Web Technologies

Unit 1: Contents

UNIT-I: HTML : Basic Syntax, Standard HTML Document Structure, Basic Text Markup, Images, Hypertext Links, Lists, Tables, Frames Forms.

CSS: Cascading style sheets : Levels of Style Sheets, Style Specification Formats, Selector Forms, Property value forms, Font Properties, List Properties, color, Alignment of Text

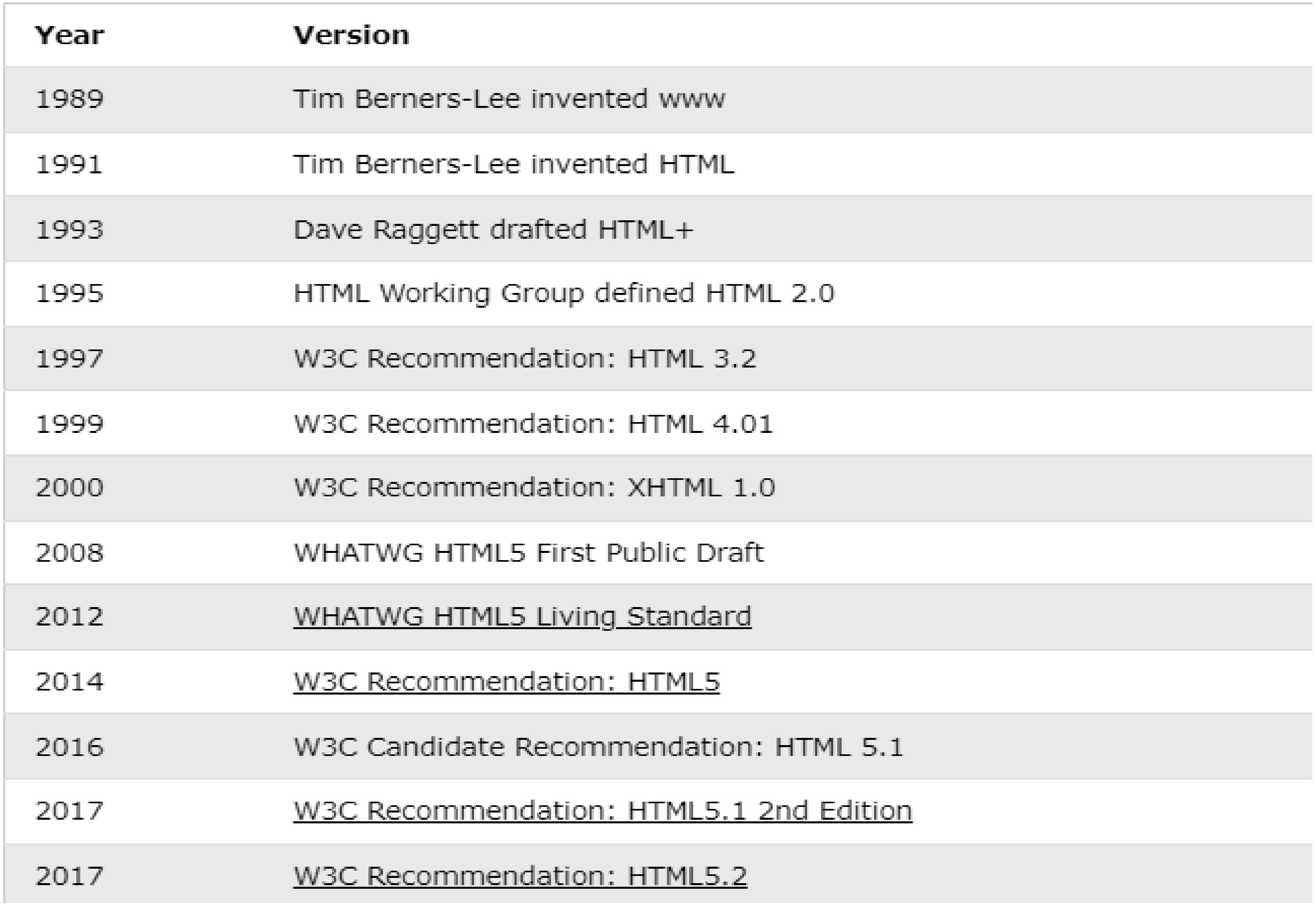
Introduction to HTML

- HTML is the standard markup language for Web pages.
- With HTML you can create your own Website.
- HTML is easy to learn - You will enjoy it!
- HTML consists of a series of elements
- HTML elements tell the browser how to display the content
- HTML elements label pieces of content such as "this is a heading", "this is a paragraph", "this is a link", etc.

HTML tags

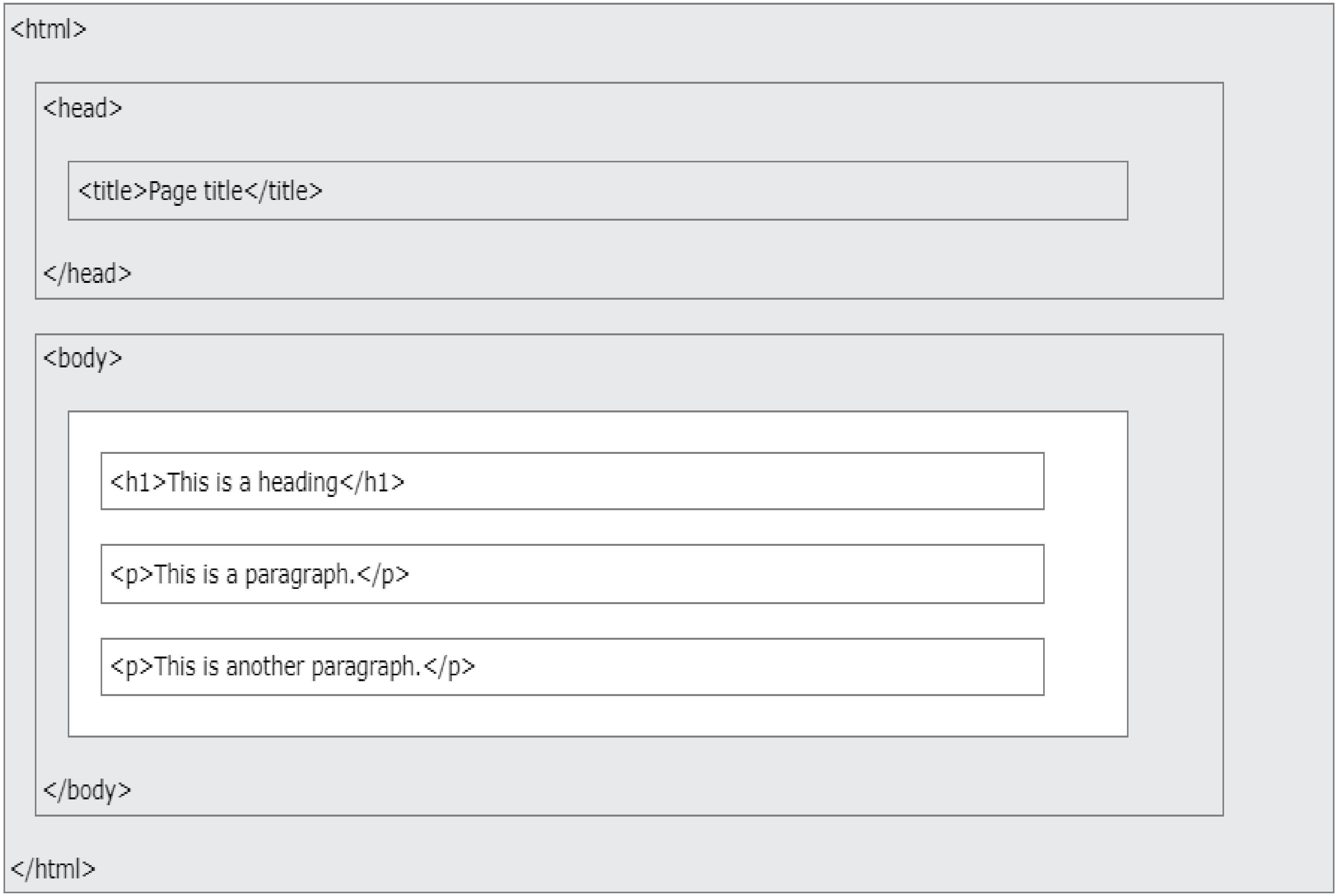
- HTML markup tags are usually called HTML tags
- HTML tags are keywords (tag names) surrounded by angle brackets like <html>
- HTML tags normally come in pairs like <b> and </b>
- The first tag in a pair is the start tag, the second tag is the end tag
- The end tag is written like the start tag, with a forward slash before the tag name
- Start and end tags are also called opening tags and closing tags

History of HTML



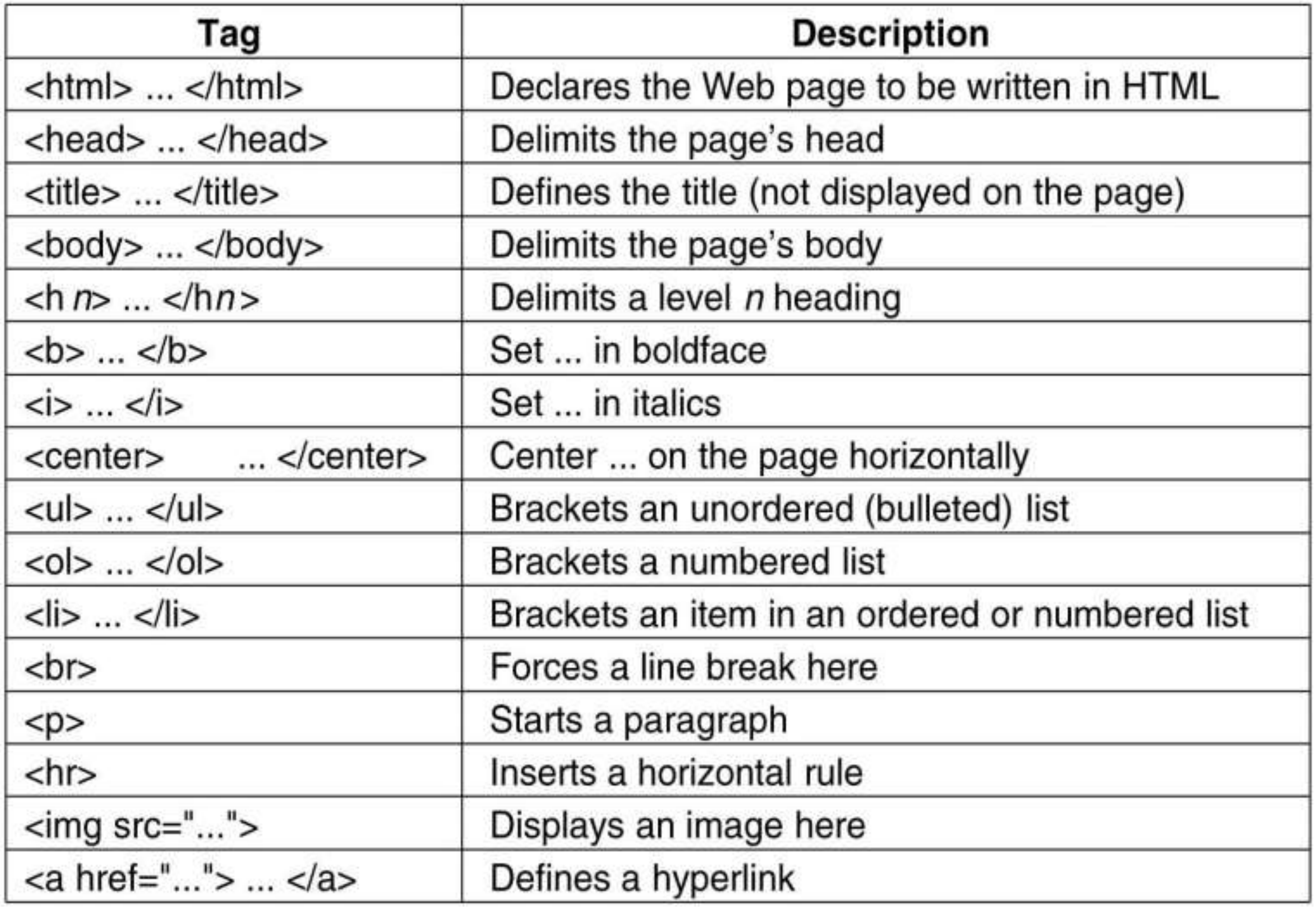
Year	Version
1989	Tim Berners-Lee invented www
1991	Tim Berners-Lee invented HTML
1993	Dave Raggett drafted HTML+
1995	HTML Working Group defined HTML 2.0
1997	W3C Recommendation: HTML 3.2
1999	W3C Recommendation: HTML 4.01
2000	W3C Recommendation: XHTML 1.0
2008	WHATWG HTML5 First Public Draft
2012	WHATWG HTML5 Living Standard
2014	W3C Recommendation: HTML5
2016	W3C Candidate Recommendation: HTML 5.1
2017	W3C Recommendation: HTML5.1 2nd Edition
2017	W3C Recommendation: HTML5.2

Structure of HTML Page



```
<html>
  <head>
    <title>Page title</title>
  </head>
  <body>
    <h1>This is a heading</h1>
    <p>This is a paragraph.</p>
    <p>This is another paragraph.</p>
  </body>
</html>
```

Some Important HTML Tags



Tag	Description
<html> ... </html>	Declares the Web page to be written in HTML
<head> ... </head>	Delimits the page's head
<title> ... </title>	Defines the title (not displayed on the page)
<body> ... </body>	Delimits the page's body
<h *n*> ... </h*n*>	Delimits a level *n* heading
<b> ... </b>	Set ... in boldface
<i> ... </i>	Set ... in italics
<center> ... </center>	Center ... on the page horizontally
<ul> ... </ul>	Brackets an unordered (bulleted) list
<ol> ... </ol>	Brackets a numbered list
<li> ... </li>	Brackets an item in an ordered or numbered list
<br>	Forces a line break here
<p>	Starts a paragraph
<hr>	Inserts a horizontal rule
<img src="...">	Displays an image here
<a href="..."> ... </a>	Defines a hyperlink

Listing of the HTML tags found during the survey, distinguishing between the correct and incorrect ones.

Correct HTML Tags	**Correct HTML Tags**	**Correct HTML Tags**	**Incorrect HTML Tags**	**Incorrect HTML Tags**	**Incorrect HTML Tags**
a	Em	option	abcsubmit	gave	ps
address	Font	p	adress	grin	rfpicr
align	Form	param	ahref	hrnoshade	shift
applet	Frame	pre	aired	htm	silence
area	Frameset	right	aircheck	http	stations
b	h[1..6]	script	and	il	svd
base	head	select	are	imgsrc	tdalign
basefont	hr	small	bgsound	inputtype	tdcolspan
blink	html	strong	bodybgcolor	is	tdnowrap
blockquote	i	sub	border	it	tdwidth
e	img	table	brt	jberg	textareawrap
body	input	td	by	jpd	thank
br	left	textarea	ceneter	krone	tsfinfo
caption	li	th	centre	m	valign
center	kbd	title	clear	marquee	vk
cite	link	tr	color	means	w
code	map	tt	colw	moore	wbr
dd	menu	u	embed	name	were
dir	meta	ul	emp	nobr	width
div	noframes	width	front	of	wireless
dl	ol		fontsize	palign	www
dt					

HTML Example

```
<!DOCTYPE html>
  <html>
  <body>
  <h1>My First Heading</h1>
  <p>My first paragraph.</p>
  </body>
  </html>
```

Explanation

- ➢ The DOCTYPE declaration defines the document type
- ➢ The text between <html> and </html> describes the web page
- ➢ The text between <body> and </body> is the visible page content
- ➢ The text between <h1> and </h1> is displayed as a heading
- ➢ The text between <p> and </p> is displayed as a paragraph

What is HTML?

- HTML, otherwise known as HyperText Markup Language, is the language used to create Web pages
- Using HTML, you can create a Web page with text, graphics, sound, and video.
- Hyper means we can navigate from one web page to another page where they need not be linear pages.

Tags

- HTML is written in the form of tags
- A tag is a keyword enclosed by pair of angle brackets (Example: < >)
- Where some text is placed between tags.
- HTML elements have two basic properties

➢ Attributes

➢ contents

More Tags...

- The opening and closing tags use the same command except the closing tag contains and additional forward slash /
- For example, the expression `<b > Warning </b>` would cause the word ‘Warning’ to appear in bold face on a Web page.
- There are some tags which has opening tag but not closing tag, also known as Empty HTML Elements.

  Eg - <br>,<hr> i.e Break and Horizontal rule

Nested Tags

- Whenever you have HTML tags within other HTML tags, you must close the nearest tag first
- Example:

    <h1> <il> The Nation </li> </h1>

Structure of a Web Page

- All Web pages share a common structure
- All Web pages should contain a pair of <HTML>, <HEAD>, <TITLE>, and <BODY> tags.

```
<html>
<head>
<title> Example </title>
</head>
<body>
    This is where you would include
    the text and images on your Web
    page.
</body>
</html>
```

Comment Statements

- Comment statements are notes in the HTML code that explain the important features of the code
- The comments do not appear on the Web page itself but are a useful reference to the author of the page and other programmers
- To create a comment statement use the <!-- Write ur comment here --> tags

HTML Elements

An HTML element is defined by a start tag, some content, and an end tag.

HTML Elements

The HTML **element** is everything from the start tag to the end tag:

<tagname>Content goes here...</tagname>

Examples of some HTML elements:

<h1>My First Heading</h1>

<p>My first paragraph.</p>

HTML Attributes

- All HTML elements can have **attributes**
- Attributes provide **additional information** about elements
- Attributes are always specified in **the start tag**
- Attributes usually come in name/value pairs like: **name="value"**

For example:

The href Attribute

The <a> tag defines a hyperlink.

The href attribute specifies the URL of the page the link goes to:

<a href="https://www.w3schools.com">Visit W3Schools</a>

The <TITLE> Tag

- Choose the title of your Web page carefully; The title of a Web page determines its ranking in certain search engines
- The title will also appear on Favorite lists, History lists, and Bookmark lists to identify your page

Headings

- Web pages are typically organized into sections with headings; To create a heading we use the expression
- <Hn>….</Hn> where n is a number between 1 and 6
- In this case, the 1 corresponds to the largest size heading while the 6 corresponds to the smallest size

Text Formatting

- Manipulating text in HTML can be tricky; Oftentimes, what you see is NOT what you get
- For instance, special HTML tags are needed to create paragraphs, move to the next line, and create headings

Text Formatting Tags

<b> **Bold Face** </b>

<strong> This is also same like bold but shows importance </strong>

<i> *Italics* </i>

<u> Underline </u>

<p> New Paragraph </p>

<br> Next Line

Example on Text Formatting Tags

```
<!DOCTYPE  html">
<head>
<title>Example on Text Formatting Tags</title>
</head>
<body>
<p><b> Bold Face </b> <br>
 <strong> this is also same like bold but shows importance </strong> </p>
<p><i>The text will appear as Italics </i> </p>
<p><u>This text will appear in Underline </u> </p>
<p> New Paragraph will be started from this tag,

we can write the paragraph as

many number of lines and sentences </p>
<br>
</body>
</html>
```

Other Formatting Tags

<pre>- Preformatted Text

<mark> - Marked text

<small> - Smaller text

<del> - Deleted text

<ins> - Inserted text

<sub> - Subscript text

<sup> - Superscript text

pre: Defines preformatted text

The <pre> tag defines preformatted text.

- Text in a <pre> element is displayed in a fixed-width font, and the text preserves both spaces and line breaks. The text will be displayed exactly as written in the HTML source code.

<mark>- Marked text

The HTML <mark> element defines text that should be marked or highlighted:

Example

```
<p> Please come with your <mark>
  Observation and Record </mark> for
  your WT Lab
</p>
```

<small> Tag

- The HTML <small> element defines smaller text:

Example:

<h6> this will display heading is small font </h6>

<small> This will display text in very small </small>

Note: h6 is smaller than compared with small attribute.

<del> - Deleted Text tag

The HTML **<del>** element defines text that has been deleted from a document. Browsers will usually strike a line through deleted text:

Example:

<p>My Interested Job is <del> Software</del> Government. </p>

<ins> - Inserted text

The HTML <ins> element defines a text that has been inserted into a document. Browsers will usually underline inserted text:

Example:

```
<p>
My Interested Job is <del> Software</del> <ins>Government </ins>.
</p>
```

<sub> - Subscript text

The HTML <sub> element defines subscript text. Subscript text appears half a character below the normal line, and is sometimes rendered in a smaller font. Subscript text can be used for chemical formulas, like H_2O:

Example:

```
<p>This is an example on <sub>subscripted</sub> text.</p>
```

HTML <sup> Element

The HTML <sup> element defines superscript text. Superscript text appears half a character above the normal line, and is sometimes rendered in a smaller font. Superscript text can be used for footnotes, like WW W[1]:

<p>This is an example on <sup>superscripted</sup> text.</p>

Example on Text Formatting Tags

```
<!DOCTYPE html>
<head>
<title>Text Formatting tags  </title>
</head>
<body>
<pre>Please come with your
         Observation and Record
                         for your WT Lab </pre>

<p> Please come with your <mark> Observation and Record </mark> for your WT Lab
</p>
<h6> this will display heading is small font </h6>
<small> This will display text in very small </small>
<p>My Interested Job is  <del> Software</del>  Government. </p>
<p>
My Interested Job is  <del> Software</del>  <ins>Government </ins>. </p>
<p>This is  an example on <sub>subscripted</sub> text.</p>
<p>This is  an example on <sup>superscripted</sup> text.</p>
</body>
</html>
```

Tables

- Tables can be used to display rows and columns of data, create multi-column text, captions for images, and sidebars
- The <table> tag is used to create a table;

the <tr> tag defines the beginning of a row while the <td> tag defines the beginning of a cell

<th> is used to define the table heading for a cell

<thead> is used to define the heading for the entire table

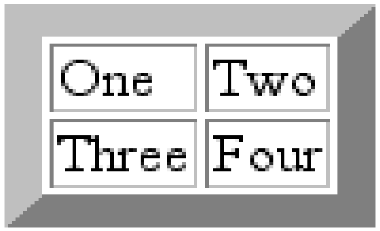
Adding a Border

- The BORDER=n attribute allows you to add a border n pixels thick around the table
- To make a solid border color, use the BORDERCOLOR="color" attribute

Creating Simple Table

```
<TABLE BORDER=10>
    <TR>
          <TD>One</TD>
          <TD>Two</TD>
    </TR>
    <TR>
          <TD>Three</TD>
          <TD>Four</TD>
    </TR>
</TABLE>
```

- Here's how it would look on the Web:



One	Two
Three	Four

Adjusting the Width

- When a Web browser displays a table, it often adds extra space. To eliminate this space use the WIDTH =n attribute in the <TABLE> and <TD> tags
- Keep in mind - a cell cannot be smaller than its contents, and if you make a table wider than the browser window, users will not be able to see parts of it.

Centering a Table

- There are two ways to center a table
  - Type <TABLE ALIGN=CENTER>
  - Enclose the <TABLE> tags in opening and closing <CENTER> tags

Wrapping Text around a Table

- It is possible to wrap text around a table. This technique is often used to keep images and captions together within an article.
- To wrap text around a table, type <TABLE ALIGN = LEFT> to align the table to the left while the text flows to the right.
- Create the table using the <TR>, <TD>, and </TABLE> tags as you normally would

Adding Space around a Table

- To add space around a table, use the HSPACE=n and VSPACE=n attributes in the <TABLE> tag
- Example:

```
<TABLE HSPACE=20 VSPACE=20>
```

Nesting Tables

- Create the inner table
- Create the outer table and determine which cell of the outer table will hold the inner table
- Test both tables separately to make sure they work
- Copy the inner table into the cell of the outer table
- Don’t nest too many tables. If you find yourself doing that, find an easier way to lay out your Web page

Changing a Cell’s Color

- To change a cell’s color, add the BGCOLOR=“color” attribute to the <TD> tag
- Example:

  <TD BGCOLOR=“blue”>

Colspan and Rowspan Attributes

You will use **colspan** attribute if you want to merge two or more columns into a single column. Similar way you will use **rowspan** if you want to merge two or more rows.

Changing the Font

- The expression <font face = “fontname”> … </font>

  can be used to change the font of the enclosed text
- To change the size of text use the expression <font size=n> …. </font> where n is a number between 1 and 7

Changing the Font

- To change the color, use <font color="red">.... </font>; The color can also be defined using hexadecimal representation (Example: #ffffff)
- These attributes can be combined to change the font, size, and color of the text all at once; For example, <font size=4 face="Courier" color="red"> .... </font>

Aligning Text

- The **align** attribute can be inserted in the <p> and <hn> tags to right justify, center, or left justify the text
- For example, <h1 align=center> The New York Times </h1> would create a centered heading of the largest size

Page Formatting

- To define the background color, use the BGCOLOR attribute in the <BODY> tag
- To define the text color, use the TEXT attribute in the <BODY> tag
- To define the size of the text, type <BASEFONT SIZE=n>

Example

```
<html>
<head>
<title> First Example Program </title>
</head>
<body>
<h1 style="background-color:tomato;">
    This is where you would include the text and images on your Web page.
</h1>
</body>
</html>
```

HTML Lists

HTML offers web authors three ways for specifying lists of information. All lists must contain one or more list elements. Lists may contain −

<ul> − An unordered list. This will list items using plain bullets.

<ol> − An ordered list. This will use different schemes of numbers to list your items.

<dl> − A definition list. This arranges your items in the same way as they are arranged in a dictionary.

HTML Un-Ordered List

An unordered list is a collection of related items that have no special order or sequence. This list is created by using HTML **<ul>** tag. Each item in the list is marked with a bullet.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Unordered List</title>
</head>
<body>
<ul>
    <li>HTML</li>
    <li>CSS</li>
    <li>Java Script</li>
    <li>Angular</li>
</ul>
</body>
 </html>
```

The type Attribute

You can use **type** attribute for <ul> tag to specify the type of bullet you like. By default, it is a disc. Following are the possible options

```
<ul type = "square">
<ul type = "disc">
<ul type = "circle">
```

HTML Ordered Lists

If you are required to put your items in a numbered list instead of bulleted, then HTML ordered list will be used. This list is created by using **<ol>** tag. The numbering starts at one and is incremented by one for each successive ordered list element tagged with <li>

Ordered List

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Unordered List</title>
</head>
<body>
<ol>
    <li>HTML</li>
    <li>CSS</li>
    <li>Java Script</li>
    <li>Angular</li>
</ol>
</body>
 </html>
```

The type Attribute

You can use **type** attribute for <ol> tag to specify the type of numbering you like. By default, it is a number. Following are the possible options –

<ol type = "1"> - Default-Case Numerals.
<ol type = "I"> - Upper-Case Numerals.
<ol type = "i"> - Lower-Case Numerals.
<ol type = "A"> - Upper-Case Letters.

<ol type = "a"> - Lower-Case Letters.

Html definition/description list

In order to create the definition or description list in html,we have 3 important tags. They are as follows:

<dl> this is root element ofdefinition list </dl>

<dt> defines the definition type or name of data

<dd> defines the data of hat definition type

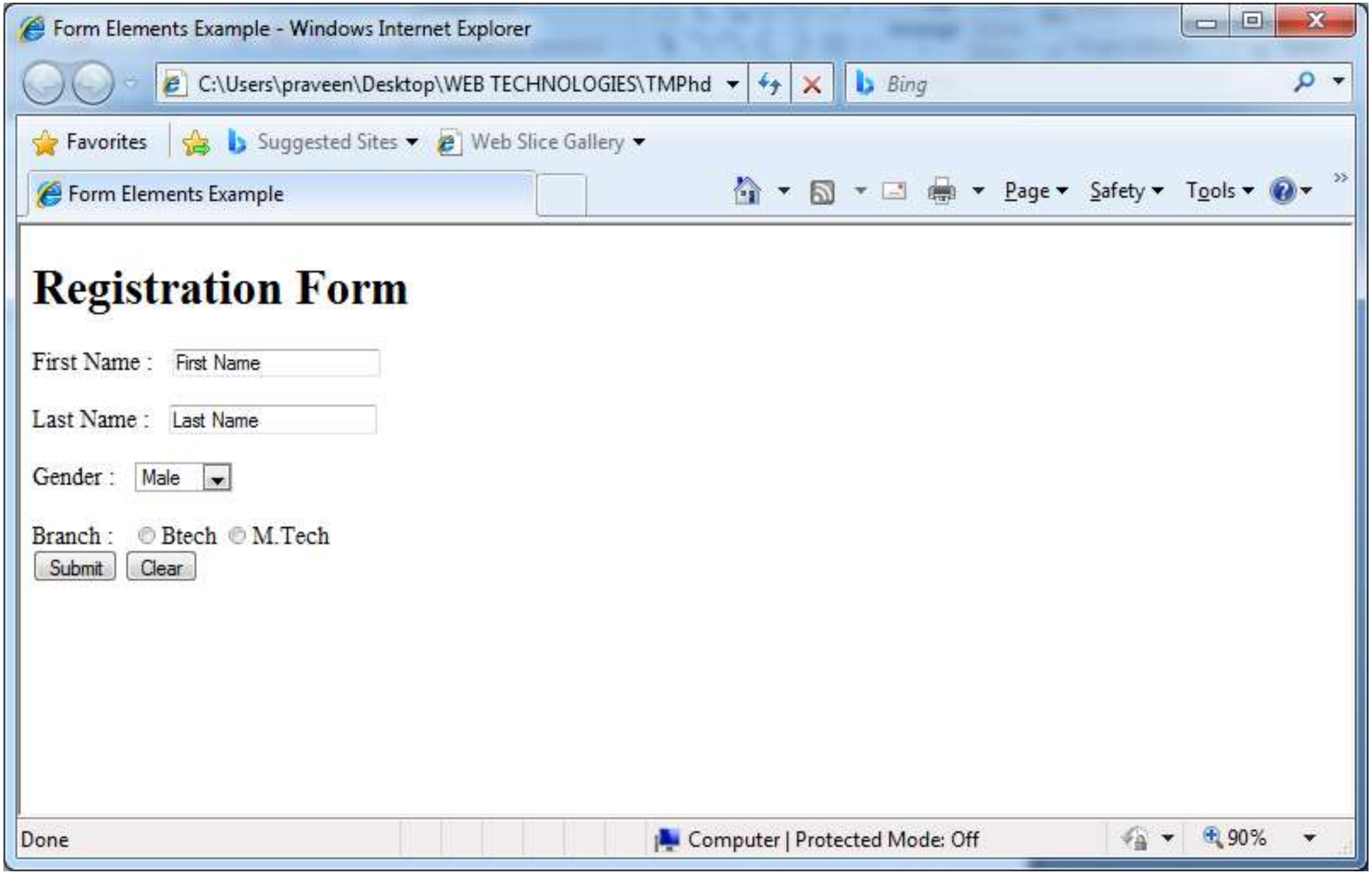
Example on definition list

```
<dl>
    <dt>Programming</dt>
          <dd>C</dd>
          <dd>CPP</dd>
          <dd>JAVA</dd>
    <dt>Designing</dt>
          <dd>HTML</dd>
          <dd>CSS</dd>
</dl>
```

HTML FORMS

HTML Forms are required when you want to collect some data from the site visitor. For example during user registration you would like to collect information such as name, email address, credit card, etc. A form will take input from the site visitor and then will post it to a back-end application such as database, ASP Script or PHP script etc.

Example


Form Elements Example - Windows Internet Explorer

C:\Users\praveen\Desktop\WEB TECHNOLOGIES\TMPhd

Registration Form

First Name : First Name

Last Name : Last Name

Gender : Male

Branch : ○ Btech ○ M.Tech

Submit Clear

Done | Computer | Protected Mode: Off | 90%

Html form tag

form -create an html form, contains form elements . form elements .

```
<form>
  form elements
</form>
```

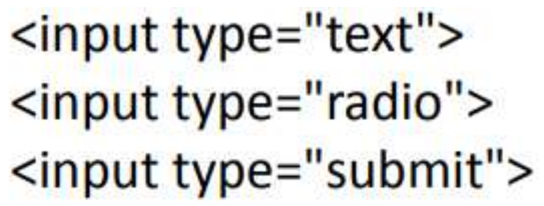
Form elements

Form Elements are text fields, text area fields, drop-down menus, radio buttons, check

boxes.

<u>Sub-Element</u>

<input> element which contains main attribute type.



```
<input type="text">
<input type="radio">
<input type="submit">
```

text type

Text Input

```
<form>
First name: <br>
<input type="text" name="firstname"><br>
Last name:<br>
<input type="text" name="lastname">
</form>
```

Radio Button Input

- Input type=“radio” defines a radio button

```
<html>
<body>
<form>
First name: <br>
<input type="radio" name="gender" value="Male"“ > Male <br>
<input type="radio" name="gender" value="female"> Female<br>
<input type="radio" name="gender" value="other"> Other <br>
</form>
</body>
</html>
```

Other tags

- <input type =“ checkbox” >
- <input type =“ file”>
- <textarea rows=“5” cols=“100”>
- </textarea>

Type = Submit button attribute

```
<body>
<form action="a.txt">
First name:<br>
<input type="text" name="firstname“> <br>
Password:<br>
<input type="password" name="pass"  ><br><br>
<input type="submit" value="Submit">
</form>
```

File Upload Box

```
<body>
<form>
<input type="file" name="fileupload"/>
</form>
</body>
```

Html form Elements

Some of the main elements in HTML form tag is:

- <input>
- <label>
- <select>
- <textarea>
- <button>
- <fieldset>
- <legend>
- <datalist>
- <output>
- <option>
- <optgroup>

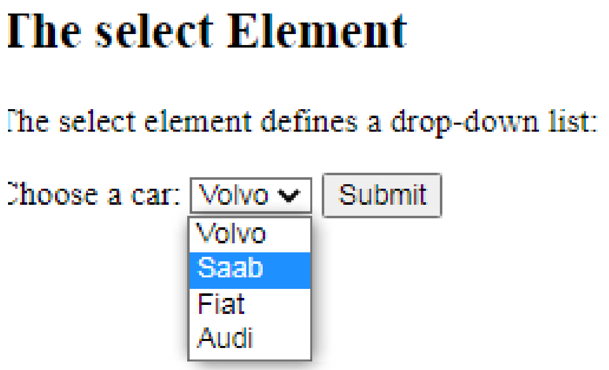
label element

The **<label>** element defines a label for several form elements. The **for** attribute of the **<label>** tag should be equal to the **id** attribute of the <**input>** element to bind them together.

Syntax:

```
<label for="fname">First name:</label>
<input type="text"  id="fname“ >
```

The select element



The <select> Element

- The <select> element defines a drop-down list:

Example

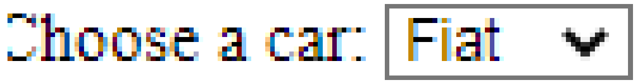
```
<label for="cars">Choose a car:</label>
  <select  id="cars" name="cars">
    <option value="volvo">Volvo</option>
    <option value="saab">Saab</option>
    <option value="fiat">Fiat</option>
    <option value="audi">Audi</option>
  </select>
```

<option> element

- The <option> elements defines an option that can be selected.
- By default, the first item in the drop-down list is selected.
- To define a pre-selected option, add the selected attribute to the option:

Syntax:

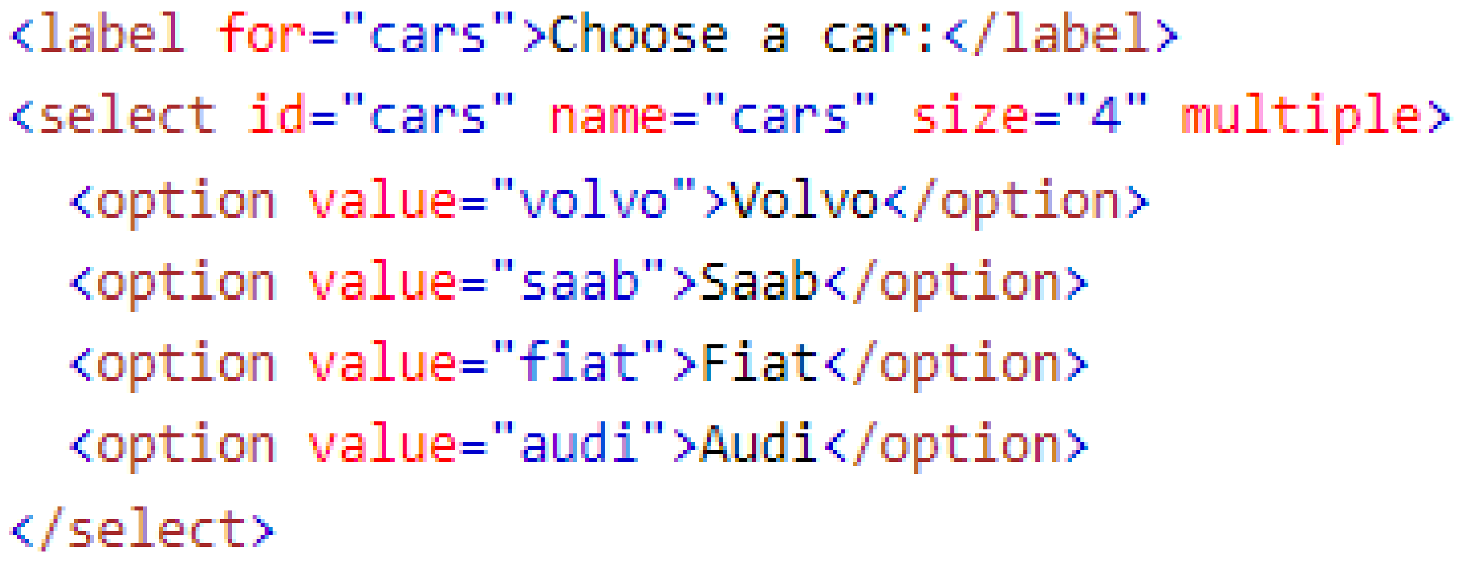
```
<option value="fiat" selected>Fiat</option>
```



Allow multiple selections

- Use the multiple attribute to allow the user to select more than one value:

Syntax:

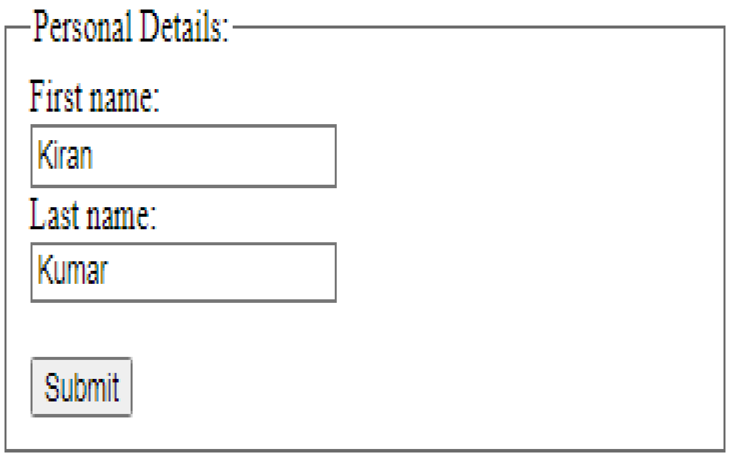


```
<label for="cars">Choose a car:</label>
<select id="cars" name="cars" size="4" multiple>
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
```

The <fieldset> and <legend> Elements

- The <fieldset> element is used to group related data in a form.
- The <legend> element defines a caption for the <fieldset> element.

```
<form action="/action_page.php">
  <fieldset>
    <legend>Personal Details:</legend>
    <label for="fname">First name:</label><br>
    <input type="text" id="fname" name="fname"
value="Kiran"><br>
    <label for="lname">Last name:</label><br>
    <input type="text" id="lname" name="lname"
value="Kumar"><br><br>
    <input type="submit" value="Submit">
  </fieldset>
</form>
```


Personal Details:

First name:

Kiran

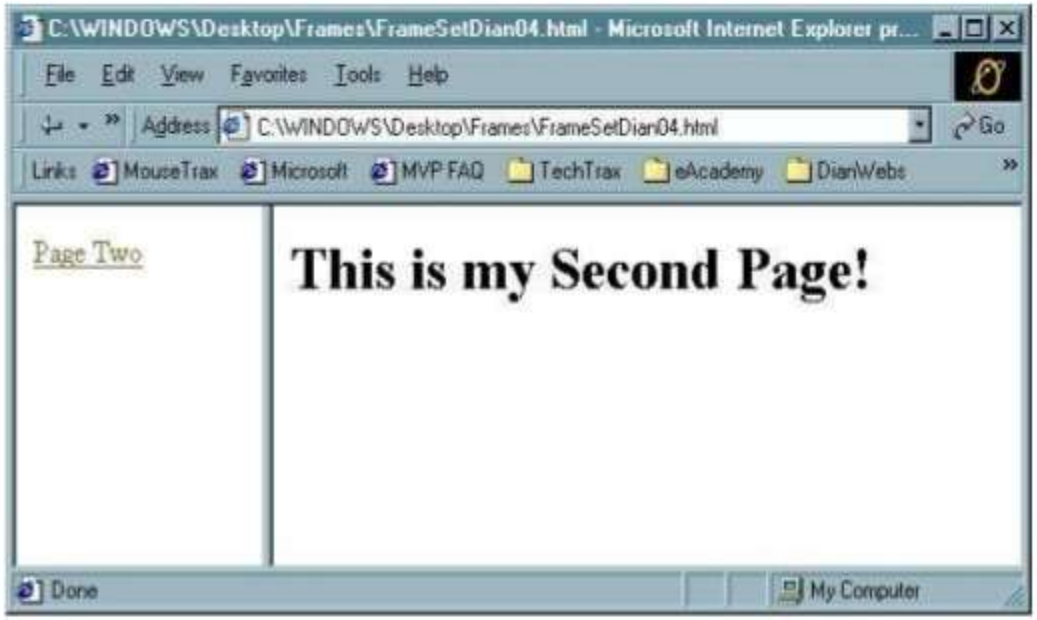
Last name:

Kumar

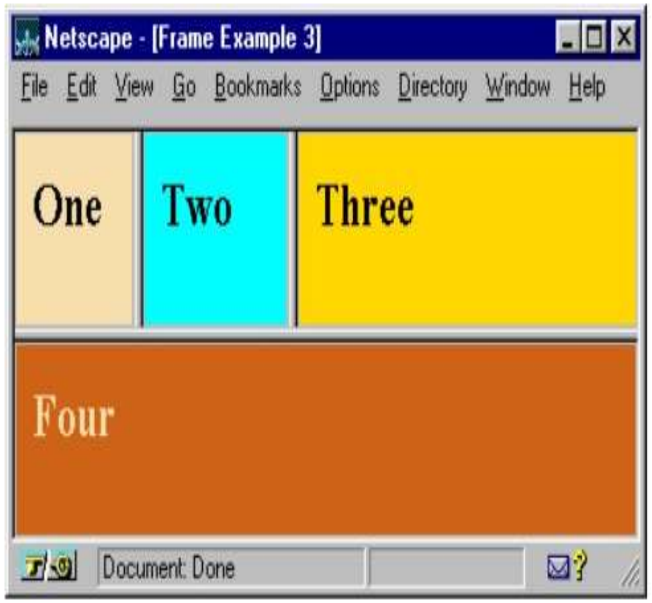
Submit

Html Frames

- Frames – enables the browser to display more than one HTML document on the same window.


C:\WINDOWS\Desktop\Frames\FrameSetDian04.html - Microsoft Internet Explorer pr...
File Edit View Favorites Tools Help
Address C:\WINDOWS\Desktop\Frames\FrameSetDian04.html
Go
Links MouseTrax Microsoft MVP FAQ TechTrax eAcademy DianWebs
Page Two
Done
My Computer

This is my Second Page!


Netscape - [Frame Example 3]
File Edit View Go Bookmarks Options Directory Window Help
One
Two
Three
Four
Document: Done

Frame Creation

HTML Frames -Creating Frames

• Divide your browser window into multiple sections where each section can load a separate HTML document.

• A collection of frames in the browser window - **A frameset.**

• The window is divided into frames with rows and columns.

<frameset> tag instead of <body> tag

• The rows attribute of <frameset> tag defines horizontal frames cols attribute defines vertical frames.

• Each frame is indicated by **<frame> tag and it defines which** HTML document shall open into the frame.

This <frame> tag is currently not used in HTML5, so we need to **use<iframe**> instead of that.

Syntax

```
<html>

<frameset cols="70%,30%">

  <frame src="https://developer.mozilla.org/en/HTML/Element/iframe" />

  <frame src="https://developer.mozilla.org/en/HTML/Element/frame" />

</frameset>

</html>
```

Cascading Style Sheets(CSS)

- CSS stands for Cascading Style Sheets
- Styles define how to display HTML elements
- Styles were added to HTML 4.0 to solve a problem
- External Style Sheets can save a lot of work
- External Style Sheets are stored in CSS files
- The extension for external style sheets is .css

CSS Solved a Big Problem ?

- HTML was never intended to contain tags for formatting a document.
- HTML was intended to define the content of a document, like:

```
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
```

When tags like <font>, and color attributes were added to the HTML 3.2 specification, it started a nightmare for web developers.

Development of large web sites, where fonts and color information were added to every single page, became a long and expensive process.

To solve this problem, the World Wide Web Consortium (W3C) created CSS.

In HTML 4.0, all formatting could be removed from the HTML document, and stored in a separate CSS file.All browsers support CSS today.

CSS Syntax

CSS Syntax

The CSS syntax is made up of three parts: a selector, a property and a value:

selector { property : value ;Font-family:”times” ;color:red ; size:10; }

The selector is normally the **HTML element/tag** you wish to define, the property is the **attribute** you wish to change, and each property can take a value. The property and value are separated by a **colon**, and surrounded by curly braces:

For example

```
body {font-family: times new roman;
color: blue;width:20pt;}
```

Or else we can also define as follows:

```
p
 {
font-family : "sans serif“
}
H1{}
H2{}
```

Multiple Attributes in one selector

If you wish to specify more than one property, you must separate each property with a semicolon. The example below shows how to define a center aligned paragraph, with a red text color:

Example:

```
p {
text-align:center;
color:red;
}
```

Levels of Style Sheets

The CSS can be declared in 3 ways:

1) External style sheet

2) Internal style sheet (inside the <head> tag)

3)Inline style (inside an HTML element)

1)Internal Style Sheet

An internal style sheet should be used when a **single document** has a unique style. You define internal styles in the head section with the <style> tag.

Example:

```
<head>
<style type="text/css">
body
{background-color: red}
p
{margin-left: 20px}
</style>
</head>
```

2) **Inline Styles**

An inline style should be used when a unique style is to be applied to a single occurrence of an element.

To use inline styles you use the style attribute in the relevant tag. The style attribute can contain any CSS property. The example shows how to change the color and the left margin of a paragraph:

For Example

```
<p style="color: red; margin-left: 20px">
                This is a paragraph</p>
```

3)External Style Sheet

An external style sheet is ideal, when the style is applied **to many pages**. Each page must link to the style sheet using the <link> tag. The <link> tag goes inside the head section.

```
<head>
 <link rel="stylesheet" type="text/css"
   href="mystyle.css">
</head>
```

Multiple Style Sheets

If some properties have been set for the same selector in different style sheets, the values will be inherited from the more specific style sheet. For example, an external style sheet has these properties for the h3 selector:

```
h3
{
color: red;
text-align: left;
font-size: 8pt
}
```

And an internal style sheet has these properties for the h3 selector:

```
h3
 {
text-align: right;
 font-size: 20pt
}
```

If the page with the internal style sheet also links to the external style sheet the properties for h3 will be:

Then the h3 properties will be as follows:

color: red;

text-align: right;

font-size: 20pt

The color is inherited from the external style sheet and the text-alignment and the font-size is replaced by the internal style sheet.

Example on CSS

```
<html>
<head>
<style type="text/css">
body {background-color: yellow}
h1 {background-color: #00ff00}
h2 {background-color: transparent}
p   {background-color: rgb(250,0,255)}
</style>
</head>
<body>
<h1> Java made simple </h1>
<h2> Introduction  </h2>
 <p> Java was invented by SunMicro … </p>
</body>
</html>
```

CSS Selectors

CSS selectors are used to "find" (or select) the HTML elements you want to style

1)CSS element selector

2)CSS id selector

3) CSS class selector

4)CSS Universal selector

5) CSS Generic selector

1) CSS Element Selector

The element selector selects HTML elements based on the element name.

For example: If we want to align the paragraph to center and change font color to red.

Syntax:

```
 p
{
    text-align: center;
    color: red;
  }
```

Simple selector form

The selector is a tag name or a list of tag names, separated by commas

Consider the following examples, in which the property is font-size and the property value is a number of points :

```
h1, h3
{
font-size: 24pt ;
}
 h2
 {
font-size: 20pt ;
}
```

2. Css Id Selector

An id selector allow the application of a style to one specific element.

General form:

```
 #specific-id
        {
     property-value list
}
```

Example:

```
#section14
{
font-size: 20;
}
```

Specifies a font size of 20 points to the

<h2 id = “section14” > Hello <h2>

3. CSS Class Selector

Used to allow different occurrences of the same tag to use different style specifications .A style class has a name, which is attached to a tag name

p.normal {property/value list}

p.warning{property/value list}

The class you want on a particular occurrence of a tag is specified with the class attribute of the tag.

For example,

<p class = “normal”> A paragraph of text that we want to be presented in ‘normal’ presentation style </p>

<p class = “warning” > A paragraph of text that is a warning to the reader ,which should be presented in an especially noticeable style. </p>

4. CSS Universal Selector

The universal selector denoted by an asterisk(*).

It applies its style to all elements in the document

For Eg:

```
 *
{
   text-align: right;
}
```

Makes all elements in the document align right side

5. CSS Generic Selector

- A generic class can be defined if you want a style to apply to more than one kind of tag. A generic class must be named, and the name must be.

Example,

.large {property : value list }

- Use it as if it were a normal style class

<h1 class = “large"> … </h1>

…

<p class = “large"> … </p> in with a period

CSS Properties

CSS include 60 different properties in 7 categories:

- Fonts
- Lists
- Alignment of text
- Margins
- Colors
- Backgrounds
- Borders

1)Fonts

1.Font-Families

- The font-family property is used to specify a list of font name.
- The browser will use the first font in the list that it supports. For example, the following could be specified.

Example: font- family: Arial, Helvetica, Courier

2.font-size

Sets the size of fonts. There are two categories of font-size values, absolute and relative.In the case of absolute category the size value could be given as length value in points, picas or pixels or keywords from the list xx-small, x-small, small, medium, large and x-large.

Eg: **font-size: 10pt**

The relative size values are smaller and larger, which adjust the font size relative to the font size of the parent element.

Eg: **font-size: 1.2em**

This sets the font size to 1.2 times the font size of the parent element.1.2em and 120% are same.

3.Font Variant

- The default value of the font-variant property is **normal, which** specifies the usual character font.
- This property can be set to **small-caps** to specify small capital letters.

4. **font-style**

Most commonly used to specify italic.

Eg: font-style: italic

5. font-weight

- Used to specify degrees of boldness.

  Eg: font-weight: **bold**

- Possible values are **bolder, lighter, bold, normal(default)**

6. Font Shorthands

For specifying more than one font properties. The values can be stated in a list as value of the font property.

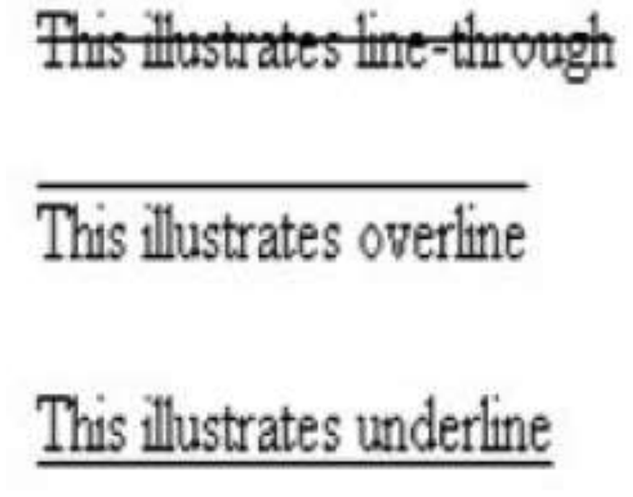
Eg: font: bold 14pt Arial Helvetica

The order which browser follows is last must be font name, second last font size and then the font style, font variant and font weight can be in any order but before the font size and names.

7. **The text-decoration property**

Specifies some special features of text. Possible values of text-decoration property are : line-through, overline, underline, none

Suppose if we apply these for a paragraph tag



~~This illustrates line-through~~

This illustrates overline

This illustrates underline

8. List properties

- Property Name: **list-style-type** can applied to both ordered and unordered list.

Unordered lists

- Bullet can be a disc (default), a square, or a circle. Set it on either the <ul> or <li> tag. On

<ul>, it applies to list items.

```
<style type = “text/CSS”>
 ul
{
list-style-type:square;
}
</style>
```

ORDERED LIST

On ordered lists - list-style-type property can be used to change the sequence values

Property value	***Sequence type***	***First four***
Decimal	Arabic numerals	1, 2, 3, 4
upper-alpha	UC letters	A, B, C, D
lower-alpha	lc letters	a, b, c, d
upper-roman	Uc Roman	I, II, III, IV
lower-roman	Lc Roman	i, ii, iii, iv

9.CSS Color properties

Color is a problem for the Web for two reasons:

1. Monitors vary widely
2. Browsers vary widely

There are three color collections

1)color: color name;

2)color: rgb(100,255,125);

3)color: #01a2cf

For hexa decimal we need to enter the color range in 0 to f letters only.

10. Alignment of text

The **text-align** property has the possible values, left (the default), center, right, or justify.

text-align: center;

Web Technologies

Unit 2: Contents

UNIT-II: Working with XML: Introduction, The syntax of XML, XML Document Structure, Document type Definition (DTD), Namespaces, XML schemas, XSLT, XML Parsers - DOM and SAX

Introduction to XML

1) What is XML?

A. XML stands for Extensible Markup Language

B. XML is a markup language much like HTML

C. XML was designed to carry data, not to display data

D. XML tags are not predefined. You must define your own tags

E. XML is designed to be self-descriptive

F. XML is a W3C Recommendation(World Wide Web Consortium)

XML Syntax

XML syntax refers to the rules that determine how an XML application can be written. The XML syntax is very straight forward, and this makes XML very easy to learn. Below are the main points to remember when creating XML documents.

XML documents must contain one **root** element that is the **parent** of all other elements

XML Document Structure

```
<? xml version="1.0" encoding="UTF-
  8"?>
<root>
   <child>
     <subchild>.....  </subchild>
   </child>
</root>
```

XML Prolog

<?xml version="1.0" encoding="UTF-8"?> is also known as **XML prolog**

Features of XML

Tag based language. tags are used to describe the content of the document

- Tags are defined by user and not by language
- Case sensitive and strict.
- Endorsed and maintained by **w3c** and used by all major companies like microsoft, oracle, Sun and IBM.
- Platform independent and language independent –any language on any platform can read process the data in xml document.
- With xml,white space is preserved

What is Well-formed XML

Ever xml document must be a well formed xml document. A well-formed XML is an document that compiles with following rules:

1. It has a single root element
2. Ever tag is opened and closed. Tags must be properly nested.
3. Values of an attribute is enclosed is quotes-single or double.

Course.xml

```
<? xml version=”1.0” encoding=“UTF-8” ?>
<course>
<name>java</name>
<fee  currency=”INR”>7500</fee>
<prereqsite> c  Language done by thomasa
  asfsafsafsafassfasfsafsafsafsafsafsa</prer
  eqsite>
</course>
```

Team.xml

```
<?xml version=”1.0” encoding=“UTF-8” ?>
   <team name=”Men InBule”
       <player>
            <name>M.S Dhoni</name>
             <age> 30</age>
        </player>
         <player>
              <name>ViratKohli</name>
               <age> 25</age>
          </player>
          <player>
                <name>Rohit</name>
                  <age> 28</age>
          </player>
</team>
```

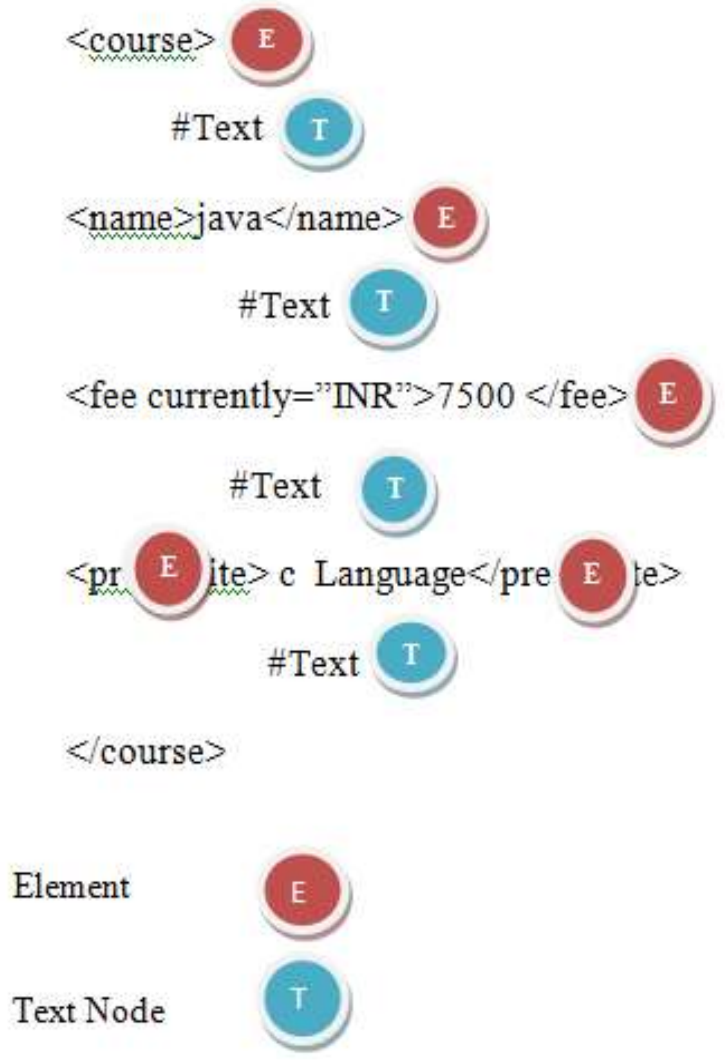
Why XML is Used?

We developers uses the XML files for following purposes:

- Storing the data for some application such as menu data or data for some comobox
- For developing the database driven websites.
- In image gallery application is can work as the data file (xml) /storing the names and location of the images.
- For shopping application it can be used to store the product details
- In travel applications XML data can be used to talk to booking gateway.
- On the web web services such as Weather services, Currency rates service etc. are using the XML language

Document Object Model

A DOM is a standard tree structure, where each node contains one of the components from an XML structure. The two most common types of nodes are **element nodes** and **text nodes**. Using DOM functions lets you create nodes, remove nodes, change their contents, and traverse the node hierarchy.



Why Use a DTD (Document Type Definition)

With a DTD, each of your XML files can carry a description of its own format. With a DTD, independent groups of people can agree to use a standard DTD for interchanging data. Your application can use a standard DTD to verify that the data you receive from the outside world is valid. You can also use a DTD to verify your own data. Seen from a DTD point of view, all XML documents (and HTML documents) are made up by the following building blocks:

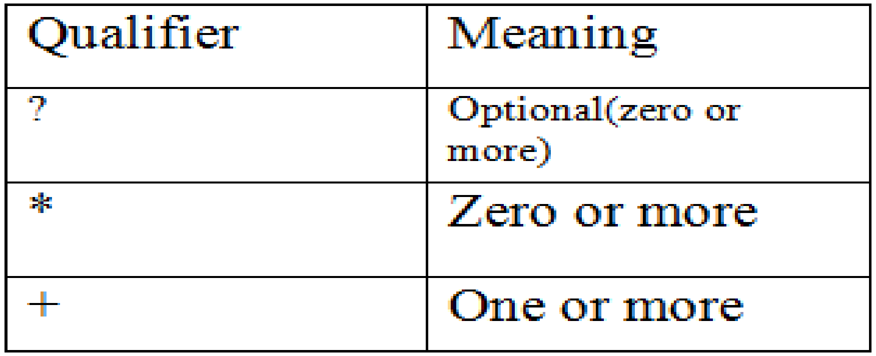
1. Elements
2. Attributes
3. Entities
4. PCDATA (Parsed Character Data)
5. CDATA (Character Data)

PCDATA

- PCDATA means parsed character data.
- Think of character data as the text found between the start tag and the end tag of an XML element.
- PCDATA is text that WILL be parsed by a parser. The text will be examined by the parser for entities and markup.
- Tags inside the text will be treated as markup and entities will be expanded.
- However, parsed character data should not contain any &, <, or > characters; these need to be represented by the &amp; &lt; and &gt; entities, respectively.

CDATA

- CDATA means character data.
- CDATA is text that will NOT be parsed by a parser. Tags inside the text will NOT be treated as markup and entities will not be expanded.
- the qualifiers you can add to an element definition are listed in below:



Qualifier	Meaning
?	Optional(zero or more)
*	Zero or more
+	One or more

You can specify what type of data an element can contain parsed character data (PCDATA) or CDATA section ,which contain character data that is not parsed. The**#** that precedes PCDATA indicates that what follows is a special word rather than an element name.

Lab Task Exercise 4

Exercise 4:

Write an XML file which will display the Book information which includes the following: (i) Title of the book (ii) Author Name (iii) ISBN number (iv) Publisher name (v) Edition (vi) Price

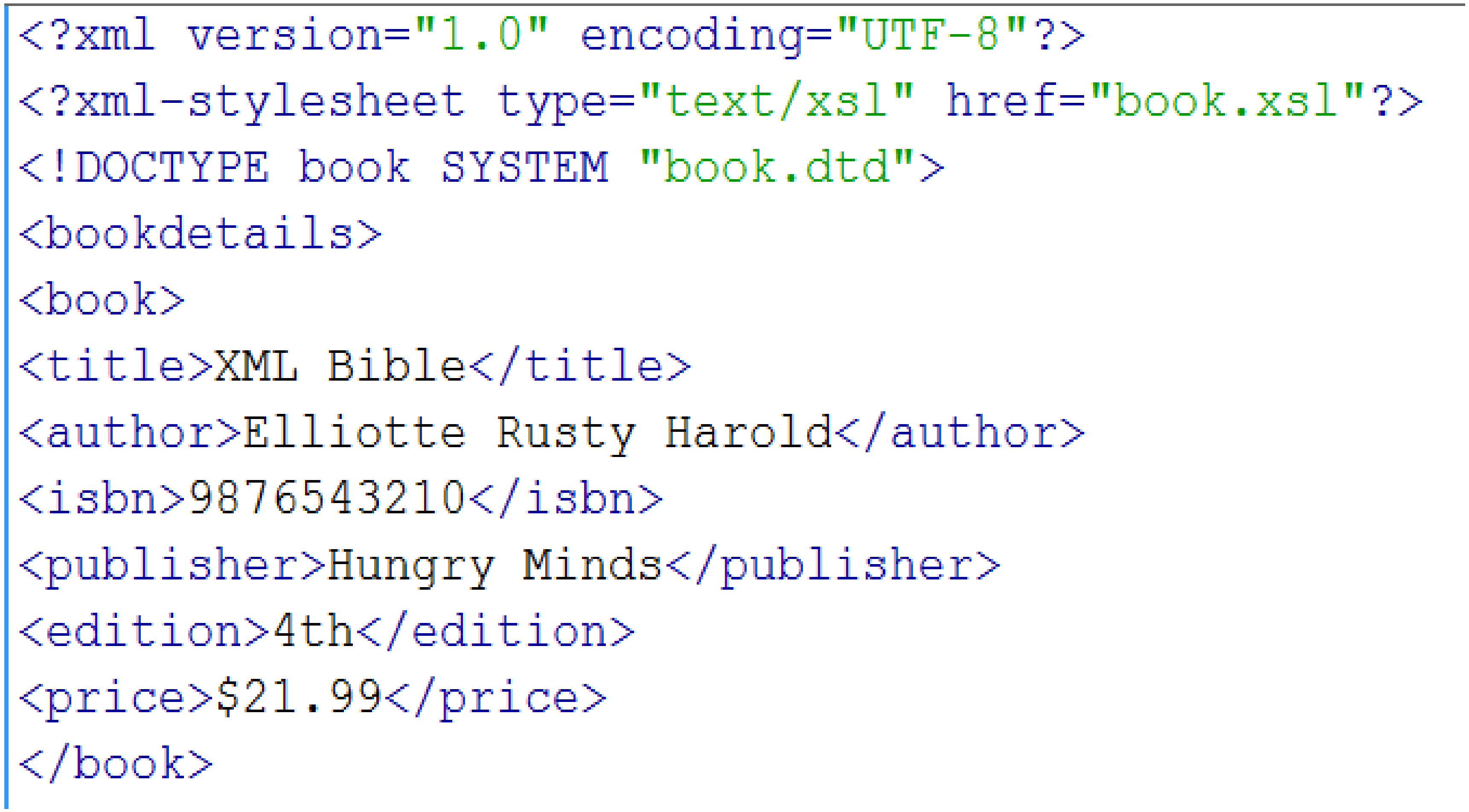
(a) Write a Document Type Definition (DTD) to validate the above XML file.

(b) Write a XML Schema Definition (XSD)

Sample DTD for Book

```
<!ELEMENT bookdetails (book+)>
<!ELEMENT book (title,author,isbn,publisher,edition,price)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT edition (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

Book.xml

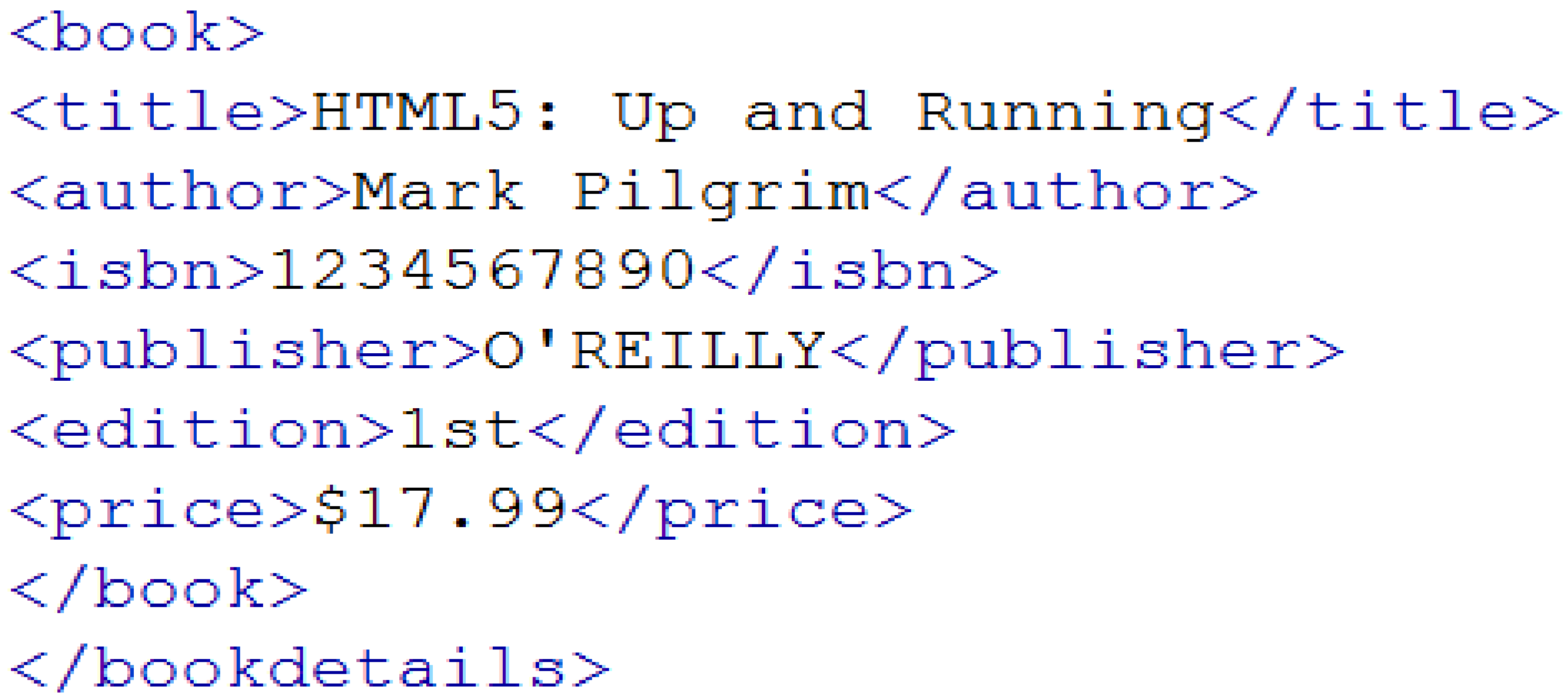


```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="book.xsl"?>
<!DOCTYPE book SYSTEM "book.dtd">
<bookdetails>
<book>
<title>XML Bible</title>
<author>Elliotte Rusty Harold</author>
<isbn>9876543210</isbn>
<publisher>Hungry Minds</publisher>
<edition>4th</edition>
<price>$21.99</price>
</book>
```

Contd..

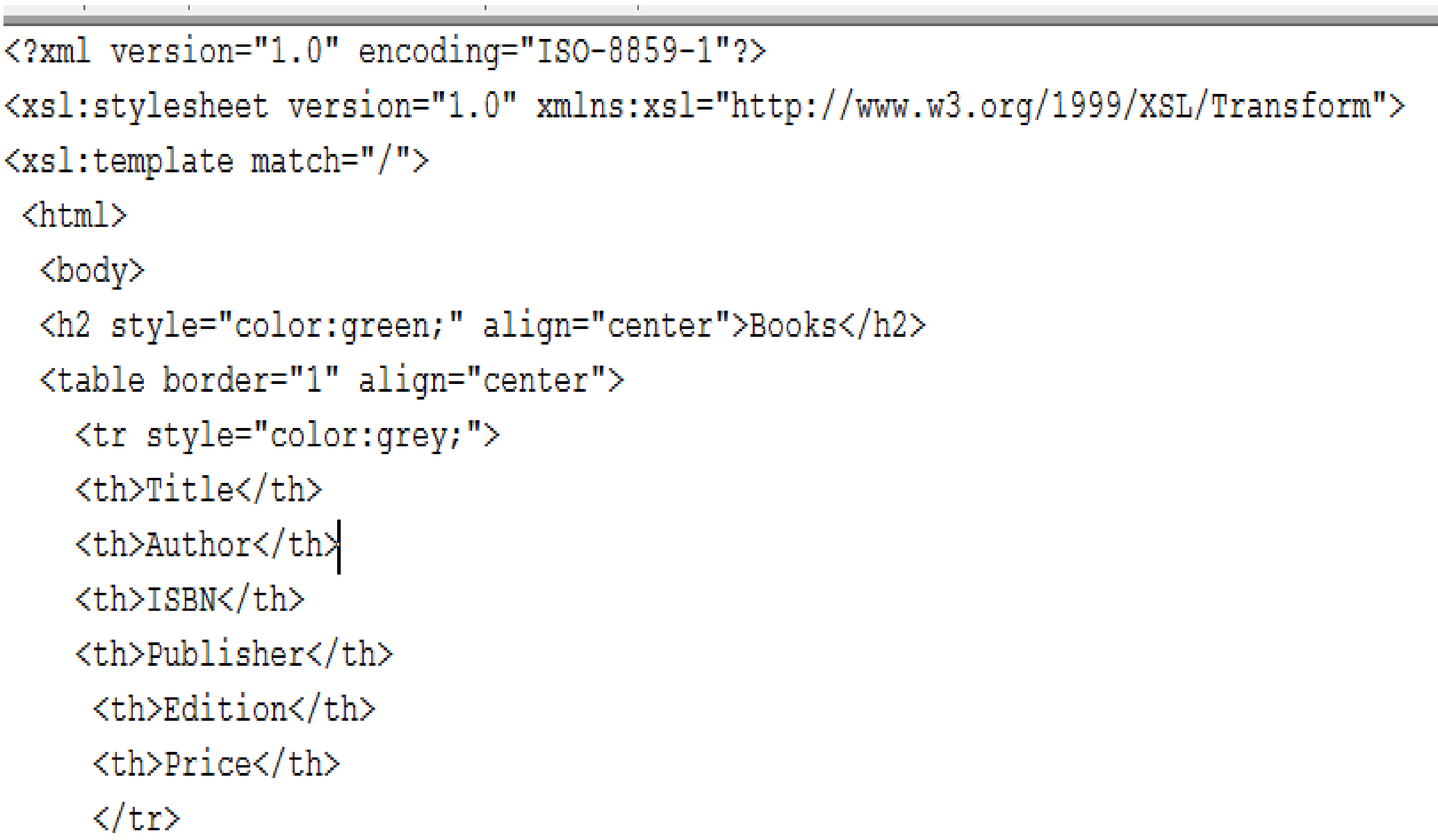
```xml
<book>
<title>AI: A Modern Approach</title>
<author>Stuart J. Russell</author>
<isbn>9876543220</isbn>
<publisher>Princeton Hall</publisher>
<edition>6th</edition>
<price>$36.09</price>
</book>
<book>
<title>Beginning Java 2</title>
<author>Ivor Horton</author>
<isbn>9876543220</isbn>
<publisher>wrox</publisher>
<edition>3th</edition>
<price>$8.95</price>
</book>
```

Contd..



```
<book>
<title>HTML5: Up and Running</title>
<author>Mark Pilgrim</author>
<isbn>1234567890</isbn>
<publisher>O'REILLY</publisher>
<edition>1st</edition>
<price>$17.99</price>
</book>
</bookdetails>
```

book.xsl



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
 <html>
  <body>
  <h2 style="color:green;" align="center">Books</h2>
  <table border="1" align="center">
    <tr style="color:grey;">
    <th>Title</th>
    <th>Author</th>
    <th>ISBN</th>
    <th>Publisher</th>
     <th>Edition</th>
     <th>Price</th>
     </tr>
```

Contd…

```
   <xsl:for-each select="bookdetails/book">
  <tr style="width:70%">

  <td style="font-family:'Comic Sans MS';
             color:red;width:70%">
      <xsl:value-of select="title"/>
 </td>

 <td style="text-transform: capitalize;
          font-weight:bold;" align="center">
     <xsl:value-of select="author"/>
 </td>

 <td style="color:blue">
      <xsl:value-of select="isbn"/>
 </td>
```

```
  <td style="color:green; font-weight:bold;"
             align="center">
       <xsl:value-of select="publisher"/>
  </td>
   <td style="pink"  align="center">
       <xsl:value-of select="edition"/>
   </td>
    <td style="color:violet; font-weight:bold;">
       <xsl:value-of select="price"/>
   </td>
</tr>
        </xsl:for-each>
       </table>
    </body>
  </html>
 </xsl:template>
</xsl:stylesheet>
```

Ways to declare DTD

- A Document Type Definition (DTD) defines the legal building blocks of an XML document. It defines the document structure with a list of legal elements and attributes.
- A DTD can be declared inline inside an XML document, or as an external reference.

Internal DTD Declaration

- If the DTD is declared inside the XML file, it should be wrapped in a DOCTYPE definition with the following syntax:

<!DOCTYPE root-element [element-declarations]>

External DTD Declaration

- If the DTD is declared in an external file, it should be wrapped in a DOCTYPE definition with the following syntax:

<!DOCTYPE root-element SYSTEM "filename">

XML Namespaces

- XML Namespaces provide a method to avoid el

Name Conflicts

- In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.

```
<table>
   <tr>
    <td>Apples</td>
    <td>Bananas</td>
   </tr>
</table>
```

- In the second XML file, the sample table element takes different values:

```
<table>
   <name>African Coffee Table</name>
   <width>80</width>
   <length>120</length>
</table>
```

If these XML fragments were added together, there would be a name conflict. Both contain a <table> element, but the elements have different content and meaning. A user or an XML application will not know how to handle these differences.

Solving the Name Conflict Using a Prefix

Name conflicts in XML can easily be avoided using a name prefix.This XML carries information about an HTML table, and a piece of furniture:

```
<h:table>
    <h:tr>
      <h:td>Apples</h:td>
      <h:td>Bananas</h:td>
    </h:tr>
</h:table>
<f:table>
    <f:name>African Coffee Table</f:name>
    <f:width>80</f:width>
    <f:length>120</f:length>
</f:table>
```

XML Namespaces - The xmlns Attribute

- When using prefixes in XML, a **namespace** for the prefix must be defined.
- The namespace can be defined by an **xmlns** attribute in the start tag of an element.
- The namespace declaration has the following syntax. xmlns:*prefix*="*URI*".

Here **Uniform Resource Identifier** (URI) is a string of characters which identifies an Internet Resource.

Example on XML Namespaces

```
<root xmlns:h="http://www.w3.org/TR/html4/"
   xmlns:f="https://www.w3schools.com/furniture">
<h:table>
     <h:tr>
       <h:td>Apples</h:td>
       <h:td>Bananas</h:td>
     </h:tr>
</h:table>
<f:table>
     <f:name>African Coffee Table</f:name>
     <f:width>80</f:width>
     <f:length>120</f:length>
</f:table>
</root>
```

XML schemas

- An XML Schema describes the structure of an XML document, just like a DTD.
- An XML document with correct syntax is called "Well Formed".
- An XML document validated against an XML Schema is both "Well Formed" and "Valid".
- XML Schemas are extensible to future additions
- XML Schemas are richer and more powerful than DTDs
- XML Schemas are written in XML
- XML Schemas Support data types
- XML Schemas supports namespace

Element	**Meaning**
<simpleType>	Describes a simple element
<complexType>	Describes a complex type
<element>	Describes a simple element
<choice>	Allows one of the specified element
<sequence>	Needs elements in the given sequence
<attribute>	Describes and attribute of an element

Most common datatypes

xs:string,

xs:decimal,

xs:integer,

xs:boolean,

xs:date,

xs:time

Example:

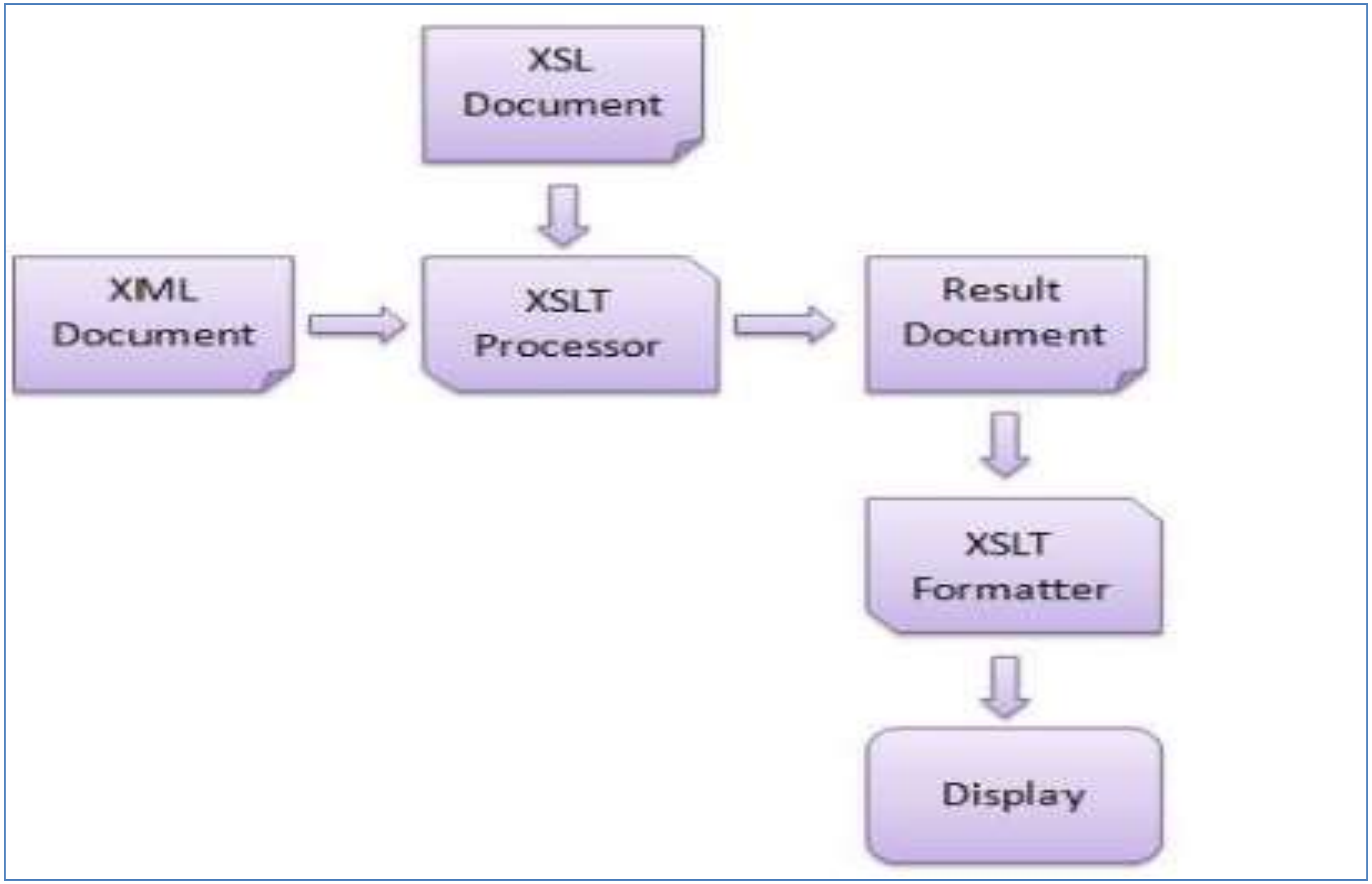
```
<xs:element name=”lastname” type=”xs:string”/>
<xs:element name=”age” type=”xs:integer”/>
<xs:element name=”dateborn” type=”xs:date”/>
```

XSLT(Extensible Style Sheet Language Transformation)

Before XSLT, first we should learn about XSL. XSL stands for **eXtensible Style sheet Language**. It is a styling language for XML just like CSS is a styling language for HTML.

XSLT stands for **XSL Transformation**. It is used to transform XML documents into other formats (like transforming XML into HTML).

Architecture of XSLT



Syntax of XSLT

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

<xsl:stylesheet version="1.0“>

<xsl:template  match="/">

<xsl:for-each  select=“xml root element/parent element">

             <xsl:value-of select=“xml child element name"/>

 </xsl:for-each>

</xsl:template>

</xsl:stylesheet>
```

If we want to link xml style sheet for an XML page

<?xml-stylesheet type = "text/xsl" href = "filename.xsl"**?>**

XML Parsers - DOM and SAX

There are two types of XML parsers namely Simple API for XML and Document Object Model.

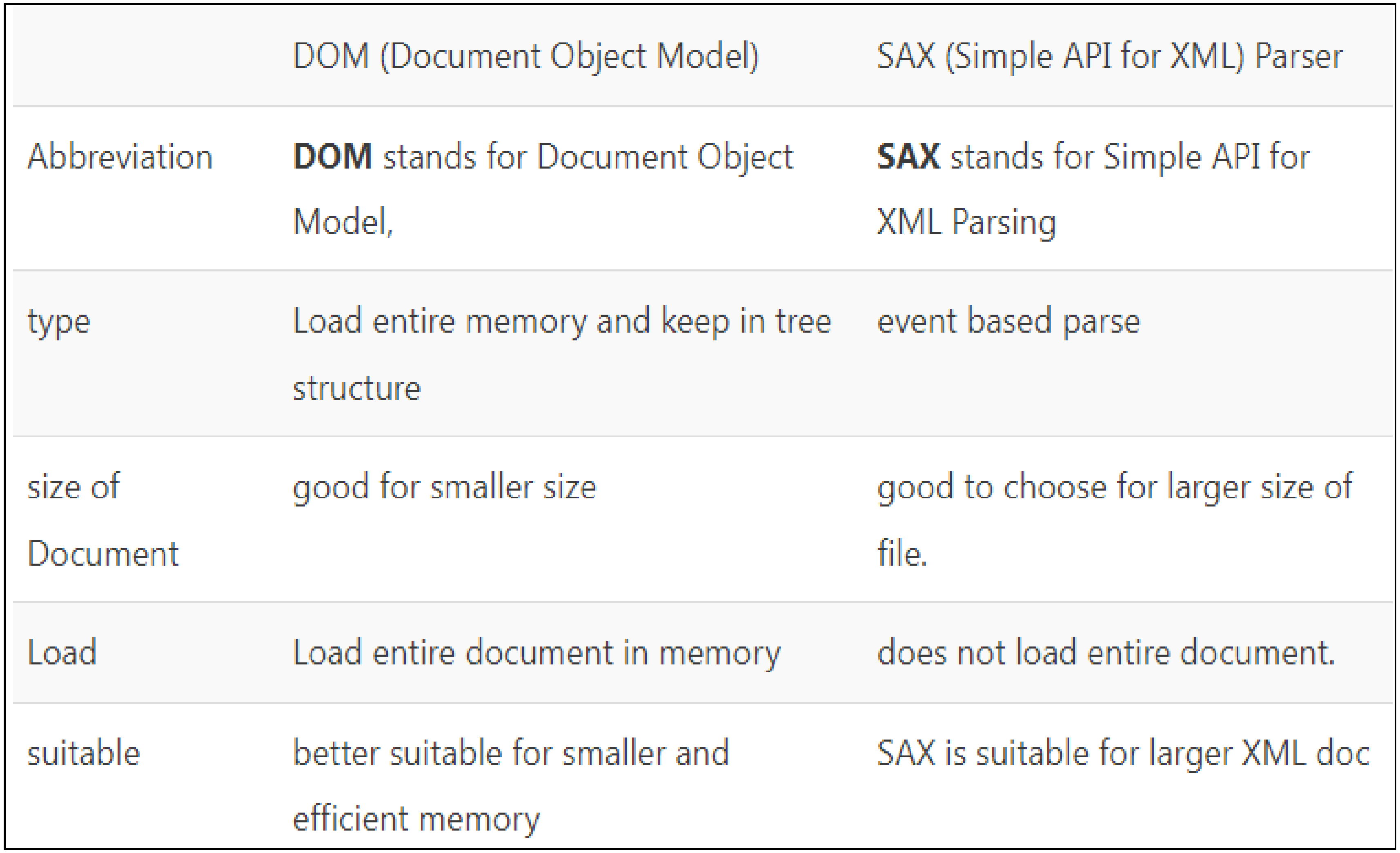
1. SAX (Simple API for XML)
2. DOM(Document Object Model)

The objective of **DOM** (**Document Object Model**) parser and **SAX** (**Simple API for XML**) parser are same but implementation is different. Both the parser work in different way internally, but intent of both are same. Internal implementation of DOM Vs SAX is different. It means, with same intent philosophy of the implementation are different. In order to understand the difference between DOM and SAX, you have to understand each one of the parsers.

Key Differences between DOM & SAX parsers

- DOM parser load full XML file in-memory and creates a tree representation of XML document, while SAX is an event based XML parser and doesn’t load whole XML document into memory.
- If you know you have sufficient amount of memory in your server you can choose DOM as this is faster because load entire XML in-memory and works as tree structure which is faster to access.
- As a thumb rule, for small and medium sized XML documents, DOM is much faster than SAX because of in memory management.
- As a thumb rule, for larger XML and for frequent parsing, SAX XML parser is better because it consume less memory.

Differences



	DOM (Document Object Model)	SAX (Simple API for XML) Parser
Abbreviation	**DOM** stands for Document Object Model,	**SAX** stands for Simple API for XML Parsing
type	Load entire memory and keep in tree structure	event based parse
size of Document	good for smaller size	good to choose for larger size of file.
Load	Load entire document in memory	does not load entire document.
suitable	better suitable for smaller and efficient memory	SAX is suitable for larger XML doc

THANK YOU

Web Technologies

Unit 3: Contents

UNIT-III: WORKING WITH DATABASE:

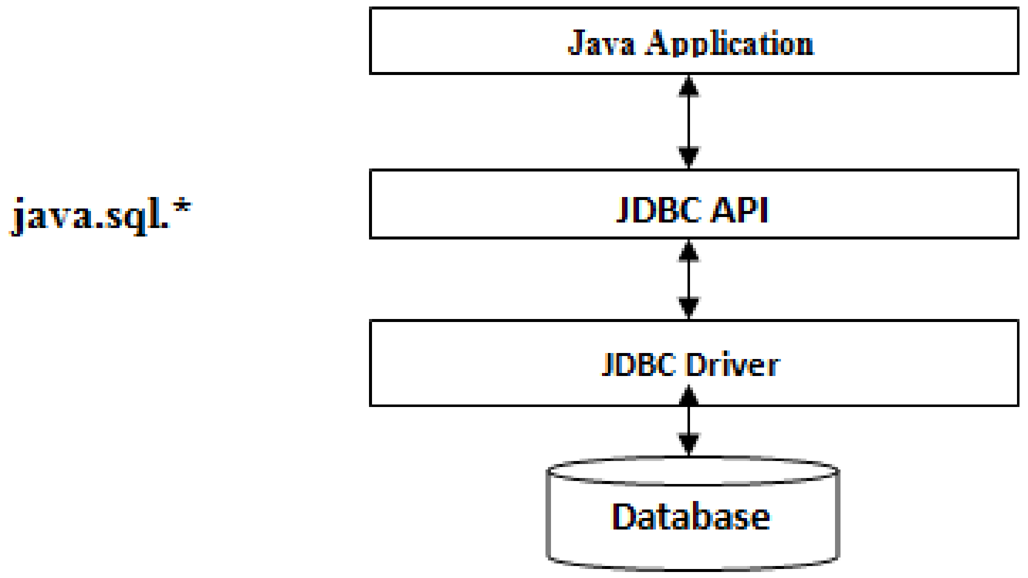
Getting started with JDBC , Defining ODBC, Introduction to JDBC, Components of JDBC, JDBC Architecture, Types of Drivers, Working with JDBC APIs, Creating a Simple Application, Working with Prepared Statement.

Getting Started with JDBC

1) What is JDBC?

Java Database Connectivity in short called as JDBC. It is a java API which enables the java programs to execute SQL statements. It is an application programming interface that defines how a java programmer can access the database in tabular format from Java code using a set of standard interfaces and classes written in the Java programming language.

The **JDBC DriverManager** class defines objects which can connect Java applications to a JDBC driver. DriverManager has traditionally been the backbone of the JDBC architecture. It is quite small and simple.

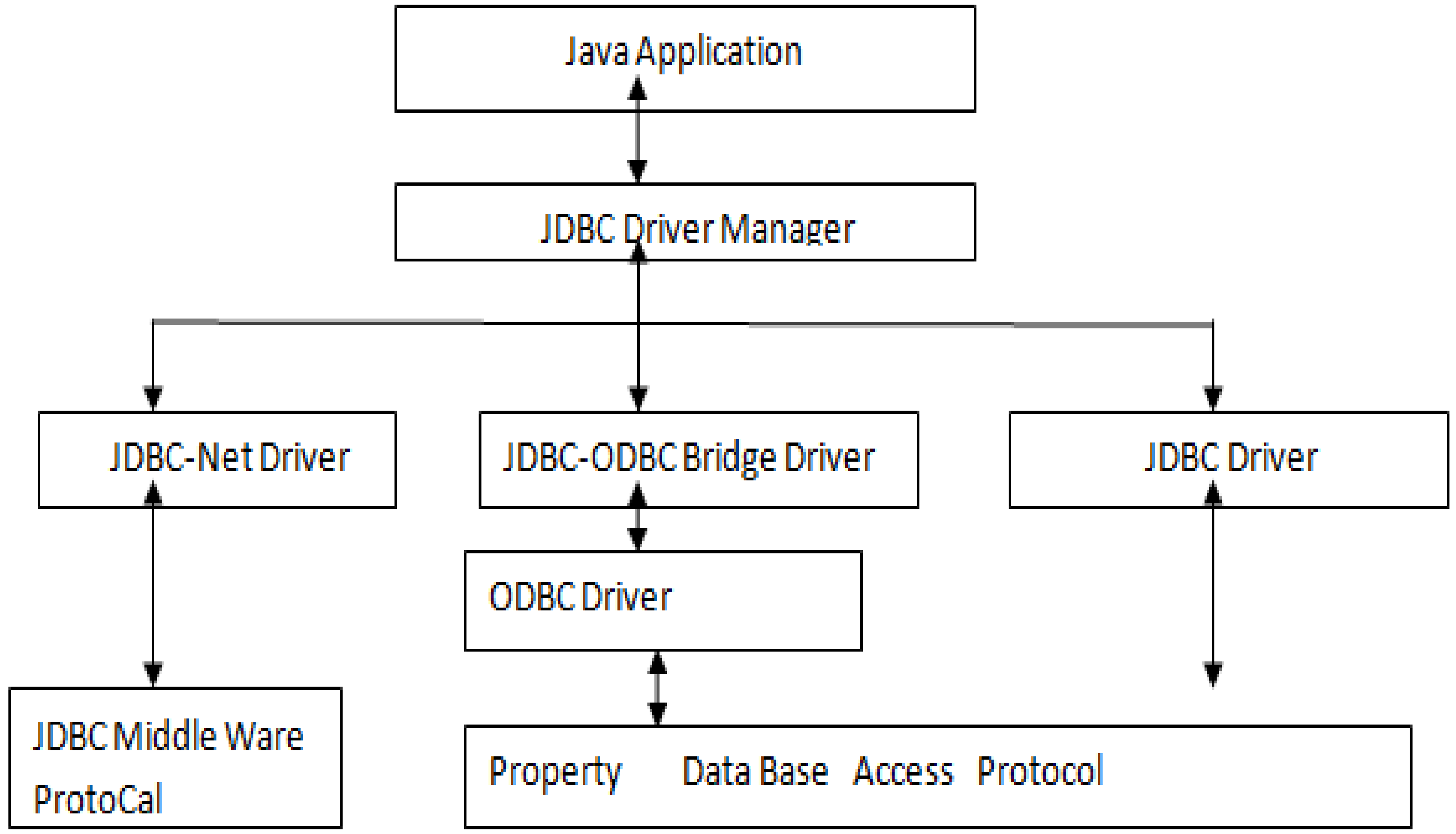


JDBC Components

JDBC provides the following components as part of the JDK.

JDBC Driver Manager	The JDBC driver manager is the backbone of the JDBC Architecture. It actually is quite small and simple; its primary function is to connect Java applications to the correct JDBC Driver and then get out of the way.
JDBC-ODBC Bridge	The JDBC-ODBC bridge allows ODBC drivers to be used as JDBC drivers. It provides a way to access less popular DBMS.If JDBC driver is not implemented for it.

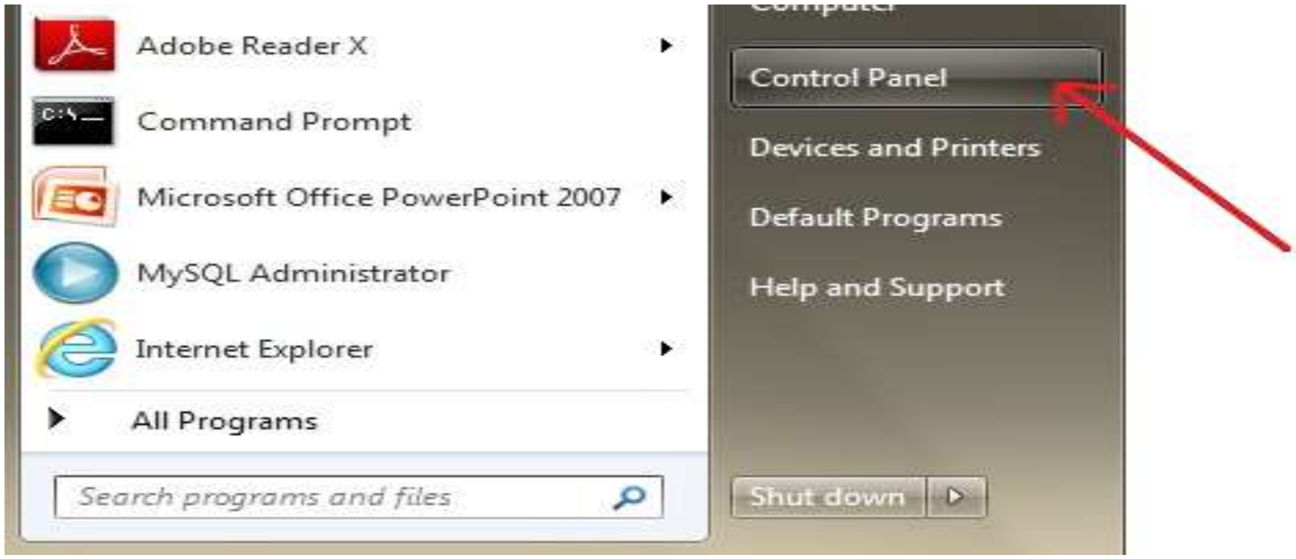
JDBC Driver Component Architecture



Defining ODBC

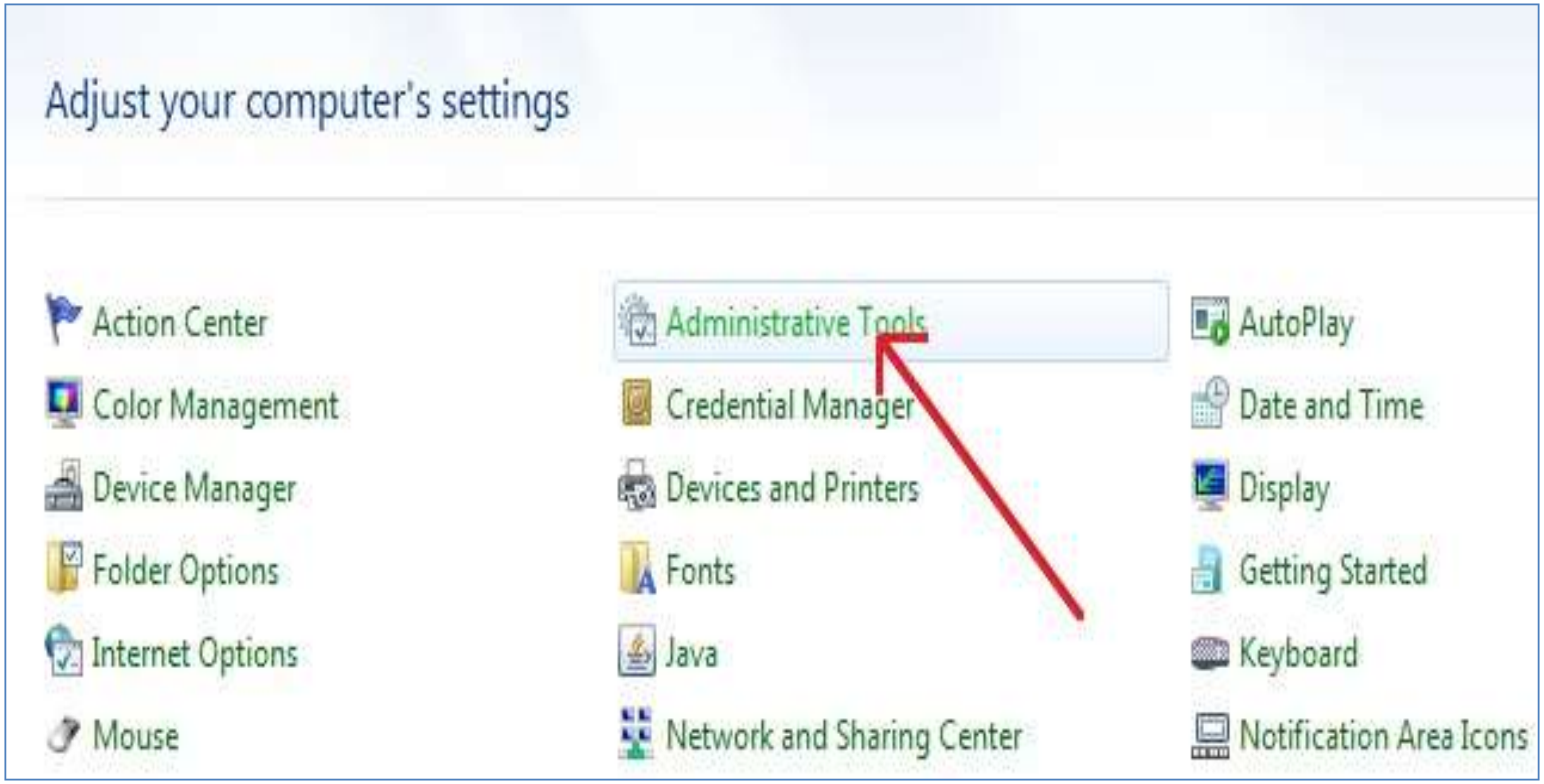
ODBC is known as Open Database Connectivity, this is present in built in every operating system. The following are the steps to check where the ODBC is present in our operating system.

Go to **Start menu..Choose Control Panel**



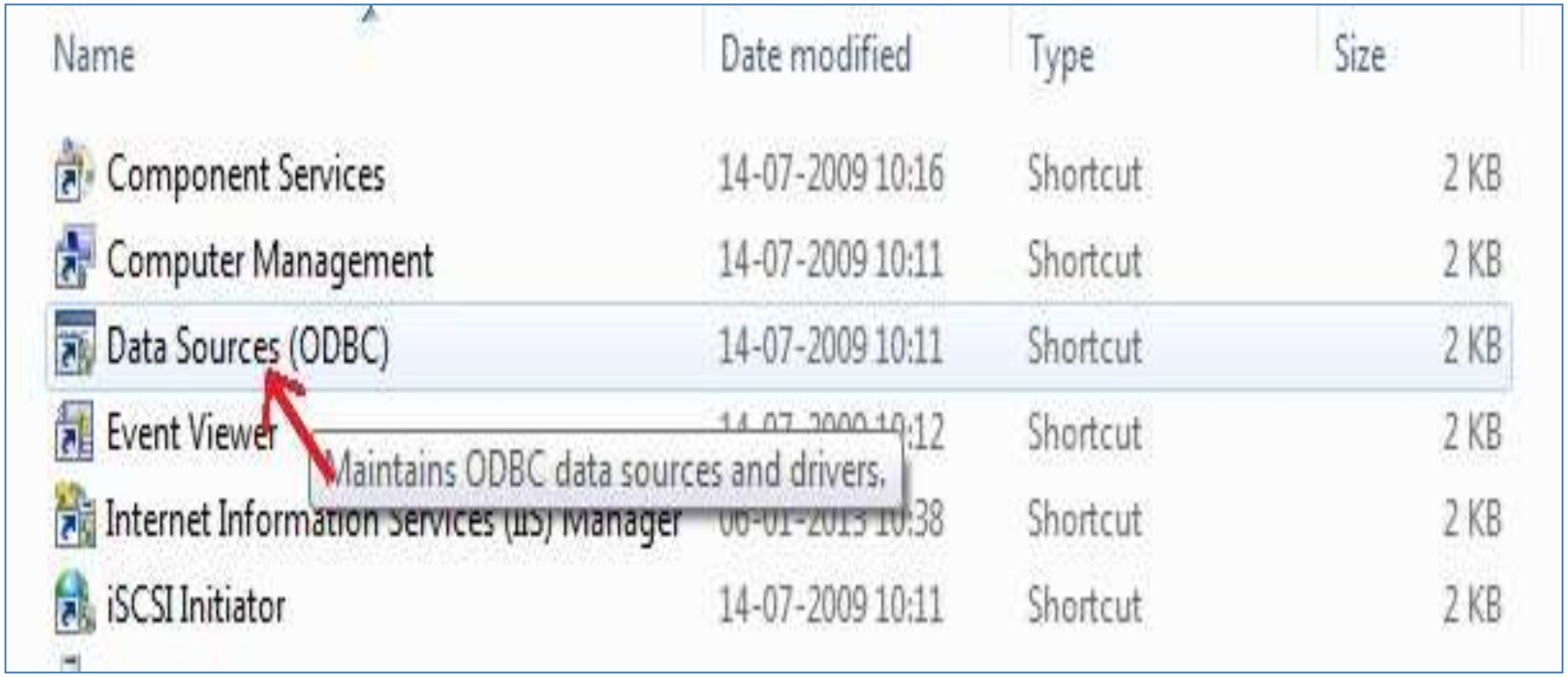
Contd…

Next choose Administrative tools from that control panel.



Contd…

After that choose Data Sources(ODBC option)

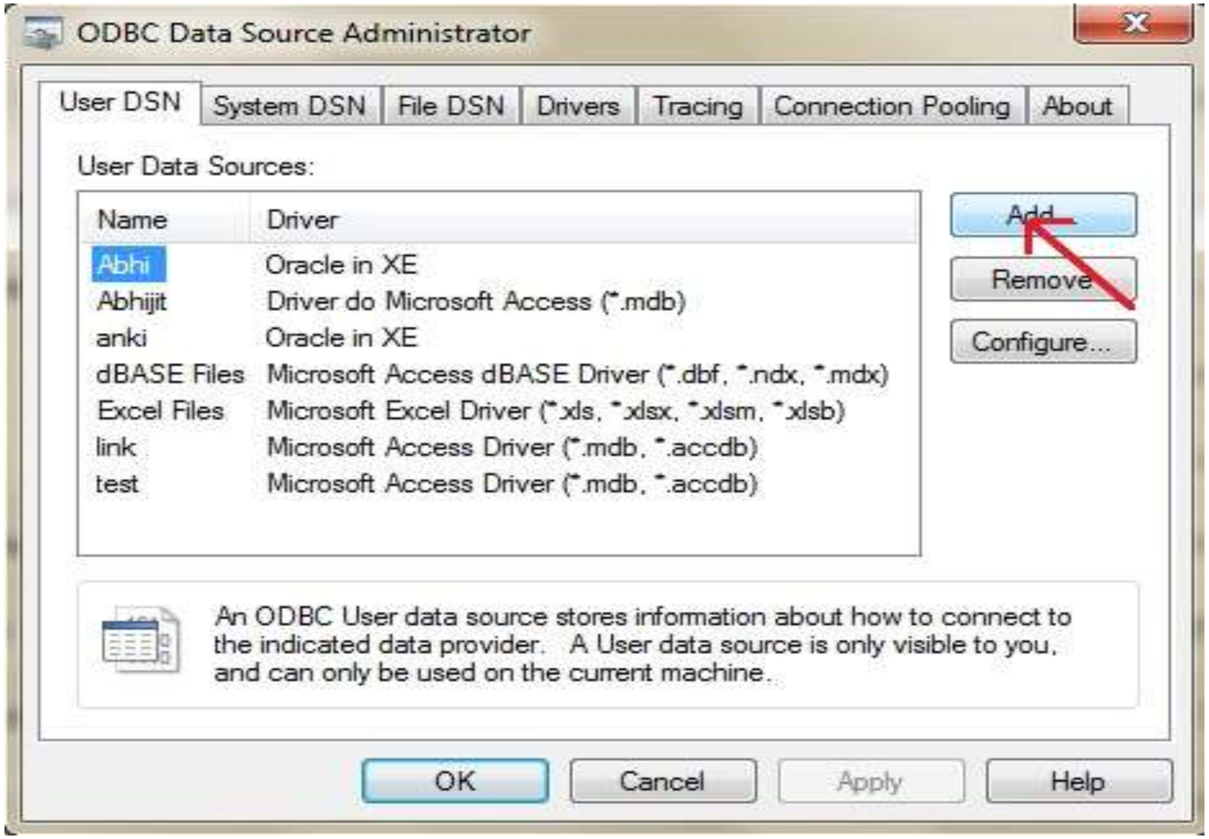


Name	Date modified	Type	Size
Component Services	14-07-2009 10:16	Shortcut	2 KB
Computer Management	14-07-2009 10:11	Shortcut	2 KB
Data Sources (ODBC)	14-07-2009 10:11	Shortcut	2 KB
Event Viewer	14-07-2009 10:12	Shortcut	2 KB
Internet Information Services (IIS) Manager	06-01-2013 10:38	Shortcut	2 KB
iSCSI Initiator	14-07-2009 10:11	Shortcut	2 KB

Maintains ODBC data sources and drivers.

Contd…

Next we can add the Data Source Name for the type of database which we want to connect



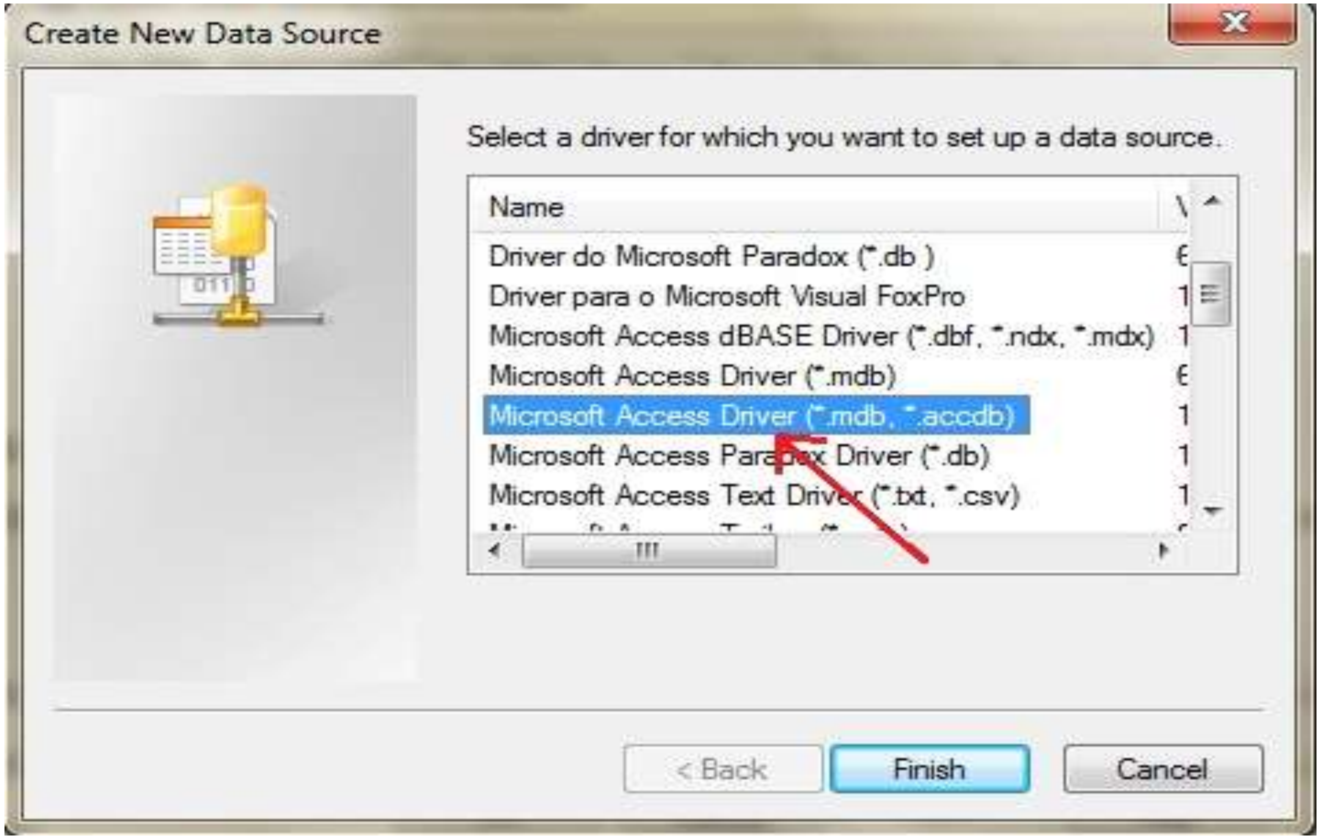
User Data Sources:

Name	Driver
Abhi	Oracle in XE
Abhijit	Driver do Microsoft Access (*.mdb)
anki	Oracle in XE
dBASE Files	Microsoft Access dBASE Driver (*.dbf, *.ndx, *.mdx)
Excel Files	Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.xlsb)
link	Microsoft Access Driver (*.mdb, *.accdb)
test	Microsoft Access Driver (*.mdb, *.accdb)

An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.

If we use 32-bit Operating system

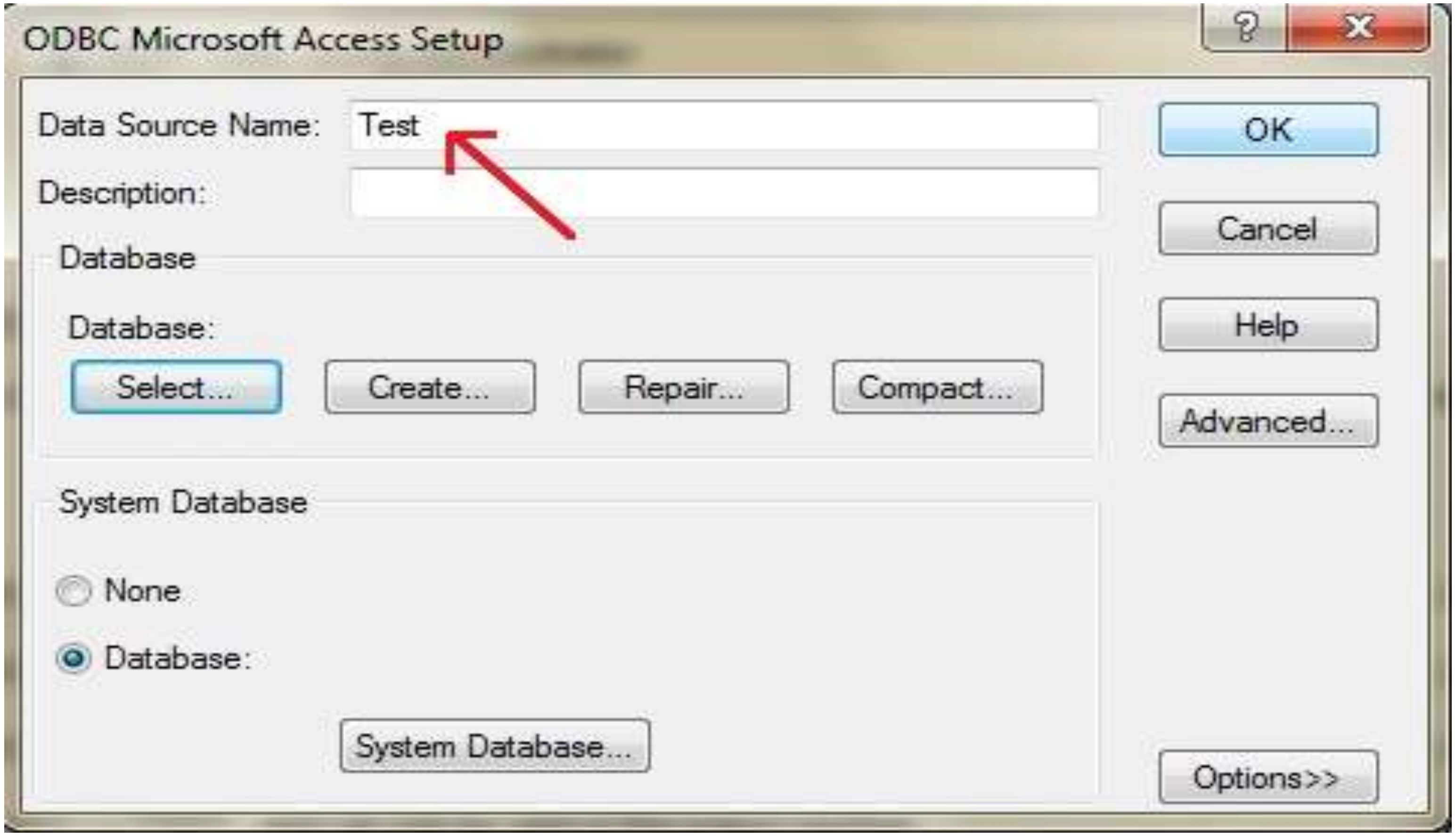
If we use 32 bit operating system in our system, then we can see the list of options in the ODBC:



Driver do Microsoft Paradox (*.db)
Driver para o Microsoft Visual FoxPro
Microsoft Access dBASE Driver (*.dbf, *.ndx, *.mdx)
Microsoft Access Driver (*.mdb)
Microsoft Access Driver (*.mdb, *.accdb)
Microsoft Access Paradox Driver (*.db)
Microsoft Access Text Driver (*.txt, *.csv)

We can choose DSN Name for that database

Here we choose **Test** as data source name.


ODBC Microsoft Access Setup

Data Source Name: Test

Description:

Database

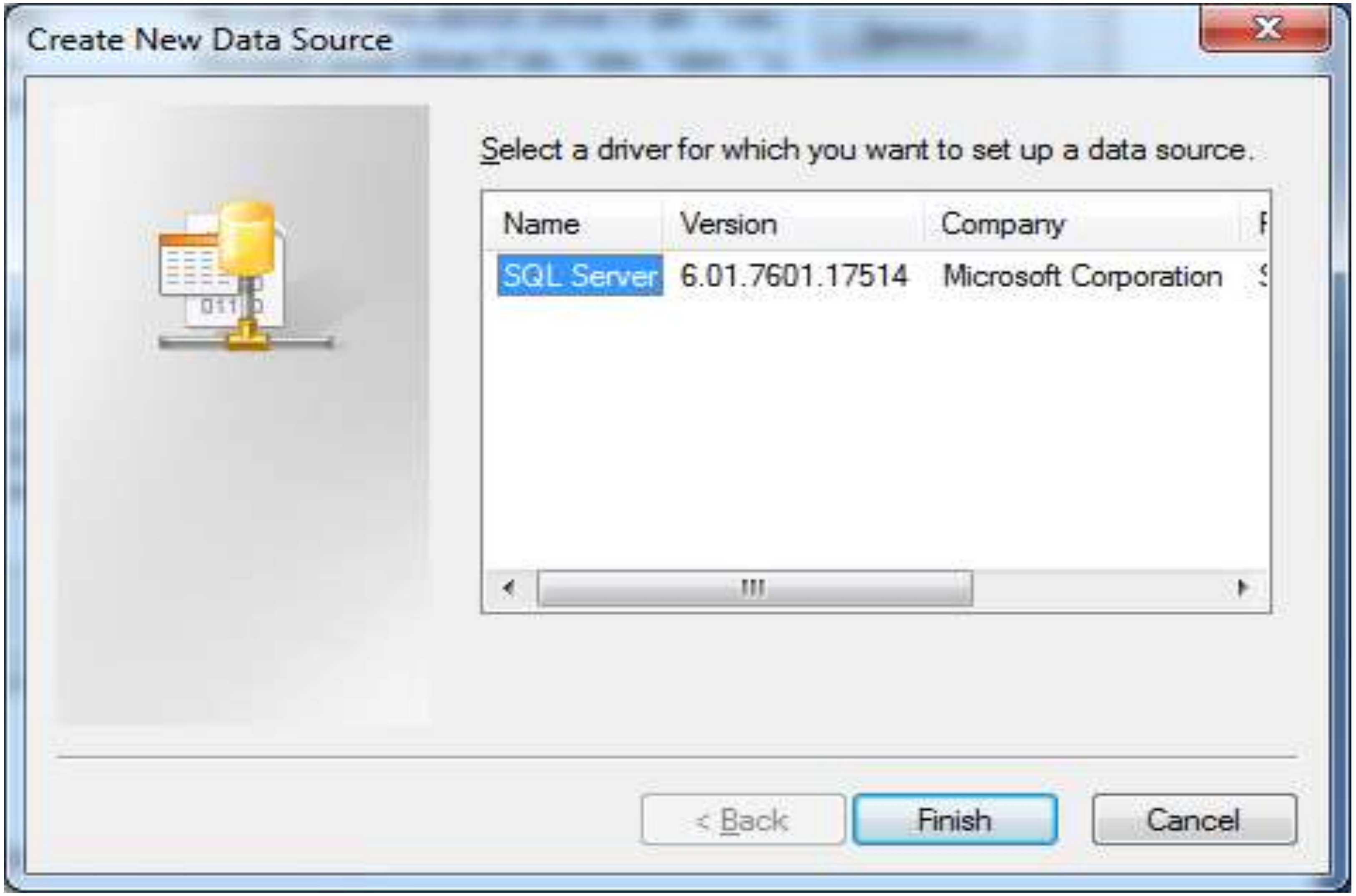
Database:

System Database

None

[✓ Database:]

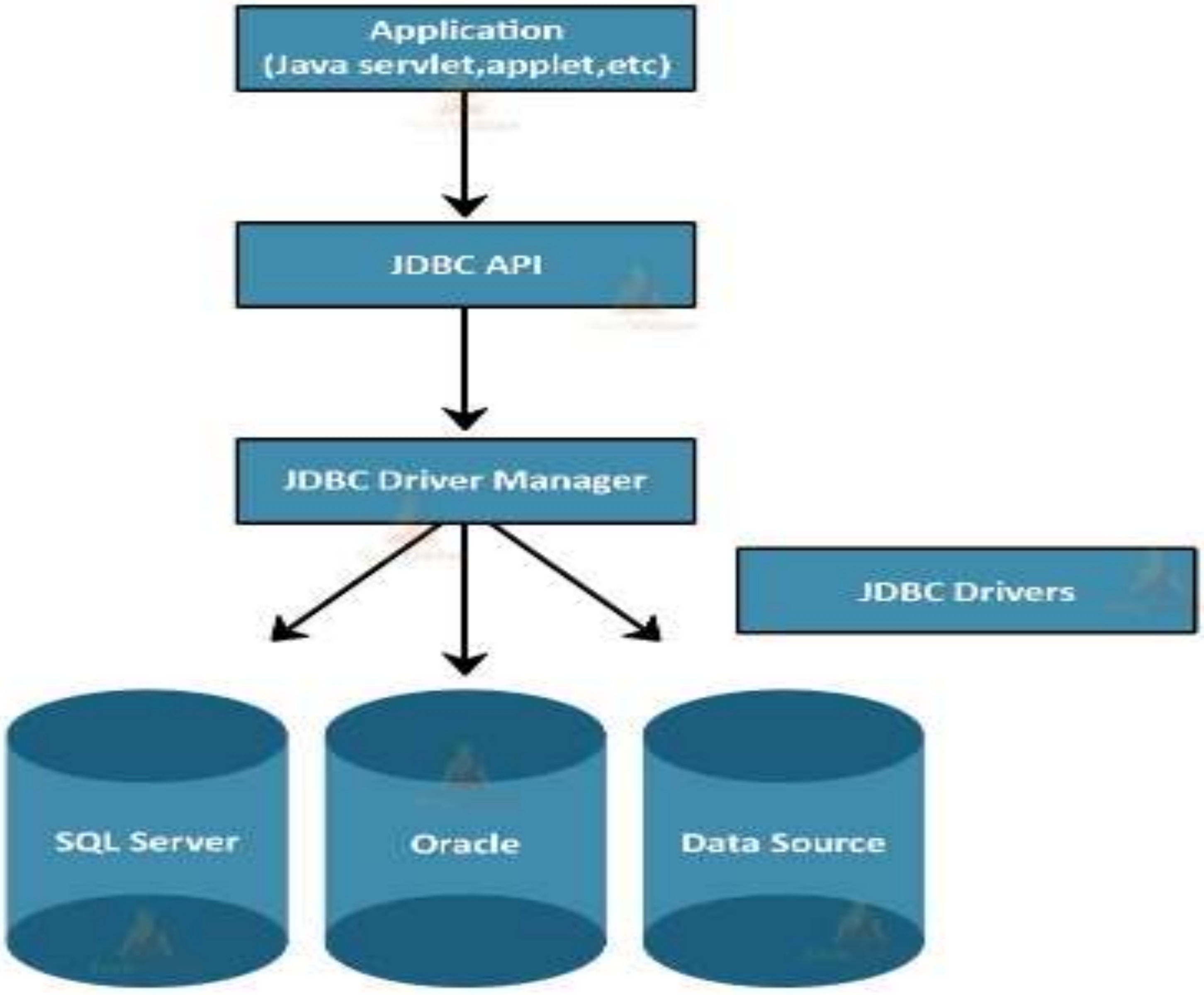
If we use 64 Bit OS,then we cannot able to see multiple databases, only SQL server can be seen



Select a driver for which you want to set up a data source.

Name	Version	Company
SQL Server	6.01.7601.17514	Microsoft Corporation

Architecture of JDBC



JDBC has four major components that are used for the interaction with the database.

1. JDBC API
2. JDBC Test Suite
3. JDBC Driver Manger
4. JDBC ODBC Bridge Driver

1) **JDBC API:** JDBC API provides various interfaces and methods to establish easy connection with different databases.

javax.sql.*;

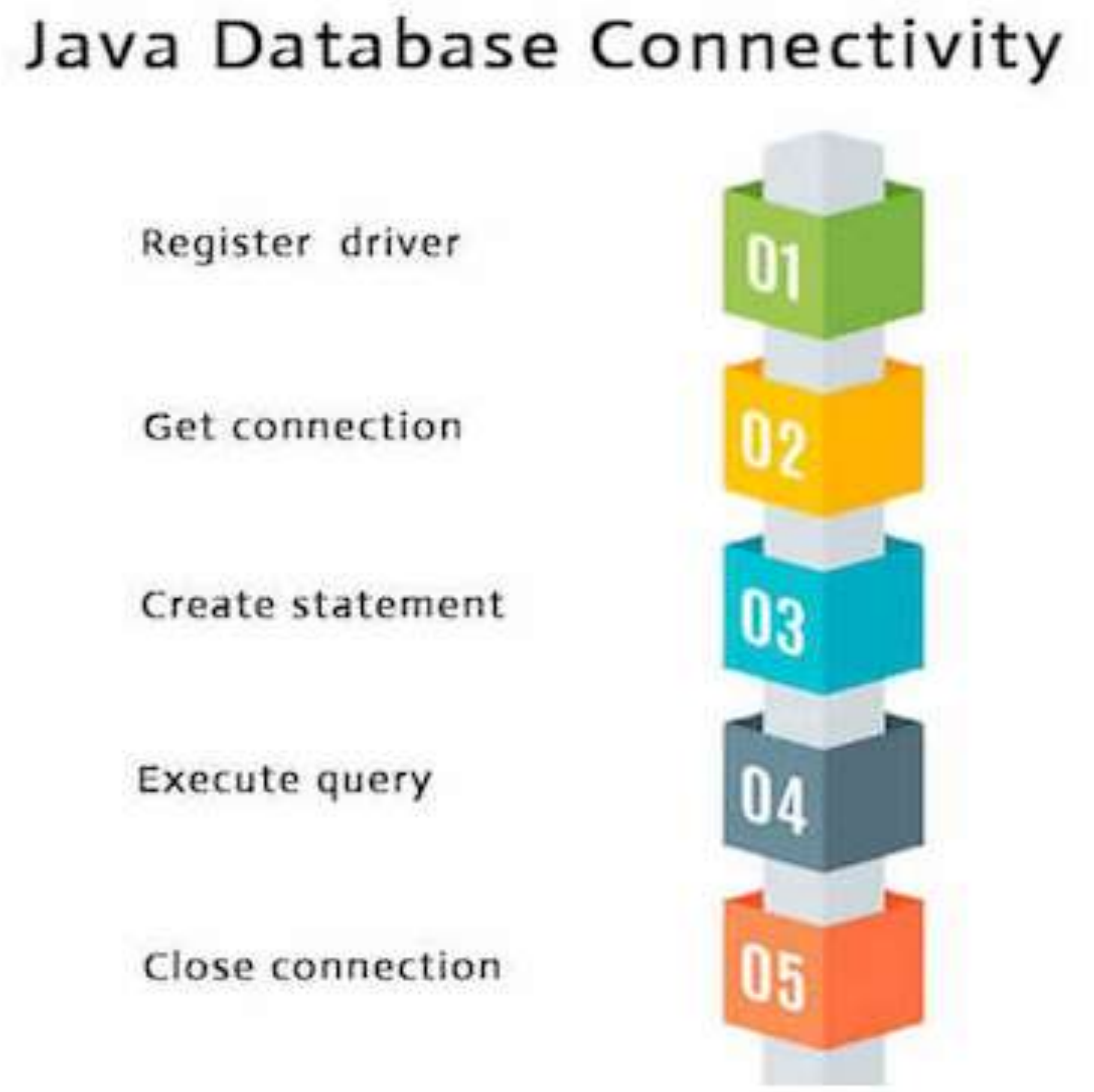
java.sql.*;

2) JDBC Test suite: JDBC Test suite facilitates the programmer to test the various operations such as deletion, updation, insertion that are being executed by the JDBC Drivers.

3) JDBC Driver manager: JDBC Driver manager loads the database-specific driver into an application in order to establish the connection with the database. The JDBC Driver manager is also used to make the database-specific call to the database in order to do the processing of a user request.

4) JDBC-ODBC Bridge Drivers: JDBC-ODBC Bridge Drivers are used to connect the database drivers to the database. The bridge does the translation of the JDBC method calls into the ODBC method call. It makes the usage of the **sun.jdbc.odbc** package that encompasses the native library in order to access the ODBC (Open Database Connectivity) characteristics.

Working with JDBC API



Components of JDBC API

In JDBC API,we can have several pre-defined classes ,interfaces and corresponding methods to execute the task. The following are the some of the classes, interfaces and methods which are present in the JDBC API.

In java, we can use JDBC with the help of following package

import java.sql.*;

Class/interface	**Usage**	**Methods**
DriverManager	This class manages the JDBC drivers. You need to register your drivers to this.	registerDriver(), getConnection() are the two methods used from this class.
Driver	This interface is the Base interface for every driver class i.e. If you want to create a JDBC Driver of your own you need to implement this interface.	**createInstance()**
Class	This is one predefined class in JDBC API which is used to connect the type of driver what we are using for the program.	The following is the method used to link driver: **forName("driver path");** **Syntax:** **Class.forName("driver path");**

Connection	This interface represents the connection with a specific database. SQL statements are executed in the context of a connection.	This interface provides methods such as close(), commit(), rollback(), createStatement(), prepareCall(), prepareStatement(), setAutoCommit() setSavepoint()
Statement	This interface represents a **static SQL statement**. Using the Statement object and its methods, you can execute an SQL statement and get the results of it.	It provides methods such as execute(), executeBatch(), executeUpdate()

PreparedStatement	precompiled SQL statement. An SQL statement is compiled and stored in a prepared statement and you can later execute this multiple times.	executeQuery(), executeUpdate(), and execute() to execute the prepared statements and getXXX(), setXXX() (where XXX is the datatypes such as long int float etc..) methods to set and get the values of the bind variables of the prepared statement.

ResultSet	This interface represents the database result set, a table which is generated by executing statements	There are several methods used in this interface to retrieve the data.They are as follows:
		
		getInt(column index)
		
		getInt(column name)
		
		getString(column name)
		
		getString(column index)

Types of Drivers

JDBC driver implementations vary because of the wide variety of operating systems and hardware platforms in which Java operates. Sun has divided the implementation types into four categories, Types 1, 2, 3, and 4 :

1) Type 1 – JDBC-ODBC Bridge Driver
2) Type 2 – JDBC-Native API
3) Type 3 – JDBC-Net pure Java
4) Type 4 – 100% Pure Java

Type 1-JDBC-ODBC Bridge Driver

1) In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine.

2) Using ODBC, requires configuring on your system a Data Source Name (DSN) that represents the target database.

3) When Java first came out, this was a useful driver because most databases only supported ODBC access but now this type of driver is recommended only for experimental use or when no other alternative is available.

Type 1 Architecture

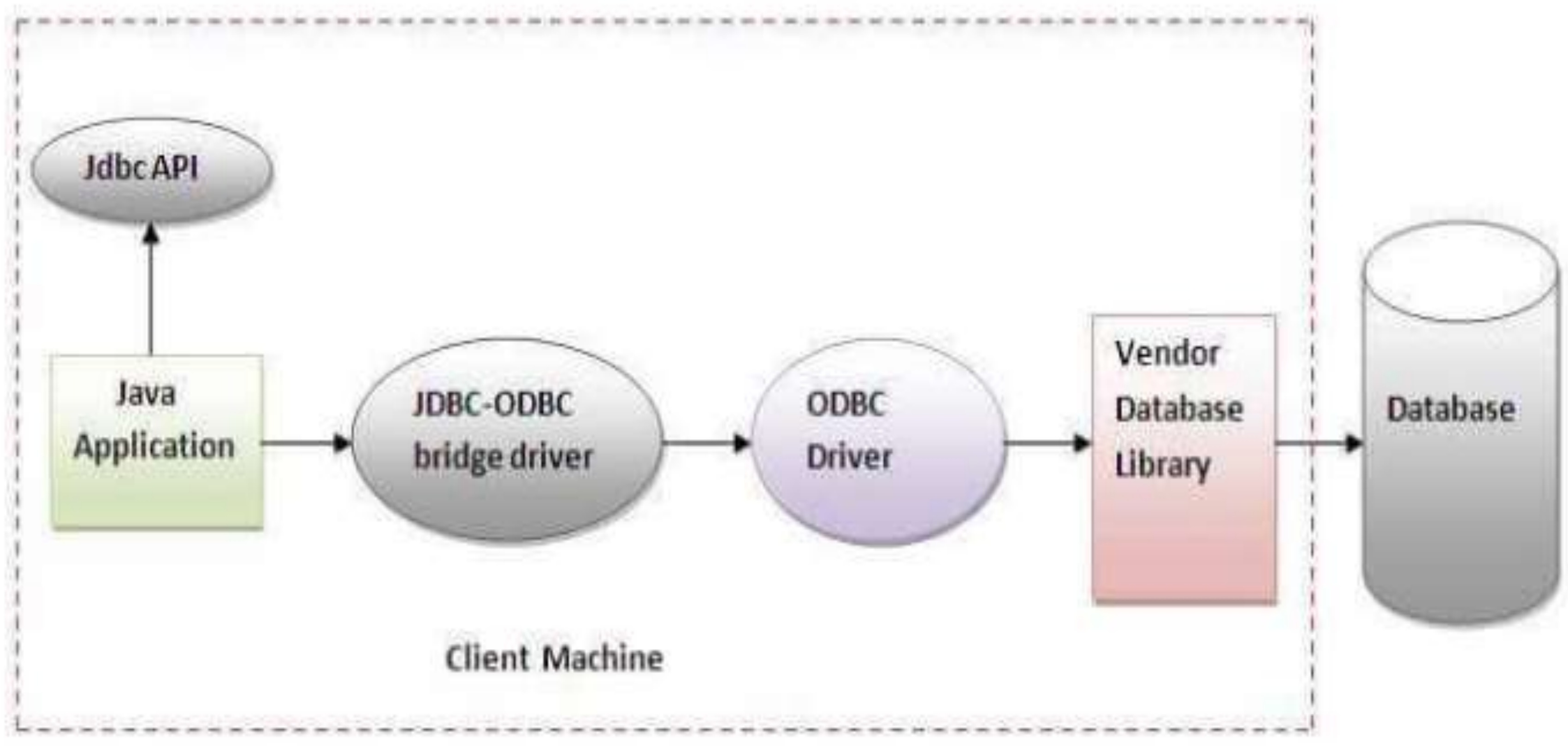
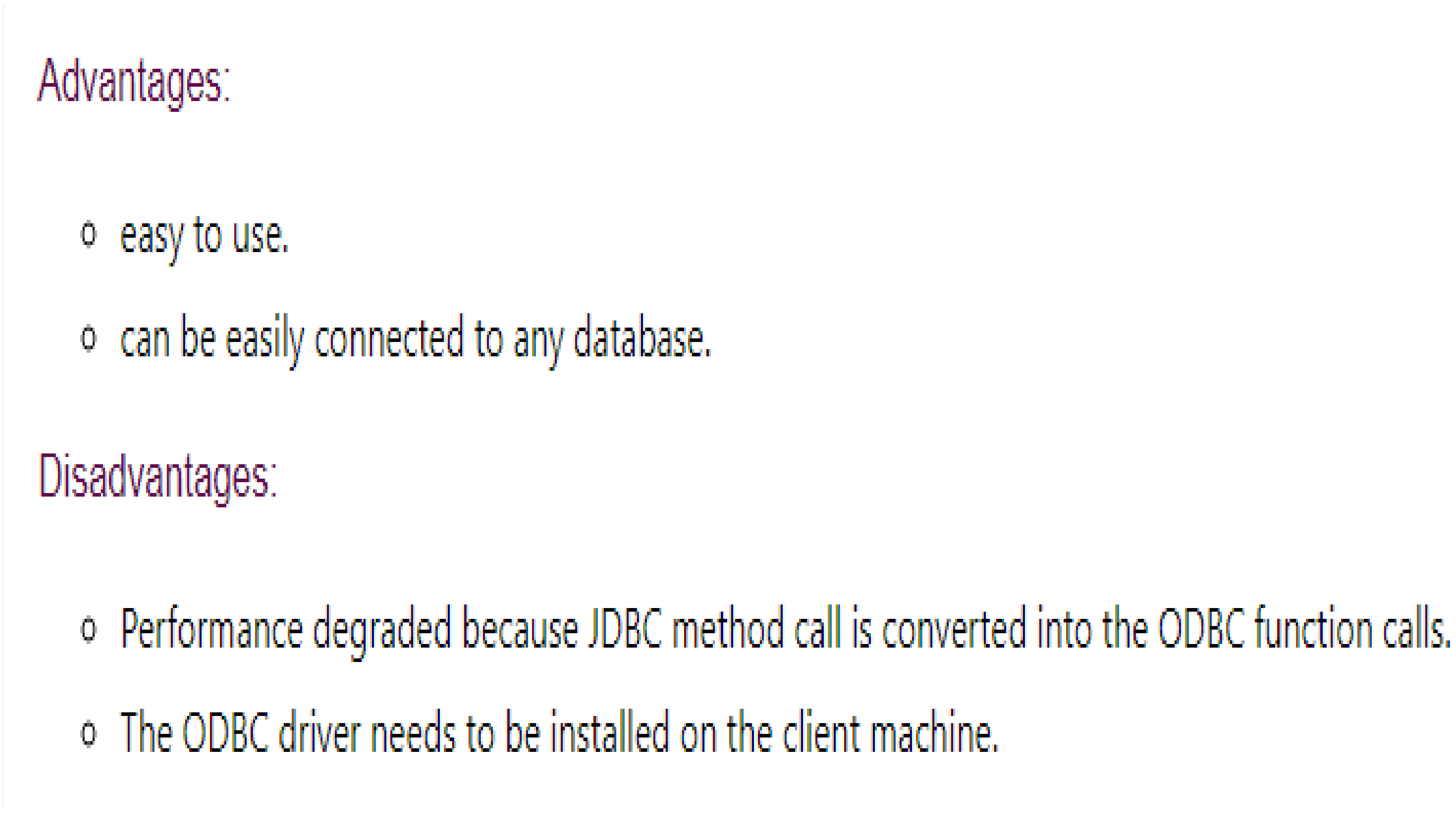


Figure- JDBC-ODBC Bridge Driver

Pros and Cons for Type 1 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.

Type-2 : Native-API driver

- The Native API driver is also known as **Type 2 driver.**
- Type 2 driver uses the native code part instead of ODBC parts. It uses the client-side libraries of the database.
- It is vender specific driver, so must be installed on each client system. This driver converts the JDBC method call into native call of database.
- The **Oracle Call Interface (OCI)** driver is an example of a Native API Driver.

Type -2 Architecture

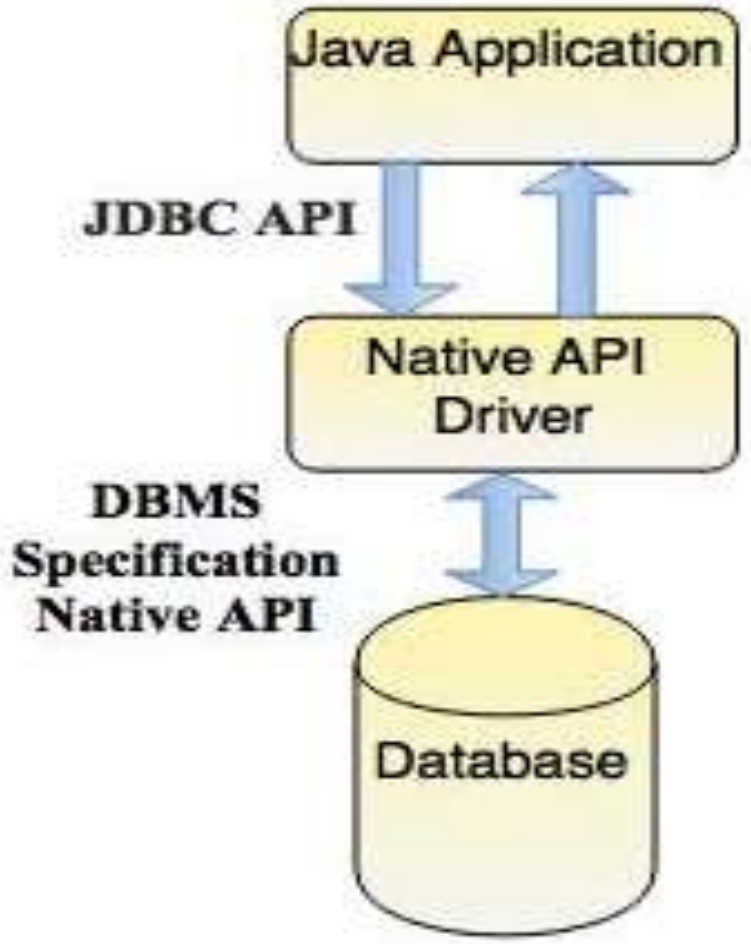
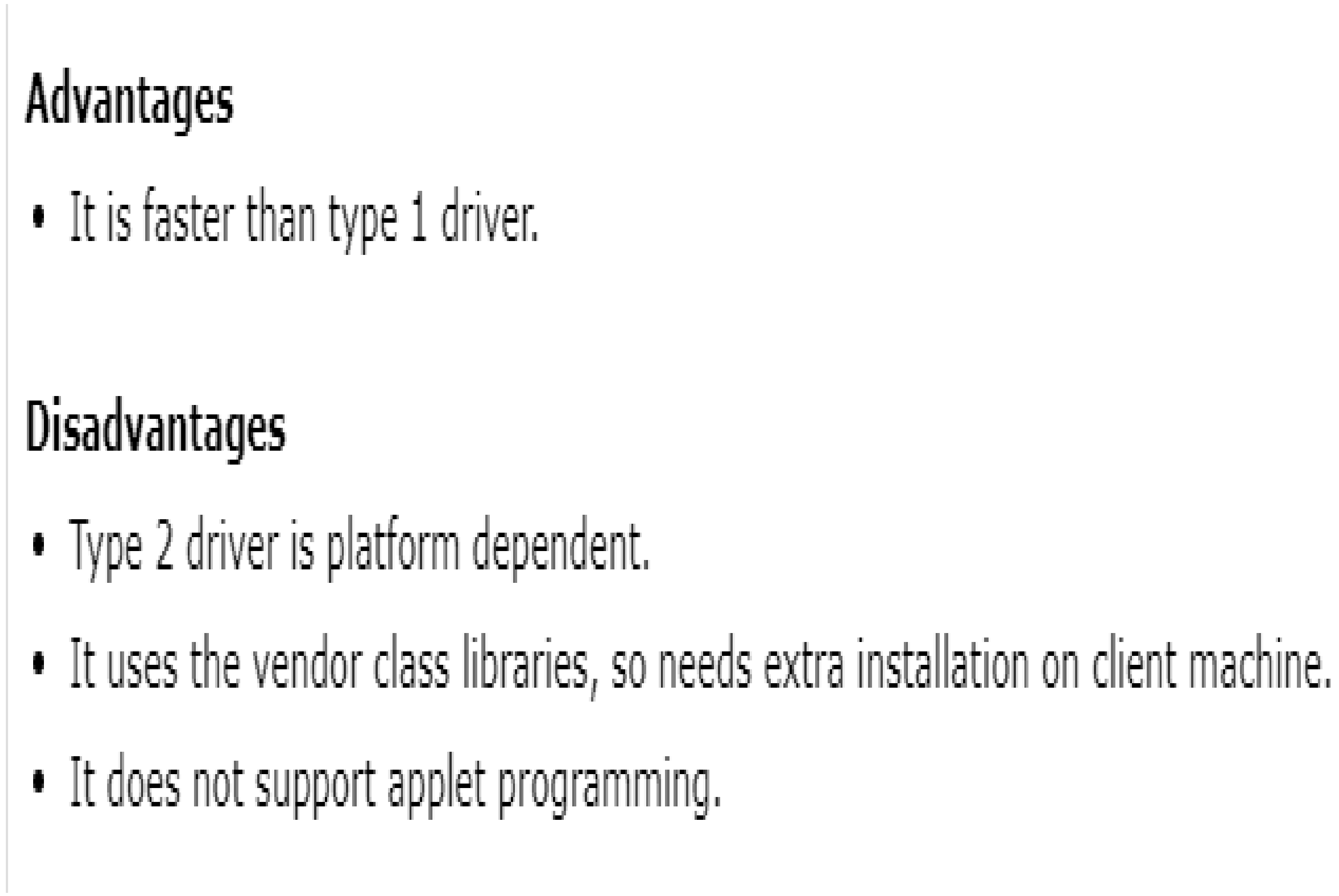


Figure: Native API Driver

Pros and Cons of Type -2



Advantages

- It is faster than type 1 driver.

Disadvantages

- Type 2 driver is platform dependent.
- It uses the vendor class libraries, so needs extra installation on client machine.
- It does not support applet programming.

Type-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.

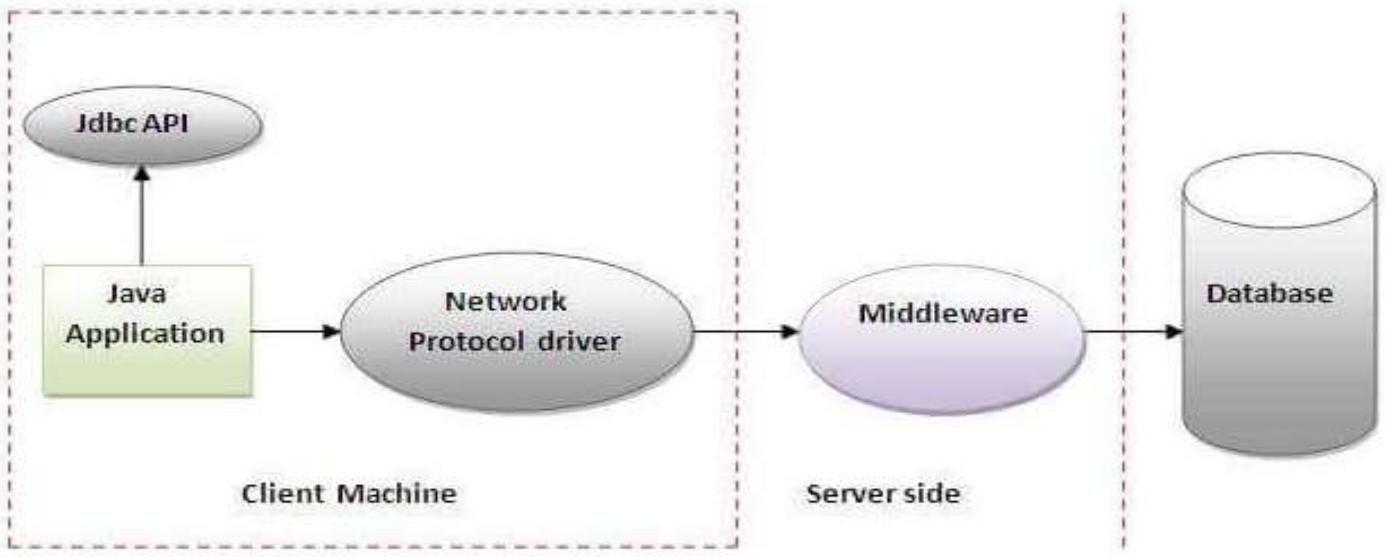
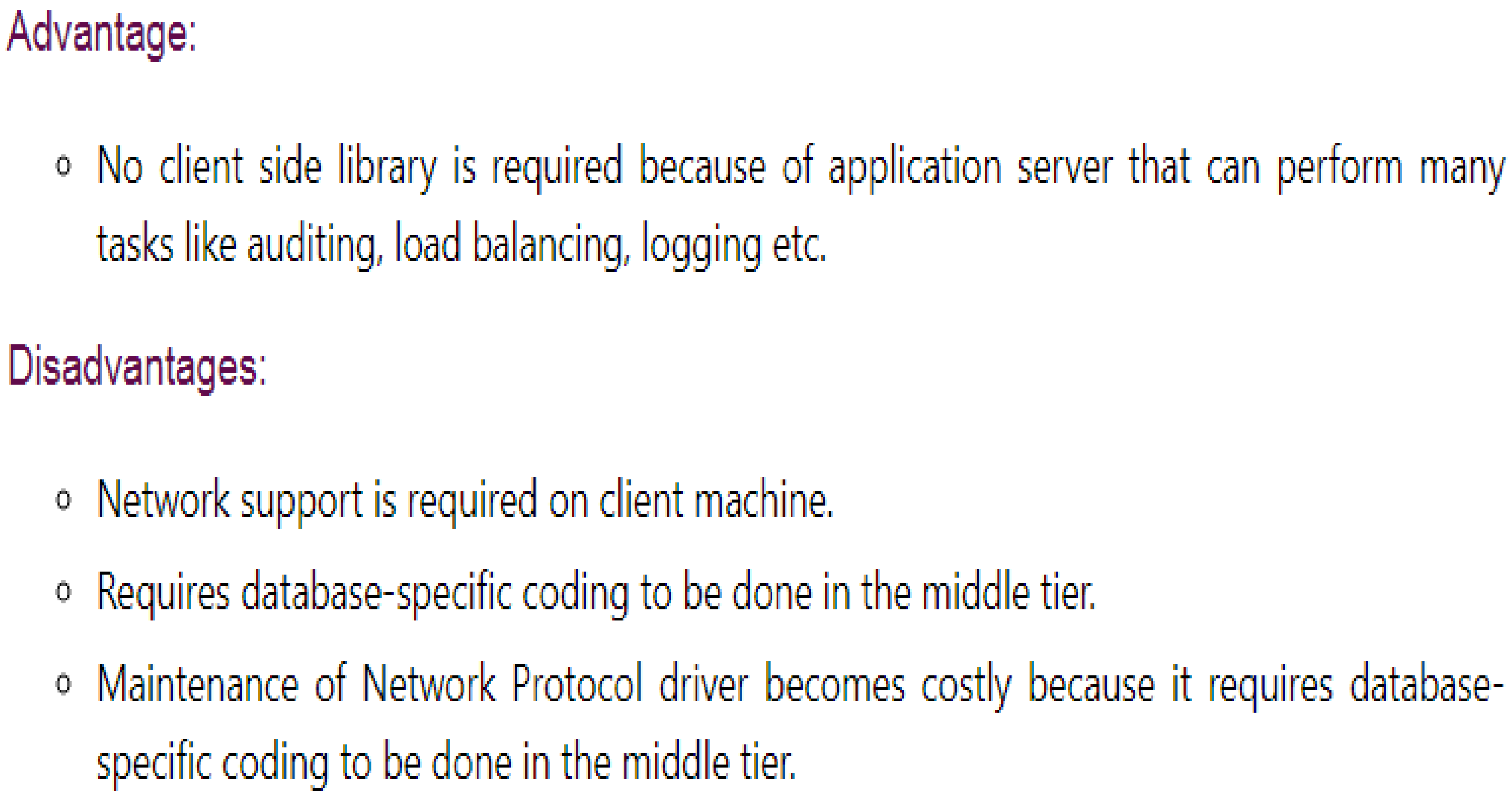


Figure- Network Protocol Driver

Pros and Cons



Advantage:

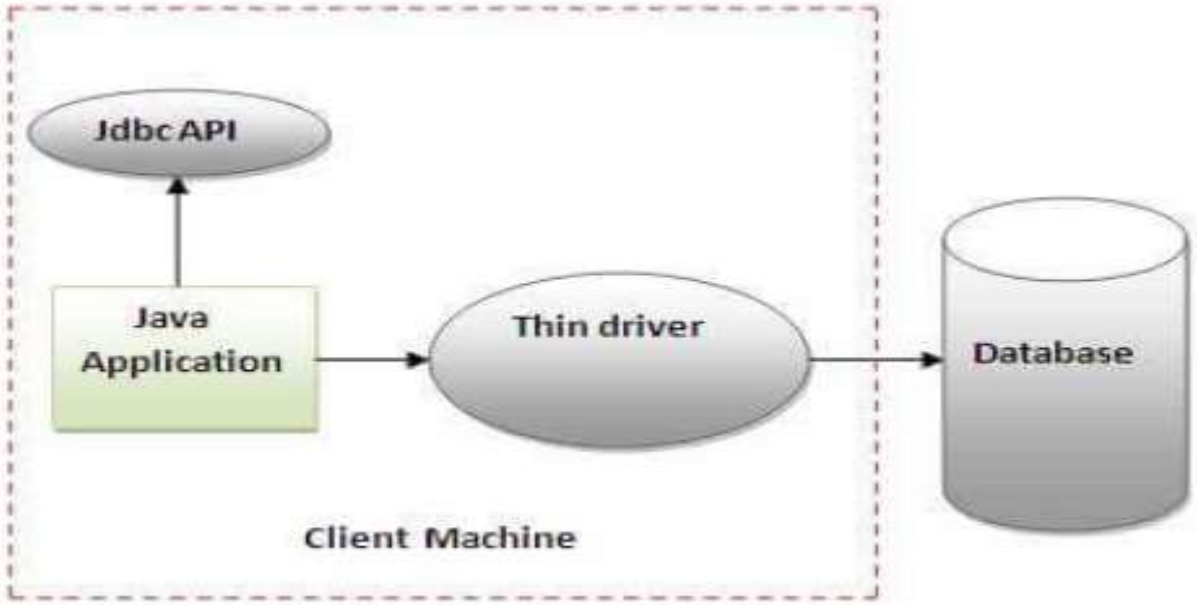
- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

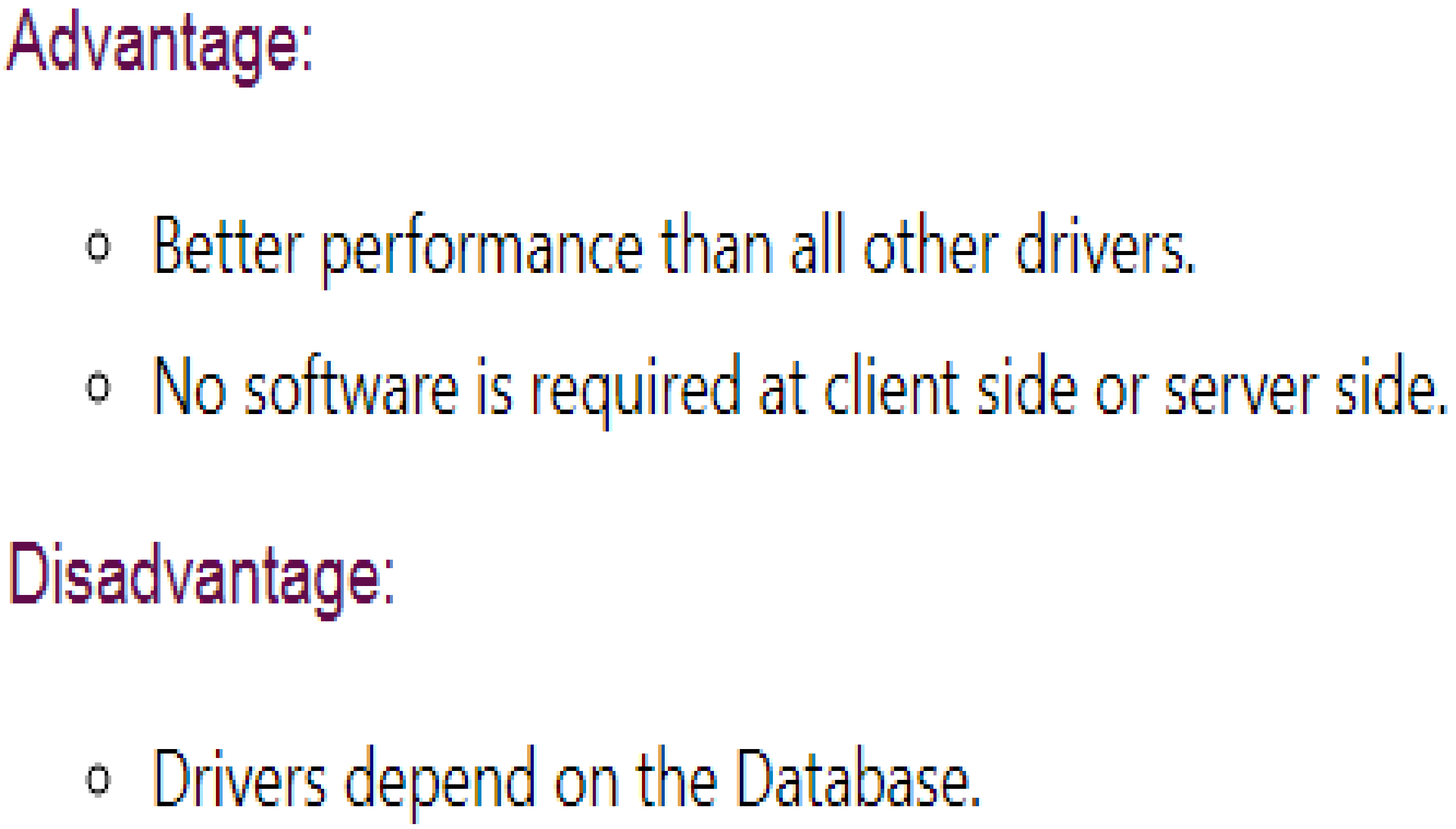
- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

Type 4 Driver: 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.



Pros and Cons



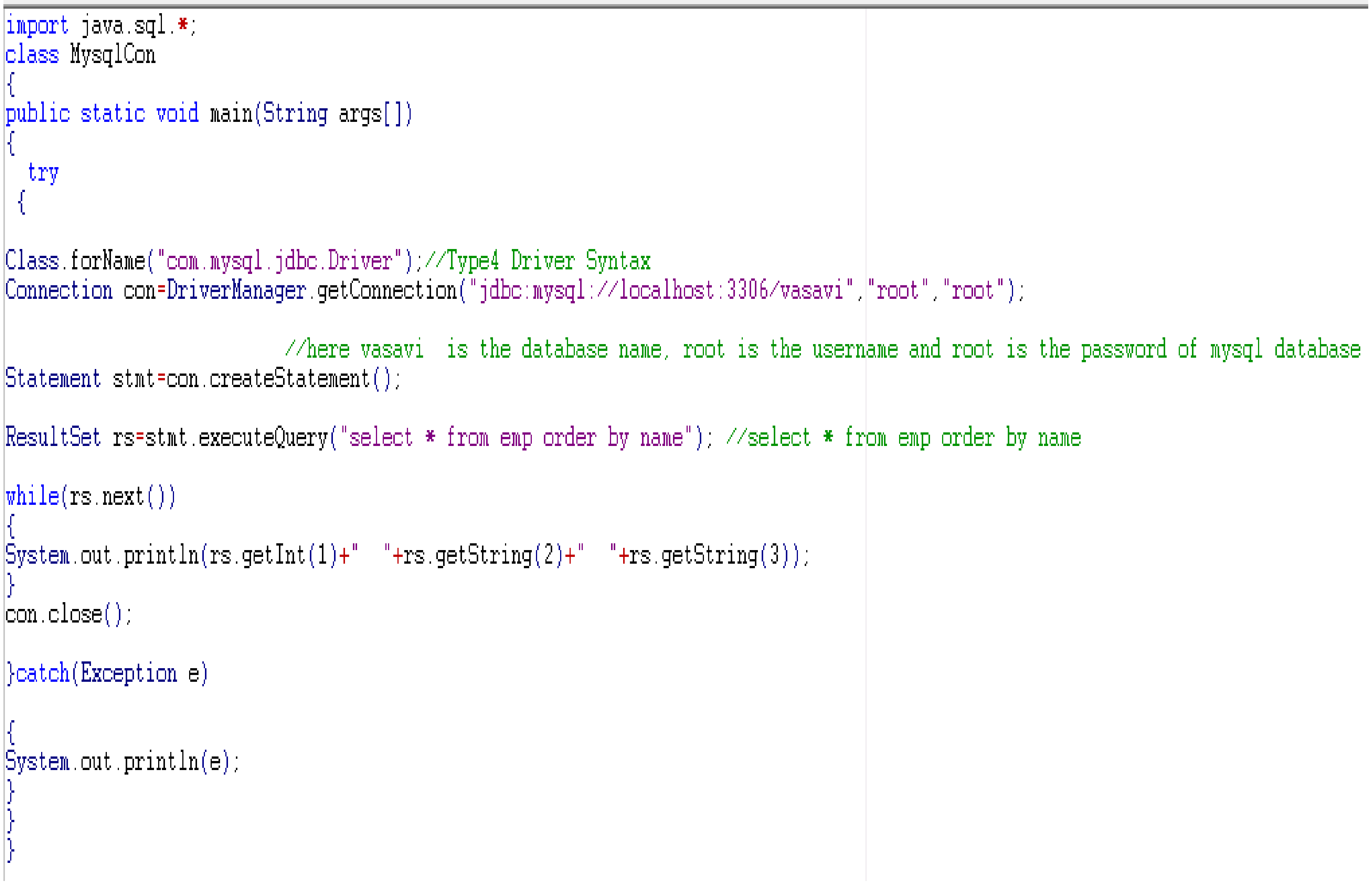
Advantage:

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

Disadvantage:

- Drivers depend on the Database.

MYSQL connectivity using Type 4 Driver for Data Retrieval

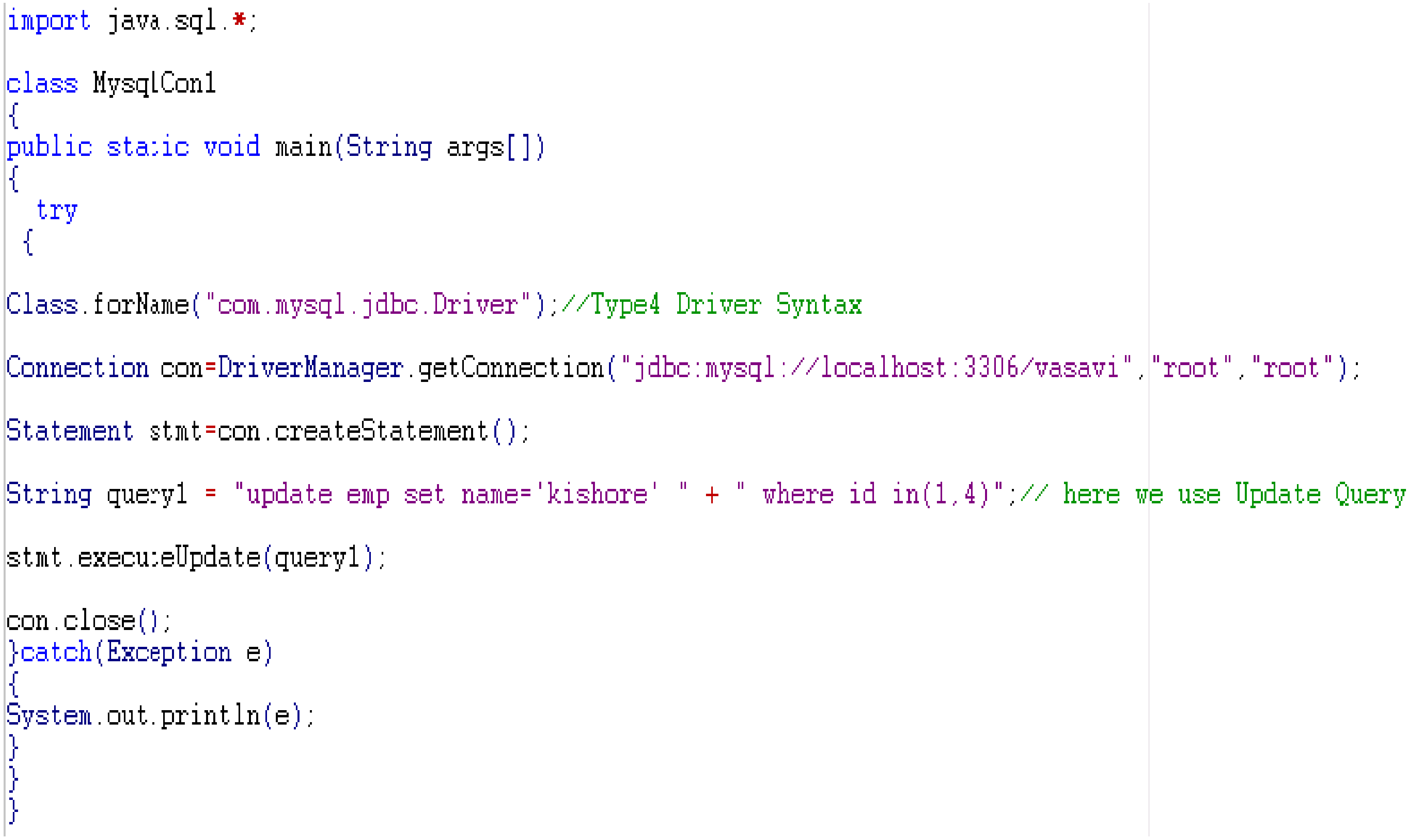


```java
import java.sql.*;
class MysqlCon
{
public static void main(String args[])
{
  try
 {

Class.forName("com.mysql.jdbc.Driver");//Type4 Driver Syntax
Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/vasavi","root","root");

                          //here vasavi  is the database name, root is the username and root is the password of mysql database
Statement stmt=con.createStatement();

ResultSet rs=stmt.executeQuery("select * from emp order by name"); //select * from emp order by name

while(rs.next())
{
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
}
con.close();

}catch(Exception e)

{
System.out.println(e);
}
}
}
```

Before executing the program,just try to create database with following tables

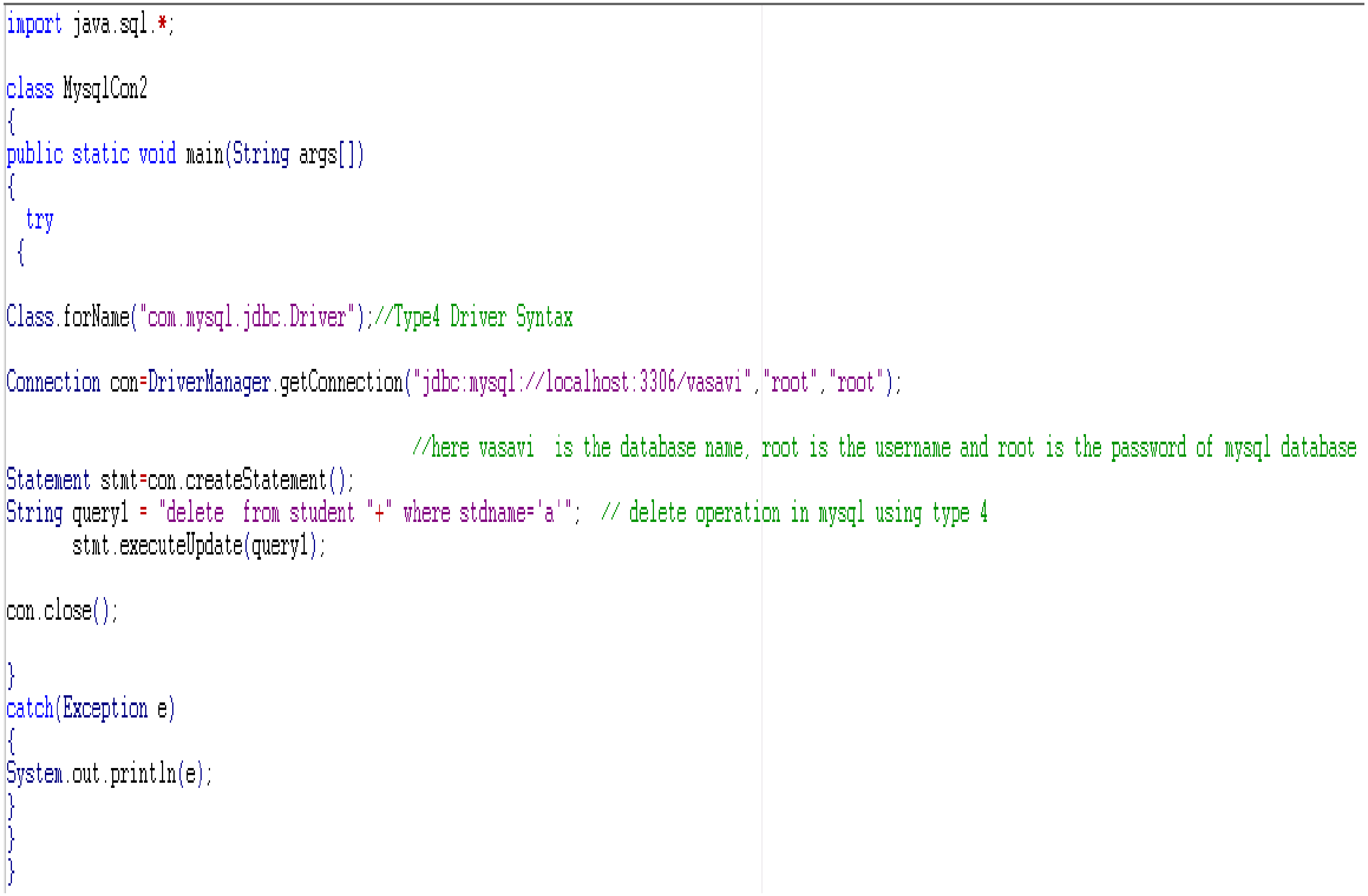
```
create database vasavi;
use vasavi;
create table emp(id int(10),name varchar(40),age int(3));
insert into emp values(1,'praveen',31);
insert into emp values(2,'Kumar',30);
insert into emp values(3,'Sowmya',29);
```

MYSQL connectivity using Type 4 Driver for Data Update



```
import java.sql.*;

class MysqlCon1
{
public static void main(String args[])
{
  try
 {

Class.forName("com.mysql.jdbc.Driver");//Type4 Driver Syntax

Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/vasavi","root","root");

Statement stmt=con.createStatement();

String query1 = "update emp set name='kishore' " + " where id in(1,4)";// here we use Update Query

stmt.executeUpdate(query1);

con.close();
}catch(Exception e)
{
System.out.println(e);
}
}
}
```

MYSQL connectivity using Type 4 Driver for Data Deletion



```
import java.sql.*;

class MysqlCon2
{
public static void main(String args[])
{
  try
 {

Class.forName("com.mysql.jdbc.Driver");//Type4 Driver Syntax

Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/vasavi","root","root");

                                          //here vasavi  is the database name, root is the username and root is the password of mysql database
Statement stmt=con.createStatement();
String query1 = "delete  from student "+" where stdname='a'";  // delete operation in mysql using type 4
       stmt.executeUpdate(query1);

con.close();

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

Database Tables for Delete Operation Program

- CREATE DATABASE "vasavi";
- USE "vasavi";
- CREATE TABLE /*!32312 IF NOT EXISTS*/ "student" ("stdno" int(50) , "stdname" varchar(50) , "stdmarks" int(10));
- Try to insert some student details in the above table.s

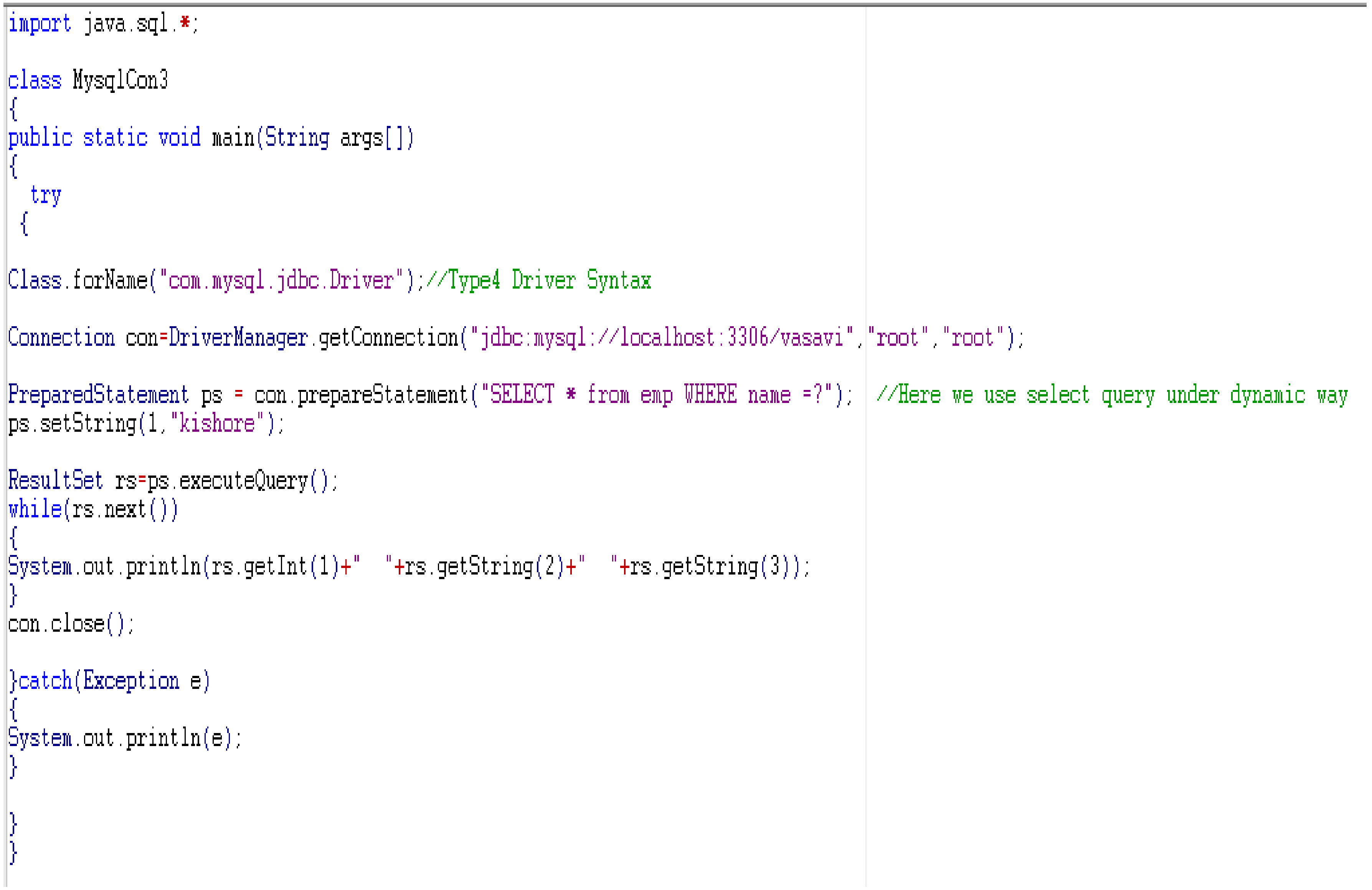
PreparedStatement

A **PreparedStatement** is a pre-compiled SQL statement. It is a **subinterface** of **Statement**. Prepared Statement objects have some useful additional features than Statement objects. Instead of hard coding queries, PreparedStatement object provides a feature to execute a parameterized query.

When PreparedStatement is created, the SQL query is passed as a parameter. This Prepared Statement contains a pre-compiled SQL query, so when the PreparedStatement is executed, DBMS can just run the query instead of first compiling it.

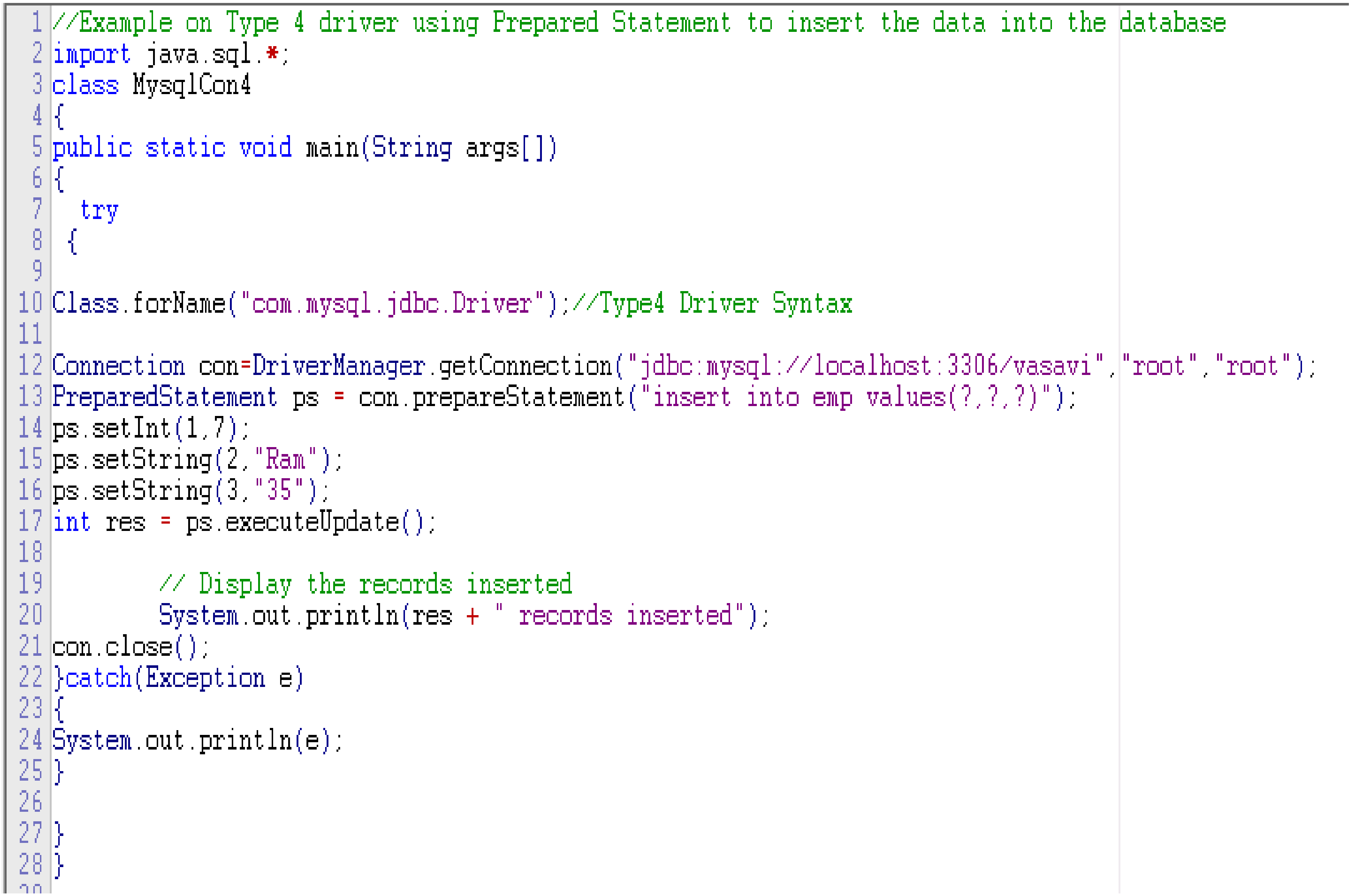
Method : prepareStatement()

Prepared Statement for Retrieving Query in Dynamic Way



```
import java.sql.*;

class MysqlCon3
{
public static void main(String args[])
{
  try
 {

Class.forName("com.mysql.jdbc.Driver");//Type4 Driver Syntax

Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/vasavi","root","root");

PreparedStatement ps = con.prepareStatement("SELECT * from emp WHERE name =?");  //Here we use select query under dynamic way
ps.setString(1,"kishore");

ResultSet rs=ps.executeQuery();
while(rs.next())
{
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
}
con.close();

}catch(Exception e)
{
System.out.println(e);
}

}
}
```

Prepared Statement for insert the records in Dynamic Way



```java
//Example on Type 4 driver using Prepared Statement to insert the data into the database
import java.sql.*;
class MysqlCon4
{
public static void main(String args[])
{
  try
 {

Class.forName("com.mysql.jdbc.Driver");//Type4 Driver Syntax

Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/vasavi","root","root");
PreparedStatement ps = con.prepareStatement("insert into emp values(?,?,?)");
ps.setInt(1,7);
ps.setString(2,"Ram");
ps.setString(3,"35");
int res = ps.executeUpdate();

        // Display the records inserted
        System.out.println(res + " records inserted");
con.close();
}catch(Exception e)
{
System.out.println(e);
}

}
}
```

THANK YOU

Web Technologies

Unit 4: Contents

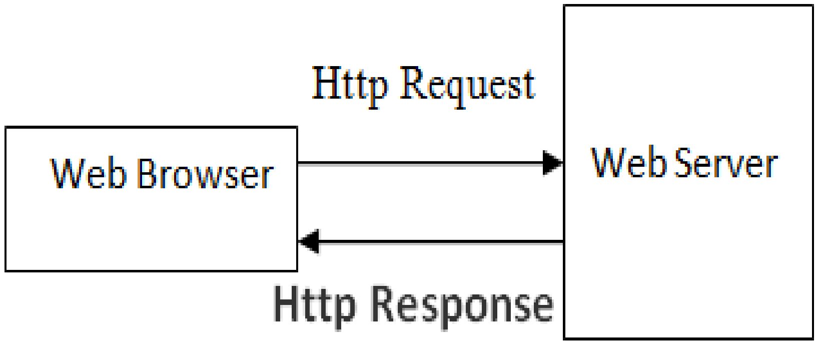
UNIT IV: Introduction to Servlets & JSP:

Introduction to servlets, Life cycle of Servlet, Limitations of servlets, Java Server Pages: JSP Overview, Components of a JSP Page: Directives, comments, Expressions, Scriplets , Declarations, implicit objects, Database Access, session tracking.

What is Web Application ?

Web Application is an application that runs in Web. A Java web application is a collection of dynamic resources (such as Servlet, Java Server Pages, Java classes and jars) and static resources (HTML pages and pictures). A Java web application can be deployed as a ".war" file. The "**.war**" file is a zip file which contains the complete content of the corresponding web application. A **Servlet** is a Java class which extends "**HttpServlet**" and answers a HTTP request within a web container.

Package: **javax.Servlet.*;**

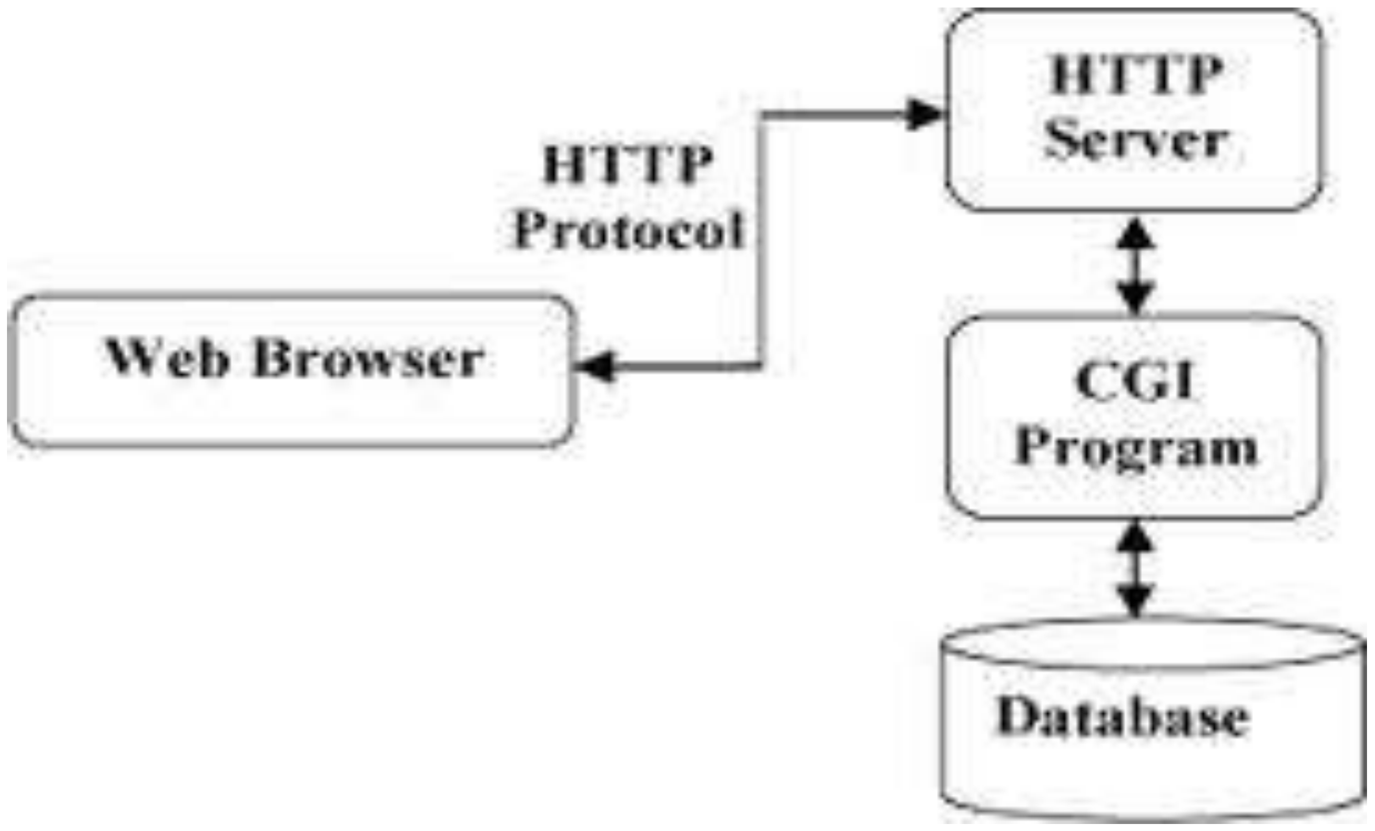


What is Web Client/Browser ?

- Web client makes a request to web server using HTTP protocol.
- Web client receives HTML sent from server and displays it to end user.
- Internet Explorer, Mozilla Firefox and Netscape Navigator are widely used web browsers.
- Browser is also responsible for running client-side scripting written in JavaScript.

HTTP PROTOCOL

- HTTP is an application protocol implemented on TCP/IP.
- It is request and response protocol.
- Client sends a request to receive information from server or invoke a process on the server.



CGI Means: Common Gateway Interface

Java Servlets

1. Servlet are **server side components** that provide a powerful mechanism for developing server side programs.
2. Servlet provide **component-based, platform-independent methods** for building Web-based applications, without the performance limitations of CGI programs.
3. Using Servlet **web developers can create fast and efficient server side application** which can run on any Servlet enabled web server.
4. Servlet run entirely inside the **Java Virtual Machine**.

5. **Servlet can access the entire family of Java APIs, including the JDBC API to access enterprise databases.**

6. Servlet can also access a library of HTTP-specific calls.

7. Servlets are **platform independent** and can be accessed by any one easily.

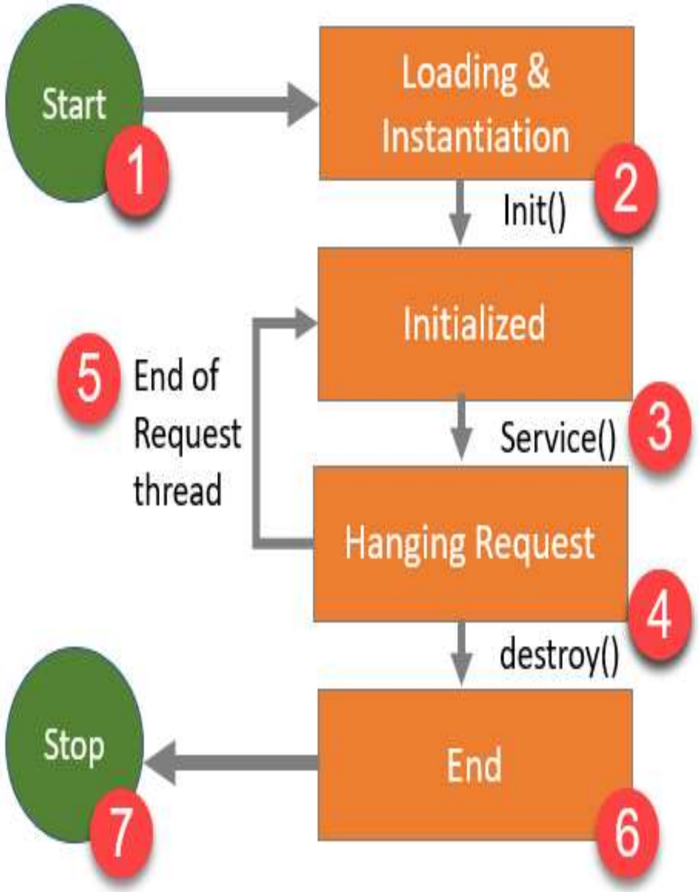
7. Servlets are one form of CGI program,which is not designed for a specific protocol. They are universal and can be accessed by any one.

8. Servlet uses the classes in the java packages javax.servlet and javax.servlet.http.

Life Cycle of Servlet

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**.



Servlet Life Cycle

Flow

1) **Start:** Execution of servlet begins.
2) **Loading & instantiation void init():** It is called when servlet is first loaded. This method lets you initialize servlet.
3) **Initialized void service()**: The purpose of this method is to serve a request. You can call it as many times as you like.
4) **Handling request and destroying servlet:** Java application must be first determined what code is needed to execute the request URL to provide a response. To destroy servlet Void destroy method is used at the end of servlet life cycle.

5) **End of Request Thread:** When service() finishes its task, either the thread ends or returns to the thread pool that is managed by servlet container.

6) **End:** Servlet lifecycle finishes.

7) **Stop:** Servlet stop executing.

Limitations of servlets

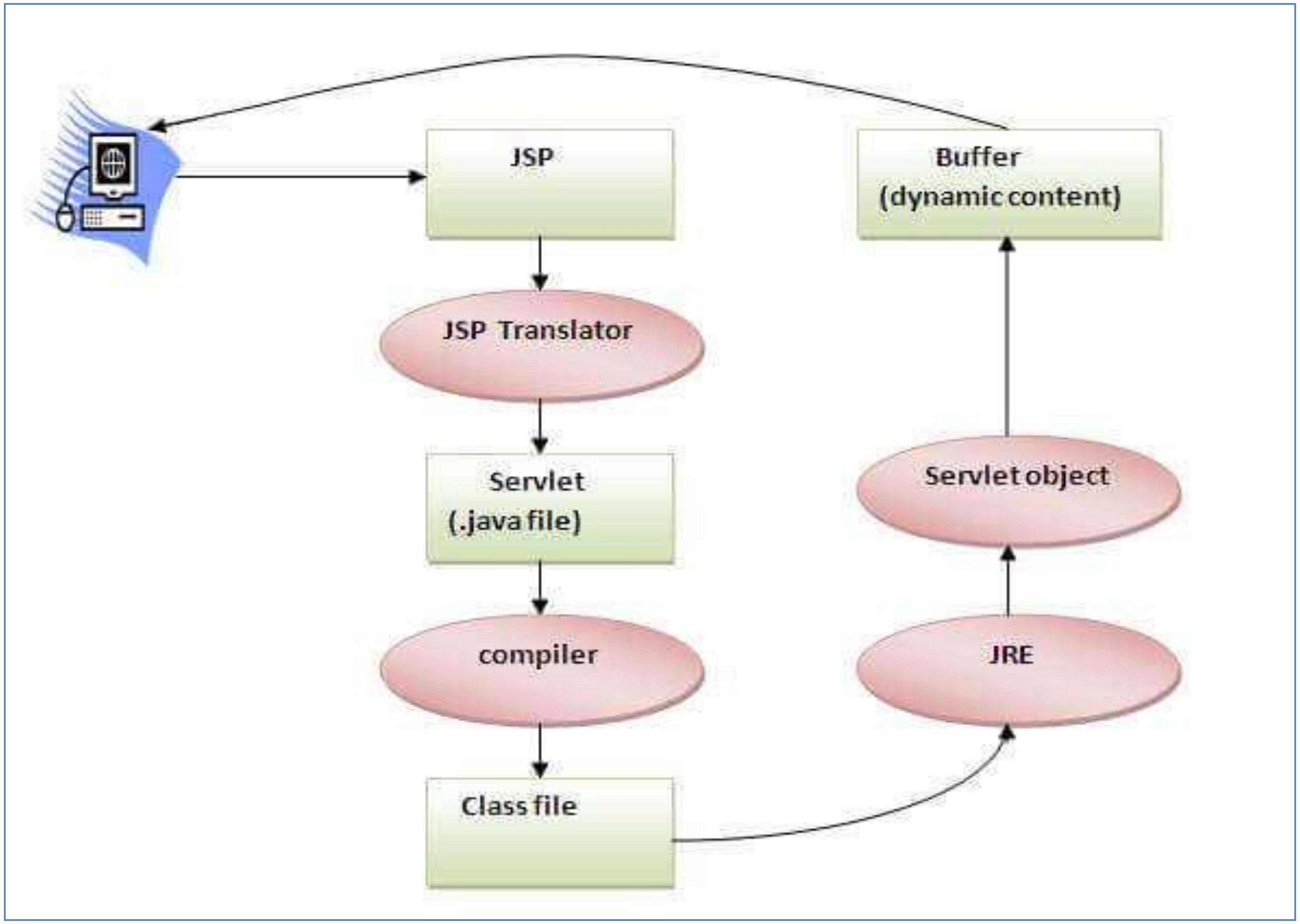
Here are the disadvantages for using Servlet:

1. One servlet is loaded into JVM. It does matter numbers of requests.
2. When there is a request, there is a thread, not a process.
3. Servlet is persistent until it destroys.
4. Designing in a servlet is difficult and slows down the application.
5. You need a JRE(Java Runtime Environment) on the server to run servlets.
6. For non-java developers, servlet is not suitable as they required to have a broad knowledge of Java servlet.
7. HTML code is mixed up with Java code therefore, changes done in one code can affect another code.
8. Writing HTML code in servlet programming is very difficult. It also makes servlet looks bulky.
9. In servlet programming, if you want to use implicit objects, you need to write some additional code in order to access them.
10. Developers must take care of exception handling because servlet programming is not thread-safe by default.

JSP(Java Server Pages)

- JSP Stands for Java Server Pages.
- It is the technology that allows you to easily create web content that has both static and dynamic components.
- Provides expression language for accessing server-side objects.
- JSP specifications extend the java Servlet API.

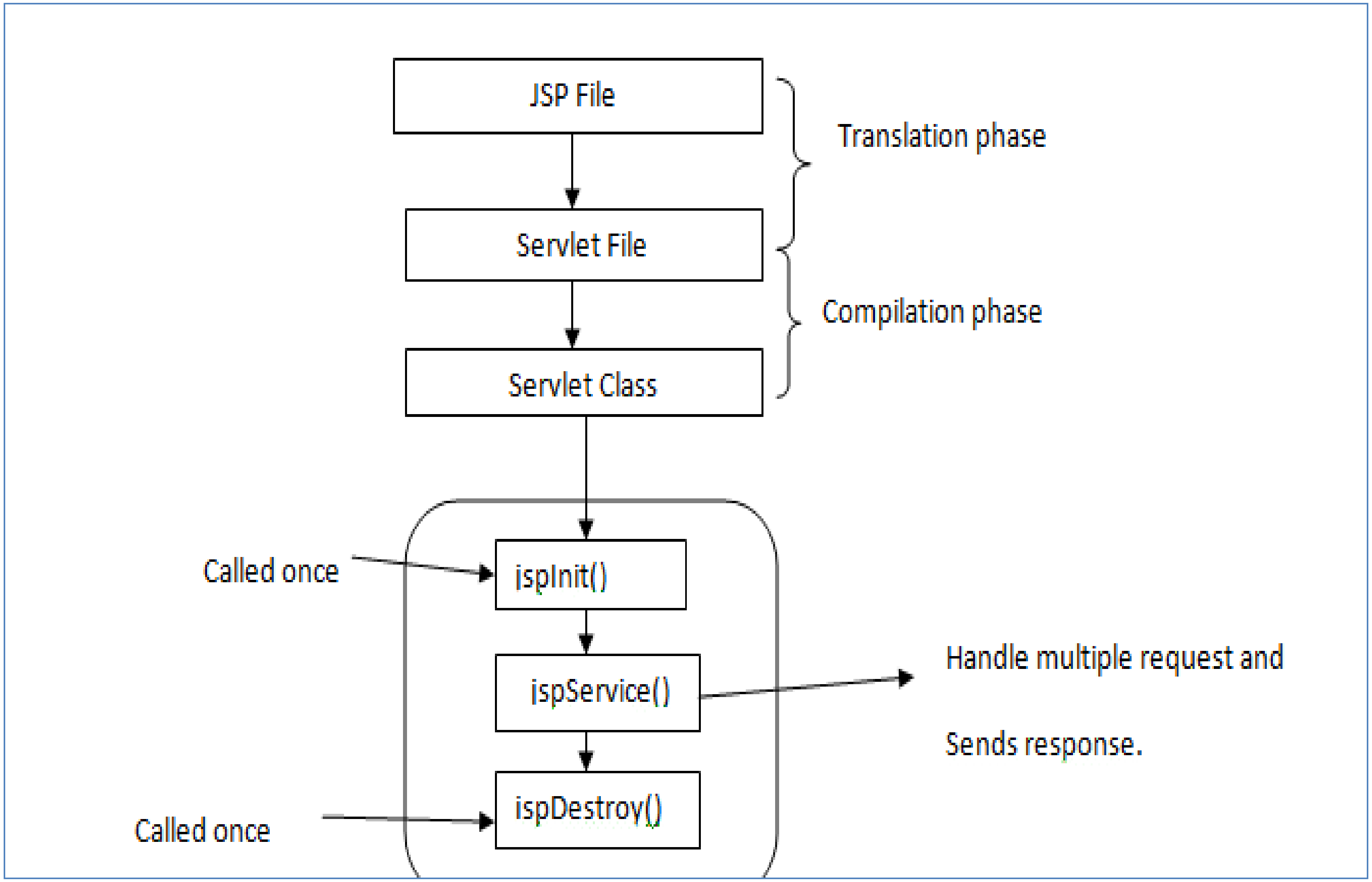
Life Cycle of JSP



Life Cycle of JSP

- A JSP Page services requests as a **servlet**.Thus,the life cycle of JSP Pages is determined by Java Servlet technology.
- When a request is mapped to a JSP page ,the web container first checks whether the JSP page’s servlet is older than the JSP page. If the servlet is older, the web container translates the JSP page into a servlet class and compiles the class

Internal Flow of JSP Page



JSP Page Interface

- Ever Jsp page should implement Servlet class
- They may Extend Servlet interface
- For JSP page there will be two main threads :

  void jspInit()-same as init()

  void jspDestroy()-same as destroy()

JSP COMPONENTS:

Directives	Are used to control how the web container translates and executes the jsp page.
Scripting Element	Are inserted into the JSP Pages servlet class.
Expression Language	Expression Are passed as parameters to calls to the JSP expression evaluator.
Jsp[set|get]Property	Elements are converted into method calls to java beans components.
Custom tags	Are converted into calls to the tag handler that implements the custom tag.

JSP Scriplets

In JSP if we want to use any component to be added within the program, we need to use <**%** and **%>**. These are known as Scriplets and for any piece of code or component we need to use Scriplets functionality.

For displaying any statement as output, we can use following statements in JSP:

- out.println(“Statement to be displayed”);
- out.write(“Statement to be printed”);
- out.print(Any expression);

1. **DIRECTIVE**

Specifies what JSP container must do.It starts with @character within the tag.

There are 3 directive-

1. page directive,
2. include directive and
3. taglib directive.

<u>**Syntax of JSP Directive:**</u>

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

2. **DECLARATIONS**

- It is a block of java code that is used to define class wide variables and methods in the generated class file.
- It is enclosed between <%! and %>

```
<%!
int a;
float b;
public void add()
{
   Piece of logic part
}
%>
```

3. **SCRIPLETS**

- It is the block of a java code that is executed when jsp is executed .
- It is enclosed in <% and %>

```
<%
out.println("HELLO JSP
    WORLD");
%>
```

4. EXPRESSIONS

- It is a shorthand notation for a scriplet that outputs a value in the response stream back to the client.
- It is enclosed in

  <%= and %>.

```
<%=new Date()%>
<input type=text value= ’<%=request.getParameter(“age”)%>’
name=’age’ >
```

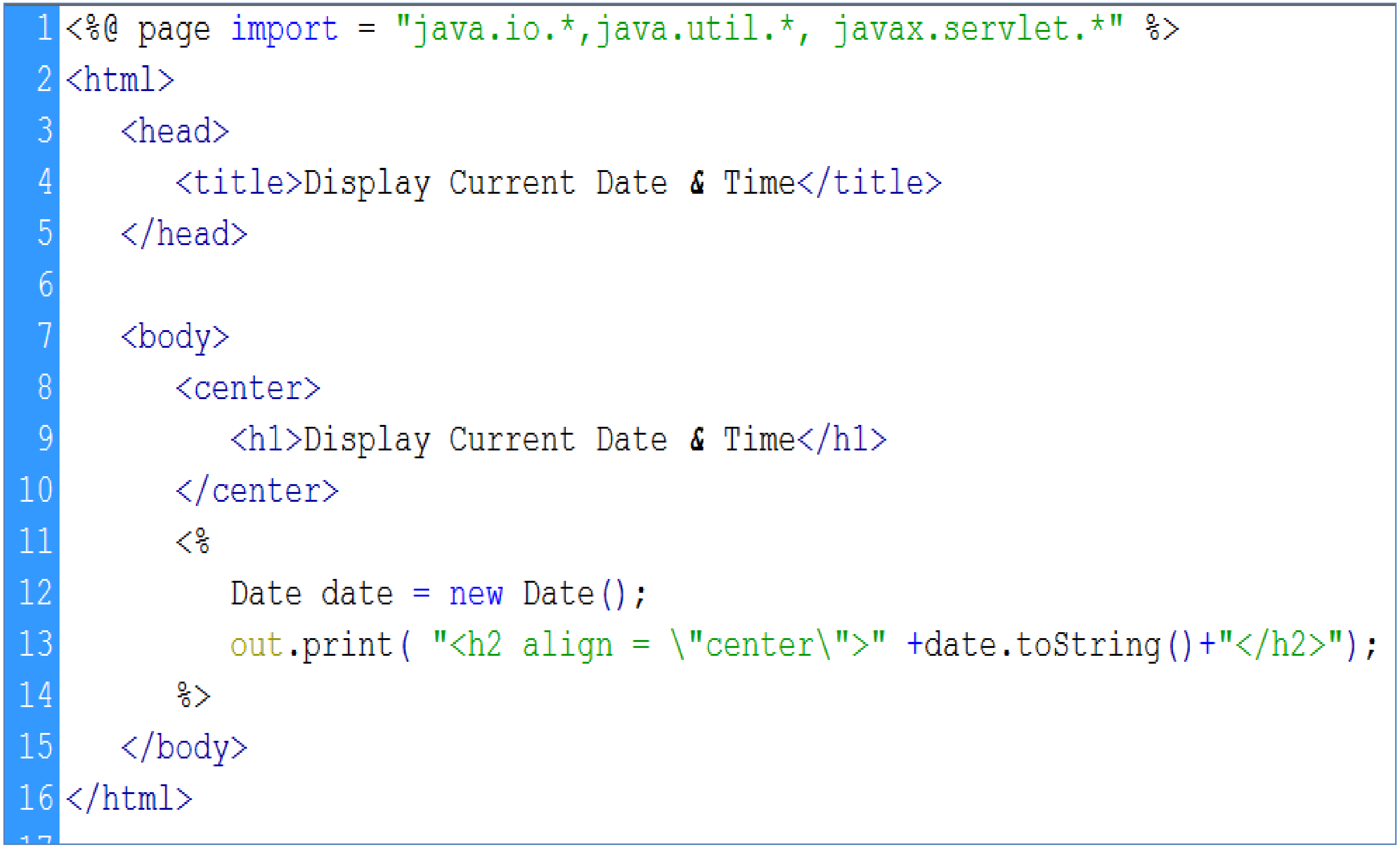
5. Comments

- It is given using <%- -   and - -%>

Syntax:

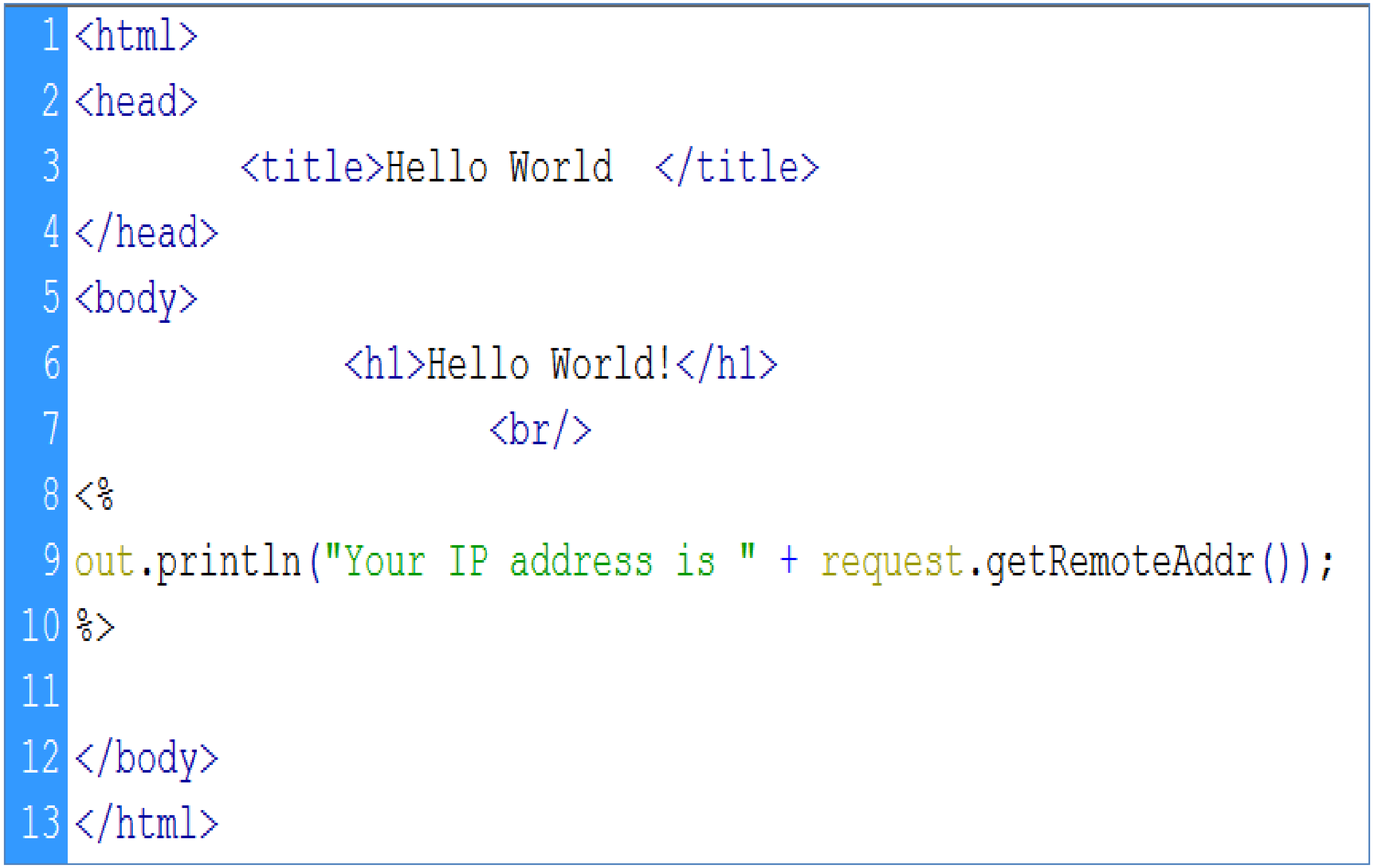
<%-- Example of SQRT function using two ways --%>

Lab Exercise 5:Create a simple JSP to print the current Date and Time.



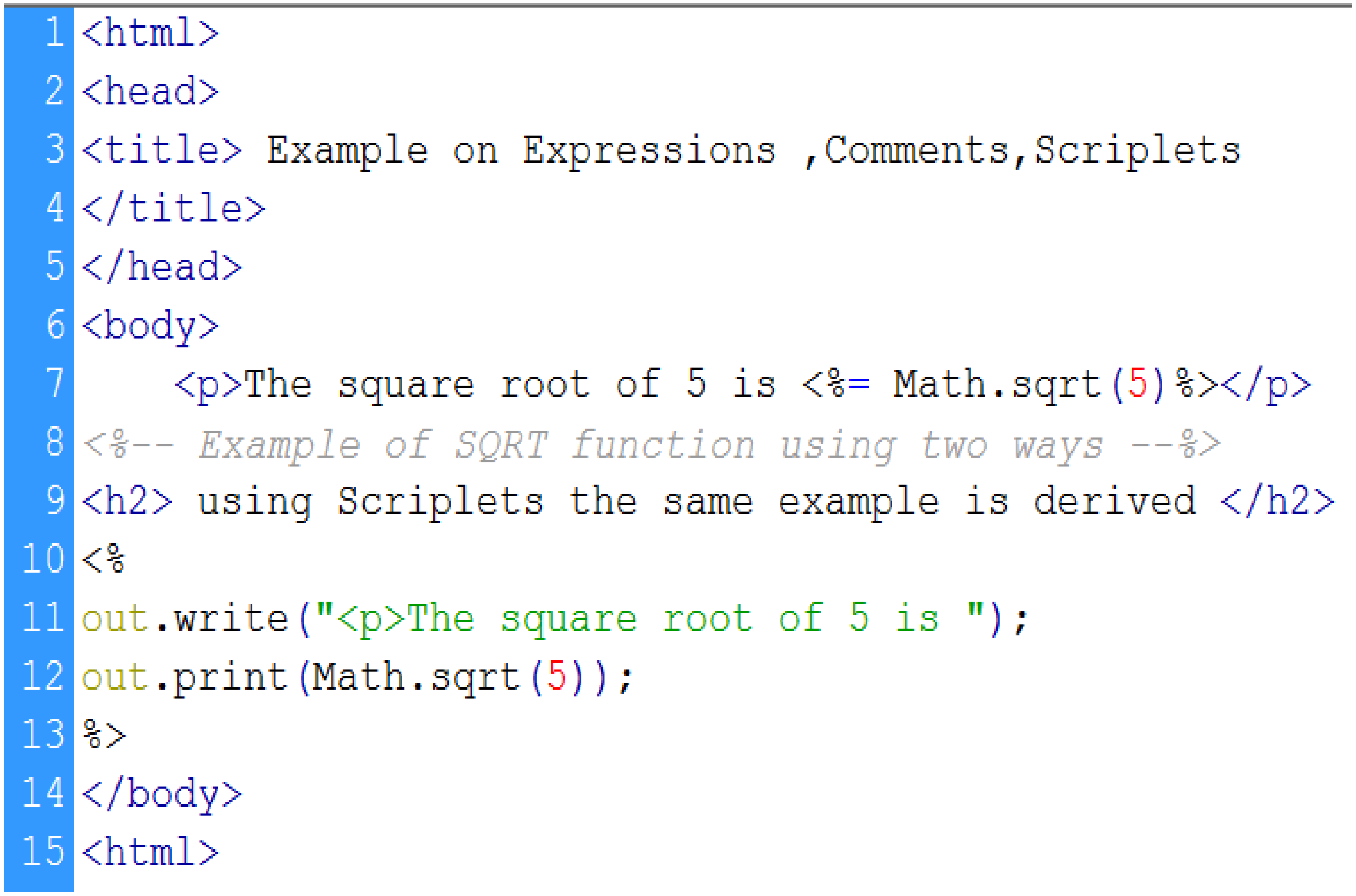
```
<%@ page import = "java.io.*,java.util.*, javax.servlet.*" %>
<html>
   <head>
      <title>Display Current Date & Time</title>
   </head>

   <body>
      <center>
         <h1>Display Current Date & Time</h1>
      </center>
      <%
         Date date = new Date();
         out.print( "<h2 align = \"center\">" +date.toString()+"</h2>");
      %>
   </body>
</html>
```

Lab Exercise 6:Create a simple JSP a) Display current IP-Address of the System b) Find out the Square root for a Number



```
<html>
<head>
        <title>Hello World  </title>
</head>
<body>
             <h1>Hello World!</h1>
                    <br/>
<%
out.println("Your IP address is " + request.getRemoteAddr());
%>

</body>
</html>
```

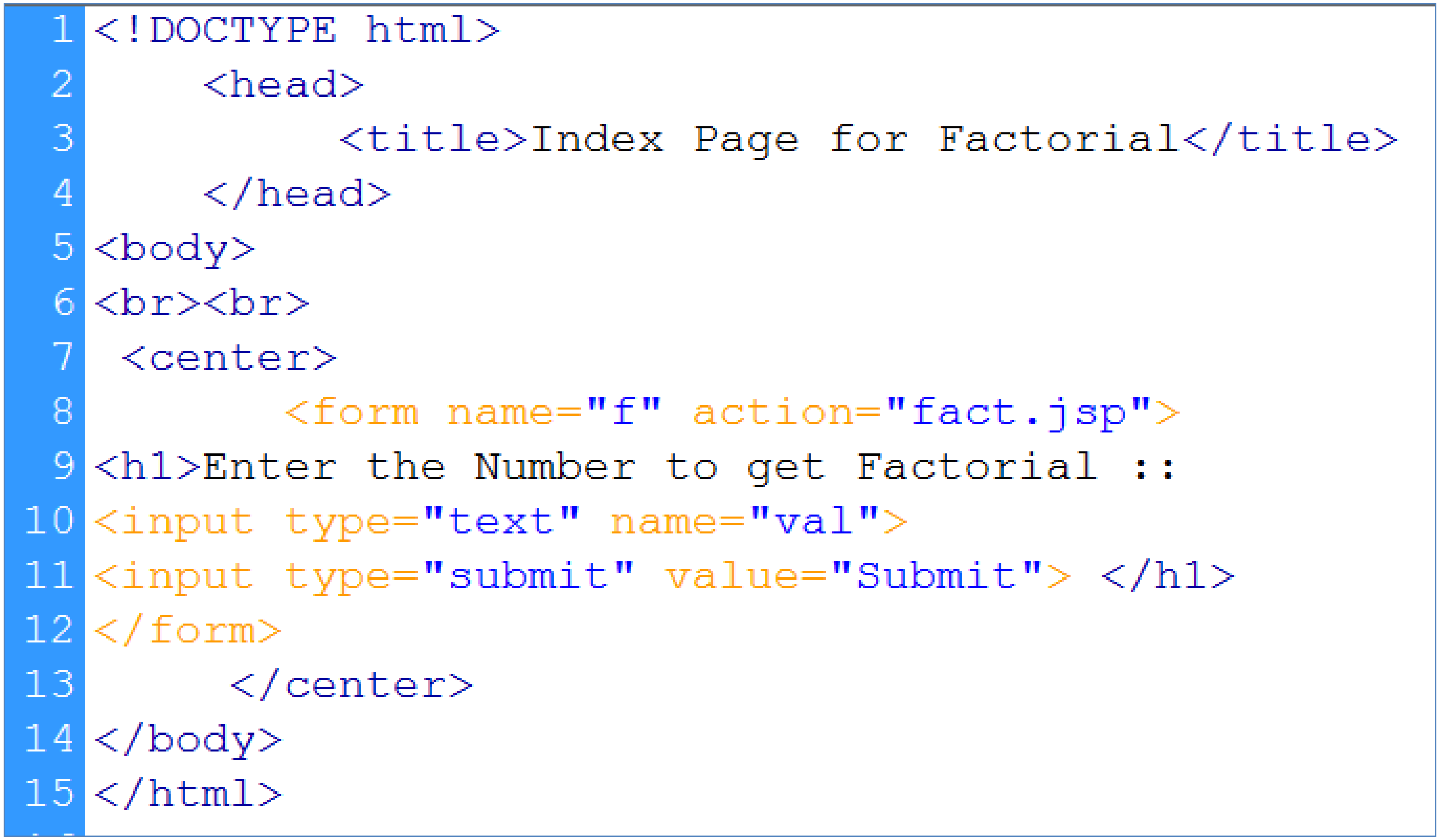
6 b) Find out the Square root for a Number



```
<html>
<head>
<title> Example on Expressions ,Comments,Scriplets
</title>
</head>
<body>
    <p>The square root of 5 is <%= Math.sqrt(5)%></p>
<%-- Example of SQRT function using two ways --%>
<h2> using Scriplets the same example is derived </h2>
<%
out.write("<p>The square root of 5 is ");
out.print(Math.sqrt(5));
%>
</body>
<html>
```

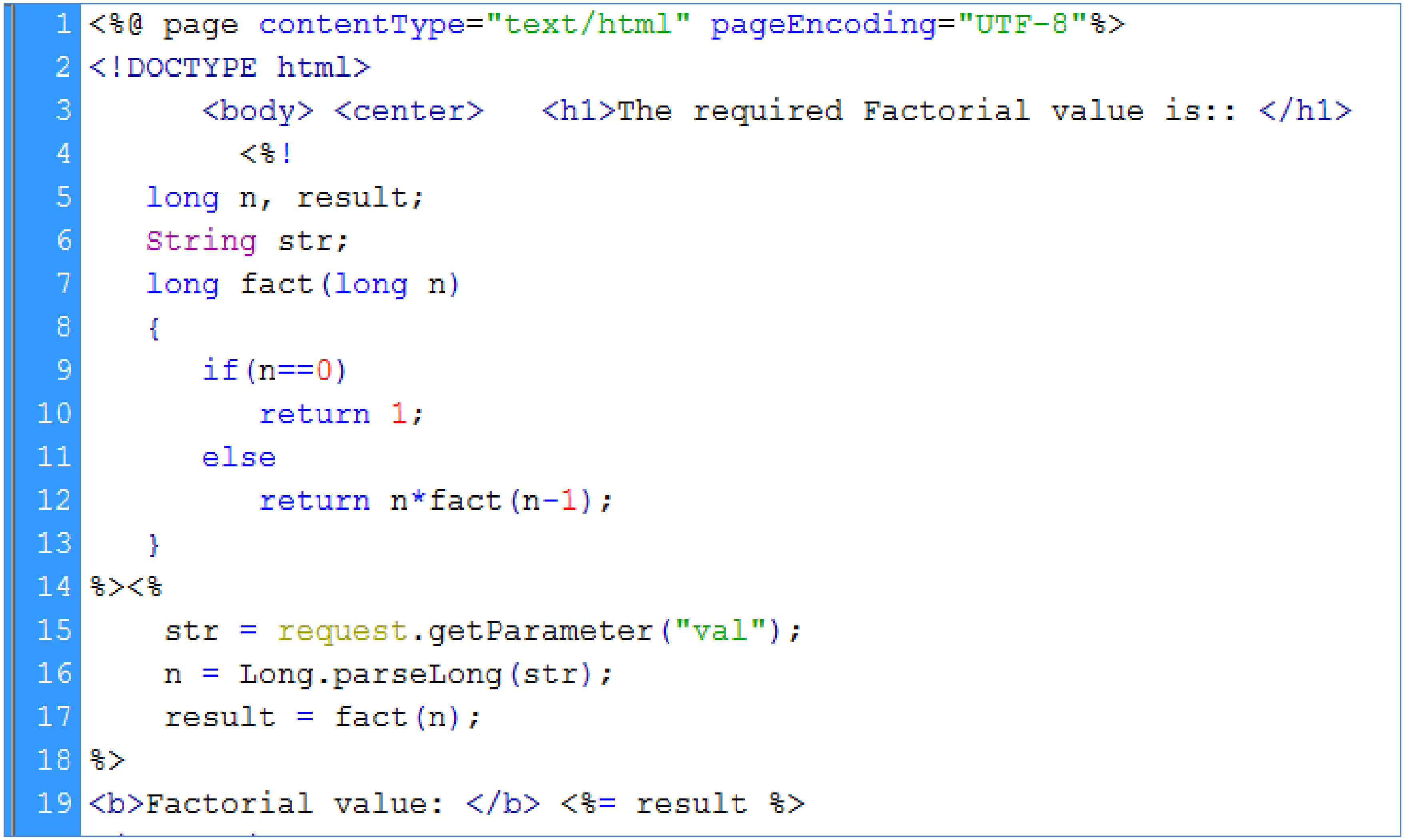
Lab Exercise 7:Develop JSP program calculates factorial values for an integer number, while the input is taken from an HTML form.

Index.jsp



```
<!DOCTYPE html>
    <head>
         <title>Index Page for Factorial</title>
    </head>
<body>
<br><br>
 <center>
       <form name="f" action="fact.jsp">
<h1>Enter the Number to get Factorial ::
<input type="text" name="val">
<input type="submit" value="Submit"> </h1>
</form>
     </center>
</body>
</html>
```

Develop JSP program calculates factorial values for an integer number, while the input is taken from an HTML form.



```
<%@ page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
      <body> <center>   <h1>The required Factorial value is:: </h1>
        <%!
   long n, result;
   String str;
   long fact(long n)
   {
      if(n==0)
         return 1;
      else
         return n*fact(n-1);
   }
%><%
    str = request.getParameter("val");
    n = Long.parseLong(str);
    result = fact(n);
%>
<b>Factorial value: </b> <%= result %>
```

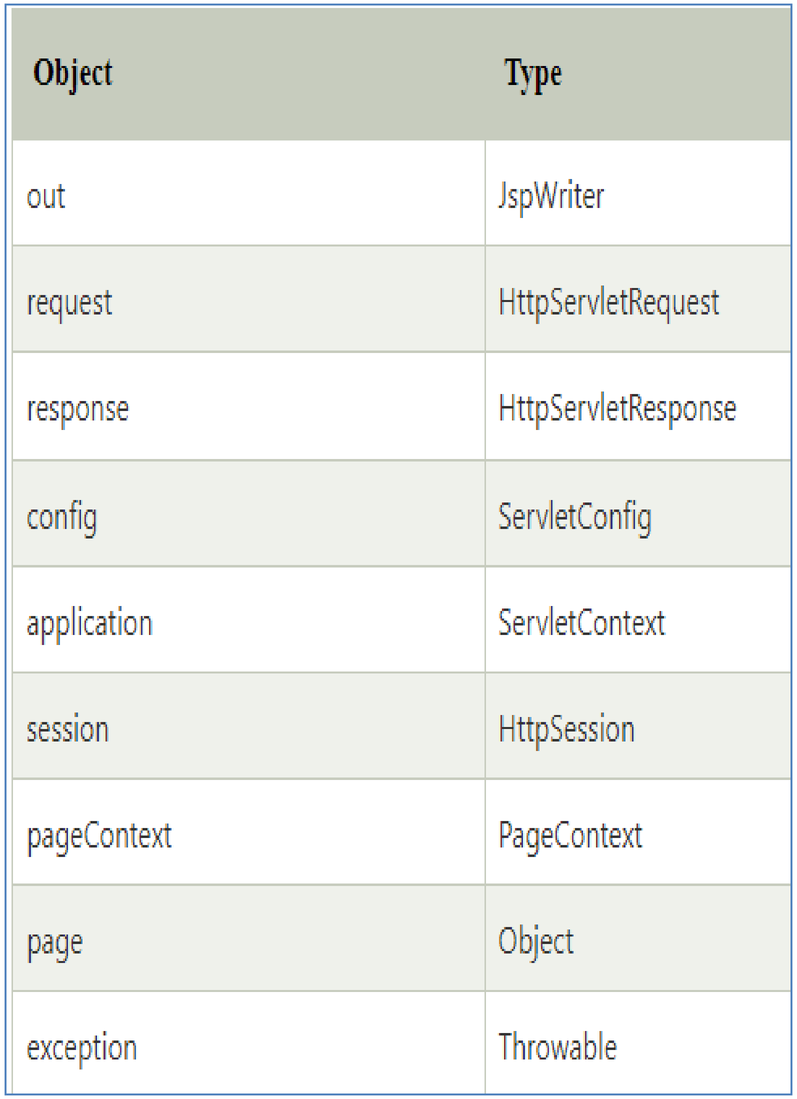
Exercise 8:Develop JSP program shows a Sample Order Form

Catalog.jsp

```
<html>
<head><title>A Catalog Order Form</title> </head>
<body> <h1 align="center">A Sample Order Form</h1>
<%!
    String item[] = {"DVD", "CD", "Diskette"};
    double price[] = {19.99, 12.99, 1.99};
    int quantity[] = {2, 9, 24};
%> <table align="center" bgcolor="lightgray" border="1" width="75%">
<tr> <td>Item</td>   <td>Price</td>   <td>Quantity</td>  <td>Total Price</td> </tr>
<% for (int i=0; i<3; i++) { %>
      <tr>
      <td><%= item[i] %></td>
      <td><%= price[i] %></td>
      <td><%= quantity[i] %></td>
      <td><%= price[i] * quantity[i] %></td>
      </tr>
<% } //end for loop %>
</table>
</body>
</html>
```

JSP IMPLICIT OBJECTS

There are **9 jsp implicit objects**. These objects are *created by the web container* that is available to all the jsp pages. The available implicit objects are out, request, config, session, application etc. A list of the **9 implicit objects** is given below:



Object	Type
out	JspWriter
request	HttpServletRequest
response	HttpServletResponse
config	ServletConfig
application	ServletContext
session	HttpSession
pageContext	PageContext
page	Object
exception	Throwable

1) JSP out implicit object

For writing any data to the buffer, JSP provides an implicit object named out. It is the object of **JspWriter**. In case of servlet you need to write:

PrintWriter out=response.getWriter ();

IN JSP we can declare out as:

- out.println();
- out.write();
- out.print();

2. request Implicit Object

The **JSP request** is an implicit object of type HttpServletRequest i.e. created for each jsp request by the web container. It can be used to get request information such as parameter, header information, remote address, server name, server port, content type, character encoding etc.

For Example

```
String name=request.getParameter("uname");
```

3) JSP response implicit object

It can be used to add or manipulate response such as redirect response to another resource, send error etc.

For example

```
<%
   response.sendRedirect("http://www.google.com");
%>
```

4) JSP config implicit object

In JSP, **config** is an implicit object of type ***ServletConfig***. This object can be used to get initialization parameter for a particular JSP page. The config object is created by the web container for each jsp page.

Generally, it is used to get initialization parameter from the web.xml file.

5) JSP Application Implicit Object

ServletConfig and **ServletContext**, both are objects created at the time of servlet initialization and used to provide some initial parameters or configuration information to the servlet. But, the difference lies in the fact that information shared by ServletConfig is for a specific servlet, while information shared by ServletContext is available for all servlets in the web application.

For example:

If we create one web site with website name, the same website name should be re-used for all the web pages hence we try to gather that information using ServletContext object.

```
getServletContext.getInitParameter("Name")
```

6) **Session Tracking Object**

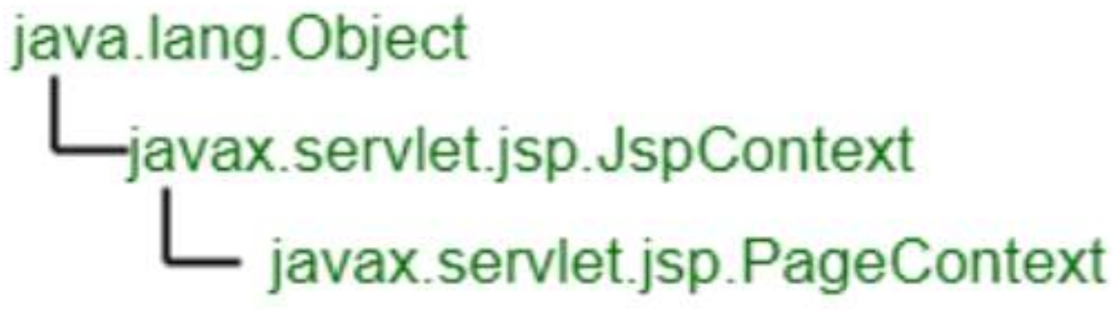
In JSP, session is an implicit object of type HttpSession.The Java developer can use this object to set,get or remove attribute or to get session information.

For Example

```
session.setAttribute("user",name);
```

7)pageContext implicit object

pageContext extends **JspContext** to contribute helpful context details while JSP technology is applied in a Servlet environment. A PageContext is an instance that gives access to all the namespaces related to a JSP page, gives access to some page attributes and a layer over the application details. Implicit objects are connected to the **pageContext** consequently.



8)page implicit Object

In JSP, page is an implicit object of type Object class.This object is assigned to the reference of auto generated servlet class. It is written as:

```
Object page=this;
```

9)Jsp exception implicit object

The exception is normally an object that is thrown at runtime. Exception Handling is the process to handle the runtime errors. There may occur exception any time in your web application. So handling exceptions is a safer side for the web developer. In JSP, there are two ways to perform exception handling:

By **errorPage** and **isErrorPage** attributes of

page directive

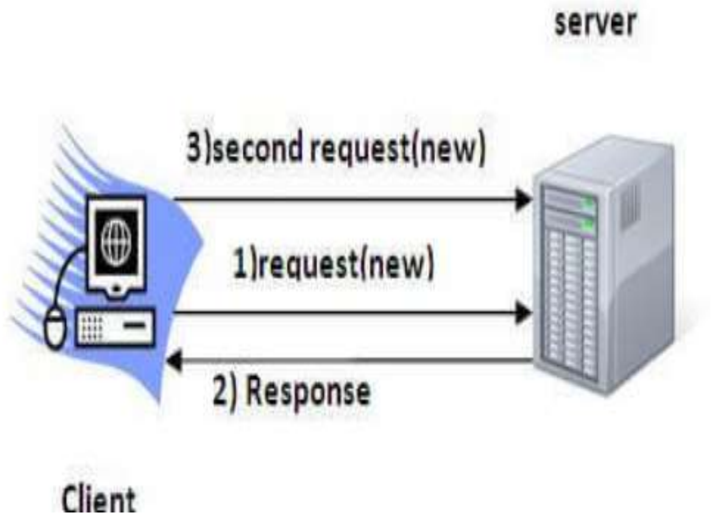
By **<error-page>** element in web.xml file

DATABASE ACCESS & SESSION TRACKING

- **Session** simply means a particular interval of time.
- **Session Tracking** is a way to maintain state (data) of an user. It is also known as **session management** in servlet.
- In general we can use any type of database for connecting JSP page to store and retrieve the data from centralized data storage location. Here we are going to use **MY-SQL** as back end database to store and retrieve the information.
- For connecting the JSP pages with My-SQL we need to use JDBC connectivity with Type 4 Driver. Once JDBC type 4 driver is constructed now the JSP pages can be easily connected with the database

Session Tracking

- **Session** simply means a particular interval of time.
- **Session Tracking** is a way to maintain state (data) of an user. It is also known as **session management** in servlet.
- **Http protocol** is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.



Database Connection Using My-SQL In JSP

Web Technologies

Unit 5: Contents

UNIT V: Fundamentals of NODE JS and Angular :

Understanding Node.js, Installing Node.js, Working with Node Packages, Creating a Node.js Application, Understanding Angular, Modules, Directives, Data Binding, Dependency Injection, Services, Creating a Basic Angular Application.

What is Node Js?

1) What is NODE JS?

Node.js is an open-source, cross-platform runtime environment, library, and development framework used to create server-side and networking JavaScript applications. It also provides developers with a vast library of JavaScript modules that simplify coding

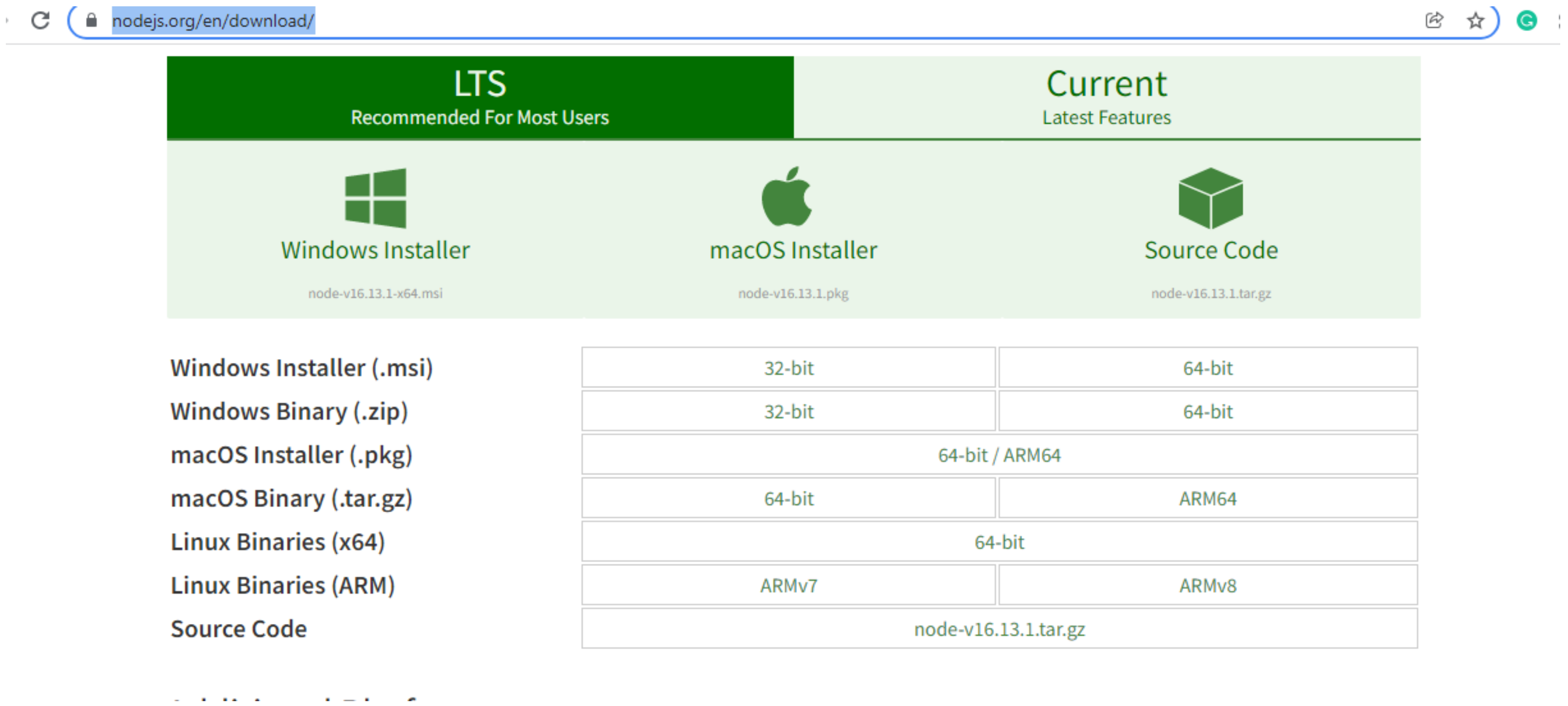
Node.js offers developers the following benefits

1. It’s open-source
2. It’s scalable. Developers can use it either for horizontal scaling or vertical scaling
3. It supports out of the box unit testing. Developers can use any JavaScript unit testing framework to test their Node.js code
4. It features built-in application programming interfaces (API) that helps developers create different types of servers
5. It is a high-performance tool, thanks to incorporating non-blocking I/O operations. It employs the JavaScript V8 engine to execute code, increasing its speed
6. It supports scripting languages like Ruby, CoffeeScript, and TypeScript
7. It enables rapid development suitable for applications that require frequent changes

Node JS Installation

Go to the URL :

https://nodejs.org/en/download/



nodejs.org/en/download/

LTS Recommended For Most Users	Current Latest Features

Windows Installer	macOS Installer	Source Code
node-v16.13.1-x64.msi	node-v16.13.1.pkg	node-v16.13.1.tar.gz

Windows Installer (.msi)	32-bit	64-bit
Windows Binary (.zip)	32-bit	64-bit
macOS Installer (.pkg)	64-bit / ARM64	
macOS Binary (.tar.gz)	64-bit	ARM64
Linux Binaries (x64)	64-bit	
Linux Binaries (ARM)	ARMv7	ARMv8
Source Code	node-v16.13.1.tar.gz	

Choose the windows platform based on ur Laptop or PC Operating System and then click on that installer

In my Pc I am using windows as operating system and my working bit is 64-bit.

Download windows installer 64 bit file

Double click on that installer and try to install the **Node.Js Installer Platform**

Installation will be completed

Working with Node Js Package

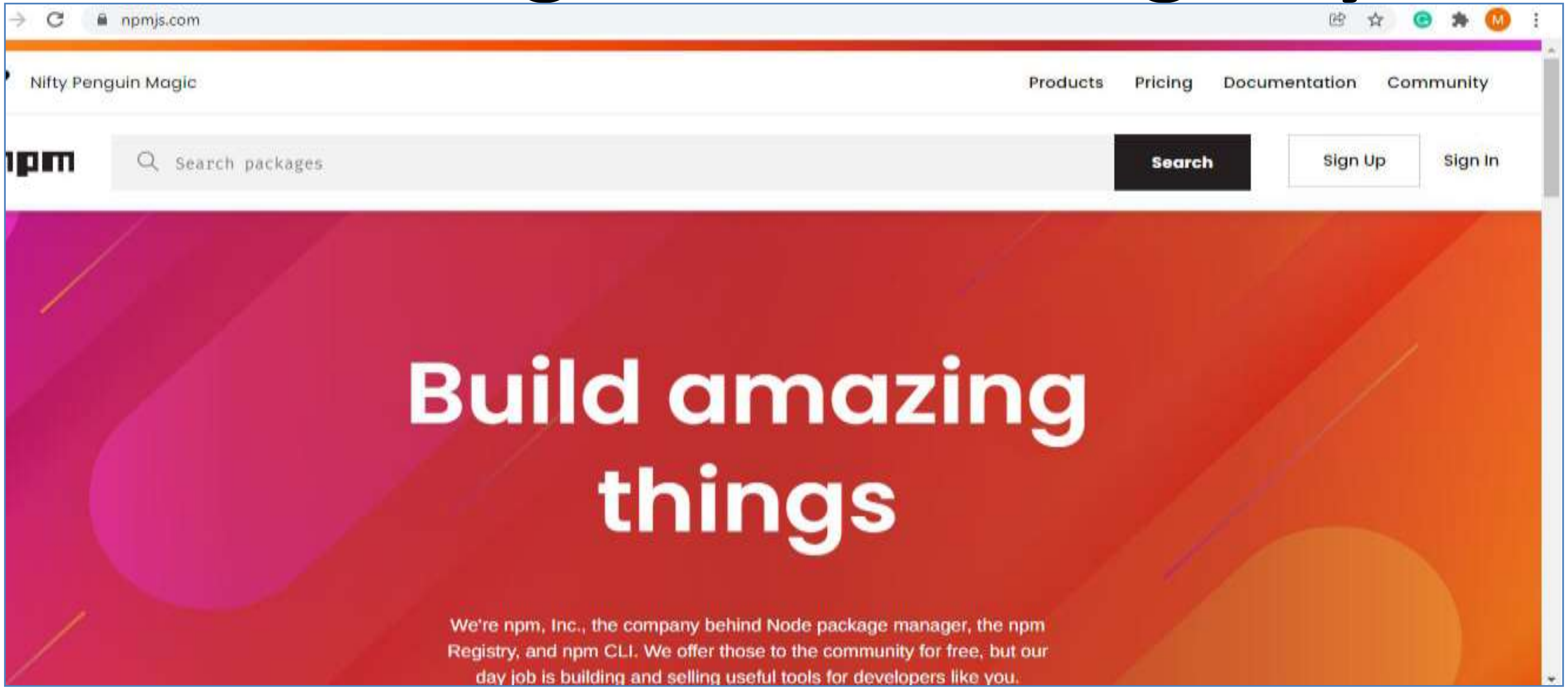
1. One of the most powerful features of the Node.js framework is the ability to easily extend it with additional Node Packaged Modules (NPMs) using the **Node Package Manager (NPM).** NPM is mainly used for referring Node Packaged Modules as **Modules**.
2. A Node Packaged Module is a packaged library that can easily be shared, reused, and installed in different projects. Many different **modules** are available for a variety of purposes.
3. Node.js modules are created by various third-party organizations to provide the needed features that Node.js lacks out of the box. This community of contributors is active in adding and updating modules.

The **Node modules** have a managed location called the **Node Package Registry** where packages are registered. This allows you to publish your own packages in a location where others can use them as well as download packages that others have created. The Node Package Registry is located at **https://npmjs.com**

From this location you can view the newest and most popular modules as well as search for specific packages

https://npmjs.com

Home Page for NPM Registry



Build amazing things

We're npm, Inc., the company behind Node package manager, the npm Registry, and npm CLI. We offer those to the community for free, but our day job is building and selling useful tools for developers like you.

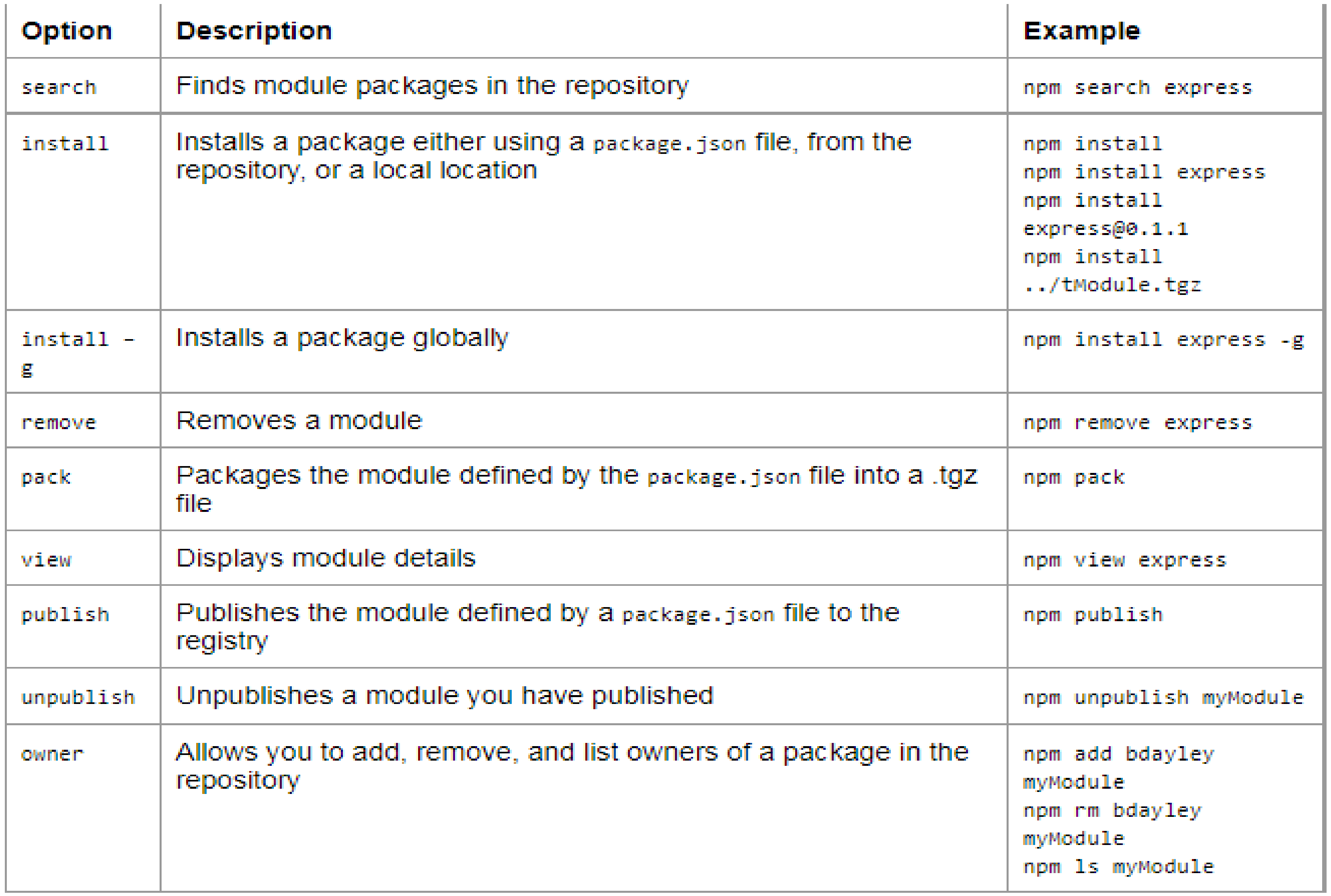
Here u can create whatever u want to create new packages or else find the details about already existing packages

USING THE NODE PACKAGE MANAGER

The **Node Package Manager** you have already seen is a command-line utility. It allows you to find, install, remove, publish, and do everything else related to **Node Package Modules**. The Node Package Manager provides the link between the **Node Package Registry** and your **development environment**.

The simplest way to really explain the **Node Package Manager** is to list some of the command-line options and what they do.

Node Package Manager Register Methods



Option	Description	Example
`search`	Finds module packages in the repository	`npm search express`

`install`	Installs a package either using a `package.json` file, from the repository, or a local location	`npm install`
		`npm install express`
		`npm install express@0.1.1`
		`npm install ../tModule.tgz`

`install -g`	Installs a package globally	`npm install express -g`
`remove`	Removes a module	`npm remove express`
`pack`	Packages the module defined by the `package.json` file into a .tgz file	`npm pack`
`view`	Displays module details	`npm view express`
`publish`	Publishes the module defined by a `package.json` file to the registry	`npm publish`
`unpublish`	Unpublishes a module you have published	`npm unpublish myModule`

`owner`	Allows you to add, remove, and list owners of a package in the repository	`npm add bdayley myModule`
		`npm rm bdayley myModule`
		`npm ls myModule`

Node.Js First Application

- Before creating an actual **"First Node Js Welcome !"** application using Node.js.
- Let us see the components of a Node.js application. A Node.js application consists of the following three important components :

1) Import required modules
2) Create Server
3) Read request and return response

1)Import required modules

We use the **require** directive to load **Node.js** modules.. In general for importing any modules in Node.js we use **require** keyword.

For Example

```
var http = require("http");
```

Here **var** is keyword which is used to create a **variable** and we use **require** to load the **http package** for initializing the **http protocol**

2)Create Server

Create server − A server which will listen to client's requests similar to Apache HTTP Server.This server is one which should be invoked by the user, but in apache tomcat we can get server automatically invoked.

We use the created http instance and call **http.createServer()** method to **create a server** instance and then we bind it at port **8081** using the **listen** method associated with the server instance.

3) **Testing Request & Response**

Let's put step 1 and 2 together in a file called **hello.js and start our HTTP server as shown below –**

I.e Step 1 and Step 2 are stored in one file and save the file with extension **.js**

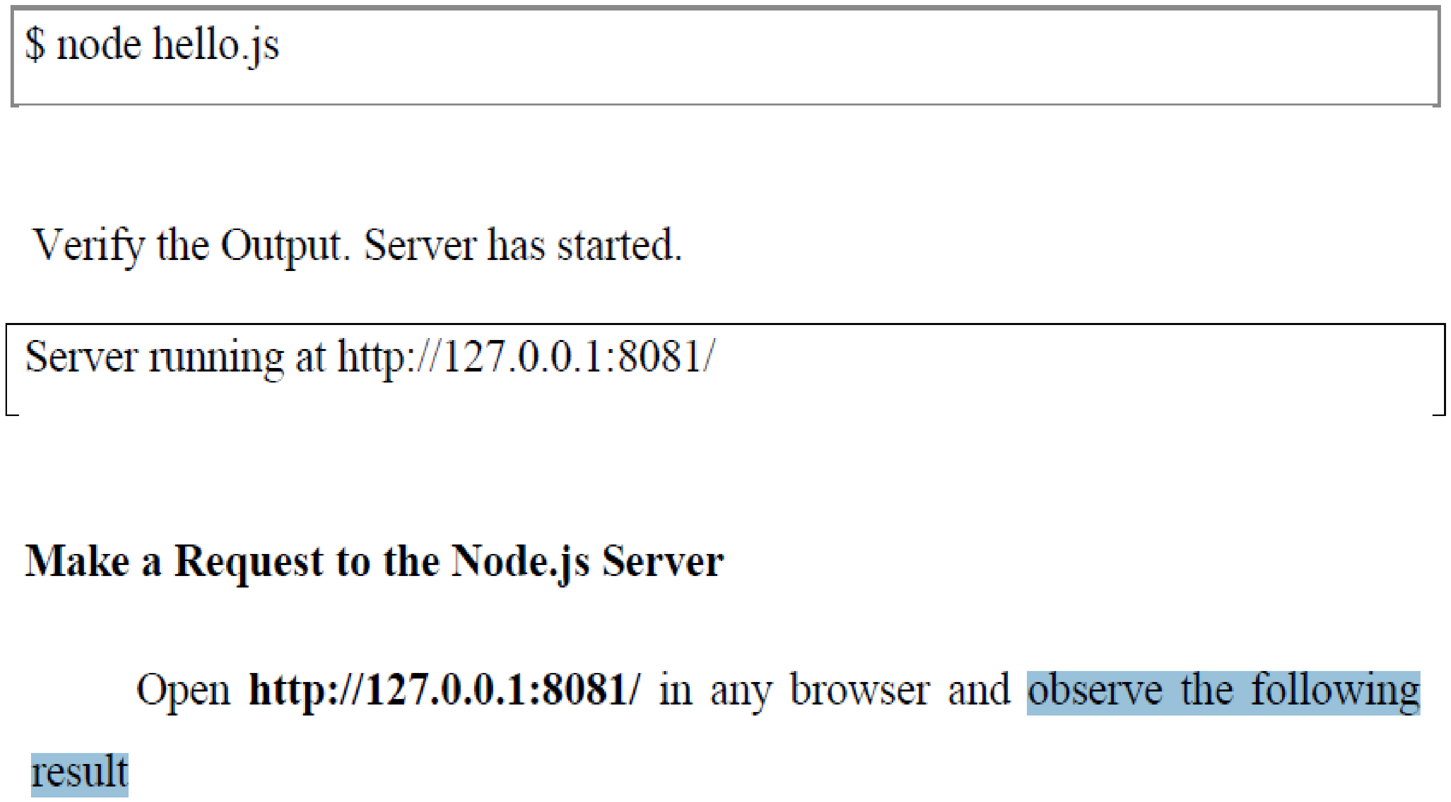
Here in our example we want to display hello world program so we try to save the filename as **hello.js**

Exercise 10: Design a simple Node JS application

```
var http = require("http");
http.createServer(function (request, response) {
  // Send the HTTP header
  // HTTP Status: 200 : OK
  // Content Type: text/plain
  response.writeHead(200, {'Content-Type': 'text/plain'});
    // Send the response body as " First Node Js Program !"
  response.end(First Node Js Program ! \n');
}).listen(8081);


// Console will print the message
console.log('Server running at http://127.0.0.1:8081/');
```

How to run the hello.js program

Now execute the hello.js to start the server as follows −



```
$ node hello.js
```

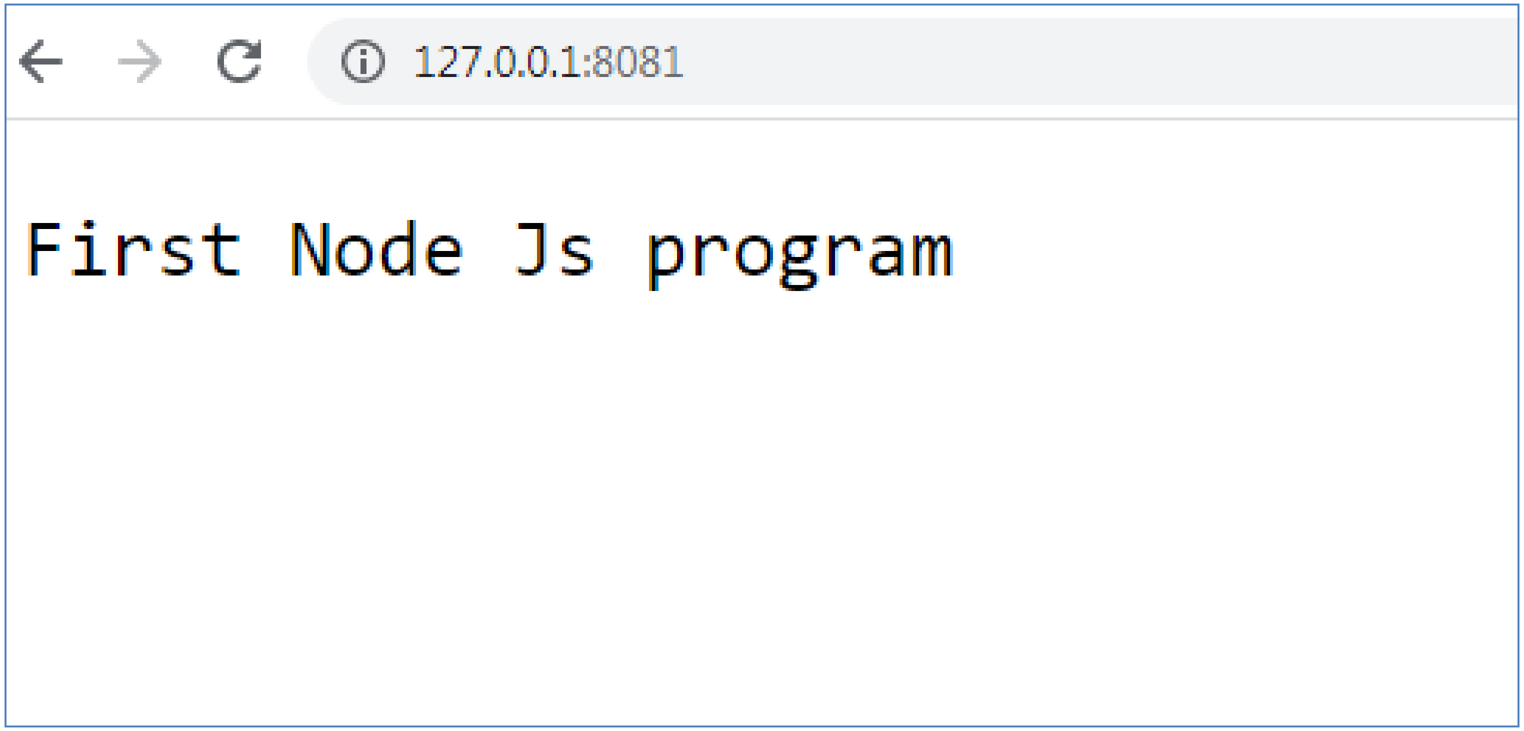
Verify the Output. Server has started.

```
Server running at http://127.0.0.1:8081/
```

Make a Request to the Node.js Server

Open **http://127.0.0.1:8081/** in any browser and observe the following result

Output



```
First Node Js program
```

HTTP Response Codes

HTTP Response Codes are classified into 5 classes

1. Informational responses (100–199)

2. Successful responses (200–299)

3. Redirection messages (300–399)

4. Client error responses (400–499)

5. Server error responses (500–599)

In our program we use **200 as response code**. That means it is **ok**

Understanding Angular JS

AngularJS is an open-source structural framework developed and maintained by Google. It lets developers use HTML as a template language, and is used to create dynamic, single-page client-side web applications.

AngularJS gives developers the following advantages:

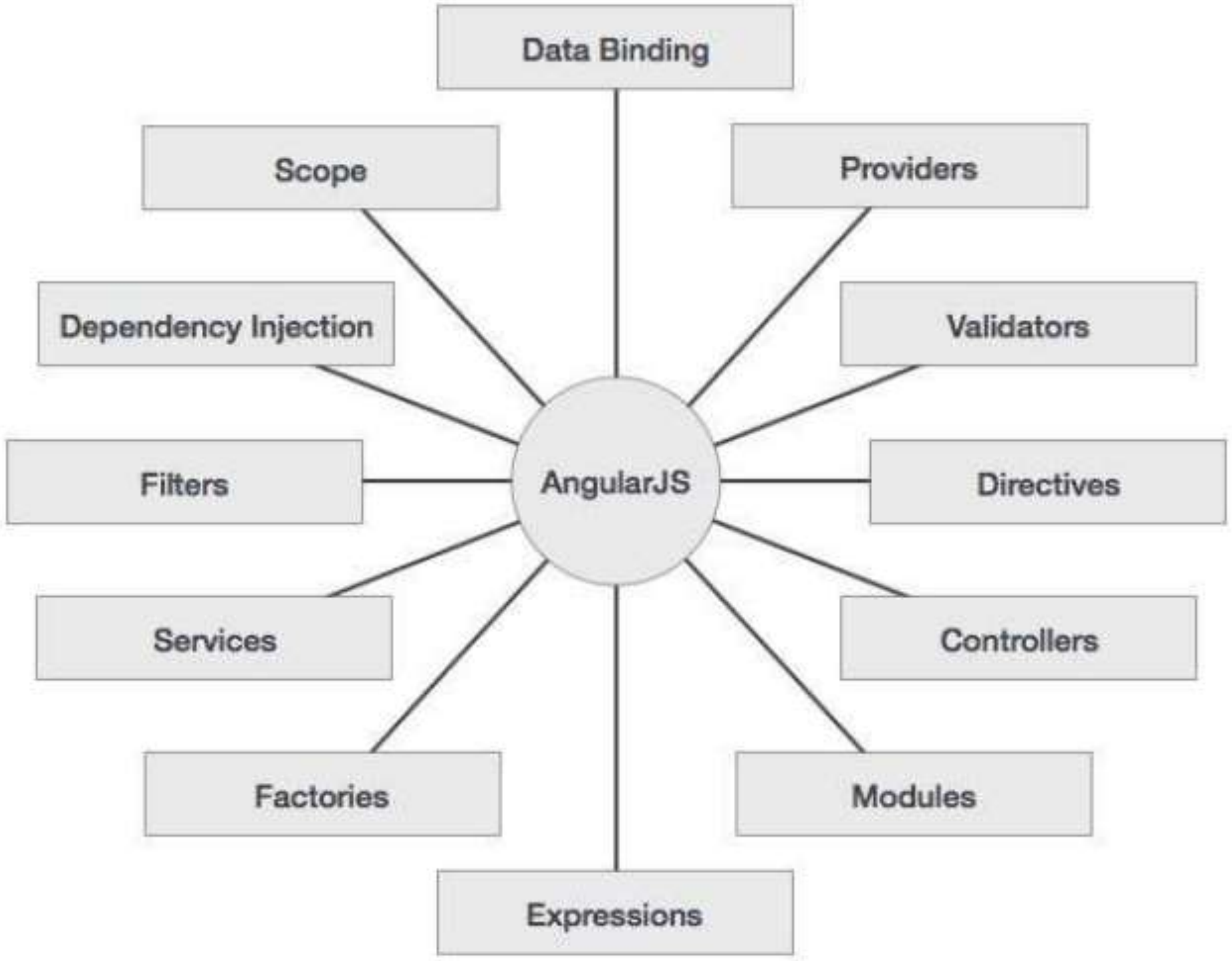
1. It's open-source
2. It's easy to use, thanks to decoupling **Document Object Models (DOM)** manipulation from application logic
3. It provides **built-in features** like filters, directives, and **automatic data binding**
4. It provides a smooth, dynamic **Model View Control** Architecture, making it easier for developers to create client-side web applications
5. It uses the **Plain Old JavaScript Objects (POJO)** data model, producing spontaneous and clean code, ideal for interactive, user-friendly web-based apps
6. It supports **object-oriented**, functional, and event-driven programming paradigms
7. It makes **unit testing easy**, thanks to its built-in test runner (**Karma)**

Disadvantages of AngularJS

Though AngularJS comes with a lot of merits, here are some points of concern:

1) **Not secure :** Being JavaScript only framework, application written in AngularJS are not safe. Server side authentication and authorization is must to keep an application secure. But this is not possible in angular.
2) **Not degradable:** If the user of your application disables JavaScript, then nothing would be visible, except the basic page.

Angular JS Concepts



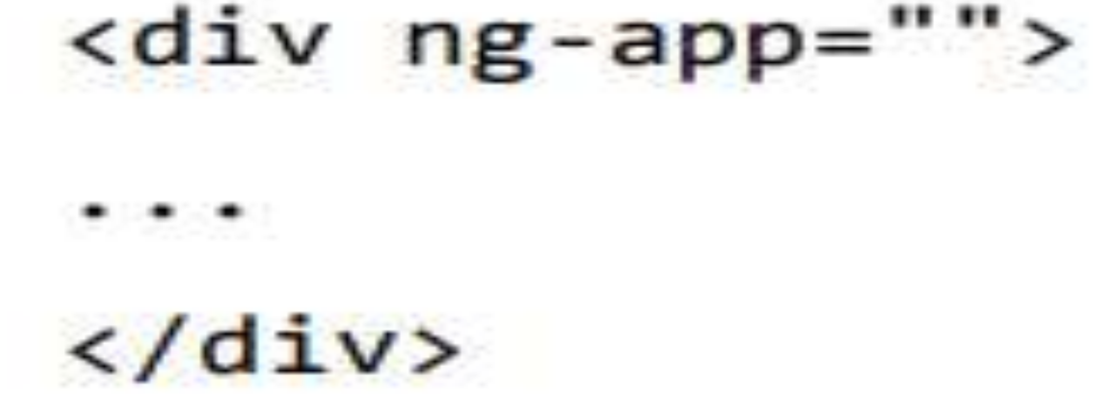
Angular Js Directives

The AngularJS framework can be divided into three major parts:

1) **ng-app :** This directive defines and links an AngularJS application to HTML.
2) **ng-model :** This directive binds the values of AngularJS application data to HTML input controls.
3) **ng-bind :** This directive binds the AngularJS application data to HTML tags

Ng-App Directive

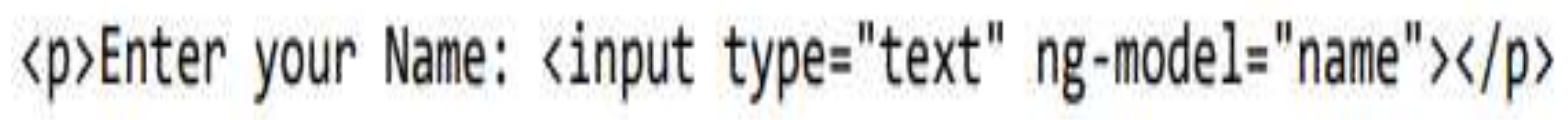
The ng-app directive starts an AngularJS Application. It defines the root element. It automatically initializes or bootstraps the application when the web page containing AngularJS Application is loaded. It is also used to load various AngularJS modules in AngularJS Application.



```
<div ng-app="">

...

</div>
```

NG MODEL DIRECTIVE

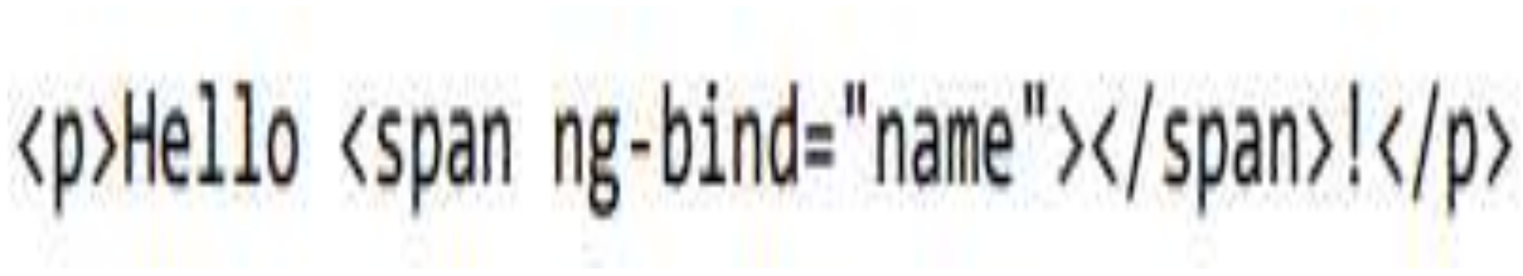
The ng-model directive defines the model/variable to be used in AngularJS Application. In the following example, we define a model named name



```
<p>Enter your Name: <input type="text" ng-model="name"></p>
```

ng-bind

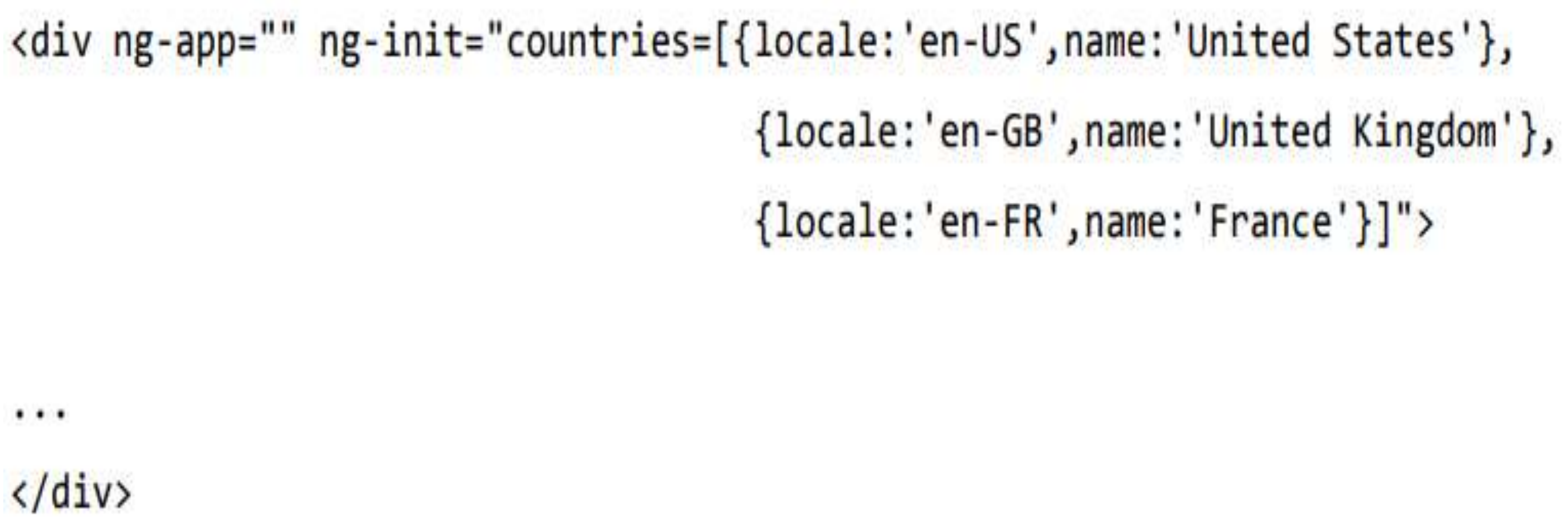
This directive binds the AngularJS Application data to HTML tags.



```
<p>Hello <span ng-bind="name"></span>!</p>
```

OTHER DIRECTIVES: ng-init directive

The ng-init directive initializes an AngularJS Application data. It is used to assign values to the variables. In the following example, we initialize an array of countries. We use JSON syntax to define



```
<div ng-app="" ng-init="countries=[{locale:'en-US',name:'United States'},
                                   {locale:'en-GB',name:'United Kingdom'},
                                   {locale:'en-FR',name:'France'}]">

...

</div>
```

NG-REPEAT DIRECTIVE

The ng-repeat directive repeats HTML elements for each item in a collection. This is used just like loop statements.

Angular MODULES: NODE Package manager (npm)

NPM supports **modularity**.

- Each modules can be bundled under a single package.
- Each module contains methods and properties.
- Modules can be a single file or a collection of multiples files/folders.
- The reason programmers are heavily reliant on modules is because of their re-usability as well as the ability to break down a complex piece of code into manageable chunks.
- Modules are of three types:
  1. **Core Modules**
  2. **local Modules**
  3. **Third-party Modules**

NPM is used to load necessary Angular JS Modules into the application and then execute the programs very easily.

How to install Angular 7

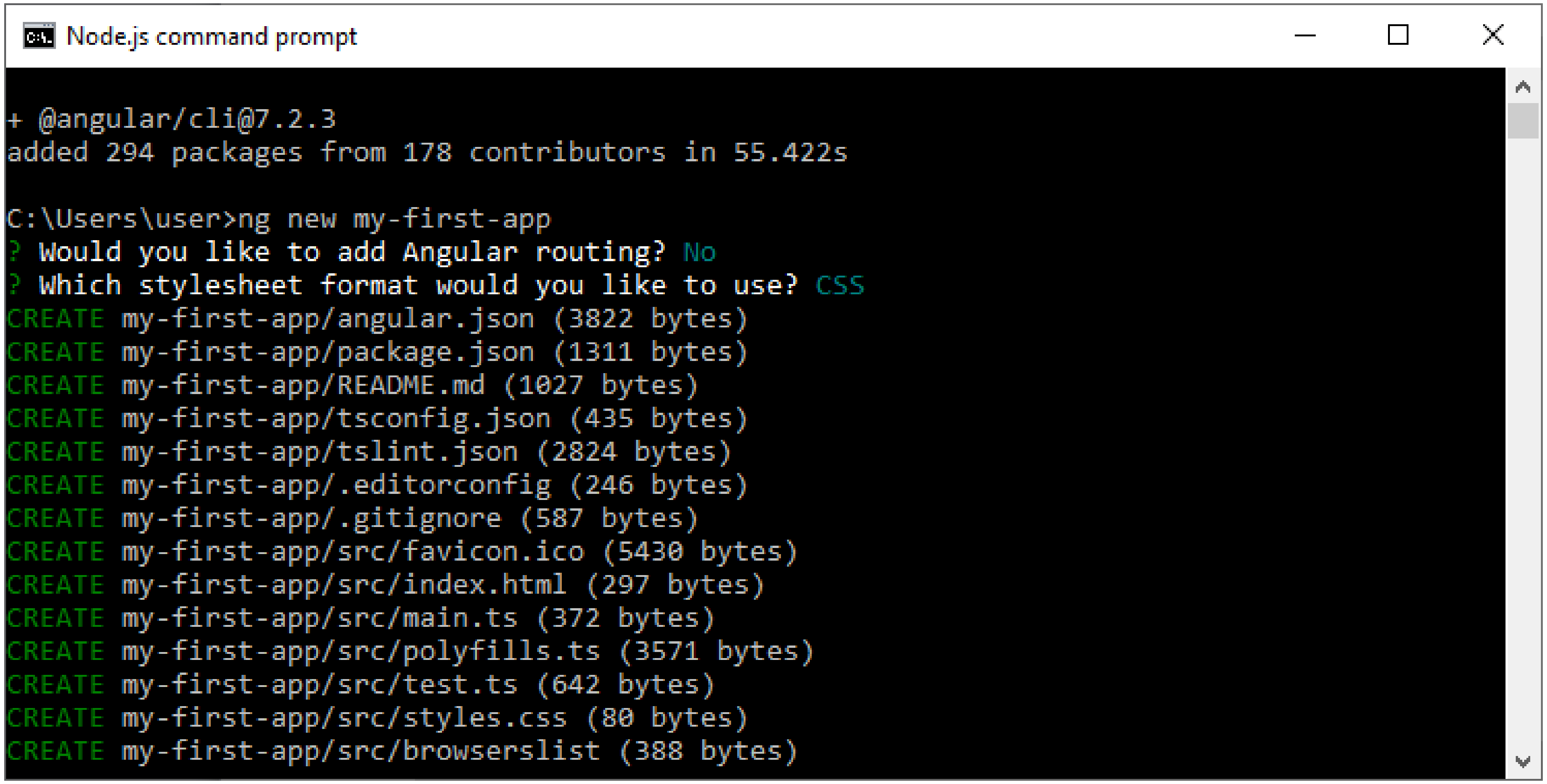
- Install Visual Studio Code IDE or JetBrains WebStorm
- You can download VS Code from **https://code.visualstudio.com**
- Run the Angular CLI command to install Angular CLI
- npm install -g @angular/cli

Or

- Your Angular 7 Environment setup is complete now.

Run the following command to create your first Angular app.

ng new my-first-app



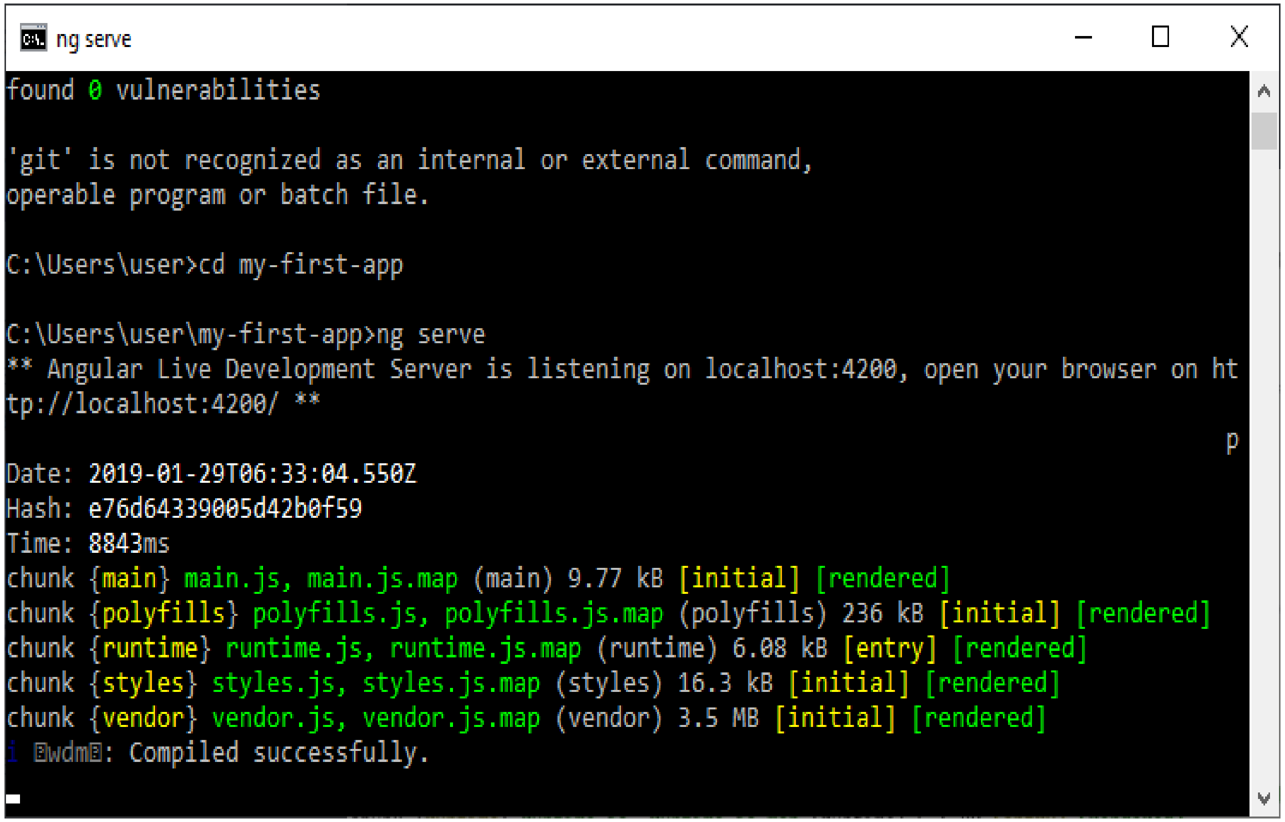
```
+ @angular/cli@7.2.3
added 294 packages from 178 contributors in 55.422s

C:\Users\user>ng new my-first-app
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use? CSS
CREATE my-first-app/angular.json (3822 bytes)
CREATE my-first-app/package.json (1311 bytes)
CREATE my-first-app/README.md (1027 bytes)
CREATE my-first-app/tsconfig.json (435 bytes)
CREATE my-first-app/tslint.json (2824 bytes)
CREATE my-first-app/.editorconfig (246 bytes)
CREATE my-first-app/.gitignore (587 bytes)
CREATE my-first-app/src/favicon.ico (5430 bytes)
CREATE my-first-app/src/index.html (297 bytes)
CREATE my-first-app/src/main.ts (372 bytes)
CREATE my-first-app/src/polyfills.ts (3571 bytes)
CREATE my-first-app/src/test.ts (642 bytes)
CREATE my-first-app/src/styles.css (80 bytes)
CREATE my-first-app/src/browserslist (388 bytes)
```

- Navigate to your first app.

  cd my-first-app

- Start your server to run app.

  ng serve



```
found 0 vulnerabilities

'git' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\user>cd my-first-app

C:\Users\user\my-first-app>ng serve
** Angular Live Development Server is listening on localhost:4200, open your browser on ht
tp://localhost:4200/ **
                                                                                         p
Date: 2019-01-29T06:33:04.550Z
Hash: e76d64339005d42b0f59
Time: 8843ms
chunk {main} main.js, main.js.map (main) 9.77 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 236 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.08 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 16.3 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 3.5 MB [initial] [rendered]
i ｢wdm｣: Compiled successfully.
```

About Data Binding

Data binding is the synchronization of data between business logic and view of the application. It serves as a bridge between two components of angular that is model part and view part. Data Binding is automatic and provides a way to wire the two important part of an application that is the UI and application data.

How data binding is possible in angular

The **directive in AngularJS** used to bind the value of the input field (such as text field, text area) to the HTML element is **ng-model**. An **ng-model** directive is used to perform data binding in angular. We don’t have to write extra code to link the model part with view part by adding few snippets of code we can bind the data with the HTML control.

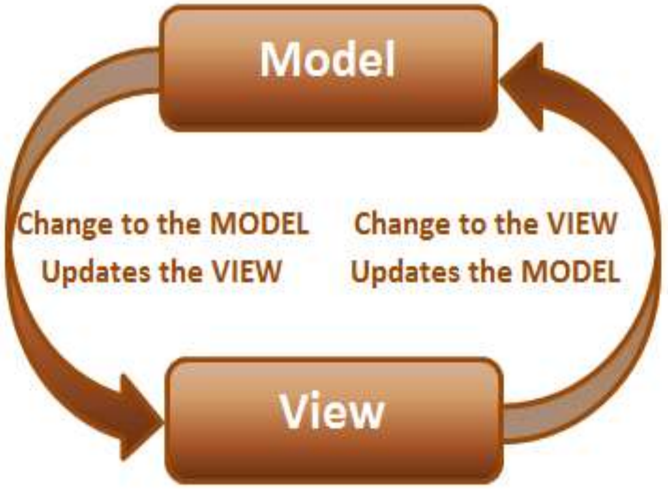
In the **Angular JS** we will be using **ng-model** and **ng-bind** for performing the binding operations.

Data Binding

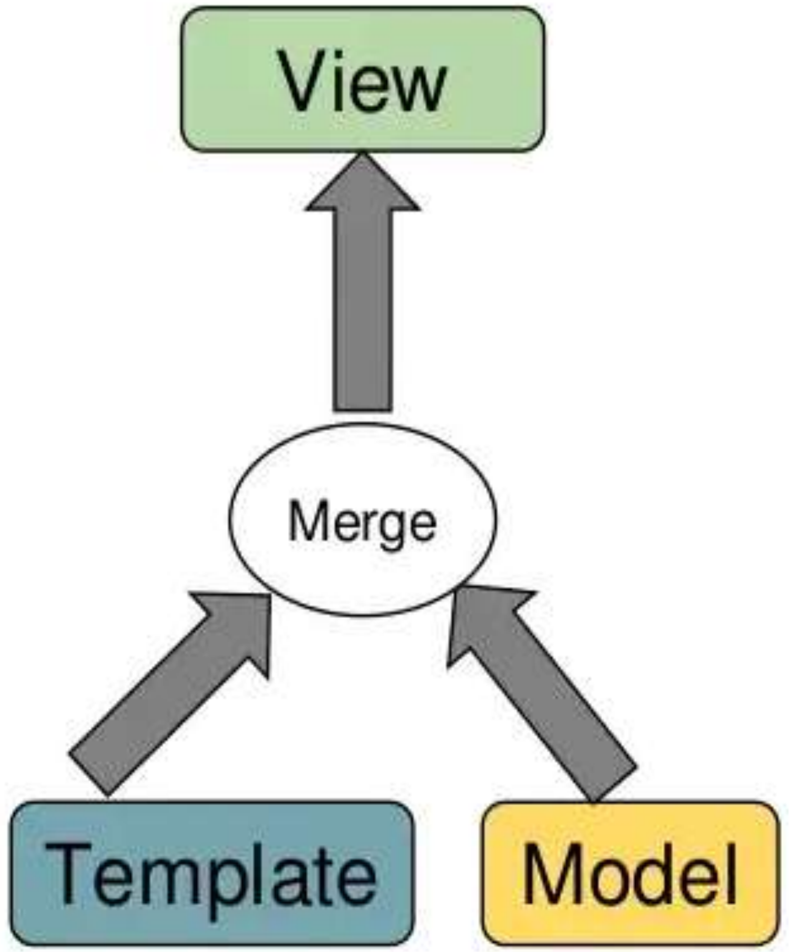
There are two types of data binding present in Angular Js. They are as follows:

1) One-way Data Binding
2) Two –Way Data Binding

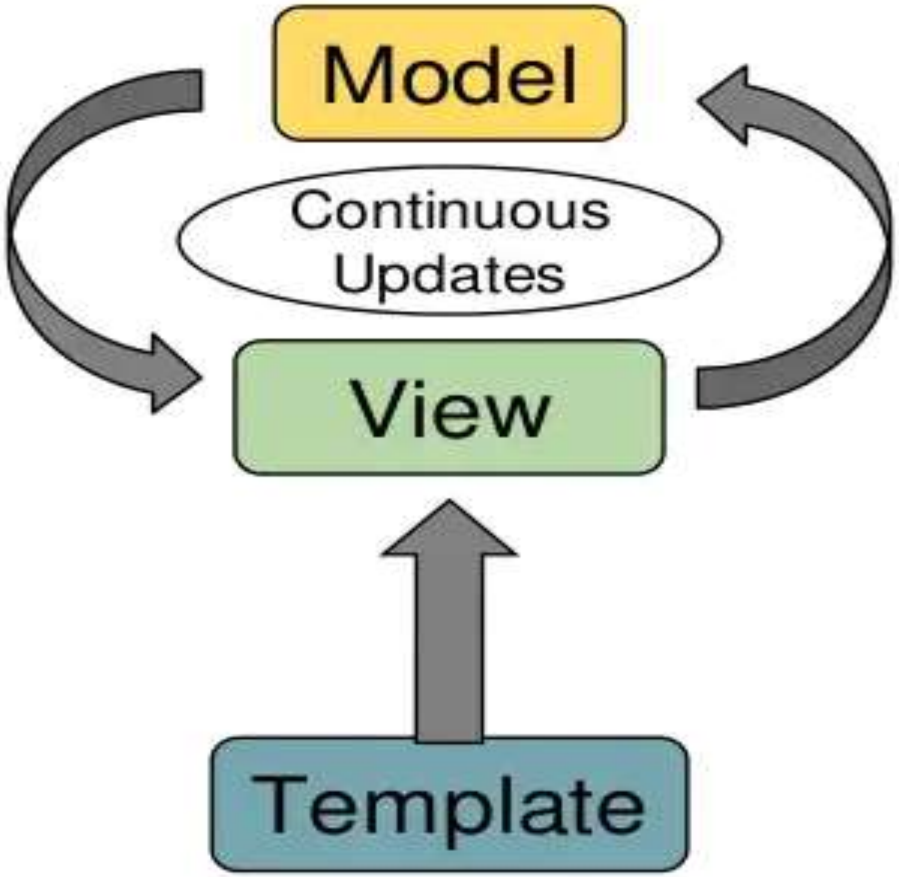
Now lets see the difference between one-way and two-way data bi


Model
Change to the MODEL Updates the VIEW
Change to the VIEW Updates the MODEL
View

1. One Way data binding



Two way data binding



You can also use double braces {{ }} to display content from the model:

Angular dependency injection

What is meant by Angular Dependecy ?

Dependency injection (DI) is a design pattern where objects are passed to another object to complete the tasks. In angular a service or component may require other dependent services to complete a task. Angular uses dependency injection design pattern to fulfil these dependencies. The advantage of dependency injection design pattern is to divide the task among deferent services.

In Angular we specify providers for services using
@Injectable(),
@ngModule() &
@Component() decorators.

Now let us discuss about each decorators in detail

@Injectable(): This is used to inject some sort of service in the angular.

@ngModule(): This is used to construct some new module

@Component(): This is used to specify some components used to execute the application.

Providers in angular dependency injection

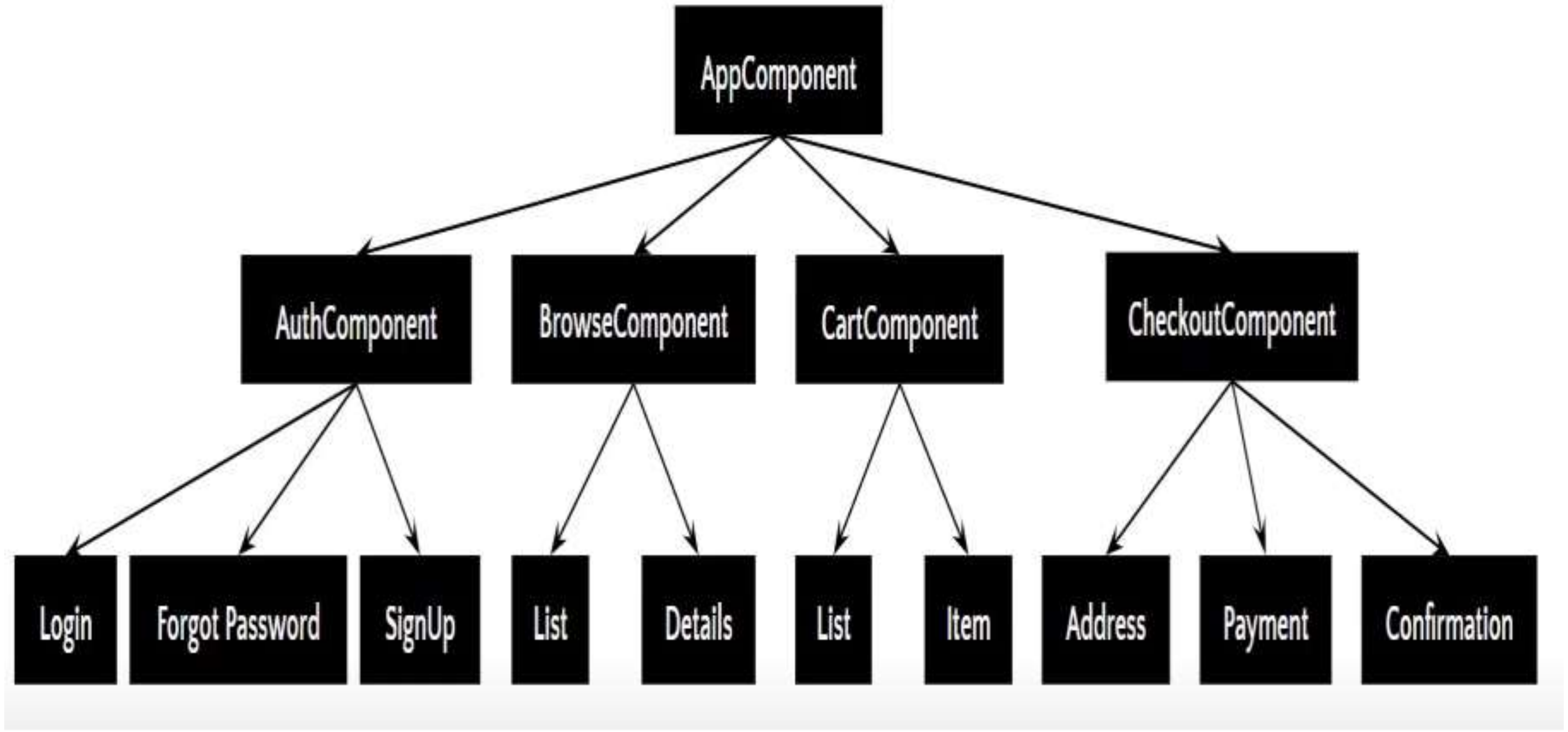
useClass: A class provider that creates and returns new instance of specified class.

useExisting: An alias provider that maps one token to another.

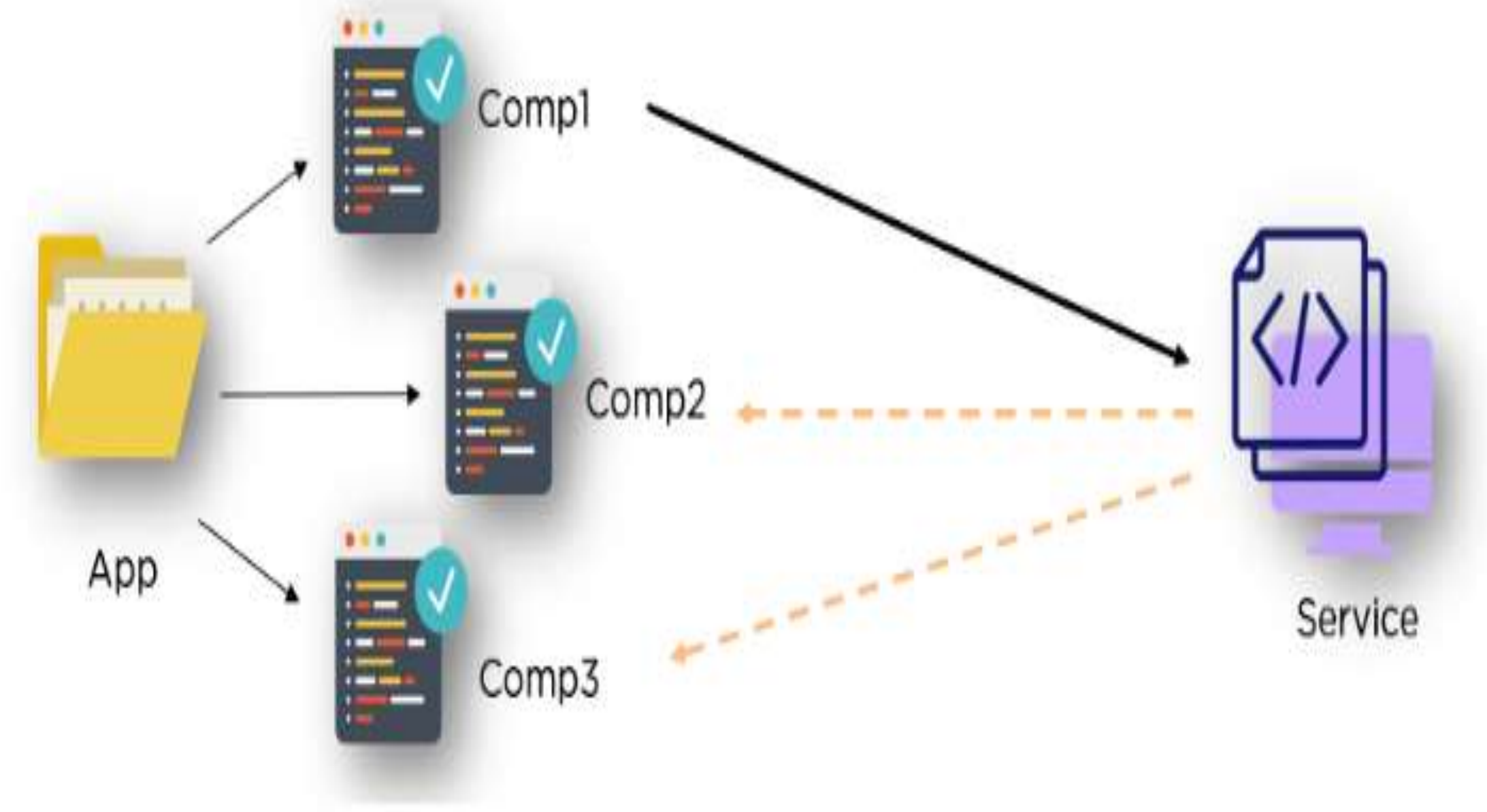
useFactory: Configures a factory provider that returns object for dependency injection.

useValue: A value provider that returns a fixed value for dependency injection.

Some Components in angular



What is meant by service ?



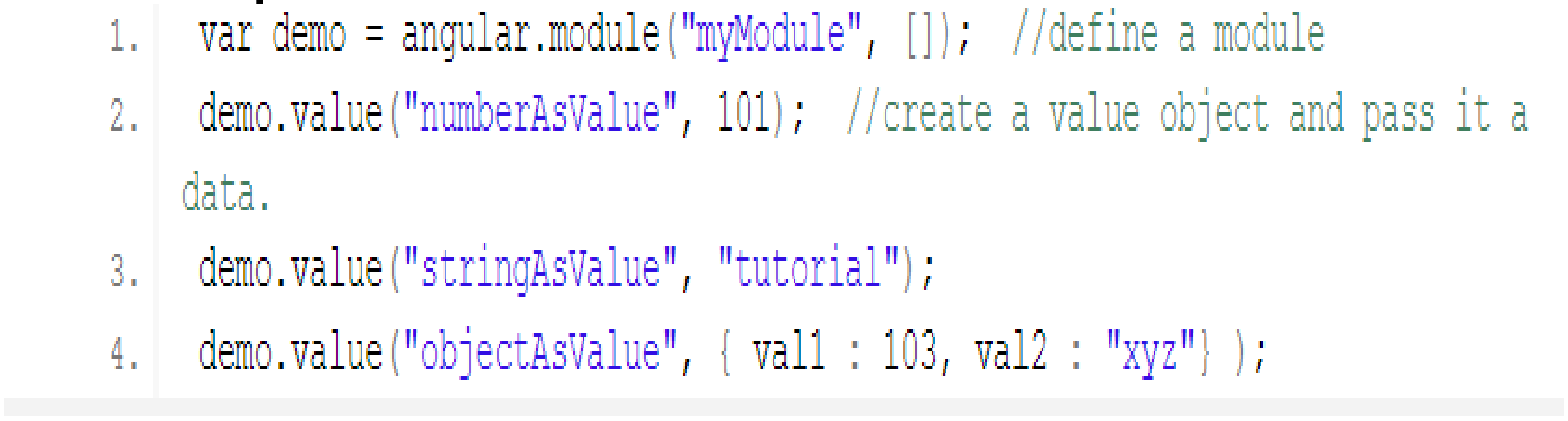
Services

AngularJS supports the concept of **Separation of Concerns** using services architecture. Services are **JavaScript functions**, which are responsible to perform only specific tasks. This makes them individual entities which are maintainable and testable. The controllers and filters can call them on requirement basis. **Services** are normally injected using the **dependency injection** mechanism of AngularJS. There are some components to inject services.

1. Value
2. Factory
3. Service
4. Provider
5. Constant

1) Value

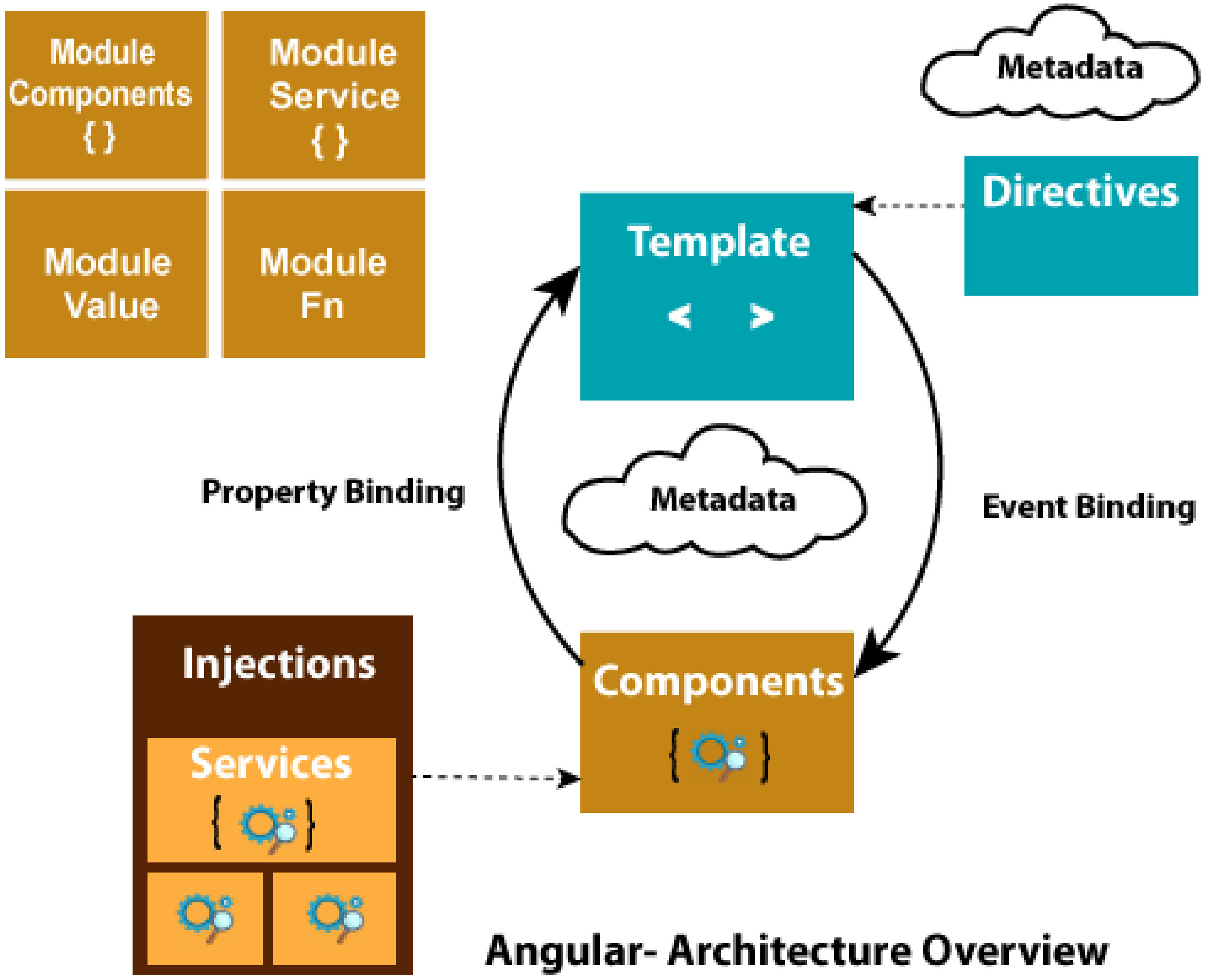
Value is an object. It can be a number, string or javascript object. It is used to pass the value to controller, service or factories in config or run phase.



```
var demo = angular.module("myModule", []);  //define a module
demo.value("numberAsValue", 101);  //create a value object and pass it a
data.
demo.value("stringAsValue", "tutorial");
demo.value("objectAsValue", { val1 : 103, val2 : "xyz"} );
```

Components

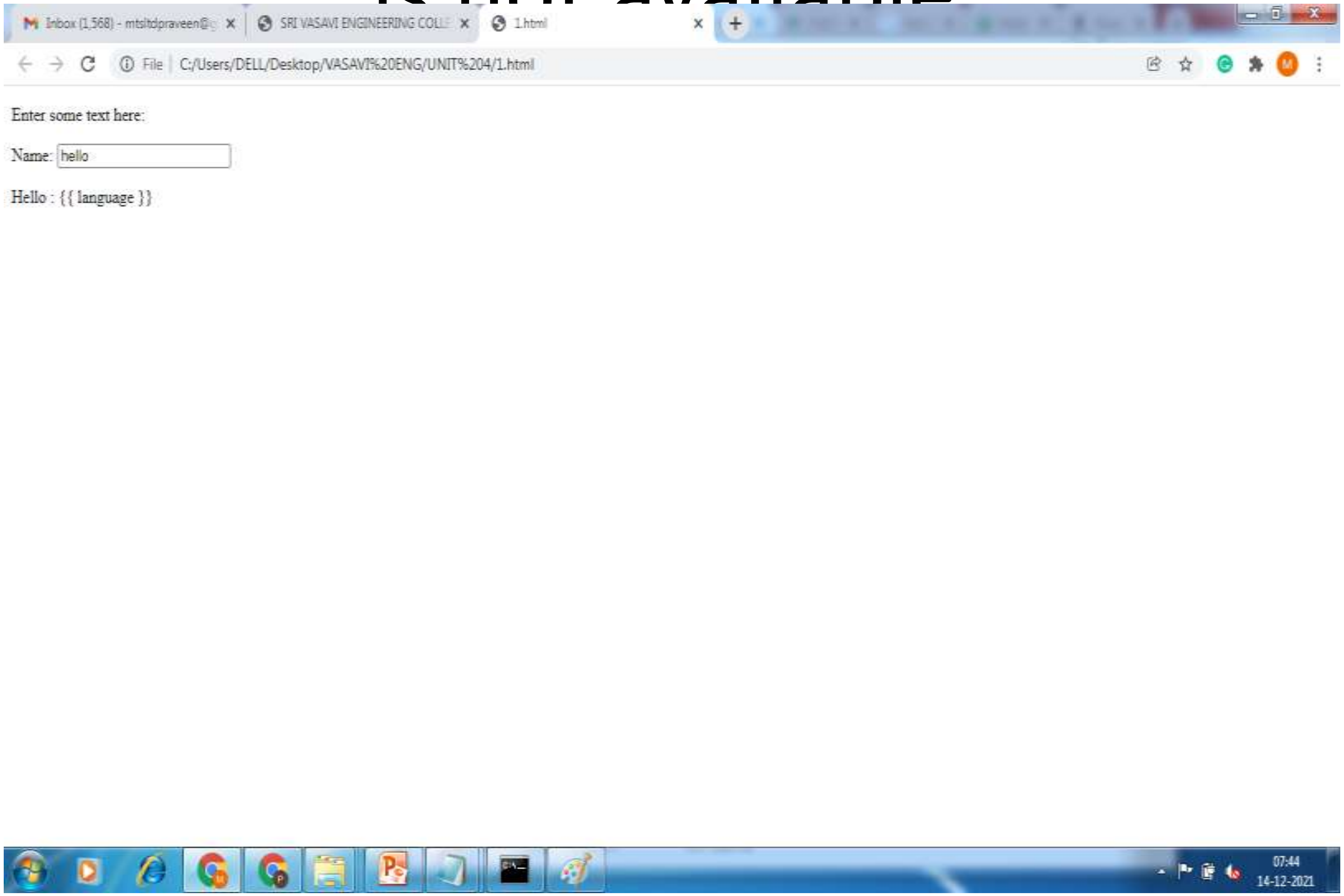
- Templat
  e
- Class
- MetaDa
  ta



Sample Program

```
<!DOCTYPE html>
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.
min.js"></script>
<body>
<div ng-app="" >
<p>Enter Some text here :</p>
<p>Name: <input type="text" ng-model="language"></p>
<p>Hello : {{ language }}</p>
</div>
</body>
</html>
```

Expected Output : When internet is not available



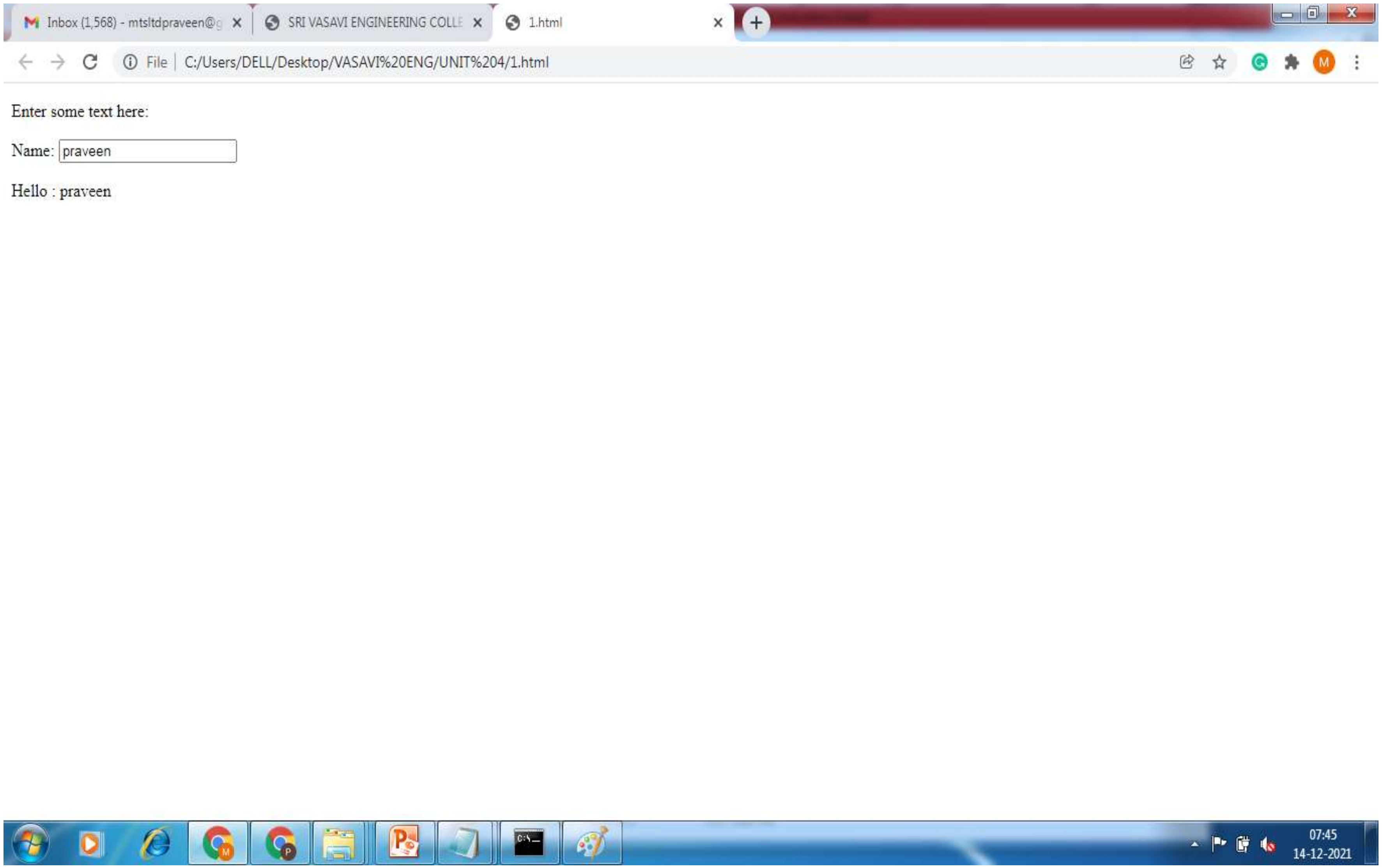
Enter some text here:

Name: hello

Hello : {{ language }}

Inbox (1,568) - mtsltdpraveen@
SRI VASAVI ENGINEERING COLL
1.html
File
C:/Users/DELL/Desktop/VASAVI%20ENG/UNIT%204/1.html
07:44
14-12-2021

When u connect internet



Enter some text here:

Name: praveen

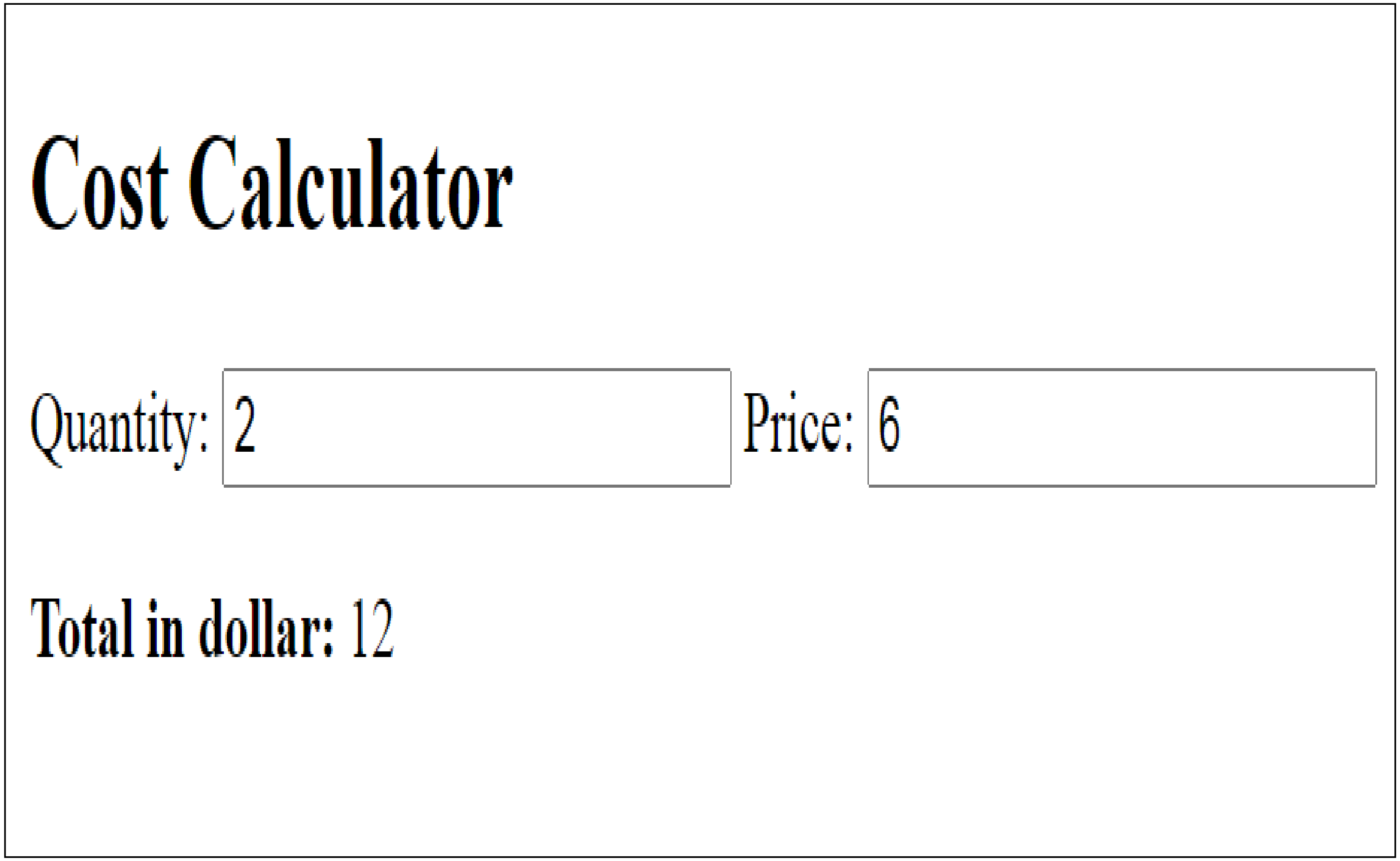
Hello : praveen

Inbox (1,568) - mtsltdpraveen@g
SRI VASAVI ENGINEERING COLLE
1.html
File
C:/Users/DELL/Desktop/VASAVI%20ENG/UNIT%204/1.html
07:45
14-12-2021

Sample application using ng-init directive

```
<!DOCTYPE html>
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.
min.js"></script>
<body>
<div ng-app="" ng-init="quantity=1;price=5">
<h2>Cost Calculator</h2>
Quantity: <input type="number" ng-model="quantity">
Price: <input type="number" ng-model="price">
<p><b>Total in dollar:</b> {{quantity * price}}</p>
</div>
</body>
```

Output


Cost Calculator

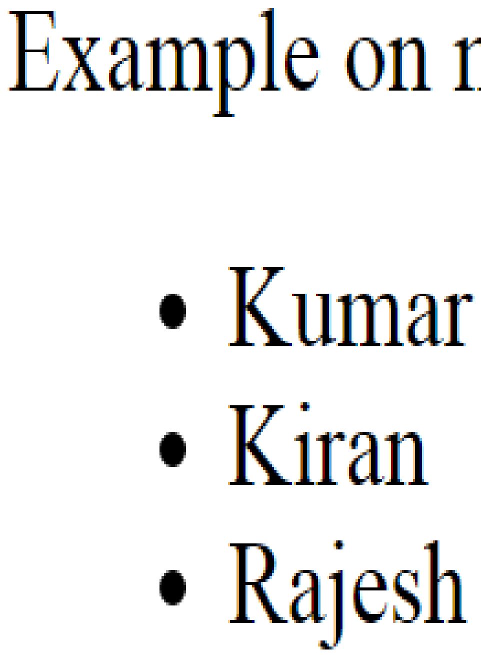
Quantity: 2 Price: 6

Total in dollar: 12

Sample application using ng-repeat directive

```
<!DOCTYPE html>
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.mi
n.js"></script>
<body>
<div ng-app="" ng-init="names=['Kumar','Kiran','Rajesh']">
  <p>Example on ng-repeat:</p>
  <ul>
    <li ng-repeat="x in names">
     {{ x }}
    </li>
  </ul>
```

Output



Example on ng-repeat:

- Kumar
- Kiran
- Rajesh

THANK U

WT LAB 1 to 10 PROGRAMS

Prepare these same program to ur Theory Exams also

Exercise 1:

Design HTML fundamental Concepts.

- Headings
- Links
- Paragraph
- Images
- Tables

1) Headings

Example on Headings

The headings will be declared under <hn> Content to be added here…</hn> , where n can be value range from 1 to 6.

```
<!DOCTYPE html>
    <html>
<head>
<title> My First Example on Headings </title>
</head>
    <body>
    <h1>My First Heading with H1 Size </h1>
<h2>My First Heading with H2 Size </h2>
<h3>My First Heading with H3 Size </h3>
<h4>My First Heading with H4 Size </h4>
<h5>My First Heading with H5 Size </h5>
<h6>My First Heading with H6 Size </h6>
</body>
    </html>
```

2) Links

The HTML <a> tag defines a hyperlink. It has the following syntax:

<a href="*url*">*link text*</a>

Example program

```
<!DOCTYPE html>
<html>
<head>
<title> Example on HTML Links </title>
</head>
    <body>
<h1>HTML Links</h1>
<p><a href="https://www.w3schools.com/">Click Here for More
    Information !</a></p>
</body>
</html>
```

3)Paragraph

<p> is the tag which is used to display the content in paragraph manner < /p>

Example program

```
<!DOCTYPE html>
<html>
<head>
<title> Example on HTML Paragraph </title>
</head>
    <body>
<p> This is one of the main tag
      Which is used to display the content in paragraph manner


                Here we can give either single line or multiple lines and display
    whole content in paragraph
</p>
</body>
</html>
```

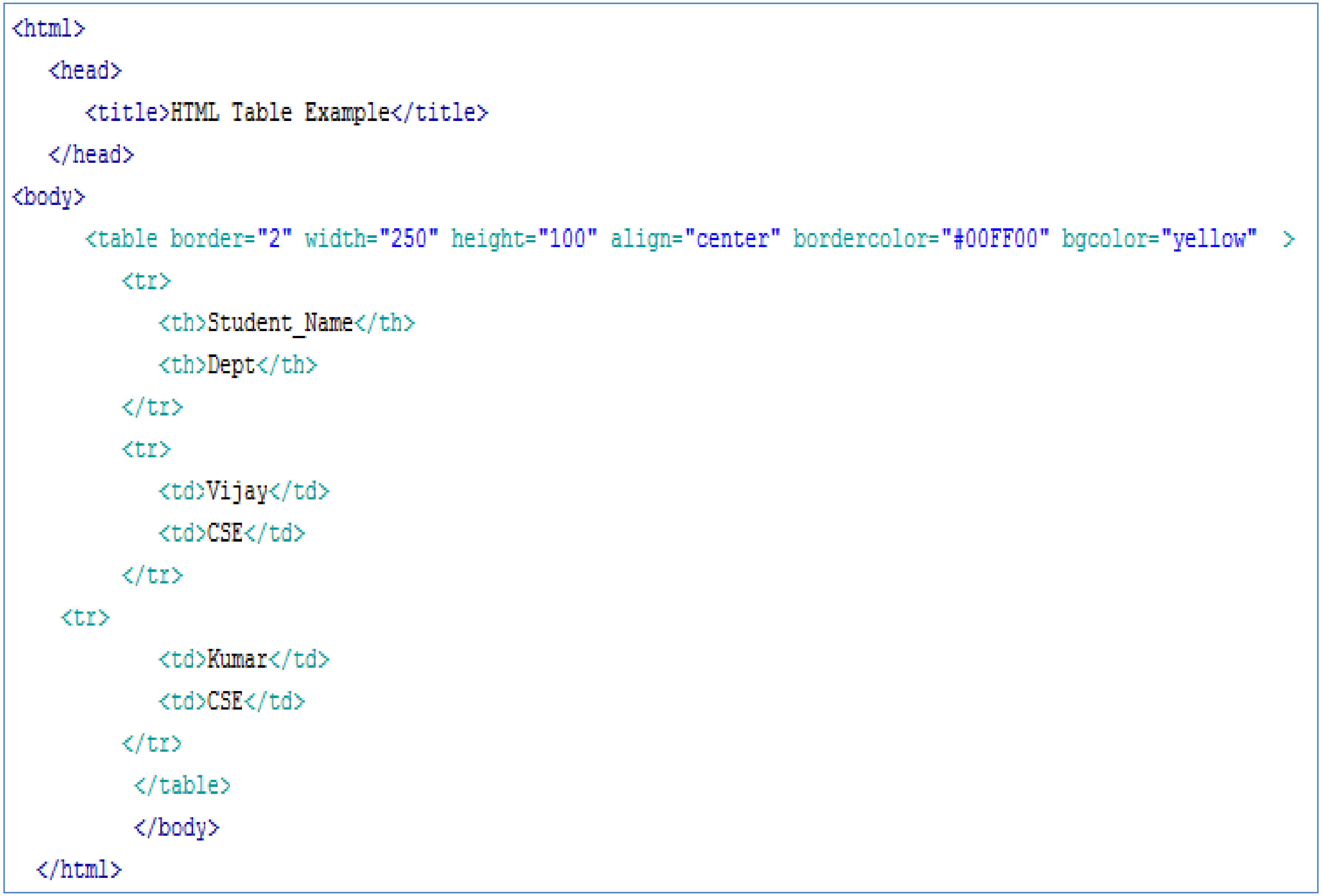
4) Images

<img src= ” is the tag which is used to link images “ >

<u>**Example**</u>

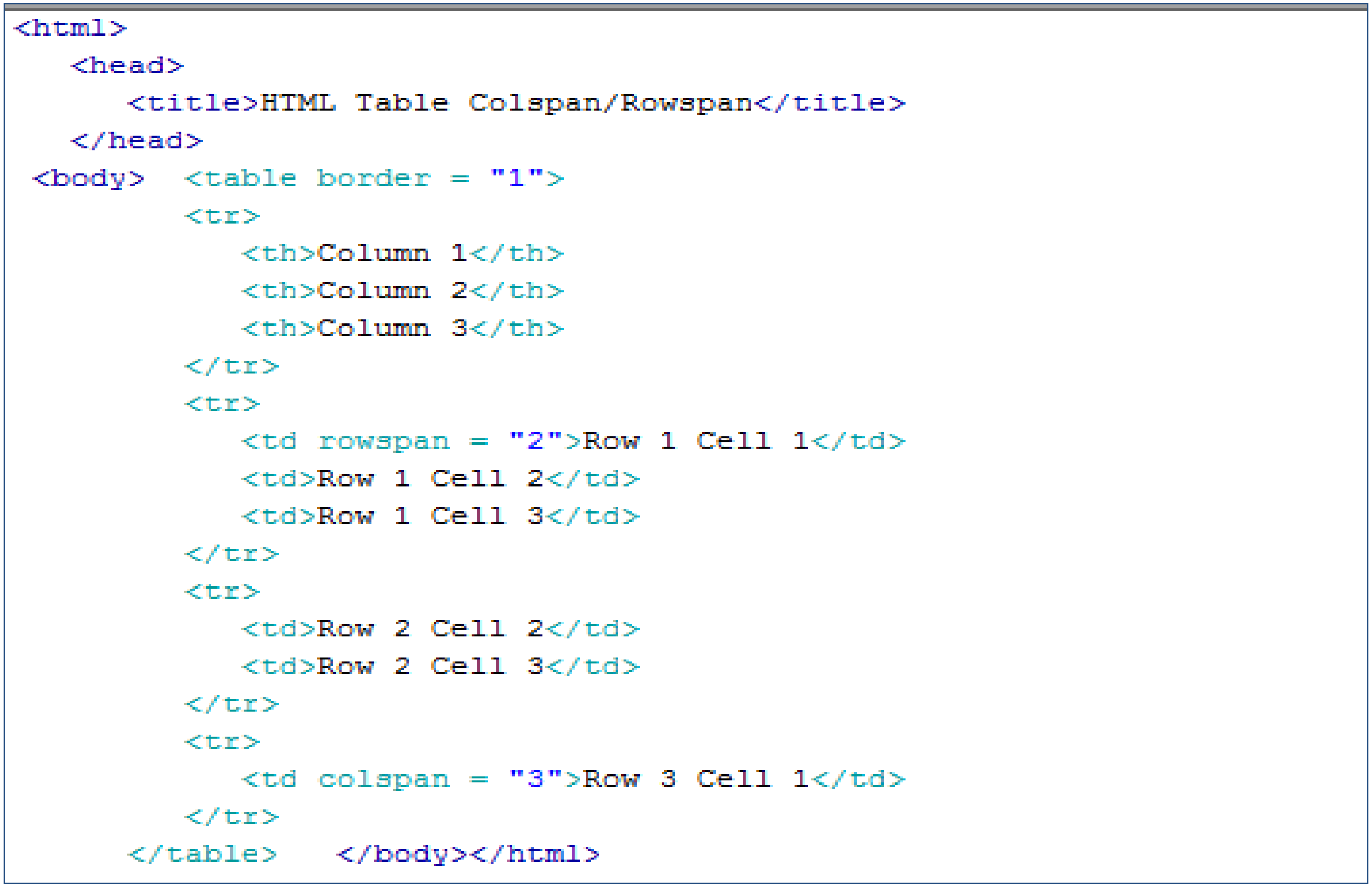
```
<!DOCTYPE html>
<html>
<body>
<h2>HTML Image</h2>
<img src=“img_girl.jpg" alt="Girl in a jacket ”
     width="500" height="600">
 </body>
</html>
```

5) tables



```
<html>
   <head>
      <title>HTML Table Example</title>
   </head>
<body>
      <table border="2" width="250" height="100" align="center" bordercolor="#00FF00" bgcolor="yellow"  >
         <tr>
            <th>Student_Name</th>
            <th>Dept</th>
         </tr>
         <tr>
            <td>Vijay</td>
            <td>CSE</td>
         </tr>
    <tr>
            <td>Kumar</td>
            <td>CSE</td>
         </tr>
          </table>
          </body>
  </html>
```

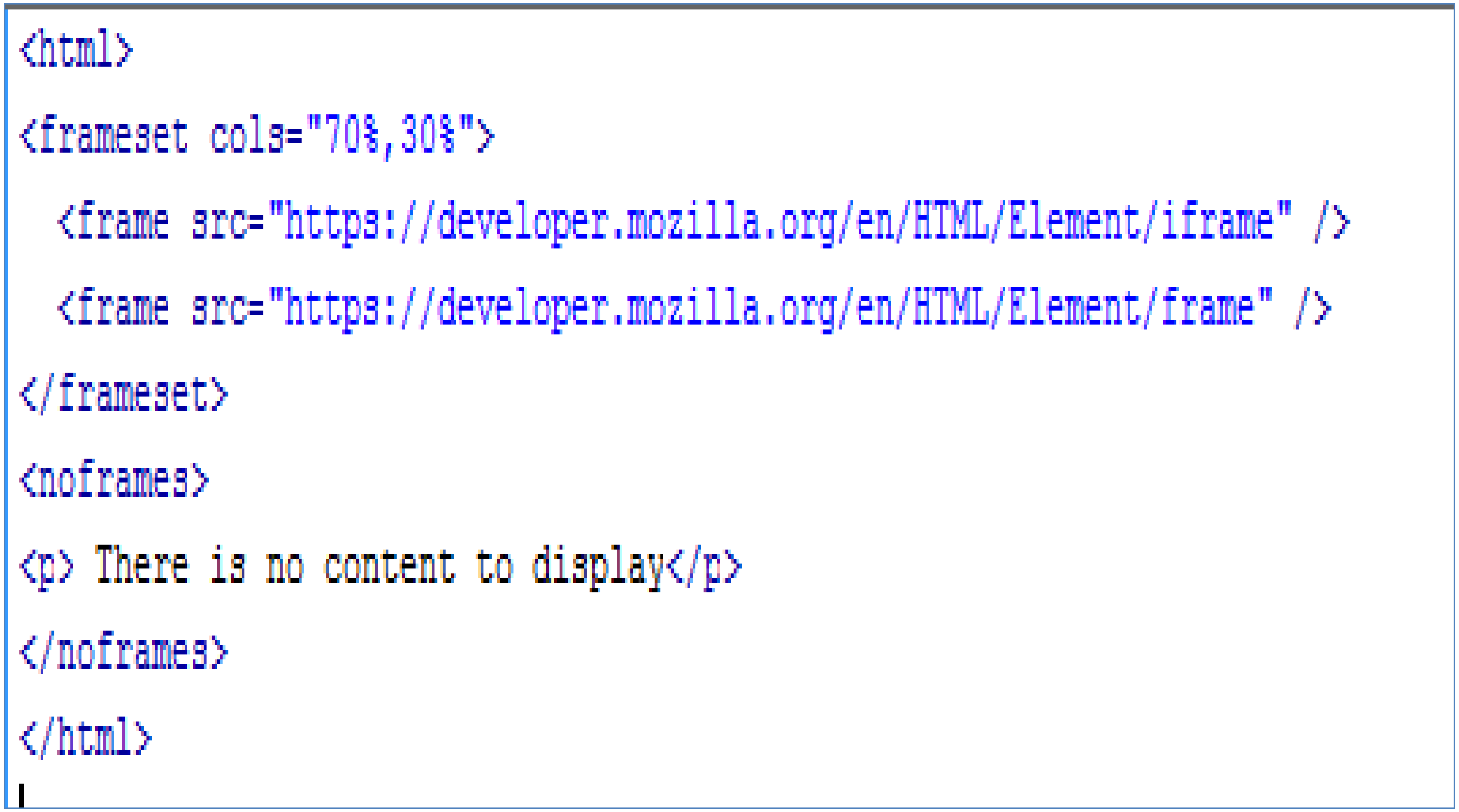
Example on rowspan and colspan



```html
<html>
   <head>
      <title>HTML Table Colspan/Rowspan</title>
   </head>
 <body>  <table border = "1">
         <tr>
            <th>Column 1</th>
            <th>Column 2</th>
            <th>Column 3</th>
         </tr>
         <tr>
            <td rowspan = "2">Row 1 Cell 1</td>
            <td>Row 1 Cell 2</td>
            <td>Row 1 Cell 3</td>
         </tr>
         <tr>
            <td>Row 2 Cell 2</td>
            <td>Row 2 Cell 3</td>
         </tr>
         <tr>
            <td colspan = "3">Row 3 Cell 1</td>
         </tr>
      </table>   </body></html>
```

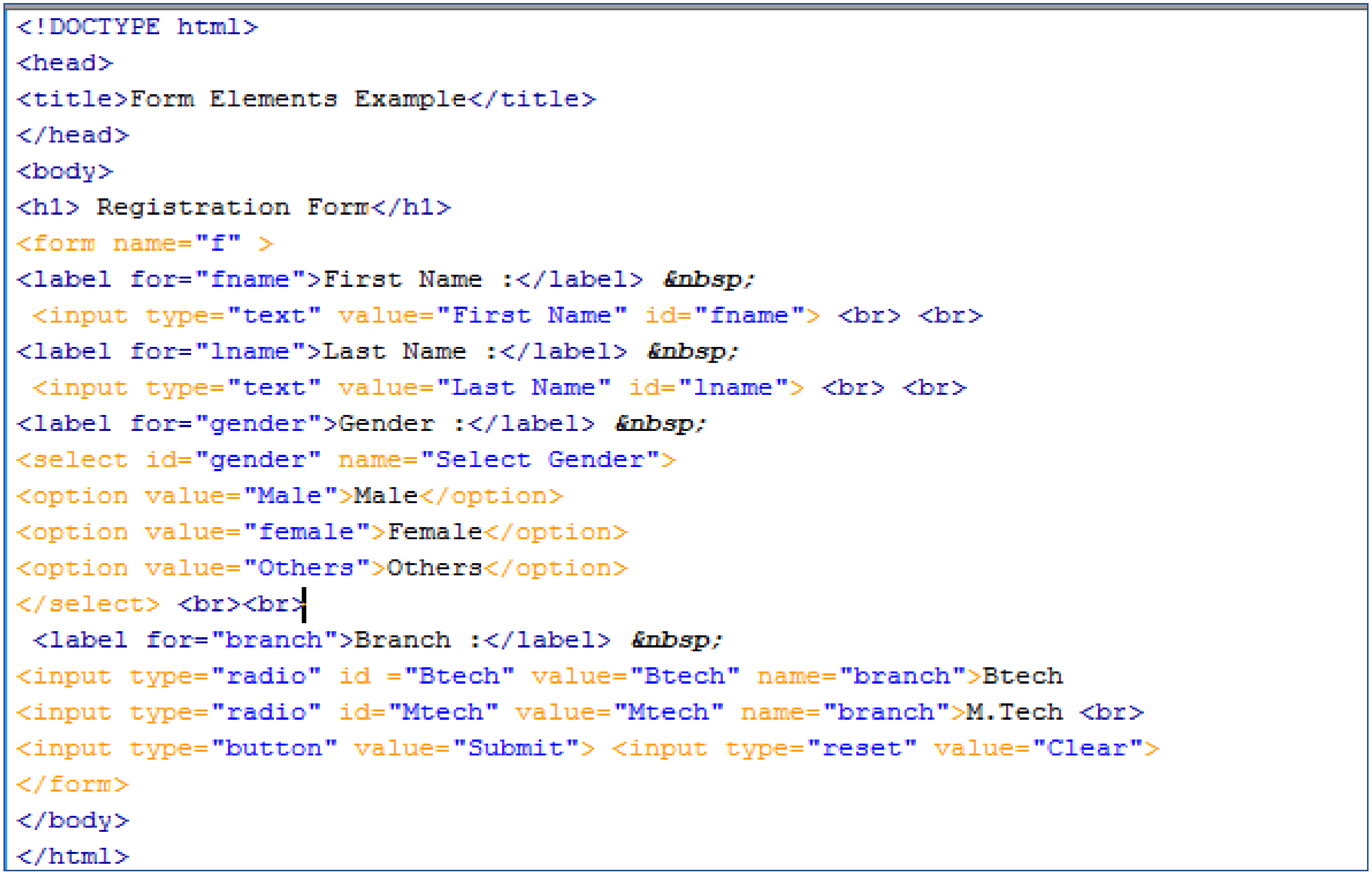
Exercise 2: Design HTML fundamental constructs. (i) Frames (ii) Forms and HTML controls

i)Frames



```
<html>

<frameset cols="70%,30%">

  <frame src="https://developer.mozilla.org/en/HTML/Element/iframe" />

  <frame src="https://developer.mozilla.org/en/HTML/Element/frame" />

</frameset>

<noframes>

<p> There is no content to display</p>

</noframes>

</html>
```

ii) Html Forms



```html
<!DOCTYPE html>
<head>
<title>Form Elements Example</title>
</head>
<body>
<h1> Registration Form</h1>
<form name="f" >
<label for="fname">First Name :</label> &nbsp;
 <input type="text" value="First Name" id="fname"> <br> <br>
<label for="lname">Last Name :</label> &nbsp;
 <input type="text" value="Last Name" id="lname"> <br> <br>
<label for="gender">Gender :</label> &nbsp;
<select id="gender" name="Select Gender">
<option value="Male">Male</option>
<option value="female">Female</option>
<option value="Others">Others</option>
</select> <br><br>
 <label for="branch">Branch :</label> &nbsp;
<input type="radio" id ="Btech" value="Btech" name="branch">Btech
<input type="radio" id="Mtech" value="Mtech" name="branch">M.Tech <br>
<input type="button" value="Submit"> <input type="reset" value="Clear">
</form>
</body>
</html>
```

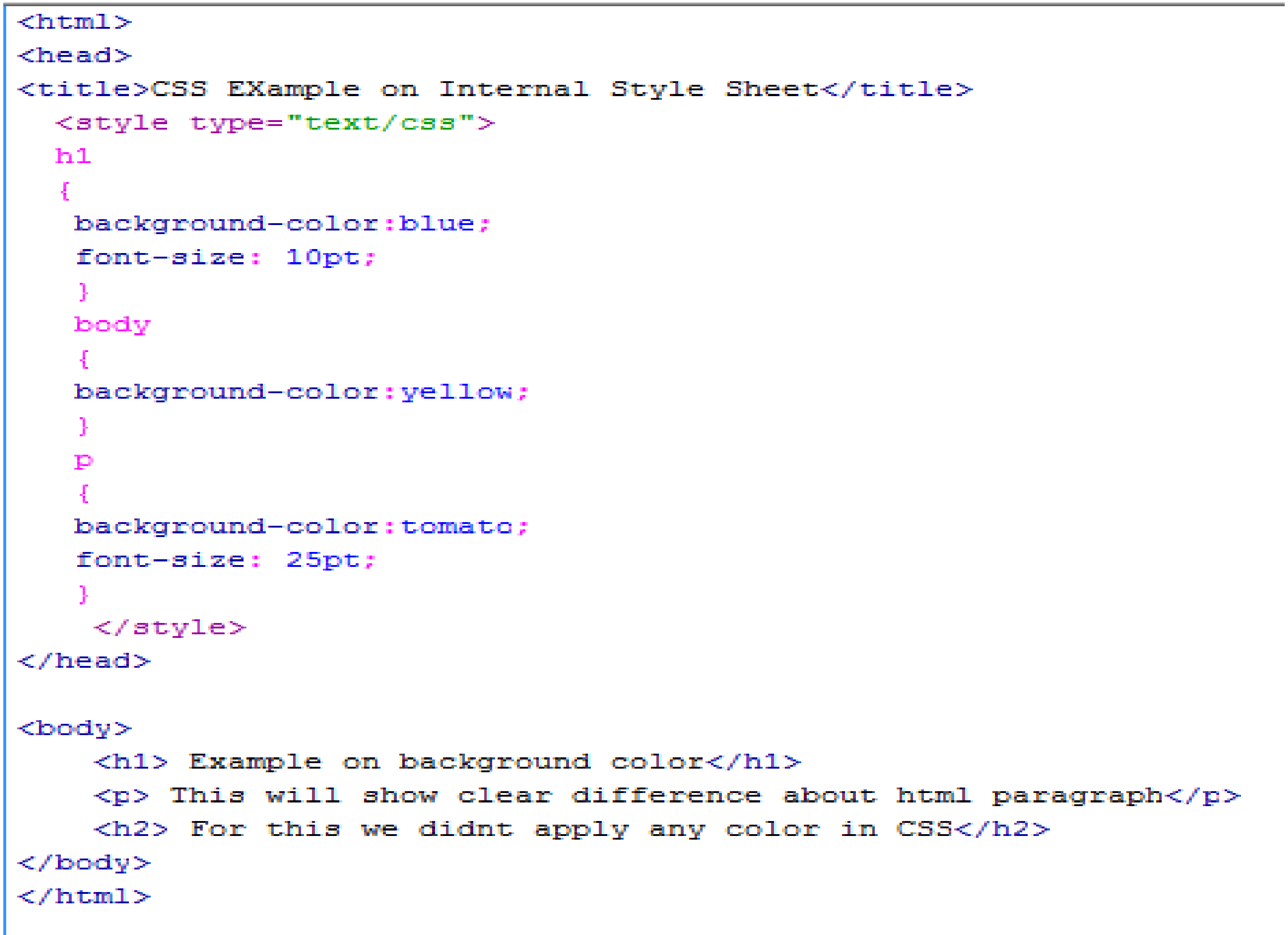
Exercise 3: Design Cascading style sheets

(i) Internal

(ii) External

(iii) Inline

1) Internal Style Sheet

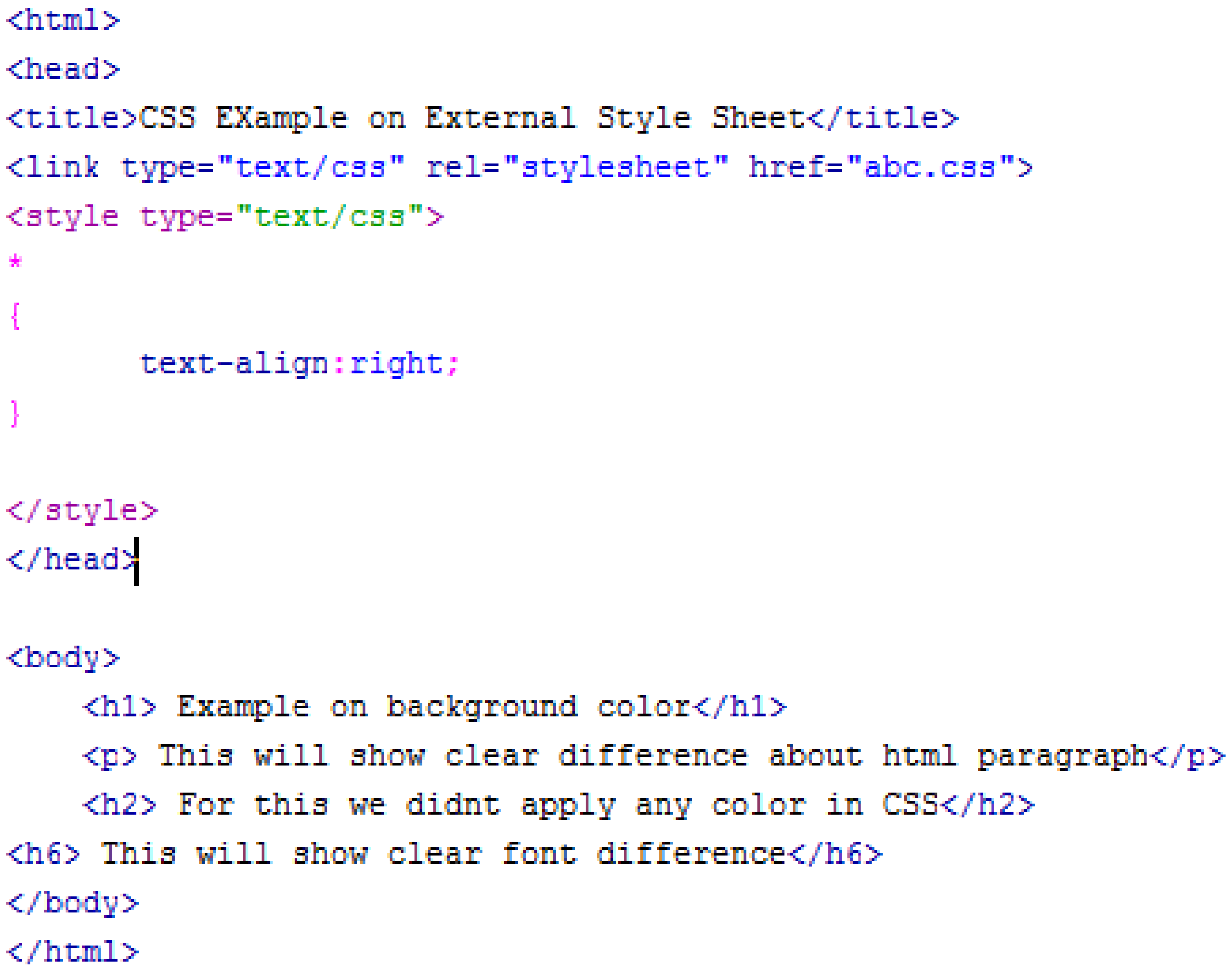


```
<html>
<head>
<title>CSS EXample on Internal Style Sheet</title>
  <style type="text/css">
  h1
  {
   background-color:blue;
   font-size: 10pt;
   }
   body
   {
   background-color:yellow;
   }
   p
   {
   background-color:tomato;
   font-size: 25pt;
   }
    </style>
</head>

<body>
    <h1> Example on background color</h1>
    <p> This will show clear difference about html paragraph</p>
    <h2> For this we didnt apply any color in CSS</h2>
</body>
</html>
```

2)Inline style sheet

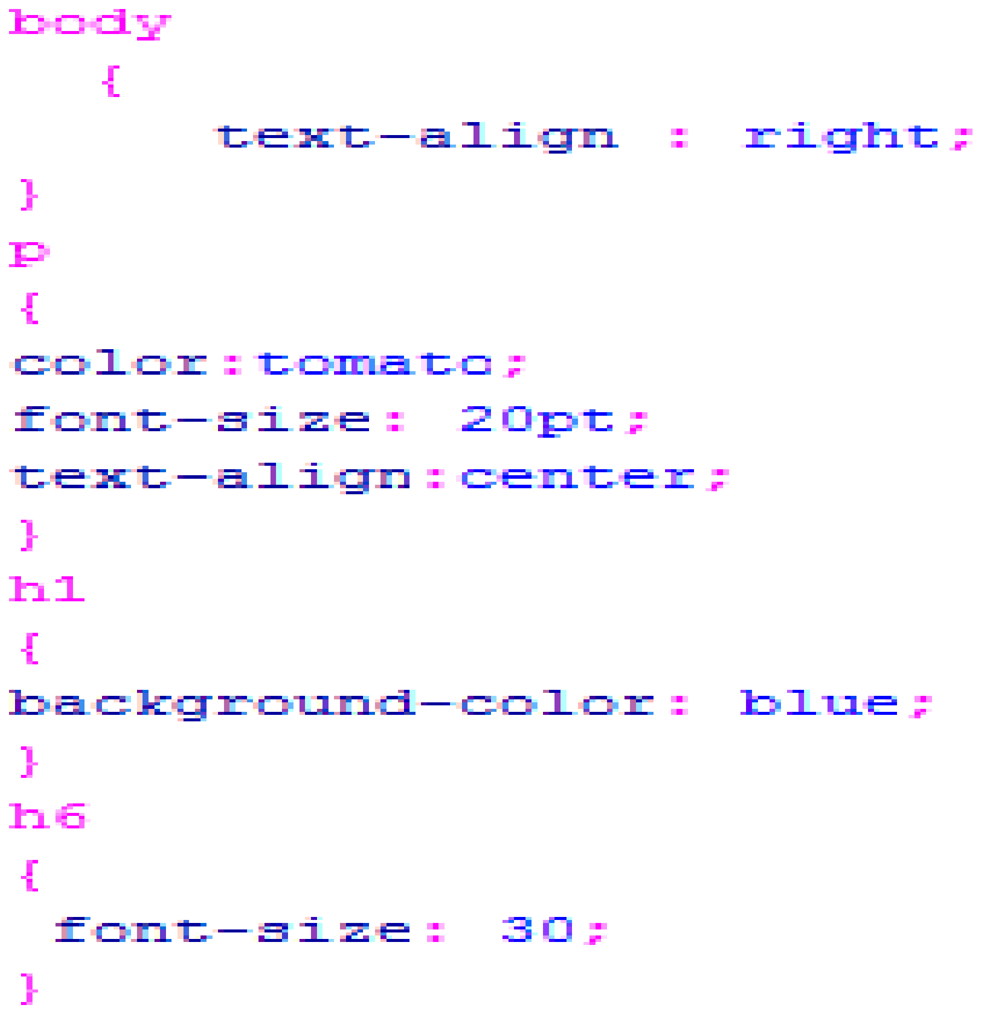
```
<!DOCTYPE html>
  <html>
  <body>
  <h1 style="color:blue;text-align:center;">
  This is a heading</h1>
  <p style="color:red;">This is a paragraph.</p>
  </body>
  </html>
```

3)External Style Sheet(External.html)



```
<html>
<head>
<title>CSS EXample on External Style Sheet</title>
<link type="text/css" rel="stylesheet" href="abc.css">
<style type="text/css">
*
{
        text-align:right;
}

</style>
</head>

<body>
    <h1> Example on background color</h1>
    <p> This will show clear difference about html paragraph</p>
    <h2> For this we didnt apply any color in CSS</h2>
<h6> This will show clear font difference</h6>
</body>
</html>
```

abc.css file



```
body
    {
         text-align  :  right;
}
p
{
color:tomato;
font-size:  20pt;
text-align:center;
}
h1
{
background-color:  blue;
}
h6
{
 font-size:  30;
}
```

Element Selectors

```
<!DOCTYPE html>
<html>
<head>
<style>
p.intro {
  background-color: yellow;
}</style>
</head>
<body>
<h1>Demo of the element.class selector</h1>
<div class="intro">
  <p>My name is Donald.</p>
  <p>I live in Duckburg.</p>
</div>
<p>My best friend is Mickey.</p>
<p class="intro">My best friend is Mickey.</p></body></html>
```

Exercise 4

Exercise 4:

Write an XML file which will display the Book information which includes the following: (i) Title of the book (ii) Author Name (iii) ISBN number (iv) Publisher name (v) Edition (vi) Price

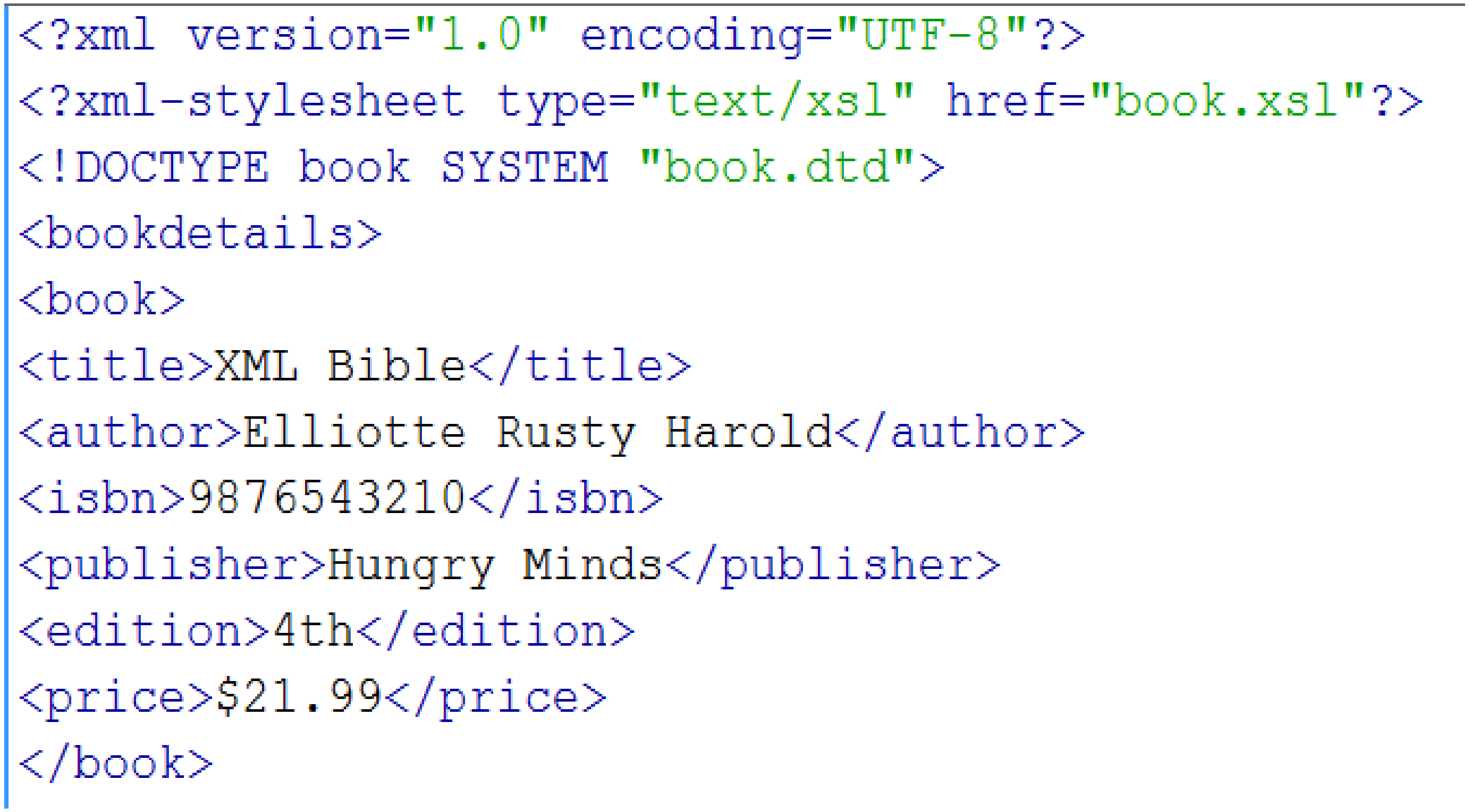
(a) Write a Document Type Definition (DTD) to validate the above XML file.

(b) Write a XML Schema Definition (XSD)

Sample DTD for Book

```
<!ELEMENT bookdetails (book+)>
<!ELEMENT book (title,author,isbn,publisher,edition,price)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT edition (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

Book.xml



```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="book.xsl"?>
<!DOCTYPE book SYSTEM "book.dtd">
<bookdetails>
<book>
<title>XML Bible</title>
<author>Elliotte Rusty Harold</author>
<isbn>9876543210</isbn>
<publisher>Hungry Minds</publisher>
<edition>4th</edition>
<price>$21.99</price>
</book>
```

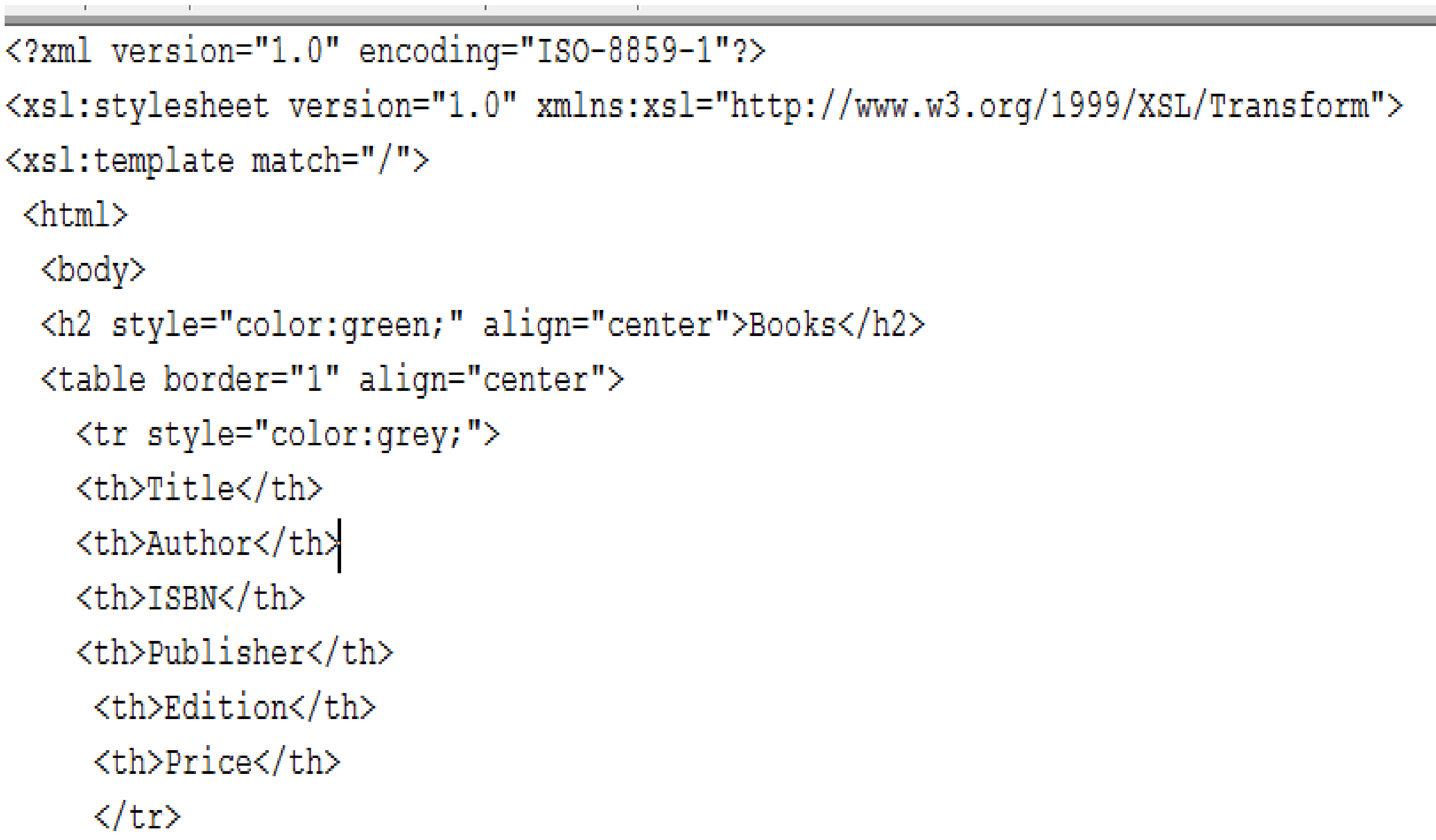
Contd..

```
<book>
<title>AI: A Modern Approach</title>
<author>Stuart J. Russell</author>
<isbn>9876543220</isbn>
<publisher>Princeton Hall</publisher>
<edition>6th</edition>
<price>$36.09</price>
</book>
<book>
<title>Beginning Java 2</title>
<author>Ivor Horton</author>
<isbn>9876543220</isbn>
<publisher>wrox</publisher>
<edition>3th</edition>
<price>$8.95</price>
</book>
```

Contd..

```xml
<book>
<title>HTML5: Up and Running</title>
<author>Mark Pilgrim</author>
<isbn>1234567890</isbn>
<publisher>O'REILLY</publisher>
<edition>1st</edition>
<price>$17.99</price>
</book>
</bookdetails>
```

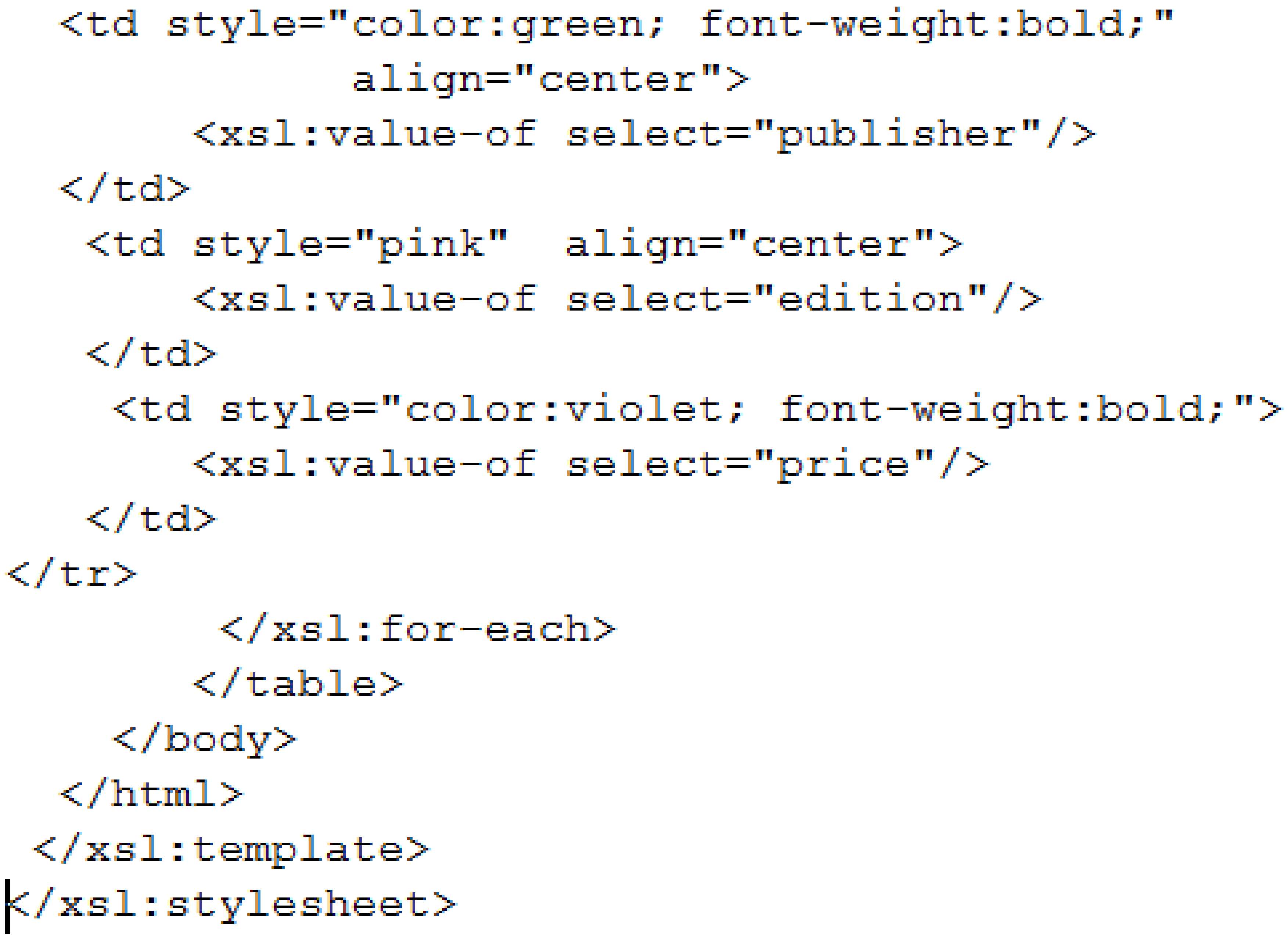
book.xsl



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
 <html>
  <body>
  <h2 style="color:green;" align="center">Books</h2>
  <table border="1" align="center">
    <tr style="color:grey;">
    <th>Title</th>
    <th>Author</th>
    <th>ISBN</th>
    <th>Publisher</th>
     <th>Edition</th>
     <th>Price</th>
     </tr>
```

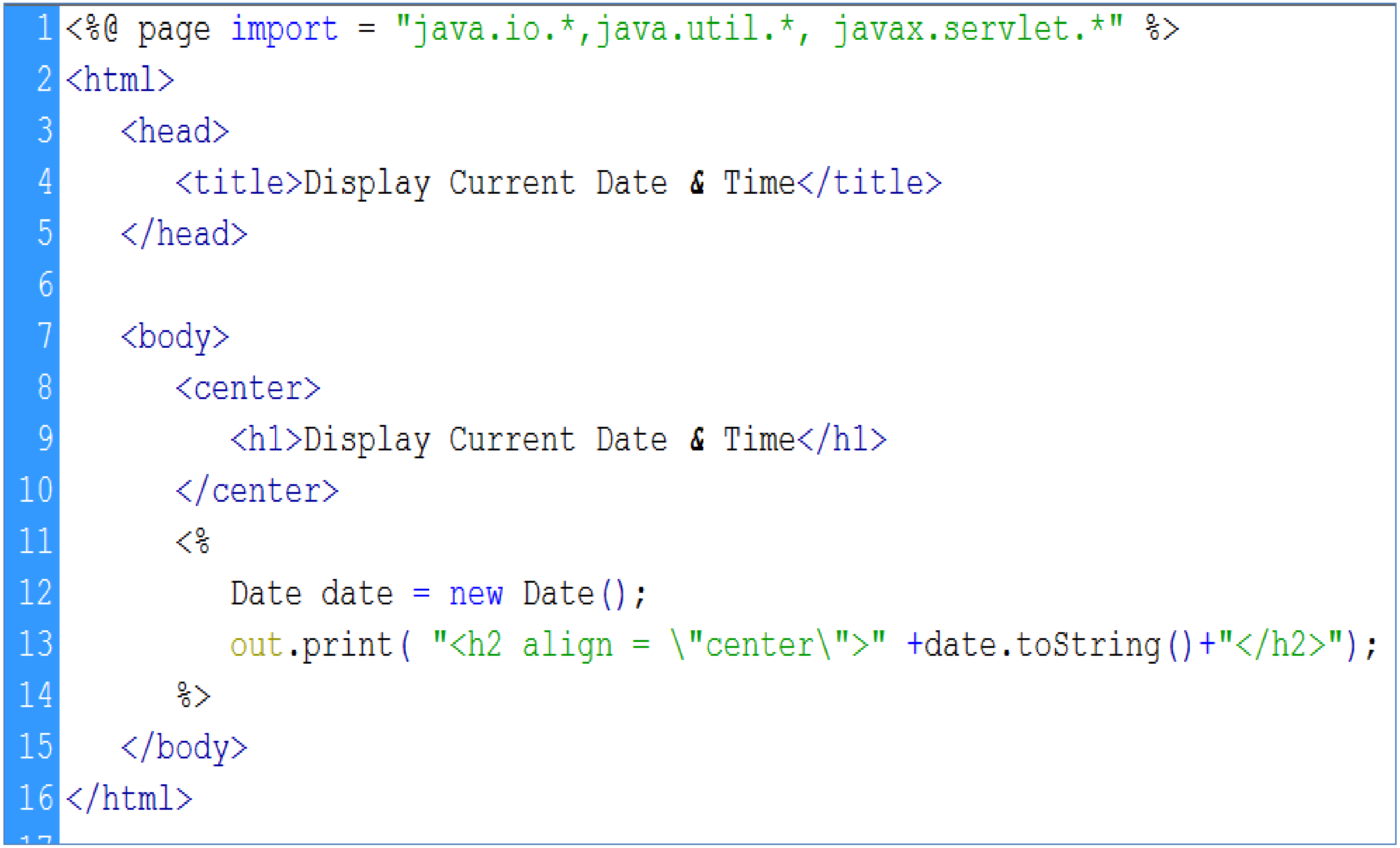
Contd...

```
    <xsl:for-each select="bookdetails/book">
   <tr style="width:70%">

   <td style="font-family:'Comic Sans MS';
              color:red;width:70%">
       <xsl:value-of select="title"/>
  </td>

  <td style="text-transform: capitalize;
        font-weight:bold;" align="center">
      <xsl:value-of select="author"/>
  </td>

  <td style="color:blue">
       <xsl:value-of select="isbn"/>
  </td>
```



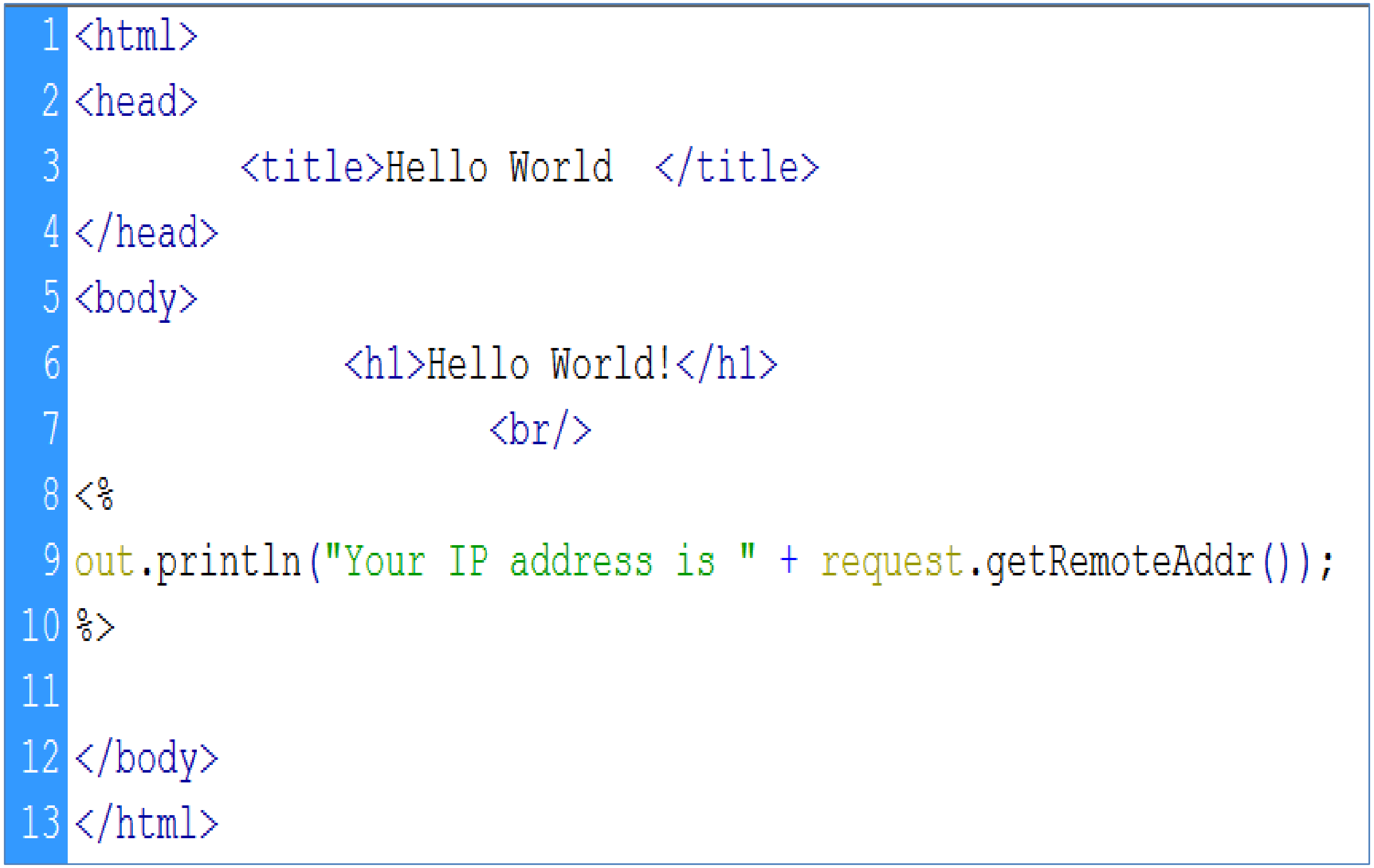
```
  <td style="color:green; font-weight:bold;"
             align="center">
       <xsl:value-of select="publisher"/>
  </td>
   <td style="pink"  align="center">
       <xsl:value-of select="edition"/>
   </td>
    <td style="color:violet; font-weight:bold;">
       <xsl:value-of select="price"/>
   </td>
</tr>
        </xsl:for-each>
       </table>
    </body>
  </html>
 </xsl:template>
</xsl:stylesheet>
```

Lab Exercise 5:Create a simple JSP to print the current Date and Time.



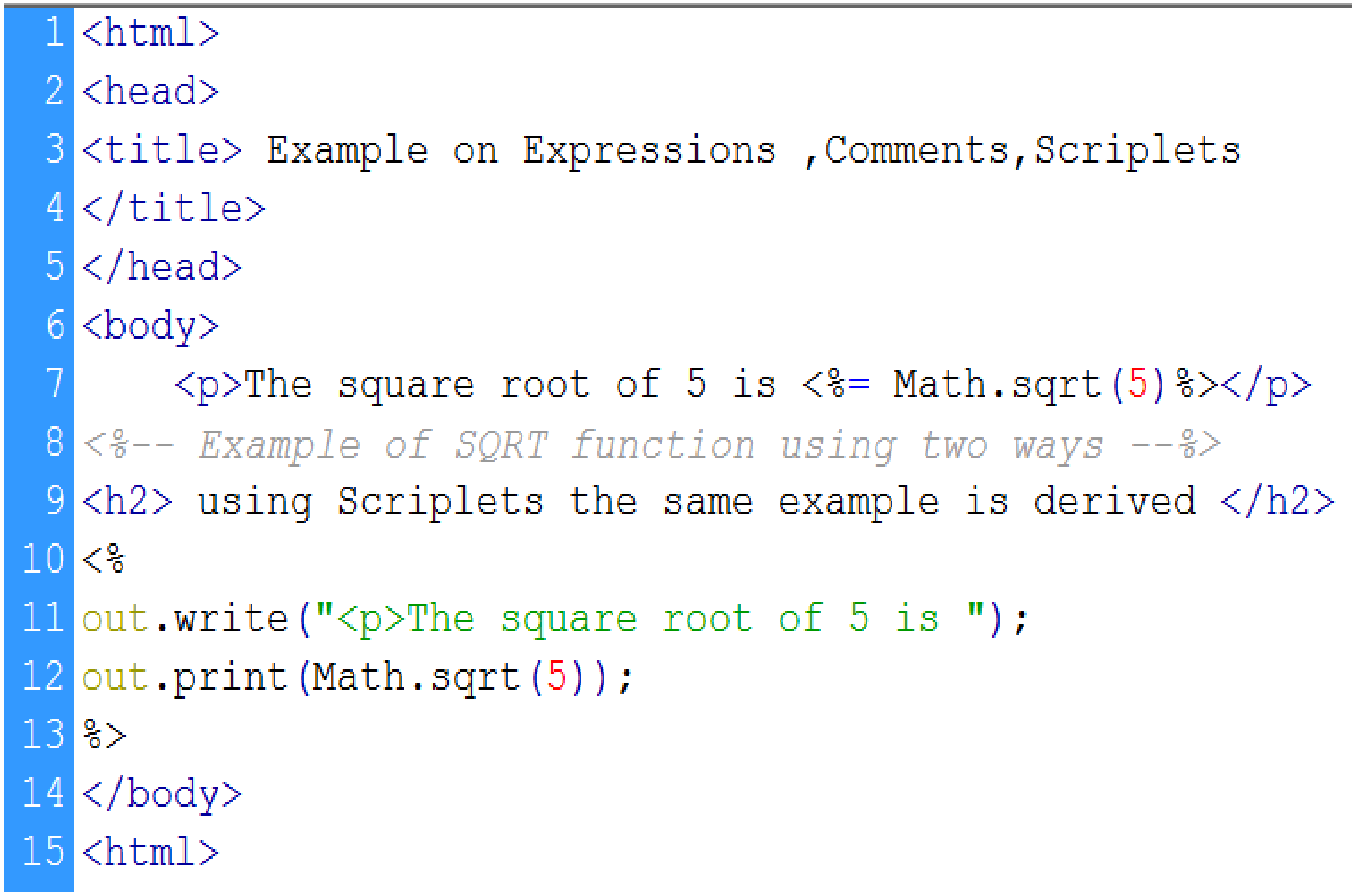
```
<%@ page import = "java.io.*,java.util.*, javax.servlet.*" %>
<html>
   <head>
      <title>Display Current Date & Time</title>
   </head>

   <body>
      <center>
         <h1>Display Current Date & Time</h1>
      </center>
      <%
         Date date = new Date();
         out.print( "<h2 align = \"center\">" +date.toString()+"</h2>");
      %>
   </body>
</html>
```

Lab Exercise 6:Create a simple JSP a) Display current IP-Address of the System b) Find out the Square root for a Number



```
<html>
<head>
        <title>Hello World  </title>
</head>
<body>
             <h1>Hello World!</h1>
                    <br/>
<%
out.println("Your IP address is " + request.getRemoteAddr());
%>

</body>
</html>
```

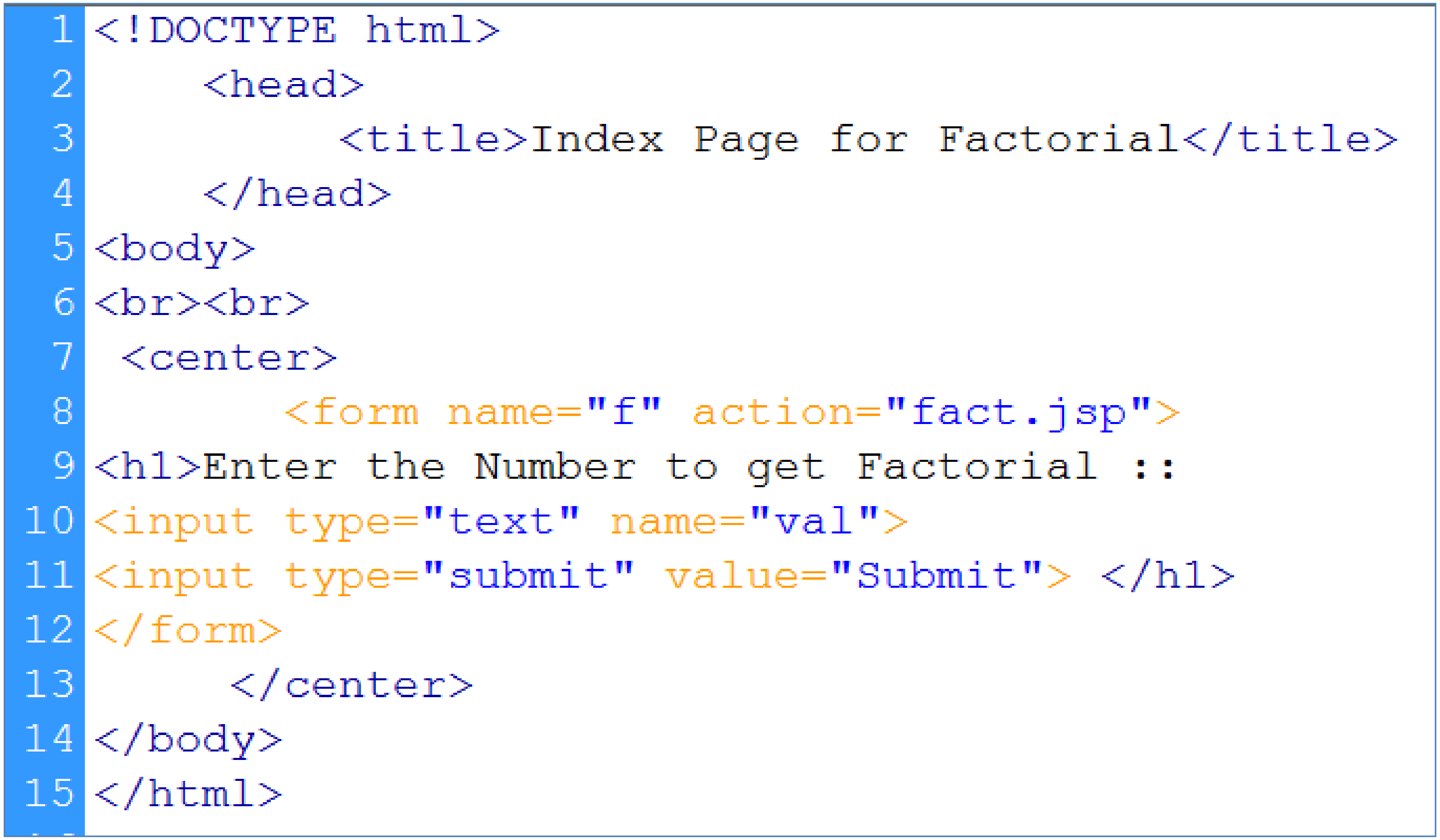
6 b) Find out the Square root for a Number



```
<html>
<head>
<title> Example on Expressions ,Comments,Scriplets
</title>
</head>
<body>
    <p>The square root of 5 is <%= Math.sqrt(5)%></p>
<%-- Example of SQRT function using two ways --%>
<h2> using Scriplets the same example is derived </h2>
<%
out.write("<p>The square root of 5 is ");
out.print(Math.sqrt(5));
%>
</body>
<html>
```

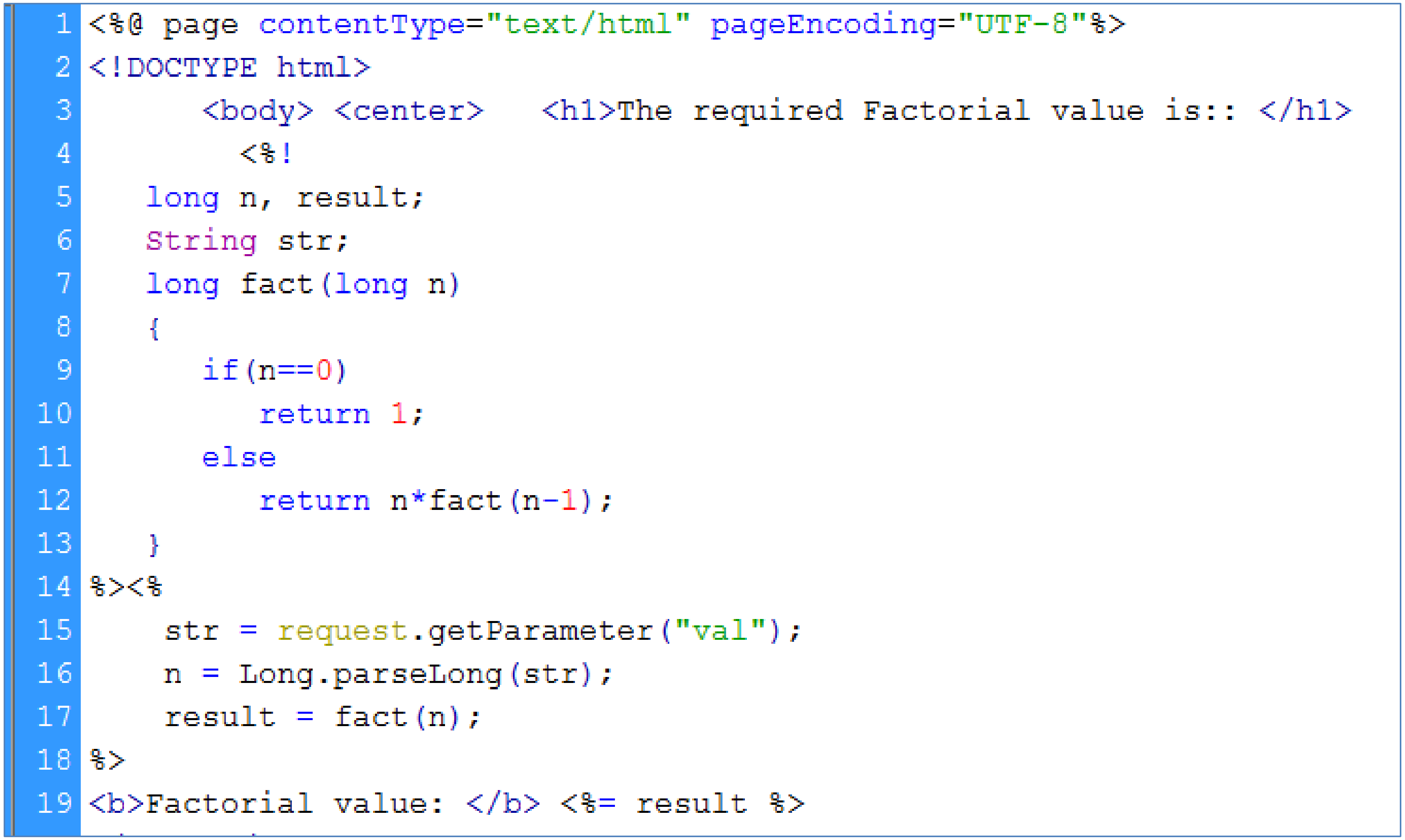
Lab Exercise 7:Develop JSP program calculates factorial values for an integer number, while the input is taken from an HTML form.

Index.jsp



```
<!DOCTYPE html>
    <head>
         <title>Index Page for Factorial</title>
    </head>
<body>
<br><br>
 <center>
       <form name="f" action="fact.jsp">
<h1>Enter the Number to get Factorial ::
<input type="text" name="val">
<input type="submit" value="Submit"> </h1>
</form>
     </center>
</body>
</html>
```

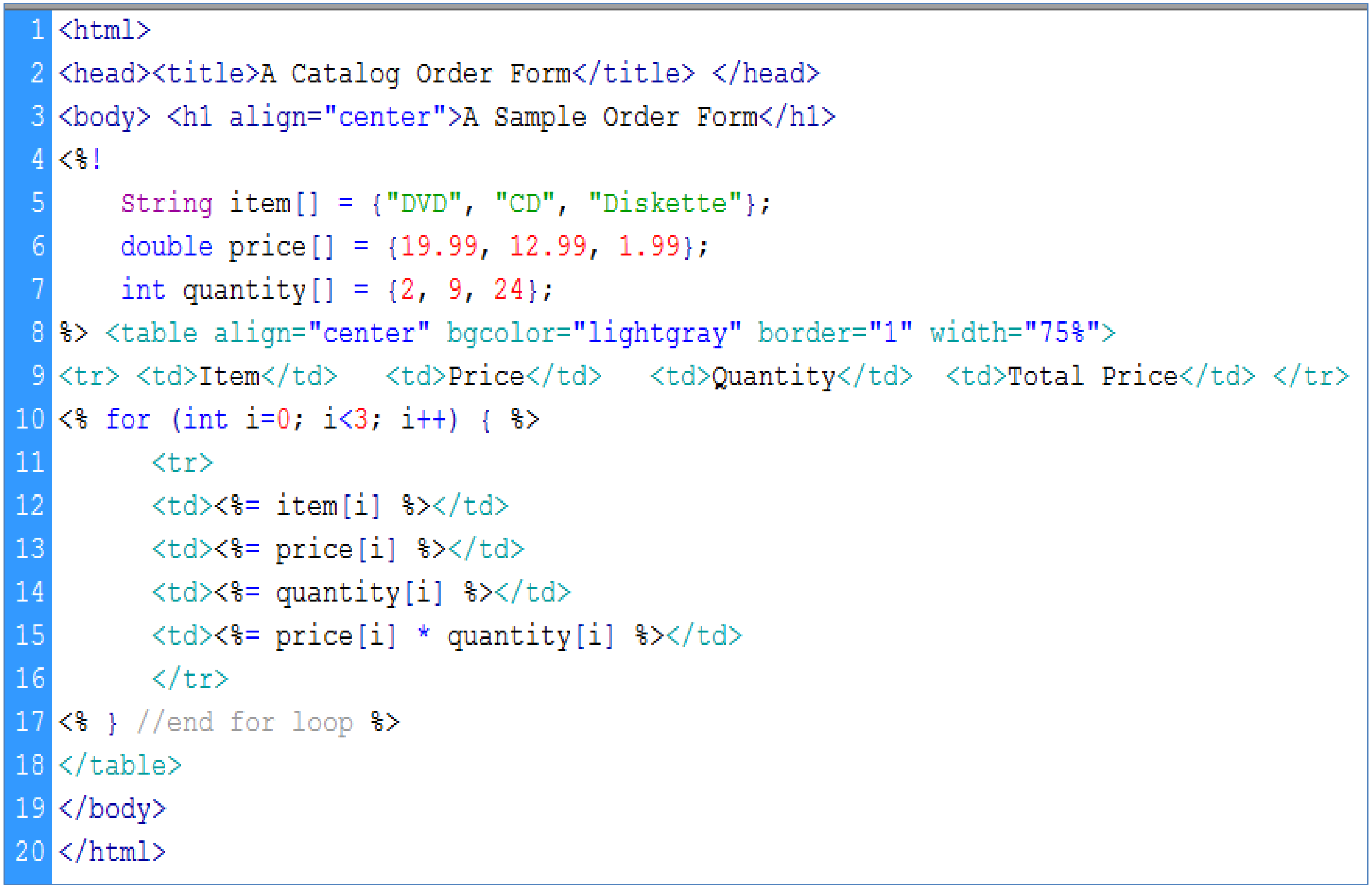
Exercise 7: Develop JSP program calculates factorial values for an integer number, while the input is taken from an HTML form.



```
<%@ page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
      <body> <center>   <h1>The required Factorial value is:: </h1>
        <%!
   long n, result;
   String str;
   long fact(long n)
   {
      if(n==0)
         return 1;
      else
         return n*fact(n-1);
   }
%><%
    str = request.getParameter("val");
    n = Long.parseLong(str);
    result = fact(n);
%>
<b>Factorial value: </b> <%= result %>
```

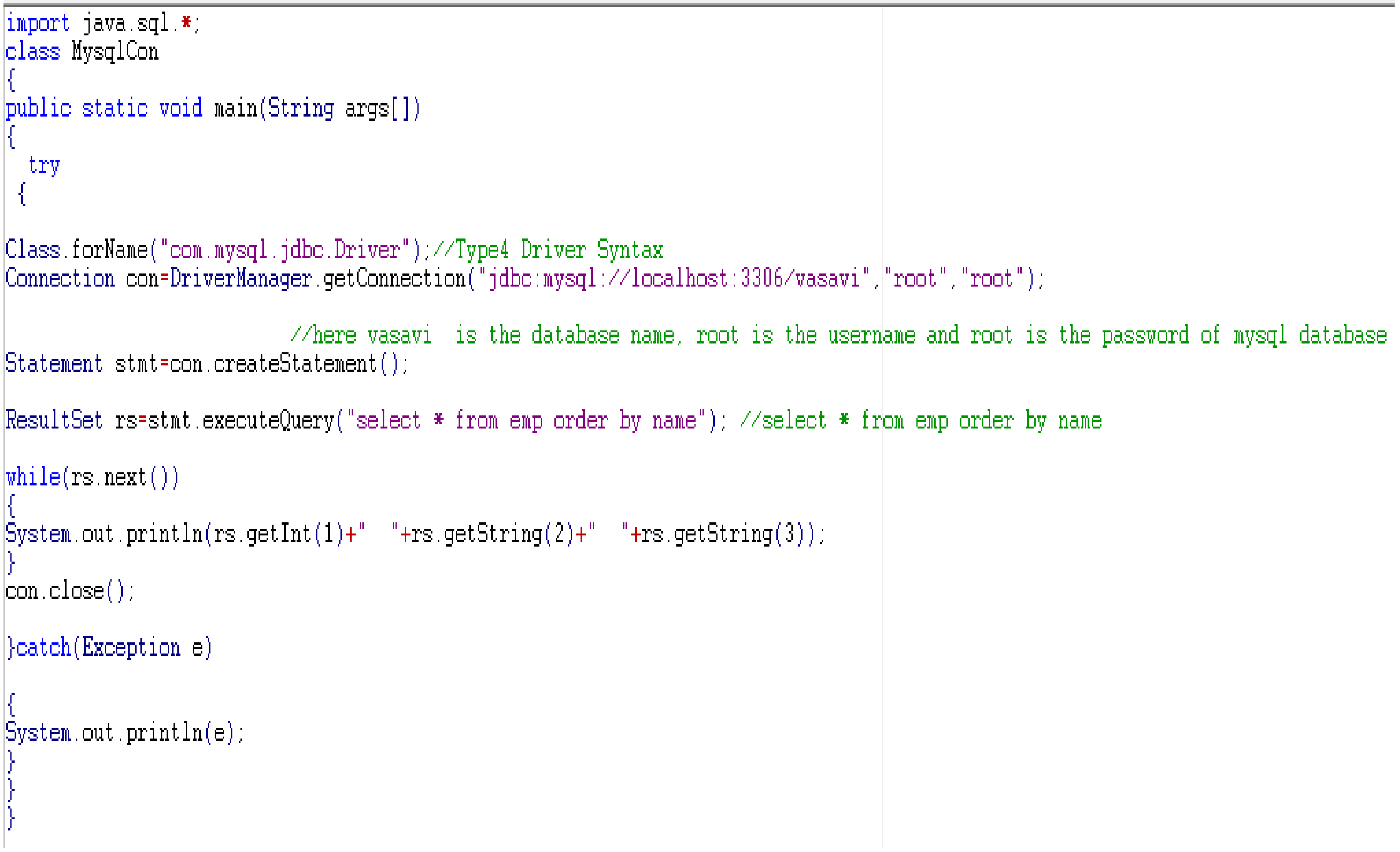
Exercise 8:Develop JSP program shows a Sample Order Form

Catalog.jsp



```
<html>
<head><title>A Catalog Order Form</title> </head>
<body> <h1 align="center">A Sample Order Form</h1>
<%!
    String item[] = {"DVD", "CD", "Diskette"};
    double price[] = {19.99, 12.99, 1.99};
    int quantity[] = {2, 9, 24};
%> <table align="center" bgcolor="lightgray" border="1" width="75%">
<tr> <td>Item</td>   <td>Price</td>   <td>Quantity</td>  <td>Total Price</td> </tr>
<% for (int i=0; i<3; i++) { %>
      <tr>
      <td><%= item[i] %></td>
      <td><%= price[i] %></td>
      <td><%= quantity[i] %></td>
      <td><%= price[i] * quantity[i] %></td>
      </tr>
<% } //end for loop %>
</table>
</body>
</html>
```

Exercise 9:
a) MYSQL connectivity using Type 4 Driver for Data Retrieval



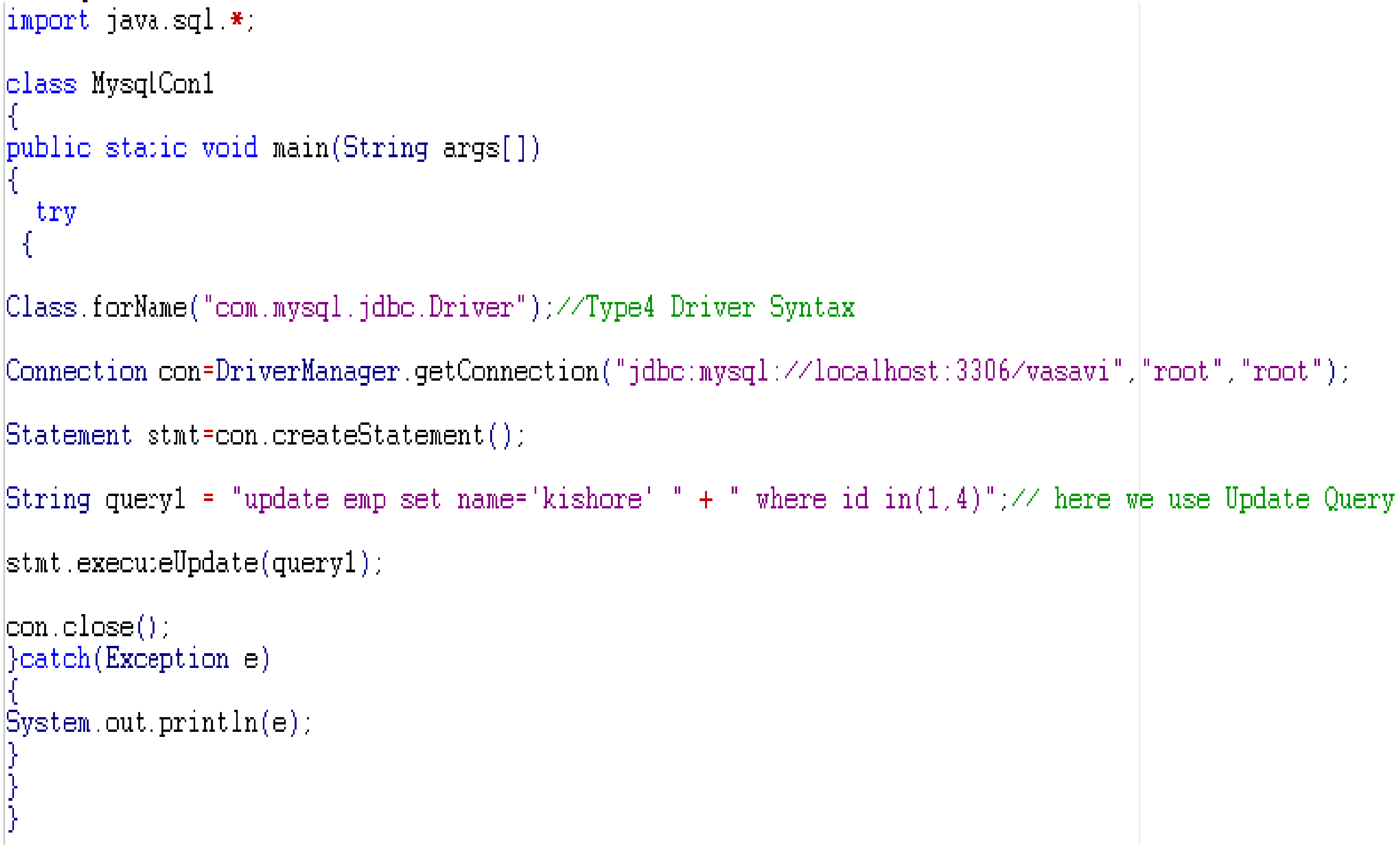
```
import java.sql.*;
class MysqlCon
{
public static void main(String args[])
{
  try
 {

Class.forName("com.mysql.jdbc.Driver");//Type4 Driver Syntax
Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/vasavi","root","root");

                          //here vasavi  is the database name, root is the username and root is the password of mysql database
Statement stmt=con.createStatement();

ResultSet rs=stmt.executeQuery("select * from emp order by name"); //select * from emp order by name

while(rs.next())
{
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
}
con.close();

}catch(Exception e)

{
System.out.println(e);
}
}
}
```

Before executing the program,just try to create database with following tables

```
create database vasavi;
use vasavi;
create table emp(id int(10),name varchar(40),age int(3));
insert into emp values(1,'praveen',31);
insert into emp values(2,'Kumar',30);
insert into emp values(3,'Sowmya',29);
```

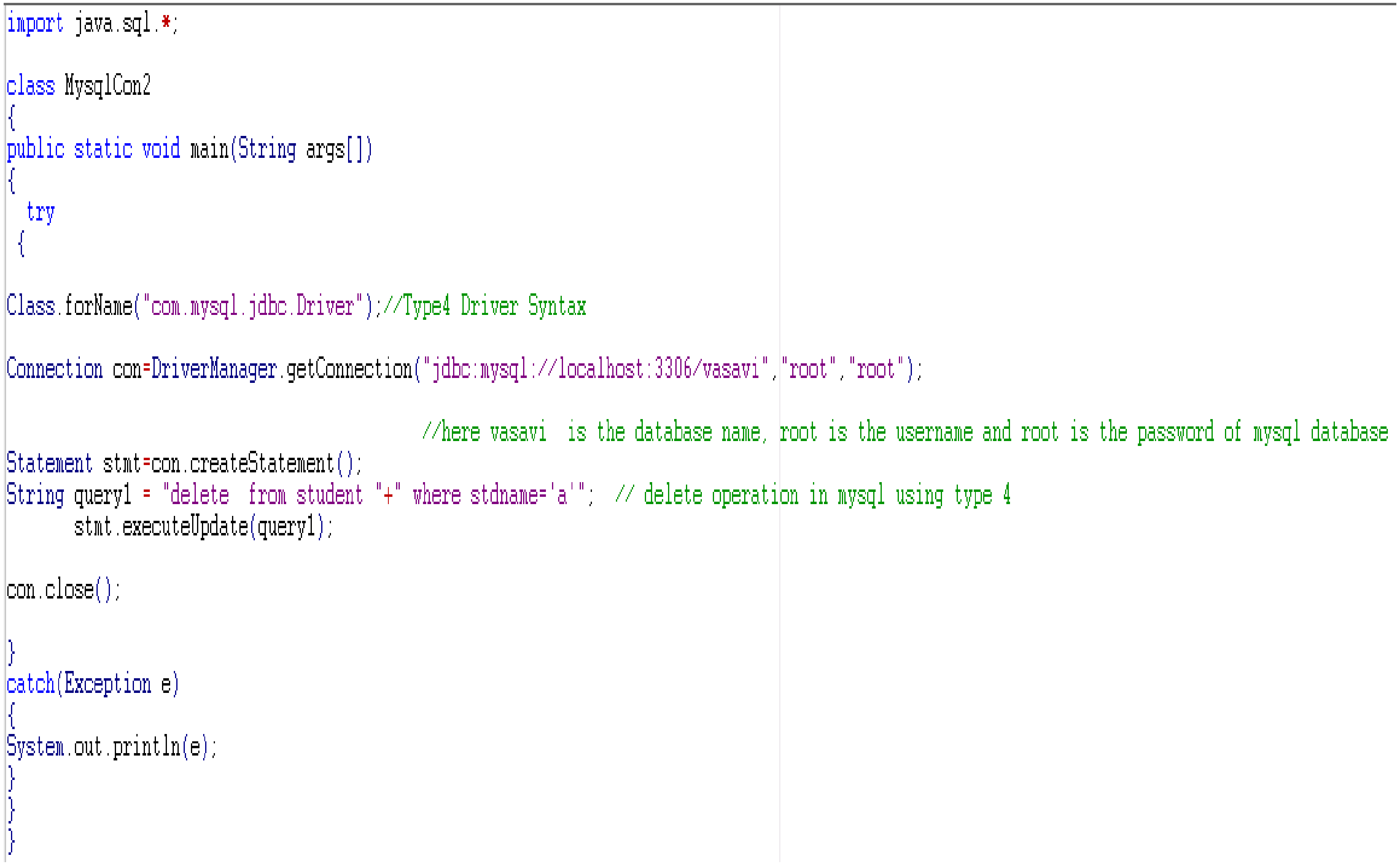
Exercise 9:

b) MYSQL connectivity using Type 4 Driver for Data Update



```
import java.sql.*;

class MysqlCon1
{
public sta:ic void main(String args[])
{
  try
 {

Class.forName("com.mysql.jdbc.Driver");//Type4 Driver Syntax

Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/vasavi","root","root");

Statement stmt=con.createStatement();

String query1 = "update emp set name='kishore' " + " where id in(1,4)";// here we use Update Query

stmt.execu:eUpdate(query1);

con.close();
}catch(Exception e)
{
System.out.println(e);
}
}
}
```

Exercise 9:
c) MYSQL connectivity using Type 4 Driver for Data Deletion

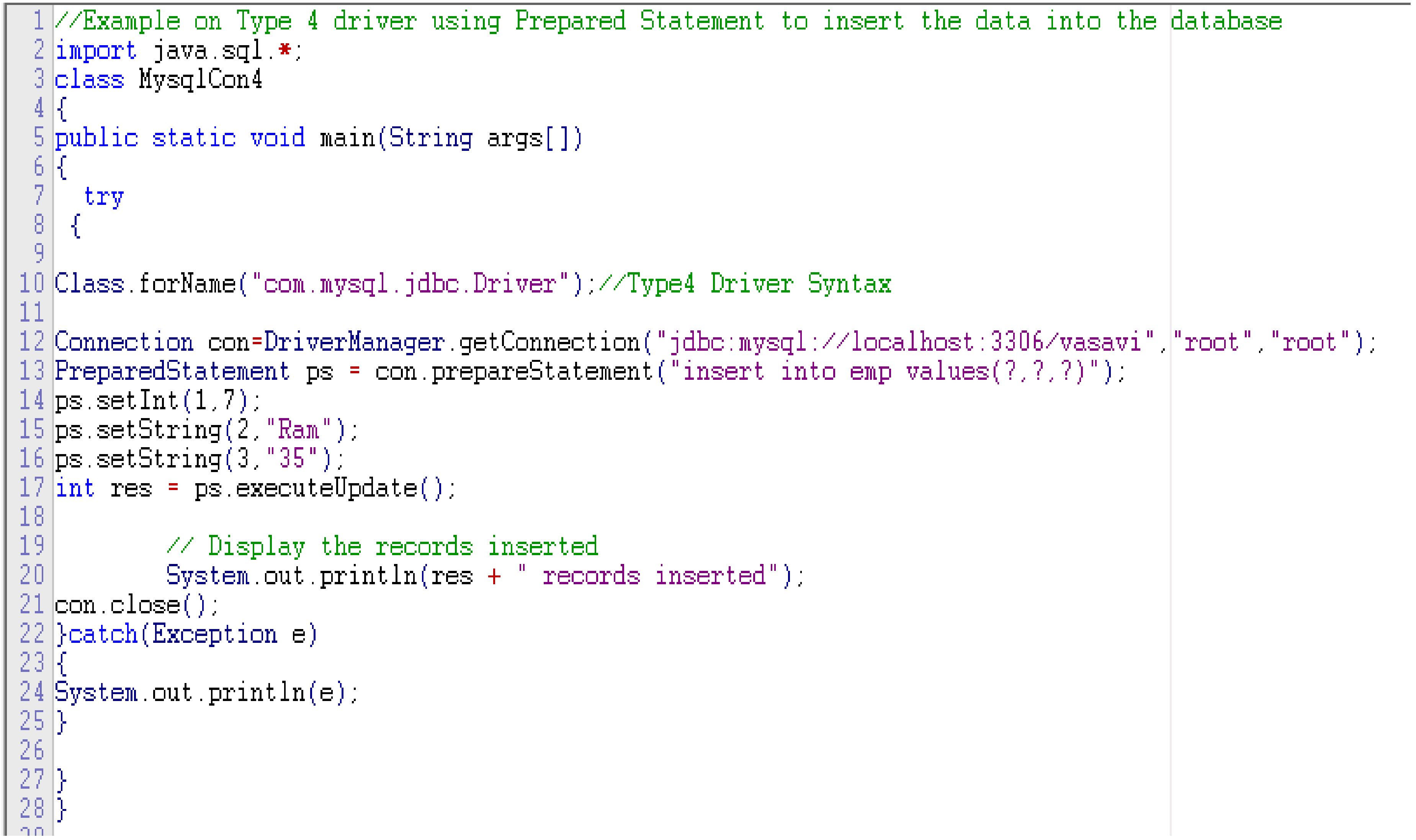


```
import java.sql.*;

class MysqlCon2
{
public static void main(String args[])
{
  try
 {

Class.forName("com.mysql.jdbc.Driver");//Type4 Driver Syntax

Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/vasavi","root","root");

                                           //here vasavi  is the database name, root is the username and root is the password of mysql database
Statement stmt=con.createStatement();
String query1 = "delete  from student "+" where stdname='a'";  // delete operation in mysql using type 4
       stmt.executeUpdate(query1);

con.close();

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

Database Tables for Delete Operation Program

- CREATE DATABASE "vasavi";
- USE "vasavi";
- CREATE TABLE /*!32312 IF NOT EXISTS*/ "student" ("stdno" int(50) , "stdname" varchar(50) , "stdmarks" int(10));
- Try to insert some student details in the above table.s

Exercise 9:
d) MYSQL connectivity using Type 4 Driver for Data Insertion using Prepared Statement



```java
//Example on Type 4 driver using Prepared Statement to insert the data into the database
import java.sql.*;
class MysqlCon4
{
public static void main(String args[])
{
  try
 {

Class.forName("com.mysql.jdbc.Driver");//Type4 Driver Syntax

Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/vasavi","root","root");
PreparedStatement ps = con.prepareStatement("insert into emp values(?,?,?)");
ps.setInt(1,7);
ps.setString(2,"Ram");
ps.setString(3,"35");
int res = ps.executeUpdate();

        // Display the records inserted
        System.out.println(res + " records inserted");
con.close();
}catch(Exception e)
{
System.out.println(e);
}


}
}
```

Exercise 10: Design a simple Node JS application

```
var http = require("http");

http.createServer(function (request, response) {

   // Send the HTTP header

   // HTTP Status: 200 : OK

   // Content Type: text/plain

   response.writeHead(200, {'Content-Type': 'text/plain'});

      // Send the response body as " First Node Js Program !"

   response.end(First Node Js Program ! \n');

}).listen(8081);


// Console will print the message

console.log('Server running at http://127.0.0.1:8081/');
```

How to run the hello.js program

Now execute the hello.js to start the server as follows –

```
$ node hello.js
```

Verify the Output. Server has started.

```
Server running at http://127.0.0.1:8081/
```

Make a Request to the Node.js Server

Open **http://127.0.0.1:8081/** in any browser and observe the following result