Biyani's Think Tank

Concept based notes

Advanced Internet Application

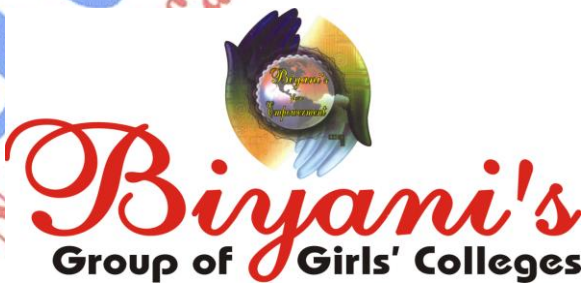
(MSC -IT (III Sem)

Peeyush Pareek

Lecturer

Deptt. of Information Technology

Biyani Girls College, Jaipur



Published by :

Think Tanks

Biyani Group of Colleges

Sector-3, Vidhyadhar Nagar,

Jaipur-302 023 (Rajasthan)

Ph : 0141-2338371, 2338591-95 • Fax : 0141-2338007

E-mail : acad@biyanicolleges.org

Website :www.gurukpo.com; www.biyanicolleges.org

First Edition : 2012

Leaser Type Setted by :

Biyani College Printing Department

Preface

I am glad to present this book, especially designed to serve the needs of the students.

The book has been written keeping in mind the general weakness in understanding the fundamental concepts of the topics. The book is self-explanatory and adopts the "Teach Yourself" style. It is based on question-answer pattern. The language of book is quite easy and understandable based on scientific approach.

Any further improvement in the contents of the book by making corrections, omission and inclusion is keen to be achieved based on suggestions from the readers for which the author shall be obliged.

I acknowledge special thanks to Mr. Rajeev Biyani, *Chairman* & Dr. Sanjay Biyani, *Director* (*Acad.*) Biyani Group of Colleges, who are the backbones and main concept provider and also have been constant source of motivation throughout this Endeavour. They played an active role in coordinating the various stages of this Endeavour and spearheaded the publishing work.

I look forward to receiving valuable suggestions from professors of various educational institutions, other faculty members and students for improvement of the quality of the book. The reader may feel free to send in their comments and suggestions to the under mentioned address.

Author

Chapter-1

Introduction to VB Script

What is VBScript?

- VBScript is a scripting language
- A scripting language is a lightweight programming language
- VBScript is a light version of Microsoft's programming language Visual Basic
- VBScript is only supported by Microsoft's browsers (Internet Explorer)

VBScript was created to allow web page developers the ability to create dynamic web pages for their viewers who used Internet Explorer. With HTML, not a lot can be done to make a web page interactive, but VBScript unlocked many tools like: the ability to print the current date and time, access to the web servers file system, and allow advanced web programmers to develop web applications.

How does VBScript Work?

When a VBScript is inserted into an HTML document, the Internet browser will read the HTML, and interpret the VBScript. The VBScript can be executed immediately, or at a later event.

JavaScript is language designed by Netscape Communications for programming Web pages. Thus VBScript and JavaScript are designed to do the same job. JavaScript has a much steeper learning curve than VBScript since JavaScript uses syntax similar to the cryptic C programming language.

Comparison chart of VBScript and JavaScript by syntax and operations:

Syntax

	VBScript	JavaScript
Arrays	Declaration (e.g., Dim,Static) Lbound, Ubound ReDim' Erase	new Arrays() new Object()
Assignment	=	=
Comments	REM Single Quote (')	// /* comment */
Control Flow	Do ….Loop For …Next, For Each …Next While … Wend	Do (statements) while (condition)

	If…Then…Else	for (expression; condition; [expression]) (
		statements
		)
		for (variable in object) (
		statements
		)
		while (condition) (
		statements1
		)
		e lse (
		statementgs2
		)
		switch (expression) (
		case label :
		statement;
		case label :
		statement;
		break;
		….
		default : statement;
		)

Error Trapping	On Error	window.onError=handler-fune window.onError(message,url,line)
Literals	Empty Nothing Null True, False User Defined Literals (e.g., 123,456,"test")	NaN Null True,false User Defined Literals(e.g.,123,456,"test")
Operators	Arithmetic+,-,*,/,\,^.Mod, Negation (-) Strings concatenation (&) Comparison=,<,>,<>,<=,>=, Is Logical Not,And,Or,Xor,Eqv,	Aritemetic +,++,-,--,*,/ % String + += Logical && !! Bitwise&^ ! ~,<< >> >>> Assignment =+= -= *=/=%=&=^=|=<<= >>= >>>= Comparison ==|=> >= < < = special ?:,delete new this typeof void
Options	Option Explicit	N/A
Declaring Procedures	Functions Sub	function function-name(arg1, arg2,…argN)
Calling Procedures	Call	function functionname(arg1,arg2,…argN) {statements;}
Exiting Procedures	Exit Function Exit Sub	
Parameters for	By Val, By Ref	

Procedures		
Procedures-level Variables	Dim Static	var variable-name
Module level Variables	Private Dim	var variable-name

When one uses Option Explicit it forces the user to declare all variables. If one tries to use a variable without declaring it an error is generated. If one doesn't use Option Explicit one is free to use variables without declaring them, however, this is bad programming practices as it is easy to misspell a variable and produce faulty code.

Operations

	VBScript	JavaScript
Divide & return an integer value	\	**N/A**
Exponentiation	**MOD**	**N/A**
Modulas	**MOD**	**%**
String Concetenation	**+ or &**	**+**
Left Shift	**N/A**	<<
Right Shift	**N/A**	>>
Bitwise AND	**AND**	**&**
Bitwise OR	**OR**	**|**
Bitwise XOR	**XOR**	^
Bitwise complement	**NOT**	**~**
Not equal to	<>	**!=**
Evaluation AND	**AND**	**&&**
Evaluation OR	**OR**	//
Evaluation XOR	**XOR**	**N/A**

The example below shows how to use VlBScript to write text on a web page:

Example

```
<html>

<body>

<script type=”text/vbscript”>

Document.write(“<h1>Hello World!</h1>”)

</script/>

</body>

</html>
```

Explanation

To insert a VBScript into an HTML page, we use the <script> tag. Inside the <script> tage we use the type attribute to define the scripting language.

So, the <script type=”text/vbscript”>

….

</script>

</body>

</html>

The document, write command is a standard VBscript command for writing output to a page.

By entering the documents, write command between the <script> and </script> tags, the browser will recognize it is a VBScript command and execute the code line. Ion this case the browser will write Hello World! To the page:

```
<html>
<body>
<script type=”text/vbscript”>
Document.write(“Hellow World”)
</script>
</body>
</html>
```

How to Handle Simple Browsers using VBScript?

Browsers that do not support scripting, will display VBScript as page content.

To prevent them from doing this, the HTML, comment tag should be used to “hide” the VBScript.

Just add an HTML comment tag <!—before the first VBScript statement, and a- -> (end of comment after the last VBScript statement, like this:

```
<html>
<body>
<script type=”text/vbscript”>
<!- -
Document.write (“Hello World!”)
- ->
</script>
</body>
</html>
```

Using an External VBScript

If you want to run the same VBScript on several pages, without having to write the same script on every page, you can write a VBScript in an external file.

Save the external VBScript file with a, vbs file extension.

Note : The external script cannot contain the <script> tag!

To use the external script, point to the, vbs file in the “Src” attribute of the <script> tag;

Example

```
<html>
<head>
<script type = “text/vbscript” src=”ex.vbs”></script>
```

<head>
<body>
</body>
</html>

Write Short note on VBScript variable:

VBScript Variables

As with algebra, VBScript variables are used to hold values or expressions.

A variable can have a short name, like x, or a more descriptive name, like carname.

Rules for VBScript variable names:

- Must begin with a letter
- Cannot contain a period(.)
- Cannot exceed 255 characters

In VBScript, all variables are of type *variant*, that can store different types of data.

Declaring/initializing VBScript Variables

Creating variables in VBScript is most often referred to as “declaring” variables

You can declare VBScript variables with the Dim, Public or the Private statement. Like this;

Dim X
Dim carname

Now you have created two variables. The name of the variables are “x” and “carname”.

You can also declare variables by using its name in a script. Like this;

Carname=”Volvo”

How you have also created a variable. The name of the variable is “carname”. However, this method is not a good practice, because you can misspell the variable name later in your script, and that can cause strange results when your script is running.

If you misspell for example the “carname” variable to “carnime”, the script will automatically create a new variable called “carnime”. To prevent your script from doing this, you can use the Option Explicit statement. This statement forces you to declare all your variables with the dim, public or private statement.

Put the option Explicit statement on the top of your script. Like this:

```
Option Explicit
Dim carname
Carname=some value
```

Assigning Values to Variables

You assign a value to a variable like this:

```
Carname=”Volvo”
X=10
```

The variable name is on the left side of the expression and the value you want to assign to the variable is on the right. Now the variable “carname” has the value of “Volvo”, and the variable “x” has the value of “10”

Lifetime of Variables

How long a variable exists is its lifetime.

When you declare a variable within procedure, the variable can only be accessed within that procedure. When the procedure exits, the variable is destroyed. These variables are called local variables. You can have local variables with the same name in different procedures, because each is recognized only by the procedure in which it is declared.

If you have declare a variable outside a procedure, all the procedures on your page can access it. The lifetime of these variables starts when they are declared, and ends when the page is closed.

VBScript Array Variables

An array variable is used to store multiple values in a single variable.

In the following example, an array containing 3 elements is declared:

Dim names(2)

The number shown in the parentheses is 2. We start at zero so this array contains 3 elements. This is a fixed-size array. You assign data to each of the elements of the array like this:

names(0)=”Tove”
names(1)=”Jani”
names(2)=”Stale”

Similarly, the data can be retrieved from any element using the index of the particular array element you want. Like this:

mother=name(0)

You can have up to 60 dimensions in an array. Multiple dimensions are declared by separating the numbers in the parentheses with commas. Here we have a two-dimensional array consisting of 5 rows and 7 columns:

Dim table (4,6)

Assign data to a two-dimensional array:

Example

```
<html>
<body>

<script type=”text/vbscript”>
Dim x (2,2)
x(0,0)=”Volvo”
x(0,1)=”BMW”
x(0,2)=”Ford”
x(1,0)=”Apple”
x(1,1)=”Orange”
x(1,2)=”Banana”
x(2,0)=”Coke”
x(2,1)=”Pepsi”
x(2,2)=”Sprite”
for i=0 to 2
    document.write (“<p>”)
    for j=0 to 2
       document.write(x(I,j) & “<br/>”)
    next
    document.wite(“</p>”)
next
</script>
```

</body>
</html>

What are :

- **VBScript Procedures**
- **Conditional Statements**
- **Looping *Statements***

VBScript has two kinds procedures:

- Sub procedure
- Function procedure

VBScript Sub Procedures

A Sub procedure:

- Is a series of statements, enclosed by the Sub and End Sub statements
- Can perform actions, but **does not return** a value
- Can take arguments
- Without arguments, it must include an empty set of parentheses ()

Sub mysub ()
some statements
End Sub

Or

Sub mysub (argument1,argument2)
some statements
End Sub

Example

Sub mysub()
Alert("Hello World")
End Sub

VBScript Function Procedures

A Function procedure:

- Is series of statements, enclosed by the Function and End Function statements
- Can perform actions and can return a value
- Can take arguments that are passed to it by a calling procedure
- Without arguments, must include an empty set of parentheses ()
- Returns a value by assigning as value to its name

```
Function my function()
  some statements
  myfuntion=some value
End Function
```

Or

```
Function myfunction(argument1,argument2)
  some statements
  myfunction=some value
End Function
```

Example

```
function myfunction()
  myfunction=Date()
end function
```

How to Call a Procedure

There are different ways to call a procedure. You can call it from within another procedure, on an event, or call it within a script.

Example

Call a procedure when the user clicks on a button:

```
<body>
<button onclick=”myfunction()”>Click me</button>
</body>
```

Procedures can be used to get a variable value;

Carname=findname()

Here you call a Function called “findname”, the Function returns a value that will be strored in the variable “carname”

Function procedures can calculate the sum of two arguments:

Example

```
Function myfunction(a,b)
myfunction =a+b
End Function

documents.write (myfujntion(5,9))
```

The function “myfunction” will return the sum of argument “a” and argument “b”. In this case 14.

Why you call a procedure you can use the Call statement, like this:

Call MyProc(argument)

Or, you can omit the Call statement, like this:

MyProc argument

Conditional Statements

Conditional statements are used to perform different actions for different decisions.

In VBScript we have four conditional statements:

- **If statement** – executes a set of code when a condition is true
- **If….Then….Else statement** – select one of two sets of lines to execute
- **If….Then….EsleIf statement**- select one of many sets of lines to execute
- **Select Case statement** – select one of many sets of lines to execute

---

If…Then….Else

Use the If…. Then….Else statement if you want to

- execute some code if a condition is true
- select one of two blocks of code to execute

If you want to execute to execute only one statement when a condition is true, you can write the code on one line:

If i=10 Then alert (“Hello”)

There is no…Else…in this syntax. You just tell the code to perform **one action** if a condition if a condition is true (in this case If i=10).

If you want to execute **more than one** statement when a condition is true, you must put each statement on separate lines, and end the statement with the keyword “End If”:

```
If i=10 Then
Alert(“Hello”)
i = i +1
End If
```

There is no…Else…in the example above either. You just tell the code to **perform multiple actions** if the condition is true.

If you want to execute a statement if a condition is true and execute another statement if the condition is not true, you must add the “Else” keyword:

Example

```
<html>
<body>
</head>
<script type=”text/vbscript>
Function greeting()
i = hour(time)
If  I < 10 Then
  Document.write(“Good morning!”)
Else
  document.write (“Have a nice day!”)
End If
End Function
</script>
</body>

</html>
```

In the example above, the first block of code will be executed if the condition is true, and the other block will be executed otherwise (if i is greater than 10)

If….Then….ElseIf

You can use the If…Then…ElseIf statement if you want to select one of many blocks of code to execute:

Example

```
<html>
<body>
</head>
<script type=”text/vbscript”>
Function greeting()
i = hour(time)
If  i = 10 Then
  document.write(“Just started…!”)
ElseIf  i = 11 then
  document.write(“Hungry!”)
ElseIf  i = 12 then
  document.write(“Ah, lunch-time!”)
DlseIf  i= 16 then
  document.write(“Time to go home!”)
Else
  document.write(“Unknown”)
End If
End Function
</script>
</head>

<body onload=”greeting()”>
</body>

</html>
```

Seclect Case

You can also use the “Select Case” statement if you want to select one of many blocks of code to execute:

Example

```
<html>
<body>
<script type=”text/vbscript”>
d = weekday(date)
Select Case  d
  Case 1
    document.write(“Sleepy Sunday)
  Case 2
    document.write(“Monday again!”)
  Case 3
    documents.write(“Just Tuesday!”)
  Case 4
    document.write (“Wednesday!”)
  Case 5
    document.write(“Finally Friday!”)
  Case 6
    document.write(“Super Saturday!!!!”)
End Select
</script>

<body>
</html>
```

This is how it works: First we have a single expression (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each Case in the structure. If there is a match. The block of code associated with that Case is executed.

Looping *Statements*

Looping statements are used to run the same block of code a specified number of times.

In VBScript we have four looping statements:

- **For…Next statement** – runs code a specified number of items
- **For Each…Next statement** – runs code for each item in a collection or each element of any array
- **Do…Loop statement** – loops while or until a condition is true
- **While…Wend statement** – Do not use it - use the Do…Loop statement instead

For …*Next* Loop

Use the **For…Next** statements to run a block of code a specified number of times.

The **For** statement specifies the counter variable (**i**), and its start and end values. The **Next** statement increases the counter variable (**i**) by one.

Example

```
<html>
<body>

<script type=”text/vbscript”>
For  i = 0 To 5
   document. Write (“The number is “& i &” <br/>”)
Next
</script>

<body>
</html>
```

The Step Keyword

With the **Step** keyword, you can increase or decrease the counter variable by the value you specify.

In the example below, the counter variable (**i**) is INCREASED by two, each time the loop repeats.

```
For  i = 10 to 2 Step – 2
  some  code
Next
```

Exit a For… Next

You can exit a For…Next statement with the Exit For keyword.

```
For  i = 1 to 10
```

```
  If  i = 5 Then Exit For
  some code
Next
```

For Each…Next Loop

A **For Each….Next** loop repeats a block of code for each item in a collection, or for each elements of an array.

Example

```
<html>
<body>

<script type=”text/vbscript”>
Dim cars(2)
cars (0)=”Volvo”
cars (1)=”Saab”
cars (2)=”BMW”

For Each x In cars
  document.write(x&”<br/>”)
Next
</script>

</body>
</html>
```

Do…Loop

If you don’t know how many repetitions you want, use a Do….Loop statement.

The Do….Loop statement repeats a block of code while a condition is true, or until a condition becomes true.

Repeat Code While a Condition is True

You use the While keyword to check a condition in a Do…Loop statement.

```
Do While i>10
  some code
Loop
```

If **i** equals 9, the code inside the loop above will never be executed.

```
Do
  Some code
Loop While  i>10
```

The code inside this loop will be executed at least one time, even if **i** is less than 10.

Repeat Code Until a Condition Becomes True

You use the Until keyword to check a condition in a Do…loop statement.

```
Do Until  i=10
  Some code
Loop
```

If **i** equals 10, the code inside the loop will never be executed.

```
Do
  Some code
Loop Until i=10
```

The code inside this loop will be executed at least one time, even if **i** is equal to 10.

Exit a Do…Loop

You can exit a Do…Loop statement with the Exit Do keyword.

```
Do Until  i=10
i=i-1
  If i<10 Then Exit Do
Loop
```

The code inside this loop will be executed as long as **i** is different from 10, and as long as **i** is greater than

JSP

Overview of JSP

JavaServer Pages (JSPs) are similar to HTML files, but provide the ability to display dynamic content within Web pages. JSP technology was developed by Sun Microsystems to separate the development of dynamic Web page content from static HTML page design. The result of this separation means that the page design can change without the need to alter the underlying dynamic content of the page. This is useful in the development life-cycle because the Web page designers do not have to know how to create the dynamic content, but simply have to know where to place the dynamic content within the page.

To facilitate embedding of dynamic content, JSPs use a number of *tags* that enable the page designer to insert the properties of a JavaBean object and script elements into a JSP file.

Here are some of the advantages of using JSP technology over other methods of dynamic content creation:

- **Separation of dynamic and static content**
  This allows for the separation of application logic and Web page design, reducing the complexity of Web site development and making the site easier to maintain.
- **Platform independence**
  Because JSP technology is Java-based, it is platform independent. JSPs can run on any nearly any Web application server. JSPs can be developed on any platform and viewed by any browser because the output of a compiled JSP page is HTML.
- **Component reuse**
  Using JavaBeans and Enterprise JavaBeans, JSPs leverage the inherent reusability offered by these technologies. This enables developers to share components with other developers or their client community, which can speed up Web site development.
- **Scripting and tags**
  JSPs support both embedded JavaScript and tags. JavaScript is typically used to add page-level functionality to the JSP. Tags provide an easy way to embed and modify JavaBean properties and to specify other directives and actions.

How JavaServer Pages work

JavaServer Pages are made operable by having their contents (HTML tags, JSP tags and scripts) translated into a servlet by the application server. This process is responsible for translating both the dynamic and static elements declared within the JSP file into Java servlet code that delivers the translated contents through the Web server output stream to the browser.

Because JSPs are server-side technology, the processing of both the static and dynamic elements of the page occurs in the server. The architecture of a JSP/servlet-enabled Web site is often referred to as *thin-client* because most of the business logic is executed on the server.

The following process outlines the tasks performed on a JSP file on the *first invocation* of the file or when the underlying JSP file is changed by the developer :

- The Web browser makes a request to the JSP page.
- The JSP engine parses the contents of the JSP file.
- The JSP engine creates temporary servlet source code based on the contents of the JSP. The generated servlet is responsible for rendering the static elements of the JSP specified at design time in addition to creating the dynamic elements of the page.
- The servlet source code is compiled by the Java compiler into a servlet class file.
- The servlet is instantiated. The *init* and *service* methods of the servlet are called, and the servlet logic is executed.
- The combination of static HTML and graphics combined with the dynamic elements specified in the original JSP page definition are sent to the Web browser through the output stream of the servlet's response object.

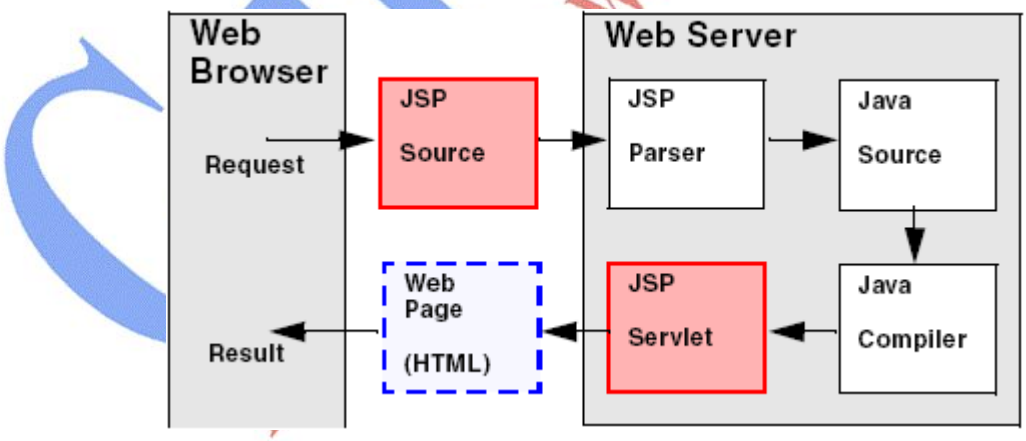


Figure: The JSP processing life-cycle on first-time invocation

Subsequent invocations of the JSP file will simply invoke the *service* method of the servlet created by the above process to serve the content to the Web browser. The servlet produced as a result of the above process remains in service until the application server is stopped, the servlet is manually unloaded, or a change is made to the underlying file, causing recompilation.

Components of JavaServer Pages

JavaServer Pages are composed of standard HTML tags and JSP tags. The available JSP tags defined in the JSP 1.0 specification are categorized as follows:

- Directives
- Declarations
- Scriptlets
- Comments
- Expressions

JSP directives

A JSP directive is a global definition sent to the JSP engine that remains valid regardless of any specific requests made to the JSP page. A directive always appears at the top of the JSP file, before any other JSP tags. This is due to the way the JSP parsing engine produces servlet code from the JSP file.

The syntax of a directive is:

<%@ directive directive_attr_name = value %>

Directives are grouped as follows:

page

The *page* directive defines page dependent attributes to the JSP engine.

<%@ page language="java" buffer="none" isThreadSafe="yes"

errorPage="/error.jsp" %>

Commonly used attributes of the page directive are listed in the following Table.

Attribute Name	Description
Language	Identifies the scripting language used in scriptlets in the JSP file or any of its

	included files. JSP supports only the value of “java”. <%@ page language = "java" %>
Extends	The fully-qualified name of the superclass for which this JSP page will be derived. Using this attribute can effect the JSP engine’s ability to select specialized superclasses based on the JSP file content, and should be used with care.
Import	When the language attribute of "java" is defined, the import attribute specifies the additional files containing the types used within the scripting environment. <%@ page import = "java.util.*" %>
Session "true" | "false"	If *true*, specifies that the page will participate in an HTTP session and enables the JSP file access to the implicit session object. The default value is *true*.

include

The *include* directive allows substitution of text or code to occur at translation time. You can use the include directive to provide a standard header on each JSP page, for example:

```
<%@ include file="copyright.html" %>
```

include directive has single attribute named file. It directs the JSP engine to substitute the text or code specified by file or URL reference. The URL reference can be another JSP file.

Declarations

A declaration block contains Java variables and methods that are called from an *expression* block within the JSP file. Code within a declaration block is usually written in Java. Code within a declaration block is often used to perform additional processing on the dynamic data generated by a JavaBean property.

The syntax of a declaration is:

```
<%! declaration(s) %>
```

For example:

```
<%!
private int getDateCount = 0;
private String getDate(GregorianCalendar gc1)
{ ...method body here...}
%>
```

Scriptlets

JSP supports embedding of Java code fragments within a JSP by using a *scriptlet* block. Scriptlets are used to embed small code blocks within the JSP page, rather than to declare entire methods as performed in a declarations block. The syntax for a scriptlet is:

<% scriptlet %>

The following example uses a scriptlet to output an HTML message based on

the time of day. You can see that the HTML elements appear *outside* the

script declarations.

<% if (Calendar.getInstance().get(Calendar.AM_PM) == Calendar.AM)

{%>

How are you this morning ?

<% } else

{ %>

How are you this afternoon ?

<% } %>

Comments

You can use two types of comments within a JSP. The first comment style, known as an *output comment,* enables the comment to appear in the output stream on the browser. This comment is an HTML formatted comment whose syntax is:

<!-- comments ... -->

The second comment style is used to fully exclude the commented block from the output and is commonly used when uncommenting a block of code that so that the commented block is never delivered to the browser. The syntax is:

<%-- comment text --%>

You can also create comments containing dynamic content by embedding a scriptlet tag inside a comment tag. For example:

<!-- comment text <%= expression %> more comment text ->

Expressions

Expressions are scriptlet fragments whose results can be converted to String objects and subsequently fed to the output stream for display in a browser. The syntax for an expression is:

<%= expression %>

The following example calls the *incrementCounter* method declared in the declarations block and prints the result.

<%= incrementCounter() %>

Accessing implicit objects

When you are writing scriptlets or expressions, there are a number of objects that you have automatic access to as part of the JSP standard without having to fully declare them or import them. The following table summarizes these implicit objects available in JSP 1.0.

You can use these implicit objects directly in your code. The following code snippet is an example of accessing the out implicit object to display a line of text in the browser:

out.println("Here is the Date Display JSP");

Object name	**Type**	**Description**
Request	javax.servlet.HttpServletRequest	The request triggering the service invocation
Response	javax.servlet.HttpServletResponse	The response to the request
pageContext	javax.servlet.jsp.PageContext	Page context of this JSP. By accessing this object, you have access to a number of convenience objects and methods such as getException, getPage, and getSession providing an explicit method of accessing JSP implementation-specific objects.

Session	javax.servlet.http.HttpSession	Session object created for the requesting client
application	javax.servlet.ServletContext	The servlet context as obtained from the servlet configuration object
Out	javax.servlet.jsp.JspWriter	Output stream writer
Page	java.lang.Object	Instance of this page's implementation class processing the current request.

Putting it all together

Following example combines many of the JSP components previously discussed in this chapter. In the example, the current date is displayed together with a count of the number of times the *getDate* function has been called.

Note that the counter continues to increment until the servlet is manually stopped, the Application Server is restarted, or the JSP page is modified, forcing a page compilation to occur.

A simple html page that accepts name and password of the user and calls a jsp name "first.jsp" which reads the values of name and password from the request and display them with some message.

first.html

```
<html>
<body>
<h1>Now Access a simple Jsp Page</h1>
<form  action="first.jsp">
Enter the the username & password<br>
<input type="text"  name="txtname"><br>
<input type="password"  name="txtpass"><br>
<input type="submit" value="submit">
</form>
```

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

first.jsp

```
<html>
<body>
<%! String s1,s2; %>
<h1>This is a smple Jsp Page</h1><br>
<b>The username & password of user is</b>
<p>

<%
 s1=request.getParameter("txtname");
 s2=request.getParameter("txtpass");
%>
The User Name is
<%
out.print(s1);
%>
<br>And Password is
<%
out.println(s2);
%>

</body>
```

```
</html>
```

Invoking a JSP by URL

A JSP can be invoked by URL, from within the <*FORM*> tag of a JSP or HTML page, or from another JSP.

To invoke a JSP by URL, use the syntax:

http://servername/path/filename.jsp

For example, to invoke the DateDisplay.jsp, use this URL:

http://localhost:8080/JspApp/DateDisplay.jsp

Deploying and testing a JSP application on Java application Server:

- Start application server and deploy tool.
- Create a new web component.
- Add the JSP file and other resource making up the application.
- Select JSP in the next step of the new application wizard.
- Specify the aliases used to access the JSP.
- Invoke the JSP using the above syntax.

Calling a servlet from a JSP

You can invoke a servlet from a JSP either as an action on a form, or directly through the jsp:include or jsp:forward tags.

Form action

Typically, you want to call a servlet as a result of an action performed on a JavaServer Page. For example, you may want to process some data entered by the user in an HTML form when they click on the *Submit* button.

To invoke a servlet within the HTML <FORM> tag, the syntax is:

<FORM METHOD="POST|GET" ACTION="application_URI / JSP_URL">

<!-- Other tags such as text boxes and buttons go here -->

</FORM>

For example:

<form method="POST"

action="DateDisplayServlet">

</form>

JSP include tag

You can include the output of a servlet in a JSP using the jsp:include tag:

<jsp:include page="HTMLServlet" />

JSP forward tag

You can forward processing from a JSP to a servlet using the jsp:forward tag:

<jsp:forward page="HTMLServlet" />

Example of include and forward:

An html page whose contents would be included in the output of a jsp page.

c.htm

```
<html>
<head>
<title>
A simple web application</title>
</head>
<body>
<b><strong> output of this page would be included in anoter web page.</strong></b>
</b>
</form>
</body>
</html>
```

A jsp page that reads the value of a request parameter, saves it into session and forwards the request to another jsp named “two.jsp”.

one.jsp

```
<html>
<head>
<title> A simple jsp page</title></head>
<body>
```

```
<%
String name = request.getParameter("txtname");
session.setAttribute("name",name);
%>
<jsp:forward page="two.jsp" />
</body>
</html>
```

A jsp page that includes the contents of c.htm as well as uses session object to display the no. of times this page is visited by the user.

two.jsp

```
<html>
<head>
<title> A simple jsp page using session object</title></head>
<body>

<form action="two.jsp">
<B> To Know your count of visit, press submit.</b>
<br><input type="submit" value="submit">
</form>
<%
Integer a = (Integer)session.getAttribute("count");
String name=(String)session.getAttribute("name");
out.print("Hello "+name+"<br>");
session.setAttribute("name",name);
if (a==null)
```

```
{
out.print("this is your first visit");
a=new Integer(1);
session.setAttribute("count",a);
}
else
{
int c= a.intValue();
c++;
out.print("you have visited this page "+c+" times.");
a=new Integer(c);
session.setAttribute("count",a);
}

%>
<jsp:include page="c.htm" />

</body>
</html>
```


jsp:useBean

The *jsp:useBean* tag is used to declare a JavaBean object that you want to use within the JSP. Before you can use the *jsp:getProperty* and *jsp:setProperty* tags, you must have first declared your JavaBean using the jsp:useBean tag. When the jsp:useBean tag is processed, the application server performs a lookup of the specified given Java object using the values

specified in the *id* and *scope* attributes. If the object is not found, it will attempt to create it using the values specified in the *scope* and *class* attributes.

The syntax for inserting a JavaBean is:

<jsp:useBean id="beanInstanceName" scope="page|request|session|application" >

optional scriptlets and tags

</jsp:useBean>

You can also embed scriptlets and tags such as *jsp:getProperty* within the jsp:useBean declaration which will be executed upon creation of the bean. This is often used to modify properties of a bean immediately after it has been created.

An example of a simple form of bean instantiation is:

<jsp:useBean id ="DateDisplayBean"

class="DateDisplayBean"/>

This example tries to locate an instance of the *DateDisplayBean* class. If no instance exists, a new instance is created. The instance can then be accessed within the JSP using the specified *id* of DateDisplayBean.

jsp:useBean attributes :

Parameter Name	Identifies the object name within the name space of the specified scope. This name is used to reference the bean throughout the JSP file and is case sensitive.
Scope	Valid values are page, request, session, and application. If omitted, the value defaults to *page* scope. ***page***: Objects declared with page scope are only valid until the response is sent back from the server or until the request is forwarded elsewhere. References to objects in page scope are only valid within the page where the object is declared. Objects declared in page scope are stored in the pagecontext object. ***request***: Objects declared within request scope are valid for the duration of the request and are accessible if the request is forwarded to a resource

	in the same runtime. Objects referenced in request scope are stored in the request object. ***session***: Session-scope objects are available for the duration of the session provided that the page is made "session aware" using the page directive. ***application***: Application-scope objects are available from pages that are processing requests within the same Web application (as defined in the application server setup) and are valid until the ServletContext object is reclaimed by the application server. Objects with this scope are stored in the application object.
Class	The name of the object's implementation class, for example: DateDisplayBean. This value is case sensitive. Specify the class attribute if you want to instantiate the bean if it does not already exist within the specified scope.
beanName	Specifies the class name or serialized file (.ser) containing the bean which is used when first creating the bean.

jsp:getProperty

Once the bean has been declared with jsp:useBean, you can access its exposed properties through the *jsp:getProperty* tag, which inserts the String value of the primitive type or object into the output stream. For primitive types, the conversion to String is performed automatically. For object types, the *toString* method of the object is called.

The syntax for the jsp:getProperty tag is

<jsp:getProperty name="beanName" property="propertyName"/>

jsp:setProperty

The properties of beans can be set by using the *jsp:setProperty* tag. The syntax for this tag is:

<jsp:setProperty name="beanName" prop_expr/>

For example, to initialize the counter variable used in dateDisplay.jsp, you could use the code:

<jsp:setProperty name="DateDisplayBean" property="counter" value="0"/>

Attribute Name	Description
Name	The name (id) of the bean instance specified in the jsp:useBean tag.
Property	The name of the property to set. By setting this value to "*", you can automate the setting of properties, provided that form-element names match the property name. For example, if a bean has a property called *dateString*, and the JSP page contains a text box named *dateString*, then the dateString property of the bean will be looked up and set automatically. For this feature to work, your beans must conform to the JavaBeans API specification 1.0.
Param	The request parameter name to give to the Bean property. Request parameters usually refer to the names of HTML form elements, and are used to implicitly set the value of a particular bean property based on the value of the HTML form element. This attribute cannot be used with the value attribute.
Value	The new value for the property.

Example of use Bean:

Following is the source code of a java bean that would be used in a jsp file. Note that it is defined in a package named "mybean". To access a java class as a bean in a jsp page the class must be defined in a package.

```
package mybean;

public class DemoBean

{
```

```
private String name;


public String getName()

{


return name;

}

public void setName(String name)

{

this.name=name;

}

public String greet()

{

return "Hello "+name;

}

}
```


An html page that accepts user name as request parameter and calls a jsp page name “demo.jsp” .

bean.htm

```
<html>

<head><title> Beans example.</title></head>

<body>
```

```
<form action="demo.jsp">

Name<input type="text" name="txtname">

<br><input type="submit" value="submit">

</form>

</body>

</html>
```

A jsp that creates and instance of DemoBean class sets its name property using request parameter and displays its value using getparameter as well as calls greet method on its object.

demo.jsp

```
<html>

<head><title>A Bean Example.</title></head>

<jsp:useBean id="demo" class="mybean.DemoBean" />

<body>

calling beans method.<br>

<jsp:setProperty name="demo" property="name" param="txtname" />

<b> value of name property is <jsp:getProperty name="demo" property="name" />

<br>

<%= demo.greet() %>

</body>

</html>
```


1. what is thin- client application?

A. A browser that uses a plug-in to process user data.
B. A distributed application where the client program does not process datA. but instead passes data for processing to an enterprise bean running

on an application server.
C. An application that cannot be stopped by a firewall.
D. An application compiled with the -thin option of the javac command.

2. when do you use a session bean?

A. When you want to be certain your application data persists between program invocations, even in the event of a crash.
B. If you need to initiate a database transaction.
C. To process transient data that can be re-created in the event of a crash.
D. To keep people who do not know how to program from accidentally changing important logic code.

3.what is bean- managed persistence?

A. When you implement entity or session bean methods to use SQL commands you provide.
B. When the bean's container handles data storage and retrieval.
C. When the J2EE server is never shut down.
D. When changes to database data are lost during a crash.

4. How are life cycle methods called?

A. By thin clients.
B. By calls to the database.
C. By the J2EE server.
D. By the bean's container.

5. what exception thrown when servlet initialization fails?

A. IOException
B. ServletException
C. RemoteException

6. what is legal about jsp scriplets?

A. A loop can begin in one Scriptlet and end in another
B. Statements in Scriptlets should follow Java Syntax

C. Semicolon is needed at the end of each statement in a Scriptlet
D. All the above

7. How can a servlet call a jsp error page?

A. This capability is not supported.
B. When the servlet throws the exception, it will automatically be caught by the calling JSP page.
C. The servlet needs to forward the request to the specific error page URL. The exception is passed along as an attribute named "javax.servlet.jsp.jspException".
D. The servlet needs to redirect the response to the specific error page, saving the exception off in a cookie.

8. what advantage does an entity bean have over a session bean?

A. Entity bean data survives crashes.
B. An entity bean represents non-persistent data.
C. It creates its own JAR file during deployment.
D. You get authentication code without having to write any.

9. In a multithread application,which tier isthe browser in?

A. First tier.
B. Second tier.
C. Third tier.
D. Fourth tier.

10. What is the output of following piece of code ?

```
int x = 2;
 switch (x) {
 case 1:System.out.println(“1?);
 case 2:
 case 3:System.out.println(“3?);
 case 4:
 case 5:System.out.println(“5?);
 }
```

A. No output
B. 3 and 5

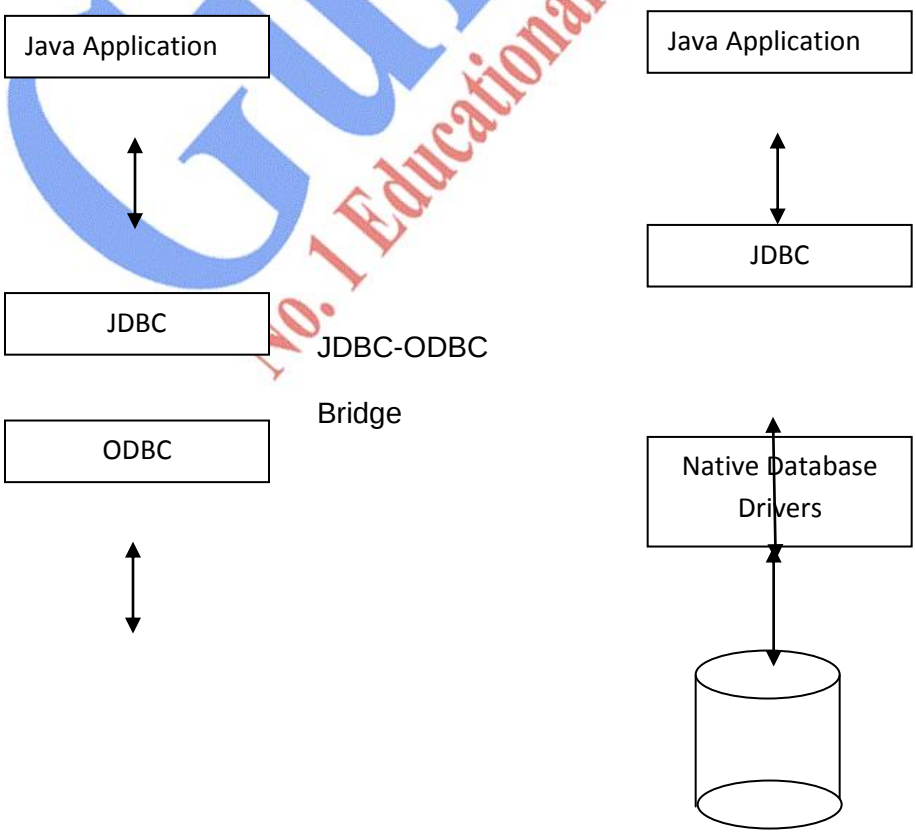
C. 1, 3 and 5
D. 3.

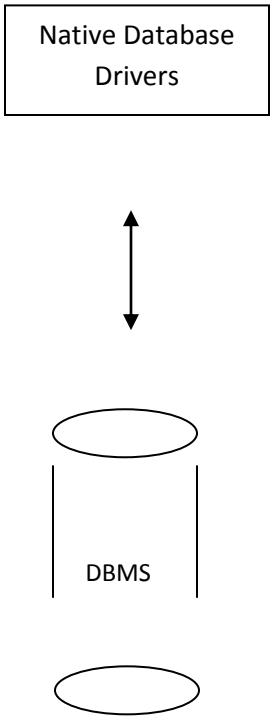
1. (b)	2. (c)	3. (a)	4. (d)	5. (b)
6. (d)	7. (c)	8. (a)	9. (a)	10. (b)

JDBC

java.sql package:

java.sql package contains classes and interfaces that form the JDBC API. Classes and interfaces in java.sql package provide a standardized and abstracted mechanism to connect a java application to a database. An application can connect to a database using various architectures, most commonly used among them are:





In the first architecture both application and database are independent of each other because of JDBC-ODBC bridge. Here independence means that application can be entirely changed without affecting the database or vice-versa.

Commonly used classes and interfaces of JDBC API:

DriverManager: This class is responsible for dynamically identifying, locating and loading database drivers as required by various database systems.

Connection: Provides the abstraction of a database connection. The main functionality provided by connection class is the facility to create Statements.

Statement: A statement object is responsible for the execution of select, insert, delete and update queries against a database connection.

ResultSet: A ResultSet object represents an application level cursor. It is used for storing the records returned by a select query and provides the functionality of extracting the values of individual fields of the current record as well as facility to set record pointer on a specific record.

PreparedStatement: A PreparedStatement is a statement that supports parameterized queries i.e. at the time of query creation placeholders are inserted in the query whose values are specified at the time of query execution.

Loading Database Drivers and Opening Connection:

Before opening the connection driver class is dynamically loaded using the forName() method.

forName(): Loads the class whose name is given as argument.

Syntax:

public static Class forName(String className) throws ClassNotFoundException;

It is defined java.lang.Class class.

e.g.: to load the JdbcOdbcDriver class following code is used.

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

getConnection(): establishes a database connection using the given connection string and returns its reference.

Syntax:

public static Connection getConnection(String connecitonString) throws SQLException;

or

public static Connection getConnection(String connectionString, String username, String password) throws SQLException;

These methods are defined in DriverManager class.

Connection string has the following format:

“mainprotocol:subprotocol:dsn”

Here dsn represents the data source name.

Steps to create a DSN:

Start → control panel → administrative tools → data sources (ODBC) → System DSN → add → select the database driver → give DSN name → ok.

e.g.: if the dsn name is ourdata and jdbc-odbc bridge is used then connection string would be

“jdbc:odbc:ourdata” and connection would be opened as

Connection c= DriverManager.getConnection(“jdbc:odbc:ourdata”);

or

Connction c = DriverManager getConnection(“jdbc:odbc:ourdata”,”scott”,”tiger”);

Creating statements and executing Queries:

createStatement(): creates a statement object.

Syntax:

public static Statement createStatement() throws SQLException;

or

public static Statement createStatement(int resultsetType, int resultsetConcurrency) throws SQLException;

these methods are defined in Connection class.

ResultSet can of the following three types:

Forward only : record pointer can be moved in the forward direction only.

Scrollabe and Insensitive: record pointer can be moved in both directions but the resultset remains immune to the changed made into the database by other concurrent users.

Srollable and Sensitive: record pointer can be moved in both direction and changes made into the database by other concurrent users are reflected in the resultset.

To specify the type of result set following static named constants are defined in ResultSet class:

ResultSet.TYPE_FORWARD_ONLY // it is the default type.

ResultSet.TYPE_SCROLL_INSENSITIVE

ResultSet.TYPE_SCROLL_SENSITIVE

By default resultsets are read-only and can't be changed. The ResultSet interface specifies two constants to indicate whether the resultset is read-only or updatable.

ResultSet.CONCUR_READ_ONLY // default concurrency type.

ResultSet.CONCUR_UPDATABLE

Methods of Statement class:

executeQuery(): executes a select query and returns its result as a ResultSet object.

Syntax:

public ResutlSet executeQuery(String query) throws SQLException;

e.g.:

to execute a select query to fetch ename, job and salary of employees from emp table following statements would be written:

Statement s = c.createStatement();

String q =" select enam, job, sal from emp";

ResultSet r = s.executeQuery(q);

executeUpdate(): executes an insert, delete, or update query and returns the no. of rows affected by the execution of the query.

Syntax:

public int executeUpdate(String query) throws SQLException;

Methods of ResultSet class:

next(): Advances the record pointer in the ResultSet by one record and returns true if the pointer is placed over a valid record after advancement otherwise returns false. The initial location of the record pointer is before the first record.

Syntax:

public boolean next();

getType(): It is not a method itself but a general signature of various methods provided by ResultSet that returns the value of the specified field of the current record after converting it into the specified java type. These methods are used for converting database types into java types.

The general signature of these methods are:

public Type getType(int fieldIndex);

following are the specific implementations the general signature:

public int getInt(int fieldIndex): returns the value of the given field after converting it into integer.

public String getString(int fieldIndex): returns the value of the given field as a String.

public boolean getBoolean(int fieldIndex): returns the value of the given field as boolean.

etc.

Putting it all together at work:

following is a java class that connects to a database, executes a select query and displays the results.

```
import java.sql.*;

class DatabaseDemo {

public static void main( String arr[ ]) {

try {

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");    // Driver class is loaded.

Connection con = DriverManager.getConnection("jdbc:odbc:ourdsn","scott","tiger");

Statement stmt = con.createStatement();      // statement is created.

String qr= "select ename, job, sal from emp";

ResultSet rset = stmt.executeQuery(qr); // query is executed and result is placed in the resultset.

String ename,job;

int sal;

while ( rset.next() ) {                    // while there are records in the ResultSet.

ename = rset.getString(1);                 // value of first field as String.

job = rset.getString(2);                   // value of second field is converted into String.

sal = rset.getInt(3);                      // value of third field is converted into integer.

System.out.println(ename+"\t"+job+"\t"+sal); }

rset.close();                              // ResutSet is closed.

stmt.close();                              // statement is closed.
```

```
con.close();                                    // connection is closed.
} catch(Exception e) {
System.out.println(e); }
}}
```

Methods for Scrolling records in a ResultSet :

first(): Sets the record pointer at the first record.

Syntax:

public boolean first() throws SQLException;

last(): Sets the record pointer at the last record.

Syntax:

public boolean last() throws SQLException;

previous(): Sets the record pointer at the previous record.

Syntax:

public boolean previous() throws SQLException;

next(): Sets the record pointer at the next record.

Syntax:

public boolean next() throws SQLException;

absolute(): Sets the record pointer at the given record.

Syntax:

public boolean absolute(int rowIndex) throws SQLException;

relative (): moves the record pointer by the no. of record relative to the current record.

Syntax:

public boolean last() throws SQLException;

Working with Prepared Statements:

A prepared statement instance can be created using either of the following methods of Connection class:

public PreaparedStatement preapareStatement(String sql) throws SQLException;

or

public PreaparedStatement preapareStatement(String sql, int resultsetType, int resultsetConcurrency) throws SQLException;

PreparedStatements support parameterized queries and the parameters in queries are represented as ?.

e.g. to insert a record in emp table having the following schema emp(empno, ename, job,sal), a parameterized query can be formed as:

String qur= "insert into emp values(?,?,?,?)";

And prepared statement can be created as:

PreparedStatement pstmt = con.prepareStatement(qur);

Values of placeholders or parameters in the prepared statement query are filled using the **setType()** methods of PreparedStatement class. The general signature of the setType() methods are:

public void setType(int parameterIndex, Object parameterValue);

Some specific implementations of setType() method are:

public void setString(int parameterIndex, Object parameterValue) : sets the value of specified parameter as String.

public void setDate(int parameterIndex, Object parameterValue) : sets the value of specified parameter as Date.

public void setInt(int parameterIndex, Object parameterValue) : sets the value of specified parameter as integer.

etc.

e.g.: any record of the schema (empno, ename, job,sal) can be inserted in the emp table using prepared statements by the following code snippet:

```
System.out.println("Enter employee's no:");
String str = b.readLine();                    // b is a BufferedReader instance.
int eno = Integer.parseInt(str);
pstmt.setInt(1,eno);                          // value of first parameter is set as integer.
System.out.println("Enter employee's name:");
str = b.readLine();
pstmt.setString(2,str);                       // value of second parameter is set as String.
System.out.println("Enter employee's job:");
str = b.readLine();
pstmt.setString(3,str);                       // value of third parameter is set as String.
System.out.println("Enter employee's salary:");
str = b.readLine();
int sal = Integer.parseInt(str);
pstmt.setInt(4,sal);                          // value of fourth parameter is set as integer.
pstmt.executeUpdate();                        // query is execute.
```

the advantage of prepared statements over simple statements is that each time a new record is to be inserted, a record is to be deleted, modified or the criteria of selecting records is changed, there is no need to create new statements or recompile the program because new queries can be formed dynamically by changing the values of parameters at run time.

1. How many JDBC driver types does Sun define?
   A. One
   B. Two
   C. Three
   D. Four

2. Where is metadata stored in MySQL?
   A. In the MySQL database *metadata*
   B. In the MySQL database *metasql*
   C. In the MySQL database *mysql*
   D. None of the above is correct.

3. Who invented Java?
   A. Netscape
   B. Microsoft
   C. Sun
   D. None of the above is correct.

4. To run a compiled Java program, the machine must have what loaded and running?
   A. Java virtual machine
   B. Java compiler
   C. Java bytecode
   D. A Web browser

5. Which JDBC driver Type(s) can be used in either applet or servlet code?

A. Both Type 1 and Type 2

B. Both Type 1 and Type 3

C. Both Type 3 and Type 4

D. Type 4 only

6. _____ is an open source DBMS product that runs on UNIX, Linux and Windows.
   A. MySQL

   B. JSP/SQL

   C. JDBC/SQL

   D. Sun ACCESS

7. What is sent to the user via HTTP, invoked using the HTTP protocol on the user's computer, and run on the user's computer as an application?
   A. A Java application

   B. A Java applet

   C. A Java servlet

   D. None of the above is correct.

8. What MySQL property is used to create a surrogate key in MySQL?
   A. UNIQUE

   B. SEQUENCE

   C. AUTO_INCREMENT

   D. None of the above -- Surrogate keys are not implemented in MySQL.

9. What is **not** true of a Java bean?
   A. There are no public instance variables.

   B. All persistent values are accessed using get*xxx* and set*xxx* methods.

   C. It may have many constructors as necessary.

D.All of the above are true of a Java bean.

10. A JSP is transformed into a(n):

A.Java applet.

B.Java servlet.

C.Either 1 or 2 above.

D.Neither 1 nor 2 above.

D	C	C	A	C
A	B	C	C	B

What is a Servlet?

A Servlet is Java program that extends the functionality of a web server in a request-response programming model. Servlets run inside a Java enabled server or application server, such as the Sun One Application Server. Servlets are loaded and executed within the Java Virtual Machine (JVM) of the Web server or application server, in much the same way that applets are loaded and executed within the JVM of the Web browser. Since servlets run inside the servers, they do not need a graphical user interface (GUI).

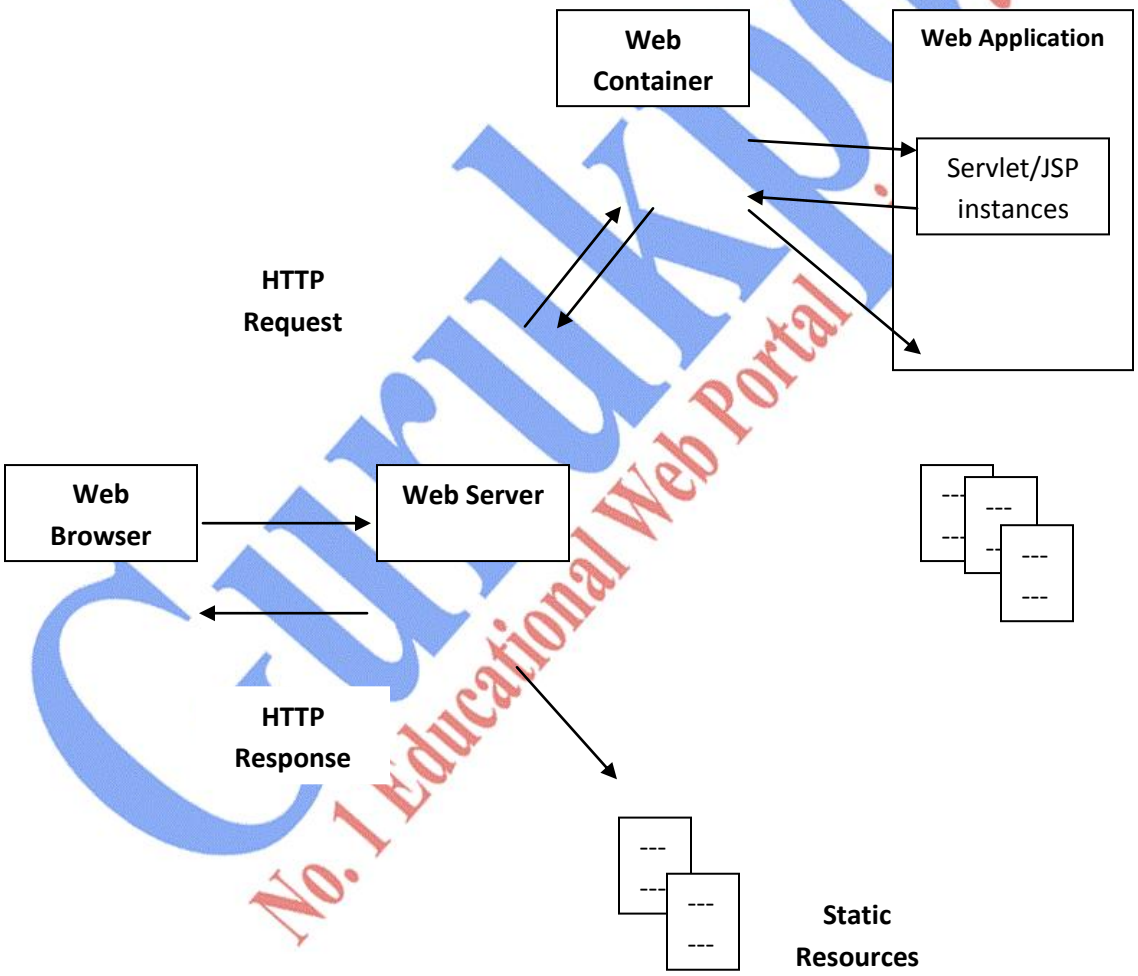
Sequence of events in a request-response model:

- The client sends a request to the server.
- If the request is for a web application the server delegates the request to the web container.
- The web container decides which web application should handle this request. Each incoming request can be mapped in a web application to a servlet, JSP or static resources such as HTML or XML pages and image files etc. if the request is for a static resource, the web container passes the resource as it is to the web server.

If the request is for a servlet then web container creates or locates the instance of the servlet and delegates the request to it.

- On receiving the request, the servlet dynamically generates an HTML page as response and gives it to the web server through the web container.
- The server sends the response back to the client.

Following diagram illustrates the working of a servlet.



Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes. The **javax.servlet** and **javax.http.servlet** packages provide interfaces and classes for writing servlets. All servlets must implement the Servlet interface, which defines life cycle methods. When implementing a generic service, you can use or extend the **GenericServlet** class provided with the Java Servlet API. The **HttpServlet** class provides methods, such as **doGet** and **doPost**, for handling HTTP-specific services. This chapter focuses on writing servlets that generate responses to HTTP requests. Some knowledge of the HTTP protocol is assumed.

A Brief Overview of Java Servlet API:

Java Servlet API is contained in javax.servlet and javax.servlet.http packages.

At the core of the servlet API is an interface named **Servlet** that specifies a contract between a web container and a servlet i.e. a web container uses this interface to reference servlets. Each servlet must implement Servlet interface. Servlet interface defines the following life-cycle methods of a servlet:

init(): This method is invoked by the web container when a servlet is loaded into memory i.e. when a servlet instance is created. As the name suggest it is used for initialization purposes.

Syntax:

public void init(ServletConfig config);

Here ServletConfig is another interface of the servlet API that provides functionality to access certain initialization parameters that can be configured while deploying a servlet.

service (): This method is invoked by the web container each time a request for the servlet is received. It contains code that generates the dynamic response.

Syntax:

Public void service (ServletRequest request, ServletResponse response);

ServletRequest and ServletResponse interfaces provide objects to access the underlying input and output streams associated with the client connection. These objects are used in service

method to read request data and to write data back to the client. These objects are instantiated by the web container and are passed to the servlet as arguments in service method.

destroy (): This method is invoked by the web container when the servlet is unloaded. It contains clean up code that is executed before winding up the servlet.

Protocol independent implementation of the Servlet interface is provided by the javax.servlet.GenericServlet class and HTTP specific implementation is provided by the javax.servlet.http.HttpServlet class.

HttpServlet class: Apart from init (), service (), and destroy () methods the HttpServlet class defines doGet() and doPost() methods that are used by the web container to serve HTTP get and post requests.

The basic difference between HTTP get and post request is that in get request, request parameters are sent as part of the request header which limits the no. of parameters that can be sent and the name and values of parameters are shown in the address bar of the client's browser which may not desirable in some applications.

In a post request the parameters are sent as part of the request body which means that any no. of parameters can be sent as well as parameters name and values are not shown in the address bar of the client's browser.

A HTTP specific servlet class extends HttpServlet and overrides either doGet() or doPost() or both.

Creation, deployment, and testing a simple web application having servlet:

A simple web application must have at least an html page having a form for requesting dynamic response from a servlet, one or more servlet classes and a deployment descriptor.

A deployment descriptor for web applications is a XML file named web.xml that contains the information required by the web container to manage the web application. It is usually generated by the deployment tool.

Q1.Which of the following are interface?
1.ServletContext

2.Servlet
3.GenericServlet
4.HttpServlet
A.1,2,3,4
B,1,2
C.1,3,4
D.1,4

Q2 1. Servlet is a Java technology based Web component.
2. Servlet servlets are platform-independent
3. Servlet has run on Web server which has a containers
4. Servlets interact with Web clients via a request/response using HTTP protocol.

A.1,2,3,4
B.1,2,3
C.1,3,4
D.None

Q3. Which of the following are class?
1.ServletContext
2.Servlet
3.GenericServlet
4.HttpServlet
A.1,2,3,4
B,1,2
C.3,4
D.1,4

Q4. Which of the following methods are main methods in life cycle of servlet?
1.init()
2 .service()
3.destroy()
4.srop()
5.wait()
A.1,2,3,4,5
B,1,2,3
C.3,4,5
D.1,4,5

Q5. init(),service() and destroy()methods are define in

1.javax.servlet.Servlet interface
2.javax.servlet.ServletHttp class
3.javax.servlet.ServletRequest interface
4.javax.servlet.ServletResponse interface

A.1,2,3,4,5
B,1

C.3,4,5
D.1,4,5

Q6. Int init() ,.service() and destroy() methods which methods are call only once at life cycle of servlets
1.init()
2.service()
3.destroy()
A.1,2,3
B,1,3
C.2
D.None

Q7. During initialization of servlet a servlet instance can throw

1.An UnavailableException
2 A ServletException
3.Both
4.None

Q8. Is Servlet thread Safe?
1.Yes
2.No
3.None

Q9. ServletContext is

1. an Interface
2.A container which is used to store an object so that it is available for whole application
3.A container which is used to store an object so that it is available for session only.
4.A container which is used to store an object so that it is available for request only.
A.1,2,3,4
B.1,2
C.1,3,4
D.4

Q10. A servlet can access the headers of an HTTP request through which following methods of the HttpServletRequest interface:
1.getHeader()
2 getHeaders()

3.getHeaderNames()
4.All
5.None

1. (b)	2. (a)	3. (c)	4. (b)	5. (b)
6. (b)	7. (c)	8. (b)	9. (b)	10. (d)

GLOSSARY

1. **ActiveX control**

   An object that you place on a form to enable or enhance a user's interaction with an application. ActiveX controls have events and can be incorporated into other controls. The controls have an .ocx file name extension.

2. **ActiveX object**

   An object that is exposed to other applications or programming tools through Automation interfaces.

3. **collection**

   An object that contains a set of related objects. An object's position in the collection can change whenever a change occurs in the collection; therefore, the position of any specific object in the collection may vary.

4. **seed**

   An initial value used to generate pseudorandom numbers. For example, the **Randomize** statement creates a seed number used by the **Rnd** function to create unique pseudorandom number sequences.

5. Connection Pool

A set of database connections managed by the application server for the various applications it manages.

6.Custom Result Mapping

The Custom Result Mapping dialog provides a similar alias ability for repeating rows in SQL result sets as the Declare Group and Repeat for Group actions do for repeating elements in a document.

7. Declare Group/Repeat Tab

This tab of the Custom Results Mapping dialog is used to create groups of result set records on one or more result set columns, create a Group Alias to use as a Context for Detail Rows, and create a Group Alias to use as a Context for Map Targets (creating Group Headers).

8. Detail Rows Tab

This tab of the Custom Results Mapping dialog allows you to create a mapping alias tied to either a document Context or a Group/Repeat alias Context. Use of the Detail Rows tab is optional.

9. DOM

A Document Object Model (DOM) is an XML document constructed as an object in a software program's memory. It provides standard methods for manipulating the object. In Composer, DOM is often synonymous with XML Document. DOMs are represented as hierarchical trees with a single root node.

10. DOM Context

The name of a DOM (Input, Output, Temp, etc.), or the name of a Repeat alias previously defined in the component. (The alias itself represents a DOM context, representing the nodepath hierarchy upstream of a given element.)

11. Execute SQL Action

Same as SQL Statement Action.

12. JDBC

A Sun trademark for the Java API for accessing relational database data. It is commonly assumed to mean Java Database Connectivity.

13. Map Target Tab

This tab of the Custom Results Mapping dialog is used to create target element names for each result set column and specify a target Context for each result set column.

14. Native Environment Pane

A pane in the JDBC Component Editor that simulates an actual SQL environment when you issue a query.

15. Query/Result Mapping Pane

(Same as the Native Environment Pane.) A pane in the JDBC Component Editor that includes three tabs: the SQL Statement tab, the Result Mapping tab, and the Results Text tab.

16. Result Mapping Tab

A tab in the Query/Result Mapping Pane that allows you to map the result of your database query to an XML document.

17. Result Text Tab

A tab in the Query/Result Mapping Pane that displays the actual data that was returned following the execution of the database query.

18. Row Target

The receiving element in a mapping operation is called the *row target*. It represents a specific position in the DOM tree of an XML file.

19. SQL Statement Action

Most commonly used to query an existing database and then map the result to an XML document.

20.**servlet**

> an application designed to run on a server in the womb of a permanently resident **CGI (Common Gateway Interface)** mother program written in **Java** that provides services for it, much the way an Applet runs in the womb of a Web browser.

Books:-

Head First Servlets and JSP: Passing the Sun Certified Web Component Developer Exam

Servlet and JSP (A Tutorial) by Budi Kurniawan

Jdbc, Servlets, And Jsp Black Book, New Edition

Servlet & JSP: A Tutorial (A Tutorial series) by Budi Kurniawan

JDBC 3: Java Database Connectivity by Bernard Van Haecke

Learning VBScript by Paul Lomax

Java Servlet & JSP Cookbook by **Bruce W. Perry**

XML: Visual QuickStart Guide (2nd Edition) by **Kevin Howard Goldberg**

Websites:-

www.**tutorials**point.com

www.**servlets**.com

www.redbooks.ibm.com

www.en.wikipedia.org

www.roseindia.net

www.mkyong.com

www.**java**-samples.com

www.w3schools.com

*www.**xml**master.org*