CSE 1710

Lecture 19

Net-Centric Programming, Part I

Learning Outcomes

- understand net-centric functionality in terms of client and server roles
- see how the Internet Protocol Suite is an example of layered abstraction
- distinguish between the WWW and the Internet
- Understand what the `URL` class encapsulates
- Understand what the `URLConnection` encapsulates
- programmatically get static content from a URL

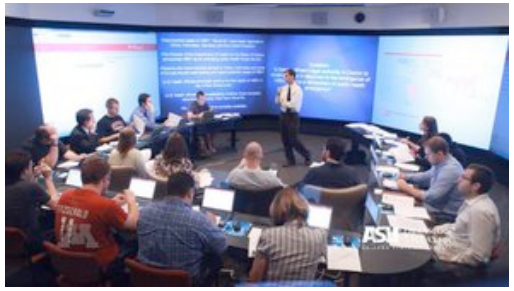
Learning Outcomes

- Understand and describe the basics of the HTTP protocol
- Use the `URL` and `URLConnection` classes to:
  - to instantiate useful objects
  - to retrieve content from web servers
    - Use string processing to manipulate query strings
- Understand the concept of a class hierarchy
  - run-time checking using instanceof
- Use the `HttpURLConnection` class to examine the request and response messages

Thinking about **Systems**

- try to imagine contexts in which humans are interacting with a set of interrelated hardware and software components, in order to support some sort of large scale activity
- sample categories:
  - operations control
  - business management
  - decision making
- who can think of specific examples?

Decision Theatres



Arizona State University – Pandemic Planning Exercise

Command Centers

PACCAR truck parts
supply chain logistics

Real-time News Reporting

CNN Election Results Reporting

Data Sources and **Data Sinks**

- within a context in which there are interrelated components (humans, sw, hw), we can identify data sources and data sinks
- data **source**: a component that produces data
- data **sink**: a component that is capable of receiving data

- Discussion: what have we dealt with so far that has an essence of a data source and data sink?

Review: what does it mean for a method *to block*?

- when a method is invoked, each statement in the body of the method is invoked in sequence
  - somewhere in the body of the method, a statement is waiting for "something" to happen
  - until this "something" happens, the method **blocks**
- the complete invocation of the method depends on some outside event, which may or may not happen in a timely fashion

Examples L20App1

`readLine()` from `Scanner`

`getResponseCode()` in `HttpURLConnection`

A crash course in the stock market

A **public company** is a company that offers its stock/shares for sale to the general public, typically through **a stock exchange**.

A public company is represented by a two- or three-character symbol e.g., "RY" for "Royal Bank of Canada"

At any given point in time, a **share** has a **selling price**. The price **fluctuates** second by second

An investor makes money by "buying low, selling high"

The stock market for this course

(We will use an adapted, simplified stock market for this module)

"Abstract Stock Exchange (ASE)"

http://www.cse.yorku.ca/~roumani/jba/ase/

Provides info on three types of stocks:

- real stocks listed on the TSE, such as Royal Bank (RY)
- made-up ***dynamic*** stocks, such as **.AA, .AB, … , .ZY, .ZZ** (all 26x26 variants)
  - the prices of dynamic stocks fluctuate
- made-up **non-dynamic** stocks, such as HR.A, …, HR.Z (26 variants)

Query Strings

Let's have a look at http://www.cse.yorku.ca/~roumani/jba/ase/

This html code contains a **form element**.

The form gathers together elements into **a meaningful whole**.

The form elements can consist of any number of input elements: text fields, radio buttons, checkboxes. In addition, the form elements typically consists of a "submit" button.

what happens when we press the form's submit button?

Query Strings

When we invoke the form's submit button, two things happen.

1) the following is composed:

```
"http://www.cse.yorku.ca/~roumani/jba/ase/se.cgi"
+ "?" + "hrss" + "=" + ".NN"
```

This evaluates to the following URL:

```
http://www.cse.yorku.ca/~roumani/jba/ase/se.cgi?hrss=.NN
```

2) The **browser** sends a "GET" request with the above-named URL

Now the **server** needs to do something in response…

The Big(ger) Picture

- **Hyptertext Transfer Protocol** (HTTP) is the protocol used to access services concerning remote html files
- It is an *application layer* protocol
- **HTTP** is just one application layer protocol; there are **many** others:
  - others include: ftp, smtp, ssh
- The **application layer protocol** is part of the Internet protocol suite (aka TCP/IP)

The Internet Protocol Suite (TCP/IP)

- Physical layer
- Data link layer
- Network layer
- Transport Layer
- Application Layer

- each layer has its own specific task to perform
- each task has its own set of issues, its own specific data unit
- the suite is a beautiful example of **layered abstraction**
  - the use of several different layers is a strategy to **confront complexity**
  - each layer encapsulates details within it
  - each layer appears as a service to the layer above it

The Internet Protocol Suite (TCP/IP)

- imagine the task – "come up with a scheme to deal with all known and future data communications over the Internet"
- to illustrate the complexity of this, let's look at each of the layers in summary…

The Internet Protocol Suite (TCP/IP)

– Physical layer
- layer deals with: data bits
- task: how to encode 0 and 1 as an analog signal, how to transmit that one bit of data from a computer's Network Interface Card (NIC) to the transmission medium (e.g., copper wire, the air, etc)
- protocols include: Ethernet, WiFi (aka IEEE 802.11), FireWire

The Internet Protocol Suite (TCP/IP)

– Data link layer
- layer deals with: frames
- task: transmit one frame from one node to another on a LAN
- protocols include: Ethernet, WiFi (aka IEEE 802.11), others (not FireWire)

The Internet Protocol Suite (TCP/IP)

– Network layer
- layer deals with: datagram (aka packet)
- task: transmit packets from one node to another on a LAN
- protocols include: IP, others

The Internet Protocol Suite (TCP/IP)

– Transport Layer
- layer deals with: segments (to/from PORT numbers)
- task: transmit messages (segment) from a process running on a node in one LAN to one running on a node in another LAN
- protocols include: TCP, others

The Internet Protocol Suite (TCP/IP)

- Application Layer
  - layer deals with: messages
  - task: provide services to user (in the form of message sending/receiving)
  - protocols include: HTTP, DNS, FTP, SSH, TELNET, SMTP, SIP, many others

Is the WWW the same as the Internet?

Why is the WWW ≠ The Internet?

- The Web **is just a portion** of the Internet
  - It is the subset of the Internet Protocol Suite that is concerned with HTML pages
  - There is more to the Internet than web pages
    - bulk of Internet is used for peer-to-peer file sharing, not for client-server html sending/receiving
- The Internet **predates** the Web
  - prior to TCP/IP (1970's), ARPANET was created (1960's)
  - The Web went "live" on Aug 6, 1991
    - Web proposed two years earlier (by Tim Berners-Lee, who was working at CERN at the time)

The DNS Application Layer Service

- is a **naming system** that maps names to IP addresses
  - for example, `130.63.92.30` is mapped to `cse.yorku.ca`
- why do we need/want this?
  - IP addresses are numeric, not easy for people and applications to use
    - difficult to remember, difficult to associate with meaning
  - IP addresses can be reassigned or otherwise change

What is a Uniform Resource Locator (URL)?

- A URL represents a reference to an object (html file) that is remote (lives on the web-server "cse.yorku.ca")
  - e.g., `http://cse.yorku.ca:80/course/1710/index.html`

- The reference has the following components:
  - the protocol: **http**
  - a path: **the location of the remote object on the host machine**
    - the remote object in this case is an **html file**
    - the information needs to "live" somewhere on the host machine
  - (optionally) a port: **default for http is 80**
    - used by the Transport Layer

More about a URL

- A URL indicates the **location of information on the host machine**
- This information can be static or dynamic
  - static information:
    - already composed, formatted as html, and stored in a file
  - dynamic information:
    - gets composed on-the-fly, gets formatted as html
    - it exists only as a run-time entity

Another Example

- `ftp://ctan.org`

  - the protocol: **ftp**
  - a path: **the location of the remote object**
    - still represents a reference to a remote object
    - the object in this case is something other than html-formatted text
    - the remote object here is a directory listing
  - (optionally) a port: default port number for ftp is 21
    - used by the Transport Layer

What services does the `URL` class provide?

- encapsulates:
  - the protocol that is being used by the URL
  - all the detail about the Internet Protocol Suite that is needed to actually obtain services according to that particular protocol
  - All protocols provides a means to establish a connection to the remote object
  - The connection to the remote object *is abstracted away* from the URL itself

The `URL` class

- The class provides a constructor that accepts a string representation of the URL
- The class `URL` is found in the `java.net` package

```
URL url = new URL("http://www.cse.yorku.ca/course/1710/index.html");
```


What services does the `URLConnection` class provide?

- encapsulates:
  - the **connection** to a remote object using the protocol specified by the URL
  - for instance, it allows the client to **open a connection**, to **request the remote object**, and the to receive it using the specified protocol
- A `URL` object **provides a service** that provides a `URLConnection` object
- You don't need to construct this object

What services does the `HttpURLConnection` class provide?

- encapsulates:
  - the **connection** to a remote object using the http protocol
  - for instance, it allows the client to **open a connection**, to **request the remote object**, and the to receive it

An Example

L19_App1

L19_App2