Networking fundamentals, The Networking classes and Interfaces, The InetAddressclass, The Socket Class, The URL class, The URLConnection Class, The HttpURLConnection Class.

Networking Fundamentals

- Java supports both the TCP and UDP protocol families.
- There are 2 types of communication:
  - Connection - oriented communication : **TCP**
  - Connection - less communication : **UDP**
- TCP is used for reliable stream-based I/O across the network
- UDP supports faster, point-to-point datagram-oriented model.
- The ***URL***, ***URLConnection*** , **Socket**, and ***ServerSocket*** classes all use TCP to communicate over the network.
- The ***DatagramPacket***, ***DatagramSocket***, and ***MulticastSocket*** classes are used by UDP to have Connection-less networking.

The Networking classes and interfaces

- Java Provides extensive support for the TCP and UDP protocol families

classes

Class	Description
InetAddress	This class represents an Internet Protocol (IP) address.
Socket	This class implements client sockets (also called just "sockets").
ServerSocket	This class implements server sockets.
URL	Represents a Uniform Resource Locator, a pointer to a "resource" on the World Wide Web.
URLConnection	The abstract class `URLConnection` is the superclass of all classes that represent a communications link between application and URL.
HttpURLConnection	**URLConnection** is a general-purpose class for accessing the attributes of a remote resource.
DatagramPacket	This class represents a datagram packet.
DatagramSocket	This class represents a socket for sending and receiving datagram packets.
MulticastSocket	The multicast datagram socket class is useful for sending and receiving IP multicast packets.

Interfaces

ContentHandlerFactory	CookiePolicy	CookieStore
DatagramSocketImplFactory	FileNameMap	ProtocolFamily
SocketIMPlFactory	SocketOption	SocketOptions
URLStreamHandlerFactory		

InetAddress

- ***InetAddress class* Represents**: An IP address.
- Can convert domain name to IP address
  1.Perform DNS look-up
- Does not have any constructor.
- To create an InetAddress object, use following **factory methods**.
  2. ***getLocalHost()*** *throws UnknownHostException*
  3. ***getByName(String hostName)*** *throws UnknownHostException*
  4. ***getByAddress()*** *throws UnknownHostException*
- All above methods returns object to InetAddress.
- ***getLocalHost***( ) method simply returns the InetAddress object that represents local host.
- ***getByName***( ) method returns an InetAddress for a host name passed to it.
- If these methods are unable to resolve the host name, they throw an UnknownHostException.
- ***getByAddress***( ), which takes an IP address and returns an ***InetAddress*** object.

```
class InetAddressTest  {

public static void main(String args[]) throws UnknownHostException {

        InetAddress ipAddress = InetAddress.getLocalHost();
        System.out.println (ipAddress);

        ipAddress = InetAddress.getByName("osborne.com");
        System.out.println (Address);

        InetAddress ipArray[] = InetAddress.getAllByName("www.nba.com");

        for ( int i=0; i< ipArray.length; i++ )
        System.out.println (ipArray [i]);
        }
}
```

χhe output produced by this program.

- ✓ `default/206.148.209.138`
- ✓ `osborne.com/198.45.24.162`
- ✓ www.nba.com/64.5.96.214
- ✓ www.nba.com/64.5.96.216

About Socket

- A socket identifies an endpoint in a network.
- Socket allows a single computer to serve many different clients at once.
- This is accomplished through the use of a port, which is a numbered socket on a particular machine.
- A server process is said to —listen to a port until a client connects to it.
- A server is allowed to accept multiple clients connected to the same port number.
- Socket communication takes place via a protocol.
  - **a. Internet Protocol**
    - breaks data into small packets and sends them to an address across a network.
    - does not guarantee to deliver said packets to the destination.
  - **b. Transmission control protocol**
    - Establish the session and transmit packets.
    - Reliable transmission of data.
  - **c. User Datagram protocol**
    - support fast, connectionless, unreliable transport of packets.
- TCP/IP reserves the lower 1,024 ports for specific protocols. Many of these will seem familiar to you if you have spent any time surfing the Internet.

- Port number 21 is for FTP; 23 is for Telnet; 25 is for e-mail; 43 is for whois; 79 is for finger, 80 is for HTTP; 119 is for net news—and the list goes on. It is up to each protocol to determine how a client should interact with the port.

Socket class

- TCP/IP sockets are used to implement reliable connections between hosts on the Internet.
- There are two kinds of TCP sockets in Java
  - The ServerSocket class is for servers. It is designed to be a "listener," which waits for clients to connect before doing anything.
  - The Socket class is for clients. It is designed to connect to server sockets and initiate protocol exchanges.
- The creation of a Socket object establishes a connection between the client and server.
- two constructors used to create client sockets:

Constructor	Description
Socket (*String* ***hostName***, *int* ***port***) ***throws*** *UnknownHostException, IOException*	Creates a socket connected to the named host and port.
Socket (*InetAddress* ***ipAddress***, *int* ***port***) *throws IOException*	Creates a Socket using a pre existing InetAddress object and a port.

- **Socket** defines several instance methods.

Methods	**Description**
InetAddress getInetAddress()	Returns the InetAddress associated with the socket object. It returns null if the socket is not connected
Int getPort()	Return the local port to which Socket object is bound. It returns -1 if not bound
InputStream getInputStream () throws IOException	Returns the inputStream associated with the invoking socket
OutputStream getOutputStream()throws IOException	Returns the outputStream associated with the invoking socket
Connect	Allows you to specify a new connection
Boolean isConnected()	Returns true if the socket is connected to a server

ServerSocket class

- The ServerSocket class is used to create servers that listen for client programs on published ports.
- Parameters to constructors are

- the port number on which clients to connect
- optional queue length, tells the system how many client connections it can leave pending before it should simply refuse connections. The default is 50.

- The constructors might throw an IOException under adverse conditions.
- Here are three of its constructors:

Constructor	Description
ServerSocket (*int port*) **throws IOException**	Creates server socket on the specified port with a queue length of 50.
ServerSocket (*int port, int maxQueue*) **throws IOException**	Creates a server socket on the specified port with a maximum queue length of maxQueue.
ServerSocket (*int port, int maxQueue, InetAddress localAddress*) **throws IOException**	Creates a server socket on the specified port with a maximum queue length of maxQueue. On a multihomed host, localAddress specifies the IP address to which this socket binds.

- **ServerSocket** has a method called ***accept***( ), which is a blocking call that will wait for a client to initiate communications and then return with a normal Socket that is then used for communication with the client.
- Support for URL connections is provided in the java.net package by the following classes:
  - URL
  - URLConnection
  - HttpURLConnection

Class URL

- the modern Internet is not about the older protocols such as whois, finger, and FTP. It is about WWW, the World Wide Web.
- The Uniform Resource Locator (URL) uniquely identify on the Internet.
- All URLs share the same basic format, although some variation is allowed.
- A URL specification is based on four components.

A. **The first** is the **protocol** to use:
   a. Separated from the rest of the locator by a colon (:).
   b. Common protocols are HTTP, FTP, gopher, and file.

> ***http:***//www.google.com/
> ***ftp***//…

B. **The second component** is the **host name** or **IP address** of the host to use;
   a. Delimited on left by (//) and on the right by (/) or optionally a colon (:)

*http:**//www.google.com:80**/ index.html*
*http:**//www.google.com/** index.h tml*

C. **The third component**, the **port number (optional)**,
   a. Delimited on left after host name by (:) and on the right by (/).
   b. It defaults to port 80, the predefined HTTP port; thus, ":80" is redundant.

D. **The fourth part** is the **actual file path**
   a. Most HTTP servers will append a file named index.html
   b. java's URL class has several constructors; each can throw a ***MalformedURLException.***

URL (*String urlSpecifier****) throws*** *MalformedURLException*

URL (*String protocolName, String hostName, int port, String path****)***
throws *MalformedURLException*

URL (*String protocolName, String hostName, String path****)***
throws *MalformedURLException*

URL (*URL url Obj , St ring urlSpecifier****) throws*** *MalformedURLException*

Example: creates a URL to osborne's download page and then examine its properties.

```
import java.net.*;
class URLDemo {
  public static void main(String args[]) throws MalformedURLException {

        URL myURL = new URL("http://www.osborne.com/downloads");

        System.out.pri ntln ("Protocol : " + myURL.getProtocol () );
        System.out.pri ntln ("Port: " + myURL.getPort( ) );
        System.out.pri ntln ("Host: " + myURL.getHost( ) );
        System.out.pri ntln ("Fil e: " + myURL.getFile( ) );
        System.out.pri ntln ("Ext:" + myURL.toExternalForm( ) );
 }
}
```

Output

```
Protocol: http
Port: -1
Host: www.osborne
File: /downloads
Ext:http://www.osborne/downloads
```

The URL Connection

- ***URLConnection*** is a general-purpose class for accessing the attributes of a remote resource.
- Once you make a connection to a remote server, you can use URLConnection to inspect the properties of the remote object before actually transporting it locally.
- These attributes are exposed by the HTTP protocol specification and, as such, only make sense for URL objects that are using the HTTP protocol.
- **URLConnection** defines several methods. Here is a sampling:

Methods	Description
int ***getContentLength()***	Returns the size in bytes of the content associated with the resource. If the length is unavailable, –1 is returned.
String ***getContentType()***	Returns the type of content in the resource else Returns null. This is the value of content-type header field.
long ***getDate()***	Returns the time and date of the response.
long ***getExpiration()***	Returns the expiration time and date of the resource. Zero is returned if the expiration date is unavailable.

- ***Example:*** creates a **URLConnection** using the **openConnection**( ) method of a **URL** object and then uses it to examine the document's properties and content:

```
import java.net.*;
import java.io.*;
import java.util.Date;
class UCDemo
{
     public static void main(String args[]) throws Exception {
          URL url = new URL("http://www.internic.net");
          URLConnection hpCon = url.openConnection();

     // get date
     long date = hpCon.getDate();
     if(date == 0)
     System.out.println("No date information.");
     else   System.out.println("Date: " + new Date(date));

     // get content type and expiration date
     System.out.println("Content-Type: " +
     hpCon.getContentType());
     date = hpCon.getExpiration();
     if(date == 0)
     System.out.println("No expiration information.");
     else    System.out.println("Expires: " + new Date(d));

     // get last-modified date
     date = hpCon.getLastModified();
     if(date == 0)
     System.out.println("No last-modified information.");
     else    System.out.println("Last-Modified: " + newDate(d));

     // get content length
```

```
      int len = hpCon.getContentLength();
      if(len == -1)  System.out.println("Content length
      unavailable.");
      else   System.out.println("Content-Length: " + len);

      if(len != 0) {
      System.out.println("=== Content ===");
      InputStream input = hpCon.getInputStream();
      int i = len, ch;

      while (((ch = input.read()) != -1)) {
      System.out.print((char) c);
      }
      input.close();
      } else {
      System.out.println("No content available.");
      }
  }
}
```

Class HttpURLConnection

- Java provides a subclass of URLConnection that provides support for HTTP connections. This class is called HttpURLConnection.
- You obtain an HttpURLConnection in the same way just shown, by calling openConnection( ) on a URL object, but you must cast the result to HttpURLConnection. (Of course, you must make sure that you are actually opening an HTTP connection.)
- Once you have obtained a reference to an HttpURLConnection object, you can use any of the methods inherited from URLConnection.
- You can also use any of the several methods defined by HttpURLConnection. Here is a sampling:

static boolean getFollowRedirects( )	Returns **true** if redirects are automatically followed and **false** otherwise. This feature is on by default.
String getRequestMethod( )	Returns a string representing how URL requests are made. The default is GET. Other options, such as POST, are available.
int getResponseCode( ) throws IOException	Returns the HTTP response code. –1 is returned if no response code can be obtained. An **IOException** is thrown if the connection fails.
String getResponseMessage( ) throws IOException	Returns the response message associated with the response code. Returns null if no message is available. An **IOException** is thrown if the connection fails.
static void setFollowRedirects(boolean *how*)	If *how* is **true**, then redirects are automatically followed. If how is **false**, redirects are not automatically followed. By default, redirects are automatically followed.
void setRequestMethod(String *how*) throws ProtocolException	Sets the method by which HTTP requests are made to that specified by *how*. The default method is GET, but other options, such as POST, are available. If *how* is invalid, a **ProtocolException** is thrown.

- The following program demonstrates HttpURLConnection. It first establishes a connection to www.google.com. Then it displays the request method, the response code, and the response message. Finally, it displays the keys and values in the response header.

```
// Demonstrate HttpURLConnection.
import java.net.*;
import java.io.*;
import java.util.*;
class HttpURLDemo
{
  public static void main(String args[]) throws Exception {
        URL ur l  = new URL("h t t p://www.goog l e.com");
        HttpURLConnection hpCon = (H t t pURLConnec tion) url .openConnection( );

                // Display request method.
                System.out.pri ntl n("Request method is " + hpCon.getRequestMethod());

                // Display response code.
                System.out.pri ntl n("Response code is " + hpCon.getResponseCode());

                // Display response message.
                System.out.pri ntl n("Response Message is " +hpCon.getResponseMessage());

                // Get a list of the header fields and a set of the header keys.
                Map<String, Li st<Stri ng>> hdrMap = hpCon.getHeaderFields();
                Set<Str ing> hdrField = hdrMap.keySet();

                System.out.pri ntln("\nHere is the header:");

                // Display all header keys and values.
                for(String k : hdrF ie l d) {
                System.out.println("Key: " + k + " Value: " + hdrMap.get(k));
}
  }
}
```

```
by www.google.com will vary over time.)
Request method is GET
Response code is 200
Response Message is OK
Here is the header:
Key: Set-Cookie Value:
[PREF=ID=4fbe939441ed966b:TM=1150213711:LM=1150213711:S=Qk81
WCVtvYkJOdh3; expires=Sun, 17-Jan-2038 19:14:07 GMT; path=/;
domain=.google.com]
```

```
Key: null Value: [HTTP/1.1 200 OK]
Key: Date Value: [Tue, 13 Jun 2006 15:48:31 GMT]
Key: Content-Type Value: [text/html]
Key: Server Value: [GWS/2.1]
Key: Transfer-Encoding Value: [chunked]
Key: Cache-Control Value: [private
```