

CSE2101: Object Oriented Programming-II (Java)

Lecture 14

Socket Programming

What is a socket?

3

- Socket is an interface between application and network (the lower levels of the protocol stack)
 - The application creates a socket
 - The socket *type* dictates the style of communication
 - reliable vs. best effort
 - connection-oriented vs. connectionless
- Once configured, the application can
 - pass data to the socket for network transmission
 - receive data from the socket (transmitted through the network by some other host)

Addresses, Ports and Sockets

4

- Like apartments and mailboxes
 - You are the application
 - Street address of your apartment building is the **IP address**
 - Your mailbox is the **port**
 - The post-office is the network
 - The **socket** is the key that gives you access to the right mailbox
- Q: How do you choose which port a socket connects to?

Addresses, Ports and Sockets

5

- Choose a port number that is registered for general use, from 1024 to 49151
 - Do not use ports 1 to 1023. These ports are reserved for use by the Internet Assigned Numbers Authority (IANA)
 - Avoid using ports 49152 through 65535. These are dynamic ports that operating systems use randomly. If you choose one of these ports, you risk a potential port conflict

Client-Server Paradigm

6

- Server waits for client to request a connection.
- Client contacts server to establish a connection.
- Client sends request.
- Server sends reply.
- Client and/or server terminate connection.

Two types of Communication

7

- Connection-oriented
 - Setup the link before communication.
 - Similar to the phone call. We need the phone number and receiver.
- Connectionless
 - No link needed to be set up before communication.
 - Similar to send a letter. We need the address and receiver.

Most popular types of sockets

8

- TCP socket
 - reliable delivery
 - in-order guaranteed
 - connection-oriented
 - bidirectional
- UDP socket
 - unreliable delivery
 - no order guarantees
 - no notion of “connection”
 - app indicates destination for each packet
 - can send or receive

JAVA TCP Sockets

9

- In Package java.net
 - java.net.Socket
 - Implements client sockets (also called just “sockets”).
 - An endpoint for communication between two machines.
 - Constructor and Methods
 - Socket(String host, int port): Creates a stream socket and connects it to the specified port number on the named host.
 - InputStream getInputStream()
 - OutputStream getOutputStream()
 - close()
 - java.net.ServerSocket
 - Implements server sockets.
 - Waits for requests to come in over the network.
 - Performs some operation based on the request.
 - Constructor and Methods
 - ServerSocket(int port)
 - Socket Accept(): Listens for a connection to be made to this socket and accepts it. This method blocks until a connection is made.

TCPClient.java

10

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

class TCPClient {

    public static void main(String argv[]) throws
Exception {
        String sentence;
        String modifiedSentence;
        BufferedReader inFromUser = new
BufferedReader(new InputStreamReader(System.in));
        Socket clientSocket = new Socket("127.0.0.1",
6789);
        DataOutputStream outToServer = new
DataOutputStream(clientSocket.getOutputStream());
```


TCPClient.java

11

```
BufferedReader inFromServer = new
    BufferedReader(new
        InputStreamReader(clientSocket.getInputStream()))
;
    sentence = inFromUser.readLine();
    outToServer.writeBytes(sentence + '\n');
    modifiedSentence = inFromServer.readLine();
    System.out.println("FROM SERVER: " +
modifiedSentence);
    clientSocket.close();
}
}
```


TCPServer.java

12

```
import java.io.*;
import java.net.*;
class TCPServer {

    public static void main(String argv[]) throws
Exception {
        String clientSentence;
        String capitalizedSentence;
        ServerSocket welcomeSocket = new
ServerSocket(6789);
        while (true) {

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

```
BufferedReader inFromClient = new BufferedReader(new
    InputStreamReader(connectionSocket.getInputStream()))
;
    DataOutputStream outToClient = new
    DataOutputStream(connectionSocket.getOutputStream());

    clientSentence = inFromClient.readLine();

    capitalizedSentence =
    clientSentence.toUpperCase() + '\n';

    outToClient.writeBytes(capitalizedSentence);
}
}
```

Socket Programming with UDP

14

- UDP
 - Connectionless and unreliable service.
 - There isn't an initial handshaking phase.
 - Doesn't have a pipe.
 - transmitted data may be received out of order, or lost
- Socket Programming with UDP
 - No need for a welcoming socket.
 - No streams are attached to the sockets.
 - the sending hosts creates “packets” by attaching the IP destination address and port number to each batch of bytes.
 - The receiving process must unravel to received packet to obtain the packet's information bytes.

- DatagramSocket:
 - **send** - Sends a datagram packet from this socket.
 - **receive** - Receives a datagram packet from this socket.
 - **setSoTimeout** - Enable/disable the specified timeout, in milliseconds.
 - **connect** - Connects the socket to a remote address for this socket.

- In Package java.net
 - java.net.DatagramSocket
 - A socket for sending and receiving datagram packets.
 - Constructor and Methods
 - DatagramSocket(int port): Constructs a datagram socket and binds it to the specified port on the local host machine.
 - void receive(DatagramPacket p)
 - void send(DatagramPacket p)
 - void close()

UDPClient.java

17

```
import java.io.*;
import java.net.*;
class UDPClient {

    public static void main(String args[]) throws
Exception {
        BufferedReader inFromUser = new
BufferedReader(new InputStreamReader(System.in));
        DatagramSocket clientSocket = new
DatagramSocket();
        InetAddress IPAddress =
InetAddress.getByName("localhost");
        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];
        String sentence = inFromUser.readLine();
        sendData = sentence.getBytes();
```



```
DatagramPacket sendPacket = new
    DatagramPacket(sendData, sendData.length,
        IPAddress, 9876);
    clientSocket.send(sendPacket);
    DatagramPacket receivePacket = new
    DatagramPacket(receiveData, receiveData.length);
    clientSocket.receive(receivePacket);
    String modifiedSentence = new
    String(receivePacket.getData());
    System.out.println("FROM SERVER:" +
        modifiedSentence);
    clientSocket.close();
}
}
```

UDPServer.java

19

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

class UDPServer {

    public static void main(String args[]) throws
    Exception {
        DatagramSocket serverSocket = new
        DatagramSocket(9876);
        byte[] receiveData = new byte[1024];
        byte[] sendData = new byte[1024];
        while (true) {
            DatagramPacket receivePacket = new
            DatagramPacket(receiveData, receiveData.length);
            serverSocket.receive(receivePacket);
```

UDPServer.java

20

```
String sentence = new String(receivePacket.getData());
    InetAddress IPAddress =
receivePacket.getAddress();
    int port = receivePacket.getPort();
    String capitalizedSentence =
sentence.toUpperCase();
    sendData = capitalizedSentence.getBytes();
    DatagramPacket sendPacket = new
DatagramPacket(sendData, sendData.length, IPAddress,
port);
    serverSocket.send(sendPacket);
}
}
}
```


Thank you