

Network Programming

Lecture 10

Overview

- Package: java.net
- Many classes for common network tasks
 - Working with URLs
 - Opening/closing sockets (communication channels)
- From a socket, one obtains the corresponding *streams*
 - One stream for input, one for output
 - Reading/writing to a socket is **just like reading/writing to a file**

Working with URLs

- URL: "Uniform Resource Locator"
 - A string representing a logical address in the network
- Basic structure of a well-formed URL
 - protocol (http, ftp, news, file,...)
 - hostname (www.cse.ohio-state.edu)
 - port number (80, 8080, 9802,...)
 - resource (docs/books/tutorial/index.html)
- Example: `http://java.sun.com:80/docs/index.html`
- A URL object can be created from either an absolute URL or a relative URL

```
URL u1 = new URL("http://java.sun.com/");
URL u2 = new URL(u1, "docs/index.html");
```
- The URL class provides methods for
 - Getting the protocol, host name, port number, filename
 - Opening a connection

Example

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

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

## Connecting to a URL

- Method `openStream()`
  - Returns a `java.io.InputStream` object, from which you can read
  - Use stream **just like for file IO** (eg 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();
 }
}
```
- Output
  - Prints out the source code for the webpage `www.osu.edu`

## Connecting to a URL (2)

- Method `openConnection()`
  - Returns a `URLConnection` object that represents a connection to the remote object referred to by the URL
  - May throw an `IOException`
- Example code

```
try {
 URL osu = new URL("http://www.osu.edu/");
 URLConnection osuCon = 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

- A communication channel between different machines on the network
  - Bidirectional stream of bytes
  - Sender writes to the stream (an output stream)
  - Receiver reads from the stream (an input stream)
- Two kinds of sockets
  - connection-oriented
  - connectionless
- TCP protocol for connection-oriented sockets
  - reliable, in order, point-to-point
- Two important classes:
  - ServerSocket – used by server
  - Socket – used by both server and client

## Skeleton of Server Side

- Create a ServerSocket object

```
ServerSocket server = new ServerSocket(port, queueLength);
```
- The server listens indefinitely (blocks) for an attempt by a client to connect

```
Socket connection = server.accept();
```
- 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
- Processing phase: the server and the client communicate via the InputStream and the OutputStream objects
- After the communication completes, the server closes the socket and the corresponding streams

```
input.close();
output.close();
connection.close();
server.close();
```

## Example Code

```
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 toClient = new PrintWriter(socket.getOutputStream(),
 true);

 System.out.print("Sending string: '" + data + "'\n");
 toClient.print(data);

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

## Skeleton of Client Side

- Create a Socket object

```
Socket connection = new Socket (serverAddr, port);
```
- Get the OutputStream and InputStream objects that enable the client to communicate with the server by sending and receiving bytes

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

  - The server and the client must send and receive the data in the same format
- Processing phase: the server and the client communicate via the InputStream and OutputStream objects
- After the communication completes, the client closes the socket and the corresponding streams

```
input.close();
output.close();
connection.close();
```

## Example Code

```
class Client {
 public static void main(String args[]) {
 try {
 Socket socket = new Socket("localhost", 1234);
 BufferedReader fromServer = new BufferedReader(
 new InputStreamReader(socket.getInputStream()));

 System.out.print("Received string: ");
 while (fromServer.ready())
 System.out.println(fromServer.readLine());
 //Read 1 line and print it

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

## More Examples

- A client attempting to make a connection
  - See PortProbe.java

```
xi% java PortProbe www.cse.ohio-state.edu
```
- A client making a connection and receiving data from the socket
  - See DayTime.java

```
xi% java DayTime www.cse.ohio-state.edu
```
- A client-server pair
  - See ReverseServer.java ReverseClient.java

```
nu% java ReverseServer
xi% java ReverseClient nu.cse.ohio-state.edu
```
  - Notice:
    - nu and xi are different machines
    - Agreed-upon port number used by both (9876)
  - Try pointing a web browser to <http://nu.cse.ohio-state.edu:9876> (while ReverseServer is running)

## More Examples

Computer Science and Engineering • The Ohio State University

- Server is singly-threaded
  - Same thread that accepts connection also handles client request/response
  - See ReverseClientSlow.java
  - `nu% java ReverseServer`
  - `xi% java ReverseClientSlow nu`
  - Try this:
    - Do not enter a string when prompted
    - Instead, start another client
    - `xi% java ReverseClientSlow nu`
    - The second client appears to connect, but receives no reply to its string
    - For a multithreaded solution that avoids this locking, see ReverseServerMT.java

## Datagrams

Computer Science and Engineering • The Ohio State University

- UDP protocol for connectionless communication
  - Reliability? No guarantee of arrival
  - Ordering? Arbitrary
  - Can be multicast
- Utility
  - Non-critical information
  - Frequent (eg periodic) broadcast
- Important classes
  - DatagramSocket
  - DatagramPacket

## Example

Computer Science and Engineering • The Ohio State University

```
class GetData {
 public static void main(String[] args) throws Exception {
 DatagramSocket dgssocket = new DatagramSocket();
 InetAddress dest = InetAddress.getByName(
 "www.cse.ohio-state.edu");
 DatagramPacket datagram;
 byte[] msg = new byte[256];

 datagram = new DatagramPacket(msg, msg.length, dest, 13);
 dgssocket.send(datagram);

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

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

- Sample execution:  
`nu% java GetData`  
Time of machine: Tue May 24 00:16:42 2005

## Supplemental Reading

Computer Science and Engineering • The Ohio State University

- Custom networking trail
  - <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>