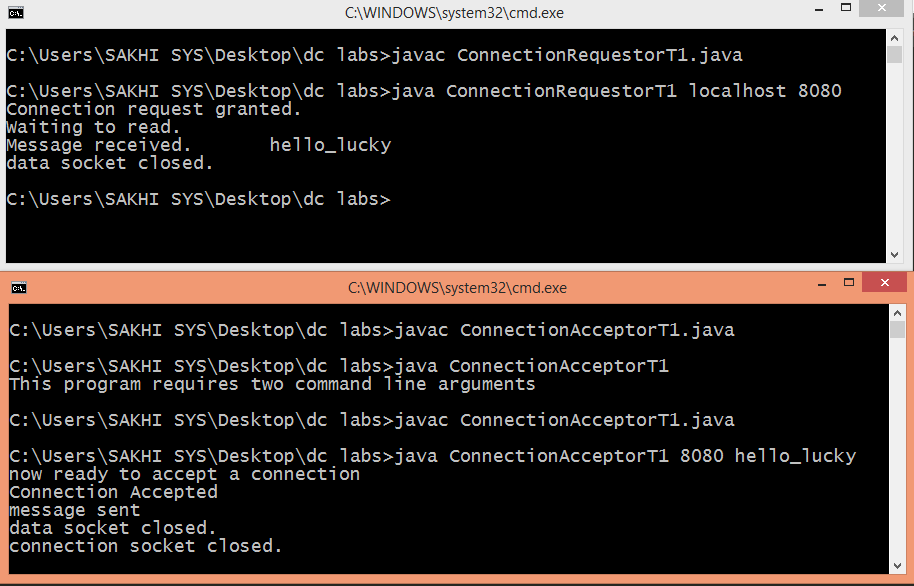
1.Compile and run the above code. Start the acceptor first and then the requestor with appropriate command line arguments. Describe and explain the output.

Connection Acceptor.java

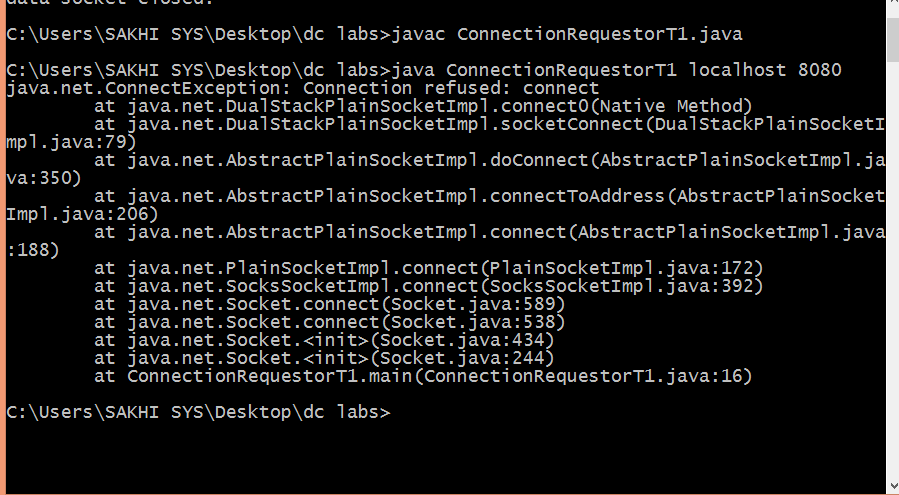


Connection request.java





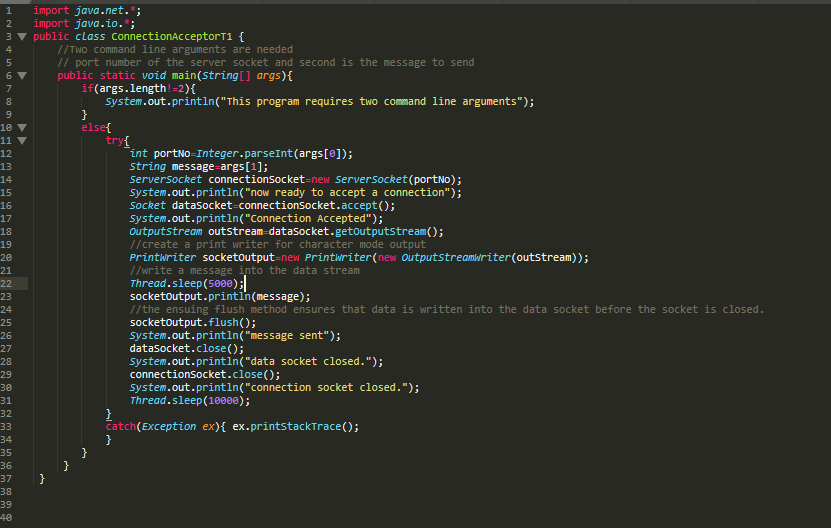
2.Now run the code again, but reverse the order of program’s execution. Start the requestor first and then the acceptor. Describe and explain the outcome.



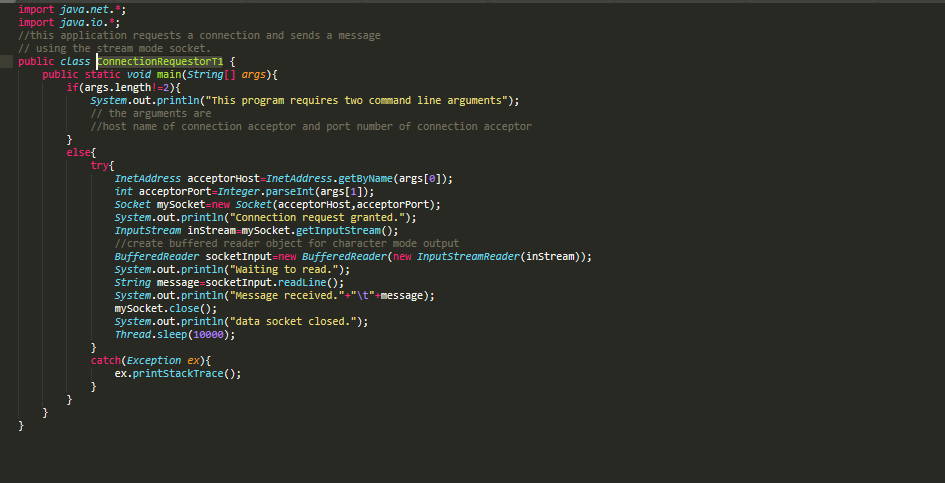
This exception means that there is no service listening on the IP/port you are trying to connect .

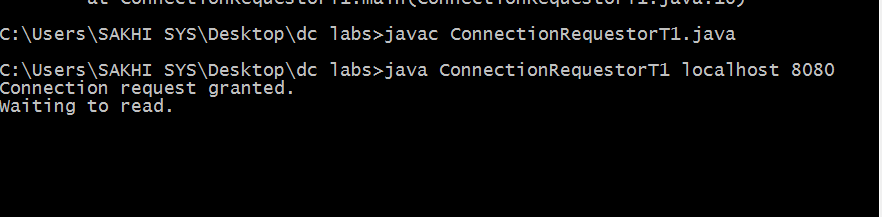
3.Add a time delay of 5 seconds in the ConnectionAcceptor process just before the message is written to the socket, then run the program. This will show you the blocking at the receiver. Show a trace of the output of the processes.

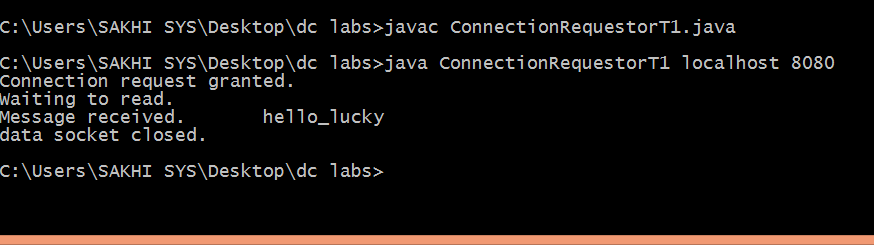
Acceptor.java

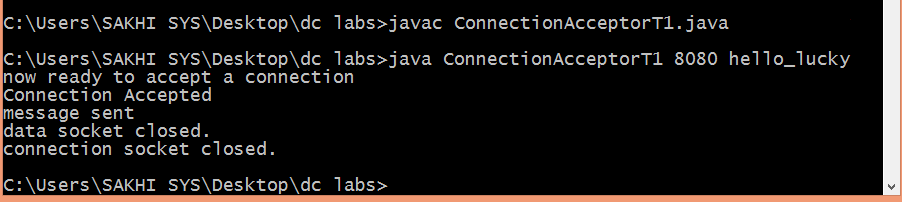


Requestor.java



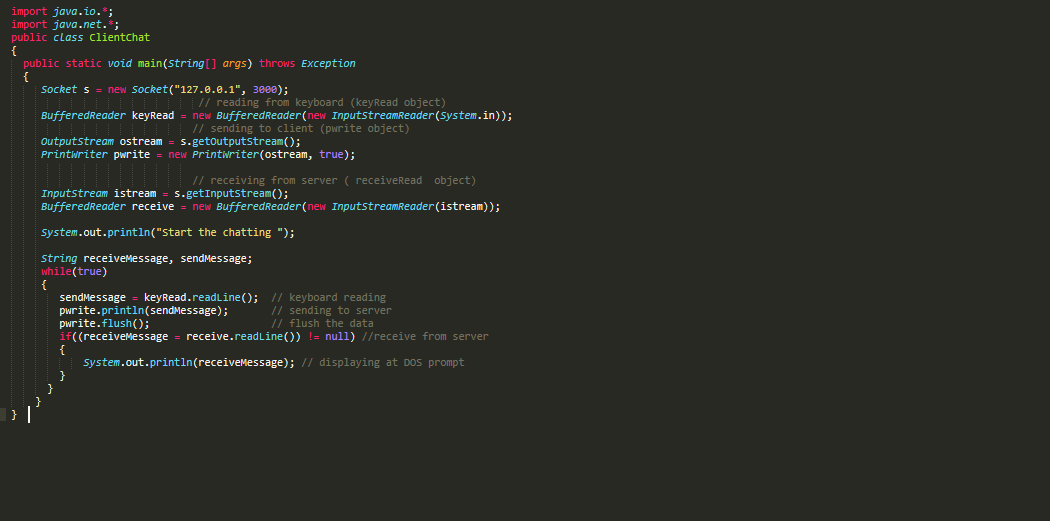




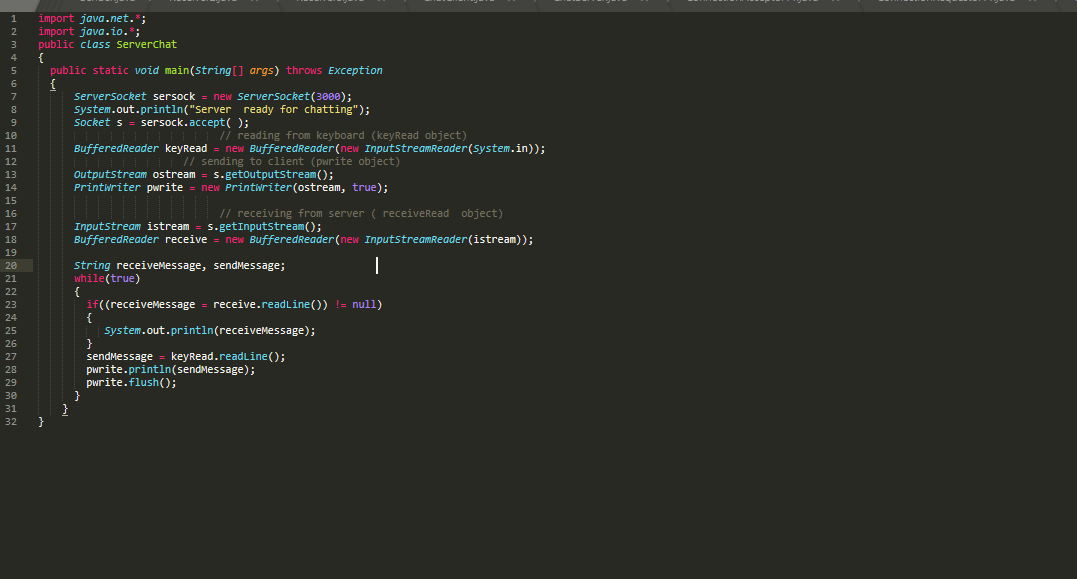


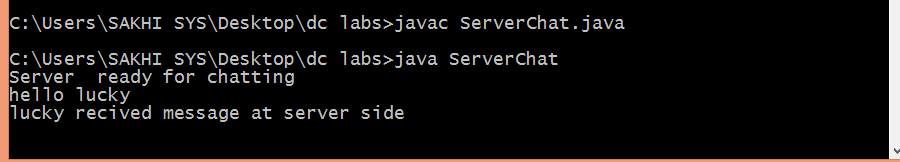
4. Modify the sample code to include two way communication between the client and the server.

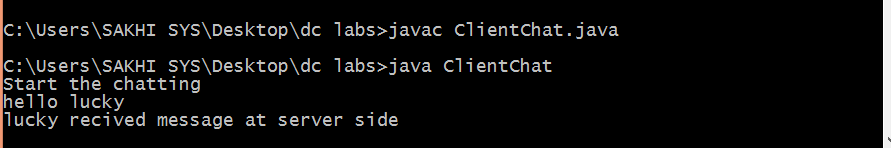
Client.java



Server.java

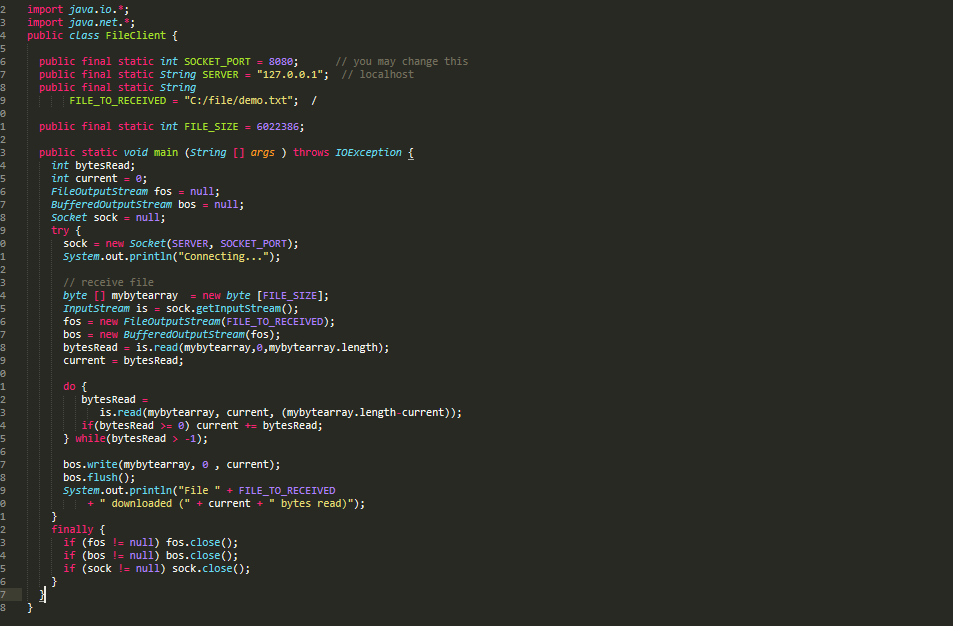




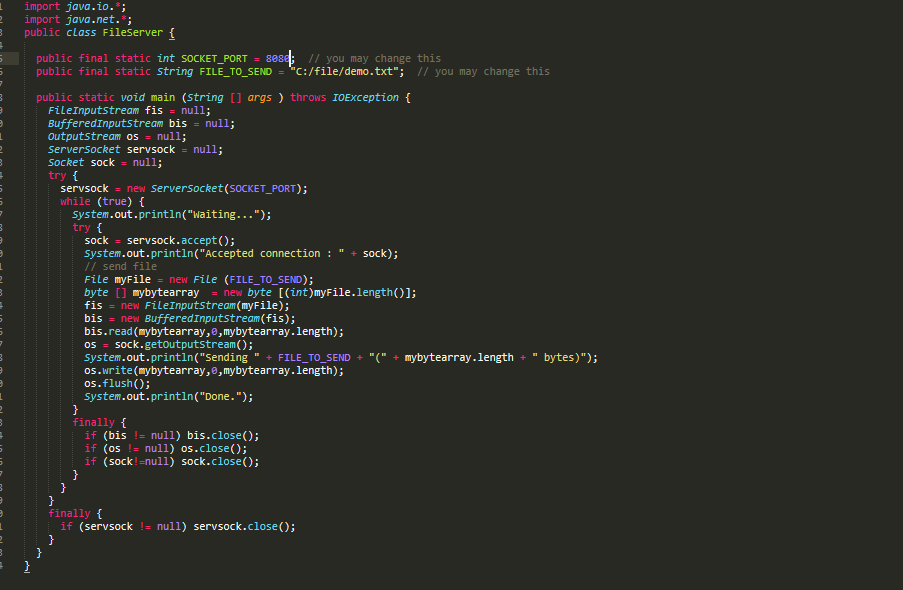


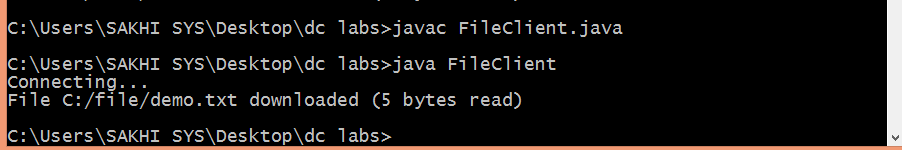
5. Modify the sample code to send complete files between the client to the server.

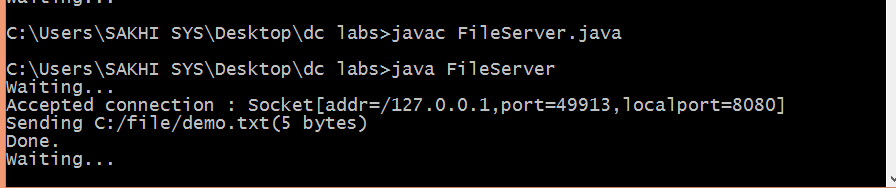
Client.java



Server.java

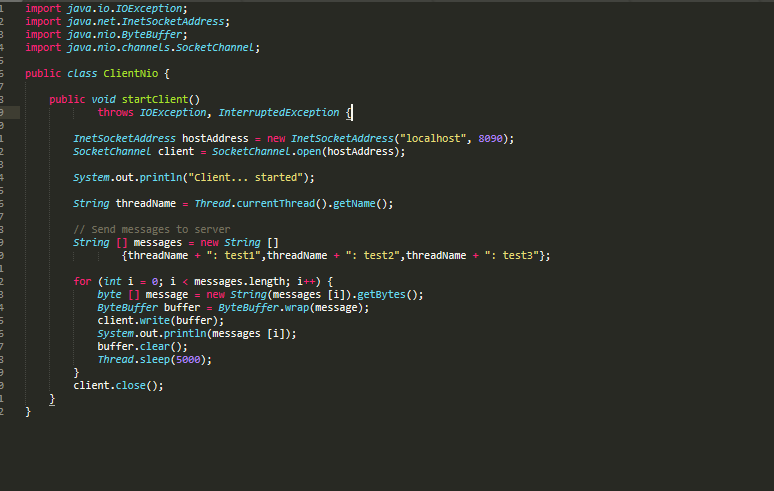






6.Explore the non-blocking java socket API in the **nio** package and implement a sample program.

clientNio.java



ServerNio.java

import java.io.\*;

import java.nio.\*;

import java.net.\*;

import java.util.\*;

public class ServerNio {

private Selector selector;

private Map<SocketChannel,List> dataMapper;

private InetSocketAddress listenAddress;

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

Runnable server = new Runnable() {

@Override

public void run() {

try {

new ServerNio("localhost", 8090).startServer();

} catch (IOException e) {

e.printStackTrace();

}

}

};

Runnable client = new Runnable() {

@Override

public void run() {

try {

new ClientNio().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 ServerNio(String address, int port) throws IOException {

listenAddress = new InetSocketAddress(address, port);

dataMapper = new HashMap<SocketChannel,List>();

}

// 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 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());

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));

}

}

