Sockets

https://docs.oracle.com/javase/tutorial/networking/sockets/index.html

Sockets

Sockets are one end of a two-way connection between programs running on a network.

Programs use sockets to communicate (send data back and forth) over the Internet (or local networks).

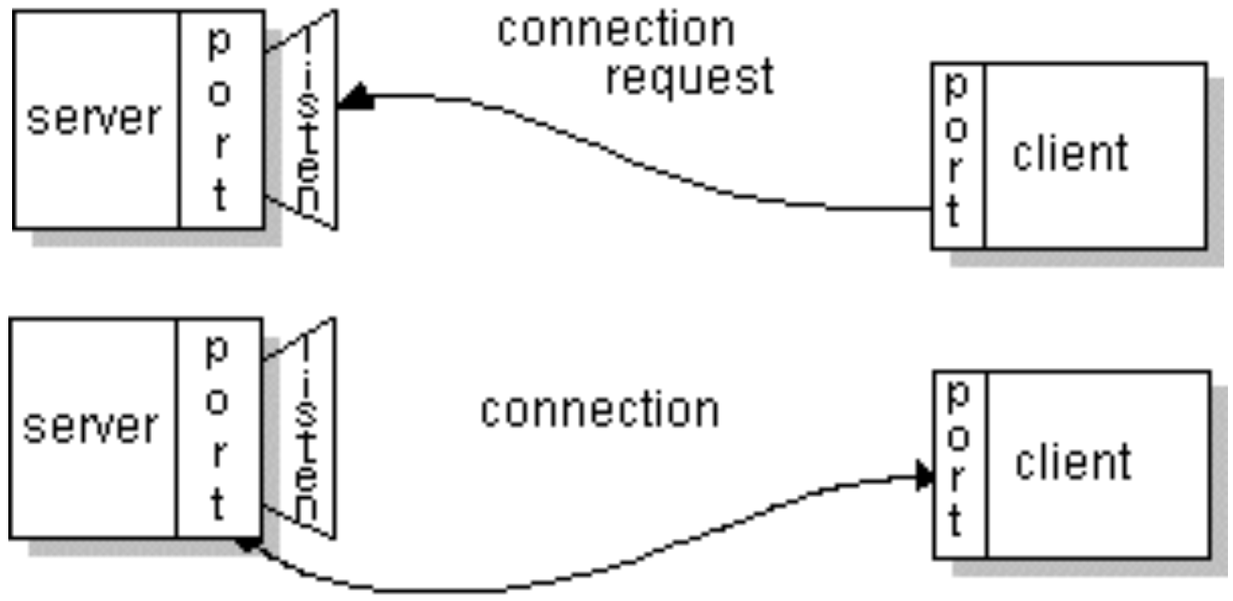
Sockets

Java provides two socket objects:

- java.net.Socket - which handles the client side connection

- java.net.ServerSocket - which handles the server side connection of the socket.

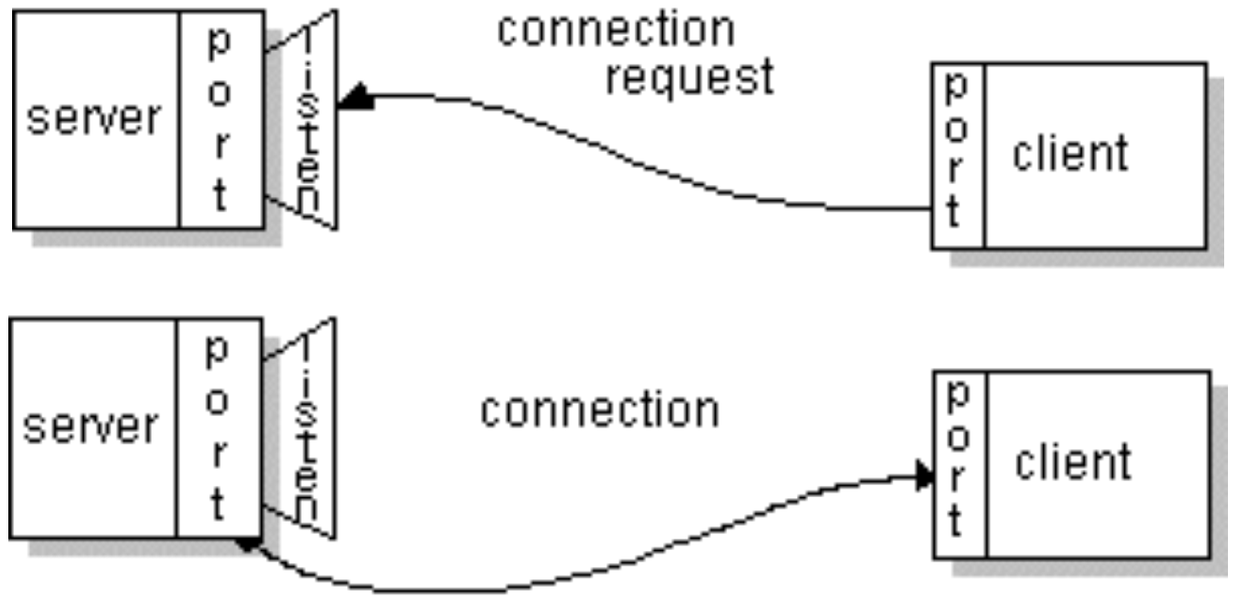
Sockets



The ServerSocket is *bound* to a port, which listens for incoming connections from clients.

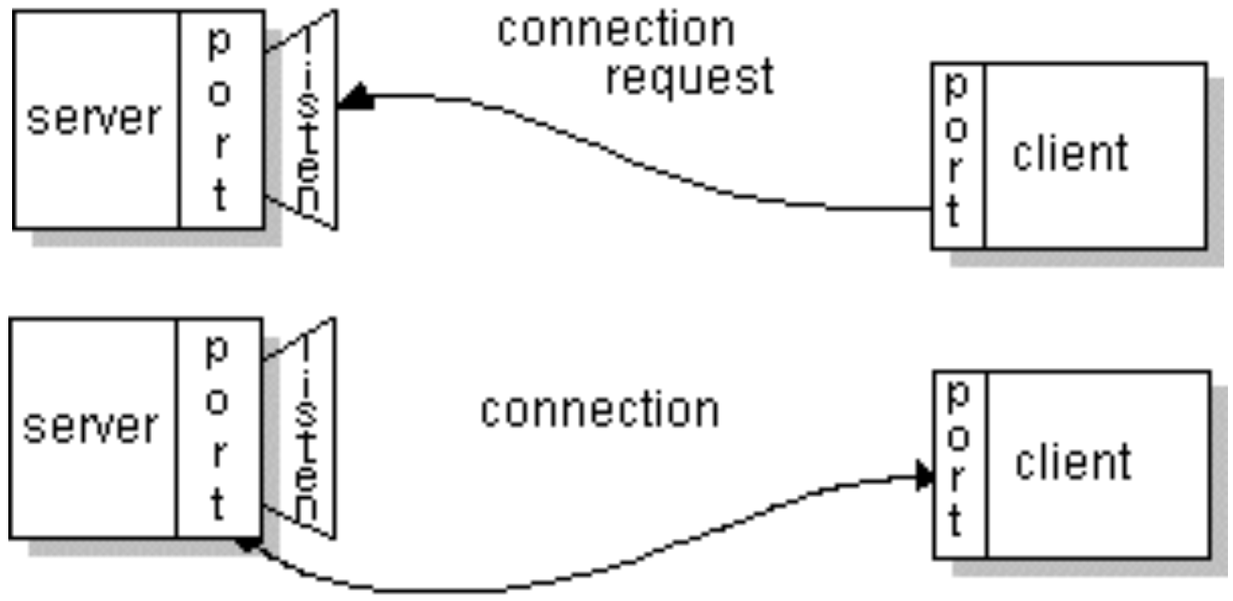
The client sockets connect to a hostname (like an ip address) and the port the ServerSocket is listening on.

Socket Definition



A *socket* is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.

Sockets



Java's Socket classes sit on top of native code, hiding system details from your Java program. This is how compiled Java clients and servers on *heterogeneous* systems (systems with different architectures and operating systems) can still work together.

Web Connections

For connecting to the web (web pages) it is probably more appropriate to use Java's URLConnection and URLEncoder classes (more on that later):

http://docs.oracle.com/javase/tutorial/networking/urls/index.html

Reading and Writing from a Socket

EchoClient.java

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

public class EchoClient {
    public static void main(String[] args) throws IOException {

        if (args.length != 2) {
            System.err.println(
                "Usage: java EchoClient <host name> <port number>");
            System.exit(1);
        }

        String hostName = args[0];
        int portNumber = Integer.parseInt(args[1]);

        try {
            Socket echoSocket = new Socket(hostName, portNumber);
            PrintWriter out = new PrintWriter(echoSocket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(new InputStreamReader(echoSocket.getInputStream()));
            BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));

            String userInput;
            while ((userInput = stdIn.readLine()) != null) {
                out.println(userInput);
                System.out.println("echo: " + in.readLine());
            }
        } catch (UnknownHostException e) {
            System.err.println("Don't know about host " + hostName);
            System.exit(1);
        } catch (IOException e) {
            System.err.println("Couldn't get I/O for the connection to " +
                hostName);
            System.exit(1);
        }
    }
}
```

http://docs.oracle.com/javase/tutorial/networking/sockets/examples/EchoClient.java

Socket I/O

```
if (args.length != 2) {
    System.err.println(
        "Usage: java EchoClient <host name> <port number>");
    System.exit(1);
}

String hostName = args[0];
int portNumber = Integer.parseInt(args[1]);
```

The EchoClient takes two arguments, the host name (which could also be an ip address) and the port number that the ServerSocket is listening on.

Socket I/O

```
String hostName = args[0];
int portNumber = Integer.parseInt(args[1]);

try {
    Socket echoSocket = new Socket(hostName, portNumber);
    PrintWriter out = new PrintWriter(echoSocket.getOutputStream(), true);

    BufferedReader in =
        new BufferedReader(
            new InputStreamReader(echoSocket.getInputStream()));

    BufferedReader stdIn =
        new BufferedReader(
            new InputStreamReader(System.in));
```

The above code gives an example of how to connect to a ServerSocket with a client Socket.

Once you get a socket connection, you can treat it extremely similar to a file; using the same BufferedReader and BufferedWriter classes and methods.

Socket I/O

```
String hostName = args[0];
int portNumber = Integer.parseInt(args[1]);

try (
    Socket echoSocket = new Socket(hostName, portNumber);
    PrintWriter out = new PrintWriter(echoSocket.getOutputStream(),
true);

    BufferedReader in =
        new BufferedReader(
            new InputStreamReader(echoSocket.getInputStream()));

    BufferedReader stdIn =
        new BufferedReader(
            new InputStreamReader(System.in))
  )
```

Note that the EchoClient gets a buffered reader both from System.in (what you type on the keyboard) and from the echoSocket (what the ServerSocket sends back).

Socket I/O

```
String userInput;
while ((userInput = stdIn.readLine()) != null) {
    out.println(userInput);
    System.out.println("echo: " + in.readLine());
}
```

So what the echo client does after connecting is repeatedly read lines from standard input and then writes those lines to the server socket (which is the ‘out’ variable).

It then reads the line sent back from the server socket and prints it to the screen.

Writing Server Sockets

EchoServer.java

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

public class EchoServer {
    public static void main(String[] args) throws IOException {
        if (args.length != 1) {
            System.err.println("Usage: java EchoServer <port number>");
            System.exit(1);
        }

        int portNumber = Integer.parseInt(args[0]);

        try {
            ServerSocket serverSocket =
                new ServerSocket(Integer.parseInt(args[0]));
            Socket clientSocket = serverSocket.accept();
            PrintWriter out =
                new PrintWriter(clientSocket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));

                System.out.println(“The server is listening at: “ +
                    serverSocket.getInetAddress() + “ on port “ +
                    serverSocket.getLocalPort());
            String inputLine;
            while ((inputLine = in.readLine()) != null) {
                out.println(inputLine);
            }
        } catch (IOException e) {
            System.out.println("Exception caught when trying to listen on port "
                + portNumber + " or listening for a connection");
            System.out.println(e.getMessage());
        }
    }
}
```

http://docs.oracle.com/javase/tutorial/networking/sockets/examples/EchoServer.java

Writing a Server Socket

```
if (args.length != 1) {
     System.err.println("Usage: java EchoServer <port number>");
     System.exit(1);
 }

 int portNumber = Integer.parseInt(args[0]);
```

The Echo Server only takes one argument, the port it will listen on.

Writing a Server Socket

```
try (
    ServerSocket serverSocket =
        new ServerSocket(Integer.parseInt(args[0]));
    Socket clientSocket = serverSocket.accept();
    PrintWriter out =
        new PrintWriter(clientSocket.getOutputStream(), true);
    BufferedReader in = new BufferedReader(
        new InputStreamReader(clientSocket.getInputStream()));
) {
```

The ServerSocket gets created listening to the port, and then waits for an incoming client connection.

The accept method blocks until a client has made a connection. (Obvious question: how do you handle multiple clients? threads!)

Writing a Server Socket

```
try (
    ServerSocket serverSocket =
        new ServerSocket(Integer.parseInt(args[0]));
    Socket clientSocket = serverSocket.accept();
    PrintWriter out =
        new PrintWriter(clientSocket.getOutputStream(), true);
    BufferedReader in = new BufferedReader(
        new InputStreamReader(clientSocket.getInputStream()));
) {
```

The ServerSocket creates streams for input and output similar to the client Socket.

Writing a Server Socket

```
            System.out.println("The server is listening at: " +
serverSocket.getInetAddress() + " on port " + serverSocket.getLocalPort());
            String inputLine;
            while ((inputLine = in.readLine()) != null) {
                out.println(inputLine);
            }
```

The getInetAddress and getLocalPort methods can get the hostname/ip and the port a server socket is listening on.

Writing a Server Socket

```
            System.out.println("The server is listening at: " +
serverSocket.getInetAddress() + " on port " + serverSocket.getLocalPort());
            String inputLine;
            while ((inputLine = in.readLine()) != null) {
                out.println(inputLine);
            }
```

The server socket repeatedly gets lines from the client, and then responds with the same line (thats why it's called an "echo").