WEB TECH NOTES :

unit 1 :

History of the Web and Internet

- **ARPANET (1969)**: First network to implement the TCP/IP protocol.
- **Tim Berners-Lee (1991)**: Created the World Wide Web (WWW) with HTML, HTTP, and web browsers.
- **Evolution**:
  - Web 1.0: Static websites.
  - Web 2.0: Interactive and dynamic content (social media, blogs).
  - Web 3.0: Semantic web, AI-driven personalization, decentralized systems.

Development Approaches :

- **Agile**: Iterative approach focusing on collaboration and flexibility.
- **Waterfall**: Linear approach with sequential phases (planning, design, development, testing, and deployment).

Protocols Governing the Web

- **HTTP/HTTPS (HyperText Transfer Protocol)**: Rules for web page requests and responses.
  - **HTTPS**: Secure version using SSL/TLS for encryption.
- **FTP (File Transfer Protocol)**: For transferring files between a client and server.
- **TCP/IP**: Underlying communication protocol suite of the internet.
- **DNS (Domain Name System)**: Translates domain names to IP addresses.

XML (eXtensible Markup Language)

- **Document Type Definition (DTD)**: Defines the structure of an XML document.
  - Specifies the allowed elements and attributes.
- **XML Schemas**: More powerful alternative to DTD, using XML to describe document structure.
- **Object Models**:
  - **DOM (Document Object Model)**: Represents the XML as a tree structure, allowing access and modification.
  - **SAX (Simple API for XML)**: Event-driven API to parse XML data.
- **Presenting and Using XML**:
  - XML can be used with XSLT (eXtensible Stylesheet Language Transformations) to present data.
  - Data can be used in web applications for structured data storage.

Using XML Processors

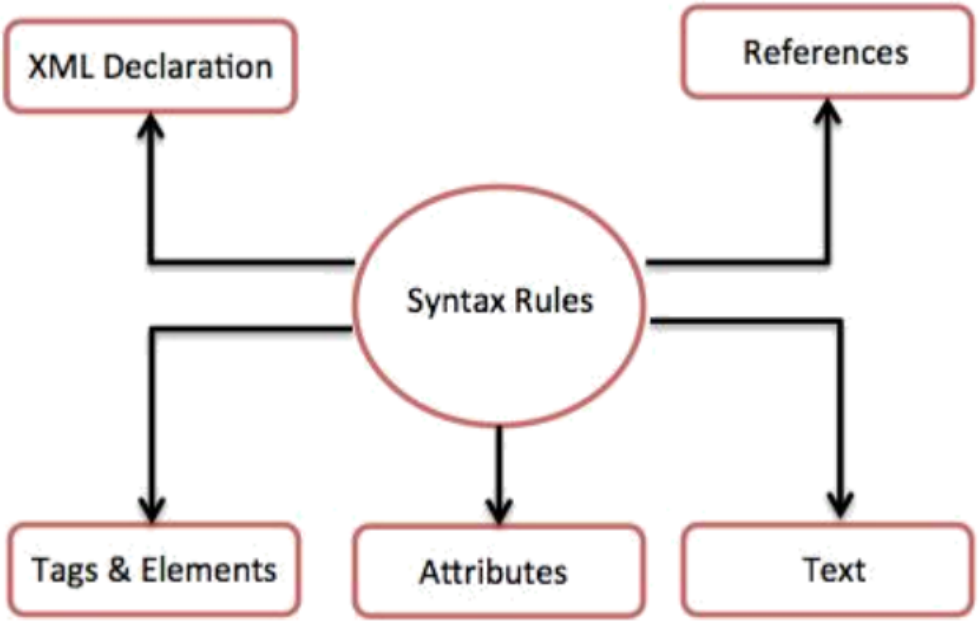
- **DOM Processor**: Loads entire XML into memory, allows read/write operations (suitable for smaller documents).
- **SAX Processor**: Event-driven, more memory efficient (best for large XML documents).

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE employees [
    <!ELEMENT employees (employee+)>
    <!ELEMENT employee (name, designation, salary)>
    <!ATTLIST employee id ID #REQUIRED>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT designation (#PCDATA)>
    <!ELEMENT salary (#PCDATA)>
]>

<employees>
```

```
    <employee id="E001">
        <name>John Doe</name>
        <designation>Software Engineer</designation>
        <salary>75000</salary>
    </employee>
    <employee id="E002">
        <name>Jane Smith</name>
        <designation>Project Manager</designation>
        <salary>90000</salary>
    </employee>
    <employee id="E003">
        <name>Emily Davis</name>
        <designation>UX Designer</designation>
        <salary>65000</salary>
    </employee>
</employees>
```

types of data in XML :

- PCDATA (Parsed Character Data) :
  This represents the text data that is parsed by the XML parser. It can include text, but not other markup elements.
- CDATA (Character Data) :
  This represents text that should
  not be parsed by the XML parser. Any content within a CDATA section is treated as literal text, meaning it won't be parsed for any XML tags or entities.



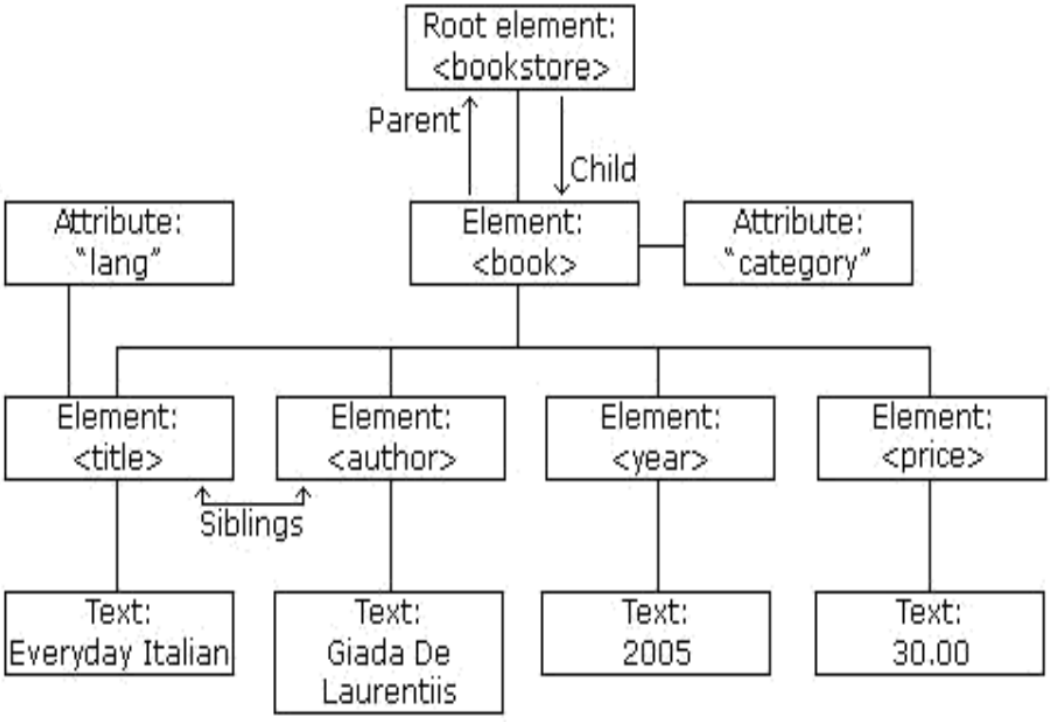
difference btw HTML and XML :

- XML was designed to carry data - with focus on what data is
- HTML was designed to display data - with focus on how data looks
- XML tags are not predefined like HTML tags
- XML documents form a tree structure that starts at "the root" and branches to
  "the leaves".

syntax rules for XML :

- The XML declaration is case sensitive and must begin with "<?xml>" where "xml" is written in lower-case.
- If document contains XML declaration, then it strictly needs to be the first statement of the XML document.
- The XML declaration strictly needs be the first statement in the XML document.
- An HTTP protocol can override the value of encoding that you put in the XML
  declaration.

XML tree structure :



well formed XML :

- XML documents must have a root element
- XML elements must have a closing tag
- XML tags are case sensitive
- XML elements must be properly nested
- XML attribute values must be quoted

XML object models : (using xml and presenting) :

XML Object Models allow us to interact with and manipulate XML data programmatically.

- **DOM (Document Object Model)**
- **SAX (Simple API for XML)**.

Document Object Model (DOM)

- The DOM parser reads the entire XML document, converts it into a tree structure in memory, and provides methods to traverse, search, and manipulate the document.
- You can add, remove, or modify nodes in this structure.

Advantages:

- Allows random access to any part of the XML document.
- Provides rich methods for searching and modifying the document.

Disadvantages:

- Loads the entire document into memory, which can be inefficient for large XML files.

DOM representation :

```
library
 └── book (id="B001")
      ├── title: "XML for Beginners"
      └── author: "John Doe"
 └── book (id="B002")
      ├── title: "Advanced XML"
      └── author: "Jane Smith"
```

Simple API for XML (SAX)

- **Definition**: SAX is an event-driven, stream-based approach to parsing XML. Unlike DOM, SAX does not load the entire document into memory; instead, it reads the document sequentially, triggering events as it encounters different parts of the document (start tags, end tags, etc.).

Advantages:

- More memory efficient since it doesn't store the whole document in memory.
- Suitable for processing large XML files.

Disadvantages:

- Not suitable for random access; the parser reads in a forward-only manner.
- You cannot traverse backward or modify the document as it’s processed sequentially.

Feature	DOM	SAX
Access Type	Random access, full document	Sequential access, no random access
Memory Usage	High (entire document is loaded)	Low (only parts of the document are processed)
Modification	Can modify the document	Cannot modify the document
Speed	Slower for large files	Faster for large files

Client-Server Architecture

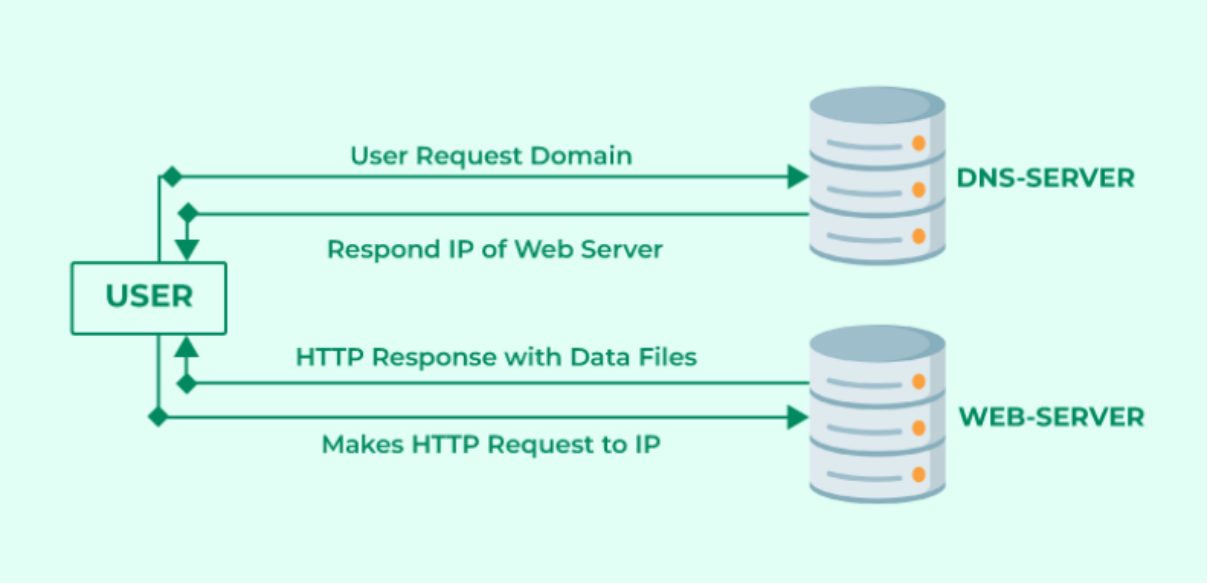
Client-server architecture is a network model where **clients** (users or devices) request resources or services from a **server**, which processes the request and returns the response. The architecture enables the distribution of tasks, workloads, and services between the client and server.

1. **Client**:
    - A client is a device or program that initiates requests to the server.
    - **Examples**: Web browsers (Google Chrome, Firefox)
2. **Server**:
    - A server is a powerful machine or software program that listens to client requests, processes them, and sends back responses.
    - Hosts services or resources (such as web pages, files, data) that clients access.
    - **Examples**: Web servers (Apache, Nginx), database servers (MySQL, PostgreSQL), file servers.

working :

- The **client** initiates a connection to the server by sending a request, such as accessing a website or fetching data from a database.

- The **server** processes the client's request, retrieves the required data or service, and sends the response back to the client.
- This communication usually happens via specific protocols, such as:
  - **HTTP/HTTPS**: Used for web-based communications.
  - **FTP**: Used for file transfer.
  - **SMTP**: Used for email.



Types of Client-Server Architecture

1. **Two-tier architecture** : only client and server (eg : simple user and database connection)
2. **Three-tier architecture** : client - application server (for logic) - database
3. **N-tier architecture**: Involves more than three layers, usually adding more specific servers for different functions (like caching, load balancing, etc.).

ADVANTAGES :

- **Centralized Resources**
- **Scalability**
- **Specialization**
- **Data Integrity**

DISADVANTAGES :

- **Single Point of Failure**
- **Server Overload**
- **Cost**

UNIT - 02 : CSS (CASCADING STYLE SHEETS)

BOILER PLATE :

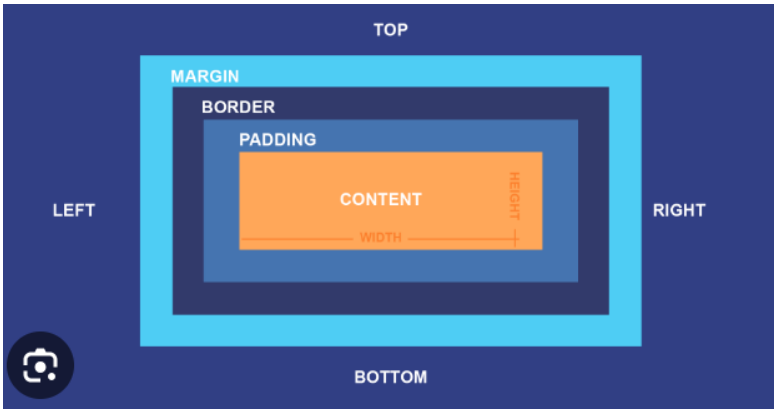
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initia
l-scale=1.0">
  <title>CSS Styling Example</title>
  <style>
    /* Background and text formatting */
    body {
      background-color: lightyellow;
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;
    }
  </style>
</head>
<body>
  <div id="header">My Webpage</div>

  <div class="content">
    <h1>Welcome</h1>
```

```
    <p>This is a paragraph styled using CSS. Below is an ex
ample of a table and a list.</p>
    <table>
      <tr>
        <th>Item</th>
        <th>Price</th>
      </tr>
      <tr>
        <td>Apple</td>
        <td>$1</td>
      </tr>
      <tr>
        <td>Banana</td>
        <td>$2</td>
      </tr>
    </table>

    <!-- List Example -->
    <ul>
      <li>List Item 1</li>
      <li>List Item 2</li>
    </ul>

  </div>
</body>
</html>
```

TYPES OF CSS :

- inline css
- internal css
- external css

BOX MODEL IN CSS :



Positioning in CSS

The `position` property specifies how elements are placed on the webpage.

- `static`: Default value; elements follow the normal flow.
- `relative`: Positioned relative to its normal position.
- `absolute`: Positioned relative to the nearest positioned ancestor.
- `fixed`: Positioned relative to the viewport.
- `sticky`: Element is static until a scroll threshold is met.

Pseudo-classes

Pseudo-classes are used to define the state of an element, such as when a user hovers over it, clicks it, or when an input field is focused.

- `:hover`: Applied when the user hovers over an element.
- `:focus`: Applied when an input is focused.
- `:nth-child()`: Targets elements based on their position in a parent.

Image Sprites are used to combine multiple images into a single image file, improving performance by reducing the number of server requests.

Attribute Selectors allow you to target HTML elements based on their attributes, providing a flexible way to style elements dynamically based on their properties.

Attribute Selectors in CSS

An **attribute selector** is a CSS selector that targets elements based on their attributes or attribute values. This allows you to apply styles to elements that match certain criteria.

```
/* Select elements with the "type" attribute set to "text"
*/
input[type="text"] {
  border: 1px solid blue;
}

/* Select anchor tags with an "href" attribute that starts
with "https" */
a[href^="https"] {
  color: green;
}

/* Select images with an "alt" attribute containing the wor
d "logo" */
img[alt*="logo"] {
  border: 2px solid black;
}
```

AJAX :

AJAX (Asynchronous JavaScript and XML) is a technique used in web development to send and receive data from a server asynchronously without interfering with the display and behavior of the existing page. This allows web applications to retrieve data from the server in the background and update parts of a webpage without needing to reload the entire page.

fetaures :

- Asynchronous
- Partial Page Updates
- Data Formats : can handle json, html , plain text etc

`XMLHttpRequest` is the built-in JavaScript object that allows you to make HTTP requests to the server.

AJAX BOILER PLATE :

```
function fetchData(url) {
    var xhr = new XMLHttpRequest(); // Create a new XMLHttp
Request object

    xhr.open('GET', url, true); // Configure the request (G
ET method, URL)

    xhr.onload = function() { // Set up a function to handl
e the response
        if (xhr.status === 200) { // Check if the request w
as successful
            console.log(xhr.responseText); // Log the respo
nse
        } else {
            console.error('Error: ' + xhr.status); // Log a
ny errors
        }
    };

    xhr.send(); // Send the request
}

// Example usage:
fetchData('https://api.example.com/data');
```

HTTP STATUS CODES :

HTTP status codes are standardized codes sent by a server in response to a client's request to indicate the outcome of the request. They are divided into several categories based on their first digit:

- **1xx**: Informational responses.

- **2xx**: Successful responses.
- **3xx**: Redirection messages.
- **4xx**: Client error responses.
- **5xx**: Server error responses.

- **100 Continue**
- **101 Switching Protocols**
- **200 OK**
- **201 Created**
- **204 No Content**

- **301 Moved Permanently**
- **302 Found**
- **304 Not Modifie**

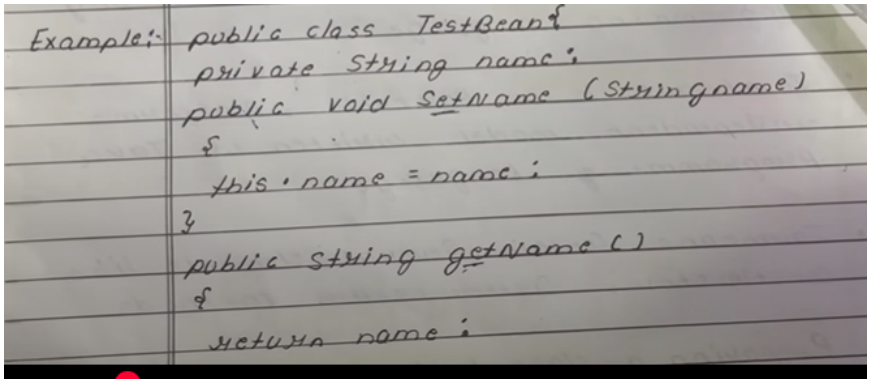
- **400 Bad Request**
- **401 Unauthorized**
- **403 Forbidden**
- **404 Not Found**

- **500 Internal Server Error**
- **502 Bad Gateway**
- **503 Service Unavailable**

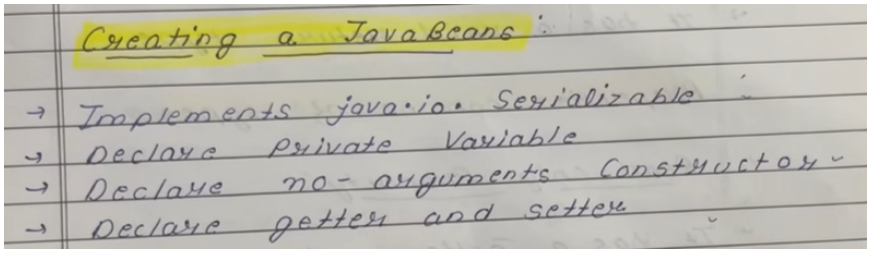
WEB TECHNOLOGY UNIT 4 :

java beans :

- reusable software components
- many objs encapsulated in one obj so can be accessed from multiple places
- for easy mantainence
- portable , platform indep
- default constructor (w/o arguments)
- has getter setter methods
- should be serializable



creating java beans :

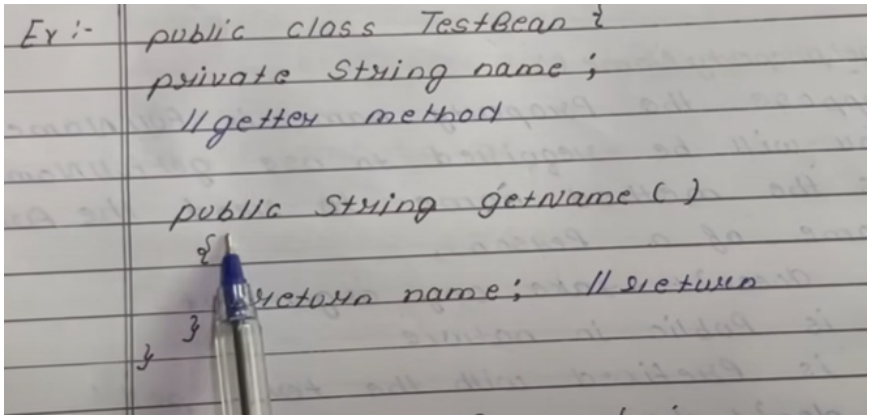


java beans properties :

- getpropertyname() - (not void, prefix get, public)
- setpropertyname() - (void, prefix set, public)

more properties :

- read only property : has only get method

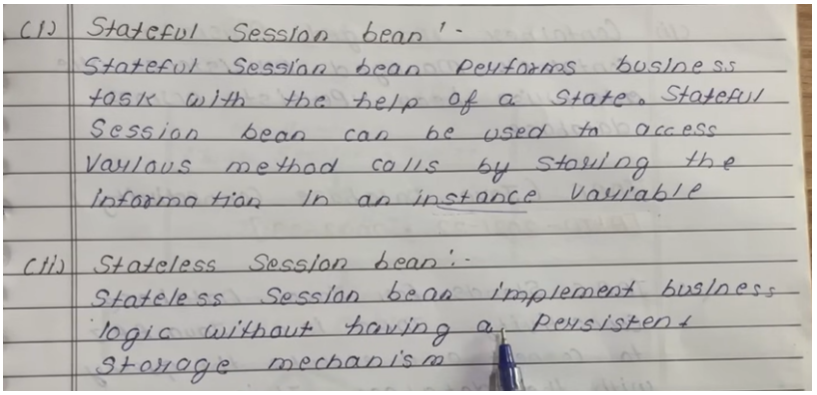


- write only property : contain only set method
- read and write property : has both get set
- indexed properties : (get set entire array)

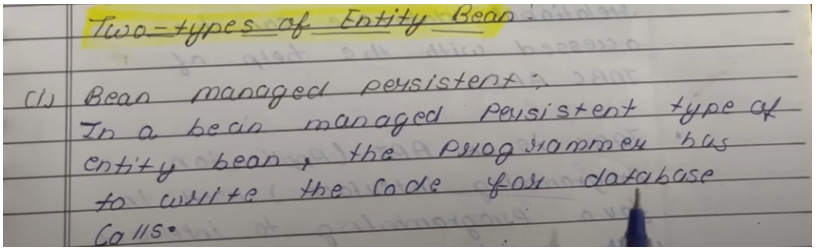
Enterprise Java Beans (EJB):

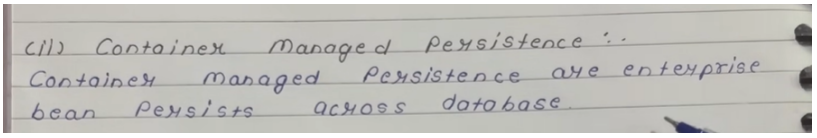
Definition: Server-side components that encapsulate **business logic of an application**

- **Types of EJB:**
  - **Session Beans**: Handle business logic and workflow
    logic that can be invoked by local or remote webservice client.
    types :



  - **Entity Beans** : Represent persistent data (replaced by java persistent API now)
    types :
    —



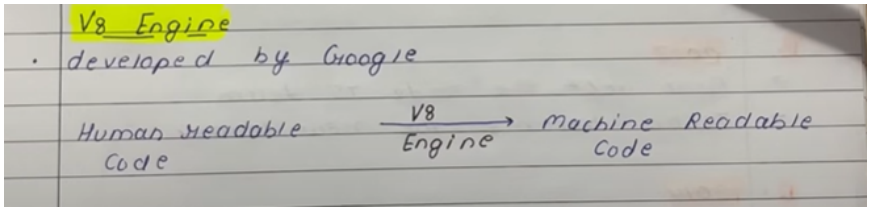


    - **Message-Driven Beans** : Process asynchronous messages, INVOKED BY PASSING MESSAGE

- **Key Features:**
    - Transaction Management
    - Security
    - Concurrency Control
    - Resource Pooling
- **Advantages:**
    - Simplified Development
    - Scalability
    - Reusability
    - Container-Managed Services

NODE JS :

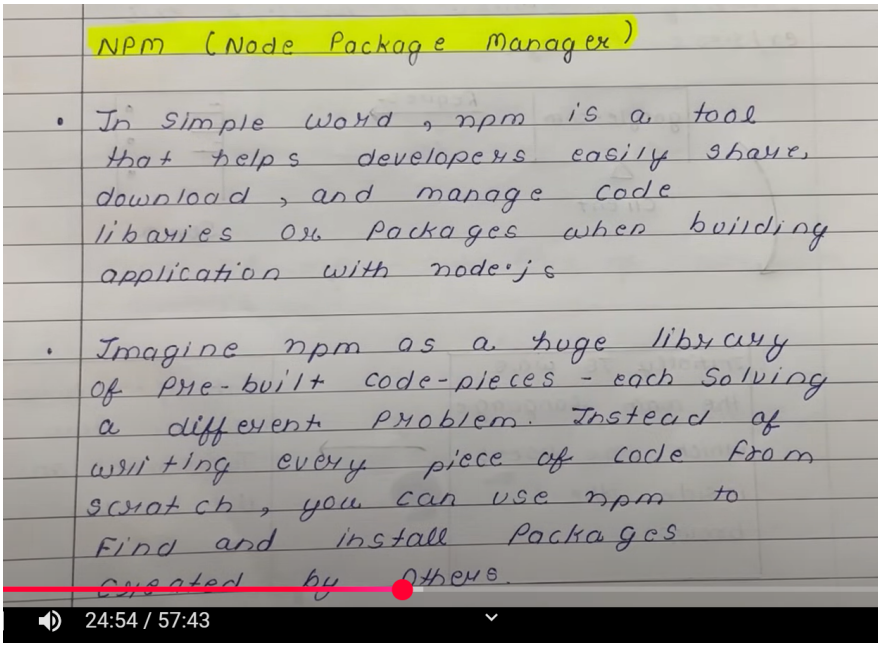
- allow FE and BE both
- dev by ryan dhal
- javascript runtime build on chrome V8 engine
- cross platform



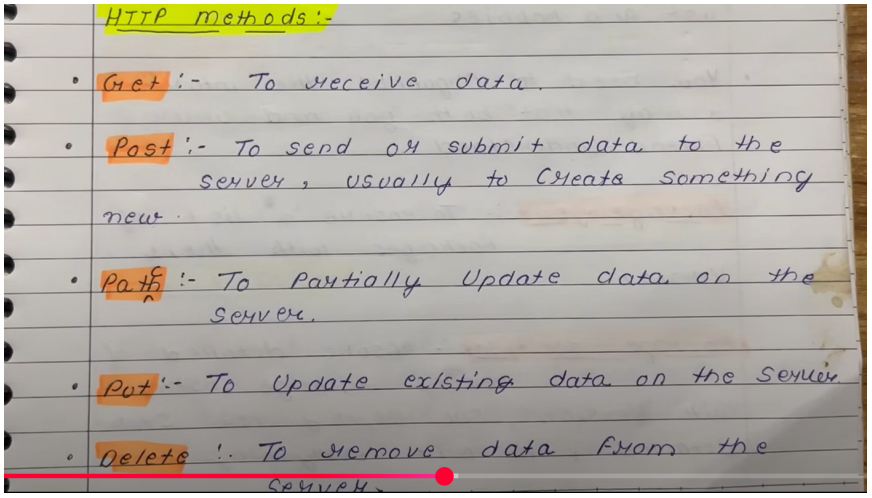
- allows REPL :

  - READ EVALUATE PRINT LOOP

NPM : node package manager :



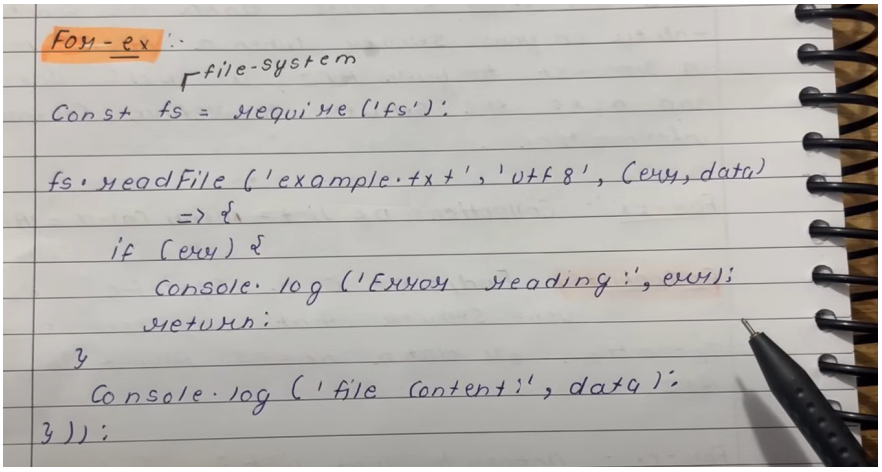
SERVER :



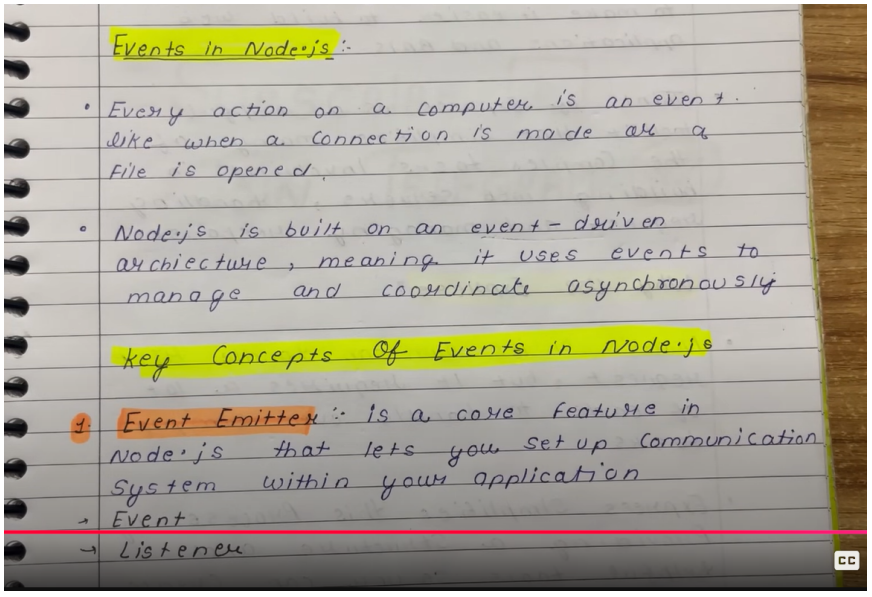
CALLBACK :

for tasks that take time like api calll and db queries

callback function : a function that is passed as argument to another function



EVENTS IN NODE JS :



express js

key features :

- routing
- middleware
- easy integrationwith db
- error handling

```
// Import Express
const express = require('express');
const app = express();

// Define a route
app.get('/', (req, res) => {
    res.send('Hello, World!');
});

// Start the server
const PORT = 3000;
app.listen(PORT, () => {
```

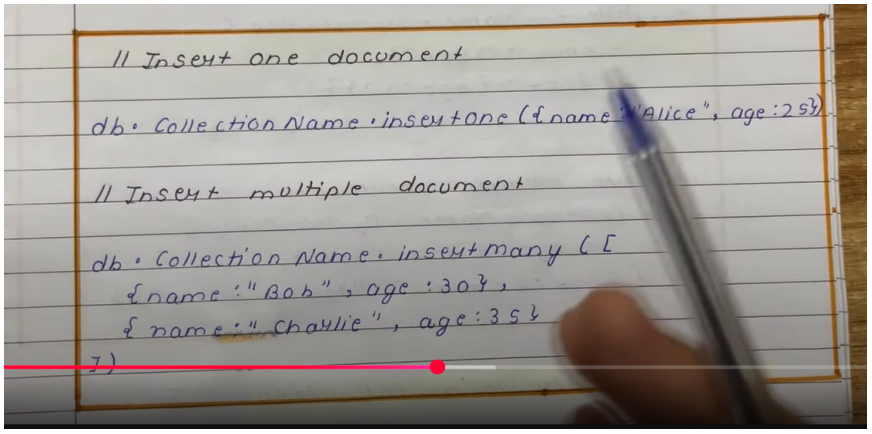
```
    console.log(`Server is running on http://localhost:${PORT
});
```

MONGO DB :

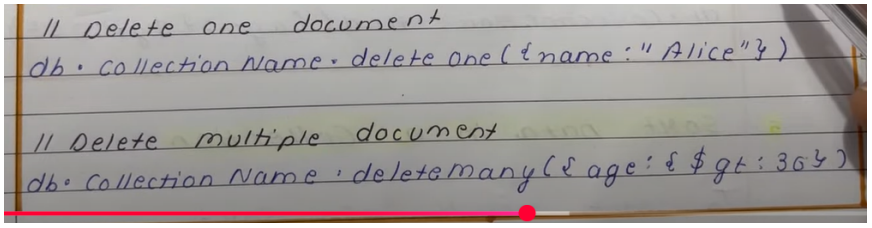
CREATE A COLLECTION IN MDB

- open mongo db
- choose ur db
- create collection
- check your collection (optional)

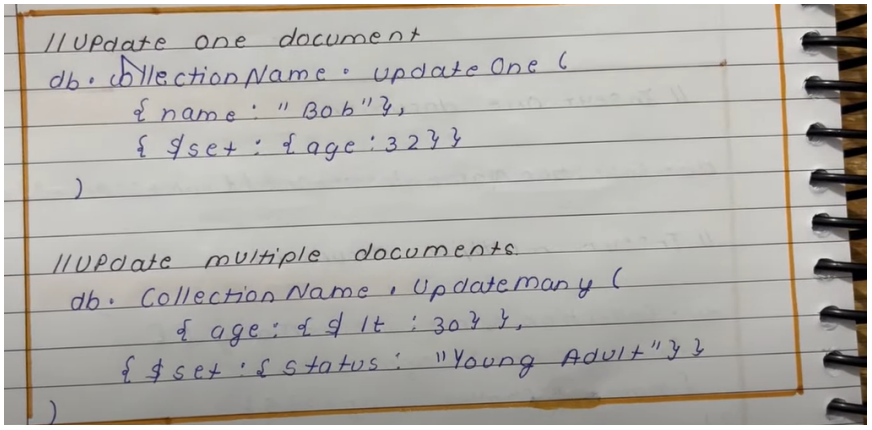
INSERT DATA TO COLLECTION :



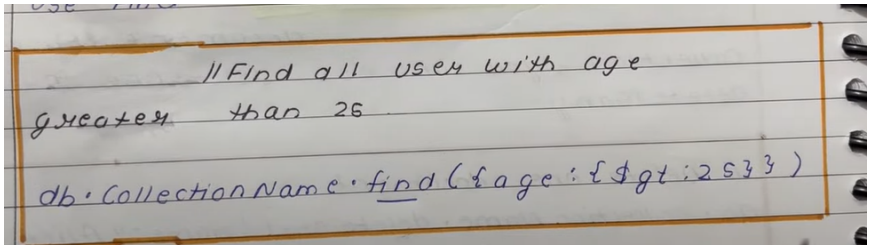
DELETE



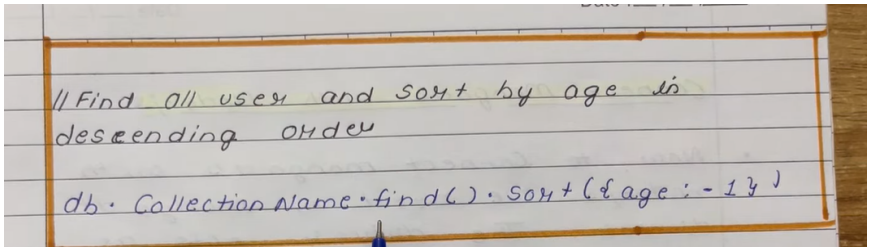
UPDATE



QUERY DATA :



SORT DATA :



node + mongo :

```
// Import MongoDB client
const { MongoClient } = require('mongodb');

// MongoDB connection URL and database name
const uri = "mongodb://127.0.0.1:27017";
const dbName = "testdb";

// Create a client instance
const client = new MongoClient(uri);
```

```
async function main() {
    try {
        // Connect to MongoDB
        await client.connect();
        console.log("Connected to MongoDB!");

        // Access the database and collection
        const db = client.db(dbName);
        const collection = db.collection("testCollection");

        // Insert a document
        const result = await collection.insertOne({ name: "Jo
        console.log("Document inserted:", result.insertedId);

        // Query the collection
        const query = { name: "John Doe" };
        const user = await collection.findOne(query);
        console.log("Found document:", user);

        // Update the document
        const updateResult = await collection.updateOne(query
        console.log("Document updated:", updateResult.modifie

        // Delete the document
        const deleteResult = await collection.deleteOne(query
        console.log("Document deleted:", deleteResult.deleted
    } catch (error) {
        console.error("Error:", error);
    } finally {
        // Close the connection
        await client.close();
    }
}

// Run the main function
main();
```

using mongoose :

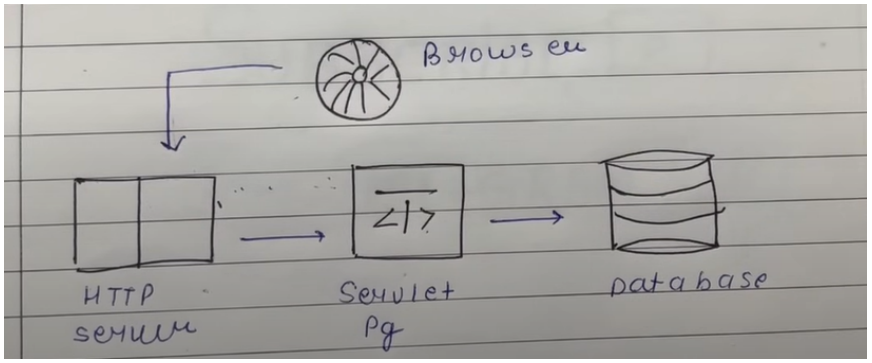
```
// Import Mongoose
const mongoose = require('mongoose');

// MongoDB connection URL
const uri = "mongodb://127.0.0.1:27017/testdb";

// Define a schema
const userSchema = new mongoose.Schema({
    name: String,
    age: Number,
});

// Create a model
const User = mongoose.model("User", userSchema);

async function main() {
    try {
        // Connect to MongoDB
        await mongoose.connect(uri, { useNewUrlParser: true,
        console.log("Connected to MongoDB!");

        // Create a new user document
        const newUser = new User({ name: "John Doe", age: 30
        const savedUser = await newUser.save();
        console.log("Document inserted:", savedUser);

        // Read the user document
        const user = await User.findOne({ name: "John Doe" })
        console.log("Found document:", user);

        // Update the user document
        const updatedUser = await User.findOneAndUpdate(
            { name: "John Doe" },
            { age: 31 },
            { new: true } // Return the updated document
        );
```

```
        console.log("Document updated:", updatedUser);

        // Delete the user document
        const deletedUser = await User.deleteOne({ name: "Joh
        console.log("Document deleted:", deletedUser.deletedC
    } catch (error) {
        console.error("Error:", error);
    } finally {
        // Close the connection
        mongoose.connection.close();
    }
}

// Run the main function
main();
```

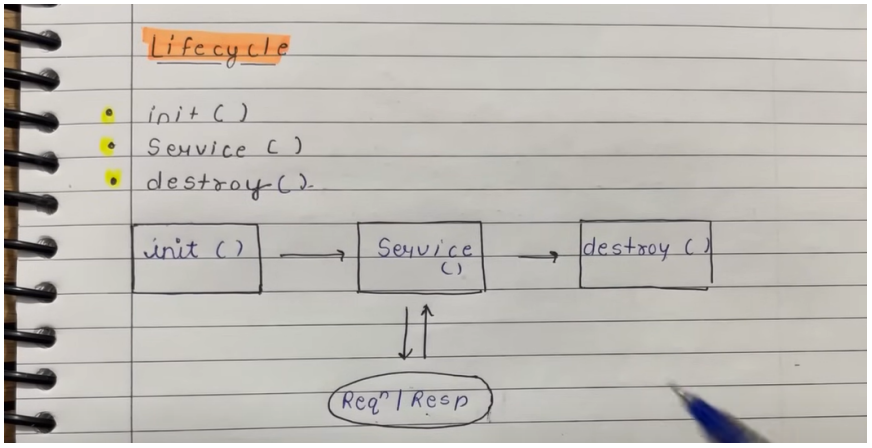
UNIT - 05

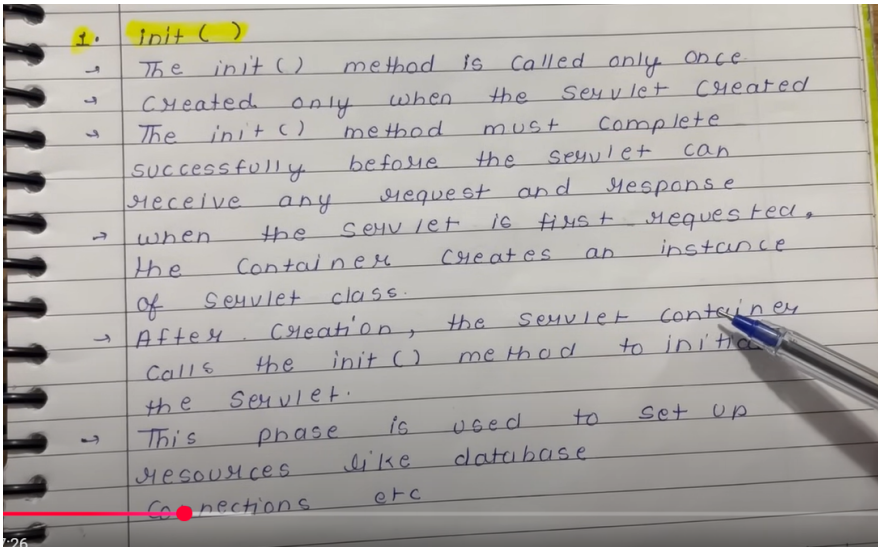
SERVLETS :

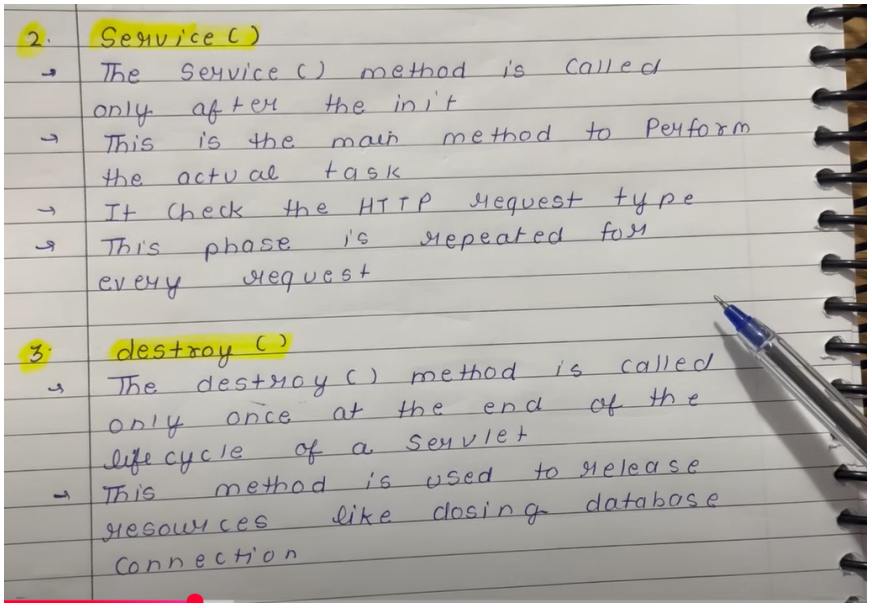
- java programs that run on java web servers
- handles request response
- works serverside
- helps to create dynamic web content

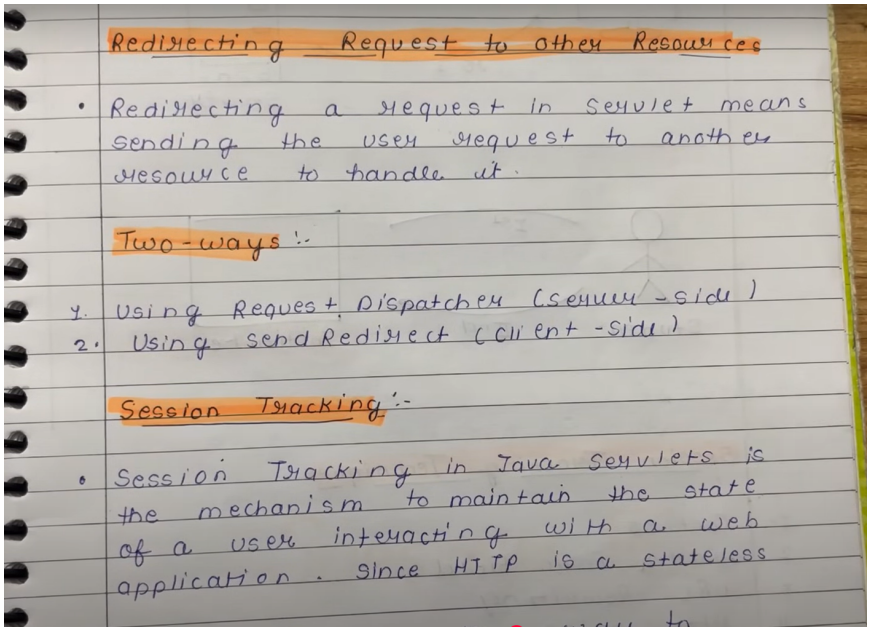


lifecycle of servelets :

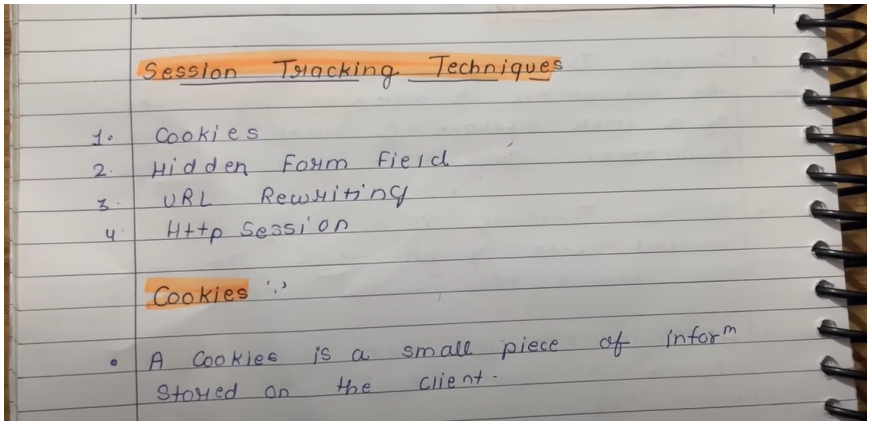


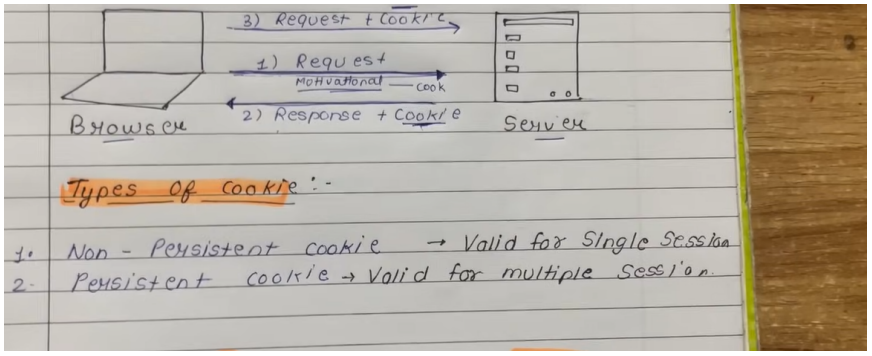




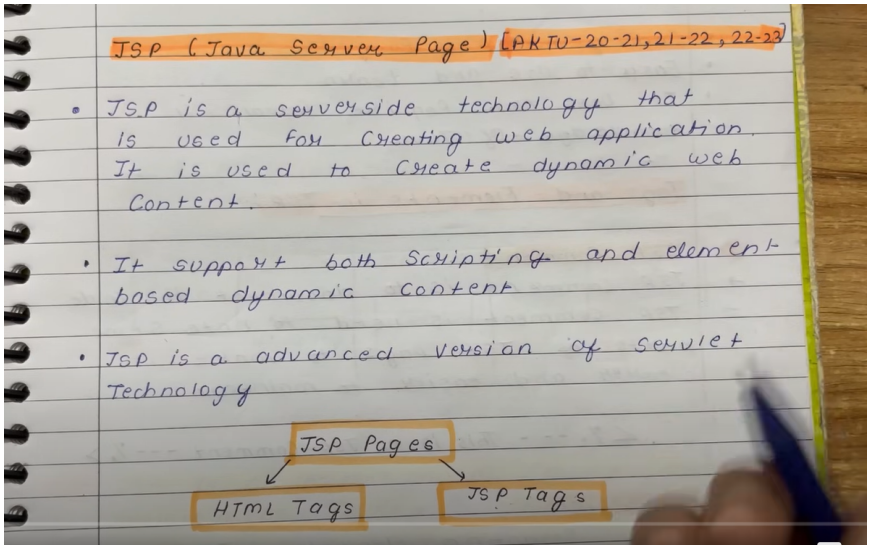


SESSION TRACKING/MANAGEMENT





JSP - JAVA SERVER PAGE :



- makes pages dynamic
- tags are reusable
- reduces hard code
- jsp is easy
- java in html

TAGS AND ELEMENTS IN JSP :

1. **JSP Comment** :-

→ JSP Comment is to document the code

→ JSP comment is used to note some parts of JSP pags to make it clearer and easier to maintain

// ** <%-- This is a JSP Comment --%>

2. **Expression** :-

→ Basic Scripting elements in JSP

→ expression is used to insert value directly to the output

<%= Expression %>

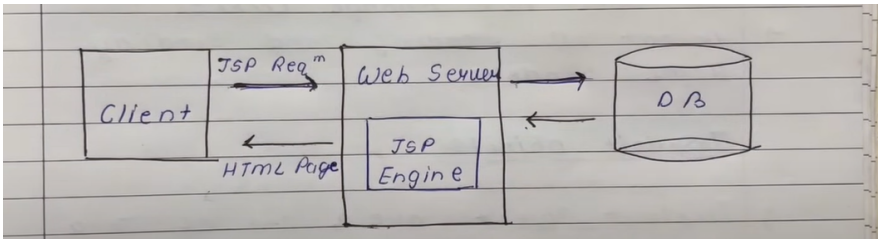
3. **Scriptlet tag** :-

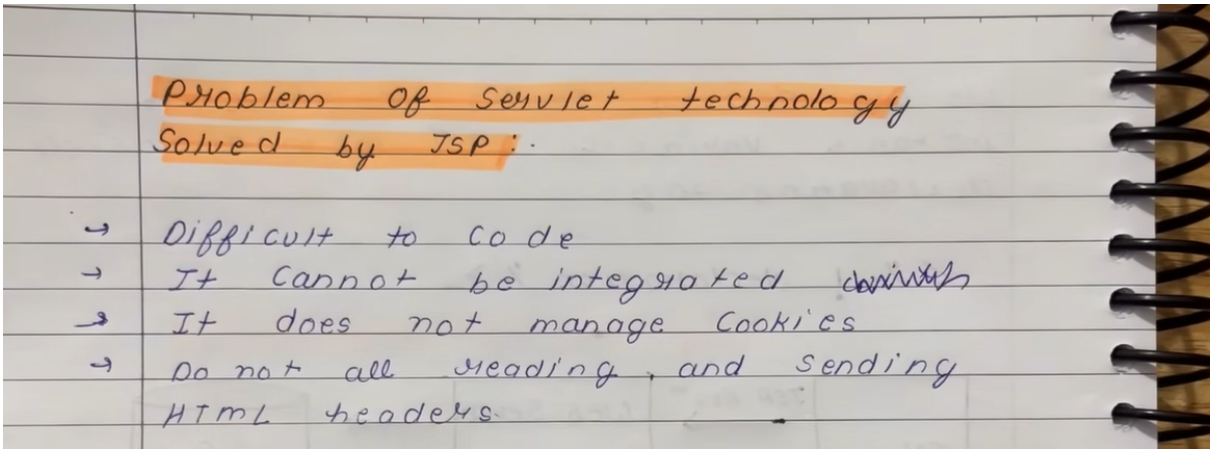
→ Insert any plain Java code inside a Scriptlet

<% Java-Code %>

We can declare static member, instance variable and methods inside declaration tag.

<%! declaration %>




Problem of Servlet technology
Solved by JSP:
Difficult to code
It cannot be integrated
It does not manage Cookies
Do not all reading, and sending
HTML headers