CS2105 Introduction to Computer Networks

Lecture 3

Socket Programming

27 August 2018

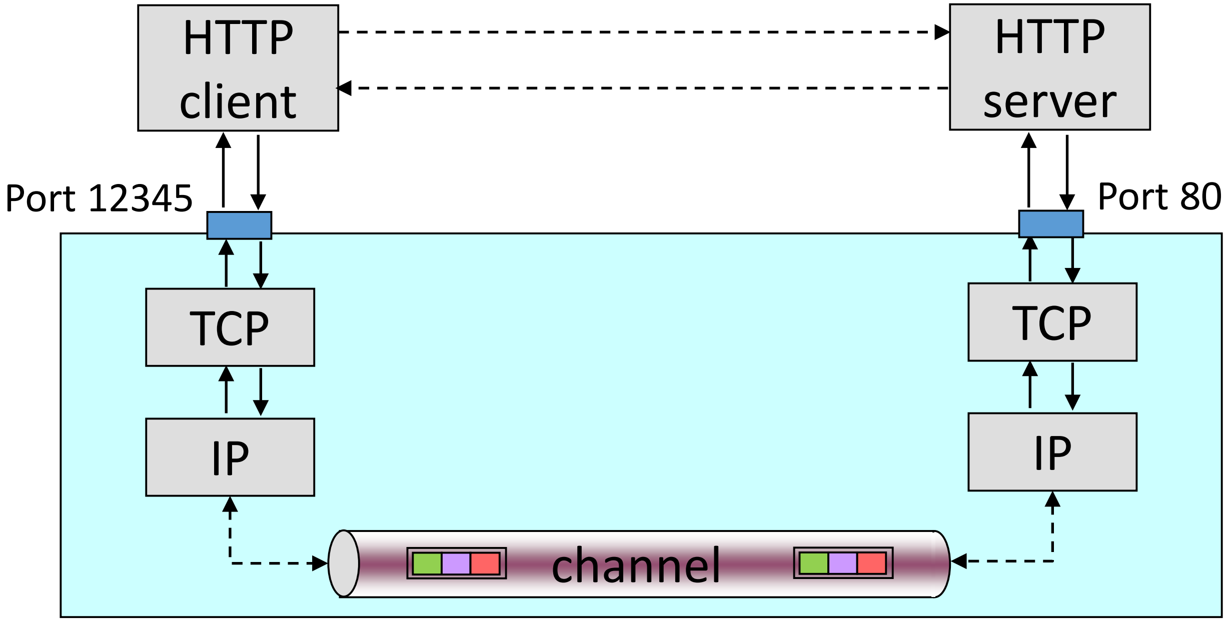
PREVIOUS LECTURE

Web and HTTP

A Web page consists of a *base HTML file* and *some other objects* referenced by the HTML file.

HTTP uses TCP as transport service.

- TCP, in turn, uses service provided by IP!



PREVIOUS LECTURE

HTTP Connections

HTTP 1.0: non-persistent

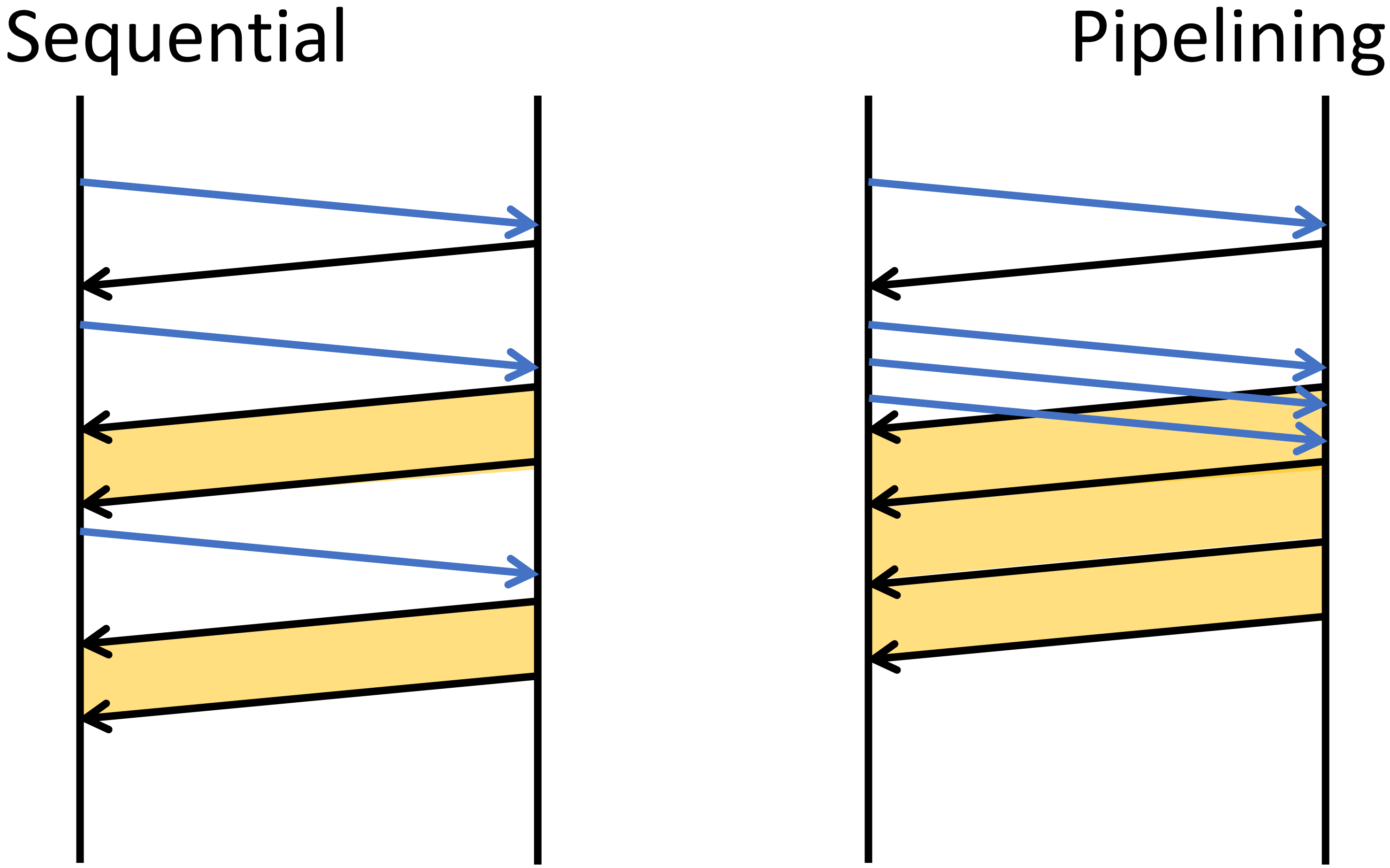
- At most one object is sent over one TCP connection.
    - connection is then closed.
- Downloading multiple objects requires multiple connections.
    - TCP connections may be launched in parallel

HTTP 1.1: persistent

- Server leaves connection open after sending a Web object.
- Multiple objects can be sent over a single TCP connection.
    - Requests may be sent in parallel

PREVIOUS LECTURE

HTTP/1.1



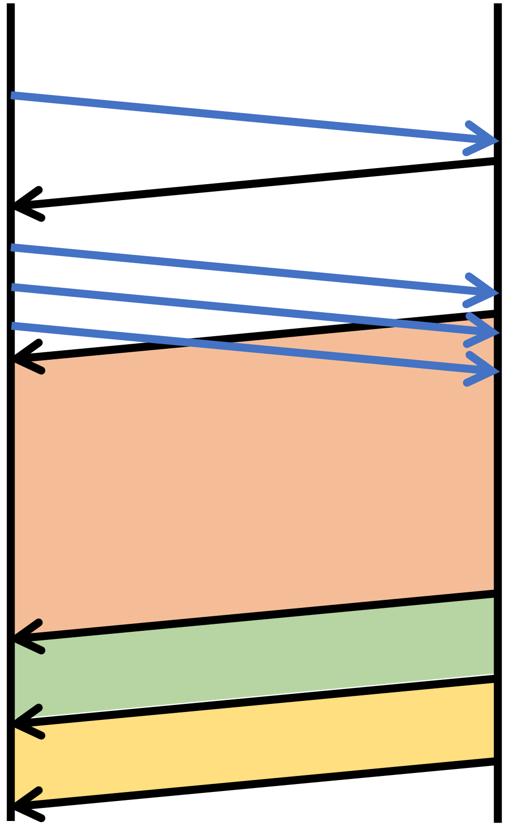
Sequential

Pipelining

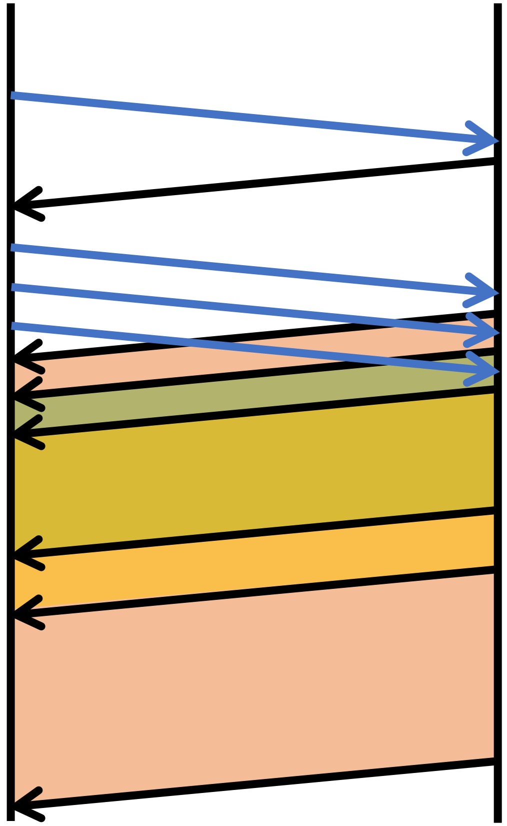
PREVIOUS LECTURE

HTTP/1.1 vs. HTTP/2

Head-of-line Blocking



Multiplexing



Learning Outcome

After this class, you are expected to:

- Know the concept of sockets.
- Differentiate between TCP and UDP sockets
- Be able to write simple client/server programs using socket programming.

Lecture 2: Roadmap

2.1 Principles of Network Applications

2.2 Web and HTTP

2.5 DNS

2.7 Socket programming with TCP

2.8 Socket programming with UDP

Recall that hosts are identified by IP addresses

A host can run several processes

process

process

process

client

server

process

process

process

port number identifies the process

Some standard port numbers:

- HTTP server: 80
- POP server: 25
- WoW: 3724

IANA coordinates the assignment of port number

Processes

- Applications runs in hosts as processes.
  - Within the same host, two processes communicate using inter-process communication (defined by OS).
  - Processes in different hosts communicate by exchanging messages (according to protocols).

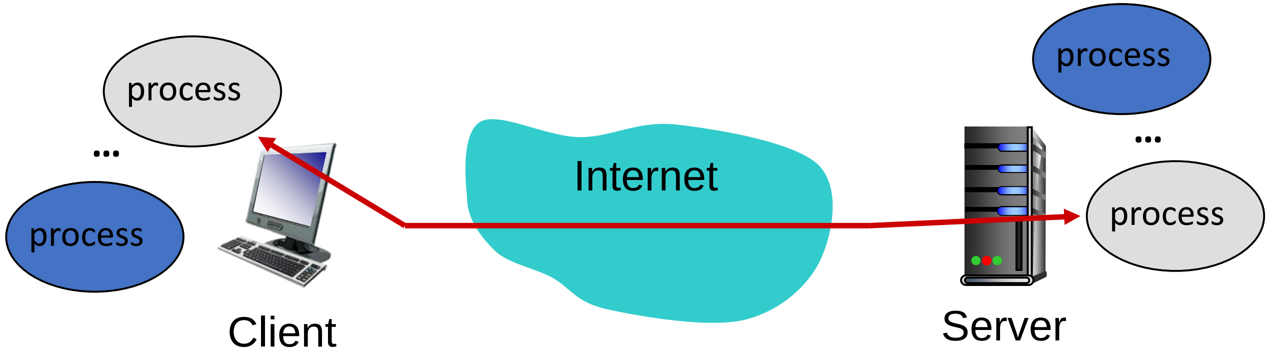
In C/S model

server process waits to be contacted

client process initiates the communication

Addressing Processes

- IP address is used to identify a host device
  - A 32-bit integer (e.g. 137.132.21.27)
- Question: is IP address of a host suffice to identify a process running inside that host?
  - Ans: no, many processes may run concurrently in a host.



Analogy

Postal service:

- deliver letter to the doorstep: home address

- dispatch letter to the right person in the house: name of the receiver as stated on the letter

Protocol service:

- deliver packet to the right host: IP address of the host

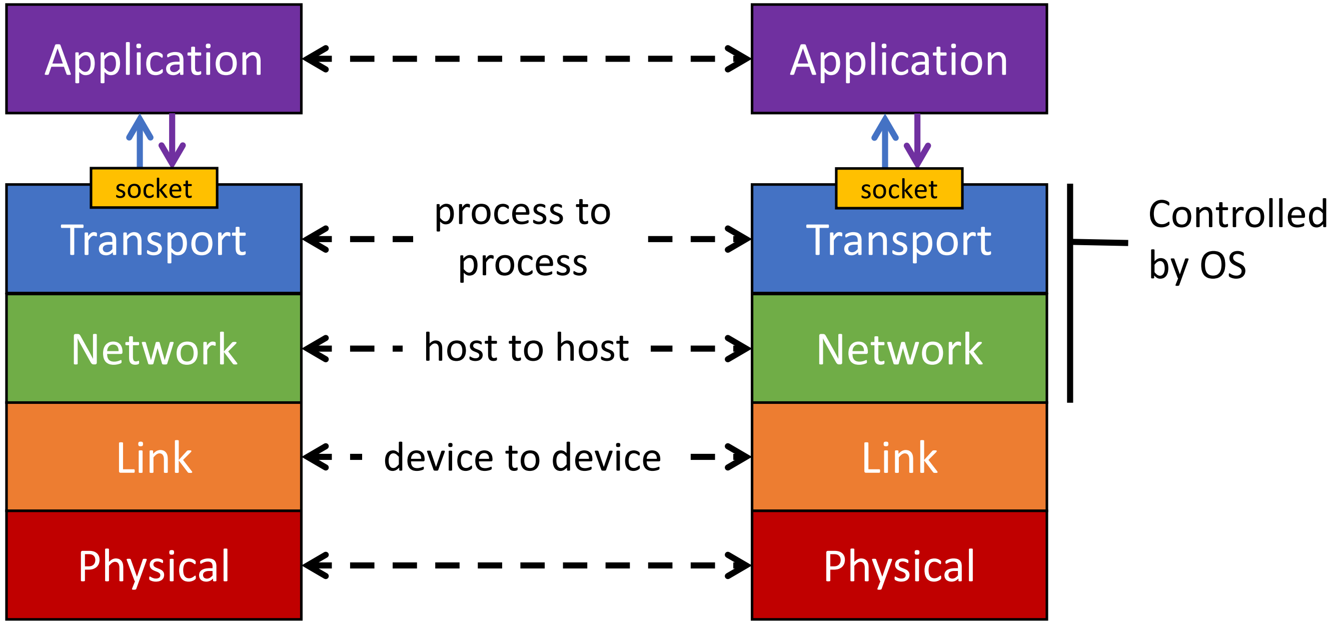
Network

- dispatch packet to the right process in the host: port number of the process

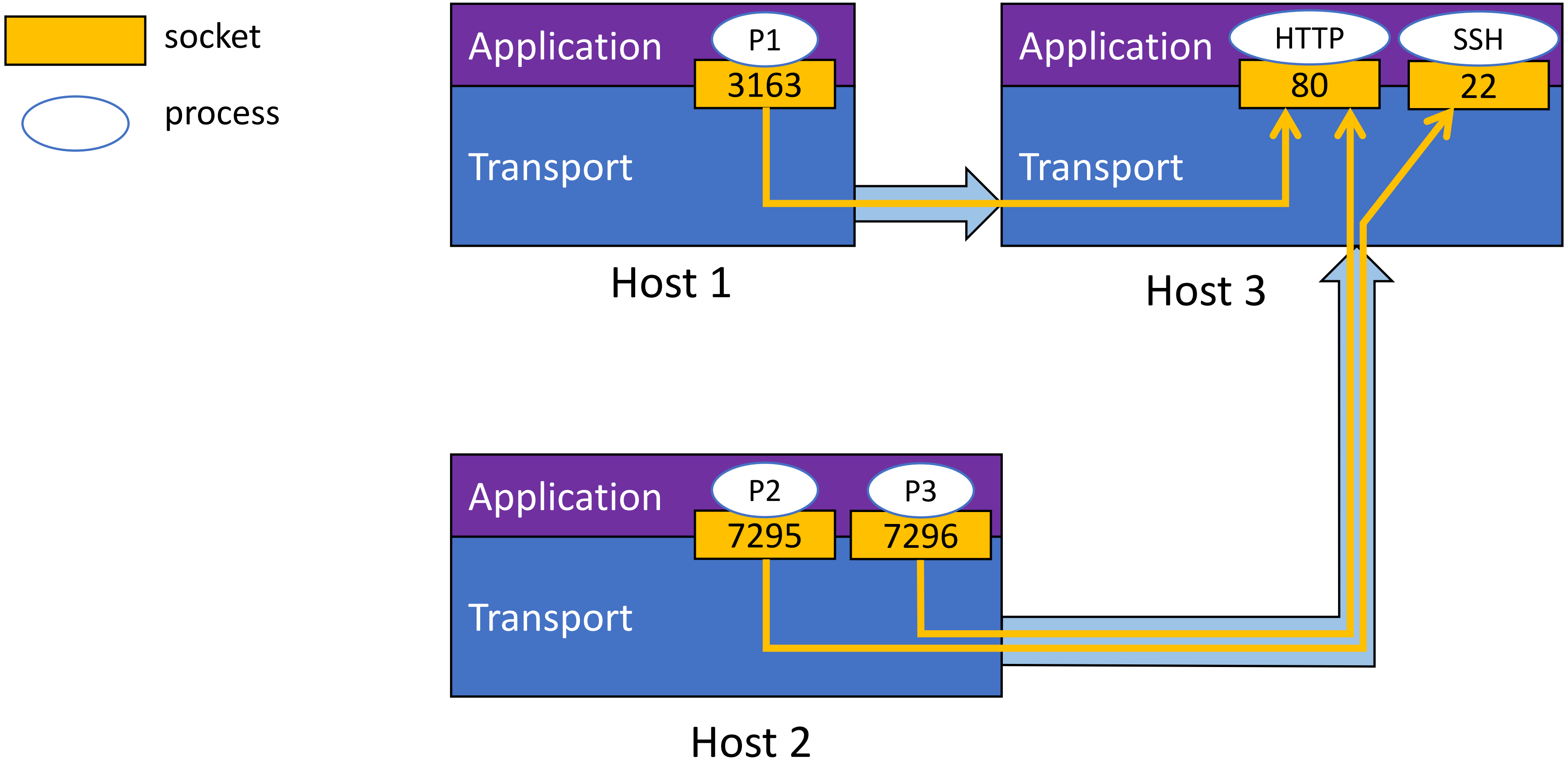
Transport

Sockets

- Socket is the software interface between app processes and transport layer protocols.
  - Process sends/receives messages to/from its socket.
  - Programming-wise: a set of API calls



Multiplexing/De-multiplexing



IP address and port number are used to locate a process

Two types of sockets

- TCP (Transmission Control Protocol)
  - Stream socket
  - Connection-oriented
  - Reliable
- UDP (User Datagram Protocol)
  - Datagram socket
  - Connection-less
  - Unreliable

Socket Programming

- Applications (or processes) treat the Internet as a black box, sending and receiving messages through sockets.
- Two types of sockets
  - stream socket (aka TCP socket) that uses TCP as its transport layer protocol.
    - Connection-oriented, reliable
  - datagram socket (aka UDP socket) that uses UDP.
    - Connection-less, unreliable (transmitted data may be lost, corrupted or received out-of-order)

Lecture 2: Roadmap

2.1 Principles of Network Applications

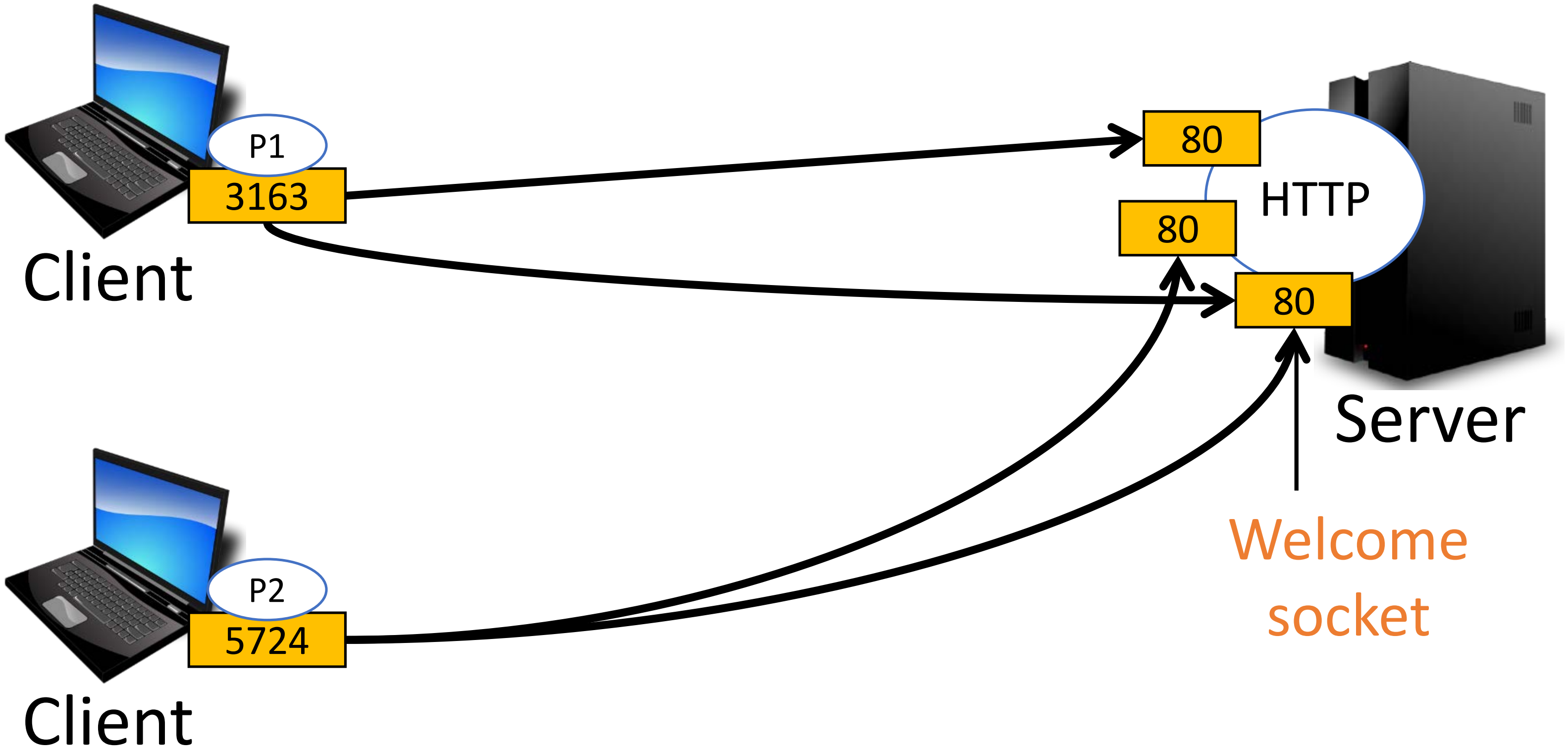
2.2 Web and HTTP

2.5 DNS

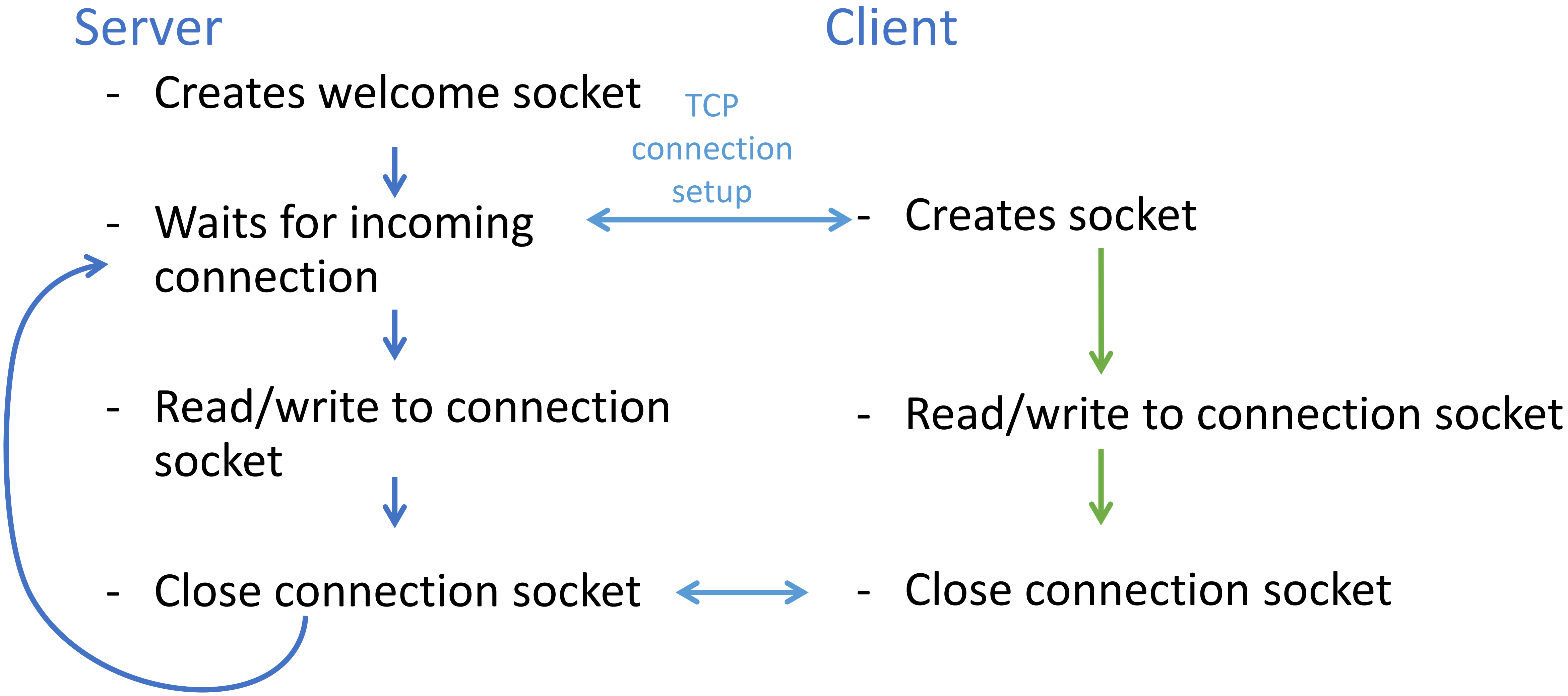
2.7 Socket programming with TCP

2.8 Socket programming with UDP

TCP Socket Programming



Client/Server Interaction



Server

- Creates welcome socket
- Waits for incoming connection
- Read/write to connection socket
- Close connection socket

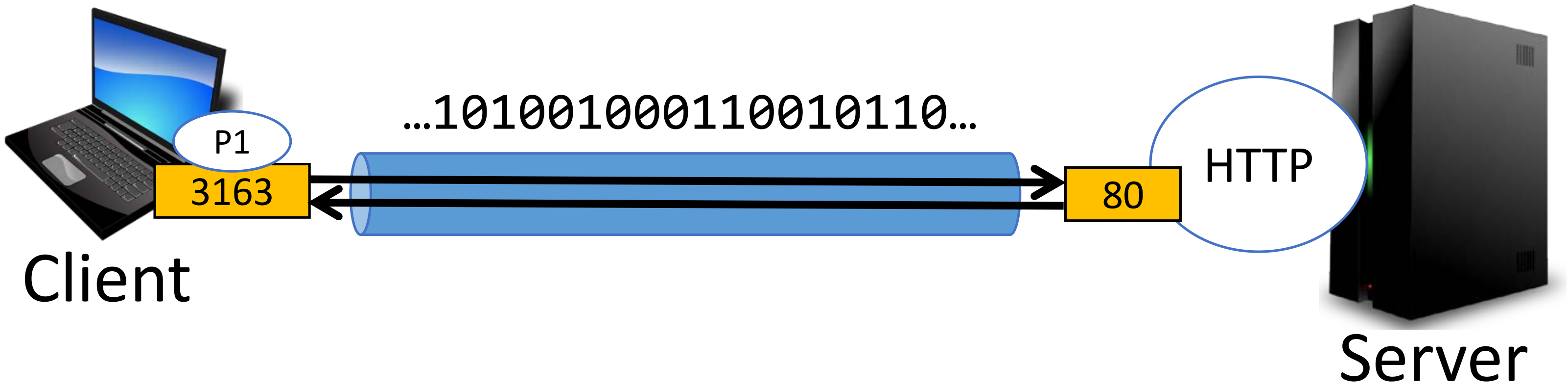
TCP connection setup

Client

- Creates socket
- Read/write to connection socket
- Close connection socket

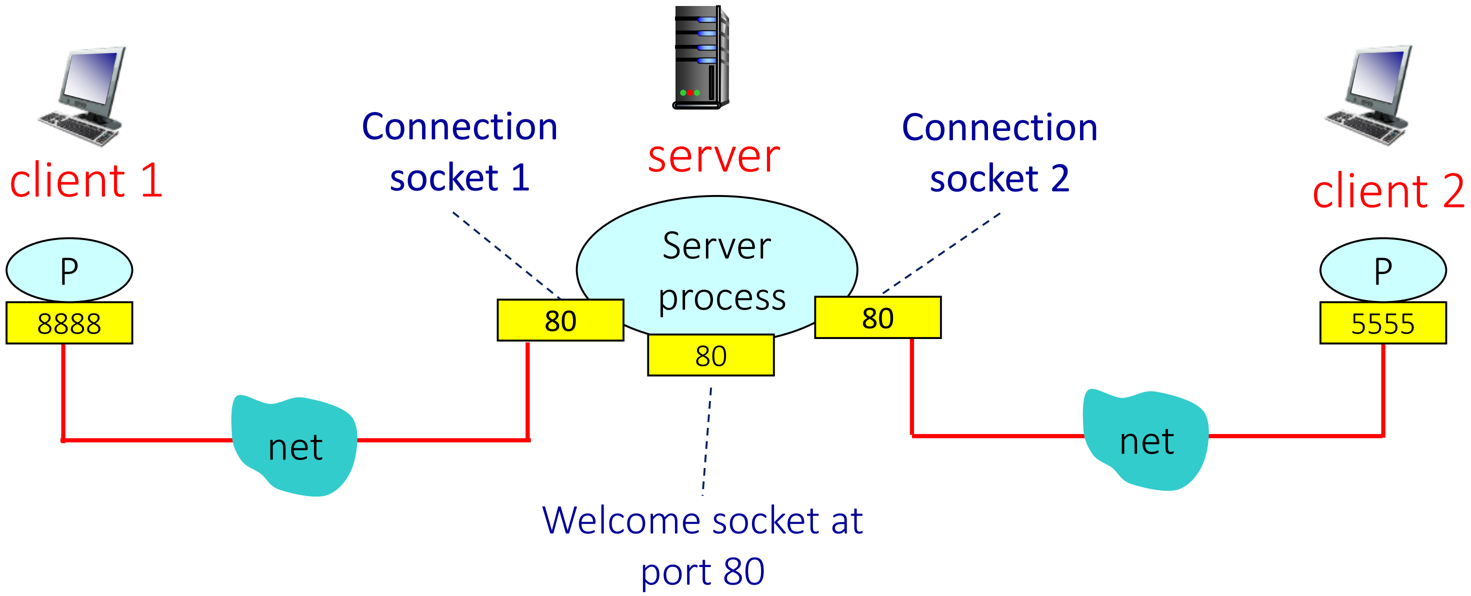
Live demo

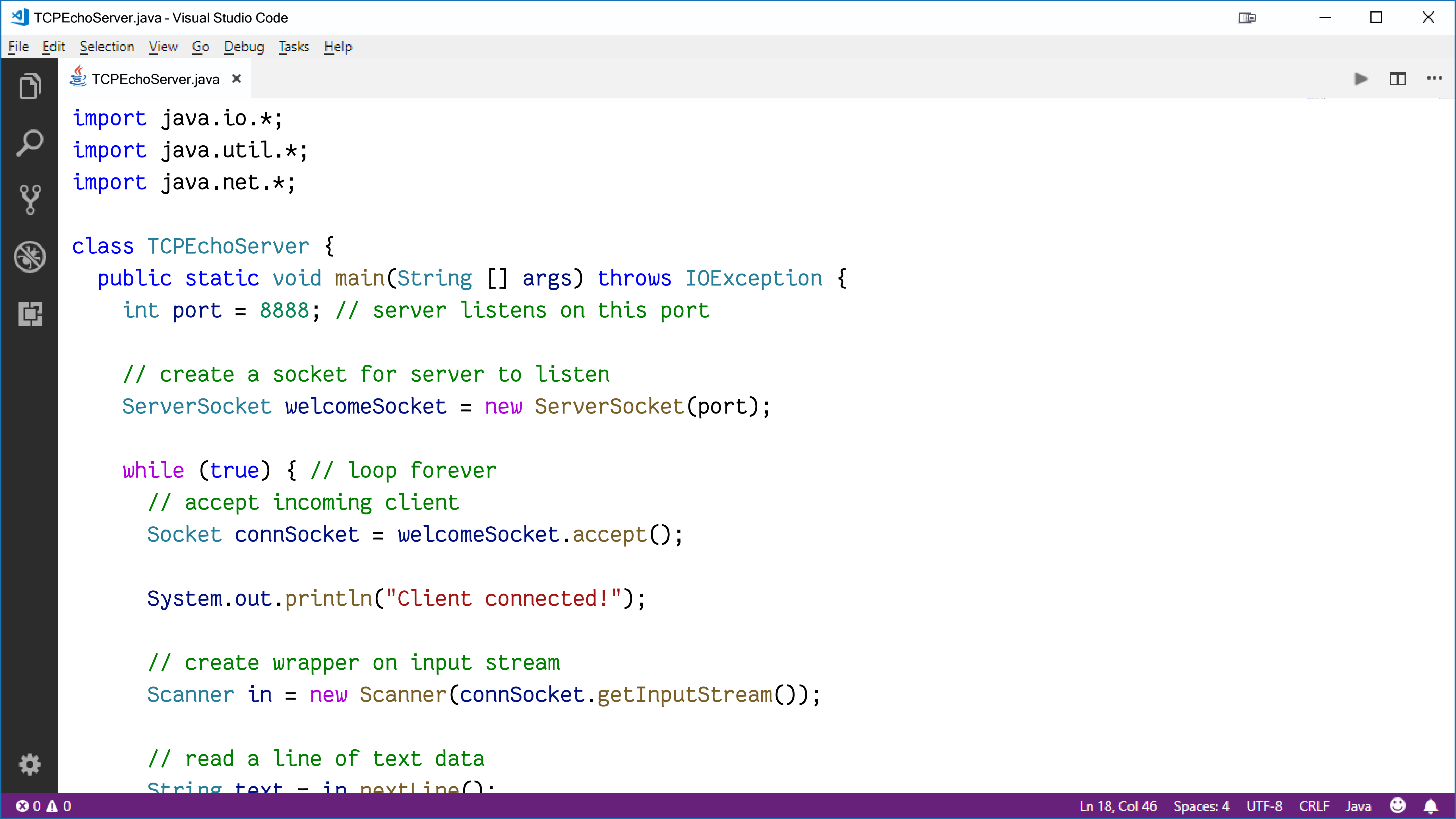
In TCP, data flow in a continuous stream



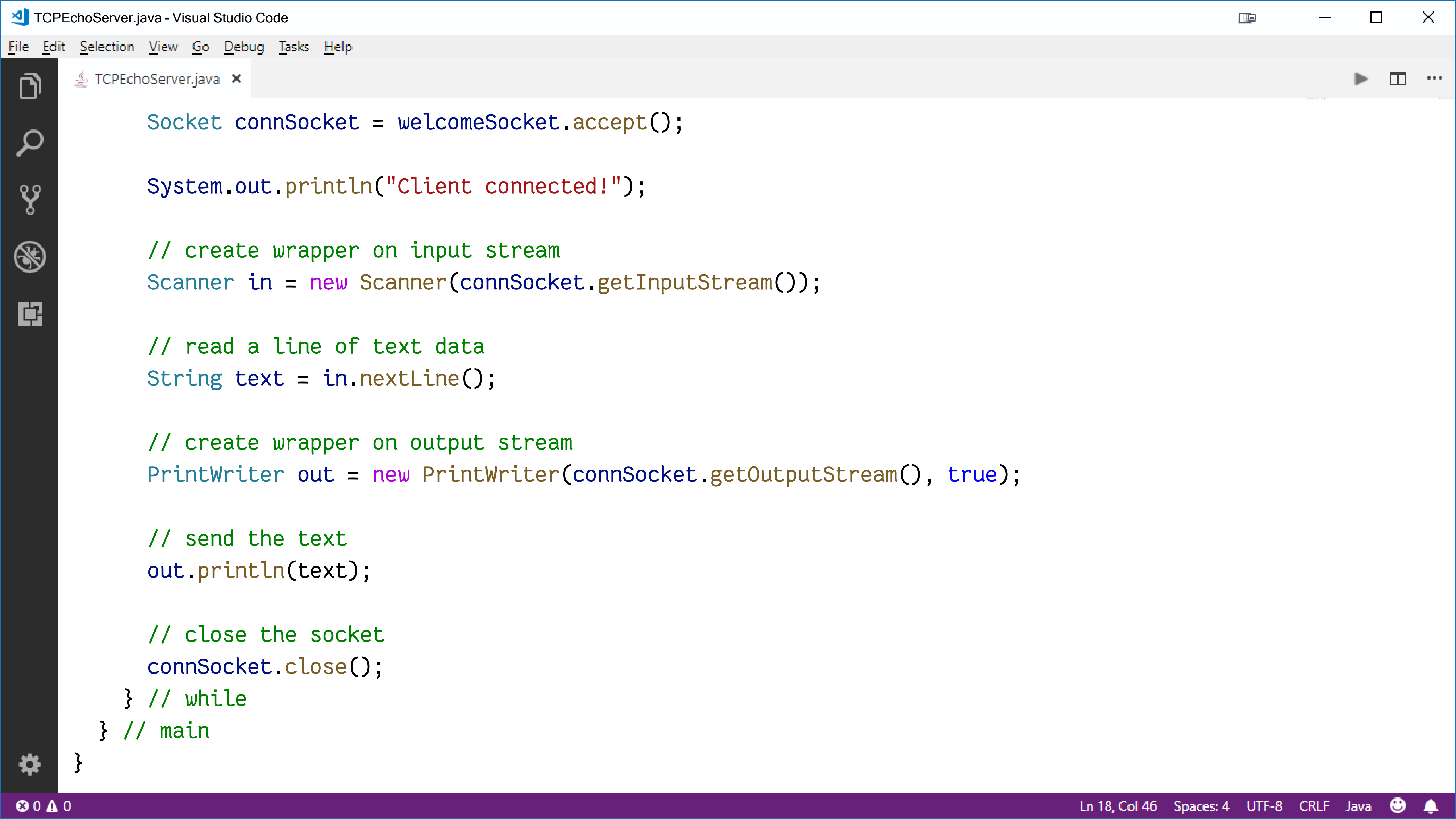
Socket Programming with TCP

- With TCP sockets, a process establishes a connection to another process.
- While the connection is in place, data flows between the processes in continuous streams.
- When contacted by client, server TCP creates a new socket for server process to communicate with client.

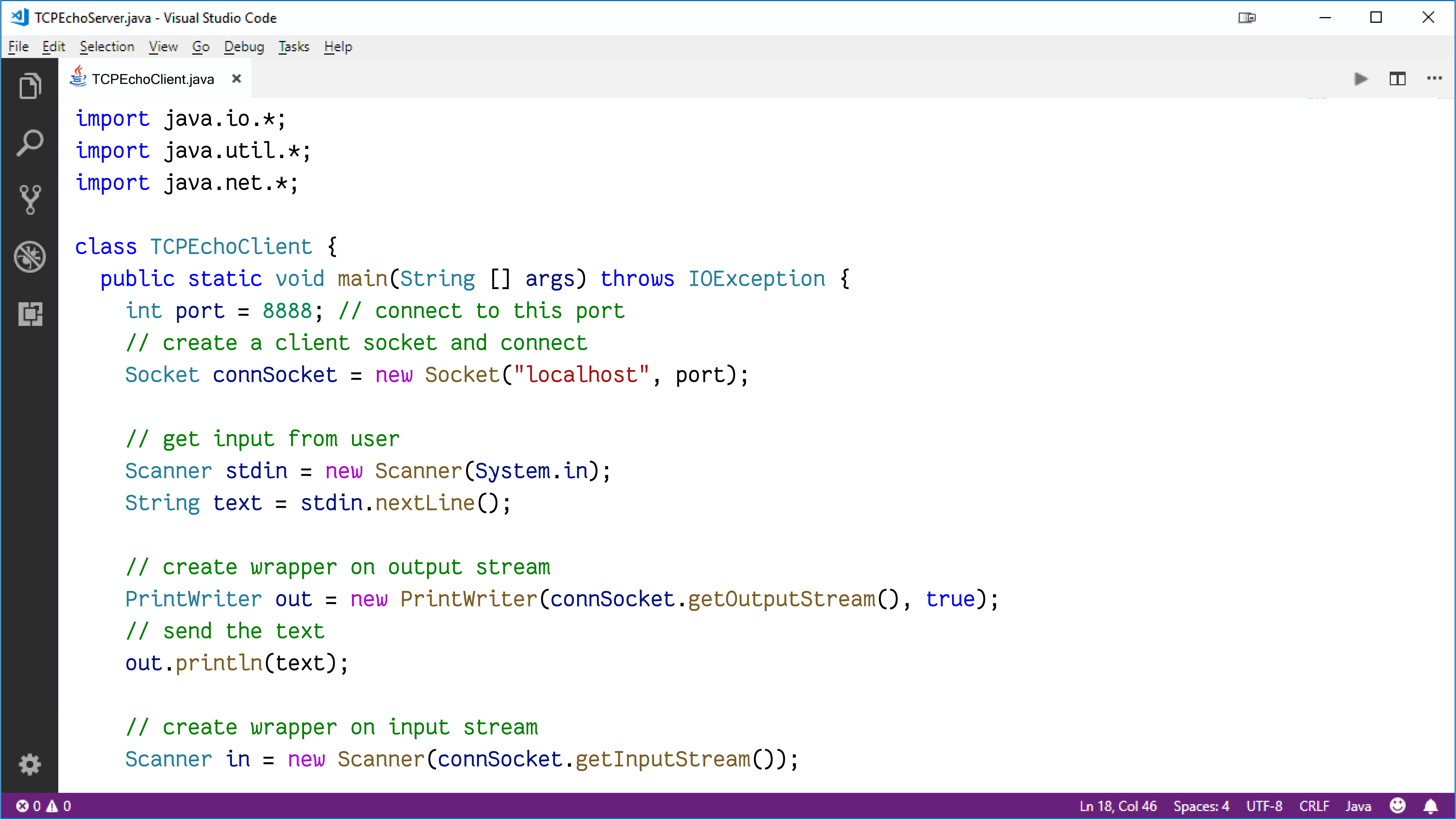




```java
import java.io.*;
import java.util.*;
import java.net.*;

class TCPEchoServer {
  public static void main(String [] args) throws IOException {
    int port = 8888; // server listens on this port

    // create a socket for server to listen
    ServerSocket welcomeSocket = new ServerSocket(port);

    while (true) { // loop forever
      // accept incoming client
      Socket connSocket = welcomeSocket.accept();

      System.out.println("Client connected!");

      // create wrapper on input stream
      Scanner in = new Scanner(connSocket.getInputStream());

      // read a line of text data
      String text = in.nextLine();
```

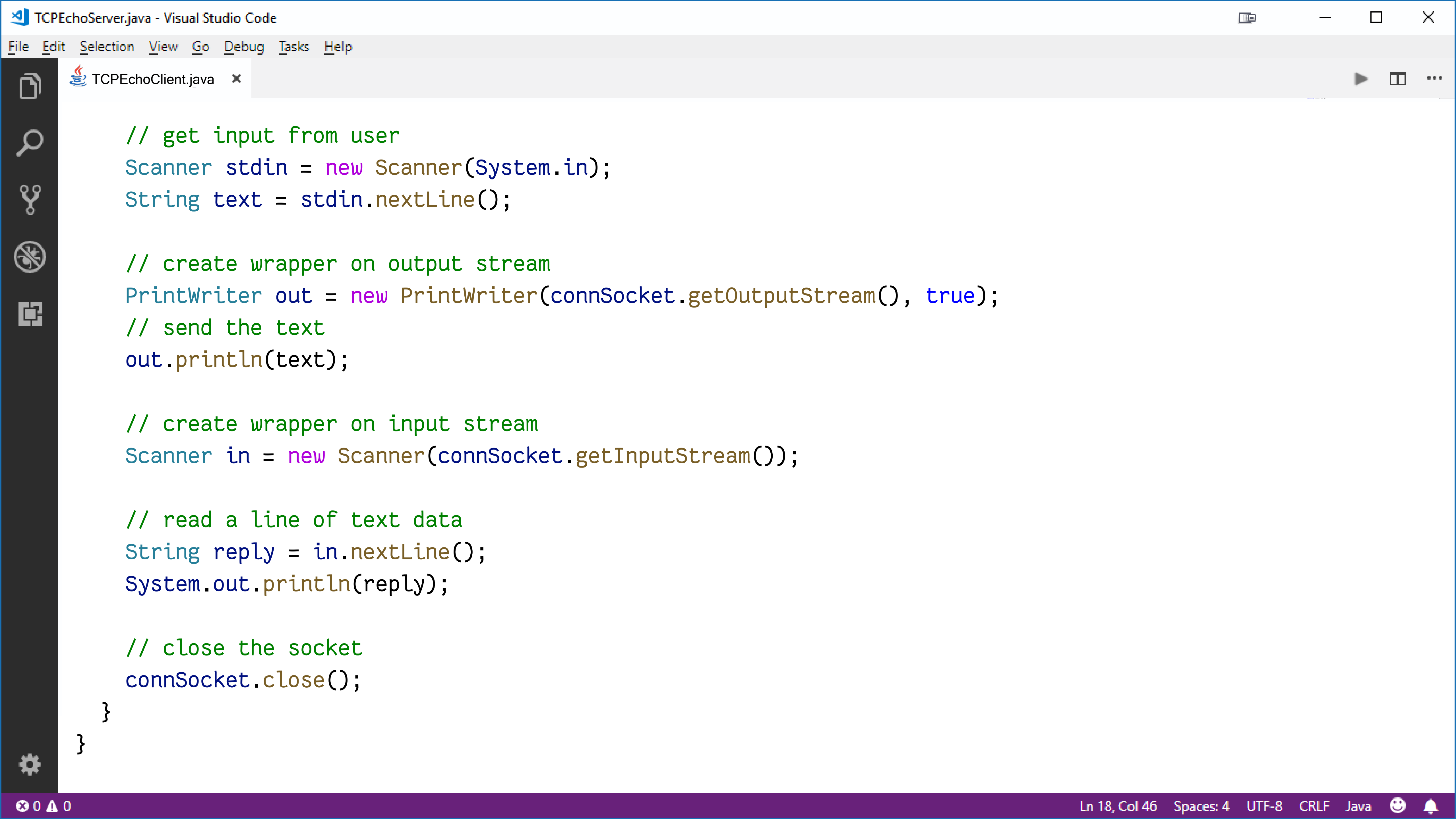


```
      Socket connSocket = welcomeSocket.accept();

      System.out.println("Client connected!");

      // create wrapper on input stream
      Scanner in = new Scanner(connSocket.getInputStream());

      // read a line of text data
      String text = in.nextLine();

      // create wrapper on output stream
      PrintWriter out = new PrintWriter(connSocket.getOutputStream(), true);

      // send the text
      out.println(text);

      // close the socket
      connSocket.close();
    } // while
  } // main
}
```



TCPEchoServer.java - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
TCPEchoClient.java ×

```
import java.io.*;
import java.util.*;
import java.net.*;

class TCPEchoClient {
  public static void main(String [] args) throws IOException {
    int port = 8888; // connect to this port
    // create a client socket and connect
    Socket connSocket = new Socket("localhost", port);

    // get input from user
    Scanner stdin = new Scanner(System.in);
    String text = stdin.nextLine();

    // create wrapper on output stream
    PrintWriter out = new PrintWriter(connSocket.getOutputStream(), true);
    // send the text
    out.println(text);

    // create wrapper on input stream
    Scanner in = new Scanner(connSocket.getInputStream());
```



```
    // get input from user
    Scanner stdin = new Scanner(System.in);
    String text = stdin.nextLine();

    // create wrapper on output stream
    PrintWriter out = new PrintWriter(connSocket.getOutputStream(), true);
    // send the text
    out.println(text);

    // create wrapper on input stream
    Scanner in = new Scanner(connSocket.getInputStream());

    // read a line of text data
    String reply = in.nextLine();
    System.out.println(reply);

    // close the socket
    connSocket.close();
  }
}
```

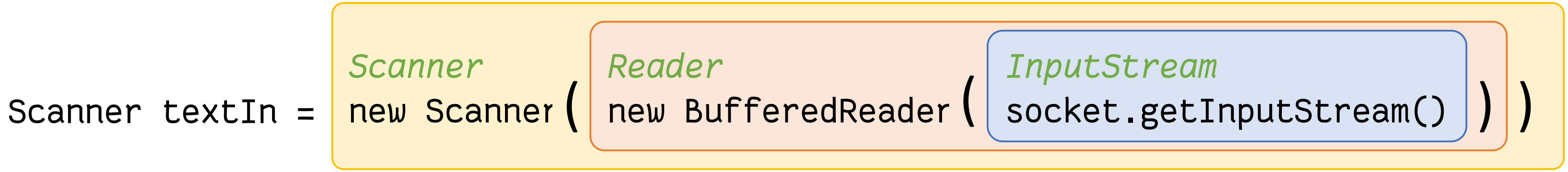
TCP Server vs Client

```java
// TCPEchoServer
// accept incoming client
Socket connSocket = welcomeSocket.accept();

System.out.println("Client connected!");

// create wrapper on input stream
Scanner in = new
    Scanner(connSocket.getInputStream());

// read a line of text data
String text = in.nextLine();

// create wrapper on output stream
PrintWriter out = new
    PrintWriter(connSocket.getOutputStream(),
    true);
// send the text
out.println(text);
```

```java
// TCPEchoClient
// create a client socket and connect
Socket connSocket = new Socket("localhost", port);

// get input from user;
String text = new Scanner(System.in).nextLine();

// create wrapper on output stream
PrintWriter out = new
    PrintWriter(connSocket.getOutputStream(), true);
// send the text
out.println(text);

// create wrapper on input stream
Scanner in = new
    Scanner(connSocket.getInputStream());

// read a line of text data
String reply = in.nextLine();
System.out.println(reply);
```

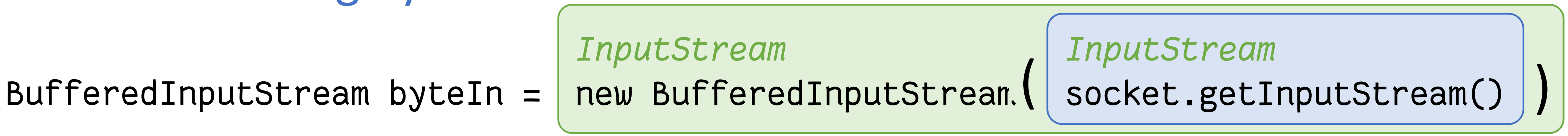
Wrapping Streams

For reading text



```
Scanner textIn = new Scanner(new BufferedReader(socket.getInputStream()))
```

For reading bytes



```
BufferedInputStream byteIn = new BufferedInputStream(socket.getInputStream())
```

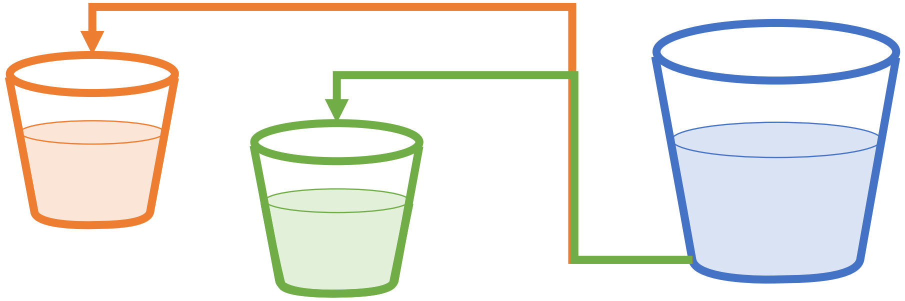
Wrapping Streams

For reading both text and bytes?

ERROR

```
                 Scanner        Reader                    InputStream
Scanner textIn = new Scanner( new BufferedReader( socket.getInputStream() ) )
```

```
                              InputStream                InputStream
BufferedInputStream byteIn = new BufferedInputStream( socket.getInputStream() )
```



No knowing which bytes will flow to which buffer

Only one InputStream

Wrapping Streams

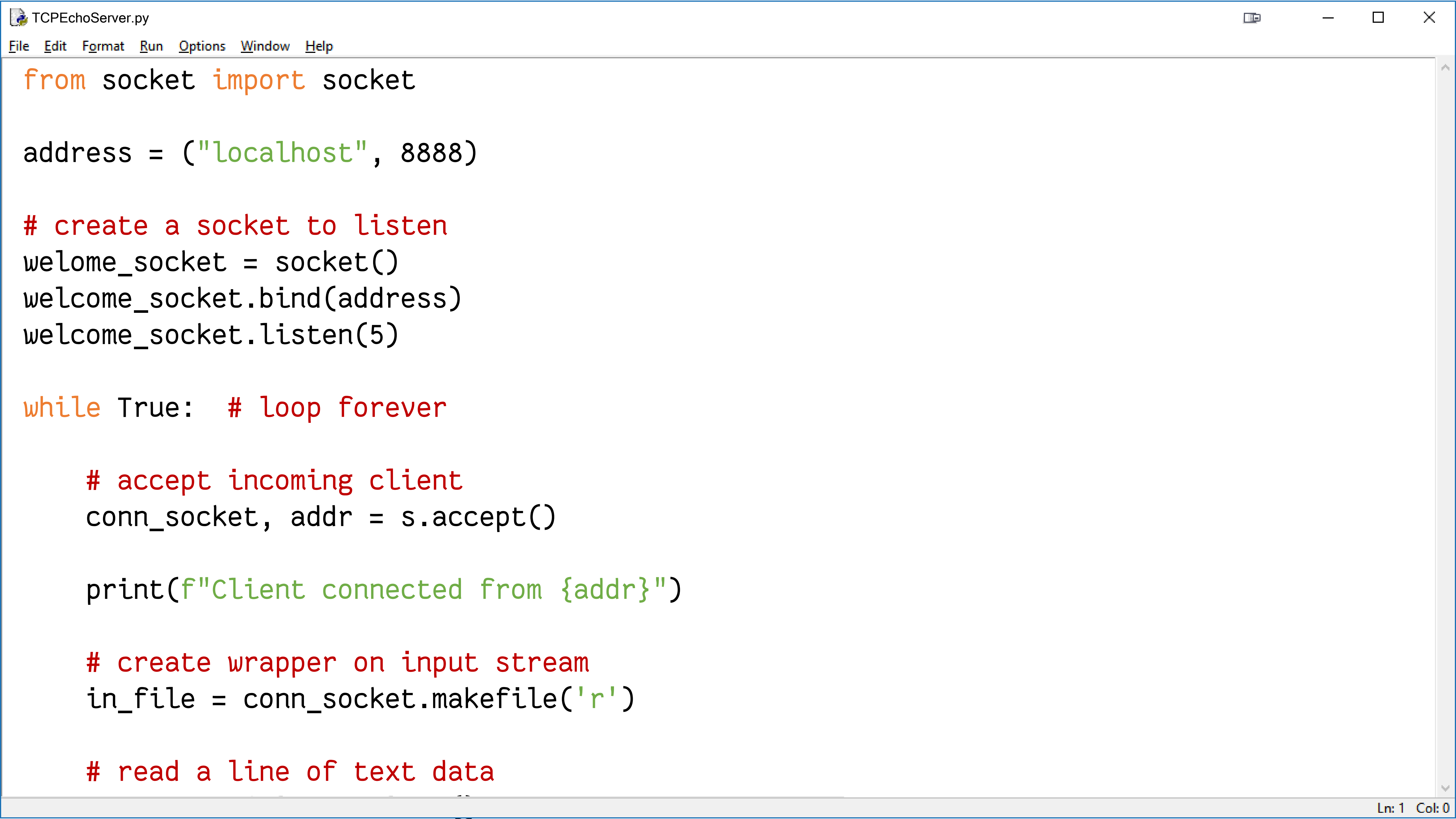
For reading both text and bytes?

`java.io.DataInputStream`

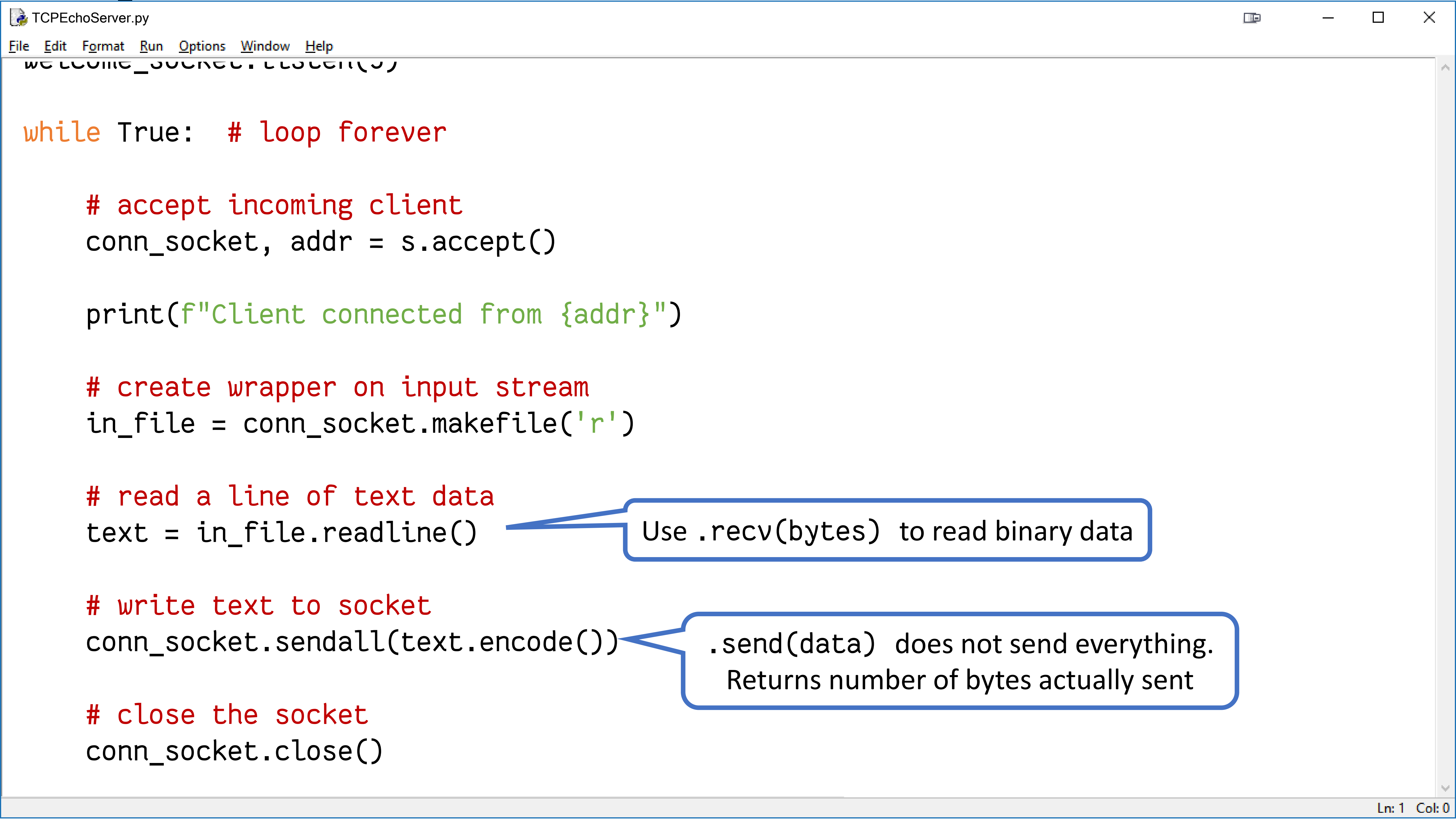
- `.read(byte[] b)`
- `.readLine()` ← Deprecated
- Because method fails to handle UTF8 properly
- But still safe if you are only reading ASCII, e.g. HTTP Headers

`java.net.HttpURLConnection`

- parses header for you



```
from socket import socket

address = ("localhost", 8888)

# create a socket to listen
welome_socket = socket()
welcome_socket.bind(address)
welcome_socket.listen(5)

while True:  # loop forever

    # accept incoming client
    conn_socket, addr = s.accept()

    print(f"Client connected from {addr}")

    # create wrapper on input stream
    in_file = conn_socket.makefile('r')

    # read a line of text data
```



```
welcome_socket.listen(5)

while True:  # loop forever

    # accept incoming client
    conn_socket, addr = s.accept()

    print(f"Client connected from {addr}")

    # create wrapper on input stream
    in_file = conn_socket.makefile('r')

    # read a line of text data
    text = in_file.readline()

    # write text to socket
    conn_socket.sendall(text.encode())

    # close the socket
    conn_socket.close()
```

Use .recv(bytes) to read binary data

.send(data) does not send everything.
Returns number of bytes actually sent

Lecture 2: Roadmap

2.1 Principles of Network Applications

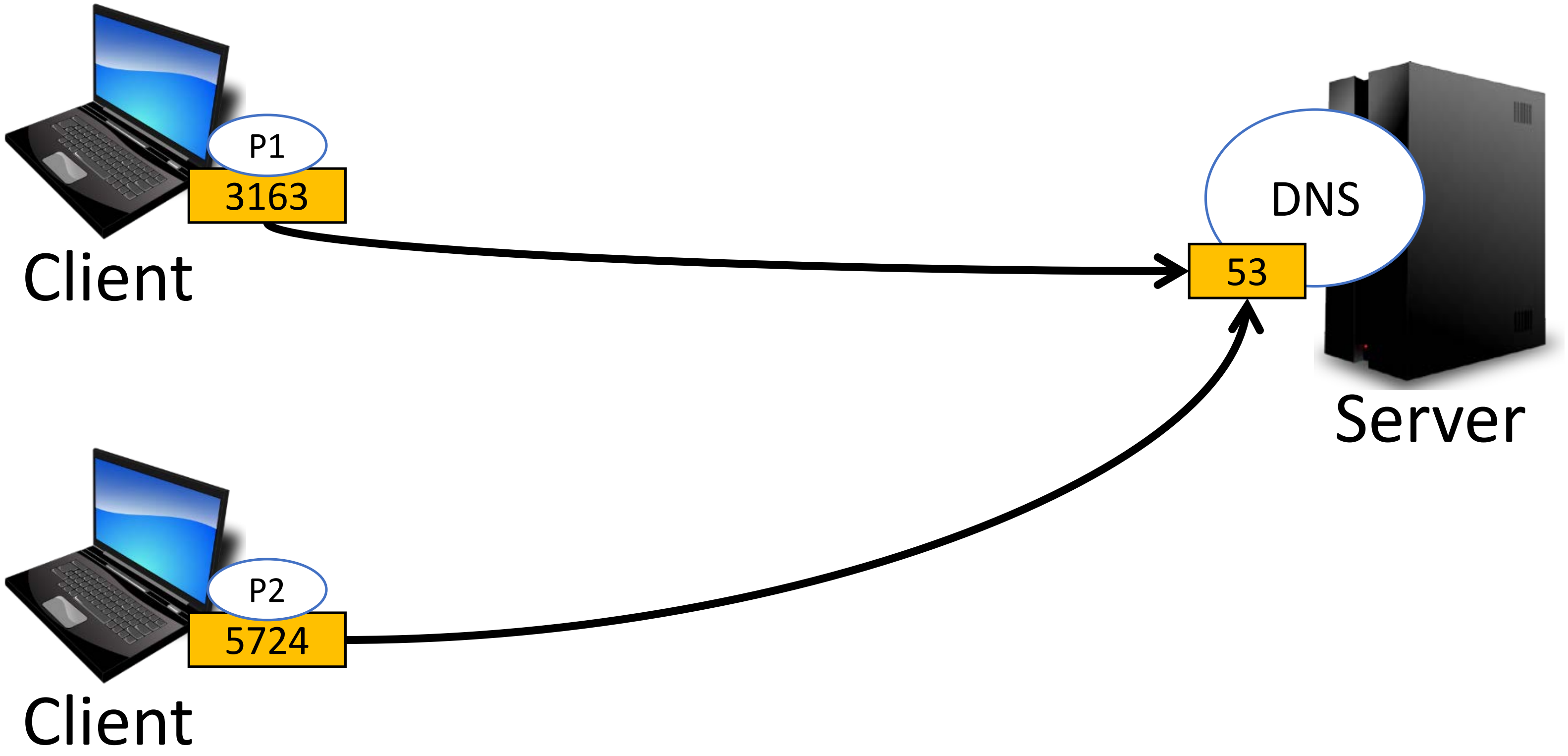
2.2 Web and HTTP

2.5 DNS

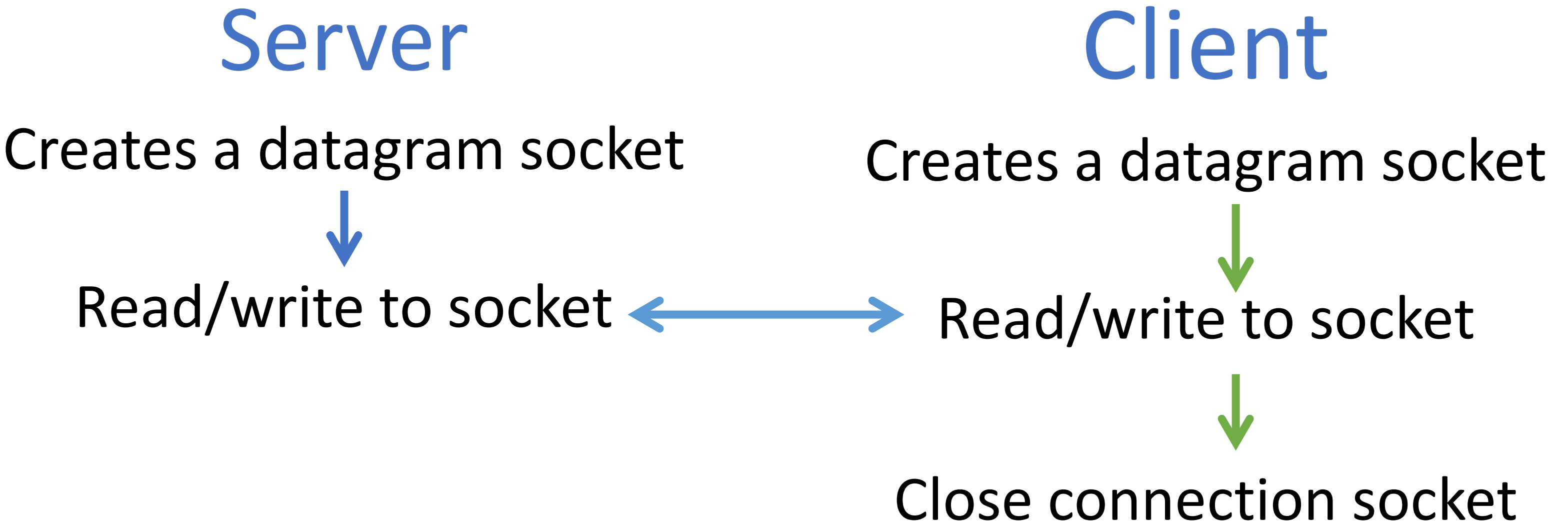
2.7 Socket programming with TCP

2.8 Socket programming with UDP

UDP Socket Programming



Client/Server Interaction



Server

Creates a datagram socket

Read/write to socket

Client

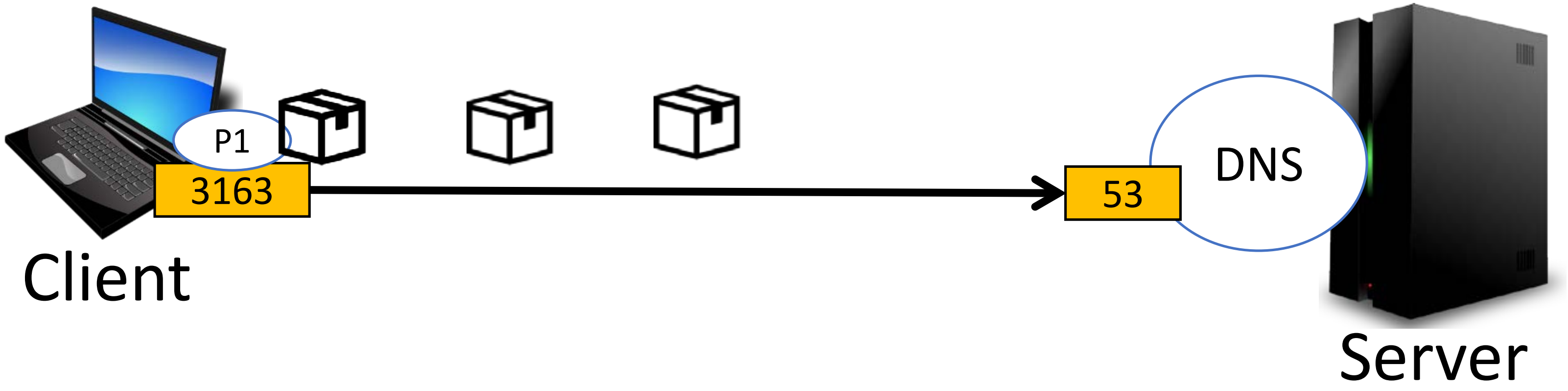
Creates a datagram socket

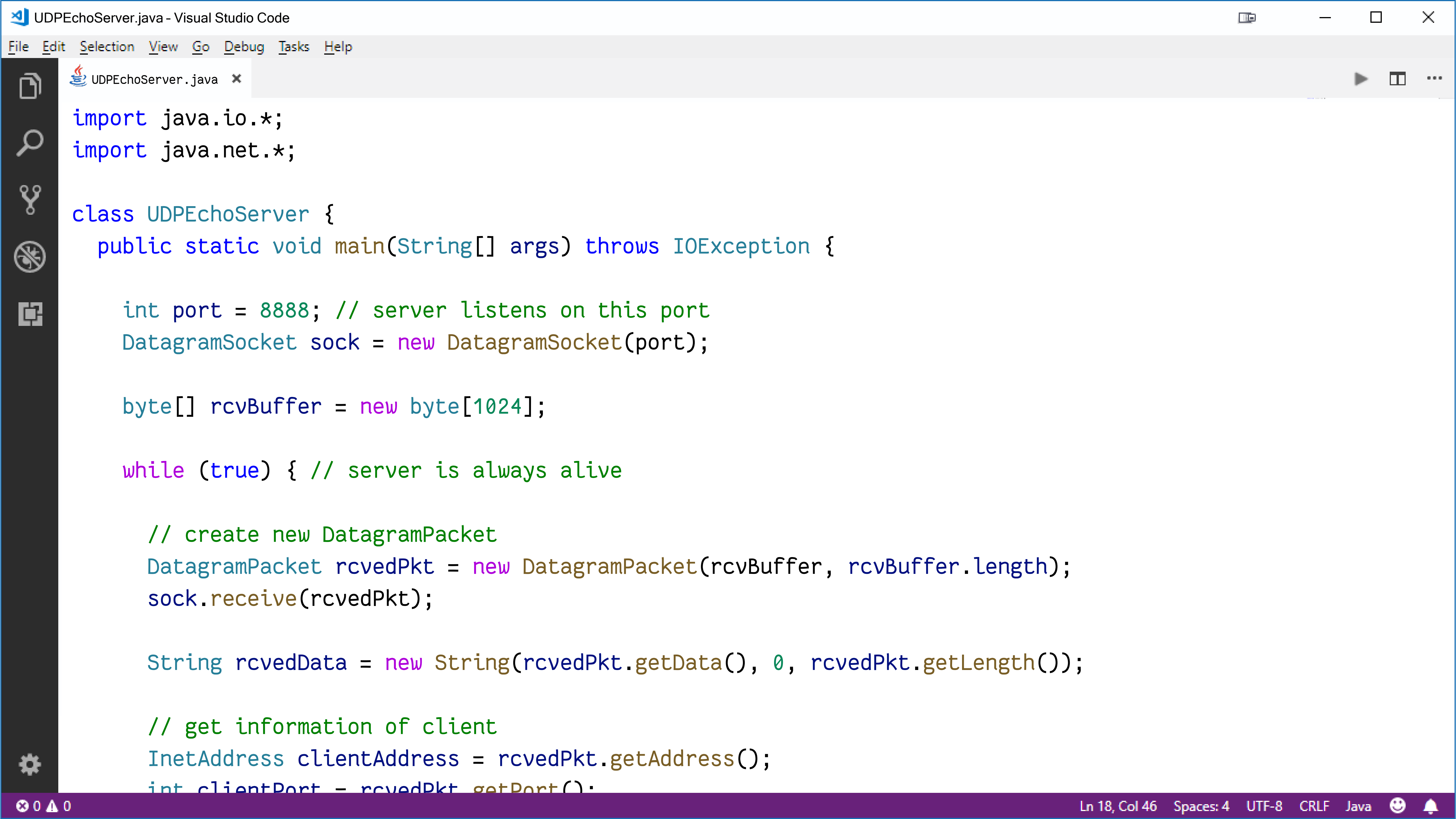
Read/write to socket

Close connection socket

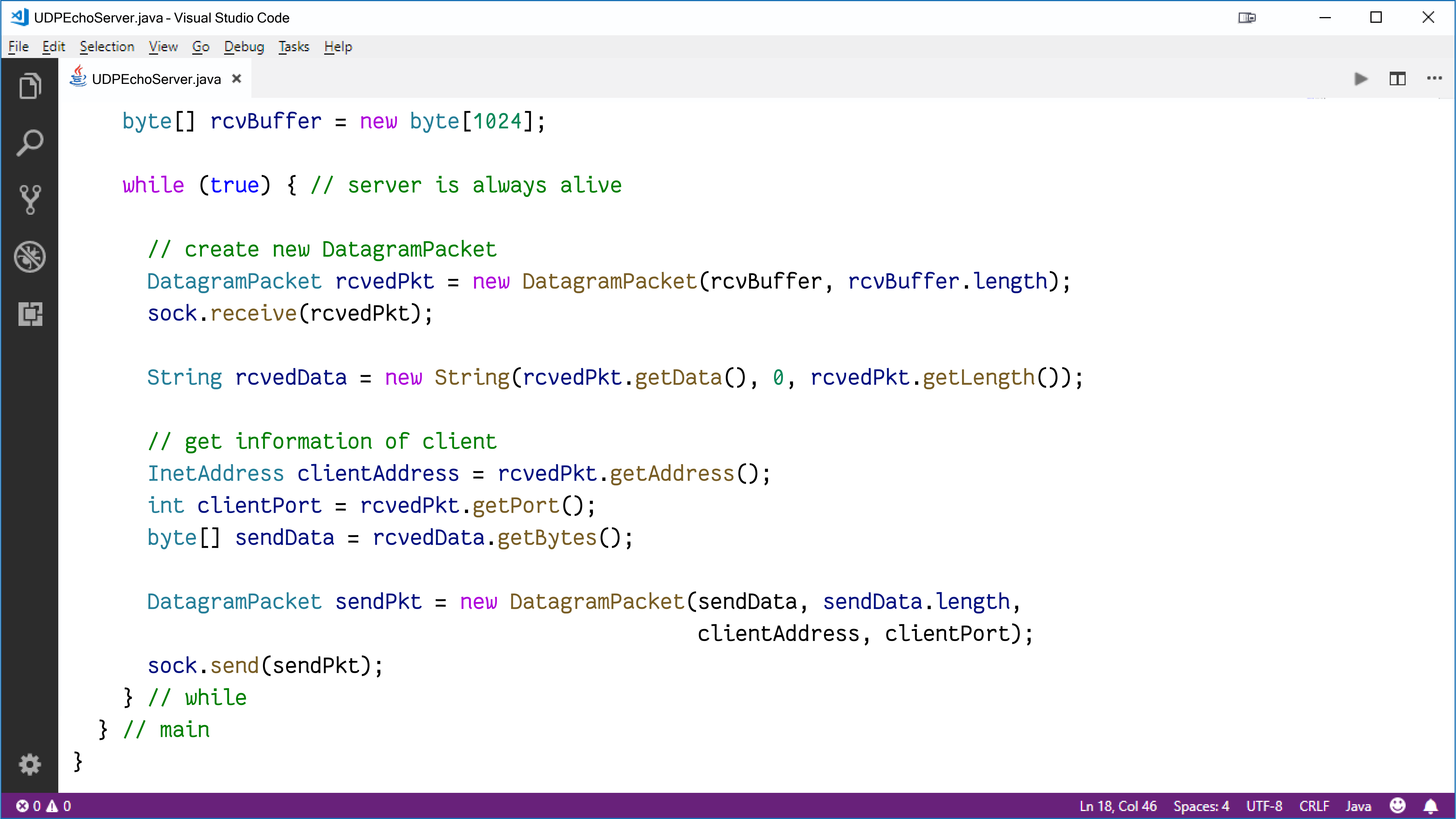
Live demo

In UDP, data is sent as datagrams (packets)

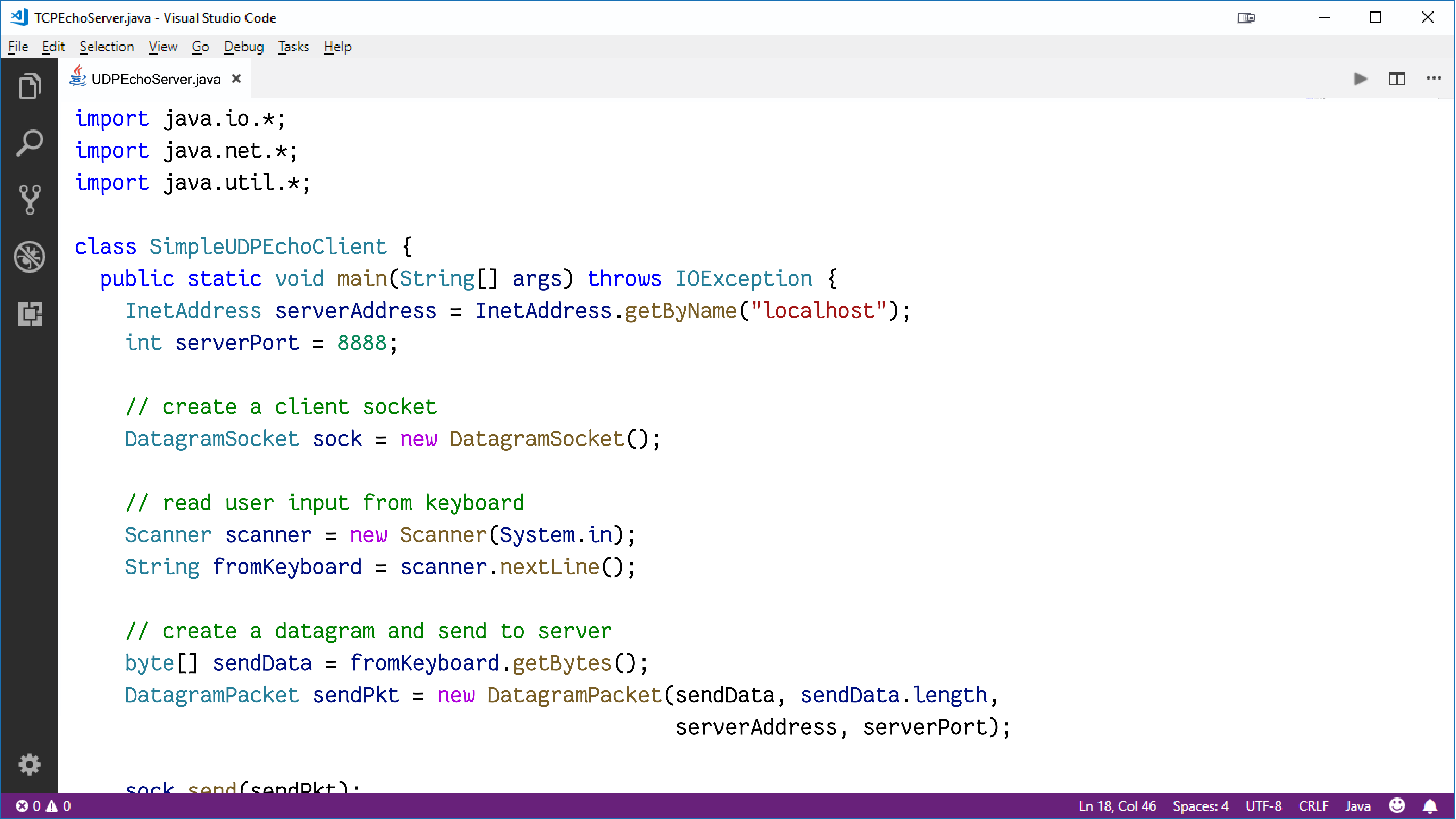




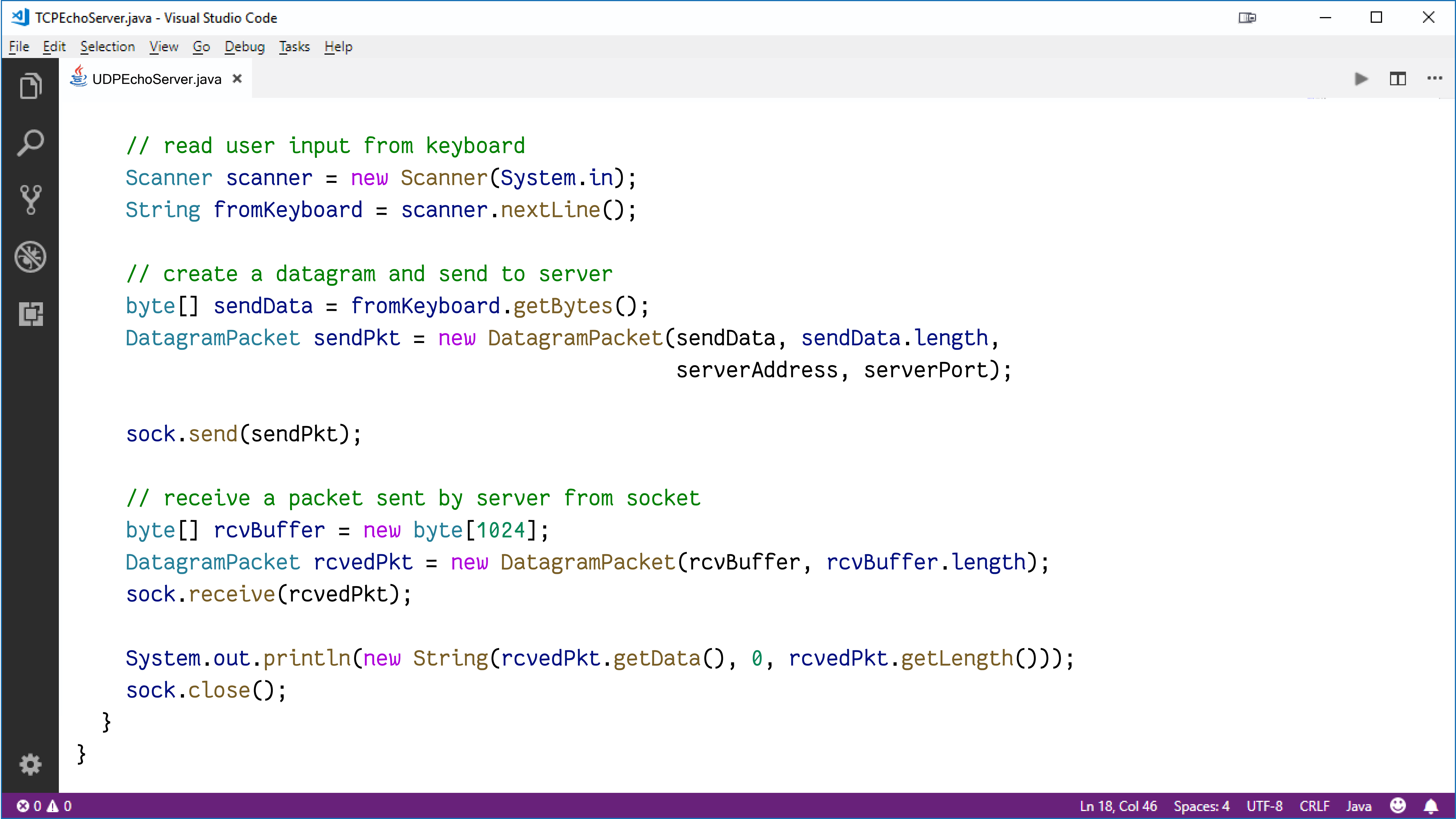
```
import java.io.*;
import java.net.*;

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

    int port = 8888; // server listens on this port
    DatagramSocket sock = new DatagramSocket(port);

    byte[] rcvBuffer = new byte[1024];

    while (true) { // server is always alive

      // create new DatagramPacket
      DatagramPacket rcvedPkt = new DatagramPacket(rcvBuffer, rcvBuffer.length);
      sock.receive(rcvedPkt);

      String rcvedData = new String(rcvedPkt.getData(), 0, rcvedPkt.getLength());

      // get information of client
      InetAddress clientAddress = rcvedPkt.getAddress();
      int clientPort = rcvedPkt.getPort();
```



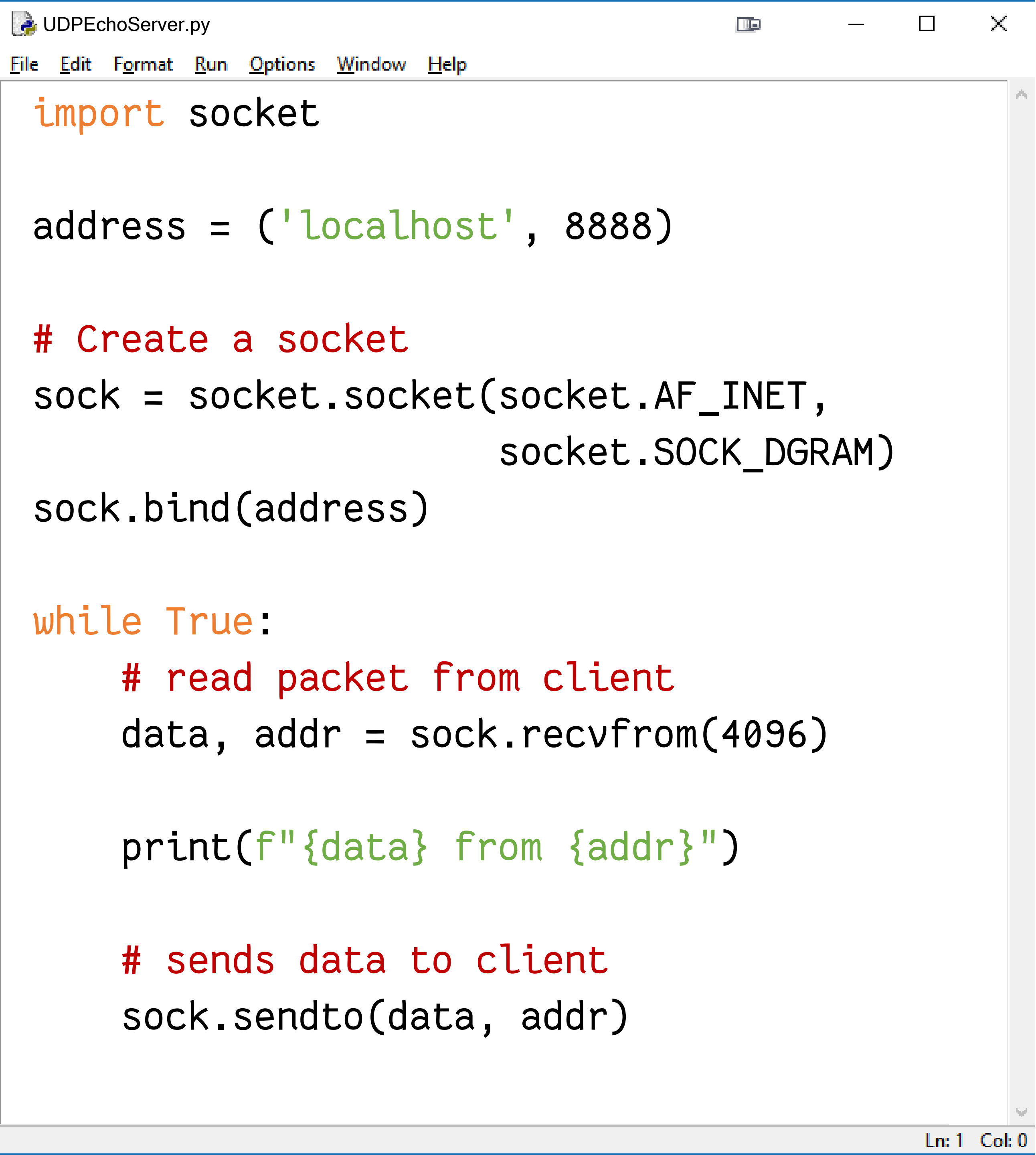
```java
    byte[] rcvBuffer = new byte[1024];

    while (true) { // server is always alive

      // create new DatagramPacket
      DatagramPacket rcvedPkt = new DatagramPacket(rcvBuffer, rcvBuffer.length);
      sock.receive(rcvedPkt);

      String rcvedData = new String(rcvedPkt.getData(), 0, rcvedPkt.getLength());

      // get information of client
      InetAddress clientAddress = rcvedPkt.getAddress();
      int clientPort = rcvedPkt.getPort();
      byte[] sendData = rcvedData.getBytes();

      DatagramPacket sendPkt = new DatagramPacket(sendData, sendData.length,
                                                  clientAddress, clientPort);

      sock.send(sendPkt);
    } // while
  } // main
}
```


TCPEchoServer.java - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
UDPEchoServer.java ×
⊗ 0 ⚠ 0
Ln 18, Col 46 Spaces: 4 UTF-8 CRLF Java

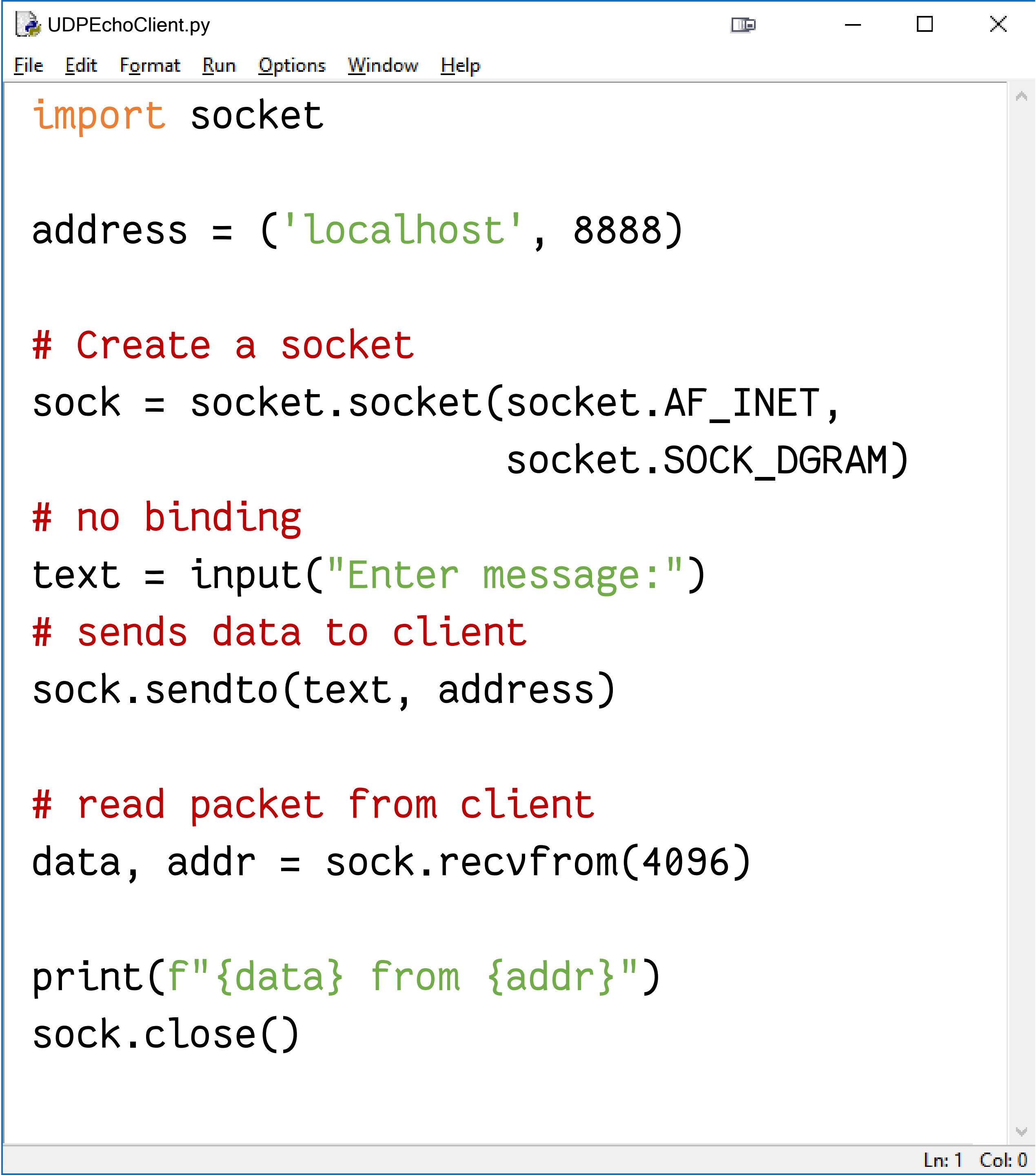
```java
import java.io.*;
import java.net.*;
import java.util.*;

class SimpleUDPEchoClient {
  public static void main(String[] args) throws IOException {
    InetAddress serverAddress = InetAddress.getByName("localhost");
    int serverPort = 8888;

    // create a client socket
    DatagramSocket sock = new DatagramSocket();

    // read user input from keyboard
    Scanner scanner = new Scanner(System.in);
    String fromKeyboard = scanner.nextLine();

    // create a datagram and send to server
    byte[] sendData = fromKeyboard.getBytes();
    DatagramPacket sendPkt = new DatagramPacket(sendData, sendData.length,
                                                serverAddress, serverPort);

    sock.send(sendPkt);
```


TCPEchoServer.java - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
UDPEchoServer.java
⊗ 0 ⚠ 0
Ln 18, Col 46 Spaces: 4 UTF-8 CRLF Java

```
    // read user input from keyboard
    Scanner scanner = new Scanner(System.in);
    String fromKeyboard = scanner.nextLine();

    // create a datagram and send to server
    byte[] sendData = fromKeyboard.getBytes();
    DatagramPacket sendPkt = new DatagramPacket(sendData, sendData.length,
                                                serverAddress, serverPort);


    sock.send(sendPkt);

    // receive a packet sent by server from socket
    byte[] rcvBuffer = new byte[1024];
    DatagramPacket rcvedPkt = new DatagramPacket(rcvBuffer, rcvBuffer.length);
    sock.receive(rcvedPkt);

    System.out.println(new String(rcvedPkt.getData(), 0, rcvedPkt.getLength()));
    sock.close();
  }
}
```



```
import socket

address = ('localhost', 8888)

# Create a socket
sock = socket.socket(socket.AF_INET,
                     socket.SOCK_DGRAM)
sock.bind(address)

while True:
    # read packet from client
    data, addr = sock.recvfrom(4096)

    print(f"{data} from {addr}")

    # sends data to client
    sock.sendto(data, addr)
```



```
import socket

address = ('localhost', 8888)

# Create a socket
sock = socket.socket(socket.AF_INET,
                     socket.SOCK_DGRAM)
# no binding
text = input("Enter message:")
# sends data to client
sock.sendto(text, address)

# read packet from client
data, addr = sock.recvfrom(4096)

print(f"{data} from {addr}")
sock.close()
```

Layering

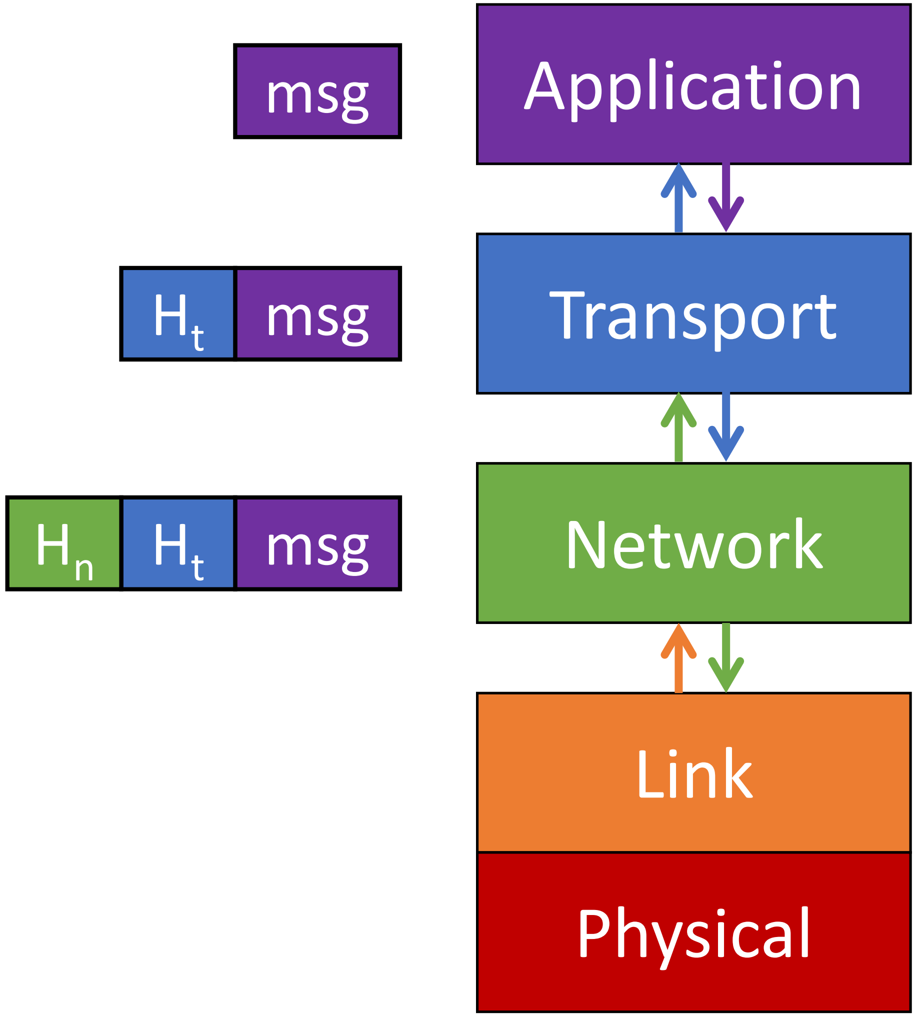
message

segment

H_t contains source and destination port

datagram

H_n contains source and destination IP address



If TCP communicates in streams, how does it make segments?

TCP Socket vs. UDP Socket

- In TCP, two processes communicate as if there is a pipe between them. The pipe remains in place until one of the two processes closes it.
  - When one of the processes wants to send more bytes to the other process, it simply writes data to that pipe.
  - The sending process doesn't need to attach a destination IP address and port number to the bytes in each sending attempt as the logical pipe has been established (which is also reliable).
- In UDP, programmers need to form UDP datagram packets explicitly and attach destination IP address / port number to every packet.

Lecture 3: Summary

Socket programming

- TCP socket
  - When contacted by client, server TCP creates new socket.
  - Server uses (client IP + port #) to distinguish clients.
  - When client creates its socket, client TCP establishes connection to server TCP.
- UDP socket
  - Server use one socket to serve all clients.
  - No connection is established before sending data.
  - Sender explicitly attaches destination IP address and port # to each packet.
  - Transmitted data may be lost or received out-of-order.