Karpagam Academy of Higher Education

(Deemed to be University)
(Established Under Section 3 of UGC Act 1956)
Coimbatore – 641 021

DEPARTMENT OF INFORMATION TECHNOLOGY

17ITU404A **INTERNET TECHNOLOGIES** **4H – 4C**

Instruction Hours / week:L: 3 T: 0 P: 0 **Marks:**Int :**40** Ext : **60** Total: **100**

SCOPE

This course relates to the interface between web servers and their clients. The course provides information includes markup languages, programming interfaces and languages, and standards for document identification adn display. The use of Web technology makes to enhance active student learning and improves their creativity in making web pages.

OBJECTIVES

- To create new web page
- To understand the fundamental features of web applications
- To understand the objects and components needed for a web designing.

UNIT – I

Java: Use of Objects, Array and ArrayList class JavaScript: Datatypes, operators, functions, control structures, events and event handling.

UNIT – II

JDBC: JDBC fundamentals, Establishing Connectivity and working with connection interface, working with statements, creating and executing SQL statements, working with ResultSet objects.

UNIT – III

JSP: Introduction to JavaServerPages, HTTP and Servlet Basics, The Problem with Servlets, The Anatomy of a JSP Page, JSP Processing, JSP Application Design with MVC, Setting Up the JSP Environment.

UNIT – IV

JSP: Implicit Objects, conditional processing, displaying values, Using an expression to Set an Attribute, declaring variables and methods, error handling and debugging, sharing data between JSP pages, Requests, and Users, Database Access.

UNIT – V

Java beans: Jaba Beans Fundamentals, JAR Files, Introspection, Developing a simple Bean, Connecting to DB.

Suggested Readings

1. Ivan Bayross (2009). Web Enabled Commercial Application Development Using Html, Dhtml, Javascript, PerlCgi. BPB Publications.
2. Cay Hortstmann (2009). BIG Java (3rd ed.). Wiley Publication
3. Herbert Schildt (2009). Java 7 The Complete Reference (8th ed.).
4. Jim Keogh (2002). The Complete Reference J2EE. TMH.
5. Hans Bergsten (2003). Java ServerPages (3rd ed.). O'Reilly.

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University Established Under Section 3 of UGC Act 1956) **Coimbatore – 641 021.**

LECTURE PLAN
DEPARTMENT OF INFORMATION TECHNOLOGY

STAFF NAME: P.A.MONISHA
SUBJECT NAME: INTERNET TECHNOLOGIES
SUB.CODE: 17ITU404A
SEMESTER: IV CLASS: II B.SC IT

S.No	Lecture Duration Period	Topics to be Covered	Support Material/Page Nos
		UNIT-I	
1.	1	Introduction to Java	T2,W1
2.	1	Introduction to Java	T2,W1
3.	1	Array and array list class	T2,W1
4.	1	Introduction to java script	T1,W1
5.	1	Datatype	T1,W1
6.	1	Operators	W2
7.	1	Functions	T1:126-129
8.	1	Control structure Event Handling	T1:133-136
9.	1	Recapitulation & Important topic	
	Total No of Hours Planned For Unit 1=9		
		UNIT-II	
1.	1	Introduction to JDBC	T4:123-124
2.	1	JDBC fundamentals	T4:124-129
3.	1	JDBC fundamentals	T4:124-129
4.	1	Establishing connectivity	T4:130-133
5.	1	working with statement	T4:135-140

6.	1	Creating and executing SQL	W1
7.	1	Creating and executing SQL	W1
8.	1	Network with Result Set Object	T4:141-148
9.	1	Recapitulation &Important Question Discussion	
	Total No of Hours Planned For Unit II=9		
		UNIT-III	
1.	1	Introduction to Java Server Pages	T4:379
2.	1	HTTP and Servlet Basics	T4:347
3.	1	The Problem with Servlets	T4:348-350
4.	1	The Anatomy of a JSP Page	T4:352
5.	1	JSP Processing	W1
6.	1	JSP Application Design with MVC	W1
7.	1	Setting up the JSP Environment	W2
8.	1	Recapitulation &Important Question Discussion	
	Total No of Hours Planned For Unit III=8		
		UNIT-IV	
1.	1	JSP: Implicit objects	T4:384
2.	1	Conditional processing	W1
3.	1	JSP: displaying values	W1
4.	1	Using an expression to set an attribute	W1
5.	1	Declaring variables and methods, Error Handling and Debugging	W1
6.	1	Sharing data between JSP pages	W1
7.	1	Requests, and users, Database access	W1
8.	1	Recapitulation &important questions discussion	
	Total No of Hours Planned For Unit IV=8		
		UNIT-V	

1.	1	Java beans	T4:18
2.	1	Java beans: java beans fundamentals	T4:443
3.	1	Jar files	T4:443
4.	1	Jar files	T4:443
5.	1	Introspection	W1
6.	1	Developing a simple bean	W1
7.	1	Developing a simple bean	W1
8.	1	Connecting to db	W1
9.	1	Connecting to db	W1
10.	1	Recapitulation & important questions discussion	
11.	1	Discussing previous years question papers	
	Total No of Hours Planned for unit V=16		
Total Planned Hours	**45**		

Suggested Readings

1. Ivan Bayross (2009). Web Enabled Commercial Application Development Using Html, Dhtml, Java script, PerlCgi. BPB Publications.
2. Cay Hortstmann (2009). BIG Java (3rd ed.). Wiley Publication
3. Herbert Schildt (2009). Java 7 The Complete Reference (8th ed.).
4. Jim Keogh (2002). The Complete Reference J2EE. TMH.
5. Hans Bergsten (2003). Java Server Pages (3rd ed.). O'Reilly.

Websites:

W1:www.http:/tutorialpoint.com
W2:www.w3.schools.com

UNIT-I

SYLLABUS

JAVA: Use of Objects, Array and ArrayList class JavaScript: Datatypes, operators, functions, control structures, events and event handling.

INTRODUCTION TO JAVA:

Java programming language was originally developed by Sun Microsystems which was initiated by James Gosling and released in 1995 as core component of Sun Microsystems' Java platform (Java 1.0 [J2SE]).

The latest release of the Java Standard Edition is Java SE 8. With the advancement of Java and its widespread popularity, multiple configurations were built to suit various types of platforms. For example: J2EE for Enterprise Applications, J2ME for Mobile Applications.

The new J2 versions were renamed as Java SE, Java EE, and Java ME respectively. Java is guaranteed to be **Write Once, Run Anywhere.**

Java is −

- **Object Oriented** − In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
- **Platform Independent** − Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.
- **Simple** − Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
- **Secure** − With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
- **Architecture-neutral** − Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.

- **Portable** − Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.
- **Robust** − Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.
- **Multithreaded** − With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.
- **Interpreted** − Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.
- **High Performance** − With the use of Just-In-Time compilers, Java enables high performance.
- **Distributed** − Java is designed for the distributed environment of the internet.
- **Dynamic** − Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

History of Java:

James Gosling initiated Java language project in June 1991 for use in one of his many set-top box projects. The language, initially called ‘Oak’ after an oak tree that stood outside Gosling's office, also went by the name ‘Green’ and ended up later being renamed as Java, from a list of random words.

Sun released the first public implementation as Java 1.0 in 1995. It promised **Write Once, Run Anywhere** (WORA), providing no-cost run-times on popular platforms.

On 13 November, 2006, Sun released much of Java as free and open source software under the terms of the GNU General Public License (GPL).

On 8 May, 2007, Sun finished the process, making all of Java's core code free and open-source, aside from a small portion of code to which Sun did not hold the copyright.

CREATING OBJECT:

An object is created from a class. In Java, the new keyword is used to create new objects.

There are three steps when creating an object from a class:

- **Declaration** − A variable declaration with a variable name with an object type.
- **Instantiation** − The 'new' keyword is used to create the object.
- **Initialization** − The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Following is an example of creating an object −

Example:

```
public class Puppy {
  public Puppy(String name) {
    // This constructor has one parameter, name.
    System.out.println("Passed Name is :" + name );
  }

  public static void main(String []args) {
    // Following statement would create an object
    myPuppy Puppy myPuppy = new Puppy( "tommy" );
  }
}
```

If we compile and run the above program, then it will produce the following result −

Output:

```
Passed Name is :tommy
```

Declaring, Instantiating and Initializing an Object

```
import java.util.Date;
class DateApp {
   public static void main (String args[]) {
      Date today = new Date();
      System.out.println(today);
   }
}
```

The main () method of the DateApp application creates a Date object named today. This single statement performs three actions: declaration, instantiation, and initialization. Date today declares to the compiler that the name today will be used to refer to an object whose type is Date, the newoperator instantiates new Date object, and Date() initializes the object.

Declaring an Object:

Declarations can appear as part of object creation as you saw above or can appear alone like this

Date today;

Either way, a declaration takes the form of

type name

where *type* is either a simple data type such as int, float, or boolean, or a complex data type such as a class like the Date class. *name* is the name to be used for the variable. Declarations simply notify the compiler that you will be using *name* to refer to a variable whose type is *type*. **Declarations do not instantiate objects.** To instantiate a Date object, or any other object, use the new operator.

Instantiating an Object:

The new operator instantiates a new object by allocating memory for it. new requires a single argument: a *constructor method* for the object to be created. The constructor method is responsible for initializing the new object.

Initializing an Object:

Classes provide constructor methods to initialize a new object of that type. In a class declaration, constructors can be distinguished from other methods because they have the same name as the class and have no return type. For example, the method signature for

Date constructor used by the DateApp application is

Date()

A constructor such as the one shown, that takes no arguments, is known as the *default constructor*. Like Date, most classes have at least one constructor, the default constructor. However, classes can have multiple constructors, all with the same name but with a different number or type of arguments. For example, the Date class supports a constructor that requires three integers:

Date(int year, int month, int day)

that initializes the new Date to the year, month and day specified by the three parameters.

ARRAY IN JAVA:

- Simple fixed sized arrays that we create in Java, like below

```
int arr[] = new int[10]
```

- An array is basic functionality provided by Java.
- Array can contain both primitive data types as well as objects of a class depending on the definition of the array.
- Array can be multi dimensional

Example: Integer addarrayobject[][] = new Integer[3][2];

- In array we insert elements using the assignment operator.

Example: addarrayobject[0]= new Integer(8) ; //new object is added to the array object

- Each array object has the length variable which returns the length of the array.

Example: arraylength= arrayobject.length ; //uses arrayobject length variable

ARRAYLIST IN JAVA:

- Dynamic sized arrays in Java that implement List interface.

```
ArrayList<Type> arrL = new ArrayList<Type>();
```

Here Type is the type of elements in ArrayList to be created

- ArrayList is part of collection framework in Java. Therefore array members are accessed using [],ArrayList has a set of methods to access elements and modify them.
- One need not to mention the size of Arraylist while creating its object. Even if we specify some initial capacity, we can add more elements.
- ArrayList only supports object entries, not the primitive data types.
- ArrayList can not contains primitive data types (like int , float , double) it can only contains Object
  Example: ArrayList arraylistobject = new ArrayList();
- Length of the ArrayList is provided by the size() method
  Example: Integer arrayobject[] = new Integer[3];
- We can insert elements into the arraylist object using the add() method **Example:** Integer addarrayobject[] = new Integer[3];
- ArrayList is always single dimensional.
  Example: Integer addarrayobject[][] = new Integer[3][2];

Example Program for Array and ArrayList:

```
import java.util.ArrayList;
import java.util.Iterator;

public class ArrayArrayListExample {

   public static void main(String[] args) {

      // ArrayList Example

      ArrayList<String> arrlistobj = new
      ArrayList<String>(); arrlistobj.add("Alive is
      awesome"); arrlistobj.add("Love yourself");
```

```
        Iterator it = arrlistobj.iterator();
        System.out.print("ArrayList object output :");
        while(it.hasNext())
        System.out.print(it.next() + " ");

          // Array Example

        String[] arrayobj = new String[3];
        arrayobj[0]= "Love yourself";
        arrayobj[1]= "Alive is awesome";
        arrayobj[2]= "Be in Present";
        System.out.print("Array object output :");
        for(int i=0; i < arrayobj.length ;i++)
        System.out.print(arrayobj[i] + " ");

    }
}
```

DIFFERENCE OF ARRAY AND ARRAY LIST

ARRAY	ARRAYLIST
Stores primitive data types and also objects	Stores only objects
Defined in Java language itself as a fundamental data structure	Belongs to collections framework
Fixed size	Growable and resizable. Elements can be added or removed
Stores similar data of one type	Can store heterogeneous data types

It is not a class	It is a class with many methods
Cannot be synchronized	Can be obtained a synchronized version
Elements retrieved with for loop	Can be retrieved with for loop and iterators
Elements accessible with index number	Accessing methods like get() etc. are available
Can be multidimensional	—

SIMILIRITIES OF ARRAY AND ARRAYLIST:

- add and get method : Performance of Array and ArrayList are similar for the add and get operations .Both operations runs in constant time.
- Duplicate elements: Both array and arraylist can contain duplicate elements.
- Null Values: Both can store null values and uses index to refer to their elements.
- Unordered: Both do not guarantee ordered elements.

JAVASCRIPTS DATATYPES:

JavaScript includes data types similar to other programming languages like Java or C#. Data type indicates characteristics of data. It tells the compiler whether the data value is numeric, alphabetic; date etc., so that it can perform the appropriate operation.

JavaScript provides different **data types** to hold different types of values. There are two types of data types in JavaScript.

1. Primitive data type
2. Non-primitive (reference) data type

JavaScript is a **dynamic type language**, means you don't need to specify type of the variable because it is dynamically used by JavaScript engine. You need to use **var** here to specify the data type. It can hold any type of values such as numbers, strings etc. For example:

```
var a=40;//holding number

var b="Rahul";//holding string
```

JavaScript primitive data types:

There are five types of primitive data types in JavaScript. They are as follows:

Data Type	Description
String	represents sequence of characters e.g. "hello"
Number	represents numeric values e.g. 100
Boolean	represents boolean value either false or true
Undefined	represents undefined value
Null	represents null i.e. no value at all

JavaScript Number Data types:

JavaScript has only one Number (numeric) data types. Number data type can store normal integer, floating-point values.

A floating-point represent a decimal integer with either decimal points or fraction expressed (refer to another decimal number).

```
1. var num1 = 5;                    // Numeric integer value
2. var num2 = 10.5;                 // Numeric float value
3. var num3 = -30.47;               // Negative numeric float value
```

```
4. var num4 = 5E4;                    // 5 x 10 Powers of 4 = 50000
5. var num5 = 5E-4                    // 5 x 10 Powers of -4 = -50000

6. var num6 = 10 / 0;                 // Number divide by zero, result is: Infinity
7. var num7 = 10 / -0;                // Number divide by negative zero, result is: -
   Infinity
8. var num8 = 10, num9 = 12.5; // Multiple variable declare and initialize
```

JavaScript String Data types:

JavaScript string data type represent textual data surrounding to single/double quotes. Each character is represent as a element that occupies the position of that string. Index value 0, starting from first character of the string.

```
1. var name = 'Hello, I am run this town.!';     // Single quote
2. var name = "Hello, I am run this town.!";     // Double quote
```

Whatever quote you use to represent string, but you should take care that quote can't repeat in string statement.

```
1. var name = "Hello, I'm run this town.!";      // Single quote use inside string
2. var name1 = ""; // empty string
```

Note : JavaScript empty string is different from the NULL value.

JavaScript Boolean Data types:

JavaScript Boolean type can have two value `true` or `false`. Boolean type is use to perform logically operator to determine condition/expression is true.

```
1. var val1 = true;
2. var val2 = false;
```

JavaScript Symbol Data types:

JavaScript Symbol data type new (Currently ECMAScript 6 Drafted) used for identifier unique object properties.

```
Symbol([description])
```

Parameters : description is optional for identify unique symbol.

```
1. var s1 = Symbol();
2. var s1 = Symbol('name');
```

JavaScript Null:

JavaScript Null specifies variable is declare but values of this variable is empty.

```
1. var str = null;                          // Value assign null
2. document.writeln(str == undefined);    // Returns true
3. document.writeln(null == undefined); // Equality check Returns true
4. document.writeln(null === undefined);          // Equality with type check Returns
   false
```

JavaScript empty variables boolean context return false.

```
1. var bool = Boolean();
2. console.log(bool);              // Default Boolean context Returns false
3. var bool = Boolean(true);
4. console.log(bool);              // Returns true
```

JavaScript undefined:

JavaScript uninitialized variables value are undefined.Uninitialized variable (value undefined) equal to null. JavaScript uninitialized variables boolean context return false.

```
1. var str;                 // Declare variable without value. Identify as undefined value.
2. document.writeln(str == null);                       // Returns true
3. document.writeln(undefined == null);   // Returns true
4. var bool = Boolean(str);          // str is undefined passed into Boolean object
5. document.writeln(bool);          // Boolean context Returns false
```

JavaScript non-primitive data types :

The non-primitive data types are as follows:

Data Type	Description

Object	represents instance through which we can access members
Array	represents group of similar values
RegExp	represents regular expression

JAVASCRIPT OPERATORS:

JavaScript operators are symbols that are used to perform operations on operands. For example:

1. var sum=10+20;

Here, + is the arithmetic operator and = is the assignment operator.

There are following types of operators in JavaScript.

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Bitwise Operators
4. Logical Operators
5. Assignment Operators
6. Special Operators

JavaScript Arithmetic Operators:

Arithmetic operators are used to perform arithmetic operations on the operands. The following operators are known as JavaScript arithmetic operators.

Operator	Description	Example
+	Addition	10+20 = 30
-	Subtraction	20-10 = 10

*	Multiplication	10*20 = 200
/	Division	20/10 = 2
%	Modulus (Remainder)	20%10 = 0
++	Increment	var a=10; a++; Now a = 11
--	Decrement	var a=10; a--; Now a = 9

JavaScript Comparison Operators:

The JavaScript comparison operator compares the two operands. The comparison operators are as follows:

Operator	Description	Example
==	Is equal to	10==20 = false
===	Identical (equal and of same type)	10==20 = false
!=	Not equal to	10!=20 = true
!==	Not Identical	20!==20 = false
>	Greater than	20>10 = true
>=	Greater than or equal to	20>=10 = true
<	Less than	20<10 = false

<=	Less than or equal to	20<=10 = false

JavaScript Bitwise Operators:

The bitwise operators perform bitwise operations on operands. The bitwise operators are as follows:

Operator	Description	Example
&	Bitwise AND	(10==20 & 20==33) = false
|	Bitwise OR	(10==20 | 20==33) = false
^	Bitwise XOR	(10==20 ^ 20==33) = false
~	Bitwise NOT	(~10) = -10
<<	Bitwise Left Shift	(10<<2) = 40
>>	Bitwise Right Shift	(10>>2) = 2
>>>	Bitwise Right Shift with Zero	(10>>>2) = 2

JavaScript Logical Operators:

The following operators are known as JavaScript logical operators.

Operator	Description	Example
&&	Logical AND	(10==20 && 20==33) = false
||	Logical OR	(10==20 || 20==33) = false
!	Logical Not	!(10==20) = true

JavaScript Assignment Operators:

The following operators are known as JavaScript assignment operators.

Operator	Description	Example
=	Assign	10+10 = 20
+=	Add and assign	var a=10; a+=20; Now a = 30
-=	Subtract and assign	var a=20; a-=10; Now a = 10
*=	Multiply and assign	var a=10; a*=20; Now a = 200
/=	Divide and assign	var a=10; a/=2; Now a = 5
%=	Modulus and assign	var a=10; a%=2; Now a = 0

JavaScript Special Operators:

The following operators are known as JavaScript special operators.

Operator	Description
(?:)	Conditional Operator returns value based on the condition. It is like if-else.
,	Comma Operator allows multiple expressions to be evaluated as single statement.
delete	Delete Operator deletes a property from the object.
in	In Operator checks if object has the given property
instanceof	checks if the object is an instance of given type

new	creates an instance (object)
typeof	checks the type of object.
void	it discards the expression's return value.
yield	checks what is returned in a generator by the generator's iterator.

JAVASCRIPT FUNCTION:

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).

```
function myFunction(p1, p2) {
  return p1 * p2;          // The function returns the product of p1 and p2
}
```

JavaScript Function Syntax:

A JavaScript function is defined with the **function** keyword, followed by a **name**, followed by parentheses **()**.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:
(*parameter1, parameter2, ...*)

The code to be executed, by the function, is placed inside curly brackets: **{}**

```
function name(parameter1, parameter2, parameter3) {
  code to be executed
}
```

Function **parameters** are listed inside the brackets () of the function definition.

Function **arguments** are the **values** received by the function when it is invoked.

Inside the function, the arguments (the parameters) behave as local variables.

A Function is much the same as a Procedure or a Subroutine, in other programming languages.

Function Invocation:

The code inside the function will execute when "something" **invokes** (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

You will learn a lot more about function invocation later in this tutorial.

Function Return:

When JavaScript reaches a **return statement**, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a **return value**. The return value is "returned" back to the "caller":

Example:

Calculate the product of two numbers, and return the result:

```
var x = myFunction(4, 3);        // Function is called, return value will end up in x

function myFunction(a, b) {
  return a * b;                  // Function returns the product of a and b
}
```

The result in x will be:

12

JavaScript Function Definitions:

JavaScript functions are **defined** with the **function** keyword.

Function Definition:

Before we use a function, we need to define it. The most common way to define a function in JavaScript is by using the **function**keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

Syntax:

The basic syntax is shown here.

```
<script type="text/javascript">
  <!--
    function functionname(parameter-list)
    {
      statements
    }
  //-->
</script>
```

Function Declarations:

Earlier in this tutorial, you learned that functions are **declared** with the following syntax:

function *functionName*(*parameters*) {
  code to be executed
}

Declared functions are not executed immediately. They are "saved for later use", and will be executed later, when they are invoked (called upon).

Example:

```
function myFunction(a, b) {
  return a * b;
}
```

Function Expressions:

A JavaScript function can also be defined using an **expression**.

A function expression can be stored in a variable:

Example:

```
var x = function (a, b) {return a * b};
```

Example:

```
var x = function (a, b) {return a * b};
var z = x(4, 3);
```

The function above is actually an **anonymous function** (a function without a name).

Functions stored in variables do not need function names. They are always invoked (called) using the variable name.

The Function() Constructor:

As you have seen in the previous examples, JavaScript functions are defined with the **function** keyword.

Functions can also be defined with a built-in JavaScript function constructor called Function().

Example:

```
var myFunction = new Function("a", "b", "return a * b");

var x = myFunction(4, 3);
```

Functions Can Be Used as Values:

JavaScript functions can be used as values:

Example:

```
function myFunction(a, b) {
  return a * b;
}

var x = myFunction(4, 3);
```

Functions are Objects:

The **typeof** operator in JavaScript returns "function" for functions.

But, JavaScript functions can best be described as objects.

JavaScript functions have both **properties** and **methods**.

The arguments.length property returns the number of arguments received when the function was invoked:

Example:

```
function myFunction(a, b) {
  return arguments.length;
}
```

CONTROL STRUCTURE:

- The control structures within JavaScript allow the program flow to change within a unit of code or function. These statements can determine whether or not given statements are executed, as well as repeated execution of a block of code.
- Most of the statements enlisted below are so-called conditional statements that can operate either on a statement or a block of code enclosed with braces ({ and }). The same structures utilize Booleans to determine whether or not a block gets executed, where any defined variable that is neither zero nor an empty string is treated as true.

Control structure actually controls the flow of execution of a program. Following are the several control structure supported by javascript.

- if … else
- switch case
- do while loop
- while loop
- for loop

If … else

The if statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

Syntax

```
if (expression){
  Statement(s) to be executed if expression is true
}
```

Example

```
<script type="text/javascript">
<!--
var age = 20;
if( age > 18 ){
  document.write("<b>Qualifies for driving</b>");
}
```

```
//-->

</script>
```

Switch case

The basic syntax of the switch statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.

Syntax

```
switch (expression)
{
 case condition 1: statement(s)
                   break;
 case condition 2: statement(s)
                   break;
  ...
 case condition n: statement(s)
                   break;
 default: statement(s)
}
```

Example

```
<script type="text/javascript">
<!--
var grade='A';
document.write("Entering switch block<br/>");
switch (grade)
{
  case 'A': document.write("Good job<br/>");
            break;
  case 'B': document.write("Pretty good<br/>");
            break;
  case 'C': document.write("Passed<br/>");
            break;
  case 'D': document.write("Not so good<br/>");
            break;
  case 'F': document.write("Failed<br/>");
            break;
  default:  document.write("Unknown grade<br/>")
}
document.write("Exiting switch block");
```

```
//-->

</script>
```

Do while Loop

The do...while loop is similar to the while loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is false.

Syntax

```
do{

   Statement(s) to be executed;

} while (expression);
```

Example

```
<script type="text/javascript">

<!--

var count = 0;

document.write("Starting Loop" + "<br/>");

do{

  document.write("Current Count : " + count + "<br/>");

  count++;

}while (count < 0);

document.write("Loop stopped!");
```

```
//-->

</script>
```

This will produce following result −

```
Starting Loop
Current Count : 0
Loop stopped!
```

While Loop

The purpose of a while loop is to execute a statement or code block repeatedly as long as expression is true. Once expression becomes false, the loop will be exited.

Syntax

```
while (expression){

  Statement(s) to be executed if expression is true

}
```

Example

```
<script type="text/javascript">

<!--

var count = 0;

document.write("Starting Loop" + "<br/>");

while (count < 10){

  document.write("Current Count : " + count + "<br/>");

  count++;
```

```
}

document.write("Loop stopped!");

//-->

</script>
```

This will produce following result −

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
```

For Loop

The for loop is the most compact form of looping and includes the following three important parts −

- The loop initialization where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The test statement which will test if the given condition is true or not. If condition is true then code given inside the loop will be executed otherwise loop will come out.
- The iteration statement where you can increase or decrease your counter.

Syntax

```
for (initialization; test condition; iteration statement){

  Statement(s) to be executed if test condition is true

}
```

Example

```
<script type="text/javascript">

<!--

var count;

document.write("Starting Loop" + "<br/>");

for(count = 0; count < 10; count++){

  document.write("Current Count : " + count );

  document.write("<br/>");

}

document.write("Loop stopped!");

//-->

</script>
```

This will produce following result which is similar to while loop −

```
Starting Loop
Current Count : 0
Current Count : 1
```

```
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
```

with

The with statement is used to extend the scope chain for a block[1] and has the following syntax:

```
with (expression) {
  // statement
}
```

Pros

The with statement can help to

- reduce file size by reducing the need to repeat a lengthy object reference, and
- relieve the interpreter of parsing repeated object references.

However, in many cases, this can be achieved by using a temporary variable to store a reference to the desired object.

Cons

The with statement forces the specified object to be searched first for all name lookups. Therefore

- all identifiers that aren't members of the specified object will be found more slowly in a 'with' block and should only be used to encompass code blocks that access members of the object.
- with makes it difficult for a human or a machine to find out which object was meant by searching the *scope chain*.
- Used with something else than a plain object, with may not be forward-compatible.

Therefore, the use of the with statement is not recommended, as it may be the source of confusing bugs and compatibility issues.

Example

```
var area;
var r = 10;

with (Math) {
  a = PI*r*r;      // == a = Math.PI*r*r
  x = r*cos(PI); // == a = r*Math.cos(Math.PI); y
  = r*sin(PI/2); // == a = r*Math.sin(Math.PI/2);
}
```

JAVASCRIPT EVENTS:

What is an Event ?

- JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.
- When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.
- Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.

- Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

- Please go through this small tutorial for a better understandingHTML Event Reference. Here we will see a few examples to understand a relation between Event and JavaScript

Event Handlers:

An event occurs when something happens in a browser window. The kinds of events that might occur are due to:

- A document loading
- The user clicking a mouse button
- The browser screen changing size

When a function is assigned to an event handler, that function is run when that event occurs.

A handler that is assigned from a script used the syntax '[element].[event] = [function];', where [element] is a page element, [event] is the name of the selected event and [function] is the name of the function that occurs when the event takes place.

Different event handlers with with different HTML tags. For example, while "onclick" can be inserted into most HTML tags to respond to that tag's onclick action, something like "onload" (see below) only works inside the <body> and <img> tags. Below are some of the most commonly used event handlers supported by JavaScript:

Event Handlers:

onclick:	Use this to invoke JavaScript upon clicking (a link, or form boxes)
onload:	Use this to invoke JavaScript after the page or an image has finished loading.
onmouseover:	Use this to invoke JavaScript if the mouse passes by some link
onmouseout:	Use this to invoke JavaScript if the mouse goes pass some link

onunload:	Use this to invoke JavaScript right after someone leaves this page.

onClick Event handler

Ok, lets see how the `onclick` event-handler can help us. Remember, any event handlers are added **inside html tags**, not inside `<script></script>` (there is an alternate way, which will not be discussed in this section). First of all, does that mean we can add event handlers inside any html tag? Noooo! In the case of the" onClick" event handler, which executes a piece of JavaScript when an element is clicked on, it can only be added to visible elements on the page such as <a>, form buttons, check boxes, a DIV etc. You wouldn't expect to be able to add this event handler inside the <head> tag for example, and you can't. With that understanding, lets see an example:

Click here for output:

```
<script>
function inform(){
    alert("You have activated me by clicking the grey button! Note that the event handler is added within the event that it handles, in this case, the form button event tag")
}
</script>

<form>
<input type="button" name="test" value="Click me" onclick="inform()">
</form>
```

The function `inform()` is invoked when the user clicks the button.

Ok, let me show you another example that adds the `onclick` event inside form radio buttons:

Try this: (it will change the background color of a document interactively)

```
<form name="go">
1
<input type="radio" name="C1" onclick="document.bgColor='lightblue'">
2
<input type="radio" name="C2" onclick="document.bgColor='lightyellow'">
3
4input type="radio" name="C3" onclick="document.bgColor='lightgreen'"<font face="Courier New">>
5
</form></font>
```

I've put the actual demo of this example onto another window. Click the button to see it.

Here used the `onclick` handler to change the background color. Notice that we just wrote in plain English the name of the bgcolor...you can do that, for most colors, or for a greater selection, use its hex value (ie: #000000).

onLoad Event handlers

The onload event handler is used to call the execution of JavaScript after a page, frame or image has completely loaded. It is added like this:

```
1<body onload="inform()"> //Execution of code //will begin after the page has loaded.
<frameset onload="inform()"> //Execution of code //will begin after the current frame has
1
loaded.
<img src="whatever.gif" onload="inform()"> //Execution of code will begin after the image has
1
loaded.
```

Lets see an example of an onload handler:

```
1<html>
2<head><title>Body onload example</title>
3</head>
4<body onload="alert('This page has finished loading!')">
5Welcome to my page
6</body>
7</html>
```

Depending on the complexity or nature of your JavaScript, sometimes it is necessary to wait until the entire page has loaded before that script is run to prevent any potential problems or make sure the aspect of the page you're manipulating has been downloaded. This is where the onload event handler comes in handy.

Executing a JavaScript function after the page has loaded is necessary sometimes

onMouseover, onMouseout

Next up, the onMouseover and onMouseout event handlers. Just like the "onClick" event, these events can be added to visible elements such as a link (<a>), DIV, even inside the <BODY> tag, and are triggered when the mouse moves over and out of the element (big surpise). Lets create a once very popular effect- display a status bar message when the mouse moves over a link:

Output: Dynamic Drive

```
1<a href="blabla.htm" onmouseover="status='DHTML code library!';return true" onmouseout="status=' '">Dynamic Drive</a>
```

Several new concepts arise here, so I'll go over each one of them.

- the "status" refers to window.status, which is how you write to the status bar.
- Note that instead of calling a function, we called directly two JavaScript statements inside the event handler :"status='Do not click here, its empty!';return true" This is ok, but you must separate multiple statements with a semicolon (;). You could have, alternatively, written everything up until "return true" as a function and then calling it:function writestatus(){status="Do not click here, its empty!" }andthen:onmouseover="writestatus();returntrue"

- So you're thinking, "what is return true?" Good question. You need to add this line of code to set the status property with the mouseover effect. Uh? I know, don't worry so much now, it really isn't important. Just remember you need this to "activate" the status onmouseover effect.
- onmouseout="status=' '" clears the status after the mouse leaves the link. Whenever the mouse moves away from the link, the status bar is "reset" again. If you don't insert this code, the status bar will still have those words you entered into it even after taking away the cursor.
- Whenever we have nested quotations, the inner ones are always singled. Ie: onclick="alert('hello')

onUnload event handler

onunload executes JavaScript *immediately after* someone leaves the page. A common use (though not that great) is to thank someone as that person leaves your page for coming and visiting.

```
<body onunload="alert('Thank you. Please come back to this site and visit us soon, ok?')">
```

There are other event handlers, many belonging to forms. These event handlers are discussed in the tutorialAccessing and validating forms using Javascript.

Here is a list of commonly used event handlers in JavaScript:

Event Handlers	Applicable inside:

onAbort	<img> tags
onBlur	window object, all form objects (ie: <input>), and <frame>.
onClick	Most visible elements such as <a>, , <body> etc.
onChange	Use this to invoke JavaScript if the mouse goes pass some link
onError	Text fields, textareas, and select lists.
onFocus	Most visible elements such as <a>, , <body> etc.
onLoad	<body>, <img>, and <frame>
onMouseover	Most visible elements such as <a>, , <body> etc.
onMouseout	Most visible elements such as <a>, , <body> etc.
onReset	<form> tag, triggered when the form is reset via <input type="reset">.
onSelect	Elements with textual content. Most commonly used inside text fields and textareas.
onSubmit	<form> tag, triggered when the form is submitted.
onUnload	<body>

Possible Questions:

Part B (2 Marks)

1) Define object with example
2) Define Arraylist with example
3) Define Array with example
4) How to define function in JavaScript?
5) What is mean by operator?
6) What is mean by event handling?
7) Define datatype with example
8) What is mean by control structure?
9) How to declare function?
10) List out what are the operators in javascript

Part C (6 Marks)

1) What is mean by array,arraylist?Explain similarities between array and arraylist?
2) Difference between Array and ArrayList
3) Explain in Detail about datatype with example
4) Explain in detail about Operators with examples
5) Explain in detail about control structure with example
6) Explain in detail about Event handling
7) What is mean by event, explain in detail about to handling java scripts event?
8) Explain in detail about Object, how to declare, Instantiating and initializing an object
9) Explain in detail about function with example
10) Explain in detail about object

KARPAGAM ACADEMY OF HIGHER EDUCATION

PART - A (ONLINE EXAMINATION)

MULTIPLE CHOICE QUESTIONS (Each question carries one mark)

SUBJECT: INTERNET TECHNOLOGY **SUB CODE: 17ITU404A** **Class: II B.Sc IT**

Unit - I

QUESTION	OPTION 1	OPTION 2	OPTION 3	OPTION4	ANSWER
Java programming language was originally developed by_____	IBM	Sun microsystem	Oracle	Microsoft	Sun microsystem
Java makes an effort to eliminate error prone situation by emphasizing mainly on compile time error checking and runtime checking	Interpreted	distributed	multithread	robust	robust
Java is guaranteed to be	write anywhere run once	write once run once	write once run anywhere	none of these	write anywhere run once
James Gosling initiated java language in	Jul-91	Jun-91	Jul-95	Jun-98	Jun-91
_____keyword used to create a instance of object	auto	new	this	static	new
constructor that takes low argument is known as _____	copy constructor	parameterized constructor	default constructor	destructor	default constructor
which of the option does not support array list	object	entries	primitive	non primitive	primitive
Defined in java language itself as a_____data structure	fundamental	non fundamental	primitive	non primitive	fundamental
Java script is the _____type language means you don't nead to specify type of variable	static	auto	register	dynamic	dynamic
_____can store heterogenous data type	array	arraylist	linked list	data link	array
Java programs can carry extensive amount of ___time information		compile	execution	none of these	execution
which compiler enables high performance	java compiler	java virtual machine	just in out	just in time	just in time
The new keyword is followed by a call to a constructor this call_____new object	declaration	initialiation	instantiation	creating	instantiation
An array is the basic functionality provided by _____	c++	java script	java	vb.net	java

_____represents instance through which can access member	class	object	array	union	object
Unintialize variable equal to _____	string	character	null	integer	null
Which is not a nonprimitive datatype are as follows	object	class	array	RegExp	class
In java script null document.writeln(str==undefined)it returns	FALSE	null	TRUE	empty	FALSE
Java script empty variable Boolean contest return_____	FALSE	TRUE	null	char	null
Java script as only one _____data type	decimal	float	numeric	string	numeric
To instantiate a data object, or any other object use the ____operator.	exist	new	free	create	new
New requires a single argument _____method for the object to be created.	constructor	destructor	argument	defaultconstructo	argument
Declaration do not _____object	initialization	instantiate	creating	generalization	initialization
In array we insert elements using _____operators	conditional	logical	assignment	relational	assignment
Array list is the part of _____frame work in java	array	list	collection	none of these	collection
Arraylist has set of _____to access and modify them _____	function	class	object	methods	methods
Java is designed for the _____environment of the internet	dynamic	distributed	add	replace	distributed
We can insert element into the arraylist object using _____method	delete	insert	add	replace	add
Length of arraylist is provided by _____method	add	size	create	update	size
There are_____types of primitive datatype in javascript	3	5	4	2	5
Javascript Boolean type can have_____value	2	1	0	3	2
_____initiated java language project	lampsalt	dennis Ritchie	james gosling	bjarne stroustrup	james gosling
Javascript symbol datatype it is used for _____unique object property.	variable	identifier	datatype	none of the above	identifier
Sun relased much of java as free and ____software under the terms of GNU.	open source	code free	JVM	None	open source

_____Java byte code is translated on the fly to native machine instruction and is not stored anywhere	dynamic	interpreted	distrubuted	robust	interpreted
GPL means	General Protected License	General Private License	General Public License	General Protocol License	General Public License
Date class supports the constructor that requires ____ integer	2	3	4	1	3
There are ____ ways to creating objects	3	4	2	5	3
javascript includes similar to other programming languages like ___	Java	C#	both a and b	C++	C#
In java script they are ____types of data types	2	3	5	6	2
Sun released the first public implementation as Java1.0 in_____	1998	1993	1995	1997	1995
In array how many types of class____	1	2	3	0	1
In arraylist the elements can _____ size	Growable	sizable	both a and b	no size	both a and b
In array elements retrieved with _____loop	for	while	dowhile	none	for
Arraylist can be retrieved with loop and ____	operators	iterators	sizable	all of the above	iterators
Arraylist can be obtained a ____version	Synchronized	Asynchronized	both a and b	none of the above	Synchronized
Array and Arraylist are similar for the ___ and ___operations	get,add	get,insert	insert,del	add,get	get,add
_____ indicates the characteristics of data	variable	identifier	datatype	keyword	datatype
Java can be easily extended since it is based on _____model	object	system	dynamic	analysis	object
Compiler in java is written in _____	ASCII	ANSIC	C#	none.	ANSIC

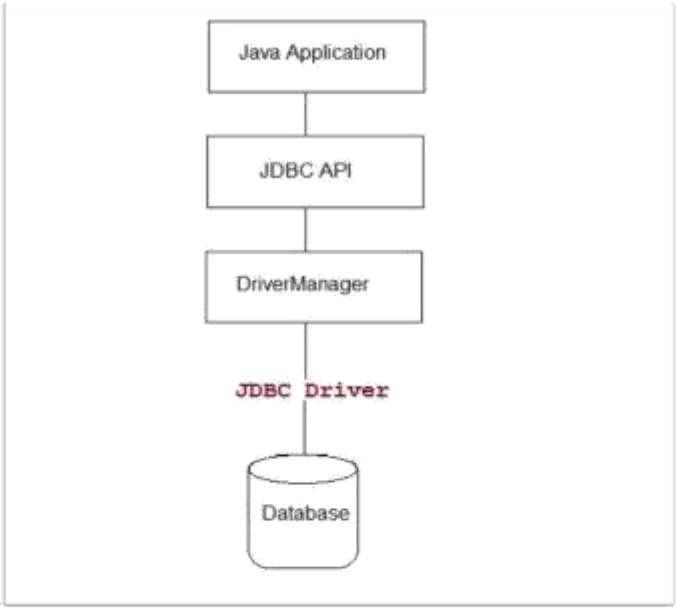
UNIT-II

SYLLABUS

JDBC: JDBC fundamentals, Establishing Connectivity and working with connection interface, working with statements, creating and executing SQL statements, working with ResultSet objects.

INTRODUCTION TO JDBC:

Java Database Connectivity(JDBC) is an **Application Programming Interface(API)** used to connect Java application with Database. JDBC is used to interact with various type of Database such as Oracle, MS Access, My SQL and SQL Server. JDBC can also be defined as the platform-independent interface between a relational database and Java programming. It allows java program to execute SQL statement and retrieve result from database.



JDBC FUNDAMENTALS:

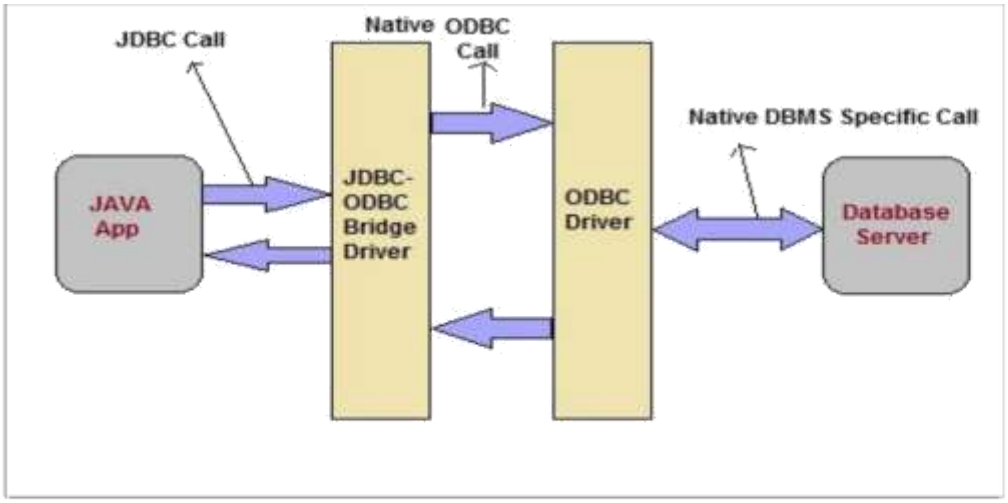
JDBC Driver

JDBC Driver is required to process SQL requests and generate result. The following are the different types of driver available in JDBC.

- **Type-1 Driver** or **JDBC-ODBC bridge**
- **Type-2 Driver** or **Native API Partly Java Driver**
- **Type-3 Driver** or **Network Protocol Driver**
- **Type-4 Driver** or **Thin Driver**

JDBC-ODBC Bridge

Type-1 Driver act as a bridge between JDBC and other database connectivity mechanism(ODBC). This driver converts JDBC calls into ODBC calls and redirects the request to the ODBC driver.



Advantage

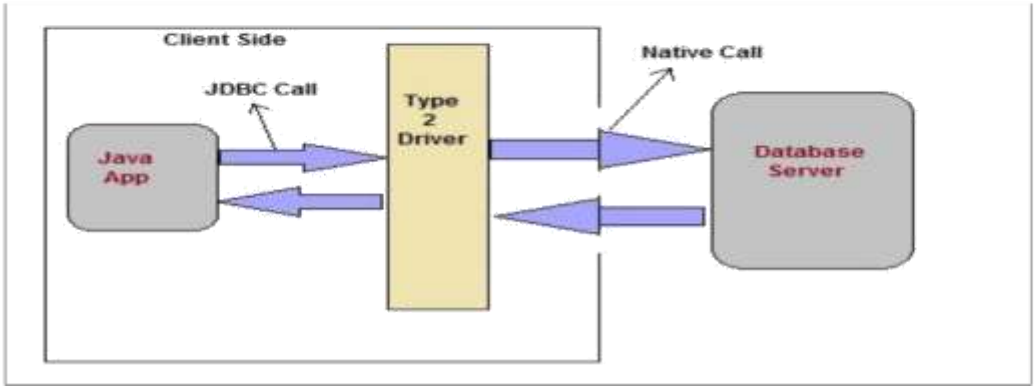
- Easy to use
- Allow easy connectivity to all database supported by the ODBC Driver.

Disadvantage

- Slow execution time
- Dependent on ODBC Driver.
- Uses Java Native Interface(JNI) to make ODBC call.

Native API Driver

This type of driver make use of Java Native Interface(JNI) call on database specific native client API. These native client API are usually written in C and C++.



Advantage

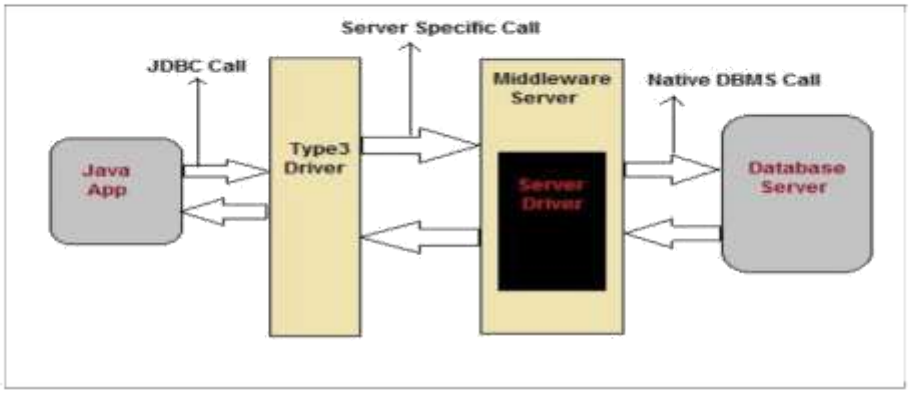
- faster as compared to **Type-1 Driver**
- Contains additional features.

Disadvantage

- Requires native library
- Increased cost of Application

Network Protocol Driver

This driver translate the JDBC calls into a database server independent and Middleware server-specific calls. Middleware server further translate JDBC calls into database specific calls.



Advantage

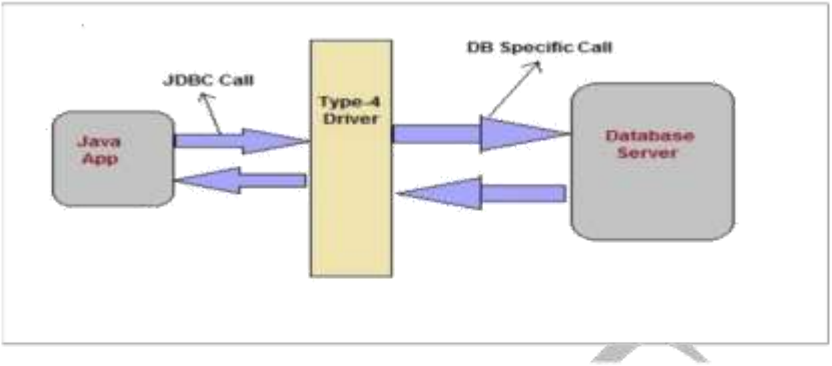
- Does not require any native library to be installed.
- Database Independency.
- Provide facility to switch over from one database to another database.

Disadvantage

- Slow due to increase number of network call.

Thin Driver

This is Driver called Pure Java Driver because. These drivers interact directly with database. It does not require any native database library, that is why it is also known as Thin Driver.



Advantage

- Does not require any native library.
- Does not require any Middleware server.
- Better Performance than other driver.

Disadvantage

- Slow due to increase number of network call.

JDBC 4.0 API

JDBC 4.0 API is mainly divided into two package

1. java.sql
2. javax.sql

java.sql package

This package include classes and interface to perform almost all JDBC operation such as creating and executing SQL Queries.

Important classes and interface of java.sql package

classes/interface	Description
java.sql.BLOB	Provide support for BLOB(Binary Large Object) SQL type.
java.sql.Connection	creates a connection with specific database
java.sql.CallableStatement	Execute stored procedures

java.sql.CLOB	Provide support for CLOB(Character Large Object) SQL type.
java.sql.Date	Provide support for Date SQL type.
java.sql.Driver	create an instance of a driver with the DriverManager.
java.sql.DriverManager	This class manages database drivers.
java.sql.PreparedStatement	Used to create and execute parameterized query.
java.sql.ResultSet	It is an interface that provide methods to access the result row-by-row.
java.sql.Savepoint	Specify savepoint in transaction.
java.sql.SQLException	Encapsulate all JDBC related exception.
java.sql.Statement	This interface is used to execute SQL statements.

javax.sql package

This package is also known as JDBC extension API. It provides classes and interface to access server-side data.

Important classes and interface of javax.sql package

classes/interface	**Description**
javax.sql.ConnectionEvent	Provide information about occurence of event.
javax.sql.ConnectionEventListener	Used to register event generated by **PooledConnection**object.

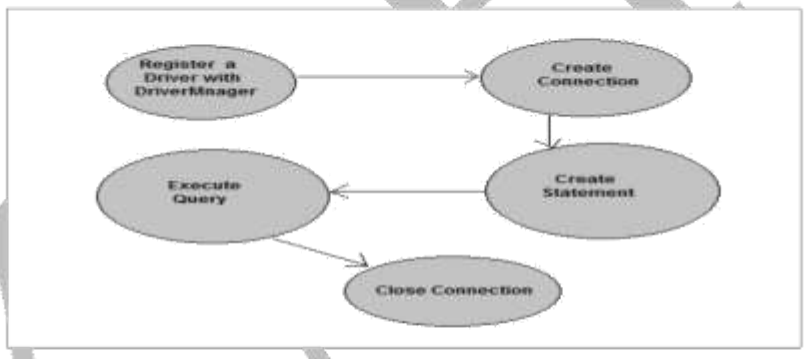
javax.sql.DataSource	Represent the **DataSource** interface used in an application.
javax.sql.PooledConnection	provide object to manage connection pools.

ESTABLISHING CONNECTIVITY AND WORKING WITH CONNECTION INTERFACE:

Steps to connect a Java Application to Database:

The following 5 steps are the basic steps involve in connecting a Java application with Database using JDBC.

1. Register the Driver
2. Create a Connection
3. Create SQL Statement
4. Execute SQL Statement
5. Closing the connection



Register the Driver

`Class.forName()` is used to load the driver class explicitly.

Example to register with JDBC-ODBC Driver

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

Create a Connection

`getConnection()` method of **DriverManager** class is used to create a connection.

Syntax

```
getConnection(String url)
getConnection(String url, String username, String password)
getConnection(String url, Properties info)
```

Example establish connection with Oracle Driver

```
Connection con = DriverManager.getConnection
                ("jdbc:oracle:thin:@localhost:1521:XE","username","password");
```

CREATING AND EXECUTING SQL STATEMENTS:

Create SQL Statement

`createStatement()` method is invoked on current **Connection** object to create a SQL Statement.

Syntax

```
public Statement createStatement() throws SQLException
```

Example to create a SQL statement

```
Statement s=con.createStatement();
```

Execute SQL Statement

`executeQuery()` method of **Statement** interface is used to execute SQL statements.

Syntax

```
public ResultSet executeQuery(String query) throws SQLException
```

Example to execute a SQL statement

```
ResultSet rs=s.executeQuery("select * from user");
 while(rs.next())
 {
  System.out.println(rs.getString(1)+" "+rs.getString(2));
 }
```

Closing the connection

After executing SQL statement you need to close the connection and release the session. The `close()` method of **Connection** interface is used to close the connection.

Syntax

```
public void close() throws SQLException
```

Example of closing a connection

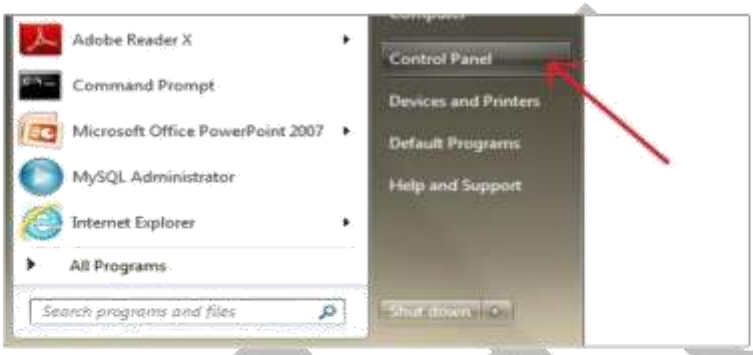
```
con.close();
```

Connecting to Access Database using Type-1 Driver

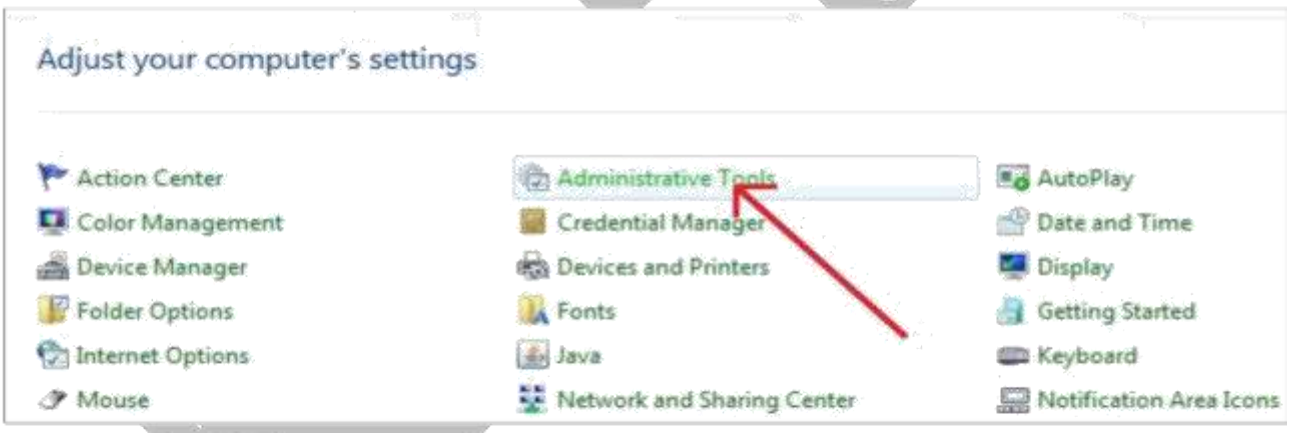
To connect a Java application with Access database using JDBC-ODBC Bridge(type-1) Driver. You need to follow the following steps

Create DSN Name

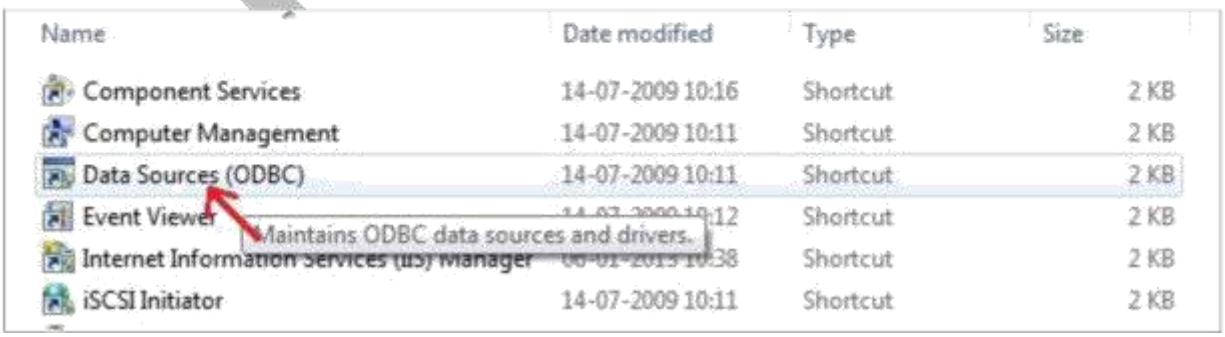
1. **Go to control panel**



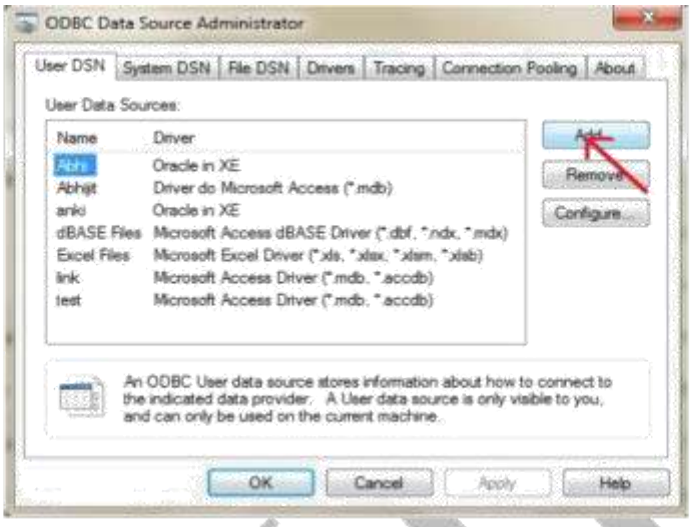
2. **Go to Administrative tools**



3. **Select Data Source(ODBC)**



4. **Add new DSN name, select add**



5. **Select Access driver from the list, click on finish**

6. **Give a DSN name, click ok**

WORKING WITH STATEMENTS:

There are 3 types of Statements, as given below:

Statement:

It can be used for general-purpose access to the database. It is useful when you are using static SQL statements at runtime.

Before you can use a Statement object to execute a SQL statement, you need to create one using the Connection object's createStatement( ) method, as in the following example −

```
Statement stmt = null;
try {
   stmt = conn.createStatement( );
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   . . .
}
```

Once you've created a Statement object, you can then use it to execute an SQL statement with one of its three execute methods.

- **boolean execute (String SQL)**: Returns a boolean value of true if a ResultSet object can be retrieved; otherwise, it returns false. Use this method to execute SQL DDL statements or when you need to use truly dynamic SQL.
- **int executeUpdate (String SQL)**: Returns the number of rows affected by the execution of the SQL statement. Use this method to execute SQL statements for which you expect to get a number of rows affected - for example, an INSERT, UPDATE, or DELETE statement.
- **ResultSet executeQuery (String SQL)**: Returns a ResultSet object. Use this method when you expect to get a result set, as you would with a SELECT statement.

PreparedStatement:

It can be used when you plan to use the same SQL statement many times. The PreparedStatement interface accepts input parameters at runtime.

The PreparedStatement interface extends the Statement interface, which gives you added functionality with a couple of advantages over a generic Statement object.

This statement gives you the flexibility of supplying arguments dynamically.

Creating PreparedStatement Object

```
PreparedStatement pstmt = null;
try {
   String SQL = "Update Employees SET age = ? WHERE id = ?";
   pstmt = conn.prepareStatement(SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   . . .
}
```

All parameters in JDBC are represented by the **?** symbol, which is known as the parameter marker. You must supply values for every parameter before executing the SQL statement.

Closing PreparedStatement Object

Just as you close a Statement object, for the same reason you should also close the PreparedStatement object.

A simple call to the close() method will do the job. If you close the Connection object first, it will close the PreparedStatement object as well. However, you should always explicitly close the PreparedStatement object to ensure proper cleanup.

```
PreparedStatement pstmt = null;
try {
```

```
  String SQL = "Update Employees SET age = ? WHERE id =

  ?"; pstmt = conn.prepareStatement(SQL); . . .


}
catch (SQLException e) {
  . . .
}
finally {
  pstmt.close();
}
```

CallableStatement:

CallableStatement can be used when you want to access database stored procedures.

Creating CallableStatement Object

Suppose, you need to execute the following Oracle stored procedure −

```
CREATE OR REPLACE PROCEDURE getEmpName
  (EMP_ID IN NUMBER, EMP_FIRST OUT VARCHAR) AS
BEGIN
  SELECT first INTO EMP_FIRST
  FROM Employees
  WHERE ID = EMP_ID;
END;
```

NOTE: Above stored procedure has been written for Oracle, but we are working with MySQL database so, let us write same stored procedure for MySQL as follows to create it in EMP database −

```
DELIMITER $$
```

```
DROP PROCEDURE IF EXISTS `EMP`.`getEmpName` $$

CREATE PROCEDURE `EMP`.`getEmpName`

  (IN EMP_ID INT, OUT EMP_FIRST VARCHAR(255))

BEGIN

  SELECT first INTO EMP_FIRST

  FROM Employees

  WHERE ID = EMP_ID;

END $$


DELIMITER ;
```

Three types of parameters exist: IN, OUT, and INOUT. The PreparedStatement object only uses the IN parameter. The CallableStatement object can use all the three.

Here are the definitions of each −

Parameter	Description
IN	A parameter whose value is unknown when the SQL statement is created. You bind values to IN parameters with the setXXX() methods.
OUT	A parameter whose value is supplied by the SQL statement it returns. You retrieve values from theOUT parameters with the getXXX() methods.
INOUT	A parameter that provides both input and output values. You bind variables with the setXXX() methods and retrieve values with the getXXX() methods.

The following code snippet shows how to employ the **Connection.prepareCall()** method to instantiate a **CallableStatement**object based on the preceding stored procedure −

```
CallableStatement cstmt = null;

try {
```

```
  String SQL = "{call getEmpName (?, ?)}";
  cstmt = conn.prepareCall (SQL);
  . . .
}
catch (SQLException e) {
  . . .
}
finally {
  . . .
}
```

The String variable SQL, represents the stored procedure, with parameter placeholders.

Using the CallableStatement objects is much like using the PreparedStatement objects. You must bind values to all the parameters before executing the statement, or you will receive an SQLException.

If you have IN parameters, just follow the same rules and techniques that apply to a PreparedStatement object; use the setXXX() method that corresponds to the Java data type you are binding.

When you use OUT and INOUT parameters you must employ an additional CallableStatement method, registerOutParameter(). The registerOutParameter() method binds the JDBC data type, to the data type that the stored procedure is expected to return.

Once you call your stored procedure, you retrieve the value from the OUT parameter with the appropriate getXXX() method. This method casts the retrieved value of SQL type to a Java data type.

Closing CallableStatement Object

Just as you close other Statement object, for the same reason you should also close the CallableStatement object.

A simple call to the close() method will do the job. If you close the Connection object first, it will close the CallableStatement object as well. However, you should always explicitly close the CallableStatement object to ensure proper cleanup.

```
CallableStatement cstmt = null;
try {
  String SQL = "{call getEmpName (?, ?)}";
```

```
  cstmt = conn.prepareCall (SQL);
  . . .
}
catch (SQLException e) {
  . . .
}
finally {
  cstmt.close();
}
```

For a better understanding, I would suggest to study Callable - Example Code

Closing Statement Object

Just as you close a Connection object to save database resources, for the same reason you should also close the Statement object.

A simple call to the close() method will do the job. If you close the Connection object first, it will close the Statement object as well. However, you should always explicitly close the Statement object to ensure proper cleanup.

```
Statement stmt = null;
try {
  stmt = conn.createStatement( );
  . . .
}
catch (SQLException e) {
  . . .
}
finally {
  stmt.close();
}
```

For a better understanding, we suggest you to study the Statement - Example tutorial

WORKING WITH RESULTSET OBJECTS.

A ResultSet object maintains a cursor that points to the current row in the result set. The term "result set" refers to the row and column data contained in a ResultSet object.

The methods of the ResultSet interface can be broken down into three categories −

- **Navigational methods:** Used to move the cursor around.
- **Get methods:** Used to view the data in the columns of the current row being pointed by the cursor.
- **Update methods:** Used to update the data in the columns of the current row. The updates can then be updated in the underlying database as well.

The cursor is movable based on the properties of the ResultSet. These properties are designated when the corresponding Statement that generates the ResultSet is created.

JDBC provides the following connection methods to create statements with desired ResultSet −

- **createStatement(int RSType, int RSConcurrency);**
- **prepareStatement(String SQL, int RSType, int RSConcurrency);**
- **prepareCall(String sql, int RSType, int RSConcurrency);**

The first argument indicates the type of a ResultSet object and the second argument is one of two ResultSet constants for specifying whether a result set is read-only or updatable.

Type of ResultSet

The possible RSType are given below. If you do not specify any ResultSet type, you will automatically get one that is TYPE_FORWARD_ONLY.

Type	Description
ResultSet.TYPE_FORWARD_ONLY	The cursor can only move forward in the result set.
ResultSet.TYPE_SCROLL_INSENSITIVE	The cursor can scroll forward and backward, 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 forward and backward, and the result set is sensitive to changes made by

	others to the database that occur after the result set was created.

Concurrency of ResultSet

The possible RSConcurrency are given below. If you do not specify any Concurrency type, you will automatically get one that is CONCUR_READ_ONLY.

Concurrency	**Description**
ResultSet.CONCUR_READ_ONLY	Creates a read-only result set. This is the default
ResultSet.CONCUR_UPDATABLE	Creates an updateable result set.

All our examples written so far can be written as follows, which initializes a Statement object to create a forward-only, read only ResultSet object −

```
try {
   Statement stmt = conn.createStatement(
                        ResultSet.TYPE_FORWARD_ONLY,
                        ResultSet.CONCUR_READ_ONLY);
}
catch(Exception ex) {
   ....
}
finally {
   ....
}
```

Navigating a Result Set

There are several methods in the ResultSet interface that involve moving the cursor, including −

S.N.	Methods & Description
1	**public void beforeFirst() throws SQLException** Moves the cursor just before the first row.
2	**public void afterLast() throws SQLException** Moves the cursor just after the last row.
3	**public boolean first() throws SQLException** Moves the cursor to the first row.
4	**public void last() throws SQLException** Moves the cursor to the last row.
5	**public boolean absolute(int row) throws** **SQLException** Moves the cursor to the specified row.
6	**public boolean relative(int row) throws SQLException** Moves the cursor the given number of rows forward or backward, from where it is currently pointing.
7	**public boolean previous() throws SQLException** Moves the cursor to the previous row. This method returns false if the previous row is off the result set.
8	**public boolean next() throws SQLException** Moves the cursor to the next row. This method returns false if there are no more rows in the result set.
9	**public int getRow() throws SQLException** Returns the row number that the cursor is pointing to.
10	**public void moveToInsertRow() throws SQLException**

	Moves the cursor to a special row in the result set that can be used to insert a new row into the database. The current cursor location is remembered.
11	**public void moveToCurrentRow() throws SQLException** Moves the cursor back to the current row if the cursor is currently at the insert row; otherwise, this method does nothing

For a better understanding, let us study Navigate - Example Code.

Viewing a Result Set

The ResultSet interface contains dozens of methods for getting the data of the current row.

There is a get method for each of the possible data types, and each get method has two versions −

- One that takes in a column name.
- One that takes in a column index.

For example, if the column you are interested in viewing contains an int, you need to use one of the getInt() methods of ResultSet −

S.N.	Methods & Description
1	**public int getInt(String columnName) throws SQLException** Returns the int in the current row in the column named columnName.
2	**public int getInt(int columnIndex) throws SQLException** Returns the int in the current row in the specified column index. The column index starts at 1, meaning the first column of a row is 1, the second column of a row is 2, and so on.

Similarly, there are get methods in the ResultSet interface for each of the eight Java primitive types, as well as common types such as java.lang.String, java.lang.Object, and java.net.URL.

There are also methods for getting SQL data types java.sql.Date, java.sql.Time, java.sql.TimeStamp, java.sql.Clob, and java.sql.Blob. Check the documentation for more information about using these SQL data types.

For a better understanding, let us study Viewing - Example Code.

Updating a Result Set

The ResultSet interface contains a collection of update methods for updating the data of a result set.

As with the get methods, there are two update methods for each data type −

- One that takes in a column name.
- One that takes in a column index.

For example, to update a String column of the current row of a result set, you would use one of the following updateString() methods −

S.N.	Methods & Description
1	**public void updateString(int columnIndex, String s) throws** **SQLException** Changes the String in the specified column to the value of s.
2	**public void updateString(String columnName, String s) throws SQLException** Similar to the previous method, except that the column is specified by its name instead of its index.

There are update methods for the eight primitive data types, as well as String, Object, URL, and the SQL data types in the java.sql package.

S.N.	Methods & Description
1	**public void updateRow()** Updates the current row by updating the corresponding row in the database.
2	**public void deleteRow()** Deletes the current row from the database
3	**public void refreshRow()** Refreshes the data in the result set to reflect any recent changes in the database.
4	**public void cancelRowUpdates()** Cancels any updates made on the current row.
5	**public void insertRow()** Inserts a row into the database. This method can only be invoked when the cursor is pointing to the insert row.

Possible Question

Part B (2 Marks)

1) What is Network Protocol Driver?
2) Write syntax for create and execute SQL Statement?
3) What is the use of SQL STATEMENT?
4) What is mean by interface?
5) How to establish connectivity?
6) What is mean by Resultset object
7) What is mean by JDBC
8) What are the types of drivers in JDBC
9) Define Statement
10) How Resultset object works in JDBC

Part C (6 Marks)

1) How to working with connection interface in JDBC?
2) Explain in details about Establishing connectivity in JDBC
3) Explain in detail about JDBC Fundamentals
4) What is mean by JDBC? Explain types of drivers in JDBC with neat diagram
5) Explain In Detail about working with statement
6) How to working with SQL statement in JDBC
7) Explain how to create and execute SQL statement with example
8) Explain in detail about result set object with example

KARPAGAM ACADEMY OF HIGHER EDUCATION

PART - A (ONLINE EXAMINATION)

MULTIPLE CHOICE QUESTIONS (Each question carries one mark)

SUBJECT: INTERNET TECHNOLOGY

Sub.code : 17ITU404A CLASS: II B.Sc IT

Unit II

QUESTIONS	OPTION 1	OPTION2	OPTION3	OPTION4	ANSWER
What is the expansion of JDBC_____	Java Database connectivity	java data connectivity	java database control	java database connection	Java Database connectivity
JDBC belongs to which interface_____	application programming interface	plat form independent interface	java national interface	None	application programming interface
What is the Type-2 driver in JDBC driver____	JDBC-ODBC bridge	Thin driver	network protocol driver	Native API partly java driver	Native API partly java driver
Which is used to make ODBC cal ____	Java network	java national	java interface	java native interface	java native interface
Native client API are usually written in _____ and_____	oracle	c & C++	C++ & java	Networking	c & C++
Which driver translates the JDBC calls into a database server independent _____	National protocol	Network protocol	JDBC driver	Networking	Network protocol
Which driver as database independency _____	ODBC driver	Network protocol	Thin driver	JDBC driver	Network protocol
Thin driver is also called _____	Thin driver	java driver	driver	driver	driver
Which deriver does not require any middle ware server_____	ODBC driver	none of the above	Thin driver	Network protocol	Thin driver
JDBC and OAPI is mainly divided into _____ packages	2	4	8	1	2
Which class/interface executes stored procedures _____	Java.sql.callable statement	java.sql.prepared statement	java.sql.save point	all the above	Java.sql.callable statement
Which class/interface provide support for date sql type_____	er	OB	e	above	e
Which classes/interface manages database drivers____	java.sql.sqlException	java.sql.connection	java.sql.driver	java.sql.Date	java.sql.Date

Which class/interface encapsulate all JDBC related exception ____	java.sql.sql Exception	java.sql.Res ultbet	java.sql.sav epoint	both a and b	java.sql.sql Exception
Which classes/interface represent the data source interface used in an application_____	java.sql.stat ement	javax.sql.dat asource	java.sql.data source	none of the above	javax.sql.dat asource
_____method of driver manager class is used to create a connect on	set connection()	gets connection()	get connection()	None of the above	get connection()
All parameters in JDBC are represented by the _____	;	!=	:	**?**	?
? symbol in JDBC is known as _____	parameter A	parameter K	parameter X	parameter Y	parameter K
callable statement can be used when you want to access _____ procedures	Database stored procedure	Database stored process	Database stored protocol	Database procedure	Database stored procedure
There are ____ types of parameters.	6	3	2	5	3
A parameter whose value is unknown when the sql statement is created___	OUT	INOUT	IN	above	IN
A parameter whose value is supplied by the sql statement it returns ____parameters	INOUT	OUT	IN	above	OUT
INOUT parameter that provides both ____ & _____	INOUT	OUT	IN	these	INOUT
paind variable use ____ method.	get XXX ()	set XXX ()	get	set	set XXX ()
_____ method is used to move the cursor around.	set methods	get methods	update methods	navigational methods	navigational methods
A_____ object maintains a cursor that points to the current row in the result set.	savepoint	result set	Driver manager	sql exception	result set
_____ methods is used to view the data in the columns of the current row.	get methods	update methods	navigational methods	set methods	get methods
_____ method is used to update the data in the coloumns of the current row update me	get methods	methods	set methods	methods	methods
The cursor is movable based on the properties of_____ .	manager	savepoint	result set	driver	result set
The cursor can only move forward in the result set .	Result set.Type_Ba ckward_onl	Result set.Type_fo rward_only;	Result set.Type_Re ad_only;	Result set.Type_W rite_only;	Result set.Type_fo rward_only;
which type of result set is used to scroll the cursor forward and backward and the result s	Type_Forw ard only	Type_scroll _Inserting	Type_scroll _backward only	Type_scroll _sensitive	Type_scroll _sensitive
Which type of result set is not sensitive.	Type_Forw ard only	Type_scroll _Insensitive	Type_Back ward only	Type_scroll _sensitive	Type_scroll _Insensitive
_____creates read only result set.	Result set.concur_u pdatable	Result set. Concur_writ ing_only	Result set.concur_ Getable	Result set.concur_ Read_only	Result set.concur_ Read_only
_____ creates updateable result set.	Resultset.co ncur_read only	Resultset.co ncur_getabl e	Resultset.co ncur_updata ble	Resultset.co ncur _write only	Resultset.co ncur_updata ble

There are several methods in the resultset interface that involve _____.	Moving the cursor	create the cursor	change the cursor	all the above	Moving the cursor
_____ Result set interface moves the cursor to the last row.	public void afterlast()	public void last()	public void before	both b & c	public void last()
When Javascript function invokes_____	when an event occurs	when a statement return	when a function define	none of these	when an event occurs
_____ returns the row number that the cursor is pointing to.	public int getRow()	public void moveto insert row()	public void moveto currentrow()	public void getRow()	public int getRow()
The resultset interface contains dozens of methods for getting the data of the _____.	set row	insert row	current row	getrow	current row
Each get method has _____versions	3	1	2	4	2
The resultset interface contains a collection of ____ for updating the data of a result set.	methods	methods	get methods	these	methods
there are _____ update methods for each datatype.	3	2	5	6	2
Among the following primitive datatypes which are used for update methods _____ str	int	char	string	these	string
There are update methods for the ____ primitive datatypes	4	2	6	8	8
Updating a row in the result set changes _____of the current row in the result set objec	insert row	current row	row	column	column
Updates the current row by updating the corresponding row in the database.	public void deleterow()	public void insertrow()	public void currentrow()	public void updaterow()	public void updaterow()
_____ deletes the current roe from the database.	public void deleterow()	public void updaterow()	public void delete()	public void insertrow()	public void deleterow()
public void cancel row updates() cancels any updates made on the _____.	insert row()	row()	row()	delete row()	row()
_____ inserts a row into the database.	public void deleterow()	public void insertrow()	public void currentrow()	public void updaterow()	public void insertrow()
JDBC is used to interact with various type of _____	database connectivity	data analysis	database	data control	database
_____ allows java program to execute sql statement and retrieve from database.	driver	driver	JDBC	ODBC	JDBC
_____ is required to process sql requests and generate result.	driver	driver	ODBC	JDBC	driver
_____ act as a bridge between JDBC and other database connectivity mechanism.	JDBC	ODBC	driver	both a and b	ODBC
Type 7 driver contains JDBC calls into ODBC calls and redirects the request to the ____	JDBC	ODBC	driver	these	ODBC
Native API driver is faster when compared with _____.	TYPE-3 driver	TYPE-4 driver	TYPE-2 driver	TYPE-1 driver	TYPE-1 driver
Which classes/ interface creates a connection with specific database.	Java. sql. BLOB	Java. sql. Connection	Java. sql.driver	Java. sql. Driver	Java. sql. BLOB
Which classes/ interface create an instance of a driver with a driver manager.	Java. sql. Callable statement	Java. sql. Driver	Java. sql.connection	Java. sql. BLOB	Java. sql.connection

Which classes/ interface create an instance of a driver with a driver manager.	Java. sql. Resultset	Java. sql. Driver	Java. sql. CLOB	none of these	Java. sql. Driver
Which classes/ interface used to create and execute parameterized query.	Java. sql. Savepoint	Java. sql.driver manager	Java. sql.driver manager	Java. sql.prepared statement	Java. sql.prepared statement
Which classes/ interface specifies savepoint in transaction.	Java. sql. Resultset	Java. sql.connecti	Java. sql. Savepoint	None of the above	Java. sql. Savepoint

UNIT-III

SYLLABUS

JSP: Introduction to JavaServerPages, HTTP and Servlet Basics, The Problem with Servlets, The Anatomy of a JSP Page, JSP Processing, JSP Application Design with MVC, Setting Up the JSP Environment.

JSP: Introduction to JavaServerPages

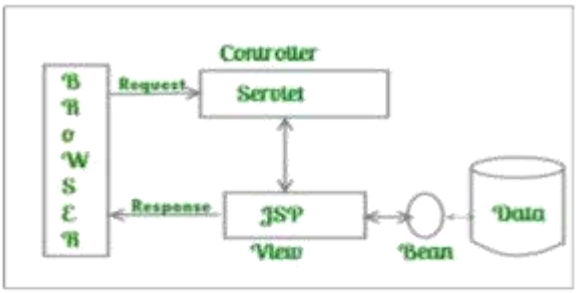
Introducing JavaServer Pages The Java 2 Enterprise Edition (J2EE) has taken the once-chaotic task of building an Internet presence and transformed it to the point where developers can use Java to efficiently create multitier, server-side applications. Today, the Java Enterprise APIs have expanded to encompass a number of areas: RMI and CORBA for remote object handling, JDBC for database interaction, JNDI for accessing naming and directory services, Enterprise JavaBeans for creating reusable business components, JMS (Java Messaging Service) for message-oriented middleware, and JTA (Java Transaction API) for performing atomic transactions. In addition, J2EE supports servlets , an extremely popular Java substitute for CGI scripts.

The combination of these technologies allows programmers to create distributed business solutions for a variety of tasks. In late 1999, Sun Microsystems added a new element to the collection of Enterprise Java tools: JavaServer Pages (JSP). JavaServer Pages are built on top of Java servlets and are designed to increase the efficiency in which programmers, and even nonprogrammers, can create web content. This book is all about JavaServer Pages.

What Is JavaServer Pages?

JavaServer Pages is a technology for developing web pages that include dynamic content. Unlike a plain HTML page, which contains static content that always remains the same, a JSP page can change its content based on any number of variable items, including the identity of the user, the user's browser type, information provided by the user, and selections made by the user.

Functionality such as this can be used to create web applications like shopping carts and employee directories. A JSP page contains standard markup language elements, such as HTML tags, just like a regular web page. However, a JSP page also contains special JSP elements that allow the server to insert dynamic content in the page. JSP elements can be used for a wide variety of purposes, such as retrieving information from a database or registering user preferences. When a user asks for a JSP page, the server executes the JSP elements, merges the results with the static parts of the page, and sends the dynamically composed page back to the browser



Why Use JSP?

I n the early days of the Web, the Common Gateway Interface (CGI) was the only tool for developing dynamic web content. However, CGI is not an efficient solution. For every request that comes in, the web server has to create a new operating system process, load an interpreter and a script, execute the script, and then tear it all down again. This is very taxing for the server and doesn't scale well when the amount of traffic increases. Numerous CGI alternatives and enhancements, such as FastCGI, mod_ perl from Apache, NSAPI from Netscape, ISAPI from Microsoft, and Java Servlets from Sun Microsystems, have been created over the years. While these solutions offer better performance and scalability, all of these technologies suffer from a common problem: they generate web pages by embedding HTML directly in programming language code. This pushes the creation of dynamic web pages exclusively into the realm of programmers. JavaServer Pages, however, changes all that.

What You Need to Get Started Before

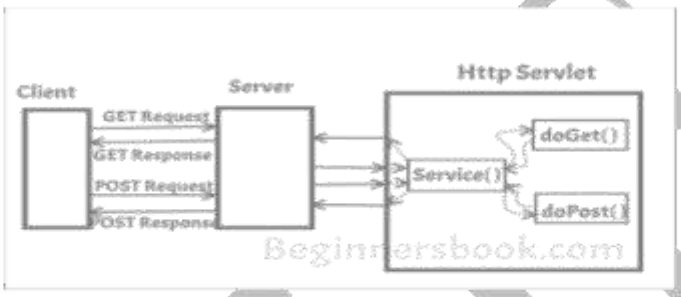
We begin, let's quickly look at what you need to run the examples and develop your own applications. You really need only three things: · A PC or workstation with a connection to the Internet, so you can download the software you need · A Java 2-compatible Java Software Development Kit (Java 2 SDK) · A JSP 1.1-enabled web server, such as Apache Tomcat from the Jakarta Project The Apache Tomcat server is the reference implementation for JSP 1.1.

All the examples in this book were tested on Tomcat. In Chapter 4, I'll show you how to download, install, and configure the Tomcat server, as well as all the examples from this book. In addition, there are a wide variety of other tools and servers that support JSP, from both open source projects and commercial companies. Close to 30 different server products support JSP to date, and roughly 10 authoring tools with varying degrees of JSP support are listed on Sun's JSP web site (http://java.sun.com/products/jsp/).

You may want to evaluate some of these products when you're ready to start developing your application, but all you really need to work with the examples in this book are a regular text editor, such as Notepad, vi, or Emacs, and of course the Tomcat server. So let's get going and take a closer look at what JSP has to offer. We need a solid ground to stand on, though, so in the next chapter we will start with the foundations upon which JSP is built: HTTP and Java servlets.

HTTP and Servlet Basics

The HTTP Request/Response Model HTTP and all extended protocols based on HTTP are based on a very simple but powerful communications model. Here's how it works: a client, typically a web browser, sends a request for a resource to a server, and the server sends back a response corresponding to the requested resource (or a response with an error message if it can't deliver the resource for some reason). A resource can be a simple HTML file, or it can be a program that stores the information sent in a database and generates a dynamic response.



This simple model implies three things you need to be aware of:

1. HTTP is a stateless protocol. This means that the server does not keep any information about the client after it sends its response, and therefore cannot recognize that multiple requests from the same client may be related.

2. Web applications cannot easily provide the kind of immediate feedback typically found in standalone GUI applications such as word processors or traditional client-server applications. Every interaction between the client and the server requires a request/response exchange. Performing a request/response exchange when a user selects an item in a list box or fills out a form element is usually too taxing on the bandwidth available to most Internet users.

3. There's nothing in the protocol that tells the server how a request is made; consequently, the server cannot distinguish between various methods of triggering the request on the client. For example, the HTTP protocol does not allow a web server to differentiate between an explicit request caused by clicking a link or submitting a form and an implicit request caused by resizing the browser window or using the browser's Back button. In addition, HTTP does not allow the server to invoke client-specific functions, such as going back in the browser history list or sending the response to a certain frame.

Requests in Detail

A user sends a request to the server by clicking a link on a web page, submitting a form, or explicitly typing a web page address in the browser's address field. To send a request, the browser needs to know which server to talk to and which resource to ask for. This information is

specified by the Uniform Resource Identifier (URI), also commonly referred to as a Uniform Resource Locator (URL). URI is the general term, while a URL is the specific type of URI used to completely identify a web resource such as an HTML page. Here is an example of a URL:

URL: http://www.gefionsoftware.com/index.html

The first part of this URL specifies that the HTTP protocol is used to request the resource. This is followed by the name of the server, www.gefionsoftware.com. The web server waits for requests to come in on a special TCP/IP port. Port number 80 is the standard port for HTTP requests. If the web server uses another port, the URL must specify the port number in addition to the server name. For example:

http://www.gefionsoftware.com:8080/index.html

This URL is sent to a server that uses port 8080 instead of 80. The last part of the URL, /index.html, identifies the resource that the client is requesting. This is sometimes called the URI path.

The client browser always makes a request by sending a request message. An HTTP request message consists of three things: a request line, request headers, and sometimes a request body.

The request line starts with the request method name, followed by a resource identifier and the protocol version used by the browser:

GET /index.html HTTP/1.0

The most commonly used request method is named GET. As the name implies, a GET request is used to retrieve a resource from the server. It's the default request method, so if you type a URL in the browser's address field or click on a link, the request will be sent to the server as a GET request.

The request headers provide additional information the server may need to process the request. The message body is included only in some types of requests, like the POST request discussed later.

Responses in Detail

When the web server receives the request, it looks at the URI and decides, based on configuration information, how to handle it. It may handle it internally by simply reading an HTML file from the filesystem, or it may forward the request to some component that is responsible for the resource corresponding to the URI.

This might be a program that uses a database to dynamically generate an appropriate response. To the client, it makes no difference how the request is handled; all it cares about is getting a

response. The response message looks similar to the request message. It consists of three things: a status line, response headers, and possibly a response body. Here's an example:

HTTP/1.0 200 OK
Last-Modified: Mon, 20 Dec 1999 23:26:42 GMT
Date: Tue, 11 Jan 2000 20:52:40 GMT
Status: 200
Content-Type: text/html
Servlet-Engine: Tomcat Web Server/3.2
Content-Length: 59

Request Parameters

Besides the URI and headers, a request message can contain additional information in the form of parameters. If the URI identifies a server-side program for displaying weather information, for example, request parameters can provide information about which city the user wants to see a forecast for.

In an e-commerce application, the URI may identify a program that processes orders, with the user's customer number and the list of items to be purchased transferred as parameters. Parameters can be sent in one of two ways: tacked on to the URI in the form of a query string , or sent as part of the request message body. Here is an example of a URI with a query string:

http://www.weather.com/forecast?city=Hermosa+Beach&state=CA

Request Methods

GET is the most commonly used request method, intended to retrieve a resource without causing anything else to happen on the server. The POST method is almost as common as GET. A POST request is intended to request some kind of processing on the server, for instance, updating a database or processing a purchase order. The way parameters are transferred is one of the most obvious differences between the GET and POST request methods. A GET request always uses a query string to send parameter values, while a POST request always sends them as part of the body (additionally, it can send some parameters as a query string, just to make life interesting). If you code a link to a URI in an HTML page using an element, clicking on the link results in a GET request being sent to the server. Since the GET request uses a query string to pass parameters, you can include hardcoded parameter values in the link URI:

****

Hermosa Beach weather forecast

****

When you use a form to send user input to the server, you can specify whether to use the GET or POST method with the method attribute, as shown below:

```
form action="/forecast" method="POST">
 City: <input name="city" type="text">
 State: <input name="state" type="text">
 <p>
 <input type="SUBMIT">
</form>
```

If the user enters "Hermosa Beach" and "CA" in the form fields and clicks on the Submit button, the browser sends a request message like this to the server:

```
POST /index.html HTTP/1.0
Host: www.gefionsoftware.com
User-Agent : Mozilla/4.5 [en] (WinNT; I)
Accept: image/gif, image/jpeg, image/pjpeg, image/png, */*
Accept-language : en
Accept-charset : iso-8859-1,*,utf-8
city=Hermosa+Beach&state=CA
```

Besides GET and POST, HTTP specifies the following methods:

OPTIONS

The OPTIONS method is used to find out what options (e.g., methods) a server or resource offers.

HEAD

HEAD method is used to get a response with all headers that would be generated by a GET request, but without the body. It can be used to make sure a link is valid or to see when a resource was last modified.

PUT

The PUT method is used to store the message body content on the server as a resource identified by the URI.

DELETE

The DELETE method is used to delete the resource identified by the URI.

TRACE

The TRACE method is used for testing the communication between the client and the server. The server sends back the request message, exactly as it was received, as the body of the response. Note that these methods are not normally used in a web application.

State Management

As I touched on earlier, HTTP is a stateless protocol; when the server sends back the response corresponding to the request, it forgets all about the transaction. If a user sends a new request, the server has no way of knowing if it is related to the previous request.

This is fine for static content such as regular HTML files, but it's a problem for web applications where a number of requests may be needed to complete a transaction. Consider a shopping cart application: the server-side application needs to allow the user to select items in multiple steps, check the inventory when the user is ready to make the purchase, and finally process the order. In this scenario, the application needs to keep track of information provided by multiple requests from the same browser. In other words, it needs to remember the client's transaction state.

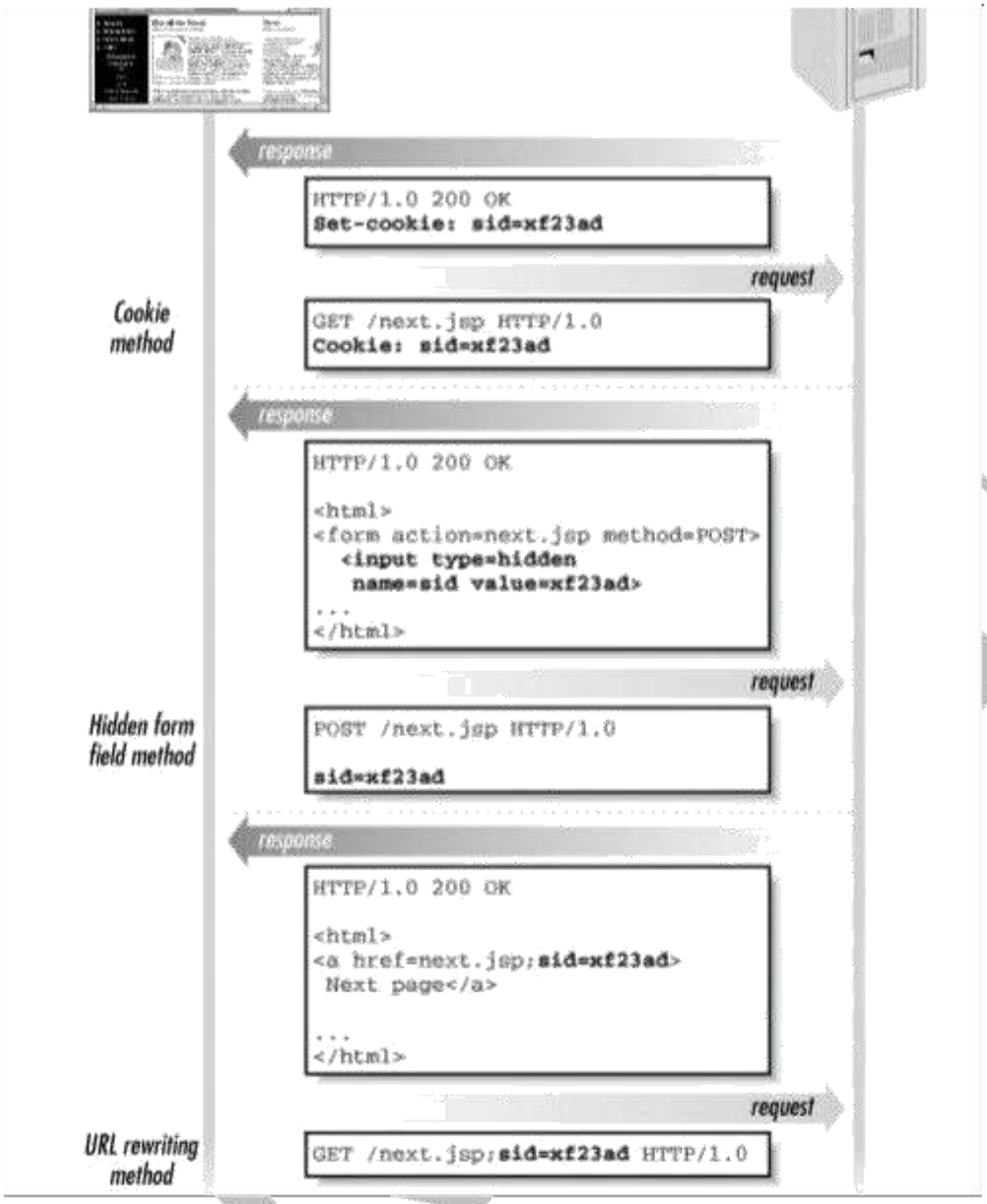
There are two ways to solve this problem, and both have been used extensively for web applications with a variety of server-side technologies. The server can either return the complete state with each response and let the browser send it back as part of the next request; or, it can save the state somewhere on the server and send back only an identifier that the browser returns with the next request. The identifier is then used to locate the state information saved on the server.

In both cases, the information can be sent to the browser in one of three ways:

·As a cookie

· Embedded as hidden fields in an HTML form

· Encoded in the URIs in the response body, typically as links to other application pages (this is known as URL rewriting)



The Problem with Servlets

In many Java servlet-based applications, processing the request and generating the response are both handled by a single servlet class. A example servlet looks like this:

```
public class OrderServlet extends HttpServlet {
 public void doGet(HttpServletRequest request,
 HttpServletResponse response)
 throws ServletException, IOException {
```

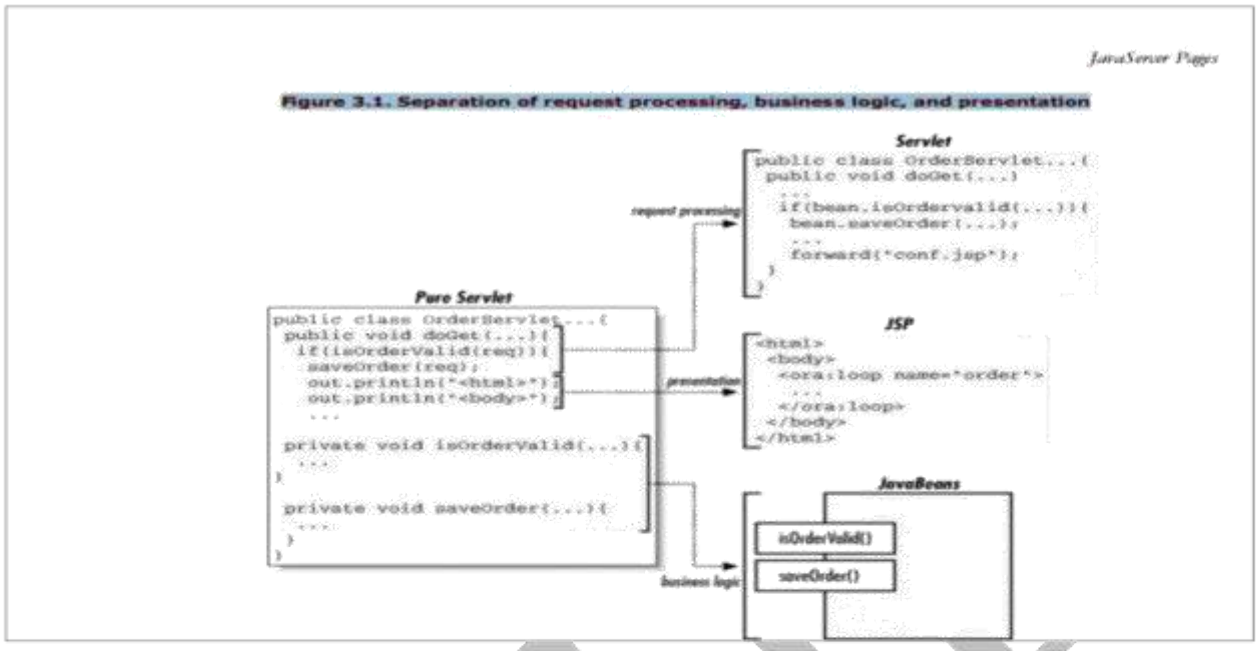
```
response.setContentType("text/html");
PrintWriter out = response.getWriter( );
if (isOrderInfoValid(request)) {
saveOrderInfo(request);
out.println("<html>");
out.println(" <head>");
out.println(" <title>Order Confirmation</title>");
out.println(" </head>");
out.println(" <body>");
out.println(" <h1>Order Confirmation</h1>");
renderOrderInfo(request);
out.println(" </body>");
out.println("</html>");
}
```

...

Detailed Java programming knowledge is needed to develop and maintain all aspects of the application, since the processing code and the HTML elements are lumped together. · Changing the look and feel of the application, or adding support for a new type of client (such as a WML client), requires the servlet code to be updated and recompiled.

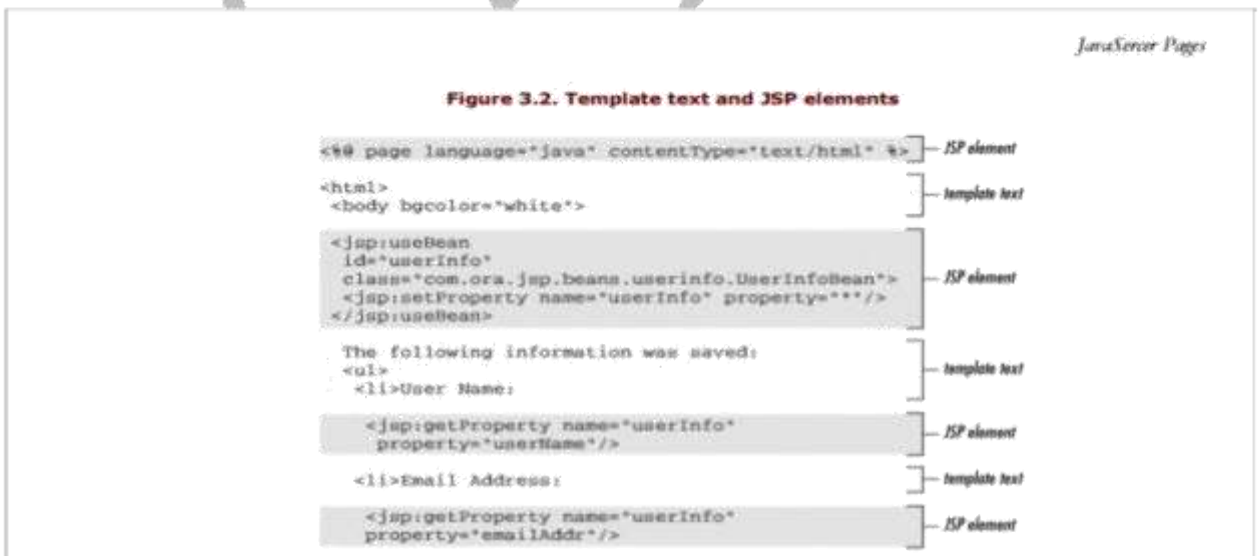
·It's hard to take advantage of web page development tools when designing the application interface. If such tools are used to develop the web page layout, the generated HTML must then be manually embedded into the servlet code, a process that is time-consuming, error-prone, and extremely boring.

Adding JSP to the puzzle lets you solve these problems by separating the request processing and business logic code from the presentation, as illustrated in Figure 3.1. Instead of embedding HTML in the code, you place all static HTML in JSP pages, just as in a regular web page, and add a few JSP elements to generate the dynamic parts of the page. The request processing can remain the domain of servlet programmers, and the business logic can be handled by JavaBeans and Enterprise JavaBeans (EJB) components.



The Anatomy of a JSP Page A JSP page is simply a regular web page with JSP elements for generating the parts of the page that differ for each request, as shown in Figure 3.2. Everything in the page that is not a JSP element is called template text .

Template text can really be any text: HTML, WML, XML, or even plain text. Since HTML is by far the most common web page language in use today, most of the descriptions and examples in this book are HTML-based, but keep in mind that JSP has no dependency on HTML; it can be used with any markup language. Template text is always passed straight through to the browser.



JSP Elements

There are three types of elements with JavaServer Pages: directive, action, and scripting elements. The directive elements, shown in Table 3.1, are used to specify information about the page itself that remains the same between page requests, for example, the scripting language used in the page, whether session tracking is required, and the name of a page that should be used to report errors, if any.

Table 3.1, Directive Elements

Element	Description
<%@ page ... %>	Defines page-dependent attributes, such as scripting language, error page, and buffering requirements
<%@ include ... %>	Includes a file during the translation phase
<%@ taglib ... %>	Declares a tag library, containing custom actions, used in the page

Table 3.2, Standard Action Elements

Element	Description
<jsp:useBean>	Makes a JavaBeans component available in a page
<jsp:getProperty>	Gets a property value from a JavaBeans component and adds it to the response
<jsp:setProperty>	Sets a JavaBeans property value
<jsp:include>	Includes the response from a servlet or JSP page during the request processing phase
<jsp:forward>	Forwards the processing of a request to a servlet or JSP page
<jsp:param>	Adds a parameter value to a request handed off to another servlet or JSP page using <jsp:include> or <jsp:forward>
<jsp:plugin>	Generates HTML that contains the appropriate client browser-dependent elements (OBJECT or EMBED) needed to execute an Applet with the Java Plugin software

Scripting elements, shown in Table 3.3, allow you to add small pieces of code to a JSP page, such as an `if` statement to generate different HTML depending on a certain condition. Like actions, they are also executed when the page is requested. You should use scripting elements with extreme care: if you embed too much code in your JSP pages, you will end up with the same kind of maintenance problems as with servlets embedding HTML.

Table 3.3, Scripting Elements

Element	Description
<% ... %>	Scriptlet, used to embed scripting code.
<%= ... %>	Expression, used to embed Java expressions when the result shall be added to the response. Also used as runtime action attribute values.
<%! ... %>	Declaration, used to declare instance variables and methods in the JSP page implementation class.

JSP Processing

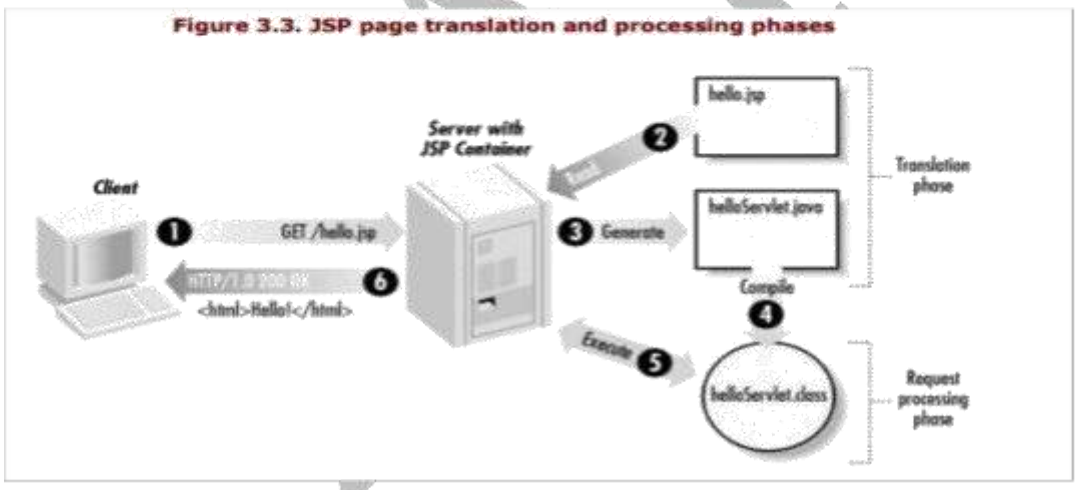
A JSP page cannot be sent as-is to the browser; all JSP elements must first be processed by the server. This is done by turning the JSP page into a servlet, and then executing the servlet. Just as a web server needs a servlet container to provide an interface to servlets, the server needs a JSP container to process JSP pages. The JSP container is often implemented as a servlet configured to handle all requests for JSP pages.

In fact, these two containers - a servlet container and a JSP container - are often combined into one package under the name web container (as it is referred to in the J2EE documentation). A JSP container is responsible for converting the JSP page into a servlet (known as the JSP page implementation class) and compiling the servlet.

These two steps form the translation phase . The JSP container automatically initiates the translation phase for a page when the first request for the page is received. The translation phase takes a bit of time, of course, so a user notices a slight delay the first time a JSP page is requested. The translation phase can also be initiated explicitly; this is referred to as precompilation of a JSP page.

The JSP container is also responsible for invoking the JSP page implementation class to process each request and generate the response. This is called the request processing phase.

Figure 3.3. JSP page translation and processing phases



As long as the JSP page remains unchanged, any subsequent processing goes straight to the request processing phase (i.e., it simply executes the class file). When the JSP page is modified, it goes through the translation phase again before entering the request processing phase. So in a way, a JSP page is really just another way to write a servlet without having to be a Java programming wiz. And, except for the translation phase, a JSP page is handled exactly like a

regular servlet: it's loaded once and called repeatedly, until the server is shut down. By virtue of being an automatically generated servlet, a JSP page inherits all of the advantages of servlets described in Chapter 2 : platform and vendor independence, integration, efficiency, scalability, robustness, and security. Let's look at a simple example of a servlet. In the tradition of programming books for as far back as anyone cares to remember, we start with an application that just writes Hello World, but this time we will add a twist: our application will also show the current time on the server. Example 3.1 shows a hand-coded servlet with this functionality.

Example 3.1. Hello World Servlet

```
public class HelloWorldServlet implements Servlet {
 public void service(ServletRequest request,
 ServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter( );
out.println("<html>");
 out.println(" <head>");
 out.println(" <title>Hello World</title>");
 out.println(" </head>");
 out.println(" <body>");
 out.println(" <h1>Hello World</h1>");
 out.println(" It's " + (new java.util.Date( ).toString( )) +
 " and all is well.");
 out.println(" </body>");
 out.println("</html>");
 }
}
```

JSP Application

Design with MVC JSP technology can play a part in everything from the simplest web application, such as an online phone list or an employee vacation planner, to full-fledged enterprise applications, such as a human resource application or a sophisticated online shopping site. How large a part JSP plays differs in each case, of course. In this section, we introduce a design model suitable for both simple and complex applications called Model-ViewController (MVC).

MVC was first described by Xerox in a number of papers published in the late 1980s. The key point of using MVC is to separate components into three distinct units: the Model, the View, and the Controller. In a server application, we commonly classify the parts of the application as: business logic, presentation, and request processing. Business logic is the term used for the manipulation of an application's data, i.e., customer, product, and order information. Presentation refers to how the application is displayed to the user, i.e., the position, font, and size. And finally, request processing is what ties the business logic and presentation parts together. In MVC terms, the Model corresponds to business logic and data, the View to the presentation logic, and the Controller to the request processing.

Why use this design with JSP? The answer lies primarily in the first two elements. Remember that an application data structure and logic (the Model) is typically the most stable part of an application, while the presentation of that data (the View) changes fairly often. Just look at all the face-lifts that web sites have gone through to keep up with the latest fashion in web design. Yet, the data they present remains the same. Another common example of why presentation should be separated from the business logic is that you may want to present the data in different languages or present different subsets of the data to internal and external users. Access to the data through new types of devices, such as cell phones and Personal Digital Assistants (PDAs), is the latest trend. Each client type requires its own presentation format. It should come as no surprise, then, that separating business logic from presentation makes it easier to evolve an application as the requirements change; new presentation interfaces can be developed without touching the business logic.

This MVC model is used for most of the examples in this book. In Part II, JSP pages are used as both the Controller and the View, and JavaBeans components are used as the Model. The

examples in Chapter 5 through Chapter 7 use a single JSP page that handles everything, while Chapter 8 through Chapter 11 show how you can use separate pages for Control and View to make the application easier to maintain. Many types of real-world applications can be developed this way, but what's more important is that this approach allows us to examine all the JSP features without getting distracted by other technologies. In Part III, we look at other possible role assignments when JSP is combined with servlets and Enterprise JavaBeans.

Setting Up the JSP Environment.

In this chapter you will learn how to install the Tomcat server and add a web application containing all the examples used in this book. You can, of course, use any web server that supports JSP 1.1, but Tomcat is a good server for development and test purposes. You can learn more about the Jakarta project and Tomcat, as well as how you can participate in the development, at the Jakarta web site: http://jakarta.apache.org.

Installing the Java Software Development Kit

Tomcat is a pure Java web server with support for the Servlet 2.2 and JSP 1.1 specifications. To use it, you must first install a Java runtime environment. If you don't already have one, you can download a Java SDK for Windows, Linux, and Solaris at http://java.sun.com/j2se/. I recommend that you install the Java 2 SDK as opposed to the slimmed-down Runtime Environment (JRE) distribution.

The reason is that JSP requires a Java compiler, which is included in the SDK but not in the JRE. Sun Microsystems has made the javac compiler from the SDK available separately for redistribution by the Apache Software Foundation. So technically, you could use the JRE and download the Java compiler as part of the Tomcat package, but even as I write this chapter, the exact legal conditions for distributing the compiler are changing.

Another alternative is to use the Jikes compiler from IBM (http://www10.software.ibm.com/developerworks/opensource/jikes/). Tomcat can be configured to use Jikes instead of the javac compiler from Sun; read the Tomcat documentation if you would like to try this. To make things simple, though, I suggest installing the Java 2 SDK from Sun. The examples were developed and tested with Java 2 SDK, Standard Edition, v1.2.2 and v1.3. I recommend that you use the latest version of the SDK available for your platform.

If you need an SDK for a platform other than Windows, Linux, or Solaris, there's a partial list of ports made by other companies at Sun's web site http://java.sun.com/cgi-bin/java-ports.cgi/

Also check your operating system vendor's web site. Most operating system vendors have their own SDK implementation available for free.

Installation of the SDK varies depending on platform but is typically easy to do. Just follow the instructions on the web site where you download the SDK.

Before you install and run Tomcat, make sure that the JAVA_HOME environment variable is set to the installation directory of your Java environment, and that the Java bin directory is included in the PATH environment variable. On a Windows system, you can see if an environment variable is set by typing the following command in a Command Prompt window:

C:\> echo %JAVA_HOME%

C:\jdk1.1.2

If JAVA_HOME is not set, you can set it and include the bin directory in the PATH like this on a Windows system

(assuming Java is installed in C:\jdk1.2.2):

C:\> set JAVA_HOME=C:\jdk1.1.2

C:\> set PATH=%JAVA_HOME%\bin;%PATH%

On a Windows 95/98 system, you can add these commands to the C:\AUTOEXEC.BAT file to set them permanently. Just use a text editor, such as Notepad, and add lines with the set commands. The next time you boot the PC, the environment variables will be set automatically. For Windows NT and 2000, you can set them permanently from the Environment tab in the System Properties tool.

If you use Linux or some other Unix platform, the exact commands depend on which shell you use. With bash, which is commonly the default for Linux, use the following commands (assuming Java is installed in

/usr/local/jdk1.2.2):

[hans@gefion /] export JAVA_HOME=/usr/local/jdk1.2.2

[hans@gefion /] export PATH=$JAVA_HOME/bin:$PATH

echo $PATH

/usr/local/jdk1.2.2/bin:/usr/local/bin:/bin:/usr/bin

Installing the Tomcat Server

You can download the Tomcat Server either in binary format or as source code that you compile yourself. If you're primarily interested in learning about JSP, I recommend that you use the binary download to run the examples in this book and develop your own applications. If you're a Java programmer and interested in seeing how Tomcat is implemented, feel free to download the source and take a look at the internals.

The binary distribution is available at http://jakarta.apache.org/downloads/binindex.html

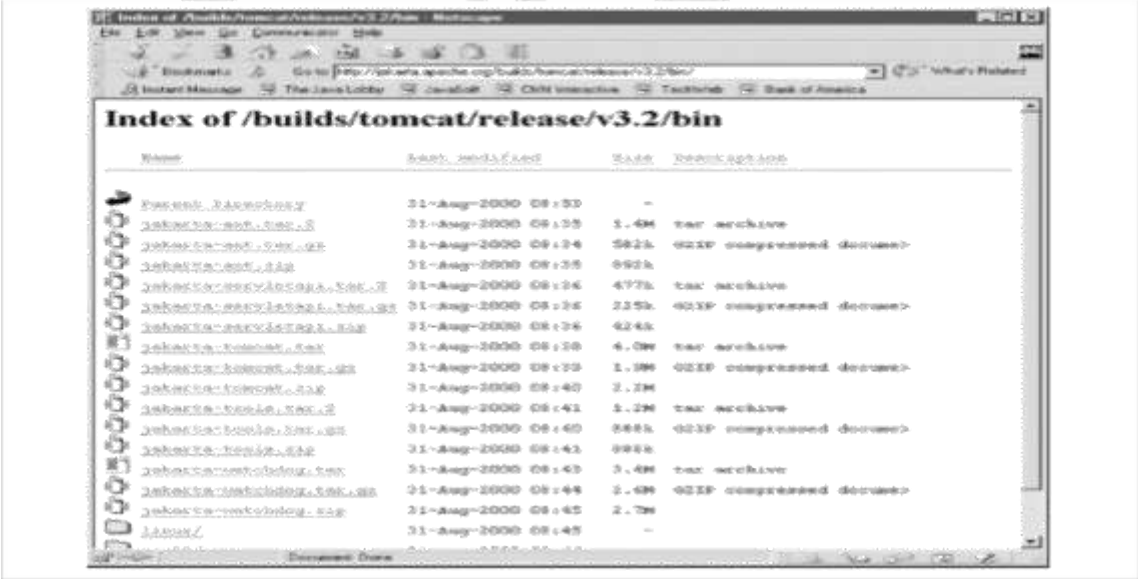
On this page you find three types of builds:

· Release builds

· Milestone builds

· Nightly builds

Release builds are stable releases that have been tested extensively and verified to comply with the servlet and JSP specifications. Milestone builds are created as intermediary steps towards a release build. They often contain new features that are not yet fully tested, but are generally known to work. A nightly build, however, may be very unstable. It's actually a snapshot of the latest source code and may have been tested only by the person who made the latest change. You should use a nightly build only if you're involved in the development of Tomcat.



Windows Platforms

The Windows files are named startup.bat, shutdown.bat, and tomcat.bat. The tomcat.bat file is the main script for controlling the server; it's called by the two other scripts startup.bat and shutdown.bat. To start the server in a separate window, change directory to the bin directory and run the startup.bat file:

C:\Jakarta> cd jakarta-tomcat\bin

C:\Jakarta\jakarta-tomcat\bin> startup

A new Command Prompt window pops up and you see startup messages like this:

2000-09-01 09:27:10 - ContextManager: Adding context Ctx(/examples)

2000-09-01 09:27:10 - ContextManager: Adding context Ctx(/admin)

Starting tomcat. Check logs/tomcat.log for error messages

2000-09-01 09:27:10 - ContextManager: Adding context Ctx( )

2000-09-01 09:27:10 - ContextManager: Adding context Ctx(/test)

2000-09-01 09:27:13 - PoolTcpConnector: Starting HttpConnectionHandler on 8080

2000-09-01 09:27:13 - PoolTcpConnector: Starting Ajp12ConnectionHandler on 8007

For some installations, this command may not work. If it doesn't work, try this instead:

1. Close the Command Prompt window and open a new one.

2. Click on the MS-DOS icon at the top-left of the window.

3. Select the Properties option.

4. Click on the Memory tab.

5. Change the Initial Environment value from Auto to 4096.

6. Click on OK and try to start the server again.

Unix Platforms

For Unix, the corresponding scripts are named startup.sh, shutdown.sh, and tomcat.sh. Start the server with this command:

[hans@gefion /usr/local/jakarta-tomcat/bin] ./startup.sh

If you want Tomcat to start each time you boot the system, you can add the following commands to your

/etc/rc.d/rc.local (or equivalent) startup script:

export JAVA_HOME=/usr/local/jdk1.2.2

export TOMCAT_HOME=/usr/local/jakarta-tomcat

$TOMCAT_HOME/bin/startup.sh &

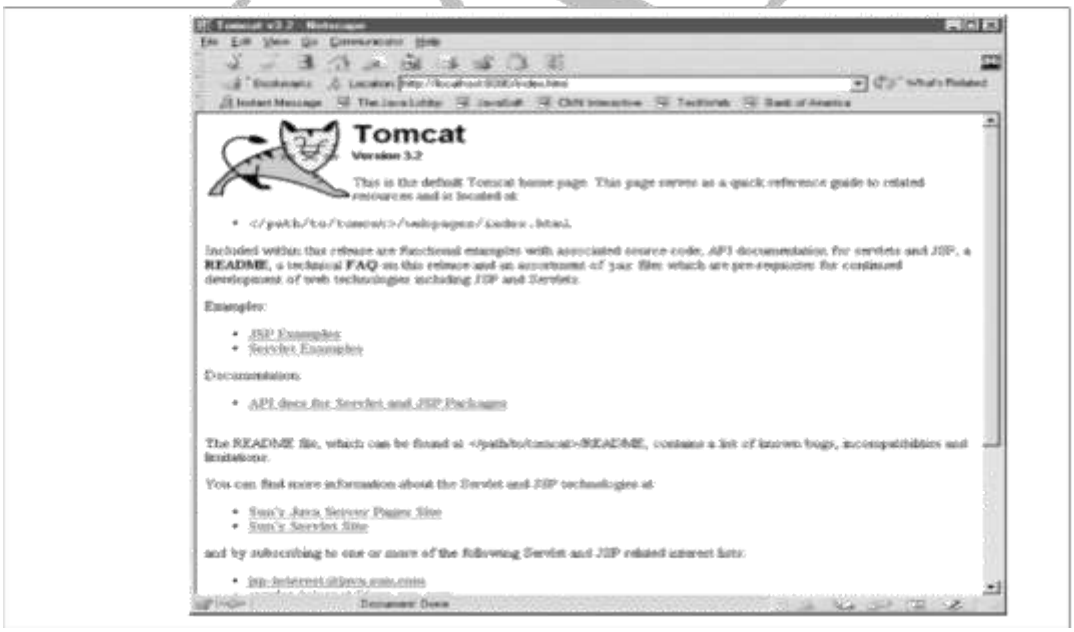
Two more subdirectories under the Tomcat home directory are then created the first time you start the

server:

Testing Tomcat

To test the server - assuming you're running Tomcat on the same machine as the browser and that you're using the default port for Tomcat (8080) - open a browser and enter the following URL in the Location/Address field:

http://localhost:8080/



Installing the Book Examples

All JSP pages, HTML pages, Java source code, and class files for the book examples can be downloaded directly from the O'Reilly web site: http://www.oreilly.com/catalog/jserverpages/

They can also be downloaded from the book web site: http://www.TheJSPBook.com The file that contains all the examples is called jspbook.zip. Save the file on your hard drive, for instance in C:\JSPBook on a Windows platform, and unpack it:

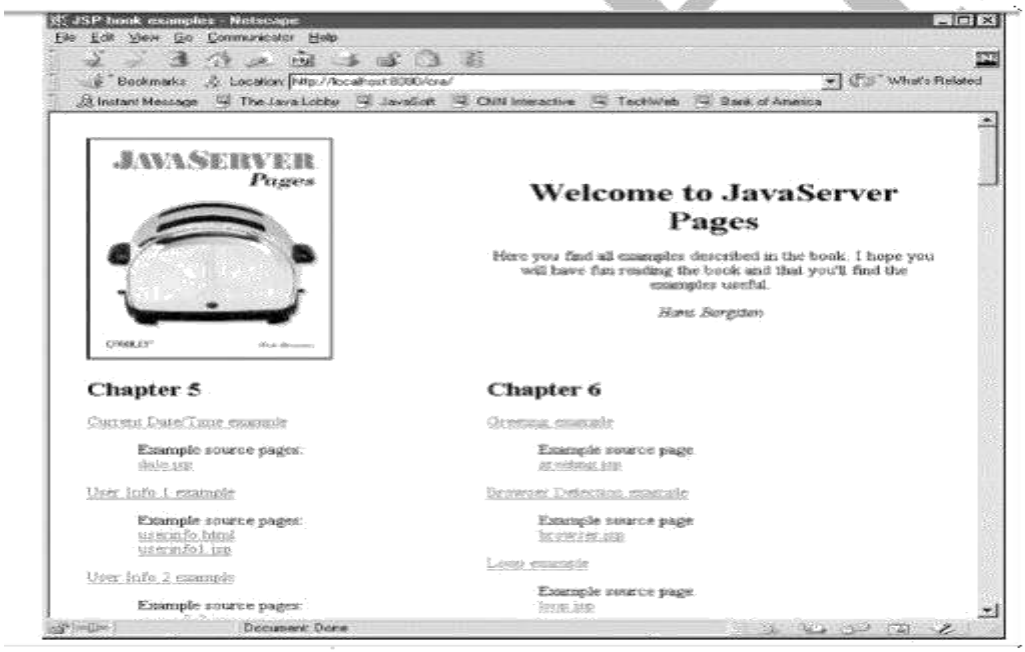
C:\JSPBook> jar xvf jspbook.zip You can use the same command on a Unix platform.

To install the example application for Tomcat, copy the web application directory structure to Tomcat's default directory for applications, called webapps.

Use this command on a Windows platform: C:\JSPBook> xcopy /s /i ora %TOMCAT_HOME%\webapps\ora On a Unix platform it looks like this:

[hans@gefion /usr/local/jspbook] cp -R ora $TOMCAT_HOME/webapps

At this point, you must shut down and restart the Tomcat server. After that, you can point your browser to the ora application with the following URL: http://localhost:8080/ora/



Possible Question

Part B (2 marks)

1) Define JSP
2) List out any 3 elements of JSP
3) Define State Management
4) Write short notes on anatomy of JSP?
5) What is mean by HTTP?
6) Define servlet
7) Define MVC
8) Define problem with servlet
9) What is mean by JSP page?
10) How to set of JSP Environment?

Part C (6 marks)

1) Explain in detail about what is JSP? what are the problems with servlet in JSP
2) Explain in detail about JSP Processing
3) What is mean by JSP, how to working with JSP environment?
4) Explain in detail about JSP environment
5) Explain in detail about JSP application design with MVC
6) Explain In Detail about JSP page
7) Explain in Detail about anatomy of JSP page
8) How to Display values in JSP
9) Explain in detail about HTTP and servlet basics
10) Explain in Detail about problem with servlets

KARPAGAM ACADEMY OF HIGHER EDUCATION

PART - A (ONLINE EXAMINATION)

MULTIPLE CHOICE QUESTIONS (Each question carries one mark)

SUBJECT: INTERNET TECHNOLOGY

SUB.CODE: 17ITU404A **CLASS : II B.Sc IT**

UNIT III

QUESTIONS	OPTION A	OPTION B	OPTION C	OPTION D	ANSWERS
1.JSP stands for _____	page	page	packet	packet	pages
2.Developers can use java to efficiently create multi tier _____ side applications.	server	client	a &b	none	server
3.Java server pages is a technology for developing web pages that include_____ content.	static	dynamic	both a&b	none	dynamic
4.CGI stands for _____	gateway	gateway	gateway	gateway	gateway
5.J2EE stands for_____	enterprise	edition	entering	entered exit	enterprise
6._____ acessing naming and directory services.	JTA	JNDI	JNDI2	JDBC	JNDI
7.Java beans uses for creating reusable _____.	components	components	components	both a&c	components
8._____is a message oriented middleware.	JTA	J2EE	CGI	JMS	JMS
9._____ is used to handling remote objects.	JSP	J2EE	RMI	CGI	RMI
10.RMI and _____ is used to handling remote objects.	corba	cobra	coarb	none	corba
11.JMS is a message oriented_____.	hardware	software	middleware	upperware	middleware
12.JTA stands for_____.	transaction	transformati	transaction	API	transaction
13._____ is used for performing atomic transactions.	J2EE	JDBC	these	JTB	none of this
14.A JSP page contains _____ elements	markup	language	language	both a&c	markup
15.JSP is used to_____data from database	retrieving	deleting	editing	all of this	retrieving
16.Jakarta project_____	Apache	windows	saffari	none of this	Apache
17.JSP supports _____different server products.	>30	30	both b&d	<31	both b&d
18._____authoring tools supports by JSP	10	11	12	13	10
19.HTTP stands for_____	transmission	transfer	texture	none of this	transfer
20.In HTTP a client is a _____	web page	web browser	server	all the above	web browser
21.A HTML file is a _____	document	reference	source	resource	resource
22.HTTP is a _____protocol	statefull	oriented	stateless	less	stateless
23.GUI stands for _____	user	graphical	user	none	user
24._____ is a example for standalone GUI application	Word excel	processor	document	word	processor

25.URI stands for _____	Resource	Relocation	Resource	Relocation	Resource
26.URL stands for_____	Resource	Resource	Resource	Resource	Resource
27.HTTP port number	8080	808	8082	8022	8080
28.The URL send to server that uses _____port number instead of 80	8022	8048	8032	8080	8080
29.In _____application,URI may identify a program.	E-Campus	Commerce	E-Tracking	Attendance	Commerce
30.URI headers request a message in the form of_____	Parameters	Arguments	both a&b	none of this	both a&b
31._____is most commonly used request method.	gets	set	get	puts	get
32._____ is used to retrieve message from server.	gets	set	get	getm	get
33.The message body request is_____.	puts	put	post	post/get	post
34._____method is used to find out the option.	select	select option	option	none of this	option
35.HEAD method is used to get _____ with all headers.	retrieving	response	request	resource	response
36._____ method is used to check that the link is valid or not.	head	put	delete	trace	head
37._____method is used to store the meaasge body.	head	none of this	delete	trace	none of this
38.put method is used to store the meaasge _____.	body	header	details	all the above	body
39._____ method is used to delete the resource indentified by URI	head	put	delete	trace	delete
40._____ method is used for testing communication between client-server.	head	put	trace	connect	trace
41.trace method is used for testing _____ between client-server	connection	on	connectivity	these	on
42.trace method is used for testing communication between _____.	staff-admin	user-admin	client-server	user-server	client-server
43.the information can be sent to the browser in _____ ways.	4	3	5	1	3
44.URIs in the response body,typically as links to other application pages is called_____.	URI rewritting	URI connection	URI writing	URI to URI	URI rewritting
45.processing the reuqest and generating the response are handled by_____.	JVM	single servlet class	double servlet class	servlet class	single servlet class
46.EJB stands for_____.	beans	java beans	beans	beans	java beans
47.<%!.....%> used for	declaration	definition	initialization	these	declaration
48.design model for simple and complex application is _____.	MCV	VCM	MVC	MV2C	MVC
49.MVC expansion_____.	controller	view	virtual	creator	controller
50.applications of MVC classifies into _____ parts	2	3	4		3
51._____ is used to manipulation of applications data	logic	components	presentation	procesing	logic
52._____ deals with view of application	logic	presentation	processing	both a&c	presentation
53._____ is combination of businesslogic and presentation	retrieving	processing	b only	both b&c	both b&c
54.most stable part of application design model is _____	model	view	controller	both a&b	model
55.PDA stands for_____	diagram	diagram	diagram	digital	digital
56.JRE expansion_____	retreiveing	environment	engine	environment	environment

57._____ is the best server for development and testing	tomcat	both a&b	apache	all the above	tomcat
58.which method is not commonly used in web application _____.	head	all the above	trace	put	all the above
59.the problems with servlets is _____.	ng	error-prone	both a&b	these	both a&b
60.there are _____ type of elements with jsp.	2	4	3	5	3

UNIT-IV

SYLLABUS

JSP: Implicit Objects, conditional processing, displaying values, Using an expression to Set an Attribute, declaring variables and methods, error handling and debugging, sharing data between JSP pages, Requests, and Users, Database Access.

What is JSP Implicit object?

- JSP implicit objects are created during the translation phase of JSP to the servlet.
- These objects can be directly used in scriplets that goes in the service method.
- They are created by the container automatically, and they can be accessed using objects.

There are 9 types of implicit objects available in the container:

What is JSP Implicit object?

- JSP implicit objects are created during the translation phase of JSP to the servlet.
- These objects can be directly used in scriplets that goes in the service method.
- They are created by the container automatically, and they can be accessed using objects.

There are 9 types of implicit objects available in the container:

1. out
2. request
3. response
4. config
5. application
6. session
7. pageContext
8. page
9. exception
10. A list of the 9 implicit objects is given below:

Object	Type
out	JspWriter
request	HttpServletRequest
response	HttpServletResponse

config	ServletConfig
application	ServletContext
session	HttpSession
pageContext	PageContext
page	Object
exception	Throwable

1.OUT

- Out is one of the implicit objects to write the data to the buffer and send output to the client in response
- Out object allows us to access the servlet's output stream
- Out is object of javax.servlet.jsp.jspWriter class
- While working with servlet, we need printwriter object

2.REQUEST

- The request object is an instance of java.servlet.http.HttpServletRequest and it is one of the argument of service method
- It will be created by container for every request.
- It will be used to request the information like parameter, header information , server name, etc.
- It uses getParameter() to access the request parameter.

3.RESPONSE

- "Response" is an instance of class which implements HttpServletResponse interface
- Container generates this object and passes to _jspservice() method as parameter
- "Response object" will be created by the container for each request.
- It represents the response that can be given to the client
- The response implicit object is used to content type, add cookie and redirect to response page

4.CONFIG

- "Config" is of the type java.servlet.servletConfig
- It is created by the container for each jsp page
- It is used to get the initialization parameter in web.xml

5.APPLICATION

- Application object (code line 10) is an instance of javax.servlet.ServletContext and it is used to get the context information and attributes in JSP.
- Application object is created by container one per application, when the application gets deployed.
- Servletcontext object contains a set of methods which are used to interact with the servlet container.We can find information about the servlet container

6.SESSION

- The session is holding "httpsession" object(code line 10).
- Session object is used to get, set and remove attributes to session scope and also used to get session information

7.PAGECONTEXT:

- This object is of the type of pagecontext.
- It is used to get, set and remove the attributes from a particular scope

Scopes are of 4 types:

- Page
- Request
- Session
- Application

8.PAGE

- Page implicit variable holds the currently executed servlet object for the corresponding jsp.
- Acts as this object for current jsp page.

9.EXCEPTION

- Exception is the implicit object of the throwable class.
- It is used for exception handling in JSP.

CONDITIONAL PROCESSING:

In most web applications, you produce different output based on runtime conditions, such as the state of a bean or the value of a request header such as UserAgent (containing information about the type of client that is accessing the page). If the differences are not too great, you can useJSPscriptingelementstocontrol whichpartsofthe JSPpage

are sent to the browser, generating alternative outputs from the same JSP page. However, if the outputs are completely different, I recommend using a separate JSP page for each alternative and passing control from one page to another. This chapter contains a number of examples in which one page is used. In the remainder of this book you'll see plenty of examples where multiple pages are used instead.

Using JavaBeans Properties:

How to use the <jsp:getProperty> and the <jsp:setProperty> actions to access a bean's properties. However, a bean is just a Java class that follows certain coding conventions, so you can also call its methods directly.

Briefly, a bean is a class with a constructor that doesn't take an argument. This makes it possible for a tool, such as the JSP container, to create an instance of the bean class simply by knowing the class name. The other condition of a bean that we are concerned with is the naming of the methods used to access its properties. The method names for reading and writing a property value, collectively known as the bean's *accessor* methods, must be composed of the keywords get and set, respectively, plus the name of the property. For instance, you can retrieve the value of a property named month in a bean with the method getMonth( ) and set it with the method setMonth( ). Individually, the accessor method for reading a property value is known as the *getter* method, and the accessor method for writing a property value is the *setter* method. A property can be read-only, write-only, or read/write depending on whether a getter method, a setter method, or both methods are provided in the class. The Java type for a property, finally, is the type returned by the getter method and the type of the setter methods argument.

Table 6.2, java.util.Date hours Property

Property Name	Java Type	Access	Description
hours	int	read	The hour as a number between 0 (midnight) and 23

Conditional Greeting Page (greeting.jsp)

```
<%@ page language="java" contentType="text/html" %>

<html>
```

```
<body bgcolor="white">

<jsp:useBean id="clock" class="java.util.Date" />

<% if (clock.getHours( ) < 12) { %>

Good morning!

<% } else if (clock.getHours( ) < 17) { %>

Good day!

<% } else { %>

Good evening!

<% } %>

</body>

</html>
```

Using Request Information:

The implicit object request is used to display different messages depending on whether the Internet Explorer or Netscape Navigator browser is used. Example shows the complete page.

Browser-Dependent Page (browser.jsp)

```
<%@ page language="java" contentType="text/html" %>

<html>

<body bgcolor="white">

<% if (request.getHeader("User-Agent").indexOf("MSIE") != -1) { %>

You're using Internet Explorer.

<%

} else

if (request.getHeader("User-Agent").indexOf("Mozilla") != 1) {

%>
```

You're using Netscape.

<% } else { %>

You're using a browser I don't know about.

<% } %>

</body>

</html>

The request object is not a bean, since it doesn't follow all the JavaBeans conventions described above, but it does provide a number of methods you can use to get information about the request. For instance, the request object's getHeader( ) method is used to get the value of a specific request header.

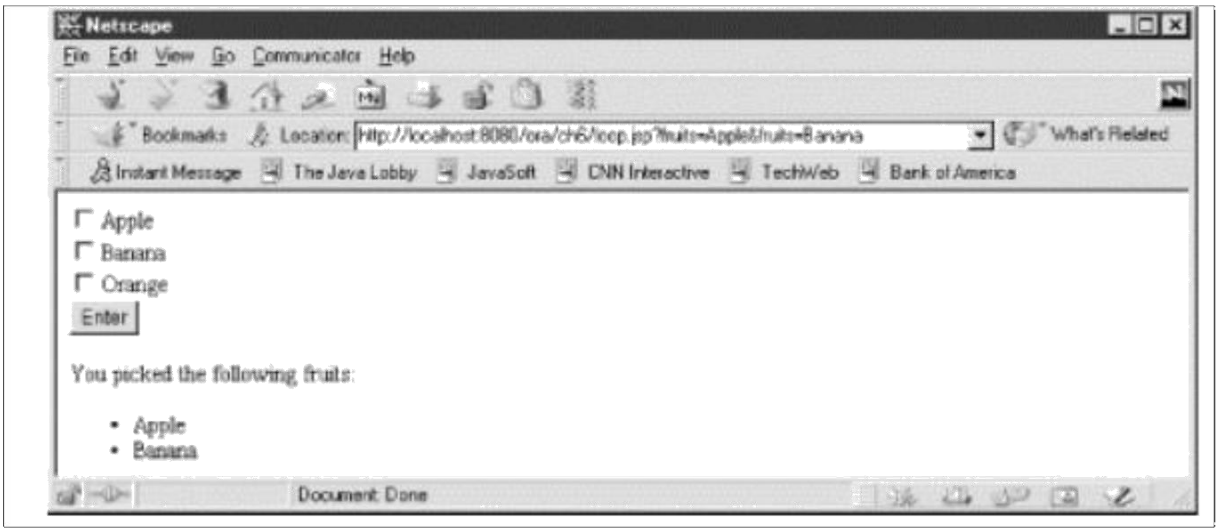
Working with Arrays:

Another common use of scriptlets is to loop over an array. In Example, we let the user pick a number of items from a group of checkboxes, and then use scriptlets to display all the choices.

Looping Over Parameter Array (loop.jsp)

```
<%@ page language="java" contentType="text/html" %>

<html>

<body bgcolor="white">

<form action="loop.jsp">

<input type="checkbox" name="fruits" value="Apple">Apple<br>

<input type="checkbox" name="fruits" value="Banana">Banana<br>

<input type="checkbox" name="fruits" value="Orange">Orange<br>

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

</form>

<%

String[] picked = request.getParameterValues("fruits");

if (picked != null && picked.length != 0) {
```

```
%>

You picked the following fruits:

<ul>

<%

for (int i = 0; i < picked.length; i++) {

out.println("<li>" + picked[i]);

}

%>

</ul>

<% } %>

</body>

</html>
```



Displaying Values:

Besides using scriptlets for conditional output, one more way to employ scripting elements is by using a JSP *expression element* to insert values into the response. A JSP expression element can be used instead of the <jsp:getProperty> action in some places, but it is also useful to insert the value of any Java expression that can be treated as a String. An expression starts with <%= and ends with %>. Note that the only syntax difference compared to a scriptlet is the equals sign (=) in the start identifier. An example is: <%= userInfo.getUserName( ) %>

```
<%@ page import="com.ora.jsp.util.*" %>

...

<tr>

<td>Name:</td>

<td><input type="text" name="userName"

value="<%= StringFormat.toHTMLString(userInfo.getUserName( )) %>" >

</td>

</tr>
```

Packages:

A large application may use many different classes, some of them part of the standard Java libraries, and others developed in-house or by third parties. To organize all these classes, Java provides the notion of a *package*. A package is a group of related classes. The fully qualified name of a class is the combination of the package name and the class name. For instance, the fully qualified name of the class used in Example is com.ora.jsp.util.StringFormat. You can always use the fully qualified name in your Java code, but to save you some typing, you can also *import* a package and then refer to the class with just the short class name. If you look at the top of Example, you see a page directive with the import attribute set to the name of the package the StringFormat class belongs to:

```
<%@ page import="com.ora.jsp.util.*" %>
```

Importing a package doesn't mean that it's physically included in the page. It only tells Java to look for classes with short names in the named package. You can use multiple page directives with import attributes in the same page, or use one with a comma-separated list of import declarations, if you need to import more than one package. In other words, this directive:

```
<%@ page import="java.util.*, com.ora.jsp.util.*" %>
```

has the same effect as these two directives:

```
<%@ page import="java.util.* " %>

<%@ page import="com.ora.jsp.util.*" %>
```

Checking Off Checkboxes Dynamically:

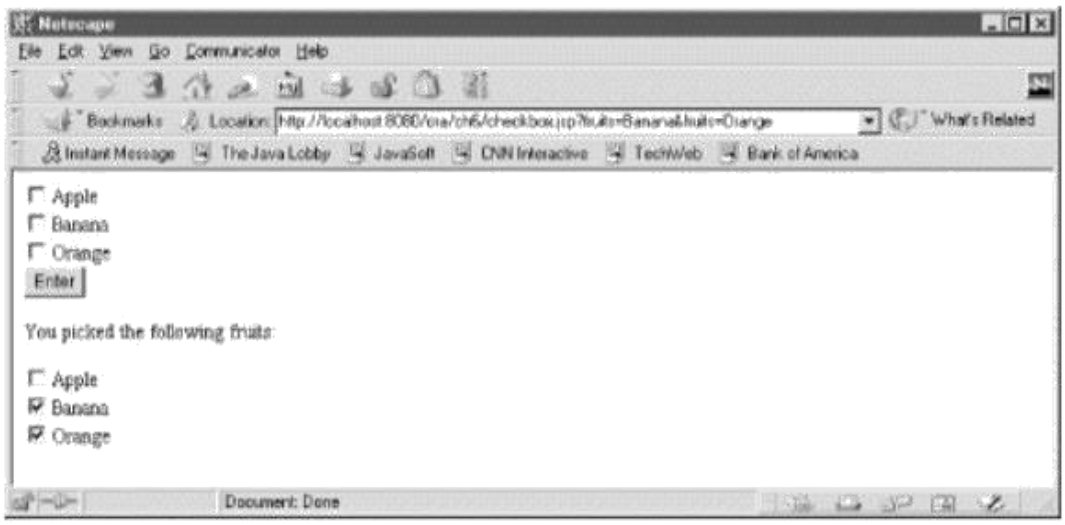
In Example, a for statement is used to loop through an array, but arrays can also be used in many other ways. If the array represents choices the user can make at one time and change at a later time, a form for changing the information can contain a set of checkboxes with the current

choices checked off. An application like this typically gets the current choices from a database, and you will see an example of this in To demonstrate a technique for dynamically checking off checkboxes in a form, however, we keep it simple and use the String[] with fruit choices from Example

Setting Checkbox Values Dynamically (checkbox.jsp)

```
<%@ page language="java" contentType="text/html" %>

<%@ page import="com.ora.jsp.util.*" %> <html>


<body bgcolor="white">

<form action="checkbox.jsp">

<input type="checkbox" name="fruits" value="Apple">Apple<br>

<input type="checkbox" name="fruits"

value="Banana">Banana<br> <input type="checkbox"

name="fruits" value="Orange">Orange<br> <input type="submit"

value="Enter"> </form>

<%

String[] picked = request.getParameterValues("fruits");

if (picked != null && picked.length != 0) { %>


You picked the following fruits:

<form>

<input type="checkbox" name="fruits"

value="Apple" <%= ArraySupport.contains(picked,

"Apple") ? "checked" : "" %> >Apple<br>

<input type="checkbox" name="fruits"

value="Banana" <%= ArraySupport.contains(picked,

"Banana") ? "checked" : "" %> >Banana<br>
```

```
<input type="checkbox" name="fruits" value="Orange"

<%= ArraySupport.contains(picked, "Orange") ?

"checked" : "" %> >Orange<br>

</form>

<% } %>

</body>

</html>
```



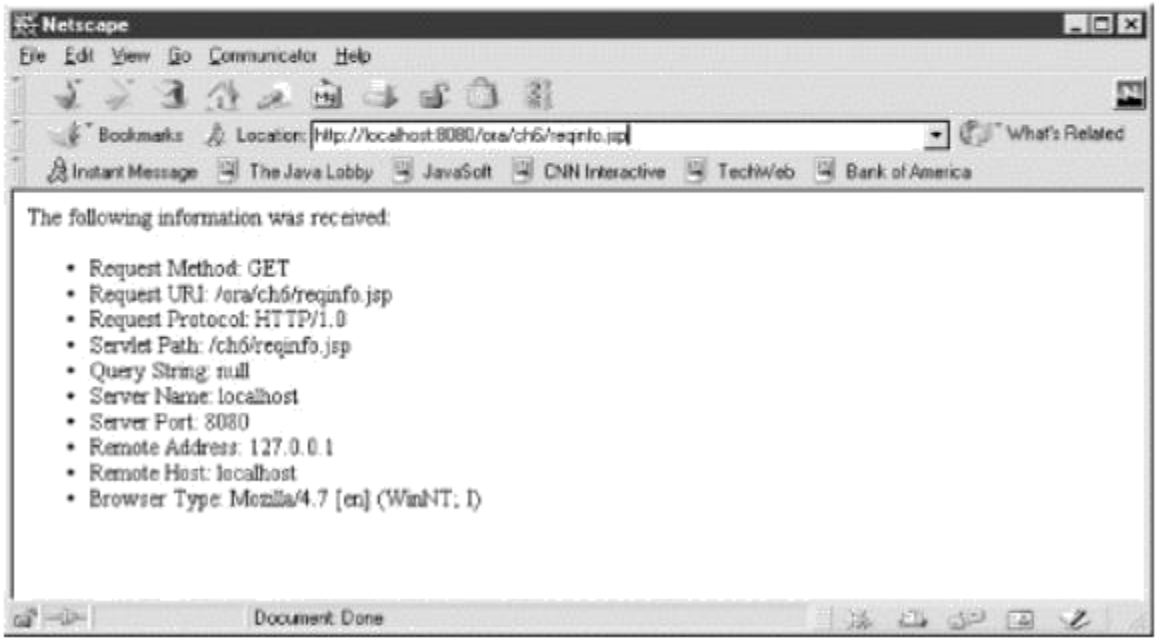
Using More Request Methods

We have already used one of the methods of the implicit request object, but this object provides a wealth of information you may be interested in. So let's use some more request methods.

Displaying Request Info (reqinfo.jsp)

```
<%@ page language="java" contentType="text/html" %>

<html>

<body bgcolor="white">

The following information was received:

<ul>

<li>Request Method: <%= request.getMethod( ) %>
```

```
<li>Request URI: <%= request.getRequestURI( ) %>
<li>Request Protocol: <%= request.getProtocol( ) %>
<li>Servlet Path: <%= request.getServletPath( ) %>
<li>Query String: <%= request.getQueryString( ) %>
<li>Server Name: <%= request.getServerName( ) %>
<li>Server Port: <%= request.getServerPort( ) %>
<li>Remote Address: <%= request.getRemoteAddr( ) %>
<li>Remote Host: <%= request.getRemoteHost( ) %>
<li>Browser Type: <%= request.getHeader("User-Agent") %>
</ul>
</body>
</html>
```



Using an Expression to Set an Attribute:

In all our JSP action element examples so far, the attributes are set to literal string values. But in many cases, the value of an attribute is not known when you write the JSP page; instead, the value must be calculated when the JSP page is requested. For situations like this, you can use a

JSP expression as an attribute value. This is called a request-time attribute value. Here is an example of how this can be used to

set an attribute of a fictitious log entry bean:

<jsp:useBean id="logEntry" class="com.foo.LogEntryBean" />

<jsp:setProperty name="logEntry" property="entryTime"

value="<%= new java.util.Date( ) %>" />

This bean has a property named entryTime that holds a timestamp for a log entry, while other properties hold the information to be logged. To set the timestamp to the time when the JSP page is requested, a <jsp:setProperty> action with a request-time attribute value is used. The attribute value is represented by the same type of JSP expression as in the previous snippet, here an expression that creates a new java.util.Date object (representing the current date and time). The requesttime attribute is evaluated when the page is requested, and the corresponding attribute is set to the result of the expression. One reason is that some attribute values must be known when the page is converted into a servlet. For instance, the class attribute value in the <jsp:useBean> action must be known in the translation phase so that the JSP container can generate valid Java code for the servlet. Request-time attributes also require a bit more processing than static string values, so it's up to the custom action developer to decide if request-time attribute values are supported or not.

Declaring Variables and Methods:

We have used two of the three JSP scripting elements in this chapter: scriptlets and expressions. There's one more, called a declaration element, which is used to declare Java variables and methods in a JSP page. My advice is this: don't use it. Let me explain why.

Java variables can be declared either within a method or outside the body of all methods, like this:

public class SomeClass {

// Instance variable

private String anInstanceVariable;

// Method

public void doSomething( ) {

String aLocalVariable;

}}

A variable declared outside the body of all methods is called an *instance variable*. Its value can be accessed from any method in the class, and it keeps its value even when the method that sets it returns. A variable declared within the body of a method is called a *local variable*. A local variable can be accessed only from the method where it's declared. When the method returns, the local variable disappears.

Each user is assigned what is called a *thread* in the server, and each thread executes the main method in the JSP object. When more than one thread executes the same code, you have to make sure the code is *thread-safe*.

This means that the code must behave the same when many threads are executing as when just one thread executes the code.Multithreading and thread-safe code strategies are best left to programmers. However, you should know that using a JSP declaration element to declare variables exposes your page to multithreading problems. That's because a variable declared using a JSP declaration element ends up as an instance variable in the generated servlet, not as a local variable in a method.

. If one thread changes the value of the instance variable, the value is changed for all threads. To put this in JSP terms, if the instance variable is changed because one user accesses the page, all users accessing the same page will use the new value.When you declare a variable within a scriptlet element instead of in a JSP declaration block, the variable ends up as a local variable in the generated servlet's request processing method. Each thread has its own copy of a local variable, so local variables can't cause any problems if more than one thread executes the same code. If the value of a local variable is changed, it will not affect the other threads.

```
<%@ page language="java" contentType="text/html" %>

<%!

int globalCounter = 0;

%>

<html>

<head>

<title>A page with a counter</title>

</head>

<body bgcolor="white">

This page has been visited: <%= ++globalCounter %> times.

<p>
```

```
<%

int localCounter = 0;

%>

This counter never increases its value: <%= ++localCounter %>

</body>

</html>
```

jspInit( ) and jspDestroy( )

servlet has two methods that the container calls when the servlet is loaded and shut down. These methods are called init( ) and destroy( ), and they allow the servlet to initialize instance variables when it's loaded and clean up when it's shut down, respectively. As you already know, a JSP page is turned into a servlet, so it has the same capability. However, with JSP, the methods are called jspInit( ) and jspDestroy( ) instead.

Again, recommend that you do not declare any instance variables for your JSP pages. If you follow this advice, there's also no reason to declare the jspInit( ) and jspDestroy( ) methods. But I know you're curious, so here's an example of how they can be used.

```
<%@ page language="java" contentType="text/html" %>

<%@ page import="java.util.Date" %>

<%!

int globalCounter = 0;

Date startDate;

public void jspInit( ) {

startDate = new Date( );

}

public void jspDestroy( ) {

ServletContext context = getServletConfig( ).getServletContext( );

context.log("test.jsp was visited " + globalCounter +

" times between " + startDate + " and " + (new Date( )));

}
```

```
%>

<html>

<head>

<title>A page with a counter</title>

</head>

<body bgcolor="white">

This page has been visited: <%= ++globalCounter %> times

since <%= startDate %>.

</body>

</html>
```

Error Handling and Debugging:

When you develop any application that's more than a trivial example, errors are inevitable. A JSP-based application is no exception. There are many types of errors you will deal with. Simple syntax errors in the JSP pages are almost a given during the development phase. And even after you have fixed all the syntax errors, you may still have to figure out why the application doesn't work as you intended due to design mistakes. The application must also be designed to deal with problems that can occur when it's deployed for production use. Users can enter invalid values and try to use the application in ways you never imagined. External systems, such as databases, can fail or become unavailable due to network problems. Since a web application is the face of a company, making sure it behaves well, even when the users misbehave and the world around it falls apart, is extremely important for a positive customer perception.

Dealing with Syntax Errors:

Element Syntax Errors

Tomcat reports some typical syntax errors in JSP directives and action elements. Example shows a version of the *date.jsp* page with a syntax error.

```
<%@ page language="java" contentType="text/html" >

<html>

<body bgcolor="white">

<jsp:useBean id="clock" class="java.util.Date" />
```

The current time at the server is:

Date: <jsp:getProperty name="clock" property="date" />

Month: <jsp:getProperty name="clock" property="month" />

Year: <jsp:getProperty name="clock" property="year" />

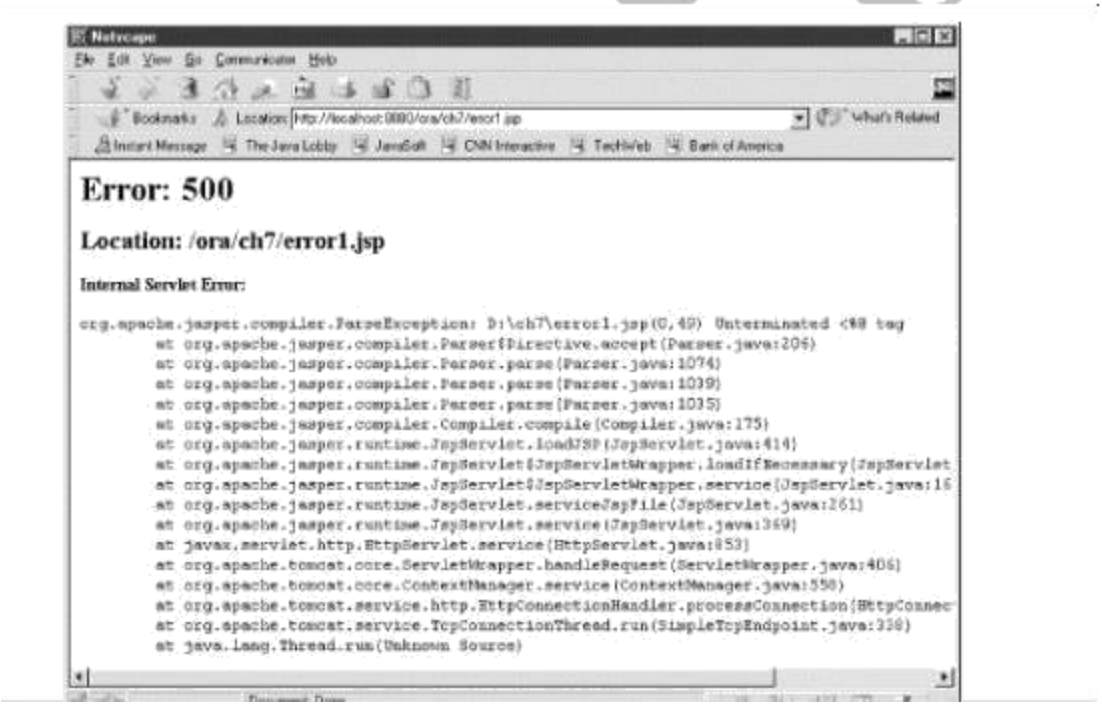
Hours: <jsp:getProperty name="clock" property="hours" />

Minutes: <jsp:getProperty name="clock" property="minutes" />

</body>

</html>

The syntax error here is that the page directive on the first line is not closed properly with %>; the percent sign is missing.



Improperly Terminated Action (error2.jsp):

<%@ page language="java" contentType="text/html" %>

<html>

<body bgcolor="white">

<jsp:useBean id="clock" class="java.util.Date" >

The current time at the server is:

Date: <jsp:getProperty name="clock" property="date" />

Month: <jsp:getProperty name="clock" property="month" />

Year: <jsp:getProperty name="clock" property="year" />

Hours: <jsp:getProperty name="clock" property="hours" />

Minutes: <jsp:getProperty name="clock" property="minutes" />

</body>

</html>

The syntax error here is almost the same as the "unterminated tag" in Example, but now it's the

<jsp:useBean> action element that's not terminated properly (it's missing the closing slash). The message

reported by Tomcat in this case is:

D:\ch7\error2.jsp(16,0) useBean tag must begin and end in the same physical file

Scripting Syntax Errors:

syntax errors in scripting elements result in error messages that are much harder to interpret. This is because of the way the JSP container deals with scripting code when it converts a JSP page into a servlet. The container reads the JSP page and generates servlet code by replacing all JSP directives and actions with code that produces the appropriate result. To do this, it needs to analyze these types of elements in detail. If there's a
syntax error in a directive or action element, it can easily tell which element is incorrect (as you saw in the previous section). Scripting elements, on the other hand, are more or less used as-is in the generated servlet code.

A syntax error in scripting code is not discovered when the JSP page is read, but instead when the generated servlet is compiled. The compiler reports an error in terms of its location in the generated servlet code (as opposed to the location in the JSP page), with messages that don't always make sense to a JSP page author. Let's look at some examples to illustrate this.

Missing End Brace (error4.jsp)

```
<%@ page language="java" contentType="text/html" %>

<html>

<body bgcolor="white">

<jsp:useBean id="clock" class="java.util.Date" />

<% if (clock.getHours( ) < 12) { %>

Good morning!

<% } else if (clock.getHours( ) < 17) { %>

Good day!

<% } else { %>

Good evening!

</body>

</html>
```

Debugging a JSP-Based Application

After you have fixed all syntax errors, pat yourself on the back and enjoy the moment. If the application is more than a trivial example, however, this moment will probably be short-lived: you will likely find that one or more things still don't work as you expected. Logic errors, such as not taking care of all possible input combinations, can easily slip into an application during development. Finding and correcting this type of problem is called debugging.

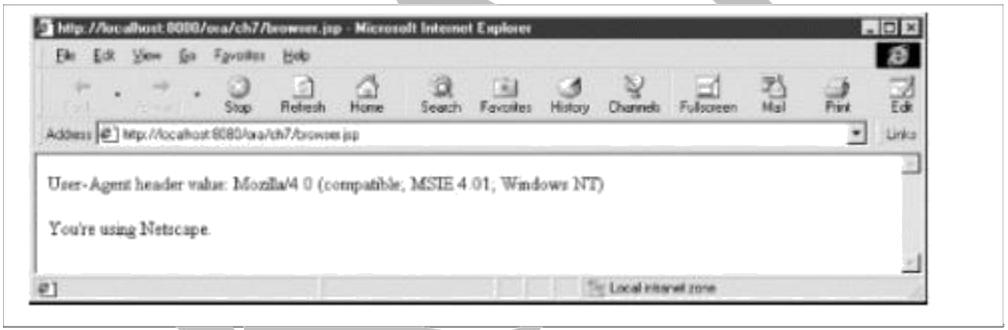
For applications developed in compiled languages such as Java, C, or C++, a tool called a debugger is often used in this phase. A debugger steps through the program line by line or runs until it reaches a break point that you have defined, and lets you inspect the values of all variables in the program. With careful analysis of the program flow in runtime, you can discover why it works the way it does, and not the way you want it to.

There are debuggers for JSP as well, such as IBM's Visual Age for Java. This product lets you debug a JSP page exactly the same way as you would a program written in a more traditional programming language.

Testing Header Values in the Wrong Order (browser.jsp)

```
<%@ page language="java" contentType="text/html" %>

<html>
```

```
<body bgcolor="white">
<% if (request.getHeader("User-Agent").indexOf("Mozilla") != -1) { %>
You're using Netscape.
<%
} else
if (request.getHeader("User-Agent").indexOf("MSIE") != 1) {
%>
You're using Internet Explorer.
<% } else { %>
You're using a browser I don't know about.
<% } %>
</body>
</html>
```



Dealing with Runtime Errors

Eventually, your application will work as you like. But things can still go wrong due to problems with external systems that your application depends on, such as a database. And even though you have tested and debugged your application, there may be runtime conditions that you didn't anticipate. Well-behaved components such as JavaBeans or JSP actions (standard and custom) deal with expected error conditions in a graceful manner. For instance, the UserInfo bean used in a valid attribute that is false unless all properties are set to valid values. Your JSP page can then test the property value and present the user with an appropriate message.

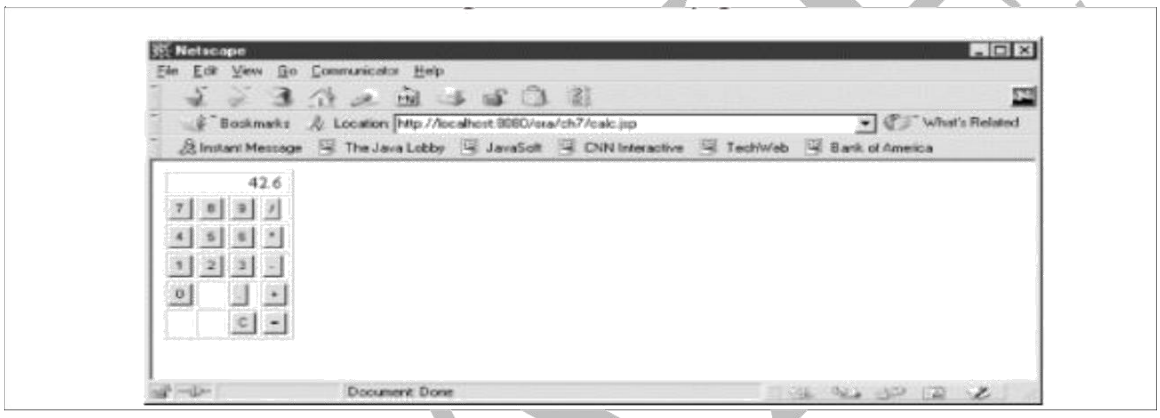
Page with an Error Page Definition (calc.jsp)

```
<%@ page language="java" contentType="text/html" %>
```

```
<%@ page errorPage="errorpage.jsp?debug=log" %>
<% request.setAttribute("sourcePage", request.getRequestURI( ));
%> <html>
<body bgcolor="white">
<jsp:useBean id="calc"
class="com.ora.jsp.beans.calc.CalcBean"> <jsp:setProperty
name="calc" property="*" /> </jsp:useBean>
<%-- Calculate the new numbers and state info --%>
<% String currentNumber = calc.getCurrentNumber( );
%> <form action="calc.jsp" method="post"> <table
border=1>
<tr>
<td colspan="4" align="right">
<%= currentNumber.equals("") ? " " :
currentNumber %>
<input type="hidden" name="currentNumber"
value="<%= currentNumber %>">
<input type="hidden" name="previousNumber"
value="<%= calc.getPreviousNumber( ) %>">
<input type="hidden" name="currentOperation"
value="<%= calc.getCurrentOperation( ) %>">
<input type="hidden"
name="previousOperation" value="<%=
calc.getPreviousOperation( ) %>"> <input
type="hidden" name="reset" value="<%=
calc.getReset( ) %>"> </td>
```

```
</tr>
<tr>
<td><input type="submit" name="digit" value=" 7 "></td>
<td><input type="submit" name="digit" value=" 8 "></td>
<td><input type="submit" name="digit" value=" 9 "></td>
<td><input type="submit" name="oper" value=" / "></td>
</tr>
<tr>
<td><input type="submit" name="digit" value=" 4 "></td>
<td><input type="submit" name="digit" value=" 5 "></td>
<td><input type="submit" name="digit" value=" 6 "></td>
<td><input type="submit" name="oper" value=" * "></td>
</tr>
<tr>
<td><input type="submit" name="digit" value=" 1 "></td>
<td><input type="submit" name="digit" value=" 2 "></td>
<td><input type="submit" name="digit" value=" 3 "></td>
<td><input type="submit" name="oper" value=" - "></td>
</tr>
<tr>
<td><input type="submit" name="digit" value=" 0
"></td> <td> </td>
<td><input type="submit" name="dot" value=" . "></td>
<td><input type="submit" name="oper" value=" + "></td>
</tr>
<tr>
```

```
<td> </td>

<td> </td>

<td><input type="submit" name="clear" value=" C "></td>

<td><input type="submit" name="oper" value=" = "></td>

</table>

</form>

</body>

</html>
```



Sharing Data Between JSP Pages, Requests, and Users

Any real application consists of more than a single page, and multiple pages often need access to the same information and server-side resources. When multiple pages are used to process the same request, for instance one page that retrieves the data the user asked for and another that displays it, there must be a way to pass data from one page to another. In an application in which the user is asked to provide information in multiple steps, such as an online shopping application, there must be a way to collect the information received with each request and get access to the complete set when the user is ready. Other information and resources need to be shared among multiple pages, requests, and all users. Examples are information about currently logged-in users, database connection pool objects, and cache objects to avoid frequent database lookups.

Passing Control and Data Between Pages

Using different JSP pages as Controller and View means that more than one page is used to process a request. To make this happen, you need to 1. Pass be able to do two things: control from one page to another.

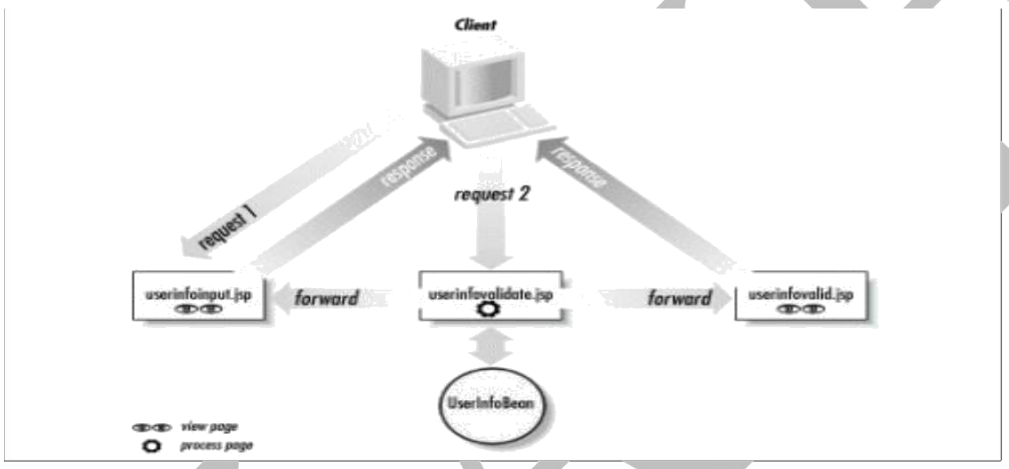
2. Pass data from one page to another.

In this section, we look at a concrete example of how to separate the different aspects of an application and how JSP supports the two requirements above. Let's revisit the User Info example developed in Chapter 5, to describe how the different aspects of an application can be separated. In this example, the business logic piece is trivial. However, it sets the stage for more advanced application examples in the next section and the remaining We can categorize the different aspects of the User Info example like this:

· Display the form for user input (presentation).

· Validate the input (request processing and business logic).

· Display the result of the validation (presentation).



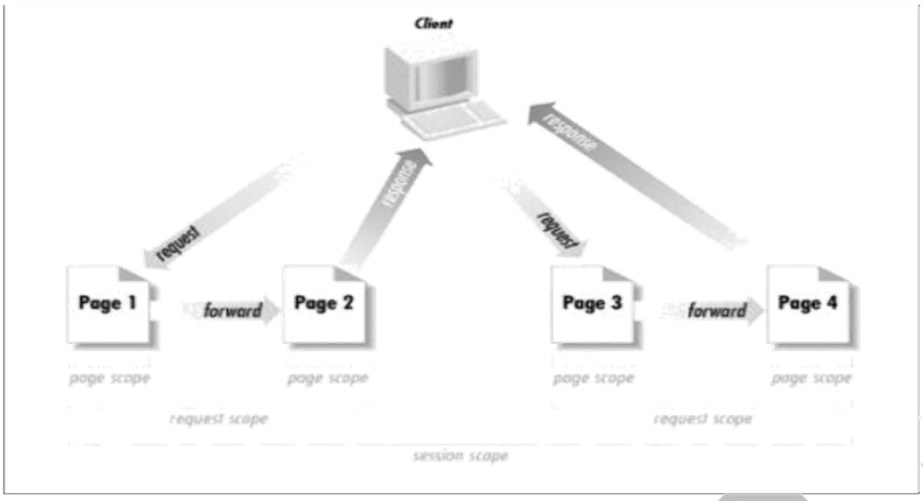
Passing Data from One Page to Another

JSP provides different *scopes* for sharing data objects between pages, requests, and users. The scope defines for how long the object is available and whether it's available only to one user or to all application users. The following scopes are defined:
· Page

· Request

· Session

· Application

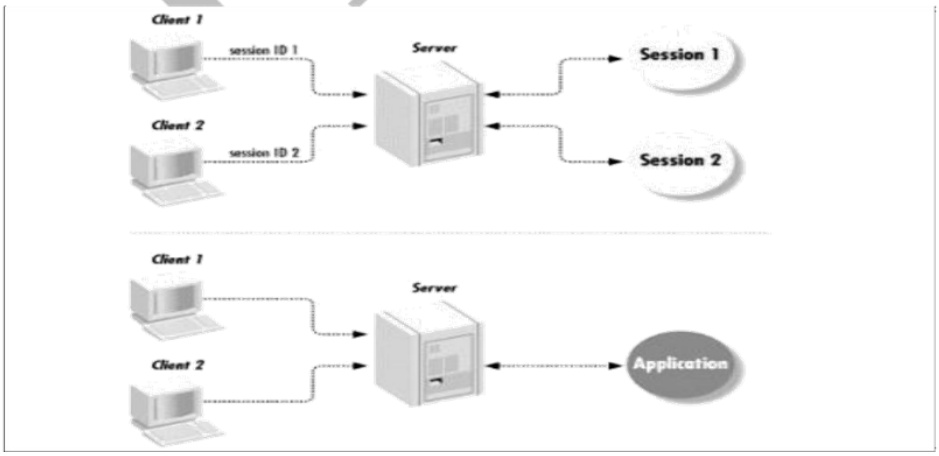


Sharing Session and Application Data:

HTTP is a stateless, request-response protocol. This means that the browser sends a request for a web resource, and the web server processes the request and returns a response. The server then forgets this transaction ever happened. So when the same browser sends a new request, the web server has no idea that this request is related to the previous one. This is fine if you're dealing with static files, but it's a problem in an interactive web application. In a travel agency application, for instance, it's important to remember the dates and destination entered to book the flight so the customer doesn't have to enter the same information again when it's time to make hotel and rental car reservations.

The way to solve this problem is to let the server send a piece of information to the browser that the browser then includes in all subsequent requests. This piece of information, called a *session ID*, is used by the server to recognize a set of requests from the same browser as related: in other words, as part of the same *session*.

A session starts when the browser makes the first request for a JSP page in a particular application. The session can be ended explicitly by the application, or the JSP container can end it after a period of user inactivity (the default value is typically 30 minutes after the last request).



URL Rewriting:

As I mentioned earlier, the session ID needed to keep track of requests within the same session can be transferred between the server and the browser in a number of different ways. One way is to encode it in the URLs created by the JSP pages, called *URL rewriting* . This approach works even if the browser doesn't support cookies (perhaps because the user has disabled them). A URL with a session ID looks like this:

counter3.jsp;jsessionid=be8d691ddb4128be093fdbde4d5be54e00

When the user clicks on a link with an encoded URL, the server extracts the session ID from the request URI and associates the request with the correct session. The JSP page can then access the session data in the same fashion as when cookies are used to keep track of the session ID, so you don't have to worry about how it's handled. What you do need to do, however, is to call a method that lets the JSP container encode the URL when needed. To see how it's done, let's create two pages that reference each other using a regular HTML link. A CounterBean in the session scope is used to increment a counter for each page.

Using Custom Actions

A custom action is just like the standard actions we've used so far. It has a start tag, which may contain attributes, and an end tag. It can also have a body. Here's what a custom action looks like:

<ora:incrementCounter scope="session"/>

The JSP specification defines how the standard set of actions can be extended with custom actions developed by Java programmers in the team or by a third party. A custom action is used in a JSP page in exactly the same way as the standard JSP actions you have seen in previous examples, such as <jsp:getProperty>.

This makes them easier to use than beans with methods that must be invoked with scripting code, since you don't have to worry about missing braces and semicolons and other syntax details. A custom action can do pretty much anything: it has access to all information about the request and can add content to the response body as well as set response headers.

Page with Counter Custom Actions (counter4.jsp)

```
<%@ page language="java" contentType="text/html" %>

<%@ taglib uri="/orataglib" prefix="ora" %>

<html>
```

```
<head>
<title>Counter page 1</title>
</head>
<body bgcolor="white">
<ora:incrementCounter scope="session"/>
<ora:incrementCounter scope="application"/>
<h1>Counter page 1</h1>
This page has been visited <b>
<ora:showCounter scope="session"/>
</b> times by the current user in the current session, and <b>
<ora:showCounter scope="application"/>
</b> times by all users since the counter was reset.
<p>
To see that a unique counter is maintained per page,
take a look at
<a href="<ora:encodeURL url="counter5.jsp" />">Counter page 2</a>.
</body>
</html>
```

Online Shopping

An online shopping site Besides showing you how the session and application scopes can be used effectively in a larger application, this example also introduces many other useful tools. You'll see a number of generic custom actions you can use in your own applications, and learn how to use the java.text.NumberFormat class to format numbers. The application consists of three pages. The main page lists all available products.

Each product is linked to a product description page, where the product can be added to the shopping cart. A product is added to the shopping cart by a request processing page. The main page with the product list is then displayed again, but now with the current contents of the shopping cart

Memory Usage Considerations

All objects you save in the application and session scopes take up memory in the server process. It's easy to calculate how much memory is used for application objects since you have full control over the number of objects you place there. But the total number of objects in the session scope depends on the number of concurrent sessions, so in addition to the size of each object, you also need to know how many concurrent users you have and how long a session lasts. Let's look at an example. The CartBean used in this chapter is small. It stores only references to ProductBean instances, not copies of the beans. An object reference in Java is 8 bytes, so with three products in the cart we need 24 bytes. The java.util.Vector object used to hold the references adds some overhead, say 32 bytes. All in all, we need 56 bytes per shopping cart bean with three products.

Here are some things you can do to keep the memory requirements under control:

· Place only those objects that really need to be unique for each session in the session scope. In the shopping cart example, for instance, each cart contains references only to the shared product beans, and the catalog bean is shared by all users.

· Set the timeout period for sessions to a lower value than the default. If you know it's rare that your users leave the site for 30 minutes and then return, use a shorter period. You can change the timeout for all sessions in an application through the application's Deployment Descriptor or call session.setMax-InactiveInterval ( ) to change it for an individual session.

Possible Question

Part B (2Marks)

1) Define implicit object
2) How to declare variable and methods?
3) What is mean by conditional processing?
4) What is mean by Error Handling?
5) What is mean by Expression?
6) What is mean sharing data?
7) What is mean by Database Access?
8) What is mean by Debugging?
9) What is mean by Request?
10) Define displaying values

Part C (6Marks)

1) Explain in details about implicit object with example
2) Explain in detail about conditional processing
3) How to Display values in JSP?
4) How to use an expression to set an attributes?
5) How to declare variable&Methods with example?
6) Explain in detail about Error handling with example
7) Explain in detail about error handling and debugging with example
8) How to sharing data between JSP pages
9) Explain in detail about request and user database access
10) Explain in detail about methods with example

KARPAGAM ACADEMY OF HIGHER EDUCATION
PART - A (ONLINE EXAMINATION)

MULTIPLE CHOICE QUESTIONS (Each question carries one mark)

SUBJECT: INTERNET TECHNOLOGY
SUB.CODE: 17ITU404A CLASS : 17ITU404A
UNIT-IV

QUESTIONS	OPTION1	OPTION2	OPTION3	OPTION4	ANSWER
JSP implicits objects are created during the _____ of JSP to the servlet.	Application phase	Translation phase	Modulation phase	object phase	Translation phase
JSP expansion_____	Java Server Page	Java Scripting Phase	Java Server Phase	Java Scripting Phase	Java Server Page
There are _____ types of implicit objects available in the container.	7	8	9	10	9
Which is not a implicit object?	out	request	accept	config	accept
The _____ object is an instance of java.servlet.http.HttpServletRequest .	out	request	response	config	request
_____ object is used to get,set and remove attributes to session scope and also used to get session information.	response	config	out	session	session
_____ is the object of javax.servlet.jsp.jspwriter class.	out	request	response	config	out
_____ is an instance of class which implements HttpServletResponse interface.	out	application	request	response	response
_____ is used to get,set and remove the attributes from a particular scope.	Pagecontext	Page	Exception	Session	Pagecontext
_____ implicit variable holds the currently executed servlet object for the corresponding jsp.	Pagecontext	Page	Exception	Session	Page
_____ is the implicit object of the throwable class.	Pagecontext	Page	Exception	Session	Exception
bean is a class with a _____ that doesnt take an argument.	constructor	object	class	javabean	constructor
The accessor method for reading a property value is known as _____	getter method	accessor method	accessor method	constructor method	getter method
The accessor method for writing a property value is known as _____	getter method	accessor method	accessor method	constructor method	getter method

The implicit object _____ is used to display diffrent messages.	request	response	accept	reject	request
The requset object's _____ method is used to get value of a specific request header.	getHeader()	getFunction()	getConstructor()	getPointer()	getHeader()
Another common use of _____ is to loop over an array	Servlets	Scriptlets	Server	none	Scriptlets
An expression starts with _____ and ends with %>.	%	<%	<%=	<	<%=
An example for a expression	<userInfo.getUserName()>	<%userInfo.getUserName()%>	<=userInfo.getUserName()=>	<=userInfo.getUserName()=>	<%userInfo.getUserName()%>
To organize all diffrent classes,Java provides the notation of a _____	Package	Packets	Groups	none	Package
A _____ is a group of related classes.	Package	Packets	Groups	none	Package
Java to look for classes with _____ names in the named package.	Short	Long	minimum	maximum	Short
Displaying Request Info is also known as _____	disreq.jsp	reqdis.jsp	inforeq.jsp	reqinfo.jsp	reqinfo.jsp
The attributes are set to literal _____ values.	string	array	constant	char	string
The value must be calculated when the JSP page is ____	compilied	requested	responsed	none	requested
We can use a JSP expression as an _____ value.	object	class	constructor	attribute	attribute
When the JSP expression is used as an attribute then it will be called as _	request time attribute value	compile time attribute value	response time attribute value	none of these	request time attribute value
The request time attribute is evaluated when the page is _____ .	compiled	requested	responsed	none	requested
JSP scripting elements are scriptlets, _____ and declaration	expressions	attributes	methods	none	expressions
_____ element id used to declare java variables and methods in a JSP page.	scriptlets	expressions	declaration	done	declaration
Java variables are can be declared either within a method or _____ of all methods.	inside the body	outside the body	between the classes	outside the method	outside the body
A variable declared outside the body of all methos is called an _____.	local variable	private variable	public variable	instance variable	instance variable
A variable declared within the body of a method is called _____	local variable	private variable	public variable	instance variable	public variable
A local variable can be accessed only from the method where it is ___	declared	initialized	compiled	none	declared
Each _____ executes the main method in the JSP object.	object	class	variable	thread	thread

When more than one thread is execute in the same code then we have to make sure the code is __.	execute	thread safe	correct	none	thread safe
_____ and thread safe code stratergies are best left to programmers	Multithreading	Multi processing	Multi tasking	none	Multithreading
If one thread changes the value of the _____ the value is changed for all threads.	local variable	private variable	public variable	instance variable	instance variable
If the value of the _____ is changed ,it will not affect the other threads.	local variable	private variable	public variable	instance variable	local variable
Servlet has _____ methods.	1	2	3	4	2
Servlet methods are init() and _____	destroy()	execute()	run()	none	destroy()
When init() is used it will allow the servlet to _____ instance variables.	initialize	declare	destroy	none	initialize
When destroy() is used it will allow the servlet to _____ when its shut down.	initialize	declare	destroy	clean	clean
Finding and correcting a type of problem is called _____.	detecting	correcting	debugging	none	debugging
Any application consists of more than one page often need access to the same information and_____ side resourses.	client	server	user	admin	server
In an application in which the _____ is asked to provide information in multiple steps	client	server	user	admin	user
Using diffrent JSP pages as ____ and view means that more than one page is used to process a request.	Constructor	Controller	Connector	none	Controller
Passing control and _____ from one page to another is supports in JSP pages.	data	object	class	none	data
The buisness logic piece is _____	thrice	trivial	both a&b	none	trivial
JSP provides diffrent _____ for sharing data objects between pages,requests& users.	string	char	object	scope	scope
Which is not a scope?	pagecontent	request	session	application	pagecontent
_____ is a stateless, request-response protocol.	HTTP	SNMP	HTML	none	HTTP
The server send a piece of information to the browser that the browser includes in all subsequent requests, this piece of information called _____.	page ID	application ID	session ID	request ID	session ID
To encode it in the URLs created by the JSP pages, called _____.	URL writing	URL rewriting	URL scanning	URL reading	URL scanning
A CounterBean in the session scope is used to _____ a counter for each page.	increment	decrement	mulitplies	divides	increment

An _____ site besides showing you how the session & application scopes can be used effectively.	account managing	image processing	online shopping	both a & b	image processing
The application consist of _____ pages.The main page lists all available products.	1	2	3	4	3
A _____ is added to the shopping cart by a req processing page.	items	modules	product	none	product
An example of a way to end the session explicity in memory requirements, _____.	login function	logout function	both a&b	none	logout function
All objects saved in the application and session scopes takeup _____ in the server process.	memory	space	data	both a&b	memory

UNIT-V

SYLLABUS

Java beans: Java Beans Fundamentals, JAR Files, Introspection, Developing a simple Bean, Connecting to DB.

JavaBeans

JavaBeans is an object-oriented programming interface from Sun Microsystems that lets you build re-useable applications or program building blocks called components that can be deployed in a network on any major operating system platform. Like Java applets, JavaBeans components (or "Beans") can be used to give World Wide Web pages (or other applications) interactive capabilities such as computing interest rates or varying page content based on user or browser characteristics.

A Java Bean is a java class that should follow following conventions:

- It should have a no-arg constructor.
- It should be Serializable.
- It should provide methods to set and get the values of the properties, known as getter and setter methods.

Why use Java Bean?

According to Java white paper, it is a reusable software component. A bean encapsulates many objects into one object, so we can access this object from multiple places. Moreover, it provides the easy maintenance.

Simple example of java bean class

```
//Employee.java
package mypack;
public class Employee implements
java.io.Serializable{ private int id;
private String name;
public Employee(){}
public void setId(int id){this.id=id;}
public int getId(){return id;}
public void setName(String name){this.name=name;}
public String getName(){return name;} }
```

How to access the java bean class?

To access the java bean class, we should use getter and setter methods.

```
package mypack;
public class Test{
public static void main(String args[]){
Employee e=new Employee();//object is created
e.setName("Arjun");//setting value to the object
System.out.println(e.getName());
}}
```

Java Beans Fundamentals

Java Enterprise Edition (Java EE) has a powerful facility dedicated to expressing the business logic of an application and for accessing a database using a JavaBeans-like concept. That facility is *Enterprise JavaBeans*, known as EJBs for short.

In this, we'll begin exploring the world of EJBs, which is a very important capability of the Java EE platform. EJBs provide infrastructure for developing and deploying mission-critical, enterprise applications. We'll first look at some EJB fundamentals, and then focus on one type of EJB: the session bean.

In this article, you will learn the following:

- The benefits of using EJBs
- The three kinds of EJBs: session, entity, and message-driven beans
- The makeup of session beans
- How to develop session beans
- Differences between stateful and stateless session beans

Understanding EJBs

Application architectures often consist of several tiers that each has its own responsibilities. One such architecture that consists of three tiers is illustrated in the Unified Modeling Language (UML) diagram shown in Figure 1

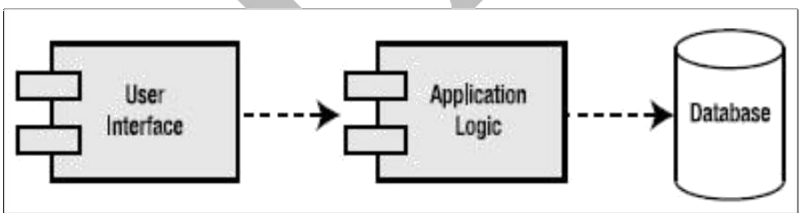


Figure 1. The classic model of a multitiered, or layered, architecture

The two elements on the left side of the diagram in Figure 1 are called *components* in the UML notation. Components represent software modules. The diagram describes what is called a *multitiered*, or *layered*, architecture. Multitiered architectures have many advantages, not the least of which is the ability to change any one of the layers without affecting all of the other layers. This is in contrast to a *single-tier* architecture, within which all aspects of the program

design coexist in a single element. Changes or actions that affect one portion of the single-tier element also potentially affect the other members of that element.

Consider the three-layer architecture shown in Figure 1, consisting of user interface, application logic, and database layers. If the database layer is changed, only the application logic layer is affected. The application logic layer shields the user interface layer from changes to the database layer. This facilitates ongoing maintenance of the application and also increases the application's ability to incorporate new technologies in its layers.These layers provide an excellent model of how EJBs fit into your overall program design. EJBs provide an application logic layer and a JavaBeans-like abstraction of the database layer. The application logic layer is also known as the *middle tier*.

Why use EJBs?

Not too long ago, when system developers wanted to create an enterprise application, they would often start by "rolling their own" (or purchasing a proprietary) application server to support the functionality of the application logic layer. Some of the features of an application server include the following:

- **Client communication:** The client, which is often a user interface, must be able to call the methods of objects on the application server via agreed-upon protocols.
- **Session state management:** You'll recall our discussions on this topic in the context of JSP (JavaServer Pages) and servlet development back in Chapter 6.
- **Transaction management:** Some operations, for example, when updating data, must occur as a unit of work. If one update fails, they all should fail.
- **Database connection management:** An application server must connect to a database, often using pools of database connections for optimizing resources.
- **User authentication and role-based authorization:** Users of an application must often log in for security purposes. The functionality of an application to which a user is allowed access is often based on the role associated with a user ID.
- **Asynchronous messaging:** Applications often need to communicate with other systems in an asynchronous manner; that is, without waiting for the other system to respond. This requires an underlying messaging system that provides guaranteed delivery of these asynchronous messages.
- **Application server administration:** Application servers must be administered. For example, they need to be monitored and tuned.

The EJB specification

The EJB specification defines a common architecture, which has prompted several vendors to build application servers that comply with this specification. Now developers can get off-the-shelf application servers that comply with a common standard, benefiting from the competition (in areas such as price, features, and performance) among those vendors.

The three kinds of EJBs

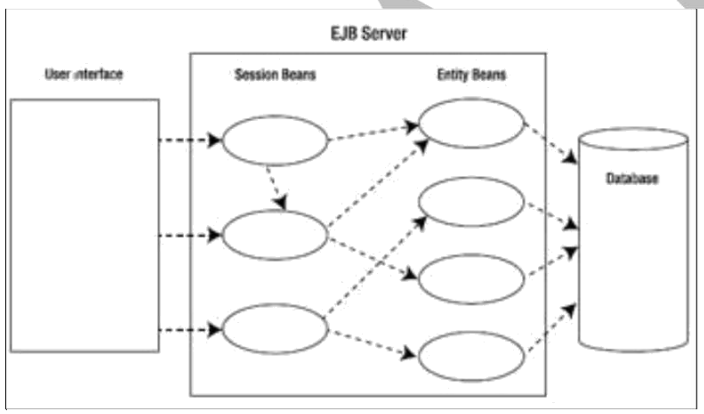
There are actually three kinds of EJBs: session beans, entity beans, and message-driven beans. Here, we will present a brief introduction to each type of bean. The balance of this article will then focus on session beans.

Session beans

One way to think about the application logic layer (middle tier) in the sample architecture shown in Figure 1 is as a set of objects that, together, implement the business logic of an application. Session beans are the construct in EJBs designed for this purpose. As shown in Figure 2, there may be multiple session beans in an application. Each handles a subset of the application's business logic.

There are two types of session beans, which are defined by their use in a client interaction:

- **Stateless:** These beans do not declare any instance (class-level) variables, so that the methods contained within can act only on any local parameters. There is no way to maintain state across method calls.
- **Stateful:** These beans can hold client state across method invocations. This is possible with the use of instance variables declared in the class definition. The client will then set the values for these variables and use these values in other method calls.



Entity beans

Before object orientation became popular, programs were usually written in procedural languages and often employed relational databases to hold the data. Because of the strengths and maturity of relational database technology, it is now often advantageous to develop object-oriented applications that use relational databases. The problem with this approach is that there is an inherent difference between object-oriented and relational database technologies, making it less than natural for them to coexist in one application. The use of entity beans is one way to get the best of both of these worlds, for the following reasons:

- Entity beans are objects, and they can be designed using object-oriented principles and used in applications as objects.
- The data in these entity bean objects are persisted in some data store, usually relational databases. All of the benefits of relational technologies—including maturity of products, speed, reliability, ability to recover, and ease of querying—can be leveraged.

Message-driven beans

- When an EJB-based application needs to receive asynchronous messages from other systems, it can leverage the power and convenience of message-driven beans. Asynchronous messages between systems can be analogous to the events that are fired from a UI component to an event handler in the same JVM.

JAR file:

A **JAR** (**J**ava **AR**chive) is a package file format typically used to aggregate many Java class files and associated metadata and resources (text, images, etc.) into one file for distribution.

JAR files are archive files that include a Java-specific manifest file. They are built on the ZIP format and typically have a .jar file extension.

JAR file allows you to efficiently deploy a set of classes and their associated resources. JAR file makes it much easier to deliver, install, and download. It is compressed.

Java Archive File • The files of a JavaBean application are compressed and grouped as JAR files to reduce the size and the download time of the files. • The syntax to create a JAR file from the command prompt is: • jar • The file_names

JAR files are packaged with the ZIP file format, so you can use them for tasks such as lossless data compression, archiving, decompression, and archive unpacking. These tasks are among the most common uses of JAR files, and you can realize many JAR file benefits using only these basic features.

Even if you want to take advantage of advanced functionality provided by the JAR file format such as electronic signing, you'll first need to become familiar with the fundamental operations.

To perform basic tasks with JAR files, you use the Java Archive Tool provided as part of the Java Development Kit (JDK). Because the Java Archive tool is invoked by using the jar command, this tutorial refers to it as 'the Jar tool'.

As a synopsis and preview of some of the topics to be covered in this section, the following table summarizes common JAR file operations:

Common JAR file operations

Operation	Command

To create a JAR file	jar cf *jar-file input-file(s)*
To view the contents of a JAR file	jar tf *jar-file*
To extract the contents of a JAR file	jar xf *jar-file*
To extract specific files from a JAR file	jar xf *jar-file archived-file(s)*
To run an application packaged as a JAR file (requires the Main-class manifest header)	java -jar *app.jar*
To invoke an applet packaged as a JAR file	<applet code=*AppletClassName.class* archive="*JarFileName.jar*" width=*width* height=*height*> </applet>

Let us see how to create a .jar file and related commands which help us to work with .jar files

1. **Create a JAR file:** To create a .jar file , we can use jar cf command in the following

   way: jar cf jarfilename inputfiles

   Here, cf represents create the file. For example , assuming our package pack is available in C:\directory , to convert it into a jar file into the pack.jar , we can give the command as:

   C:\> jar cf pack.jar pack Now

   , pack.jar file is created

2. **Viewing a JAR file:** To view the contents of .jar files, we can use the command

   as: jar tf jarfilename

   Here , tf represents table view of file contents. For example, to view the contents of our pack.jar file , we can give the command:

   C:/> jar tf pack.jar

   Now , the contents of pack.jar are displayed as:

   META-INF/

   META-

   INF/MANIFEST.MF pack/

```
pack/class1.class
pack/class2.class
..
..
```

where class1 , class2 etc are the classes in the package pack. The first two entries represent that there is a manifest file created and added to pack.jar. The third entry represents the sub-directory with the name pack and the last two represent the files name in the directory pack.

When we create .jar files , it automatically receives the default manifest file. There can be only one manifest file in an archive , and it always has the pathname.

```
META-INF/MANIFEST.MF
```

This manifest file is useful to specify the information about other files which are packaged.

3. **Extracting a JAR file:** To extract the files from a .jar file , we can

```
use: jar xf jarfilename
```

Here, xf represents extract files from the jar files. For example , to extract the contents of our pack.jar file, we can write:

```
C:\> jar xf pack.jar
```

This will create the following directories in C:\

```
META-INF
pack // in this directory , we can see class1.class and class2.class.
```

4. **Updating a JAR File** The Jar tool provides a 'u' option which you can use to update the contents of an existing JAR file by modifying its manifest or by adding files. The basic command for adding files has this format:

```
jar uf jar-file input-file(s)
```

here uf represent update jar file. For example , to update the contents of our pack.jar file, we can write:

```
C:\>jar uf pack.jar
```

5. **Running a JAR file:** In order to run an application packaged as a JAR file (requires the Main-class manifest header) , following command can be used:

```
C:\>java -jar pack.jar
```

The various options that you can specify while creating a JAR file are:

c: Indicates the new JAR file is created.

f: Indicates that the first file in the file_names list is the name of the JAR file.

m: Indicates that the second file in the file_names list is the name of the manifest file.

t: Indicates that all the files and resources in the JAR file are to be displayed in a tabular format.

v: Indicates that the JAR file should generate a verbose output.

x: Indicates that the files and resources of a JAR file are to be extracted.

o: Indicates that the JAR file should not be compressed.

Introspection:

1. Introspection can be defined as the technique of obtaining information about bean properties, events and methods.

2. Basically introspection means analysis of bean capabilities.

3. Introspection is the automatic process by which a builder tool finds out which properties, methods, and events a bean supports.

4. Introspection describes how methods, properties, and events are discovered in the beans that you write.

5. This process controls the publishing and discovery of bean operations and properties Without introspection, the JavaBeans technology could not operate

6. Basically introspection means analysis of bean capabilities There are two ways in which the developer of a Bean can indicate which of its properties, events, and methods should be exposed by an builder tool.

The first method, simple naming conventions are used. These allow the introspection mechanisms to infer information about a Bean.

In the second way, an additional class is provided that explicitly supplies this

information. **Introspection**

Introspection is the automatic process of analyzing a bean's design patterns to reveal the bean's

properties, events, and methods. This process controls the publishing and discovery of bean operations and properties. This lesson explains the purpose of introspection, introduces the Introspection API, and gives an example of introspection code.

Purpose of Introspection

A growing number of Java object repository sites exist on the Internet in answer to the demand for centralized deployment of applets, classes, and source code in general. Any developer who has spent time hunting through these sites for licensable Java code to incorporate into a program has undoubtedly struggled with issues of how to quickly and cleanly integrate code from one particular source into an application.

The way in which introspection is implemented provides great advantages, including:

1. *Portability* - Everything is done in the Java platform, so you can write components once, reuse them everywhere. There are no extra specification files that need to be maintained independently from your component code. There are no platform-specific issues to contend with. Your component is not tied to one component model or one proprietary platform. You get all the advantages of the evolving Java APIs, while maintaining the portability of your components.
2. *Reuse* - By following the JavaBeans design conventions, implementing the appropriate interfaces, and extending the appropriate classes, you provide your component with reuse potential that possibly exceeds your expectations.

Introspection API

The JavaBeans API architecture supplies a set of classes and interfaces to provide introspection.

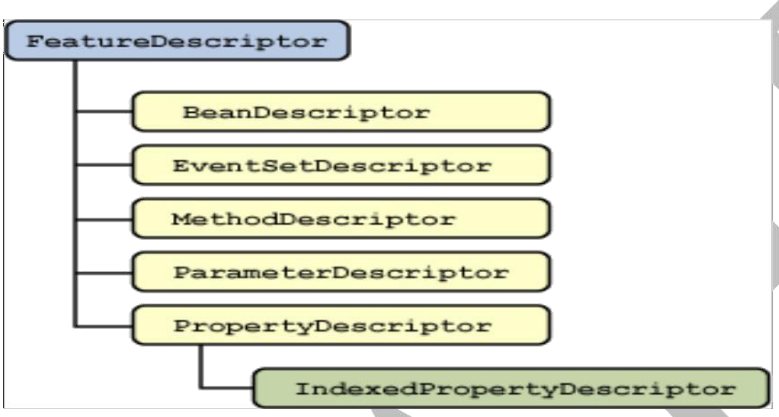
The BeanInfo(in the API reference documentation) interface of the java.beans package defines a set of methods that allow bean implementors to provide explicit information about their beans. By specifying BeanInfo for a bean component, a developer can hide methods, specify an icon for the toolbox, provide descriptive names for properties, define which properties are bound properties, and much more.

The getBeanInfo(beanName)(in the API reference documentation) of the Introspector(in the API reference documentation) class can be used by builder tools and other automated environments to provide detailed information about a bean. The getBeanInfo method relies on the naming

conventions for the bean's properties, events, and methods. A call to getBeanInfo results in the introspection process analyzing the bean's classes and superclasses.

The Introspector class provides descriptor classes with information about properties, events, and methods of a bean. Methods of this class locate any descriptor information that has been explicitly supplied by the developer through BeanInfo classes. Then the Introspector class applies the naming conventions to determine what properties the bean has, the events to which it can listen, and those which it can send.

The following figure represents a hierarchy of the FeatureDescriptor classes:



Each class represented in this group describes a particular attribute of the bean. For example, the isBound method of the PropertyDescriptorclass indicates whether a PropertyChangeEventevent is fired when the value of this property changes.

Developing a Simple Bean

This section presents an example that shows how to develop a simple Bean and connect it to other components via BDK.

Following steps to be adopted to create a new Bean:

- Create the Java source file.
- Compile the source file.
- Create a manifest file.
- Generate a JAR file.
- Start the BDK.
- Test.

This example creates a simple bean button

```
C:\BDK\beanbox>run
C:\BDK\beanbox>if "Windows_NT" == "Windows_NT" setlocal
C:\BDK\beanbox>set CLASSPATH=classes;..\lib\methodtracer.jar;..\infobus.jar
C:\BDK\beanbox>java sun.beanbox.BeanBoxFrame
```

Diagram 25.9 Type run in the command prompt

Working of the Code:

To create a bean, type the above code and save it as *SimpleBean.java*. Comiple the above code using the javac.exe compiler as under:

javac SimpleBean.java

Create a manifest file with a **.mf** extension as under. The Manifest file is a text file which contains the name of the class file.

SimpleBean.mf
Name: SimpleBean.class

Java-Bean: True

Creating a jar (Java Archive) file

As you are aware that *javac.exe* creates a separate **.class** file for every class defined in the program. Hence, if a program has five classes, the compiler will create five class file each with their respective name and a **.class** extension

This proved to be drawback as the applet loader makes separate connections for each file while loading the applet. To overcome this problem, the engineers at Java Soft, the *Sun Microsystems subsidiary for Java* introduced the idea fo compressing all the files together which can be unzipped at the applet's client machine when the applet executes.

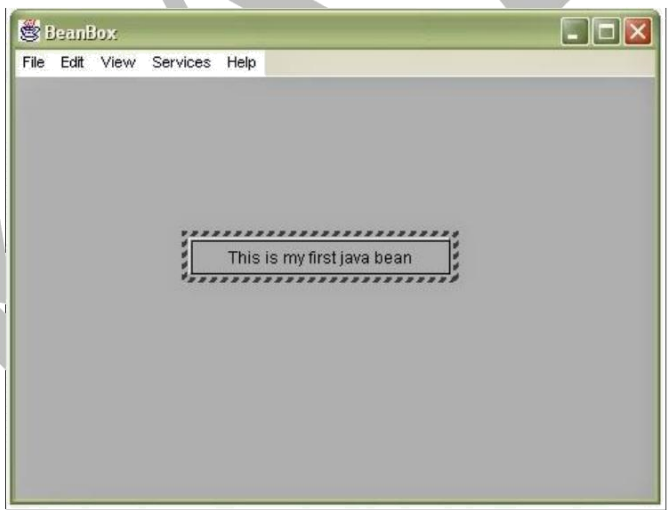
Jar is a zip or compression utility tailored to Java's needs. Pass the names of all the files in the project to the JAR utility, which compresses all of them together and creates a compressed file with a **.jar**extension.

Usage of JAR is as under
jar [-cvfmOM][jar-file name][manifest-file name] list of .class file
Options:

Options:	Description
-c	Creates a new archive.
-t	List table of contents for archive.
-x	Extracts named (or all) files from archive.
-f	Specify archive file name.
-m	Include manifest information from specified manifest file.
-O	Store only: use no Zip compression.
-M	Do not create a manifest file for the entries.

Creating a JAR file for the example code created eariler on.

This will create a jar file:*SimpleBean.jar*. Copy this file in the *jars* directory under the BDK folder and start the Bean Box. The ToolBox of the Bean Box now displays the bean created underline.



Connecting to DB

Creating a Java Bean that issues an SQL statement
You can use the Create a Java™ Bean that executes an SQL statement wizard to create a Java Bean that issues a specific SQL statement.

Before you begin

Prerequisites: To use this wizard, you must create and save an SQL statement in a data development project.

About this task

To create a Java Bean that issues an SQL statement:

Procedure

1. Switch to the Data perspective.
2. In the **SQL Scripts** folder for a data development project in the Data Project Explorer, right-click the SQL statement that you want to use, and select **Generate Java Bean** from the pop-up menu.
3. On the Java Class Specification page of the wizard, complete the following steps.

a. In the **Source Folder** field, type the name of the folder where you want to create the Java Bean or click **Browse** to select the folder.

b. In the **Package** field, type the name of Java package in which you want to create your Java Bean or click **Browse** to select the package. Leave this field empty to create the new class in the default package.

c. In the **Name** field, type a name for the new class. To conform to Java conventions, the class name must start with a capital letter.

d. If the SQL statement returns a result set, specify whether you want to include a parameter in the execute() method to limit the size of the result set. By default, the execute() method retrieves all rows in the result set.

4. On the Specify Runtime Database Connection Information page, specify whether to use a data source connection or a driver manager connection. A data source is defined in an application server that implements Java Database Connectivity (JDBC), and is generally the preferred way for Web applications to connect to a database, because it provides pooled connections. When the application initializes, the server requests a pool of database connections. Each time a Web application requires a database connection, the server provides one from the pool. When the Web application is done with the connection, it releases the connection back to the pool. Because connecting to the database is one of the slowest operations an application can perform, this approach is usually the most efficient. In contrast, for a driver manager connection, each time a Web application requires a database connection, it requests a connection from the database server.

   - **Use DataSource Connection:** Enter the Java Naming and Directory Interface (JNDI) name of the data source, as defined in the server configuration.
   - **Use DriverManager Connection:** Type the fully-qualified class name of the driver in the **Driver Name** field, and the associated JNDI address in the **URL** field. For example, for DB2® enter COM.ibm.db2.jdbc.app.DB2Driver for the driver name and jdbc:db2:SAMPLE for the URL.

   Specify how the Java Bean will provide user authentication. To run the SQL statement, you must supply a user ID and password that are valid for the database. You can specify that the Java Bean will provide a valid user ID and password within its execute() method. This means that the Java Bean will always connect using the same user ID and password. If you do not include the user ID and

password inside the execute() method, the application must provide the user ID and password as input parameters to the execute() method.

- To include user authentication within the method, click **Inside the execute() method**. Type a user ID and password in the appropriate fields to access the database. The initial values are the ones originally used to load the existing database model. The password will be masked when you type it in the field.
- To require the user ID and password to be provided as parameters to the execute()method, click **By the execute() method's caller**

On the final page of the wizard, review the specifications for the new Java Bean, then click **Finish** to complete the wizard.

Results

The wizard creates the Java Bean, along with all the necessary deployment descriptors to build, run, and deploy it with your application.

Possible Question:

Part B (2Marks)

1) What is mean by java bean?
2) Define Introspection
3) What is mean by JAR FILES?
4) Write short notes developing simple java bean
5) What is mean by Datatype
6) How to develop simple bean

Part C (6 Marks)

1) What is mean by introspection? how it works in javabean?
2) What is javabean? write the steps to connect DB to simple bean?
3) What is javabean, how to connect javabean with database?
4) What is javabean, how to create and develop simple bean with example?
5) Explain in detail about Connecting Javabean to DB
6) How to create simple bean,with example?
7) Explain in detail about Introspection
8) How to develop simple bean with example?
9) Explain in Detail about JAR files
10) What is mean by javabean?,Explain In Detail about fundamental concepts of javabeans?

KARPAGAM ACADEMY OF HIGHER EDUCATION

PART - A (ONLINE EXAMINATION)

MULTIPLE CHOICE QUESTIONS (Each question carries one mark)

SUBJECT: INTERNET TECHNOLOGY

SUB.CODE : 17ITU404A CLASS:II B.Sc IT

UNIT-V

QUESTIONS	OPTION1	OPTION2	OPTION3	OPTION4	Answer
Java bean is an ……	Structure oriented	Object oriented	Both a&b	none	Object oriented
Program building blocks are called as……	Components	Elements	Applications	none	Components
Java class should follow …… following convention	Constructor	arg constructor	no-arg constructor	Both a&b	no-arg constructor
EJB stands for ……	Edision java bean	Element java bean	Enterprise java bean	none	Enterprise java bean
EJBs provide infrastructure for developing and deploying ……..	EJB fundamentals	EJB functions	EJB elements	Mission critical	Mission critical
UML stands for ……	Unified Modified Language	User Modified Language	Unified Markup Language	User Markup Language	Unified Modified Language
The Application Logic layer is also known as …….	Upper tier	Middle tier	Lower tier	Bottom tier	Middle tier
Application often need to communicate with other systems is an ……. Manner.	Synchronous	Asynchronous	Authentication	none	Asynchronous
There are ….. kinds of EJB.	1	2	3	4	3
Session beans are ……..	Stateful	Stateless	Both a&b	none	Both a&b

Entity beans are ……	Elements	objects	Attributes	Class	objects
JAR stands for …….	Java ARchitechture	Java ARchive	Java Arc	none	Java ARchive
JAR files are ……..	Archive files	Auto files	Compress files	De-compress files	Archive files
JDK stands for ……	Java Development Kit	Java Details Kit	Java Design Kit	none	Java Development Kit
To create a JAR file …….	jar cf jar-file input-file(s)	jar tf jar-file	jar xf jar-file	jar xf jar-file archived-file(s)	jar cf jar-file input-file(s)
To extract the contents of a JAR file ……..	jar cf jar-file input-file(s)	jar tf jar-file	jar xf jar-file	jar xf jar-file archived-file(s)	jar xf jar-file
To view the content of a JAR file …….	jar cf jar-file input-file(s)	jar tf jar-file	jar xf jar-file	jar xf jar-file archived-file(s)	jar tf jar-file
To extract specific files from a JAR file ……	jar cf jar-file input-file(s)	jar tf jar-file	jar xf jar-file	jar xf jar-file archived-file(s)	jar xf jar-file archived-file(s)
To run an application packaged as a JAR file ……	java –jar app.jar	jar tf jar-file	jar xf jar-file	jar xf jar-file archived-file(s)	java –jar app.jar

To update a JAR file…….	java –jar app.jar	jar tf jar-file	jar uf jar-file input file(s)	jar xf jar-file archived-file(s)	jar uf jar-file input file(s)
……..…… is the automatic process of analyzing a bean's design patterns to reveal the bean's properties ,events,and methods.	Introspection	Extraspection	Abstraction	none	Introspection
Advantages of introspection is ……..	Portability	Reuse	Inheritance	Both a&b	Both a&b
Introspection means analysis Of bean …….	Capabilities	Reusability	inheritance	comparability	Capabilities
…… is the automatic process	Introspection	Extraspection	Abstraction	none	Introspection
which is a java package…….	Beaninfo	infojava	javaifo	none	Beaninfo
API expansion is ……	advanced package interface	Application package interface	application programming interface	Advanced Programming interchange	Application package interface
IsBound Method comes under …… class	Bean descriptor	method descriptor	property descriptor	none	property descriptor
BDK means …….	Button development kit	bean development kit	bean describing kit	none	bean development kit
How many steps to be adapted to create a new bean……	7	3	10	6	6
Manifest file denoted as……..	.mf	.manif	.manifest	.mfexe	.mf
JVM stands for……	java virtual machine	java virtual mechanism	java versibility machine	none	java virtual machine
creates new archive……	–c	-a	-t	-f	–c

List table of contents for archive…….	–c	-a	-t	-f	-t
Extracts named files from archive……	–c	-a	-x	-f	-x
Specify archive file name……	–c	-a	-x	-f	-f
Include manifest information from specified manifest file…….	–m	-a	-x	**-f**	–m
Store only : use no ZIP compression……	–c	-a	-x	-O	-O
Do not create a manifest file for the entries…	–c	-M	-x	-f	-M
By default the execute() method retrieves……	all rows in result set	all columns in result set	limited rows in result set	limited rows in result set	limited rows in result set
JDBC is used to……	connectivity between web application to DB	connectivity between program to DB	connectivity between a pplication to DB	none	connectivity between program to DB
JNDI stands……	Java Naming & Directory Interface	Java Naming & Dictionary Interface	Java Namespace & Directory Interface	Java Naming & Domain Interface	Java Naming & Domain Interface
For user authentication …… method is use	inside the execute()	outside the execute()	bottom the execute()	down the execute()	inside the execute()
On the final page of the wizard, review the specification for the new java bean, then click ----- to complete the wizard.	Complete	finish	final	conculsion	final
The password will be _____ when you type it in the field .	covered	masked	changed	duplicate	duplicate
The data source is defined in an ____	application server	web application	SQL server	database server	SQL server
___ is a zip or compression utility tailored to java's needs.	jar	jar extension	pass	applet	applet
There are _____ advantages in introspection.	2	3	4	6	4

The introspector class provides _____ class.	developer	information	beaninfo	descriptor	descriptor
A call to_____ results in the introspection process analyzing the bean classes and super class	setbeaninfo	getbeaninfo	beansetinfo	beangetinfo	getbeaninfo
Basically introspection means analysis of _____ capabilities	beans	applets	classes	API	beans
Introspection describes how methods , properties , and _____ are discovered in the beans that you write.	event	tool	properties	methods	event
Introspection describes how methods, _____ , and event are discovered beans that you write.	event	tool	properties	methods	properties
Introspection describes how _____ , properties , and event are discovered beans that you write.	event	tool	properties	methods	methods
Indicates the new jar file is created	c	f	t	x	c
Indicates that the first file in the file _ name list is the name of the jar file.	c	f	t	x	f
Indicates that the second file in the file _ name list is the name of the manifest file	c	m	t	x	m
Indicates that all the files and resources in the jar file are to be displayed in a tabular form	c	t	x	m	t
Indicates that the jar file should generate a verbose output.	c	t	v	m	v
Indicates that the files and resources of a jar are to be extracted.	c	x	o	t	x
Indicates that the jar file should not be compressed.	c	v	SQL server	t	o