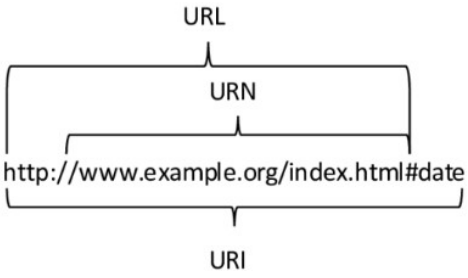
Unit 3: URLs and URIs

3.1 URIs: URLs and Relative URLs
3.2 The URL Class: Creating New URLs, Retrieving Data From a URL, Splitting a URL into Pieces, Equality & Comparison and Conversion
3.3 The URI Class: Constructing a URI, The Parts of the URI, Resolving Relative URIs, Equality & Comparison and String Representation
3.4 x-www-form-urlencoded: URL Encoder and URL Decoder
3.5 Proxies: System Properties, The ProxyClass and The ProxySelector Class
3.6 Communicating with Server-Side Programs Through GET
3.7 Accessing Password-Protected Sites: The Authenticator Class, The PasswordAuthentication Class and The JPasswordField Class

Introduction:

- The **URL** class is the simplest way for a Java program to locate and retrieve data from the network.
- We do not need to worry about the details of the protocol being used, the format of the data being retrieved, or how to communicate with the server; simply tell Java the URL and it gets the data



URI(Uniform Resource Identifier):

In Java, understanding and working with URLs and URIs involves using classes from the `java.net` package, such as `URI` and `URL`.

The `java.net.URI` class represents a Uniform Resource Identifier, providing methods for parsing, manipulating, and accessing components of a URI. It does not provide network access functionalities but is useful for working with URIs in a generic way.

Unit 3: URLs and URIs

```java
import java.net.*;
public class URIDemo{
    Run main | Debug main
    public static void main(String[] args) throws Exception {

        try {
            URI uri = new URI("http://google.com:80/search?q=java+uri#section1");
            System.out.println("Scheme: " + uri.getScheme());
            System.out.println("Authority: " + uri.getAuthority());
            System.out.println("Host: " + uri.getHost());
            System.out.println("Port: " + uri.getPort());
            System.out.println("Path: " + uri.getPath());
            System.out.println("Query: " + uri.getQuery());
            System.out.println("Fragment: " + uri.getFragment());
              // Normalize the URI
              URI normalizedURI = uri.normalize();
              System.out.println("Normalized URI: " + normalizedURI);
              // Resolve a relative URI against the base URI
              URI relativeURI = new URI("/search?q=java+examples");
              URI resolvedURI = uri.resolve(relativeURI);
              System.out.println("Resolved URI: " + resolvedURI);
              // Relativize the resolved URI against the base URI
              URI relativizedURI = uri.relativize(resolvedURI);
              System.out.println("Relativized URI: " + relativizedURI);

            } catch (URISyntaxException ex){
              System.out.println("Invalid URI syntax: "+ ex.getMessage());
            }
    }
}
```

Unit 3: URLs and URIs

Output:

```
Scheme: http
Authority: google.com:80
Host: google.com
Port: 80
Path: /search
Query: q=java+uri
Fragment: section1
Normalized URI: http://google.com:80/search?q=java+uri#section1
Resolved URI: http://google.com:80/search?q=java+examples
Relativized URI: ?q=java+examples
```

Creating a URI:

```
URI uri = new URI("http://google.com:80/search?q=java+uri#section1");
```

This line creates a new URI object with the host name "google.com".

Displaying URI Parts:

```
System.out.println("Scheme: " + uri.getScheme());
System.out.println("Authority: " + uri.getAuthority())
System.out.println("Host: " + uri.getHost());
System.out.println("Port: " + uri.getPort());
System.out.println("Path: " + uri.getPath());
System.out.println("Query: " + uri.getQuery());
System.out.println("Fragment: " + uri.getFragment());
```

Normalizing the URI:

```
// Normalize the URI
URI normalizedURI = uri.normalize();
System.out.println("Normalized URI: " + normalizedURI);
```

The `normalize` method removes any redundant path elements (like `.` or `..`) from the URI.

Unit 3: URLs and URIs

Resolving a Relative URI:

```
// Resolve a relative URI against the base URI
URI relativeURI = new URI("/search?q=java+examples");
URI resolvedURI = uri.resolve(relativeURI);
System.out.println("Resolved URI: " + resolvedURI);
```

The `resolve` method resolves a relative URI against the base URI.

Relativizing a URI:

```
// Relativize the resolved URI against the base URI
URI relativizedURI = uri.relativize(resolvedURI);
System.out.println("Relativized URI: " + relativizedURI);
```

The `relativize` method computes a relative URI that, when resolved against the original URI, yields the specified URI.

Unit 3: URLs and URIs

URL (Uniform Resource Locator): A URL is a specific type of URI that provides the means to locate a resource on the network. It defines the mechanism used to access the resource and the location where it resides.

```java
import java.net.*;
public class URLDemo{
    Run main | Debug main
    public static void main(String[] args) throws Exception {
        //Create a URL object
        URL url = new URL("http://google.com:80/search?q=java+url#section1");
        // Get and print different parts of the URL
        System.out.println("Protocol: " + url.getProtocol());
        System.out.println("Host: " + url.getHost());
        System.out.println("Port: " + url.getPort());
        System.out.println("Path: " + url.getPath());
        System.out.println("Query: " + url.getQuery());
        System.out.println("File: " + url.getFile());
        System.out.println("Ref: " + url.getRef());


    }
}
```

Output:

```
Protocol: http
Host: google.com
Port: 80
Path: /search
Query: q=java+url
File: /search?q=java+url
Ref: section1
```

Unit 3: URLs and URIs

URL in Java: The `java.net.URL` class extends `java.net.URI` and adds network access functionalities, allowing you to open connections and read from or write to resources specified by the URL.

```java
import java.net.*;

public class URLTest{
    Run main | Debug main
    public static void main(String[] args) throws Exception {
        try {
            URL url = new URL("https://www.oreilly.com/");
            BufferedReader br = new BufferedReader(new InputStreamReader(url.openStream()));
            String line;
            while((line = br.readLine()) != null){
                System.out.println(line);
            }
            br.close();
            } catch (MalformedURLException e) {
                System.err.println(e);
            }
    }
}
```

Output:

```
      <div class="sectionFigure">
        <button class="button-left" id="home-experts-slider-button-left" type="button"></button>

        <div class="home-experts-slider">
          <div class="home-experts-panels" id="home-experts-panels">
            <a href="https://learning.oreilly.com/search/?query=author%3A%22Arianne%20Dee%22&extended_publisher_data=true&highlight=true&in
tudies=true&include_courses=true&include_playlists=true&include_collections=true&include_notebooks=true&include_sandboxes=true&include_scena
ount=false&source=suggestion&sort=date_added&facet_json=true&json_facets=true&page=0&include_facets=false" style="background-image: url(http
-online-training/39040-720x720.jpg)">
              <div class="text">
                <span class="name">Arianne Dee</span>
                <span class="affiliation">Pearson</span>
              </div>
            </a>
            <a href="https://learning.oreilly.com/search/?query=author%3A%22Sari%20Greene%22&extended_publisher_data=true&highlight=true&in
tudies=true&include_courses=true&include_playlists=true&include_collections=true&include_notebooks=true&include_sandboxes=true&include_scena
ount=false&source=user&sort=date_added&facet_json=true&json_facets=true&page=0&include_facets=false" style="background-image: url(https://c
me_sari_greene_720x720.jpg)">
```

Unit 3: URLs and URIs

Relative URL

A relative URL refers to a URL that is specified relative to another URL rather than starting from the root of the file system or the web server. Relative URLs are often used when referencing resources within the same web application or when navigating within a file system structure.

```java
import java.net.*;

public class RelativeURLExample{
    Run main | Debug main
    public static void main(String[] args) throws Exception{
        try {
            URL baseUrl = new URL("http://www.ibiblio.org");
            URL relativeUrl = new URL(baseUrl,"index.html");

            System.out.println("Absolute URL " + relativeUrl);

        } catch (MalformedURLException e) {
            System.err.println(e);

        }

    }
}
```


Output

```
Absolute URL http://www.ibiblio.org/index.html
```

Unit 3: URLs and URIs

Difference between URL and URI

URL	URI
Describe the identity of a device	Technique to identify the item
Protocols used to access links to a webpage, a component or a program on a webpage	Regardless of the method utilised, helps to identify one resource from the other
About the type of protocols to be used	No protocol is specified
Is a type of URI	URI is the superset of URL
Access the location or address of the resource	Find the resource
Its components are protocol, domain, path, hash, query string and so on	Components included are scheme, authority, path, query and more
Example- https://google.in	Example-:urn:isbn:0-284-56889-3

Unit 3: URLs and URIs

URL Equality and Comparison

In Java, when you need to compare URLs for equality or perform any kind of comparison between them, you should consider a few important points due to the nature of URLs:

1) Using `equals()` method:

- The `equals()` method in Java can be used to compare two `URL` objects for equality. It checks if both objects represent the same URL.

```java
import java.net.*;

public class URLEquality {
    Run main | Debug main
    public static void main(String[] args) {

        try {
        URL ibiblio = new URL("http://www.ibiblio.org/");
        URL metalab = new URL("http://metalab.unc.edu/");

        if(ibiblio.equals(metalab)){
            System.out.println(ibiblio + "is same as " + metalab);
        }else{
            System.err.println(ibiblio + "is not same as" + metalab);
        }

        } catch (MalformedURLException e) {
            System.err.println(e);

        }

    }
}
```

Output:

```
http://www.ibiblio.org/is same as http://metalab.unc.edu/
```

This comparison checks both the protocol (http, https, etc.), host name, port number, and file/path part of the URLs.

2) String Representation Comparison:

- If you have URLs as strings, you can compare them using `String.equals()` method. This is straightforward but does not validate the URL syntax

```
String url1 = "https://example.com";
String url2 = "https://example.com";

if (url1.equals(url2)) {
    System.out.println("URLs are equal");
} else {
    System.out.println("URLs are not equal");
}
```

3) Normalization:

- URLs that appear different but actually point to the same resource (e.g., due to differences in percent-encoding or trailing slashes) can be normalized using `URL`'s `toURI()` method and then compared.

```
URL url1 = new URL("https://example.com/path/file%20name");
URL url2 = new URL("https://example.com/path/file%20name");

URI uri1 = url1.toURI();
URI uri2 = url2.toURI();

if (uri1.equals(uri2)) {
    System.out.println("URLs are equal after normalization");
} else {
    System.out.println("URLs are not equal");
}
```

This approach ensures that URLs with different percent-encoding but pointing to the same resource are considered equal.

4) Hash Code Comparison:

In Java, comparing URL hash codes directly is not a typical way to determine equality of URLs because hash codes are not guaranteed to be unique for different objects. Instead, you should compare the URLs themselves using their string representations.

```java
import java.net.URL;

public class URLComparisonExample {
    public static void main(String[] args) throws Exception {
        // Example URLs
        String urlString1 = "https://www.example.com/path1";
        String urlString2 = "https://www.example.com/path1";

        // Creating URL objects
        URL url1 = new URL(urlString1);
        URL url2 = new URL(urlString2);

        // Comparing URLs using equals method
        if (url1.equals(url2)) {
            System.out.println("URLs are equal");
        } else {
            System.out.println("URLs are not equal");
        }
    }
}
```

Unit 3: URLs and URIs

Which protocols does a virtual machine support?

```
try {
        URL u1 = new URL("http://www.ambition.edu.np/");
        System.out.println(u1.getProtocol()); // http
        URL u2 = new URL("verbatim:http://www.adc.org/");
        System.out.println(u2.getProtocol()); // error
    }
     catch (MalformedURLException ex) {
        System.err.println(ex);
}
```

Methods that retrieve data from a URL

- public InputStream openStream(): connects to the resource referenced by the URL, performs any necessary handshaking between the client and the server, and returns an Input Stream from which data can be read.
- public URLConnection openConnection(): opens a socket to the specified URL and returns a URLConnection object.
- public Object getContent(): method retrieves the data referenced by the URL and tries to make it into some type of object.

Unit 3: URLs and URIs

openStream()

- public final InputStream openStream() throws IOException

```
try {
    URL u = new URL("http://www.tufohss.edu.np");
    InputStream in = u.openStream();
    int c;
    while ((c = in.read()) != -1)
        System.out.write(c);
    in.close();
} catch (IOException ex) {
        System.err.println(ex);
}
```

openConnection()

- **public URLConnection openConnection() throws IOException**

```
try {
    URL u = new URL("http://www.tufohss.edu.np");
 try {
 URLConnection uc = u.openConnection();
 InputStream in = uc.getInputStream();
 // read from the connection...
 } catch (IOException ex) {
 System.err.println(ex);
 }
} catch (MalformedURLException ex) {
 System.err.println(ex);
}
```

getContent()

- **public final Object getContent() throws IOException**

```
try {
URL u = new URL("http://www.oreilly.com/graphics_new/animation.gif");
Object o = u.getContent();
System.out.println("I got a " + o.getClass().getName());
} catch (MalformedURLException ex) {
System.err.println(args[0] + " is not a parseable URL");
} catch (IOException ex) {
System.err.println(ex);
}
```

x-www-form-urlencoded

The `x-www-form-urlencoded` format is a way of encoding key-value pairs in HTTP requests. It's commonly used with `POST` requests where data is sent to the server from a web form. Here's how encoding and decoding work in this format:

URL Encoding (Encoder)

URL encoding transforms data into a format that can be transmitted over the internet and is safe for use within URLs. It involves replacing non-alphanumeric characters with percent-encoded hexadecimal sequences (`%xx`). Here are the main steps:

1. **Convert characters**: Convert each character that is not a letter, digit, or one of the special characters `-._~` to its hexadecimal ASCII value. For example, space (' ') becomes `%20`.
2. **Handle space**: The space character (' ') can be encoded as either `+` or `%20`, although `%20` is more common.
3. **Handle special characters**: Certain characters have special meanings in URLs (like `&`, `=`, `?`, etc.). These must be encoded as `%xx` where `xx` is the hexadecimal value of the character.

URL Decoding (Decoder)

URL decoding is the reverse process of URL encoding. It takes encoded data and converts it back into its original form:

1. **Convert %xx sequences**: Identify `%xx` sequences in the string and convert them back to the corresponding ASCII characters.
2. **Handle + signs**: Convert `+` signs back to spaces (' ').

Example

Let's encode and then decode a simple key-value pair (`name=Ram Bahadur`):

Encoding:

- Original string: `name=Ram Bahadur`
- URL encoded: `name%3DRam+Bahadur`

In this example:

- Space is encoded as `+`.
- The = character is encoded as %3D.

Decoding:

- URL encoded string: `name%3DRam+Bahadur`
- After decoding: `name=Ram Bahadur`

```java
import java.io.UnsupportedEncodingException;
import java.net.*;


public class URLEncodeDecodeExample {
    Run main | Debug main
    public static void main(String[] args) throws Exception{
        String originalString = "name=Ram Bahadur";
        //Encoding
        String encodedString = "";
        try{
            encodedString = URLEncoder.encode(originalString,"UTF-8");
            System.out.println("Encoded: "+ encodedString);
        } catch(UnsupportedEncodingException e){
            System.err.println("Unsupported encoding: " + e.getMessage());
        }
        // Decoding
        String decodedString = "";
        try {
            decodedString = URLDecoder.decode(encodedString, "UTF-8");
            System.out.println("Decoded: " + decodedString);
        } catch (UnsupportedEncodingException e) {
            System.err.println("Unsupported encoding: " + e.getMessage());
        }

    }
}
```

Output:

```
Encoded: name%3DRam+Bahadur
Decoded: name=Ram Bahadur
```

Unit 3: URLs and URIs

Communicating with Server-Side Programs Through GET

Communicating with server-side programs using the GET method involves sending data as part of the URL query string. This method is often used for retrieving data or performing operations that do not require significant security, as the data sent is visible in the URL. Here's a detailed overview:

Basics of GET Requests

- **URL Structure**: The data is appended to the URL as query parameters.
- **Format**: `http://example.com/page?key1=value1&key2=value2`
- **Visibility**: Parameters are visible in the URL, suitable for non-sensitive data.

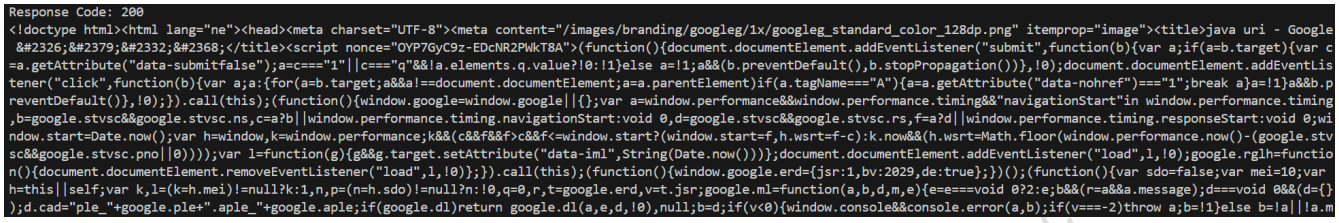
```java
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

public class GetRequestDemo{
    Run main | Debug main
    public static void main(String[] args) throws Exception {
        try {
            String url = "http://google.com:80/search?q=java+uri#section1";
            URL obj = new URL(url);
            HttpURLConnection con = (HttpURLConnection) obj.openConnection();
            con.setRequestMethod("GET");
            int responseCode = con.getResponseCode();
            System.out.println("Response Code: " + responseCode);

            BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()));
            String inputLine;
            StringBuilder response = new StringBuilder();
            while ((inputLine = in.readLine()) != null) {
                response.append(inputLine);
            }
            in.close();

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

Output



Accessing Password-Protected Sites: The Authenticator Class, The PasswordAuthentication Class and The JPasswordField Class

Accessing Password-Protected Sites

- Java's URL class **can access** sites that use **HTTP authentication**, though you'll of course need to *tell it what username and password to use.*
- **cookie-based authentication** is more challenging, not least because this varies a lot from one site to another

The `Authenticator` Class

The `Authenticator` class is part of the `java.net` package and provides a way to handle HTTP authentication (e.g., Basic Authentication). It allows you to set a global authentication mechanism for all HTTP requests in your application.

I.e.

Unit 3: URLs and URIs

- the java.net package includes an Authenticator class you can use to provide a username and password for sites that protect themselves using HTTP authentication:

  public abstract class Authenticator extends Object

- **Methods**

```
Authenticator.setDefault(new DialogAuthenticator());
// Sets the authenticator to be used when a HTTP server requires authentication.
```

Methods from the Authenticator superclass

- protected final InetAddress getRequestingSite() // requesting for the authorization,
- protected final int getRequestingPort()
- protected final String getRequestingProtocol()
- protected final String getRequestingPrompt()
- protected final String getRequestingScheme()
- protected final String getRequestingHost()
- protected final String getRequestingURL()
- protected RequestorType getRequestorType() // requester is a Proxy or a Server.

Example Methods form the Authenticator :

```
ClassAuthenticator obj1 = new ClassAuthenticator();
Authenticator.setDefault(new ClassAuthenticator());
obj1.getPasswordAuthentication() ;
```

```
public static class ClassAuthenticator extends Authenticator
 {
    protected PasswordAuthentication getPasswordAuthentication()
    {
          System.out.println("Port Requesting :" +  getRequestingPort());
          String username = "javaTpoint";
          String password = "java";
          return new PasswordAuthentication(username, password.toCharArray());
    }
```

The `PasswordAuthentication` Class

The `PasswordAuthentication` class is used in conjunction with the `Authenticator` class to store the username and password.

I.e.

- **PasswordAuthentication** is a very simple final class that supports **two read-only properties**: username and password.
- **Syntax**

```
public PasswordAuthentication(String userName, char[] password)
```

- Each is **accessed** via a getter method:

```
public String getUserName( )
public char[] getPassword( )
```

Example

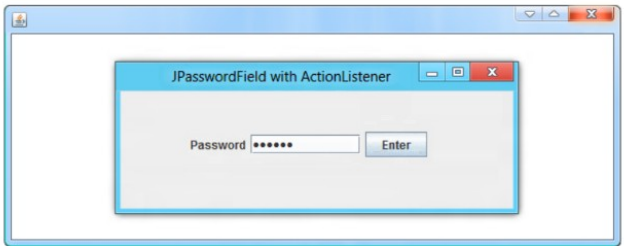
```
String userName = "user";
char[] password = { 'p', 'a', 's', 's' };
PasswordAuthentication auth = new PasswordAuthentication(userName, password);
System.out.println("UserName: " + auth.getUserName());
System.out.println( "Password: "   + passwordAuthentication.getPassword());
```

The `JPasswordField` Class

The `JPasswordField` class is part of the Swing library (`javax.swing`) and provides a graphical component to securely accept password input from the user.

I.e

- One useful tool for asking users for their passwords in a more or less secure fashion is the JPasswordField component from Swing:
- public class JPasswordField extends JTextField



Example JPasswordField:

```
import javax.swing.*;
public class PasswordFieldExample {
   public static void main(String[] args) {
   JFrame f=new JFrame("Password Field Example");
    JPasswordField value = new JPasswordField();
    JLabel l1=new JLabel("Password:");
      l1.setBounds(20,100, 80,30);
       value.setBounds(100,100,100,30);
         f.add(value);  f.add(l1);
         f.setSize(300,300);
         f.setLayout(null);
         f.setVisible(true);
}
```

Proxies in Java: System Properties, ProxyClass, and ProxySelector

Java offers several mechanisms for handling proxies, each serving a different purpose:

1. System Properties:

- These are environment variables you can set to configure proxy settings for your entire Java application.
- Common properties include:
  - `http.proxyHost`: hostname of the proxy server for HTTP connections.
  - `http.proxyPort`: port number of the proxy server for HTTP connections.
  - Similar properties exist for `https`, `ftp`, etc., specifying proxy settings for different protocols.

```
System.setProperty("http.proxyHost", "192.168.254.254");

System.setProperty("http.proxyPort", "9000");

System.setProperty("http.nonProxyHosts","java.oreilly.com|xml.oreilly.com");
```

- Setting these properties allows Java applications to automatically use the configured proxy for network connections.

2. java.lang.reflect.ProxyClass (Dynamic Proxies):

- This class is used to create dynamic proxy classes at runtime.
- A dynamic proxy implements one or more interfaces and intercepts method calls made on those interfaces.
- You define an `InvocationHandler` object that determines how the intercepted methods are handled.
- This allows for functionalities like:
  - Security checks before delegating method calls to the real object.
  - Logging method invocations.
  - Caching results of method calls.

In short,

- The Proxy class allows more fine-grained control of proxy servers from within a Java program.
- Specifically, it allows you to choose different proxy servers for different remote hosts.
- The proxies themselves are represented by instances of the java.net.Proxy class.
- **Example:**

```
SocketAddress address = new InetSocketAddress("proxy.example.com", 80);
Proxy proxy = new Proxy(Proxy.Type.HTTP, address);
```

3. java.net.ProxySelector:

- This class is used to select the appropriate proxy server for a specific network request.
- You can set a custom `ProxySelector` implementation to define your own logic for choosing a proxy based on factors like URL, protocol, or user authentication.

- This allows for more granular control over proxy usage compared to system properties.
- The default `ProxySelector` uses system properties for configuration.

In short,

- **Each running virtual machine** has a single java.net.ProxySelector object it uses to *locate the proxy server for different connections*
- To change the Proxy Selector, pass the new selector to the static **ProxySelector.setDefault()** method, like so:

```
ProxySelector selector = new LocalProxySelector(); // returns list of proxies
ProxySelector.setDefault(selector);
```