Internet Networking
Introduction

In this module, you will learn what goes on “under the hood” when you send an email message or when you retrieve a web page from a remote server. You will also learn how to write your own programs that fetch data from sites across the Internet and how to write server programs that can serve information to other programs.

Objectives

- To understand the concept of sockets
- To learn how to send and receive data through sockets
- To implement network clients and servers
- To communicate with web servers and server-side applications through the HTTP

The Internet Protocol and Application Level Protocols

Before starting networking programming in Java, you should have an understanding of some networking basics. You need to familiarize yourself with general Internet knowledge and those terms that are frequently used on the networks, such as Internet, WWW, Internet Protocol, TCP/IP, TCP connection, HTTP, URL etc.. These topics are covered in Chapter 24.1 and 24.2. A brief summary is presented below.

- The Internet is a worldwide collection of networks, routing equipment, and computers using a common set of protocols to define how each party will interact with each other.

- The World Wide Web (or the Web), is an Internet service that links together millions of electronic documents called Web pages located on computers around the world.

- Internet Services are the services developed by the Internet community to help user exchange data and access resources, such as Electronic mail (Email), Internet Relay Chat (IRC), File Transfer Protocol (FTP), WWW etc.

- TCP/IP is the abbreviation for Transmission Control Protocol over Internet Protocol, the pair of communication protocols that is used to establish reliable transmission of data between two computers on the Internet.

- A TCP connection requires the Internet addresses and port numbers of both end points.

- HTTP, or Hypertext Transfer Protocol, is the protocol that defines communication between web browsers and web servers.

- An URL, or Uniform Resource Locator, is a pointer to an information resource (such as a web page or an image) on the World Wide Web.

Computers running on the Internet communicate to each other using either the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP), as this diagram illustrates:
When you write Java programs that communicate over the network, you are programming at the application layer. Typically, you don't need to concern yourself with the TCP and UDP layers. Instead, you can use the classes in the java.net package. These classes provide system-independent network communication. However, to decide which Java classes your programs should use, you do need to understand how TCP and UDP differ.

Please read through Chapter 24.1 and 24.2 for the detailed explanation.

**Socket, Client/Server**

A network socket is a lot like an electrical socket. Various plugs around the network have a standard way of delivering their payload. Anything that understands the standard protocol can “plug in” to the socket and communicate. The same idea applies to network sockets, except we talk about TCP/IP packets and IP addresses rather than electrons and street addresses. Internet Protocol (IP) is a low-level routing protocol that breaks data into small packets and sends them to an address across a network, which does not guarantee to deliver said packets to the destination. Transmission Control Protocol (TCP) is a higher-level protocol that manages to robustly string together these packets, sorting and retransmitting them as necessary to reliably transmit your data.

You often hear the term client/server mentioned in the context of networking. A server is anything that has some resource that can be shared. There are compute servers, which provide computing powers; web servers, which store web pages. A client is simply any other entity that wants to gains access to a particular server. The server is a permanently available resource, while the client is free to “unplug” after it has been served.

Please note a network socket is different from a port. A socket is an operating system abstraction similar to a file descriptor; it is part of the Application Program Interface (API). A program creates a socket, specifies that it will be used with TCP/IP, and then fills in details such as whether the socket will be used by a client or a server. A port is a transport-layer abstraction that is part of the TCP/IP suite.

Note that after creating a socket, a program specifies a port to be used with that socket.
**A Client Program**

In Java, you use Socket object to establish TCP connection. To communicate with the other end point of the connection, use the input and output streams attached to the socket.

For example:

```java
Socket s = new Socket (hostname, portnumber); //create Socket object
InputStream instream = s. getInputStream();   //get input stream
OutputStream outstream = s.getOutputStream(); //get output stream

// transmission data here, read, write etc.

s.close(); //close socket
```

The WebGet.java example in the text book page 914-916 is an example of client programs.

**A Server Program**

The server program waits for clients to connect to a particular port. To listen to incoming connections, you use a server socket. The `ServerSocket` class is used by server applications to listen for client connections. The `accept` method of the `ServerSocket` waits for a client connection, then the server program obtains a socket through which it communicates with the client.

For example:

```java
ServerSocket server = new ServerSocket (8888); //8888 is port No.
Socket s = server.accept(); //wait for client

BankService service = new BankService(s,bank); // carry out the service
```

A Bank service example is described in the text book. Please read through Chapter section 24.4 on your own.

**URL Connections**

If you are trying to connect to the Web, the URL class and related classes (URLConnection, URLEncoder) are probably more appropriate than the socket classes. The `URLConnection` class makes it easy to communicate with a web server without having to issue HTTP commands. In fact, URLs are a relatively high-level connection to the Web and use sockets as part of the underlying implementation.

**Create a URL object:**

The easiest way to create a URL object is from a String that represents the human-readable form of the URL address. This is typically the form that another person will use for a URL. For example, the URL for the CQU site, takes the following form:

```java
http://www.cqu.edu.au/
```

In your Java program, you can use a String containing this text to create a URL object:
URL cqu = new URL("http://www.cqu.edu.au/");
The URL object created above represents an absolute URL. An absolute URL contains all of the information necessary to reach the resource in question. You can also create URL objects from a relative URL address. For example,

    URL cqu = new URL("http://www.cqu.edu.au/");
    URL student = new URL (cqu, “students/index.htm”);

Get the URLConnection object:
Once you have a URL object, which is where you are going to connect, you then need to get a URLConnection object.

    URLConnection connection = cqu.openConnection();

Then you call the getInputStream method to obtain an input stream,

    InputStream instream = connection.getInputStream();

Now, you are able to read URL files from the input stream, you can also turn the stream into a scanner in the usual way, and read input from the scanner.

The program example URLGet.java in page 926 and page 927 demonstrates how to retrieve a web page file by using URL Connection in Java program.

Actually, you can retrieve web page information via a Java program directly. Examine the following example.

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

class URLReader
{
    public static void main(String[] args) throws Exception
    {
        URL yahoo = new URL("http://www.yahoo.com/");
        BufferedReader in = new BufferedReader(
        new InputStreamReader( yahoo.openStream()));

        String inputLine;

        while ((inputLine = in.readLine()) != null)
        {
            System.out.println(inputLine);
            in.close();
        }
    }
}
```
Activity

The Exercise 24.9 (page 932) asks you to write a chat server and client program. The chat server accepts connections from clients. Whenever one of the clients sends a chat message, it is displayed for all other clients to see. Use a protocol with three commands: LOGIN name, CHAT message, and LOGOUT.

The solution to this application is attached below (from Big Java book site), you can try these programs and save them in a specific directory, compile all these java files. After that, start ChatServer from a console window (e.g. java ChatServer). Open another console window, run ChatClient (e.g. java ChatClient). Open the 3rd console window, run ChatClient (e.g. java ChatClient) again. Now, you should be able to send chat message between these two chat client windows. The following screen shots are some chat examples. You may like to try it.

Programs (Consist of 5 programs):

- ChatServer.java

```java
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

/**
   * A server that executes the Simple Chat Room Access Protocol.
   * It can accept simultaneous connections from multiple clients.
   */
public class ChatServer
{
    public static void main(String[] args) throws IOException
    {
        final int ROOM_SIZE = 10;
        ChatRoom chatRoom = new ChatRoom(ROOM_SIZE);
        final int PORT = 8888;
        ServerSocket server = new ServerSocket(PORT);
```
System.out.println("Waiting for chatters to connect...");

Socket s;
while (true)
{
    s = server.accept();
    ChatService service = new ChatService(s, chatRoom);
    Thread t = new Thread(service);
    t.start();
}
}
ChatClient.java

```java
import java.io.InputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;

/**
 * This program tests the chat server.
 */
public class ChatClient
{
    public static void main(String[] args)
        throws IOException
    {
        final int PORT = 8888;
        final String HOST = "localhost";

        System.out.println("Welcome to the chat room!\n");
        System.out.println("Please enter your command.\n");
        System.out.println("Usage:  LOGIN username\n");
        System.out.println("        CHAT message\n");
        System.out.println("        LOGOUT\n");
        System.out.println("Press ENTER to send your message.\n");

        Socket s = new Socket(HOST, PORT);
        InputStream inStream = s.getInputStream();
        OutputStream outStream = s.getOutputStream();
        final Scanner in = new Scanner(inStream);
        final PrintWriter out = new PrintWriter(outStream);
        final Scanner console = new Scanner(System.in);

        class InputRunnable implements Runnable
        {
            public void run()
            {
                while (!Thread.interrupted())
                {
                    String line = console.nextLine();
                    out.println(line);
                    out.flush();
                }
            }
        }

        InputRunnable runnable = new InputRunnable();
        Thread t = new Thread(runnable);
        t.start();

        while (in.hasNextLine())
        {
            String response = in.nextLine();
            System.out.println(response);
        }
        t.interrupt();
        System.out.println("Hit ENTER to exit.");
    }
}
```
import java.util.HashMap;
import java.util.Collection;
import java.util.Iterator;
import java.util.ArrayList;
import java.util.ListIterator;
import java.io.PrintWriter;
/**
 * A chat room consisting of multiple chatters.
 */
public class ChatRoom
{
    /**
     * Constructs a chat room with a given number of capacity.
     * @param aCapacity the capacity of the room
     */
    public ChatRoom(int aCapacity)
    {
        capacity = aCapacity;
        chatterHash = new HashMap<String, Chatter>(capacity);
        activeService = new ArrayList<ChatService>(capacity);
    }
    /**
     * Register a chatter to the room.
     * @param aName the name to register
     */
    public void register(String aName)
    {
        chatterHash.put(aName, new Chatter(aName));
    }
    /**
     * De-register a chatter to the room.
     * @param aName the name to de-register
     */
    public void leave(String aName)
    {
        chatterHash.remove(aName);
    }
    public void add(ChatService cs)
    {
        activeService.add(cs);
    }
    /**
     * Broadcast a message to everyone in the room.
     * @param msg the message to be broadcast
     * @param out the place to write the message
     */
    public void broadcast(String requestor, String msg, ChatService chatService)
    {
        for (ChatService cs : activeService)
        {
            if (cs != chatService && cs.getUserName() != null)
            {
                cs.putMessage(requestor + ':' + msg);
            }
        }
    }
    private ArrayList<ChatService> activeService;
    int capacity;
    private HashMap<String, Chatter> chatterHash;
ChatService.java

```java
import java.io.InputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;

/**
   * Executes Simple Chat Room Access Protocol commands from a socket.
   */
public class ChatService implements Runnable {
    /**
     * Constructs a service object that processes commands from a socket for a chat room.
     * @param aSocket the socket
     * @param aChatRoom the chat room
     */
    public ChatService(Socket aSocket, ChatRoom aChatRoom) {
        s = aSocket;
        chatRoom = aChatRoom;
        chatRoom.add(this);
        loggedIn = false;
        loggedOut = false;
    }

    /**
     * Executes all commands until the LOGOUT command or the end of input.
     */
    public void run() {
        try {
            try {
                InputStream inStream = s.getInputStream();
                OutputStream outStream = s.getOutputStream();
                in = new Scanner(inStream);
                out = new PrintWriter(outStream);
                while (!loggedOut && in.hasNext()) {
                    String command = in.next();
                    String response = executeCommand(command);
                    putMessage(response);
                }
            }
            finally {
                s.close();
            }
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```
/**
   * Send the message through the socket.
   * @param msg the message needs to be send
   */
   public void putMessage(String msg)
   {
      if (out != null)
      {
         out.println(msg);
         out.flush();
      }
   }

   /**
   * Executes a single command.
   * @param command the command
   * @return the reply to send to the client.
   */
   public String executeCommand(String command)
   {
      if (command.equals("LOGIN"))
      {
         userName = in.next();
         chatRoom.register(userName);
         chatRoom.broadcast(userName, "LOGIN", this);
         loggedIn = true;
         return "Chat Room Manager: Hello, " + userName + ".";
      }
      else if (!loggedIn)
      {
         in.nextLine(); // ignore the rest of the command
         return "ChatRoom Manager: You must LOGIN first";
      }
      else if (command.equals("CHAT"))
      {
         String message = in.nextLine();
         chatRoom.broadcast(userName, message, this);
         return userName + ": " + message;
      }
      else if (command.equals("LOGOUT"))
      {
         chatRoom.broadcast(userName, "LOGOUT", this);
         chatRoom.leave(userName);
         loggedOut = true;
         return "Good-Bye!";
      }
      in.nextLine(); // ignore the rest of the command
      return "Chat Room Manager: Invalid command";
   }

   /**
   * Returns the user name of this service.
   * @return the user name of this service
   */
   public String getUserName()
   {
      return userName;
   }

   private String userName;
   private Socket s;
   private ChatRoom chatRoom;
   private PrintWriter out;
   private Scanner in;
   private boolean loggedIn;
private boolean loggedOut;
}

- Chatter.java

```java
/**
 * A class represent a chat room user.
 */
public class Chatter {
  /**
   * Constructs a chatter with a name.
   */
  public Chatter(String aName) {
    name = aName;
  }
  /**
   * Returns the name
   * @return the name
   */
  public String getName() {
    return name;
  }
  String name;
}

Review questions
Programming exercises: None

References