Chapter 30

Networking and Sockets

Introduction

- The Internet is a huge part of computing today. Without an Internet connection most of us are lost.
- E-mail, Web browsers, and IM clients are all useless without the ability to open a connection to a another computer.
- To allow a program to open a network connection, we use sockets. Today we will be learning about TCP sockets, and how to use them to allow two programs to communicate over a network.

Introduction

- We will see that opening up a socket connection is very easy
- We will also see that sending data back and forth between two programs is identical to reading and writing from binary files.
- Lets begin by looking at how to open a server socket, and how to connect to it with a client socket.

Server Sockets

- To establish a server, you need to create a server socket and attach it to a port.
- The port identifies the TCP service on the socket.
- Port number range from 0 – 65,536 (unsigned 16 bit numbers)
- Ports 0 – 1024 are reserved for privileged services, so we shouldn't use them unless we are writing our own implementation of a privileged service.

Server Sockets

- A server socket object can bind to any port not already in use by another program.
- It is probably save to use your birthday as a port number for your programs (Eg. 4985 for me)
- You will know right away if the port is in use because it will throw a BindException.
- If this happens, just change the port number and try again.

Server Sockets

```
import java.net.*;
import javax.swing.*;
import java.io.*;
import java.awt.*;

/**
 * The AreaServer class defines a simple server that reads in
 * a radius from a single client, and responds with the correct
 * area.
 *
 */
public class AreaServer extends JFrame{
    private ServerSocket socket;
    private DataInputStream inStream;
    private DataOutputStream outStream;
    private Thread serverThread;

    private JTextArea jtaOutput = new JTextArea();

    public AreaServer(){
        setLayout(new BorderLayout());
        add(new JScrollPane(jtaOutput), BorderLayout.CENTER);

        try{
                socket = new ServerSocket(4985);
```

Server Sockets

```
serverThread = new Thread(new Runnable(){
    public void run() {
        double radiusFromClient = 0.0;
        try{
            jtaOutput.append("Waiting for client to connect\n");
            Socket client = socket.accept();

            jtaOutput.append("\tclient connected!");

            inStream = new DataInputStream(client.getInputStream());
            outStream = new DataOutputStream(client.getOutputStream());

            while(true){
                radiusFromClient = inStream.readDouble();
                jtaOutput.append("\nRadius recieved from client: " +
                    radiusFromClient);

                double area = radiusFromClient * radiusFromClient *
                    Math.PI;
                outStream.writeDouble(area);
                jtaOutput.append("\n\tArea returned: " + area);
            }

        }catch(IOException ioe){
            ioe.printStackTrace();
        }
    }
});
```

Server Sockets

```
            serverThread.start();

        }catch (BindException be){
            be.printStackTrace();
        }
        catch (IOException ioe){
            ioe.printStackTrace();
        }
    }

    public static void main(String[] args){
        JFrame frame = new AreaServer();
        frame.setTitle("Area Server");
        frame.setSize(500,500);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Server Sockets

- In this example we are establishing a server with a ServerSocket object.
- We bind the ServerSocket to a specific port number. `socket = new ServerSocket(4985);`
- Next we need to listen or a connection request, and assign that incoming connection a socket. `Socket client = socket.accept();`
- After we have our connection, we need to open input and output streams so we can communicate.

```
inStream = new DataInputStream(client.getInputStream());

outStream = new DataOutputStream(client.getOutputStream());
```

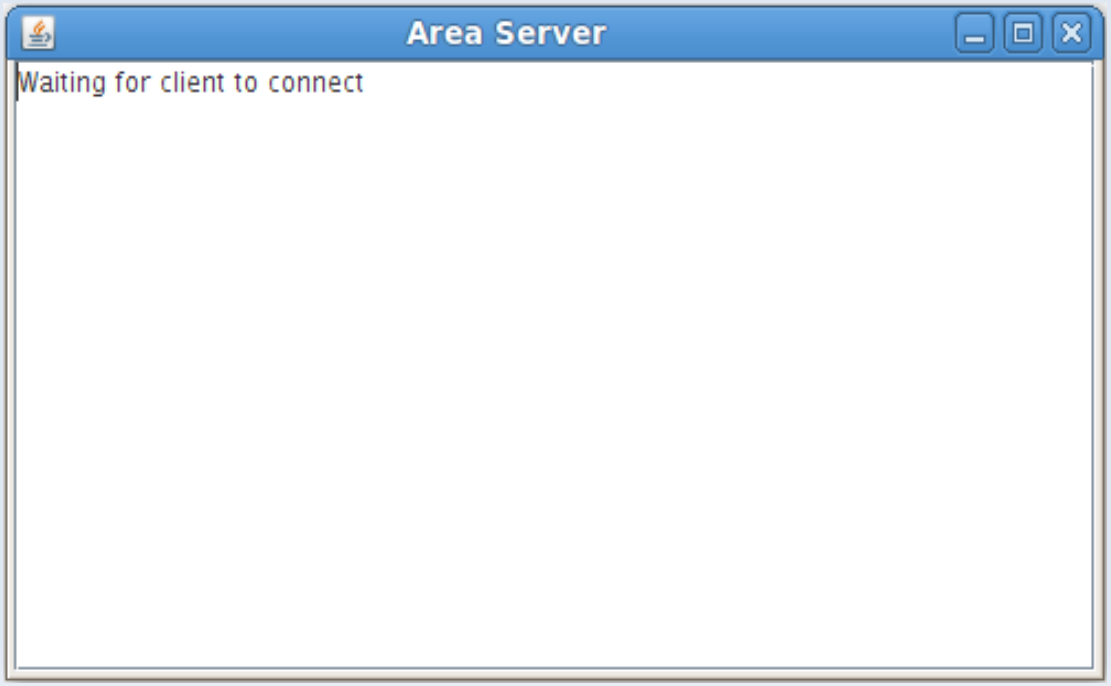
Server Sockets

- Once we have opened communications via our DataStreams, we can now listen for a radius, and respond with an area.

```
radiusFromClient = inStream.readDouble();
double area = radiusFromClient * radiusFromClient * Math.PI;
outStream.writeDouble(area);
```

Server Sockets

- We will continue to do this until the server is shutdown manually.
- Now that we have a server set up, lets write a client that can communicate with, and use the server.



Clients

- Clients are different then servers because they don't need to bind to any specific addresses or ports.
- A client simply creates a Socket object that contains the address and port number that it wants to connect to.
- The address is a string that can be a hostname, or an IP Address.
- Once the connection is established, you simply open up the streams for communications.

Clients

```
import java.io.*;
import java.util.*;
import java.net.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

/**
 * The area client defines a client that connects to the
 * area server. The client sends radiuses, and gets back
 * areas.
 *
 */
public class AreaClient extends JFrame{
   private JTextField jtfRadius = new JTextField();
   private JTextArea jtaOutput = new JTextArea();

   private DataInputStream inStream;
   private DataOutputStream outStream;
```

Clients

```
public AreaClient(){
    JPanel inputPanel = new JPanel(new GridLayout(1,2,0,0));
    inputPanel.add(new JLabel("Enter a radius"));
    inputPanel.add(new JScrollPane(jtfRadius));

    setLayout(new BorderLayout());
    add(inputPanel, BorderLayout.NORTH);
    add(jtaOutput, BorderLayout.CENTER);

    jtfRadius.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
            Scanner inputScanner = new Scanner(jtfRadius.getText());
            if (inputScanner.hasNextDouble()){
                double radius = inputScanner.nextDouble();
                try{
                    jtaOutput.append("Sending radius: " + radius + "\n");
                    outStream.writeDouble(radius);
                    double area = inStream.readDouble();
                    jtaOutput.append("\tRecieved area: " + area + "\n");

                }catch(IOException ioe){
                    ioe.printStackTrace();
                }
            }
            else{
                jtaOutput.append("Invalid input\n");
                return;
            }
        }
    });
```

Clients

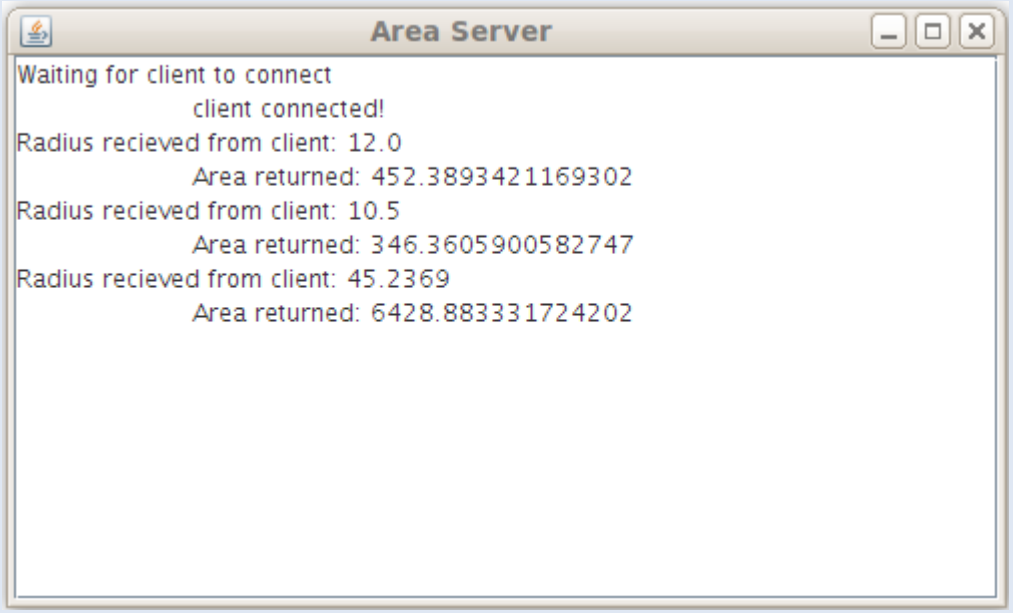
```
        try{
            Socket socket = new Socket("localhost", 4985);
            inStream = new DataInputStream(socket.getInputStream());
            outStream = new DataOutputStream(socket.getOutputStream());
        }catch(IOException ioe){
            ioe.printStackTrace();
        }
    }

    public static void main(String[] args){
        JFrame frame = new AreaClient();
        frame.setTitle("Area Client");
        frame.setSize(500,500);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

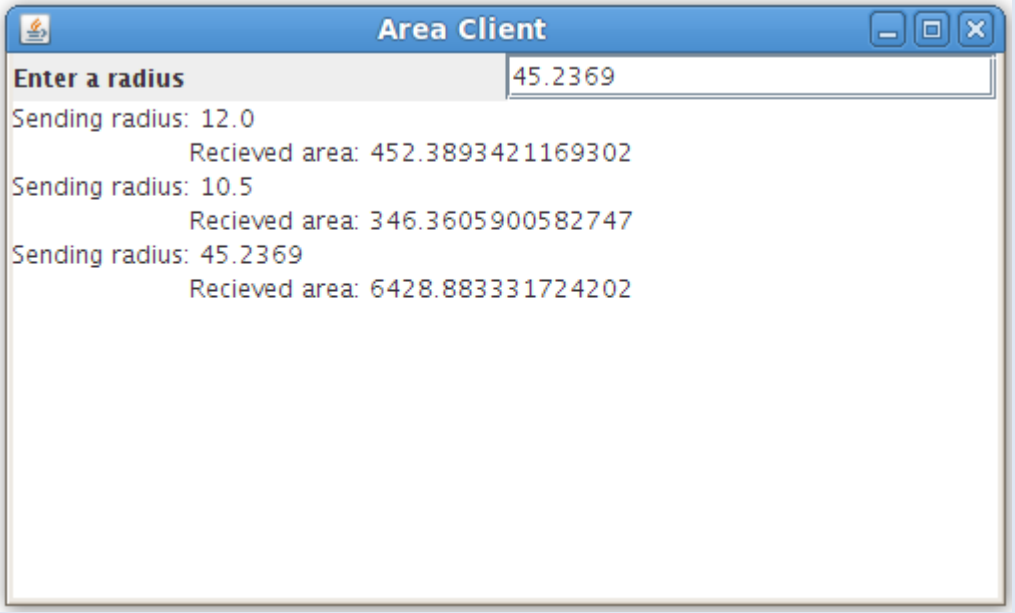
Clients

- In the previous example, The client opens up a socket connection to localhost:4985.
- I used localhost as the hostname because both the server, and client program are running on the same computer.
- If they were running on different computers, The socket would be the host name, or IP address of the remote computer. If you have more then one computer at home I suggest you try this out! Its really cool to see your two computer talking!

Clients

- After the connection has been established, we opened up the DataStreams for communication and began the session.





Client Server Interaction

Server

```
socket = new ServerSocket(4985);
Socket client = socket.accept();
```

Client

```
Socket socket = new Socket("localhost", 4985);
```

```
outStream.writeDouble(radius);
```

```
radiusFromClient = inStream.readDouble();
//process area
outStream.writeDouble(area);
```

```
double area = inStream.readDouble();
```

The InetAddress object

- In the last program, we were only talking to a single client, because that was all we were able to handle.
- In future sever programs it is important to deal with any number of clients that might want to connect and use the services provided.
- Before we can server lots of users, its good to be able to identify them.
- The InetAddress class provides all of the necessary tools to identify a client based on a socket connection.

The InetAddress object

Modification of the AreaServer class to display the connecting clients IP and hostname.

```
serverThread = new Thread(new Runnable(){
    public void run() {
        double radiusFromClient = 0.0;
        try{
            jtaOutput.append("Waiting for client to connect\n");
            Socket client = socket.accept();

            InetAddress clientInfo = client.getInetAddress();
            jtaOutput.append("\tclient connected!\n");
            jtaOutput.append("\tclient hostname: " + clientInfo.getHostName());
            jtaOutput.append("\tclient IP: " + clientInfo.getHostAddress());

            inStream = new DataInputStream(client.getInputStream());
            outStream = new DataOutputStream(client.getOutputStream());
```

The InetAddress object

```
Waiting for client to connect
                    client connected!
                    client hostname: localhost      client IP: 127.0.0.1
Radius recieved from client: 100.0
                    Area returned: 31415.926535897932
Radius recieved from client: 10036.0
                    Area returned: 3.16425283573643E8
Radius recieved from client: 84.0
                    Area returned: 22167.07776372958
Radius recieved from client: 0.236
                    Area returned: 0.1749741444343371
```

Since we are connecting locally, we only get the localhost hostname and loop back address. If you were to connect remotely you would see more interesting information.

Serving Multiple clients

- Now that we can tell our clients apart, Lets modify the AreaServer class to allow as many connections as we want.
- This becomes a much more useful system when it allows multiple clients to interact with it.
- We will use threads to perform this task, as well as locks to share the output display.

Multi threaded server

```
import java.net.*;
import javax.swing.*;
import java.io.*;
import java.awt.*;

/**
 * The AreaServer class defines a simple server that reads in
 * a radius from a single client, and responds with the correct area.
 *
 */
public class AreaServer extends JFrame{
	private ServerSocket socket;
	private DataInputStream inStream;
	private DataOutputStream outStream;
	private Thread serverThread;

	private JTextArea jtaOutput = new JTextArea();
```

Multi threaded server

```
public AreaServer(){
        setLayout(new BorderLayout());
        add(new JScrollPane(jtaOutput), BorderLayout.CENTER);

        try{
            socket = new ServerSocket(4985);
            output("Server ready, waiting for requests");

            serverThread = new Thread(new Runnable(){
                public void run() {
                    try{
                        while(true){
                            Socket client = socket.accept();
                            InetAddress clientInfo = client.getInetAddress();
                            output("client connected: " + clientInfo.getHostAddress());
                            ClientHandler ch = new ClientHandler(client);
                            new Thread(ch).start();
                        }

                    }catch(IOException ioe){
                        ioe.printStackTrace();
                    }
                }
            });

            serverThread.start();

        }catch (BindException be){
            be.printStackTrace();
        }
        catch (IOException ioe){
            ioe.printStackTrace();
        }
    }
```

Multi threaded server

```
private void output(String message){
    synchronized(jtaOutput){
        jtaOutput.append(message + "\n");
    }
}

class ClientHandler implements Runnable{
    private Socket client;
    private InetAddress clientInfo;

    public ClientHandler(Socket client){
        this.client = client;
    }

    public void run(){
      try{

        DataInputStream inStream = new DataInputStream(client.getInputStream());
        DataOutputStream outStream = new DataOutputStream(client.getOutputStream());
        clientInfo = client.getInetAddress();
```

Multi threaded server

```
                while(true){
                    double radiusFromClient = inStream.readDouble();
                    output(String.format("Radius recieved from %s: %f",
                            clientInfo.getHostAddress(), radiusFromClient));

                    double area = radiusFromClient * radiusFromClient * Math.PI;
                    outStream.writeDouble(area);
                    output(String.format("\tArea returned to %s: %f",
                            clientInfo.getHostAddress(), area));
                }
            }catch(EOFException eof){
                output(clientInfo.getHostAddress() + " has disconnected");
            }
            catch(IOException ioe){
                output(clientInfo.getHostAddress() + " has had an IOException");
            }
        }
    }

    public static void main(String[] args){
        JFrame frame = new AreaServer();
        frame.setTitle("Area Server");
        frame.setSize(500,300);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Multi threaded server

- In the previous example, The server has an endless loop that listens for connections (using the ServerSocket.accecpt() method)
- Once the server has a connection, it spawns a handler thread to take care of that single client.
- Once the thread has been started, the server can now wait for the next connections and continue.
- The handler thread now takes care of all of the communications between the user and the server.

Multi threaded server

Area Client

Enter a radius 86

Sending radius: 109.0
Recieved area: 37325.26231730033
Sending radius: 52.0
Recieved area: 8494.8665353068
Sending radius: 86.0
Recieved area: 23235.21926595011

```
Server ready, waiting for requests
client connected: 192.168.1.3
client connected: 127.0.0.1
Radius recieved from 127.0.0.1: 12.000000
        Area returned to 127.0.0.1: 452.389342
Radius recieved from 192.168.1.3: 109.000000
        Area returned to 192.168.1.3: 37325.262317
Radius recieved from 127.0.0.1: 0.235000
        Area returned to 127.0.0.1: 0.173494
Radius recieved from 192.168.1.3: 52.000000
        Area returned to 192.168.1.3: 8494.866535
```

Enter a radius 25

Sending radius: 12.0
Recieved area: 452.3893421169302
Sending radius: 0.235
Recieved area: 0.1734944542944963
Sending radius: 10.0
Recieved area: 314.1592653589793
Sending radius: 25.0
Recieved area: 1963.4954084936207

Multi threaded server

```
                Area returned to 127.0.0.1: 452.389342
Radius recieved from 192.168.1.3: 109.000000
                Area returned to 192.168.1.3: 37325.262317
Radius recieved from 127.0.0.1: 0.235000
                Area returned to 127.0.0.1: 0.173494
Radius recieved from 192.168.1.3: 52.000000
                Area returned to 192.168.1.3: 8494.866535
Radius recieved from 127.0.0.1: 10.000000
                Area returned to 127.0.0.1: 314.159265
Radius recieved from 127.0.0.1: 25.000000
                Area returned to 127.0.0.1: 1963.495408
Radius recieved from 192.168.1.3: 86.000000
                Area returned to 192.168.1.3: 23235.219266
192.168.1.3 has disconnected
127.0.0.1 has disconnected
```

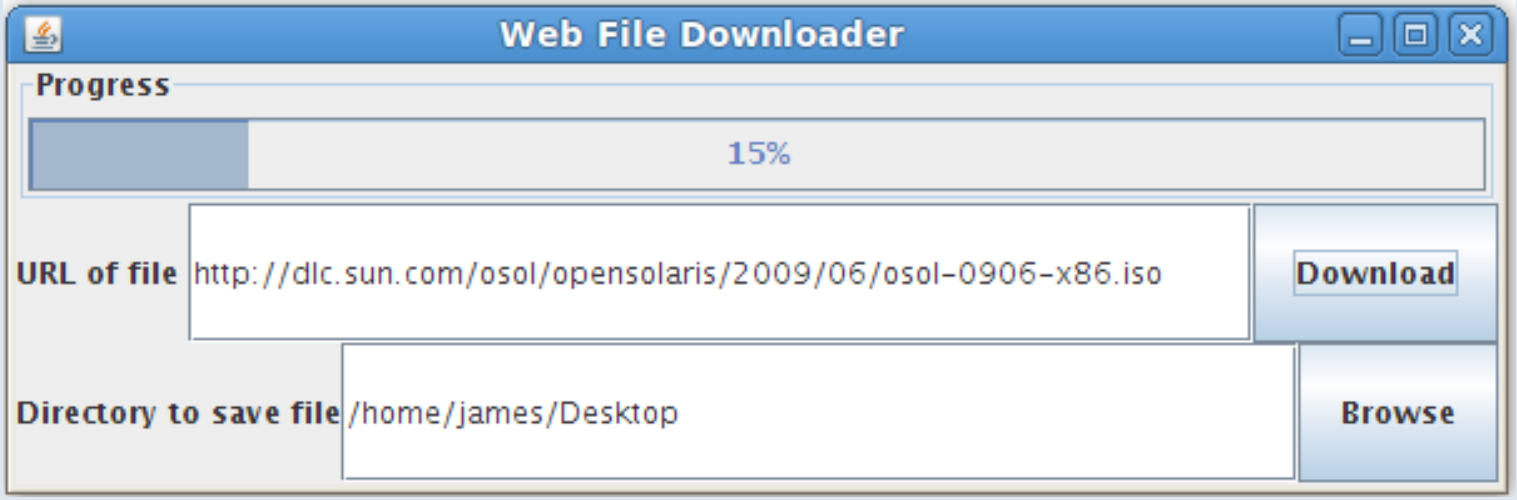
The uses have quit, we know this because of the EOFException caused by the stream

Downloading content from web servers

- So far we have dealt with designing our own clients and servers.
- What if we simply want to write a client to access an already we established protocol like HTTP, FTP, or any other service?
- We are in luck, Java gives us that ability to connect to another computer via a URL.
- The URL can then be used to access the files on the remote system.

Downloading content

- Please refer to the hand out for the example on using URL's to download files.



Test it out for yourself, try this URL
ftp://ftp.freebsd.org/pub/FreeBSD/doc/en_US.ISO8859-1/books/handbook/book.pdf.zip

Lab Assignment

- No Lab for today,

  Homework
- Page 1014 #30.13 (Multiple client chat)

Acknowledgments

Introduction to Java Programming by Y. Daniel Liang