***READING M1***

***Conceptual knowledge  
The layers of a network  
Differences between IP, TCP, and UDP layers  
Basic understanding of IP addresses and ports  
Basic understanding of network routing mechanisms***

TCP, UDP, sockets, datagrams, and ports

Table

Description automatically generated

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

* Reliable
* In order
* Makes sure data is received

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

* Unreliable
* May not receive
* May be out of order
* The UDP protocol provides for communication that is not guaranteed between two applications on the network. UDP is not connection-based like TCP. Rather, it sends independent packets of data from one application to another.

**Ports**:

* Data transmitted over the Internet is accompanied by addressing information that identifies the computer and the port for which it is destined.
  + The computer is identified by its 32-bit IP address, which IP uses to deliver data to the right computer on the network.
  + Ports are identified by a 16-bit number, which TCP and UDP use to deliver the data to the right application.
* Definition: The TCP and UDP protocols use ports to map incoming data to a particular process running on a computer.
* Port numbers range from 0 to 65,535 because ports are represented by 16-bit numbers. The port numbers ranging from 0 - 1023 are restricted; they are reserved for use by well-known services such as HTTP and FTP and other system services. These ports are called well-known ports. Your applications should not attempt to bind to them.

**Networking Classes in the JDK**

* Through the classes in java.net, Java programs can use TCP or UDP to communicate over the Internet. The URL, URLConnection, Socket, and ServerSocket classes all use TCP to communicate over the network. The DatagramPacket, DatagramSocket, and MulticastSocket classes are for use with UDP.

***READING M2***

***Conceptual knowledge  
Understanding of the input/output streams   
Understanding of the use of threading model in the construction of input/output streams***

Basic I/O

I/O Streams

A stream is a sequence of data. A program uses an input stream to read data from a source, one item at a time. A program uses an output stream to write data to a destination, one item at time.

I/O in Java is built on streams. Input streams read data; output streams write data.  
OutputStream’s fundamental method is write(int b)

The flush() method breaks the deadlock by forcing the buffered stream to send its data

even if the buffer isn’t yet full.

**Byte Streams**

Programs use *byte streams* to perform input and output of 8-bit bytes. All byte stream classes are descended from [InputStream](https://docs.oracle.com/javase/8/docs/api/java/io/InputStream.html) and [OutputStream](https://docs.oracle.com/javase/8/docs/api/java/io/OutputStream.html).

**Buffered Streams**

Most of the examples we've seen so far use *unbuffered* I/O. This means each read or write request is handled directly by the underlying OS. This can make a program much less efficient, since each such request often triggers disk access, network activity, or some other operation that is relatively expensive.

inputStream = new BufferedReader(new FileReader("xanadu.txt"));

outputStream = new BufferedWriter(new FileWriter("characteroutput.txt"));

There are four buffered stream classes used to wrap unbuffered streams: [BufferedInputStream](https://docs.oracle.com/javase/8/docs/api/java/io/BufferedInputStream.html) and [BufferedOutputStream](https://docs.oracle.com/javase/8/docs/api/java/io/BufferedOutputStream.html) create buffered byte streams, while [BufferedReader](https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html) and [BufferedWriter](https://docs.oracle.com/javase/8/docs/api/java/io/BufferedWriter.html) create buffered character streams.

***Practical knowledge in Java  
How to instantiate different types of input/output streams in Java  
How to use input/output streams in Java to read/write to a file  
How to use the concurrency mechanisms in the construction of reader and writer functionality***

System.in.println()

System.out.println()

inputStream = new BufferedReader(new FileReader(file\_name));

In addition to these two classes, the java.io package provides several raw reader and

writer classes that read characters without directly requiring an underlying input

stream, including:

• FileReader

• FileWriter

• StringReader

• StringWriter

• CharArrayReader

• CharArrayWriter

***READING M3***

***Conceptual knowledge***

***Understanding of sockets using TCP protocol***

TCP provides a reliable, point-to-point communication channel that client-server applications on the Internet use to communicate with each other. To communicate over TCP, a client program and a server program establish a connection to one another. Each program binds a socket to its end of the connection. To communicate, the client and the server each reads from and writes to the socket bound to the connection.

Definition: A socket is one end-point 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.

An endpoint is a combination of an IP address and a port number. Every TCP connection can be uniquely identified by its two endpoints. That way you can have multiple connections between your host and the server.

***Client-server model of socket communication on top of TCP***

 Socket classes are used to represent the connection between a client program and a server program. The java.net package provides two classes--Socket and ServerSocket--that implement the client side of the connection and the server side of the connection, respectively.

Graphical user interface, text, application

Description automatically generated

The java.net package in the Java platform provides a class, Socket, that implements one side of a two-way connection between your Java program and another program on the network. The Socket class sits on top of a platform-dependent implementation, hiding the details of any particular system from your Java program. By using the java.net.Socket class instead of relying on native code, your Java programs can communicate over the network in a platform-independent fashion.

Additionally, java.net includes the ServerSocket class, which implements a socket that servers can use to listen for and accept connections to clients.

***Handling concurrency issues when reading from and reading to TCP sockets***

***Practical knowledge in Java***

***Programming using Java TCP socket libraries***

***Java TCP socket error handling in Java***

***Use of threading to read and write to Java TCP sockets***

***Use of Java's input/output streams to read from and write to TCP sockets***

1. Open a socket.
2. Open an input stream and output stream to the socket.
3. Read from and write to the stream according to the server's protocol.
4. Close the streams.
5. Close the socket.

InetAddress objects can then be used to create sockets, server sockets, and so forth. The primary purpose of the SocketAddress class is to provide a convenient store for

transient socket connection information such as the IP address and port that can be

reused to create new sockets, even after the original socket is disconnected and garbage

collected.

The InetSocketAddress class (which is the only subclass of SocketAddress in the JDK,

and the only subclass I’ve ever encountered) is usually created with a host and a port

(for clients) or just a port (for servers):

**public** InetSocketAddress(InetAddress address, **int** port)

**public** InetSocketAddress(String host, **int** port)

**public** InetSocketAddress(**int** port)

Socket objects have several properties that are accessible through getter methods:

• Remote address

• Remote port

• Local address

• Local port

Here are the getter methods for accessing these properties:

**public** InetAddress getInetAddress()

**public int** getPort()

**public** InetAddress getLocalAddress()

**public int** getLocalPort()

***READING M4***

Some applications that you write to communicate over the network will not require the reliable, point-to-point channel provided by TCP. Rather, your applications might benefit from a mode of communication that delivers independent packages of information whose arrival and order of arrival are not guaranteed.

The UDP protocol provides a mode of network communication whereby applications send packets of data, called datagrams, to one another. A datagram is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed. The DatagramPacket and DatagramSocket classes in the java.net package implement system-independent datagram communication using UDP.

[**What Is a Datagram?**](https://docs.oracle.com/javase/tutorial/networking/datagrams/definition.html)

Definition: A datagram is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed.

The java.net package contains three classes to help you write Java programs that use datagrams to send and receive packets over the network: [DatagramSocket](https://docs.oracle.com/javase/8/docs/api/java/net/DatagramSocket.html), [DatagramPacket](https://docs.oracle.com/javase/8/docs/api/java/net/DatagramPacket.html), and [MulticastSocket](https://docs.oracle.com/javase/8/docs/api/java/net/MulticastSocket.html)An application can send and receive DatagramPackets through a DatagramSocket. In addition, DatagramPackets can be broadcast to multiple recipients all listening to a MulticastSocket.

[**Writing a Datagram Client and Server**](https://docs.oracle.com/javase/tutorial/networking/datagrams/clientServer.html)

The server side is a quote server that listens to its DatagramSocket and sends a quotation to a client whenever the client requests it. The client side is a simple program that simply makes a request of the server.

The client application sends a single datagram packet to the server indicating that the client would like to receive a quote of the moment. The client then waits for the server to send a datagram packet in response.

[**Broadcasting to Multiple Recipients**](https://docs.oracle.com/javase/tutorial/networking/datagrams/broadcasting.html)

This section modifies the quote server so that instead of sending a quotation to a single client upon request, the quote server broadcasts a quote every minute to as many clients as are listening. The client program must be modified accordingly.

**Note:**

Many firewalls and routers are configured not to allow UDP packets. If you have trouble connecting to a service outside your firewall, or if clients have trouble connecting to your service, ask your system administrator if UDP is permitted.