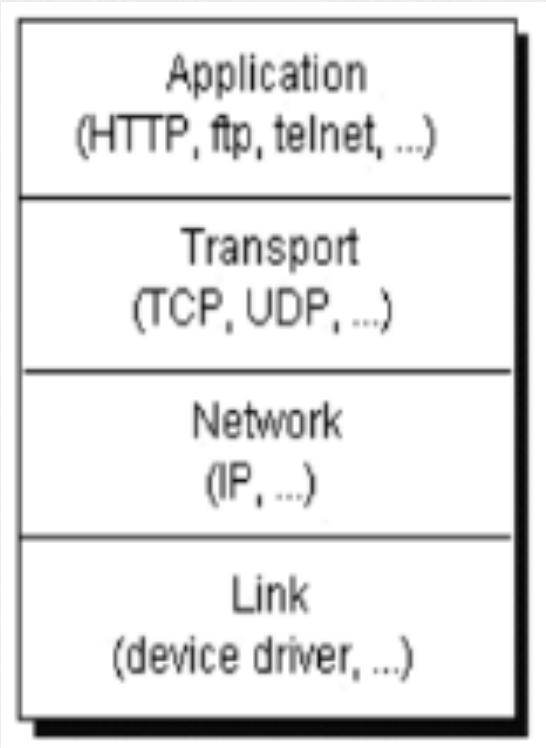

WEBONISE LAB

Networking in Java

Introduction

- When we write java programs that communicate over the network, we are programming at the application layer. Typically we don't need to concern ourself with the TCP and UDP layers.

- We use the classes in the java.net.package

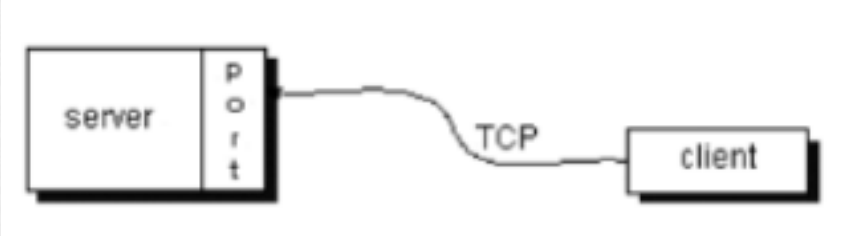


Protocols

- TCP(Transmission Control Protocol) : It is a connection based protocol that provides a reliable flow of data between computers.

- UDP(User Datagram Protocol) : is a protocol that sends the independent packets of data called datagram, from one computer to another with no guarantees about arrival. UDP is not connection based like TCP

- The TCP and UDP protocols use ports to map incoming data to a particular process running on a computer.



Protocols

- Java programs uses TCP and UDP to communicate over the network.
- Classes use TCP to communicate over the network
  - URL
  - URLConnection
  - Socket
  - ServerSocket
- Classes use UDP to communicate over the network
  - DatagramPacket
  - DatagramSocket

URL

- URL is acronym for Uniform Resource Locater and is a reference (an address) to a source on the internet.

In your program, you can use a `String` containing this text to create a `URL` object :

```
try {
URL gamelan = new URL("http://www.gamelan.com/");
} catch (MalfordUrlException ex) {
   // exception handler code here
}
```

Sample

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

public class ParseURL {
    public static void main(String[] args) throws Exception {
        URL aURL = new URL("http://java.sun.com:80/docs/books/tutorial"
                           + "/index.html?name=networking#DOWNLOADING");
        System.out.println("protocol = " + aURL.getProtocol());
           System.out.println("authority = " + aURL.getAuthority());
        System.out.println("host = " + aURL.getHost());
        System.out.println("port = " + aURL.getPort());
        System.out.println("path = " + aURL.getPath());
        System.out.println("query = " + aURL.getQuery());
        System.out.println("filename = " + aURL.getFile());
        System.out.println("ref = " + aURL.getRef());
    }
}
```

Output

protocol = http

authority = java.sun.com:80

host = java.sun.com

port = 80

path = /docs/books /tutorial/index.html

query = name=networking

filename = /docs/books/tutorial/index.heml?name=networking

ref = DOWNLOADING

Socket

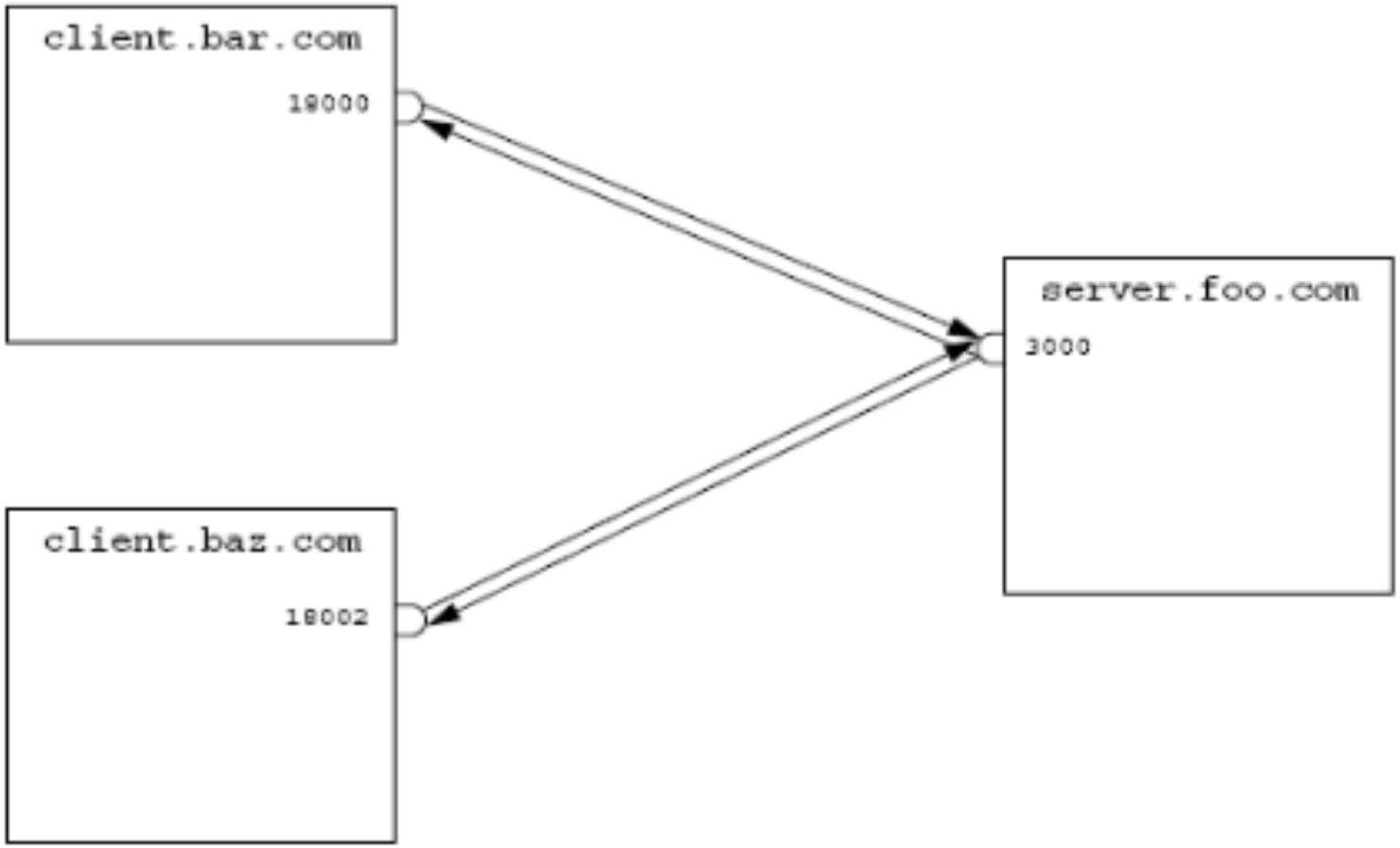
Sockets

- Sockets hold two streams: an input streams and an output streams.
- Each end of the socket has a pair of streams.

Setting up the Connection

Set up of a network connection is similar to a telephone system: One end must dial the other end, which must be listening.

Socket connection



Socket

- Java provides Socket based API that shields a programmer from lot of low level code writing while making it pretty simple to create network based applications.

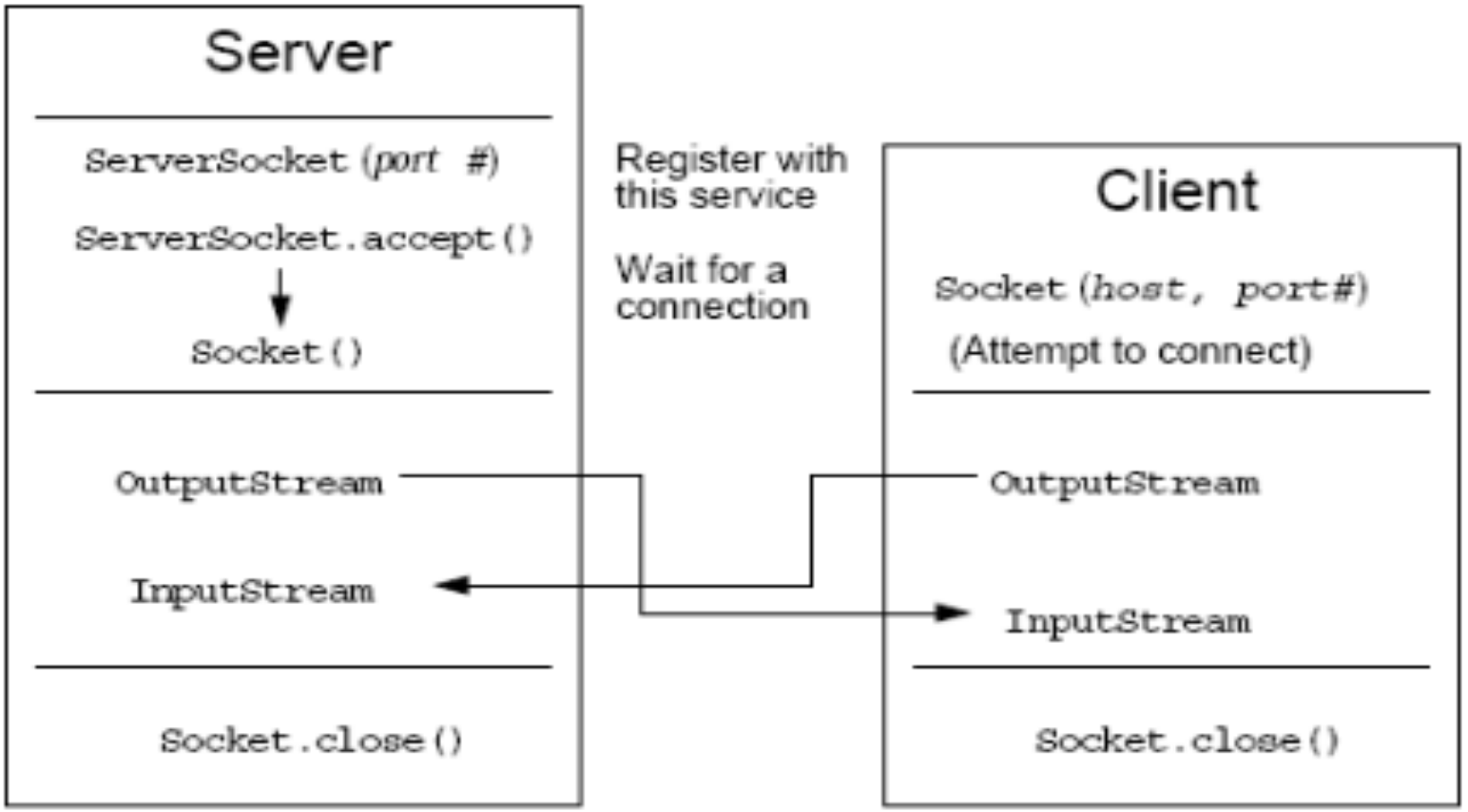
- A network socket is like an electric socket. Various plugs around the network have a standard way to deliver their payload. Anything that understands standard networking protocol can plug-in to the socket and communicate.

- In the networking environment, a **Socket** on the servers allows client to plug-in and access a server's resources. Servers Sockets allows a computer single handedly server different clients different kinds of information

Ports

- To address the connection, include the following:
  - The address or name of remote machine
  - A port number to identify the purpose at the server
- Port numbers range from 0-65535.

Setup C/S connection



C/S Connection

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

public class SimpleServer {
  public static void main(String args[]) {
    ServerSocket s = null;

    // Register your service on port 5432
    try {
      s = new ServerSocket(5432);
    } catch (IOException e) {
      e.printStackTrace();
    }
```

C/S Connection

```
// Run the listen/accept loop forever
while (true) {
  try {
    // Wait here and listen for a connection
    Socket s1 = s.accept();

    // Get output stream associated with the socket
    OutputStream s1out = s1.getOutputStream();
    BufferedWriter bw = new BufferedWriter(
      new OutputStreamWriter(s1out));

    // Send your string!
    bw.write("Hello Net World!\n");
```

C/S Connection

```
            // Close the connection, but not the server socket
            bw.close();
            s1.close();

          } catch (IOException e) {
            e.printStackTrace();
          } // END of try-catch

        } // END of while(true)

      } // END of main method

    } // END of SimpleServer program
```

C/S Connection

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

public class SimpleClient {

  public static void main(String args[]) {

    try {
      // Open your connection to a server, at port 5432
      // localhost used here
      Socket s1 = new Socket("127.0.0.1", 5432);

      // Get an input stream from the socket
      InputStream is = s1.getInputStream();
```

Thank You