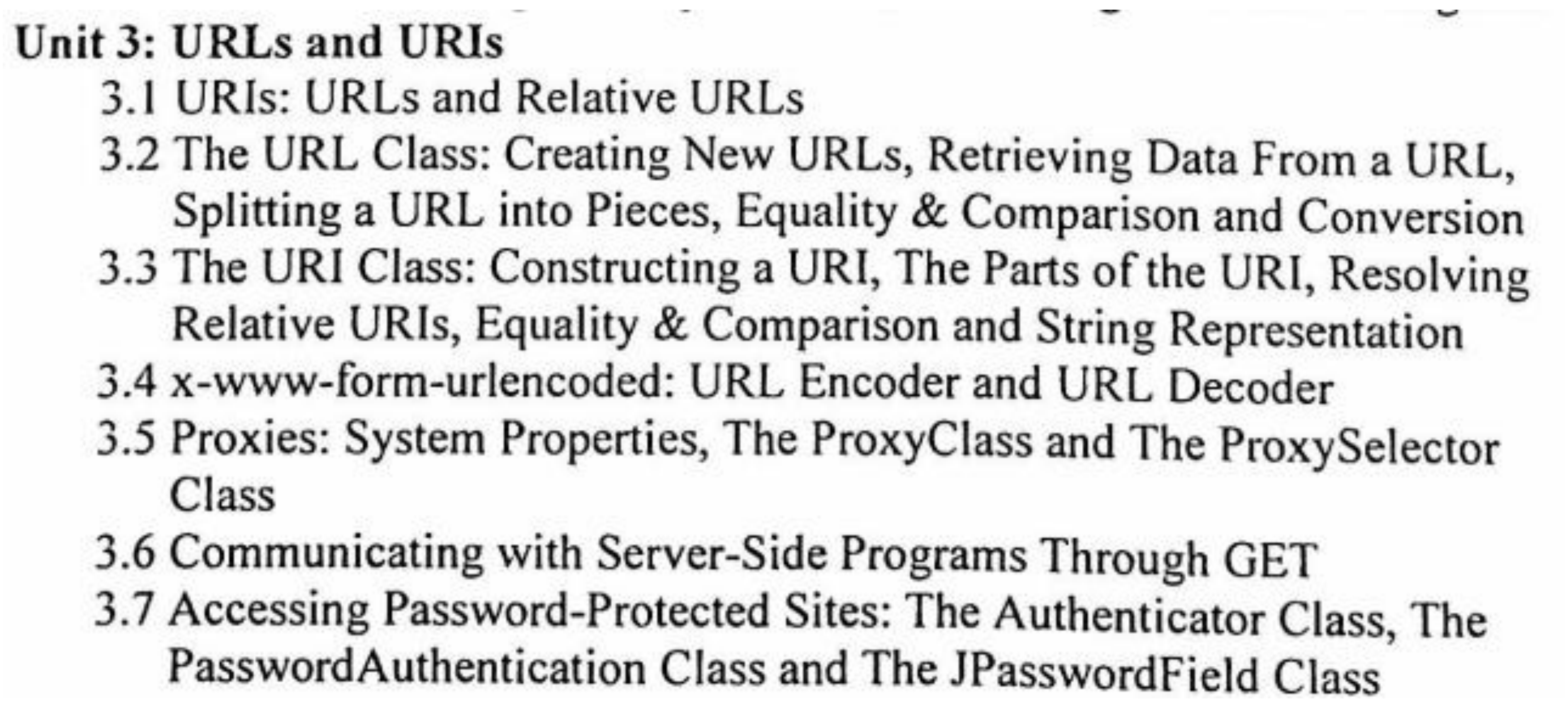
UNIT-3: URLS AND URIS



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

Krishna Pd. Acharya

URL URN URI

- A URL (Uniform Resource Locator) is a string of characters that specifies the address of a resource on the Internet. It typically includes the protocol (such as "http://" or "https://"), the domain name or IP address of the server hosting the resource, and the path to the specific resource.
- A URN (Uniform Resource Name) is another type of identifier for a resource, but it is designed to be persistent and location-independent. This means that a URN should remain valid even if the resource is moved to a different location or server. URNs are typically constructed using a naming authority, a namespace identifier, and a unique identifier for the resource.
- Let's say we want to identify a specific version of the novel "Pride and Prejudice" by Jane Austen. We could use a URL to locate the book on a specific website. For example, the URL for the book on Project Gutenberg, a website that hosts free public domain eBooks, would be: http://www.gutenberg.org/ebooks/1342
- This URL specifies the protocol (http://), the domain name (www.gutenberg.org), and the path to the specific resource (/ebooks/1342), which is the book "Pride and Prejudice" on the Project Gutenberg website.
-

INTERNET ADDRESS

- However, if we want to identify the book itself, rather than its location on a specific website, we could use a URN. For example, the URN for this version of "Pride and Prejudice" could be:

  urn:isbn:978-1-62513-096-4

- This URN uses the ISBN (International Standard Book Number) of the book as its unique identifier. It also uses the "isbn" namespace identifier to specify that the identifier is based on the ISBN standard. The URN does not specify the location of the book on a specific website, but rather identifies the book itself, regardless of its location.
- A URI (Uniform Resource Identifier) is a string of characters that identifies a resource on the Internet, including both URLs and URNs. In other words, a URI is a more general term that encompasses both URLs and URNs.
- A URI can be used to identify any kind of resource, not just web pages. For example, a URI can be used to identify a file on a server, an email address, a telephone number, or any other type of resource.
- URI is a general term for a string of characters that identifies a resource on the Internet, which can include both URLs and URNs.

URL URN URI

- For example, here are some examples of different types of URIs:
- URL: http://www.example.com/index.html
- URN: urn:isbn:978-3-16-148410-0
- URI for a file: file:///C:/Users/UserName/Documents/example.txt
- URI for an email address: mailto:example@example.com
- URI for a telephone number: tel:+1-555-555-5555
- A relative URL is a type of URL that only includes the path to a resource relative to the current page's URL, rather than the complete address. Relative URLs are often used when linking to other pages within the same website or when linking to resources on the same server.
- Relative URLs do not include the domain name or protocol (http, https) and can be either absolute or relative. An absolute relative URL specifies the complete path to a resource, starting from the root directory of the website, while a relative relative-URL specifies the path to a resource relative to the current page's URL.

URL URN URI

- For example, here are some examples of different types of URIs:
  - URL: http://www.example.com/index.html
  - URN: urn:isbn:978-3-16-148410-0
  - URI for a file: file:///C:/Users/UserName/Documents/example.txt
  - URI for an email address: mailto:example@example.com
  - URI for a telephone number: tel:+1-555-555-5555
- A **relative URL** is a type of URL that only includes the path to a resource relative to the current page's URL, rather than the complete address. Relative URLs are often used when linking to other pages within the same website or when linking to resources on the same server.
- Relative URLs do not include the domain name or protocol (http, https) and can be either absolute or relative. An absolute relative URL specifies the complete path to a resource, starting from the root directory of the website, while a relative relative-URL specifies the path to a resource relative to the current page's URL.

URL URN URI

1. ./page.html: This is a relative URL that points to a file called page.html in the same directory as the current page.
2. ../images/picture.jpg: This is a relative URL that points to a file called picture.jpg in a directory called images, which is one level up in the directory structure from the current page.
3. /contact.html: This is a relative URL that points to a file called contact.html in the root directory of the website. The slash at the beginning of the URL indicates that the path is relative to the root directory.
4. ../../../file.txt: This is a relative URL that points to a file called file.txt that is three levels up in the directory structure from the current page.
5. #section1: This is a relative URL that points to a section of the current page identified by the ID "section1". The pound sign at the beginning of the URL indicates that the link is an anchor link within the current page.

CREATING NEW URLS

- The java.net.URL class in Java is used to represent a Uniform Resource Locator (URL), which is a reference to a web resource that specifies its location on the internet. A URL consists of several components, including a protocol, a hostname, a port number, a path, and possibly other components such as a query string or a fragment identifier.
- The URL class provides several constructors and methods for working with URLs. Here are some of the key ones:

Constructors:

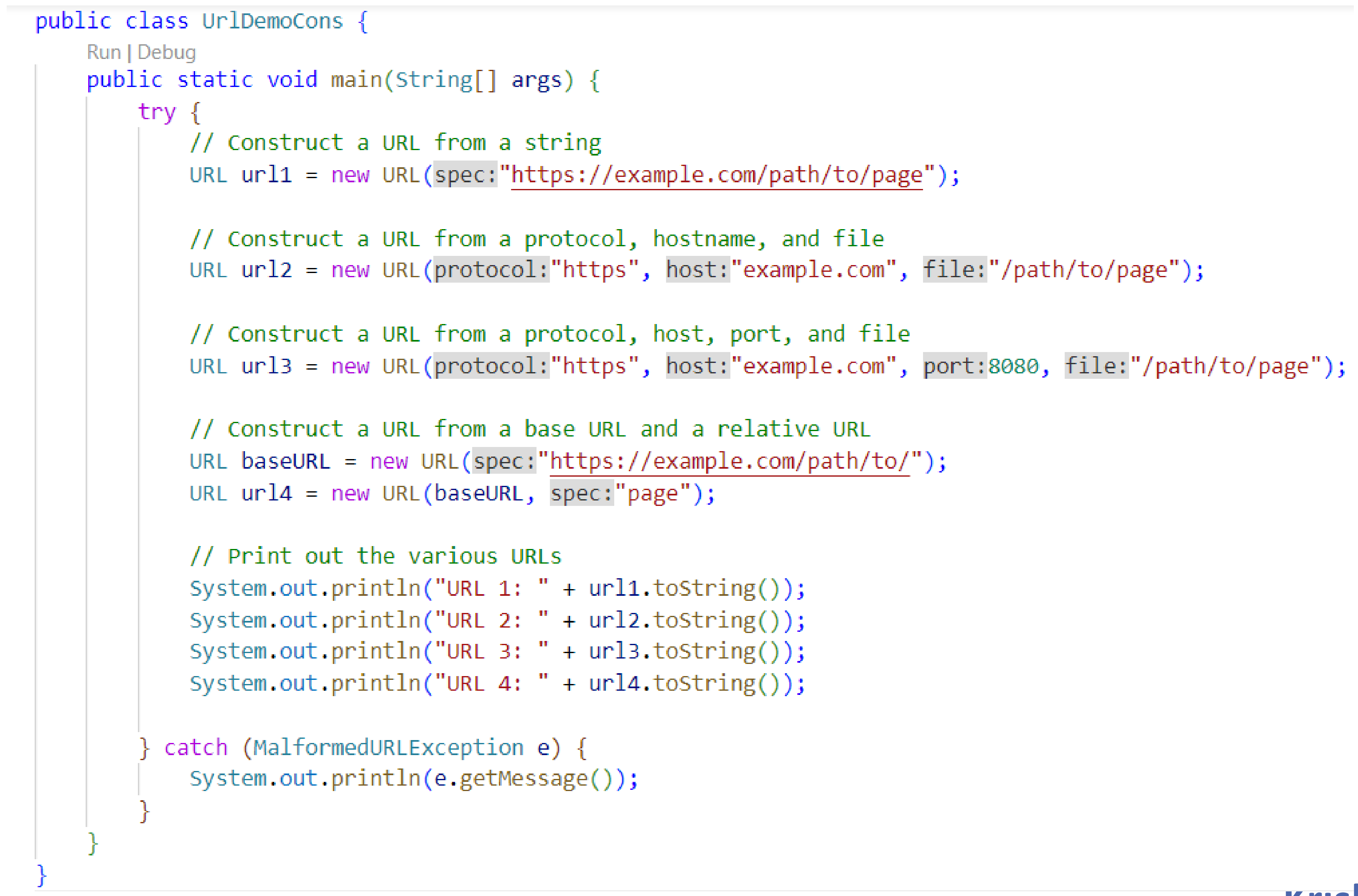
- public **URL(String spec)** throws MalformedURLException: Creates a new URL object from a string representing the URL.
- public **URL(String protocol, String host, String file)** throws MalformedURLException: Creates a new URL object from the specified protocol, host, and file.
- public URL(String protocol, String host, int port, String file) throws MalformedURLException: Creates a new URL object from the specified protocol, host, port, and file.
- public URL(URL context, String spec) throws MalformedURLException: Creates a new URL object from a context URL and a URL string.

CREATING NEW URLS

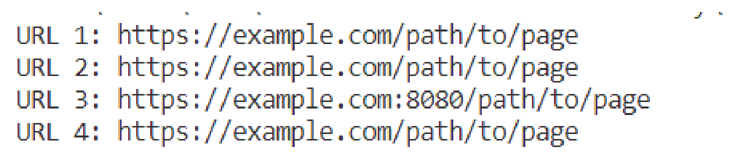
Methods:

- public String **getProtocol():** Returns the protocol of the URL.
- public **String getHost()**: Returns the host name of the URL.
- public **int getPort():** Returns the port number of the URL.
- public **String getPath():** Returns the path of the URL.
- public **String getQuery():** Returns the query string of the URL.
- public **String getRef():** Returns the fragment identifier of the URL.
- public **String toString():** Returns a string representation of the URL.
- public URLConnection openConnection() throws IOException: Opens a connection to the URL and returns a URLConnection object that can be used to read from or write to the resource.

CREATING NEW URLS

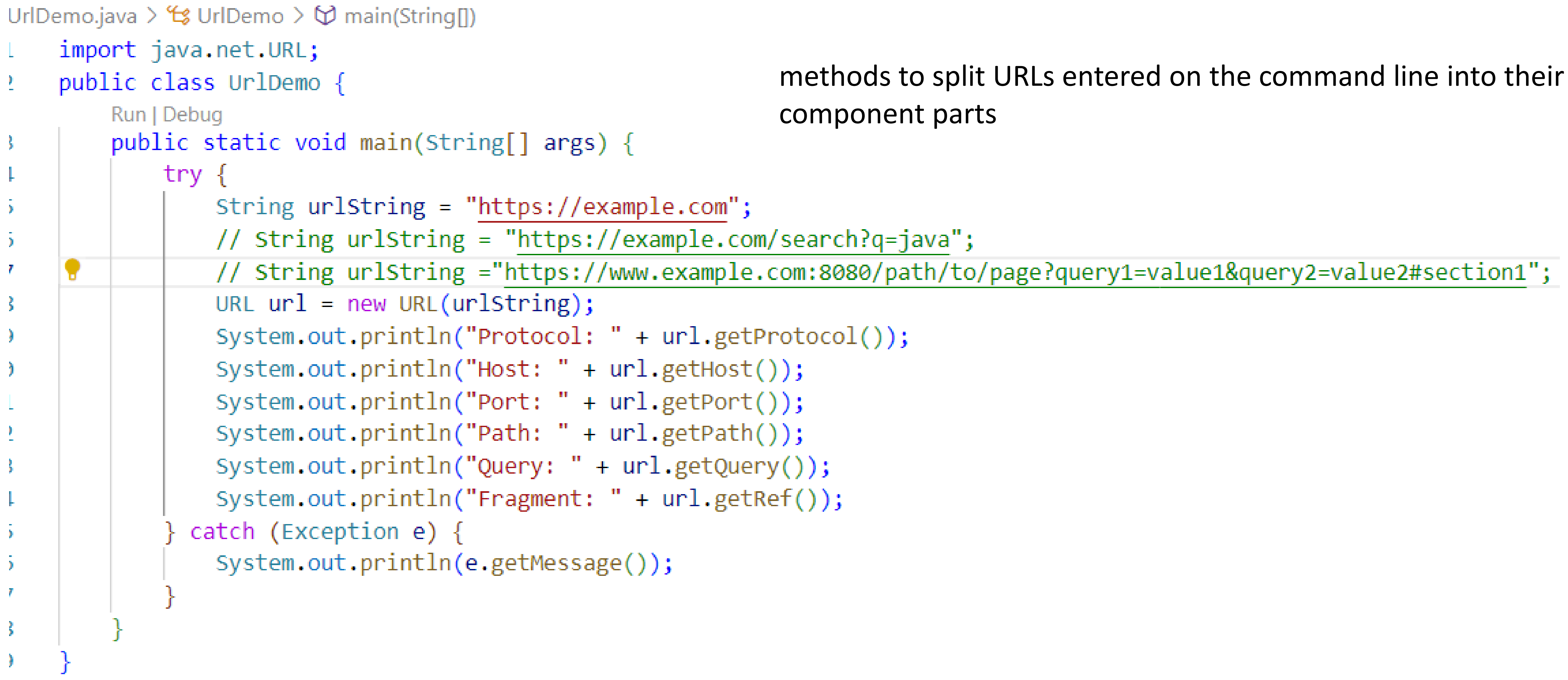


```
public class UrlDemoCons {
    Run | Debug
    public static void main(String[] args) {
        try {
            // Construct a URL from a string
            URL url1 = new URL(spec:"https://example.com/path/to/page");

            // Construct a URL from a protocol, hostname, and file
            URL url2 = new URL(protocol:"https", host:"example.com", file:"/path/to/page");

            // Construct a URL from a protocol, host, port, and file
            URL url3 = new URL(protocol:"https", host:"example.com", port:8080, file:"/path/to/page");

            // Construct a URL from a base URL and a relative URL
            URL baseURL = new URL(spec:"https://example.com/path/to/");
            URL url4 = new URL(baseURL, spec:"page");

            // Print out the various URLs
            System.out.println("URL 1: " + url1.toString());
            System.out.println("URL 2: " + url2.toString());
            System.out.println("URL 3: " + url3.toString());
            System.out.println("URL 4: " + url4.toString());

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



```
URL 1: https://example.com/path/to/page
URL 2: https://example.com/path/to/page
URL 3: https://example.com:8080/path/to/page
URL 4: https://example.com/path/to/page
```

CREATING NEW URLS



UrlDemo.java > UrlDemo > main(String[])

```java
import java.net.URL;
public class UrlDemo {
    Run | Debug
    public static void main(String[] args) {
        try {
            String urlString = "https://example.com";
            // String urlString = "https://example.com/search?q=java";
            // String urlString ="https://www.example.com:8080/path/to/page?query1=value1&query2=value2#section1";
            URL url = new URL(urlString);
            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("Fragment: " + url.getRef());
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

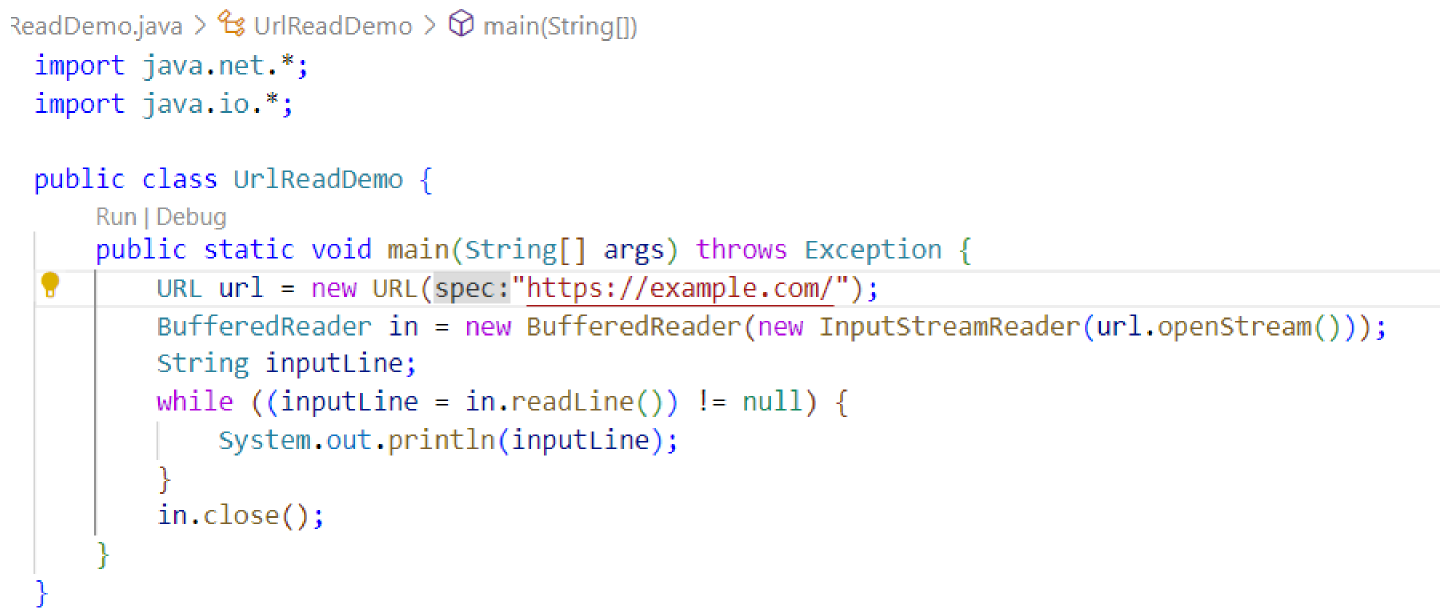
methods to split URLs entered on the command line into their component parts

RETRIEVING DATA FROM A URL

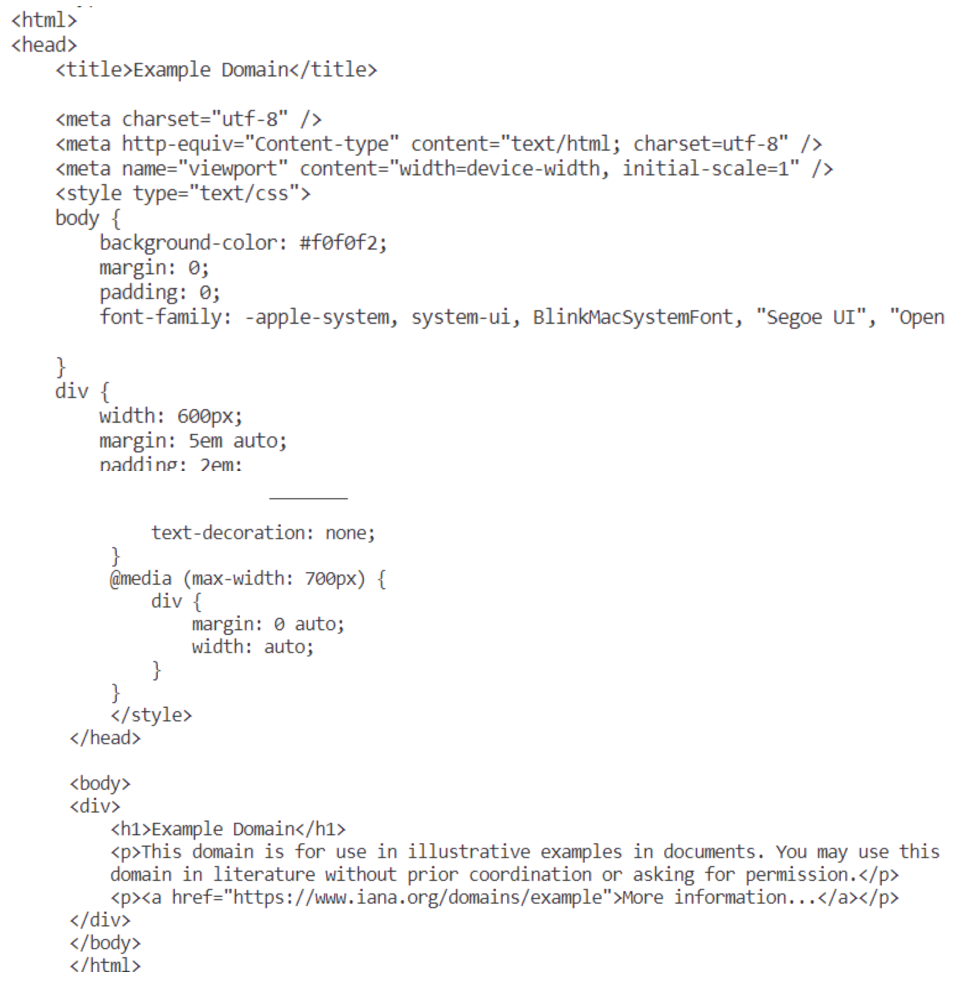
- Retrieving data from a URL in Java involves creating a connection to the URL and then reading the data from the connection's input stream.

1. public InputStream **openStream()** throws IOException: This method opens a connection to the URL and returns an input stream that can be used to read data from the resource.
2. public **URLConnection openConnection()** throws IOException: This method returns a URLConnection object that represents the connection to the URL. You can use this object to retrieve information about the connection, such as the content type, response code, and headers.
3. public **URLConnection openConnection**(Proxy proxy) throws IOException: This method returns a URLConnection object that uses the specified proxy to connect to the URL.
4. public Object **getContent()** throws IOException: This method retrieves the content of the resource as an object. The actual type of the object depends on the content type of the resource.
5. public Object **getContent**(Class[] classes) throws IOException: This method retrieves the content of the resource as an object of the specified class.

RETRIEVING DATA FROM A URL



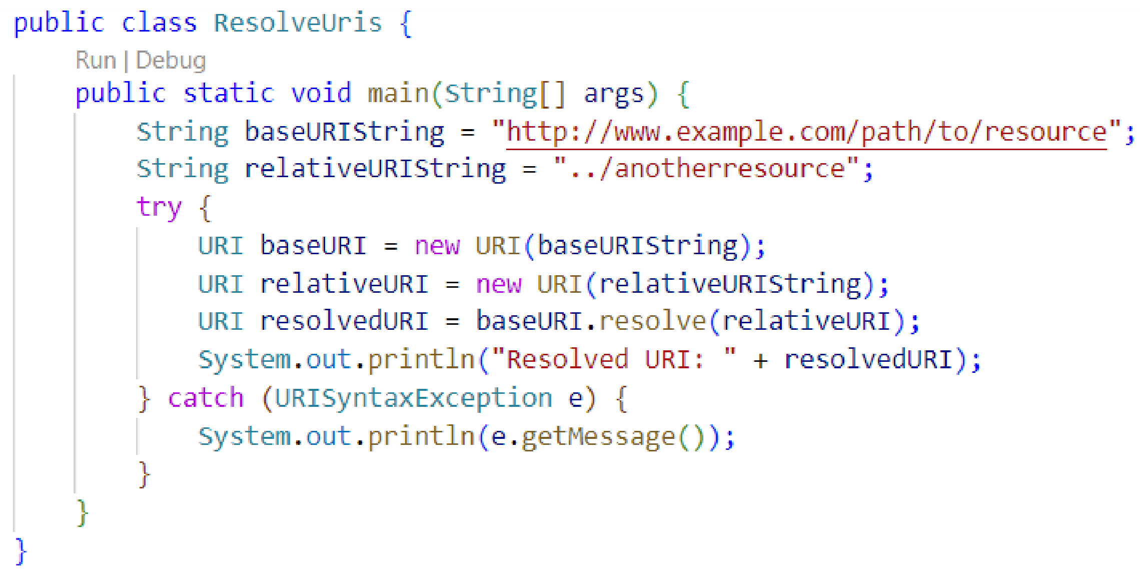
```
ReadDemo.java > UrlReadDemo > main(String[])
import java.net.*;
import java.io.*;

public class UrlReadDemo {
    Run | Debug
    public static void main(String[] args) throws Exception {
        URL url = new URL(spec:"https://example.com/");
        BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()));
        String inputLine;
        while ((inputLine = in.readLine()) != null) {
            System.out.println(inputLine);
        }
        in.close();
    }
}
```



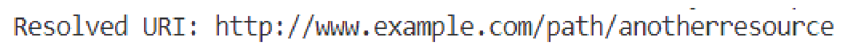
```
<html>
<head>
    <title>Example Domain</title>

    <meta charset="utf-8" />
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <style type="text/css">
    body {
        background-color: #f0f0f2;
        margin: 0;
        padding: 0;
        font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open

    }
    div {
        width: 600px;
        margin: 5em auto;
        padding: 2em;


            text-decoration: none;
        }
        @media (max-width: 700px) {
            div {
                margin: 0 auto;
                width: auto;
            }
        }
        </style>
    </head>

    <body>
    <div>
        <h1>Example Domain</h1>
        <p>This domain is for use in illustrative examples in documents. You may use this
        domain in literature without prior coordination or asking for permission.</p>
        <p><a href="https://www.iana.org/domains/example">More information...</a></p>
    </div>
    </body>
    </html>
```

RETRIEVING DATA FROM A URL

In Java, you can use the **java.net.URI** class to resolve relative URIs against a base URI. The URI class provides a method called resolve() that takes a relative URI and returns the absolute URI by resolving it against a base URI.



```java
public class ResolveUris {
    Run | Debug
    public static void main(String[] args) {
        String baseURIString = "http://www.example.com/path/to/resource";
        String relativeURIString = "../anotherresource";
        try {
            URI baseURI = new URI(baseURIString);
            URI relativeURI = new URI(relativeURIString);
            URI resolvedURI = baseURI.resolve(relativeURI);
            System.out.println("Resolved URI: " + resolvedURI);
        } catch (URISyntaxException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

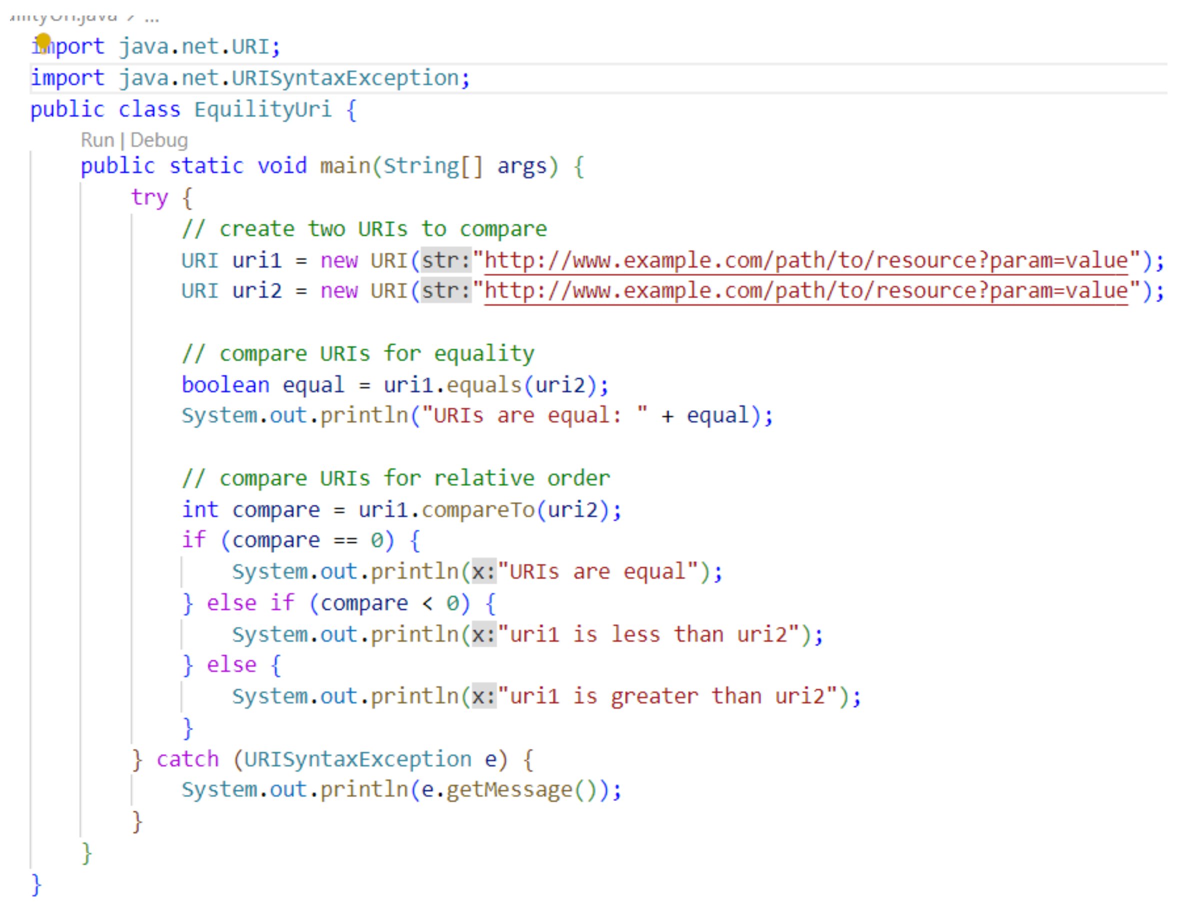


```
Resolved URI: http://www.example.com/path/anotherresource
```

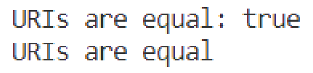
Krishna Pd. Acharya

EQUALS , COMPARETO AND STRING REPRESENTATION

In Java, you can use the java.net.URI class to compare URIs for equality and to compare the relative order of two URIs. The URI class provides the equals() method to compare two URIs for equality, and the compareTo() method to compare two URIs for relative ordering.



```
ilityUri.java > ...
import java.net.URI;
import java.net.URISyntaxException;
public class EquilityUri {
    Run | Debug
    public static void main(String[] args) {
        try {
            // create two URIs to compare
            URI uri1 = new URI(str:"http://www.example.com/path/to/resource?param=value");
            URI uri2 = new URI(str:"http://www.example.com/path/to/resource?param=value");

            // compare URIs for equality
            boolean equal = uri1.equals(uri2);
            System.out.println("URIs are equal: " + equal);

            // compare URIs for relative order
            int compare = uri1.compareTo(uri2);
            if (compare == 0) {
                System.out.println(x:"URIs are equal");
            } else if (compare < 0) {
                System.out.println(x:"uri1 is less than uri2");
            } else {
                System.out.println(x:"uri1 is greater than uri2");
            }
        } catch (URISyntaxException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

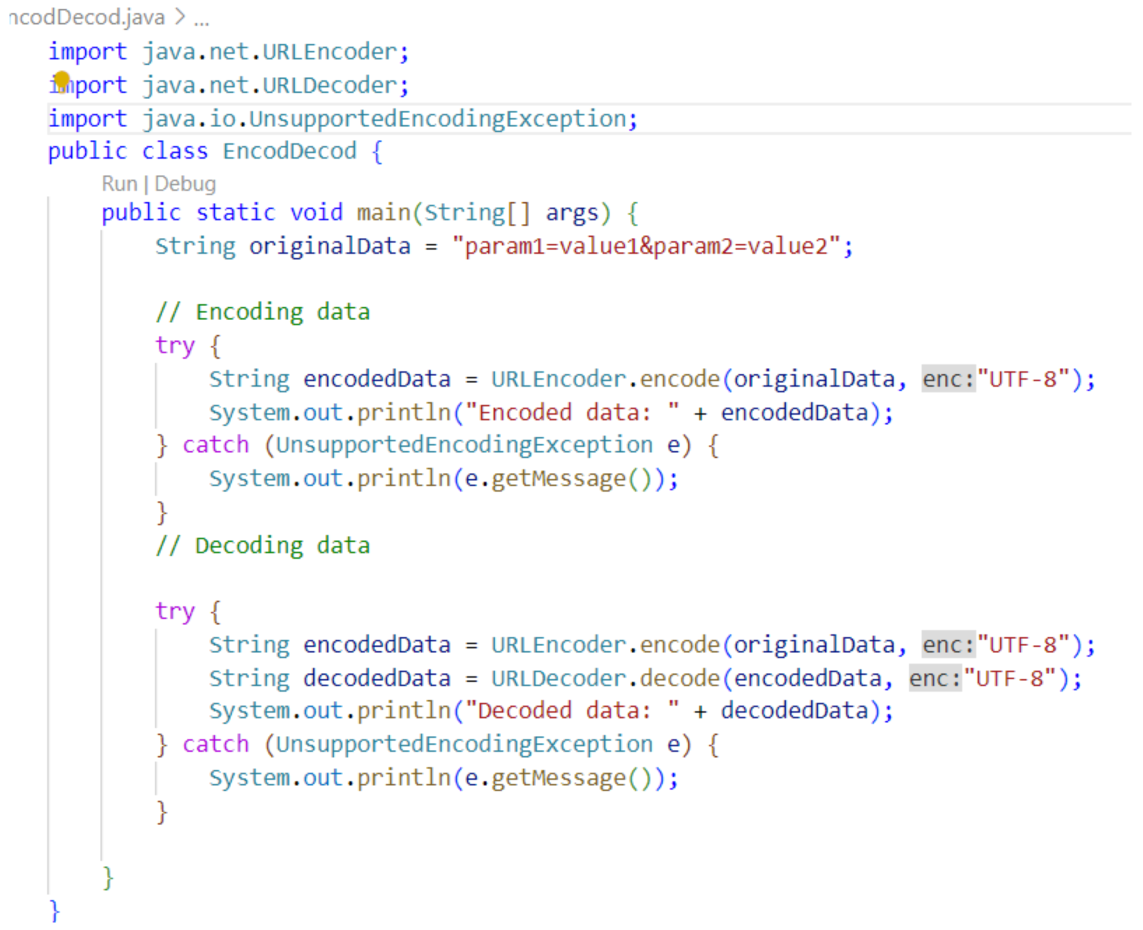


```
URIs are equal: true
URIs are equal
```

Krishna Pd. Acharya

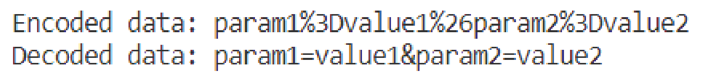
X-WW-FROM-URLENCODED

➤ The x-www-form-urlencoded format is a simple encoding scheme used to send data as key-value pairs in an HTTP request or response. In Java, you can encode and decode data in x-www-form-urlencoded format using the java.net.URLEncoder and java.net.URLDecoder classes, respectively.



ncodDecod.java > ...

```java
import java.net.URLEncoder;
import java.net.URLDecoder;
import java.io.UnsupportedEncodingException;
public class EncodDecod {
    Run | Debug
    public static void main(String[] args) {
        String originalData = "param1=value1&param2=value2";

        // Encoding data
        try {
            String encodedData = URLEncoder.encode(originalData, enc:"UTF-8");
            System.out.println("Encoded data: " + encodedData);
        } catch (UnsupportedEncodingException e) {
            System.out.println(e.getMessage());
        }
        // Decoding data

        try {
            String encodedData = URLEncoder.encode(originalData, enc:"UTF-8");
            String decodedData = URLDecoder.decode(encodedData, enc:"UTF-8");
            System.out.println("Decoded data: " + decodedData);
        } catch (UnsupportedEncodingException e) {
            System.out.println(e.getMessage());
        }

    }
}
```



```
Encoded data: param1%3Dvalue1%26param2%3Dvalue2
Decoded data: param1=value1&param2=value2
```

PROXIES: SYSTEM PROPERTIES

- In network programming, a proxy is an intermediary server that acts as an intermediary between a client and another server. Proxies are often used to improve performance, provide security, and/or enforce policies.
- When a client sends a request to a server through a proxy, the request is first sent to the proxy server, which then forwards the request to the destination server on behalf of the client. The destination server sends its response back to the proxy, which then sends the response back to the client.

Proxies can be used for various purposes, including:

1. Caching: Proxies can store frequently accessed resources locally to improve performance by reducing the number of requests to the destination server.
2. Security: Proxies can be used to filter requests based on policies, block malicious traffic, and hide the identity of the client by masking its IP address.
3. Content filtering: Proxies can be used to block access to certain websites or content based on policy.
4. Load balancing: Proxies can be used to distribute incoming requests across multiple servers to improve performance and prevent overload on any single server.

PROXIES: SYSTEM PROPERTIES

Here are some common proxy-related system properties in Java:

- http.proxyHost - specifies the hostname of the HTTP proxy server to use
- http.proxyPort - specifies the port number of the HTTP proxy server to use
- http.proxyUser - specifies the username to use for authentication with the HTTP proxy server
- http.proxyPassword - specifies the password to use for authentication with the HTTP proxy server
- https.proxyHost - specifies the hostname of the HTTPS proxy server to use
- https.proxyPort - specifies the port number of the HTTPS proxy server to use
- https.proxyUser - specifies the username to use for authentication with the HTTPS proxy server
- https.proxyPassword - specifies the password to use for authentication with the HTTPS proxy server
- To set these system properties in your Java application, you can use the System.setProperty() method. For example, to set the http.proxyHost and http.proxyPort properties, you can use the following code:

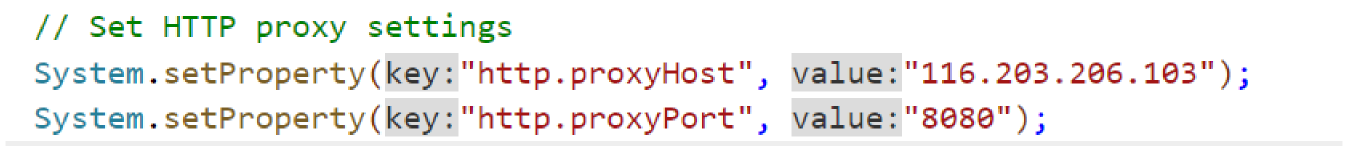
```
System.setProperty("http.proxyHost", "my.proxy.server.com");
System.setProperty("http.proxyPort", "8080");
```

- This will set the HTTP proxy host to my.proxy.server.com and the port to 8080. You can also set the other proxy-related properties in a similar manner.

PROXIES: PROXY CLASS

Proxy class is used to represent a proxy server. It has two main constructors:

- **Proxy(Type type, SocketAddress sa):** This constructor creates a Proxy object of the specified type (**HTTP, SOCKS**) and associates it with the specified socket address.
- **Proxy(Type type, SocketAddress sa, String userName, String password):** This constructor creates a Proxy object of the specified type and associates it with the specified socket address, and provides **credentials for authentication to the proxy server**.
- Once a Proxy object is created, it can be used to open a connection to a URL using the **openConnection(Proxy proxy)** method of the URL class.



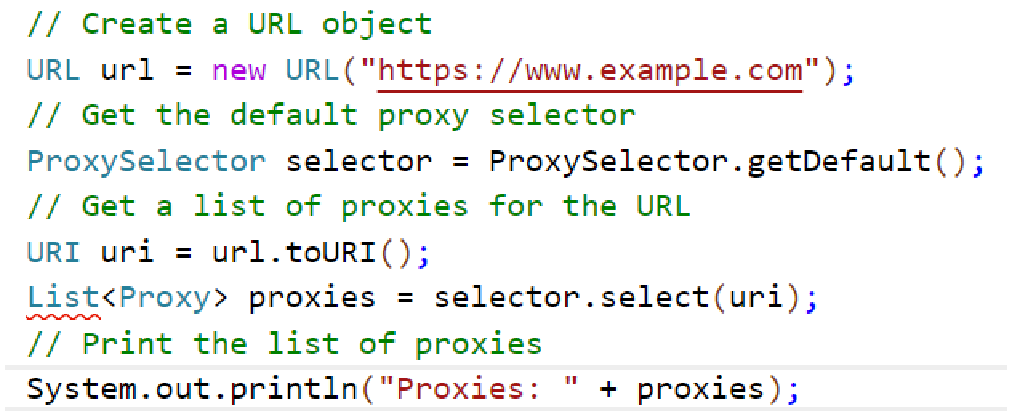
```
// Set HTTP proxy settings
System.setProperty(key:"http.proxyHost", value:"116.203.206.103");
System.setProperty(key:"http.proxyPort", value:"8080");
```

PROXIES: PROXYSELECTOR

The ProxySelector class is an **abstract class**, which means that you cannot create an instance of this class directly. Instead, you can use the **ProxySelector.getDefault**() method to get the default proxy selector for the system.

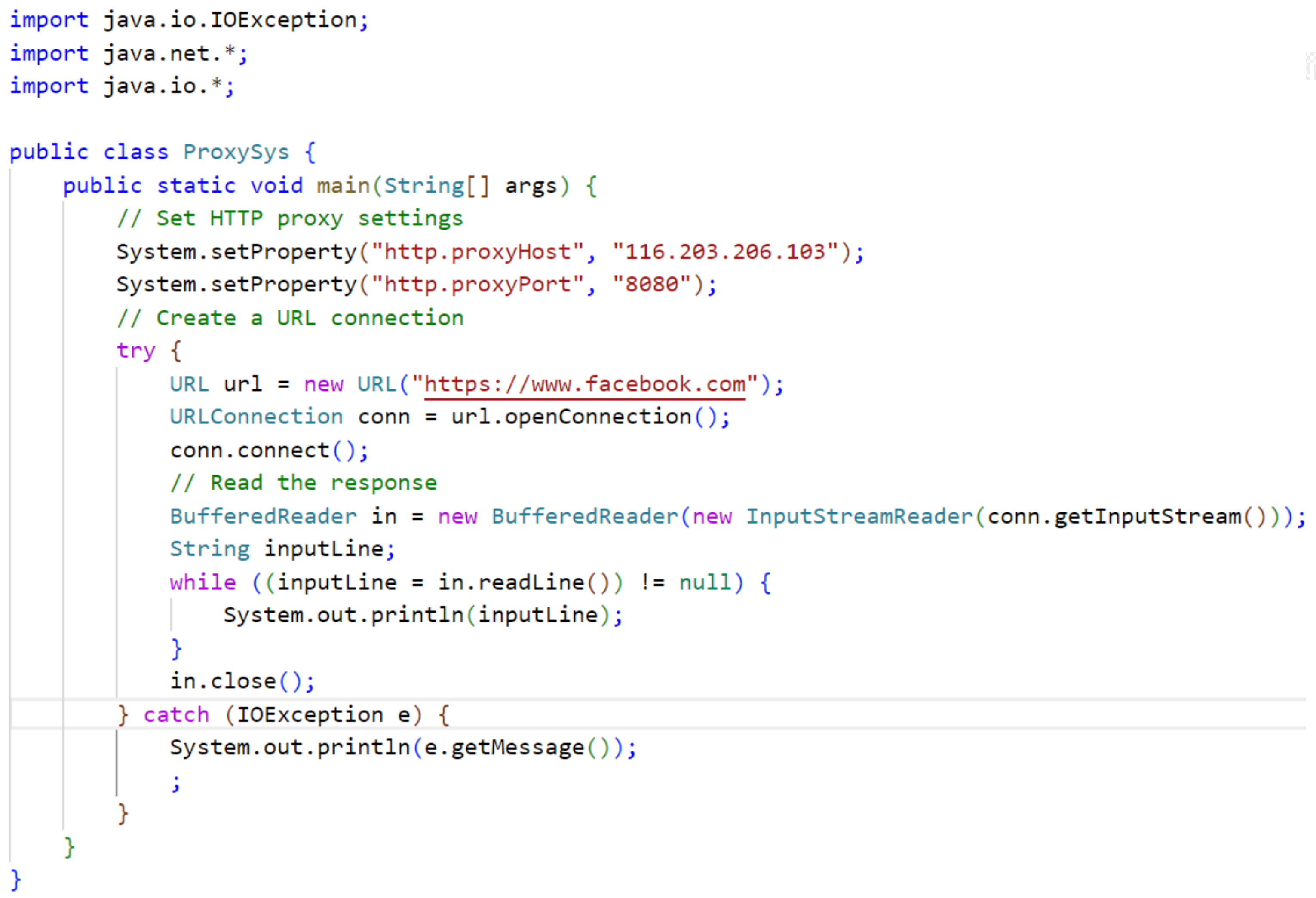
Once you have obtained a ProxySelector instance, you can use it to set up a proxy for a URL **connection by calling the openConnection()** method of the URL class and passing in a Proxy object. You can get the Proxy object by calling the **select()** method of the ProxySelector class and passing in the URI of the URL connection.

Here's an example of how to use the ProxySelector class: refer: ProxySelectorExample.java file



```
// Create a URL object
URL url = new URL("https://www.example.com");
// Get the default proxy selector
ProxySelector selector = ProxySelector.getDefault();
// Get a list of proxies for the URL
URI uri = url.toURI();
List<Proxy> proxies = selector.select(uri);
// Print the list of proxies
System.out.println("Proxies: " + proxies);
```

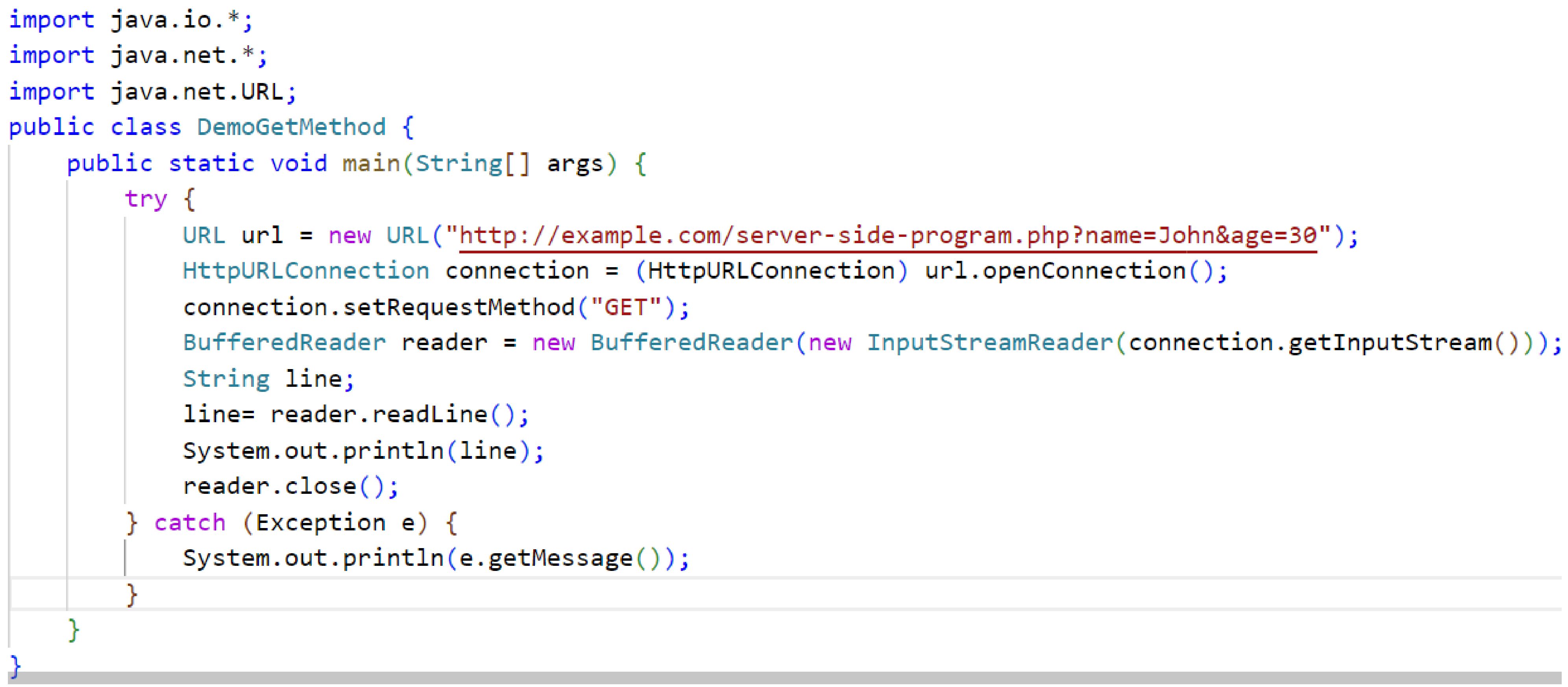
PROXIES



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

public class ProxySys {
    public static void main(String[] args) {
        // Set HTTP proxy settings
        System.setProperty("http.proxyHost", "116.203.206.103");
        System.setProperty("http.proxyPort", "8080");
        // Create a URL connection
        try {
            URL url = new URL("https://www.facebook.com");
            URLConnection conn = url.openConnection();
            conn.connect();
            // Read the response
            BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
            String inputLine;
            while ((inputLine = in.readLine()) != null) {
                System.out.println(inputLine);
            }
            in.close();
        } catch (IOException e) {
            System.out.println(e.getMessage());
            ;
        }
    }
}
```

COMMUNICATING WITH SERVER-SIDE PROGRAMS THROUGH GET

In HTTP, the GET method is used to retrieve information from a server. This method is often used to communicate with server-side programs and retrieve dynamic content. Here's an example of how to communicate with a server-side program using the GET method in Java

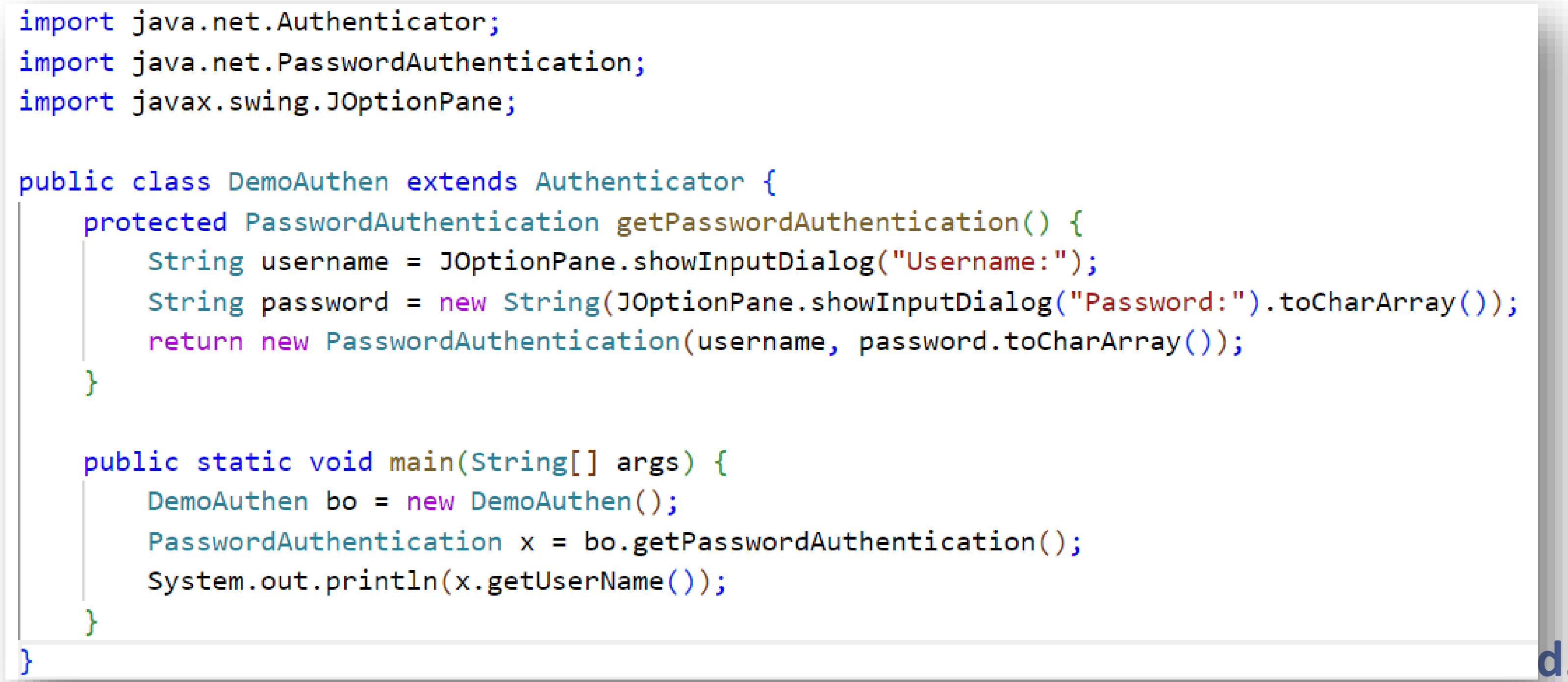


```java
import java.io.*;
import java.net.*;
import java.net.URL;
public class DemoGetMethod {
    public static void main(String[] args) {
        try {
            URL url = new URL("http://example.com/server-side-program.php?name=John&age=30");
            HttpURLConnection connection = (HttpURLConnection) url.openConnection();
            connection.setRequestMethod("GET");
            BufferedReader reader = new BufferedReader(new InputStreamReader(connection.getInputStream()));
            String line;
            line= reader.readLine();
            System.out.println(line);
            reader.close();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

ACCESSING PASSWORD-PROTECTED SITES

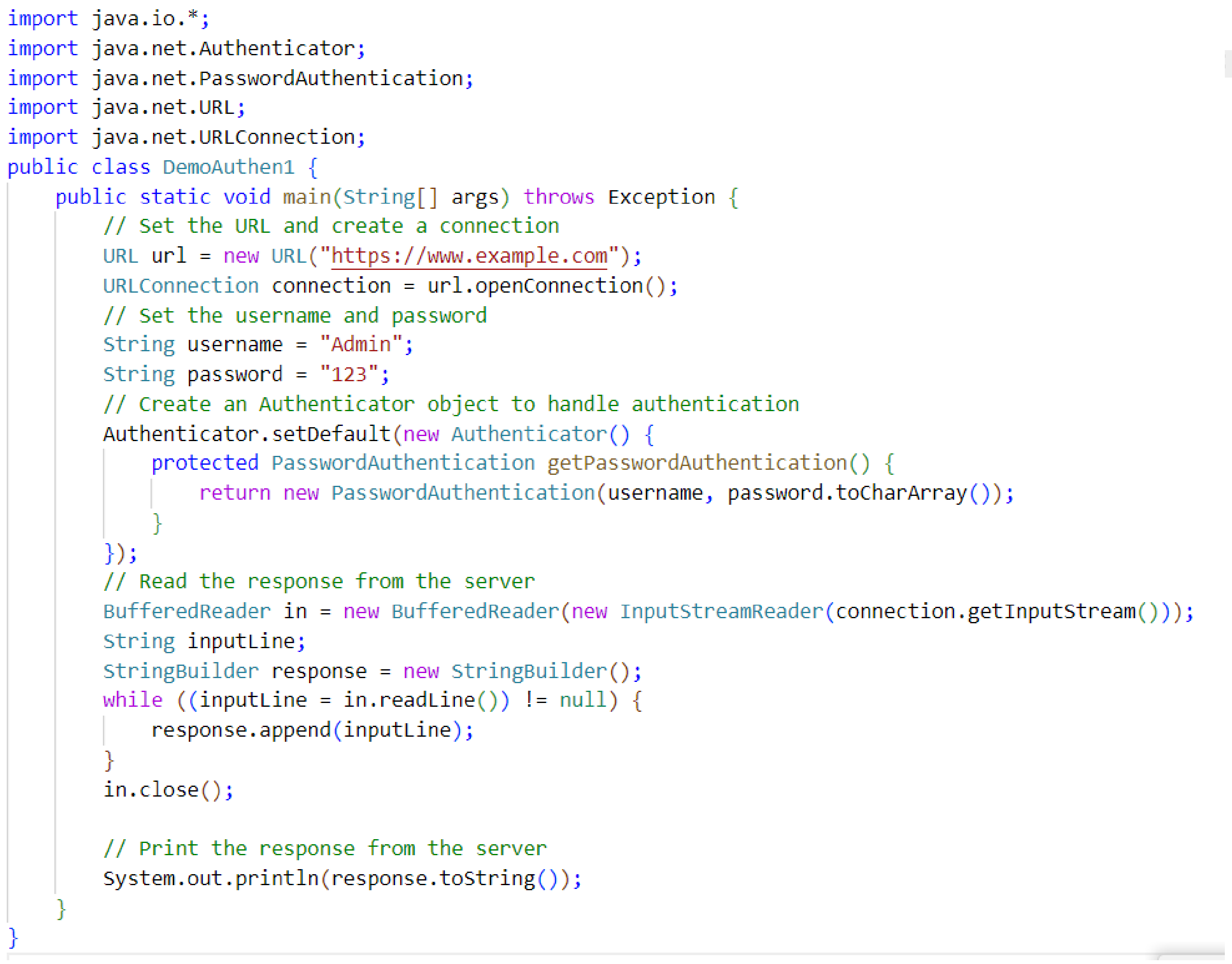
- The Authenticator class in Java is used for implementing authentication schemes for password-protected sites or for accessing resources that require authentication. It provides a framework for creating an authentication mechanism that can be used by all URLConnection objects.

- The Authenticator class is a part of the java.net package and is an abstract class that defines a single abstract method: protected PasswordAuthentication getPasswordAuthentication(). This method is called when authentication is required and returns a PasswordAuthentication object containing the username and password required for authentication.



```java
import java.net.Authenticator;
import java.net.PasswordAuthentication;
import javax.swing.JOptionPane;

public class DemoAuthen extends Authenticator {
    protected PasswordAuthentication getPasswordAuthentication() {
        String username = JOptionPane.showInputDialog("Username:");
        String password = new String(JOptionPane.showInputDialog("Password:").toCharArray());
        return new PasswordAuthentication(username, password.toCharArray());
    }

    public static void main(String[] args) {
        DemoAuthen bo = new DemoAuthen();
        PasswordAuthentication x = bo.getPasswordAuthentication();
        System.out.println(x.getUserName());
    }
}
```

ACCESSING PASSWORD-PROTECTED SITES



```java
import java.io.*;
import java.net.Authenticator;
import java.net.PasswordAuthentication;
import java.net.URL;
import java.net.URLConnection;
public class DemoAuthen1 {
    public static void main(String[] args) throws Exception {
        // Set the URL and create a connection
        URL url = new URL("https://www.example.com");
        URLConnection connection = url.openConnection();
        // Set the username and password
        String username = "Admin";
        String password = "123";
        // Create an Authenticator object to handle authentication
        Authenticator.setDefault(new Authenticator() {
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(username, password.toCharArray());
            }
        });
        // Read the response from the server
        BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
        String inputLine;
        StringBuilder response = new StringBuilder();
        while ((inputLine = in.readLine()) != null) {
            response.append(inputLine);
        }
        in.close();

        // Print the response from the server
        System.out.println(response.toString());
    }
}
```

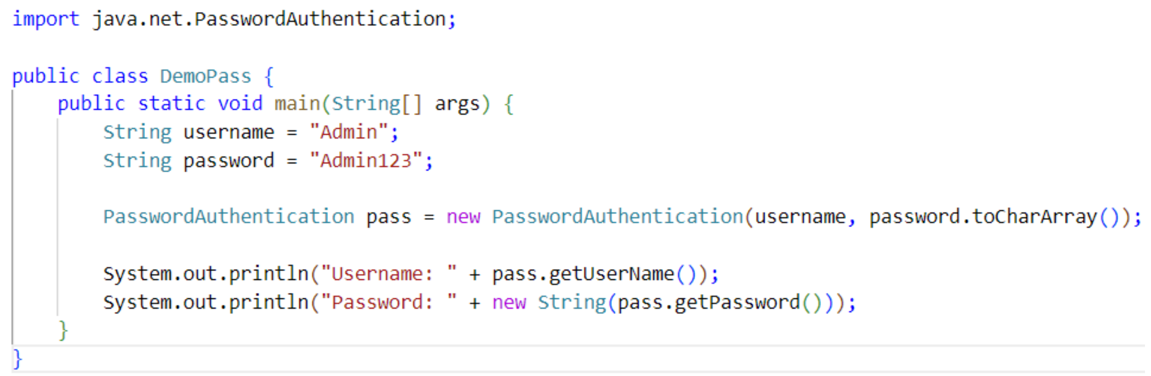
ACCESSING PASSWORD-PROTECTED SITES

- The **PasswordAuthentication** class in Java is a simple class that represents a username and password combination. It is used in **conjunction with the Authenticator class** to provide authentication credentials when accessing password-protected resources.
- The PasswordAuthentication class is a part of the **java.net package** and has two constructors:

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

public **PasswordAuthentication**(String userName, byte[] password)

The **PasswordAuthentication** class also provides two methods:
public String getUserName()
public char[] getPassword()



```java
import java.net.PasswordAuthentication;

public class DemoPass {
    public static void main(String[] args) {
        String username = "Admin";
        String password = "Admin123";

        PasswordAuthentication pass = new PasswordAuthentication(username, password.toCharArray());

        System.out.println("Username: " + pass.getUserName());
        System.out.println("Password: " + new String(pass.getPassword()));
    }
}
```

JPASSWORDFIELD CLASS

- The J**PasswordField** class in Java is a **subclass of the JTextField class** that is used for **creating text fields** that can accept password input. When text is entered into a **JPasswordField**, it is displayed as a series of asterisks or other masking characters to protect the password from being seen.
- The **JPasswordField class** is a part of the **javax.swing** package and has several constructor:

JPasswordField()

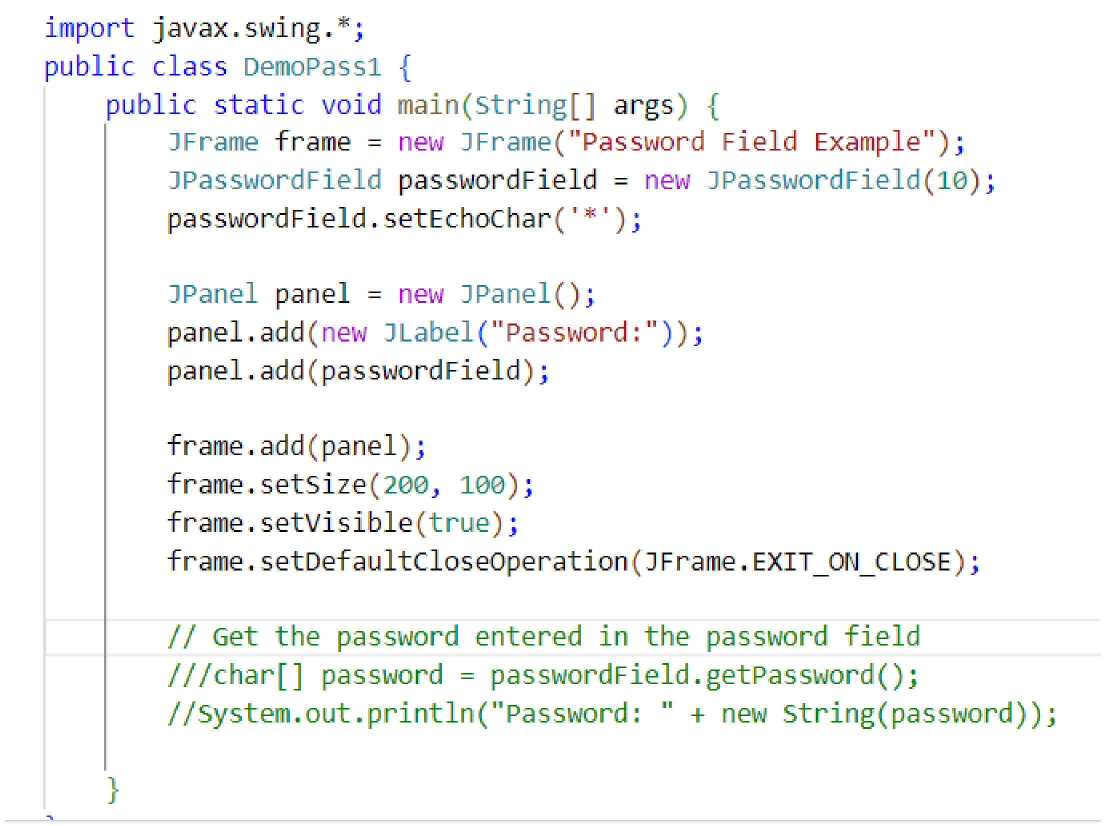
JPasswordField(int columns)

JPasswordField(String text)

JPasswordField(String text, int columns)

- The **JPasswordField** class provides a method **char[] getPassword()** that returns the password entered into the field as a char array. This is different from **getText()** method provided by **JTextField** which returns the password as a String, which is less secure.

JPASSWORDFIELD CLASS



```java
import javax.swing.*;
public class DemoPass1 {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Password Field Example");
        JPasswordField passwordField = new JPasswordField(10);
        passwordField.setEchoChar('*');

        JPanel panel = new JPanel();
        panel.add(new JLabel("Password:"));
        panel.add(passwordField);

        frame.add(panel);
        frame.setSize(200, 100);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Get the password entered in the password field
        ///char[] password = passwordField.getPassword();
        //System.out.println("Password: " + new String(password));

    }
```