▼ Unit 2

▼ Servlets:

- Servlet technology is used to create a web application
- There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse, etc.
- Servlet is an interface that must be implemented for creating any Servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
- Servlet is a web component that is deployed on the server to create a dynamic web page.

HTTP methods:

- The Hypertext Transfer Protocol (HTTP) is the data communication protocol used to establish communication between client and server
- It provides the standardized way for computers to communicate with each other.
- It uses the reliable TCP connections by default on TCP port 80.
- It is stateless means each request is considered as the new request.

The request sent by the computer to a web server, contains all sorts of potentially interesting information; it is known as HTTP requests.

HTTP Request	Description
GET	This method retrieves information from the given server using a given URI. GET request can retrieve the data. It can not apply other effects on the data..
POST	The POST request sends the data to the server. Asks the server to accept the body info attached. It is like GET request with extra info sent with the request.
HEAD	Asks for only the header part of whatever a GET would return. Just like GET but with no body.
TRACE	Asks for the loopback of the request message, for testing or troubleshooting.
PUT	Says to put the enclosed info (the body) at the requested URL.
DELETE	Says to delete the resource at the requested URL.
OPTIONS	Asks for a list of the HTTP methods to which the thing at the request URL can respond

Get vs. Post:

GET	POST
1) In case of Get request, only **limited amount of data** can be sent because data is sent in header.	In case of post request, **large amount of data** can be sent because data is sent in body.
2) Get request is **not secured** because data is exposed in URL bar.	Post request is **secured** because data is not exposed in URL bar.
3) Get request **can be bookmarked.**	Post request **cannot be bookmarked.**
4) Get request is **idempotent** . It means second request will be ignored until response of first request is delivered	Post request is **non-idempotent.**
5) Get request is **more efficient** and used more than Post.	Post request is **less efficient** and used less than get.

Web Server:

- A web server can be characterized as software that receives HTTP requests, processes them, and sends back responses
- A web server is a software program that, as its name implies, serves websites to users. It does this by responding to HTTP requests from the user's computer
- Traditional web servers only allow one transaction at once.
- Examples of popular web servers include Apache HTTP Server, Nginx, Microsoft Internet Information Services (IIS) etc.

Web Container:

- A web container, on the other hand, is an application that includes a web server as well as additional components like a servlet container, Enterprise JavaBean (EJB) container, and so forth.

- web container has many different components included in one package. This may improve your application's security, stability, and performance since everything is configured out of the box.
- Web containers typically support more than one transaction running at the same time
- Examples of web containers include Apache Tomcat, Red Hat JBoss, and IBM WebSphere Application Server.

In summary, while a web server primarily serves static content and handles HTTP requests, a web container executes dynamic web applications written in Java, managing servlets and JSP pages within the context of a Java-based web application.

Servlet Interface:

- **Servlet interface provides** common behavior to all the servlets.
- Servlet interface defines methods that all servlets must implement.
- Servlet interface needs to be implemented for creating any servlet (either directly or indirectly). It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods.

Methods of Servlet interface:

- public void init(ServletConfig config): initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
- public void service(ServletRequest request,ServletResponse response): provides response for the incoming request. It is invoked at each request by the web container.
- public void destroy(): is invoked only once and indicates that servlet is being destroyed.
- public ServletConfig getServletConfig( ): Returns a ServletConfig object that contains any initialization parameters
- public String getServletInfo( ): Returns a string describing the servlet.

GenericServlet:

- **GenericServlet** is an abstract class in Java that provides a generic, protocol-independent implementation of the **Servlet** interface. It's designed to be subclassed to create servlets that handle specific protocols like HTTP, FTP, or SMTP.
- GenericServlet class can handle any type of request so it is protocol-independent.

Methods of GenericServlet interface:

- public void init(ServletConfig config): initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
- public void service(ServletRequest request,ServletResponse response): provides response for the incoming request. It is invoked at each request by the web container.
- public void destroy(): is invoked only once and indicates that servlet is being destroyed.
- **public String getServletInfo()** returns information about servlet such as writer, copyright, version etc.
- **public ServletConfig getServletConfig()** returns the object of ServletConfig.

HttpServlet Class:

- **HttpServlet** is an abstract class in Java that extends **GenericServlet** and provides specific support for HTTP protocol-based servlets. It's designed to handle HTTP requests and generate HTTP responses, making it suitable for creating web applications that communicate over the HTTP protocol.
- **HttpServlet** extends the servlet lifecycle methods provided by **GenericServlet** (**init()**, **service()**, **destroy()**), but it also introduces additional methods specifically for handling HTTP requests. These methods include **doGet()**, **doPost()**, **doPut()**, **doDelete()**, **doOptions()**, **doHead()**, and **doTrace()**

Methods of HttpServlet class:

- **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
- **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.

Life Cycle of Servlet Class:

The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

- **Servlet class is loaded:**

  The servlet class is loaded when the first request for the servlet is received by the web container.

- **Servlet instance is created:**
  The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

- **init method is invoked:**
  The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet.

- **service method is invoked:**
  The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method.

- **destroy method is invoked:**
  The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc.

ServletConfig:

- An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file.
- The core advantage of ServletConfig is that you don't need to edit the servlet file if information is modified from the web.xml file.
- The life cycle of the ServletConfig object is specific to each servlet, and any data declared within it is shared only within that servlet.
- The scope of the ServletConfig object is limited to a particular servlet
- The server creates and removes the ServletConfig object; it cannot be created or removed by the programmer.

How to get the object of ServletConfig

1. **getServletConfig() method** of Servlet interface returns the object of ServletConfig.

```
ServletConfig config=getServletConfig();
//Now we can call the methods of ServletConfig interface
```

ServletContext:

- There is only one ServletContext object per web application.
- If any information is shared to many servlet, it is better to provide it from the web.xml file using the **<context-param>** element.
- The ServletContext object can be used to get configuration information from the web.xml file.
- The ServletContext object can be used to set, get or remove attribute from the web.xml file.

How to get the object of ServletContext interface

1. **getServletContext() method** of ServletConfig interface returns the object of ServletContext.
2. **getServletContext() method** of GenericServlet class returns the object of ServletContext.

Servlet Communication:

- The communication between the Java servlets is known as Servlet communication.
- It is sending users request, and the response object passed to a servlet to another servlet.

- We are using the method getParameter(), which is basically used to get the input from user value in a specified field.
- So, when Servlet request the object is passed from one servlet to another servlet, then we will use this method to get the input that the user has given in an HTML/JSP form.
- **getServletContext()**:
  It gets the ServletContext object. It is an interface, or we say it may be an implementation class of ServletContext
- **getRequestDispatcher():**

  getRequestDispatcher() is a method of the ServletContext interface. It returns the RequestDispatcher object with some implementation class is written. The purpose of this getRequestDispatcher() method is to pass the request to other servlets.

Session Management Techniques:

- **Session Tracking** is a way to maintain state (data) of an user. It is also known as **session management** in servlet.
- Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.
- There are four techniques used in Session tracking:

1. **Cookies:**
   - A **cookie** is a small piece of information that is persisted between the multiple client requests.
   - By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user
   - **Non-persistent cookie:** It is **valid for single session** only. It is removed each time when user closes the browser.
   - **Persistent cookie:** It is **valid for multiple session** . It is not removed each time when user closes the browser. It is removed only if user logout or signout.
   - **javax.servlet.http.Cookie** class provides the functionality of using cookies.

2. **Hidden Form Field:**
   - In case of Hidden Form Field **a hidden (invisible) textfield** is used for maintaining the state of an user.
   - In such case, we store the information in the hidden field and get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.

3. **URL Rewriting:**
   - In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource. We can send parameter name/value pairs using the following format:

     `url?name1=value1&name2=value2&??`
   - A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&).
   - When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a Servlet, we can use getParameter() method to obtain a parameter value.

4. **HttpSession:**
   - In such case, container creates a session id for each user.The container uses this id to identify the particular user
   - An object of HttpSession can be used to perform two tasks:
     1. bind objects
     2. view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.

▼ JSP:

- **JSP** technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet
- A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development.

- In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

JSP Life Cycle:

Following steps are involved in the JSP life cycle:

1. **Translation of JSP page to Servlet:** This is the first step of the JSP life cycle. This translation phase deals with the Syntactic correctness of JSP. For Example, test.jsp file is translated to test.java.
2. **Compilation of JSP page(Compilation of JSP into test.java):** Here the generated java servlet file (test.java) is compiled to a class file (test.class).
3. **Classloading:** The classloader loads the Java class file into the memory. The loaded Java class can then be used to serve incoming requests for the JSP page.
4. **Instantiation(Object of the generated Servlet is created):** Here an instance of the class is generated
5. **Initialization(jspInit() method is invoked by the container):** jspInit() method is called only once during the life cycle immediately after the generation of the Servlet instance from JSP.
6. **Request processing(_jspService()is invoked by the container):** _jspService() method is used to serve the raised requests by JSP.
7. **JSP Cleanup (jspDestroy() method is invoked by the container)**: In order to remove the JSP from the use by the container or to destroy the method for servlets jspDestroy()method is used.

JSP Implicit Objects:

1. **JSP out implicit object:** For writing any data to the buffer, JSP provides an implicit object named out. It is the object of JspWriter
2. **JSP request implicit object:** The **JSP request** is an implicit object of type HttpServletRequest i.e. created for each jsp request by the web container.
3. **JSP response implicit object:** In JSP, response is an implicit object of type HttpServletResponse. The instance of HttpServletResponse is created by the web container for each jsp request. It can be used to add or manipulate response such as redirect response to another resource, send error etc.
4. **JSP config implicit object:** In JSP, config is an implicit object of type *ServletConfig*. This object can be used to get initialization parameter for a particular JSP page. The config object is created by the web container for each jsp page
5. **JSP application implicit object:** In JSP, application is an implicit object of type *ServletContext*. The instance of ServletContext is created only once by the web container when application or project is deployed on the server.
6. **JSP session implicit object:** In JSP, session is an implicit object of type HttpSession.The Java developer can use this object to set,get or remove attribute or to get session information.
7. **JSP pageContext implicit object: I**n JSP, pageContext is an implicit object of type PageContext class.The pageContext object can be used to set,get or remove attribute from one of the following scopes: page, request, session, application
8. **JSP page implicit object:** In JSP, page is an implicit object of type Object class.This object is assigned to the reference of auto generated servlet class.
9. **JSP exception implicit object:** In JSP, exception is an implicit object of type java.lang.Throwable class. This object can be used to print the exception. But it can only be used in error pages.

JSP Directives:

JSP directives are the elements of a JSP source code that guide the web container on how to translate the JSP page into its respective servlet.

There are three different JSP directives available:

- **Page Directive:** The page directive defines attributes that apply to an entire JSP page.

  Different properties/attributes:

  - **import**: This tells the container what packages/classes are needed to be imported
  - **contentType**: This defines the format of data that is being exchanged between the client and the server
  - **info**: Defines the string which can be printed using the 'getServletInfo()' method.

- **buffer**: Defines the size of the buffer that is allocated for handling the JSP page
- **language**: Defines the scripting language used in the page
- **isELIgnored**: This attribute tells if the page supports expression language
- **errorPage**: Defines which page to redirect to, in case the current page encounters an exception.

- **Include directive :** JSP include directive is used to include other files into the current jsp page. These files can be html files, other sp files etc. The advantage of using an include directive is that it allows code re-usability.

- **Taglib Directive :** The taglib directive is used to mention the library whose custom-defined tags are being used in the JSP page.

JSP Scripting Elements:

The scripting elements provides the ability to insert java code inside the jsp. There are three types of scripting elements:

- **JSP scriptlet tag:**
  - A scriptlet tag is used to execute java source code in JSP.
  - Syntax: `<%  java source code %>`

- **JSP expression tag:**
  - The code placed within **JSP expression tag** is *written to the output* stream of the response.
  - So you need not write out.print() to write data. It is mainly used to print the values of variable or method.
  - Syntax: `< %=  statement % >`

- **JSP declaration tag:**
  - The **JSP declaration tag** is used *to declare fields and methods*.
  - The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet.
  - So it doesn't get memory at each request.
  - Syntax: `< %!  field or method declaration % >`