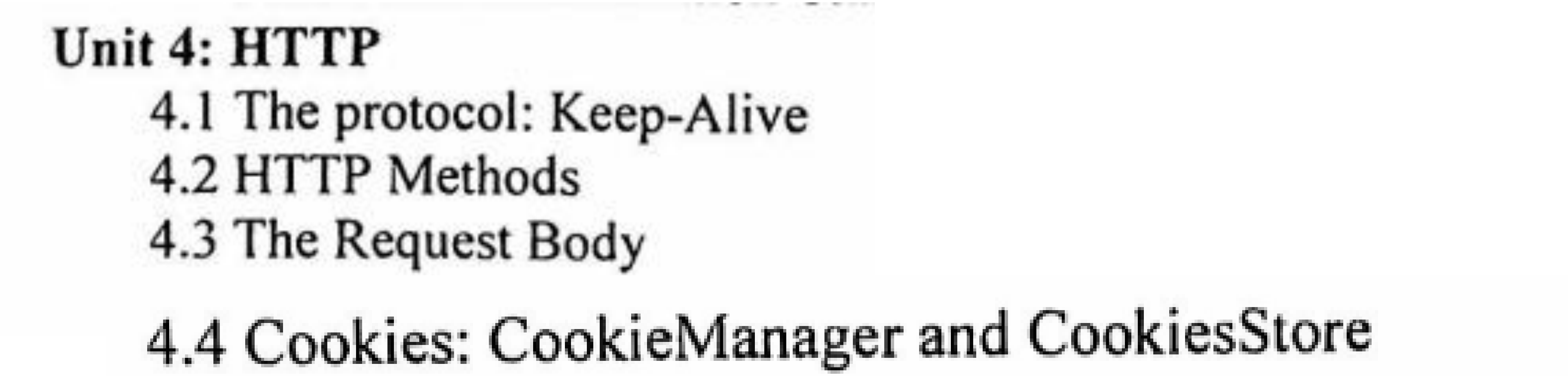
UNIT-4 HTTP



Unit 4: HTTP

4.1 The protocol: Keep-Alive

4.2 HTTP Methods

4.3 The Request Body

4.4 Cookies: CookieManager and CookiesStore

HTTP

- HTTP (Hypertext Transfer Protocol) is an **application layer protocol** that operates **on top of the TCP/IP** (Transmission Control Protocol/Internet Protocol) stack. It was **originally designed for transmitting hypertext documents**, but it has since become the primary protocol used for transmitting data over the World Wide Web.

- HTTP is a **stateless protocol**, which means that each request and response is independent of all previous requests and responses. This design allows web servers to **handle a large number of clients simultaneously**, without having to maintain **state information for each client**. However, this also means that **if a client wants to maintain state information**, such as **login credentials or shopping cart contents**, it must be managed by the client itself or by a **separate mechanism such as cookies**.

- In addition to the standard HTTP protocol, there are **several extensions and variants**, such as **HTTPS** (HTTP Secure), which uses **SSL/TLS encryption** to provide secure communication, and **HTTP/2**, which provides several performance enhancements over the original HTTP protocol.

HTTP FROM CLIENT TO SERVER

- The **client opens a TCP connection** to the server's IP address on the appropriate port (usually **port 80** for **HTTP**, or **port 443** for **HTTPS**).
- The client **sends an HTTP request** to the server over the TCP connection. The request typically includes the **HTTP method** (e.g. **GET or POST**), the URL of the requested resource, and **any headers or message** body data.
- The **server receives** the request and processes it. This may involve **fetching data from a database** or performing other server-side processing.
- The server sends **an HTTP response back to the client over the same TCP connection**. The response typically includes a status code (**e.g. 200 for success, 404 for not found**), any headers, and any message body data.
- The client receives the response and processes it. This may involve **rendering HTML content in a browser**, or using the response data in a programmatic way.
- The **client closes the TCP connection**, unless it is using HTTP keep-alive to keep the connection open for future requests

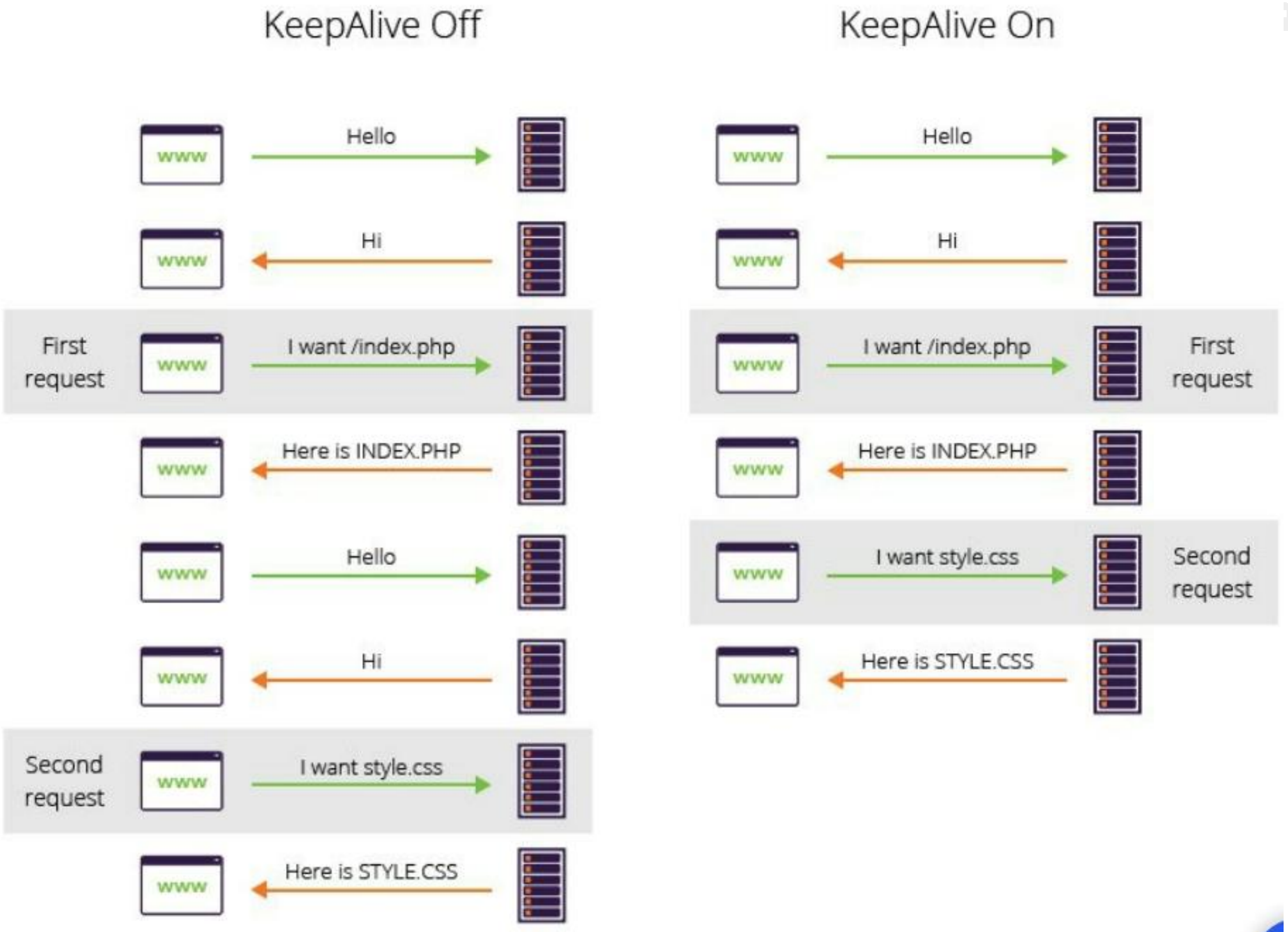
HTTP FROM CLIENT TO SERVER AND KEEP-ALIVE IS A FEATURE OF THE HTTP

- HTTP headers are **additional metadata** that are sent along with an HTTP request or response. They provide information about the **message,** such as the **content type**, **language, encoding**, and **caching information**. Headers are used to convey information that is **not directly related to the content of the message**, but which can be important for **processing or interpreting** the message.

- **Keep-Alive is a feature of the HTTP protocol** that allows the **client and server to keep the TCP connection open** after a request has been made, so that **multiple requests** and responses can be sent over the same connection. Without Keep-Alive, the client and server would **need to open and close** a new TCP connection for each request/response cycle, which can be **inefficient and slow down** the response time.

- When Keep-Alive is used, the **server keeps the connection open for a specified period of time**, during which the client can send additional requests without needing to re-establish a new TCP connection. The server can also send additional responses over the same connection. This can **improve the overall performance** of the connection by **reducing the overhead of establishing new TCP connections** for each request/response cycle.

HTTP FROM CLIENT TO SERVER



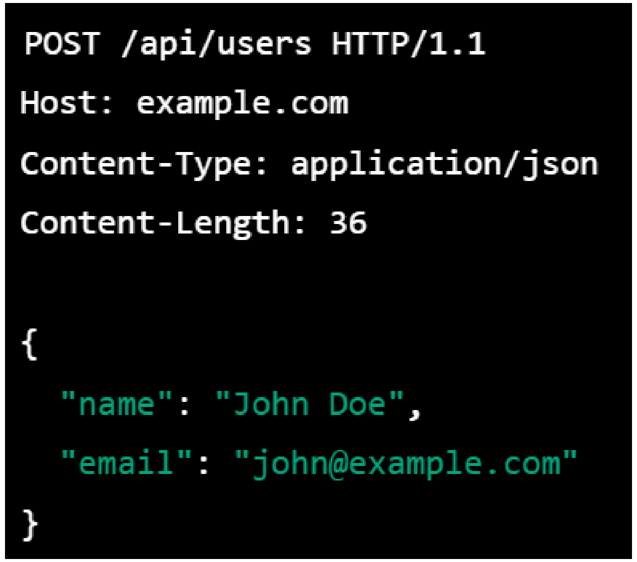
Krishna Pd. Acharya

HTTP METHODS

- HTTP defines **several methods**, also known as "**ver**bs", that indicate the **action to be performed on a resource**. The most common HTTP methods are:
- GET: **retrieves a resource from the server**. The response body contains the resource being requested.
- POST: **sends data to the server to be processed**. The request body contains the data to be processed, and the response body typically contains the result of the processing.
- PUT: **updates a resource on the server**. The request body contains the updated version of the resource, and the response body typically contains a confirmation of the update.
- DELETE: **deletes a resource from the server**. The request body is typically empty, and the response body typically contains a confirmation of the deletion.
- HEAD: **retrieves only the header information of a resource**, without the response body.
- OPTIONS: retrieves information about the **communication options available for a resource**, such as the supported HTTP methods and content types.
- CONNECT: **establishes a network connection to a resource**, typically used for secure communication through a proxy.
- TRACE: **retrieves a diagnostic trace of the HTTP request and response**, useful for debugging and troubleshooting.

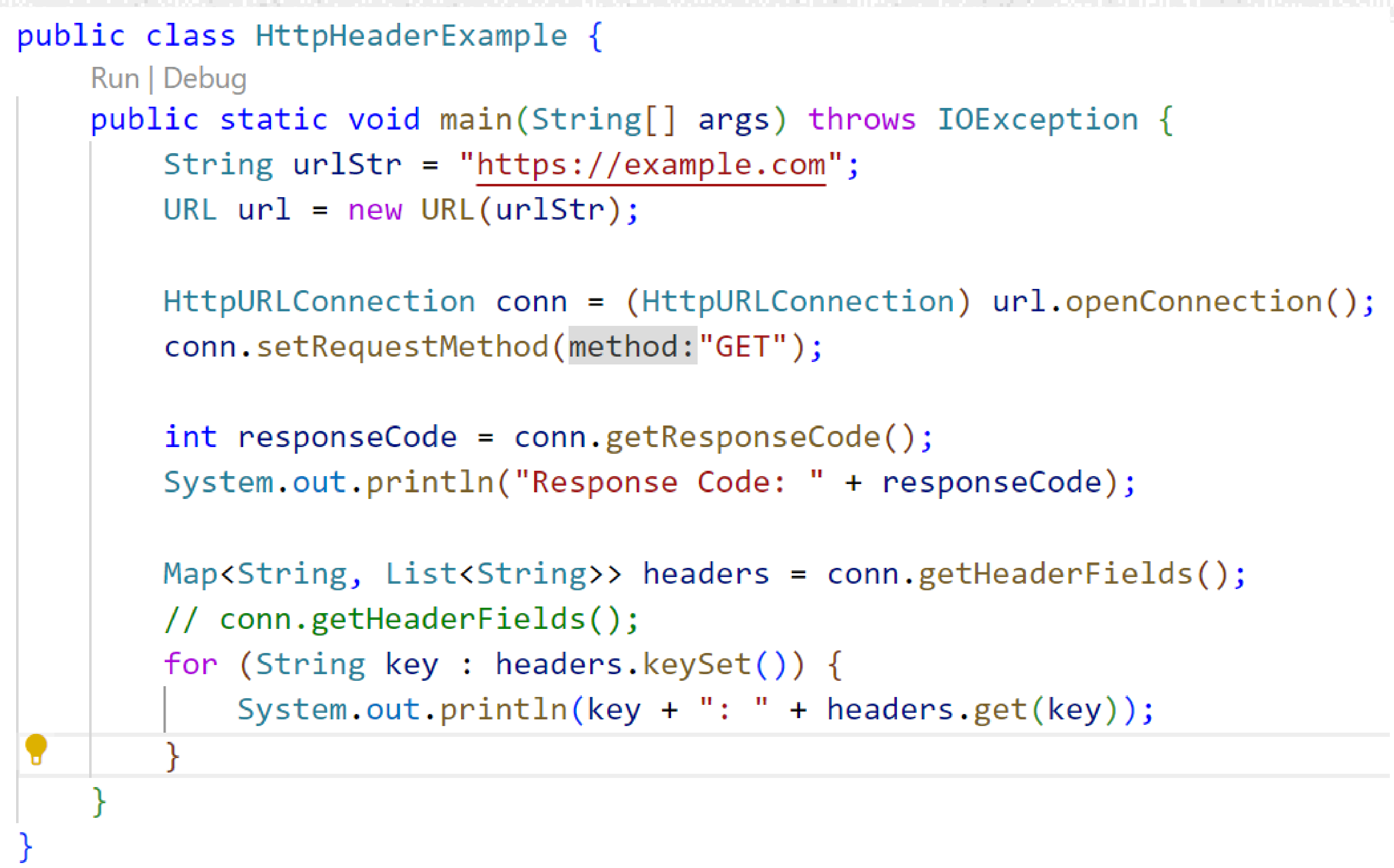
HTTP REQUEST BODY

- The **HTTP request body** is the data that is sent from the **client to the server as part of an HTTP request message**. The request body is **optional** and **not all HTTP** requests have **a request body.** Whether a request has a body or not **depends on the HTTP method being used,** and the type of data being sent.
- The HTTP methods that typically include **a request body are POST and PUT**. These methods are used to **send data to the server for processing, updating, or creating new resources**. For example, when submitting a form or uploading a file to a web server, the data is usually sent in the **request body using the POST method**.



```
POST /api/users HTTP/1.1
Host: example.com
Content-Type: application/json
Content-Length: 36

{
  "name": "John Doe",
  "email": "john@example.com"
}
```

PROGRAM TO OBTAIN HTTP HEADER

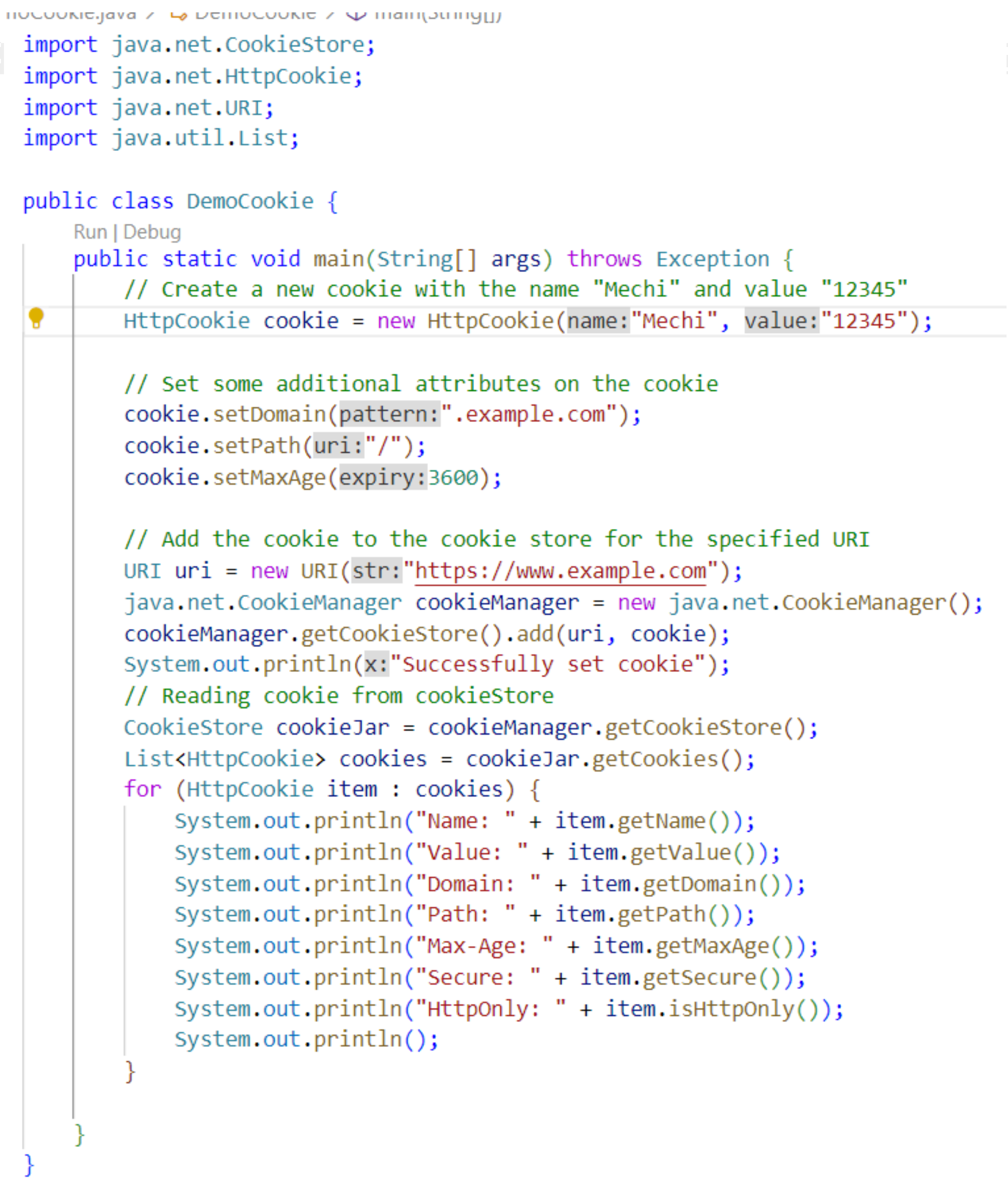


```java
public class HttpHeaderExample {
    Run | Debug
    public static void main(String[] args) throws IOException {
        String urlStr = "https://example.com";
        URL url = new URL(urlStr);

        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod(method:"GET");

        int responseCode = conn.getResponseCode();
        System.out.println("Response Code: " + responseCode);

        Map<String, List<String>> headers = conn.getHeaderFields();
        // conn.getHeaderFields();
        for (String key : headers.keySet()) {
            System.out.println(key + ": " + headers.get(key));
        }
    }
}
```

CookieManager AND CookieStore.

- The **java.net package** in Java provides two classes for managing cookies in HTTP connections: **CookieManager** and **CookieStore**.
- The **CookieManager** class is **responsible for storing** and **retrieving cookies** from a **CookieStore**. It is also responsible for managing the **cookie policy**, which determines **whether to accept or reject cookies based on various criteria such as expiration time, secure flag, domain, and path.**
- The CookieStore interface defines methods for **adding, getting, and removing cookies** from a cookie store. The **CookieManager class implements this interface** to provide a concrete implementation of the cookie store.

CookieManager AND CookieStore.



```java
import java.net.CookieStore;
import java.net.HttpCookie;
import java.net.URI;
import java.util.List;

public class DemoCookie {
    public static void main(String[] args) throws Exception {
        // Create a new cookie with the name "Mechi" and value "12345"
        HttpCookie cookie = new HttpCookie(name:"Mechi", value:"12345");

        // Set some additional attributes on the cookie
        cookie.setDomain(pattern:".example.com");
        cookie.setPath(uri:"/");
        cookie.setMaxAge(expiry:3600);

        // Add the cookie to the cookie store for the specified URI
        URI uri = new URI(str:"https://www.example.com");
        java.net.CookieManager cookieManager = new java.net.CookieManager();
        cookieManager.getCookieStore().add(uri, cookie);
        System.out.println(x:"Successfully set cookie");
        // Reading cookie from cookieStore
        CookieStore cookieJar = cookieManager.getCookieStore();
        List<HttpCookie> cookies = cookieJar.getCookies();
        for (HttpCookie item : cookies) {
            System.out.println("Name: " + item.getName());
            System.out.println("Value: " + item.getValue());
            System.out.println("Domain: " + item.getDomain());
            System.out.println("Path: " + item.getPath());
            System.out.println("Max-Age: " + item.getMaxAge());
            System.out.println("Secure: " + item.getSecure());
            System.out.println("HttpOnly: " + item.isHttpOnly());
            System.out.println();
        }

    }
}
```