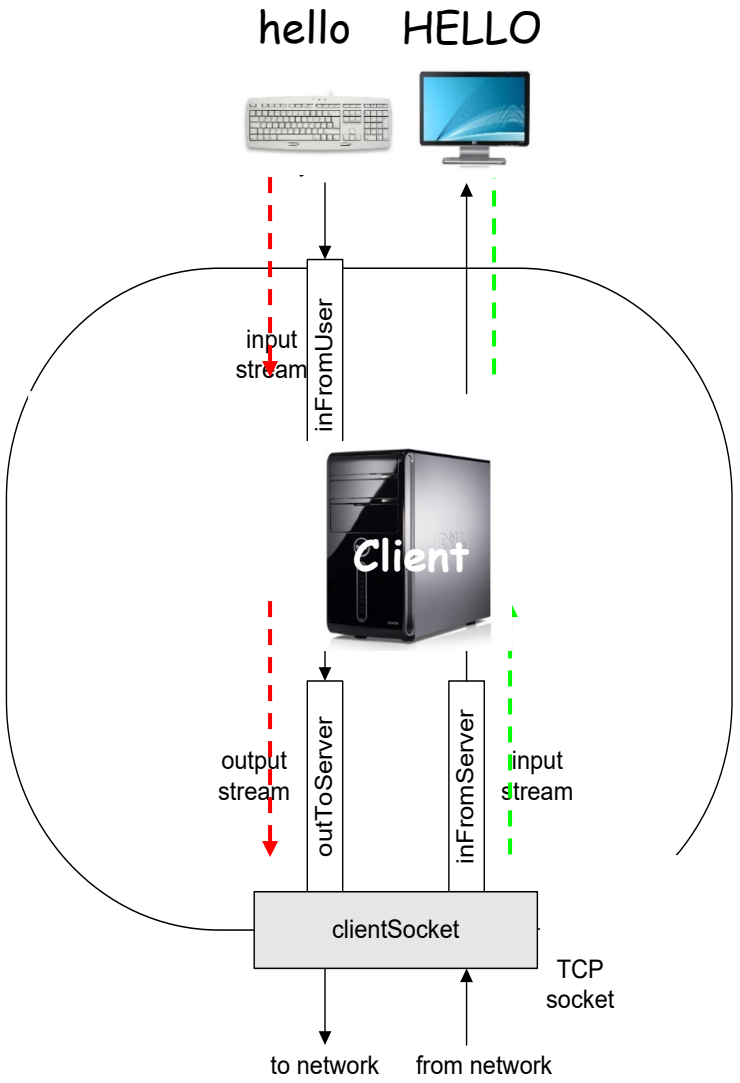
Example [Java socket programming – unicast communication]

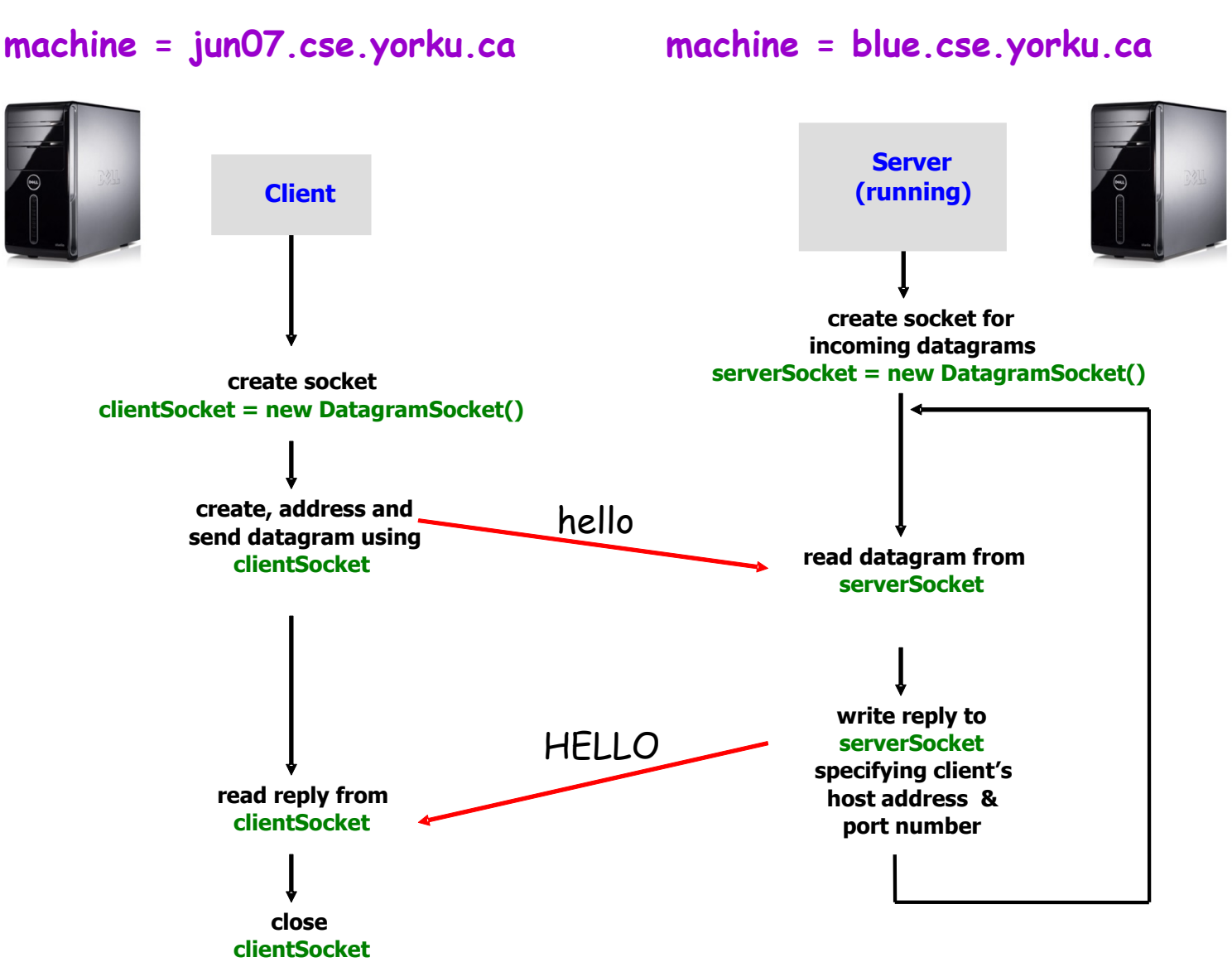
Use the following simple client/server application to demonstrate socket programming for both TCP and UDP:

1) **A client reads a line from its standard input (keyboard) and sends line out through its socket to the server.**
2) **The server reads a line from its connection socket.**
3) **The server converts the line to upper case.**
4) **The server sends the modified line out through its socket to the client.**
5) **The client reads the modified line from its socket and prints the line on its standard output (monitor).**

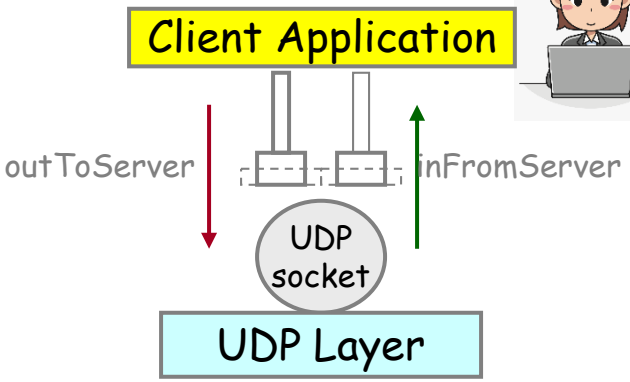


Server: HELLO

Java Socket Programming with UDP



Java Socket Programming with UDP (cont.)




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

class UDPClient {

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

1     BufferedReader inFromUser = new BufferedReader (new
2     InputStreamReader(System.in));

3     byte[] sendData = new byte[1204];
4     byte[] receiveData = new byte[1204];

5     DatagramSocket clientSocket = new DatagramSocket();

6     InetAddress IPAddress = InetAddress.getByName("blue.cse.yorku.ca");

      ..........................................................

7     String sentence = inFromUser.readLine();

8     sendData = sentence.getBytes();
```

create input stream attached to keyboard

byte arrays sendData and receiveData will hold data that client sends and receives in datagrams

create client socket - host does NOT contact server upon execution of this line!

translate hostname to IP address using DNS

store inFromUser to sendData buffer

server runs on blue

Java Socket Programming with UDP (cont.)

server runs port 7777

construct datagram with data, length, server IP address and port number

send datagram

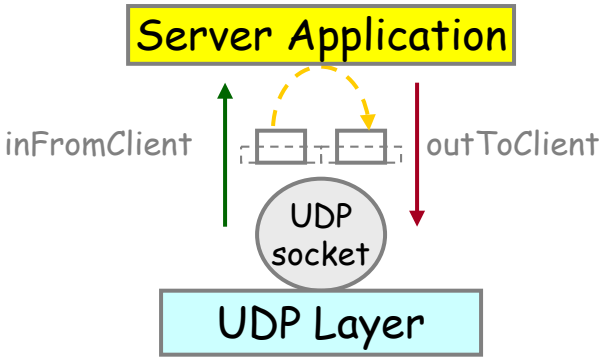
while waiting for response, create placeholder for packet

read datagram

extract data from receivePacket buffer and perform type conversion

```
 9  DatagramPacket sendPacket =
10  new DatagramPacket(sendData, sendData.length, IPAddress, 7777);

11  clientSocket.send(sendPacket);

12  DatagramPacket receivePacket =
13  new DatagramPacket(receiveData, receiveData.length);

14  clientSocket.receive(receivePacket);

15  String modifiedSentence = new String(receivePacket.getData());

16  System.out.println("FROM SERVER: "+modifiedSentence.trim());

17  clientSocket.close();

    }
}
```

Java Socket Programming with UDP (cont.)



server on blue.cs.yorku.ca …

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

class UDPServer {

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

1       DatagramSocket serverSocket = new DatagramSocket(7777);

2       byte[] receiveData = new byte[1024];
3       byte[] sendData = new byte[1024];

        ..........................................................

4       while(true) {

5           DatagramPacket receivePacket =
6           new DatagramPacket(receiveData, receiveData.length);

7           serverSocket.receive(receivePacket);
```

create datagram socket at port 7777

Why do we have to specify port number in this case?!

Java Socket Programming with UDP (cont.)


```
8  String sentence = new String(receivePacket.getData());

9  InetAddress IPAddress = receivePacket.getAddress();

10 int port = receivePacket.getPort();

11 String capitalizedSentence = sentence.toUpperCase() + '\n';

12 sendData = capitalizedSentence.getBytes();

13 DatagramPacket sendPacket = new DatagramPacket(sendData,
14 sendData.length, IPAddress, port);

15 serverSocket.send(sendPacket);

   }
  }
 }
```

get IP address of the sender

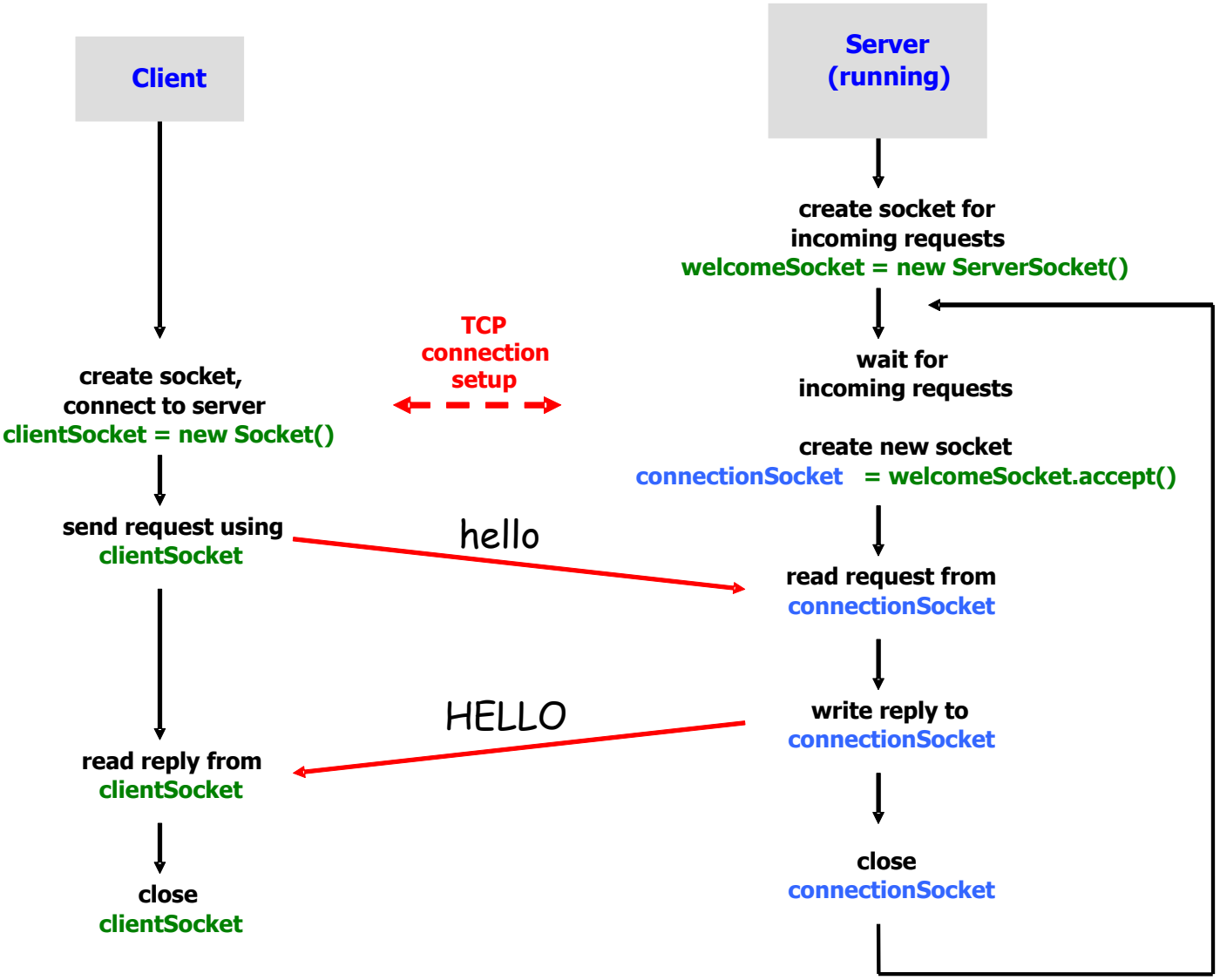
get port number of the sender

create datagram to send to client

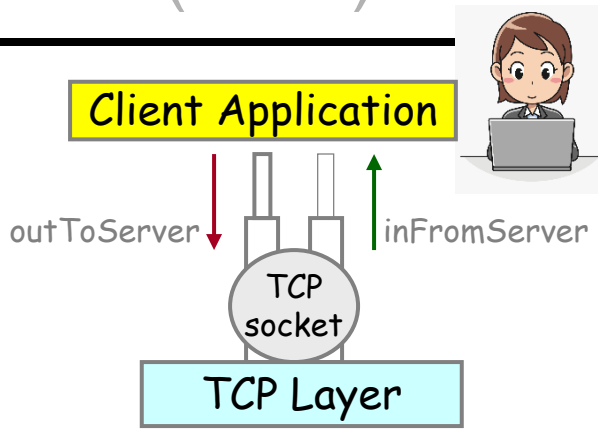
write datagram to socket

loop back and wait for another datagram

Java Socket Programming with TCP



Java Socket Programming with TCP (cont.)




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

class TCPClient {

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

1       String sentence;
2       String modifiedSentence;

3       BufferedReader inFromUser = new BufferedReader (new
4       InputStreamReader(System.in));

5       Socket clientSocket = new Socket("blue.cs.yorku.ca", 5555);

6       DataOutputStream outToServer =
7       new DataOutputStream(clientSocket.getOutputStream());

8       BufferedReader inFromServer = new BufferedReader (new
9       InputStreamReader(clientSocket.getInputStream()));
```

create input stream attached to keyboard

create socket; connect it to server

server runs on blue - port 5555

create output stream attached to socket

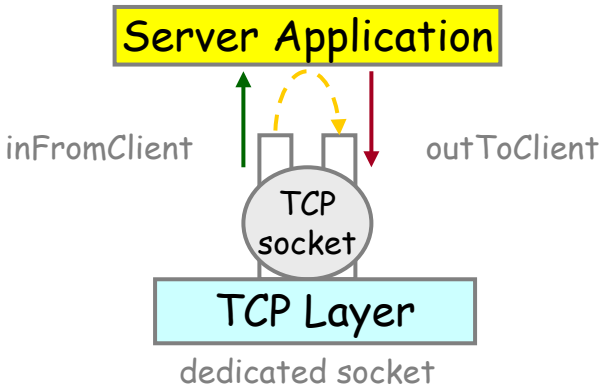
create input stream attached to socket

place line typed by user into 'sentence'; 'sentence' continues to gather characters until a carriage return

send line to server

print to monitor line read from server

```
10  sentence = inFromUser.readLine();
11  outToServer.writeBytes(sentence + '\n');
12  modifiedSentence = inFromServer.readLine();
13  System.out.println("FROM SERVER: "+modifiedSentence);
14  clientSocket.close();
    }
}
```



server on blue.cs.yorku.ca …

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

class TCPServer {

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

1     String clientSentence;
2     String capitalizedSentence;

3     ServerSocket welcomeSocket = new ServerSocket(5555);

4     while(true) {

5       Socket dedicatedSocket = welcomeSocket.accept();

6       BufferedReader inFromClient = new BufferedReader (new
7       InputStreamReader(dedicatedSocket.getInputStream()));
```

create welcoming socket at port 5555

wait for contact-request by clients

once a request arrives, allocate new socket

create & attach input stream to new socket

Java Socket Programming with TCP (cont.)

create & attach output stream to new socket

read from socket

write to socket

end of while loop – wait for another client to connect

```
8  DataOutputStream outToClient =
9  new DataOutputStream(dedicatedSocket.getOutputStream());

10 clientSentence = inFromClient.readLine();

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

12 outToClient.writeBytes(capitalizedSentence);

        }
    }
}
```

NOTE: This version of TCP Server is NOT actually serve clients concurrently, but it can be easily modified (with threads) to do so.