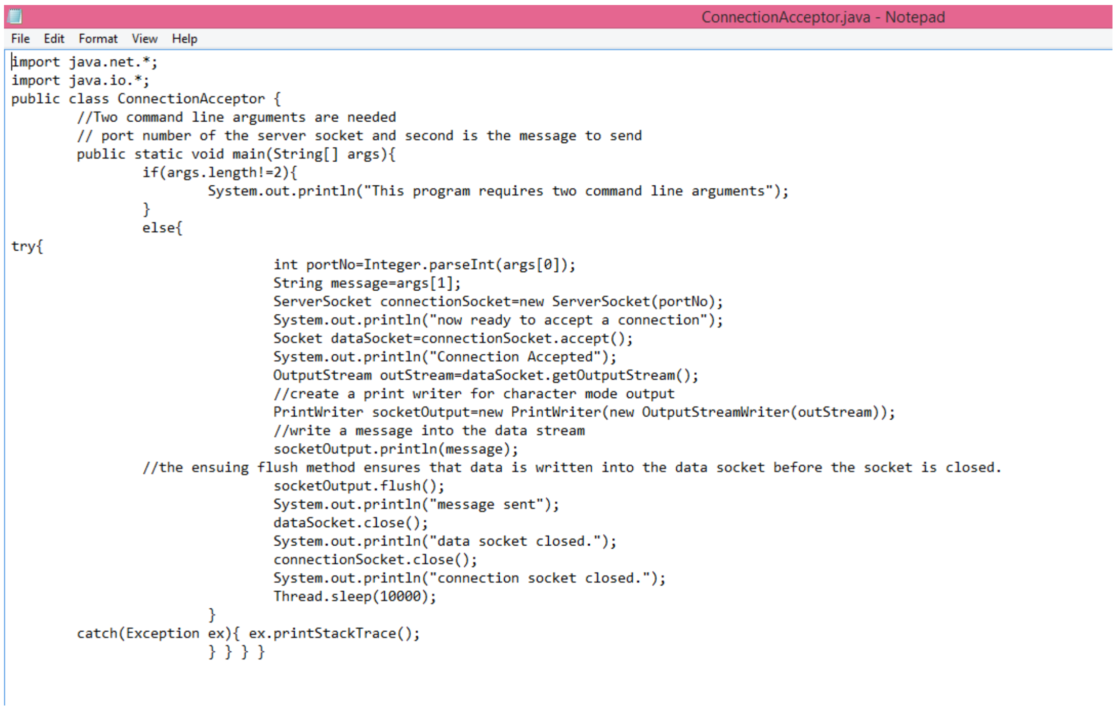
Lab no:2

Objective: To become familiar with Stream Socket API.

Task No: 01

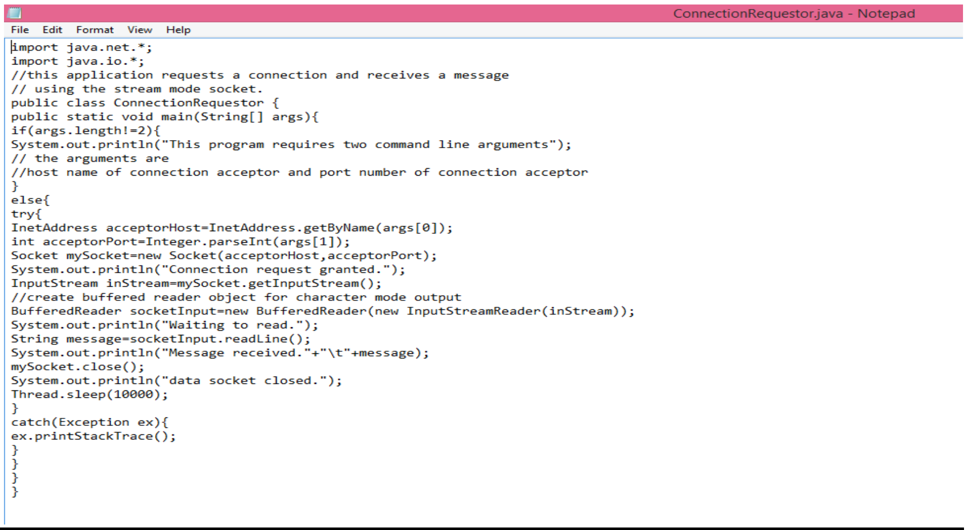
Compile and run the above code. Start the acceptor first and then the requestor with appropriate command line arguments. Describe and explain the output.

Acceptor:

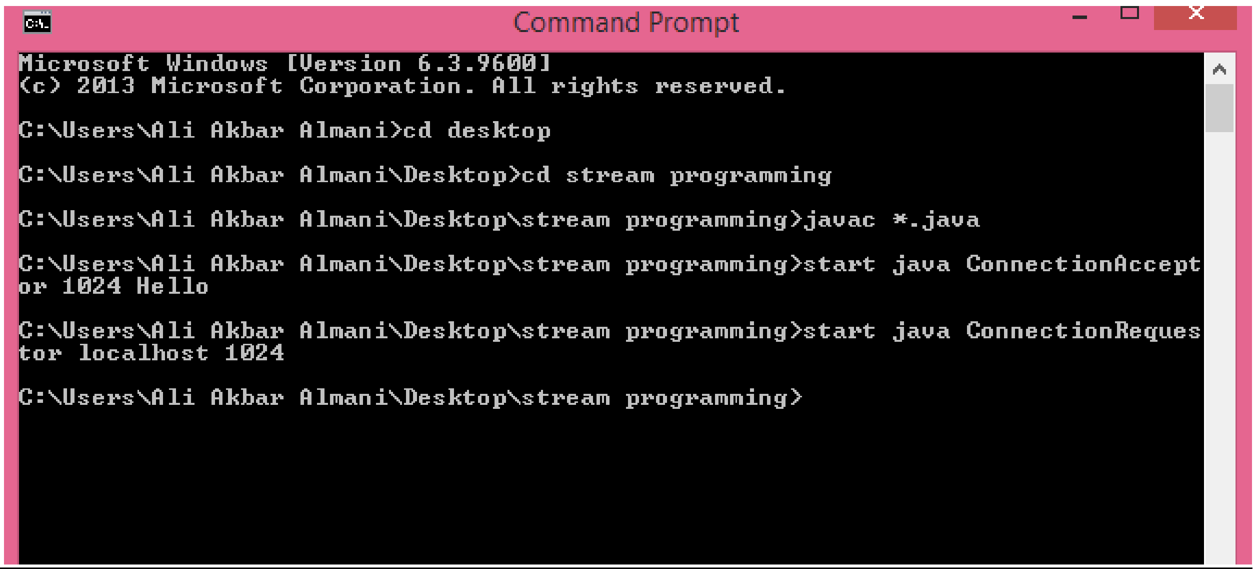


```
import java.net.*;
import java.io.*;
public class ConnectionAcceptor {
        //Two command line arguments are needed
        // port number of the server socket and second is the message to send
        public static void main(String[] args){
                if(args.length!=2){
                        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();
                        } } } }
```

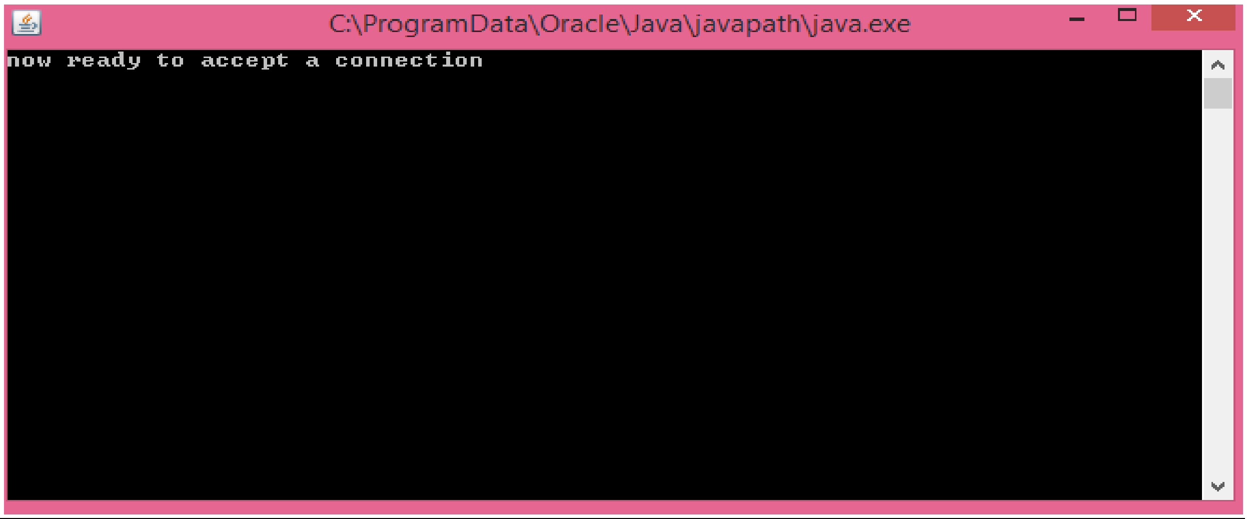
Requestor:


ConnectionRequestor.java - Notepad
File Edit Format View Help

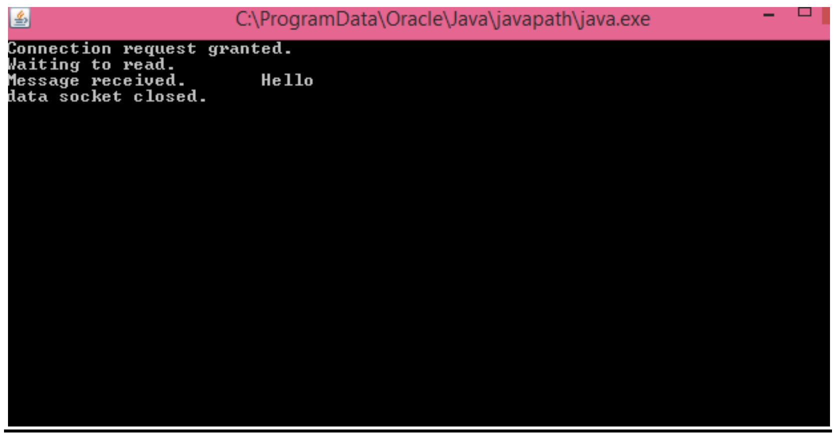
```java
import java.net.*;
import java.io.*;
//this application requests a connection and receives a message
// using the stream mode socket.
public class ConnectionRequestor {
public static void main(String[] args){
if(args.length!=2){
System.out.println("This program requires two command line arguments");
// 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();
}
}
}
}
```



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Ali Akbar Almani>cd desktop

C:\Users\Ali Akbar Almani\Desktop>cd stream programming

C:\Users\Ali Akbar Almani\Desktop\stream programming>javac *.java

C:\Users\Ali Akbar Almani\Desktop\stream programming>start java ConnectionAccept
or 1024 Hello

C:\Users\Ali Akbar Almani\Desktop\stream programming>start java ConnectionReques
tor localhost 1024

C:\Users\Ali Akbar Almani\Desktop\stream programming>
```



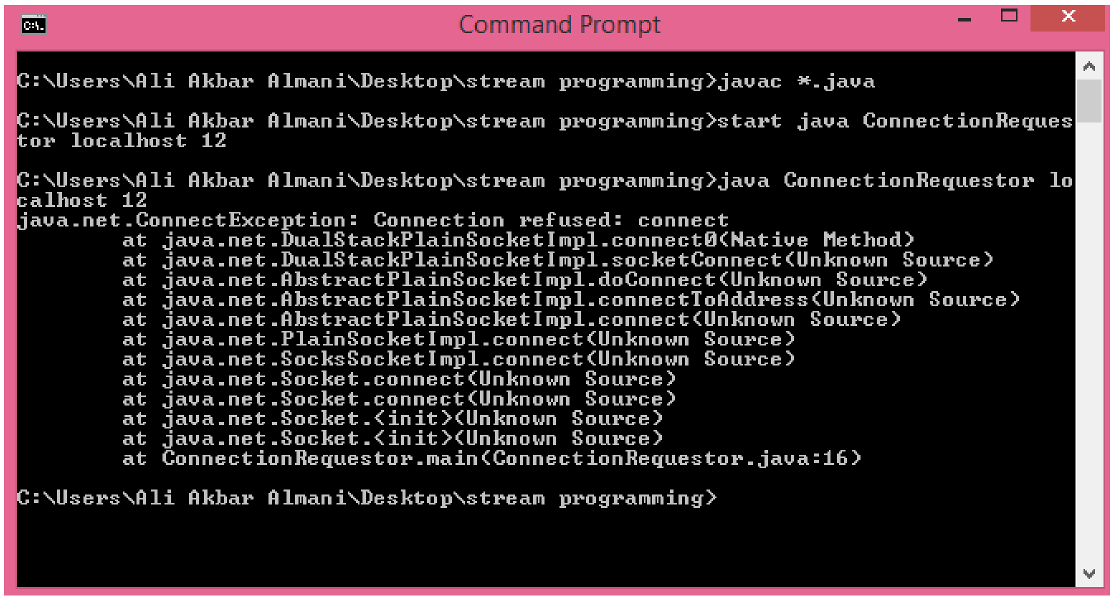
```
now ready to accept a connection
```


C:\ProgramData\Oracle\Java\javapath\java.exe

```
Connection request granted.
Waiting to read.
Message received.       Hello
data socket closed.
```

<u>Task No: 02</u>

<u>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.</u>

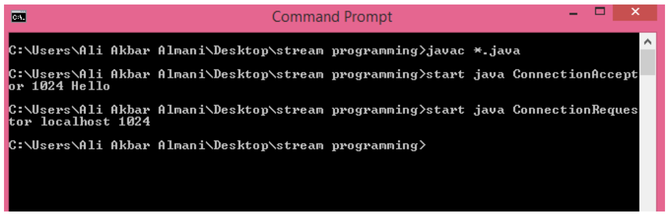

Command Prompt

```
C:\Users\Ali Akbar Almani\Desktop\stream programming>javac *.java

C:\Users\Ali Akbar Almani\Desktop\stream programming>start java ConnectionReques
tor localhost 12

C:\Users\Ali Akbar Almani\Desktop\stream programming>java ConnectionRequestor lo
calhost 12
java.net.ConnectException: Connection refused: connect
        at java.net.DualStackPlainSocketImpl.connect0(Native Method)
        at java.net.DualStackPlainSocketImpl.socketConnect(Unknown Source)
        at java.net.AbstractPlainSocketImpl.doConnect(Unknown Source)
        at java.net.AbstractPlainSocketImpl.connectToAddress(Unknown Source)
        at java.net.AbstractPlainSocketImpl.connect(Unknown Source)
        at java.net.PlainSocketImpl.connect(Unknown Source)
        at java.net.SocksSocketImpl.connect(Unknown Source)
        at java.net.Socket.connect(Unknown Source)
        at java.net.Socket.connect(Unknown Source)
        at java.net.Socket.<init>(Unknown Source)
        at java.net.Socket.<init>(Unknown Source)
        at ConnectionRequestor.main(ConnectionRequestor.java:16)

C:\Users\Ali Akbar Almani\Desktop\stream programming>
```

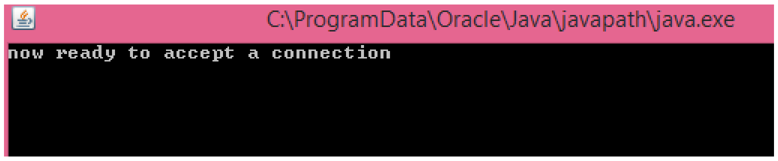
As the exception occurs there is no acceptor to accept the requestors connection.

<u>Task No :03</u>

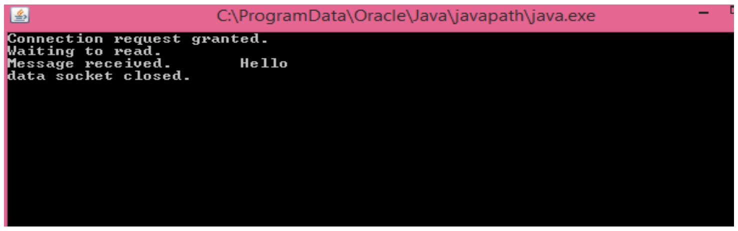
<u>Add a time delay of 5 seconds in the ConnectionAcceptor process just before the message is written to the socket, then run the program. This will show you the blocking at the receiver. Show a trace of the output of the processes.</u>


Command Prompt

```
C:\Users\Ali Akbar Almani\Desktop\stream programming>javac *.java

C:\Users\Ali Akbar Almani\Desktop\stream programming>start java ConnectionAccept
or 1024 Hello

C:\Users\Ali Akbar Almani\Desktop\stream programming>start java ConnectionReques
tor localhost 1024

C:\Users\Ali Akbar Almani\Desktop\stream programming>
```



```
now ready to accept a connection
```

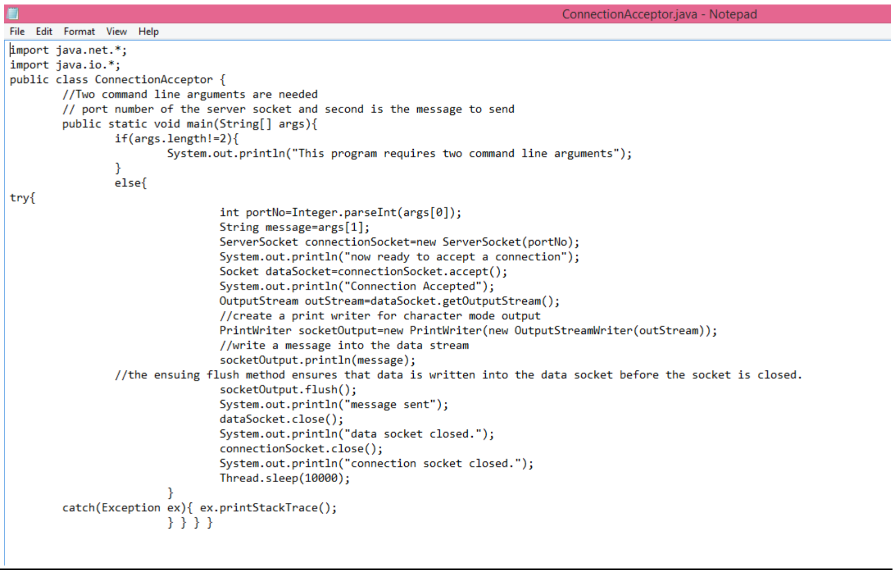


```
Connection request granted.
Waiting to read.
Message received.        Hello
data socket closed.
```

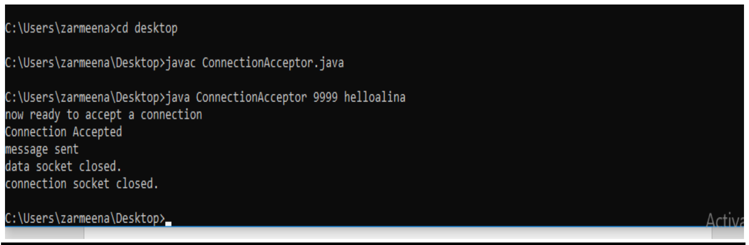
Task No: 04

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

- **Server:**

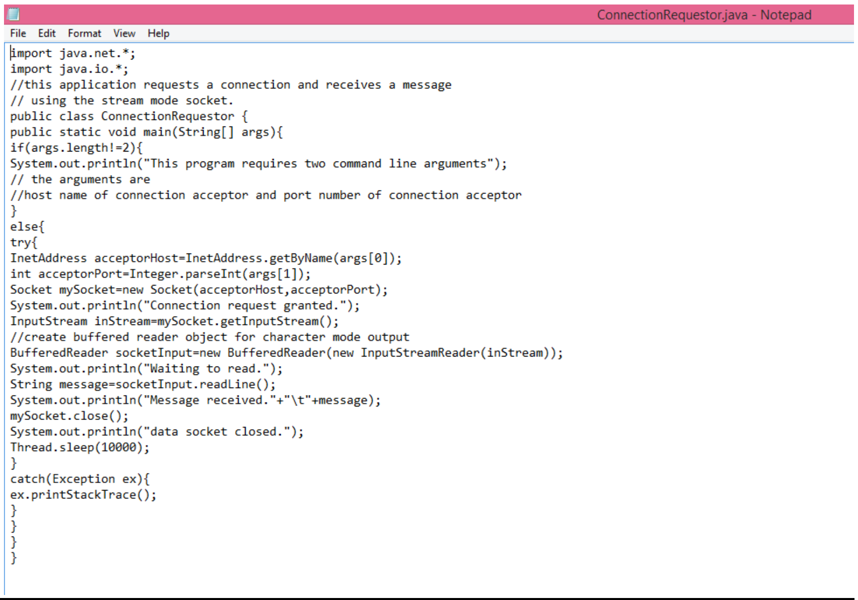


```
import java.net.*;
import java.io.*;
public class ConnectionAcceptor {
        //Two command line arguments are needed
        // port number of the server socket and second is the message to send
        public static void main(String[] args){
                if(args.length!=2){
                        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();
                        } } } }
```

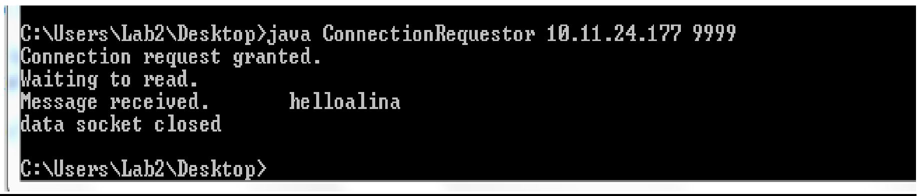


```
C:\Users\zarmeena>cd desktop

C:\Users\zarmeena\Desktop>javac ConnectionAcceptor.java

C:\Users\zarmeena\Desktop>java ConnectionAcceptor 9999 helloalina
now ready to accept a connection
Connection Accepted
message sent
data socket closed.
connection socket closed.

C:\Users\zarmeena\Desktop>_
```

- **Client:**



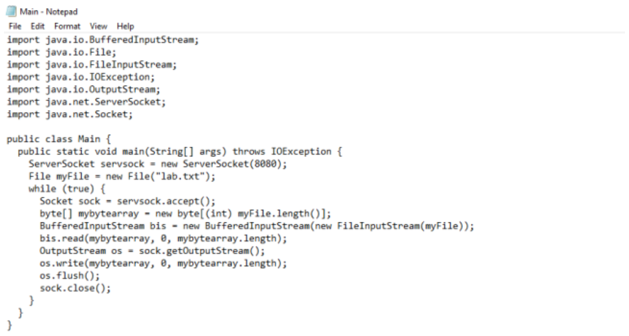
```
import java.net.*;
import java.io.*;
//this application requests a connection and receives a message
// using the stream mode socket.
public class ConnectionRequestor {
public static void main(String[] args){
if(args.length!=2){
System.out.println("This program requires two command line arguments");
// 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();
}
}
}
}
```



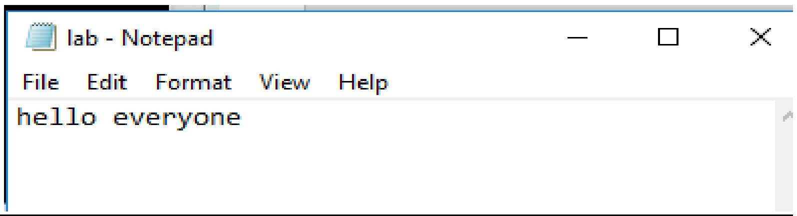
```
C:\Users\Lab2\Desktop>java ConnectionRequestor 10.11.24.177 9999
Connection request granted.
Waiting to read.
Message received.       helloalina
data socket closed

C:\Users\Lab2\Desktop>
```

Task No: 05

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

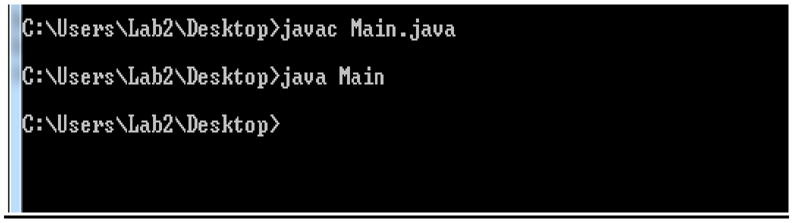


```
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;

public class Main {
  public static void main(String[] args) throws IOException {
    ServerSocket servsock = new ServerSocket(8080);
    File myFile = new File("lab.txt");
    while (true) {
      Socket sock = servsock.accept();
      byte[] mybytearray = new byte[(int) myFile.length()];
      BufferedInputStream bis = new BufferedInputStream(new FileInputStream(myFile));
      bis.read(mybytearray, 0, mybytearray.length);
      OutputStream os = sock.getOutputStream();
      os.write(mybytearray, 0, mybytearray.length);
      os.flush();
      sock.close();
    }
  }
}
```

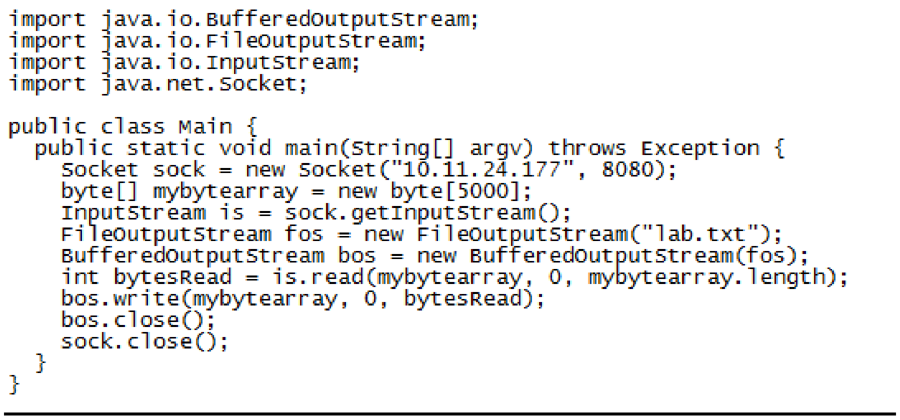


```
hello everyone
```

Client:

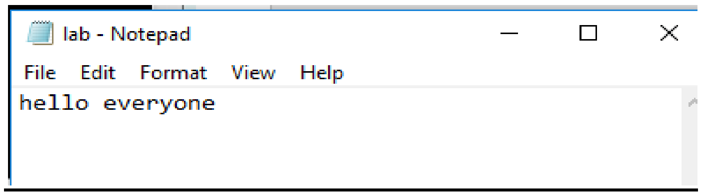


```
C:\Users\Lab2\Desktop>javac Main.java

C:\Users\Lab2\Desktop>java Main

C:\Users\Lab2\Desktop>
```



```
import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.net.Socket;

public class Main {
  public static void main(String[] argv) throws Exception {
    Socket sock = new Socket("10.11.24.177", 8080);
    byte[] mybytearray = new byte[5000];
    InputStream is = sock.getInputStream();
    FileOutputStream fos = new FileOutputStream("lab.txt");
    BufferedOutputStream bos = new BufferedOutputStream(fos);
    int bytesRead = is.read(mybytearray, 0, mybytearray.length);
    bos.write(mybytearray, 0, bytesRead);
    bos.close();
    sock.close();
  }
}
```

Received file on client pc:

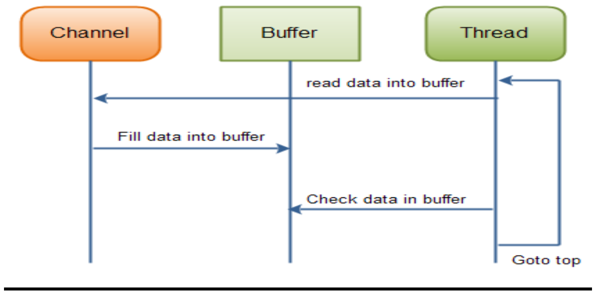


```
hello everyone
```

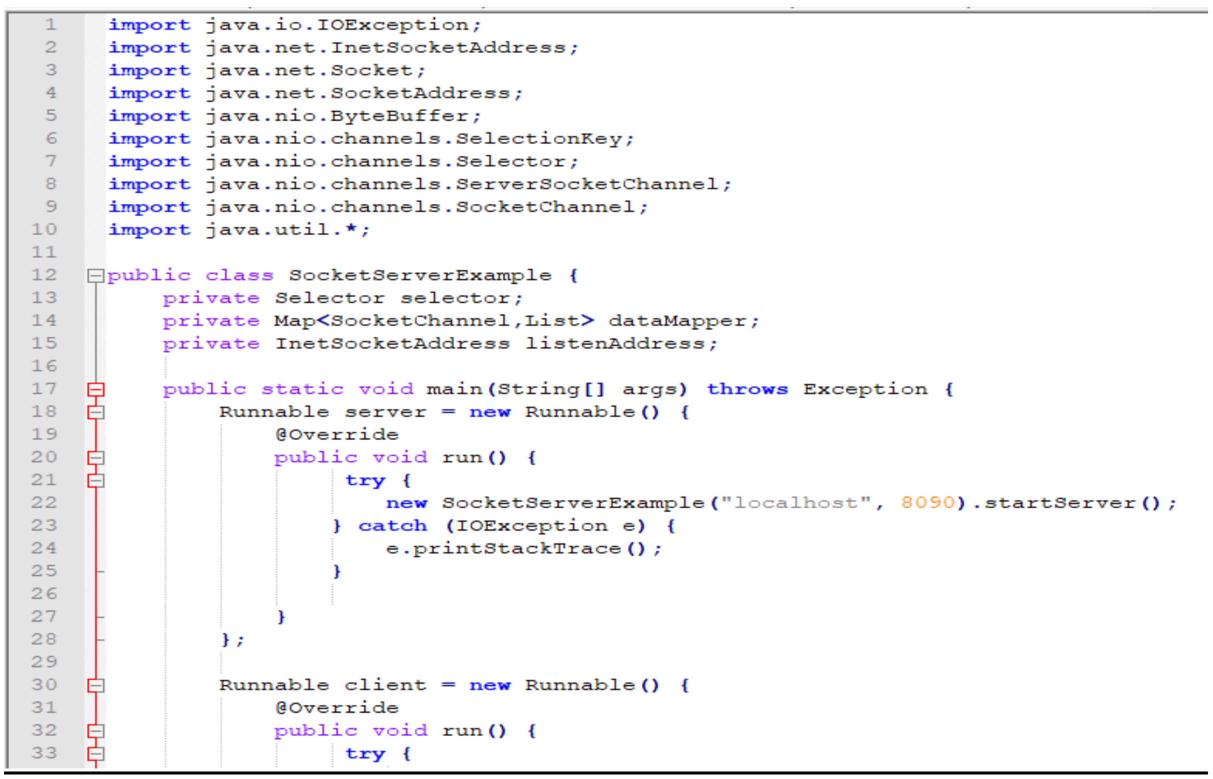
Task No: 06

Explore the non-blocking java socket API in the niopackage and implement a sample program.

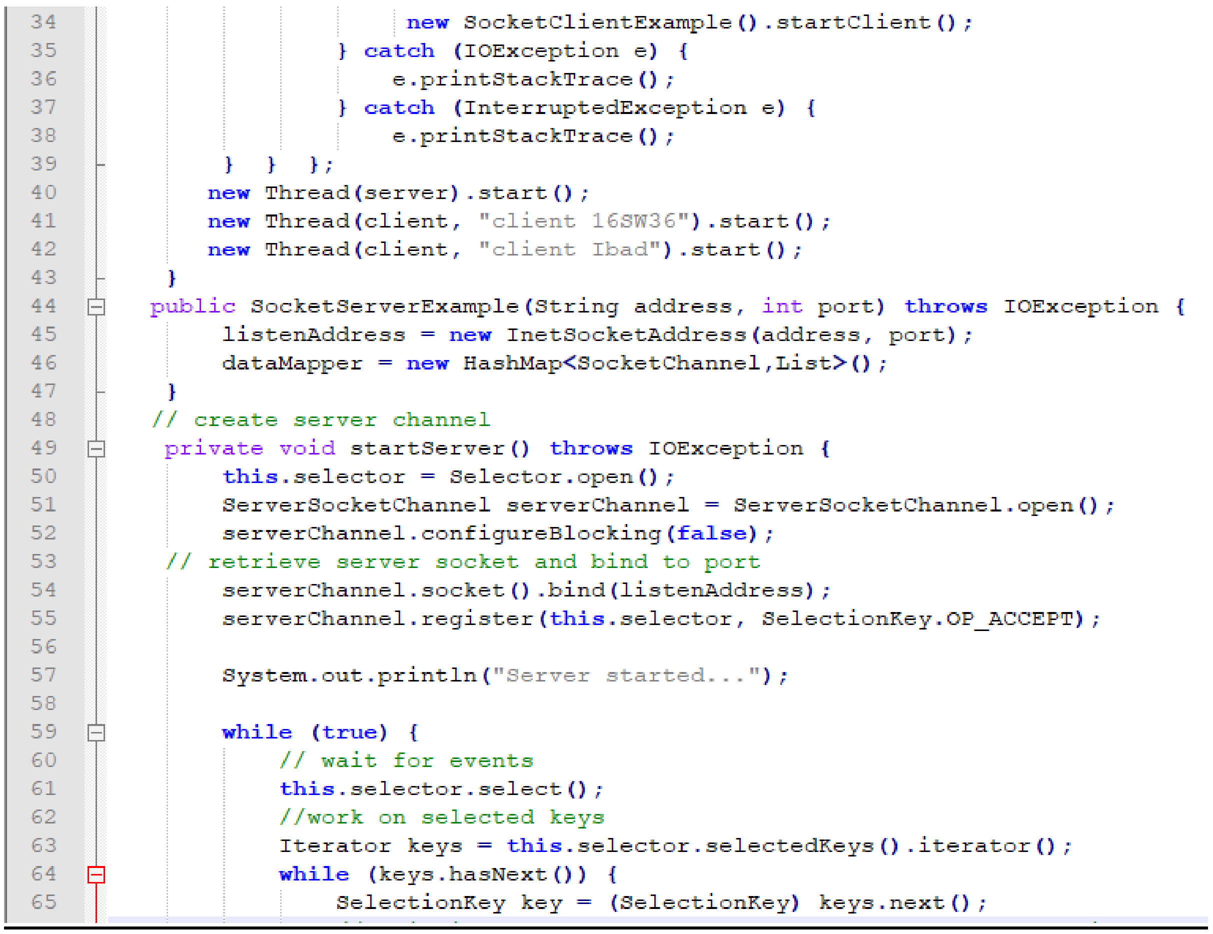
Java NIO's non-blocking mode enables a thread to request reading data from a channel, and only get what is currently available, or nothing at all, if no data is currently available.



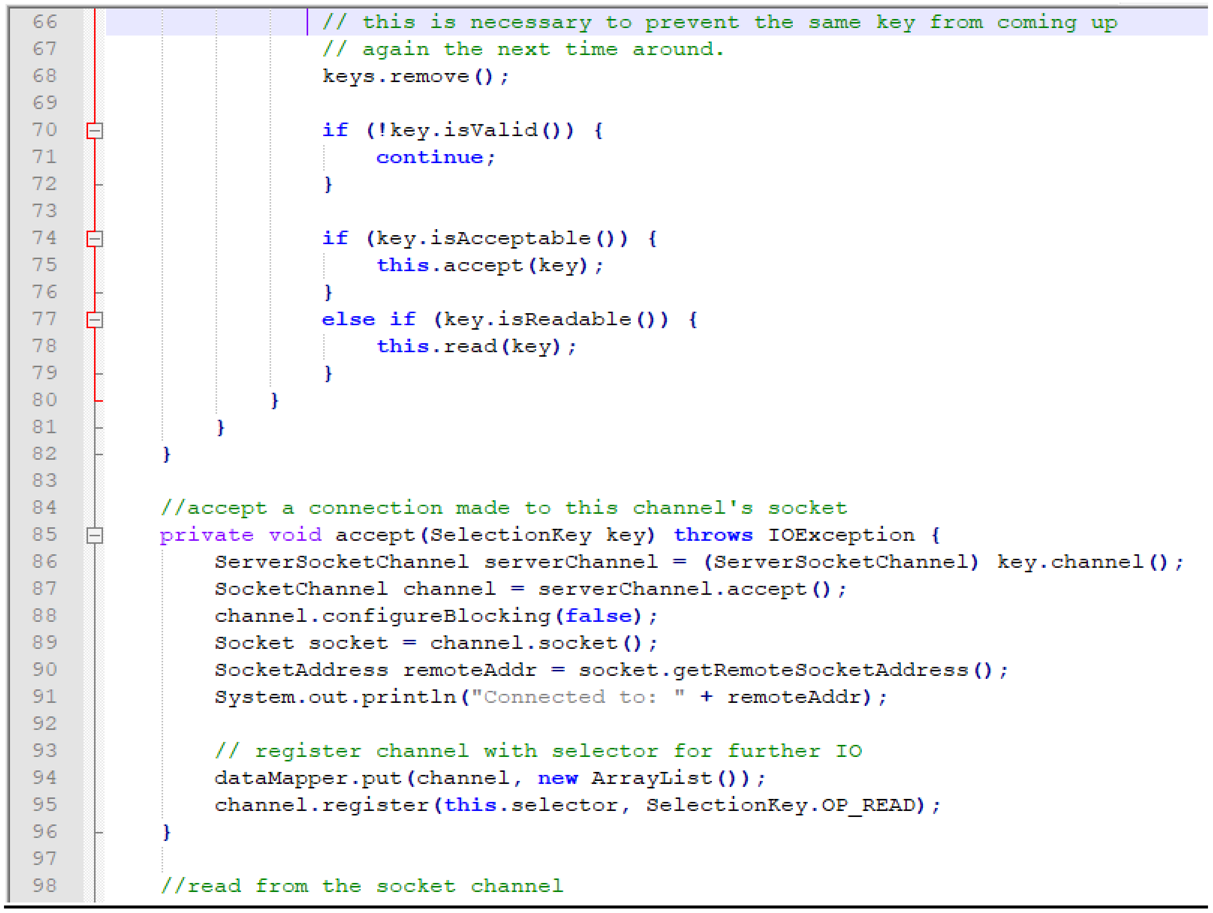
- **Sample Program:**
- **Server:**



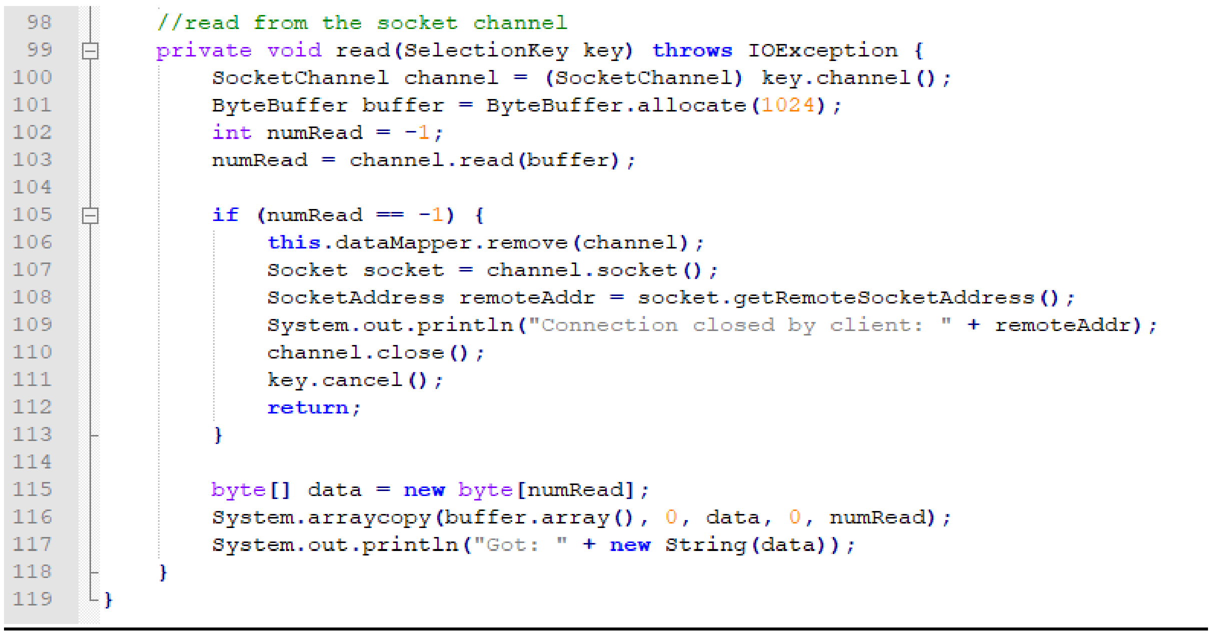
```java
import java.io.IOException;
import java.net.InetSocketAddress;
import java.net.Socket;
import java.net.SocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.SelectionKey;
import java.nio.channels.Selector;
import java.nio.channels.ServerSocketChannel;
import java.nio.channels.SocketChannel;
import java.util.*;

public class SocketServerExample {
    private Selector selector;
    private Map<SocketChannel,List> 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 {
```



```java
                    new SocketClientExample().startClient();
                } catch (IOException e) {
                    e.printStackTrace();
                } catch (InterruptedException e) {
                    e.printStackTrace();
        }   }   };
        new Thread(server).start();
        new Thread(client, "client 16SW36").start();
        new Thread(client, "client Ibad").start();
    }
    public SocketServerExample(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();
```

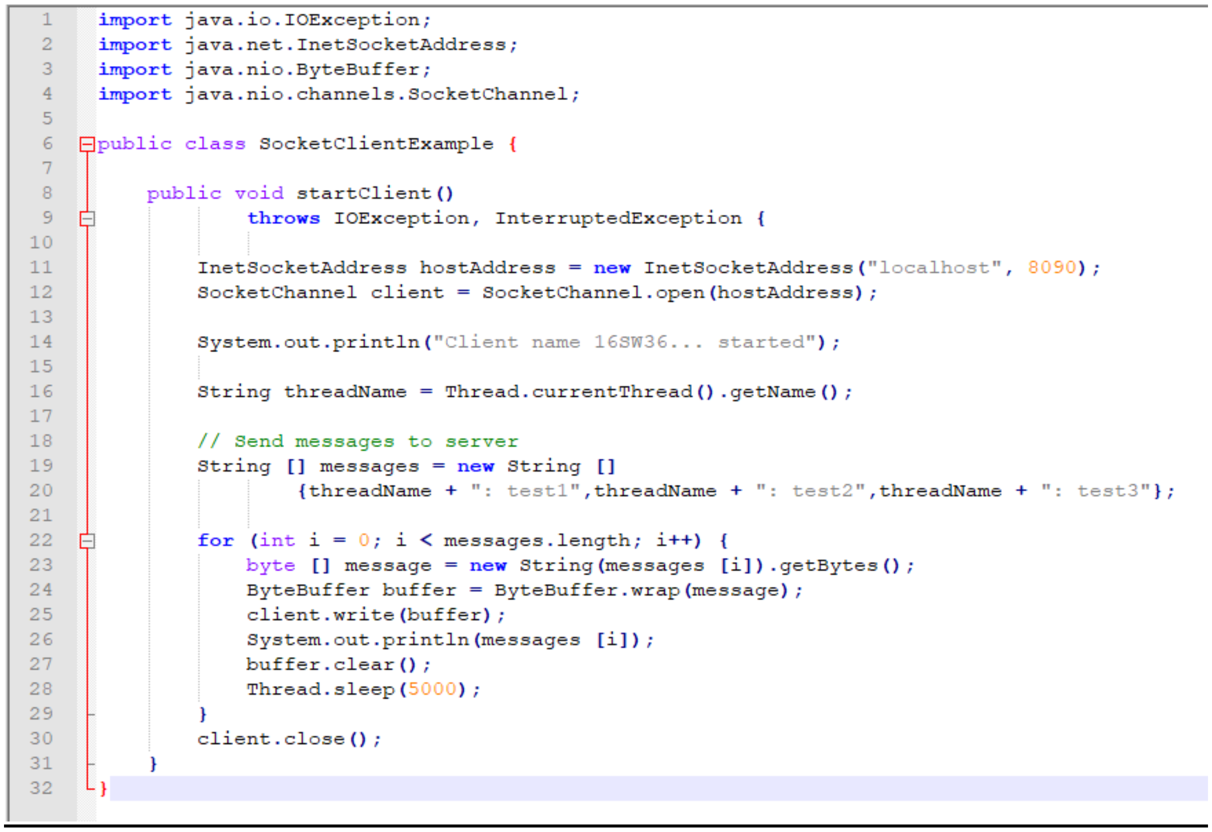


```
                // this is necessary to prevent the same key from coming up
                // again the next time around.
                keys.remove();

                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
```

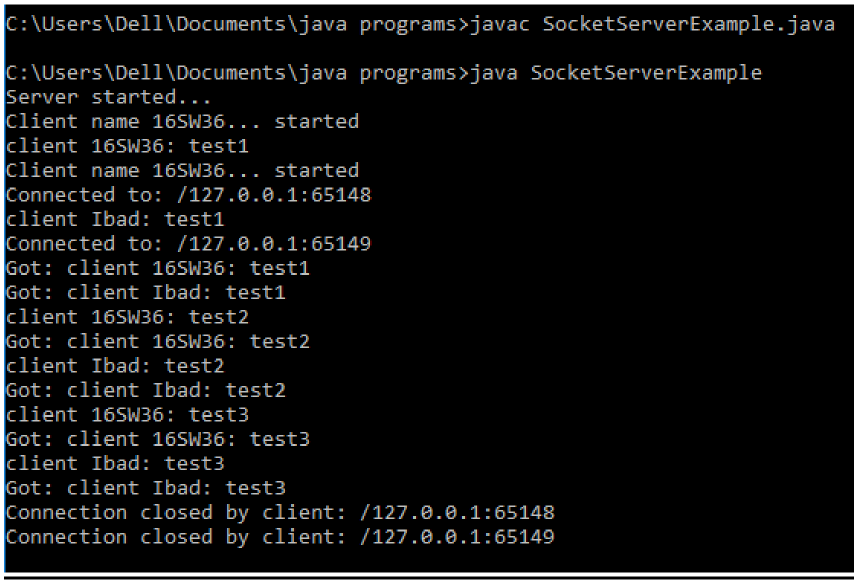


```
    //read from the socket channel
    private void read(SelectionKey key) throws IOException {
        SocketChannel channel = (SocketChannel) key.channel();
        ByteBuffer buffer = ByteBuffer.allocate(1024);
        int numRead = -1;
        numRead = channel.read(buffer);

        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));
    }
}
```

- **Client:**



```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 name 16SW36... 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();
    }
}
```



```
C:\Users\Dell\Documents\java programs>javac SocketServerExample.java

C:\Users\Dell\Documents\java programs>java SocketServerExample
Server started...
Client name 16SW36... started
client 16SW36: test1
Client name 16SW36... started
Connected to: /127.0.0.1:65148
client Ibad: test1
Connected to: /127.0.0.1:65149
Got: client 16SW36: test1
Got: client Ibad: test1
client 16SW36: test2
Got: client 16SW36: test2
client Ibad: test2
Got: client Ibad: test2
client 16SW36: test3
Got: client 16SW36: test3
client Ibad: test3
Got: client Ibad: test3
Connection closed by client: /127.0.0.1:65148
Connection closed by client: /127.0.0.1:65149
```