

UDP & TCP

TUTORIAL 2

UDP Programming

- Connection less protocol which means that one program can send a load of packets to another and that would be the end of the relationship
- There is no guarantee of delivery, ordering, or duplicate protection of data
- UDP is faster because error recovery is not attempted.
- UDP is suitable for applications that need fast, efficient transmission, such as games.

UDP Programming

From class notes:

sender program

create a datagram socket and
bind it to any local port;
place data in a byte array;
create a datagram packet, specifying
the data array and the receiver's
address;
invoke the send method of the
socket with a reference to the
datagram packet;

receiver program

create a datagram socket and
bind it to a specific local port;
create a byte array for receiving the data;
create a datagram packet, specifying
the data array;
invoke the receive method of the
socket with a reference to the
datagram packet;

UDP Programming

So we need to create 2 classes:

- A client class (sender)
- A server class (receiver)

UDP Programming

- Create a new Project named UDPProgramming
- In your src default package, create 2 classes:
 - Client
 - Server
- Check the main method for both

UDP Programming - Client

1. Create a Datagram socket and bind it to any local port
2. Place data in byte array
3. Create a datagram packet and specify data array and receiver address
4. Invoke the send method with a reference to the packet

```
1 import java.net.*;
2 import java.io.*;
3 public class UDPCClient {
4
5     public static void main(String args[]){
6         //args give message contents and destination hostname
7         DatagramSocket aSocket = null;
8         try{
9             aSocket = new DatagramSocket();
10            byte [] m = "hello".getBytes();
11            InetAddress aHost = InetAddress.getByName("localhost");
12            int serverPort = 6789;
13
14            DatagramPacket request = new DatagramPacket(m, "hello".length(), aHost, serverPort);
15            aSocket.send(request);
16
17            byte[] buffer = new byte[1000];
18            DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
19            aSocket.receive(reply);
20            System.out.println("Reply: " + new String (reply.getData()));
21        }
22        catch(SocketException e){
23            System.out.println("Socket: " + e.getMessage());
24        }
25        catch(IOException e){
26            System.out.println("IO: " + e.getMessage());
27        }
28        finally {
29            if(aSocket != null) aSocket.close();
30        }
31    }
32 }
33 }
```

UDP Programming - Server

1. Create a datagram socket and bind it to a port
2. Create a byte array to receive the data
3. Create a datagram packet and specify the data array
4. Invoke the receive method of the socket with a reference the datagram packet

UDPServer.java ✕

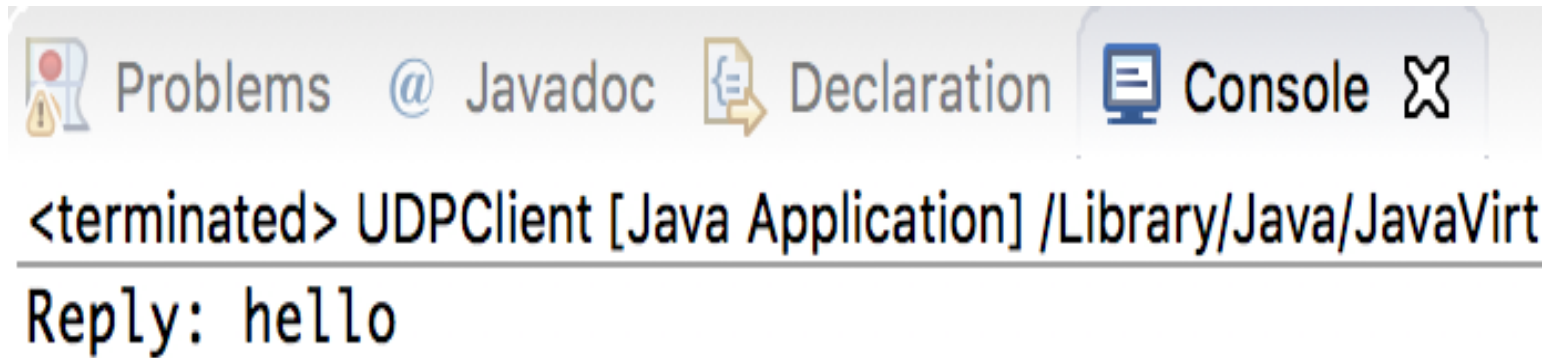
```
1 import java.net.*;
2 import java.io.*;
3 public class UDPServer {
4
5     public static void main(String[] args) {
6         DatagramSocket aSocket = null;
7         try{
8             //create a socket at agreed port
9             aSocket = new DatagramSocket(6789);
10            byte[] buffer = new byte[1000];
11
12            while(true){
13                DatagramPacket request = new DatagramPacket(buffer, buffer.length);
14                aSocket.receive(request);
15                DatagramPacket reply = new DatagramPacket(request.getData(), request.getLength(),
16                    request.getAddress(), request.getPort());
17                aSocket.send(reply);
18            }
19        }
20        catch(SocketException e){
21            System.out.println("Socket: " + e.getMessage());
22        }
23        catch(IOException e){
24            System.out.println("IO: " + e.getMessage());
25        }
26        finally {
27            if(aSocket != null) aSocket.close();
28        }
29    }
30 }
31
32 }
```

UDP Programming - Run

How to make it work?

1. Run the UDPServer
2. Run the UDPClient

UDP Programming - Output



The screenshot shows a standard IDE interface with a toolbar at the top containing icons for Problems, Javadoc, Declaration, Console, and a close button. The Console tab is active, displaying the output of a Java application. The output text is as follows:

```
<terminated> UDPClient [Java Application] /Library/Java/JavaVirt  
Reply: hello
```

TCP Programming

- Connection oriented protocol
- It provides reliable, ordered, and error-checked delivery of data
- TCP is suited for applications that require high reliability, and transmission time is relatively less critical.
- The speed for TCP is slower than UDP.
- WWW, email, remote administration and file transfer rely on TCP

TCP Programming

Write a TCP client and TCP server.

The client sends a message to the server, the server reverses that messages and sends it back.

Create a class TCPServer that accepts input string from client and reverses it.

Create a class TCPClient that gets values from the user.

It also sends requests to server to reverse the string.

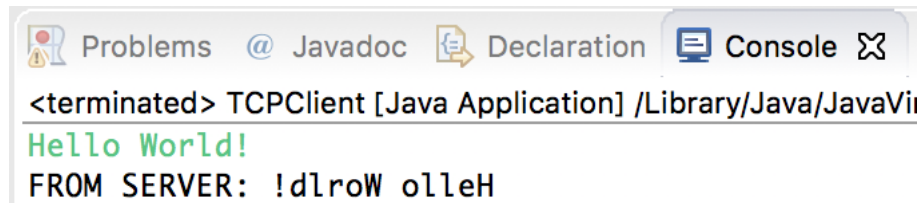
```
1 import java.net.*;
2 import java.io.*;
3 public class TCPServer {
4
5     public static void main(String[] args) throws IOException{
6         //Initialization
7         String server_inputMsg = "";
8         String server_reverseMsg = "";
9
10        //Create a socket at agreed port(5000)
11        ServerSocket serverSocket = new ServerSocket(5000);
12
13        while(true){
14            //Establish the connection between the client and the server
15            Socket connectionSocket = serverSocket.accept();
16
17            //Get InputStream at server to get values from client
18            BufferedReader inFromClient =
19                new BufferedReader(new InputStreamReader(connectionSocket.getInputStream()));
20
21            //Get OutputStream at server to send values to client
22            DataOutputStream outToClient = new DataOutputStream(connectionSocket.getOutputStream());
23
24            //Get the input message from client and then print
25            server_inputMsg = inFromClient.readLine();
26            System.out.println("Received: " + server_inputMsg);
27
28            //Reverse the string
29            server_reverseMsg = new StringBuffer(server_inputMsg).reverse().toString() + "\n";
30
31            //Send the result to the client
32            outToClient.writeBytes(server_reverseMsg);
33        }
34    }
35 }
```

```
1 import java.net.*;
2
3 public class TCPClient {
4
5     public static void main(String[] args) throws Exception{
6         //Initialization
7         String client_inputMsg = "";
8         String client_reverseMsg = "";
9
10        //Create a socket at agreed port
11        Socket clientSocket = new Socket("localhost", 5000);
12
13        //Get OutputStream at client to send values to server
14        DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream());
15
16        //Get InputStream at client to get values from server
17        BufferedReader inFromServer = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
18
19        //Get the input from the user
20        BufferedReader inFromUser = new BufferedReader( new InputStreamReader(System.in));
21        client_inputMsg = inFromUser.readLine();
22
23        //Send the message received from user to server to be reversed
24        outToServer.writeBytes(client_inputMsg + "\n");
25
26        client_reverseMsg = inFromServer.readLine();
27        System.out.println("FROM SERVER: " + client_reverseMsg);
28
29        //close the socket for good measure
30        clientSocket.close();
31    }
32 }
```

TCP Programming

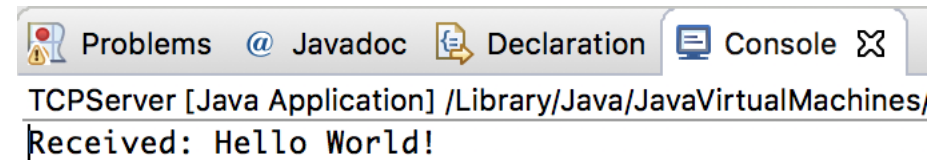
- To get Output:
 - Run the TCPServer first
 - Run the TCPClient next
 - Enter a string at the console of TCPClient

At TCPClient

A screenshot of the Eclipse IDE's console window for the TCPClient application. The window title is "TCPClient [Java Application] /Library/Java/JavaVirtualMachines/...". The console output shows the application has terminated, followed by the text "Hello World!" in green, and then "FROM SERVER: !dlroW olleH" in black.

```
<terminated> TCPClient [Java Application] /Library/Java/JavaVirtualMachines/  
Hello World!  
FROM SERVER: !dlroW olleH
```

At TCPServer

A screenshot of the Eclipse IDE's console window for the TCPServer application. The window title is "TCPServer [Java Application] /Library/Java/JavaVirtualMachines/...". The console output shows the text "Received: Hello World!" in black.

```
TCPServer [Java Application] /Library/Java/JavaVirtualMachines/  
Received: Hello World!
```


References

1. https://www.tutorialspoint.com/java/java_networking.htm
2. <https://docs.oracle.com/javase/tutorial/networking/overview/networking.html>

Lecture Note: Network and Process Communication, Dr. Jayakumar

<http://www.cs.rutgers.edu/~pxk/417/notes/sockets/udp.html>

Distributed Systems Concepts and Design, Fifth Edition, ISBN: 0-13-214301-1