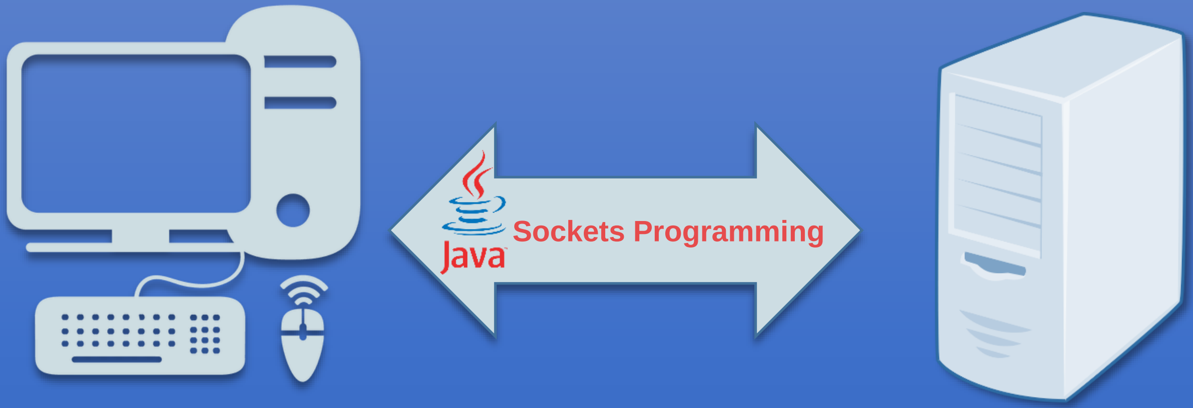


Sockets Programming

Lecture 14

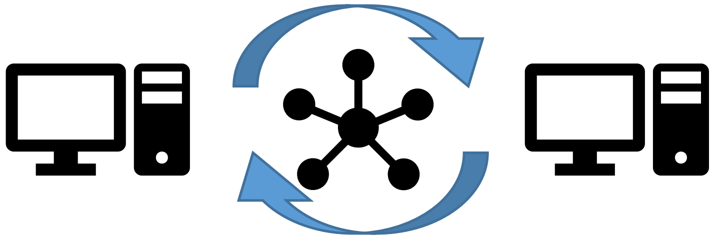
Dr. Tamer ABUHMED
Java Programming Course (SWE2023)
College of Computing
SUNG KYUN KWAN UNIVERSITY

Outline

- Introduction to Java Sockets Programming
- Socket class
- JAVA TCP Sockets
- Socket Constructors & Methods
- Socket I/O
- Server Socket
- Client/server TCP socket interaction (Example)
- Client/server TCP socket interaction (Swing Example)

Java Sockets Programming

- What is Socket? Generally refers to a stream connecting processes running in different address spaces (across a network or on the same machine).

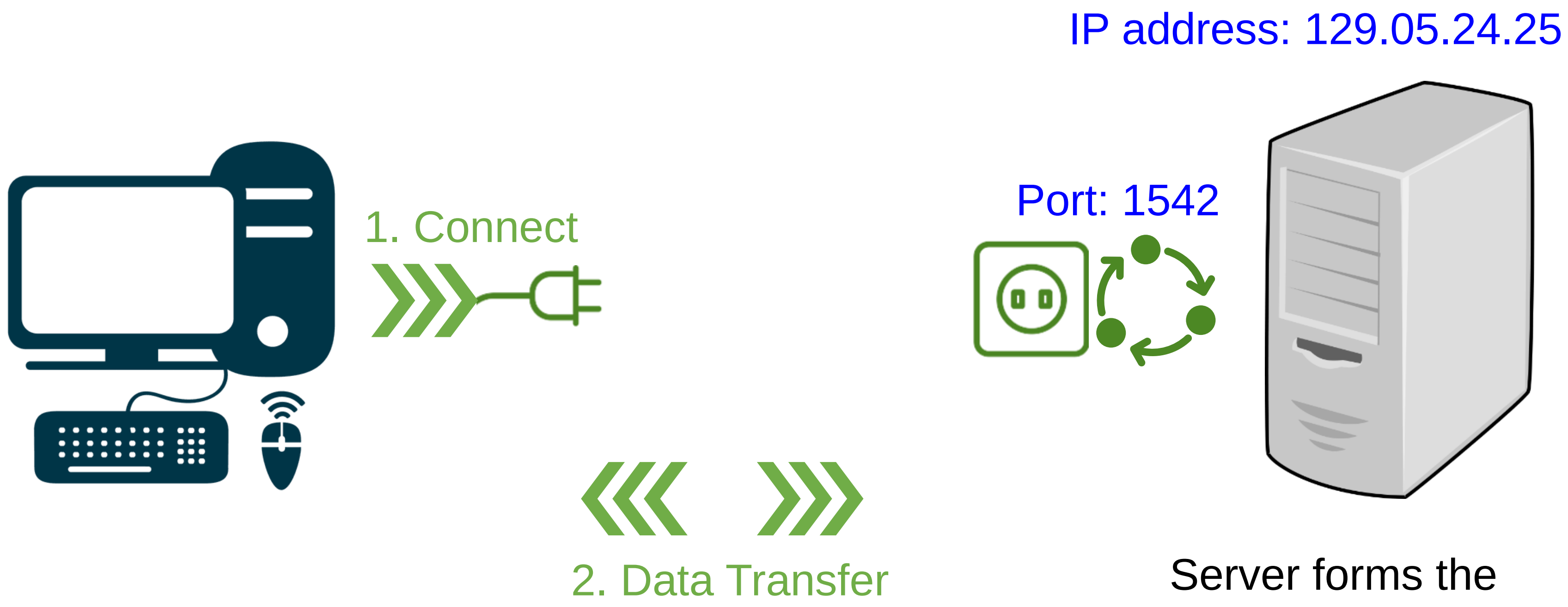


- The package java.net provides support for sockets programming (and more).
- Typically you import everything defined in this package with:

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

Java Sockets Programming

Socket programming is a way of connecting two nodes on a network to communicate with each other.



Server forms the listener socket while client reaches out to the server.

Classes

`InetAddress`

`Socket`

`ServerSocket`

`DatagramSocket`

`DatagramPacket`

InetAddress class

- provides methods to get the IP of any host name
- 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`**

Sample Code: Lookup.java

- Uses InetAddress class to lookup hostnames found on command line.

```
> java Lookup www.yahoo.com www.skku.edu
www.yahoo.com:106.10.250.11
www.skku.edu:115.145.133.39
```

Lookup.java

```
public class Lookup {
 public static void main(String[] args) {

    for (String s : args) {
       String hostname = s;
            try {

          InetAddress a = InetAddress.getByName(hostname);

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

          } catch (UnknownHostException e) {

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

              }
            }
        }
}
```

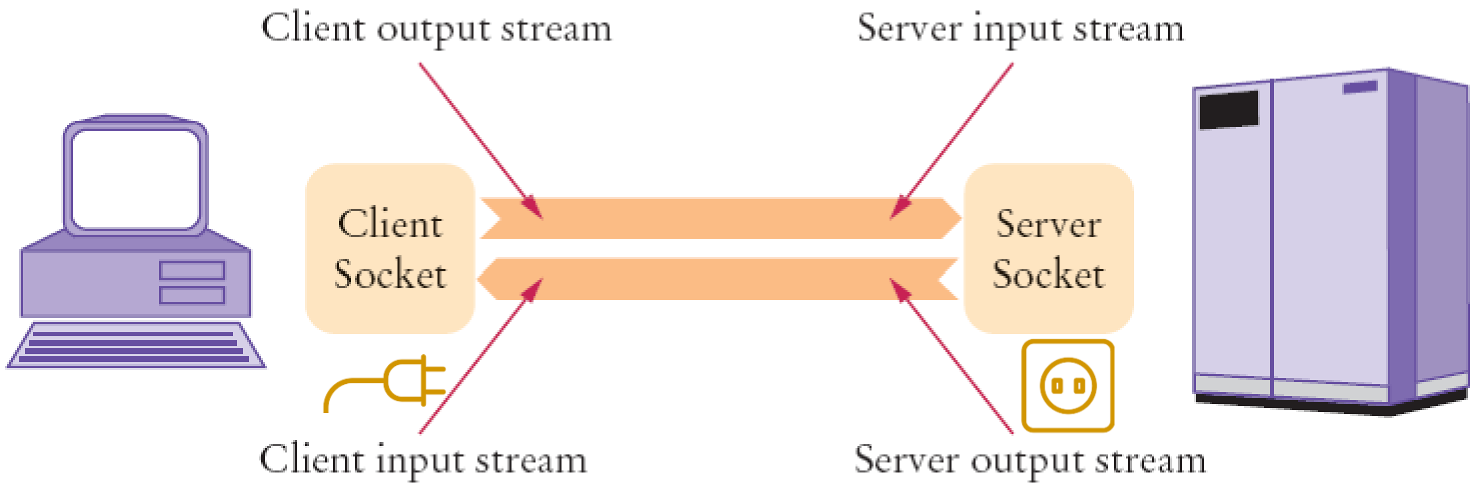
Socket class

- 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

- 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.

Sockets


Client output stream
Server input stream
Client Socket
Server Socket
Client input stream
Server output stream

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

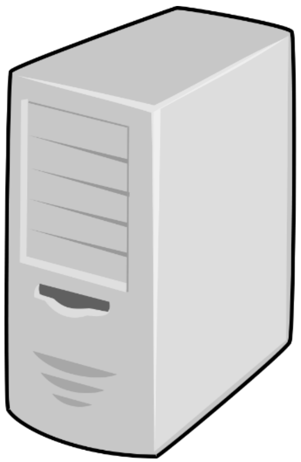
```
// writes b.length bytes
void write(byte[] b);

// 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

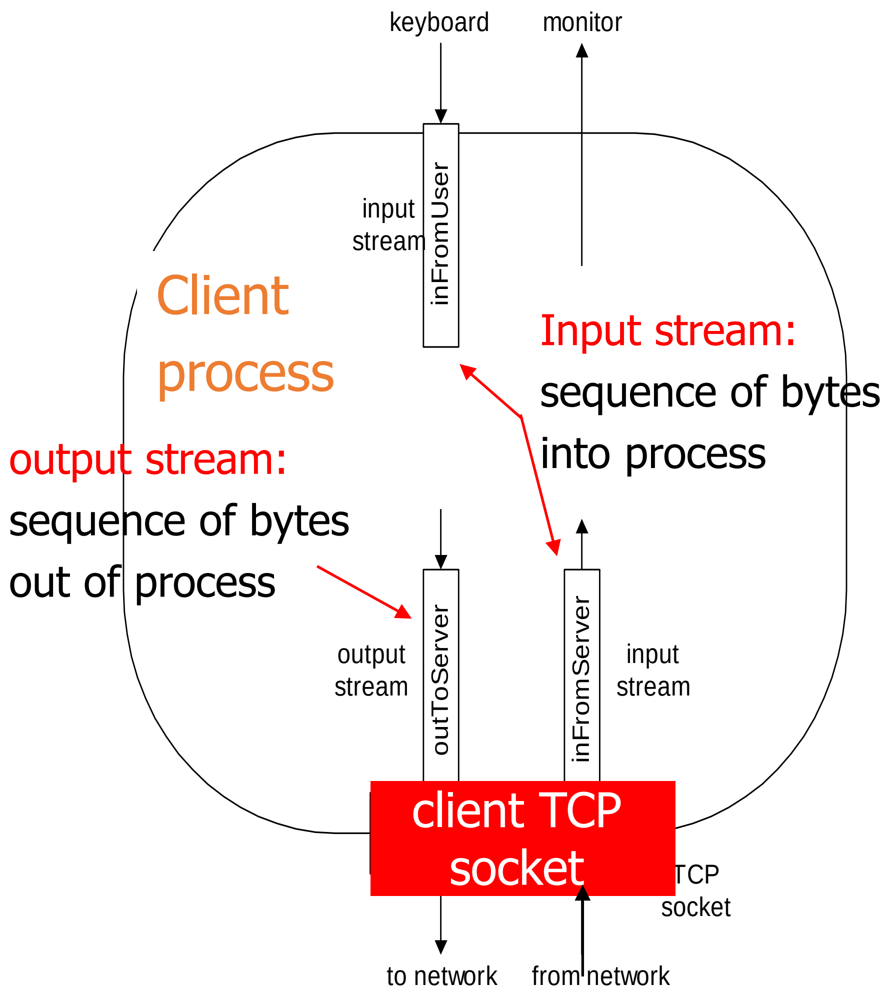
```
Socket accept();

void close();

InetAddress getInetAddress();

int getLocalPort();
```

throw `IOException, SecurityException`

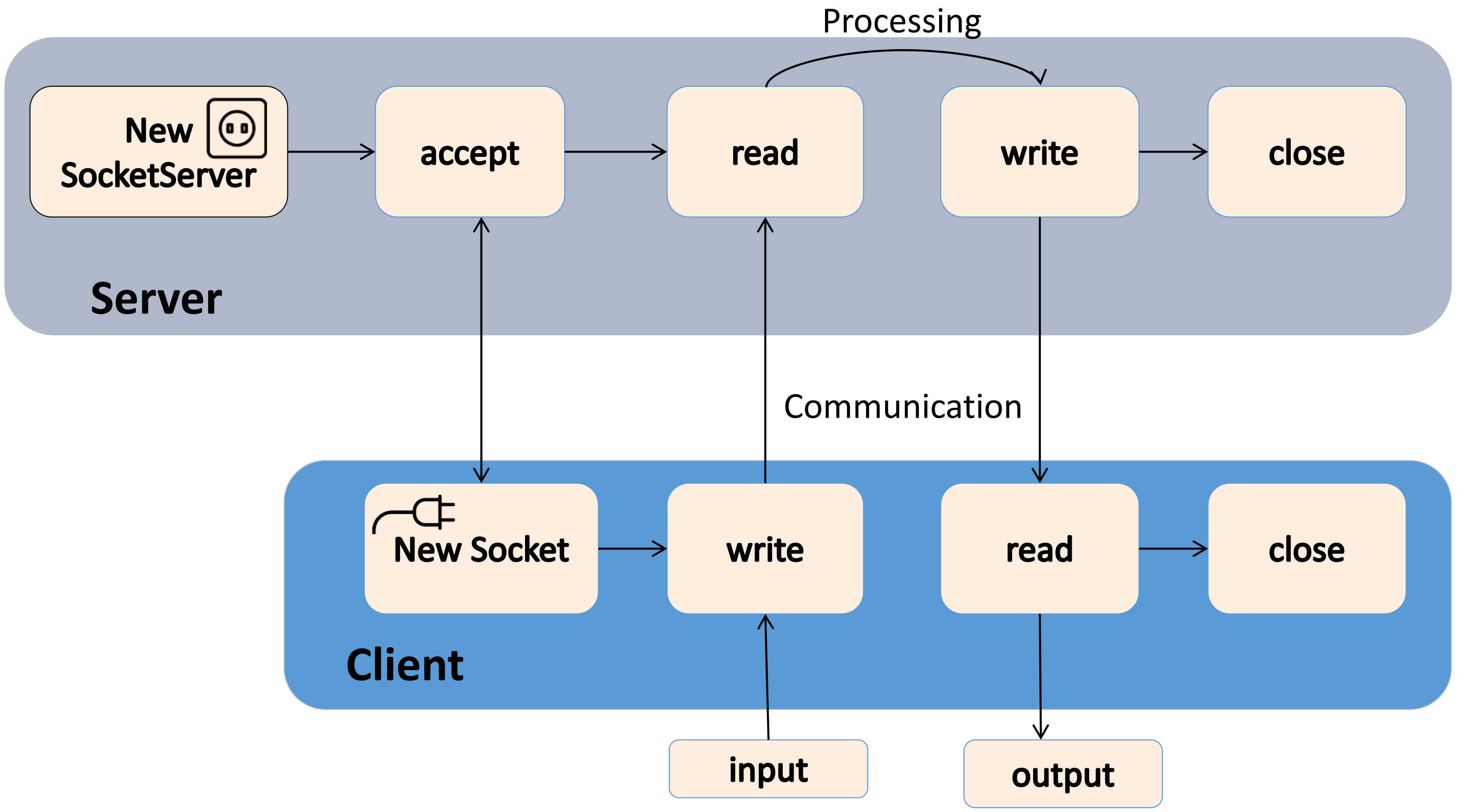
Socket programming with TCP

Example client-server app:

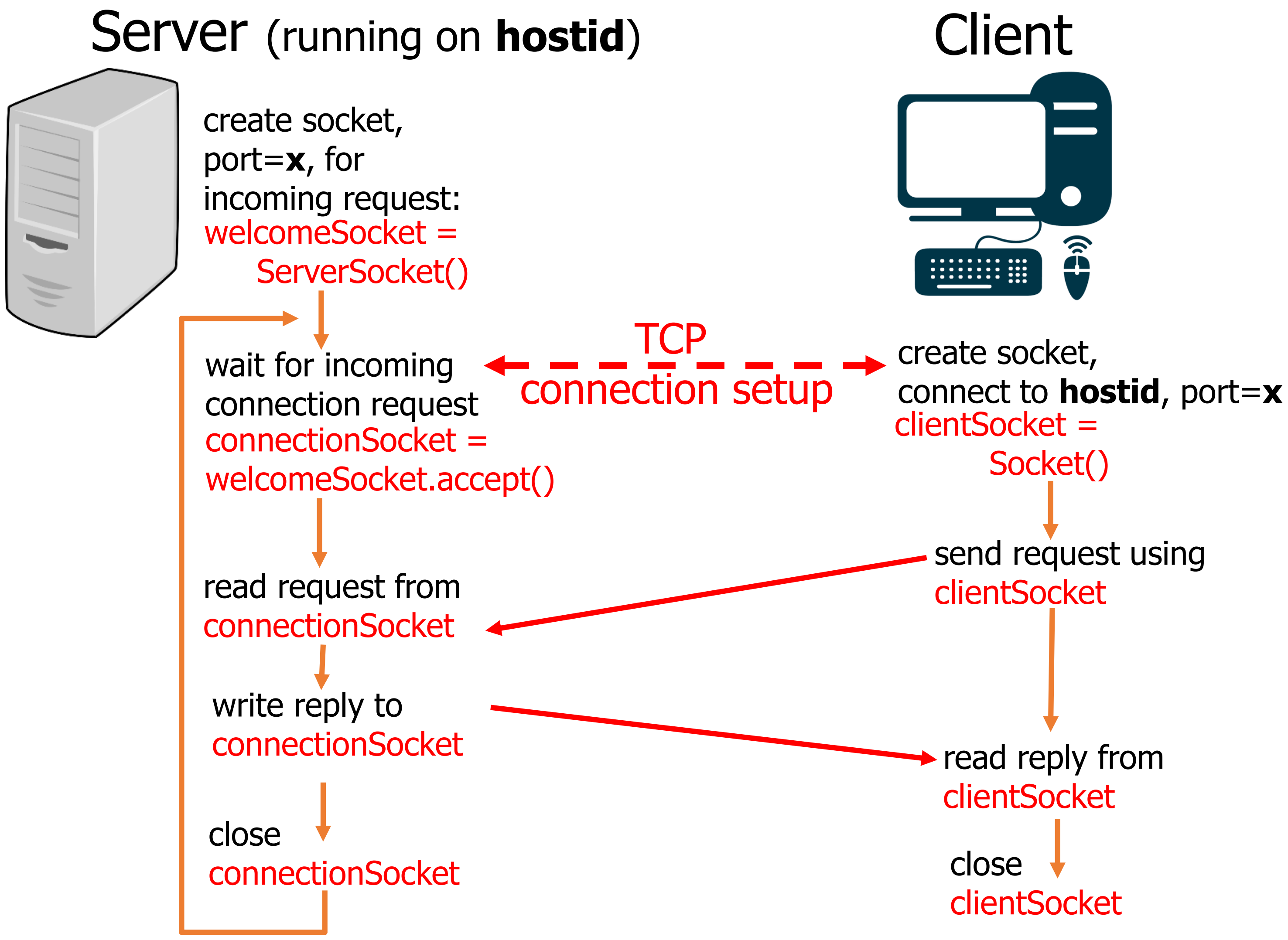
- client reads line from standard input (**inFromUser** stream) , sends to server via socket (**outToServer** stream)
- server reads line from socket
- server converts line to uppercase, sends back to client
- client reads, prints modified line from socket (**inFromServer** stream)



Client/server socket interaction: TCP



Client/server socket interaction: TCP



Server (running on **hostid**)

create socket,
port=**x**, for
incoming request:
welcomeSocket =
ServerSocket()

wait for incoming
connection request
connectionSocket =
welcomeSocket.accept()

read request from
connectionSocket

write reply to
connectionSocket

close
connectionSocket

Client

create socket,
connect to **hostid**, port=**x**
clientSocket =
Socket()

send request using
clientSocket

read reply from
clientSocket

close
clientSocket

TCP Simple Client Example

```
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("hostname", 6789);
DataOutputStream outToServer =  new
DataOutputStream(clientSocket.getOutputStream());
```

TCP Simple Client Example Cont.

```
    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();

    }
}
```

TCP Simple Server Example

```
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()));
```

TCP Simple Server Example Cont.

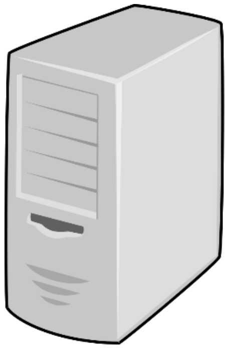
```
            DataOutputStream  outToClient =
                new
   DataOutputStream(connectionSocket.getOutputStream());

            clientSentence = inFromClient.readLine();

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

            outToClient.writeBytes(capitalizedSentence);

      }
   }
}
```

TCP Client/Server

Using Swing GUI

Establishing a Server Using Stream Sockets

- Establishing a server in Java requires five steps.
- *Step 1 is to create a ServerSocket object.*
- `ServerSocket` constructor
    - `ServerSocket server = new ServerSocket( portNumber, queueLength );`
- The constructor establishes the port number where the server waits for connections from clients—a process known as binding the server to the port.
- Each client will ask to connect to the server on this port.



Establishing a Server Using Stream Sockets (cont.)

- Programs manage each client connection with a Socket object.
- In *Step 2*, the server listens indefinitely (or blocks) for an attempt by a client to connect.
- To listen for a client connection, the program calls `ServerSocket` method accept, as in
  - `Socket connection = server.accept();`
  - returns a `Socket` when a connection with a client is established.
- The `Socket` allows the server to interact with the client.

Establishing a Server Using Stream Sockets (cont.)

- *Step 3* is to get the `OutputStream` and `InputStream` objects that enable the server to communicate with the client by sending and receiving bytes.
  - The server invokes method getOutputStream on the `Socket` to get a reference to the `Socket`'s `OutputStream` and invokes method getInputStream on the `Socket` to get a reference to the `Socket`'s `InputStream`.
- *Step 4* is the processing phase, in which the server and the client communicate via the `OutputStream` and `InputStream` objects.
- In *Step 5*, when the transmission is complete, the server closes the connection by invoking the close method on the streams and on the `Socket`.

Establishing a Server Using Stream Sockets

```java
public class Server extends JFrame
{
   private JTextField enterField; // inputs message from user
   private JTextArea displayArea; // display information to user
   private ObjectOutputStream output; // output stream to client
   private ObjectInputStream input; // input stream from client
   private ServerSocket server; // server socket
   private Socket connection; // connection to client
   private int counter = 1; // counter of number of connections

   // set up GUI
   public Server()
   {
      super( "Server" );

      enterField = new JTextField(); // create enterField
      enterField.setEditable( false );
      enterField.addActionListener(
         new ActionListener()
         {
            // send message to client
            public void actionPerformed( ActionEvent event )
            {
               sendData( event.getActionCommand() );
               enterField.setText( "" );
            } // end method actionPerformed
         } // end anonymous inner class
      ); // end call to addActionListener

      add( enterField, BorderLayout.NORTH );

      displayArea = new JTextArea(); // create displayArea
      add( new JScrollPane( displayArea ), BorderLayout.CENTER );

      setSize( 300, 150 ); // set size of window
      setVisible( true ); // show window
   } // end Server constructor
```

```java
 // set up and run server
   public void runServer()
   {
      try // set up server to receive connections; process
connections
      {
         server = new ServerSocket( 12345, 100 ); // create
ServerSocket

         while ( true )
         {
            try
            {
               waitForConnection(); // wait for a connection
               getStreams(); // get input & output streams
               processConnection(); // process connection
            } // end try
            catch ( EOFException eofException )
            {
               displayMessage( "\nServer terminated connection" );
            } // end catch
            finally
            {
               closeConnection(); //  close connection
               ++counter;
            } // end finally
         } // end while
      } // end try
      catch ( IOException ioException )
      {
         ioException.printStackTrace();
      } // end catch
   } // end method runServer
// wait for connection to arrive, then display connection info
private void waitForConnection() throws IOException
   {
displayMessage( "Waiting for connection\n" );
connection = server.accept(); // allow server to accept connection
displayMessage( "Connection " + counter + " received from: " +
         connection.getInetAddress().getHostName() );
   } // end method waitForConnection
   // get streams to send and receive data
```

Establishing a Server Using Stream Sockets

```java
private void getStreams() throws IOException
 {   // set up output stream for objects
output = new ObjectOutputStream( connection.getOutputStream() );
output.flush(); // flush output buffer to send header information

// set up input stream for objects
input = new ObjectInputStream( connection.getInputStream() );
displayMessage( "\nGot I/O streams\n" );
   } // end method getStreams
   // process connection with client
private void processConnection() throws IOException
   {
      String message = "Connection successful";
      sendData( message ); // send connection successful message
      // enable enterField so server user can send messages
      setTextFieldEditable( true );
do // process messages sent from client
  {
    try // read message and display it
      {  message = ( String ) input.readObject(); // read new message
         displayMessage( "\n" + message ); // display message
         } // end try
      catch ( ClassNotFoundException classNotFoundException )
         {displayMessage( "\nUnknown object type received" );
         } // end catch
      } while ( !message.equals( "CLIENT>>> TERMINATE" ) );
   } // end method processConnection
   // close streams and socket
private void closeConnection()
   {
      displayMessage( "\nTerminating connection\n" );
      setTextFieldEditable( false ); // disable enterField
      try
      {  output.close(); // close output stream
         input.close(); // close input stream
         connection.close(); // close socket
      } // end try
```

```java
catch ( IOException ioException )
     {
        ioException.printStackTrace();
     } // end catch
   } // end method closeConnection

private void sendData( String message )
   {// send message to client
     try // send object to client
     {  output.writeObject( "SERVER>>> " + message );
        output.flush(); // flush output to client
        displayMessage( "\nSERVER>>> " + message );
     } // end try
     catch ( IOException ioException )
     {
        displayArea.append( "\nError writing object" );
     } // end catch
   } // end method sendData

   // manipulates displayArea in the event-dispatch thread
   private void displayMessage( final String messageToDisplay )
   {
     SwingUtilities.invokeLater(
        new Runnable()
        {
           public void run() // updates displayArea
           {
             displayArea.append( messageToDisplay ); // append message
           } // end method run
        } // end anonymous inner class
     ); // end call to SwingUtilities.invokeLater
   } // end method displayMessage

   // manipulates enterField in the event-dispatch thread
   private void setTextFieldEditable( final boolean editable )
   { SwingUtilities.invokeLater(
        new Runnable()
        {
           public void run() // sets enterField's editability
           {
             enterField.setEditable( editable );
           }} // end method run  then // end inner class
); } // end call to SwingUtilities.invokeLater then // end method
} // end class Server
```

Establishing a Client Using Stream Sockets (1/3)

- Establishing a simple client in Java requires four steps.
- In *Step 1*, the `Socket` constructor establishes a connection to the server.
  - `Socket connection = new Socket(serverAddress, port);`
  - If the connection attempt is successful, this statement returns a `Socket`.
  - A connection attempt that fails throws an instance of a subclass of `IOException`.
  - An `UnknownHostException` occurs when the system is unable to resolve the server name.

Establishing a Client Using Stream Sockets (2/2)

- In *Step 2*, the client uses `Socket` methods `getInputStream` and `getOutputStream` to obtain references to the `Socket`'s `InputStream` and `OutputStream`.
- *Step 3* is the processing phase in which the client and the server communicate via the `InputStream` and `OutputStream` objects.
- In *Step 4*, the client closes the connection when the transmission is complete by invoking the `close` method on the streams and on the `Socket`.

Establishing a Client Using Stream Sockets

```
public class Client extends JFrame
{
   private JTextField enterField; // enters information from user
   private JTextArea displayArea; // display information to user
   private ObjectOutputStream output; // output stream to server
   private ObjectInputStream input; // input stream from server
   private String message = ""; // message from server
   private String chatServer; // host server for this application
   private Socket client; // socket to communicate with server

   // initialize chatServer and set up GUI
   public Client( String host )
   {
      super( "Client" );

      chatServer = host; // set server to which this client connects
      enterField = new JTextField(); // create enterField
      enterField.setEditable( false );
      enterField.addActionListener(
         new ActionListener()
         {
            // send message to server
            public void actionPerformed( ActionEvent event )
            {
               sendData( event.getActionCommand() );
               enterField.setText( "" );
            } // end method actionPerformed
         } // end anonymous inner class
      ); // end call to addActionListener

      add( enterField, BorderLayout.NORTH );

      displayArea = new JTextArea(); // create displayArea
      add( new JScrollPane( displayArea ), BorderLayout.CENTER );

      setSize( 300, 150 ); // set size of window
      setVisible( true ); // show window
   } // end Client constructor
```

```
 // connect to server and process messages from server
   public void runClient()
   {
      try // connect to server, get streams, process connection
      {
         connectToServer(); // create a Socket to make connection
         getStreams(); // get the input and output streams
         processConnection(); // process connection
      } // end try
      catch ( EOFException eofException )
      {
         displayMessage( "\nClient terminated connection" );
      } // end catch
      catch ( IOException ioException )
      {
         ioException.printStackTrace();
      } // end catch
      finally
      {
         closeConnection(); // close connection
      } // end finally
   } // end method runClient

// connect to server
private void connectToServer() throws IOException
   {
     displayMessage( "Attempting connection\n" );

      // create Socket to make connection to server
   client = new Socket( InetAddress.getByName( chatServer ), 12345 );
   // display connection information
      displayMessage( "Connected to: " +
         client.getInetAddress().getHostName() );
   } // end method connectToServer
```

Establishing a Client Using Stream Sockets

```java
// get streams to send and receive data
private void getStreams() throws IOException
   {
      // set up output stream for objects
   output = new ObjectOutputStream( client.getOutputStream() );
   output.flush(); // flush output buffer to send header information

      // set up input stream for objects
      input = new ObjectInputStream( client.getInputStream() );

      displayMessage( "\nGot I/O streams\n" );
   } // end method getStreams


// process connection with server
private void processConnection() throws IOException
   {
      // enable enterField so client user can send messages
      setTextFieldEditable( true );

  do // process messages sent from serve
   {
     try // read message and display it
      {
        message = ( String ) input.readObject(); // read new message
            displayMessage( "\n" + message ); // display message
        } // end try
        catch ( ClassNotFoundException classNotFoundException )
        {
            displayMessage( "\nUnknown object type received" );
        } // end catch

      } while ( !message.equals( "SERVER>>> TERMINATE" ) );
   } // end method processConnection
```

```java
// close streams and socket
   private void closeConnection()
   {  displayMessage( "\nClosing connection" );
      setTextFieldEditable( false ); // disable enterField

      try
      {  output.close(); // close output stream
         input.close(); // close input stream
         client.close(); // close socket
      } // end try
      catch ( IOException ioException )
      { ioException.printStackTrace();
      } // end catch
   } // end method closeConnection

// send message to server
   private void sendData( String message )
   {
      try // send object to server
      {  output.writeObject( "CLIENT>>> " + message );
         output.flush(); // flush data to output
         displayMessage( "\nCLIENT>>> " + message );
      } // end try
      catch ( IOException ioException )
      {
         displayArea.append( "\nError writing object" );
      } } // end catch then // end method sendData

   // manipulates displayArea in the event-dispatch thread
   private void displayMessage( final String messageToDisplay )
   {
      SwingUtilities.invokeLater(
         new Runnable()
         {
            public void run() // updates displayArea
            {
               displayArea.append( messageToDisplay );
            }} // end method run then // end anonymous inner class
      ); // end call to SwingUtilities.invokeLater
   } // end method displayMessage

   // manipulates enterField in the event-dispatch thread
   private void setTextFieldEditable( final boolean editable )
   {
      SwingUtilities.invokeLater(
         new Runnable()
         {
            public void run() // sets enterField's editability
            {
               enterField.setEditable( editable );
            } }  // end method run then // end anonymous inner class
      ); // end call to SwingUtilities.invokeLater
   } } // end method setTextFieldEditable  then // end class Client
```

Summary

- Introduction to Java Sockets Programming
- Socket class
- JAVA TCP Sockets
- Socket Constructors & Methods
- Socket I/O
- Server Socket
- Client/server TCP socket interaction (Example)
- Client/server TCP socket interaction (Swing Example)