ITEC 1630
Week 11: Networking

Yves Lespérance

Readings: Horstmann Ch. 24

Networks & Distributed Systems

- Computers are usually hooked up to a local area network (LAN) and/or the Internet
- Many *distributed applications* run on networked computers
- Computers on a network use *protocols* to communicate, e.g. IP, AppleTalk

TCP/IP

- *Internet Protocol (IP)* is used to deliver data across the Internet
- In IP, data is broken up into smaller *packets* that may be sent along different routes
- *Transmission Control Protocol (TCP)* runs on top of IP and ensures *reliable* transmission

IP addresses

- IP uses internet addresses, e.g. `64.233.167.147`
- Domain names are more convenient, e.g. `www.google.ca`
- Domain Naming Service (DNS) translates between the two
- IP also uses port numbers (short), e.g. 80 for web servers
- Telnet can be used to open a connection and display data sent & received

Application Level Protocols

- Application level protocols run on top of TCP/IP and support communication for specific applications
- E.g. Post Office Protocol (POP) to retrieve email
- E.g. Simple Mail Transfer Protocol (SMTP) to send email

HTTP

- Hypertext Transfer Protocol (HTTP) is used by web browsers and servers
- HTTP uses Uniform Resource Locators (URL) to identify web server hosts and files on them, e.g. `http://www.cse.yorku.ca/~lesperan/courses/ITEC1630/index.html`

HTTP Commands

- `GET` *`file`* `HTTP/1.0` retrieves *`file`*
- `PUT` stores item on server
- `DELETE` removes item from server
- `POST` sends input to server side command and gets results
- `HEAD`, `OPTIONS`, `TRACE`, etc.

Clients & Servers

- A *client* is a program that connects to a server program over the net and uses TCP/IP to communicate
- A *server* is a program that accepts connections from clients
- Each uses a *socket* to send and receive data

Using Sockets in a Client

- To create a socket: `Socket s = new Socket(`*`hostname, port`*`);`
- To obtain its input and output streams:
  ```
  InputStream ins = s.getInputStream();
  OutputStream ous = s.getOutputStream();
  ```
- To close connection: `s.close();`

Sending & Receiving Text

- To read from socket:
  ```
  Scanner in = new Scanner(ins);
  String input = in.nextLine();
  ```
- To send a command through socket:
  ```
  PrintWriter out = new PrintWriter(ous);
  out.print(command);
  out.flush();
  ```
- E.g. WebGet, BankClient

Designing a Server

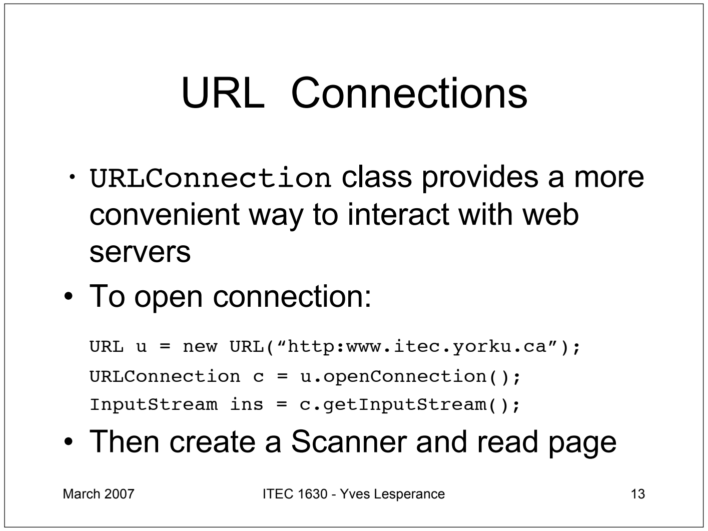
- Decide on a protocol for clients to interact with the server
- To listen for incoming connections, create a server socket: `ServerSocket serv = new ServerSocket(`*`port`*`);`
- To wait for connection and get socket: `Socket s = serv.accept();`
- Then obtain I/O streams and send/receive data as seen before

Handling Clients

- Usually, want to spawn new thread to handle each client who connects, e.g.

```
while(true)
{  Socket s = serv.accept();
   Runnable run = new HandleClient(s);
   Thread t = new Thread(run);
    t.start();
}
```

- When done, kill server
- E.g. BankServer

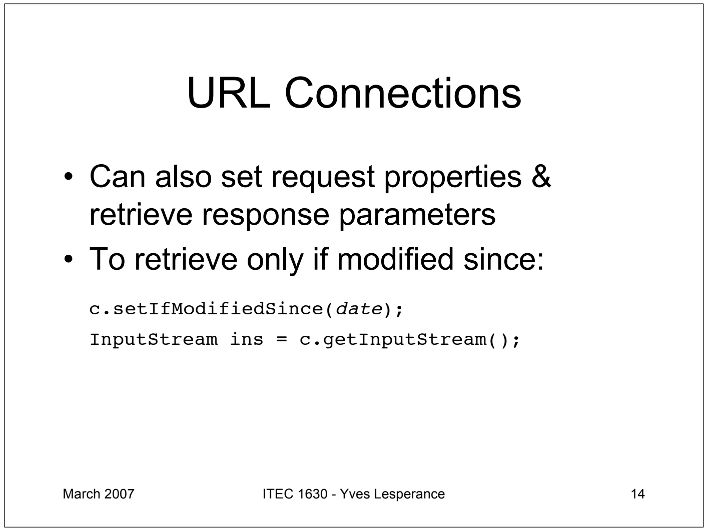


URL Connections

- `URLConnection` class provides a more convenient way to interact with web servers
- To open connection:

```
URL u = new URL(“http:www.itec.yorku.ca”);
URLConnection c = u.openConnection();
InputStream ins = c.getInputStream();
```

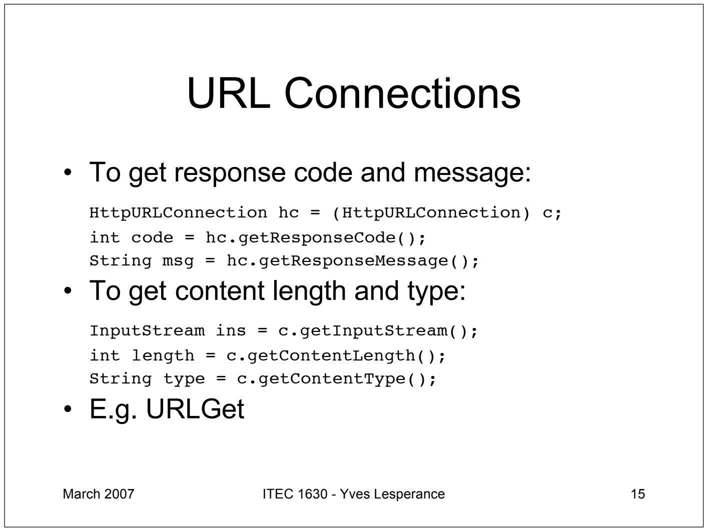
- Then create a Scanner and read page



URL Connections

- Can also set request properties & retrieve response parameters
- To retrieve only if modified since:

```
c.setIfModifiedSince(date);
InputStream ins = c.getInputStream();
```



URL Connections

- To get response code and message:

```
HttpURLConnection hc = (HttpURLConnection) c;
int code = hc.getResponseCode();
String msg = hc.getResponseMessage();
```

- To get content length and type:

```
InputStream ins = c.getInputStream();
int length = c.getContentLength();
String type = c.getContentType();
```

- E.g. URLGet