M19_SAVI1674_06_SE_C19.indd Page 1106 13/03/15 5:27 PM f-w-155 /203/PH01992/9780134041674_SAVITCH/SAVITCH_ABSOLUTE_JAVA6_SE_9780134041674/SE/MA...

19.2 Networking with Stream Sockets

Since in order to speak, one must first listen, learn to speak by listening.

MEVLANA RUMI

Transmission Control Protocol (TCP)

When computers want to communicate with each other over a network, each computer must speak the same "language." In other words, the computers need to communicate using the same *protocol*. One of the most common protocols today is **TCP**, or the **Transmission Control Protocol**. For example, the HTTP protocol used to transmit Web pages is based on TCP. TCP is a stream-based protocol in which a stream of data is transmitted from the sender to the receiver. TCP is considered a reliable protocol because it guarantees that data from the sender is received in the same order in which it was sent. An analogy to TCP is the telephone system. A connection is made when the phone is dialed and the participants communicate by speaking back and forth. In TCP, the receiver must first be listening for a connection, the sender initiates the connection, and then the sender and receiver can transmit data. The program that is waiting for a connection is called the **server** and the program that initiates the connection is called the **client**.

server

client

User Datagram Protocol (UDP)

An alternate protocol is **UDP**, or the **User Datagram Protocol**. In UDP, packets of data are transmitted but no guarantee is made regarding the order in which the packets are received. An analogy to UDP is the postal system. Letters that are sent might be received in an unpredictable order, or lost entirely with no notification. Although Java provides support for UDP, we will only introduce TCP in this section.

Sockets

sockets

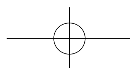
port

Network programming is implemented in Java using **sockets**. A socket describes one end of the connection between two programs over the network. A socket consists of an address that identifies the remote computer and a **port** for both the local and remote computer. The port is assigned an integer value between 0 and 65,535 that is used to identify which program should handle data received from the network. Two applications may not bind to the same port. Typically, ports 0 to 1,024 are reserved for use by well-known services implemented by your operating system.

VideoNote **Networking with Streams**

The process of client/server communication is shown in Display 19.6. First, the server waits for a connection by listening on a specific port. When a client connects to this port, a new socket is created that identifies the remote computer, the remote port, and the local port. A similar socket is created on the client. Once the sockets are created on both the client and the server, data can be transmitted using streams in a manner very similar to the way we implemented file I/O in Chapter 10.

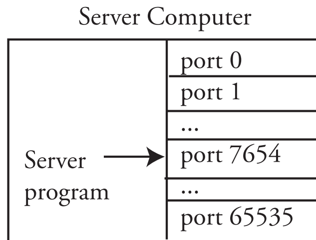
Display 19.7 shows how to create a simple server that listens on port 7654 for a connection. Once it receives a connection, a new socket is returned by the `accept( )` method. From this socket, we create a `BufferedReader`, just as if we were reading from a text file described in Chapter 10. Data is transmitted to the socket using a `DataOutputStream`, which is similar to a `FileOutputStream`. The `ServerSocket`



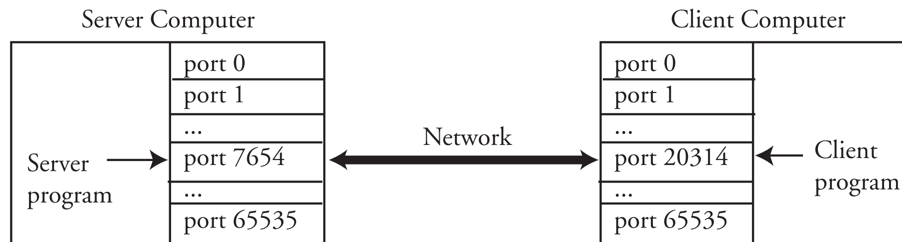
M19_SAVI1674_06_SE_C19.indd Page 1107 13/03/15 5:27 PM f-w-155-user /203/PH01992/9780134041674_SAVITCH/SAVITCH_ABSOLUTE_JAVA6_SE_9780134041674/SE/MA... EQA

Display 19.6 **Client/Server Network Communication through Sockets**

1. The server listens and waits for a connection on port 7654.



2. The client connects to the server on port 7654. It uses a local port that is assigned automatically, in this case, port 20314.



The server program can now communicate over a socket bound locally to port 7654 and remotely to the client's address at port 20314.

The client program can now communicate over a socket bound locally to port 20314 and remotely to the server's address at port 7654.

and `Socket` classes are in the `java.net` package, while the `BufferedReader` and `DataOutputStream` classes are in the `java.io` package. Once the streams are created, the server expects the client to send a name. The server waits for the name with a call to `readLine( )` on the `BufferedReader` object and then sends back the name concatenated with the current date and time. Finally, the server closes the streams and sockets.

localhost

Display 19.6 shows how to create a client that connects to our date and time server. First, we create a socket with the name of the computer running the server along with the corresponding port of 7654. If the server program and client program are running on the same computer, then you can use **`localhost`** as the name of the machine. Your computer understands that any attempt to connect across a network to the machine named `localhost` really corresponds to a connection with itself. Otherwise, the hostname should be set to the name of the computer (e.g., my.server.com). After a connection is made, the client creates stream objects, sends its name, waits for a reply, and prints the reply.

M19_SAVI1674_06_SE_C19.indd Page 1108 13/03/15 5:27 PM f-w-155-user /203/PH01992/9780134041674_SAVITCH/SAVITCH_ABSOLUTE_JAVA6_SE_9780134041674/SE/MA... EQA

Display 19.7 Date and Time Server (part 1 of 2)

```
 1  import java.util.Date;
 2  import java.net.ServerSocket;
 3  import java.net.Socket;
 4  import java.io.DataOutputStream;
 5  import java.io.BufferedReader;
 6  import java.io.InputStreamReader;
 7  import java.io.IOException;

 8  public class DateServer
 9  {
10    public static void main(String[] args)
11    {
12        Date now = new Date();

13        try
14        {
15          System.out.println("Waiting for a connection on port 7654.");
16          ServerSocket serverSock = new ServerSocket(7654);
17          Socket connectionSock = serverSock.accept();

18          BufferedReader clientInput = new BufferedReader(
19              new InputStreamReader(connectionSock.getInputStream()));
20          DataOutputStream clientOutput = new DataOutputStream(
21              connectionSock.getOutputStream());

22          System.out.println("Connection made, waiting for client " +
23              "to send their name.");
24          String clientText = clientInput.readLine();
25          String replyText = "Welcome, " + clientText +
26              ", Today is " + now.toString() + "\n";
27          clientOutput.writeBytes(replyText);
28          System.out.println("Sent: " + replyText);

29          clientOutput.close();
30          clientInput.close();
31          connectionSock.close();
32          serverSock.close();
33        }
34        catch (IOException e)
35        {
            System.out.println(e.getMessage());
36        }
37    }
38  }
```

M19_SAVI1674_06_SE_C19.indd Page 1109 13/03/15 5:27 PM f-w-192 /203/PH01992/9780134041674_SAVITCH/SAVITCH_ABSOLUTE_JAVA6_SE_9780134041674/SE/MA... EQA

Display 19.7 **Date and Time Server** (part 2 of 2)

Sample Dialogue *Output when the client program in Display 19.8 connects to the server program*

```
Waiting for a connection on port 7654.
Connection made, waiting for client to send their name.
Sent: Welcome, Dusty Rhodes, Today is Sun Mar 1 12:18:21 AKDT 2015
```

Display 19.8 **Date and Time Client** (part 1 of 2)

```
1  import java.net.Socket;
2  import java.io.DataOutputStream;
3  import java.io.BufferedReader;
4  import java.io.InputStreamReader;
5  import java.io.IOException;

6  public class DateClient
7  {
8    public static void main(String[] args)
9    {
10       try
11       {
12         String hostname = "localhost";
13         int port = 7654;

14         System.out.println("Connecting to server on port " + port);
15         Socket connectionSock = new Socket(hostname, port);

16         BufferedReader serverInput = new BufferedReader(
17             new InputStreamReader(connectionSock.getInputStream()));
18         DataOutputStream serverOutput = new DataOutputStream(
19             connectionSock.getOutputStream());

20         System.out.println("Connection made, sending name.");
21         serverOutput.writeBytes("Dusty Rhodes\n");

22         System.out.println("Waiting for reply.");
23         String serverData = serverInput.readLine();
24         System.out.println("Received: " + serverData);

25         serverOutput.close();
26         serverInput.close();
27         connectionSock.close();
28       }
```

`localhost` refers to the same, or local, machine that the client is running on. Change this string to the appropriate hostname (e.g., my.server.com) if the server is running on a remote machine.

(continued)

M19_SAVI1674_06_SE_C19.indd Page 1110 13/03/15 5:27 PM f-w-155-user /203/PH01992/9780134041674_SAVITCH/SAVITCH_ABSOLUTE_JAVA6_SE_9780134041674/SE/MAI... EQA

Display 19.8 **Date and Time Client** (part 2 of 2)

```
29          catch (IOException e)
30          {
31            System.out.println(e.getMessage());
32          }
33      }
34  }
```

Sample Dialogue *Output when client program connects to the server program in Display 19.7*

```
Connecting to server on port 7654
Connection made, sending name.
Waiting for reply.
Received: Welcome, Dusty Rhodes, Today is Sun Mar 1 12:18:21 AKDT 2015
```

Note that the socket and stream objects throw checked exceptions. This means that their exceptions must be caught or declared in a `throws` block.

Sockets and Threading

blocking

If you run the program in Display 19.7, then you will notice that the server waits, or **blocks**, at the `serverSock.accept( )` call until a client connects to it. Both the client and server also block at the `readLine( )` call if data from the socket is not yet available. In a client with a GUI, you would notice this as a nonresponsive program while it is waiting for data. For the server, this behavior makes it difficult to handle connections with more than one client. After a connection is made with the first client, the server will become nonresponsive to the client's requests while it waits for a second client.

The solution to this problem is to use threads. One thread will listen for new connections while another thread handles an existing connection. Section 19.1 describes how to create threads and make a GUI program responsive. On the server, the `accept( )` call is typically placed in a loop and a new thread is created to handle each client connection:

```
while (true)
{
  Socket connectionSock = serverSock.accept( );
  ClientHandler handler = new ClientHandler(connectionSock);
  Thread theThread = new Thread(handler);
  theThread.start( );
}
```

In this code, `ClientHandler` is a class that implements `Runnable`. The constructor keeps a reference to the socket in an instance variable, and the `run( )` method would handle all communications. A complete implementation of a threaded server is left as Programming Projects 19.7 and 19.8.

M19_SAVI1674_06_SE_C19.indd Page 1111 13/03/15 5:27 PM f-w-192 /203/PH01992/9780134041674_SAVITCH/SAVITCH_ABSOLUTE_JAVA6_SE_9780134041674/SE/MA...

The URL Class

Java's URL class will retrieve the HTML from a website into a stream while eliminating several details involved in creating a socket. The URL class also illustrates the flexibility of streams and the power of polymorphism. Code that reads from the keyboard or from a file can be used almost verbatim to read from a website; all we need to do is change the source of the stream that is connected to the Scanner object. To use the URL class, import java.net.URL, create a URL object, and then use the stream when creating a Scanner object. From that point on, reading from the Scanner will read data from the URL specified in the URL object. The following code listing will output the HTML of www.wikipedia.org:

```
URL website = new
    URL("http://www.wikipedia.org");
Scanner inputStream = new Scanner(
    new InputStreamReader(
          website.openStream()));

while (inputStream.hasNextLine())
{
  String s = inputStream.nextLine();
  System.out.println(s);
}
inputStream.close();
```

Self-Test Exercises

5. What is the purpose of a port in the context of a socket?
6. Consider a threaded server that is expected to have up to 100 clients connected to it at one time. Why might this server require a large amount of resources such as memory, disk space, or processor time?

19.3 JavaBeans

Insert tab A into slot B.

Common assembly instruction

JavaBeans

JavaBeans refers to a framework that facilitates software building by connecting software components from diverse sources. Some of the components might be standard existing pieces of software. Some might be designed for the particular application. Typically, the various components were designed and coded by different teams. If the components are all designed within the JavaBeans framework, it simplifies the process of integrating the components and means that the components produced can more easily be reused for future software projects. JavaBeans have been widely used. For example, the AWT and Swing packages were built within the JavaBeans framework.