Java Socket Programming

Faculty of Information Technology
Hanoi University

Java Sockets Programming

- The package java.net provides support for sockets programming (and more).
- Typically you import everything defined in this package with:

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

Classes

`InetAddress`

`Socket`

`ServerSocket`

`DatagramSocket`

`DatagramPacket`

InetAddress class

- static methods you can use to create new InetAddress objects.
  - `getByName(String host)`
  - `getAllByName(String host)`
  - `getLocalHost()`

```
InetAddress x = InetAddress.getByName(
                                 “cse.unr.edu”);
```

- Throws **UnknownHostException**

```
try {

  InetAddress a = InetAddress.getByName(hostname);
  System.out.println(hostname + ":" + a.getHostAddress());

} catch (UnknownHostException e) {

      System.out.println("No address found for " + hostname);

}
```

Socket class

- Corresponds to active TCP sockets only!
  - client sockets
  - socket returned by accept();
- Passive sockets are supported by a different class:
  - ServerSocket
- UDP sockets are supported by
  - DatagramSocket

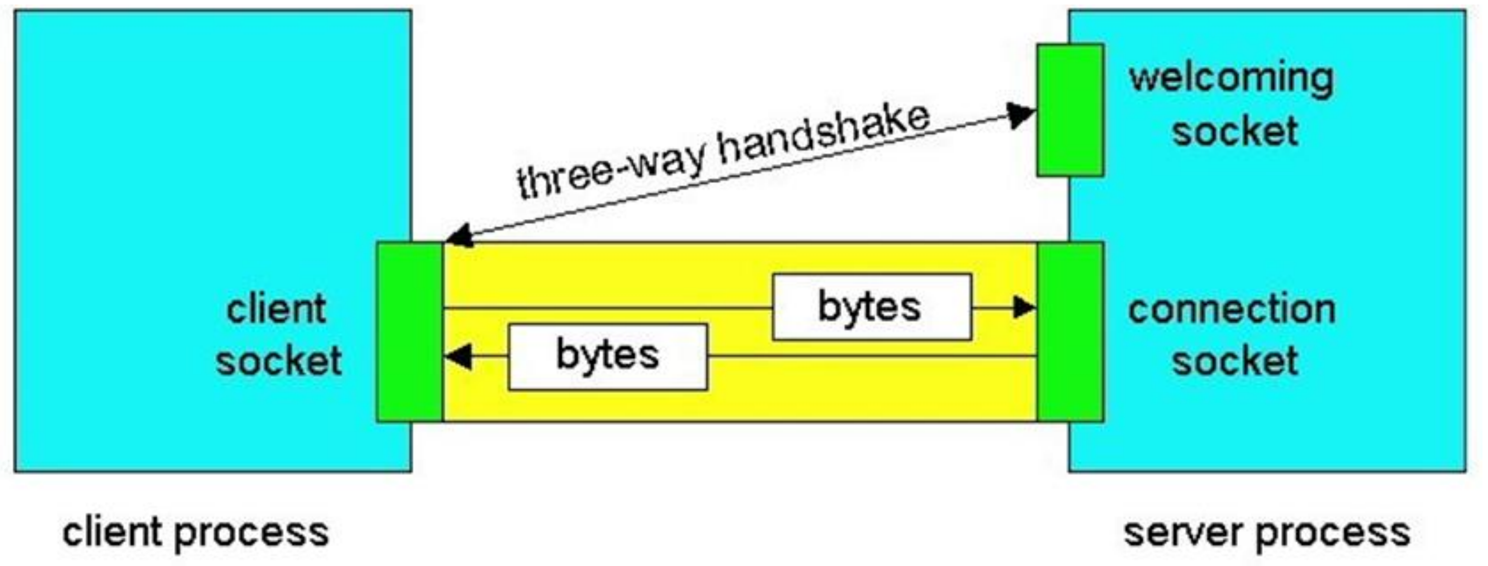
JAVA TCP Sockets (Client Socket)

- 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()***

ServerSocket

- 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.

Sockets



Client socket, welcoming socket (passive) and connection socket (active)

Socket Constructors

- Constructor creates a TCP connection to a named TCP server.
  - There are a number of constructors:

```
Socket(InetAddress server, int port);

Socket(InetAddress server, int port,
          InetAddress local, int localport);

Socket(String hostname, int port);
```

Socket Methods

```
void close();
InetAddress getInetAddress();
InetAddress getLocalAddress();
InputStream getInputStream();
OutputStream getOutputStream();
```

- Lots more (setting/getting socket options, partial close, etc.)

Socket I/O

- Socket I/O is based on the Java I/O support
  - in the package `java.io`

- InputStream and OutputStream are abstract classes
  - common operations defined for all kinds of InputStreams, OutputStreams…

InputStream Basics

```
// reads some number of bytes and
// puts in buffer array b
int read(byte[] b);

// reads up to len bytes
int read(byte[] b, int off, int len);
```

Both methods can throw `IOException`.

Both return -1 on EOF.

OutputStream Basics

```
// writes b.length bytes
void write(byte[] b);

// writes len bytes starting
// at offset off
void write(byte[] b, int off, int len);
```

Both methods can throw **`IOException`**.

ServerSocket Class (TCP Passive Socket)

- Constructors:

```
ServerSocket(int port);

ServerSocket(int port, int backlog);

ServerSocket(int port, int backlog,
               InetAddress bindAddr);
```

ServerSocket Methods

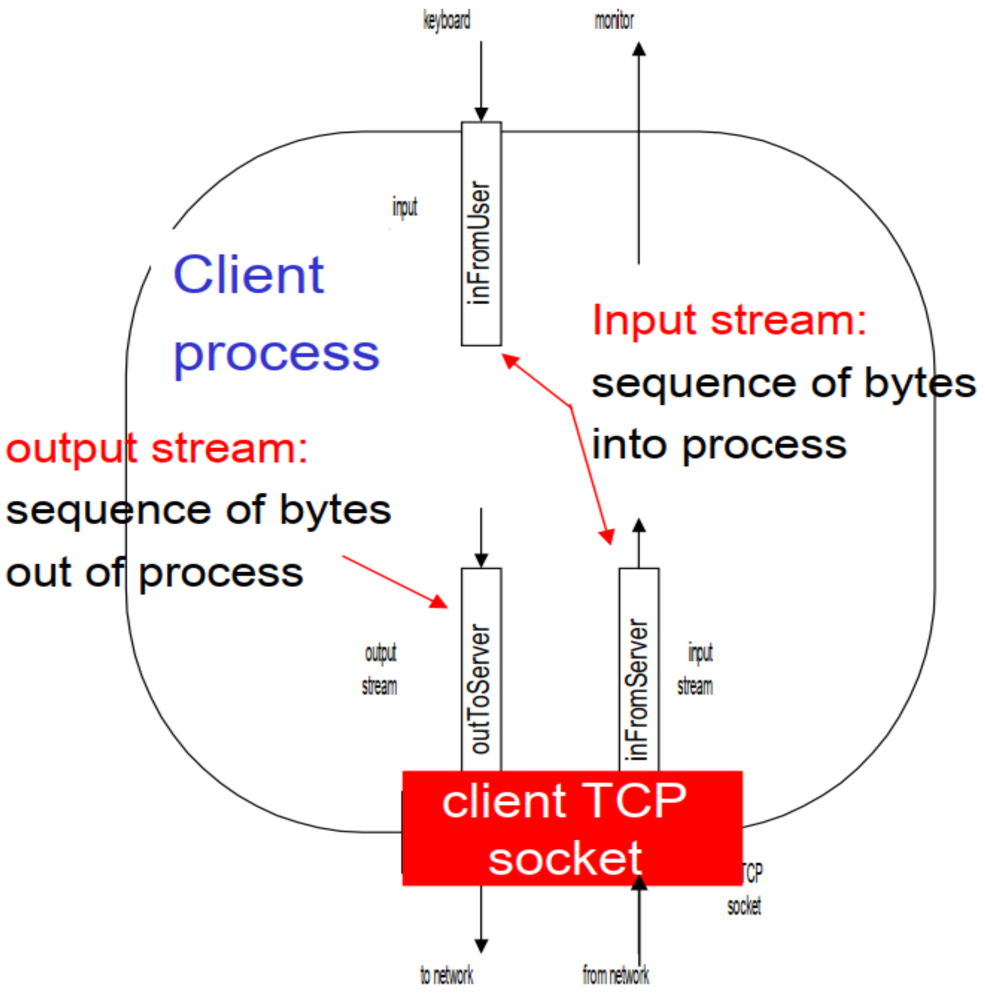
```
Socket accept();

void close();

InetAddress getInetAddress();

int getLocalPort();
```

throw `IOException, SecurityException`

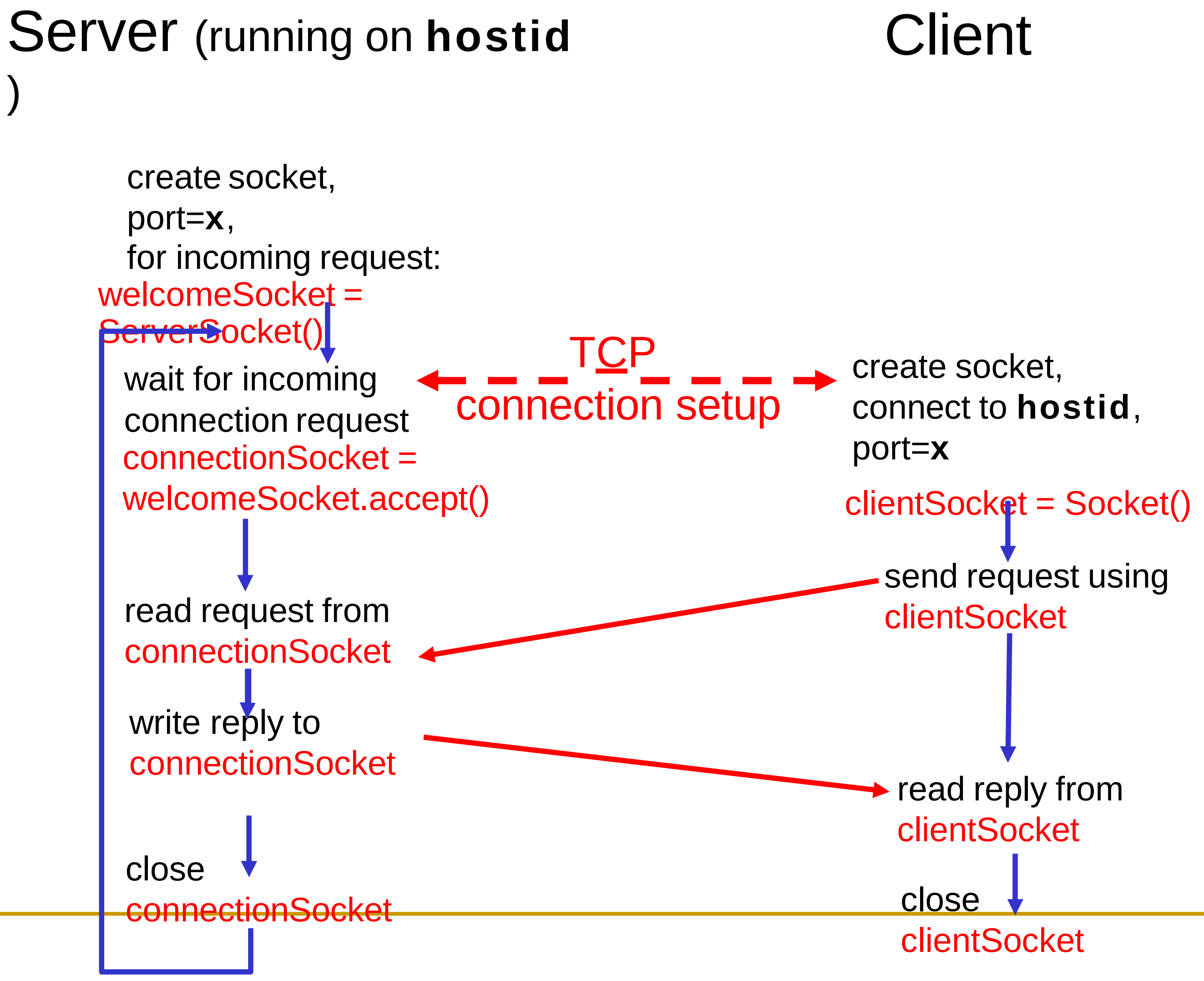
Socket programming with TCP

Example client-server app:

- client reads line from standard input (**inFrom User** stream) , sends to server via socket (**outTo Server** stream)
- server reads line from socket
- server converts line to uppercase, sends back to client
- client reads, prints modified line from socket (**inFrom Server** stream)



Client/server socket interaction: TCP


Server (running on **hostid**)
Client
create socket, port=**x**, for incoming request:
welcomeSocket = ServerSocket()
wait for incoming connection request
connectionSocket = welcomeSocket.accept()
TCP connection setup
create socket, connect to **hostid**, port=**x**
clientSocket = Socket()
send request using clientSocket
read request from connectionSocket
write reply to connectionSocket
read reply from clientSocket
close connectionSocket
close clientSocket

Sample Echo Server

TCPEchoServer.java
And TCPClient.java

Save both files, compile and run on separate terminal.
First TCPEchoServer and then TCPClient

Based on code from:
    TCP/IP Sockets in Java

TCPEchoServer.java

```
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);
System.out.println("Server is waiting to accept user... ");

while(true) {

Socket connectionSocket = welcomeSocket.accept();
        System.out.println("Accept a client!");
```

```
      BufferedReader inFromClient = new BufferedReader(new
InputStreamReader(connectionSocket.getInputStream()));
      DataOutputStream outToClient = new
DataOutputStream(connectionSocket.getOutputStream());
      clientSentence = inFromClient.readLine();
      capitalizedSentence = clientSentence.toUpperCase() +
"\n";

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

TCPClient.java

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

public class TCPClient {
      public static void main(String argv[]) throws Exception {
            String sentence;
            String modifiedSentence;
            BufferedReader inFromUser = new BufferedReader(new
InputStreamReader(System.in));
            while(true) {
                  System.out.println("Please enter your
message");
                  Socket clientSocket = new
Socket("localhost", 6789);
                  DataOutputStream outToServer = new
DataOutputStream  (clientSocket.getOutputStream());
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();
                }
            }
}
```

UDP Sockets

- `DatagramSocket` class
- `DatagramPacket` class needed to specify the payload
  - incoming or outgoing

Socket Programming with UDP

- UDP
  - Connectionless and unreliable service.
  - There isn’t an initial handshaking phase.
  - 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.

JAVA UDP Sockets

- 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()

DatagramSocket Constructors

```
DatagramSocket();

DatagramSocket(int port);

DatagramSocket(int port, InetAddress a);
```

All can throw SocketException or SecurityException

Datagram Methods

```
void connect(InetAddress, int port);

void close();

void receive(DatagramPacket p);

void send(DatagramPacket p);
```

Lots more!

DatagramPacket

- Contain the payload
  - (a byte array, length of byte array, InetAddress, port)
- Can also be used to specify the destination address
  - when not using connected mode UDP

DatagramPacket Constructors

For receiving:

```
DatagramPacket( byte[] buf, int len);
```

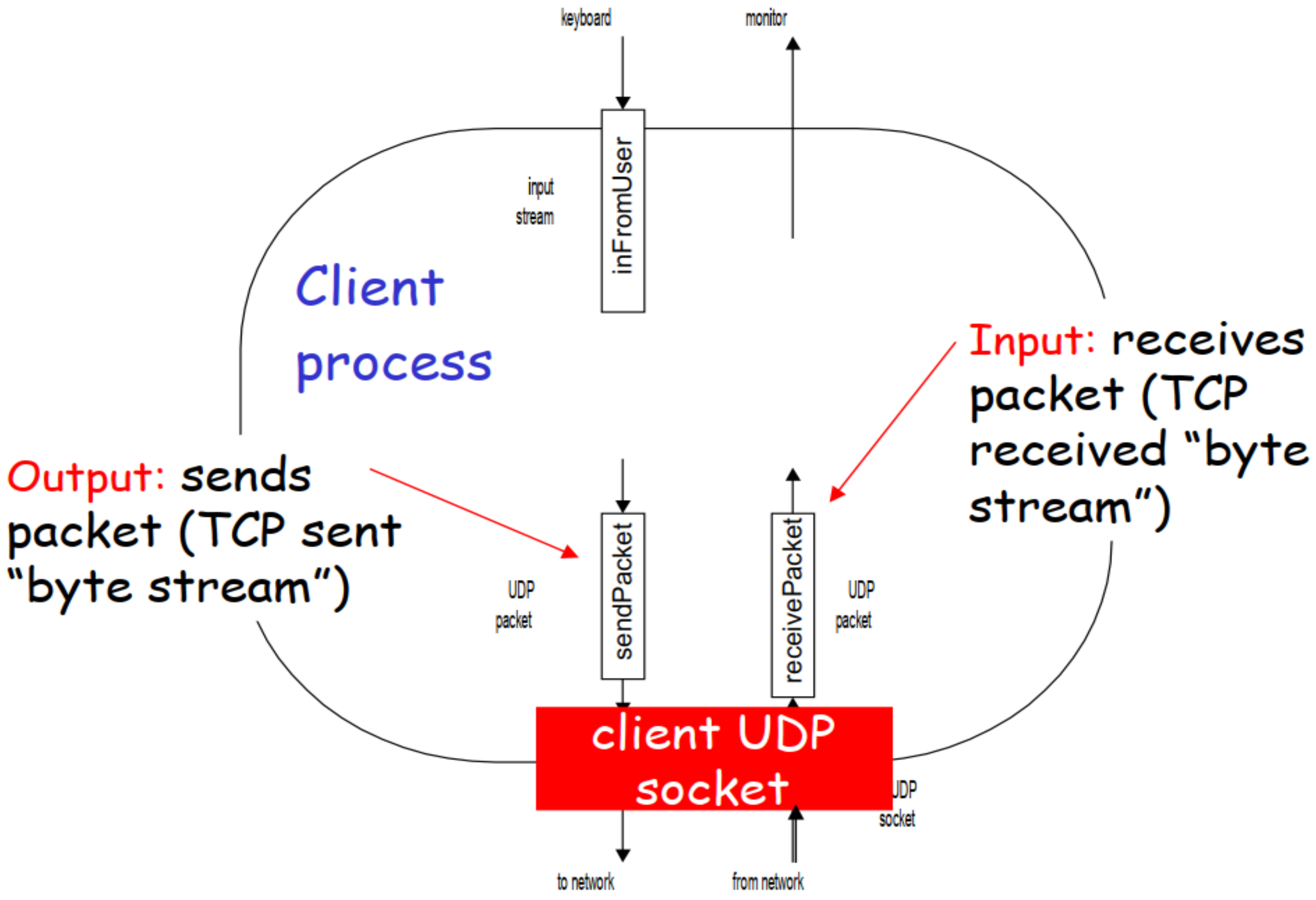
For sending:

```
DatagramPacket( byte[] buf, int len
                InetAddress a, int port);
```

DatagramPacket methods

```
byte[] getData();
void setData(byte[] buf);

void setAddress(InetAddress a);
void setPort(int port);

InetAddress getAddress();
int getPort();
```

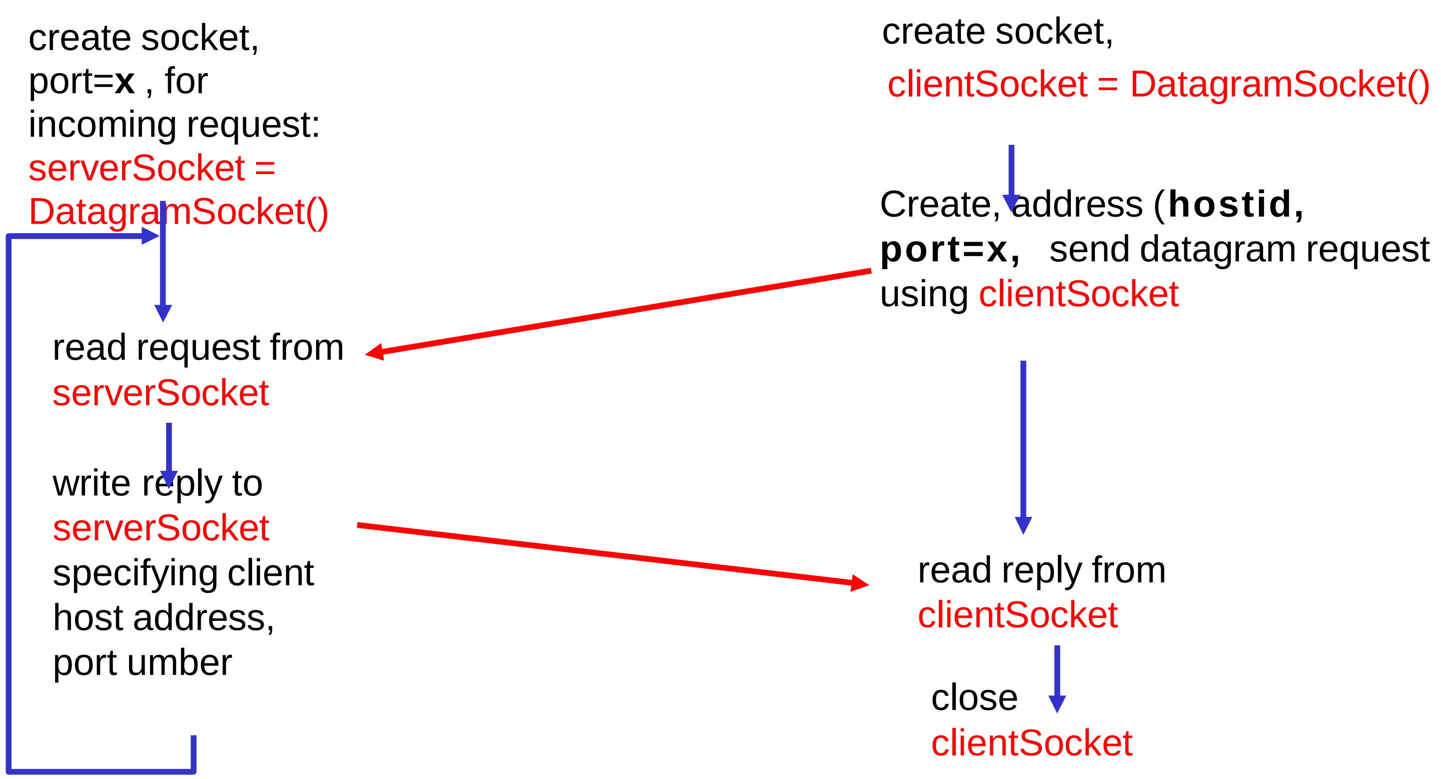
Example: Java client (UDP)



Client/server socket interaction: UDP

Server (running on **hostid**)

Client


create socket, port=**x** , for incoming request:
serverSocket = DatagramSocket()
read request from serverSocket
write reply to serverSocket specifying client host address, port umber
create socket,
clientSocket = DatagramSocket()
Create, address (**hostid, port=x,** send datagram request using clientSocket
read reply from clientSocket
close clientSocket

Sample UDP code

UDPEchoServer.java

Simple UDP Echo server.

Test using nc as the client (netcat):

```
> nc –u hostname port
```

UDPEchoServer.java

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

class UDPEchoServer {
      public static void main(String args[]) throws Exception {
            int port = 9876;
            DatagramSocket serverSocket = new
DatagramSocket(port);

            byte[] receiveData = new byte[1024];
            byte[] sendData     = new byte[1024];
            while(true) {
                  DatagramPacket receivePacket = new
DatagramPacket (receiveData, receiveData.length);
                  serverSocket.receive(receivePacket);
                  String sentence = new
String(receivePacket.getData());
                  InetAddress IPAddress =
receivePacket.getAddress();
                  int clientPort = receivePacket.getPort();
                  String capitalizedSentence =
sentence.toUpperCase();
                  sendData = capitalizedSentence.getBytes();
                  DatagramPacket sendPacket = new
DatagramPacket  (sendData, sendData.length, IPAddress, clientPort);
                  serverSocket.send(sendPacket);

            }
      }
}
```

UDPClient.java

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

public class UDPClient {
	public static void main(String args[]) throws Exception {
		BufferedReader inFromUser = new BufferedReader(new 
InputStreamReader (System.in));
		int port = 9876;
		DatagramSocket clientSocket = new 
DatagramSocket();
		InetAddress IPAddress = 
InetAddress.getByName("localhost");

		byte[] sendData = new byte[1024];
		byte[] receiveData = new byte[1024];
while(true) {
                        System.out.println("Please enter your message");
                        String sentence = inFromUser.readLine();
                        sendData = sentence.getBytes();
                        DatagramPacket sendPacket = new DatagramPacket
(sendData,  sendData.length, IPAddress, port);
                        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();
                }

        }
}
```

Socket functional calls

- socket (): Create a socket
- bind(): bind a socket to a local IP address and port #

- listen(): passively waiting for connections
- connect(): initiating connection to another socket
- accept(): accept a new connection

- Write(): write data to a socket
- Read(): read data from a socket
- sendto(): send a datagram to another UDP socket
- recvfrom(): read a datagram from a UDP socket

- close(): close a socket (tear down the connection)

Java URL Class

- Represents a Uniform Resource Locator
  - scheme (protocol)
  - hostname
  - port
  - path
  - query string

Parsing

- You can use a URL object as a *parser*:

```
URL u = new URL("http://www.cs.unr.edu/");

System.out.println("Proto:" + u.getProtocol());

System.out.println("File:" + u.getFile());
```

URL construction

- You can also build a URL by setting each part individually:

```
URL u = new URL(“http”,
                www.cs.unr.edu,80,”/~mgunes/”);

System.out.println(“URL:” + u.toExternalForm());

System.out.println(“URL: “ + u);
```

Retrieving URL contents

- URL objects can retrieve the documents they refer to!
  - actually this depends on the protocol part of the URL.
  - HTTP is supported
  - File is supported ("file://c:\foo.html")
  - You can get "Protocol Handlers" for other protocols.

- There are a number of ways to do this:

```
Object getContent();

InputStream openStream();

URLConnection openConnection();
```

Getting Header Information

- There are methods that return information extracted from response headers:

```
String getContentType();

String getContentLength();

long getLastModified();
```

URLConnection

- Represents the connection (not the URL itself).
- More control than URL
  - can write to the connection (send POST data).
  - can set request headers.
- Closely tied to HTTP