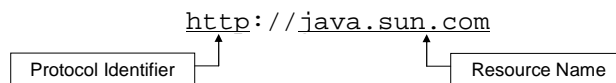


Lecture 9

Network programming

Manipulating URLs

- URL is an acronym for *Uniform Resource Locator* and is a reference (an address) to a resource on the Internet.
- Sample structure of a URL. The resource name part may contain: host name, file name, port number(optional) and reference (optional)



- you can create a URL object in Java from an absolute URL or a relative URL.
- The URL class provides several methods implemented on URL objects. You can get the protocol, host name, port number, and filename from a URL.

Example code

```
import java.net.*;
import java.io.*;
public class ParseURL {
    public static void main(String[] args) throws Exception {
        URL aURL = new URL("http://java.sun.com:80/docs/books/" +
            "tutorial/index.html#DOWNLOADING");
        System.out.println("protocol = " +
            aURL.getProtocol()); System.out.println("host = " +
            aURL.getHost()); System.out.println("filename = " +
            aURL.getFile()); System.out.println("port = " +
            aURL.getPort()); System.out.println("ref = " +
            aURL.getRef());
    }
}
```

Output of the above code:

```
protocol = http
host = java.sun.com
filename = /docs/books/tutorial/index.html
port = 80
ref = DOWNLOADING
```

Connecting with a URL (1)

- `openStream()`: returns a `java.io.InputStream` object, from which you can read easily as reading from an input stream. It may throw an `IOException`

Example code

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

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

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

This prints out the source code for the webpage www.osu.edu

Connecting with a URL (2)

- `openConnection()`: Returns a `URLConnection` object that represents a connection to the remote object referred to by the URL. It may throw an `IOException`

```
try {  
    URL osu = new URL("http://www.osu.edu/");  
    URLConnection osuConnection = osu.openConnection();  
} catch (MalformedURLException e) { // new URL() failed  
    . . .  
} catch (IOException e) {  
    . . .  
}
```

- The `URLConnection` class provides many methods to communicate with the URL, such as reading and writing.

Sockets

- Sometimes you need a low-level network communication, such as a client-server application
- The TCP protocol provides a reliable point-to-point communication channel via the sockets.
- A socket is an endpoint for reliable communication between two machines. To connect with each other, each of the client and the server binds a socket to its end for reading and writing.
- The `java.net` package provides two classes — `Socket` and `ServerSocket` — to implement the client and the server, respectively.

Establishing a simple server

- Five steps:

- 1). Create a `ServerSocket` object

```
ServerSocket server=new ServerSocket(port,queueLength);
```

- 2). The server listens indefinitely (or blocks) for an attempt by a client to connect

```
Socket connection = server.accept();
```

- 3). Get the `OutputStream` and `InputStream` objects that enable the server to communicate with the client by sending and receiving bytes

```
InputStream input = connection.getInputStream();  
OutputStream output = connection.getOutputStream();
```

- * You can get a stream of other data types from the `InputStream` and `OutputStream`

- 4). *Processing phase*: the server and the client communicate via the `InputStream` and the `OutputStream` objects

- 5). After the communication completes, the server closes the connection by invoking `close()` on the `Socket` and the corresponding streams

Establishing a simple client

- Four steps:

- 1). Create a `Socket` object

```
Socket connection = new Socket (serverAddress, port);
```

- 2). Get the `OutputStream` and `InputStream` of the `Socket`. The server and the client must send and receive the data in the same format

- 3). *Processing phase*: the server and the client communicate via the `InputStream` and the `OutputStream` objects

- 4). After the communication completes, the client closes the connection.

A simple server/client pair example

The server side

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

class Server {
    public static void main(String args[]) {
        String data = "Let's test if we can connect...";
        try {
            ServerSocket server_socket = new ServerSocket(1234);
            System.out.println("I've started, dear clients...");

            Socket socket = server_socket.accept();

            System.out.print("Server has connected!\n");
            PrintWriter outToClient = new PrintWriter(socket.getOutputStream(),
true);
            System.out.print("Sending string: '" + data + "'\n");
            outToClient.print(data);

            outToClient.close();
            socket.close();
            server_socket.close();
        }
        catch(Exception e) {
            System.out.print("Whoops! It didn't work!\n");
        }
    }
}
```

(this example is got from <http://www.cise.ufl.edu/~amyles/tcpchat/>)

A simple server/client pair example (cont.)

The client side

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

class Client {
    public static void main(String args[]) {
        try {
            Socket socket = new Socket("localhost", 1234);

            BufferedReader inFromServer = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

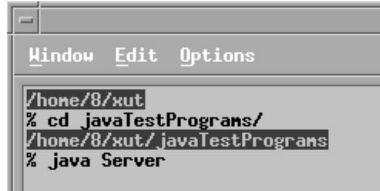
            System.out.print("Received string: `");
            while (inFromServer.ready())
                System.out.println(in.readLine()); //Read one line and output it

            inFromServer.close();
        }
        catch(Exception e) {
            System.out.print("Whoops! It didn't work!\n");
        }
    }
}
```

(this example is got from <http://www.cise.ufl.edu/~amyles/tcpchat/>)

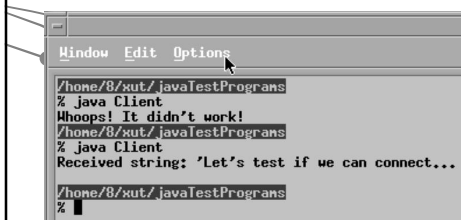
A simple server/client pair example (cont.)

- What happens on the screen if you run the code?
 - First run `Server.java`

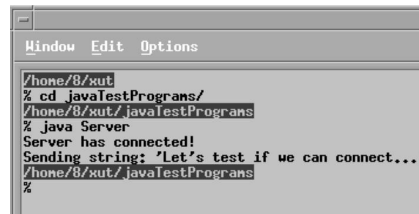


```
Window Edit Options
/home/8/xut
% cd javaTestPrograms/
/home/8/xut/javaTestPrograms
% java Server
```

- Then run `Client.java`



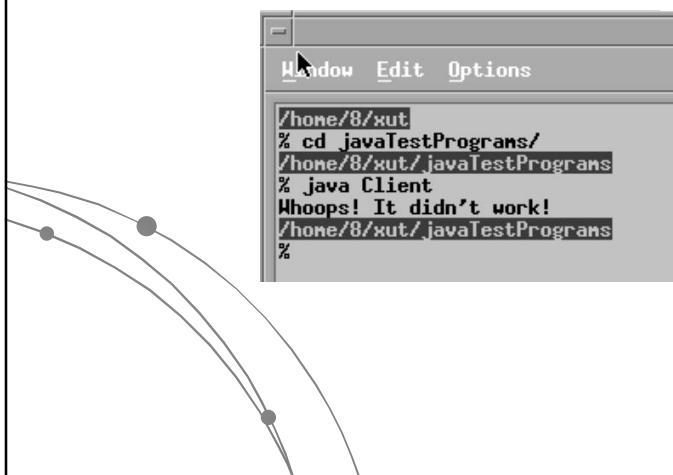
```
Window Edit Options
/home/8/xut/javaTestPrograms
% java Client
Whoops! It didn't work!
/home/8/xut/javaTestPrograms
% java Client
Received string: 'Let's test if we can connect...'
/home/8/xut/javaTestPrograms
% █
```



```
Window Edit Options
/home/8/xut
% cd javaTestPrograms/
/home/8/xut/javaTestPrograms
% java Server
Server has connected!
Sending string: 'Let's test if we can connect...'
/home/8/xut/javaTestPrograms
%
```

A simple server/client pair example (cont.)

- If you run `Client.java` without running `Server.java`



```
Window Edit Options
/home/8/xut
% cd javaTestPrograms/
/home/8/xut/javaTestPrograms
% java Client
Whoops! It didn't work!
/home/8/xut/javaTestPrograms
% █
```

Datagrams

- The UDP protocol provides a mode of network communication whereby *datagrams* are sent over the network. `DatagramSockets` are used for the connection.
- A datagram's arrival, arrival time and order of arrival is not guaranteed. It's used whenever an information needs to be broadcast periodically

UDP example

```
import java.net.*;

class GetDate {
    final static int PORT_DAYTIME = 13;
    public static void main(String[] args) throws Exception {
        DatagramSocket dgsocket = new DatagramSocket();
        InetAddress destination =
            InetAddress.getByName("news.cis.ohio-state.edu");
        DatagramPacket datagram;
        byte[] msg = new byte[256];

        datagram = new DatagramPacket(msg, msg.length, destination,
            PORT_DAYTIME);
        dgsocket.send(datagram);

        datagram = new DatagramPacket(msg, msg.length);
        dgsocket.receive(datagram);

        String received = new String(datagram.getData());
        System.out.println("Time of machine:" + received);
        dgsocket.close();
    }
}
```

Sample execution:

```
alpha> java GetDate
Time of machine: Tue May 24 00:16:42 2005
```

Supplemental reading

- *Custom networking*
<http://java.sun.com/docs/books/tutorial/networking/index.html>
- *Java™ Programming Language Basics, Socket Communications*
<http://developer.java.sun.com/developer/onlineTraining/Programming/BasicJava2/socket.html>

