INTERNET TECHNOLOGIES 5th SEMESTER CORE - 11

What is an object in Java?



An entity that has state and behavior is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.

An object has three characteristics:

* State: represents the data (value) of an object.
* Behavior: represents the behavior (functionality) of an object such as deposit, withdraw, etc.
* Identity: An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

For Example, Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behavior.

An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

Object Definitions:

* An object is *a real-world entity*.
* An object is *a runtime entity*.
* The object is *an entity which has state and behavior*.
* The object is *an instance of a class*.

Example 1:

*File: TestStudent2.java*

1. class Student{
2. int id;
3. String name;
4. }
5. class TestStudent2{
6. public static void main(String args[]){
7. Student s1=new Student();
8. s1.id=101;
9. s1.name="Sonoo";
10. System.out.println(s1.id+" "+s1.name);//printing members with a white space
11. }
12. }

Output: Sonoo

*EXAMPLE 2:*

*File: TestEmployee.java*

1. class Employee{
2. int id;
3. String name;
4. float salary;
5. void insert(int i, String n, float s) {
6. id=i;
7. name=n;
8. salary=s;
9. }
10. void display(){System.out.println(id+" "+name+" "+salary);}
11. }
12. public class TestEmployee {
13. public static void main(String[] args) {
14. Employee e1=new Employee();
15. Employee e2=new Employee();
16. Employee e3=new Employee();
17. e1.insert(101,"ajeet",45000);
18. e2.insert(102,"irfan",25000);
19. e3.insert(103,"nakul",55000);
20. e1.display();
21. e2.display();
22. e3.display();
23. }
24. }

Output:

ajeet 45000.0

irfan 25000.0

nakul 55000.0

# Array vs ArrayList in Java

Array is a fixed length data structure whereas ArrayList is a variable length Collection class. We cannot change length of array once created in Java but ArrayList can be changed.

We cannot store primitives in ArrayList, it can only store objects. But array can contain both primitives and objects in Java. Since Java 5, primitives are automatically converted in objects which is known as auto-boxing.

EXAMPLE 3:

1. import java.util.\*;
2. public class ListExample {
3. public static void main(String[] args) {
4. List<Integer> list=new ArrayList<>();
5. list.add(Integer.valueOf(10));//storing Integer object
6. list.add(20);//Now compiler converts it into Integer.valueOf(20) which is object
7. list.add(30);
8.
9. System.out.println("Traversing List...");
10. for(Integer i:list){
11. System.out.println(i);
12. }
13. }
14. }

Output:

Traversing List...

10

20

30

## Example 4:

The following program illustrates several of the methods supported by ArrayList −

import java.util.\*;

public class ArrayListDemo {

 public static void main(String args[]) {

 // create an array list

 ArrayList al = new ArrayList();

 System.out.println("Initial size of al: " + al.size());

 // add elements to the array list

 al.add("C");

 al.add("A");

 al.add("E");

 al.add("B");

 al.add("D");

 al.add("F");

 al.add(1, "A2");

 System.out.println("Size of al after additions: " + al.size());

 // display the array list

 System.out.println("Contents of al: " + al);

 // Remove elements from the array list

 al.remove("F");

 al.remove(2);

 System.out.println("Size of al after deletions: " + al.size());

 System.out.println("Contents of al: " + al);

 }

}

This will produce the following result −

## Output

Initial size of al: 0

Size of al after additions: 7

Contents of al: [C, A2, A, E, B, D, F]

Size of al after deletions: 5

Contents of al: [C, A2, E, B, D]

# Javascript Data Types

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:

1. var a=40;//holding number
2. 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 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 |

## The Concept of Data Types

In programming, data types is an important concept.

To be able to operate on variables, it is important to know something about the type.

Without data types, a computer cannot safely solve this:

var x = 16 + "Volvo";

Does it make any sense to add "Volvo" to sixteen? Will it produce an error or will it produce a result?

JavaScript will treat the example above as:

var x = "16" + "Volvo";

When adding a number and a string, JavaScript will treat the number as a string.

# 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. |

EXAMPLE 6:

var x = 5 + 5;
var y = "5" + 5;
var z = "Hello" + 5;

The result of *x*,*y*, and *z* will be:

10
55
Hello5

[Try it Yourself »](https://www.w3schools.com/JS/tryit.asp?filename=tryjs_oper_concat5)

# avaScript Functions

JavaScript functions are used to perform operations. We can call JavaScript function many times to reuse the code.

#### Advantage of JavaScript function

There are mainly two advantages of JavaScript functions.

1. Code reusability: We can call a function several times so it save coding.
2. Less coding: It makes our program compact. We don’t need to write many lines of code each time to perform a common task.

## JavaScript Function Syntax

The syntax of declaring function is given below.

1. function functionName([arg1, arg2, ...argN]){
2. //code to be executed
3. }

JavaScript Functions can have 0 or more arguments.

## JavaScript Function Example 7:

Let’s see the simple example of function in JavaScript that does not has arguments.

1. <script>
2. function msg(){
3. alert("hello! this is message");
4. }
5. </script>
6. <input type="button" onclick="msg()" value="call function"/>

## JavaScript Function Arguments

We can call function by passing arguments. Let’s see the example of function that has one argument.

1. <script>
2. function getcube(number){
3. alert(number\*number\*number);
4. }
5. </script>
6. <form>
7. <input type="button" value="click" onclick="getcube(4)"/>
8. </form>

JAVASCRIPT CONTROL AND LOOPING STRUCTURES:

## 1. If  Statement

* IF statement is a conditional branching statement.
* In 'IF' statement, if the condition is true a group of statement is executed. And if the condition is false, the following statement is skipped.

#### Syntax : If statement

if(condition)
{
     //Statement 1;
     //Statement 2;
}

#### Example : Simple Program for IF Statement

<html>
     <body>
     <script type="text/javascript">
          var num = prompt("Enter Number");
          if (num > 0)
          {
               alert("Given number is Positive!!!");
          }
     </script>
     </body>
</html>

Output:



## 2. If – Else Statement

* If – Else is a two-way decision statement.
* It is used to make decisions and execute statements conditionally.

Flow Diagram of If – Else Statement


#### Syntax : If-Else statement

if (condition)
{
     //Statement 1;
}
else if(condition)
{
     //Statement 2;
}
else
{
     //Statement 3;
}

#### Example : Simple Program for If-Else Statement

<html>
     <head>
     <script type="text/javascript">
          var no = prompt("Enter a Number to find Odd or Even");
          no = parseInt(no);
          if (isNaN(no))
          {
               alert("Please Enter a Number");
          }
          else if (no == 0)
          {
               alert("The Number is Zero");
          }
          else if (no % 2)
          {
               alert("The Number is Odd");
          }
          else
          {
               alert("The Number is Even");
          }
     </script>
     </head>
</html>

Output:









## 3. Switch Statement

* Switch is used to perform different actions on different conditions.
* It is used to compare the same expression to several different values.

Flow Diagram of Switch Statement


#### Syntax

switch(expression)
{
     case condition 1:
          //Statements;
          break;
     case condition 2:
          //Statements;
          break;
     case condition 3:
          //Statements;
          break;
     .
     .
     case condition n:
          //Statements;
          break;
     default:
          //Statement;
}

#### Example : Simple Program for Switch Statement

<html>
<head>
     <script type="text/javascript">
     var day = prompt("Enter a number between 1 and 7");
     switch (day)
     {
          case (day="1"):
               document.write("Sunday");
               break;
          case (day="2"):
               document.write("Monday");
               break;
          case (day="3"):
               document.write("Tuesday");
               break;
          case (day="4"):
               document.write("Wednesday");
               break;
          case (day="5"):
               document.write("Thursday");
               break;
          case (day="6"):
               document.write("Friday");
               break;
          case (day="7"):
               document.write("Saturday");
               break;
          default:
               document.write("Invalid Weekday");
               break;
     }
     </script>
</head>
</html>

Output:



## 4. For Loop

* For loop is a compact form of looping.

It includes three important parts:
1. Loop Initialization
2. Test Condition
3. Iteration

* All these three parts come in a single line separated by semicolons(;).

Flow Diagram of 'For' Loop


#### Syntax

for(initialization; test-condition; increment/decrement)
{
     //Statements;
}

#### Example : Palindrome Program using For Loop

<html>
<body>
     <script type="text/javascript">
     function palindrome()
     {
          var revstr = " ";
          var strr = document.getElementById("strr").value;
          var i = strr.length;
          for(var j=i; j>=0; j--)
          {
               revstr = revstr+strr.charAt(j);
          }
          if(strr == revstr)
          {
               alert(strr+" - is Palindrome");
          }
          else
          {
               alert(strr+" - is not a Palindrome");
          }
     }
     </script>
     <form>
          Enter a String or Number: <input type="text" id="strr" name="checkpalindrome"><br>
          <input type="submit" value="Check" onclick="palindrome();">
     </form>
</body>
</html>

Output:





## 5. For-in Loop

* For-in loop is used to traverse all the properties of an object.
* It is designed for looping through arrays.

#### Syntax

for (variable\_name in Object)
{
     //Statements;
}

## 6. While Loop

* While loop is an entry-controlled loop statement.
* It is the most basic loop in JavaScript.
* It executes a statement repeatedly as long as expression is true.
* Once the expression becomes false, the loop terminates.

Flow Diagram of While Loop


#### Syntax

while (condition)
{
     //Statements;
}

#### Example : Fibonacci Series Program using While Loop

<html>
<body>
     <script type="text/javascript">
          var no1=0,no2=1,no3=0;
          document.write("Fibonacci Series:"+"<br>");
          while (no2<=10)
          {
               no3 = no1+no2;
               no1 = no2;
               no2 = no3;
               document.write(no3+"<br>");
          }
     </script>
</body>
</html>

Output:


## 7. Do-While Loop

* Do-While loop is an exit-controlled loop statement.
* Similar to the While loop, the only difference is condition will be checked at the end of the loop.
* The loop is executed at least once, even if the condition is false.

Flow Diagram of Do – While


#### Syntax

do
{
     //Statements;
}
while(condition);

#### Example : Simple Program on Do-While Loop

<html>
<body>
     <script type ="text/javascript">
     var i = 0;
     do
     {
          document.write(i+"<br>")
          i++;
     }
     while (i <= 5)
     </script>
</body>
</html>

Output:
0
1
2
3
4
5

## Difference between While Loop and Do – While Loop

|  |  |
| --- | --- |
| While Loop | Do – While Loop |
| In while loop, first it checks the condition and then executes the program. | In Do – While loop, first it executes the program and then checks the condition. |
| It is an entry – controlled loop. | It is an exit – controlled loop. |
| The condition will come before the body. | The condition will come after the body. |
| If the condition is false, then it terminates the loop. | It runs at least once, even though the conditional is false. |
| It is a counter-controlled loop. | It is a iterative control loop. |

## 8. Break Statement

* Break statement is used to jump out of a loop.
* It is used to exit a loop early, breaking out of the enclosing curly braces.

Syntax:
break;

Flow Diagram of Break Statement


## 9. Continue Statement

* Continue statement causes the loop to continue with the next iteration.
* It skips the remaining code block.

Flow Diagram of Continue Statement

Syntax:
continue;

## What is an Event in javascript?

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 understanding [HTML Event Reference](https://www.tutorialspoint.com/html/html_events_ref.htm). Here we will see a few examples to understand a relation between Event and JavaScript −

## onclick Event Type

This is the most frequently used event type which occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type.

### Example

<html>

 <head>

 <script type = "text/javascript">

 <!--

 function sayHello() {

 alert("Hello World")

 }

 //-->

 </script>

 </head>

 <body>

 <p>Click the following button and see result</p>

 <form>

 <input type = "button" onclick = "sayHello()" value = "Say Hello" />

 </form>

 </body>

</html>

### Output

## onsubmit Event Type

onsubmit is an event that occurs when you try to submit a form. You can put your form validation against this event type.

### Example

The following example shows how to use onsubmit. Here we are calling a validate() function before submitting a form data to the webserver. If validate() function returns true, the form will be submitted, otherwise it will not submit the data.

Try the following example.

<html>

 <head>

 <script type = "text/javascript">

 <!--

 function validation() {

 all validation goes here

 .........

 return either true or false

 }

 //-->

 </script>

 </head>

 <body>

 <form method = "POST" action = "t.cgi" onsubmit = "return validate()">

 .......

 <input type = "submit" value = "Submit" />

 </form>

 </body>

</html>

## onmouseover and onmouseout

These two event types will help you create nice effects with images or even with text as well. The onmouseover event triggers when you bring your mouse over any element and the onmouseout triggers when you move your mouse out from that element. Try the following example.

<html>

 <head>

 <script type = "text/javascript">

 <!--

 function over() {

 document.write ("Mouse Over");

 }

 function out() {

 document.write ("Mouse Out");

 }

 //-->

 </script>

 </head>

 <body>

 <p>Bring your mouse inside the division to see the result:</p>

 <div onmouseover = "over()" onmouseout = "out()">

 <h2> This is inside the division </h2>

 </div>

 </body>

</html>

## What is JDBC?

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

The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

* Making a connection to a database.
* Creating SQL or MySQL statements.
* Executing SQL or MySQL queries in the database.
* Viewing & Modifying the resulting records.

Fundamentally, JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database. Java can be used to write different types of executables, such as −

* Java Applications
* Java Applets
* Java Servlets
* Java ServerPages (JSPs)
* Enterprise JavaBeans (EJBs).

All of these different executables are able to use a JDBC driver to access a database, and take advantage of the stored data.

JDBC provides the same capabilities as ODBC, allowing Java programs to contain database-independent code.

## Pre-Requisite

Before moving further, you need to have a good understanding of the following two subjects −

* [Core JAVA Programming](https://www.tutorialspoint.com/java/index.htm)
* [SQL or MySQL Database](https://www.tutorialspoint.com/mysql/index.htm)

## JDBC Architecture

The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers −

* JDBC API: This provides the application-to-JDBC Manager connection.
* JDBC Driver API: This supports the JDBC Manager-to-Driver Connection.

The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.

The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.

Following is the architectural diagram, which shows the location of the driver manager with respect to the JDBC drivers and the Java application −



## Common JDBC Components

The JDBC API provides the following interfaces and classes −

* DriverManager: This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.
* Driver: This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.
* Connection: This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.
* Statement: You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.
* ResultSet: These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
* SQLException: This class handles any errors that occur in a database application.

# JDBC - Database Connections

The programming involved to establish a JDBC connection is fairly simple. Here are these simple four steps −

* Import JDBC Packages: Add import statements to your Java program to import required classes in your Java code.
* Register JDBC Driver: This step causes the JVM to load the desired driver implementation into memory so it can fulfill your JDBC requests.
* Database URL Formulation: This is to create a properly formatted address that points to the database to which you wish to connect.
* Create Connection Object: Finally, code a call to the *DriverManager*object's *getConnection( )* method to establish actual database connection.

## Import JDBC Packages

The Import statements tell the Java compiler where to find the classes you reference in your code and are placed at the very beginning of your source code.

To use the standard JDBC package, which allows you to select, insert, update, and delete data in SQL tables, add the following *imports* to your source code −

import java.sql.\* ; // for standard JDBC programs

import java.math.\* ; // for BigDecimal and BigInteger support

## Register JDBC Driver

You must register the driver in your program before you use it. Registering the driver is the process by which the Oracle driver's class file is loaded into the memory, so it can be utilized as an implementation of the JDBC interfaces.

You need to do this registration only once in your program. You can register a driver in one of two ways.

## Approach I - Class.forName()

The most common approach to register a driver is to use Java's Class.forName() method, to dynamically load the driver's class file into memory, which automatically registers it. This method is preferable because it allows you to make the driver registration configurable and portable.

The following example uses Class.forName( ) to register the Oracle driver −

try {

 Class.forName("oracle.jdbc.driver.OracleDriver");

}

catch(ClassNotFoundException ex) {

 System.out.println("Error: unable to load driver class!");

 System.exit(1);

}

You can use getInstance() method to work around noncompliant JVMs, but then you'll have to code for two extra Exceptions as follows −

try {

 Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();

}

catch(ClassNotFoundException ex) {

 System.out.println("Error: unable to load driver class!");

 System.exit(1);

catch(IllegalAccessException ex) {

 System.out.println("Error: access problem while loading!");

 System.exit(2);

catch(InstantiationException ex) {

 System.out.println("Error: unable to instantiate driver!");

 System.exit(3);

}

## Approach II - DriverManager.registerDriver()

The second approach you can use to register a driver, is to use the static DriverManager.registerDriver() method.

You should use the *registerDriver()* method if you are using a non-JDK compliant JVM, such as the one provided by Microsoft.

The following example uses registerDriver() to register the Oracle driver −

try {

 Driver myDriver = new oracle.jdbc.driver.OracleDriver();

 DriverManager.registerDriver( myDriver );

}

catch(ClassNotFoundException ex) {

 System.out.println("Error: unable to load driver class!");

 System.exit(1);

}

## Database URL Formulation

After you've loaded the driver, you can establish a connection using the DriverManager.getConnection() method. For easy reference, let me list the three overloaded DriverManager.getConnection() methods −

* getConnection(String url)
* getConnection(String url, Properties prop)
* getConnection(String url, String user, String password)

Here each form requires a database URL. A database URL is an address that points to your database.

Formulating a database URL is where most of the problems associated with establishing a connection occurs.

Following table lists down the popular JDBC driver names and database URL.

|  |  |  |
| --- | --- | --- |
| RDBMS | JDBC driver name | URL format |
| MySQL | com.mysql.jdbc.Driver | jdbc:mysql://hostname/ databaseName |
| ORACLE | oracle.jdbc.driver.OracleDriver | jdbc:oracle:thin:@hostname:port Number:databaseName |
| DB2 | COM.ibm.db2.jdbc.net.DB2Driver | jdbc:db2:hostname:port Number/databaseName |
| Sybase | com.sybase.jdbc.SybDriver | jdbc:sybase:Tds:hostname: port Number/databaseName |

All the highlighted part in URL format is static and you need to change only the remaining part as per your database setup.

## Create Connection Object

We have listed down three forms of DriverManager.getConnection()method to create a connection object.

## Using a Database URL with a username and password

The most commonly used form of getConnection() requires you to pass a database URL, a *username*, and a *password*:

Assuming you are using Oracle's thin driver, you'll specify a host:port:databaseName value for the database portion of the URL.

If you have a host at TCP/IP address 192.0.0.1 with a host name of amrood, and your Oracle listener is configured to listen on port 1521, and your database name is EMP, then complete database URL would be −

jdbc:oracle:thin:@amrood:1521:EMP

Now you have to call getConnection() method with appropriate username and password to get a Connection object as follows −

String URL = "jdbc:oracle:thin:@amrood:1521:EMP";

String USER = "username";

String PASS = "password"

Connection conn = DriverManager.getConnection(URL, USER, PASS);

## Using Only a Database URL

A second form of the DriverManager.getConnection( ) method requires only a database URL −

DriverManager.getConnection(String url);

However, in this case, the database URL includes the username and password and has the following general form −

jdbc:oracle:driver:username/password@database

So, the above connection can be created as follows −

String URL = "jdbc:oracle:thin:username/password@amrood:1521:EMP";

Connection conn = DriverManager.getConnection(URL);

## Using a Database URL and a Properties Object

A third form of the DriverManager.getConnection( ) method requires a database URL and a Properties object −

DriverManager.getConnection(String url, Properties info);

A Properties object holds a set of keyword-value pairs. It is used to pass driver properties to the driver during a call to the getConnection() method.

To make the same connection made by the previous examples, use the following code −

import java.util.\*;

String URL = "jdbc:oracle:thin:@amrood:1521:EMP";

Properties info = new Properties( );

info.put( "user", "username" );

info.put( "password", "password" );

Connection conn = DriverManager.getConnection(URL, info);

## Closing JDBC Connections

At the end of your JDBC program, it is required explicitly to close all the connections to the database to end each database session. However, if you forget, Java's garbage collector will close the connection when it cleans up stale objects.

Relying on the garbage collection, especially in database programming, is a very poor programming practice. You should make a habit of always closing the connection with the close() method associated with connection object.

To ensure that a connection is closed, you could provide a 'finally' block in your code. A *finally* block always executes, regardless of an exception occurs or not.

To close the above opened connection, you should call close() method as follows −

conn.close();

# JDBC - Statements, PreparedStatement and CallableStatement

Once a connection is obtained we can interact with the database. The JDBC *Statement, CallableStatement,* and *PreparedStatement* interfaces define the methods and properties that enable you to send SQL or PL/SQL commands and receive data from your database.

They also define methods that help bridge data type differences between Java and SQL data types used in a database.

The following table provides a summary of each interface's purpose to decide on the interface to use.

|  |  |
| --- | --- |
| Interfaces | Recommended Use |
| Statement | Use this for general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters. |
| PreparedStatement | Use this when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime. |
| CallableStatement | Use this when you want to access the database stored procedures. The CallableStatement interface can also accept runtime input parameters. |

## The Statement Objects

### Creating Statement Object

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.

## 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();

}

# JDBC - Result Sets

The SQL statements that read data from a database query, return the data in a result set. The SELECT statement is the standard way to select rows from a database and view them in a result set. The *java.sql.ResultSet* interface represents the result set of a database query.

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 ResultSet

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

|  |  |
| --- | --- |
| S.N. | Methods & Description |
| 1 | public void beforeFirst() throws SQLExceptionMoves the cursor just before the first row. |
| 2 | public void afterLast() throws SQLExceptionMoves the cursor just after the last row. |
| 3 | public boolean first() throws SQLExceptionMoves the cursor to the first row. |
| 4 | public void last() throws SQLExceptionMoves the cursor to the last row. |
| 5 | public boolean absolute(int row) throws SQLExceptionMoves the cursor to the specified row. |
| 6 | public boolean relative(int row) throws SQLExceptionMoves the cursor the given number of rows forward or backward, from where it is currently pointing. |
| 7 | public boolean previous() throws SQLExceptionMoves the cursor to the previous row. This method returns false if the previous row is off the result set. |
| 8 | public boolean next() throws SQLExceptionMoves 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 SQLExceptionReturns the row number that the cursor is pointing to. |
| 10 | public void moveToInsertRow() throws SQLExceptionMoves 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 SQLExceptionMoves 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](https://www.tutorialspoint.com/jdbc/navigate-result-sets.htm).

## 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 SQLExceptionReturns the int in the current row in the column named columnName. |
| 2 | public int getInt(int columnIndex) throws SQLExceptionReturns 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](https://www.tutorialspoint.com/jdbc/viewing-result-sets.htm).

## 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 SQLExceptionChanges the String in the specified column to the value of s. |
| 2 | public void updateString(String columnName, String s) throws SQLExceptionSimilar 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.

Updating a row in the result set changes the columns of the current row in the ResultSet object, but not in the underlying database. To update your changes to the row in the database, you need to invoke one of the following methods.

|  |  |
| --- | --- |
| 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. |

# JDBC - SQL Syntax

Structured Query Language (SQL) is a standardized language that allows you to perform operations on a database, such as creating entries, reading content, updating content, and deleting entries.

SQL is supported by almost any database you will likely use, and it allows you to write database code independently of the underlying database.

This chapter gives an overview of SQL, which is a prerequisite to understand JDBC concepts. After going through this chapter, you will be able to Create, Create, Read, Update, and Delete (often referred to as CRUD operations) data from a database.

For a detailed understanding on SQL, you can read our [MySQL Tutorial](https://www.tutorialspoint.com/mysql/index.htm%22%20%5Co%20%22MySQL%20Tutorial%22%20%5Ct%20%22_blank).

## Create Database

The CREATE DATABASE statement is used for creating a new database. The syntax is −

SQL> CREATE DATABASE DATABASE\_NAME;

## Example

The following SQL statement creates a Database named EMP −

SQL> CREATE DATABASE EMP;

## Drop Database

The DROP DATABASE statement is used for deleting an existing database. The syntax is −

SQL> DROP DATABASE DATABASE\_NAME;

Note: To create or drop a database you should have administrator privilege on your database server. Be careful, deleting a database would loss all the data stored in the database.

## Create Table

The CREATE TABLE statement is used for creating a new table. The syntax is −

SQL> CREATE TABLE table\_name

(

 column\_name column\_data\_type,

 column\_name column\_data\_type,

 column\_name column\_data\_type

 ...

);

## Example

The following SQL statement creates a table named Employees with four columns −

SQL> CREATE TABLE Employees

(

 id INT NOT NULL,

 age INT NOT NULL,

 first VARCHAR(255),

 last VARCHAR(255),

 PRIMARY KEY ( id )

);

## Drop Table

The DROP TABLE statement is used for deleting an existing table. The syntax is −

SQL> DROP TABLE table\_name;

## Example

The following SQL statement deletes a table named Employees −

SQL> DROP TABLE Employees;

## INSERT Data

The syntax for INSERT, looks similar to the following, where column1, column2, and so on represents the new data to appear in the respective columns −

SQL> INSERT INTO table\_name VALUES (column1, column2, ...);

## Example

The following SQL INSERT statement inserts a new row in the Employees database created earlier −

SQL> INSERT INTO Employees VALUES (100, 18, 'Zara', 'Ali');

## SELECT Data

The SELECT statement is used to retrieve data from a database. The syntax for SELECT is −

SQL> SELECT column\_name, column\_name, ...

 FROM table\_name

 WHERE conditions;

The WHERE clause can use the comparison operators such as =, !=, <, >, <=,and >=, as well as the BETWEEN and LIKE operators.

## Example

The following SQL statement selects the age, first and last columns from the Employees table, where id column is 100 −

SQL> SELECT first, last, age

 FROM Employees

 WHERE id = 100;

The following SQL statement selects the age, first and last columns from the Employees table where *first* column contains *Zara* −

SQL> SELECT first, last, age

 FROM Employees

 WHERE first LIKE '%Zara%';

## UPDATE Data

The UPDATE statement is used to update data. The syntax for UPDATE is −

SQL> UPDATE table\_name

 SET column\_name = value, column\_name = value, ...

 WHERE conditions;

The WHERE clause can use the comparison operators such as =, !=, <, >, <=,and >=, as well as the BETWEEN and LIKE operators.

## Example

The following SQL UPDATE statement changes the age column of the employee whose id is 100 −

SQL> UPDATE Employees SET age=20 WHERE id=100;

## DELETE Data

The DELETE statement is used to delete data from tables. The syntax for DELETE is −

SQL> DELETE FROM table\_name WHERE conditions;

The WHERE clause can use the comparison operators such as =, !=, <, >, <=,and >=, as well as the BETWEEN and LIKE operators.

## Example

The following SQL DELETE statement deletes the record of the employee whose id is 100 −

SQL> DELETE FROM Employees WHERE id=100;

### Introduction to JSP

JSP technology is used to create dynamic web applications. JSP pages are easier to maintain then a Servlet. JSP pages are opposite of Servlets as a servlet adds HTML code inside Java code, while JSP adds Java code inside HTML using JSP tags. Everything a Servlet can do, a JSP page can also do it.

JSP enables us to write HTML pages containing tags, inside which we can include powerful Java programs. Using JSP, one can easily separate Presentation and Business logic as a web designer can design and update JSP pages creating the presentation layer and java developer can write server side complex computational code without concerning the web design. And both the layers can easily interact over HTTP requests.

#### In the end a JSP becomes a Servlet

JSP pages are converted into Servlet by the Web Container. The Container translates a JSP page into servlet class source(.java) file and then compiles into a Java Servlet class.

#### Why JSP is preffered over servlets?

* JSP provides an easier way to code dynamic web pages.
* JSP does not require additional files like, java class files, web.xml etc
* Any change in the JSP code is handled by Web Container(Application server like tomcat), and doesn't require re-compilation.
* JSP pages can be directly accessed, and web.xml mapping is not required like in servlets.

#### Advantage of JSP

* Easy to maintain and code.
* High Performance and Scalability.
* JSP is built on Java technology, so it is platform independent.
* **JSP** technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.
* A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc.

### Advantages of JSP over Servlet

* There are many advantages of JSP over the Servlet. They are as follows:

#### 1) Extension to Servlet

* JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

#### 2) Easy to maintain

* JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.

#### 3) Fast Development: No need to recompile and redeploy

* If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

#### 4) Less code than Servlet

* In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

### The Lifecycle of a JSP Page

The JSP pages follow these phases:

* Translation of JSP Page
* Compilation of JSP Page
* Classloading (the classloader loads class file)
* Instantiation (Object of the Generated Servlet is created).
* Initialization ( the container invokes jspInit() method).
* Request processing ( the container invokes \_jspService() method).
* Destroy ( the container invokes jspDestroy() method).



As depicted in the above diagram, JSP page is translated into Servlet by the help of JSP translator. The JSP translator is a part of the web server which is responsible for translating the JSP page into Servlet. After that, Servlet page is compiled by the compiler and gets converted into the class file. Moreover, all the processes that happen in Servlet are performed on JSP later like initialization, committing response to the browser and destroy.

### Creating a simple JSP Page

To create the first JSP page, write some HTML code as given below, and save it by .jsp extension. We have saved this file as index.jsp. Put it in a folder and paste the folder in the web-apps directory in apache tomcat to run the JSP page.

**index.jsp**

Let's see the simple example of JSP where we are using the scriptlet tag to put Java code in the JSP page. We will learn scriptlet tag later.

<html>

<body>

<% out.print(2\*5); %>

</body>

</html>

It will print **10** on the browser.

### How to run a simple JSP Page?

Follow the following steps to execute this JSP page:

* Start the server
* Put the JSP file in a folder and deploy on the server
* Visit the browser by the URL http://localhost:portno/contextRoot/jspfile, for example, http://localhost:8888/myapplication/index.jsp

### JSP Scripting Element

JSP Scripting element are written inside <% %> tags. These code inside <% %> tags are processed by the JSP engine during translation of the JSP page. Any other text in the JSP page is considered as HTML code or plain text.

Example:

<html>

 <head>

 <title>My First JSP Page</title>

 </head>

 <%

 int count = 0;

 %>

 <body>

 Page Count is <% out.println(++count); %>

 </body>

Just to experiment, try removing the <% %> scriplet tag from the above code and run it as JSP. You will see that everything is printed as it is on the browser, because without the scriplet tag, everything is considered plain HTML code.

#### There are five different types of scripting elements

|  |  |
| --- | --- |
| Scripting Element | Example |
| Comment | <%-- comment --%> |
| Directive | <%@ directive %> |
| Declaration | <%! declarations %> |
| Scriptlet | <% scriplets %> |
| Expression | <%= expression %> |

#### JSP Comment

JSP Comment is used when you are creating a JSP page and want to put in comments about what you are doing. JSP comments are only seen in the JSP page. These comments are not included in servlet source code during translation phase, nor they appear in the HTTP response. Syntax of JSP comment is as follows :

<%-- JSP comment --%>

Simple Example of JSP Comment

<html>

 <head>

 <title>My First JSP Page</title>

 </head>

 <%

 int count = 0;

 %>

 <body>

 *<%-- Code to show page count --%>*

 Page Count is <% out.println(++count); %>

 </body>

### Implicit Objects in JSP

JSP provide access to some implicit object which represent some commonly used objects for servlets that JSP page developers might need to use. For example you can retrieve HTML form parameter data by using request variable, which represent the HttpServletRequest object.

#### Following are the JSP implicit object

|  |  |
| --- | --- |
| Implicit Object | Description |
| request | The HttpServletRequest object associated with the request. |
| response | The HttpServletRequest object associated with the response that is sent back to the browser. |
| out | The JspWriter object associated with the output stream of the response. |
| session | The HttpSession object associated with the session for the given user of request. |
| application | The ServletContext object for the web application. |
| config | The ServletConfig object associated with the servlet for current JSP page. |
| pageContext | The PageContext object that encapsulates the enviroment of a single request for this current JSP page |
| page | The page variable is equivalent to this variable of Java programming language. |
| exception | The exception object represents the Throwable object that was thrown by some other JSP page. |

All of them are very useful and you will slowly get to know all of them as you will move ahead into your career working on live projects. For example: When you will create an application where in User sessions have to be created session will come into picture, request is used when you have form submissions in your application etc.

# MVC in JSP

1. [MVC in JSP](https://www.javatpoint.com/MVC-in-jsp)
2. [Example of following MVC in JSP](https://www.javatpoint.com/MVC-in-jsp#mvcex)

MVC stands for Model View and Controller. It is a design pattern that separates the business logic, presentation logic and data.

Controller acts as an interface between View and Model. Controller intercepts all the incoming requests.

Model represents the state of the application i.e. data. It can also have business logic.

View represents the presentaion i.e. UI(User Interface).

#### Advantage of MVC (Model 2) Architecture

1. Navigation Control is centralized
2. Easy to maintain the large application



### MVC Example in JSP

In this example, we are using servlet as a controller, jsp as a view component, Java Bean class as a model.

In this example, we have created 5 pages:

* **index.jsp** a page that gets input from the user.
* **ControllerServlet.java** a servlet that acts as a controller.
* **login-success.jsp** and **login-error.jsp** files acts as view components.
* **web.xml** file for mapping the servlet.

*File: index.jsp*

<form action="ControllerServlet" method="post">

Name:<input type="text" name="name"><br>

Password:<input type="password" name="password"><br>

<input type="submit" value="login">

</form>

*File: ControllerServlet*

package com.javatpoint;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.RequestDispatcher;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class ControllerServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)

            throws ServletException, IOException {

        response.setContentType("text/html");

        PrintWriter out=response.getWriter();

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

        String password=request.getParameter("password");

        LoginBean bean=new LoginBean();

        bean.setName(name);

        bean.setPassword(password);

        request.setAttribute("bean",bean);

        boolean status=bean.validate();

        if(status){

            RequestDispatcher rd=request.getRequestDispatcher("login-success.jsp");

            rd.forward(request, response);

        }

        else{

            RequestDispatcher rd=request.getRequestDispatcher("login-error.jsp");

            rd.forward(request, response);

        }

    }

    @Override

    protected void doGet(HttpServletRequest req, HttpServletResponse resp)

            throws ServletException, IOException {

        doPost(req, resp);

    }

}

*File: LoginBean.java*

package com.javatpoint;

public class LoginBean {

private String name,password;

public String getName() {

    return name;

}

public void setName(String name) {

    this.name = name;

}

public String getPassword() {

    return password;

}

public void setPassword(String password) {

    this.password = password;

}

public boolean validate(){

    if(password.equals("admin")){

        return true;

    }

    else{

1. return false;

    }

}

}

*File: login-success.jsp*

<%@page import="com.javatpoint.LoginBean"%>

<p>You are successfully logged in!</p>

<%

LoginBean bean=(LoginBean)request.getAttribute("bean");

out.print("Welcome, "+bean.getName());

%>

*File: login-error.jsp*

<p>Sorry! username or password error</p>

<%@ include file="index.jsp" %>

*File: web.xml*

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

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"

id="WebApp\_ID" version="3.0">

  <servlet>

  <servlet-name>s1</servlet-name>

  <servlet-class>com.javatpoint.ControllerServlet</servlet-class>

  </servlet>

  <servlet-mapping>

  <servlet-name>s1</servlet-name>

  <url-pattern>/ControllerServlet</url-pattern>

  </servlet-mapping>

</web-app>

# JSP - Form Processing

In this chapter, we will discuss Form Processing in JSP. You must have come across many situations when you need to pass some information from your browser to the web server and ultimately to your backend program. The browser uses two methods to pass this information to the web server. These methods are the GET Method and the POST Method.

## The Methods in Form Processing

Let us now discuss the methods in Form Processing.

### GET method

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ? character as follows −

http://www.test.com/hello?key1=value1&key2=value2

The GET method is the default method to pass information from the browser to the web server and it produces a long string that appears in your browser's Location:box. It is recommended that the GET method is better not used. if you have password or other sensitive information to pass to the server.

The GET method has size limitation: only 1024 characters can be in a request string.

This information is passed using QUERY\_STRING header and will be accessible through QUERY\_STRING environment variable which can be handled using getQueryString() and getParameter() methods of request object.

### POST method

A generally more reliable method of passing information to a backend program is the POST method.

This method packages the information in exactly the same way as the GET method, but instead of sending it as a text string after a ? in the URL it sends it as a separate message. This message comes to the backend program in the form of the standard input which you can parse and use for your processing.

JSP handles this type of requests using getParameter() method to read simple parameters and getInputStream() method to read binary data stream coming from the client.

## Reading Form Data using JSP

JSP handles form data parsing automatically using the following methods depending on the situation −

* getParameter() − You call request.getParameter() method to get the value of a form parameter.
* getParameterValues() − Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
* getParameterNames() − Call this method if you want a complete list of all parameters in the current request.
* getInputStream() − Call this method to read binary data stream coming from the client.

## GET Method Example Using URL

The following URL will pass two values to HelloForm program using the GET method.

http://localhost:8080/main.jsp?first\_name=ZARA&last\_name=ALI

Below is the main.jsp JSP program to handle input given by web browser. We are going to use the getParameter() method which makes it very easy to access the passed information −

<html>

 <head>

 <title>Using GET Method to Read Form Data</title>

 </head>

 <body>

 <h1>Using GET Method to Read Form Data</h1>

 <ul>

 <li><p><b>First Name:</b>

 <%= request.getParameter("first\_name")%>

 </p></li>

 <li><p><b>Last Name:</b>

 <%= request.getParameter("last\_name")%>

 </p></li>

 </ul>

 </body>

</html>

## GET Method Example Using Form

Following is an example that passes two values using the HTML FORM and the submit button. We are going to use the same JSP main.jsp to handle this input.

<html>

 <body>

 <form action = "main.jsp" method = "GET">

 First Name: <input type = "text" name = "first\_name">

 <br />

 Last Name: <input type = "text" name = "last\_name" />

 <input type = "submit" value = "Submit" />

 </form>

 </body>

</html>

# Exception Handling in JSP

1. [Exception Handling in JSP](https://www.javatpoint.com/exception-handling-in-jsp)
2. [Example of exception handling in jsp by the elements of page directive](https://www.javatpoint.com/exception-handling-in-jsp#jspexcepex1)
3. [Example of exception handling in jsp by specifying the error-page element in web.xml file](https://www.javatpoint.com/exception-handling-in-jsp#jspexcepex2)

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

1. By **errorPage** and **isErrorPage** attributes of page directive
2. By **<error-page>** element in web.xml file

### Example of exception handling in jsp by the elements of page directive

In this case, you must define and create a page to handle the exceptions, as in the error.jsp page. The pages where may occur exception, define the errorPage attribute of page directive, as in the process.jsp page.

There are 3 files:

* index.jsp for input values
* process.jsp for dividing the two numbers and displaying the result
* error.jsp for handling the exception

#### index.jsp

<form action="process.jsp">

No1:<input type="text" name="n1" /><br/><br/>

No1:<input type="text" name="n2" /><br/><br/>

<input type="submit" value="divide"/>

</form>

#### process.jsp

<%@ page errorPage="error.jsp" %>

<%

String num1=request.getParameter("n1");

String num2=request.getParameter("n2");

int a=Integer.parseInt(num1);

int b=Integer.parseInt(num2);

int c=a/b;

out.print("division of numbers is: "+c);

%>

#### error.jsp

<%@ page isErrorPage="true" %>

<h3>Sorry an exception occured!</h3>

Exception is: <%= exception %>

JSP JAVABEAN

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

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

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

## JavaBeans Properties

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

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

|  |  |
| --- | --- |
| S.No. | Method & Description |
| 1 | getPropertyName()For example, if property name is *firstName*, your method name would be getFirstName() to read that property. This method is called accessor. |
| 2 | setPropertyName()For example, if property name is *firstName*, your method name would be setFirstName() to write that property. This method is called mutator. |

A read-only attribute will have only a getPropertyName() method, and a write-only attribute will have only a setPropertyName() method.

## JavaBeans Example

Consider a student class with few properties −

package com.tutorialspoint;

public class StudentsBean implements java.io.Serializable {

 private String firstName = null;

 private String lastName = null;

 private int age = 0;

 public StudentsBean() {

 }

 public String getFirstName(){

 return firstName;

 }

 public String getLastName(){

 return lastName;

 }

 public int getAge(){

 return age;

 }

 public void setFirstName(String firstName){

 this.firstName = firstName;

 }

 public void setLastName(String lastName){

 this.lastName = lastName;

 }

 public void setAge(Integer age){

 this.age = age;

 }

}

## Accessing JavaBeans

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

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

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

Following example shows how to use the useBean action −

<html>

 <head>

 <title>useBean Example</title>

 </head>

 <body>

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

 <p>The date/time is <%= date %>

 </body>

</html>

You will receive the following result − −

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

## Accessing JavaBeans Properties

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

<jsp:useBean id = "id" class = "bean's class" scope = "bean's scope">

 <jsp:setProperty name = "bean's id" property = "property name"

 value = "value"/>

 <jsp:getProperty name = "bean's id" property = "property name"/>

 ...........

</jsp:useBean>

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

Following example shows how to access the data using the above syntax −

<html>

 <head>

 <title>get and set properties Example</title>

 </head>

 <body>

 <jsp:useBean id = "students" class = "com.tutorialspoint.StudentsBean">

 <jsp:setProperty name = "students" property = "firstName" value = "Zara"/>

 <jsp:setProperty name = "students" property = "lastName" value = "Ali"/>

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

 </jsp:useBean>

 <p>Student First Name:

 <jsp:getProperty name = "students" property = "firstName"/>

 </p>

 <p>Student Last Name:

 <jsp:getProperty name = "students" property = "lastName"/>

 </p>

 <p>Student Age:

 <jsp:getProperty name = "students" property = "age"/>

 </p>

 </body>

</html>

## What is JAR?

JAR stands for Java ARchive. It's a file format based on the popular ZIP file format and is used for aggregating many files into one. Although JAR can be used as a general archiving tool, the primary motivation for its development was so that Java applets and their requisite components (.class files, images and sounds) can be downloaded to a browser in a single HTTP transaction, rather than opening a new connection for each piece. This greatly improves the speed with which an applet can be loaded onto a web page and begin functioning. The JAR format also supports compression, which reduces the size of the file and improves download time still further. Additionally, individual entries in a JAR file may be digitally signed by the applet author to authenticate their origin.

JAR is:

* the only archive format that is cross-platform
* the only format that handles audio and image files as well as class files
* backward-compatible with existing applet code
* an open standard, fully extendable, and written in java
* the preferred way to bundle the pieces of a java applet

# What is introspection in JavaBeans?

Introspection is the ability to ask a JavaBean component what properties and events it supports. By default, introspection is supported by reflection, where you name methods with certain naming patterns, like set/getProperty() and add/removeListener(). You can also explicitly expose the bean's behavior through creation of a BeanInfo class related to the bean. The BeanInfo class can be used for many things, like setting up an icon for the component when used in a bean builder tool like JBuilder, limiting the properties available, or mapping methods to properties that don't follow standard naming patterns.

# Connect from database using JSP Bean file

EXAMPLE

package myexample;

import java.io.\*;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.Statement;

import java.sql.ResultSet;

public class bean

{

  private int msgid;

  private String message;

  private Connection connection=null;

  private ResultSet rs = null;

  private Statement st = null;

  String connectionURL = "jdbc:mysql://192.168.10.59/messagepaging";

  public bean()

  {

 try {

 // Load the database driver

  Class.forName("com.mysql.jdbc.Driver");

  // Get a Connection to the database

  connection = DriverManager.getConnection(connectionURL, "root", "root");

  }catch(Exception e){

  System.out.println("Exception is ;"+e);

  }

  }

  public void setmsgid(int msgid)

  {

  this.msgid = msgid;

  }

  public int getmsgid()

  {

  return (this.msgid);

  }

  public void setmessage(String message)

  {

  this.message = message;

  }

  public String getmessage()

  {

  return (this.message);

  }

  public void insert()

  {

 try

 {

  String sql = "insert into message(id,message)

 values('"+msgid+"','"+message+"')";

  Statement s = connection.createStatement();

  s.executeUpdate (sql);

  s.close ();

  }catch(Exception e){

  System.out.println("Exception is ;"+e);

  }

  }

 }

 EXAMPLE:

Print squares and cubes using alert from 5 to 15’

<html>

 <head>

 </head>

 <body>

 <table border='4'>

 <thead>

 <tr>

 <th></th>

 <th> Number </th>

 <th> Squared </th>

 <th> Cubed </th>

 </tr>

 </thead>

 <tbody>

 <tr>

 <td></td>

 <td id="number\_1" align="center">-</td>

 <td id="squared\_1" align="center">-</td>

 <td id="cubed\_1" align="center">-</td>

 </tr>

 </tbody>

 </table>

 </body>

</html>

<script>

 var a = prompt("Please enter a number from 5 - 15! \n", "");

 //Compute da business

 if(!isNaN(a)){

 var sqpt = a \* a;

 var cbpt = a \* a \* a;

 //alert(a);

 document.getElementById("number\_1").innerHTML = a;

 document.getElementById("squared\_1").innerHTML = sqpt;

 document.getElementById("cubed\_1").innerHTML = cbpt;

 }else{

 alert("Please enter a valid Number!");

 }

</script>

FACTORIAL OF A NUMBER EXAMPLE:

<!doctype html>

<html>

<head>

<script>

function show(){

var i, no, fact;

fact=1;

no=Number(document.getElementById("num").value);

for(i=1; i<=no; i++)

{

fact= fact\*i;

}

document.getElementById("answer").value= fact;

}

</script>

</head>

<body>

Enter Num: <input id="num">

<button onclick="show()">Factorial</button>

<input id="answer">

</body>

</html>