UNIT II JAVA NETWORKING FUNDAMENTALS

- Overview of Java Networking
  - TCP
  - UDP
  - InetAddress and Ports
- Socket Programming
- Working with URLs
- Internet Protocols simulation
  - HTTP
  - SMTP
  - POP
  - FTP
- Remote Method Invocation
- Multithreading Concepts.

Presented by,
B.Vijayalakshmi
Computer Centre
MIT Campus
Anna University

Working with URLs

- URL stands for **Uniform Resource Locator**
- It is a **reference (an address) to a resource on the Internet**.
- We provide URLs to our favourite Web browser so that it can locate files on the Internet in the same way that we provide addresses on letters so that the post office can locate our correspondents.
- Java programs that interact with the Internet also may use URLs to find the resources on the Internet they wish to access.
- Java programs can use a **class called URL in the java.net package to represent a URL address**.

What Is a URL?

- URL represents a resource on the World Wide Web, such as a Web page or FTP directory.
- Example: **http://www.mitindia.edu/en/**

Components of URL:

- **Protocol:** May be http, smtp, ftp. Commonly used is http.
- **Server name or IP Address:** delimited on the left by // & on right side by / or a colon
- **Port Number:** It is an optional attribute. If we write http://www.mitindia.edu: 80/en, 80 is the port number. If port number is not mentioned in the URL, it returns -1.
- **File Name or directory name:** In this case, en is the directory name.

Creating a URL

- Java have a class called **URL** in the **java.net** package to represent a URL address.

Constructors

- ➢ **URL(String address)**
  → It creates a URL object from the specified String.
- ➢ **URL(String protocol, String host, String file)**
  → Creates a URL object from the specified protcol, host, and file name.
- ➢ **URL(String protocol, String host, int port, String file)**
  → Creates a URL object from protocol, host, port and file name.
- ➢ **URL(URL context, String spec)**
  → Creates a URL object by parsing the given spec in the given context.

- Each of the four URL constructors throws a MalformedURLException if the arguments to the constructor refer to a null or unknown protocol.
- Typically, we want to catch and handle this exception by embedding our URL constructor statements in a try/catch pair

Creating a URL Cont'd

URL(String address) throws MalformedURLException

- The easiest way to create a URL object is from a String that represents the human-readable form of the URL address.
- This is typically the form that another person will use for a URL.
- In our Java program, we can use a String containing this text to create a URL object:

```
URL myURL = new URL("http://example.com/");
```

- The URL object created above represents an ***absolute URL***.
- An absolute URL contains all of the information necessary to reach the resource in question.
- We can also create URL objects from a *relative URL* address.

Creating a URL Cont'd

URL(URL context, String spec)

- **Creating a URL Relative to Another**
- A relative URL contains only enough information to reach the resource relative to (or in the context of) another URL.
- Relative URL specifications are often used within HTML files.
- For example, suppose we write an HTML file called JoesHomePage.html.
- Within this page, are links to other pages, PicturesOfMe.html and MyKids.html, that are on the same machine and in the same directory as JoesHomePage.html.
- The links to PicturesOfMe.html and MyKids.html from JoesHomePage.html could be specified just as filenames, like this:

```
<a href="PicturesOfMe.html">Pictures of Me</a>
<a href="MyKids.html">Pictures of My Kids</a>
```

- These URL addresses are *relative URLs*.
- That is, the URLs are specified relative to the file in which they are contained — JoesHomePage.html.

Creating a URL Cont’d

URL(URL context, String spec)

- In our Java programs, we can create a URL object from a relative URL specification. For example, suppose we know two URLs at the site example.com:

  http://example.com/pages/page1.html

  http://example.com/pages/page2.html

- We can create URL objects for these pages relative to their common base URL: http://example.com/pages/ like this:

```
URL myURL = new URL("http://example.com/pages/");
URL page1URL = new URL(myURL, "page1.html");
URL page2URL = new URL(myURL, "page2.html");
```

- **This constructor is also useful for creating URL objects for named anchors (also called references) within a file.**
- For example, suppose the page1.html file has a named anchor called BOTTOM at the bottom of the file.
- We can use the relative URL constructor to create a URL object for it like this:

```
URL page1BottomURL = new URL(page1URL,"#BOTTOM");
```

Creating a URL Cont’d

URL(String protocol, String host, String file)

URL(String protocol, String host, int port, String file)

- These two constructors are useful when we do not have a String containing the complete URL specification, but we do know various components of the URL.
- For example, suppose we design a network browsing panel similar to a file browsing panel that allows users to choose the protocol, host name, port number, and filename. We can construct a URL from the panel's components.
- The first constructor creates a URL object from a protocol, host name, and filename. Example: new URL("http", "example.com", "/pages/page1.html");
- This is equivalent to new URL("http://example.com/pages/page1.html");
- Note that the filename contains a forward slash at the beginning. This indicates that the filename is specified from the root of the host.
- The second URL constructor adds the port number to the list of arguments used in the previous constructor:

  URL url2 = new URL("http", "example.com", 80, "pages/page1.html");
- This creates a URL object for the following URL:

  http://example.com:80/pages/page1.html

URL addresses with Special characters

- Some URL addresses contain special characters, for example the space character. Like this,

```
http://example.com/hello world/
```

- To make these characters legal they need to be encoded before passing them to the URL constructor.

```
URL url = new URL("http://example.com/hello%20world");
```

- Encoding the special character(s) in this example is easy as there is only one character that needs encoding, but for URL addresses that have several of these characters or if you are unsure when writing our code what URL addresses we will need to access
- we can use the multi-argument constructors of the **java.net.URI** class to automatically take care of the encoding for us.

```
URI uri = new URI("http", "example.com", "/hello world/", "");
```

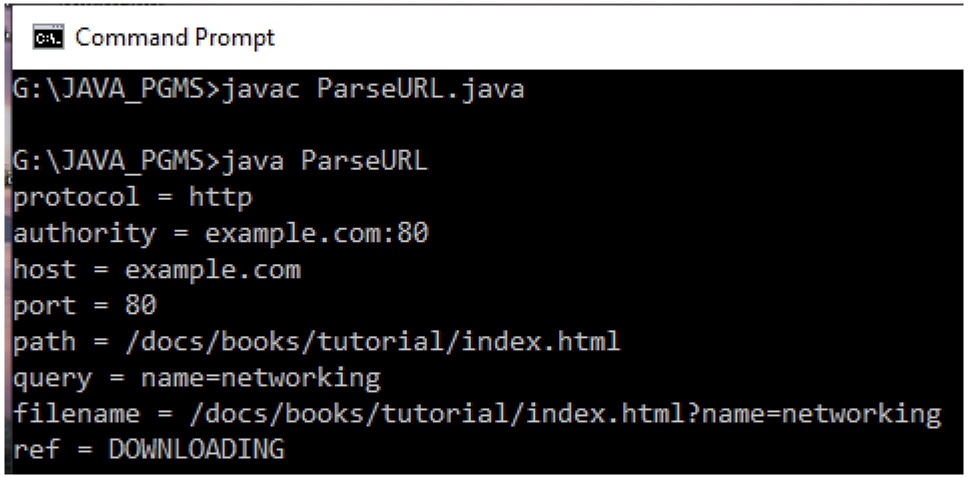
- And then convert the URI to a URL: **URL url = uri.toURL();**

Parsing a URL

- The URL class provides **several methods that let us query URL objects**.
- We can get the protocol, authority, host name, port number, path, query, filename, and reference from a URL using these accessor methods:
- <u>**URL Methods:**</u>
  - ➢ **getProtocol()→**Returns the protocol identifier component of the URL.
  - ➢ **getAuthority()→**Returns the authority component of the URL.
  - ➢ **getHost()→**Returns the host name component of the URL.
  - ➢ **getPort()→**Returns the port number component of the URL. The getPort method returns an integer that is the port number. If the port is not set, getPort returns -1.
  - ➢ **getPath()→**Returns the path component of this URL.
  - ➢ **getQuery()→**Returns the query component of this URL.
  - ➢ **getFile()→**Returns the filename component of the URL. The getFile method returns the same as getPath, plus the concatenation of the value of getQuery, if any.**getRef** Returns the reference component of the URL.
- **Note:** Remember that **not all URL addresses contain these components**. The URL class provides these methods because HTTP URLs do contain these components and are perhaps the most commonly used URLs. The URL class is somewhat HTTP-centric.
- We can use these get*XXX* methods to get information about the URL regardless of the constructor that we have used to create the URL object.

Parsing a URL -Example

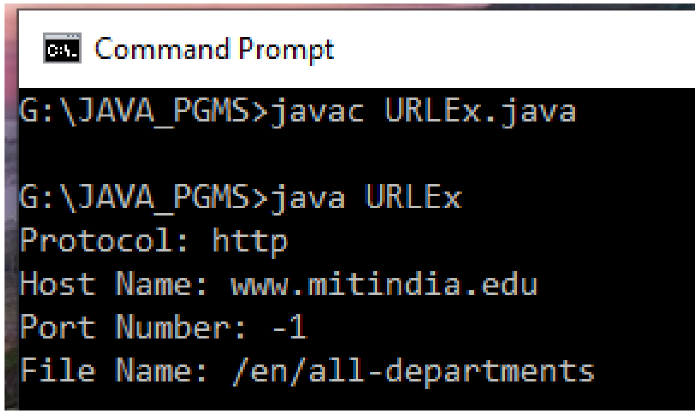
```
import java.net.*;
import java.io.*;
public class ParseURL
{
    public static void main(String[] args) throws Exception
    {
        URL aURL = new URL("http://example.com:80/docs/books/tutorial"
                        + "/index.html?name=networking#DOWNLOADING");
        System.out.println("protocol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



```
G:\JAVA_PGMS>javac ParseURL.java

G:\JAVA_PGMS>java ParseURL
protocol = http
authority = example.com:80
host = example.com
port = 80
path = /docs/books/tutorial/index.html
query = name=networking
filename = /docs/books/tutorial/index.html?name=networking
ref = DOWNLOADING
```

Parsing a URL –Example2

```
import java.io.*;
import java.net.*;
public class URLEx
{
   public static void main(String[] args)
   {
            try
            {
                        URL url=new URL("http://www.mitindia.edu/en/all-departments");

                        System.out.println("Protocol: "+url.getProtocol());
                        System.out.println("Host Name: "+url.getHost());
                        System.out.println("Port Number: "+url.getPort());
                        System.out.println("File Name: "+url.getFile());

            }catch(Exception e){System.out.println(e);}
   }
}
```



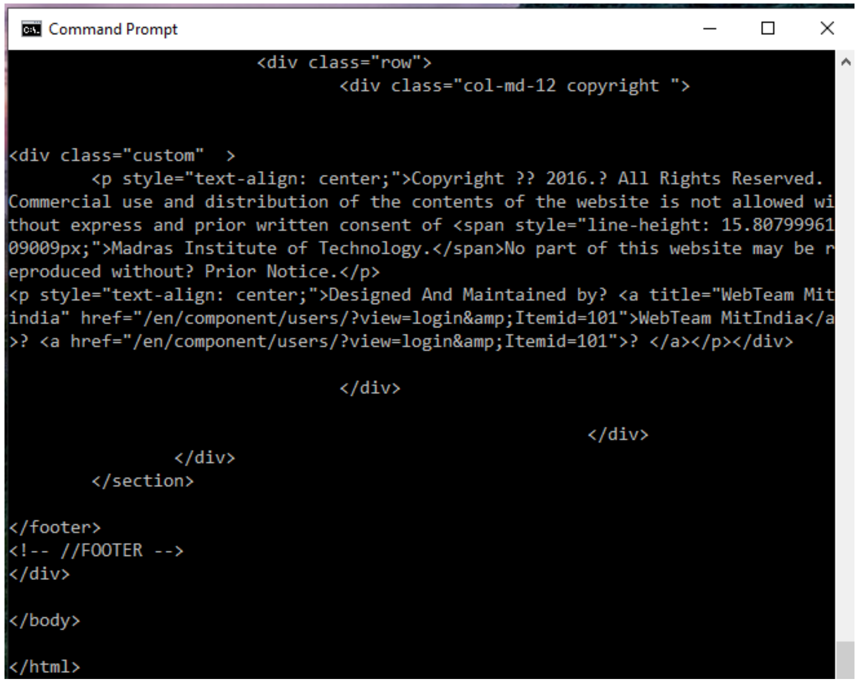
```
G:\JAVA_PGMS>javac URLEx.java

G:\JAVA_PGMS>java URLEx
Protocol: http
Host Name: www.mitindia.edu
Port Number: -1
File Name: /en/all-departments
```

Reading Directly from a URL

- After we have successfully created a URL, we can call the URL's **openStream()** method to get a stream from which we can read the contents of the URL.
- The openStream() method returns a java.io.InputStream object , so reading from a URL is as easy as reading from an input stream.
- **Steps for reading URL Content from webserver:**
    1. Create a URL object from the String representation.
    2. Create a new BufferedReader, using a new InputStreamReader with the URL input stream.
    3. Read the text, using readLine() API method of BufferedReader.
- When we run the program, we should see, scrolling by in our command window, the HTML commands and textual content from the HTML file located at given URL.
- Alternatively, the program might hang or might see the following error message, IOException: java.net.UnknownHostException:

Reading Directly from a URL

```
import java.net.*;
import java.io.*;
public class URLRead
 {
  public static void main(String[] args) throws Exception
  {
     try
      {
               URL url = new URL("http://www.mitindia.edu/en");
               BufferedReader reader = new BufferedReader(
                          new InputStreamReader(url.openStream()));
               String line;
               while ((line = reader.readLine()) != null)
                          System.out.println(line);
               reader.close();
     }catch(Exception ex){ System.out.println(ex);
    }
  }
}
```



```
                        <div class="row">
                                <div class="col-md-12 copyright ">


<div class="custom"  >
        <p style="text-align: center;">Copyright ?? 2016.? All Rights Reserved.
Commercial use and distribution of the contents of the website is not allowed wi
thout express and prior written consent of <span style="line-height: 15.80799961
09009px;">Madras Institute of Technology.</span>No part of this website may be r
eproduced without? Prior Notice.</p>
<p style="text-align: center;">Designed And Maintained by? <a title="WebTeam Mit
india" href="/en/component/users/?view=login&amp;Itemid=101">WebTeam MitIndia</a
>? <a href="/en/component/users/?view=login&amp;Itemid=101">? </a></p></div>

                                </div>

                                                        </div>
                </div>
        </section>

</footer>
<!-- //FOOTER -->
</div>

</body>

</html>
```

Reading Directly from a URL and Write into a file

```
import java.net.*;
import java.io.*;
public class URLRead2
 {
  public static void main(String[] args) throws Exception
  {
    try
     {
            URL url = new URL("http://www.mitindia.edu/en");
            BufferedReader reader = new BufferedReader(
                        new InputStreamReader(url.openStream()));
            String line;
            FileOutputStream file=new FileOutputStream("webPage.txt",true);
            int i;
            while((i=reader.read())!=-1)
            {
                  System.out.print((char)i);
                 file.write(i);
            }
            reader.close();
            file.close();
     }catch(Exception ex){ System.out.println(ex);
   }
  }
}
```

URLConnection

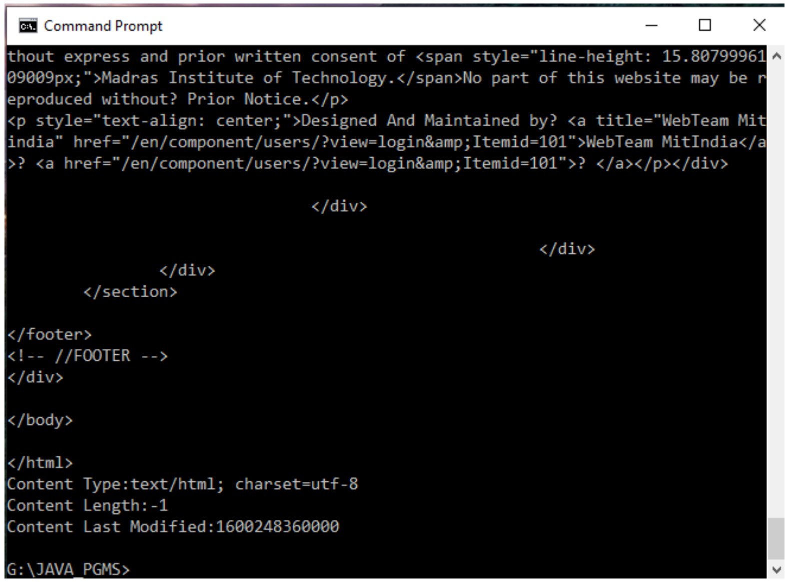
- The **Java URLConnection** class represents a communication link between the URL and the application.
- The URLConnection class contains many methods that let us to communicate with the URL over the network.
- URLConnection is an HTTP-centric class; that is, many of its methods are useful only when we are working with HTTP URLs.
- This class can be used to **read and write data** to the specified resource referred by the URL
- **Methods**
  - **URLConnection openConnection():** opens the connection to the specified URL.
  - **Object getContent():** retrieves the content of the URLConnection.
  - **getContentLength():** returns the length of the content header field and -1 if not known.
  - **getDate():** returns value of date in header field.
  - **OutputStream getOutputStream():** returns the output stream to this connection.
  - **InputStream getInputStream():** returns the input stream to this open connection.

Reading from a URLConnection

- After we have successfully created a URL object, we can call the URL object's openConnection method to get a URLConnection object, or one of its protocol specific subclasses, e.g. java.net.HttpURLConnection
- Reading from a URL can be done by openStream() method as we did in ReadURL program previously
- However, reading from a URLConnection instead of reading directly from a URL might be more useful
- This is because we can use the URLConnection object for other tasks (like writing to the URL) at the same time.
- We can also use this URLConnection object to setup parameters and general request properties that we may need before connecting.

Reading from a URLConnection

```
import java.io.*;
import java.net.*;
public class URLConnectionEx
{
 public static void main(String[] args)
 {
    try
    {
      URL url=new URL("http://www.mitindia.edu/en/");
      URLConnection urlcon=url.openConnection();
      InputStream stream=urlcon.getInputStream();
      int i;
      while((i=stream.read())!=-1)
      {
            System.out.print((char)i);
      }
      System.out.println("\nContent Type:"+urlcon.getContentType());
      System.out.println("Content Length:"+urlcon.getContentLength());
      System.out.println("Content Last Modified:"+urlcon.getLastModified());
     }catch(Exception e){System.out.println(e);}
 }
}
```


Command Prompt

```
thout express and prior written consent of <span style="line-height: 15.80799961
09009px;">Madras Institute of Technology.</span>No part of this website may be r
eproduced without? Prior Notice.</p>
<p style="text-align: center;">Designed And Maintained by? <a title="WebTeam Mit
india" href="/en/component/users/?view=login&amp;Itemid=101">WebTeam MitIndia</a
>? <a href="/en/component/users/?view=login&amp;Itemid=101">? </a></p></div>

                                </div>

                                                        </div>
                </div>
        </section>

</footer>
<!-- //FOOTER -->
</div>

</body>

</html>
Content Type:text/html; charset=utf-8
Content Length:-1
Content Last Modified:1600248360000

G:\JAVA_PGMS>
```

Writing to a URLConnection

- Many of these HTML forms use the HTTP POST METHOD to send data to the server.
- Thus writing to a URL is often called *posting to a URL*.
- The server recognizes the POST request and reads the data sent from the client.
- For a Java program to interact with a server-side process it simply must be able to write to a URL, thus providing data to the server.
- It can do this by following these steps:
  - ➢ Create a URL.
  - ➢ Retrieve the URLConnection object.
  - ➢ Set output capability on the URLConnection.
  - ➢ Open a connection to the resource.
  - ➢ Get an output stream from the connection.
  - ➢ Write to the output stream.
  - ➢ Close the output stream.

Send HTTP POST Request

To send an HTTP POST request along with parameters, you need to constructor the parameters in the following form: param1=value1&param2=value2&param3=value3

The following code snippet demonstrates how to send a login request to Twitter via HTTP POST:

```
String url = "https://twitter.com/sessions";
String email = "yourname@gmail.com";
String password = "yourpass";
 URL urlObj = new URL(url);
HttpURLConnection httpCon = (HttpURLConnection) urlObj.openConnection();
 httpCon.setDoOutput(true);
httpCon.setRequestMethod("POST");
 String parameters = "username=" + email;
parameters += "password=" + password;
OutputStreamWriter writer = new OutputStreamWriter(
 httpCon.getOutputStream());
writer.write(parameters);
writer.flush();
```

Of course this won’t work because Twitter requires more complex parameters for the login process (i.e. authentication key).