JAVA ESSENTIALS

The Java programming language includes many keywords reserved for use only by Java. You cannot use keywords as variable names or values in your code. If you use a Java keyword inappropriately, the Java compiler will usually detect the error and stop compiling the code. The following is a list of Java reserved keywords.

PRIMITIVE DATA TYPES

`boolean` Holds the value `true` or `false`.	`byte` Holds an integer value ranging from -128 to 127.	`char` Holds a single Unicode character.	`double` Holds a 64-bit floating-point value ranging from ±4.9E-324 to ±1.7976931348623157E+308.
`float` Holds a 32-bit floating-point value ranging from ±1.4E-45 to ±3.4028235E+38.	`int` Holds a 32-bit integer value ranging from -2,147,483,648 to 2,147,483,647.	`long` Holds a 64-bit integer value ranging from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.	`short` Holds a 16-bit integer value ranging from -32,768 to 32,767.

FLOW CONTROL

`break` Exits the flow of a loop or switch and continues execution from the line following the loop or switch.	`case` Defines an option within a `switch` statement and the block of code that should be executed for that option.	`continue` Exits the flow of a loop and re-evaluates the loop condition.
`default` Defines a default option within a `switch` statement and the block of code that should be executed when there is no matching `case` statement.	`do` Defines a block of code that is executed once and repeats as long as a condition is true.	`else` Defines a block of code that is executed if a condition is false in an `if` statement.
`for` Defines a block of code that repeats depending on the value of a counter.	`if` Defines a block of code that is executed if a condition is true.	`return` Exits a method and returns control to the calling method, possibly returning a value.
`switch` Defines a variable or field whose current value may induce the execution of the block of code defined by a matching `case` statement.	`while` Defines a block of code that repeats as long as a condition is true.	

EXCEPTION HANDLING

`catch` Defines a block of code that handles exceptions thrown by a `try` block of code.	`finally` Defines a block of code that is executed in a `try-catch` structure whether an exception is thrown or not.	`throw` Exits a method and passes control to the block of code that handles the thrown exception.	`throws` Defines the exceptions that a method may throw.	`try` Defines a block of code that may throw an exception.

MODIFIERS

`abstract` Indicates that a method's functionality should be implemented by a subclass of the class that contains the method. Also, indicates that a class has such methods.	`final` Indicates that a class, method or field cannot be modified.	`native` Declares a Java method which is written in another language.	`private` Indicates that a method or field can be accessed only by the class that contains it.
`protected` Indicates that a method or field can be accessed only by the class that contains it or by a subclass of the class that contains it.	`public` Indicates that a method or field can be accessed by any class.	`static` Indicates that a method or field is a class member and can be instantiated only in the class that contains it.	`strictfp` Indicates that a method or class stores intermediate results using strict floating-point guidelines set by the IEEE 754 standard.
`synchronized` Indicates that a method cannot be executed by two threads at the same time and cannot be interrupted during execution. Also, indicates an object that can be accessed by only one synchronized block of code at a time.		`void` Indicates a method that does not return a value.	`volatile` Indicates that a field may be changed by multiple threads.

OBJECTS

`class` Defines a block of code that outlines the methods and fields of a class.	`extends` Specifies which class a subclass inherits methods and fields from.
`implements` Specifies interfaces from which a class implements methods and fields.	`interface` Indicates a block of code that defines an interface.

REFERENCE TYPES

`instanceof` Tests an object to determine if the object is an instance of a particular data type or class.	`new` Creates a new object in memory and calls the constructor method.
`super` Refers to the parent class of an object.	`this` Refers to the current object.

PACKAGES

`import` Specifies packages to be used by the source file.	`package` Specifies which package the classes of a source file belong to.

MISCELLANEOUS RESERVED WORDS

In addition to the keywords above, you also may not use `const`, `false`, `goto`, `null`, `transient` and `true` as names in your JSP code.

JAVASERVER PAGES ESSENTIALS

Legend:

`plain text` = required	**`bold`** = default
`italics` = user-defined	| = or
[] = optional	{} = required choice
... = list of items	+ = can repeat

COMMENTS

HTML Comment

An HTML comment is sent to the Web browser, but is not displayed. The information may be viewed by users who view the HTML source code.

Syntax:

```
<!-- comment [<%= expression %>] -->
```

Example:

```
<!-- Session ID: <%= session.getId() %> -->
```

Hidden Comment

A hidden comment is discarded before any processing of the JSP page takes place and is not sent to the Web browser.

Syntax:

```
<%-- comment --%>
```

Example:

```
<%-- Page created by: Jade Williams --%>
```

SCRIPTING ELEMENTS

Declaration

A declaration allows you to define variables and methods that will be used throughout a JSP page.

Syntax:

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

Example:

```
<%! String siteName = "ABC Corporation"; %>
```

Expression

An expression allows you to generate output on a JSP page.

Syntax:

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

Example:

```
<%= request.getAttribute("firstName") %>
```


SCRIPTING ELEMENTS (CONTINUED)

Scriptlet

A scriptlet is a block of code embedded within a JSP page, which performs tasks such as generating output.

Syntax:

```
<% code_fragment %>
```

Example:

```
<% out.println("ABC Corporation Web site") %>
```

DIRECTIVES

Include Directive

The `include` directive allows you to use one file in several different JSP pages.

Syntax:

```
<%@ include file="relativeURL" %>
```

Example:

```
<%@ include file="pages/footer.html" %>
```

Taglib Directive

The `taglib` directive allows you to define a tag library and a prefix that can be used to reference the custom tags.

Syntax:

```
<%@ taglib uri="tagLibURI" prefix="tagPrefix" %>
```

Example:

```
<%@ taglib uri="http://www.mysite.com/chart"
  prefix="barchart" %>
```

Page Directive

The `page` directive allows you to specify information about the configuration of a JSP page.

Syntax:

```
<%@ page [language="java"] [extends="package.class"]
  [import="{package.class|package.*}, ..."]
  [isThreadSafe="true|false"]
  [session="true|false"] [info="text"]
  [buffer="none|8kb|sizekb"] [autoFlush="true|false"]
  [errorPage="relativeURL"] [isErrorPage="true|false"]
  [contentType="{mimeType[; charset=characterSet]
              |text/html; charset=iso-8859-1}"] %>
```

Example:

```
<%@ page isErrorPage="true" contentType="text/plain" %>
```

JAVASERVER PAGES ESSENTIALS

CUSTOM TAGS

Custom Tag

A custom tag invokes custom actions included in a tag library. Custom actions are re-usable modules that may create and access objects and affect the output stream.

Syntax:

```
<tagPrefix:name attribute="value"+ ... />
```

or

```
<tagPrefix:name attribute="value"+ ... >
JSPContent
</tagPrefix:name>
```

Example:

```
<barchart:Vertical values="1, 4, 3" />
```

ACTIONS

<jsp:forward> Action

The `<jsp:forward>` action instructs a Web server to stop processing the current JSP page and start processing another.

Syntax:

```
<jsp:forward page="{relativeURL|<%= expression %>}" />
```

or

```
<jsp:forward page="{relativeURL|<%= expression %>}" >
    <jsp:param name="parameterName"
      value="{parameterValue|<%= expression %>}" />+
</jsp:forward>
```

Example:

```
<jsp:forward page="/search" />
```

<jsp:setProperty> Action

The `<jsp:setProperty>` action sets the value or values of a property in a JavaBean.

Syntax:

```
<jsp:setProperty name="beanID"
  {property="*"
  |property="propertyName" [param="parameterName"]
  |property="propertyName"
    value="{string|<%= expression %>}"} />
```

Example:

```
<jsp:setProperty name="lineBean" property="color"
  value="red" />
```


ACTIONS (CONTINUED)

<jsp:getProperty> Action

The `<jsp:getProperty>` action accesses a property of a JavaBean and can display the property in a JSP page.

Syntax:

```
<jsp:getProperty name="beanID" property="propertyName" />
```

Example:

```
<jsp:getProperty name="movieBean" property="title" />
```

<jsp:plugin> Action

The `<jsp:plugin>` action generates HTML code to display an applet or JavaBean using a Java plug-in in a Web browser. The plug-in is downloaded from a specified location if the Web browser is not capable of displaying the applet or JavaBean.

Syntax:

```
<jsp:plugin type="bean|applet" code="fileName.class"
  codebase="classURL"
  [name="instanceName"] [archive="archiveURI, ..."]
  [align="bottom|top|middle|left|right"]
  [height="hPixels"] [width="wPixels"]
  [hspace="horPixels"] [vspace="verPixels"]
  [jreversion="JREVersionNumber|1.1"]
  [nspluginurl="PluginURL"] [iepluginurl="PluginURL"] >
    [<jsp:params>
      [<jsp:param name="parameterName"
        value="{parameterValue|<%= expression %>}" />]+
    </jsp:params>]
    [<jsp:fallback> alternateText </jsp:fallback>]
</jsp:plugin>
```

Example:

```
<jsp:plugin type="applet" code="fireworks.class"
  codebase="/java" height=250 width=187>
    <jsp:fallback> Unable to load applet </jsp:fallback>
</jsp:plugin>
```

<jsp:include> Action

The `<jsp:include>` action includes a file, such as an HTML or JSP page, in the current JSP page.

Syntax:

```
<jsp:include page="{relativeURL|<%= expression %>}"
  flush="true" />
```

or

```
<jsp:include page="{relativeURL|<%= expression %>}"
  flush="true" >
    <jsp:param name="parameterName"
      value="{parameterValue|<%= expression %>}" />+
</jsp:include>
```

Example:

```
<jsp:include page="welcome.jsp" flush="true" />
```

<jsp:useBean> Action

The `<jsp:useBean>` action associates a JSP page with a specific JavaBean.

Syntax:

```
<jsp:useBean id="beanID"
  scope="page|request|session|application"
  {class="package.class" [type="package.class"]
  |[beanName="{package.class|<%= expression %>}"]
  type="package.class"}/>
```

or

```
<jsp:useBean id="beanID"
  scope="page|request|session|application"
  {class="package.class" [type="package.class"]
  |[beanName="{package.class|<%= expression %>}"]
  type="package.class"} >
otherElements
</jsp:useBean>
```

Example:

```
<jsp:useBean id="lineBean" scope="session"
  class="myBeans.lnBean" />
```

JSP IMPLICIT OBJECTS QUICK REFERENCE

REQUEST OBJECT

The `request` object retrieves and controls information sent from a client to the Web server. The `request` object is a subclass of the `javax.servlet.ServletRequest` class.

METHODS

`Object` **`getAttribute(String name)`**
Returns the value of an attribute of the `request` object.

`Enumeration` **`getAttributeNames()`**
Returns a list of names of available attributes in the `request` object.

`String` **`getCharacterEncoding()`**
Returns the name of the character set used by the `request` object.

`int` **`getContentLength()`**
Returns the length of the content body for the `request` object, in bytes.

`String` **`getContentType()`**
Returns the content type of the `request` object.

`ServletInputStream` **`getInputStream()`**
Uses a `ServletInputStream` object to retrieve a request as binary data.

`Locale` **`getLocale()`**
Returns the preferred regional setting in which the client computer accepts information.

`Enumeration` **`getLocales()`**
Returns a list of regional settings in which the client computer accepts information, in decreasing order of preference.

`String` **`getParameter(String name)`**
Returns the value of a parameter in the `request` object.

`Enumeration` **`getParameterNames()`**
Returns a list of names of parameters contained in the `request` object.

`String[]` **`getParameterValues(String name)`**
Returns an array containing all the values of a parameter.

`String` **`getProtocol()`**
Returns the name and version of the protocol used by the `request` object.

`BufferedReader` **`getReader()`**
Uses a `BufferedReader` object to retrieve a request as character data.

`String` **`getRealPath(String virtualPath)`**
Returns the real path that corresponds to a virtual path. Deprecated. Use `String ServletContext.getRealPath(String virtualPath)` instead.

`String` **`getRemoteAddr()`**
Returns the Internet Protocol (IP) address of the client computer that made the request.

`String` **`getRemoteHost()`**
Returns the name or IP address of the client computer that made the request.

METHODS (Continued)

`RequestDispatcher` **`getRequestDispatcher(String path)`**
Returns a `RequestDispatcher` object to be used with the resource located at the specified path.

`String` **`getScheme()`**
Returns the scheme used to make the request, such as http, https or ftp.

`String` **`getServerName()`**
Returns the name of the server that received the request.

`int` **`getServerPort()`**
Returns the number of the server port the request was received on.

`boolean` **`isSecure()`**
Returns a boolean value indicating whether the request was made through a secure channel.

`void` **`removeAttribute(String name)`**
Removes an attribute from the `request` object.

`void` **`setAttribute(String name, Object attribute)`**
Adds an attribute to the `request` object.

RESPONSE OBJECT

The `response` object sends and controls information from the Web server to a client. The `response` object is a subclass of the `javax.servlet.ServletResponse` class.

METHODS

`void` **`flushBuffer()`**
Sends information stored in the buffer to a client immediately.

`int` **`getBufferSize()`**
Returns the buffer size of the `response` object, in bytes.

`String` **`getCharacterEncoding()`**
Returns the name of the character set used by the `response` object.

`Locale` **`getLocale()`**
Returns the regional setting assigned to the `response` object.

`ServletOutputStream` **`getOutputStream()`**
Returns a `ServletOutputStream` object that allows the `response` object to write binary data to the client.

`PrintWriter` **`getWriter()`**
Returns a `PrintWriter` object that allows the `response` object to write character data to the client.

`boolean` **`isCommitted()`**
Returns a boolean value indicating if the `response` object has written the status code and headers.

`void` **`reset()`**
Clears information, status code and headers from the buffer.

JSP IMPLICIT OBJECTS QUICK REFERENCE

RESPONSE OBJECT (CONTINUED)

METHODS (Continued)

`void` **`setBufferSize`**`(int size)`
Sets the buffer size for the `response` object, in bytes.

`void` **`setContentType`**`(String type)`
Sets the content type of the `response` object.

`void` **`setContentLength`**`(int len)`
Sets the length of the content body for the `response` object, in bytes.

`void` **`setLocale`**`(Locale loc)`
Sets the regional setting for the `response` object.

PAGECONTEXT OBJECT

A `pageContext` object provides access to the namespaces associated with a JSP page, page attributes and implementation details. The `pageContext` object is a subclass of the `javax.servlet.jsp.PageContext` class.

FIELDS

`String` **`APPLICATION`**
Stores the `application` object in the name table of the `pageContext` object.

`int` **`APPLICATION_SCOPE`**
Indicates that a reference has application scope.

`String` **`CONFIG`**
Stores the `config` object in the name table of the `pageContext` object.

`String` **`EXCEPTION`**
Stores an uncaught exception in the attribute list of the `request` object and in the name table of the `pageContext` object.

`String` **`OUT`**
Stores the `out` object in the name table of the `pageContext` object.

`String` **`PAGE`**
Stores the `Servlet` object in the name table of the `pageContext` object.

FIELDS (Continued)

`int` **`PAGE_SCOPE`**
Indicates that a reference has page scope.

`String` **`PAGECONTEXT`**
Stores the `pageContext` object in its own name table.

`String` **`REQUEST`**
Stores the `request` object in the name table of the `pageContext` object.

`int` **`REQUEST_SCOPE`**
Indicates that a reference has request scope.

`String` **`RESPONSE`**
Stores the `response` object in the name table of the `pageContext` object.

`String` **`SESSION`**
Stores the `session` object in the name table of the `pageContext` object.

`int` **`SESSION_SCOPE`**
Indicates that a reference has session scope.

METHODS

`Object` **`findAttribute`**`(String name)`
Searches for an attribute in all valid scopes and returns the value.

`void` **`forward`**`(String relativeUrlPath)`
Forwards the current `request` and `response` objects to another resource found at the specified path.

`Object` **`getAttribute`**`(String name)`
Returns the value of an attribute in the page scope.

`Object` **`getAttribute`**`(String name, int scope)`
Returns the value of an attribute in the specified scope.

`Enumeration` **`getAttributeNamesInScope`**`(int scope)`
Returns a list of names of the available attributes in the specified scope.

`int` **`getAttributesScope`**`(String name)`
Returns the scope of an attribute.

`Exception` **`getException`**`()`
Returns any `exception` object that was passed to the JSP page.

`JspWriter` **`getOut`**`()`
Returns the `out` object that is being used for client response.

JSP IMPLICIT OBJECTS QUICK REFERENCE

PAGECONTEXT OBJECT (CONTINUED)

METHODS (Continued)

`Object` **`getPage()`**
Returns the `Servlet` associated with the `pageContext` object.

`ServletRequest` **`getRequest()`**
Returns the `request` object associated with the `pageContext` object.

`ServletResponse` **`getResponse()`**
Returns the `response` object associated with the `pageContext` object.

`ServletConfig` **`getServletConfig()`**
Returns the `config` object associated with the `pageContext` object.

`ServletContext` **`getServletContext()`**
Returns the `application` object associated with the `pageContext` object.

`HttpSession` **`getSession()`**
Returns the `session` object associated with the `pageContext` object.

`void` **`handlePageException(Exception exception)`**
Executes code when a specific error is encountered.

`void` **`include(String relativeUrlPath)`**
Processes the resource found at the specified path as part of the `request` and `response` objects currently being processed.

`void` **`initialize(Servlet servlet, ServletRequest request, ServletResponse response, String errorPageURL, boolean needsSession, int bufferSize, boolean autoFlush)`**
Initializes a `pageContext` object.

`JspWriter` **`popBody()`**
Returns the previous `out` object saved by the previous call of the `pushBody` method and updates the value of the `OUT` attribute in the page scope of the `pageContext` object.

`BodyContent` **`pushBody()`**
Returns a new `BodyContent` object, saves the current `out` object and updates the value of the `OUT` attribute in the page scope of the `pageContext` object.

`void` **`release()`**
Resets the `pageContext` object.

`void` **`removeAttribute(String name)`**
Removes an attribute.

`void` **`removeAttribute(String name, int scope)`**
Removes an attribute in the specified scope.

`void` **`setAttribute(String name, Object attribute)`**
Adds an attribute with page scope.

`void` **`setAttribute(String name, Object attribute, int scope)`**
Adds an attribute with the specified scope.

SESSION OBJECT

The `session` object stores session information about a client computer as the client navigates a Web site. The `session` object is a subclass of the `javax.servlet.http.HttpSession` class.

METHODS

`Object` **`getAttribute(String name)`**
Returns the information stored in a session value of the `session` object.

`Enumeration` **`getAttributeNames()`**
Returns a list of all session values available in the `session` object.

`long` **`getCreationTime()`**
Returns the time the session started, measured in milliseconds since January 1, 1970.

`String` **`getId()`**
Returns the session ID.

`long` **`getLastAccessedTime()`**
Returns the last time the client sent a request during the session, measured in milliseconds since January 1, 1970.

`int` **`getMaxInactiveInterval()`**
Returns the session timeout, in seconds.

`HttpSessionContext` **`getSessionContext()`**
Deprecated. This method will be removed in a future version of the Java Servlet API.

`Object` **`getValue(String name)`**
Returns the information stored in a session value of the `session` object.
Deprecated. Use `Object getAttribute(String name)` instead.

`String[]` **`getValueNames()`**
Returns a list of all session values available in the `session` object.
Deprecated. Use `Enumeration getAttributeNames()` instead.

`void` **`invalidate()`**
Terminates the current session.

`boolean` **`isNew()`**
Returns `true` if the client computer has not yet joined the session.

`void` **`putValue(String name, Object value)`**
Creates a session value for the `session` object.
Deprecated. Use `void setAttribute(String name, Object attribute)` instead.

`void` **`removeAttribute(String name)`**
Removes a session value from the `session` object.

`void` **`removeValue(String name)`**
Removes a session value from the `session` object.
Deprecated. Use `void removeAttribute(String name)` instead.

`void` **`setAttribute(String name, Object attribute)`**
Creates a session value for the `session` object.

`void` **`setMaxInactiveInterval(int interval)`**
Sets the session timeout, in seconds.

JSP IMPLICIT OBJECTS QUICK REFERENCE

APPLICATION OBJECT

The `application` object stores and shares information for use during an active application. The `application` object is a subclass of the `javax.servlet.ServletContext` class.

METHODS

`Object` **`getAttribute`**`(String name)` Returns the value of an attribute of the `application` object.	`int` **`getMinorVersion`**`()` Returns the minor version of the Java Servlet API that the server supports.
`Enumeration` **`getAttributeNames`**`()` Returns a list of names of available attributes in the `application` object.	`RequestDispatcher` **`getNamedDispatcher`**`(String name)` Returns a `RequestDispatcher` object to be used by the named `application` object.
`ServletContext` **`getContext`**`(String path)` Returns a `ServletContext` object to be used with the resource located at the specified path.	`String` **`getRealPath`**`(String virtualPath)` Returns the real path that corresponds to the specified virtual path.
`String` **`getInitParameter`**`(String name)` Returns the value of an initialization parameter of the `application` object.	`RequestDispatcher` **`getRequestDispatcher`**`(String path)` Returns a `RequestDispatcher` object to be used with the resource located at the specified path.
`Enumeration` **`getInitParameterNames`**`()` Returns a list of names of available initialization parameters in the `application` object.	`URL` **`getResource`**`(String path)` Returns a URL to the resource that is mapped to the specified path.
`int` **`getMajorVersion`**`()` Returns the major version of the Java Servlet API that the server supports.	`InputStream` **`getResourceAsStream`**`(String path)` Returns the resource located at the specified path as an `InputStream` object.
`String` **`getMimeType`**`(String file)` Returns the MIME type of a file.	`String` **`getServerInfo`**`()` Returns information about the server on which the servlet is running.

METHODS (Continued)

`Servlet` **`getServlet`**`(String name)` Deprecated. This method will be removed in a future version of the Java Servlet API.	`void` **`log`**`(String msg)` Writes a message to a servlet log file.
`Enumeration` **`getServletNames`**`()` Deprecated. This method will be removed in a future version of the Java Servlet API.	`void` **`log`**`(String msg, Throwable throwable)` Writes an explanatory message and a stack trace to the servlet log file for a given error.
`Enumeration` **`getServlets`**`()` Deprecated. This method will be removed in a future version of the Java Servlet API.	`void` **`removeAttribute`**`(String name)` Removes an attribute from the `application` object.
`void` **`log`**`(Exception exception, String msg)` Writes an explanatory error message to the servlet log file. Deprecated. Use `void log(String msg, Throwable throwable)` instead.	`void` **`setAttribute`**`(String name, Object attribute)` Creates an attribute for the `application` object.

OUT OBJECT

The `out` object is a buffered output stream that sends output to the client. The `out` object is a subclass of the `javax.servlet.jsp.JspWriter` class.

FIELDS

`boolean` **`autoFlush`** Indicates whether the buffer flushes automatically.	`int` **`NO_BUFFER`** Indicates that the `out` object is not buffered.
`int` **`bufferSize`** Stores the buffer size used by the `out` object, in bytes.	`int` **`UNBOUNDED_BUFFER`** Indicates that the `out` object is buffered and is using an unlimited buffer size.
`int` **`DEFAULT_BUFFER`** Indicates that the `out` object is buffered and is using the default buffer size.	

JSP IMPLICIT OBJECTS QUICK REFERENCE

OUT OBJECT (CONTINUED)

METHODS

`void` **`clear()`**
Clears the buffer. If the buffer has been flushed, this method causes an error to occur.

`void` **`clearBuffer()`**
Clears the buffer. This method does not cause an error if the buffer has been flushed.

`void` **`close()`**
Flushes the buffer and closes the output stream.

`void` **`flush()`**
Flushes the buffer.

`int` **`getBufferSize()`**
Returns the buffer size used by the `out` object, in bytes.

`int` **`getRemaining()`**
Returns the size of the unused area in the buffer, in bytes.

`boolean` **`isAutoFlush()`**
Returns `true` if the buffer flushes automatically.

`void` **`newLine()`**
Writes the line separator string to start a new line in the output stream.

`void` **`print(boolean b)`**
Prints a boolean value.

`void` **`print(char c)`**
Prints a character.

`void` **`print(char[] s)`**
Prints an array of characters.

`void` **`print(double d)`**
Prints a double-precision floating-point number.

`void` **`print(float f)`**
Prints a floating-point number.

`void` **`print(int i)`**
Prints an integer.

METHODS (Continued)

`void` **`print(long l)`**
Prints a long integer.

`void` **`print(Object obj)`**
Prints an object.

`void` **`print(String s)`**
Prints a string.

`void` **`println()`**
Writes the line separator string to terminate the current line.

`void` **`println(boolean b)`**
Prints a boolean value and then terminates the line.

`void` **`println(char c)`**
Prints a character and then terminates the line.

`void` **`println(char[] s)`**
Prints an array of characters and then terminates the line.

`void` **`println(double d)`**
Prints a double-precision floating-point number and then terminates the line.

`void` **`println(float f)`**
Prints a floating-point number and then terminates the line.

`void` **`println(int i)`**
Prints an integer and then terminates the line.

`void` **`println(long l)`**
Prints a long integer and then terminates the line.

`void` **`println(Object obj)`**
Prints an object and then terminates the line.

`void` **`println(String s)`**
Prints a string and then terminates the line.

JSP IMPLICIT OBJECTS QUICK REFERENCE

CONFIG OBJECT

The `config` object contains information about the servlet configuration. The `config` object is a subclass of the `javax.servlet.ServletConfig` class.

METHODS

`String` **`getInitParameter(String name)`**
Returns the value of the initialization parameter of the servlet.

`ServletContext` **`getServletContext()`**
Returns a reference to the `application` object in which the servlet is executing.

`Enumeration` **`getInitParameterNames()`**
Returns a list of names of the servlet's initialization parameters.

`String` **`getServletName()`**
Returns the name of the current servlet.

PAGE OBJECT

The `page` object refers to the JSP page itself. The `page` object is a subclass of the `java.lang.Object` class.

METHODS

`Object` **`clone()`**
Creates and returns a copy of the `page` object.

`Class` **`getClass()`**
Returns the runtime class of the `page` object.

`boolean` **`equals(Object obj)`**
Indicates whether another object is the same as the `page` object.

`int` **`hashCode()`**
Returns a hash code value for the `page` object.

`void` **`finalize()`**
Performs cleanup tasks when there are no more references to the `page` object.

`void` **`notify()`**
Wakes up a single thread that is waiting on the `page` object's monitor.

METHODS (Continued)

`void` **`notifyAll()`**
Wakes up all threads that are waiting on the `page` object's monitor.

`String` **`toString()`**
Returns a string representation of the `page` object.

`void` **`wait()`**
Causes the current thread to wait until another thread awakens the `page` object.

`void` **`wait(long timeout)`**
Causes the current thread to wait until another thread awakens the `page` object or the specified amount of time, measured in milliseconds, has elapsed.

`void` **`wait(long timeout, int nanos)`**
Causes the current thread to wait until another thread awakens the `page` object, another thread interrupts the current thread or the specified amount of time, measured in nanoseconds, has elapsed.

EXCEPTION OBJECT

The `exception` object contains information about a runtime error and is available only in an error page. An error page must contain the `isErrorPage=true` attribute in the `page` directive. The `exception` object is a subclass of the `java.lang.Throwable` class.

METHODS

`Throwable` **`fillInStackTrace()`**
Fills the `exception` object with current error information.

`void` **`printStackTrace(PrintStream s)`**
Prints information about the `exception` object to the specified print stream.

`String` **`getLocalizedMessage()`**
Returns an error message according to regional settings.

`void` **`printStackTrace(PrintWriter s)`**
Prints information about the `exception` object to the specified print writer.

`String` **`getMessage()`**
Returns the error message describing the exception.

`String` **`toString()`**
Returns a short description of this `exception` object.

`void` **`printStackTrace()`**
Prints information about the `exception` object to the standard error stream.

JAVA.SQL QUICK REFERENCE

The `java.sql` package provides interfaces and classes that allow Java programs to access and manipulate information in a database. The `java.sql` package is also known as the Java DataBase Connectivity 2.0 core Application Program Interface, or JDBC 2.0 core API.

INTERFACES

`Array`
Provides methods for handling data in an SQL `ARRAY` type.

`Blob`
Provides methods for handling data in an SQL `BLOB` type.

`CallableStatement`
Provides methods for executing SQL stored procedures.

`Clob`
Provides methods for handling data in an SQL `CLOB` type.

`Connection`
Provides methods for controlling a connection to a database.

`DatabaseMetaData`
Provides comprehensive information about a database.

`Driver`
This interface must be implemented by every driver class.

`PreparedStatement`
Provides methods for handling precompiled SQL statements.

`Ref`
Provides a method for retrieving the SQL name of a data structure type referenced by the `Ref` object and SQL `REF`.

`ResultSet`
Provides methods for handling the information generated by the execution of a statement that queries a database.

`ResultSetMetaData`
Provides methods for obtaining information about the columns in a `ResultSet` object.

`SQLData`
Provides methods for manipulating SQL user-defined types.

INTERFACES (Continued)

`SQLInput`
Provides methods for manipulating data in an input stream.

`SQLOutput`
Provides methods for writing data back to a database.

`Statement`
Provides methods for executing an SQL statement and obtaining the results.

`Struct`
Provides methods for retrieving information about an SQL structured type.

CLASSES

`Date`
Provides methods for manipulating an SQL `DATE`, which is measured in milliseconds since January 1, 1970.

`DriverManager`
Provides methods for managing a set of JDBC drivers.

`DriverPropertyInfo`
Provides methods for discovering and supplying properties for database connections.

`SQLPermission`
Holds the name of the permission that is checked by the `SecurityManager` when there is a call to one of the `setLogWriter` methods.

`Time`
Provides methods for manipulating an SQL `TIME` value, which is the time of day, based on the number of milliseconds since January 1, 1970.

`Timestamp`
Provides methods for manipulating an SQL `TIMESTAMP` value, which is the time of day and a nanosecond component. The time of day is based on the number of milliseconds since January 1, 1970.

`Types`
Defines the constants that represent generic SQL types.

JAVASERVER PAGES AND ACTIVE SERVER PAGES

Active Server Pages (ASP) is an alternative technology for generating dynamic Web pages. ASP, developed by Microsoft, and JSP have many similarities. Both technologies allow programmers to insert dynamic Web content into HTML pages using special tags, access information in databases, store information about a client computer throughout a session and use encapsulated components, such as ActiveX in ASP and JavaBeans in JSP.

There are many differences between the two technologies as well. ASP pages are processed almost exclusively by Microsoft Web servers—Internet Information Server (IIS) or Personal Web Server, while JSP pages can be processed by any server that supports Java servlets.

ASP code is usually written using VBScript or JScript, which are Microsoft proprietary scripting languages. JSP code, however, is written using Java, which is platform independent and can run on any computer that has a Java virtual machine. Java has more flexibility and fewer limitations than the scripting languages used in ASP.

There are also differences in the way JSP and ASP pages are processed by Web servers. Every time an ASP page is requested by a client, the code in the ASP page is interpreted by the Web server and the results are then sent to the Web browser. The first time a JSP page is requested, the code in the JSP page is compiled into a servlet by the Web server. The server then processes the servlet to generate the HTML code which is sent to the Web browser. The Web server keeps a copy of the servlet for future requests. The next time the page is requested, the precompiled servlet can simply be processed. Processing precompiled servlets is faster than re-interpreting the code in ASP pages each time a page is requested.

The following pages illustrate the differences between JSP and ASP implicit objects and their most common functions, as well as other JSP and ASP features, such as scripting elements and comments.

REQUEST OBJECT

Function	JSP	ASP
Object name	`request`	`Request`
Retrieve certification information	N/A	`ClientCertificate (String key[String field])`
Retrieve cookies	`getCookies()`	`Cookies(String name) [(String key)]`
Retrieve form data	`getParameter(String name)`, `getParameterNames()` and `getParameterValues(String name)`	`Form(String element) [(int index)]`
Retrieve query data	`getParameter(String name)` and `getQueryString()`	`QueryString(String element) [(int index)]`
Retrieve HTTP headers	`getHeaderNames()`, `getHeader(String name)`, `getIntHeader(String name)` and `getDateHeader(String name)`	`ServerVariables(String serverVar)`

RESPONSE OBJECT

Function	JSP	ASP
Object name	`response`	`Response`
Enable or disable buffering	The JSP `response` object does not support this function, but this can be done using the `<%@ page buffer="sizekb|none" %>` directive.	`Buffer = True|False`
Enable or disable proxy server caching	`setHeader("Cache-Control", "no-cache")`	`CacheControl = Public|Private`
Create a cookie	`addCookie(Cookie name)`	`Cookies(String name)[(String key).attribute] = value`
Add an HTTP header	`setHeader(String name, String value)`	`AddHeader String Name, String Value`
Load a new page	`sendRedirect(String AbsURL)` This function needs an encoded URL from `encodeRedirectURL(String url)` if URL rewriting is being used to track a session.	`Redirect String url`
Send an error to client	`sendError(int code, String msg)`	N/A
Encode a URL	`encodeURL(String url)` This function appends the session ID to the URL if URL rewriting is being used to track a session.	The ASP `Response` object does not support this function, but this can be done using the `Server.URLencode(String url)` method.
Set the MIME type	`setContentType(String mimeType)`	`ContentType = String mimeType`

PAGECONTEXT OBJECT

Function	JSP	ASP
Object name	`pageContext`	No similar object

JAVASERVER PAGES AND ACTIVE SERVER PAGES

SESSION OBJECT

Function	JSP	ASP
Object name	`session`	`Session`
Terminate a session	`invalidate()`	`Abandon`
Create a session variable	`setAttribute(String name, Object attribute)`	`Session(String name) = "Variable Data"`
Create a session object	`setAttribute(String name, Object attribute)`	`Set Session(String name) = Server.CreateObject(String name)`
Retrieve a session variable	`getAttribute(String name)`	`My_Variable = Session(String name)`
Retrieve a session object	`getAttribute(String name)`	`Set My_Object = Session(String name)`
Remove a session variable or object	`removeAttribute(String name)`	`Contents.Remove(String name)`
Retrieve session variable or object names	`getAttributeNames()`	`For Each Key in Session.Contents` `  Response.Write(Key & " : " & Session(Key) & " ")` `Next`
Retrieve the session ID	`getId()`	`SessionID`
Set the session timeout	`setMaxInactiveInterval(int seconds)`	`Timeout(int Minutes)`
Retrieve the session timeout	`getMaxInactiveInterval()`	N/A
Retrieve the preferred regional setting in which the client computer accepts information	The JSP `session` object does not support this function, but this can be done using the `request.getLocale()` method.	`LCID`
Disable the session	This function can be done using the `<%@ page session="false" %>` directive.	This function can be done using the `<%@ EnableSessionState=False %>` directive.

APPLICATION OBJECT

Function	JSP	ASP
Object name	`application`	`Application`
Create a variable	`setAttribute(String name, Object object)`	`Application(String name) = "Variable Data"`
Create an object	`setAttribute(String name, Object object)`	`Set Application(String name) = Server.CreateObject(String name)`
Retrieve a variable	`getAttribute(String name)`	`My_Variable = Application(String name)`
Retrieve an object	`getAttribute(String name)`	`Set My_Object = Application(String name)`
Remove a variable or object	`removeAttribute(String name)`	`Contents.Remove(String name)`
Retrieve variable or object names	`getAttributeNames()`	`For Each Key in Application.Contents` `  Response.Write(Key & " : " & Application(Key) & " ")` `Next`
Lock and unlock variables	The JSP `application` object does not support this function, but this can be done using thread control.	`Lock` and `Unlock`
Retrieve information about the server	`getServerInfo()`	N/A
Determine the servlet API version	`getMajorVersion()` and `getMinorVersion()`	N/A
Write to the servlet log file	`log(String msg)`	The ASP `Application` object does not support this function, but this can be done using the `Response.AppendToLog(String msg)` method.
Determine the MIME type of a file	`getMimeType(String file)`	N/A
Find a virtual path's corresponding real path	`getRealPath(String virtualpath)`	The ASP `Application` object does not support this function, but this can be done using the `Server.MapPath(String path)` method.
Find the URL to a resource	`getResource(String path)`	N/A

JAVASERVER PAGES AND ACTIVE SERVER PAGES

OUT OBJECT

Function	JSP	ASP
Object name	`out`	`Response`
Write to the output buffer	`print(data)`	`Write(data)`
Write binary data	The JSP `out` object does not support this function, but this can be done using the `OutputStream.write (Byte[] buffer)` method.	`BinaryWrite(data)`
Clear the buffer	`clearBuffer()`	`Clear()`
Flush the buffer	`flush()`	`Flush()`
Close the output stream	`close()`	`End()` This method stops the processing of the current page.

CONFIG OBJECT

Function	JSP	ASP
Object name	`config`	No similar object
Determine the name of the current servlet	`getServletName()`	N/A
Return a reference to the application object	`getServletContext()`	N/A
Retrieve the names of the servlet's initialization parameters	`getInitParameterNames()`	N/A
Retrieve the value of an initialization parameter	`getInitParameter(String name)`	N/A

PAGE OBJECT

Function	JSP	ASP
Object name	`page`	No similar object

ERROR OBJECT

Function	JSP	ASP
Object name	`exception`	`ASPError`
Retrieve an error message	`getMessage()`	`Description()`
Retrieve a detailed error description	`toString()`	`ASPDescription()`
Print information about an error	`printStackTrace(PrintStream s)` or `printStackTrace(PrintWriter s)`	N/A
Determine the position of an error in the source file	N/A	`Line` and `Column`

SERVER OBJECT

Function	JSP	ASP
Object name	No similar object, but other JSP implicit objects have methods that support most functions of the ASP `Server` object.	`Server`
Create an object	To create an object, use standard Java syntax.	`CreateObject(Object id)`
Apply HTML encoding to a string	N/A	`HTMLEncode(String s)`
Find a virtual path's corresponding real path	This function can be done using the `application.getRealPath(String virtualPath)` method.	`MapPath(String virtualPath)`
Encode a URL	This function can be done using the `response.encodeURL(String url)` method.	`URLEncode(String url)`
Forward control to a new page	This function can be done using the `<jsp:forward page="String path" />` action.	`Transfer(String path)`
Set the amount of time a script can run on the server before it is terminated	N/A	`ScriptTimeout = int Seconds`

JAVASERVER PAGES AND ACTIVE SERVER PAGES

SCRIPTING ELEMENTS

Element	JSP	ASP
Declarations: define functions, methods and variables that will be used by the scriptlets in a Web page.	<%! Method and variable declarations %>	<% Function and variable declarations %>
Expressions: generate output directly to a Web page.	<%= Variable name or call to a method %>	<%= Variable name or call to a function %>
Scriptlets: embed blocks of code within a Web page to perform tasks such as generating output.	<% JSP code %>	<% ASP code %>

SCRIPTING COMMENTS

Comment Type	JSP	ASP
Scriptlet comments: add information to the scriptlet code.	// a single-line comment or /* a multiple-line comment */	' a single-line comment
Hidden comments: add information to the server-side file and are not sent to the Web browser.	<%-- a hidden comment --%>	N/A

INCLUDE COMMANDS

Include Type	JSP	ASP
Static includes: include files before the processing of a page.	<%@ include file="String url" %> or <jsp:include file="String url" flush="true" > <jsp:param name="String name" value="String value" /> </jsp:include>	<!--#include file="String relativePath"--> or <!--#include virtual="String virtualPath"-->
Dynamic includes: include a file during the processing of a page.	<%@ include page="String url" %> or <jsp:include page="String url" flush="true" > <jsp:param name="String name" value="String value" /> </jsp:include>	Server.Execute("String url")

FILE REDIRECTION

Redirection Type	JSP	ASP
Server redirection: stops the execution of the current page and transfers control to a new page.	<jsp:forward page="String path" /> or <jsp:forward page="String path" > <jsp:param name="String name" value="String value" /> </jsp:forward>	Server.Transfer(String path)