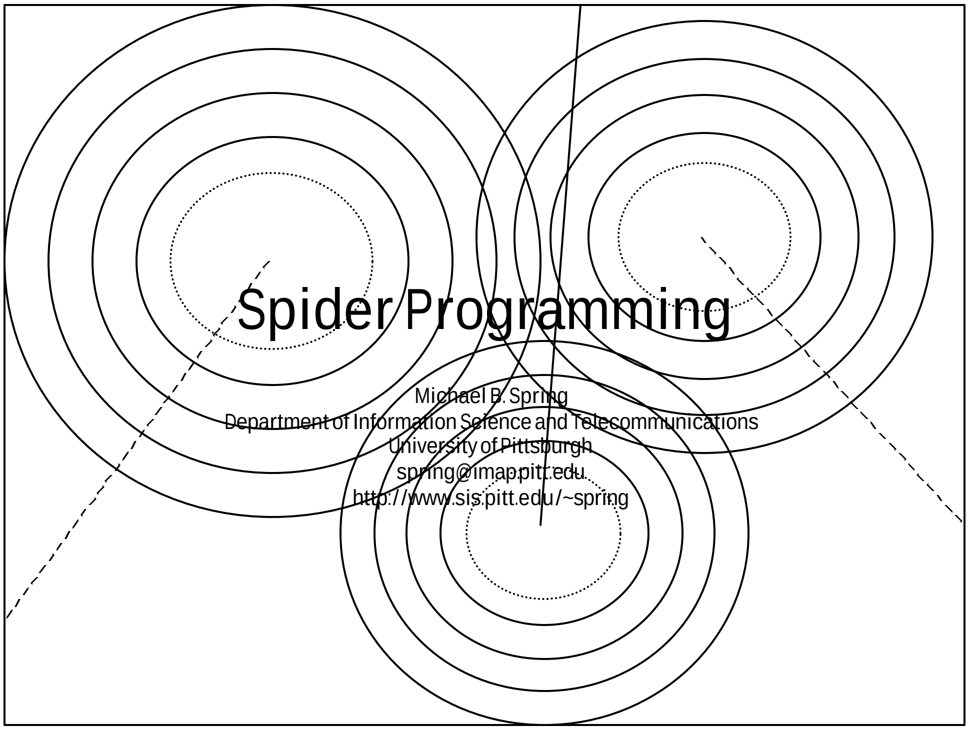
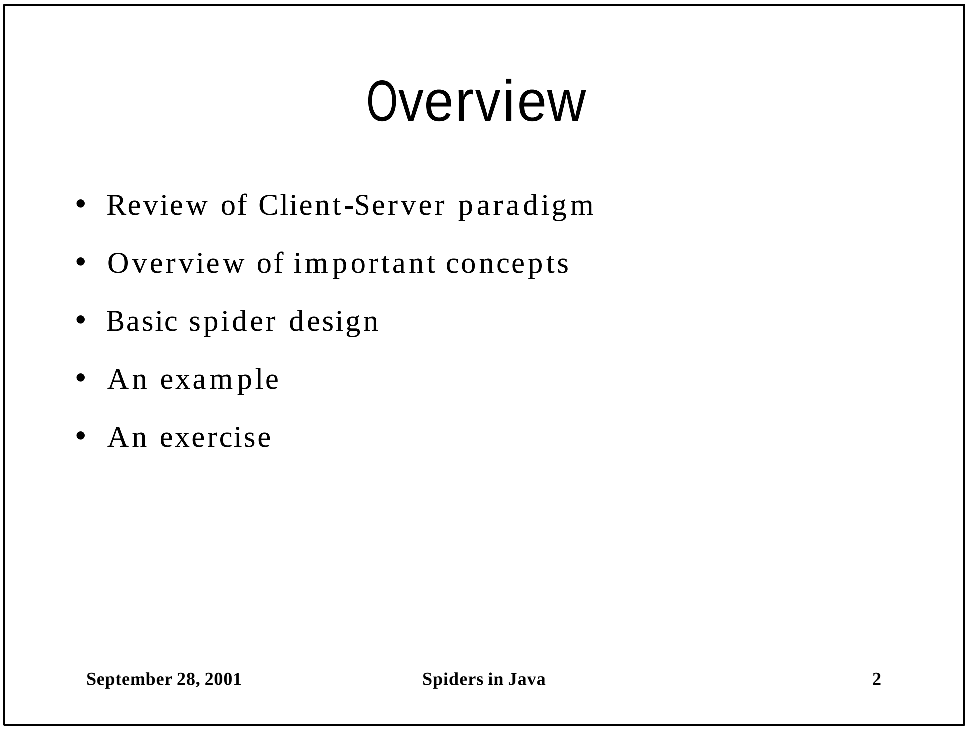


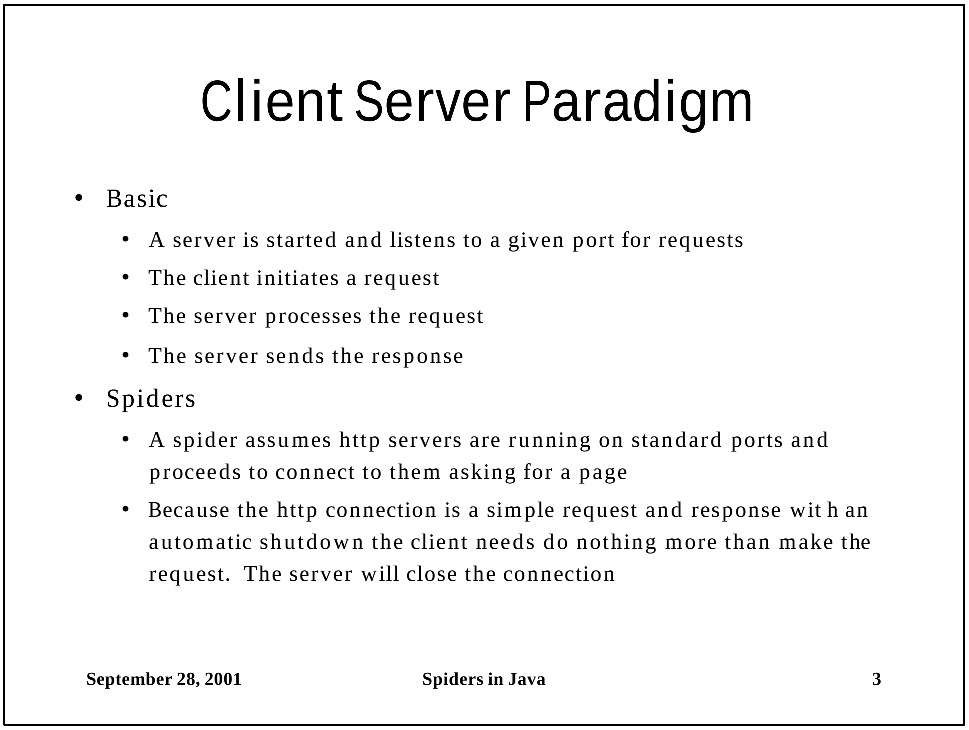
Spider Programming

Michael B. Spring
Department of Information Science and Telecommunications
University of Pittsburgh
spring@imap.pitt.edu
http://www.sis.pitt.edu/~spring



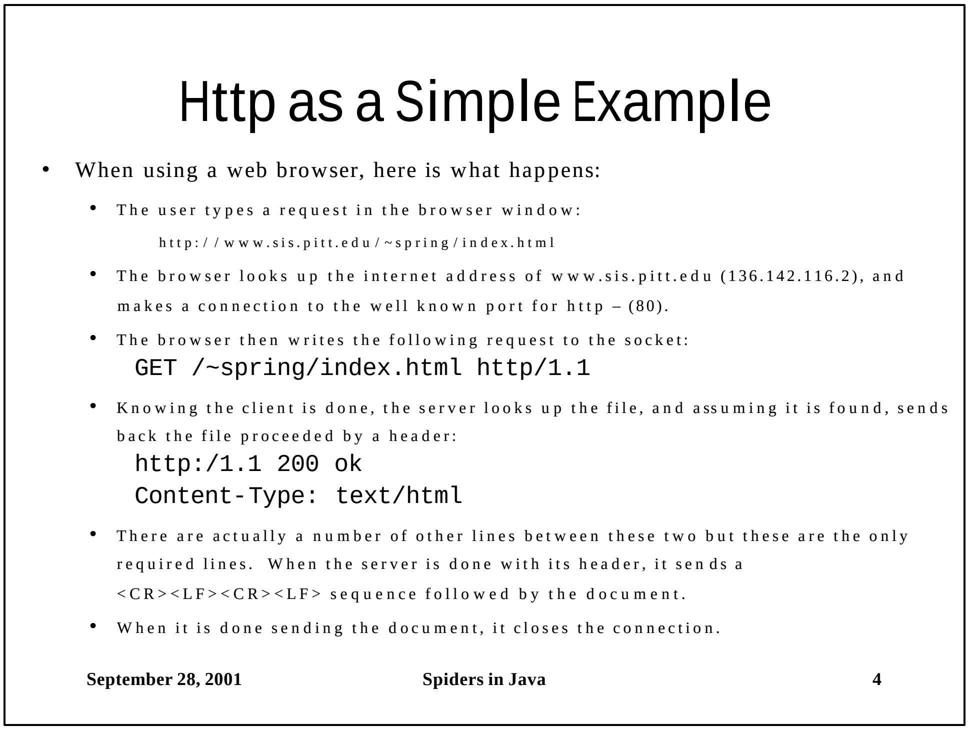
Overview

- Review of Client-Server paradigm
- Overview of important concepts
- Basic spider design
- An example
- An exercise



Client Server Paradigm

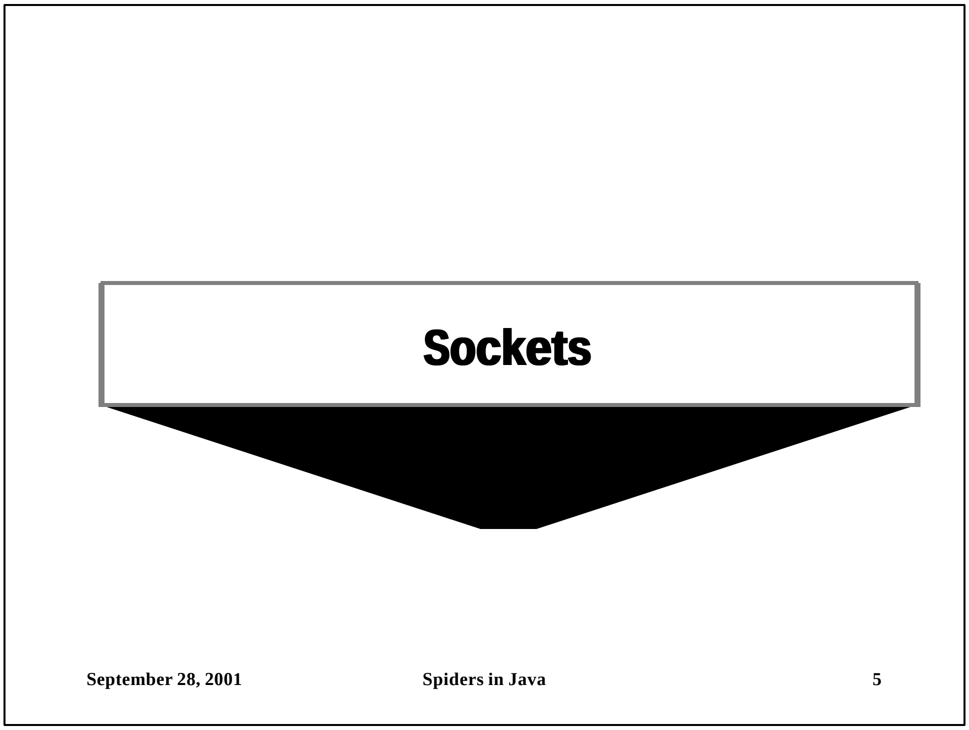
- Basic
  - A server is started and listens to a given port for requests
  - The client initiates a request
  - The server processes the request
  - The server sends the response
- Spiders
  - A spider assumes http servers are running on standard ports and proceeds to connect to them asking for a page
  - Because the http connection is a simple request and response with an automatic shutdown the client needs do nothing more than make the request. The server will close the connection



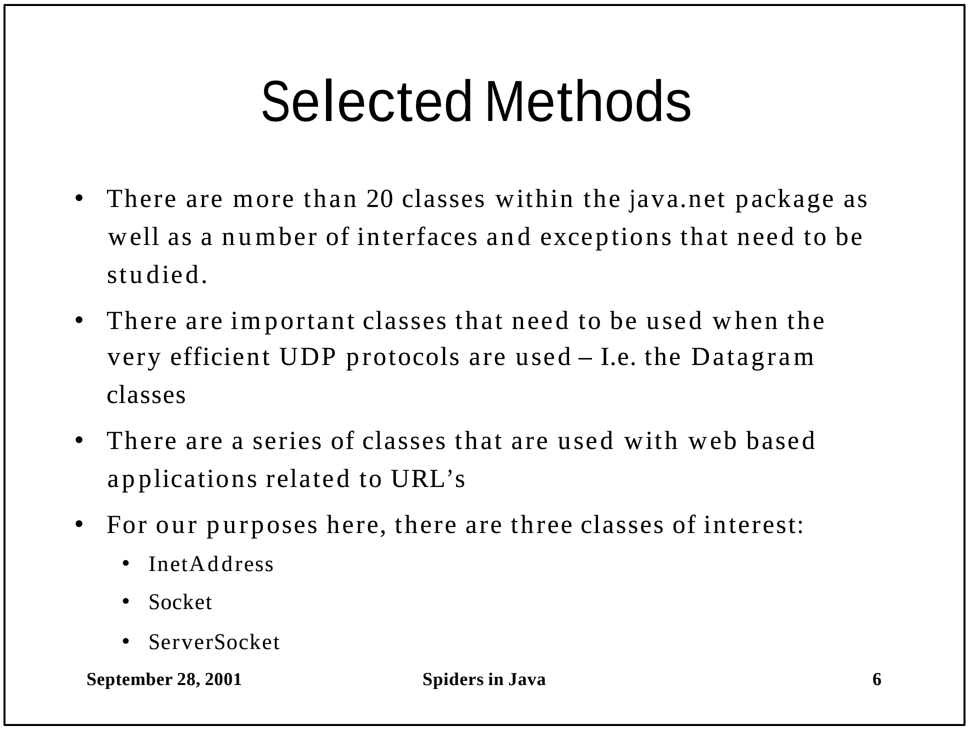
Http as a Simple Example

- When using a web browser, here is what happens:
  - The user types a request in the browser window:

    http://www.sis.pitt.edu/~spring/index.html
  - The browser looks up the internet address of www.sis.pitt.edu (136.142.116.2), and makes a connection to the well known port for http – (80).
  - The browser then writes the following request to the socket:
    ```
    GET /~spring/index.html http/1.1
    ```
  - Knowing the client is done, the server looks up the file, and assuming it is found, sends back the file proceeded by a header:
    ```
    http:/1.1 200 ok
    Content-Type: text/html
    ```
  - There are actually a number of other lines between these two but these are the only required lines. When the server is done with its header, it sends a <CR><LF><CR><LF> sequence followed by the document.
  - When it is done sending the document, it closes the connection.



Sockets



Selected Methods

- There are more than 20 classes within the java.net package as well as a number of interfaces and exceptions that need to be studied.
- There are important classes that need to be used when the very efficient UDP protocols are used – I.e. the Datagram classes
- There are a series of classes that are used with web based applications related to URL’s
- For our purposes here, there are three classes of interest:
  - InetAddress
  - Socket
  - ServerSocket

InetAddress Class

- Socket programming anticipated numerous schemes for addressing machines on networks. Most implementations still allow for this, but in reality, there is only one address type used – internet addresses.
- An internet address is a binary identifier that is four bytes long. Humans have trouble with this long a string of ones and zeros, so two alternate forms are also used:
  - Dotted decimal notations such as the STRING 136.142.116.26
  - Domain names such as the STRING cport.sis.pitt.edu
- InetAddress class is a final class with methods that provide for conversion:
  - InetAddress a = InetAddress.getByName(String)
- There are also methods to convert an InetAddress to the dotted decimal notation (getHostAddress) and domain name (getHostName)

Socket Class

- The Socket Class has a large number of constructors and methods. The most used form would be:
  - Socket S= new Socket(InetAddress a, int port);
- This establishes a connection to the process listening to Port port on the machine at address a.
- The Socket class has a number of utility methods to set the characteristics of the channel and to query attributes of the connection.
- Three methods are essential to developing clients and servers:
  - getOutputStream() which gets a stream to write to
  - getInputStream() wheich gets a stream to read from
  - close() which closes the socket connection.

Socket In Detail

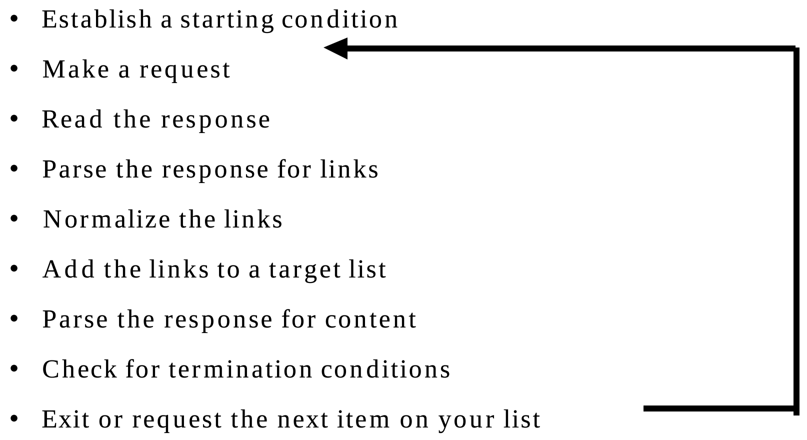
- Be sure that all exceptions are handled appropriately

```
InetAddress Host;
try {// Create a Socket to make connection
  Host = InetAddress.getByName(“www.pitt.edu“);
  S = new Socket(Host , 80 );
  }
catch (UnknownHostException eh)
  {System.out.println( "Host not found" );
  }
catch ( IOException es )
  {System.out.println("Can't create socket");
   es.printStackTrace();
  }
```

Skeleton of a Client

```
// convert dotted decimal string to an address
InetAddress Host = InetAddress.getByName("127.0.0.1");
//open a connection to the host on port 32638
S = new Socket(Host , 32638 );
// get the raw input and output streams as object streams
// letting Java do encoding
Sout = new ObjectOutputStream( S.getOutputStream() );
Sin = new ObjectInputStream( S.getInputStream() );
// write
Sout.writeObject( “GET filename http/1.1”);
Sout.flush();
// read
response = (String) Sin.readObject();
// read response and process it
Sout.close();
Sin.close();
S.close();
```

Basic Spider Paradigm



- Establish a starting condition
- Make a request
- Read the response
- Parse the response for links
- Normalize the links
- Add the links to a target list
- Parse the response for content
- Check for termination conditions
- Exit or request the next item on your list

Make a Request

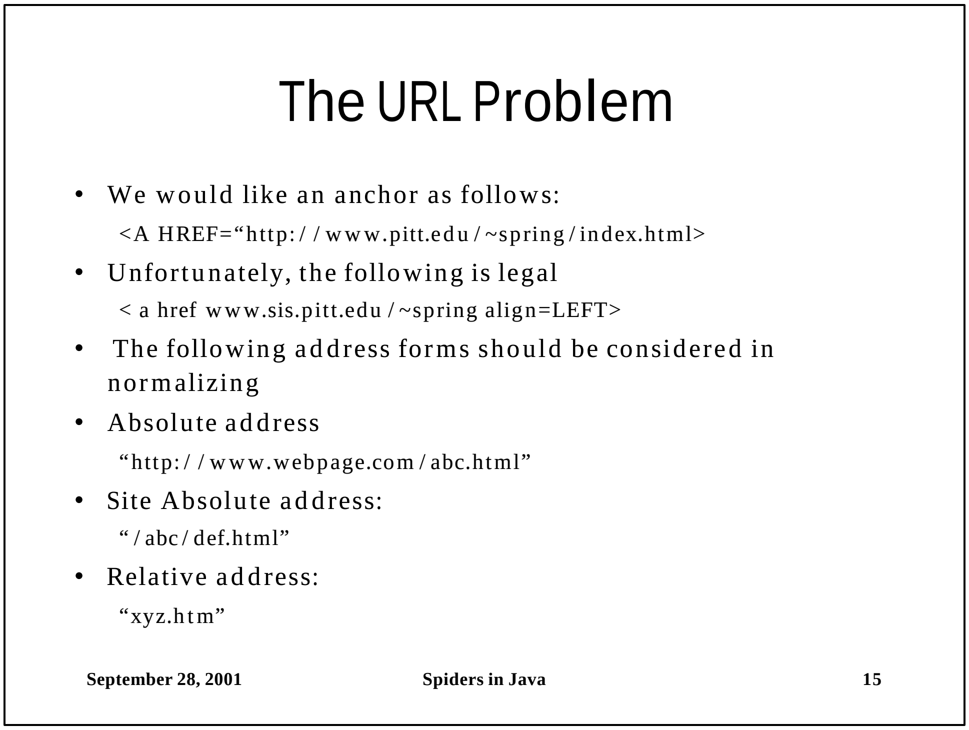
- Gets one or more pages
  - Opens a socket to a machine on Port 80
  - Writes a request:
    - “GET /homepage.html HTTP/1.0<CR><LF><CR><LF>”
  - Note: you can include any number of headers
  - Proceed to read the response

Read the Response

- Use strstr like method to find the end of the header (CRLFCRLF)
- Parse the headers into name value pairs
  - Find the length of the body from the header called “Content-Length”
  - This value indicates the length of the body only and excludes the length of the header
- Read the rest of the reponse (Make sure your read loop reads the entire response)
- Handle the response based on the content type

Parse the response for anchors

- Find all the elements on the page which will contain URLs – anchors, frames, images, maps.
- Process the page elements to find the URLs
- Find href attributes in <a> Anchor elements and obtain the literal string associated with the href
- There are at least 4 problems associated with this process:
  - HTML is case insensitive regarding attributes
  - The ‘=’ is NOT required
  - “ “ quotes are NOT required
  - The string literal may be absolute, site absolute, or relative



The URL Problem

- We would like an anchor as follows:

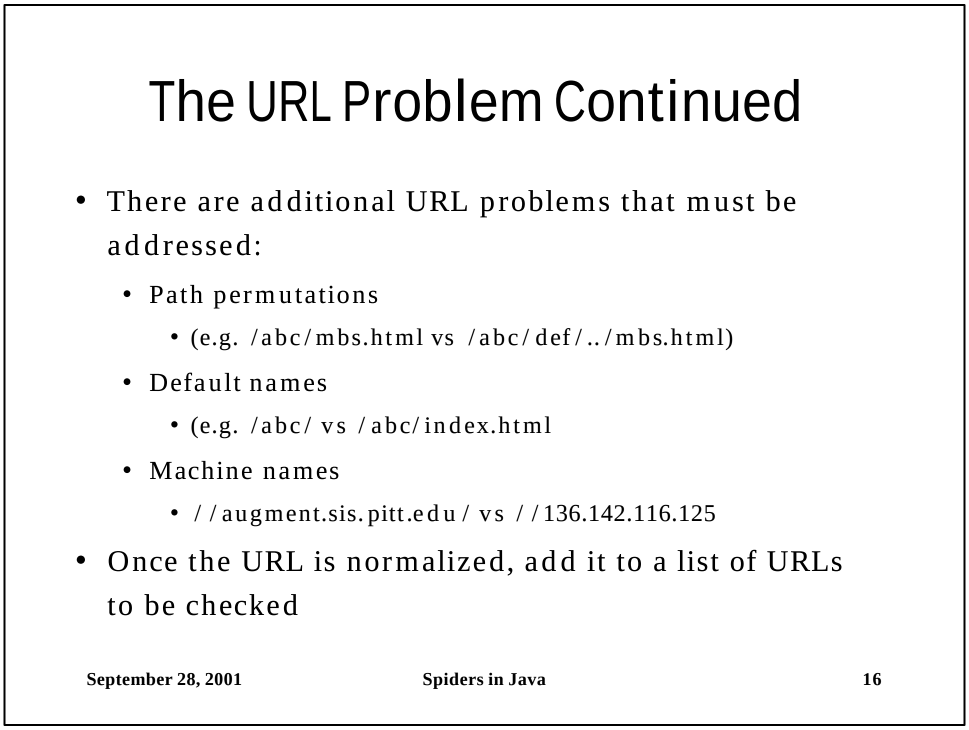
  <A HREF=“http://www.pitt.edu/~spring/index.html>
- Unfortunately, the following is legal

  < a href www.sis.pitt.edu/~spring align=LEFT>
- The following address forms should be considered in normalizing
- Absolute address

  “http://www.webpage.com/abc.html”
- Site Absolute address:

  “/abc/def.html”
- Relative address:

  “xyz.htm”



The URL Problem Continued

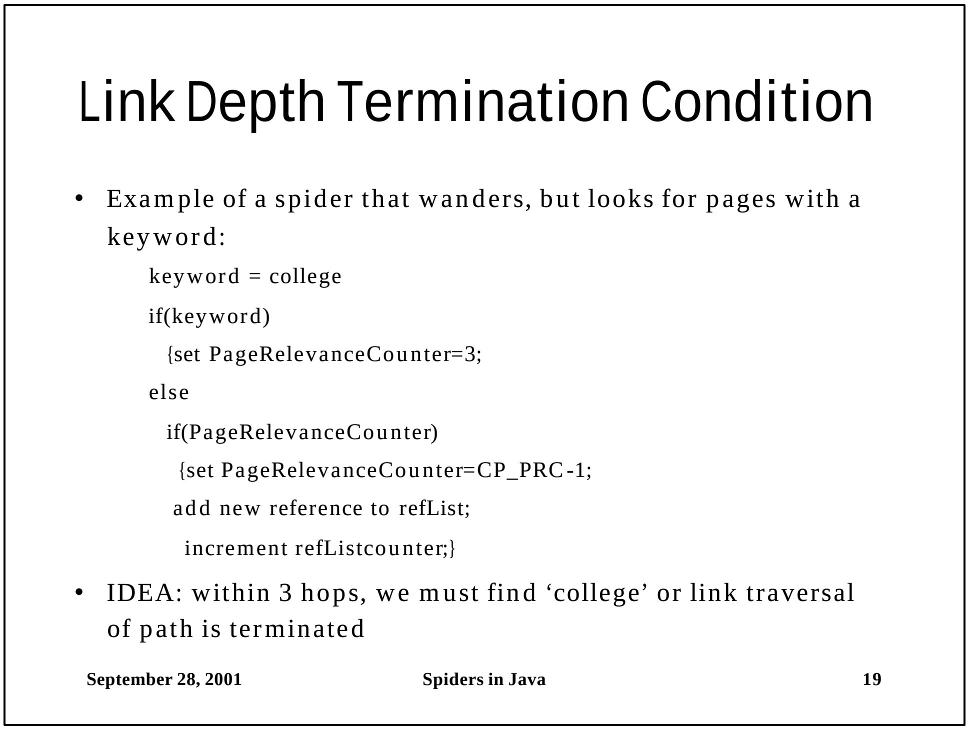
- There are additional URL problems that must be addressed:
  - Path permutations
    - (e.g. /abc/mbs.html vs /abc/def/../mbs.html)
  - Default names
    - (e.g. /abc/ vs /abc/index.html
  - Machine names
    - //augment.sis.pitt.edu/ vs //136.142.116.125
- Once the URL is normalized, add it to a list of URLs to be checked

Parse the response for Content

- Invoke a method on the page that analyzes the page as per your spider function:
  - Check for images
    - Methods for image analysis
  - Gather statistics on the page
    - Size, links, incoming and outgoing, tables, prices, ectc
  - Check for site related matters
    - Modification date, existence, form, etc
  - Look for term occurrence
    - Within a page
    - Within pages separated by less than n links
  - etc

Termination Condition

- The easiest termination condition – often used during development – is to get a single page and stop.
- You can also terminate after some number of pages – 1000.
- You can terminate at exhausting some finite resource – all the pages on a given site
- You can terminate after some complex conditon – don’t follow any link trail for more than five links without finding a given condition – e. g. a particular keyword

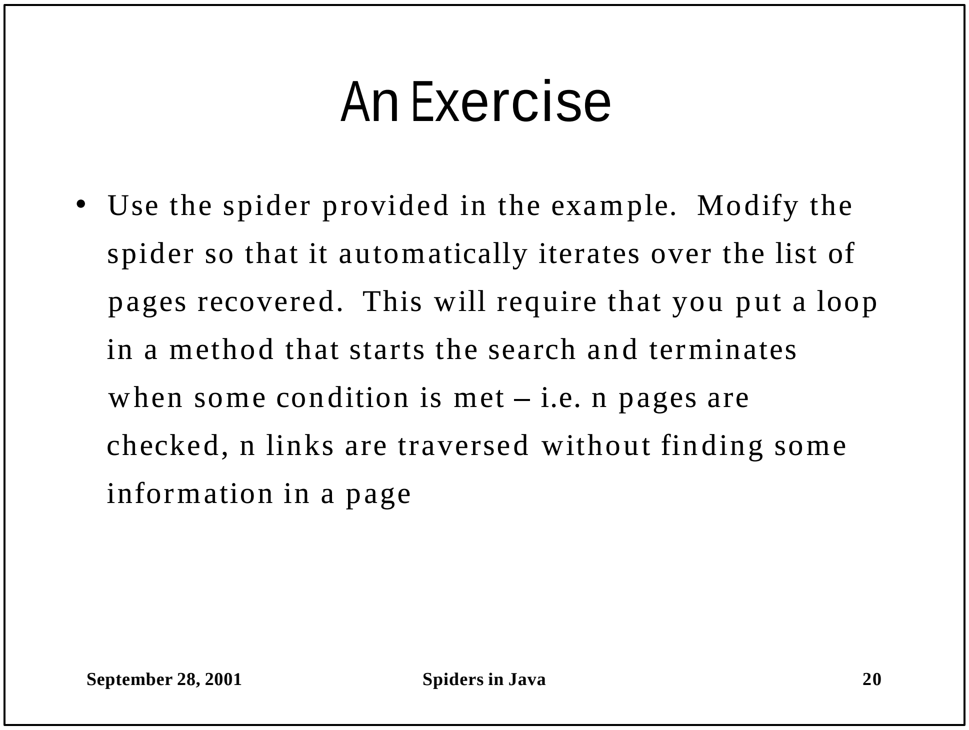


Link Depth Termination Condition

- Example of a spider that wanders, but looks for pages with a keyword:

```
keyword = college
if(keyword)
  {set PageRelevanceCounter=3;
else
  if(PageRelevanceCounter)
    {set PageRelevanceCounter=CP_PRC-1;
   add new reference to refList;
     increment refListcounter;}
```

- IDEA: within 3 hops, we must find ‘college’ or link traversal of path is terminated



An Exercise

- Use the spider provided in the example. Modify the spider so that it automatically iterates over the list of pages recovered. This will require that you put a loop in a method that starts the search and terminates when some condition is met – i.e. n pages are checked, n links are traversed without finding some information in a page