IT6503 WEB PROGRAMMING
QUESTION BANK
UNIT I
SCRIPTING
PART- A (2 Marks)

1. Define URI.

URIs have been known by many names: WWW addresses, Universal Document Identifiers, Universal Resource Identifiers , and finally the combination of Uniform Resource Locators (URL) and Names (URN).As far as HTTP is concerned, Uniform Resource Identifiers are simply formatted strings which identify--via name, location, or any other characteristic--a network resource.

2. How will you create a password field in a HTML form?

<input type=”password” name=”pwd” size=”15”>

3. List any four common browsers.

- Google Chrome
- Netscape Navigator
- Microsoft Internet Explorer
- Mozilla

4. List and explain any two HTML elements

i) Form element

Action attribute specifies a url to which the information collected on the form should be sent when user submits the form method attribute to make the request.

ii) div element almost identical to span

The tag defines logical divisions (defined) in your Web page.

Div → block element span → inline element

5. Write HTML code to display an image.

<img src="smiley.gif" alt="Smiley face">

The height and width attributes are used to specify the height and width of an image.

The attribute values are specified in pixels by default:

<img src="smiley.gif" alt="Smiley face" width="42" height="42">

6. Write html code to create the following table

W	X
Y	Z

```
<TABLE>
<TR>
<TD>W</TD>
<TD>X</TD>
</TR>
<TR>
<TD>Y</TD>
<TD>Z</TD>
</TR>
</TABLE>
```

7. List the different basic protocols used in internet

- SMTP (email)
- FTP (file transfer)
- HTTP (transfer of Web documents)
- Telnet

8. Define URI, URL, and URN.

- URI (Uniform Resource Identifier): It identifies an object on the Internet.
- URL (Uniform Resource Locator): It is a specification for identifying an object such as a file, newsgroup, and CGI program or e-mail address by indicating the exact location on the internet.
- URN (Uniform Resource Name): It is a method for referencing an object without declaring the full path to the object.

9. Define MIME. List the predefined MIME content types.

MIME (Multipurpose Internet Mail Extensions) is an open standard for sending multipart, multimedia data through Internet email. The predefined MIME content types:

Text, Multipart, Message, Image, Audio, Video, Model and Application

10. Define HTML and protocol tunneling.

HTML is a simple page description language, which enables document creation for the web. Protocol tunneling is the process of encapsulating one protocol within another protocol that operates on the same layer.

11. What is meant by loop back address?

A zone that enables the server to direct traffic to itself. The host number is almost always 127.0.0.1.

12. Mention some text formatting tags.

-   - is used for introducing various paragraphs.
-
 - this tag is used for giving an empty blank line.
- HEADING TAGS - <h1> </h1> .. <h6> </h6> is used to introduce various headings.

<h1> is the biggest and h6 is the smallest heading tag.

- <HR> TAG – is used to draw lines and horizontal rules.
- <B>,<I>,<U> for bold, italic and underline respectively.

13. What do you mean by Relative URL?

Partial internet address which points to a directory or file in relation to the current directory or file. When creating a link in a website's code, if just a filename is given without any other modifications, that is an indicator to find that file within the current directory and link to it. Modifiers can be added to a relative URL which indicates that the file is found in a higher directory, or code can be added to indicate that the file is in a deeper directory. Relative URLs are often easier to maintain than absolute URLs.

14. Mention the need for cascading style sheets.

CSS are powerful mechanism for adding styles (e.g. Fonts, Colors, Spacing) to web documents.

- They enforce standards and uniformity throughout a web site and provide numerous attributes to create dynamic effects.
- The advantage of a style sheet includes the ability to make global changes to all documents from a single location. Style sheets are said to cascade when they combine to specify the appearance of a page.

15. Explain array creation in Java script with example.

An array can be defined in three ways.

The following code creates an Array object called myCars:

- var myCars=new Array(); // regular array (add an optional integer myCars[0]="Saab";
- // argument to control array's size) myCars[1]="Volvo"; myCars[2]="BMW";
- var myCars=new Array("Saab","Volvo","BMW"); // condensed array
- var myCars=["Saab","Volvo","BMW"]; // literal array

16. List and explain any two java script built in objects.

String Object

The String object is used to manipulate a stored piece of text.

Example

var val = new String("Hello world!");
document.write(val.length)

Date Object

The Date object is used to work with dates and times.

We define a Date object with the new keyword.

Example

var myDate=new Date()

17. List the ways of positioning an element within a browser window.

- **Absolute positioning**

An element which is positioned absolute does not obtain any space in the document. This means that it does not leave an empty space after being positioned.To position an element absolutely, the position property is set as absolute. You can subsequently use the properties left, right, top, and bottom to place the box.

- **Relative positioning**

To position an element relatively, the property position is set as relative. The difference between absolute and relative positioning is how the position is being calculated. The position for an element which is relatively positioned is calculated from the original position in the document. That means that you move the element to the right, to the left, up or down. This way, the element still obtains a space in the document after it is positioned.

18. List the different methods defined in document and window object of java script.

Document Object Methods

- document.title - Sets or returns the title of the document
- document.URL - Returns the full URL of the HTML document
- document.write()- Writes HTML expressions or JavaScript code to a document

Window Object Methods

- alert() - Displays an alert box with a message and an OK button
- close() - Closes the current window
- confirm() - Displays a dialog box with a message and an OK and a Cancel button

19. What is JavaScript?

A lightweight programming language that runs in a Web browser (client-side). Embedded in HTML files and can manipulate the HTML itself and JavaScript is a scripting language. Interpreted, not compiled.

20. What is meant by ID selectors?

The #id selector styles the element with the specified id. The rule applies only to the document element with the given id attribute. In HTML element IDs must have unique values within the same page.

Example

Style the element with id="firstname":

```
#firstname {
   background-color: yellow;
            }
```

21. Which function can be used to create an alert popup dialog box? Write an example to create an alert popup box.

alert () function can be used to create an alert popup dialog box.

Example

```
<html>
<head>
<script type="text/javascript">
function compute ()
{
alert ("Enter the phone no");
}
</script>
</head>
</html>
```

22. Define scriplets.

Scriplets enable you to create small, reusable web applications that can be used in any web page. Scriplets are created using HTML, scripting and Dynamic HTML. To include them in an HTML document use the <OBJECT> tag.

23. Mention the advantages of java/java script

- Use sending data continuously File storage
- Massively parallel computing

- Smart forms – includes various controls like text box, radio button, text area control etc.
- Peer-to-Peer Interaction – used in various client/server model.
- Games – Combine the ability to easily include networking in your programs with java‟s powerful graphics and you have the recipe for truly awesome multiplayer games.
- Chat – Used in various chat applications.
- Whiteboards – Java programs are not limited to sending ext and data across the network.
- A number of programmers have developed whiteboard software that allows users in diverse locations to draw on their computers

24. Differentiate client side scripting and server side scripting

client side scripting	server side scripting
The client is the system on which the Web browser is running.	The server is where the Web page and other content lives.
JavaScript is the main client-side scripting language for the Web.	JSP and Servlets is the server-side scripting language for the Web.
Client-side scripts are interpreted by the browser.	The server sends pages to the user/client on request.

25. How will you include CSS in a web site?

Inline

Inline styles are when you just insert the type of style you want inside another tag, using the style attribute. This is usually the least useful way to use CSS.

<p style="width:100%; color:#660099; text-align:right; background-color:#ffcc00;" >

Embedded

Styles can also be placed in the document using the <style> tag. The <style> tag is usually placed in the head section of the document, where it will apply to the whole document.

```
<style>     <!--
        p { color:#009900;
        font-family:"comic sans ms",sans-serif; }
        h1 { color:#660000; font-size:12pt; }
  </style>
```

External styles

Styles can also be set in an external style sheet which is linked to the page with a <link> tag. For example the style sheet for this site is included like this:

<link rel="stylesheet" type="text/css" href="class.css" />

26. Define the GET() and POST() method.

GET():

The GET method means retrieves whatever information (in the form of an entity) is identified by the Request-URI. If the Request-URI refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process.

POST():

The POST method is used to request that the destination server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line.

PART B (16 MARKS)

1. **Explain HTML elements in detail also State the types of lists supported by HTML and explain them in detail.**

Headings:h1 and Friends

- Headings are produced using h1, h2, …, h6 elements:
- Should use h1 for highest level, h2 subheading, etc.
- h1 largest and bold
- h6 normal text

Spacing: pre and br

- Use pre to retain format of text and display using monospace font:
- XML allows two syntactic representations of empty elements
- Empty tag syntax
 is recommended for browser compatibility
- XML parsers also recognize syntax
</br> (start tag followed immediately by end tag), but many browsers do not understand this for empty elements
- br element represents line break
- br is example of an empty element, i.e., element that is not allowed to have content

Formatting Text Phrases: span, strong, tt, etc..

- Text can be formatted in various ways:
- Style of word and phrases
- Marking the text has content of span element
- Use a phrase element that specifies semantics of text (not style directly):
- Phrase element strong and emphasis (strong and em)
- Use a font style element b and i
- Not recommended, but frequently used

Text
Text

Horizontal rule: hr

- Horizontal rule is produced using hr
- Adds a horizontal line to the document
- Also an empty element
- Style can be modified using style sheet technology

Images: The img Element

Images can be embedded using img element

```
<img
    src="http://www.w3.org/Icons/valid-xhtml10"
    alt="Valid XHTML 1.0!" height="31" width="88"
    style="float:right" />
```

- Attributes:
  - src: URL of image file (required) via HTTP GET
  - Two GET request
    - Request the HTML document
    - Browser generates a GET request to this URL to get the img.
  - alt: text description of image (required)
  - height / width: dimensions of area that image will occupy (recommended)
    - To reserve space for page images by the browser
- If height and width not specified for image, then browser may need to rearrange the client area after downloading the image (poor user interface for Web page)
- If height and width specified are not the same as the original dimensions of image, browser will resize the image
- Default units for height and width are "picture elements" (pixels)
- Represent one dot on a display
- Display composed of grid of such dots
- Each dot in a particular colour forms the image

Links: The a element

- Core hypertext part of HTML

```
See
<a href="http://www.w3.org/TR/html4/index/elements.html">the
  W3C HTML 4.01 Element Index</a>
for a complete list of elements.
```

- Clickable link within a document
- Hyperlinks are produced by the anchor element a
- Default appearance with underlined can be changed using style sheets
- Clicking on a hyperlink causes browser to issue GET request to URL specified in href attribute and render response in client area
- Content of anchor element is text of hyperlink (avoid leading/trailing space in content)
- a element called anchor. Link has two ends→ anchors and direction
- Link starts at "source" anchor and points to destination

- Anchors can be used as source (previous example) or destination
- To link to a point within a webpage, for example to link to content below the first fold, a destination anchor must be added.
- A destination anchor is created by adding a fragment identifier.

```
<a id="section1" name="section1"></a>
```

- The fragment portion of a URL is used to reference a destination anchor

```
<a href="http://www.example.org/PageWithAnchor.html#section1">...
```

- Browser scrolls so destination anchor is at (or near) top of client area

Comments

- Comments are intended to be read by programmer ignored by software
- Comments are a special form of tag

<!—Comment1-->

Nesting Elements

- If one element is nested within another element, then the content of the inner element is also content of the outer element
- XHTML requires that elements be properly nested

Text

HTML Lists

HTML offers authors several mechanisms for specifying lists of information. All lists must contain one or more list elements.

1. Unordered HTML List

- The first item
- The second item
- The third item
- The fourth item
- Ordered HTML List

Example

```
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

2. Ordered information.
1. The first item
2. The second item
3. The third item
4. The fourth item
5. HTML Description List

Example

```
<ol>
  <li>Coffee</li>
  <li>Milk</li>
</ol>
```

3. Definitions.

- The first item
- Description of item
- The second item
- Description of item

Example

```
<dl>
  <dt>Coffee</dt>
  <dd>- black hot drink</dd>
  <dt>Milk</dt>
  <dd>- white cold drink</dd>
</dl>
```

2. Explain the use of relative URL, Special Characters, Entity References and Attributes in HTML with an example.

Relative URLs

- Relative URLs make it easy to have everything working when switching from a staging/dev environment to a live environment with a different domain.
- When you use an absolute URL, your browser has to do a DNS lookup. Relative URLs avoid that problem so you gain a few microseconds.
- Because absolute URLs use up more characters, they increase the file size of your pages by a few hundred bytes at most.

Example

```
<A href="/suppliers.html">Suppliers</A>
```

Special Characters

HTML Entities and/or ISO Latin-1 codes can be placed in source code like any other alphanumeric characters to produce special characters and symbols that cannot be generated in HTML with normal keyboard commands.

Char	Entity	Description
‘	‘	left single quote
’	’	right single quote
,	‚	single low-9 quote
“	“	left double quote
‹	‹	single left-pointing angle quote
›	›	single right-pointing angle quote

Entity References

HTML symbols like mathematical operators, arrows, technical symbols and shapes, are not present on a normal keyboard.

To add these symbols to an HTML page, you can use the HTML entity name.

If no entity name exists, you can use the entity number.

If the character does not have an entity name, you can use a decimal (or hexadecimal) reference.

Char	Number	Entity	Description
∋	∋	∋	ni
∏	∏	∏	prod
∑	∑	∑	sum

Example

I will display €
I will display €
I will display €

HTML Attributes

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

HTML, CSS, JavaScript, XML, SQL, PHP, ASP, etc.

3. Create a registration form for an educational web site with E-Learning resources. All form controls should have appropriate name attributes. Use the GET method for form submission, and specify an empty string for the action attribute.

```
<html>
<body>
   <form action=" " method="GET"  >
         <div align="center">
                  Username *: <input type="username" name="username" />
                  Password *: <input type="password" name="pwd" />
                  Surname *: <input type="surname" name="surname" />
                  Other Names *: <input type="other names" name="names" />
                  Date of Birth *: <input type="date of birth" name="dob" />
                  Email *: <input type="email" name="email" />
                  Telephone: <input type="telephone" name="tel" />
                  Address *: <input type="address" name="add" />
                  Post Code *: <input type="postcode" name="ptc" />
         <input type="submit" value="Submit" />
         </div>
   </form>
 <p>
course registration
</ p>
 </body>
</html>
```

4. Explain the significance of XHTML with the help of a real time application. Write necessary code snippets

Extensible Hypertext Markup Language (XHTML) is a family of XML markup languages that mirror or extend versions of the widely used Hypertext Markup Language (HTML), the language in which Web pages are formulated.

Significance

- **Sustainability.**

Web applications tend towards XML. Using XHTML now instead of HTML makes any future conversion of the website easier.

- **Extensibility.**

Owing to the extensibility of XML, XHTML documents can be supplemented with other forms of markup, MathML (Math Markup Language) SVG (Scalable Vector Graphics) or your own markup variants, thanks to the use of namespaces.

- **Compatibility.**

Because XHTML documents are written in compliance with the rules of XML, XML-processing programmes can effortlessly convert an XHTML file to another format (e.g. PDF, RSS or RTF).

- **Efficiency of processing applications.**

Once browsers support XHTML documents and the strict rules of XML, they will become quicker thanks to shorter error processing routines. At present, a great deal of the processing power of a browser is still spent on liberal error processing of documents containing malformed HTML markup.

Features

- XHTML requires strict adherence to coding rules.
- XHTML encourages a more structured and conceptual way of thinking about content and, combined with the style sheet, a more creative way of displaying it.
- XHTML makes it easier for people to dream up and add new elements.

Code snippets

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
     <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
     <title>Page Title</title>
     <link rel="stylesheet" href="style.css" type="text/css" media="screen" charset="utf-8"/>
</head>
<body>
</body>
</html>
```

5. How do you create frames? Why do we need them? Develop an application to explain the same.

Creating Frames

- To use frames on a page we use <frameset> tag instead of <body> tag.
- The <frameset> tag defines how to divide the window into frames.
- The rows attribute of <frameset> tag defines horizontal frames and cols attribute defines vertical frames.
- Each frame is indicated by <frame> tag and it defines which HTML document shall open into the frame.

Format

<frameset rows="pixels|%|*">

Attribute Values

Value	Description
pixels	The row height in pixels (like "100px" or just "100")
%	The row height in percent of the available space (like "50%")
*	The rest of the available space should be assigned this row

Need for Frame

- Frames are a way of organizing your website. They allow you to divide up your window into various segments for different purposes.
- Another reason might be to have your entire site's links visible on the page, while the actual 'content' - i.e. text scrolls as much as it needs.

Example

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
   "http://www.w3.org/TR/html4/frameset.dtd">
<html>
<head>
<title>Web Technology</title>
</head>
<frameset cols="200,*" frameborder="0" border="0" framespacing="0">
   <frame name="menu" src="menu_1.html" marginheight="0" marginwidth="0" scrolling="auto" noresize>
   <frame name="content" src="content.html" marginheight="0" marginwidth="0" scrolling="auto" noresize>
<noframes>
<p> Frame Example
</frameset>
</html>
```

6. List and explain in detail the various selector strings.

- One or more style sheets-- Statements
- Each line in style1.css called Rule
- Form of rule called-- Rulest
- Consist of two parts

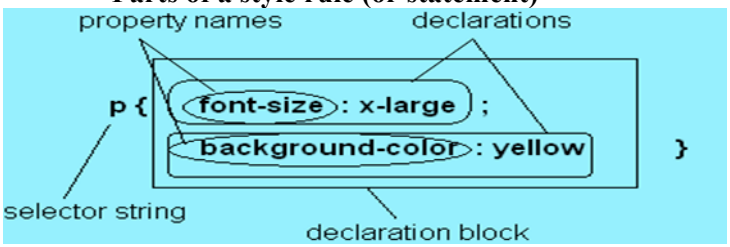
•Selector string

- Indicates the elements to which the rule should apply

•Declaration block in { }

- Specifies a value for one style property of those elements
- List of declaration separated by semicolons
- It is syntactically legal to split rules over several lines or write multiple rules on a single line

- **Parts of a style rule (or statement)**



- Type selector : Type selector matches elements with the corresponding element type name.

```
h1,h2,h3,h4,h5,h6 { background-color : purple }
```

- Universal selector : Every possible element type
- The universal selector matches any element type.

```
* { font-weight : bold }
```

- ID selector : Matches an element that has a specific id attribute value. Since id attributes must have unique values, it can never match more than one element in a document.

```
  #p1, #p3 { background-color : aqua }
<p id="P1" class="takeNote">
   Paragraph with id="P1" and class="takeNote".
</p>
.
.
<p id="p3">
   This paragraph (id="p3") contains a
   <a href="http://www.example.net">hyperlink</a>.
```

EXAMPLE PROGRAM:

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>
      Selectors.html
    </title>
    <link rel="stylesheet" type="text/css" href="sel-demo.css" />
  </head>
  <body>
    <h1>Selector Tests</h1>
    <p id="p1" class="takeNote">
      Paragraph with id="p1" and class="takeNote"
    </p>
     <p id="p2" class="special">
       Second paragraph. <span class="takeNote special cool"> This span
       belongs to classes takeNote, special, and cool.</span>
       <ul>
         <li>Span's within this list are in <span>small-cap</span>
             style.</li>
         <ol>
           <li>This item spaces letters.</li>
         </ol>
       </ul>
     </p>
     <p id="p3">
       This paragraph (id="p3") contains a
       <a href="http://www.example.net">hyperlink</a>.
       <ol>
         <li>This item contains a span but does not display it in
           <span>small capes</span>,nor does it space letters.</li>
       </ol>
     </p>
  </body>
</html>
```

```
h1,h2,h3,h4,h5,h6 { background-color : purple }
* { font-weight : bold }
#p1, #p3 { background-color : aqua }
#p1, .takeNote { font-style : italic }
span.special { font-size : x-large }
a:link { color : black }
a:visited { color : yellow }
a:hover { color : green }
a:active { color : red }
ul span { font-variant : small-caps }
ul ol li { letter-spacing : 1em }
```

- **Class selector :**

```
#p1, .takeNote { font-style : italic }
span.special { font-size : x-large }
```

The difference between an ID and a class is that an ID can be used to identify one element, whereas a class can be used to identify more than one.

- Descendant selector : *used to select elements that are descendants of another element in the document tree.*

ul span { font-variant : small-caps }
ul ol li { letter-spacing : lem }

Predefined pseudo classes associated with anchor element

- a:link { color : black } – link that has not been visited recently
- a:visited { color : yellow } – visited once next time loaded
- a:hover { color : green } – *positioning cursor over link without clicking the mouse button cause link to change green*

a:active { color : red } - *clicking and holding the mouse button change the colour to red*

7 . Explain about Style Rule Cascading and Inheritance

Rule cascading:

*{ font-weight : bold }àapplies to every element of HTML

#p1, #p3 { background-color : aqua }

#p3{ font-weight : bold }

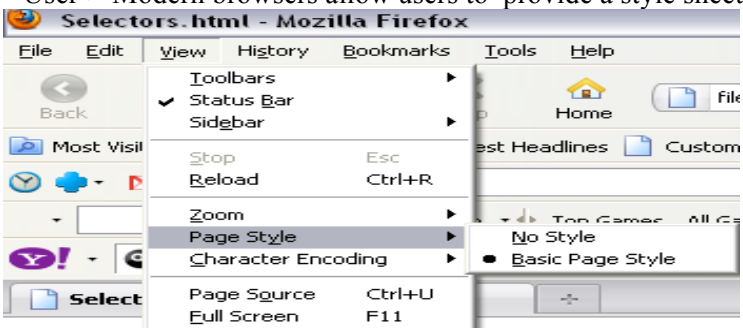
What if more than one style declaration applies to a property of an element?

- Multiple declaration
- Browser applies rule cascading
- A multistage sorting process that selects a single declaration that supply the property value
- The CSS rule cascade determines which style rule's declaration applies
- Once declaration identified, associate **origin and weight** with every declaration

•PERSON WHO WROTE THE DOCUMENT
•PERSON WHO IS VIEWING THE DOCUMENT
•PERSON WHO WROTE THE BROWSER SOFTWARE

- **Origin of a declaration is one of the following:**

•Author-> declaration is part of an external or embed style sheet or part of the value specified by style attribute
•User agent –>A browser define default style property values for HTML elements
• User-> Modern browsers allow users to provide a style sheet



- **User style rules defined in two ways:**

•**Edit|Preferences|Appearance,** Fonts and colors panels allow a user to select various style options
•User can explicitly create a style sheet file the browser will input when it is started
•Features provided by IE6

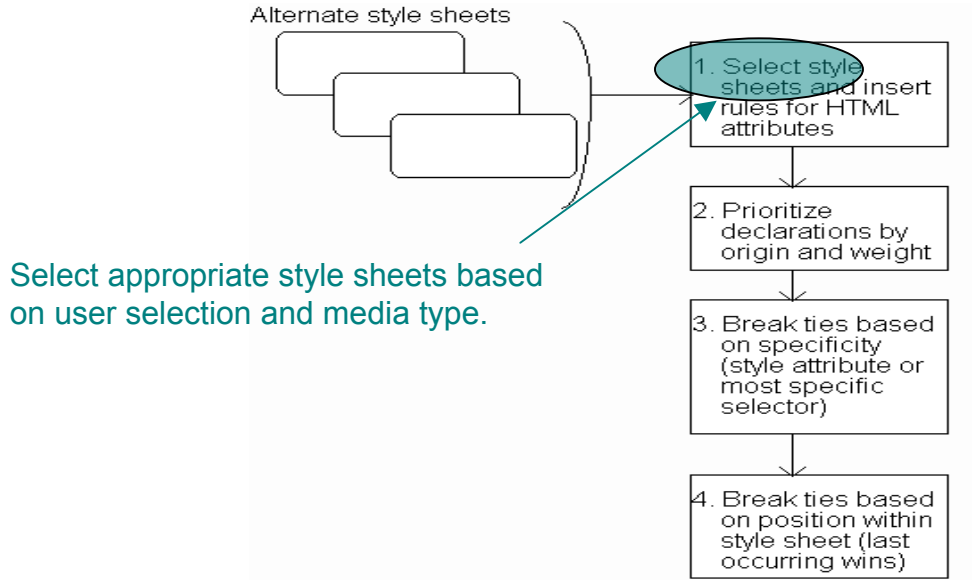
- **Tools|Internet Options**
- **Two weight values**

•Normal
•Important

- User/important highest priority in CSS to accommodate users with special needs

•Rules made important by adding "!important":

p{text-indent:3em;font-size:larger !important}



Cascading order

What style will be used when there is more than one style specified for an HTML element?

all the styles will "cascade" into a new "virtual" style sheet by the following rules, where number four has the highest priority:

1. **Browser default**
2. **External style sheet**
3. **Internal style sheet (in the head section)**
4. **Inline style (inside an HTML element)**

So, an inline style (inside an HTML element) has the highest priority, which means that it will override a style defined inside the <head> tag, or in an external style sheet, or in a browser (a default value).

Note: If the link to the external style sheet is placed after the internal style sheet in HTML <head>, the external style sheet will override the internal style sheet!

Style Inheritance

- Cascading based on structure of style sheets
- Inheritance based on tree structure of document
- What if no style declaration applies to a property of an element?
- Generally, the property value is inherited from the nearest ancestor element that has a value for the property
- If no ancestor has a value (or the property does not inherit) then CSS defines an initial value that is used
- Property values:

•Specified: value contained in declaration

- Absolute: value can be determined without reference to context (*e.g.*, 2cm)
- Relative: value depends on context (*e.g.*, larger)

•Computed: browser performs calculation depends on particular relative value

•absolute representation of relative value (*e.g.*, larger might be 1.2 x parent font size)
•Actual: value actually used by browser (*e.g.*, computed value might be rounded)
• Most properties inherit computed value
•Exception discussed later: line-height
• A little thought can usually tell you whether a property inherits or not
•Example: height does not inherit

8. Explain about the various style sheets with examples. (Internal, External, Inline)

To create an inline style
a. Add the style attribute to the HTML tag.
b. The style declaration must be enclosed within double quotation marks.
To create an embedded style
c. Insert a <style> tag within the head section of HTML file.
d. Within the <style> tag, enclose the style declarations need to the entire Web page.
e. The style sheet language identifies the type of style used in the document.
f. The default and the most common language is "text/css" for use with CSS.
To create an External styles
g. Create a text file containing style declarations
h. Create a link to that file in each page of the Web site using a <link> tag.
i. Specify the link attributes, such as href, rel, and type.
j. Link a style sheet, the value of the href attribute should be the "URL" of the linked document, the value of the rel attribute should be "stylesheet" and the value of the type attribute should be "text/css".

EXTERNAL.CSS:

```
body{ background-color: gray;}
p { color: blue; }
h3{ color: white; }
```

EXTERNAL.HTML:

```
<html>
<head>
<link rel="stylesheet" type="text/css" href="EXTERNAL.css" /><!—Link tag for External CSS-->
</head>
<body>
<h3> A White Header </h3>
<p> This paragraph has a blue font.
The background color of this page is gray because
we changed it with CSS! </p>
</body>
</html>
```

INTERNAL.HTML:

```
<html>
<head>
<style>  <!—Style tag for Internal CSS-->
body { background-color: blue; }
p { color: white; }
</style>
</head>
<body>
<h2>Internal CSS</h2>
<p>This page uses internal CSS.  Using the style tag we are able to modify
the appearance of HTML elements.</p>
</body>
</html>
```

INLINE.HTML:

```
<html>
<head>
</head>
<body>
<h2>InLINE CSS</h2>
<p style="color:sienna;margin-left:20px"><!—Style Attribute(INLINE)-->
This page uses INLINE CSS.  Using the style ATTRIBUTE we are able to modify
the appearance of HTML elements.
</p>
</body>
</html>
```

9. (i) Write java script to find sum of first n even number and display the result. Get the value of n from user.[8]

```
<html >
<head>
<title>Calc Numbers</title>
<script type='text/javascript'>
function calc()
{
// calc() is designed to sum numbers entered in an input field
var t = document.getElementById("num").value; // the numbers entered
var numbers = new Array();
var sum = 0; // variable to hold the sum of the numbers
var num = ''; // string to hold a list of the numbers
if (numbers.length == 1)
 {
num = '0';
 }
for (i = 0; i < numbers.length; i++)
{
sum = sum + eval(numbers[i]);
}
// create the text to display with the result
result = "<p>The sum of the numbers " + num + " is " + sum +"<"+"/p>";
// find the placeholder for displaying the result
var resElement = document.getElementById("res");
// display the result
resElement.innerHTML = result;
}
</script>
</head>
<body>
<h1>Calc Numbers</h1>
<p>Enter some numbers separated by spaces:</p>
<input id='num' onclick="this.value="" onchange='calc()'>
<input type="button" value="Calc">
<div id='res'></div>
</body>
</html>
```

(ii) Write java script to find factorial of a given number.

```
<html>
<head>
<script type="text/javascript">
function factorial(f,n)
{
l=1;
for(i=1;i<=n;i++)
l=l*i;
f.p.value=l;
}
</script>
</head>
<body>
<form>
number:<input type="text" name="t"></br>
<input type="button" value="submit" onClick="factorial(this.form,t.value)"></br>
result:<input type="text" name="p"></br>
</form>
</body>
</html>
```

10. Discuss Javascript objects in detail with suitable examples.

- An object is a set of properties
- A property consists of a unique (within an object) name with an associated value
- The type of a property depends on the type of its value and can vary dynamically
- Object properties do not have data types

Ex: Single property prop of an object o

```
o.prop = true;      prop is Boolean
o.prop = "true";    prop is now String
o.prop = 1;         prop is now Number
```

- There are no classes in JavaScript
- Object constructors defined to create objects and automatically define properties for the objects created
- Instead, properties and methods can be created and deleted dynamically

```
var o1 = new Object();          Create an object o1
o1.testing = "This is a test"; Create property testing
delete o1.testing;              Delete testing property
```

- Objects are created using new expression
- First line creates a variable named o1 initialize its value of type object by calling built-in constructor Object()

```
new Object()    Constructor and argument list
```

- Second line adds a property named testing to the o1 object and assigns a string value to this property
- A constructor is a function
- When called via new expression, a new empty Object is created and passed to the constructor along with the argument values
- Constructor performs initialization on the object
- Can add properties and methods to object
- Can add object to an inheritance hierarchy from which it can inherit additional properties and methods
- The Object() built-in constructor
- Does not add any properties or methods directly to the object
- default toString() and valueOf() methods (used for conversions to String and Number, resp.)

```
o1.testing = "This is a test";
```

- Assign a value to an object property

- Property does not exist in the object
- Property with the given name is created in the object and assigned the specified value
- delete used to remove a property from an object
- Object initializer notation can be used to create an object (using Object() constructor) and one or more properties in a single statement

Enumerating Properties

- To know which property an object has at any given time
- Special form of for statement used to iterate through all properties of an object:

```
var hash = new Object();
hash.kim = "85";
hash.sam = "92";
hash.lynn = "78";
for (var aName in hash) {
  window.alert(aName + " is a property of hash.");
}
```

UNIT II
JAVA
PART- A (2 Marks)

1. **Define object?**
   Defining variables of a class data type is known as class instantiation and such variables are called objects. Object is a instance of a class.
2. **Define class?**
   Object oriented programming constructs support a data type called class. A class encloses both the data and functions. The enclosed data and function in a class are called data member and member function respectively.
3. **Define encapsulation?**
   It is a mechanism that associates the code and data it manipulates into a single unit. In C++, this is supported by a construct called class. An instance of a class is known as an object, which represents a real world entity.
4. **Define data abstraction?**
   The technique of creating new data type that are well suited to an application to be programmed is known as data abstraction. The class is a construct in C++ for creating user defined data types called ADTS.
5. **Define message passing in oops?**
   It is a process of invoking an operation of object. In response to a message, the corresponding method is executed in the object.
6. **List out all object oriented concepts?**
   List of object oriented concepts are Encapsulation, data abstraction, Inheritance, polymorphism, message passing, extensibility, persistence, delegation, generality
7. **What do you mean by inheritance?**
   It allows the extension and reuse of existing code without having to rewrite the code from scratch. Inheritance involves the creation of new classes (derived classes) from the existing ones(base classes), thus enabling the creation of a hierarchy of classes that simulate the class and sub class concept of the real world.
8. **Define polymorphism?**
   It allows a single name/operator to be associated with different operation depending on the type of data passed to it.
9. **List out all access specifiers?**
   The list of access specifiers are private, public, and protected.

Private:

Private members can be accessed by only members of the same class and can not be accessed by non members.

Public:

public members can be accessible to both members and non members.

Protected : it can be accessed by members of subclasses and own members and can not be accessed by non members

10. What is static in java.

A static data member is defined with a static keyword preceding it in the class defintion. It must have a definition outside the body of the class. It is shared between all the objects of a class.

11. How can you achieve Multiple inheritance (MI) in java?

Java's interface mechanism can be used to implement inheritance, with one important difference from c++ way of doing MI: the inherited interface must be abstract. This obviates the need to choose between different implementations as with interfaces there are no implementations.

12. What is a string buffer class and how does it differs from string class.

In contrast to the String class which implements immutable character strings, the StringBuffer class implements mutable character strings. Not only can the character string in a string buffer gets changed, but the buffer's capacity can also change dynamically. The capacity of a String Buffer is the maximum number of characters that a string buffer can accommodate, before its size is automatically augmented

String Class:

```
String str = new String ("Stanford  ");
Str+="Lost!!";
```

StringBuffer Class:

```
StringBuffer str = new StringBuffer ("Stanford ");
str.append("Lost!!");
```

13. What are the difference between the abstract class and interface?

An interface cannot implement any methods, whereas an abstract class can.

A class can implement many interfaces but can have only one super class.

An interface is not part of the class hierarchy. Unrelated classes can implement the same interface

14. Define Packages.

A *package* is a grouping of related types providing access protection and name space management.

15. How is an interface? How it is used in java?

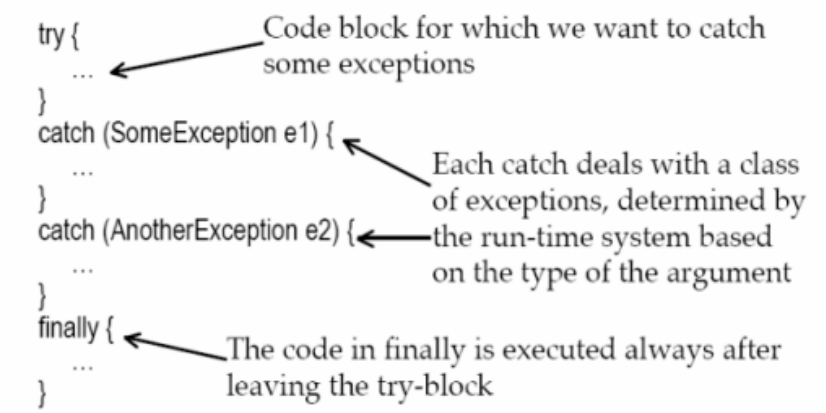
A class that implements an interface adheres to the protocol defined by that interface. To declare a class that implements an interface, include an implements clause in the class declaration. A class can implement more than one interface (the Java platform supports multiple inheritance for interfaces), so the implements keyword is followed by a comma-separated list of the interfaces implemented by the class

16. What is an abstract method?

An abstract method is a method whose implementation is deferred to a sub class.

17. Explain the significance of try-catch blocks?

Whenever the exception occurs in Java, we need a way to tell the JVM what code to execute. To do this, we use the try and catch keywords. The try is used to define a block of code in which exceptions may occur. One or more catch clauses match a specific exception to a block of code that handles it.



18. What is the use of finally block?

The finally block encloses code that is always executed at some point after the try block, whether an exception was thrown or not. This is right place to close files, release your network sockets, connections, and perform any other cleanup your code requires.

19. How to create custom exceptions?

By extending the Exception class or one of its subclasses.

Example:

```
Class MyException extends Exception
 {
        public MyException() { super(); }
    public MyException(String s) { super(s); }
 }
```

20. How Threads are created in Java?

Threads are created in two ways. They are by extending the Thread class and by implementing Runnable interface.

21. What is multithreaded programming?

Multithreaded program contains 2 or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution.

22. What is thread priority?

Java assigns to each thread a priority that determines how that thread should be treated with respect to the others. Thread priorities are integers that specify the relative priority of one thread to another.

23. What is the difference between the Reader/Writer class hierarchy and the InputStream/OutputStream class hierarchy?

The Reader/Writer class hierarchy is character-oriented, and the InputStream/OutputStream class hierarchy is byte-oriented.

24. Describe life cycle of thread?

A thread is a execution in a program. The life cycle of a thread include:

- Newborn state
- Runnable state
- Running state
- Blocked state
- Dead state

25. What is HashMap and Map?

Map is Interface which is part of Java collections framework. This is to store Key Value pair, and Hashmap is class that implements that using hashing technique.

PART B (16 Mark)

1. Write a program to perform matrix multiplication

Program:

```
Class MatrixMultiply{
  public static void main(String[] args)  {
    int array[][] = {{5,6,7},{4,8,9}};
    int array1[][] = {{6,4},{5,7},{1,1}};
    int array2[][] = new int[3][3];
    int x= array.length;
    System.out.println("Matrix 1 : ");
      for(int i = 0; i < x; i++) {
      for(int j = 0; j <= x; j++) {
        System.out.print(" "+ array[i][j]);
      }
    System.out.println();
    }
    int y= array1.length;
    System.out.println("Matrix 2 : ");
      for(int i = 0; i < y; i++) {
      for(int j = 0; j < y-1; j++) {
        System.out.print(" "+array1[i][j]);
      }
    System.out.println();
    }

      for(int i = 0; i < x; i++) {
      for(int j = 0; j < y-1; j++) {
        for(int k = 0; k < y; k++){

          array2[i][j] += array[i][k]*array1[k][j];
        }
      }
     }
    System.out.println("Multiply of both matrix : ");
    for(int i = 0; i < x; i++) {
      for(int j = 0; j < y-1; j++) {
        System.out.print(" "+array2[i][j]);
      }
    System.out.println();
    } }}
```

2. Discuss the different levels of access protection available in java.

Access specifiers:

private	member	Accessible only in its class(which defins it).
protected	member	Accessible only within its package and its subclasses
public	class	Accessible anywhere

	interface	Accessible anywhere
	member	Accessible anywhere its class is.

Example;

```
Package P1;
Class A
{
        Private I;
        Protected J;
        Public K;
 Void set()
        {
        I=10;
        J=20;
        K=30;
        }
}

Package P1;
Class B extends A
{
        Void set()
        {
                J=20;
                K=30;
        }
}



Package P1;
Class C
{
        Void set()
        {
                A  A1=new A();
                A1.J=20;
                A1.K=30;
        }
}

Package P2;
Class D extends A
{
        Void set()
        {
                J=20;K=30;
        }
}
```

```
Package P2;
Class E
{
        Void set()
        {
                A A1=new A();
                A1.K=30;
                }
}
```

3. What does it mean that a method or class is abstract? Can we make an instance of an abstract class? Explain it with example

- An abstract class is a shared superclass
  - Provides a high-level partial implementation of some concept
  - Cannot be instantiated
  - An abstract method is one that cannot meaningfully be implemented by a class
    - A generic operation
    - Part of an abstract class
  - Must be implemented by a concrete subclass
    - Each concrete subclass can implement the method differently

Program:

```
abstract public class Draw {
abstract public void point();
}

public class Circle extends Draw {
        private double radius;
        public Circle(double radius)
        {
                this.radius=radius;
        }
        public void point()
        {
                double area=3.14*radius*radius;
                System.out.println("The Area of Circle"+area);
        }
        void display4()
        {
                System.out.println("The value of radius"+radius);
        }
}
public class Rectangle extends Draw {
        private double breath;
        private double length;
        public Rectangle(double breath,double length)
        {
                this.breath=breath;
                this.length=length;
        }
        public void point()
        {
                double area=breath*length;
```

```
        System.out.println("The Area of Rectangle"+area);
}
void display3()
{
        System.out.println("The value of breath"+breath+" , "+"The value of length"+length);
}}
```

4. Explain with examples of various types of constructors. Why constructor does not have return types in java? State the use of finalize() method. Explain in with proper example.

Constructor

A function with the same name as the class itself responsible for construction and returning objects of the class is called constructor.

Types of Constructor

Copy constructor

Copy constructor is used to copy the value of one object into another. When one object is copied to another using initialization, they are copied by executing the copy constructor.

The copy constructor contains the object as one of the passing argument.

Default constructor

A constructor without any argument is called default constructor.

Ex

```
Class complex
{
Public:   void complex();
};
```

1. Parameterized constructor

A constructor with one or more than one argument is called parameterized constructor.

Ex

```
Class complex
{
Public:
Complex  complex( int real, intimag);
};
```

2. Multiple constructors

A class with more than one constructor comes under multiple constructor. Multiple constructors is constructed with different argument list. That is either with empty default constructor or with parameterized constructor.

```
public class Point {
  int x, y;

  Point(int x, int y)
  {
    this.x = x;
    this.y = y;
  }
  Point(Point p)
  {
    this(p.x, p.y);
  }
  void move(int dx, int dy) {
```

```
    x += dx;
    y += dy;
   }
   public String toString() {
    return "(" + x + ", " + y + ")";
   }
  }
    protected void finalize() throws Throwable
     {
     }
```

- Every class inherits the finalize() method from java.lang.Object
- The method is called by the garbage collector when it determines no more references to the object exist
- The Object finalize method performs no actions but it may be overridden by any class
- Normally it should be overridden to clean-up non-Java resources ie closing a file
- If overriding finalize () it is good programming practice to use a try-catch-finally statement and to always call super .finalize() (JPL pg 47-48). This is a safety measure to ensure you do not inadvertently miss closing a resource used by the objects calling class

```
protected void finalize() throws Throwable
{
        try
        {
            close();        // close open files
        }
      finally
        {
            super.finalize();
        }
}
```

- Any exception thrown by finalize() during garbage collection halts the finalization but is otherwise ignored

Finalize() is never run more than once on any object

5. Give an example where interface can be used to support multiple inheritances. Develop a standalone java program for the example.

It is just to **remove ambiguity**, because **multiple inheritance** can cause ambiguity in few scenarios.

```
public class Main {
    public static void main(String[] args)
     {
     shapeA circleshape=new circle();
    circleshape.Draw();
     circleshape.Draw();
     }
     }

 interface shapeA
{
   public   String baseclass="shape";
   public void Draw();
}
 interface shapeB
{
```

```
    public   String baseclass="shape2";
    public void Draw2();
}
  class circle implements shapeA,shapeB
{
   public   String baseclass="shape3";
   public void Draw() {
      System.out.println("Drawing Circle here:"+baseclass);
   }
   @Override
   public void Draw2() {
       System.out.println("Drawing Circle here:"+baseclass);
   }
}
```

6. Explain abstract classes with an example program? Also describe the properties of abstract classes.

An *abstract class* is a class that is declared abstract. It may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be sub classed.

An *abstract method* is a method that is declared without an implementation (without braces, and followed by a semicolon).

```
abstract void moveTo(double deltaX, double deltaY);
```

If a class includes abstract methods, then the class itself *must* be declared abstract, as in:

```
public abstract class GraphicObject {
  // declare fields
  // declare nonabstract methods
  abstract void draw();
}
abstract class GraphicObject {
   int x, y;
   ...
   void moveTo(int newX, int newY) {
      ...
   }
   abstract void draw();
   abstract void resize();
}
class Circle extends GraphicObject {
   void draw() {
      ...
   }
   void resize() {
      ...
   }
}
class Rectangle extends GraphicObject {
   void draw() {
      ...
   }
   void resize() {
      ...
   }
```

```
}
```

When an abstract class is subclassed, the subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, then the subclass must also be declared abstract.

7. How to declare and initialize a string in java and also explain the different string handling functions with suitable examples.

String class:

- It is a predefined class in Java.
- It is a non-mutable class.
- It supports several constructors.

STRING CLASS CONSTRUCTORS

String class offers several constructors. The four basic constructors that are widely used are explained below.

Program:

```
public class Stringexp {
        void exp3()
        {
                charcharArray[] = { 'b', 'i', 'r', 't', 'h', ' ', 'd', 'a', 'y' };
                String s = new String( "hello" );

                // use String constructors
                String s1 = new String();
                String s2 = new String( s );
                String s3 = new String(charArray );
                String s4 = new String(charArray, 6, 3 );

                System.out.println("\ns1 ="+s1+"\ns2="+s2+"\ns3="+s3+"\ns4="+s4 );
        }
        public static void main(String[] args) {

        Stringexp s=new Stringexp();
        s.exp3();
}
}
public class TestOne {
public void disp1()
{

        long []counts=new long[40];
        String g=new String("Hellow are you there");
        for(int i=0;i<g.length();i++)
        {
        counts[g.charAt(i)]++;

        }
        for(int i=0;i<counts.length;i++)
        {
                System.out.println(counts[i]);
        }
}
```

```
public static void main(String[] args)
{
TestOne t=new TestOne();
String a="hai";
String s=new String("Hellow");
String a1=new String("Hellow");
String s1=new String("Hai");
String s2=new String("hellowMr how r you");
System.out.println(a);
System.out.println(s.charAt(0));
System.out.println(s.indexOf("l"));
System.out.println(s.compareTo(a1));
System.out.println(s.length());
System.out.println(s2.substring(3, 17));
//System.out.println(a);
//System.out.println(a1);
if(s==a1)
{
        System.out.println("right");
}
else
{
        System.out.println("wrong");
}
System.out.println(a1.indexOf("l"));
System.out.println(a1.lastIndexOf("l"));
}
}
```

8.i. What is meant by stream? What are the types of streams and classes? Explain in detail.

An I/O Stream represents an input source or an output destination. A stream can represent many different kinds of sources and destinations, including disk files, devices, other programs, and memory arrays.

The ObjectInputStream class supports the reading of objects from input streams.

Getting Input from User:

```
package io;
import java.io.*;
public class IO1
{
        public static void main(String args[])throws IOException
        {
                boolean i;
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
                System.out.println("Enter . to Stop");
                do {
                        String s=br.readLine();
                        System.out.println(s);
                        i=s.contains(".");
                        }while(!i);
        }
```

```
}
```

Getting Input from File and Write to File:

```
package io;
import java.io.*;
public class IO2
{
        public static void main(String args[])throws IOException
        {
                int v;
                FileReader fin=null;
                String inf,outf;
                String s=new String();
                try
                {
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
                System.out.println("Enter the input file");
                inf=br.readLine();
                System.out.println("Enter the output file");
                outf=br.readLine();

                File file=new File(inf);
                fin=new FileReader(file);
                BufferedReader gs=new BufferedReader(fin);
                String linestring=gs.readLine();
                while(linestring!=null)
                {
                s=s.concat(linestring+"    ");
                linestring=gs.readLine();
                }

                File outFile = null;
                FileOutputStream fis = null;
                BufferedOutputStream bos = null;
                outFile = new File(outf);
System.out.println("contents are written into  "+outFile.getCanonicalPath());
                        fis = new FileOutputStream(outFile);
                        bos = new BufferedOutputStream(fis);
                        bos.write(s.getBytes());
                        bos.close();
                } catch (IOException e) {
                        e.printStackTrace();
                }
                }
        }
```

ii. List and discuss the role of various buffer classes used in java programming.

The **StringBuffer** and **StringBuilder** classes are used when there is a necessity to make a lot of modifications to Strings of characters.

StringBuffer Methods:

Here is the list of important methods supported by StringBuffer class:

1. **public StringBuffer append(String s)**
Updates the value of the object that invoked the method. The method takes boolean, char, int, long, Strings etc.
2. **public StringBuffer reverse()**
The method reverses the value of the StringBuffer object that invoked the method.
3. **public delete(int start, int end)**
Deletes the string starting from start index until end index.
4. **public insert(int offset, int i)**
This method inserts an string s at the position mentioned by offset.
5. **replace(int start, int end, String str)**
This method replaces the characters in a substring of this StringBuffer with characters in the specified String.

9. Create a try block that is likely to generate three types of exception and then incorporate necessary catch blocks to catch and handle them appropriately.

1. Error occurred in execution time
2. Abnormal termination of program
3. Wrong execution result
4. Provide an exception handling mechanism in language system
5. Improve the reliability of application program
6. Allow simple program code for exception check and handling into source
7. Treat exception as an object

GENERAL SYNTAX:

```
try {

        //statements – one of which is capable of throwing an exception
    }

catch (ExceptionTypeName objName)
{

        //one or more statements to execute if this exception occurs

}
```

EXAMPLE:

```
package Exception;
public class MainClass {
          public static void main(String args[]) {
            int d, a;
          try {
               d = 0;
               a = 42 / d;
               System.out.println("This will not be printed.");
             }
          catch (ArithmeticException e) {
               System.out.println("Division by zero.");
             }
             System.out.println("After catch statement.");
```

```
    }
}
```

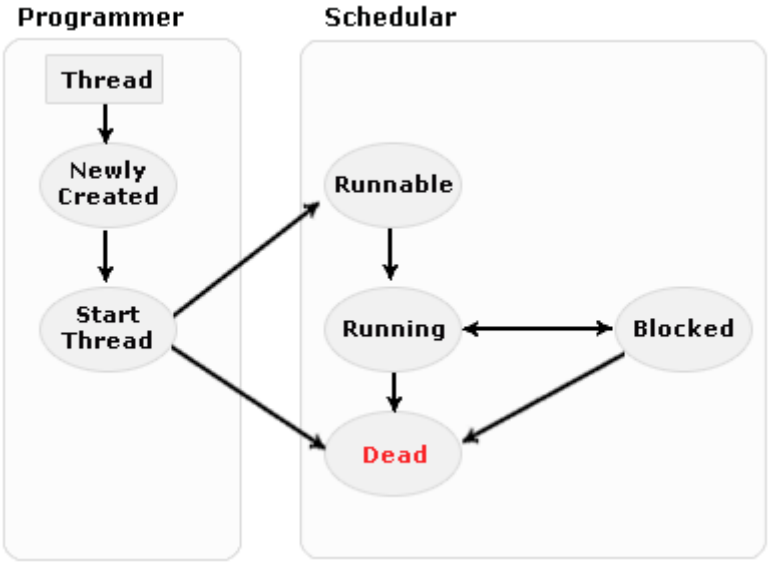
10. Explain creating a thread, extending the thread class and an example of using the thread class.

THREADS:

Threads arelight weight processes

EXTENDING THREAD CLASS

```
public class ThreadSample extends Thread {

        public void run()
        {
                try
                {
                        for(int i=5;i>0;i--)
                        {
                                System.out.println("Child Thread"+i);
                                Thread.sleep(1000);
                        }
                }
                catch(InterruptedException e)
                {
                        System.out.println("Child interrupted");
                }
                System.out.println("Exiting Child Thread");
        }
}

public class MainThread
{
public static void main(String[] arg)
{
        ThreadSample d=new ThreadSample();
        d.start();
        try
{
                for(int i=5;i>0;i--)
                {
                        System.out.println("Main Thread"+i);
                        Thread.sleep(5000);
                }
        }
        catch(InterruptedException e)
{
                System.out.println("Main interrupted");
        }
        System.out.println("Exiting Main Thread");
}
}
```



1. **New state ?** After the creations of Thread instance the thread is in this state but before the start() method invocation. At this point, the thread is considered not alive.

2. **Runnable (Ready-to-run) state ?**A thread start its life from Runnable state. A thread first enters runnable state after the invoking of start() method but a thread can return to this state after either running, waiting, sleeping or coming back from blocked state also. On this state a thread is waiting for a turn on the processor.

3. **Running state ?**A thread is in running state that means the thread is currently executing. There are several ways to enter in Runnable state but there is only one way to enter in Running state: the scheduler select a thread from runnable pool.

4. **Dead state ?** A thread can be considered dead when its run() method completes. If any thread comes on this state that means it cannot ever run again.
5. **Blocked -** A thread can enter in this state because of waiting the resources that are hold by another thread.

11. Explain in detail about classes implementing List interface with suitable example?

The List interface extends **Collection** and declares the behavior of a collection that stores a sequence of elements.

- Elements can be inserted or accessed by their position in the list, using a zero-based index.
- A list may contain duplicate elements.
- In addition to the methods defined by **Collection**, List defines some of its own, which are summarized in the following below Table.
- Several of the list methods will throw an UnsupportedOperationException if the collection cannot be modified, and a ClassCastException is generated when one object is incompatible with another.

SN	Methods with Description
1	**void add(int index, Object obj)** Inserts obj into the invoking list at the index passed in index. Any pre-existing elements at or beyond the point of insertion are shifted up. Thus, no elements are overwritten.
2	**boolean addAll(int index, Collection c)**

	Inserts all elements of c into the invoking list at the index passed in index. Any pre-existing elements at or beyond the point of insertion are shifted up. Thus, no elements are overwritten. Returns true if the invoking list changes and returns false otherwise.
	Object get(int index)
3	Returns the object stored at the specified index within the invoking collection.
	int indexOf(Object obj)
4	Returns the index of the first instance of obj in the invoking list. If obj is not an element of the list, .1 is returned.
	int lastIndexOf(Object obj)
5	Returns the index of the last instance of obj in the invoking list. If obj is not an element of the list, .1 is returned.
	ListIterator listIterator( )
6	Returns an iterator to the start of the invoking list.
	ListIterator listIterator(int index)
7	Returns an iterator to the invoking list that begins at the specified index.
	Object remove(int index)
8	Removes the element at position index from the invoking list and returns the deleted element. The resulting list is compacted. That is, the indexes of subsequent elements are decremented by one
	Object set(int index, Object obj)
9	Assigns obj to the location specified by index within the invoking list.
	List subList(int start, int end)
10	Returns a list that includes elements from start to end.1 in the invoking list. Elements in the returned list are also referenced by the invoking object.

Example:

Above interface has been implemented in various classes like ArrayList or LinkedList, etc. Following is the example to explain few methods from various class implementation of the above collection methods:

```
import java.util.*;

public class CollectionsDemo {

  public static void main(String[] args) {
    List a1 = new ArrayList();
    a1.add("Zara");
    a1.add("Mahnaz");
    a1.add("Ayan");
    System.out.println(" ArrayList Elements");
```

```
    System.out.print("\t" + a1);

    List l1 = new LinkedList();
    l1.add("Zara");
    l1.add("Mahnaz");
    l1.add("Ayan");
    System.out.println();
    System.out.println(" LinkedList Elements");
    System.out.print("\t" + l1);
  }
}
```

UNIT III
JDBC
PART- A (2 Marks)

1. What is JDBC?

JDBC stands for Java Database Connectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

2. What are the common JDBC API components?

JDBC API consists of following interfaces and classes: DriverManager, Driver, Connection, Statement, ResultSet, SQLException.

3. What is a JDBC DriverManager?

JDBC DriverManager is a class that manages a list of database drivers. It matches connection requests from the java application with the proper database driver using communication subprotocol.

4. What is a connection?

Connection interface consists of methods for contacting a database. The connection object represents communication context.

5. What is a statement?

Statement encapsulates an SQL statement which is passed to the database to be parsed, compiled, planned and executed.

6. What is a ResultSet?

These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data. The java.sql.ResultSet interface represents the result set of a database query.

7. What are types of ResultSet?

There are three constants which when defined in result set can move cursor in resultset backward, forward and also in a particular row.

- ResultSet.TYPE_FORWARD_ONLY: The cursor can only move forward in the result set.
- ResultSet.TYPE_SCROLL_INSENSITIVE: The cursor can scroll forwards and backwards, and the result set is not sensitive to changes made by others to the database that occur after the result set was created.
- ResultSet.TYPE_SCROLL_SENSITIVE: The cursor can scroll forwards and backwards, and the result set is sensitive to changes made by others to the database that occur after the result set was created.

8. What are the basic steps to create a JDBC application?

- Following are the basic steps to create a JDBC application:
- Import packages containing the JDBC classes needed for database programming.
- Register the JDBC driver, so that you can open a communications channel with the database.
- Open a connection using the DriverManager.getConnection () method.
- Execute a query using an object of type Statement.
- Extract data from result set using the appropriate ResultSet.getXXX () method.

- Clean up the environment by closing all database resources relying on the JVM's garbage collection.

9. What are the standard isolation levels defined by JDBC?

- The standard isolation levels are:
- TRANSACTION_NONE
- TRANSACTION_READ_COMMITTED
- TRANSACTION_READ_UNCOMMITTED
- TRANSACTION_REPEATABLE_READ
- TRANSACTION_SERIALIZABLE

10. What does setAutoCommit do?

- setAutoCommit() invoke the commit state query to the database. To perform batch updation we use the setAutoCommit() which enable us to execute more than one statement together, which in result minimize the database call and send all statement in one batch.
- setAutoCommit() allowed us to commit the transaction commit state manually the default values of the setAutoCommit() is true.

11. What are types of JDBC drivers?

There are four types of drivers defined by JDBC as follows:

- **JDBC/ODBC:** These require an ODBC (Open Database Connectivity) driver for the database to be installed. It is used for local connection.
- **Native API (partly-Java driver):** This type of driver uses a database API to interact with the database. It also provides no host redirection.
- **Network Protocol Driver:** It makes use of a middle-tier between the calling program and the database. The client driver communicates with the net server using a database-independent protocol and the net server translates this protocol into database calls.
- **Native Protocol Drive:** This has a same configuration as a type 3 driver but uses a wire protocol specific to a particular vendor and hence can access only that vendor's database.

12. How can you load the drivers?

- It is very simple and involves just one line of code to load the driver or drivers we want to use.
- For example, We want to use the JDBC-ODBC Bridge driver, the following code will load it:
- Class.forName(”sun.jdbc.odbc.JdbcOdbcDriver”);
- Driver documentation will give you the class name to use. For instance, if the class name is jdbc.DriverHELLO, you would load the driver with the following line of code:
- Class.forName(”jdbc.DriverHELLO”);

13. What Is a Socket?

A socket is one end-point of a two-way communication link between two programs running on the network. Socket classes are used to represent the connection between a client program and a server program. The java.net package provides two classes--Socket and ServerSocket--that implement the client side of the connection and the server side of the connection, respectively.

14. What is a JavaBean?

JavaBeans are reusable software components written in the Java programming language, designed to be manipulated visually by a software develpoment environment, like JBuilder or VisualAge for Java. They are similar to Microsoft’s ActiveX components, but designed to be platform-neutral, running anywhere there is a Java Virtual Machine (JVM).

15. What are some advantages and disadvantages of Java Sockets?

- **Advantages of Java Sockets:**
  - Sockets are flexible and sufficient. Efficient socket based programming can be easily implemented for general communications.
  - Sockets cause low network traffic. Unlike HTML forms and CGI scripts that generate and transfer whole web pages for each new request, Java applets can send only necessary updated information.
- **Disadvantages of Java Sockets:**

- Security restrictions are sometimes overbearing because a Java applet running in a Web browser is only able to establish connections to the machine where it came from, and to nowhere else on the network .
- Despite all of the useful and helpful Java features, Socket based communications allows only to send packets of raw data between applications. Both the client-side and server-side have to provide mechanisms to make the data useful in any way.
- Since the data formats and protocols remain application specific, the re-use of socket based implementations is limited.

16. How do I create an InetAddress instance?

In case of InetAddress the three methods getLocalHost, getByName, getByAllName can be used to create instances.

E.g.

```
InetAddress add1;
InetAddress add2;
try
{
add1 = InetAddress.getByName("java.sun.com");
add2 = InetAddress.getByName("199.22.22.22″);
}
catch(UnknownHostException e)
{
}
```

17. What is the difference between URL instance and URLConnection instance?

A URL instance represents the location of a resource, and a URLConnection instance represents a link for accessing or communicating with the resource at the location.

18. When MalformedURLException and UnknownHostException throws?

When the specified URL is not connected then the URL throw MalformedURLException and If InetAddress' methods getByName and getLocalHost are unable to resolve the host name they throw an UnknownHostException.

19. What is the difference between TCP and UDP?

TCP	**UDP**
Connection oriented transport protocol	Connection less protocol
Sends data as a stream of bytes	Datagram service
Guarantee of delivery	No guarantee of delivery.

20. What is RMI?

RMI stands for Remote Method Invocation. Traditional approaches to executing code on other machines across a network have been confusing as well as tedious and error-prone to implement. The nicest way to think about this problem is that some object happens to live on another machine, and that you can send a message to the remote object and get a result as if the object lived on your local machine.

21. Explain RMI Architecture?

RMI uses a layered architecture, each of the layers could be enhanced or replaced without affecting the rest of the system. The details of layers can be summarised as follows:

- Application Layer: The client and server program

- Stub & Skeleton Layer: Intercepts method calls made by the client/redirects these calls to a remote RMI service.
- Remote Reference Layer: Understands how to interpret and manage references made from clients to the remote service objects.
- Transport layer: Based on TCP/IP connections between machines in a network. It provides basic connectivity, as well as some firewall penetration strategies.

22. What are the services in RMI ?

An RMI "service" could well be any Java method that can be invoked remotely. The other service is the JRMP RMI naming service which is a lookup service.

23. How many types of protocol implementations does RMI have?

RMI has at least three protocol implementations: Java Remote Method Protocol(JRMP), Internet Inter ORB Protocol(IIOP), and Jini Extensible Remote Invocation(JERI). These are alternatives, not part of the same thing, All three are indeed layer 6 protocols for those who are still speaking OSI reference model.

24. What is Registry Service for RMI?

The registration of the remote object must be done by the server in order for the client to look it up, is called the RMI Registry. In RMI, the client must contact an RMI registry, so that the server side application will be able to contact the client's registry which points the client in the direction of the service. The client registers the service with the registry so that it is transparent to even for the server.

25. Explain how URL convention is used for accessing the registry.

The class rebind () method of java.rmi.Naming class is used to specify the port number . For example if the registry is running on a port number 3271 of an application named HelloRMIRegistry the following is the usage of the URL to reference the remote object:

Naming.rebind ("//host:1111/RMIRegistry", obj);

The URL stored on the web page needs to specify the non-default port number.When the server's remote objects created by the server can include the URL from which the stub class can dynamically be downloaded to the client.

PART- B(16 Marks)

1. What is a JavaBeans component? How will you use the JSP language elements for accessing Beans in your JSP pages?

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

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

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

JavaBeans Properties:

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

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

Method	Description

get**PropertyName**()	For example, if property name is *firstName*, your method name would be getFirstName() to read that property. This method is called accessor.
set**PropertyName**()	For example, if property name is *firstName*, your method name would be setFirstName() to write that property. This method is called mutator.

A read-only attribute will have only a get**PropertyName**() method, and a write-only attribute will have only a set**PropertyName**() method.

JavaBeans Example:

Consider a student class with few properties:

```
package com.tutorialspoint;

public class StudentsBean implements java.io.Serializable
{
  private String firstName = null;
  private String lastName = null;
  private int age = 0;

  public StudentsBean() {
  }
  public String getFirstName(){
    return firstName;
  }
  public String getLastName(){
    return lastName;
  }
  public int getAge(){
    return age;
  }
  public void setFirstName(String firstName){
    this.firstName = firstName;
  }
  public void setLastName(String lastName){
    this.lastName = lastName;
  }
  public void setAge(Integer age){
    this.age = age;
  }
}
```

Accessing JavaBeans:

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

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

Here values for the scope attribute could be page, request, session or application based on your requirement. The value of the **id** attribute may be any value as a long as it is a unique name among other useBean declarations in the same JSP.

Following example shows its simple usage:

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

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

</body>
</html>
```

This would produce following result:

The date/time is Thu Sep 30 11:18:11 GST 2010

Accessing JavaBeans Properties:

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

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

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

Following is a simple example to access the data using above syntax:

```
<html>
<head>
<title>get and set properties Example</title>
</head>
<body>

<jsp:useBean id="students"
            class="com.tutorialspoint.StudentsBean">
  <jsp:setProperty name="students" property="firstName"
            value="Zara"/>
  <jsp:setProperty name="students" property="lastName"
            value="Ali"/>
```

```
  <jsp:setProperty name="students" property="age"
                value="10"/>
</jsp:useBean>

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

</body>
</html>
```

Let us make StudentsBean.class available in CLASSPATH and try to access above JSP. This would produce following result:

Student First Name: Zara

Student Last Name: Ali

Student Age: 10

2. Write a program for banking application using JDBC (consider 5 customer create a/c no and type, set some minimum balance do credit and debit operation and print consolidated report for month wise transaction.)
   Program:

```
import java.sql.*;
import java.sql.DriverManager;
import java.util.*;
public class Connectivity {
    Connection conn=null;
    Statement st=null;
    String sql = null;
    ResultSet rs=null;
   public void connect() throws ClassNotFoundException, SQLException  {
    Class.forName("org.apache.derby.jdbc.ClientDriver");
    conn=DriverManager.getConnection("jdbc:derby://localhost:1527/shop","root","root");
    st=conn.createStatement();
    System.out.println("Enter ur pin no");
    Scanner sn=new Scanner(System.in);
    String pin=sn.next();
    sql="select * from root.bank where ano='"+pin+"'";
    rs=st.executeQuery(sql);
    if(rs.next()){
       System.out.println("enter ur choice \n 1. withdraw 2.deposit 3.balance check");
       int ch=sn.nextInt();
```

```
   switch(ch){
      case 1:
         int balance=Integer.parseInt(rs.getString("balance"));
          System.out.println("Enter amount");
          int n=sn.nextInt();
          balance=balance-n;
         String bl=""+balance;
          if(balance>0){
             sql="UPDATE root.bank SET balance="+balance+" WHERE ano='"+pin+"'";
             if(st.executeUpdate(sql)>0){
                 sql="select * from root.bank where ano='"+pin+"'";
rs=st.executeQuery(sql);
if(rs.next()){
   System.out.println("balance is"+rs.getString("balance"));

          System.out.println("success");
}              }
          }
          else{
                       System.out.println("failed");
          }
              break;
       case 2:
            int balance1=Integer.parseInt(rs.getString("balance"));
          System.out.println("Enter amount");
          int n1=sn.nextInt();
          balance1=balance1+n1;

          if(balance1>0){
             sql="UPDATE root.bank SET balance="+balance1+" WHERE ano='"+pin+"'";
             if(st.executeUpdate(sql)>0){
                 sql="select * from root.bank where ano='"+pin+"'";
rs=st.executeQuery(sql);
if(rs.next()){
   System.out.println("balance is"+rs.getString("balance"));

          System.out.println("success");
}              }
          }
          else{
                       System.out.println("failed");
          }

         break;
       case 3:

                 sql="select * from root.bank where ano='"+pin+"'";
rs=st.executeQuery(sql);
if(rs.next()){
   System.out.println("balance is"+rs.getString("balance"));

          System.out.println("success");
```

```
    }

            break;
         default:
            System.out.println("wrong choice");
            break;
      }
    }
}
   public static void main(String a[]) throws ClassNotFoundException, SQLException{
      Connectivity con=new Connectivity();
      con.connect();

   }
}
```

Output:

```
Enter ur pin no
1234
enter ur choice
 1. withdraw 2.deposit 3.balance check
2
Enter amount
100
balance is2100
success
```

3. Explain the JDBC architecture and its driver types.

Introduction to JDBC:

JDBC stands for Java Database Connectivity. It is set of Java API‘s(application programming interface) used for executing SQL statements. This API consists of a set of classes and interfaces to enable programmers to write pure Java Database applications. JDBC is a software layer that allows developers to write real client –server projects in Java. JDBC does not concern itself with specific DBMS functions. JDBC API defines how an application opens a connection, communicates with a database, executes SQL statements, and retrieves query result. Following fig. will illustrate the role of JDBC. JDBC is based on the X/OPEN call level interface (CLI) for SQL. Call Level Interface is a library of function calls that supports SQL statements. CLI requires neither host variables nor other embedded SQL concepts that would make it less flexible from a programmer‘s perspective. It is still possible, however, to maintain and use specific functions of a database management system when accessing the database through a CLI.

JDBC was designed to be very compact, simple interface focusing on the execution of raw SQL statements and retrieving the results. The goal of creating JDBC is to create an interface that keeps simple tasks, while ensuring the more difficult and uncommon tasks are at least made possible.
The following are the characteristics of JDBC.
It is a call-level SQL interface for java
It does not restrict the type of queries passed to an underlying DBMS driver
JDBC mechanism are simple to understand and use

It provides a java interface that stays consistent with the rest of the Java system
JDBC may be implemented on top of common SQL level APIs.

Microsoft ODBC API offers connectivity to almost all databases on all platforms and is the most widely used programming interface for accessing relational database. But ODBC cannot be directly used with java programs due to various reasons enumerated in the JDBC vs. ODBC section. Hence the need for JDBC came into existence.
It is possible to access various relational databases like Sybase, Oracle, Informix, Ingers, using JDBC API. Using JDBC, we can write individual programs to connect to individual database or one program that take care of connecting to the respective database.

Java and JDBC:

The combination of java with JDBC is very useful because it lets the programmer run his/her program on different platforms, Java programs are secure, robust, automatically downloaded from the network and java is a good language to create database applications. JDBC API enables Java applications to interact with different types of database. It is possible to publish vital information from a remote database on a web page using a java applet. With increasing inclination of programmers towards Java, knowledge about JDBC is essential.

Some of the advantages of using Java with JDBC are as follows:

Easy and economical
Continued usage of already installed databases
Development time is short
Installation and version control simplified

How does JDBC work

JDBC defines a set of API objects and methods to interact with the underlying database. A Java program first opens a connection to the database, makes a statement object, passes SQL statements to the underlying database management system (DBMS) through the statement object and retrieve the results as well as information about the result set.
There are two types of interfaces – low –level interface and high-level interface. While high level interfaces are user-friendly, low-level interfaces are not. JDBC is a low-level API interface, ie. it used to invoke or call SQL commands directly. The required SQL statements are passed as strings to Java methods.
Some of the current trend that are being developed to add more features to JDBC are embedded SQL for java and direct mapping of relational database to java classes.
Embedded SQL enables mixing of java into SQL statements. These statements are translated into JDBC calls using SQL Processor. In this type of direct mapping of relational database tables to java, each row of the table becomes an instance of the class and each column value corresponds to an attribute of that instance. Mapping is being provided that makes rows of multiple tables to form a java class.

JDBC VS ODBC:

The most widely used interface to access relational database today is Microsoft‘s ODBC API. ODBC performs similar tasks as that of JDC(Java Development Connection) and yet JDBC is preferred due to the following reasons :
ODBC cannot be directly used with Java because it uses a C interface. Calls from Java to native C code have a number of drawbacks in the security, implementation, robustness and automatic portability of applications.

ODBC makes use of pointers which have been totally removed from Java
ODBC mixes simple and advanced features together and has complex options for simple queries. But JDBC is designed to keep things simple while allowing advanced capabilities when required.
JDBC API is a natural Java Interface and is built on ODBC. JDBC retains some of the basic features of ODBC like X/Open SQL Call Level Interface.
JDBC is to Java programs and ODBC is to programs written in languages other than Java.
ODBC is used between applications and JDBC is used by Java programmers to connect to databases.

Details about JDBC

The JDBC API is in the package java.sql it consists of 8 interfaces, 6 classes and 3 exceptions in JDK1.1.
Interfaces:
CallableStatement
Connection
DatabaseMetaData
Driver
PreparedStatement
ResultSet
ResultSetMetaData
Statement

Classes:

Date
DriverManager
DriverPropertyInfo
Time
Timestamp
Types

Exceptions:
DataTruncation
SQLException
SQLWarning

JDBC DRIVER MODEL

JDBC Driver Types
There are 4 types of JDBC drivers. Commonest and most efficient of which are type 4 drivers. Here is the description of each of them:
JDBC Type 1 Driver - They are JDBC-ODBC Bridge drivers ie. Translate JDBC into ODBC and use Windows ODBC built in drivers. They delegate the work of data access to ODBC API. They are the slowest of all. SUN provides a JDBC/ODBC driver implementation.

JDBC Type 2 Driver - They mainly use native API for data access ie. Converts JDBC to data base vendors native SQL calls and provide Java wrapper classes to be able to be invoked using JDBC drivers like Type 1 drivers; requires installation of binaries on each client.
JDBC Type 3 Driver - Translates JDBC to a DBMS independent network protocol. They are written in 100% Java and use vendor independent Net-protocol to access a vendor independent remote listener. This listener in turn maps the vendor independent calls to vender dependent ones. This extra step adds complexity and decreases the data access efficiency.

JDBC Type 4 Driver - They are also written in 100% Java and are the most efficient among all driver types. It compiles into the application, applet or servlet; doesn't require anything to be installed on client machine, except JVM. It also converts JDBC directly to native API used by the RDBMS.

The JDBC API supports both two-tier and three-tier processing models for database access.

4. Explain in detail the various steps used in implementing RMI with suitable example

Steps to Implementing RMI

1. There are essentially four software parts to implementing RMI. These are: - client program (does the request) - server program (implements the request) - stub interface (used by the client so that it knows what functions it can access on the server side) - skeleton interface (used by the server as the interface to the stub on the client side)

2. The information flows as follows: the client talks to the stub, the stub talks to the skeleton and the skeleton talks to the server.

3. The Client. The client requires the client program itself and an interface to the server object that we are going to connect to. An interface example is shown below:

Public interface WeatherIntf extends java.rmi.Remote { Public String getWeather() throws java.rmi.RemoteException; } Figure 1. The Client Interface.

The important thing to note about the interface it that it defines an interface to the function to be called on the server. In this case, the function to be called is getWeather(). The client program itself is shown in Figure 2. The main points of the client program are as follows. First, to implement RMI, we need to use the "java.rmi" package. This package contains all the guts to implement RMI. In the main function, we first instantiate a Remote object of the type that we want to use. In this case, we are going to remotely connect to the WeatherServer object through the WeatherIntf interface. Once we have the object from the server then we can call its functions like we would as if the object were located on our computer.

```
Import java.rmi.*;
public class RMIdemo {  public static void main(String[] args) {  try {  Remote robj =
Naming.lookup("//192.168.0.9/WeatherServer");  WeatherIntf weatherserver = (WeatherIntf) robj;
String forecast = weatherserver.getWeather();  System.out.println("The weather will be " +
forecast);
  } catch (Exception e) {System.out.println(e.getMessage()); }  }  } Figure 2. The Client
Program.
```

Once the Client and Interface have been written, they have to be compiled into class files.

4. The Server. It is interesting to note that the Server program also requires the Interface class as shown in Figure 1. This makes sense since the Server will be implementing the Interface. To use the Server, we need to use the "java.rmi" package and the "java.rmi.server.UnicastRemoteObject" package. The Server IS a UnicastRemoteObject, and is declared as such in the class declaration shown below. This means that the system will be a remote object that can be called from a client. Note that the method that we are going to call from the client is the getWeather() function and it can throw a RemoteException. The interesting function to see here is the main function. To implement RMI, we need to set a security manager with permissions that will allow clients to access functions on this pc from another pc. This is easily done by setting the System Security Manager to a RMISecurityManager. This will allow other pcs to call functions on this pc. Then we create an instance of this class simply so that we can pass the object as an argument to the Naming.Rebind. Rebind is the way a server announces its service to the Registry. The Registry is a program that contains a table of services that can be used on a server by client programs.

```
Import java.rmi.*; import java.rmi.server.UnicastRemoteObject;
public class WeatherServer extends UnicastRemoteObject implements WeatherIntf {  public
WeatherServer () throws java.rmi.RemoteException {  super();  }  public String getWeather()
throws RemoteException {  return Math.random()>0.5? "sunny" : "rainy";  }  public static
void main(String[] args) {  System.setSecurityManager(new RMISecurityManager());  try {
WeatherServer myWeatherServer = new WeatherServer();  Naming.rebind("/WeatherServer",
myWeatherServer);  } catch (Exception e) { System.out.println(e.getMessage()); }  }
```

} Figure 3. The Server Program.

5. Compiling the code. So far we have created three java programs: the client, the server and the interface. Each of these must be compiled into class files. This can be done in the IDE or on the command line by using "javac WeatherIntf.java", "javac WeatherServer.java", or "javac RMIdemo.java" for example.

Once the three class files have been compiled, we now have to create the stub and the skeleton for the network communications between them. The is done by using the rmic compiler and the server program. The following can be entered at the command prompt: "rmic WeatherServer". This will create "WeatherServer_Stub.class" and "WeatherServer_Skel.class" for the client and server respectively.

6. Running the Program. The Server needs the skeleton, the interface and the server program class files. The Client needs the Stub, the interface and the client program class files. The best place to put these files is in the Java Runtime Environment directory. On my machine it is: C:\Program Files\JavaSoft\JRE\1.3\lib\ext.

On the Server side we need to start the registry so that the server functions can be made public to other machines. This is done by starting the rmi registry via typing at the command prompt: "rmiregistry". The registry program will continue to run in this window until CTRL-C is pressed.

To run the server with the RMISecurityManager, we have to define the permissions that we want to grant clients. We do this via a permit file. The permit file that I used is shown in Figure 4. Basically, I set connect and accept permissions on the socket connections. I also set read permissions on files in the tmp directory just to illustrate, although it is not required in this demo. Grant { permission java.net.SocketPermission "*", "connect"; permission java.net.SocketPermission "*', "accept"; permission java.io.FilePermission "/tmp/*", "read"; }; Figure 4. The permit file.

Now we can start the server and make it use the permit file by typing the following at the command prompt: "java –Djava.security.policy=permit WeatherServer"

The Client can then be started on the client machine by typing the following at its command prompt: "java RMIdemo"

7. Some things to note. Note that all that is needed on the Client side is the main client program, and the stub (WeatherIntf stub). The Server side requires that rmiregistry is running, the main server program is started with the permit file, and the skeleton (WeatherIntf_skeleton).

5. How do RMI clients contact remote RMI servers? Explain with detailed architecture of RMI.

RMI (Remote Method Invocation)

1. Remote Method Invocation (RMI)
2. Understanding stub and skeleton
   1. stub
   2. skeleton
3. Requirements for the distributed applications
4. Steps to write the RMI program
5. RMI Example

The RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects stub and skeleton.

Understanding stub and skeleton

RMI uses stub and skeleton object for communication with the remote object.

A remote object is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

stub

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

In the Java 2 SDK, an stub protocol was introduced that eliminates the need for skeletons.

Understanding requirements for the distributed applications

If any application performs these tasks, it can be distributed application.

.

1. The application need to locate the remote method
2. It need to provide the communication with the remote objects, and

3. The application need to load the class definitions for the objects.

The RMI application have all these features, so it is called the distributed application.

Steps to write the RMI program

The is given the 6 steps to write the RMI program.

1. Create the remote interface
2. Provide the implementation of the remote interface
3. Compile the implementation class and create the stub and skeleton objects using the rmic tool
4. Start the registry service by rmiregistry tool
5. Create and start the remote application
6. Create and start the client application

RMI Example

In this example, we have followed all the 6 steps to create and run the rmi application. The client application need only two files, remote interface and client application. In the rmi application, both client and server interacts with the remote interface. The client application invokes methods on the proxy object, RMI sends the request to the remote JVM. The return value is sent back to the proxy object and then to the client application.

1) create the remote interface

For creating the remote interface, extend the Remote interface and declare the RemoteException with all the methods of the remote interface. Here, we are creating a remote interface that extends the Remote interface. There is only one method named add() and it declares RemoteException.

copy to clipboard

```
1. import java.rmi.*;
2. public interface Adder extends Remote{
3. public int add(int x,int y)throws RemoteException;
4. }
```

---

2) Provide the implementation of the remote interface

Now provide the implementation of the remote interface. For providing the implementation of the Remote interface, we need to

- Either extend the UnicastRemoteObject class,
- or use the exportObject() method of the UnicastRemoteObject class

In case, you extend the UnicastRemoteObject class, you must define a constructor that declares RemoteException.

copy to clipboard

```
1.  import java.rmi.*;
2.  import java.rmi.server.*;
3.  public class AdderRemote extends UnicastRemoteObject implements Adder{
4.  AdderRemote()throws RemoteException{
5.  super();
6.  }
7.  public int add(int x,int y){return x+y;}
8.  }
```

---

3) create the stub and skeleton objects using the rmic tool.

Next step is to create stub and skeleton objects using the rmi compiler. The rmic tool invokes the RMI compiler and creates stub and skeleton objects.

copy to clipboard

```
1.  rmic AdderRemote
```

---

4) Start the registry service by the rmiregistry tool

Now start the registry service by using the rmiregistry tool. If you don't specify the port number, it uses a default port number. In this example, we are using the port number 5000.

copy to clipboard

```
1.  rmiregistry 5000
```

---

5) Create and run the server application

Now rmi services need to be hosted in a server process. The Naming class provides methods to get and store the remote object. The Naming class provides 5 methods.

1. public static java.rmi.Remote lookup(java.lang.String) throws java.rmi.NotBoundException, java.net.MalformedURLException, java.rmi.RemoteException; it returns the reference of the remote object.

2. public static void bind(java.lang.String, java.rmi.Remote) throws java.rmi.AlreadyBoundException, java.net.MalformedURLException, java.rmi.RemoteException; it binds the remote object with the given name.

3. public static void unbind(java.lang.String) throws java.rmi.RemoteException, java.rmi.NotBoundException, java.net.MalformedURLException; it destroys the remote object which is bound with the given name.

4. public static void rebind(java.lang.String, java.rmi.Remote) throws java.rmi.RemoteException, java.net.MalformedURLException; it binds the remote object to the new name.

5. public static java.lang.String[] list(java.lang.String) throws java.rmi.RemoteException, java.net.MalformedURLException; it returns an array of the names of the remote objects bound in the registry.

In this example, we are binding the remote object by the name sonoo.

copy to clipboard

```
1.  import java.rmi.*;

2.  import java.rmi.registry.*;

3.  public class MyServer{

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

```
5.  try{
6.  Adder stub=new AdderRemote();
7.  Naming.rebind("rmi://localhost:5000/sonoo",stub);
8.  }catch(Exception e){System.out.println(e);}
9.  }
10. }
```

---

6) Create and run the client application

At the client we are getting the stub object by the lookup() method of the Naming class and invoking the method on this object. In this example, we are running the server and client applications, in the same machine so we are using localhost. If you want to access the remote object from another machine, change the localhost to the host name (or IP address) where the remote object is located.

copy to clipboard

```
1.  import java.rmi.*;
2.  public class MyClient{
3.  public static void main(String args[]){
4.  try{
5.  Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/sonoo");
6.  System.out.println(stub.add(34,4));
7.  }catch(Exception e){}
8.  }
9.  }
```

---

download this example of rmi

---

copy to clipboard

1. For running this rmi example,

   1) compile all the java files

. javac *.java

   2)create stub and skeleton object by rmic tool

   rmic AdderRemote

   3)start rmi registry in one command prompt

   rmiregistry 5000

   4)start the server in another command prompt

. java MyServer

. 5)start the client application in another command prompt

java MyClient

6. Create an Application for online shopping with JDBC connectivity
   PROGRAM:

```
import java.sql.*;
import java.sql.DriverManager;
import java.util.*;
public class Connectivity {
    Connection conn=null;
    Statement st=null;
    String sql = null;
    ResultSet rs=null;
   public void connect() throws ClassNotFoundException, SQLException  {
    Class.forName("org.apache.derby.jdbc.ClientDriver");
    conn=DriverManager.getConnection("jdbc:derby://localhost:1527/shop","root","root");
    st=conn.createStatement();
    System.out.println("Select the item:");
    System.out.println("1.Television");
    System.out.println("2.Mobile");
    System.out.println("3.Radio");
    System.out.println("enter the no of items:");
    Scanner i=new Scanner(System.in);
    int n=i.nextInt();
    int total=0;
```

```
 int total1=0;
for(int j=0;j<n;j++)
{
   System.out.println("enter item id:");
   int choice=i.nextInt();
   if(choice==1)
   {

      String sql="select tv from root.price";
      rs=st.executeQuery(sql);
      while(rs.next())
      {
      total1=rs.getInt("tv");
   }
      total=total+total1;
}
 if(choice==2)
   {

      String sql="select mobile from root.price";
      rs=st.executeQuery(sql);
      while(rs.next())
      {
      total1=rs.getInt("mobile");
   }2
      total=total+total1;
}
 if(choice==3)
   {

      String sql="select radio from root.price";
      rs=st.executeQuery(sql);
      while(rs.next())
      {
      total1=rs.getInt("radio");
   }
      total=total+total1;
}
 } System.out.println("total amount:"+total);
}
   public static void main(String a[]) throws ClassNotFoundException, SQLException{
     Connectivity con=new Connectivity();
     con.connect();

   }
}
```

OUTPUT:

#	TV	MOBILE	RADIO
1	10000	5000	3000

Select the item:
1.Television
2.Mobile
3.Radio
enter the no of items:
2
enter item id:
1
enter item id:
2
total amount:15000

7. Explain Socket, ServerSocket, InetAddress classes. Write a java program to find an IP address of the machine on which the program runs.

Socket Programming:

Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server.

When the connection is made, the server creates a socket object on its end of the communication. The client and server can now communicate by writing to and reading from the socket.

The java.net.Socket class represents a socket, and the java.net.ServerSocket class provides a mechanism for the server program to listen for clients and establish connections with them.

The following steps occur when establishing a TCP connection between two computers using sockets:

- The server instantiates a ServerSocket object, denoting which port number communication is to occur on.

- The server invokes the accept() method of the ServerSocket class. This method waits until a client connects to the server on the given port.

- After the server is waiting, a client instantiates a Socket object, specifying the server name and port number to connect to.

- The constructor of the Socket class attempts to connect the client to the specified server and port number. If communication is established, the client now has a Socket object capable of communicating with the server.

- On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket.

After the connections are established, communication can occur using I/O streams. Each socket has both an OutputStream and an InputStream. The client's OutputStream is connected to the server's InputStream, and the client's InputStream is connected to the server's OutputStream.

TCP is a twoway communication protocol, so data can be sent across both streams at the same time. There are following usefull classes providing complete set of methods to implement sockets.

ServerSocket Class Methods:

The java.net.ServerSocket class is used by server applications to obtain a port and listen for client requests

The ServerSocket class has four constructors:

SN

Methods with Description

1 public ServerSocket(int port) throws IOException

Attempts to create a server socket bound to the specified port. An exception occurs if the port is already bound by another application.

2 public ServerSocket(int port, int backlog) throws IOException

Similar to the previous constructor, the backlog parameter specifies how many incoming clients to store in a wait queue.

3 public ServerSocket(int port, int backlog, InetAddress address) throws IOException

Similar to the previous constructor, the InetAddress parameter specifies the local IP address to bind to. The InetAddress is used for servers that may have multiple IP addresses, allowing the server to specify which of its IP addresses to accept client requests on

4 public ServerSocket() throws IOException

Creates an unbound server socket. When using this constructor, use the bind() method when you are ready to bind the server socket

If the ServerSocket constructor does not throw an exception, it means that your application has successfully bound to the specified port and is ready for client requests.

Here are some of the common methods of the ServerSocket class:

SN Methods with Description

1 public int getLocalPort()

Returns the port that the server socket is listening on. This method is useful if you passed in 0 as the port number in a constructor and let the server find a port for you.

2 public Socket accept() throws IOException

Waits for an incoming client. This method blocks until either a client connects to the server on the specified port or the socket times out, assuming that the time-out value has been set using the setSoTimeout() method. Otherwise, this method blocks indefinitely

3 public void setSoTimeout(int timeout)

Sets the time-out value for how long the server socket waits for a client during the accept().

4 public void bind(SocketAddress host, int backlog)

Binds the socket to the specified server and port in the SocketAddress object. Use this method if you instantiated the ServerSocket using the no-argument constructor.

When the ServerSocket invokes accept(), the method does not return until a client connects. After a client does connect, the ServerSocket creates a new Socket on an unspecified port and returns a reference to this new Socket. A TCP connection now exists between the client and server, and communication can begin.

Socket Class Methods:

The java.net.Socket class represents the socket that both the client and server use to communicate with each other. The client obtains a Socket object by instantiating one, whereas the server obtains a Socket object from the return value of the accept() method.

The Socket class has five constructors that a client uses to connect to a server:

SN Methods with Description

1 public Socket(String host, int port) throws UnknownHostException, IOException.

This method attempts to connect to the specified server at the specified port. If this constructor does not throw an exception, the connection is successful and the client is connected to the server.

2 public Socket(InetAddress host, int port) throws IOException

This method is identical to the previous constructor, except that the host is denoted by an InetAddress object.

3 public Socket(String host, int port, InetAddress localAddress, int localPort) throws IOException.

Connects to the specified host and port, creating a socket on the local host at the specified address and port.

4 public Socket(InetAddress host, int port, InetAddress localAddress, int localPort) throws IOException.

This method is identical to the previous constructor, except that the host is denoted by an InetAddress object instead of a String

5 public Socket()

Creates an unconnected socket. Use the connect() method to connect this socket to a server.

When the Socket constructor returns, it does not simply instantiate a Socket object but it actually attempts to connect to the specified server and port.

Some methods of interest in the Socket class are listed here. Notice that both the client and server have a Socket object, so these methods can be invoked by both the client and server.

SN Methods with Description

1 public void connect(SocketAddress host, int timeout) throws IOException

This method connects the socket to the specified host. This method is needed only when you instantiated the Socket using the no-argument constructor.

2 public InetAddress getInetAddress()

This method returns the address of the other computer that this socket is connected to.

3 public int getPort()

Returns the port the socket is bound to on the remote machine.

4 public int getLocalPort()

Returns the port the socket is bound to on the local machine.

5 public SocketAddress getRemoteSocketAddress()

Returns the address of the remote socket.

6 public InputStream getInputStream() throws IOException

Returns the input stream of the socket. The input stream is connected to the output stream of the remote socket.

7 public OutputStream getOutputStream() throws IOException

Returns the output stream of the socket. The output stream is connected to the input stream of the remote socket

8 public void close() throws IOException

Closes the socket, which makes this Socket object no longer capable of connecting again to any server

InetAddress Class Methods:

This class represents an Internet Protocol (IP) address. Here are following usefull methods which you would need while doing socket programming:

SN Methods with Description

1 static InetAddress getByAddress(byte[] addr)

Returns an InetAddress object given the raw IP address .

2 static InetAddress getByAddress(String host, byte[] addr)

Create an InetAddress based on the provided host name and IP address.

3 static InetAddress getByName(String host)

Determines the IP address of a host, given the host's name.

4 String getHostAddress()

Returns the IP address string in textual presentation.

5 String getHostName()

Gets the host name for this IP address.

6 static InetAddress InetAddress getLocalHost()

Returns the local host.

7 String toString()

Converts this IP address to a String.

Example:

java program to find an IP address of the machine on which the program runs.

```
package javaapplication2;

import java.util.*;
```

```
import java.lang.*;

import java.net.*;

 public class JavaApplication2 {

 public static void main(String args[]) {

      try{

         InetAddress ownIP=InetAddress.getLocalHost();

         System.out.println("IP of my system is := "+ownIP.getHostAddress());

      }catch (Exception e){

         System.out.println("Exception caught ="+e.getMessage())

}}}
```

8. Write a client program to send any string from its standard input to the server program. The server program reads the string, finds number of characters and digits and sends it back to client program. Use connection-oriented or connection-less communication.

SERVER:

```
import java.io.*;

import java.util.*;

class Server

{

  public static void main(String args[])

        {

              ServerSocket ss;

              Socket s;

              PrintStream ps;

              DataInputStream dis;

              String str;

              try
```

```
                {
                ss=new ServerSocket(8020);
                s=ss.accept();
                ps=new PrintStream(s.getOutputStream());
                dis=new DataInputStream(s.getInputStream());
                str=dis.readLine();
                System.out.println("string from client is "+str);
                System.out.println("length of the string is "+str.length());
          ps.println(str.length());
                ps.close();}
                catch(IOException e)
                {
                        System.out.println("The exception is :"+e);
                }}}
```

CLIENT:

```
import java.net.*;
import java.io.*;
class Stringlength
{
public static void main (String args[])
        {
                Socket soc;
                DataInputStream dis;
                String str;
```

```
PrintStream ps;
try
{
        InetAddress ia=InetAddress.getLocalHost();
        soc=new Socket(ia,8020);
        dis=new DataInputStream(soc.getInputStream());
ps=new PrintStream(soc.getOutputStream());
ps.println("hello");
        System.out.println("word requested is hello");
        String len=dis.readLine();
        System.out.println("len of the string is"+len);
}
catch(IOException e)
{
        System.out.println("THE EXCEPTION is :"+e);

}}}
```

OUTPUT:

SERVER:

```
C:\java>java Server
string from client is hello
length of the string is 5
```

CLIENT:

```
C:\java>java Stringlength
string from client is hello
```

length of the string is 5

9. (i) what is URL? what are its usual constituents?(6)

A URL is actually a link, which connects you to a website. It is also called as a Uniform Resource Locator. This is a unique address for a file that is accessible on the Internet. eg. http://www.website.com/products/laptop.html A common way to get to a web site is to enter the URL of its home page file in your Web browser's address line. The target website is contacted using a hypertext transfer protocol, which identifies the website and transfer the contents to the end user. The Uniform Resource Locator was created in 1994 by Tim Berners-Lee, Marc Andreessen, Mark P. McCahill, Alan Emtage, Peter J. Deutsch and Jon Postel, as part of the URI. The URL consists of various components to describe the full length of it.

Every URL consists of the 1) the scheme name or protocol followed by a colon, then depending on scheme, 2) a hostname (server name or alternatively an IP address), 3) a port number, the path of the resource to be fetched or the program to be run, then, for programs such as Common Gateway Interface (CGI) scripts, 4) a query string and with HTML documents an anchor (optional) for where the page should start to be displayed.

The basic components in a URL are:

http: //www.website.com /products/ laptop.html

protocol server name directory name page or file name

1) Protocol comes first and ends with a colon eg. http:

2) Server name comes then, prefixed with a double slash eg. //www.website.com

Server name further divided into:

a) Server's name eg. www

b) Domain name eg. website.com

3) Directory which comes next with slashes before and after eg. /products/

4) Actual Page / file name comes last eg. laptop.html

A URL for a particular image on a web site will look like this:

http://searchnetworking.website.com/animals/images/koala.gif

A URL for a file meant to be downloaded using the File Transfer Protocol (FTP) would require that the "ftp" protocol be specified like this:

ftp://www.somecompany.com/whitepapers/widgets.ps

A URL to email a message to someone would be like this:

mailto:somebody@mail.com

ABSOLUTE URL AND RELATIVE URL

Absolute URL

An Absolute URL is, thus, something that is independent or free from any relationship. When you use an absolute URL, you point directly to a file. Hence, an absolute URL specifies theexact location of a file/directory on the internet. It also follows that each absolute URL is unique

Eg: http://www.webdevelopersnotes.com/images/email.gif

specifies an image file email.giflocated in the images directory, under www.webdevelopersnotes.com domain name.

Relative URL

A relative URL points to a file/directory in relation to the present file/directory

Using an Absolute URL in an <img> tag

Using a Relative URL in an <IMG> tag

.

We, thus, instruct the browser to first go one level up (i.e. to the document root) and then move to the images directory and pick up the file email.gif.

The two peroids (..) instruct the server to move up one directory (which is the root directory)

Adv:

- It is easy to transfer a web site from one domain name to another.
- Also, relative URLs would be shorter than absolute URLs and hence the file size of the web page would reduce .

(ii) write a simple java program that connects to HTTP servers to fetch(use URL and URL connection classes)

Get connected with web server by using sock.getInetAddress() method of net.Socket class.

Program:

```
import java.net.InetAddress;

import java.net.Socket;

public class cli {

  public static void main(String[] args) {

    try {

      InetAddress addr;

      Socket sock = new Socket("www.google.com", 80);

      addr = sock.getInetAddress();

      System.out.println("Connected to " + addr);

      sock.close();

    }

catch (java.io.IOException e) {

     System.out.println("Can't connect to " + args[0]);

      System.out.println(e);

    }

  }
```

```
}
```

Result:

The above code sample will produce the following result.

```
Connected to www.google.com/74.125.130.99
```

10. Develop a socket program to send a set of sales data (use your own sales figures) one by one to the server. Write the corresponding server side program to calculate the commission for each sales man and return the result to the client.

Server:

```
import java.io.*;

import java.net.ServerSocket;

import java.net.Socket;

public class Salesserver {

   public static void main(String args[])

        {

                ServerSocket ss;

                Socket s;

                PrintStream ps;

                DataInputStream dis;

                int str;

                try

                {

                ss=new ServerSocket(8020);

                s=ss.accept();

                ps=new PrintStream(s.getOutputStream());

                dis=new DataInputStream(s.getInputStream());
```

```
                str=Integer.parseInt(dis.readLine());
                System.out.println("no of salesman:"+str);
              for(int i=0;i<str;i++)
              {
               int sale=Integer.parseInt(dis.readLine());
               System.out.println("no of sales for salesman"+i+1+":"+sale);
               System.out.println("commission for salesman"+(i+1)+":"+(sale*100));
               ps.println(sale);
               ps.println(sale*100);
              }
                ps.close();

                }
                catch(IOException e)
                {
                        System.out.println("The exception is :"+e);
                }
        }

}
```

Client:

```
import java.net.*;
```

```
import java.io.*;
import java.util.Scanner;
public class Salesclient
{
        public static void main (String args[])
        {
                Socket soc;
                DataInputStream dis;
                String str;
                PrintStream ps;
                try
                {
                        InetAddress ia=InetAddress.getLocalHost();
                        soc=new Socket(ia,8020);
                        dis=new DataInputStream(soc.getInputStream());
              ps=new PrintStream(soc.getOutputStream());
              System.out.println("enter no of salesman");
              Scanner sn=new Scanner(System.in);
              int n=sn.nextInt();
              ps.println(n);
                        for(int i=0;i<n;i++){
                 System.out.println("enter sale"+(i+1));
                 int m=sn.nextInt();
                 ps.println(m);
                       int len=0;
```

```
                    len=Integer.parseInt(dis.readLine());
                     System.out.println("commision for salesman"+(i+1)+":"+(len*100));
                  }
                   }
                   catch(IOException e)
                   {
                        System.out.println("THE EXCEPTION is :"+e);
                   }
          }
}
```

OUTPUT:

SERVER:

```
C:\java>java Salesserver
no of salesman:2
no of sales for salesman01:2
commission for salesman1:200
no of sales for salesman11:3
commission for salesman2:300
```

CLIENT:

```
C:\java>java Salesserver
no of salesman:2
no of sales for salesman01:2
```

commission for salesman1:200

no of sales for salesman11:3

commission for salesman2:300

UNIT IV
APPLETS
PART- A (2 Marks)

1. **What is an Applet?**
   Applet is a Java application, which can be executed in JVM, enabled web browsers.
2. **What are methods available in the Applet class?**
   - init - To *initialize* the applet each time it's loaded (or reloaded).
   - start - To *start* the applet's execution, such as when the applet's loaded or when the user revisits a page that contains the applet.
   - stop - To *stop* the applet's execution, such as when the user leaves the applet's page or quits the browser.
   - destroy - To perform a *final cleanup* in preparation for unloading.
3. **Distinguish between paint and update method?**
   paint is basic display method. Many applets implement the paint method to draw the applet's representation within a browser page. Update is a method that can use along with paint to improve drawing performance.

4. **What is AWT?**
   A collection of graphical user interface (GUI) components that were implemented using native-platform versions of the components. These components provide that subset of functionality which is common to all native platforms. Largely supplanted by the Project Swing component set.
5. **List out some UI components available in AWT?**
   - Buttons (java.awt.Button)
   - Checkboxes (java.awt.Checkbox)
   - Single-line text fields (java.awt.TextField)
   - Larger text display and editing areas (java.awt.TextArea)
   - Labels (java.awt.Label)
   - Lists (java.awt.List)
   - Pop-up lists of choices (java.awt.Choice)
   - Sliders and scrollbars (java.awt.Scrollbar)
   - Drawing areas (java.awt.Canvas)
   - Menus (java.awt.Menu, java.awt.MenuItem, java.awt.CheckboxMenuItem)
   - Containers (java.awt.Panel, java.awt.Window and its subclasses)
6. **Write some methods, which are used to add UI components in Applet?**
   - add - Adds the specified Component.
   - remove - Removes the specified Component.
   - setLayout - Sets the layout manager.
7. **Write the Html code to load an Applet in the browser?**
   <APPLET CODE=*AppletSubclass*.class WIDTH=*anInt*
   HEIGHT=*anInt*>
   </APPLET>

8. How parameters are passed to an Applet?

The parameters are passed to an Applet by using param tag

<APPLET CODE=*AppletSubclass*.class WIDTH=*anInt*
HEIGHT=*anInt*>
<PARAM NAME=*parameter1Name* VALUE=*aValue*>
<PARAM NAME=*parameter2Name* VALUE=*anotherValue*>
</APPLET>

9. Write the attributes of the Applet tag?

< APPLET
[CODEBASE = *codebaseURL*]
CODE = *appletFile*
[ALT = *alternateText*]
[NAME = *appletInstanceName*]
WIDTH = *pixels*
HEIGHT = *pixels*
[ALIGN = *alignment*]
[VSPACE = *pixels*]
[HSPACE = *pixels*]
>
[< PARAM NAME = *appletParameter1* VALUE = *value*
>]
[< PARAM NAME = *appletParameter2* VALUE = *value*
>]
. . .
[*alternateHTML*]
</APPLET>

10. What is the difference between applications and applets?-

a)Application must be run on local machine whereas applet needs no explicit installation on local machine.

b)Application must be run explicitly within a java-compatible virtual machine whereas applet loads and runs itself automatically in a java-enabled browser.

c)Application starts execution with its main method whereas applet starts execution with its init method.

d)Application can run with or without graphical user interface whereas applet must run within a graphical user interface.

11. When do you use codebase in applet?

When the applet class file is not in the same directory, codebase is used.

12. What is the lifecycle of an applet?-

- init() method - Can be called when an applet is first loaded
- start() method - Can be called each time an applet is started.
- paint() method - Can be called when the applet is minimized or maximized.
- stop() method - Can be used when the browser moves off the applet's page.
- destroy() method - Can be called when the browser is finished with the applet

13. Explain in brief the interaction between a web server and a Servlet.

Web Server is a machine that has a HTTPD service running. A web server is the one that handles HTTP requests and generates HTTP responses. A Servlet is a server-side entity for servicing HTTP requests.

14. How is session tracking achieved by URL rewriting?

- **URL Rewriting** can be used in place where we don't want to use cookies. It is used to maintain the session. Whenever the browser sends a request then it is always interpreted as a new request because http protocol is a stateless protocol as it is not persistent. Whenever we want that out request object to stay

alive till we decide to end the request object then, there we use the concept of session tracking. In session tracking firstly a session object is created when the first request goes to the server. Then server creates a token which will be used to maintain the session. The token is transmitted to the client by the response object and gets stored on the client machine. By default the server creates a cookie and the cookie get stored on the client machine

15. Explain the Servlet API life cycle methods in brief.

- init(): called when servlet is instantiated; must return before any other methods will be called. If intialization processing causes error it throw throw an exception called UnavailableException
- service(): method called directly by server when an HTTP request is received; this in turn calls doGet()
- destroy(): called when server shuts down(taking a servlet out of service

16. What is the purpose of cookies?

The main purpose of cookies is to identify users and possibly prepare customized Web pages for them. When you enter a Web site using cookies, you may be asked to fill out a form providing such information as your name and interests. This information is packaged into a cookie and sent to your Web browser which stores it for later use. The next time you go to the same Web site the server can use this information to present you with custom Web pages. Instead of seeing just a generic welcome page you might see a welcome page with your name on it.

17. What is the difference between doGet () and doPost ()?

doGet()	**doPost()**
In doGet() the parameters are appended to the URL and sent along with header information.	In doPost(), on the other hand will (typically) send the information through a socket back to the webserver and it won't show up in the URL bar.
The amount of information you can send back using a GET is restricted as URLs can only be 1024 characters.	You can send much more information to the server this way - and it's not restricted to textual data either. It is possible to send files and even binary data such as serialized Java objects!
doGet() is a request for information; it does not (or should not) change anything on the server. (doGet() should be idempotent)	doPost() provides information (such as placing an order for merchandise) that the server is expected to remember
Parameters are not encrypted	Parameters are encrypted
doGet() is faster if we set the response content length since the same connection is used. Thus increasing the performance	doPost() is generally used to update or post some information to the server.doPost is slower compared to doGet since doPost does not write the content length
doGet() should be idempotent. i.e. doGet should be able to be repeated safely many times	This method does not need to be idempotent. Operations requested through POST can have side effects for which the user can be held accountable.

18. What is the difference between HttpServlet and GenericServlet?

Generic Servlet	**HttpServlet**

GenericServlet class is direct subclass of Servlet interface.	HttpServlet class is the direct subclass of Generic Servlet.
Generic Servlet is protocol independent.	HttpServlet is protocol dependent.
It handles all types of protocol like HTTP, SMTP, FTP etc	It handles only http protocol.
Generic Servlet only supports service() method. It handles only simple request	HttpServlet supports public void service(ServletRequest req,ServletResponse res) and protected void service(HttpServletRequest req,HttpServletResponse res).
Generic Servlet only supports service() method.	HttpServlet supports also doGet(),doPost(),doPut(),doDelete(),doHead(),doTrace(),doOptions()etc.

19.What are the difference between Parameters and Attributes?

Parameters	**Attributes**
A "parameter" is a form field name/value pair passed from the HTML side of the world	An "attribute" is a Java object name/value pair passed only through the internal Java Server processes.
Parameter value is a String	Attribute value is Object.
Parameters come from the client request.	Attributes are set by the server side
Parameters are read only i.e. generally can be retrieved, but not set. whereas	Attributes are read/write .You can also set attributes programmatically and retrieve them later. This is very useful in the MVC pattern.

20. How Cookie entry will be made?

Each entry is made up of

- A name value pair which stores whatever data you want to save.
- A expiry date, after which time the entry will be deleted.
- The web domain and path and that the entry should be associated.

21.What is inter-servlet communication?

It is the communication between servlets. There are many ways to communicate between servlets which includes request dispatching, Http redirect, Servlet chaining, Http request, shared session , request, or application objects, direct method invocation and shared static or instance variables.

22 .What are the types of authentication?

- **Http authentication:** User name and password protection.
- **Form based authentication**: Unauthorized access to the form is protected.

23. Write the purpose of URL rewriting

- In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource. We can send parameter name/value pairs using the following format:
- *url?name1=value1&name2=value2&??*

- A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand (&). When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server.

24. Write two basic differences between JSP and servlet.

JSP	Servlet
A JSP is typically oriented more towards displaying information Ex: Display a report	A servlet is more oriented towards processing information. Ex: Process a user submitted form
Coding JSP is easier	Coding JSP is Complex
JSP is based in Java, an object oriented language.	Servlets are designed to work with in request response processing model

25. What are implicit objects? List them?

Certain objects that are available for the use in JSP documents without being declared first. These objects are parsed by the JSP engine and inserted into the generated servlet. The implicit objects re listed below

- request
- response
- pageContext
- session
- application
- out
- page
- exception

26. What are the different scope values for the <jsp:useBean>?

- page
- request
- session
- application

PART- B(16 Marks)

1. Describe the AWT controls in Java

```
import java.awt.*;
import javax.swing.*;
class sample1
{
public static void main(String[] args)
{
JButton a1=new JButton("ok");
JButton b1=new JButton("submit");
JButton c1=new JButton("hellow");
JButton d1=new JButton("world");
JButton e1=new JButton("java");
JFrame a=new JFrame("Example AWT");

Container panel = new JPanel(new FlowLayout(FlowLayout.CENTER));

panel.add(a1);
panel.add(b1);
panel.add(c1);
panel.add(d1);
panel.add(e1);
```

```
a.add(panel);
a.setSize(500,500);
a.setVisible(true);
}

}
```

Border Layout

```
import java.awt.*;
import javax.swing.*;
class sample
{
public static void main(String[] args)
{

JButton a1=new JButton("ok");
JButton b1=new JButton("submit");
JButton c1=new JButton("hellow");
JButton d1=new JButton("world");
JButton e1=new JButton("java");

JFrame a=new JFrame("Example AWT");

Container panel = new JPanel(new BorderLayout());

panel.add(a1,BorderLayout.NORTH);
panel.add(b1,BorderLayout.SOUTH);
panel.add(c1,BorderLayout.WEST);
panel.add(d1,BorderLayout.EAST);
panel.add(e1,BorderLayout.CENTER);

a.add(panel);
a.setSize(500,500);
a.setVisible(true);
}

}
```

Menu Creation

```
public void MenuCreation() {
        MenuBar menubar=new MenuBar();
        Menu file=new Menu("File");
        Menu shape=new Menu("Shape");
        Menu about=new Menu("About");
file.add(new MenuItem("Exit",new MenuShortcut(kcontrolX))).addActionListener(new WindowHandler());
shape.add(new MenuItem("Rectangle",new MenuShortcut(kcontrolR))).addActionListener(new
    WindowHandler());
shape.add(new MenuItem("Triangle",new MenuShortcut(kcontrolT))).addActionListener(new
    WindowHandler());
```

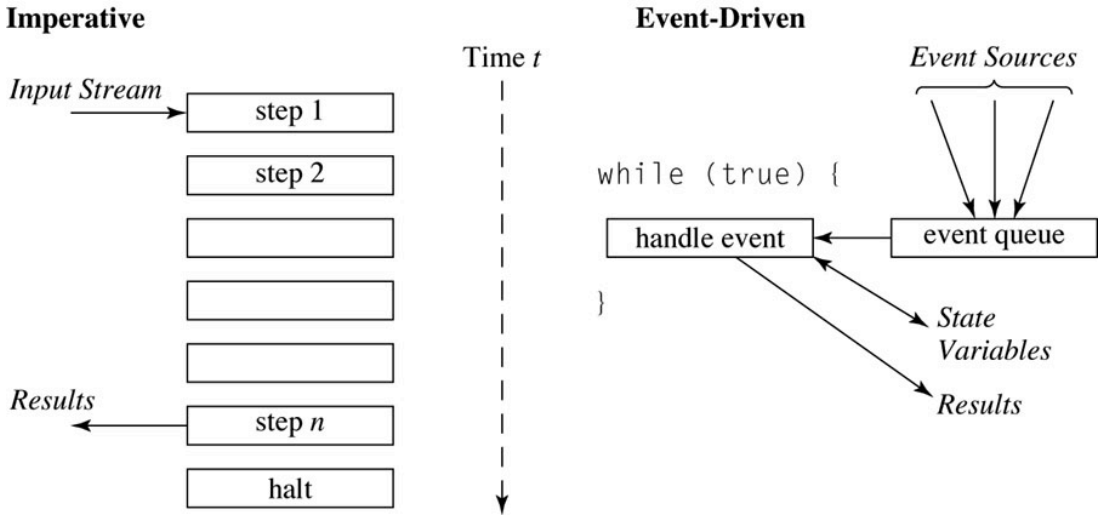
```
shape.add(new MenuItem("Circle",new MenuShortcut(kcontrolC))).addActionListener(new
    WindowHandler());
shape.add(new MenuItem("Polygon",new MenuShortcut(kcontrolP))).addActionListener(new
    WindowHandler());
shape.add(new MenuItem("Draw Polygon",new MenuShortcut(kcontrolD))).addActionListener(new
    WindowHandler());
about.add(new MenuItem("About",new MenuShortcut(kcontrolA))).addActionListener(new
    WindowHandler());
      menubar.add(file);
      menubar.add(shape);
      menubar.add(about);
      if(null==this.getMenuBar())
      {
            this.setMenuBar(menubar);
      }     }
```

2. What is event handling in java? List out the available event classes and listener interfaces with suitable example. (16)

Once the program was complete we would:
i) provide input
ii) run the program
iii) display the answer(s)
Things always happened in the same sequence.

The program always terminated when the process was complete (except for some Prolog programs we will not mention).

Event-driven programs do not have a set sequence of instructions to execute. They also do not have a predetermined finish. The most common example of event-driven programming is found in ***graphical user interfaces*** (GUIs):



EVENT DRIVEN PROGRAM:
i) Input to event-driven programs come from ***event sources*****:** sensors, input devices, objects on a web page.

ii) Events occur asynchronously and are placed in an ***event queue*** as they arise.

iii) Events are removed from the event queue and processed ("***handled***") by the program's main processing loop.

As a result of handling an event the program may produce output or modify the value of a ***state variable***. There is no predefined starting or stopping point. Java provides support for event-driven programming through certain ***classes*** and ***methods*** that can be used to design program interaction.

EVENT HANDLERS:

To respond to events we need to implement special methods called ***handlers***.

Each class of events predefines the names of the handlers that can be written for it

Event Type	Event Source	Handler Required
Button selection	Button	actionPerformed
Text entry	TextField	actionPerformed
Menu selection	Choice	itemStateChanged
Mouse		mousePressed, mouseReleased, mouseClicked, mouseExited, mouseEntered
Mouse motion		mouseDragged, mouseMoved

Different types of Interfaces in Event Handling:

1) Mouse Listener
2) Mouse Motion Listener
3) Window Listener
4) Item Listener

Different types of classes in Event Handling:

1. ComponentAdapter
2. ContainerAdapter
3. FocusAdapter
4. KeyAdapter
5. MouseAdapter
6. MouseMotionAdapter
7. WindowAdapter

3. Design and write a java program to display the buttons from a digital telephone. As a number is dialed play the respective tone. After a six-digit number is input, the program should play a ringtone. The termination of this call should be initiated after playing this tone for 20 seconds.

```
import java.awt.Container;

import java.awt.event.ActionEvent;
```

```
import java.awt.event.ActionListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.*;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.SwingConstants;
import javax.swing.UIManager;

public class Scientific extends JFrame implements ActionListener{
    JTextField tfield;
    double temp,temp1,result,a;
    static double m1,m2;
    int k=1,x=0,y=0,z=0;
    char ch;
    JButton b1,b2,b3,b4,b5,b6,b7,b8,b9,zero,b#,b*;
    Container cont;
    JPanel textpanel,buttonpanel;

    Scientific(){
      cont=getContentPane();
      cont.setLayout(new BorderLayout());
      JPanel textpanel=new JPanel();
      tfield=new JTextField(25);
      tfield.setHorizontalAlignment(SwingConstants.RIGHT);

      textpanel.add(tfield);
```

```
buttonpanel=new JPanel();
buttonpanel.setLayout(new GridLayout(8,4,2,2));
boolean t=true;

b1=new JButton("1");
buttonpanel.add(b1);
b1.addActionListener(this);

b2=new JButton("2");
buttonpanel.add(b2);
b2.addActionListener(this);

b3=new JButton("3");
buttonpanel.add(b3);
b3.addActionListener(this);
b4=new JButton("4");
buttonpanel.add(b4);
b4.addActionListener(this);

b5=new JButton("5");
buttonpanel.add(b5);
b5.addActionListener(this);

b6=new JButton("6");
buttonpanel.add(b6);
b6.addActionListener(this);

b7=new JButton("7");
buttonpanel.add(b7);
```

```
b7.addActionListener(this);

b8=new JButton("8");
buttonpanel.add(b8);
b8.addActionListener(this);

b9=new JButton("9");
buttonpanel.add(b9);
b9.addActionListener(this);


zero=new JButton("0");
buttonpanel.add(zero);
zero.addActionListener(this);


b#=new JButton("#");
buttonpanel.add(plus);
plus.addActionListener(this);


b*=new JButton("*");
buttonpanel.add(mul);
mul.addActionListener(this);

cont.add("Center",buttonpanel);
cont.add("North",textpanel);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);   }
```

```
        public void actionPerformed(ActionEvent e) {


        if(s.equals("1")){


  Sound()
  tfield.setText(tfield.getText()+"1");
  String s=tfield.getText()
  Int a=s.length();
  If(a>=6)
  {
  Ringtone()
  exit(0):
  }
        }
        if(s.equals("2")){


  Sound()
  tfield.setText(tfield.getText()+"2");
  String s=tfield.getText()
  Int a=s.length();
  If(a>=6)
  {
  Ringtone()
  exit(0):


  }
        }
        if(s.equals("3")){
  Sound()
```

```
tfield.setText(tfield.getText()+"3");
String s=tfield.getText()
Int a=s.length();
If(a>=6)
{
Ringtone();
exit(0):

}
        }
        if(s.equals("4")){
Sound()
tfield.setText(tfield.getText()+"4");
String s=tfield.getText()
Int a=s.length();
If(a>=6)
{
Ringtone();
exit(0):
}    }
        if(s.equals("5")){

Sound()
tfield.setText(tfield.getText()+"5");
String s=tfield.getText()
Int a=s.length();
If(a>=6)
{
Ringtone();
```

```
exit(0):


}
        }
        if(s.equals("6")){
Sound()
tfield.setText(tfield.getText()+"6");
String s=tfield.getText()
Int a=s.length();
If(a>=6)
{
Ringtone();
exit(0):


}
}
        if(s.equals("7")){
Sound()
tfield.setText(tfield.getText()+"7");
String s=tfield.getText()
Int a=s.length();
If(a>=6)
{
Ringtone();
exit(0):
}


        }
        if(s.equals("8")){
```

```
Sound()
tfield.setText(tfield.getText()+"8");
String s=tfield.getText()
Int a=s.length();
If(a>=6)
{
Ringtone();
exit(0):

}       }
        if(s.equals("9")){
Sound()
tfield.setText(tfield.getText()+"9");
String s=tfield.getText()
Int a=s.length();
If(a>=6)
{
Ringtone();
exit(0):

}
     }
        if(s.equals("0")){
Sound()
tfield.setText(tfield.getText()+"0");
String s=tfield.getText()
Int a=s.length();
If(a>=6)
```

```
{
Ringtone();
exit(0):

}

        }


        if(s.equals("*")){
                }
        if(s.equals("#")){


    }
    public static void main(String args[]){
      try{
    UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");          }
      catch(Exception e){
      }
      Scientific f=new Scientific();
      f.setTitle("Phone");
      f.pack();
      f.setVisible(true);        }}
```

4. Discuss in detail about JSP Expression Language (EL) and JSP Markup.

JSP MARKUP

- Three types of markup elements:
  - Scripting
  - Directive
  - Action

2.1 SCRIPTING:

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

- Scriptlet tag (<% …%>)
- Expression tag (<%= statement %>)
- Declaration tag (<%! int data=50; %>)

2.2 DIRECTIVE

- Instructs JSP translator

PAGE directive:

The **page** directive is used to provide instructions to the container that pertain to the current JSP page. You may code page directives anywhere in your JSP page. By convention, page directives are coded at the top of the JSP page.

Syntax of page directive:

<%@ page attribute="value" %>

You can write XML equivalent of the above syntax as follows:

<jsp:directive.page attribute="value" /> **Attributes:**

buffer	Specifies a buffering model for the output stream.
contentType	Defines the character encoding scheme.
errorPage	Defines the URL of another JSP that reports on Java unchecked runtime exceptions.
extends	Specifies a superclass that the generated servlet must extend
import	Specifies a list of packages or classes for use in the JSP as the Java import statement does for Java classes.
language	Defines the programming language used in the JSP page.
session	Specifies whether or not the JSP page participates in HTTP sessions

INCLUDE DIRECTIVE :

The **include** directive is used to includes a file during the translation phase. This directive tells the container to merge the content of other external files with the current JSP during the translation phase. You may code *include* directives anywhere in your JSP page.

The general usage form of this directive is as follows:

<%@ include file="relative url" >

You can write XML equivalent of the above syntax as follows:

<jsp:directive.include file="relative url" />

ACTION:

- Standard: provided by JSP itself
- Custom: provided by a tag library such as JSTL.

- **JSTL** is divided into several functional areas, each with its own namespace:

TABLE 8.6: JSTL functional areas.

Functional Area	Namespace Name Suffix
Core	`core`
XML Processing	`xml`
Functions	`functions`
Database	`sql`
Internationalization	`fmt`

5. Discuss in Detail About JSP Tag Libraries.

ACTION:

- Standard: provided by JSP itself
- Custom: provided by a tag library such as JSTL.

- **JSTL** is divided into several functional areas, each with its own namespace:

TABLE 8.6: JSTL functional areas.

Functional Area	Namespace Name Suffix
Core	`core`
XML Processing	`xml`
Functions	`functions`
Database	`sql`
Internationalization	`fmt`

Namespace prefix is

`http://java.sun.com/jsp/jstl/`

JSTL CORE ACTIONS:

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

TABLE 8.7: Some JSTL core actions.

Action	Purpose
`set`	Assign a value to a scoped variable, creating the variable if necessary
`remove`	Destroy a scoped variable
`out`	Write data to `out` implicit object, escaping XML special characters
`url`	Create a URL with query string
`if`	Conditional (if-then) processing
`choose`	Conditional (if-then-elseif) processing
`forEach`	Iterate over a collection of items

- Common variables:

– var

- Represents name of a scoped variable that is assigned to by the action
- Must be a string literal, not an EL expression

– scope

- Specifies scope of scoped variable as one of the literals page, request, session, or application

set action

– Setting (and creating) a scoped variable

out action

– Normally used to write a string to the out JSP implicit object

– Automatically escapes XML special characters

if action

– General form includes scoped variable to receive test value

```
<c:set var="age" value="20" scope="session"></c:set>

<c:if test="${age ge 18}" var="x">
<h3><c:out value="WELCOME"></c:out></h3>
</c:if>

<h3><c:out value="${x}"></c:out></h3>
```

<c:if test="${x}">
<h3><c:out value="WELCOME"></c:out></h3>
</c:if>

Output:

WELCOME
true
WELCOME

remove action

– Only attributes are var and scope
– Removes reference to the specified scoped variable from the scope object

<c:set var="x" value="10" scope="session"></c:set>
<c:set var="y" value="20" scope="session"></c:set>
<h3>Product :<c:out value="${x*y}"></c:out></h3>
<c:remove var="x" scope="session"/>
<c:remove var="y" scope="session"/>
<h3>Product :<c:out value="${x*y}"></c:out></h3>

Output:

Product :200
Product :0

choose action:

<c:set var="salary" scope="session" value="5000"/>
Your salary is : <c:out value="${salary}"/>
<c:choose>
<c:when test="${salary > 1000}">
Salary is very good.
</c:when>
<c:otherwise>
Salary is very low
</c:otherwise>
</c:choose>

Output:

Your salary is : 5000
Salary is very good.

forEach action:

Used to increment a variable (writes 2, 4, 6, 8 to the out object)

<c:forEach begin="1" end="10" step="4" var="x">
Begin Index value :${x}
</c:forEach>

Output:

Begin Index value :1
Begin Index value :5
Begin Index value :9

url action

– value attribute is a URL to be written to the out JSP implicit object
– URL's beginning with / are assumed relative to context path
– param elements can be used to define parameters that will be URL encoded

curl.jsp:

<c:url value="/URLTest.jsp" var="x" >
<c:param name="d1" value="SCJP"/>
<c:param name ="d2" value="SCWCD"/>
</c:url>

```
<h1>The modified url : ${x},</h1><br>
<a href="${x}">click Here </a>
```

Output:

The modified url : /JSTL/URLTest.jsp?d1=SCJP&d2=SCWCD,

URLTest.jsp

```
<h2> This is URLTest JSP  </h2>
      <h2>First Parameter : : :${param.d1}</h2>
    <h2>Second Parameter : : :${param.d2}</h2>
```

Output:

URL:

This is URLTest JSP

First Parameter : : :SCJP

Second Parameter : : :SCWCD

JSTL FUNCTIONS (<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>**):**

– Function name followed by parenthesized, comma-separated list of EL expression arguments.

– Tag libraries define all functions

– Function names usually include a namespace prefix associated with the tag library.

<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>

${fn:toLowerCase(param.name)}

JSTL SQL (<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/sql" %>**):**

The JSTL SQL tag library provides tags for interacting with relational databases (RDBMSs) such as Oracle, mySQL, or Microsoft SQL Server.

The **<sql:query>** tag executes an SQL SELECT statement and saves the result in a scoped variable.

```
<sql:query dataSource="${snapshot}" var="result">
SELECT * from Emp;
</sql:query>
```

The <sql:setDataSource> tag sets the data source configuration variable or saves the data-source information in a scoped variable

```
<sql:setDataSource var="snapshot" driver="sun.jdbc.odbc.JdbcOdbcDriver"
   url="jdbc:odbc:NEO"//Data Source Name
   user=""  password=""/>
```

FORMATTING TAGS: (<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>)

The JSTL formatting tags are used to format and display text, the date, the time, and numbers for internationalized Web sites. Following is the syntax to include Formatting library in your JSP:

XML TAGS (<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>**)**

The JSTL XML tags provide a JSP-centric way of creating and manipulating XML documents. Following is the syntax to include JSTL XML library in your JSP.

The JSTL XML tag library has custom tags for interacting with XML data. This includes parsing XML, transforming XML data, and flow control based on XPath expressions.

6. Explain the architecture of Servlet in detail with a sample Servlet program.

Servlet Architecture overview:

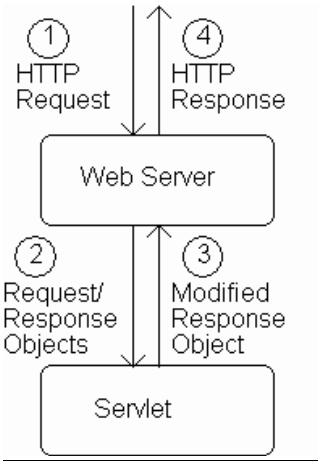
- The combination of HTML JavaScript and DOM is sometimes referred to as Dynamic HTML (DHTML)
- Web pages that include scripting are often called dynamic pages
- A simple HTML document without scripting known as static document

Web server response can be static or dynamic

- **Static:** When browser has requested an HTML document
- HTML document is retrieved from the file system and returned to the client
- Web server not responsible for generating content of response

- Find and send the content
- **Dynamic:** HTML document is generated by a program in response to an HTTP request
  Eg: Visiting a search engine website
- Java servlets are one technology for producing dynamic server responses
- Servlet is a Java class instantiated by the server to produce a dynamic response
- A particular method is called on this instance when server receives HTTP request
- Code in the servlet method can obtain information about the request and produce information to be included in the response
- It is done by calling methods on parameter objects passed to the method
- When the servlet returns control to the server , it creates a response from the information dynamically generated by servlet

Web Server-Servlet Interaction



Web server Operation

1. When an HTTP request is received by a servlet capable server
   1. It determines based on URL whether the request handled by the servlet
   2. Any URL in which the path component begins with servlet
2. The servlet determines from URL which servlet handle the request and calls a method on that servlet
   1. Two parameters are passed to the method
   2. An object implementing the HttpservletRequest interface
   3. An object implementing the HttpservletResponse interface
   4. Both are defined as part of Java Servlet API which is implemented by server
- First method to access the information contained in the request message
- Second object to record the information that the servlet wishes to include in the HTTP response message

3. Servlet method executes by calling methods on
   HttpservletRequest and HttpservletResponse objects passed to it.
- The information stored in the latter object by the servlet method includes an HTML document along with some HTTP header information
- When servlet method finishes its processing it returns control to the server

4. The server formats the information stored in the HttpservletResponse object by the servlet into an HTTP response message then send to the client

INDEX.JSP:

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
```

```
        <FORM action ="Servlet2" method="get">
            USERNAME:<input type="text" name="user"/>
            PASSWORD:<input type="password" name="pass"/>
            <INPUT TYPE="SUBMIT" VALUE="SUBMIT"/>
        </FORM>
    </body>
</html>
```

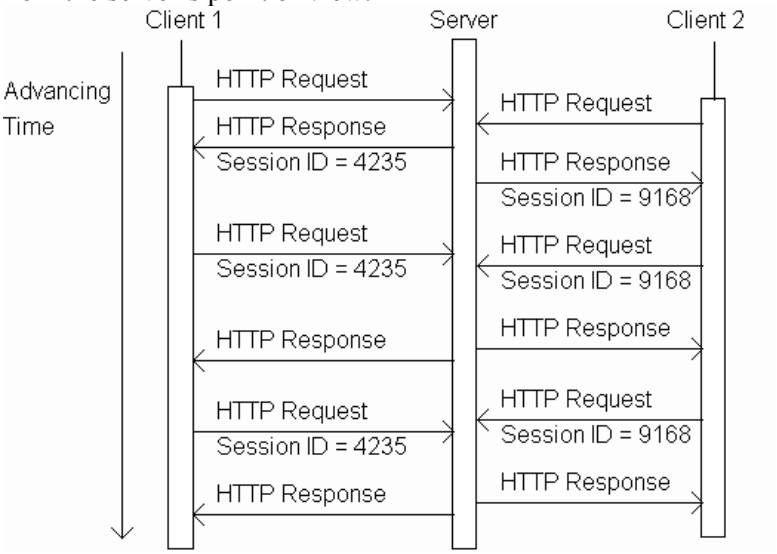
SERVLET2.JAVA:

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet(name="Servlet2", urlPatterns={"/Servlet2"})
public class Servlet2 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

            String u=request.getParameter("user");
            String p=request.getParameter("pass");
            System.out.println("USER IS"+u);
            System.out.println("PASS IS"+p);
            if(u.equals("User1")&&p.equals("pass"))
            {
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>WELCOME :" + u + "</h1>");
            /*out.println("<h1>PASSWORD IS " + p + "</h1>");*/
            out.println("</body>");
            out.println("</html>");
            }
        else
        {
         out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>" + u + " is not a valid user</h1>");
            out.println("</body>");
            out.println("</html>");
        }       }       }
```

7. What is a session? Explain how client state is maintained using session and also explain about session tracking and session management using an example.

SESSIONS

- A session refers to the entire interaction between a client and a server from the time of the client's first request, which generally begins the session, to the time the session is terminated.
- The session could be terminated by the client's request, or the server could automatically close it after a certain period of time.
- Without session management, each time a client makes a request to a server, it's a brand new user with a brand new request from the server's point of view.



Server knows that all of these requests are from the same client. The set of requests is known as a *session*

INDEX.JSP:

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>JSP Page</title>
   </head>
   <body>
      <form action="FirstSession" method="post">
         <input type="submit" value="Welcome to Online Purchase"/>
      </form>
   </body>
</html>
```

FirstSession.java

```
public class FirstSession extends HttpServlet
{
  protected void doPost(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException
{
        response.setContentType("text/html;charset=UTF-8");
       PrintWriter out = response.getWriter();
       try
{
         HttpSession session = request.getSession();
         String id=session.getId();
         System.out.println(id);
```

```
      String ses_name=(String) session.getAttribute("ses_attr");
           System.out.println("Session_name:"+ses_name);
        if(session.isNew() || ses_name == null)
        {
                printSignInForm(out,"Greeting",id);
/*For URL Rewriting(Cookies Turned Off)
printSignInForm(out,response.encodeURL("Greeting"),id);
*/
        }
        else
        {
                welcomeBackPage(out,ses_name,id);
        }
       } finally {
          out.close();
       }
   }
public void printSignInForm(PrintWriter out, String action,String sess_id)
     {
       out.println("<Form method='post' action="+action+">");
       out.println("Session Id:" + sess_id);
                out.println("Please Sign in :");
       out.println("<input type=text name=signIn>");
       out.println("<br><input type=submit name=signin value=SignIn>");
     }
     private void welcomeBackPage(PrintWriter out,String signIn,String sess_id)
     {
                out.println("Session Id:" + sess_id);
       out.println("Hey you're <h3>" +signIn+ " </h3>Welcome back!!!");
     }
}
```

Greeting.java

```
public class Greeting extends HttpServlet {
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      try {
            String user = request.getParameter("signIn");
       HttpSession session = request.getSession();
                String id=session.getId();
       if(user != null)
       {
                printThanks(out,user,id);
                session.setAttribute("ses_attr",user);
       }
     }
finally
{
         out.close();
        }
   }
```

```
private void printThanks(PrintWriter out,String signIn,String sess_id)
    {
                out.println("Session Id:" + sess_id);
       out.println("Thanks for signing in, <h3> "+signIn+ " </h3>");
        out.println("<br> Please <a href=FirstSession>visit again !</a>");
/*For URL Rewriting(Cookies Turned Off)
out.println("<br> Please <a href=" + response.encodeURL("FirstSession") +">visit again !</a>");
    */
    }
}
```

HttpSession interface:

METHODS

Method	Description
public HttpSession getSession()	Will cause one session to be created.
public HttpSession getSession(boolean)	true = will cause one to be created; false = will return null (no session)
public Object getAttribute(String name)	Returns the object bound with the specified name in this session, or null if no object is bound under the name.
public void setAttribute(String name, Object value)	Binds an object to this session, using the name specified.
public void removeAttribute(String name)	Removes the object bound with the specified name from this session.

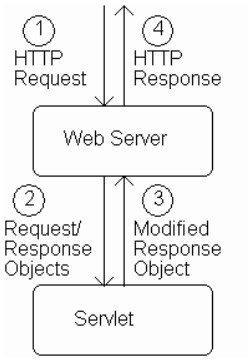
Session termination:

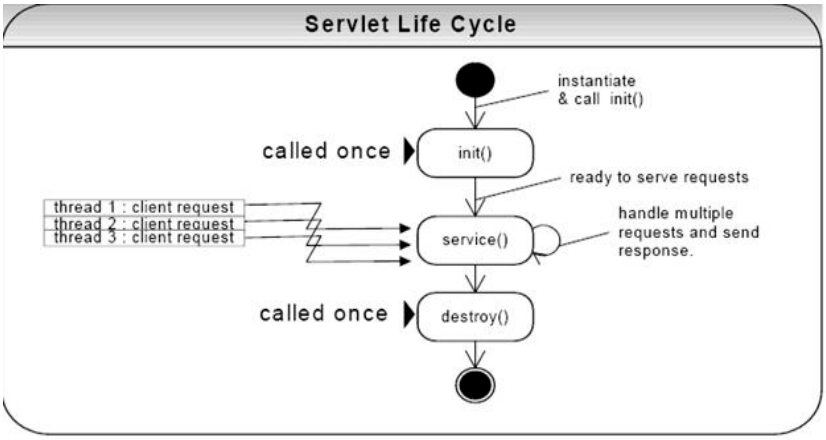
- By default, each session expires if a server-determined length of time elapses between a session's HTTP requests
  - Server destroys the corresponding session object
- Servlet code can:
  - Terminate a session by calling invalidate() method on session object
  - Set the expiration time-out duration (secs) by calling setMaxInactiveInterval(int)

8. Discuss in Detail about Servlet life cycle and Parameter Data.

Difference Between Static and Dynamic HTML?

- Static: HTML document is retrieved from the file system and returned to the client
- Dynamic: HTML document is generated by a program in response to an HTTP request
- Java servlets are one technology for producing dynamic server responses
- Servlet is a class instantiated by the server to produce a dynamic response





What are all the Servlet API life cycle methods

Servlet API life cycle methods

– init(): called when servlet is instantiated; must return before any other methods will be called

– service(): method called directly by server when an HTTP request is received; default service() method calls doGet() (or related methods covered later)

– destroy(): called when server shuts down

PARAMETER DATA:

- The request object (which implements HttpServletRequest) provides information from the HTTP request to the servlet
- One type of information is parameter data, which is information from the query string portion of the HTTP request

query string with one parameter

```
http://www.example.com/servlet/PrintThis?arg=aString
```

parameter name: arg

parameter value: aString

GET vs. POST method for forms:

– GET:

- Query string is part of URL
- Length of query string may be limited
- Recommended when parameter data is not stored but used only to request information (*e.g.*, search engine query)

– POST:

- Query string is sent as body of HTTP request
- Length of query string is unlimited
- Recommended if parameter data is intended to cause the server to update stored data
- Most browsers will warn you if they are about to resubmit POST data to avoid duplicate updates

9. Discuss URL Rewriting .Explain how if Differentiates from sessions and cookies

URL Rewriting

- Some users TURN OFF cookies and we need to use URL Rewriting.
- In Url rewriting no Cookies or Cookies Should be Blocked.
- Original (relative) URL:
  href="URLEncodedGreeting"
- URL containing session ID:
  href="URLEncodedGreeting;jsessionid=0157B9E85"
- Path parameter is treated differently than query string parameter

– Ex: invisible to getParameter()

INDEX.JSP:

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>JSP Page</title>
   </head>
   <body>
      <form action="FirstSession" method="post">
         <input type="submit" value="Welcome to Online Purchase"/>
      </form>
   </body>
</html>
```

FirstSession.java

```
public class FirstSession extends HttpServlet
{
  protected void doPost(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException
{
        response.setContentType("text/html;charset=UTF-8");
       PrintWriter out = response.getWriter();
       try
{
         HttpSession session = request.getSession();
         String id=session.getId();
         System.out.println(id);
     String ses_name=(String) session.getAttribute("ses_attr");
          System.out.println("Session_name:"+ses_name);
       if(session.isNew() || ses_name == null)
       {
                printSignInForm(out,"Greeting",id);
/*For URL Rewriting(Cookies Turned Off)
printSignInForm(out,response.encodeURL("Greeting"),id);
*/
       }
       else
       {
                welcomeBackPage(out,ses_name,id);
       }
      } finally {
         out.close();
      }
   }
public void printSignInForm(PrintWriter out, String action,String sess_id)
     {
       out.println("<Form method='post' action="+action+">");
       out.println("Session Id:" + sess_id);
                out.println("Please Sign in :");
       out.println("<input type=text name=signIn>");
       out.println("<br><input type=submit name=signin value=SignIn>");
     }
     private void welcomeBackPage(PrintWriter out,String signIn,String sess_id)
     {
```

```
                out.println("Session Id:" + sess_id);
        out.println("Hey you're <h3>" +signIn+ " </h3>Welcome back!!!");
      } }
Greeting.java
public class Greeting extends HttpServlet {
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      try {
            String user = request.getParameter("signIn");
       HttpSession session = request.getSession();
                String id=session.getId();
       if(user != null)
       {
                printThanks(out,user,id);
                session.setAttribute("ses_attr",user);
       }        }
finally
{
         out.close();
        }   }
private void printThanks(PrintWriter out,String signIn,String sess_id)   {
                 out.println("Session Id:" + sess_id);
        out.println("Thanks for signing in, <h3> "+signIn+ " </h3>");
         out.println("<br> Please <a href=FirstSession>visit again !</a>");
/*For URL Rewriting(Cookies Turned Off)
out.println("<br> Please <a href=" + response.encodeURL("FirstSession") +">visit again !</a>");        */
     }}
```

10. Write a servlet to illustrate the principles of cookies and Explain.

COOKIES

- A cookie is a name/value pair in the Set-Cookie header field of an HTTP response
- The main difference between cookies and sessions is that cookies are stored in the user's browser, and sessions are not.
- A cookie can keep information in the user's browser until deleted.
- If a person has a login and password, this can be set as a cookie in their browser so they do not have to re-login to your website every time they visit.

TABLE 6.3: Key `Cookie` class methods.

Method	Purpose
`Cookie(String name, String value)`	Constructor to create a cookie with given name and value
`String getName()`	Return name of this cookie
`String getValue()`	Return value of this cookie
`void setMaxAge(int seconds)`	Set delay until cookie expires. Positive value is delay in seconds, negative value means that the cookie expires when the browser closes, and 0 means delete the cookie.

Servlets can set cookies explicitly

– Cookie class used to represent cookies

– **request.getCookies()** returns an array of Cookie instances representing cookie data in HTTP request

– **response.addCookie(Cookie)** adds a cookie to the HTTP response

Cookies.java:

```
public class Cookies extends HttpServlet
{
   private static final int oneYear=60*60*24*365;
   protected void processRequest(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      try {
                                int count = 0;
     Cookie[] cookies = request.getCookies();
                                if(cookies != null)
                                {
                                  for(int i=0; (i<cookies.length) && (count==0); i++)
                                  {
                                          if(cookies[i].getName().equals("COUNT"))
                                          {
                                          count = Integer.parseInt(cookies[i].getValue());
                                          }
                                  }
                                }
count++;
Cookie cookie = new Cookie("COUNT", new Integer(count).toString());
cookie.setMaxAge(oneYear);
response.addCookie(cookie);
out.println("<p>You have visited content type page " +count+" time(s) \n in the past year , or since clearing
     your cookies</p>");
out.close();
}
finally
{
```

```
out.close();
}
}
```

UNIT V
XML AND WEB SERVICES
PART- A (2 Marks)

1. What is meant by a XML namespace?

XML Namespaces provide a method to avoid element name conflicts. When using prefixes in XML, a so-called **namespace** for the prefix must be defined. The namespace is defined by the **xmlns attribute** in the start tag of an element. The namespace declaration has the following syntax. **xmlns:*prefix*="*URI*".**

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

2. What is the purpose of XSLT?

The XSLT stands for XSL Transformations and XSL stands for extensible Style sheet Language. The XSLT is used for defining the XML document transformation and presentations.

3. What does XSLT mean?

XSLT stands for XSL Transformations and is a language used to transform XML documents into XHTML documents or to other XML documents

4. What is XQuery?

XQuery is a query and functional programming language that is designed to query and transform collections of structured and unstructured data, usually in the form of XML, text and with vendor-specific extensions for other data formats (JSON, binary, etc.).

5. What are complex types?

complex types are an important aspects of xml schema that allow application developers to define application-specific data types that can be checked by programs that check XML document for validity. XML schema divides complex types into two categories: those with *simple content* & those with *complex content.*

6. What are all the Transformation techniques?

- XSLT - it is an XML- based languages used to transform XML documents into others format such as HTML for web display.
- XLINK - highlighting that element or taking the user directly to that point in the document.
- XPATH - xpath gets its name from its use of a payh notation to navigate through the hierarchical tree structure of an XML document

XQUERY - it is W3C initiative to define a standard set of constructs for querying & searching XML document

7. What is meant by WSDL?

- WSDL stands for Web Services Description Language
- WSDL is based on XML

- WSDL is used to describe Web services
- WSDL is used to locate Web services
- WSDL is an XML-based language for locating and describing Web services

8. Define Serialization.

Object Serialization supports the encoding of objects, and the objects reachable from them, into a stream of bytes; and it supports the complementary reconstruction of the object graph from the stream. Serialization is used for lightweight persistence and for communication via sockets or Remote Method Invocation (RMI).

9. List the basic concepts behind JAX-RPC technology.

Java API for XML-based RPC (JAX-RPC) allows a java application to invoke a Java-based Web Service with a known description while still being consistent with its WSDL description. It can be seen as Java RMIs over Web services.

10. What is UDDI?

UDDI means Universal Description, Discovery and Integration. UDDI is a protocol for communicating with registries. The core of UDDI is the UDDI Business Registry, a global, pubic, online directory.

11. List some examples of web services.

- Geo IP: http://www.webservicex.net/geoipservice.asmx?op=GetGeoIP
- Whois: http://www.webservicex.net/whois.asmx?op=GetWhoIS
- SMS: http://www.webservicex.net/sendsmsworld.asmx

12. State the uses of WSDL.

- WSDL stands for Web Services Description Language.
- WSDL is a document written in XML.
- WSDL is an XML-based language for locating and describing Web services.

13. State the significance of a WSDL document.

The WSDL is a Web Service Descriptor Language which is based on XML.

ELEMENT	DESCRIPTION
Types	It Specifies the data types of the symbols used by the web services.
Messages	It specifies the messages used by the web services.
Porttype	It specifies the name of the operations
Binding	It specifies the name of the protocol of the web services, typically it is SOAP.

14. Give an example of a web services registry and its function.

It refers to a place in which service providers can impart information about their offered services and potential clients can search for services

Example: IBM - WebSphere Service Registry, Oracle Service Registry etc.,

15. What is the purpose of XML schema?

- The schemas are more specific and provide the support for data types.
- The schema is aware of namespace
- The XML Schema is written in XML itself and has a large number of built-in and derived types.
- The xml schema is the W3C recommendation. Hence it is supported by various XML validator and XML Processors.

16. Define the need for SOAP.

Simple Object Access Protocol (SOAP) is a protocol based on XML. It is used by the web services for exchange of information. The Client- Server communication is based on RPC. The HTTP does not design to handle the distributed objects that are required by the RPC. Hence another application protocol is build over HTTP which popularly known as SOAP. SOAP allows talking different applications that are running in two different operating systems.

17. What is the use of web services?

- Web services encompass a set of related standards that can enable two computers
- The data is passed back and forth using standard protocols such as HTTP, the same protocol used to transfer ordinary web pages.
- Web services operate using open, text-based standards that enable components written in different languages and on different platforms to communicate.
- They are ready to use pieces of software on the Internet. XML, SOAP, Web Services Description Language (WSDL) and Universal Description, Discovery and Integration (UDDI) are the standards on which web services rely.
- UDDI is another XML based format that enables developers and business to publish and locate Web services on a network.

18. List out some web service technologies?

- XML
- SOAP
- WSDL

19. What is SOAP?

Service Oriented Architecture Protocol.It provides a standard, extensible framework for packaging and exchanging XML messages. In the context of this architecture, SOAP also provides a convenient mechanism for referencing capabilities (typically by use of headers).

20. Define SOAP structure.

The SOAP envelope

<Envelope> is the root element in every SOAP message, and contains two child elements, an optional <Header> element, and a mandatory <Body> element.

The SOAP header

<Header> is an optional sub element of the SOAP envelope, and is used to pass application-related information that is to be processed by SOAP nodes along the message path; see The SOAP header.

The SOAP body

<Body> is a mandatory sub element of the SOAP envelope, which contains information intended for the ultimate recipient of the message; see The SOAP body.

The SOAP fault

<Fault> is a sub element of the SOAP body, which is used for reporting errors; see The SOAP fault.

XML elements in <Header> and <Body> are defined by the applications that make use of them, although the SOAP specification imposes some constraints on their structure.

21. Explain DTD for XML Schemas.

- XML documents are processed by applications
- Applications have assumptions about XML documents
- DTDs allow to formalize some of these constraints
- Part of the constraint checking must still be programmed

22. What is JWSDP?

Java Web Service Developer Pack (JWSDP) is a tool. Using the JWSDP tool the simple implementation files written in java can be converted to Web Service.

23. What are the disadvantages of schema?

- The XML schema is complex to design and hard to learn
- The XML document cannot be if the corresponding schema file is absent.

- Maintaining the schema for large and complex operations sometimes slows down the processing of XML document

24. Mention some of the disadvantageous of web services

Web services standards features such as transactions are currently nonexistent or still in their infancy compared to more mature distributed computing open standards such as CORBA. Web services may suffer from poor performance compared to other distributed computing approaches such as RMI, CORBA, or DCOM.

25. Why do you want to describe a web service?

Web Services Description Language (WSDL) is a document written in XML. The document describes a Web service. It specifies the location of the service and the operations (or methods) the service exposes

PART- B (16 Marks)

1. **List and explain the XML syntax rules in detail. Explain how a XML document can be displayed on a browser.**

- An XML document is one that follows certain syntax rules (most of which we followed for XHTML)

- An XML document consists of
  - Markup
    - Tags, which begin with < and end with >
    - References, which begin with & and end with ;
      - Character, e.g.
      - Entity, e.g. <
  - Character data: everything not markup

Staff1.xml:

```
<?xml version="1.0"?>
<company>
        <staff id="1001">
                <firstname>yong</firstname>
                <lastname>mook kim</lastname>
                <nickname>mkyong</nickname>
                <salary>100000</salary>
        </staff>
        <staff id="2001">
                <firstname>low</firstname>
                <lastname>yin fong</lastname>
                <nickname>fong fong</nickname>
                <salary>200000</salary>
        </staff>
</company>
```

- Element tags and elements
  - Three types
    - Start, e.g. <message>
    - End, e.g. </message>
    - Empty element, e.g.
- XML Rules:
  - Start and end tags must properly nest
  - Corresponding pair of start and end element tags plus everything in between them defines an element
  - Character data may only appear within an element
  - Start and empty-element tags may contain attribute specifications separated by white space
    - Syntax: *name = quoted value*

Well-formed XML document:

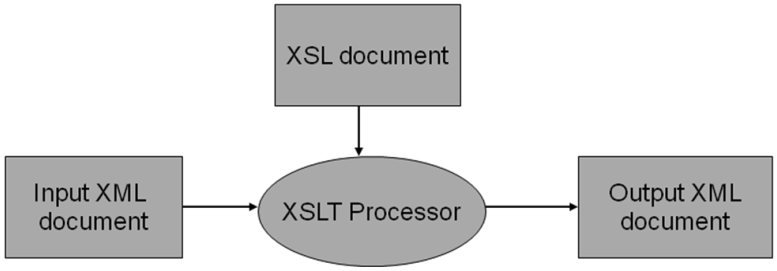
- A well-formed XML document
  - follows the XML syntax rules and
  - has a single root element
- Well-formed documents have a tree structure
- Many XML parsers (software for reading/writing XML documents) use tree representation internally

XML parsers

- Two types of XML parsers:
  - Validating
    - Requires document type declaration
    - Generates error if document does not
      - Conform with DTD and
      - Meet XML validity constraints
        - » Example: every attribute value of type ID must be unique within the document
  - Non-validating
    - Checks for well-formedness
    - Can ignore external DTD

2. **Given an XSLT document and a source XML document explain the XSLT transformation process that produces a single result XML document.**

- **The Extensible Stylesheet Language (XSL) is an XML vocabulary typically used to transform XML documents from one form to another form**



- JAXP allows a Java program to use the **Extensible Stylesheet Language (XSL)** to extract data from one XML document, process that data, and produce another XML document containing the processed data.

For example, XSL can be used to extract information from an XML document and embed it within an XHTML document so that the information can be viewed using a web browser.

TRANSFORMER:

Main.java:

```
import java.io.FileOutputStream;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;
public class Main
{
   static String xml="D://WebTech//trans.XML";
   static String xslt="D://WebTech//trans.XSL";
   static String output="D://WebTech//trans.HTML";
   public static void main(String[] args)
   {
      try
      {
      TransformerFactory tf = TransformerFactory.newInstance();
```

```
        Transformer tr = tf.newTransformer(new StreamSource(xslt));
        tr.transform(new StreamSource(xml),new StreamResult(new FileOutputStream(output)));
        System.out.println("Output to " + output);
        }
        catch(Exception e)
        {
         }
    }
}
```

trans.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>
            <h1>Indian Languages details</h1>
            <table border="1">
               <tr>
                  <th>Language</th>
                  <th>Family/Origin</th>
                  <th>No. of speakers</th>
                  <th>Region</th>
               </tr>
       <xsl:for-each select="language">
               <tr>
                  <td><xsl:value-of select="name"/></td>
                  <td><xsl:value-of select="family"/></td>
                  <td><xsl:value-of select="users"/></td>
                  <td><xsl:value-of select="region"/></td>
               </tr>
             </xsl:for-each>
            </table>
         </body>
      </html>
   </xsl:template>
</xsl:stylesheet>
```

trans.xml:

```
<?xml version="1.0"?>
<!--<?xml-stylesheet type="text/xsl" href="trans.xsl"?>-->
<language>
<name>Kannada</name>
<region>Karnataka</region>
<users>38M</users>
<family>Dravidian</family>
</language>
```

trans.html:

Indian Languages details

Language	Family/Origin	No. of speakers	Region
Kannada	Dravidian	38M	Karnataka

3. **Explain the role of XML namespaces with examples.**

- XML Namespace: Collection of element and attribute names associated with an XML vocabulary (such as XHTML)
- Namespace Name: Absolute URI that is the name of the namespace
  - Ex: http://www.w3.org/1999/xhtml is the namespace name of XHTML 1.0
- Default namespace for elements of a document is specified using a form of the xmlns attribute:

Another form of xmlns attribute known as a namespace declaration can be used to associate a namespace prefix with a namespace name:

xmlns:h="http://www.w3.org/TR/html4/"

- In **a namespace-aware** XML application, all element and attribute names are considered qualified names
  - A qualified name has an associated expanded name that consists of a namespace name and a local name
  - Ex: <table> is a qualified name with expanded name <null, table>
  - Ex: <h:table> is a qualified name with expanded name
    < http://www.w3.org/TR/html4, table>

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. This XML carries HTML table information:

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

This XML carries information about a table (a piece of furniture):

```
<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.An XML parser will not know how to handle these differences.

Solving the Name Conflict Using a Prefix

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

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

4. **Explain in detail the XML schema, built in and user defined data type in detail.**

XML Schema is an XML-based alternative to DTD.

An XML schema describes the structure of an XML document.

The XML Schema language is also referred to as XML Schema Definition (XSD).

XML Schema:

- XML Schema
  - Defines a number of simple data types, including
    - Range of allowed values
    - How values are represented as strings
  - Provides facilities for defining data structures in terms of simple types or other data structures
- Can also be used in place of XML DTD
- Built-in data types

```
<part name="latitude" type="xsd:decimal" />
```

  - Types corresponding to Java primitive types: boolean, byte,int, double, *etc*.
    - String representations much as Java
      - Exception: can use 0 for false, 1 for true
    - No char; use string instead
  - XML DTD types (ID, CDATA, *etc*.)
- Built-in data types
  - integer and decimal (arbitrary precision)
  - dates, times, and related subtypes
  - URLs
  - XML namespace qualified names
  - binary data
  - some restricted forms of the above, *e.g*., nonNegativeInteger
- XML Schema namespace defining built-in types is called the document namespace
  - Standard prefix for this namespace is xsd

TABLE 9.1: JAX-RPC mappings between supported Java classes and XML Schema built-in data types.

Java Class	XML Schema Type
String	string
java.math.BigDecimal	decimal
java.math.BigInteger	integer
java.util.Calendar	dateTime
java.util.Date	dateTime
java.xml.namespace.QName	QName
java.net.URI	anyURI

- **Plus Java primitive types (int, *etc*.)**
- Mapping from XML Schema data types to Java:
  - Primitives: one-for-one mapping
  - date, time, dateTime: map to Calendar
  - most others: map to String
- Elements in the document namespace can declare user-defined data types
- Two XML Schema data types:
  - Complex: requires markup to represent within an XML document
  - Simple: can be represented as character data
- **User-defined data types** are declared in the types element of a WSDL
  - Example: ExchangeValue
- In WSDL, user-defined types can be used
  - To define other data types within types element
  - To specify data types of parameters and return values in message elements

Restrictions for Datatypes

enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
pattern	Defines the exact sequence of characters that are acceptable
totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

Inclusive and Exclusive:

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxExclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Enumeration:

```
<xs:element name="car">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Pattern:

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-z]"/>
    </xs:restriction>
```

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

Minlength and Maxlength:

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="5"/>
      <xs:maxLength value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

5. **Describe the significance and working of WSDL with an example.**

Describing Web Services:WSDL.

Web Service ν Is a server application uses HTTP to accept and return SOAP documents, where content of the document is specified by a WSDL document that uses XML schema to define data types.

All the elements in the content of definitions optional ωMost WSDL document contain types element followed by atleast one of these elements:

- message
- portType
- binding
- Service
- Types element defines data types used by input and output parameter

•message is to define an input parameter list for an operation or to the value returned by the operation •One part element for each parameter

•This associates a name and a data type with each parameter

•portType element defines an abstract level the operation of a web service •The content of this element consist of an operation elements one for each operation •Names of these elements are not required to be unique •The content of an operation will be one input and one output this represents a form of an operation called request-response form used by web services.

Binding element specify a way in which the operations specified abstractly in portType can be accessed remotely by a client •The content of each operation element is a pair of input and output element with no attribute specification. •Contain a single soap:body element provides communication details .Whether or not the associated message should be encoded •What form of encoding should be used

Each operation element contains soap: operation element •Used for various purposes depending on WS implementation •An empty string specified as the value for this field it means it has not been used by the web service

Service element provides a overall name for the web service Contains one or more port elements each associates a binding with an internet address The address is specified by including soap: address element in the content of port

•Location attribute replaced with absolute URL by the server when the web services WSDL URL is browsed

types uses XML Schema to define data types ν message elements define parameter lists and return types using types and XML Schema

portType defines abstract API for operation‟s using message‟s

binding specifies how message‟s will be communicated and operation‟s called

service associates URL with binding

6. **Describe the major elements of SOAP.**

SOAP is a simple XML based protocol to let applications exchange information over HTTP.
Or more simply: SOAP is a protocol for accessing a Web Service.

SOAP Building Blocks

A SOAP message is an ordinary XML document containing the following elements:

- A required Envelope element that identifies the XML document as a SOAP message
- An optional Header element that contains header information
- A required Body element that contains call and response information
- An optional Fault element that provides information about errors that occurred while processing the message

All the elements above are declared in the default namespace for the SOAP envelope:
http://www.w3.org/2001/12/soap-envelope
and the default namespace for SOAP encoding and data types is:
http://www.w3.org/2001/12/soap-encoding

Syntax Rules

Here are some important syntax rules:

- A SOAP message MUST be encoded using XML
- A SOAP message MUST use the SOAP Envelope namespace
- A SOAP message MUST use the SOAP Encoding namespace
- A SOAP message must NOT contain a DTD reference
- A SOAP message must NOT contain XML Processing Instructions

Skeleton SOAP Message

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
...
...
</soap:Header>
<soap:Body>
...
...
<soap:Fault>
...
...
</soap:Fault>
</soap:Body>
</soap:Envelope>
```

SOAP ElementS:

SOAP Envelope Element

The mandatory SOAP Envelope element is the root element of a SOAP message.
The required SOAP Envelope element is the root element of a SOAP message. It defines the XML document as a SOAP message.
Note the use of the xmlns:soap namespace. It should always have the value of:
http://www.w3.org/2001/12/soap-envelope
and it defines the Envelope as a SOAP Envelope:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
...
```

Message information goes here
...
</soap:Envelope>

The xmlns:soap Namespace

A SOAP message must always have an Envelope element associated with the "http://www.w3.org/2001/12/soap-envelope" namespace.
If a different namespace is used, the application must generate an error and discard the message.

The encodingStyle Attribute

The SOAP encodingStyle attribute is used to define the data types used in the document. This attribute may appear on any SOAP element, and it will apply to that element's contents and all child elements. A SOAP message has no default encoding.

Syntax

soap:encodingStyle="URI"

SOAP Header Element

The optional SOAP Header element contains header information.

The optional SOAP Header element contains application specific information (like authentication, payment, etc) about the SOAP message. If the Header element is present, it must be the first child element of the Envelope element.
Note: All immediate child elements of the Header element must be namespace-qualified.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
<m:Trans
xmlns:m="http://www.w3schools.com/transaction/"
soap:mustUnderstand="1">234</m:Trans>
</soap:Header>
...
...
</soap:Envelope>
```

The example above contains a header with a "Trans" element, a "mustUnderstand" attribute value of "1", and a value of 234.
SOAP defines three attributes in the default namespace ("http://www.w3.org/2001/12/soap-envelope"). These attributes are: actor, mustUnderstand, and encodingStyle. The attributes defined in the SOAP Header defines how a recipient should process the SOAP message.

The actor Attribute

A SOAP message may travel from a sender to a receiver by passing different endpoints along the message path. Not all parts of the SOAP message may be intended for the ultimate endpoint of the SOAP message but, instead, may be intended for one or more of the endpoints on the message path.
The SOAP actor attribute may be used to address the Header element to a particular endpoint.

Syntax soap:actor="URI"

Example

<?xml version="1.0"?>

The mustUnderstand Attribute

The SOAP mustUnderstand attribute can be used to indicate whether a header entry is mandatory or optional for the recipient to process.

If you add "mustUnderstand="1" to a child element of the Header element it indicates that the receiver processing the Header must recognize the element. If the receiver does not recognize the element it must fail when processing the Header.

Syntax soap:mustUnderstand="0|1"

SOAP Body Element

The mandatory SOAP Body element contains the actual SOAP message.
The required SOAP Body element contains the actual SOAP message intended for the ultimate endpoint of the message.
Immediate child elements of the SOAP Body element may be namespace-qualified. SOAP defines one element inside the Body element in the default namespace ("http://www.w3.org/2001/12/soap-envelope"). This is the SOAP Fault element, which is used to indicate error messages.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body>
<m:GetPrice xmlns:m="http://www.w3schools.com/prices">
<m:Item>Apples</m:Item>
</m:GetPrice>
</soap:Body>
</soap:Envelope>
```

The example above requests the price of apples. Note that the m:GetPrice and the Item elements above are application-specific elements. They are not a part of the SOAP standard.

A SOAP response could look something like this:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body>
<m:GetPriceResponse xmlns:m="http://www.w3schools.com/prices">
<m:Price>1.90</m:Price>
</m:GetPriceResponse>
</soap:Body>
</soap:Envelope>
```

SOAP Fault Element

The optional SOAP Fault element is used to hold error and status information for a SOAP message.
An error message from a SOAP message is carried inside a Fault element.
If a Fault element is present, it must appear as a child element of the Body element. A Fault element can only appear once in a SOAP message.
The SOAP Fault element has the following sub elements:

Sub Element	**Description**
<faultcode>	A code for identifying the fault
<faultstring>	A human readable explanation of the fault
<faultactor>	Information about who caused the fault to happen
<detail>	Holds application specific error information related to the Body element

SOAP Fault Codes

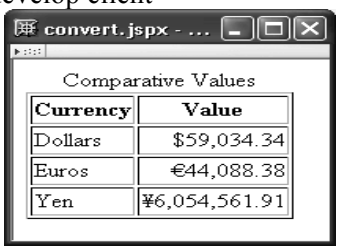
The faultcode values defined below must be used in the	**Description**

faultcode element when describing faults: **Error**	
VersionMismatch	Found an invalid namespace for the SOAP Envelope element
MustUnderstand	An immediate child element of the Header element, with the mustUnderstand attribute set to "1", was not understood
Client	The message was incorrectly formed or contained incorrect information
Server	There was a problem with the server so the message could not proceed

7. **Explain the creation of a java web service Client in detail with examples.**

Writing a Java Web service Client

- Goal: write a JSP-based client
  - Input: currency and value
  - Output: table of equivalent values
- Use wscompile tool to develop client



- Input xml document for wscompile tool
- Configuration file input to wscompile to create client
- Child element wsdl specifies the URL of a WSDL document

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
    xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <wsdl
    location=
    "http://localhost:8080/converter/currency?WSDL"
    packageName="myCurConClient" />
</configuration>
```

- Wscompile generate proxy object
- Proxy object is a java object called on by a client software in order to access the web service

JWSDP: Client

- **Directory structure (wscompile generates content of classes and src)**
- **Run wscompile**

```
webapps
  [[ other web application document base directories ]]
  ConverterClient
    WEB-INF
      classes
        myCurConClient
      src
        myCurConClient
```

- **Run wscompile**

Wscompile –gen –keep –d classes –s src config.xml

- Wscompile tool creates a class implementing the interface
- Interface is shared between webservice server and clients via the wsdl document.
- On server side the class implementing the interface is written
- On client side the interface is automatically generated by wscompile tool

Structs will be represented as JavaBeans classes, regardless of how they are defined on the server

```
public class ExchangeValues {
    protected double dollars;
    protected double euros;
    protected double yen;
    ...
    public double getDollars() {
        return dollars;
    }

    public void setDollars(double dollars) {
        this.dollars = dollars;
    }
    ...
```

- Bean obtaining and calling proxy object:
- JSP document convert.jspx calls on javaBeans class to perform currency conversion and displays result in HTML table
- Document is placed in ConverterClient directory

```
public ExchangeValues getExValues() {
    ExchangeValues ev = null;
    CurCon curCon =
        (new HistoricCurrencyConverter_Impl()).getCurConPort();
    try {
        if (currency.equals("euros")) {
            ev = curCon.fromEuros(value);
        }
        else if (currency.equals("yen")) {
            ev = curCon.fromYen(value);
        }
        else {
            ev = curCon.fromDollars(value);
        }
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    return ev;
}
```

- **JSP document using the bean:**

```
<c:set var="exvals" value="${client.exValues}" />
...
    <tr>
      <td>Euros</td>
      <td style="text-align:right">
        <fmt:formatNumber
          type="currency" currencySymbol="&#8364;">
          ${exvals.euros}
        </fmt:formatNumber>
      </td>
    </tr>
```

8. **Explain the basic concept of RPC**

Remote Procedure Call

JAX-RPC (Java API for XML-Based RPC) is an application program interface (API) in the Java Web Services Developer Pack (WSDP) that enables Java developers to include remote procedure calls (RPCs) with Web services or other Web-based applications. JAX-RPC is aimed at making it easier for applications or Web services to call other applications or Web services.

JAX-RPC provides a programming model for the development of SOAP (Simple Object Access Protocol)-based applications. The JAX-RPC programming model simplifies development by abstracting SOAP protocol-level runtime mechanisms and providing mapping services between Java and the Web Services Description Language (WSDL).

Sequence of events during an RPC

- The client calls the client stub. The call is a local procedure call, with parameters pushed on to the stack in the normal way.
- The client stub packs the parameters into a message and makes a system call to send the message. Packing the parameters is called marshalling.
- The client's local operating system sends the message from the client machine to the server machine.
- The local operating system on the server machine passes the incoming packets to the server stub.
- The server stub unpacks the parameters from the message. Unpacking the parameters is called unmarshalling.
- Finally, the server stub calls the server procedure. The reply traces the same steps in the reverse direction.

Example :

```
<?xml version="1.0"?>
<methodCall>
   <methodName>examples.getStateName</methodName>
   <params>
      <param><value><i4>41</i4></value></param>
   </params>
</methodCall>
```

9. **Illustrate the principles of WSDL, XML and SOAP and their interaction between them in web service applications**

XML:

An XML document consists of

a. Markup
   i. Tags, which begin with < and end with >
   ii. References, which begin with & and end with ;
      1. Character, e.g.
      2. Entity, e.g. <

b. Character data: everything not markup

SOAP:

Simple Object Access Protocol (SOAP) is a protocol based on XML. It is used by the web services for exchange of information. The Client- Server communication is based on RPC. The HTTP does not design to handle the distributed objects that are required by the RPC. Hence another application protocol is build over HTTP which popularly known as SOAP. SOAP allows talking different applications that are running in two different operating systems.

Describing Web Services:WSDL.

Web Service ν Is a server application uses HTTP to accept and return SOAP documents, where content of the document is specified by a WSDL document that uses XML schema to define data types.

All the elements in the content of definitions optional ωMost WSDL document contain types element followed by atleast one of these elements:

- message
- portType
- binding
- Service
- Types element defines data types used by input and output parameter

types uses XML Schema to define data types ν message elements define parameter lists and return types using types and XML Schema

portType defines abstract API for operation‟s using message‟s

binding specifies how message‟s will be communicated and operation‟s called

service associates URL with binding

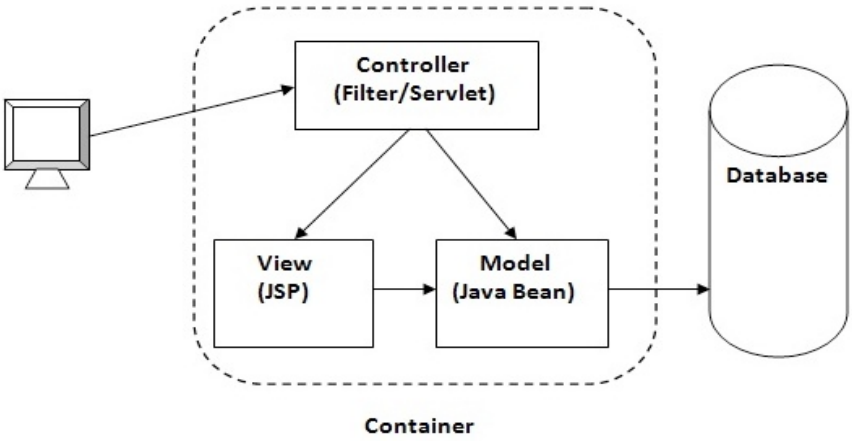
10. **Explain the model view controller architecture pattern in detail.**

Model 2 is based on the MVC (Model View Controller) design pattern. The MVC design pattern consists of three modules model, view and controller.

Model The model represents the state (data) and business logic of the application.

View The view module is responsible to display data i.e. it represents the presentation.

Controller The controller module acts as an interface between view and model. It intercepts all the requests i.e. receives input and commands to Model / View to change accordingly.



1. Define beans to represent the data
2. Use a servlet to handle requests
   – Servlet reads request parameters, checks for missing and malformed data, etc.
3. Populate the beans
   – The servlet invokes business logic (application-specific code) or data-access code to obtain the results. Results are placed in the beans that were defined in step 1.
4. Store the bean in the request, session, or servlet context
   – The servlet calls setAttribute on the request, session, or servlet context objects to store a reference to the beans that represent the results of the request.
5. Forward the request to a JSP page.
   – The servlet determines which JSP page is appropriate to the situation and uses the forward method of RequestDispatcher to transfer control to that page.
6. Extract the data from the beans.
   – The JSP page accesses beans with jsp:useBean and a scope matching the location of step 4. The page then uses jsp:getProperty to output the bean properties.

– The JSP page does not create or modify the bean; it merely extracts and displays data that the servlet created.

index.jsp:

```
<html>
   <head>
            <title>JSP Page</title>
   </head>
   <body>
     <form action="ControllerServlet" method="post">
       Name:<input type="text" name="name"><br>
       Password:<input type="password" name="password"><br>
       <input type="submit" value="login">
     </form>
   </body>
</html>
```

ControllerServlet.java:

```
public class ControllerServlet extends HttpServlet {
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      try {
      String name=request.getParameter("name");
      String password=request.getParameter("password");
      LoginBean bean=new LoginBean();
      bean.setName(name);
      bean.setPassword(password);
      HttpSession session=request.getSession();
      session.setAttribute("bean",bean);
      boolean status=bean.validate();

      if(status){
         RequestDispatcher rd=request.getRequestDispatcher("Success.jsp");
         rd.forward(request, response);
      }
      else{
         RequestDispatcher rd=request.getRequestDispatcher("Failure.jsp");
         rd.forward(request, response);
      }
      } finally {
         out.close();
      }
   }
```

LoginBean.java:

```
public class LoginBean
{
private String name,password;

public String getName()
{
return name;
}
```

```
public void setName(String name)
{
this.name = name;
}
public String getPassword()
{
return password;
}
public void setPassword(String password)
{
this.password = password;
}
public boolean validate()
{
if(password.equals("admin"))
{
return true;
}
else
{
return false;
}
}
}
```

Success.jsp:

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>JSP Page</title>
   </head>
   <body>
     <p>You are successfully logged in!</p>
      Welcome: <c:out value="${sessionScope.bean.name}"/>    </body>
</html>
```

Failure.jsp:

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>JSP Page</title>
   </head>
   <body>
          <p>Sorry! username or password error</p>
          <%@ include file="index.jsp" %>
   </body>
</html>
```