Java Network Programming

- The ***java.net*** package contains the ***Socket*** class. This class speaks TCP (connection-oriented protocol).

- The ***DatagramSocket*** class uses UDP (connectionless protocol).

- The java.net.Socket class represents a single side of a socket connection on either the client or server. In addition, the server uses the ***java.net.ServerSocket*** class to wait for connections from clients.

- The server creates a *ServerSocket* object and waits, blocked in a call to its *accept()* method, until a connection arrives. When a connection request arrives, the *accept()* creates a *Socket* object. The server uses this *Socket* object to communicate with the client.

E.g. of a client:

```
try {
        Socket server = new Socket("foo.bar.com", 2500);
        InputStream in = server.getInputStream();
        OutputStream out = server.getOutputStream();
        //sends a string
        PrintStream pout = new PrintStream(out);
        Pout.println("Hello!");

        //Read a string
        DataInputStream din = new DataInputStream(in);
        String response = din.readLine();
        Server.close();
}
catch (IOException e) {
        System.out.println("Error connecting to host");
}
```

E.g. of a server:

```
try {
        ServerSocket listener = new ServerSocket(2500);
        while (!finished) {
                Socket aclient = listener.accept(); //waits for a
                                                    //connection
                InputStream in = aclient.getInputStream();
                OutputStream out = aclient.getOutputStream();
                //Read a string
                DataInputStream din = new DataInputStream(in);
                String request = din.readLine();

                //Write a string
                PrintStream pout = new PrintStream(out);
                pout.println("Goodbye!");
                aclient.close();
        }
        listener.close();
}
catch(IOException e) { }
```

For Multithreaded Servers:

```
import java.net.*;
import java.io.*;
import java.util.*;
public class TinyHttpd {
        public static void main(String argv[]) throws IOException {
                int port = Integer.parseInt(argv[0]);
                Serversocket ss = new Serversocket(port);
                while (true)
                                new TinyHttpdConn(ss.accept());
                                //accept() returns a new socket to handle client
        }
}

//create a thread to service each web client request
class TinyHttpdConn extends Thread {
        Socket sock;
        TinyHttpdConn(Socket s) {          //constructor
                        sock = s;
                        setPriority(NORM_PRIORITY - 1);
                        start();
        }
        public void run() {
                        try { …}
        }
}
```

Java I/O Facilities

- All fundamental I/O in Java is based on streams. A stream represents a flow of data with a writer at one end and a reader at the other.

- *InputStream* and *OutputStream* are the lowest level interface for all streams. These are abstract classes that define the basic functionality for reading and writing an unstructured sequence of bytes. All other streams are built on top of these two streams.

- These two streams are sub-classed to provide *DataInputStream* and *DataOutputStream* that let you read or write strings and primitive data types.
- BufferedInputStream, BufferedOutputStream, PrintStream.
- PrintWriter and the equivalent Reader (refer to java.sun.com/docs)

E.g. DataInputStream dis = new DataInputStream(System.in);
//read from std input
String line = dis.readLine();
//This wraps the std input stream in a DataInputStream
//and readLine() reads bytes upto '\n'

To read an int:

```
int i = dis.readInt();
Or
String text = dis.readLine();
int i = Integer.parseint(text);
Or
String text = dis.readLine();
int i = new Integer(text).intValue();
//This creates an Int wrapper and calls its intValue() method
```

The DataOutputStream class provides write methods that correspond to the read methods in DataInputStream.
e.g. writeInt()

Files

Example:

```
//read from stdin and write to a file
DataInputStream dis = new DataInputStream(System.in);
String s = dis.readLine();
File out = new File("/tmp/foo.txt");
if (out.canWrite()) {
        FileOutputStream fos = new FileOutputStream(out);
        PrintStream pfos = new PrintStream(fos);
        pfos.println(s);
}
```

Example

```
//read data from a file and send out the data read via a socket
File file = new F ile(args[0]);
FileInputStream fis = new FileInputStream(file);
DataInputStream data = new DataInputStream(fis);

OutputStream out = sock.getOutputStream();
PrintStream pfos = new PrintStream(out);

while (num < fis.available()) {
                String s = data.readLine();
                num += s.length();
                pfos.println(s);
}
```