Advanced Programming

Computer Science Program

Chapter 6

Networking in Java

Introduction

- A network is a collection of computers and other devices that can send data to and receive data from each other.
- What a network program do?
  - Retrieve data
  - Send data – Once a connection between two machines is established, Java programs can send data across the connection just as easily as they can receive from it.
  - Peer-to-peer interaction
    - Games
    - Chat
    - File sharing
  - Servers
  - Searching the web
  - E-commerce

Introduction (cont'd)

- When a computer needs to communicate with another computer, it needs to know the other computer's address.
- An Internet Protocol (IP) address uniquely identifies the computer on the Internet.
- TCP and UDP
  - TCP enables two hosts to **establish a connection** and exchange streams of data.
  - TCP **guarantees delivery** of data and also guarantees that packets will be **delivered in the same order** in which they were sent.
  - UDP is a connectionless protocol.

Client/Server Computing

- Networking is tightly integrated in Java.
- Java API provides the classes for creating sockets to facilitate program communications over the Internet.
- **Sockets** are the endpoints of logical connections between two hosts and can be used to send and receive data.
- Java treats socket communications much as it treats I/O operations; thus programs can read from or write to sockets as easily as they can read from or write to files.
- Network programming usually involves a server and one or more clients.

Client/Server Computing (Cont'd)

- The client sends requests to the server, and the server responds.
- The client begins by attempting to establish a connection to the server.
- The server can accept or deny the connection.
- Once a connection is established, the client and the server communicate through sockets.
- The server must be running when a client attempts to connect to the server.
- The server waits for a connection request from a client.

The Server Socket

- To establish a server, you need to create a server socket and attach it to a port, which is where the server listens for connections.
- Port is a software address of a computer on the network.
- The port identifies the TCP service on the socket.
- A socket is a communication path to a port.
- To communicate program over the network, give a way of addressing the port. How? Create a socket and attach it to the port.
- Port numbers range from 0 to 65536, but port numbers 0 to 1024 are reserved for privileged services.
- For instance, the email server runs on port 25, and the Web server usually runs on port 80.

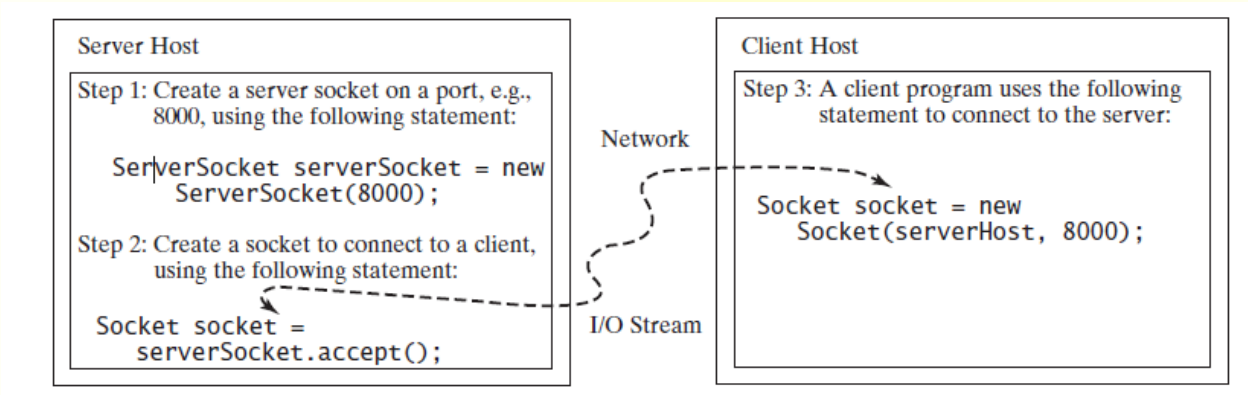
The Server Socket (cont'd)

- You can choose any port number that is not currently used by any other process.
- The following statement creates a server socket **serverSocket:**
  - ServerSocket serverSocket = **new ServerSocket(port);**
- After a server socket is created, the server can use the following statement to listen for connections:
  - Socket socket = serverSocket.accept();

The Client Socket

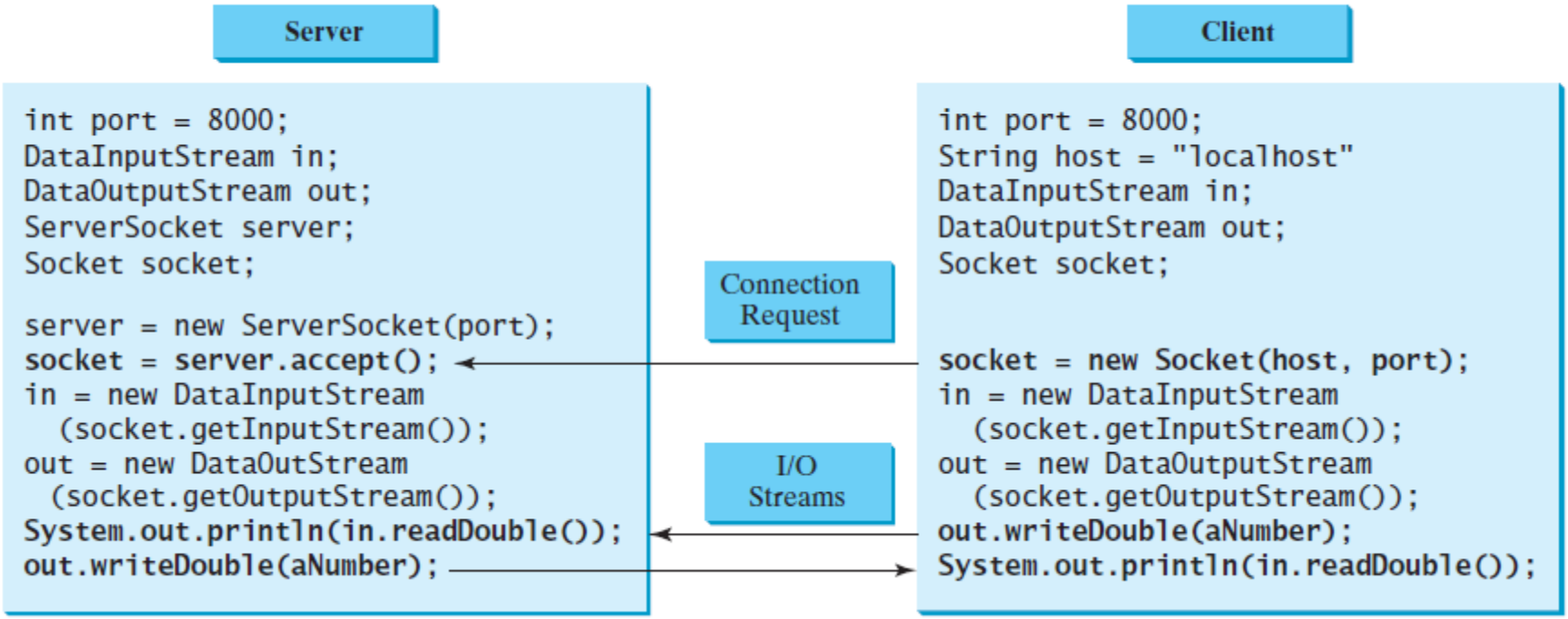
- The client issues the following statement to request a connection to a server:
  - Socket socket = new Socket(serverName, port);
    - ServerName is the server's Internet host name or IP address.
- The following statement creates a socket at port 8000 on the client machine to connect to the host 130.254.204.36:
  - Socket socket = new Socket("130.254.204.36", 8000)

The Client Socket(cont'd)



The server creates a server socket and, once a connection to a client is established, connects to the client with a client socket.

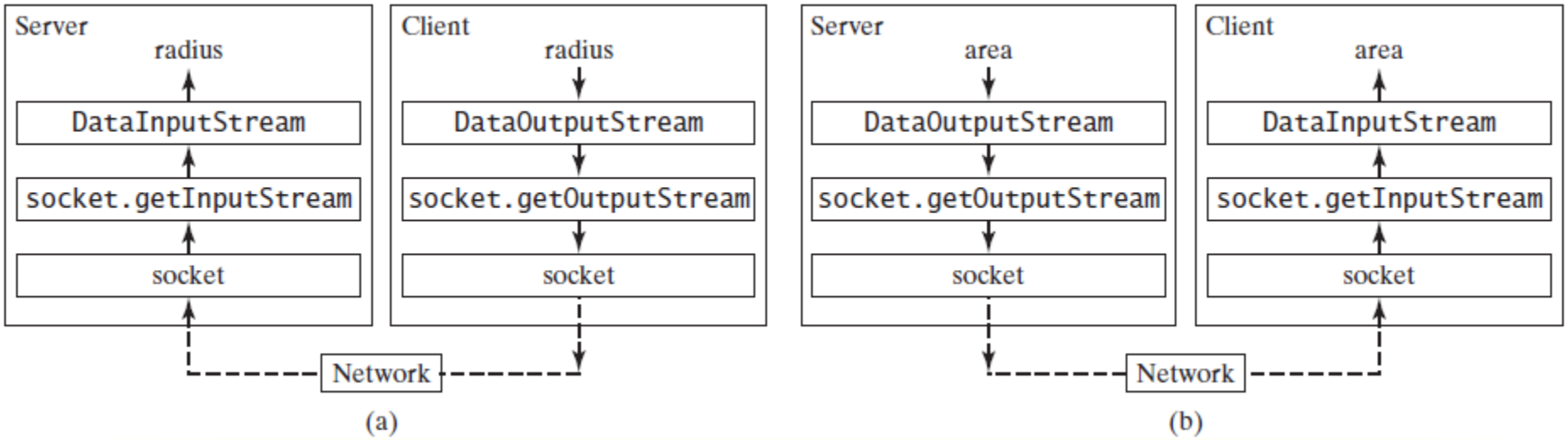
Data Transmission Through Sockets



The server and client exchange data through I/O streams on top of the socket.

Data Transmission…

- Example: The client sends the radius to the server; the server computes the area and sends it to the client.



(a) The client sends the radius to the server.

(b) The server sends the area to the client.

Example: Sever

```
//Server.java
import java.io.*;
import java.net.*;
import java.util.*;
import java.awt.*;
import javax.swing.*;
public class Server extends JFrame {
    // Text area for displaying contents
     private JTextArea jta = new JTextArea();
     public static void main(String[] args) {
        new Server();
     }
     public Server() {
         // Place text area on the frame
          setLayout(new BorderLayout());
          add(new JScrollPane(jta), BorderLayout.CENTER);
          setTitle("Server");
```

Example: Sever(cont'd)

```
        setSize(500, 300);
         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         setVisible(true); // It is necessary to show the frame here!
try {
            // Create a server socket
         ServerSocket serverSocket = new ServerSocket(8000);
         jta.append("Server started at " + new Date() + '\n');
         // Listen for a connection request
         Socket socket = serverSocket.accept();
         // Create data input and output streams
         DataInputStream inputFromClient = new DataInputStream(
          socket.getInputStream());
         DataOutputStream outputToClient = new DataOutputStream(
          socket.getOutputStream());
         while (true) {
         // Receive radius from the client
          double radius = inputFromClient.readDouble();
         // Compute area
         double area = radius * radius * Math.PI;
         // Send area back to the client
```

Example: Sever(cont'd)

```
            outputToClient.writeDouble(area);
             jta.append("Radius received from client: " + radius + '\n');
             jta.append("Area found: " + area + '\n');
        }
    }
     catch(IOException ex) {
              System.err.println(ex);
    }
    }
}
```

Example: Client

```
import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Client extends JFrame {
 // Text field for receiving radius
 private JTextField jtf = new JTextField();
 // Text area to display contents
 private JTextArea jta = new JTextArea(); // IO streams
 private DataOutputStream toServer;
 private DataInputStream fromServer;
public static void main(String[] args) {
 new Client();
 }
```

Example: Client(cont'd)

```
public Client() {
// Panel p to hold the label and text field
 JPanel p = new JPanel();
 p.setLayout(new BorderLayout());
 p.add(new JLabel("Enter radius"), BorderLayout.WEST);
 p.add(jtf, BorderLayout.CENTER);
 jtf.setHorizontalAlignment(JTextField.LEFT);
 setLayout(new BorderLayout());
 add(p, BorderLayout.NORTH);
 add(new JScrollPane(jta), BorderLayout.CENTER);
 jtf.addActionListener(new TextFieldListener());
setTitle("Client");
 setSize(500, 300);
 setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 setVisible(true); // It is necessary to show the frame here!
```

Example: Client(cont'd)

```
try {
// Create a socket to connect to the server
Socket socket = new Socket("localhost",8000);
// Socket socket = new Socket("130.254.204.36", 8000);
// Socket socket = new Socket("drake.Armstrong.edu", 8000);
// Create an input stream to receive data from the server
fromServer = new DataInputStream(
  socket.getInputStream());
// Create an output stream to send data to the server
toServer =
 new DataOutputStream(socket.getOutputStream());
}
catch (IOException ex) {
jta.append(ex.toString() + '\n');
}
}
```

Example: Client(cont'd)

```
private class TextFieldListener implements ActionListener {
 public void actionPerformed(ActionEvent e) {
 try {
 // Get the radius from the text field
 double radius = Double.parseDouble(jtf.getText().trim());
 // Send the radius to the server
 toServer.writeDouble(radius);
 toServer.flush();
 // Get area from the server
 double area = fromServer.readDouble() ;
 // Display to the text area
 jta.append("Radius is " + radius + "\n");
 jta.append("Area received from the server is "
 + area + '\n');
 }
 catch (IOException ex) {
 System.err.println(ex);
 }
 }
 }
```

The InetAddress Class

- To know who is connecting to sever, You can use the **InetAddress class.**
- InetAddress has three static methods
  - public static InetAddress getByName(String hostName)
  - public static InetAddress[] getAllByName(String hostName)
  - public static InetAddress getLocalHost( )

InetAddress (cont'd)

- Example - A program that prints the address of host

```
import java.net.*;
public class HostName {
  public static void main (String[] args) {
    try {
      InetAddress address = InetAddress.getByName("www.google.com");
      System.out.println(address);
    }
    catch (UnknownHostException ex) {
      System.out.println("Could not find www.google.com");

    }
 }
}
```

InetAddress (cont'd)

- Some computers have more than one Internet address.
- Given a hostname, InetAddress.getAllByName() returns an array that contains all the addresses corresponding to that name.
- Example

```
import java.net.*;
public class AllAddressOfGoogle {
    public static void main (String[] args) {
      try {
            InetAddress[] addresses =
            netAddress.getAllByName("www.google.com");
            for (int i = 0; i < addresses.length; i++) {
                System.out.println(addresses[i]);
             }
       }
       catch (UnknownHostException ex) {
            System.out.println("Could not find www.google.com");
```

InetAddress (cont'd)

- Example

```
import java.net.*;
public class MyAddress {
 public static void main (String[] args) {
    try {
      InetAddress address = InetAddress.getLocalHost( );
      System.out.println(address);
    }
    catch (UnknownHostException ex) {
      System.out.println("Could not find this computer's address.");

    }
  }

}
```

Serving Multiple Clients

- Multiple clients are quite often connected to a single server at the same time.
- Typically, a server runs continuously on a server computer, and clients from all over the Internet can connect to it.
- You can use threads to handle the server's multiple clients simultaneously.
- Simply create a thread for each connection.
- Here is how the server handles the establishment of a connection:

```
while (true) {
    Socket socket = serverSocket.accept(); // Connect to a client
    Thread thread = new ThreadClass(socket);
    thread.start();
}
```

- The server socket can have many connections.
- Each iteration of the **while loop creates a new** connection.
- Whenever a connection is established, a new thread is created to handle communication between the server and the new client; and this allows multiple connections to run at the same time.