



VidyaJyothi Institute of Technology

(Approved by AICTE, New Delhi, Accredited by NAAC, Permanently Affiliated to JNTUH, Hyderabad)

An Autonomous Institution

Aziznagar Gate, ChilkurBalaji Road,
Hyderabad – 500075, Telangana, India

www.vjit.ac.in

Department of Information Technology

Course file

Regulations : R19
Batch : 2019-2023
Academic year : 2020-2021
Program : B.Tech
Course name : WT
Year/ Sem : III / II
Course Code : A36545
Pre-Requisites : pps - I & II
Course Coordinator : Venkatesh. W

Index

INDEX

S.NO.	ITEM DESCRIPTION
1	Course Information Sheet
2	Syllabus
3	Text Books, Reference Books, Web/Internet Resources
4	Time table
5	Program Educational Objectives(PEOS) and Program Outcomes(POs)
6	Program Specific Outcomes(PSOs)
7	Course Outcomes(COs)
8	Mapping of Course Outcomes, POs and PSOs
9	Course Schedule
10	Lecture Plan/Teaching Plan
11	Unit wise Date of Completion and Remarks
12	Assignment Questions
13	Unit wise Question Bank
14	Mid Question Papers
15	End Exam papers
16	Content Beyond Syllabus
17	Unit wise PPTs and lecture notes
18	CO Attainment - Direct and Indirect
19	Course end survey form



Syllabus

WEB TECHNOLOGIES

III Year B.Tech. IT – II Sem R-19

L	T	P	C
3	0	0	3

Course Outcomes:

1. Develop static and dynamic web pages using HTML and javascript.
2. Understand the XML tags and to parse XML data with java.
3. Develop web applications using server side programming with PHP.
4. Implement web applications using JDBC and Servlets.
5. Apply web applications with JSP.

UNIT – I

Introduction to HTML: HTML tags, Lists, Tables, Images, Forms, Frames, Cascading Style Sheets

Client Side Scripting: Java Script Language – Declaring variables, Scope of variables, Functions, Objects in java scripts, Dynamic HTML with java scripts, Form Validation.

UNIT – II

XML: Introduction to XML, Defining XML tags their attributes and values, Document Type Definition, XML Schema, Document Object Model, and XHTML.

Parsing XML Data: DOM and SAX Parsers in java.

UNIT – III

Introduction to PHP:

Declaring variables, data types, arrays, strings, operators, expressions, control structures, functions, Reading data from web form controls like text boxes, radio buttons, lists etc. Handling File Uploads. Connecting to database (MySQL as reference), executing simple queries, handling results, Handling sessions and cookies.

File Handling in PHP: File operations like opening, closing, reading, writing, appending, deleting etc. binary files listing directories.


UNIT – IV

Introduction to Servlets: Common Gateway Interface (CGI), The Servlet API, Life cycle of a Servlet, Deploying a Servlet, Reading Servlet parameters, Reading Initialization parameters, Handling HTTP Request & Responses, Using Cookies and Sessions,

Introduction to JDBC: JDBC Drivers, JDBC Process, Connecting to a Database using JDBC

UNIT – V

Introduction to JSP: The Anatomy of a JSP Page, Introduction to MVC Architecture, JSP Processing, Declarations, Directives, Expressions, Code Snippets, Implicit Objects, Using Beans in JSP Pages, Using Cookies and Session for Session Tracking, Connecting to Database using JSP.



**Text Books
&
Reference Books**

Text Books:

1. Programming the World Wide Web 7th Edition by Robert W. Sebesta
2. Web Technologies Uttam K Roy, Oxford University Press

Reference Books:

1. The Complete Reference PHP – **Steven Holzner** , Tata McGraw-Hill
2. Web Programming, Building Internet Applications , Chris Bates 2nd edition , Wiley Dreamtech
3. Java Script , D Flanagan, O'Reilly, SPD
4. Java Server Pages- Hans Bergsten , SPD O'Reilly




Time Table

Time Table

A.Y - 2020-21

Class Hour Time	1	2	3	LUNCH BREAK	4	5
	9:30 - 10:30	10:40 - 11:40	11:50 - 12:50		2:00- 3:00	3:05- 3:55
MON			WT			II
TUE	WT					
WED						
THU		WT				
FRI					WT	
SAT						



Program Educational
Objectives(PEOs)&
Program Outcomes(POs)



Vidya Jyothi Institute of Technology

(Affiliated to JNTUHH)

Aziznagar Gate, C.B. Post, Hyderabad-500 075

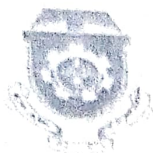
DEPARTMENT OF INFORMATION TECHNOLOGY

Program Educational Objectives (PEOs)

PEO1: Core Capabilities / Competence: Impart profound knowledge in humanities and basic sciences along with core engineering concepts for practical understanding and project development.

PEO2: Career Advancement: Enrich analytical and industry based technical skills through ICT for accomplishing research, higher education and entrepreneurship

PEO3: Life-Long Learning: Infuse life-long learning, professional ethics, adaptation to innovation and effective communication skills with a sense of social awareness.



Vidya Jyothi Institute of Technology

DEPARTMENT OF INFORMATION TECHNOLOGY

Program Outcomes

- PO1 Engineering Knowledge:** Apply knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.
- PO2 Problem Analysis:** Identify, Formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.
- PO3 Design/Development of Solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety and the cultural, societal and environmental considerations.
- PO4 Conduct Investigations of Complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of the information to provide valid conclusions.
- PO5 Modern Tool Usage:** Create, Select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
- PO6 The Engineer and Society:** Apply Reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues, and the consequent responsibilities relevant to the professional engineering practice.
- PO7 Environment and Sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts and demonstrate the knowledge of and need for sustainable development.
- PO8 Ethics:** Apply Ethical Principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- PO9 Individual and Teamwork:** Function effectively as an individual and as a member or leader in diverse teams and in multidisciplinary settings.
- PO10 Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large such as, being able to comprehend and with write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
- PO11 Project Management and Finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team to manage projects and in multi disciplinary environments.
- PO12 Life-Long Learning:** Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs)

PSO 1: Enhanced ability in applying mathematical abstractions and algorithmic design along with programming tools to solve complexities involved in efficient programming.

PSO 2: Develop effective software skills and documentation ability for graduates to become Employable/ Higher studies/ Entrepreneur/ Researcher.

Course Outcomes

After completing this course the student must demonstrate the knowledge and ability to

1. Create static and dynamic web pages using HTML and java script.
2. Analyze the XML and how to parse XML data with java.
3. Develop web applications using server side scripting language-PHP.
4. Implement the web applications using JDBC and java servlets.
5. Apply web applications with Java Server Pages.

Mapping of Course Outcomes with POs, PSOs & PEOs

Course Outcomes	POs	PSOs
CO1	1, 3, 5, 6, 7, 8, 9, 11, 12	1, 2
CO2	1, 2, 3, 5, 11, 12	1, 2
CO3	1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12	1, 2
CO4	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12	1, 2
CO5	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12	1, 2

Articulation matrix of Course outcomes with POs

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	3		3		3	2	2	1	2		3	2
CO2	3	1	3		2						2	2
CO3	3	3	3		3	2	3	2	3	2	3	3
CO4	3	2	3	3	2	3	3	1	2	2	3	3
CO5	3	2	3	2	2	3	2	1	2	2	3	3

Web Technologies

Articulation matrix of Course outcomes with PSOs

	PSO1	PSO2
CO1	3	3
CO2	3	3
CO3	3	3
CO4	3	3
CO5	3	3

Course Schedule

Distribution of Hours in Unit – Wise

Unit	Topic	Reference Section	Total No. of Hours
I	Introduction to HTML Client Side Scripting	Text Book1- (Chapters 2,4)	15
II	XML Parsing XML Data AZAX	Text Book2 - (Chapter 6, 7, 8, 9, 17)	5
III	Introduction to PHP File Handling in PHP	Text Book1- (Chapters 11)	15
IV	Introduction to Servlets Introduction to JDBC	Text Book2 - (Chapter 20)	8
V	Introduction to JSP Introduction to MVC Architecture	Text Book2 - (Chapter 21, 22)	9
Total contact classes for Syllabus coverage			52
<i>Tutorial Classes : 2 per Unit</i>			
<i>Assignment Tests : 02 (Before Mid1 & Mid2 Examinations)</i>			

Lecture Plan /Teaching Plan



Vidya Jyothi Institute of Technology
(An Autonomous Institution)
(Accredited by NAAC, Approved by AICTE & Permanently Affiliated to JNTUH)

Department of Information Technology

Subject: Web Technologies

Class: IIIB.Tech, II SEM

Lesson Plan

Lecture No.	Topics to be Covered	Suggested Books	Page Nos	Scheduled date	Completed date
UNIT I					
1.	Introduction to HTML				
2.	HTML tags	T2(4.4 – 4.5.10)	90-101	20/2/2021	30/2
3.	Lists, Tables	T2(4.5.11)	101-103	21/02	21/2
4.	Images, Forms, Frames	T2(4.6.4 – 4.6.5)	110 - 117	06/04	06/4
5.	Cascading Style Sheets	T2(5.1 – 5.6)	135-160	07/04	07/04
6.	Client Side Scripting - Java Script Language – Declaring variables, Scope of variables	T2(13.1-13.7)	383-416	09/04	09/04
7.					
8.	Functions, Objects in java script	T2(13.8 – 13.9)	416-442	12/04	12/04
9.	Dynamic HTML with java script	T2(15.1 – 15.5)	505-522	16/04	16/04
10.	Form Validation	T2(16.1 - 16.1.2)	523-541	19/04	19/04
UNIT –II					
11.	XML: Introduction to XML, Defining XML tags their attributes and values	T2(6.1 - 6.9)	163-182	20/04	20/04
12.	Document Type Definition	T2(7.1 – 7.7)	183-205	22/04	22/04
13.	XML Schema, Document Object Model	T2(8.1 – 8.10)	216-233	26/04	26/04
14.	Parsing XML Data: DOM and SAX Parsers in java	T2(9.1-9.8)	258-290	27/04	27/04
UNIT –III					
15.	Introduction to PHP: Declaring variables	T1(11.1-11.5)	449-459	30/04	30/04
16.	Data types arrays, strings, operators and expressions	T1(11.1-11.5 & 11.7)	449-459 & 463-470	01/05	01/05
17.	Control structures	T1(11.6)	460-462	02/05	02/05
18.	Functions, Reading data from web form controls like text boxes, radio buttons, lists etc	T1(11.8-11.10)	471-486	04/05	04/05
19.	Handling File Uploads	R1(Chapter 5)	187-191	7/05	07/05
20.	Connecting to database (MySQL as reference), executing simple queries, handling results	T1(13.5)	557-565	8/05	08/05
21.					
22.	Handling cookies	T1(11.12- 11.13)	487-490	15/05	15/05
23.	Handling sessions			16/05	16/05
24.	File operations like opening, closing, Reading, writing, appending, deleting etc, Binary files	R1(Chapter 9)	319-357	17/05	17/05
25.					
26.					
26	Revision Class				
UNIT-IV					
28.	Introduction to Servlets: Common Gateway Interface (CGI)	T2(19.1-19.10)	593-607	18/05	19/05

29.	Life cycle of a Servlet, Deploying a servlet	T2(20.1-20.6)	608-613	2/06	02/06
30.	The Servlet API	T2(20.7-20.8)	614-618	4/06	04/06
31.	Reading Servlet parameters, Reading Initialization parameters	T2(20.9)	618-619	7/6	7/06
32.	Handling HTTP Request	T2(20.10)	619-620	10/6	10/06
33. 34.	Using Cookies and Sessions	T2(20.12)	623-624	19/6	19/06
35.	Introduction to JDBC: JDBC Drivers, JDBC Process	T2(21.13-21.14)	676-677	22/6	22/06
36.	Connecting to the database using JDBC	T2(21.15-21.20)	677-692	23/06	23/06
37.	Revision Class				
UNIT-V					
38. 39.	Introduction to JSP: The Anatomy of a JSP Page, JSP Processing	T2(21.1-21.6)	631-640	01/7	01/07
40.	JSP Declarations, JSP Directives	T2(21.7-21.8)	640-664	6/7	06/07
41. 42.	Expressions, Code Snippets, Implicit Objects	T2(21.7-21.8)	640-664	9/7	9/07
43.	Using Beans in JSP Pages	T2(21.9)	664-667	14/7	14/7
44.	Using Cookies	T2(21.10)	668-673	20/7	21/7
45.	Using Session for session tracking	T2(21.10)	668-673	28/7	28/9
46. 47.	Connecting to database in JSP	T2(21.15-21.20)	677-692	20/7	30/7
48.	Introduction to MVC Architecture	T2(22.12)	739-742	03/8	03/08
49.	Case study:1 for creating static web pages			05/8	05/08
50.	Case study:2 for creating code snippet for web pages			07/8	07/08
51.	Case study:3 for creating web pages for identifying cookie ids			08/08	08/08
52.	Case study:4 for creating web pages using different style sheets			14/08	14/08

TEXT BOOKS

1. Programming the World Wide Web (7th Edition) 7th Edition by Robert W. Sebesta
2. Web Technologies Uttam K Roy, Oxford University Press

REFERENCE BOOKS

1. The Complete Reference PHP – Steven Holzner , Tata McGraw-Hill
2. Web Programming, Building Internet Applications , Chris Bates 2nd edition , Wiley Dreamtech
3. Java Script , D Flanagan, O'Reilly, SPD
4. Java Server Pages- Hans Bergsten , SPD O'Reilly

Unit wise Date of Completion and Remarks

Date of Unit completion & Remarks

Unit - I	
Date:	19/04/21
Remarks:	Additional class were taken for form validations, only one tutorial class were taken.
Unit - II	
Date:	27/04/21
Remarks:	The demonstration of XML DTD was not planned earlier, hence a addition classes was taken. ^{Therefore} Delayed to complete unit in time.
Unit - III	
Date:	12/05/21
Remarks:	along with PHP Parser, MySQL, the additional tool XAMPP was used for PHP Programming, the syllabus is covered in time.
Unit - IV	
Date:	23/06/21
Remarks:	JDBC concepts was explained w.r.t Laboratory Experiments also during the class hour. syllabus is covered in time.
Unit - V	
Date:	14/08/21
Remarks:	completed in time.



Unit wise Assignment
Questions

Unit Wise Assignments (With different Levels of thinking – Blooms Taxonomy and Course Outcomes)

Unit – I	
1	Design a Registration page using HTML and Write a javascript to validate the above registration page. (Level-6, CO1)
2	Differentiate internal and external stylesheets. (Level-3, CO1)
Unit – II	
1	Design an XML Schema and validate an XML document against it. (Level-6, CO2)
2	Identify the properties of XMLHttpRequest object and Explain the limitations of AJAX. (Level-2, CO2)
Unit – III	
1	Explain the Super global variables in PHP and write the login form validation using PHP (Level-3, CO-3)
2	Design a Registration page and insert the data in mySql Database using PHP (Level-6, CO-3)
Unit – IV	
1	Explain the JDBC drivers and the process to connect with the database. (Level-3, CO4)
2	Differentiate between ServletConfig and ServletContext. (Level-3, CO4)
Unit – V	
1	List and Explain the implicit objects in JSP with Examples (Level-3, CO-5)
2	Explain MVC Architecture in detail. (Level-3, CO-5)

Unit wise Question Bank

Unit Wise Questions

Questions	Blooms Level
UNIT- I	
Short Answer Questions	
1. What are HTML Tags? Explain.	2
2. What are HTML elements and attributes ?	1
3. What are the types of HTML lists?	1
4. Explain HTML blocks?	2
5. List few HTML Tags with their syntaxes.	1
6. What is java script? Explain the advantages of java script.	2
7. Discuss the features of javascript	2
8. How to embed java script into an HTML page ?	2
9. Differentiate client side validation and server side validation	3
10. How to declare global and local variables in java script ?	1
11. Explain the scope of the variables in java script	2
12. How to write functions in java script ?	1
13. Define array with an example	1
14. Discuss various ways of declaring array in java script	2
15. Write a program to compute the factorial of a given no using java script	3
16. Specify the In built objects supported by java script	2
17. Demonstrate String object with an example	3
18. List the methods and properties of DATE object	1
19. List the methods and properties of History object	1
20. List the methods and properties of Navigator object	1
21. Write a program to change the background and foreground colors of a web page using java script.	3
22. List the various data types or literals supported by java script	1
23. List the various operators supported by java script	1
24. Write the methods supported by document object	1
25. Differentiate various dialog boxes of window object(alert, prompt, & confirm)	3
26. List the various events supported Javascript	1
Long Answer Questions	
1. List and explain HTML tags in detail	3
2. Design the Login& Registrations pages using HTML.	6
3. Explain the control statements supported by Java Script	3

6. Differentiate SAX and DOM parsers with an example	3
Unit-III	
Short Answer Questions	
1. Compare scripting language and programming language.	3
2. Differentiate client side scripting and server side scripting.	3
3. How PHP is different from HTML?	3
4. State the declaration of the variables in PHP	1
5. Discuss various data types and how they are implemented in PHP	2
6. Define array. List different types of Arrays in PHP .	1
7. How do we declare strings in PHP?	1
8. Classify the operators in PHP	2
9. What is the difference between for and foreach statement in PHP	3
10. Demonstrate the difference between echo and print with an example.	3
11. What is MIME?	1
12. Identify the methods available in submitting form data.	1
13. Compare GET and POST method.	3
14. Discuss Session	2
15. Discuss How can we register the variables into a session	2
16. How many ways we can pass the variable through the navigation between the pages?	2
17. How can we know the total number of elements in any Array?	2
18. How to store the uploaded file to the final location?	2
19. Demonstrate mysql_error().	3
20. Differentiate include() and require() functions	3
21. How do you define a constant?	3
22. Demonstrate how to pass a variable by value in PHP	4
23. Demonstrate how to pass a variable by reference in PHP	4
24. Write a PHP code to create database.	4
25. Write a PHP code to create table.	4
26. Write a PHP code to insert values in table.	4
27. Write a PHP code to delete values from table.	4
28. Write a PHP code to update values in a table.	4
29. Write a PHP code to display the contents of table.	4
30. What are cookies? How do we handle cookies in PHP.	3
31. Illustrate special set of tags <?= and ?> do in PHP	2
32. Compare PHP and JavaScript	3
Long Answer Questions	
1. Define array. Explain array declaration and various built in array functions.	3

2. Tabulate the inbuilt functions of strings with an example program	3
3. Compare and contrast various control structures –for loop, while loop, do-while loop and for-each loop.	3
4. Explain how do we declare functions and types of functions in PHP	3
5. Explain various file handling operations in PHP.	3
6. Design a form having name, email id etc and validate these fields using PHP.	6
7. Design a login form and check whether the user is authorized or not using PHP.	6
8. Demonstrate the steps to create a database using PHP, Explain with example.	3
9. How do we upload files in php? Give a program to implement file uploading.	3
10. Develop code to connect PHP with MySQL	3
UNIT-IV	
Short Answer Questions	
1. Discuss the purpose of CGI.	2
2. Define Servlet.	1
3. Differentiate CGI and servlet	3
4. List the major operations of servlets	1
5. Compare Generic servlet and Http servlet	3
6. Differentiate doGet() and doPost() method	3
7. When init() method of servlet gets called?	1
8. When service() method of servlet gets called?	1
9. Differentiate get method and post method	3
10. List the methods of ServletRequest and ServletResponse interface	1
11. List the methods of ServletConfig interface	1
12. How to read form data into servlet?	1
13. Write the code to read initialization parameters	3
14. How to read http header information using servlet?	3
15. Differentiate HttpServletRequest and HttpServletResponse	3
16. How to write html contents using servlets?	3
17. List different types of JDBC drivers	1
18. Differentiate Statement object and PreparedStatement object	3
19. Where we apply Callable Statement object ?	3
20. What is the purpose of Resultset	1
21. Compare and contrast executeQuery(), executeUpdate() and execute() methods	3
Long Answer Questions	
1. Demonstrate servlet life cycle.with an example program	3

2. Classify Servlet API	1
3. Explain handling Http request and Http Response using servlets with an example	2
4. Define Cookie . Write a program to implement Cookies using sevlets	3
5. Explain session tracking using servlets	2
6. Explain the purpose various types of JDBC drivers	2
7. Explain the steps of JDBC Processing or Explain the steps to connect to database using JDBC	2
8. Demonstrate PreparedStatement object for inserting records into Database using JDBC	3
9. Write a program to validate user name password using Servlets and JDBC	3
10. Demonstrate ResultSet object to display the records of a table	3
UNIT V	
Short Answer Questions	
1. What are the features of JSP ?	1
2. Differentiate JSP and Servlet	3
3. List the steps for JSP Processing	1
4. Discuss the Anatomy of a JSP page	2
5. List out the various components of JSP	1
6. Write the syntax for code scriptlets using JSP	2
7. Demonstrate JSP expressions with an example	3
8. How to declare variables using JSP ?	1
9. list commonly used JSP directives	1
10. Design a web page to display current date and time using JSP expressions	6
11. How to pass parameters using JSP or Explain <jsp:param> tag ?	1
12. What is the difference between include directive and include action ?	3
13. Explain session object with an example	2
14. Differentiate cookie and session	3
15. Differentiate JSP Directives and JSP Actions	3
16. List the attributes for JSP page directive	1
17. List JSP implicit objects	1
18. Write the syntax and example of JSP forward action	1
19. What are the methods used by session object to set and retrieve session data	1
20. Discuss URL rewriting in session tracking	2
21. Discuss hidden type in session tracking	2
Long Answer Questions	
1. Explain JSP Processing with a neat diagram	2
2. Explain JSP Scripting elements or Explain the components of JSP with	2

syntax and examples	
3. Explain JSP directives with examples	2
4. Explain JSP actions with examples	2
5. How to share Session Data and Application Data using JSP?	3
6. Demonstrate session tracking using JSP with an example	3
7. Demonstrate cookies using JSP with an example	3
8. How to use Beans using JSP actions explain with an example program ?	3
9. Write a program to validate username and password using JSP and JDBC	3
10. Explain the steps to install Apache Tomcat Web Server	3



Mid Question Papers

VidyaJyothi Institute of Technology (Autonomous)
 (Accredited by NAAC & MIA. Approved By New Delhi, Permanently Affiliated to MTV Hyderabad)
 (Aziz Nagar, C.B.Post, Hyderabad -500075)

III B.TechII Semester Mid Exam 1-SET-1		
Total Duration(H:M): 1:30	Course: Web Technologies	Max.Marks: 20
Branch: Information Technology		Date:

Q.NO	Questions	Marks	CO	BL
Part-A				
Answer All The Questions				
1.	Discuss the features of JavaScript	2	1	2
2.	What is a valid XML document?	2	2	1
3.	Discuss various data types and how they are implemented in PHP	2	3	2
PART-B				
4.a)	List HTML tags and explain “table” and “frameset” tags of HTML in detail	3	1	2
b)	Design the Login page using HTML	2	1	6
(OR)				
5.a)	Explain JavaScript functions and variable scope with an example	3	1	3
b)	Differentiate inline, internal and external CSS	2	1	3
6.	Compare and contrast XML DTD and XML schema (XSD)	5	2	3
(OR)				
7.	Differentiate SAX and DOM parsers with an example	5	2	3
8.	Compare and contrast various control structures –for loop, while loop, do-while loop and for-each loop in PHP	4	3	3
(OR)				
9.	Define array. Explain array declaration and various built in array functions in PHP	4	3	3



Vidya Jyothi Institute of Technology (Autonomous)

(Accredited by NAAC & NBA, Approved By A.I.C.T.E., New Delhi, Permanently Affiliated to JNTU, Hyderabad)
(Aziz Nagar, C.B Post, Hyderabad - 500075)

III Year B.Tech II Semester 2nd Mid Exam

Branch: Information Technology

Duration: 90Min

Sub: Web Technologies

Marks: 20

Date: 22.04.2019

Session: FN

Course Outcomes:

1. Create static and dynamic web pages using HTML and java script.
2. Analyze the XML and how to parse XML data with java.
3. Develop web applications using server side scripting language-PHP.
4. Implement the web applications for JDBC and Java Servlets.
5. Design web applications with Java Server Pages.

Bloom Levels:

Remember	I
Understand	II
Apply	III
Analyze	IV
Evaluate	V
Create	VI

PART-A (3Q×2M = 6 Marks)		Outcomes		BL	Marks
ANSWER ALL THE QUESTIONS		CO	PO		
1	List of the File opening modes in PHP.	3	1, 3, 5, 6, 7, 8, 9, 11, 12	I	2
2	Write a short note on Servlet API.	4	1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12	II	2
3	What is JSP Directive Element?	5	1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12	I	2
PART-B (4+5+5= 14 Marks)		Outcomes		BL	Marks
ANSWER ALL THE QUESTIONS		CO	PO		
4.i)	Explain with an example how Cookies are created in PHP.	3	1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12	IV	4
[OR]					
ii)	Explain PHP sessions with an Example.	3	1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12	III	4
5.i)	Describe a life cycle of a Java Servlet and write a simple servlet that reads three parameters from the form data.	4	1, 3, 5, 6, 7, 8, 9, 11, 12	III	5
[OR]					
ii)	Differentiate Common Gateway Interface (CGI) and Servlets.	4	1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12	III	5
6. i)	Discuss about implicit objects in JSP with examples.	5	1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12	III	5
[OR]					
ii)	Explain about MVC architecture and concept of java beans.	5	1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12	IV	5

End Exam Papers



B.Tech. III Year II Semester Regular/Supply Examination, OCTOBER/NOVEMBER-2020

SUBJECT : Web Technologies

BRANCH : CSE & IT

Time: 2 Hours

Max. Marks: 75

Note: This question paper contains EIGHT questions and answer any FIVE questions. Each question carries 15 marks.

Bloom's Level:

Remember	L1
Understand	L2
Apply	L3
Analyze	L4
Evaluate	L5
Create	L6

ANSWER ANY FIVE QUESTIONS		50X15M = 75 M	Bloom's Level	Marks
1.a)	Write about the Importance of Dynamic HTML and Advantages of DHTML.		L4	8M
b)	Describe the Advantages and Disadvantages of Java Script. Apply the following Inline CSS rules for two interlinked HTML pages first.html and second.html i)In the first page Ordered lists should have a background color of #FFCC99 and unordered lists should have a background color of #CCFFCC. All list elements should be in italics. ii)In the second page Links should never show the default underlining and upon hovering they should become neon green (#33ff33).		L4	7M 10M
b)	Develop a JavaScript program to get the current date.		L5	5M
3.a)	Explain with examples of Document Object Model(DOM) and SAX.		L4	8M
b)	What is the difference between JavaScript and AJAX?		L4	7M
4.a)	Describe Document Type Definition.		L4	5M
b)	Give the importance of XML Parser using Java Script? Explain.		L4	10M
5.a)	Develop a PHP file on Reading data from Web Form controls like Text Box values are "username", and "password".		L4	10M
b)	Explain setcookie() function in PHP.		L5	5M
6.a)	Define Operator? Explain Different Operator used in PHP with suitable example.		L4	7M
b)	How can we create a database using PHP and MySQL?		L5	8M
7.a)	Describe the lifecycle of servlet.		L4	5M
b)	List out various types of Session Tracking Techniques. Develop a servlet program using HTTP Session.		L5	10M
8.a)	Elaborate the process involved in JSP page translation and processing phases.		L4	10M
b)	Explain Model View Controller Architecture.		L5	5M



Vidya Jyothi Institute of Technology (Autonomous)

(Accredited by NAAC & NBA, Approved By A.I.C.T.E., New Delhi, Permanently Affiliated to JNTU, Hyderabad)
(Aziz Nagar, C.B.Post, Hyderabad -500075)

R15

Subject Code:A16522

B.Tech. III Year II Semester Examinations - MAY 2019

SUBJECT : WEB TECHNOLOGIES

BRANCH : CSE&IT

Time: 3 Hours

Max. Marks:75

Note: This question paper contains two *Parts A and B*.

Part A is compulsory which carries 25 Marks. Answer all the questions.

Part B consists of 5 questions. Answer all the questions.

Bloom Levels:

Remember	L1	Analyze	L4
Understand	L2	Evaluate	L5
Apply	L3	Create	L6

PART - A

B.L 25M

ANSWER ALL THE QUESTIONS

1	What is meant by HTML?	L1	2M
2	What are the different types of lists in HTML?	L2	3M
3	State rules to define tags in XML.	L1	2M
4	How can you declare attributes in XML? Give an example.	L1	3M
5	List various string functions in PHP.	L1	2M
6	State the rules for declaring variables in PHP.	L1	3M
7	What are various drivers of JDBC?	L4	2M
8	Write the advantages of Servlets over CGI?	L1	3M
9	What are the features of JavaScript?	L1	2M
10	How does one access cookie in a java script?	L1	3M

PART - B

B.L 50M

ANSWER ALL THE QUESTIONS

11.i.a)	Discuss the creation of HTML document with frames.	L6	5M
b)	Explain how nested tables are created using HTML.	L5	5M

[OR]

ii.a)	Explain different types of cascading style styles with suitable examples.	L5/L4	5M
b)	Discuss the scoping rules for the Java script.	L6	5M

12.i.a)	Compare and contrast between DOM and SAX parsers in java.	L4	5M
b)	Write the advantages of XML schemas over DTDs.	L5	5M

[OR]

ii.a)	Create a XML document to store voter ID, voter name, address and date of birth details. Create a DTD to validate the document.	L6	5M
-------	--	----	----

b)	What are the limitations of Document Type Definitions (DTDs)? How these limitations are overcome using XML schema?	L1	5M
----	--	----	----

13.i.a)	Write a PHP script to add and remove users from a MySQL table.	L5	5M
b)	Explain the user defined functions in PHP with an example.	L2	5M

[OR]

ii.a)	Discuss the different data types and operators supported in PHP.	L6/L4	5M
b)	Write a PHP program for passing by value and passing by reference.	L5	5M

14.i.a)	Explain the differences between Generic Servlet and HttpServlet.	L2	5M
b)	Explain about the database connectivity using JDBC.	L5	5M

[OR]

ii.a)	What potential advantages do servlets have over CGI programs? Explain.	L1	5M
b)	Write note on Common Gateway Interface (CGI).	L5	5M

15.i.a)	Write a JavaScript to display whether given number is a prime or not.	L5	5M
b)	How is Client side Java Script different from Server side Java Script?	L1	5M

[OR]

ii.	Explain in detail of how to use Java Beans in JSP pages with suitable example.	L2	10M
-----	--	----	-----

Content beyond Syllabus

S.No	Name of the Topic
1	Angular JS
2	JQuery
3	Node JS



Unit Wise PPTs and Lecture Notes

HTML LMN

What is HTML?

- Defines the structure of the website. If a human body is a website, then HTML would be our skeleton.
- Just like our skeleton is made up of various bones (skull, thigh bone, etc), the HTML file will have various 'elements' or 'tags' where each element or tag will affect the 'structure' of the website
- IT IS NOT A PROGRAMMING LANGUAGE

Components of an HTML file

- **Head**
 - Defines the structure of the head of the website.
 - Head is the part of the website which 'describes' the website. The head mainly deals with **meta** tags which contain metadata (data about data) of the website. The **title** tag contains the title of the website (the name which one sees on browser window tabs)
 - Also used to link external CSS files or other files to HTML files using the **link** tag or the **script** tag (for backend scripts (script = something written in a backend language like javascript or python or php))
- **Body**
 - This is the section of the website which the user sees and interacts with
 - Contains the bulk of the website

Tags in HTML

- The structure of any website contains the various 'things' which will be in the website like text, videos, images, headings, navbar, footer, etc etc. But how do we actually 'code' them? By using **tags**. They act as keywords to **render** certain types of elements (paragraphs, headings, sections, etc)
- Structure of a tag : tags can be **stand-alone tags** or **closing tags**
 - Closing tags are of the form : <tag> Something or nothing </tag>
 - Stand-alone don't need the closing tag and just do fine without them. They look something like: <aatmanirbhar>
 - Every closing tag will have some form of content within the opening and closing tag which will be content to be rendered

HTML attributes

- Some elements require additional information to function properly and this is done via attributes.
- Eg: the tag is used to display an image, but for that it needs to know one important thing. **The location of the image**. So, we use the **src** attribute to **provide** the img tag with the location of the image which we want to display.

Frequently used tags in HTML

1. **<div>/div>** : Defines a section of a website. BEST. TAG. EVER
2. **<h1>/h1>** : The value next to the letter 'h' ranges from [1, 6] and defines a heading text (basically a bold p tag)
3. **<p>/p>** : Used to define any text in website
4. **** : Used to render images on the website
5. **<video>** : Used to render videos on the website
6. **<form>/form>** : Used to structure forms (eg: google forms) in a website
7. **<input>** : Used to specify form inputs (text area, radio button, checkbox, etc)
8. **<a>/a>** : Used to 'hyperlink' (link) a resource (pdf, website, etc)
9. **/ul>** : Defines an 'unordered' list (not numbered)
10. **/ol>** : Defines an 'ordered' list (numbered)
11. **/li>** : Defines a list item inside a list (both ul and ol)

Best way to learn HTML : Don't 'learn it'. Practice it.

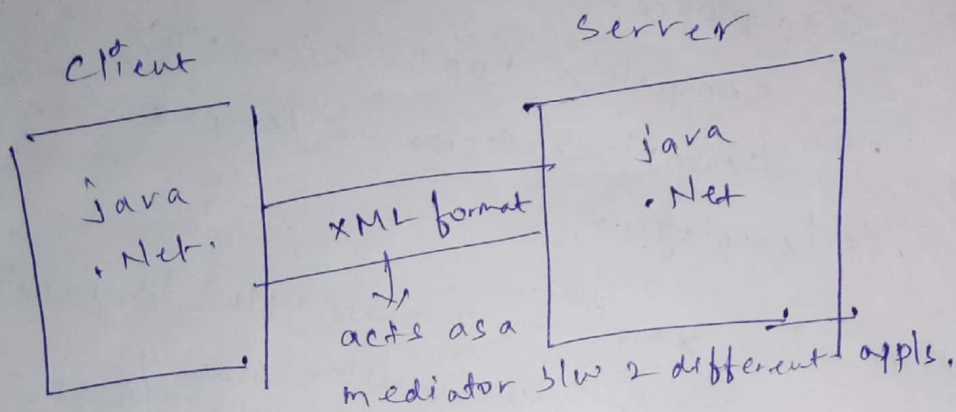
XML (Extensible Markup Language)

Markup → enclosing the textual info. in between tags.

Ex: `<emp>`
`<empname> Eswar </empname>`
`</emp>`

XML Tags

- Userdefined. (HTML - predefined tags)



→ HTML vs XML

- * Predefined - userdefined.
- * Limited feature - extensible feature.
- * not case sensitive - case sensitive tags
- * used to display the data - used to describe the data
- * all closing tags are optional - for every open tag there must be closed tag.

Note: → XML is a common format / platform & language independent.

→ only one version 1.0 & given by W3C.

Uses of XML

- Used to describe the data.
- XML documents - possible to maintain text based databases,
- XML docs. are used to transport ^{the data} from one lang/appl. to another appl.

Note: XML is ^{based} text document is saved with .xml.

Ex:

```
<employee>
  <empNo> 1001 </empNo>
  <empName> Eswar </empName>
</employee>
```

→ TO view XML ^{docs.} → browser slw is required.
XML editor slw.

XML document contains

- Elements
- Attributes
- entity references.

- element:

data from open tag to closing tag.

- Element can contain
- * child elements
 - * Text data
 - * attributes.
 - * mix of all.

- element names are
- case sensitive,
 - start with a letter or underscore
 - can contain letters, digits, hyphens, underscores & periods.
 - no spaces

→ Attribute — name-value pair.
— Used to define additional info. of an element.
— value associated with open tag.

Ex: employee empNo = "101" name = "Eswar"
</employee>

Note: 1) Attribute values must be enclosed in single/
double quotes.

2) Attribute names are unique ~~for~~ ⁱⁿ each element.

3) Attributes order is not impl. in an ".

→ Entity Reference:

<Person>

<name> Eswar </name>

<age> the person age is 18 </age>

</Person>

↓
illegal.

Note: in place of \leq , we use entity reference

< \Rightarrow < ; → entity reference

> \Rightarrow > ;

& \Rightarrow & ;

' \Rightarrow ' ;

" \Rightarrow " ;

Note: only < ~~&~~ ~~&~~ and & are illegal.

Syntax:

& entity reference name;

→ we can also create our own entity reference.
Note: → long length strings are represented using
entity references.
(shortcuts)

XML Document: (well formed doc.)

DTD: or XSD:

- begins with prolog
- must maintain unique root element
- must have closing tags for all elements.
- XML elements are case sensitive.
- attribute values are enclosed with quotes
- In place of spl chars. use entity references.

* DTD (Document Type Definition)

- DTD is an XML technique used to define the structure of an XML document.
- DTD is a text based document with .dtd extension
- DTD contains element, attributes & entity reference declarations.

Element declaration syntax:

<!ELEMENT element-name (content model) >

↳ specifies

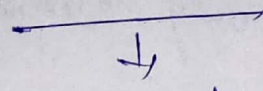
allows child element/
text data / both.

Ex: DTD

<!ELEMENT employee (empno, name, sal) >

child elements

<!ELEMENT> empno (#PCDATA) >



used only for element which allows only Text data

<!ELEMENT> ~~name~~ (#PCDATA) >

<!ELEMENT> sal (#PCDATA) >

#PCDATA (Parsable Character Data)

→ It is content model - used only for element which allows only Text data.

→ It can be string / numeric data.

declared in XML

XML

<employee >

<empno > 101 </empno >

<~~name~~ > Eswar </name >

<sal > 5000 </sal >

</employee >

* Content Model

In Content Model

5 types of elements are possible.

1) Text only elements → # PC DATA

2) child only element → ex: employee → allows only children.

3) empty element

4) ANY element

5) mixed element.

after
cardinality

* Mapping DTD with XML

In XML doc. for linking XML with DTD,

we use `<!DOCTYPE>`

Types of DTDs

1) Internal
writing DTD's within the XML only.

2) External

separate DTD's are used for linking with XML
(.dtd)

Internal DTD Syntax:-

```
<!DOCTYPE root elements [ DTD Rules ] >  
XML Elements
```



```

Ex: Progl. xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE employee [

```

```

  <!ELEMENT employee (empno, name, sal) >
    <!ELEMENT empno (#PCDATA) >
    <!ELEMENT name (#PCDATA) >
    <!ELEMENT sal (#PCDATA) >
] >

```

```

<employee>
  <empno> 101 </empno>
  <name> Eswar </name>
  <sal> 5000 </sal>
</employee>

```

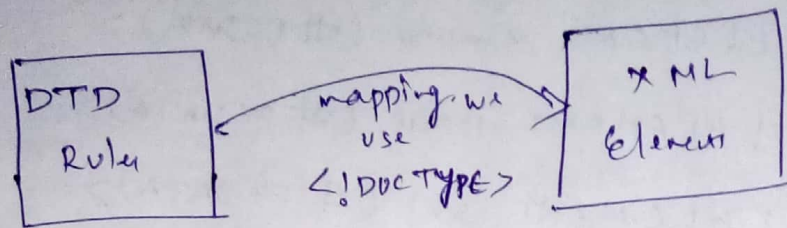
Disadvantages

- specific to only one xml file, written in xml
- not reusable
- readability decreases.

External DTD's:

→ DTD & XML Elements are written in ~~in~~ different files/documents

→



1) Private DTD's

2) Public DTD's

1) Private DTD's:

System for mapping private DTD's with XML in XML

```
<!DOCTYPE rootElement SYSTEM "DTDFileName.dtd">
```

↳ indicates private DTD

Ex: employee.dtd

```
Parent { <!ELEMENT employee (empNo, name, sal) >
Child { <!ELEMENT empNo (#PCDATA) >
        <!ELEMENT name (#PCDATA) >
        <!ELEMENT sal (#SAL) >
```

prog2.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE employee SYSTEM "D://XML/employee.dtd">
<employee>
  <empNo> 101 </empNo>
  <name> eswar </name>
  <sal> 5000 </sal>
</employee>
```

② Public DTD's:

of any DTD, which is not particular to the specific project.

Ex: Hibernate, Spring

Syntax:

```

<!DOCTYPE rootElement PUBLIC "-//vendorname//
version//EN" "dtdfilename.dtd"

```

"+" → for registered ^{DTD} with ISO

"-" → for not registered

registration, vendorname, version, language are not mandatory.

Ex:

Q 1

Specifying child element with separator | or ,

① comma " , " — all the child elements are mandatory to specify

Ex:-

<!ELEMENT book (name, title)>

<!ELEMENT name (#PCDATA)

<!ELEMENT title (#PCDATA)

<book>

<name>

<title>

</book>

② " | " vertical line — either or can be specified but not both.

Ex:- <!ELEMENT book (name | title)>

<

<

<book>

<name>

</book>

Cardinality operators:-

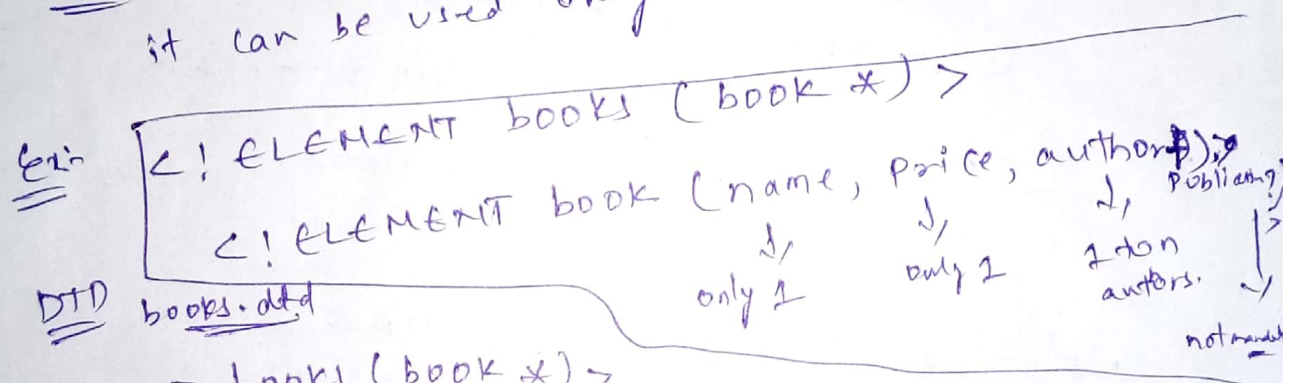
In a XML doc, an element can occur ^{zero or more} n no. of times

- to specify, how many no. of times, an element can occur can be specified using cardinality.

Types operators

- * → 0 to n
- + → 1 to n
- ? → 0 or 1.

Note:- If cardinality is not specified by default it can be used only once.



```
<!ELEMENT books (book*) >
```

```
<!ELEMENT book
  ((name|title), price, author+, publication?) >
```

```

<!ELEMENT name (#PCDATA) >
<! " title " >
<! " price " >
<! " author " >
<! " publication " >
  
```

XML
<!DOCTYPE books SYSTEM "books.dtd">
<books>

<book>

<name>

<price>

more than
2 { <author>
<author>
<publication>

</book> </book>
<book> . . . </book>
</books>

Mixed-element declaration (refer pg. ③)

Ex:

<!ELEMENT employees (employee*)>
<!ELEMENT employee (#PCDATA|empname|sal)*>
<!ELEMENT empname (#PCDATA)>
<!ELEMENT sal (#PCDATA)>

Note: employee - mixed element
empname & sal → text only element
employees - child only element.

XML

<!DOCTYPE employees . . . >

<employees>

<employees>

the emp name is <empname> & salary </empname>

and emp sal is <sal> 5000 </sal>

</employees>

</employees>

④ EMPTY elements:

in html
 & </br/> are empty elements.
no child's & ^{no} text data.

in xml -

```
<!ELEMENT employee EMPTY >
```

Note: No child elements & no text data.
but, it can contain attributes

```
Ex: <!DOCTYPE employees [
  <!ELEMENT employees (employee*) >
  <!ELEMENT employee EMPTY >
]
```

```
>
<employees>
  <employee> <employees> no space.
  (or)
  <employees>
</employees>
```

⑤ ANY Elements:

it is possible to main text data, child, empty or mixed type of elements.

```
Ex: <!DOCTYPE employees [ children.
  <!ELEMENT employees (employee+) >
  <!ELEMENT employee ANY >
  <!ELEMENT empname (#PCDATA) >
  <! " empSal (#PCDATA) >
]
```

XML
XML
employees >

mixed { employees >
the empname is <empname> Eswar </empname>

Text data only { <employee>
this employee info </employee>

empty -> <employee> </employee>

only child element { <employee>
<empsal> 1000 </empsal>
</employee>

</employees>

Attributes:

- Name-value pair
- Provides extra info. of an element.

Declaration of an attribute in DTD:

Syntax:

<!ATTLIST elementname attributename
attributedatatype attributespecifier otherInfo >

Ex: <!ELEMENT employees (employee+)> ^{1 min 1d max 0}

<!ELEMENT employee EMPTY> ^{no text data / no child.}

<!ATTLIST employee empno CDATA ~~#REQUIRED~~
#REQUIRED >

<!ATTLIST employee name CDATA #REQUIRED >

<!ATTLIST employee sal CDATA #REQUIRED >

employees

<employee empno="191" name="ABC" sal="5000"/>

<employee

</employee>

NOTE: CDATA - Only char data.

Attribute specifiers:

Used to specify attribute 's

→ mandatory attribute — #REQUIRED

→ optional " — #IMPLIED

→ fixed ' — #FIXED

→ default ' — Any specifier is not specified then attribute will be default.

Ex:in

```
<!ELEMENT students (student +)>
```

```
<!ELEMENT stu EMPTY>
```

mandatory —

```
<!ATTLIST stu sid CDATA #REQUIRED>
```

optional —

```
<!ATTLIST stu sname CDATA #IMPLIED>
```

default & fixed —

```
<!ATTLIST stu sfee CDATA #FIXED "1000">
```

default —

```
<!ATTLIST stu ssource DATA "JAVA">
```

```
<students>
```

```
<stu sid='10' ssource="C" sfee='1000' />
```

```
<stu sid='108' sname='Ewar' />
```

```
<stu sid='109' ssource="JAVA" sfee='1000' />
```

```
</students>
```

Attributes Datatypes:

CDATA (Character Data) Ex: empno="101" ; name="abc"

ENUMERATED - It allows any one of value from the specified list.

ID - The ^{attribute} value must be unique

IDREF - Reusability (one element attribute value into another element attribute)

IDREFS

NMTOKEN

NMTOKENS } → // to CDATA, but NMTOKEN it allows only alphabets, -, , .

NOTATION → used for media type like images/gif

ENTITY

ENTITIES

Ex: ENUMERATED (to specify list of possible values)

<!ELEMENT Payments EMPTY>

<!ATTLIST Payment-type (DD) CHECK | CASH) #REQUIRED >

Ex: ID

<!ATTLIST stu sid ID #REQUIRED >

Note: - ^{value} Should not start with number, but can start with alphabet or -

Ex: IDREF

<student >

sname="abc" sid="" scid="1"
scname="JP" scfee="500" >

<Sname sname="" sid="" scid="1" scname="JP" scfee="500"

Here scid, scname, scfee is common attr

ADOLTYPE students [

<student>

<!Element students (course*, student*) >

<!~~ATT~~LIST course cid ID #REQUIRED >

<!~~ATT~~ course cname CDATA #REQ >

<!~~ATT~~ course cfee CDATA #REQ >

<!ELE stu EMPTY >

<!ATT stu sid ID #REQ >

<!ATT stu sname CDATA #REQ >

<!ATT stu scourse IDREF #REQ >

] >

<student>

<course cid="A101" cname="JP" cfee="500" / >

<course cid="B10" cname="DS" cfee="300" / >

<course cid="C10" cname="CPP" cfee="5000" / >

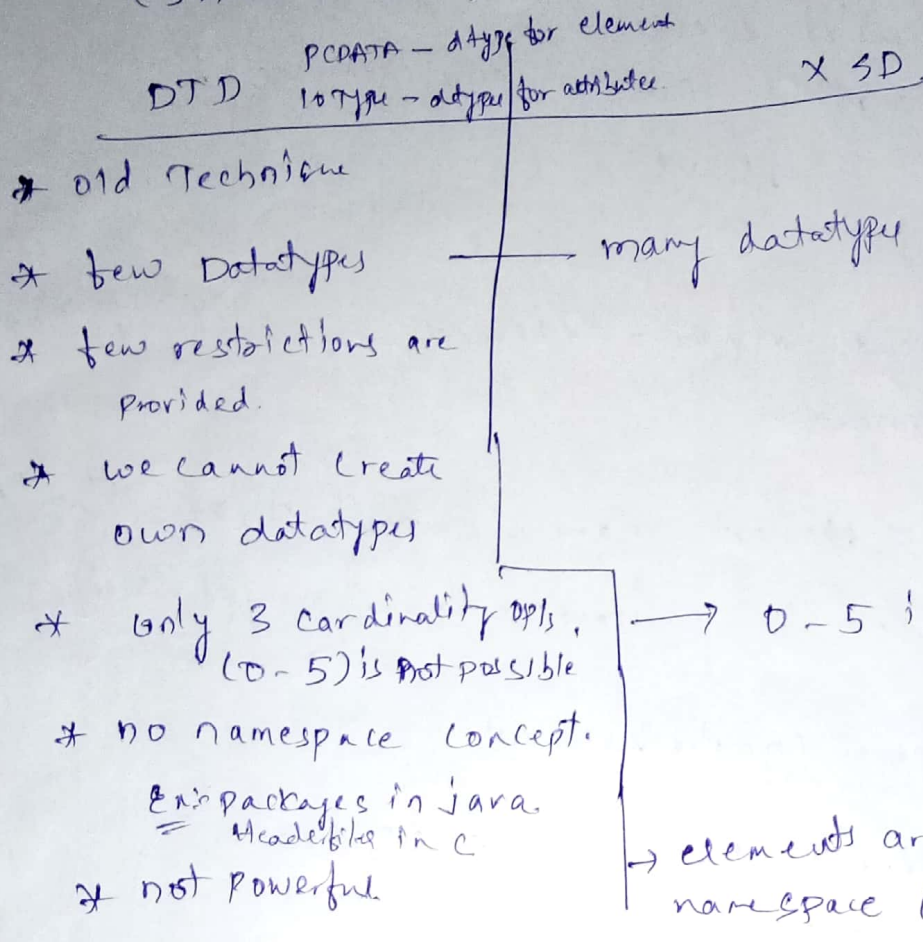
<stu sid="120" sname="abc" scourse="A101" / >

</students>

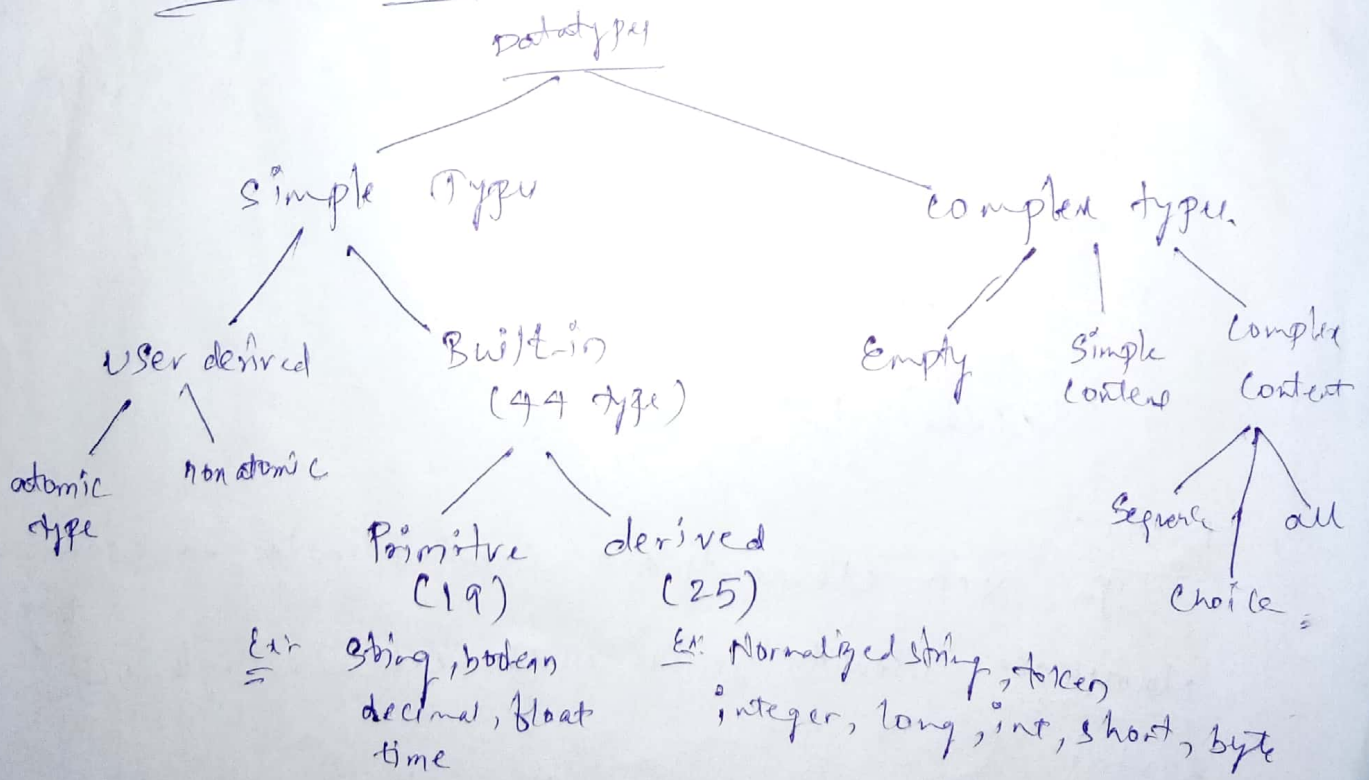
Note: cid must unique. ID type.

* XML Schema Definition: (XSD):

→ It is also used to ~~define~~^{define} the structure of an XML doc. (similar to DTD)



* Datatypes in XSD:



~~XSD~~ XSD elements:

- 1) simple elements → elements (contains only text data, no child & no attributes)
- 2) complex elements.
↳ it can contain child elements & attributes/both

Creating an element - syntax:

```
<element name = "element-name" type = "datatype" />  
</element>
```

ex:

```
<element name = "empno" type = "int" />  
< " name = "ename" type = "string" />
```

Syntax - complex element

```
<element name = "element-name" >
```

```
<complexType >
```

```
<sequence >
```

```
<element name = "child1" type = "datatype" />
```

```
<element name = "child2" type = "dt" />
```

```
⋮
```

```
</sequence >
```

```
</complexType >
```

```
</element >
```

Note:

In XSD elements are created inside the <schema> tag.

Namespace: used to group all elements as a single unit.

syntax:

```
<schema targetNamespace = "url" >
```

Cardinality operator

Ex: schemaemp.xsd

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"

targetNamespace="https://www.eswar.com"

xmlns="https://www.eswar.com"

elementFormDefault="qualified">

<xs:element name="emp">

<xs:complexType>

<xs:sequence>

<xs:element name="empno" type="xs:int" />

<xs:element name="empname" type="xs:string" />

<xs:element name="empsal" type="xs:decimal" />

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

schemaemp.~~xml~~ xml

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

<emp

xmlns="https://www.eswar.com"

xmlns: xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="https://www.eswar.com schemaemp.xsd">

<empno>1201 </empno>

<empname>eswar </empname>

<empsal>5000.50 </empsal>

</emp>

XML schema has lot many datatypes,

Most com. xs: string,

xs: decimal

xs: integer

xs: boolean

xs: date

xs: time.

Ex:

<xs: element name="age" type="xs: integer" />

<xs: element name="DOB" type="xs: date" />

XML - <age> 30 </age>

XML - <DOB> 2000-08-20 </DOB>

Default & fixed values for simple elements

Syntax - Default

<xs: element name="dept" type="type" default="m" />

Ex:

<xs: element name="dept" type="string" default="IT" />

Syntax - Fixed

<xs: element name="dept" type="string" fixed="CSE" />

XSD - Attributes

(12)

→ simple elements cannot have attributes. If an element has attributes, it is considered as complex type.

Syntax:

`<xs:attribute name="abc" type="xyz" />`

Ex:- `<xs:attribute name="dept" type="string" />`

XML - `<empname dept="IT" > Eswar </empname >`
↓ element name ↓ attribute name of empname element.

default value - attribute:

Ex:-
`<xs:attribute name="dept" type="string" default="IT" />`

fixed value attribute

Ex:-
`<xs:attribute name="dept" type="string" fixed="IT" />`

optional & required attributes:

to specify that the attribute is required, use the "use" attribute

Ex:-
`<xs:attribute name="firstname" type="string" use="required" />`

Note: Attributes are optional by default.

XSD - Restrictions:

— Restrictions are used to define acceptable values for an XML element or attribute.

Restriction on values:

Ex: If age must be b/w 0 & 100

```
<xs: element name="age" >
```

```
  <xs: simpleType >
```

```
    <xs: restriction base="xs: integer" >
```

```
      <xs: minInclusive value="0" / >
```

```
      <xs: maxInclusive value="100" / >
```

```
    </xs: restriction >
```

```
  </xs: simpleType >
```

```
</xs: element >
```

Restriction on set of values: (enumeration)

```
<xs: element name="branch" >
```

```
  <xs: simpleType >
```

```
    <xs: restriction base="xs: string" >
```

```
      <xs: enumeration value="IT" / >
```

```
      <xs: enumeration value="CSE" / >
```

```
      <xs: enumeration value="ECE" / >
```

```
    < / >
```

```
  < / >
```

```
< / >
```

Note: acceptable values for branch element are only *IT, CSE, ECE

Restriction on a set of values:

one letter in
only lowercase

<xs: restriction base="string" >

<xs: pattern value="[a-z]" / >

<xs: restriction >

[A-Z] — Capital letters — only 1 char

([a-z])* — more occurrences of lowercase a to z

*male | female — either male or female

[0-9] — digit.

[a-zA-Z0-9]{8} — exactly 8 characters in a row
& those chars must be lowercase, uppercase or numbers
from 0-9.

Restriction on length

<xs: restriction base="string" >

<xs: minLength value="5" / >

<xs: maxLength value="10" / >

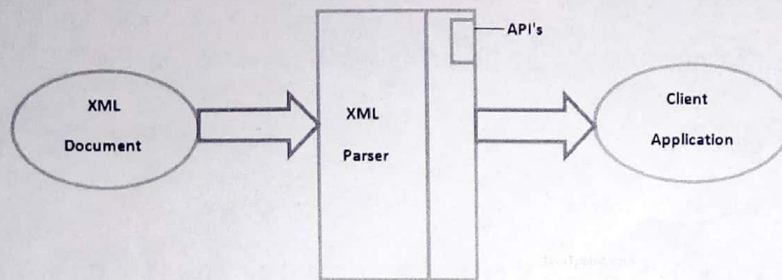
< / >

XML Parsers

An XML parser is a software library or package that provides interfaces for client applications to work with an XML document. The XML Parser is designed to read the XML and create a way for programs to use XML.

XML parser validates the document and check that the document is well formatted.

Let's understand the working of XML parser by the figure given below:



Types of XML Parsers

These are the two main types of XML Parsers:

1. DOM
2. SAX

DOM (Document Object Model)

A DOM document is an object which contains all the information of an XML document. It is composed like a tree structure. The DOM Parser implements a DOM API. This API is very simple to use.

Features of DOM Parser

A DOM Parser creates an internal structure in memory which is a DOM document object and the client applications get information of the original XML document by invoking methods on this document object.

DOM Parser has a tree based structure.

Advantages

- 1) It supports both read and write operations and the API is very simple to use.

2) It is preferred when random access to widely separated parts of a document is required.

Disadvantages

- 1) It is memory inefficient. (consumes more memory because the whole XML document needs to be loaded into memory).
- 2) It is comparatively slower than other parsers.

SAX (Simple API for XML)

A SAX Parser implements SAX API. This API is an event based API and less intuitive.

Features of SAX Parser

It does not create any internal structure.

Clients do not know what methods to call, they just override the methods of the API and place their own code inside the method.

It is an event based parser, it works like an event handler in Java.

Advantages

- 1) It is simple and memory efficient.
- 2) It is very fast and works for huge documents.

Disadvantages

- 1) It is event-based so its API is less intuitive.
- 2) Clients never know the full information because the data is broken into pieces.

Example ProgramDOMSteps**Test.java**

```
import java.io.IOException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
```

1. Create DocumentBuilderFactory class obj;
2. Create DocumentBuilder object
3. Parse the XML document

```
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
```

```
public class Test {
```

```
    public static void main(String[] args) {
```

```
        // TODO Auto-generated method stub
```

```
        DocumentBuilderFactory fact = DocumentBuilderFactory.newInstance();
```

```
        try
```

```
        {
```

```
            DocumentBuilder build = fact.newDocumentBuilder();
```

```
            Document doc = build.parse("F://employee.xml");
```

```
            Element ele = doc.getDocumentElement();
```

```
            System.out.println(ele.getNodeName());
```

```
            NodeList list = ele.getChildNodes();
```

```
            for (int i = 0; i < list.getLength(); i++)
```

```
            {
```

```
                Node n = list.item(i);
```

```
                if (n.getNodeType() == n.ELEMENT_NODE)
```

```
                    System.out.println(n.getNodeName() + "-->");
```

```
            n.getTextContent();
```

```
            }
```

```
        } catch (ParserConfigurationException | IOException | SAXException)
```

```
        {
```

```
            e.printStackTrace();
```

```
        }
```

```
    }
```

```
}
```

Employee.xml

```
<employee>
<ename>Eswar</ename>
<rollno>1208</rollno>
</employee>
```

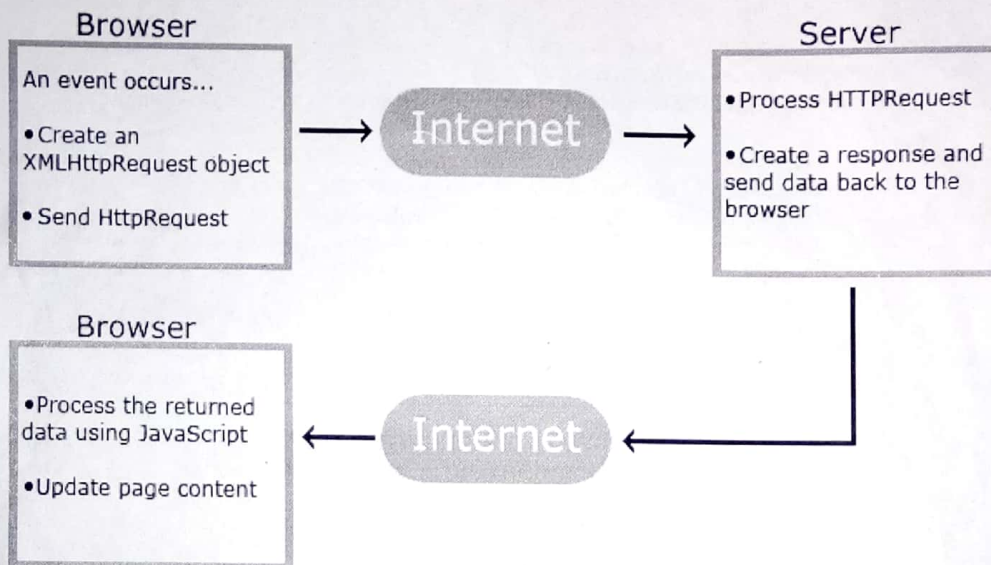
AJAX(Asynchronous JavaScript And XML)

AJAX is not a programming language.

AJAX just uses a combination of:

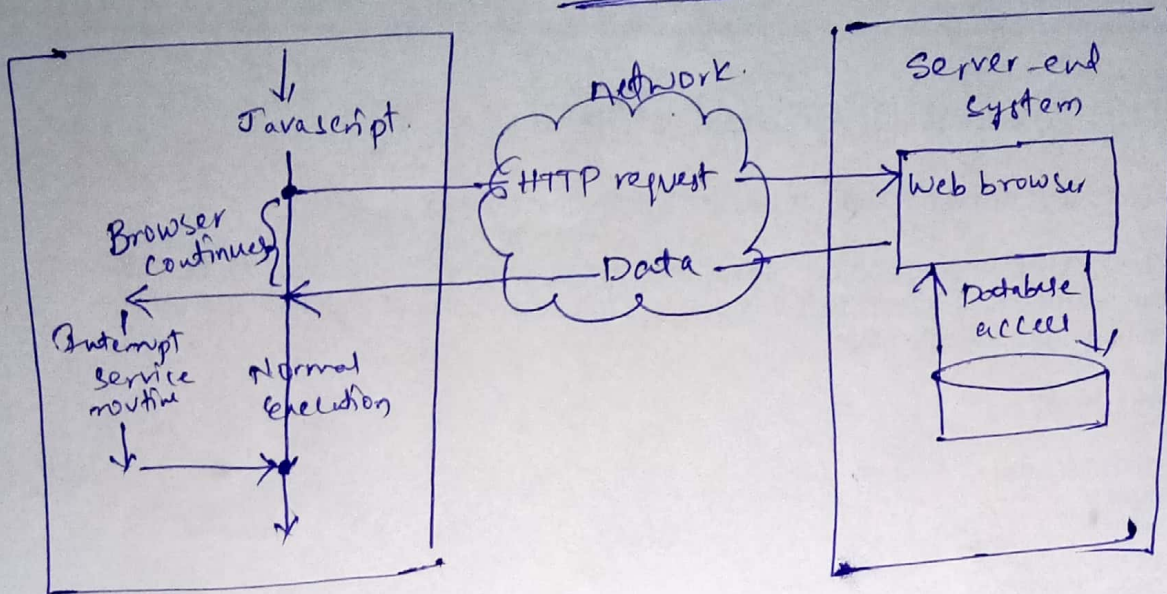
- A browser built-in XMLHttpRequest object (to request data from a web server)
- JavaScript and HTML DOM (to display or use the data)

AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.



1. An event occurs in a web page (the page is loaded, a button is clicked)
2. An XMLHttpRequest object is created by JavaScript *& specify a handler*
3. The XMLHttpRequest object sends a request to a web server
4. The server processes the request
5. The server sends a response back to the web page
6. The response is read by JavaScript
7. Proper action (like page update) is performed by JavaScript

Ajax framework



Properties & methods of XMLHttpRequest object.

State	State Value	Description
un-initialized	0	After creating the XMLHttpRequest obj, but before calling the open() method.
connection established	1	After calling open(), but before calling send()
Request sent	2	After calling send()
Processing	3	After calling send(), but before getting the response
Completed & response is ready	4	After the request has been completed, & the response data have been completely received from the server.

The XMLHttpRequest Object

All modern browsers support the XMLHttpRequest object.

The XMLHttpRequest object can be used to exchange data with a server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

Az1.html

```
<!DOCTYPE html>
<html>
<body>

<div id="demo">
<h1>The XMLHttpRequest Object</h1>
<button type="button" onclick="loadDoc()">Change Content</button>
</div>

<script>
functionloadDoc() {
varxhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
if (this.readyState == 4 &&this.status == 200)
{
document.getElementById("demo").innerHTML = this.responseText;
}
else
{
document.getElementById("demo").innerHTML = "<strong> waiting...</strong>";
}
};
xhttp.open("GET", "exAzax.html", true);
xhttp.send();
}
</script>

</body>
</html>
```

exAzax.html

This is an Example of Ajax

PHP – Environment setup

In order to develop and run PHP Web pages three vital components need to be installed on your computer system.

- **Web Server** – PHP will work with virtually all Web Server software, including Microsoft's Internet Information Server (IIS) but then most often used is freely available Apache Server. Download Apache for free here – <https://httpd.apache.org/download.cgi>
- **Database** – PHP will work with virtually all database software, including Oracle and Sybase but most commonly used is freely available MySQL database. Download MySQL for free here – <https://www.mysql.com/downloads/>
- **PHP Parser** – In order to process PHP script instructions a parser must be installed to generate HTML output that can be sent to the Web Browser. This tutorial will guide you how to install PHP parser on your computer.
<http://php.net/downloads.php>

Note: Bundle of these three components are available in the 3rd party. They are

- XAMPP Server (preferable)
- WAMP Server
- EasyPHP

XAMPP server

X – Cross platform

A – Apache

M – Maria DB

P – PHP

P – Perl

<https://www.apachefriends.org/download.html>

Hello World Program

Step1- Install XAMPP

Step2 -

- Assume you installed xampp in C Drive.
- Go to: **C:\xampp\htdocs**
- Create your own folder, name it for example as **eswar**.

Step3 -

- Now create your first php program in xampp and name it as **“hello.php”**
(Write in notepad & save in location - **C:\xampp\htdocs\eswar\hello.php**)
- Program – hello.php

```
<html>
<head>
<title>PHP Execution</title>
</head>

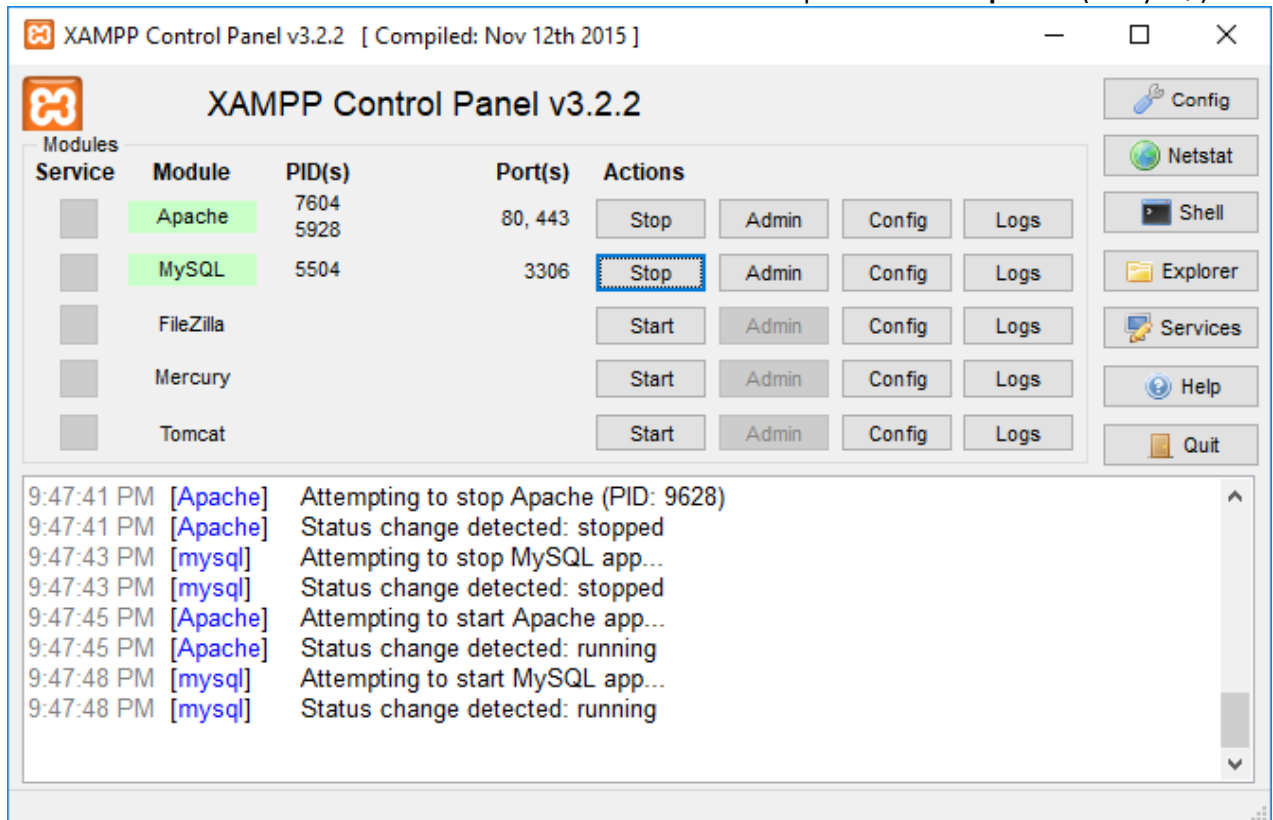
<body>

<?php
echo "<h1>Hello World </h1>";
?>

</body>
</html>
```

Step4 -

- Now double click on “**XAMPP CONTROL PANEL**” on desktop and START “**Apache**” (& MySQL)

**Step5 -**

- Type **localhost** on your browser and press enter:
It will show the following:



Step6 -

- Now type the following on browser:
http://localhost/eswar/
Below screenshot shows php files created under folder “eswar”

**Step7 -**

- Click on “**hello.php**” and it will show the following:
-



PHP – Variable Types

The main way to store information in the middle of a PHP program is by using a variable.

Here are the most important things to know about variables in PHP.

- All variables in PHP are denoted with a leading dollar sign (\$).
- The value of a variable is the value of its most recent assignment.
- Variables are assigned with the = operator, with the variable on the left-hand side and the expression to be evaluated on the right.
- Variables can, but do not need, to be declared before assignment.
- Variables in PHP do not have intrinsic types - a variable does not know in advance whether it will be used to store a number or a string of characters.
- Variables used before they are assigned have default values.
- PHP does a good job of automatically converting types from one to another when necessary.
- PHP variables are Perl-like.

PHP has a total of eight data types which we use to construct our variables –

- **Integers** – are whole numbers, without a decimal point, like 4195.
- **Doubles** – are floating-point numbers, like 3.14159 or 49.1.
- **Booleans** – have only two possible values either true or false.
- **NULL** – is a special type that only has one value: NULL.
- **Strings** – are sequences of characters, like 'PHP supports string operations.'
- **Arrays** – are named and indexed collections of other values.
- **Objects** – are instances of programmer-defined classes, which can package up both other kinds of values and functions that are specific to the class.
- **Resources** – are special variables that hold references to resources external to PHP (such as database connections).

The first five are *simple types*, and the next two (arrays and objects) are compound - the compound types can package up other arbitrary values of arbitrary type, whereas the simple types cannot.

Variable Scope

Scope can be defined as the range of availability a variable has to the program in which it is declared. PHP variables can be one of four scope types –

- Local variables
- Function parameters
- Global variables
- Static variables

Variable Naming

Rules for naming a variable is –

- Variable names must begin with a letter or underscore character.
- A variable name can consist of numbers, letters, underscores but you cannot use characters like +, -, %, (,), . & , etc

There is no size limit for variables.

Example Program on integers

```
<?php
    $num1 = 120;
    $num2 = 3.14;
    $sum= $num1 + $num2;
    print("result= $sum<br>");
    print("$num1 + $num2 = $sum");
?>
```

OUTPUT:

```
result= 123.14  
120 + 3.14 = 123.14
```

PHP Integer

An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.

Rules for integers:

- An integer must have at least one digit
- An integer must not have a decimal point
- An integer can be either positive or negative

NOTE:The PHP var_dump() function returns the data type and value:

```
<!DOCTYPE html>  
<html>  
<body>
```

```
<?php  
$x = 5985;  
var_dump($x);  
?>
```

```
</body>  
</html>  
OUTPUT: int(5985)
```

Example Program on Boolean

```
<?php  
    if (TRUE)  
        print("True statement <br>");  
    else  
        print("Else Statement <br>");  
?>
```

OUTPUT:

True statement

Strings

They are sequences of characters, like "PHP supports string operations". Following are valid examples of string

```
$string_1 = "This is a string in double quotes";  
$string_2 = 'This is a somewhat longer, singly quoted string';  
$string_39 = "This string has thirty-nine characters";  
$string_0 = ""; // a string with zero characters
```

Singly quoted strings are treated almost literally, whereas doubly quoted strings replace variables with their values as well as specially interpreting certain character sequences.

Example:

```
<?php
    $variable = "Eeswar";
    $literally = 'My name is $variable';

print($literally);
print "<br>";

    $literally = "My name is $variable";
print($literally);
?>
```

OUTPUT:

```
My name is $variable
My name is Eeswar
```

PHP Array

An array stores multiple values in one single variable.

```
<!DOCTYPE html>
<html>
<body>

<?php
$cars = array("Volvo", "BMW", "Toyota");
var_dump($cars);
?>

</body>
</html>
```

OUTPUT:

```
array(3) { [0]=> string(5) "Volvo" [1]=> string(3) "BMW" [2]=> string(6) "Toyota" }
```

PHP Array-ex2

```
<!DOCTYPE html>
<html>
<body>

<?php
$exarray = array(1,2,4,67.89,"vjit");
var_dump($exarray);
?>

</body>
</html>
```

OUTPUT:

```
array(5) { [0]=>int(1) [1]=>int(2) [2]=>int(4) [3]=> float(67.89) [4]=>
string(4) "vjit"}
```

PHP echo and print Statements

echo and print are more or less the same. They are both used to output data to the screen.

The differences is: echo has no return value while print has a return value of 1 so it can be used in expressions. echo can take multiple parameters (although such usage is rare) while print can take one argument. echo is marginally faster than print.

PHP Object

- An object is a data type which stores data and information on how to process that data.
- In PHP, an object must be explicitly declared.
- First we must declare a class of object. For this, we use the class keyword. A class is a structure that can contain properties and methods:

```
<!DOCTYPE html>
<html>
<body>

<?php
class Display {
function Display() {
    $this->model = "HelloPHP";
    }
}
// create an object
$obj1 = new Display();

// show object properties
echo $obj1->model;
?>

</body>
</html>
```

OUTPUT:

HelloPHP

PHP String Functions

- `echo strlen("Hello world!"); // outputs 12`
- `echo str_word_count("Hello world!"); // outputs 2`
- `echo strrev("Hello world!"); // outputs !dlrowolleH`
- `echo strpos("Hello world!", "world"); // outputs`
- `echo str_replace("world", "Dolly", "Hello world!"); // outputs Hello Dolly!`

PHP Operators

Operators are used to perform operations on variables and values.

PHP divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Increment/Decrement operators
- Logical operators
- String operators
- Array operators

PHP Arithmetic operators

Operator	Name	Example	Result
+	Addition	$\$x + \y	Sum of $\$x$ and $\$y$
-	Subtraction	$\$x - \y	Difference of $\$x$ and $\$y$
*	Multiplication	$\$x * \y	Product of $\$x$ and $\$y$
/	Division	$\$x / \y	Quotient of $\$x$ and $\$y$
%	Modulus	$\$x \% \y	Remainder of $\$x$ divided by $\$y$
**	Exponentiation	$\$x ** \y	Result of raising $\$x$ to the $\$y$ 'th power

PHP Assignment Operators

Assignment	Same as...	Description
$x = y$	$x = y$	The left operand gets set to the value of the expression
$x += y$	$x = x + y$	Addition
$x -= y$	$x = x - y$	Subtraction
$x *= y$	$x = x * y$	Multiplication
$x /= y$	$x = x / y$	Division
$x \% = y$	$x = x \% y$	Modulus

PHP Comparison Operators

The PHP comparison operators are used to compare two values (number or string):

Operator	Name	Example	Result
==	Equal	$\$x == \y	Returns true if $\$x$ is equal to $\$y$

===	Identical	$\$x === \y	Returns true if $\$x$ is equal to $\$y$, and they are of the same type
!=	Not equal	$\$x != \y	Returns true if $\$x$ is not equal to $\$y$
<>	Not equal	$\$x <> \y	Returns true if $\$x$ is not equal to $\$y$
!==	Not identical	$\$x !== \y	Returns true if $\$x$ is not equal to $\$y$, or they are not of the same type
>	Greater than	$\$x > \y	Returns true if $\$x$ is greater than $\$y$
<	Less than	$\$x < \y	Returns true if $\$x$ is less than $\$y$
>=	Greater than or equal to	$\$x >= \y	Returns true if $\$x$ is greater than or equal to $\$y$
<=	Less than or equal to	$\$x <= \y	Returns true if $\$x$ is less than or equal to $\$y$

PHP Increment / Decrement Operators

- The PHP increment operators are used to increment a variable's value.
- The PHP decrement operators are used to decrement a variable's value.

Operator	Name	Description
++ $\$x$	Pre-increment	Increments $\$x$ by one, then returns $\$x$
$\$x$ ++	Post-increment	Returns $\$x$, then increments $\$x$ by one
-- $\$x$	Pre-decrement	Decrements $\$x$ by one, then returns $\$x$
$\$x$ --	Post-decrement	Returns $\$x$, then decrements $\$x$ by one

PHP Logical Operators

The PHP logical operators are used to combine conditional statements.

Operator	Name	Example	Result
And	And	$\$x$ and $\$y$	True if both $\$x$ and $\$y$ are true
Or	Or	$\$x$ or $\$y$	True if either $\$x$ or $\$y$ is true
Xor	Xor	$\$x$ xor $\$y$	True if either $\$x$ or $\$y$ is true, but not both
&&	And	$\$x \&\& \y	True if both $\$x$ and $\$y$ are true
	Or	$\$x \ \ \y	True if either $\$x$ or $\$y$ is true

!	Not	!\$x	True if \$x is not true
---	-----	------	-------------------------

PHP String Operators

PHP has two operators that are specially designed for strings.

Operator	Name	Example	Result
.	Concatenation	\$txt1 . \$txt2	Concatenation of \$txt1 and \$txt2
.=	Concatenation	\$txt1 .= \$txt2	Appends \$txt2 to \$txt1

PHP Array Operators

The PHP array operators are used to compare arrays.

Operator	Name	Example	Result
+	Union	\$x + \$y	Union of \$x and \$y
==	Equality	\$x == \$y	Returns true if \$x and \$y have the same key/value pairs
===	Identity	\$x === \$y	Returns true if \$x and \$y have the same key/value pairs
!=	Inequality	\$x != \$y	Returns true if \$x is not equal to \$y
<>	Inequality	\$x <> \$y	Returns true if \$x is not equal to \$y
!==	Non-	\$x !== \$y	Returns true if \$x is not identical to \$y

PHP Conditional Statements

Very often when you write code, you want to perform different actions for different conditions. You can use conditional statements in your code to do this.

In PHP we have the following conditional statements:

- **if statement** - executes some code if one condition is true
- **if...else statement** - executes some code if a condition is true and another code if that condition is false
- **if...elseif...else statement** - executes different codes for more than two conditions
- **switch statement** - selects one of many blocks of code to be executed

PHP Loops

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal code-lines in a script, we can use loops to perform a task like this.

In PHP, we have the following looping statements:

- **while** - loops through a block of code as long as the specified condition is true
- **do...while** - loops through a block of code once, and then repeats the loop as long as the specified condition is true
- **for** - loops through a block of code a specified number of times
- **foreach** - loops through a block of code for each element in an array

The PHP for Loop

The for loop is used when you know in advance how many times the script should run.

Syntax

```
for (init counter; test counter; increment counter) {  
    code to be executed;  
}
```

The PHP foreach Loop

The foreach loop works only on arrays, and is used to loop through each key/value pair in an array.

Syntax

```
foreach ($array as $value) {  
    code to be executed;  
}
```

For every loop iteration, the value of the current array element is assigned to \$value and the array pointer is moved by one, until it reaches the last array element.

The following example demonstrates a loop that will output the values of the given array (\$colors):

Example:

```
<?php  
$colors = array("red", "green", "blue", "yellow");  
  
foreach ($colors as $value) {  
    echo "$value <br>";  
}  
?>
```

P1.html

```
<html>  
<body>  
  
<form action="welcome.php" method="post">
```

```
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>
```

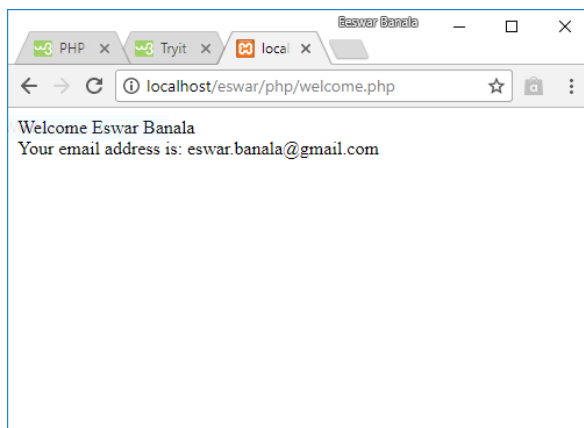
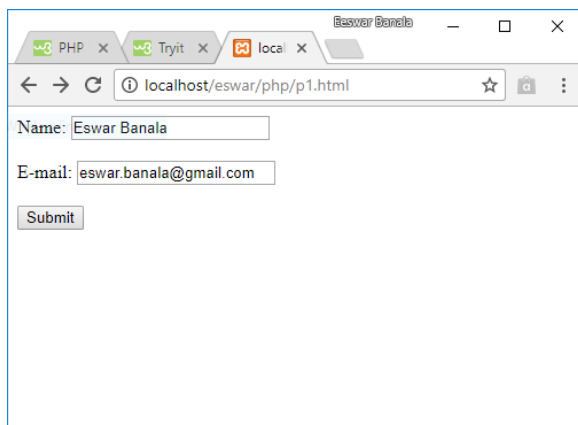
```
</body>
</html>
```

Welcome.php

```
<html>
<body>
```

```
Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>
```

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



P4.php (internal)

```
<!DOCTYPE HTML>
<html>
<head>
```

```
</head>
<body>

<?php
// define variables and set to empty values
$name = $email = $gender = $comment = $website = "";

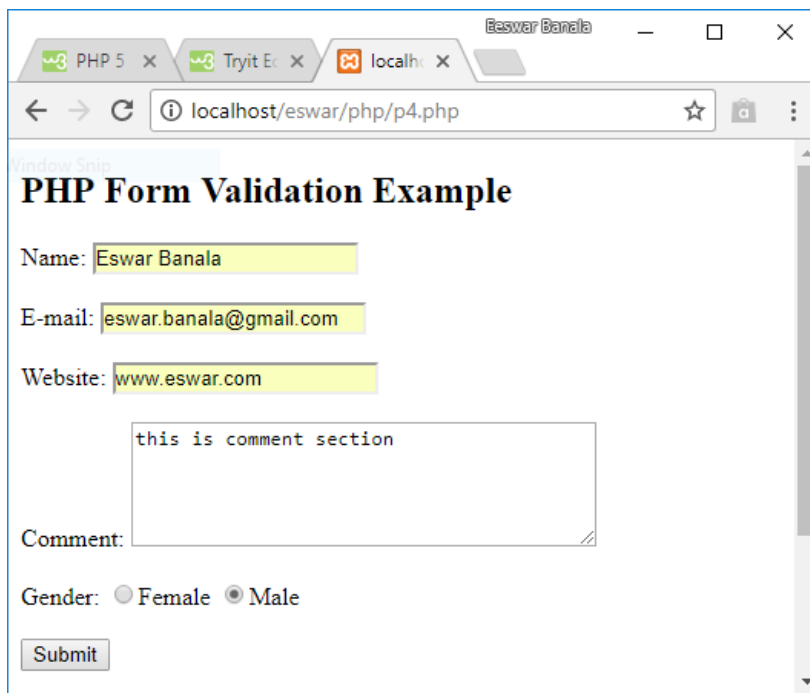
if ($_SERVER["REQUEST_METHOD"] == "POST")
{
    $name = $_POST["name"];
    $email = $_POST["email"];
    $website = $_POST["website"];
    $comment = $_POST["comment"];
    $gender = $_POST["gender"];
}

?>

<h2>PHP Form Validation Example</h2>
<form method="post" action="<?php $_SERVER["PHP_SELF"];?>">
    Name: <input type="text" name="name">
<br><br>
    E-mail: <input type="text" name="email">
<br><br>
    Website: <input type="text" name="website">
<br><br>
    Comment: <textarea name="comment" rows="5" cols="40"></textarea>
<br><br>
    Gender:
<input type="radio" name="gender" value="female">Female
<input type="radio" name="gender" value="male">Male
<br><br>
<input type="submit" name="submit" value="Submit">
</form>

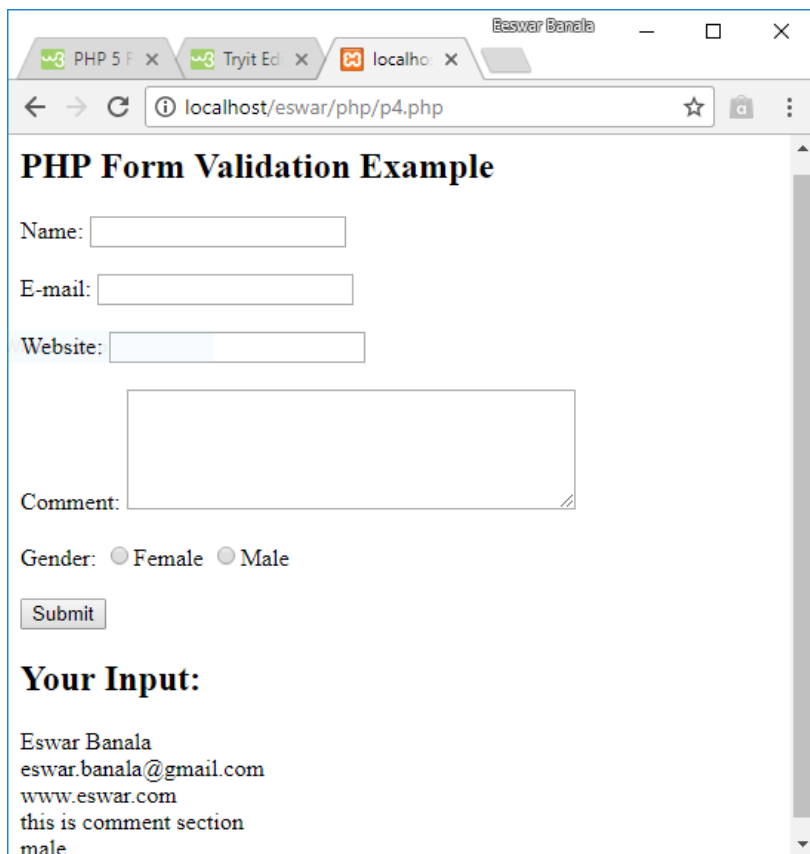
<?php
echo "<h2>Your Input:</h2>";
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;
echo "<br>";
echo $gender;
?>

</body>
</html>
```



A screenshot of a web browser window titled "Eswar Banala" showing a PHP form validation example. The browser's address bar displays "localhost/eswar/php/p4.php". The form is titled "PHP Form Validation Example" and contains the following fields and controls:

- Name:
- E-mail:
- Website:
- Comment:
- Gender: Female Male
- Submit:



A screenshot of a web browser window titled "Eswar Banala" showing the same PHP form validation example. The browser's address bar displays "localhost/eswar/php/p4.php". The form is titled "PHP Form Validation Example" and contains the following fields and controls:

- Name:
- E-mail:
- Website:
- Comment:
- Gender: Female Male
- Submit:

Your Input:

Eswar Banala
eswar.banala@gmail.com
www.eswar.com
this is comment section
male

Both GET and POST create an array (e.g. `array(key => value, key2 => value2, key3 => value3, ...)`). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.

Both GET and POST are treated as `$_GET` and `$_POST`. These are superglobals, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

`$_GET` is an array of variables passed to the current script via the URL parameters.

`$_POST` is an array of variables passed to the current script via the HTTP POST method.

When to use GET?

Information sent from a form with the GET method is **visible to everyone** (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.

GET may be used for sending non-sensitive data.

Note: GET should NEVER be used for sending passwords or other sensitive information!

When to use POST?

Information sent from a form with the POST method is **invisible to others** (all names/values are embedded within the body of the HTTP request) and has **no limits** on the amount of information to send.

Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.

However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

Developers prefer POST for sending form data.

PHP - File Upload

Some rules to follow for the File Upload in PHP

- Make sure that the form uses **method="post"**
- The form also needs the attribute: **enctype="multipart/form-data"**. It specifies which content-type to use when submitting the form

Note:The type="file" attribute of the <input> tag shows the input field as a file-select control, with a "Browse" button next to the input control

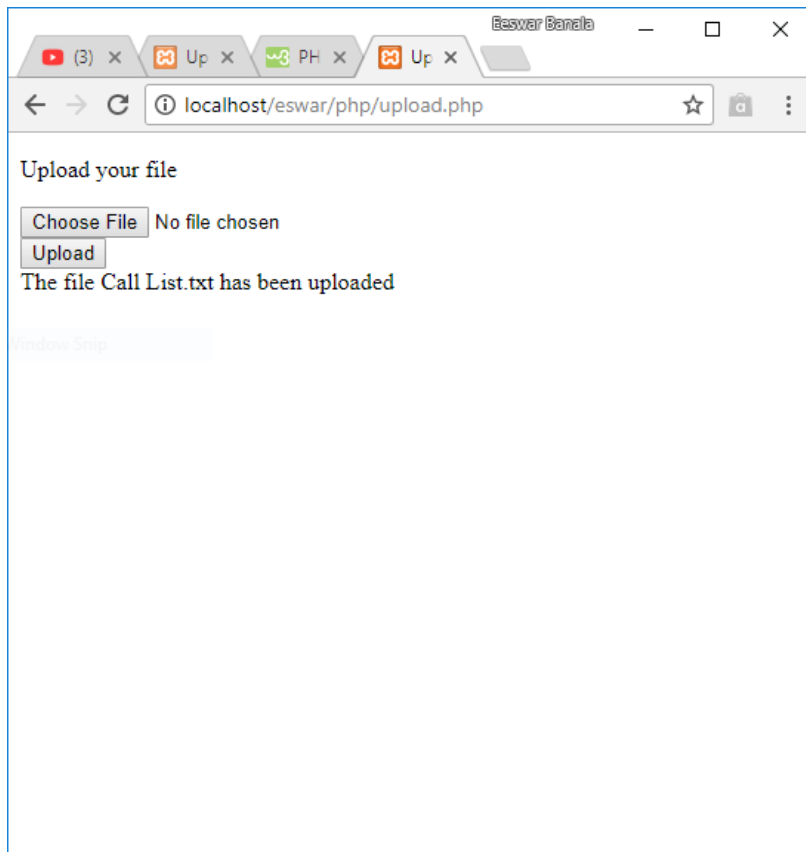
Upload.php

```
<!DOCTYPE html>
<html>
<head>
<title>Upload your files</title>
</head>
<body>
<form enctype="multipart/form-data"action="upload.php" method="POST">
<p>Upload your file</p>
<input type="file" name="uploaded_file"></input><br><br>
<input type="submit" value="Upload"></input>
</form>
</body>
</html>

<?PHP
if(!empty($_FILES['uploaded_file']))
{
    $dir = "uploads/";
    $targetFile = $dir. basename( $_FILES['uploaded_file']['name']);
if(move_uploaded_file($_FILES['uploaded_file']['tmp_name'], $targetFile))
    {

echo "The file ". basename( $_FILES['uploaded_file']['name']).
" has been uploaded";
    } else{
echo "There was an error uploading the file, please try again!";
    }
}
?>
```

NOTE: Before executing the above program create a folder with name **"uploads"** in **"C:\xampp\htdocs\eswar\php"**. Uploaded file will be saved in this uploads folder.



PHP Connect to MySQL

MySQL database using: **MySQLi extension** (the "i" stands for improved)

Steps to connect Database(MySql)

1) Open and Create a Connection to MySQL and database

Before we can access data in the MySQL database, we need to be able to connect to the server:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "wt";

// Create connection
$conn = new mysqli($servername, $username, $password,$dbname);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";
?>
```

2) Execute an SQL Query and Handle the results

```
$sql = "SELECT empid, empname, esal FROM emp";
$result = $conn->query($sql);

if ($result->num_rows > 0)
{
    // output data of each row
    while($row = $result->fetch_assoc())
    {
        echo "ID: " . $row["empid"]. " Name: " . $row["empname"]. " Sal:" .
        $row["esal"]. "<br>";
    }
}
else
{
    echo "0 results";
}
```

3) Close the Connection

The connection will be closed automatically when the script ends. To close the connection before, use the following:

```
$conn->close();
```

empid	empname	esal ▲ 1
1202	Employee2	123
1201	Employee1	2345
1205	Employee5	12345
1204	Employee4	23145
1203	Employee3	23455

Db4selectall.php

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbname="wt";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn)
{
die("Connection failed: " . mysqli_connect_error());
}
else
    echo "Connected succesfully..<br>";

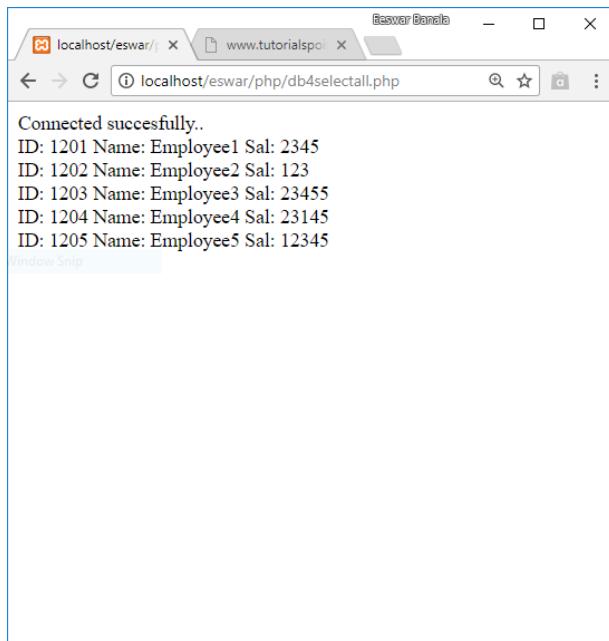
//selection of data
$sql = "SELECT empid, empname, esal FROM emp";
$result = $conn->query($sql);

// output data of each row

if ($result->num_rows > 0)
{
while($row = $result->fetch_assoc())
{
echo "ID: " . $row["empid"]. " Name: " . $row["empname"]. " Sal: " .
$row["esal"]. "<br>";
}
}
else
{
echo "0 results";
}

//connection close
mysqli_close($conn);

?>
```



Log.php

```
<html>
<!--
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Login Form</title>
</head>
-->
<body>
<form method="post" action="vallogin.php" >

<label for="users_email">Email</label>
<input type="text" name="users_email" id="users_email"><br><br>

    <label for="users_pass">Password</label>
    <input type="password" name="users_pass"
id="users_pass"></input><br><br>

    <input type="submit" value="Login"/>
<input type="reset" value="Reset"/>

</form>
</body>
</html>
```

Vallogin.php

```
<?php

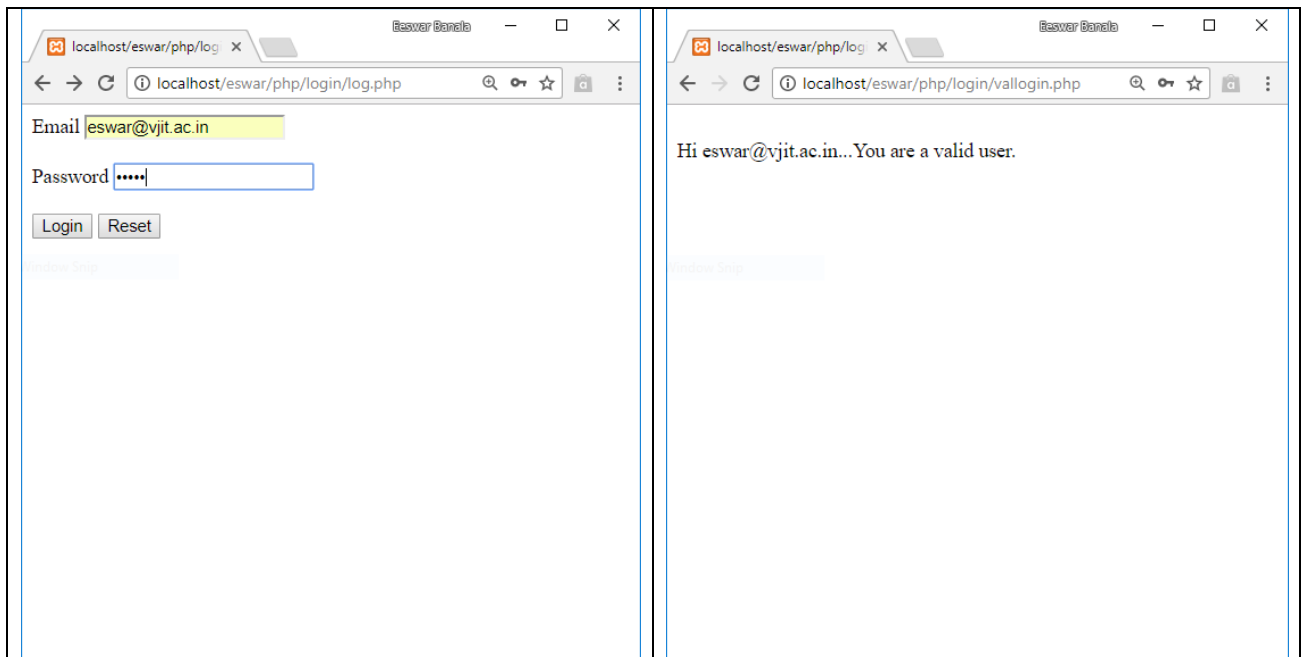
$servername = "localhost";
$username = "root";
$password = "";
$dbname="eswar";
```

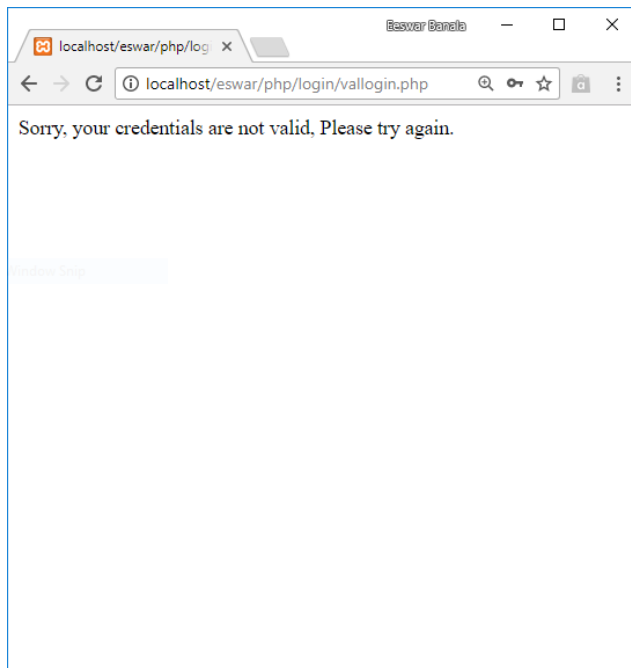
```
// Grab User submitted information
$email = $_POST["users_email"];
$pass = $_POST["users_pass"];

// Connect to the database
$con= mysqli_connect($servername, $username, $password, $dbname);
// Make sure we connected successfully
if(!$con)
{
die('Connection Failed'.mysql_error());
}

//selection of data
$sql= "SELECT email, password FROM users WHERE email = '$email'";
$result = $con->query($sql);
$row = $result->fetch_assoc();

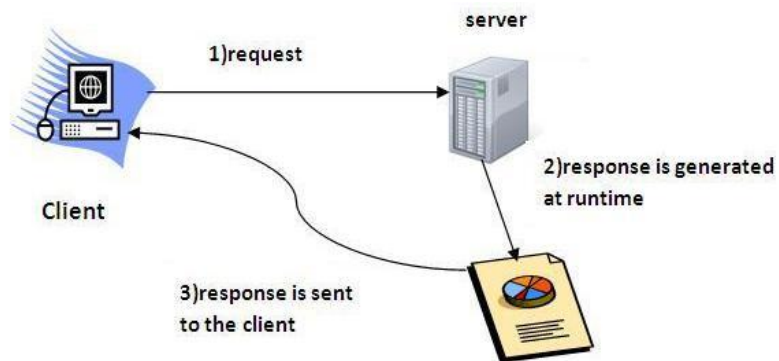
if($row["email"]==$email && $row["password"]==$pass)
echo "<br> Hi ". $email . "...You are a valid user.";
else
echo"Sorry, your credentials are not valid, Please try again.";
?>
```





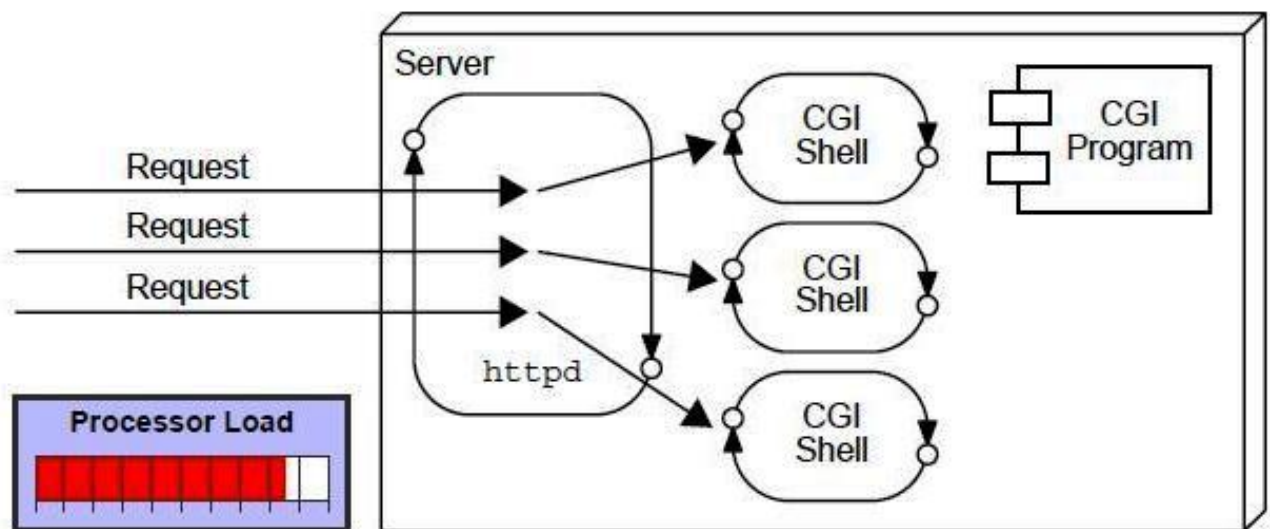
Servlets

- Servlet is a technology i.e. used to create web application.
- Servlet is an API that provides many interfaces and classes including documentations.
- Servlet is an interface that must be implemented for creating any servlet.
- Servlet is a class that extend the capabilities of the servers and respond to the incoming request. It can respond to any type of requests.
- Servlet is a web component that is deployed on the server to create dynamic web page.



CGI(Common Gateway Interface)

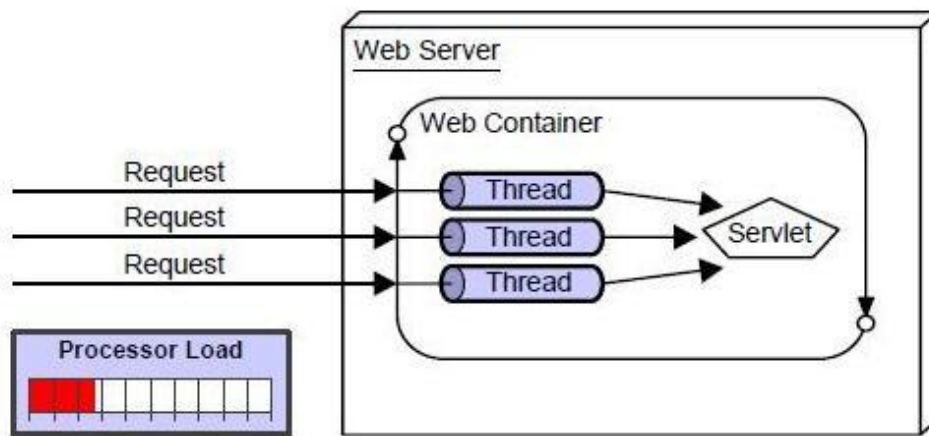
CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.



Disadvantages of CGI

There are many problems in CGI technology:

1. If number of clients increases, it takes more time for sending response.
2. For each request, it starts a process and Web server is limited to start processes.
3. It uses platform dependent language e.g. C, C++, perl.

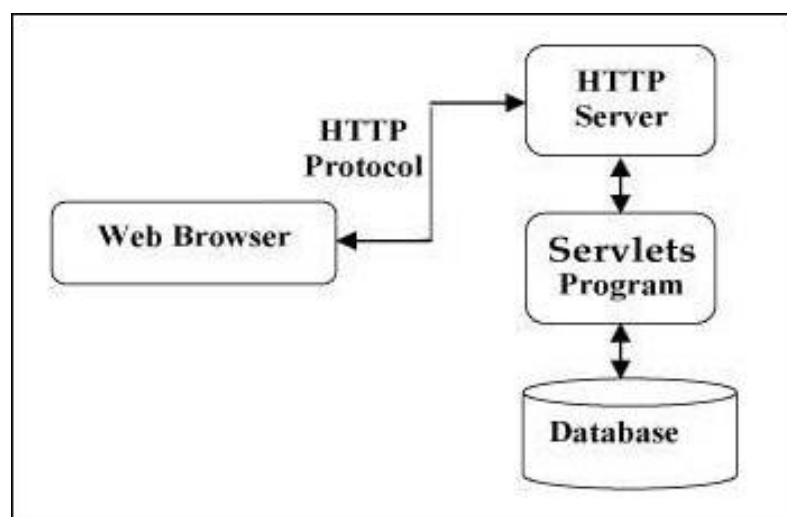


Advantages Of Servlet

There are many advantages of servlet over CGI. The web container creates threads for handling the multiple requests to the servlet. Threads have a lot of benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The basic benefits of servlet are as follows:

1. **better performance:** because it creates a thread for each request not process.
2. **Portability:** because it uses java language.
3. **Robust:** Servlets are managed by JVM so we don't need to worry about memory leak, garbage collection etc.
4. **Secure:** because it uses java language..

Servlets Architecture



Servlets Packages

- Java Servlets are Java classes run by a web server that has an interpreter that supports the Java Servlet specification.
- Servlets can be created using the **javax.servlet** and **javax.servlet.http** packages, which are a standard part of the Java's enterprise edition, an expanded version of the Java class library that supports large-scale development projects.

Servlets - Life Cycle

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet.

- The servlet is initialized by calling the **init()** method.
- The servlet calls **service()** method to process a client's request.
- The servlet is terminated by calling the **destroy()** method.
- Finally, servlet is garbage collected by the garbage collector of the JVM.

1) init() method

The init method is called only once. It is called only when the servlet is created, and not called for any user requests afterwards. So, it is used for one-time initializations, just as with the init method of applets.

```
public void init() throws ServletException
{
    // Initialization code...
}
```

2) service() method

The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.

```
public void service(ServletRequest request, ServletResponse response)
throws ServletException, IOException
{
}
```

- The service () method is called by the container and service method invokes doGet, doPost, doPut, doDelete, etc.
- The doGet() and doPost() are most frequently used methods with in each service request.

The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
public void doGet(HttpServletRequest request, HttpServletResponse
response)
throws ServletException, IOException
{
    // Servlet code
}
```

The doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
public void doPost(HttpServletRequest request, HttpServletResponse
response)
throws ServletException, IOException
{
    // Servlet code
}
```

3) The destroy() Method

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

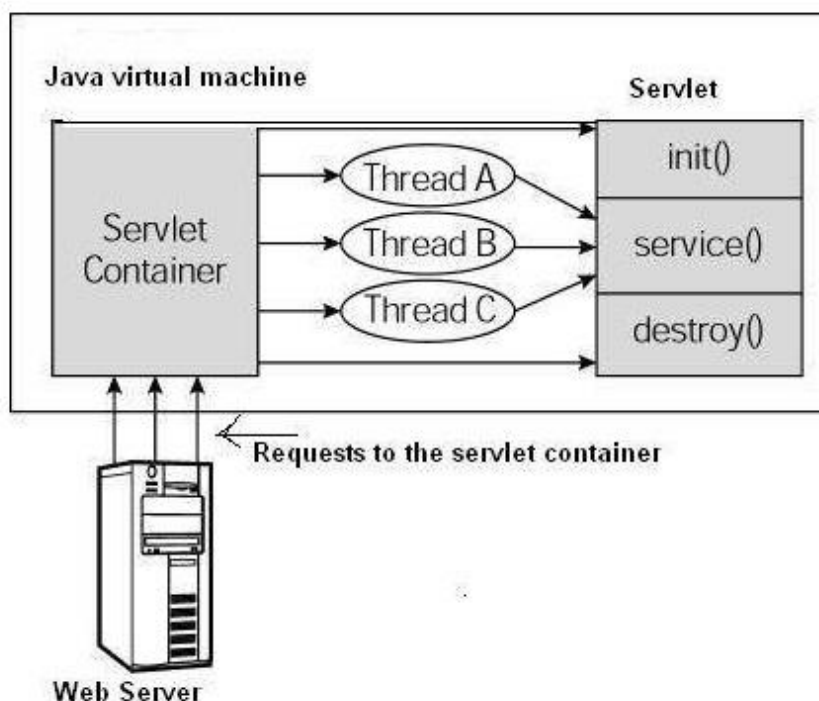
After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this –

```
public void destroy()
{
    // Finalization code...
}
```

Architecture Diagram

The following figure depicts a typical servlet life-cycle scenario.

- First the HTTP requests coming to the server are delegated to the servlet container.
- The servlet container loads the servlet before invoking the service() method.
- Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the service() method of a single instance of the servlet.



Steps to Running Your First Servlet

Once Tomcat is installed and configured, you can put it to work. Six steps take you from writing your servlet to running it. These steps are as follows:

1. Create a directory structure.

```
F:\DateApp(Root folder)
  |-----WEB-INF
    |-----classes
    |       |-----DateServlet.java
    |-----web.xml
```

2. Write the servlet source code(DateServlet.java). You need to import the javax.servlet package and the javax.servlet.http package in your source file.

3. Compile your source code and observe the .class file in respective package.

```
F:\DateApp\WEB-INF\classes\javac -d . DateServlet.java
```

4. Create a deployment descriptor(web.xml).

5. Set classpath for Tomcat

This PC properties >Advanced System Settings >Environment variables > System Variables > New > Variable-name = CLASSPATH and variable-value = "C:\Program Files (x86)\Apache Software Foundation\Tomcat 9.0\lib\servlet-api.jar;"> OK > OK > OK
Note: (check if tomcat path is set correctly or not - in command prompt type: "javapjavax.servlet.GenericServlet").

6. Start and Run Tomcat – open Tomcat9.exe

```
"C:\Program Files (x86)\Apache Software Foundation\Tomcat 9.0\bin\Tomcat9.exe"
```

7. Deployment of servlet - Copy root web application folder(DateApp) and paste in "Tomcat-home\webapps" folder.

8. Call your servlet from a web browser.

<http://localhost:8080/DateApp/test1>

```
|           |-----url-pattern in web.xml file
|-----Root web application folder
```

DateServlet.java

```
package org.it.servlet;
import javax.servlet.*;
import java.io.*;
import java.util.*;
public class DateServlet extends GenericServlet
{
    public void service(ServletRequest req, ServletResponse res) throws
ServletException, IOException
    {
        PrintWriter pw=null;
        Date date=null;
        //set response content type
        res.setContentType("text/html");
        //get PrintWriter object
        pw=res.getWriter();
        //write request processing
        date =new Date();
        //write output to response object
        pw.println("<h1 style ='text-align:center'>Date and Time is " +
date + "</h1>");
        //close PrintWriter
        pw.close();
    }
}
```

Web.xml(in WEB-INF)

```
<web-app>
    <servlet>
        <servlet-name>date</servlet-name>
        <servlet-class>org.it.servlet.DateServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>date</servlet-name>
        <url-pattern>/test1</url-pattern>
    </servlet-mapping>
</web-app>
```

OUTPUT:

Date and Time is Sun Mar 04 10:26:57 IST 2018



GenericServlet Class

- This class defines a generic, protocol-independent servlet.
- To write a generic servlet, you need only override the abstract service method.

Syntax

public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException

HttpServlet Class

Provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site. A subclass of HttpServlet must override at least one method, usually one of these:

- doGet(), if the servlet supports HTTP GET requests
- doPost(), for HTTP POST requests
- init() and destroy(), to manage resources that are held for the life of the servlet

Syntax

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

```
{  
}
```

javax.servlet - Package

The javax.servlet package contains a number of classes and interfaces that describe and define the contracts between a servlet class and the runtime environment provided for an instance of such a class by a conforming servlet container.

ServletResponse - Interface

- Defines an object to assist a servlet in sending a response to the client. The servlet container creates a **ServletResponse** object and passes it as an argument to the servlet's service method.
- To send character data, use the **PrintWriter** object returned by **getWriter()**.

Methods – ServletResponse interface

- 1) **setContentType(String)** - Sets the content type of the response being sent to the client, if the response has not been committed yet.
- 2) **getWriter()** - Returns a **PrintWriter** object that can send character text to the client.

Returns:

A **PrintWriter** object that can return character data to the client

Throws:

java.io.IOException - if an input or output exception occurred

- 3) **reset()** - Clears any data that exists in the buffer as well as the status code and headers.

ServletRequest – Interface

- Defines an object to provide client request information to a servlet. The servlet container creates a **ServletRequest** object and passes it as an argument to the servlet's service method.
- A **ServletRequest** object provides data including parameter name, values and attributes.

Methods – ServletResponse interface – (Reading Form Data using Servlet)

- 1) **getParameter(String)** - Returns the value of a request parameter as a **String**, or null if the parameter does not exist.

NOTE: Use this method when parameter has only one value (Ex: Text Box).

Returns:

A String representing the single value of the parameter

- 2) **getParameterValues(String)** - Returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist.

NOTE: Use this method when parameter has more than one value (Ex: Check Box).

Returns:

An array of String objects containing the parameter's values

- 3) **getParameterNames()** – Call this method if you want a complete list of all parameters in the current request.

javax.servlet.http - Package

The javax.servlet.http package contains a number of classes and interfaces that describe and define the contracts between a servlet class running under the HTTP protocol and the runtime environment provided for an instance of such a class by a conforming servlet container.

HttpServletRequest - Interface

The servlet container creates an HttpServletRequest object and passes it as an argument to the servlet's service methods (doGet, doPost, etc).

Ex: Reading Form Data using Servlet**HelloFormPost.html(C:\xampp\tomcat\webapps\LogAppPost\HelloForm.html)**

```
<html>
<body>
<form action = "HelloFormPost" method = "POST">
    First Name: <input type = "text" name = "first_name">
<br><br>
    Last Name: <input type = "text" name = "last_name" />
    <br><br>
<input type = "submit" value = "Submit" />
</form>
</body>
</html>
```

HelloFormPost.java(C:\xampp\tomcat\webapps\LogAppPost\WEB-INF\classes\HelloFormPost.java)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloFormPost extends HttpServlet {

    public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

        // Set response content type
        response.setContentType("text/html");
```

```

PrintWriter out = response.getWriter();
out.println("<b>First Name</b>: "
           + request.getParameter("first_name") + "<br>" +
"<b>Last Name</b>: "
           + request.getParameter("last_name") + "<br>");
    }
}

```

The screenshot shows a Windows command prompt window with the following text:

```

C:\Windows\System32\cmd.exe
C:\xampp\tomcat\webapps\LogAppPost\WEB-INF\classes>javac HelloFormPost.java
C:\xampp\tomcat\webapps\LogAppPost\WEB-INF\classes>

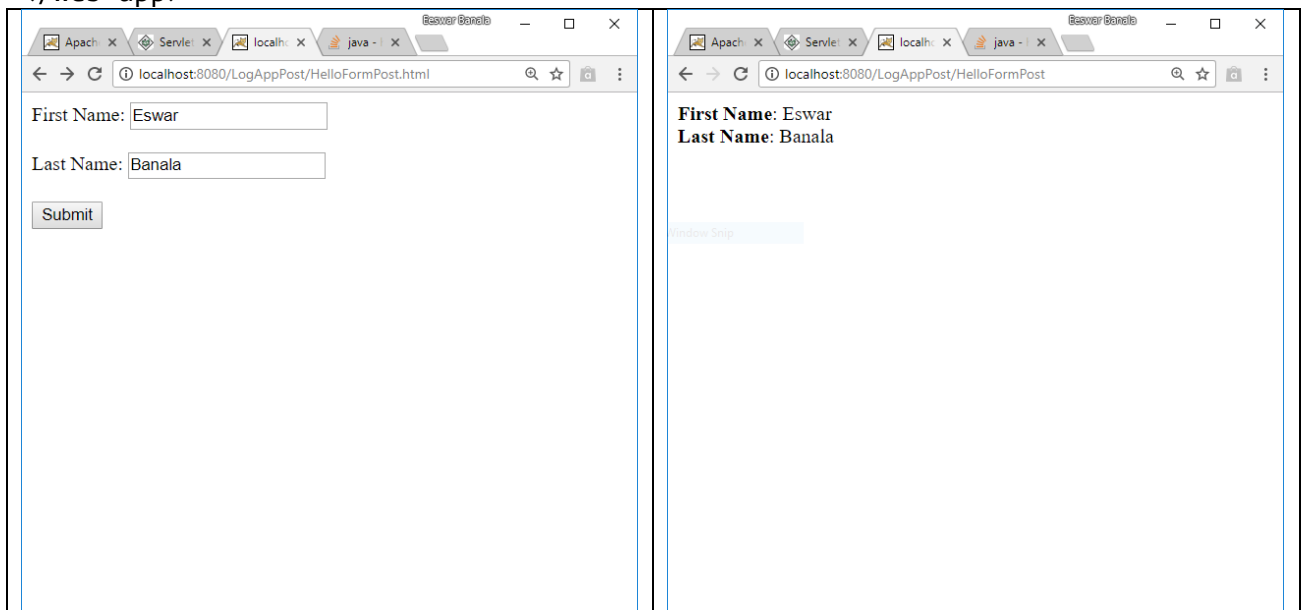
```

web.xmlC:\xampp\tomcat\webapps\LogAppPost\WEB-INF\web.xml)

```

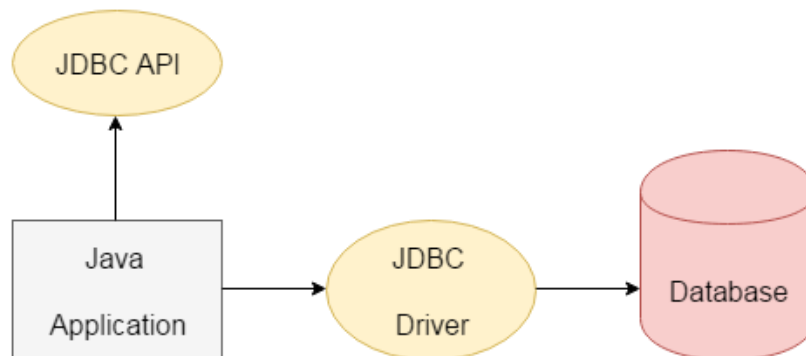
<web-app>
<servlet>
<servlet-name>HelloFormPost</servlet-name>
<servlet-class>HelloFormPost</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>HelloFormPost</servlet-name>
<url-pattern>/HelloFormPost</url-pattern>
</servlet-mapping>
</web-app>

```



Introduction to JDBC

Java JDBC is a java API to connect and execute query with the database. JDBC API uses jdbc drivers to connect with the database.



Why use JDBC

Before JDBC, ODBC API was the database API to connect and execute query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

JDBC Driver

JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

1) JDBC-ODBC bridge driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

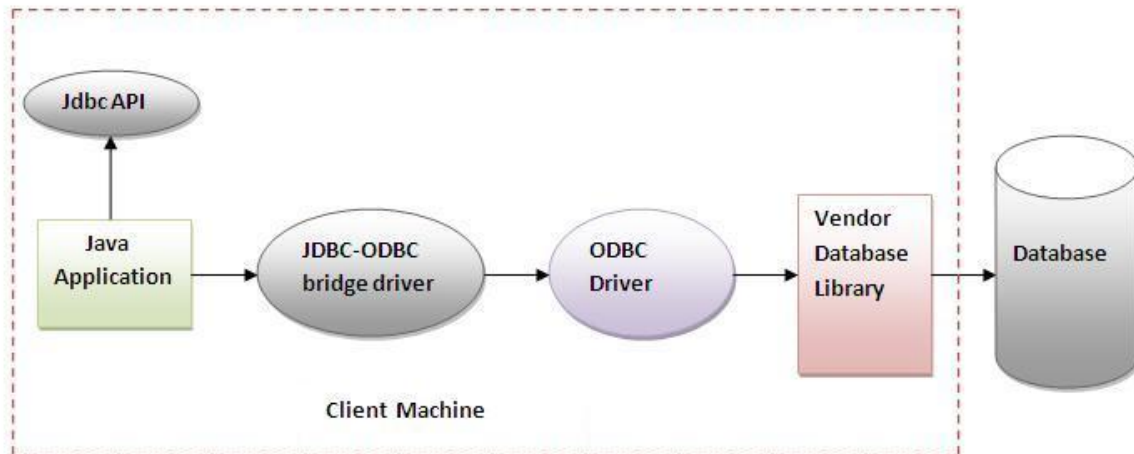


Figure- JDBC-ODBC Bridge Driver

Advantages:

- easy to use.
- can be easily connected to any database.

Disadvantages:

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

2) Native-API driver

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

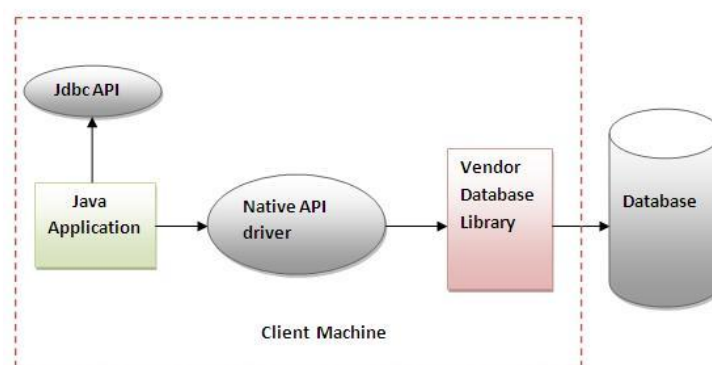


Figure- Native API Driver

Advantage:

- performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

- The Native driver needs to be installed on the each client machine.
- The Vendor client library needs to be installed on client machine.

3) Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

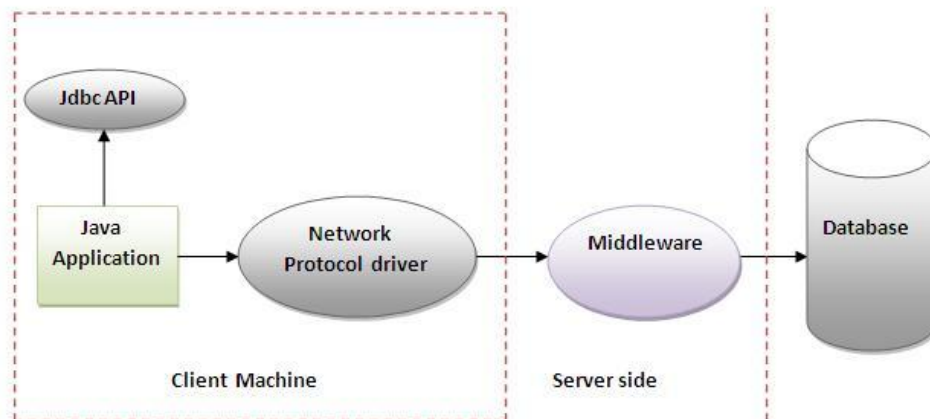


Figure- Network Protocol Driver

Advantage:

- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

4) Thin driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

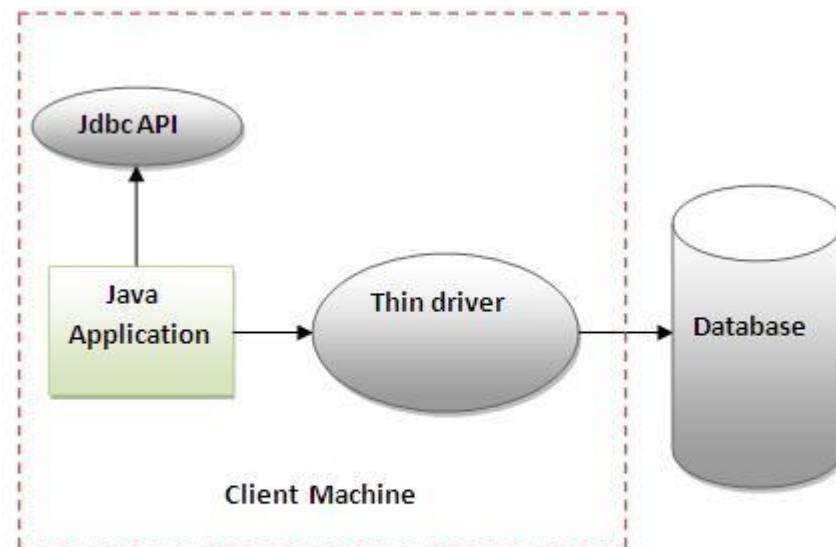


Figure- Thin Driver

Advantage:

- Better performance than all other drivers.
- No software is required at client side or server side.

Disadvantage:

- Drivers depends on the Database.

5 Steps to connect to the database in java

There are 5 steps to connect any java application with the database in java using JDBC. They are as follows:

- Register the driver class
- Creating connection
- Creating statement
- Executing queries
- Closing connection

Example to connect to the mysql database in java

For connecting java application with the mysql database, you need to follow 5 steps to perform database connectivity.

In this example we are using MySQL as the database. So we need to know following informations for the mysql database:

1. **Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**.
2. **Connection URL:** The connection URL for the mysql database is **jdbc:mysql://localhost:3306/sonoo** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and sonoo is the database name. We may use any database, in such case, you need to replace the sonoo with your database name.
3. **Username:** The default username for the mysql database is **root**.
4. **Password:** Password is given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

Let's first create a table in the mysql database, but before creating table, we need to create database first.

```
create database sonoo;
use sonoo;
create table emp(id int(10),name varchar(40),age int(3));
```

Example to Connect Java Application with mysql database

In this example, sonoo is the database name, root is the username and password.

```
import java.sql.*;
class ExSqlCon{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/sonoo","root","root");
//here sonoo is database name, root is username and password
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getInt(3));
con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

OUTPUT

```
1201 eswar 12
1208 Banala 25
1216 Koti 30
1217 Kiran 35
```

mysql(MySQL5.7 command line client)

```
mysql> create databsesonoo;
```

- Use sonoo;
- Create table emp(id int(10), name varchar(40), age int(3));
- Insert into emp values(1201,'eswar',15);
- Select * from emp;

Setting class path for mysql-connector.jar(in eclipse)

Select project in project explorer>right click> properties> libraries menu> Add External JARs > browse and select the file “ mysql-connector.jar” file> open>ok

Example to connect to the Oracle database in java

For connecting java application with the oracle database, you need to follow 5 steps to perform database connectivity. In this example we are using Oracle10g as the database. So we need to know following information for the oracle database:

1. **Driver class:** The driver class for the oracle database is **oracle.jdbc.driver.OracleDriver**.
2. **Connection URL:** The connection URL for the oracle10G database is **jdbc:oracle:thin:@localhost:1521:xe** where jdbc is the API, oracle is the database, thin is the driver, localhost is the server name on which oracle is running, we may also use IP address, 1521 is the port number and XE is the Oracle service name. You may get all these information from the tnsnames.ora file.
3. **Username:** The default username for the oracle database is **system**.
4. **Password:** Password is given by the user at the time of installing the oracle database.

F:\Subjects\WT\18-19\RegisterApp\Register.html

```

<html>
  <body>
    <form action = "RegisterServlet" method = "POST">
      <Center><h1> Registration Page </h1></center>

      First Name: <input type = "text" name = "first_name">
      <br /></br/>
      Last Name: <input type = "text" name = "last_name" >
      </br></br>
      Email: <input type = "text" name = "email_id">
      <br /></br/>
      Password: <input type = "text" name = "pwd">
      <br /></br/>
      <input type = "submit" value = "Register" />
    </form>
  </body>
</html>

```

F:\Subjects\WT\18-19\RegisterApp\WEB-INF\classes RegisterServlet.java

```

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
// Extend HttpServlet class
public class RegisterServlet extends HttpServlet
{
    public void doPost(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException
    {
        // Set response content type
        response.setContentType("text/html");

        String fn=request.getParameter("first_name");
        String ln=request.getParameter("last_name");
        String ei=request.getParameter("email_id");
        String pw=request.getParameter("pwd");
        PrintWriter out = response.getWriter();
        out.println("<h1>Registration Details</h1><br>");
        out.println("<b>First Name</b>:" + fn + "<br>");
        out.println("<b>Last Name</b>:" + ln + "<br>");
        out.println("<b>Email</b>:" + ei + "<br>");
        out.println("<b>Password</b>:" + pw + "<br>");
        try
        {
            Class.forName("com.mysql.jdbc.Driver");
            Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/2016","root
","");
            String sql="INSERT INTO signup (fn,ln,email,pwd)" +
"VALUES(?,?,?,?)";
            PreparedStatement pst = con.prepareStatement(sql);
            pst.setString (1,fn); //grabed element String fn

```

```

        pst.setString (2,ln);
        pst.setString (3,ei);
        pst.setString (4,pw);
        pst.execute();
        out.println("<br>Record inserted successfully..
Registration was completed...</body></html>");
        con.close();
    }catch(Exception e)
    {
        out.println(e);
    }
}
}
}

```

F:\Subjects\WT\18-19\RegisterApp\WEB-INF\web.xml

```

<web-app>
  <servlet>
    <servlet-name>R</servlet-name>
    <servlet-class>RegisterServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>R</servlet-name>
    <url-pattern>/RegisterServlet</url-pattern>
  </servlet-mapping>
</web-app>

```

Servlets - Cookies Handling

Cookies are text files stored on the client computer and they are kept for various information tracking purpose. Java Servlets transparently supports HTTP cookies.

There are three steps involved in identifying returning users –

- Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
- Browser stores this information on local machine for future use.
- When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.
- Cookies are usually set in an HTTP header.

Servlet Cookies Methods

- **public void setMaxAge(int expiry)**
This method sets how much time (in seconds) should elapse before the cookie expires. If you don't set this, the cookie will last only for the current session.
- **public int getMaxAge()**
This method returns the maximum age of the cookie, specified in seconds, By default, -1 indicating the cookie will persist until browser shutdown.
- **public String getName()**
This method returns the name of the cookie. The name cannot be changed after creation.
- **public String getValue()**
This method gets the value associated with the cookie.
- **public void setDomain(String pattern)**

- **public String getDomain()**

Setting Cookies with Servlet

Setting cookies with servlet involves three steps –

(1) Creating a Cookie object – You call the Cookie constructor with a cookie name and a cookie value, both of which are strings.

```
Cookie cookie1 = new Cookie("cookie name", "value");
```

(2) Setting the maximum age – You use setMaxAge to specify how long (in seconds) the cookie should be valid. Following would set up a cookie for 2 hours.

```
cookie1.setMaxAge(60 * 60 * 2);
```

(3) Sending the Cookie into the HTTP response headers – You use response.addCookie to add cookies in the HTTP response header as follows –

```
response.addCookie(cookie1);
```

CookiesDemo.html(C:\xampp\tomcat\webapps\CookiesDemo\CookiesDemo.html)

```
<html>
<body>
<form action = "CookiesDemo" method = "GET">
    First Name: <input type = "text" name = "first_name">
<br />
    Last Name: <input type = "text" name = "last_name" /><br>
<input type = "submit" value = "Submit" />
</form>
</body>
</html>
```

CookiesDemo.java (C:\xampp\tomcat\webapps\CookiesDemo\WEB-INF\classes\CookiesDemo.java)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class CookiesDemo extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

    // Create cookies for first and last names.
    Cookie firstName = new Cookie("first_name", request.getParameter("first_name"));
    Cookie lastName = new Cookie("last_name", request.getParameter("last_name"));

    // Set expiry date after 2 Hrs for both the cookies.
    firstName.setMaxAge(60*60*2);
    lastName.setMaxAge(60*60*2);

    // Add both the cookies in the response header.
    response.addCookie(firstName );
    response.addCookie(lastName );

    // Set response content type
    response.setContentType("text/html");

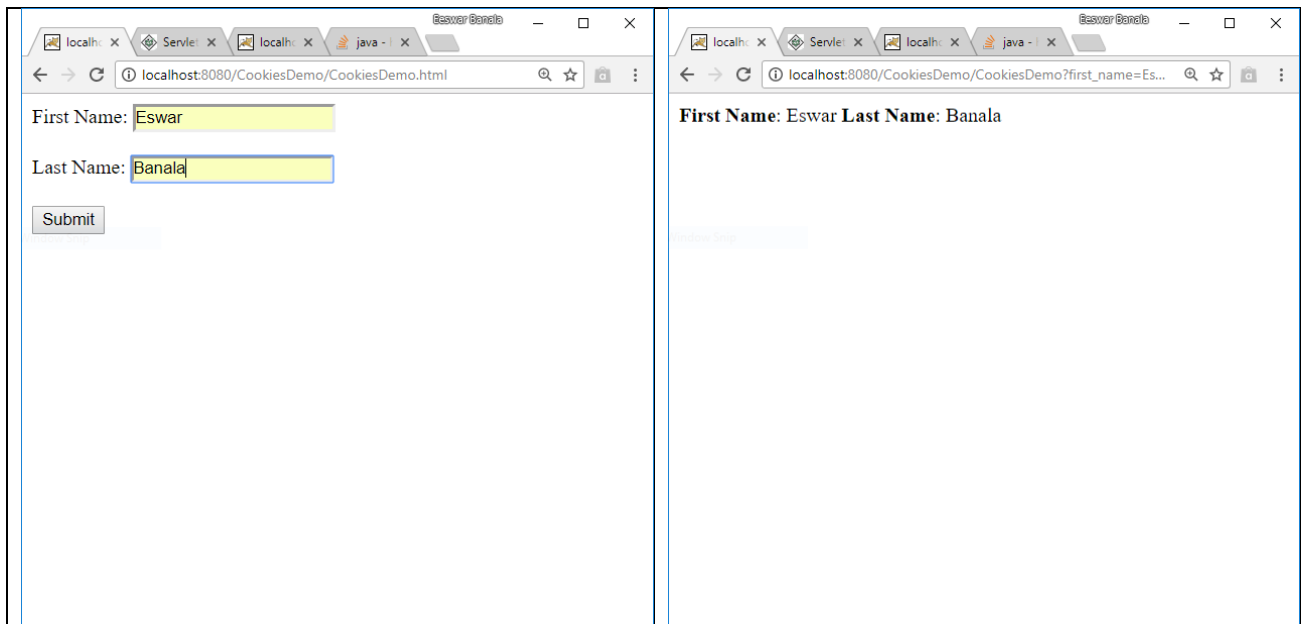
    PrintWriter out = response.getWriter();
```

```
out.println("<b>First Name</b>: "
           + request.getParameter("first_name") + "\n" +
" <b>Last Name</b>: "
           + request.getParameter("last_name") + "\n"
           );
    }
}
```

web.xml(C:\xampp\tomcat\webapps\CookiesDemo\WEB-INF\web.xml)

```
<web-app>
    <servlet>
        <servlet-name>cookies</servlet-name>
        <servlet-class>CookiesDemo</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>cookies1</servlet-name>
        <servlet-class>ReadCookiesDemo</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>cookies</servlet-name>
        <url-pattern>/CookiesDemo</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>cookies1</servlet-name>
        <url-pattern>/ReadCookiesDemo</url-pattern>
    </servlet-mapping>
</web-app>
```



ReadCookiesDemo.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

```
// ExtendHttpServlet class
public class ReadCookiesDemo extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

        //Cookie cookie = null;
        Cookie[] cookies = null;

        // Get an array of Cookies associated with this domain
        cookies = request.getCookies();

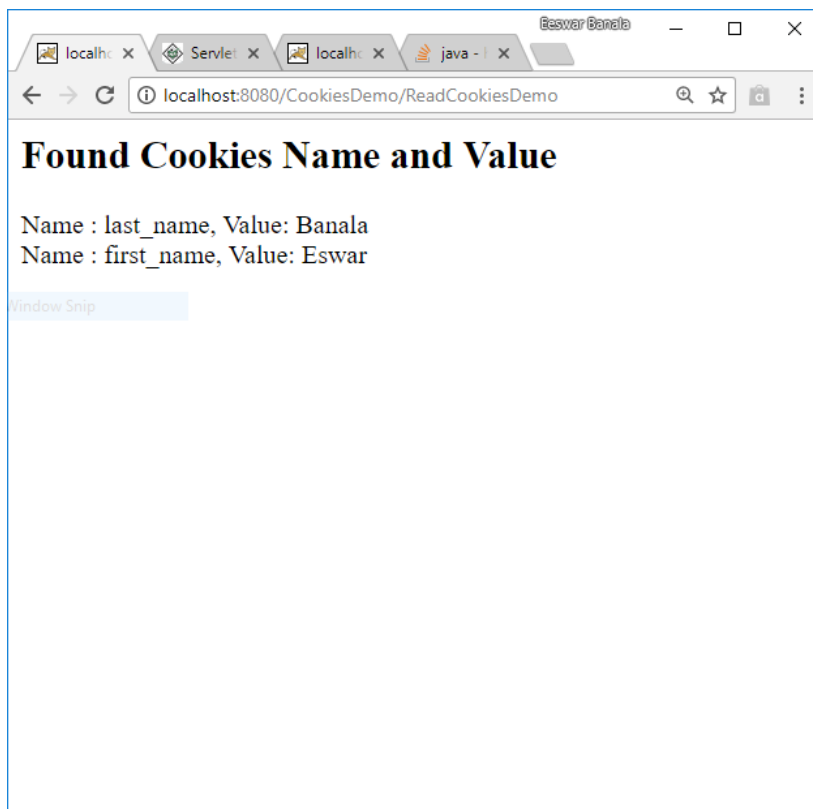
        // Set response content type
        response.setContentType("text/html");

        PrintWriter out = response.getWriter();

        if( cookies != null ) {
            out.println("<h2> Found Cookies Name and Value</h2>");

            for (int i = 0; i <cookies.length; i++) {
                //cookie = cookies[i];
                out.print("Name : " + cookies[i].getName( ) + ", ");
                out.print("Value: " + cookies[i].getValue( ) + " <br/>");
            }
            } else {
            out.println("<h2>No cookies founds</h2>");
        }

    }
}
```



Sessions - Servlets

The HttpSession Object

Servlet provides HttpSession Interface which provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.

The servlet container uses this interface to create a session between an HTTP client and an HTTP server. The session persists for a specified time period, across more than one connection or page request from the user.

You would get HttpSession object by calling the public method **getSession()** of HttpServletRequest, as below –

```
HttpSession s1 = req.getSession();
```

Sr.No.	Method & Description
1	<p>public Object getAttribute(String name)</p> <p>This method returns the object bound with the specified name in this session, or null if no object is bound under the name.</p>

2	public Enumeration getAttributeNames() This method returns an Enumeration of String objects containing the names of all the objects bound to this session.
3	public long getCreationTime() This method returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
4	public String getId() This method returns a string containing the unique identifier assigned to this session.
5	public long getLastAccessedTime() This method returns the last accessed time of the session, in the format of milliseconds since midnight January 1, 1970 GMT
6	public int getMaxInactiveInterval() This method returns the maximum time interval (seconds), that the servlet container will keep the session open between client accesses.
7	public void invalidate() This method invalidates this session and unbinds any objects bound to it.
8	public boolean isNew() This method returns true if the client does not yet know about the session or if the client chooses not to join the session.
9	public void removeAttribute(String name) This method removes the object bound with the specified name from this session.
10	public void setAttribute(String name, Object value) This method binds an object to this session, using the name specified.

11	<p>public void setMaxInactiveInterval(int interval)</p> <p>This method specifies the time, in seconds, between client requests before the servlet container will invalidate this session.</p>
----	--

Example**SessionTrack.java**

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// Extend HttpServlet class
public class SessionTrack extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        // Create a session object if it is already not created.
        HttpSession s1 = request.getSession(true);

        // Get session creation time.
        Date createTime = new Date(s1.getCreationTime());

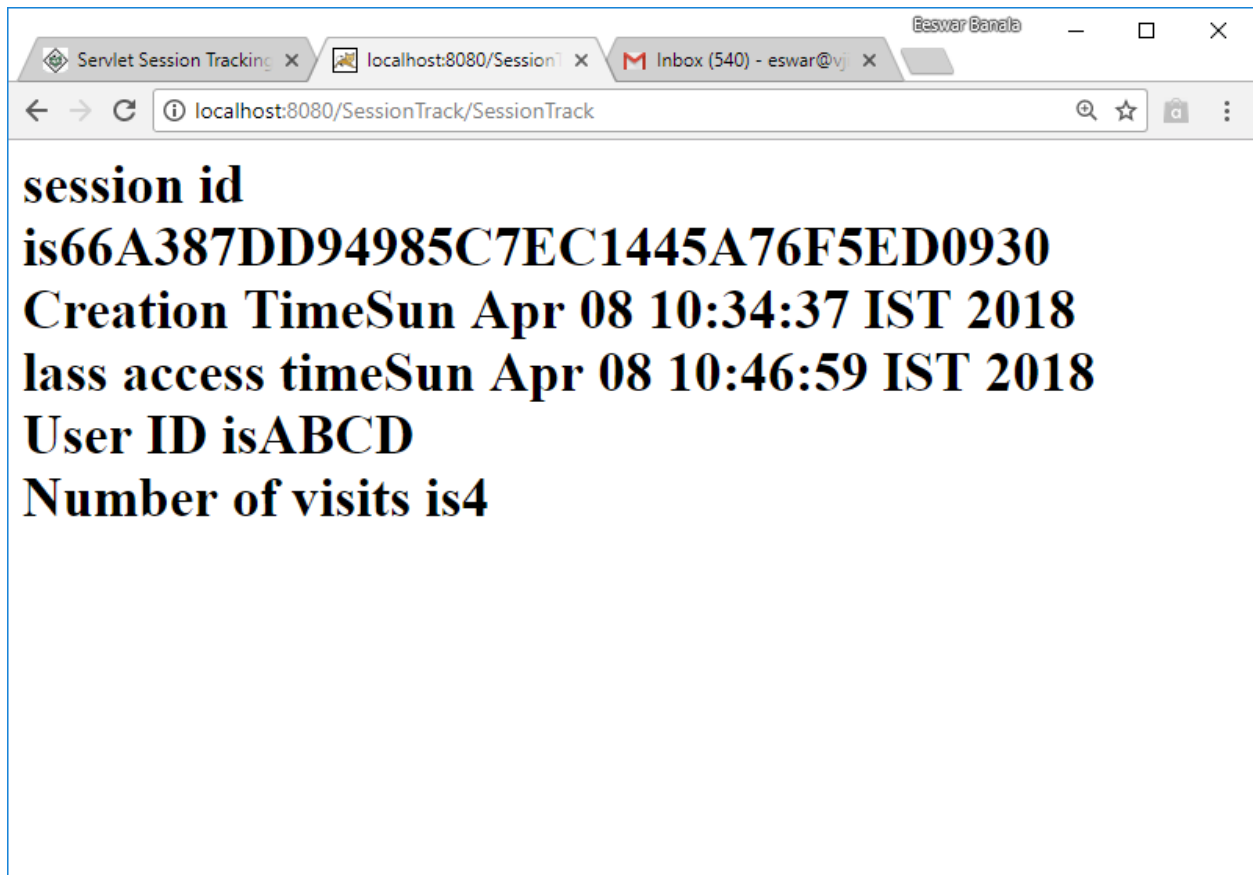
        // Get last access time of this web page.
        Date lastAccessTime = new Date(s1.getLastAccessedTime());
        Integer visitCount = new Integer(0);
        String visitCountKey = new String("visitCount");
        String userIDKey = new String("userID");
        String userID = new String("ABCD");

        // Check if this is new comer on your web page.
        if (s1.isNew())
        {
            s1.setAttribute(userIDKey, userID);
        }
        else
        {
            visitCount = (Integer)s1.getAttribute(visitCountKey);
            visitCount = visitCount + 1;
            userID = (String)s1.getAttribute(userIDKey);
        }
        s1.setAttribute(visitCountKey, visitCount);

        // Set response content type
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        out.println("<h1>session id is" + s1.getId() +
                    "<br>Creation Time"+ createTime +
                    "<br>last access time" + lastAccessTime +
```

```
");  
    "<br>User ID is" +userID +  
    "<br>Number of visits is" + visitCount+"</h1>"  
};  
}
```



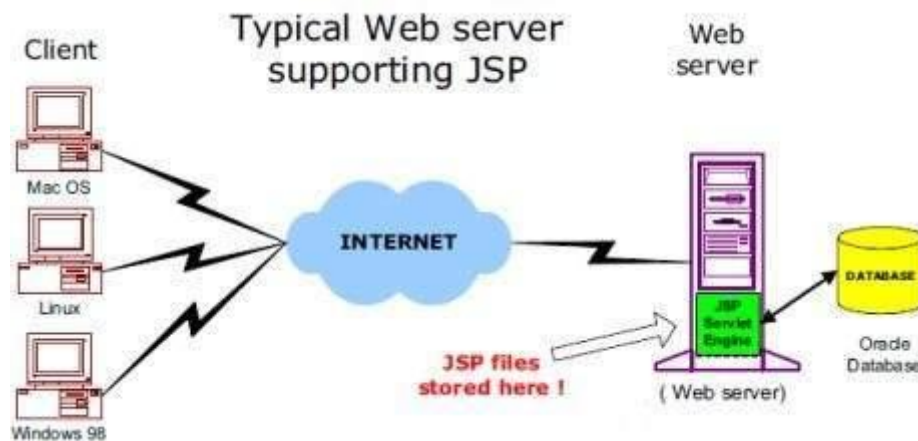
Java Server Pages (JSP)

JSP - Architecture

The web server needs a JSP engine, i.e, a container to process JSP pages. The JSP container is responsible for intercepting requests for JSP pages. This tutorial makes use of Apache which has built-in JSP container to support JSP pages development.

A JSP container works with the Web server to provide the runtime environment and other services a JSP needs. It knows how to understand the special elements that are part of JSPs.

Following diagram shows the position of JSP container and JSP files in a Web application.

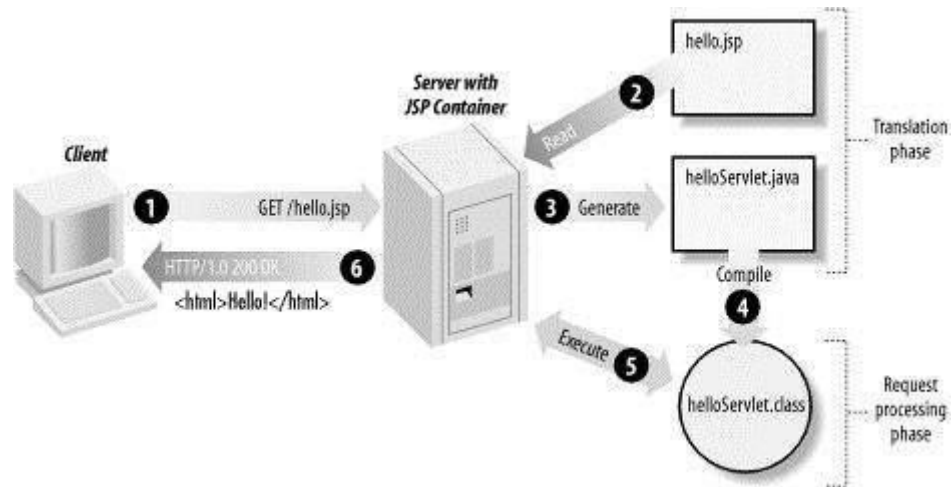


JSP Processing

The following steps explain how the web server creates the Webpage using JSP –

- As with a normal page, your browser sends an HTTP request to the web server.
- The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with **.jsp** instead of **.html**.
- The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to `println()` statements and all JSP elements are converted to Java code. This code implements the corresponding dynamic behavior of the page.
- The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
- A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format. The output is further passed on to the web server by the servlet engine inside an HTTP response.
- The web server forwards the HTTP response to your browser in terms of static HTML content.
- Finally, the web browser handles the dynamically-generated HTML page inside the HTTP response exactly as if it were a static page.

All the above mentioned steps can be seen in the following diagram –



Typically, the JSP engine checks to see whether a servlet for a JSP file already exists and whether the modification date on the JSP is older than the servlet. If the JSP is older than its generated servlet, the JSP container assumes that the JSP hasn't changed and that the generated servlet still matches the JSP's contents. This makes the process more efficient than with the other scripting languages (such as PHP) and therefore faster.

So in a way, a JSP page is really just another way to write a servlet without having to be a Java programming wiz. Except for the translation phase, a JSP page is handled exactly like a regular servlet.

JSP - Lifecycle

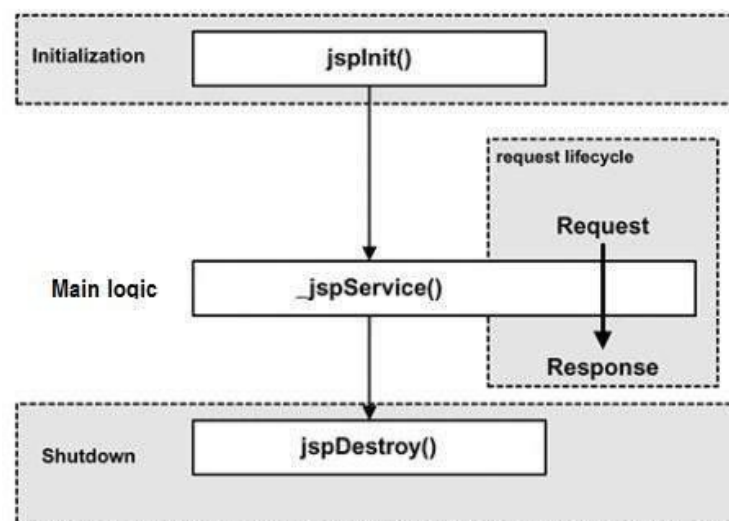
A JSP life cycle is defined as the process from its creation till the destruction. This is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet.

Paths Followed By JSP

The following are the paths followed by a JSP –

- Compilation
- Initialization
- Execution
- Cleanup

The four major phases of a JSP life cycle are very similar to the Servlet Life Cycle. The four phases have been described below –



JSP Compilation

When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page. If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.

The compilation process involves three steps –

- Parsing the JSP.
- Turning the JSP into a servlet.
- Compiling the servlet.

JSP Initialization

When a container loads a JSP it invokes the `jspInit()` method before servicing any requests. If you need to perform JSP-specific initialization, override the `jspInit()` method –

```
public void jspInit()
{
    // Initialization code...
}
```

JSP Execution

This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed.

Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the `_jspService()` method in the JSP.

```
void _jspService(HttpServletRequest request, HttpServletResponse response)
{
    // Service handling code...
}
```

JSP Cleanup

The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.

The `jspDestroy()` method is the JSP equivalent of the `destroy` method for servlets. Override `jspDestroy` when you need to perform any cleanup, such as releasing database connections or closing open files.

The `jspDestroy()` method has the following form –

```
public void jspDestroy()
{
    // Your cleanup code goes here.
}
```

Elements of JSP

The elements of JSP have been described below –

1. The Scriptlet

A scriptlet can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.

Syntax of Scriptlet –

```
<% code fragment %>
```

Note: Any text, HTML tags, or JSP elements you write must be outside the scriptlet.

Example

```
<html>
  <head><title>Hello World</title></head>
  <body>
    Hello World! <br>
    <%
      out.println("Your IP address is " + request.getRemoteAddr());
    %>
  </body>
</html>
```

2. JSP Declarations

A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file.

syntax for JSP Declarations –

```
<%! declaration; [ declaration; ]+ ... %>
```

Example

```
<%! int i = 0; %>
<%! int a, b, c; %>
<%! Circle a = new Circle(2.0); %>
```

3. JSP Expression

- A JSP expression element contains a scripting language expression that is evaluated, converted to a String, and inserted where the expression appears in the JSP file.
- Because the value of an expression is converted to a String, you can use an expression within a line of text, whether or not it is tagged with HTML, in a JSP file.
- The expression element can contain any expression that is valid according to the Java Language Specification but you cannot use a semicolon to end an expression.

Syntax of JSP Expression –

```
<%= expression %>
```

Example

```
<html>
  <head><title>A Comment Test</title></head>
  <body>
    <p>Today's date: <%= (new java.util.Date()).toLocaleString()%></p>
  </body>
</html>
```

4. JSP Comments

JSP comment marks text or statements that the JSP container should ignore. A JSP comment is useful when you want to hide or "comment out", a part of your JSP page.

Syntax of the JSP comments –

```
<%-- This is JSP comment --%>
```

5. JSP Directives

A JSP directive affects the overall structure of the servlet class.

Syntax of the JSP Directives –

```
<%@ directive attribute="value" %>
```

There are three types of directive tag –

S.No.	Directive & Description
1	<%@ page ... %> Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.
2	<%@ include ... %> Includes a file during the translation phase.
3	<%@ taglib ... %> Declares a tag library, containing custom actions, used in the page

6. JSP Actions

JSP actions use constructs in XML syntax to control the behavior of the servlet engine. You can dynamically insert a file, reuse JavaBeans components, forward the user to another page, or generate HTML for the Java plugin.

Syntax for the Action element, as it conforms to the XML standard –

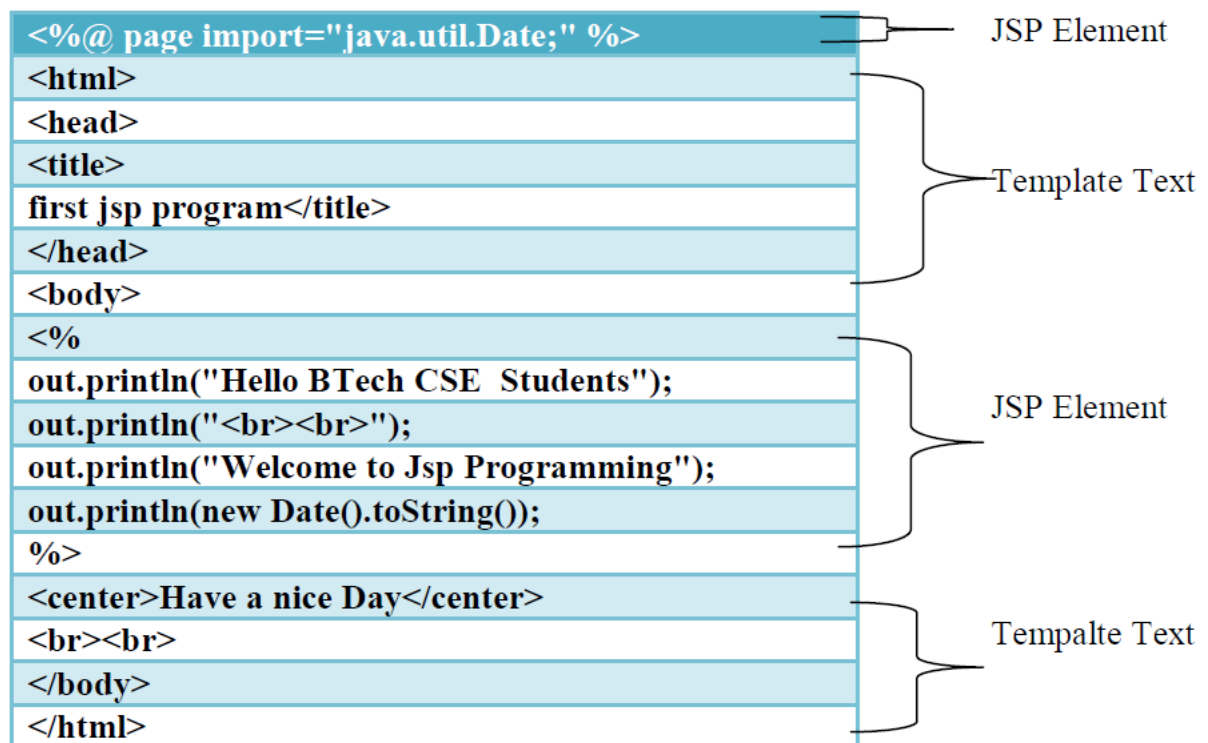
```
<jsp:action_name attribute="value" />
```

Action elements are basically predefined functions. Following table lists out the available JSP Actions –

S.No.	Syntax & Purpose
1	jsp:include Includes a file at the time the page is requested.
2	jsp:useBean Finds or instantiates a JavaBean.
3	jsp:setProperty Sets the property of a JavaBean.
4	jsp:getProperty Inserts the property of a JavaBean into the output.
5	jsp:forward Forwards the requester to a new page.

6	jsp:plugin Generates browser-specific code that makes an OBJECT or EMBED tag for the Java plugin.
7	jsp:element Defines XML elements dynamically.
8	jsp:attribute Defines dynamically-defined XML element's attribute.
9	jsp:body Defines dynamically-defined XML element's body.
10	jsp:text Used to write template text in JSP pages and documents.

Anatomy of JSP



JSP Implicit Objects

JSP supports nine automatically defined variables, which are also called implicit objects. These variables are –

S.No.	Object & Description
1	request This is the HttpServletRequest object associated with the request.
2	response This is the HttpServletResponse object associated with the response to the client.
3	out This is the PrintWriter object used to send output to the client.
4	session This is the HttpSession object associated with the request.
5	application This is the ServletContext object associated with the application context.
6	config This is the ServletConfig object associated with the page.
7	pageContext This encapsulates use of server-specific features like higher performance JspWriters .
8	page This is simply a synonym for this , and is used to call the methods defined by the translated servlet class.
9	Exception The Exception object allows the exception data to be accessed by designated JSP.

Using Beans in JSP Pages

A JavaBean is a specially constructed Java class written in the Java and coded according to the JavaBeans API specifications.

Following are the unique characteristics that distinguish a JavaBean from other Java classes –

- It provides a default, no-argument constructor.
- It should be serializable and that which can implement the **Serializable** interface.
- It may have a number of properties which can be read or written.
- It may have a number of "**getter**" and "**setter**" methods for the properties.

JavaBeans Properties

A JavaBean property is a named attribute that can be accessed by the user of the object. The attribute can be of any Java data type, including the classes that you define.

A JavaBean property may be **read**, **write**, **read only**, or **write only**. JavaBean properties are accessed through two methods in the JavaBean's implementation class –

S.No.	Method & Description
1	getPropertyName() For example, if property name is <i>firstName</i> , your method name would be getFirstName() to read that property. This method is called accessor.
2	setPropertyName() For example, if property name is <i>firstName</i> , your method name would be setFirstName() to write that property. This method is called mutator.

A read-only attribute will have only a **getPropertyName()** method, and a write-only attribute will have only a **setPropertyName()** method.

Example: (StudentsBean.java)

```
package com;
public class StudentsBean implements java.io.Serializable
{
    private String firstName = null;
    private String lastName = null;

    public StudentsBean()
    {
    }
    //Get Methods
    public String getFirstName()
    {
        return firstName;
    }
    public String getLastName()
    {
        return lastName;
    }

    //Set Methods
    public void setFirstName(String firstName)
    {
        this.firstName = firstName;
    }
}
```

```

    public void setLastName(String lastName)
    {
        this.lastName = lastName;
    }
}

```

Accessing JavaBeans

The **useBean** action declares a JavaBean for use in a JSP. Once declared, the bean becomes a scripting variable that can be accessed by both scripting elements and other custom tags used in the JSP. The full **syntax for the useBean tag is as follows –**

```
<jsp:useBean id = "bean's name" />
```

The value of the **id** attribute may be any value as long as it is a unique name among other **useBean declarations** in the same JSP.

Example shows how to use the useBean action –

```

<html>
  <head>
    <title>useBean Example</title>
  </head>

  <body>
    <jsp:useBean id = "date" class = "java.util.Date" />
    <p>The date/time is <%= date %>
  </body>
</html>

```

Accessing JavaBeans Properties

Along with **<jsp:useBean...>** action, you can use the **<jsp:getProperty/>** action to access the get methods and the **<jsp:setProperty/>** action to access the set methods.

Syntax –

```

<jsp:useBean id = "id" class = "bean's class" scope = "bean's scope">
  <jsp:setProperty name = "bean's id" property = "property name"
    value = "value"/>
  <jsp:getProperty name = "bean's id" property = "property name"/>
  .....
</jsp:useBean>

```

The name attribute references the id of a JavaBean previously introduced to the JSP by the useBean action. The property attribute is the name of the **get** or the **set** methods that should be invoked.

Example (StudentsBean.jsp) shows how to access the data using the above syntax –

```

<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<%@ page import="com.StudentsBean"%>
<html>
  <head>
    <title>get and set properties Example</title>
  </head>

  <body>
    <jsp:useBean id = "students" class = "com.StudentsBean">
      <jsp:setProperty name = "students" property = "firstName" value =
"Eeswar"/>
      <jsp:setProperty name = "students" property = "lastName" value =
"Banala"/>
    </jsp:useBean>
  </body>
</html>

```



```
</jsp:useBean>

<p>Student First Name:
  <jsp:getProperty name = "students" property = "firstName"/>
</p>

<p>Student Last Name:
  <jsp:getProperty name = "students" property = "lastName"/>
</p>

</body>
</html>
```

Session Tracking in JSP

The JSP engine exposes the HttpSession object to the JSP author through the implicit **session** object. Since **session** object is already provided to the JSP programmer, the programmer can immediately begin storing and retrieving data from the object without any initialization or **getSession()**.

Here is a summary of important methods available through the session object –

S.No.	Method & Description
1	public void setAttribute(String name, Object value) This method binds an object to this session, using the name specified.
2	public Object getAttribute(String name) This method returns the object bound with the specified name in this session, or null if no object is bound under the name.
3	public Enumeration getAttributeNames() This method returns an Enumeration of String objects containing the names of all the objects bound to this session.
4	public long getCreationTime() This method returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
5	public String getId() This method returns a string containing the unique identifier assigned to this session.
6	public long getLastAccessedTime() This method returns the last time the client sent a request associated with the this session, as the number of milliseconds since midnight January 1, 1970 GMT.
7	public int getMaxInactiveInterval() This method returns the maximum time interval, in seconds, that the servlet container will keep this session open between client accesses.
8	public void invalidate() This method invalidates this session and unbinds any objects bound to it.
9	public boolean isNew() This method returns true if the client does not yet know about the session or if the client chooses not to join the session.
10	public void removeAttribute(String name) This method removes the object bound with the specified name from this session.

Session Tracking Example

W11.a.jsp (Setting the Session variables)

```
<%@page language="java" import="java.util.*" import="java.text.*"
errorPage=""%>
<form method="get" action="w11b.jsp">
<%
SimpleDateFormat sd = new SimpleDateFormat("dd.MM.yyyy 'at' HH:mm:ss z");
```

```
Date d=new Date();
sd.setTimeZone(TimeZone.getTimeZone("IST"));
%>
<p align="right"> Time:<%=sd.format(d)%></p>
<%
String un=request.getParameter("uname");
session.setAttribute("user",un);
session.setAttribute("time",d.getTime());
%>
Hello <%=un%>
<br><br>
<input type="submit" value="Logout">
</form>
```

W11b.jsp (Accessing the Session variables)

```
<%@page language="java" import="java.util.*" errorPage=""%>
<%
Date d2=new Date();
String un=(String)session.getAttribute("user");
Long t1=(Long)session.getAttribute("time");
Long t2=d2.getTime();
%>
Thank you <%=un%>
<br><br>
Session duration: <%= (t2-t1)/(60*60)%> seconds
<% session.invalidate();%>
```

W11.html

```
<html>
<head> <title> SESSION LOGIN </title> </head>
<body>
<center>
<form action="w11a.jsp" method="get">
Enter Name: <input type="text" name="uname"> <br><br>
<input type="submit" value="LOGIN" name="register">
</form>
</center>
</body>
</html>
```

MVC Architecture in JSP

MVC stands for Model View and Controller. It is a **design pattern** that separates the business logic, presentation logic and data.

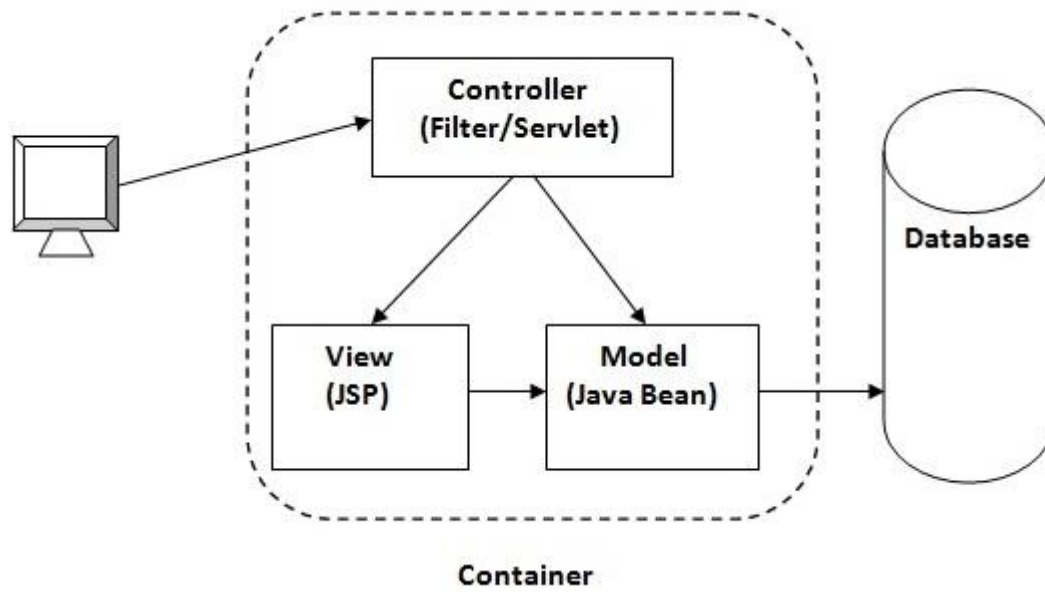
Controller acts as an interface between View and Model. Controller intercepts all the incoming requests.

Model represents the state of the application i.e. data. It can also have business logic.

View represents the presentation i.e. UI(User Interface).

Advantage of MVC (Model 2) Architecture

1. Navigation Control is centralized
2. Easy to maintain the large application





CO - PO Attainment



VIDYA JYOTHI INSTITUTE OF TECHNOLOGY

DEPARTMENT OF INFORMATION TECHNOLOGY
Course: WEB TECHNOLOGIES

Batch: 2017-21

CL ASS III year II SEM

S.No	Reg.No	MID-1 Threshold 60%									MID-2 Threshold 60%						Threshold 60% (45M) End Exam (75M)	
		Assignment1 (5M)	Theory (20M)	PART-B						Assignment2 (5M)	Theory (20M)	PART-B						
				M1- Q1 (2M)	M1-Q2 (2M)	M1- Q3 (2M)	M1- Q4 (5M)	M1-Q5 (5M)	M1-Q6 (4M)			M2- Q1 (2M)	M2-Q2 (2M)	M2- Q3 (2M)	M2-Q4 (4M)	M2-Q5 (5M)		M2-Q6 (5M)
1	17911A1255	5	12	1	2	1	3	2	3	5	11	0	2	1	3	2	3	17
2	17911A1201	5	19	1	2	2	5	5	4	5	18	1	1	2	4	5	5	54
3	17911A1202	5	19	2	2	1	5	5	4	5	18	2	2	0	4	5	5	49
4	17911A1203	5	19	2	2	2	4	5	4	5	18	2	2	2	4	5	3	52
5	17911A1205	5	18	2	2	2	5	4	3	5	17	2	2	2	3	3	5	36
6	17911A1206	5	19	2	2	2	5	5	3	5	18	2	2	2	2	5	5	29
7	17911A1208	5	18	2	2	1	5	4	4	5	17	2	2	1	3	4	5	35
8	17911A1209	5	15	2	2	1	5	2	3	5	14	2	2	1	3	1	5	29
9	17911A1211	5	15	1	1	1	5	4	3	5	14	1	1	1	3	4	5	47
10	17911A1212	5	15	1	1	2	3	4	4	5	14	1	1	1	4	4	4	45
11	17911A1213	5	15	1	1	1	4	4	4	5	14	1	0	1	4	4	4	46
12	17911A1214	5	19	2	2	2	5	4	4	5	18	1	2	2	4	4	5	42
13	17911A1215	5	20	2	2	2	5	5	4	5	19	2	1	2	4	5	5	33
14	17911A1216	5	15	2	2	2	4	3	2	5	14	2	2	1	2	3	4	39
15	17911A1218	5	16	2	1	1	5	3	4	5	15	2	1	1	4	3	4	54
16	17911A1219	5	14	2	2	1	2	3	4	5	13	2	2	1	4	2	2	55
17	17911A1220	5	16	2	1	2	5	2	4	5	15	2	1	2	3	2	2	48
18	17911A1221	5	17	2	1	1	5	4	4	5	16	2	1	1	4	3	5	60
19	17911A1223	5	18	2	2	2	5	4	3	5	17	2	2	2	3	4	4	47
20	17911A1225	5	20	2	2	2	5	5	4	5	19	2	2	1	4	5	5	48
21	17911A1226	5	15	2	2	1	5	2	3	5	14	2	1	1	3	2	5	54
22	17911A1227	5	17	1	2	2	5	3	4	5	16	0	2	2	4	3	5	47
23	17911A1228	5	20	2	2	2	5	5	4	5	19	2	1	2	4	5	5	34
24	17911A1229	5	15	2	2	0	5	4	2	5	14	2	2	0	2	4	4	54
25	17911A1230	5	13	1	1	1	3	4	3	5	12	1	1	1	3	3	3	15
26	17911A1231	5	16	1	1	2	5	4	3	5	15	1	1	2	2	4	5	16
27	17911A1232	5	17	1	2	2	4	4	4	5	16	1	2	2	3	4	4	47
28	17911A1233	5	14	1	1	2	4	3	3	5	13	1	1	2	3	2	4	46
29	17911A1234	5	18	1	2	1	5	5	4	5	17	1	2	1	4	5	4	15
30	17911A1235	5	17	2	2	1	5	3	4	5	16	2	2	0	4	3	5	42

ASSESSMENT OF COs FOR THE COURSE							
COs	Method	value	CO Attain	Assignments	CO Attainment (Internal - Theory)	CO Attainment (End Exam)	Overall CO Attainment
CO1	M1	3.0	3.0	3.0	2.7	2.00	2.21
	Q1	3.0					
CO2	M1	3.0	3.0				
	Q2	3.0					
	M1	3.0					
CO3	Q5	3.0	3.0				
	M1	3.0					
	Q3	3.0					
	M1	3.0					
CO4	Q7	3.0	3.0				
	M2	3.0					
	Q1	3.0					
	M2	3.0					
CO5	Q4	3.0	1.5				
	M2	3.0					
	Q2	3.0					
CO5	M2	3.0	1.5				
	Q3	3.0					
	M2	0.0					



Course End Survey Form

1. Are you able to create static and dynamic web pages using HTML and java script?38

RESPONSES

Slight 3

Moderate 3

Substantial 32

2. Are you able to analyze the XML and how to parse XML data with java?38

RESPONSES

Slight 3

Moderate 5

Substantial 30

3. Are you able to develop web applications using server side scripting language-PHP?

Slight 2

Moderate 2

Substantial 34

4. Are you able to implement the web applications using JDBC and java servlets?

Slight 1

Moderate 4

Substantial 33

5. Are you able to apply web applications with Java Server Pages?

Slight 3

Moderate 5

Substantial 30