**JAVA SERVLETS**

**What is Java Servlet?**

Java Servlet is a server side technology that runs on a web application server. It acts as middle layer between requests coming from clients and other server/databases in a web application. It is used to create dynamic web pages in a web application.

**Brief History**

Pavni Diwanji while working at Sun Microsystems, created the Servlet1 specification and version 1.0 was finalized in June 1997. The specification for version 2.2 was developed by Java Community Process in August 1999. Servlet 3 was released on December 2009. The latest version which is Servlet 4.0 was released on September 2017.

**Installation**

**JDK Installation**

Step 1: To download JAVA, go to their official website

Step 2: Download JDK and JRE, before you can download the JDK and JRE you must first click on the checkbox “Accept the license agreement”

Step 3: Install the JDK by clicking the download file, and just click the “Next” button to proceed

Step 4: After the Java is installed, you need to set the classpath. To set the path just follow this:

* Click the **Start Button**
* Right click on the ***Computer*** icon and choose ***Properties***
* Click the ***Advanced system settings*** which is in the upper left of the window
* Click ***Environment Variables.*** In the *System Variables* section, find the ***PATH*** environment variable and select it. Click **Edit,** add/paste the directory of the *bin folder* of the JDK file installed to the ***PATH*** environment

**Eclipse Installation**

Step 1: To download Eclipse, go to their official website

Step 2: Click the Eclipse download file to install Eclipse

Step 3: After installing, open Eclipse and a window will apear asking where to store the workspace. The workspace is where the collection of projects are located

**Apache Tomcat Installation**

Step 1: Download the latest version of Tomcat

Step 2: After downloading, unzip the apache-tomcat zip file and install

Step 3: Configure the Tomcat Server by editing 4 configuration files (**web.xml, server.xml, tomcat-users.xml and context.xml**)

**Java Servlet Life Cycle**

**Servlet Initialization**

* The ***init()*** method is only called once when the servlet is created.

**Servlet Request Handling**

* The ***service()***method is called to handle requests coming from clients and responds back to clients.
* The method calls the *doGet method* to handle GET requests and the *doPost* method to POST requests.

**Servlet Termination**

* The ***destroy()*** method is called when the life cycle of servlet ends

**Java Servlet Variable Scope**

* **Iteration level** is for current server access only. This is the request and response level (Local Variables: doGet and doPost is used here)
* **User/Session level** is used to store the user specific data
* **Application level** is used to store global data which is going to be shared in the entire application

**Variable**

A variable contains a data value and is assigned with a specific datatype. Here are the three types of variables in Java:

* Local variable – it is declared inside the method
* Instance variable – it is declared inside the class but outside the method and it is not declared as static.
* Static variable – it is declared as static and it cannot be local.

Here is a sample code:

***class****Sample{//start of class*

***int****a=10;//instance variable*

***static******int****b=100;//static variable*

***void****method(){*

***int****c=50;//local variable*

*}*

*}//end of class*

**Data Types**

Data types are classification of values stored in the variable. There are two types of data types in Java:

* Primitive data types
  + Under this data type are Strings, Arrays etc.
* Non-primitive data types
  + Under this data type are Boolean and Numeric.

**Array**

Set of multiple elements with similar data type is called array. There are two types of array:

* Single Dimensional Array

Here are three different syntax to declare an array:

* *dataType[] array\_name;*
* *dataType []array\_name;*
* *dataType array\_name[];*
* Multidimensional Array

Here are four different syntax to declare multidimensional array:

* *dataType[][] arrRefVar;*
* *dataType [][]arrRefVar;*
* *dataType arrRefVar[][];*
* *dataType []arrRefVar[];*

**Java if Statement**

The Java *if statement* is used to test the condition. It checks boolean condition: *true* or *false*.

Common types of *if* statement:

* *if statement*
* *if-else statement*
* *if-else-if ladder*

**if Statement**

The Java *if* statement tests the condition and it executes the *if block* if the condition is true.

Here is a syntax of *if* statement:

*if(condition){*

*//code to be executed if condition is true*

*}*

**if-else Statement**

It executes the *if block* if condition is true otherwise if false *else block* is executed.

Here is a syntax of *if-else* statement:

***if****(condition){*

*//code to be executed if condition is true*

*}****else****{*

*//code to be executed if condition is false*

*}*

**if-else-if ladder Statement**

Executes one condition from multiple statements.

Here is a syntax of *if-else-if ladder* statement:

***if****(condition1){*

*//code to be executed if condition1 is true*

*}****else******if****(condition2){*

*//code to be executed if condition2 is true*

*}*

***else******if****(condition3){*

*//code to be executed if condition3 is true*

*}*

***else****{*

*//code to be executed if all the conditions are false*

*}*

**Java Switch**

The *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement**.**

Here is a syntax for *switch* statement:

***switch****(expression){*

***case****value1:*

*//code*

***break****;  //optional*

***case****value2:*

*//code*

***break****;  //optional*

*......*

***default****:*

*//code to be executed if all cases are not matched*

*}*

**Java For Loop**

The *for loop* is used to iterate a part of the program several times. It is recommended to use for loop if the number of iteration is fixed.

Here is a syntax for *for loop*:

***for****(initialization;condition;increment/decrement){*

*//code*

*}*

**Java While Loop**

The *while loop* is used to iterate a part of the program several times. It is recommended to use while loop if the number of iteration is not fixed.

Here is a syntax for *while loop*:

***while****(condition){*

*//code*

*}*

**Java Do-While Loop**

The *do-while loop* is used to iterate a part of the program several times. The loop is executed at least once because condition is checked after loop body.

Here is a syntax *for do-while loop*:

***do****{*

*//code*

*}****while****(condition);*

**Java AND Operator**

The logical “&&” operator only checks the second condition if the first condition is true.

The bitwise “&” operator always checks both conditions whether first condition is true or false.

Here is a sample code using the AND operator:

***class****OperatorExample1{*

***public******static******void****main(String args[]){*

***int****a=10;*

***int****b=5;*

***int****c=20;*

*System.out.println(a>b&&a<c);//true && true = true*

*System.out.println(a<b&a<c);//false & true = false*

*}}*

**Java OR Operator**

The logical “||” operator only checks the second condition if the first condition is false.

The bitwise “|” operator always checks both conditions whether first condition is true or false.

Here is a sample code using the OR operator:

***class****OperatorExample2{*

***public******static******void****main(String args[]){*

***int****a=10;*

***int****b=5;*

***int****c=20;*

*System.out.println(a<b||a<c);//false || true = true*

*System.out.println(a>b|a<c);//true | true = true*

*}}*

**HTTP Servlet**

HttpServlet is an abstract class. It provides methods such as doGet, doPost, and many more. Here are some methods in HttpServlet:

* **doGet** methodhandles the GET request.
* **doPost** methodhandles the POST request.
* **doHead** methodhandles the HEAD request.
* **doOptions** methodhandles the OPTIONS request.
* **doPut** method handles the PUT request.
* **doTrace** methodhandles the TRACE request.
* **doDelete** methodhandles the DELETE request.
* **getLastModified** methodreturns the time when HttpServletRequest was last modified

**HTTP Request**

**Parameters**

Request parameters are request from clients from the browser. The request is sent as a part of the URL or part of the body of an HTTP request.

**Headers**

Request header consists of its name followed by a colon and its value pairs. It allows the client and the server to pass the request or response.

Sample Code:

*String contentLength = request.getHeader("Content-Length");*

**InputStream**

A client sends a request from the browser. To be able to access the request body, you can use InputStream directing it to the request body.

Sample Code:

*InputStream requestBodyInput = request.getInputStream();*

**Session**

A session object holds the data about a user in each request.

Sample Code:

*HttpSession session = request.getSession();*

**Servlet Context**

The ServletContext holds information about the web application.

Sample Code:

*ServletContext context = request.getSession().getServletContext();*

**HTTP Response**

**Writing HTML**

You can obtain the PrintWriter from the HttpResponse object to be able to send back HTML to the browser like this:

*PrintWriter writer = response.getWriter();*

*writer.write("<html><body>GET/POST response</body></html>");*

**Headers**

The HttpRequest contains HTTP headers. Before any data is written on the response, you must first set the headers like this:

*response.setHeader("Header-Name", "Header Value");*

**Content-Type**

It is a response header that indicates the media type of the content that the browser is responding to the client.

To obtain the type of a content from a response header just execute this command:

*response.setHeader("Content-Type", "text/html");*

**Writing Text**

Instead of HTML, you can give back the text file only to the browser like this:

*response.setHeader("Content-Type", "text/plain");*

*PrintWriter writer = response.getWriter();*

*writer.write("This is just plain text!");*

**Redirecting to a different web address**

From your servlet, you can change and redirect the web addres like this:

*response.sendRedirect("http://www.google.com");*

**Creating a Servlet using Tomcat Server**

To create a servlet, we will use **Apache Tomcat Server**. Follow this steps:

* First, create a directory structure
* Second, create a Servlet

The servlet can be created in any of these three ways:

* By implementing Servlet interface,
* By inheriting GenericServlet class
* By inheriting HttpServlet class
* Third, compile the Servlet
* Fourth, create a deployment configuration file
* Fifth, start the server and project
* Lastly, access the servlet

**What is JSP?**

Java Server Pages or also known as JSP is a server-side programming technology that is used to create dynamic web pages just like Servlet.

**Life Cycle Methods of JSP**

**JSP Initialization**

* **jsp*Init()*** method

**JSP Request Handling**

* **jsp*Service()***method

**JSP Termination**

* ***jspDestroy()*** method

**JSP Scripting Elements**

The scripting elements lets you insert java code inside the JSP. Here are the three types of scripting elements:

* Scriptlet tag is used to execute java source code in JSP.

Here is the sample syntax:

*<%  java source code %>*

* Expression tag is mainly used to print the values of variable or method.

Here is the sample syntax:

***<****%=  statement %****>***

* Declaration tag is used *to declare fields and methods*.

Here is the sample syntax:

***<****%!  field or method declaration %****>***

**JSP Directive Elements**

The **JSP directives** command the web container how to convert a JSP page into the corresponding servlet.

There are three types of directives:

* page directive
* include directive
* taglib directive

**JSP page directive** defines attributes that apply to an entire JSP page.

Here is the sample syntax for page directive:

*<%@ page attribute="value"* %>

**JSP include directive** isused to include the contents of any resource it may be jsp file, html file or text file.

Here is the sample syntax for include directive:

*<%@ include file="resourceName" %>*

**JSP taglib directive** is used to define a tag library that defines many tags.

Here is the sample syntax for taglib directive:

*<%@ taglib uri="uriofthetaglibrary" prefix="prefixoftaglibrary"* %>