# Introduction to Java Servlets

Today we all are aware of the need of creating dynamic web pages i.e the ones which have the capability to change the site contents according to the time or are able to generate the contents according to the request received by the client. If you like coding in Java, then you will be happy to know that using Java there also exists a way

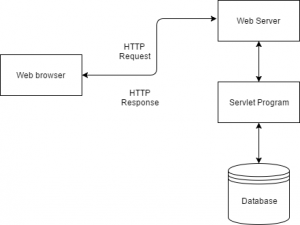
to generate dynamic web pages and that way is Java Servlet. But before we move forward with our topic let’s first understand the need for server-side extensions.

Servlets are the Java programs that run on the Java-enabled web server or application server. They are used to handle the request obtained from the webserver, process the request, produce the response, then send a response back to the webserver.

Properties of Servlets are as follows:

* Servlets work on the server-side.
* Servlets are capable of handling complex requests obtained from the webserver.

Servlet Architecture is can be depicted from the image itself as provided below as follows:



Execution of Servlets basicallyinvolves six basic steps:

1. The clients send the request to the webserver.
2. The web server receives the request.
3. The web server passes the request to the corresponding servlet.
4. The servlet processes the request and generates the response in the form of output.
5. The servlet sends the response back to the webserver.
6. The web server sends the response back to the client and the client browser displays it on the screen.

Now let us do discuss eccentric point that why do we need For Server-Side extensions?

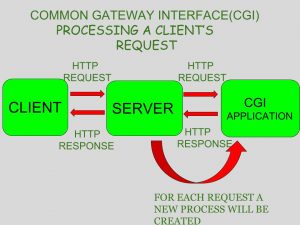
The **server-side extensions** are nothing but the technologies that are used to create dynamic Web pages. Actually, to provide the facility of dynamic Web pages, Web pages need a container or Web server. To meet this requirement, independent Web server providers offer some proprietary solutions in the form of **APIs**(Application Programming Interface).   
These **APIs** allow us to build programs that can run with a Web server. In this case, **Java Servlet** is also one of the component APIs of **Java Platform Enterprise Edition** which sets standards for creating dynamic Web applications in Java.

Before learning about something, it’s important to know the need for that something, it’s not like that this is the only technology available for creating dynamic Web pages. The Servlet technology is similar to other Web server extensions such as **Common Gateway Interface**(CGI) scripts and **Hypertext Preprocessor** (PHP). However, Java Servlets are more acceptable since they solve the limitations of **CGI** such as low performance and low degree scalability.

**What is CGI?**

**CGI** is actually an external application that is written by using any of the programming languages like **C** or **C++** and this is responsible for processing client requests and generating dynamic content.   
In CGI application, when a client makes a request to access dynamic Web pages, the Web server performs the following operations :

* It first locates the requested web page *i.e* the required CGI application using URL.
* It then creates a new process to service the client’s request.
* Invokes the CGI application within the process and passes the request information to the application.
* Collects the response from the CGI application.
* Destroys the process, prepares the HTTP response, and sends it to the client.



So, in **CGI** server has to create and destroy the process for every request. It’s easy to understand that this approach is applicable for handling few clients but as the number of clients increases, the workload on the server increases and so the time is taken to process requests increases.

**Difference between Servlet and CGI**

| Servlet | CGI(Common Gateway Interface) |
| --- | --- |
| Servlets are portable and efficient. | CGI is not portable |
| In Servlets, sharing data is possible. | In CGI, sharing data is not possible. |
| Servlets can directly communicate with the webserver. | CGI cannot directly communicate with the webserver. |
| Servlets are less expensive than CGI. | CGI is more expensive than Servlets. |
| Servlets can handle the cookies. | CGI cannot handle the cookies. |

**Servlets API’s:**   
Servlets are build from two packages:

* javax.servlet(Basic)
* javax.servlet.http(Advance)

Various classes and interfaces present in these packages are:

| Component | Type | Package |
| --- | --- | --- |
| Servlet | Interface | javax.servlet.\* |
| ServletRequest | Interface | javax.servlet.\* |
| ServletResponse | Interface | javax.servlet.\* |
| GenericServlet | Class | javax.servlet.\* |
| HttpServlet | Class | javax.servlet.http.\* |
| HttpServletRequest | Interface | javax.servlet.http.\* |
| HttpServletResponse | Interface | javax.servlet.http.\* |
| Filter | Interface | javax.servlet.\* |
| ServletConfig | Interface | javax.servlet.\* |

**Advantages of a Java Servlet**

* Servlet is **faster** than CGI as it doesn’t involve the creation of a new process for every new request received.
* Servlets, as written in Java, are platform-independent.
* Removes the overhead of creating a **new process** for each request as Servlet doesn’t run in a separate process. There is only a single instance that handles all requests concurrently. This also saves the memory and allows a Servlet to easily manage the client state.
* It is a server-side component, so Servlet inherits the **security** provided by the Web server.
* The **API** designed for Java Servlet automatically acquires the advantages of the Java platforms such as platform-independent and portability. In addition, it obviously can use the wide range of APIs created on Java platforms such as **JDBC** to access the database.
* Many Web servers that are suitable for personal use or low-traffic websites are offered for free or at extremely **cheap costs** eg. Java servlet. However, the majority of commercial-grade Web servers are rather expensive, with the notable exception of Apache, which is free.

**The Servlet Container**

**Servlet container**, also known as **Servlet engine** is an integrated set of objects that provide a run time environment for Java Servlet components.

In simple words, it is a system that manages Java Servlet components on top of the Web server to handle the Web client requests.

**Services provided by the Servlet container :**

* **Network Services:** Loads a Servlet class. The loading may be from a local file system, a remote file system or other network services. The Servlet container provides the network services over which the request and response are sent.
* **Decode and Encode MIME-based messages:** Provides the service of decoding and encoding MIME-based messages.
* **Manage Servlet container:** Manages the lifecycle of a Servlet.
* **Resource management**Manages the static and dynamic resources, such as HTML files, Servlets, and JSP pages.
* **Security Service:** Handles authorization and authentication of resource access.
* **Session Management:** Maintains a session by appending a **session ID** to the URL path.

# Life Cycle of a Servlet

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 19 Oct, 2021

 Read

 Discuss

The entire life cycle of a Servlet is managed by the **Servlet container** which uses the **javax.servlet.Servlet** interface to understand the Servlet object and manage it. So, before creating a Servlet object, let’s first understand the life cycle of the Servlet object which is actually understanding how the Servlet container manages the Servlet object.

**Stages of the Servlet Life Cycle**: The Servlet life cycle mainly goes through four stages,

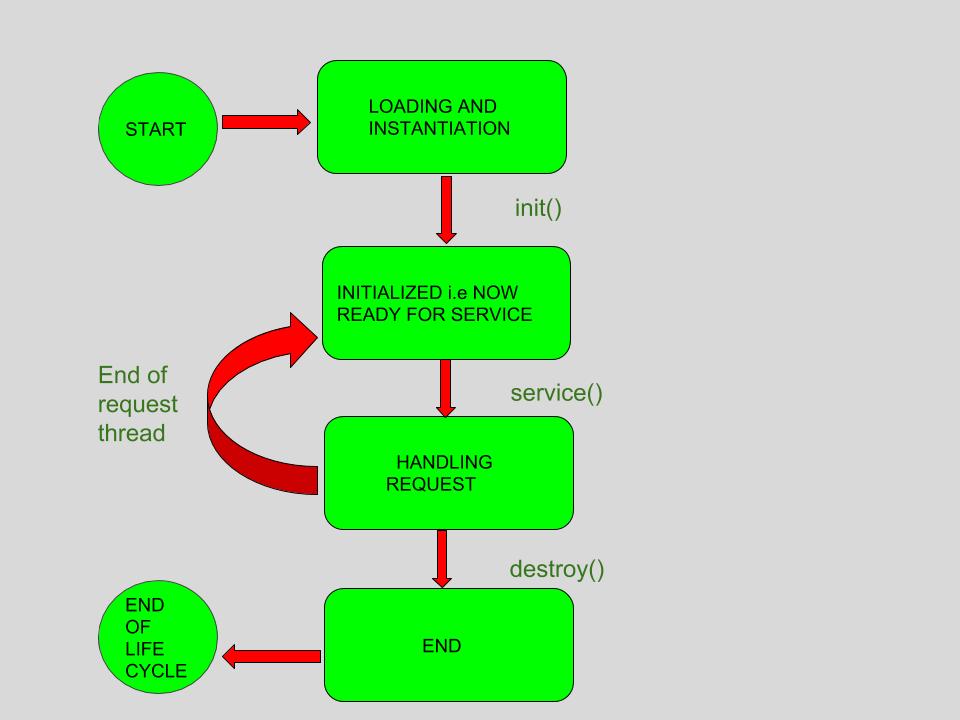
* Loading a Servlet.
* Initializing the Servlet.
* Request handling.
* Destroying the Servlet.

Let’s look at each of these stages in details:

1. **Loading a Servlet**: The first stage of the Servlet lifecycle involves loading and initializing the Servlet by the Servlet container. The Web container or Servlet Container can load the Servlet at either of the following two stages :
   * Initializing the context, on configuring the Servlet with a zero or positive integer value.
   * If the Servlet is not preceding stage, it may delay the loading process until the Web container determines that this Servlet is needed to service a request.

The Servlet container performs two operations in this stage :

* + **Loading :** Loads the Servlet class.
  + **Instantiation :** Creates an instance of the Servlet. To create a new instance of the Servlet, the container uses the no-argument constructor.



1. **Initializing a Servlet**: After the Servlet is instantiated successfully, the Servlet container initializes the instantiated Servlet object. The container initializes the Servlet object by invoking the **Servlet.init(ServletConfig)** method which accepts ServletConfig object reference as parameter.

The Servlet container invokes the **Servlet.init(ServletConfig)** method only once, immediately after the **Servlet.init(ServletConfig)** object is instantiated successfully. This method is used to initialize the resources, such as JDBC datasource.

Now, if the Servlet fails to initialize, then it informs the Servlet container by throwing the **ServletException** or **UnavailableException**.

1. **Handling request**: After initialization, the Servlet instance is ready to serve the client requests. The Servlet container performs the following operations when the Servlet instance is located to service a request :
   * It creates the **ServletRequest** and **ServletResponse** objects. In this case, if this is a HTTP request, then the Web container creates **HttpServletRequest** and **HttpServletResponse** objects which are subtypes of the **ServletRequest** and **ServletResponse** objects respectively.
   * After creating the request and response objects it invokes the Servlet.service(ServletRequest, ServletResponse) method by passing the request and response objects.

The **service()** method while processing the request may throw the **ServletException** or **UnavailableException** or **IOException**.

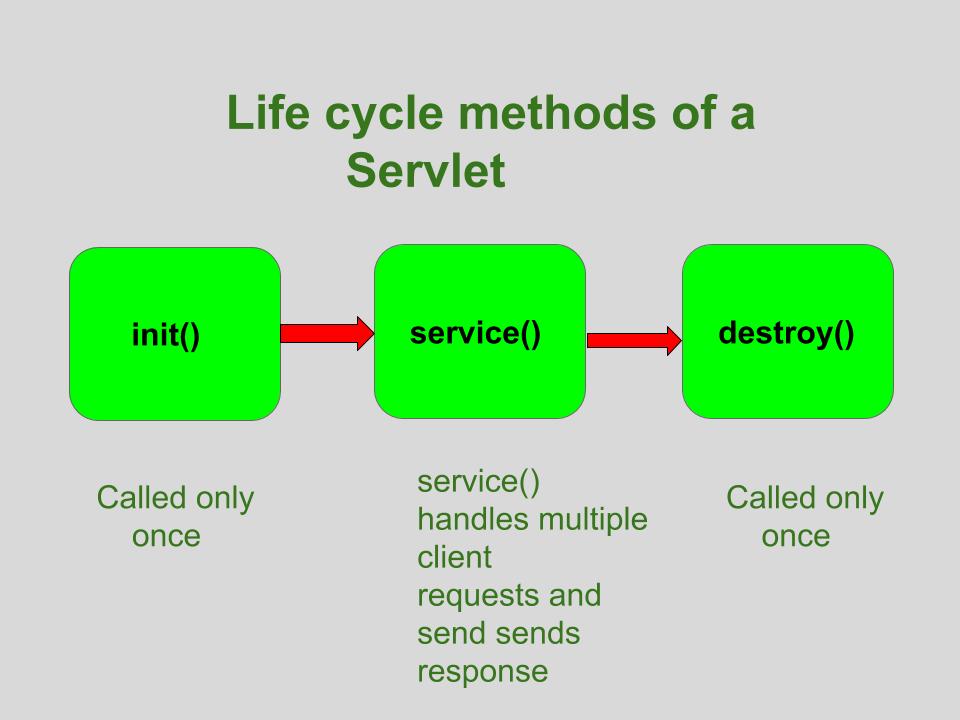
1. **Destroying a Servlet**: When a Servlet container decides to destroy the Servlet, it performs the following operations,
   * It allows all the threads currently running in the service method of the Servlet instance to complete their jobs and get released.
   * After currently running threads have completed their jobs, the Servlet container calls the **destroy()** method on the Servlet instance.

After the **destroy()** method is executed, the Servlet container releases all the references of this Servlet instance so that it becomes eligible for garbage collection.

**Servlet Life Cycle Methods**

There are three life cycle methods of a Servlet :

* init()
* service()
* destroy()



Let’s look at each of these methods in details:

1. **init() method**: The **Servlet.init()** method is called by the Servlet container to indicate that this Servlet instance is instantiated successfully and is about to put into service.
2. //init() method
3. public class MyServlet implements Servlet{
4. public void init(ServletConfig config) throws ServletException {
5. //initialization code
6. }
7. //rest of code
8. }
9. **service() method**: The **service()** method of the Servlet is invoked to inform the Servlet about the client requests.
   * This method uses **ServletRequest** object to collect the data requested by the client.
   * This method uses **ServletResponse** object to generate the output content.
10. // service() method
11. public class MyServlet implements Servlet{
12. public void service(ServletRequest res, ServletResponse res)
13. throws ServletException, IOException {
14. // request handling code
15. }
16. // rest of code
17. }
18. **destroy() method**: The **destroy()** method runs only once during the lifetime of a Servlet and signals the end of the Servlet instance.
19. //destroy() method
20. public void destroy()

As soon as the **destroy()** method is activated, the Servlet container releases the Servlet instance.

**Servlet Life Cycle:**  
Servlet life cycle can be defined as the stages through which the servlet passes from its creation to its destruction.  
The servlet life cycle consists these stages:

* Servlet is borned
* Servlet is initialized
* Servlet is ready to service
* Servlet is servicing
* Servlet is not ready to service
* Servlet is destroyed

**Life cycle methods:**  
Life cycle methods are those methods which are used to control the life cycle of the servlet. These methods are called in specific order during the servlets’s entire life cycle.  
The class **Servlet** provides the methods to control and supervise the life cycle of servlet. There are three life cycle methods in the Servlet interface. There are as follows:

* **init() method :**
  1. A servlet’s life begins here .
  2. This method is called only once to load the servlet.Since it is called only once in it’s lifetime,therefore “connected architecture” code is written inside it because we only want once to get connected with the database.  
     **Now Question Arises is that:-**  
     Q.Why can’t we write connected architecture code inside the constructor, since constructor also run only once in it’s entire life?  
     Ans. Suppose if the connection doesn’t get established, then we can throw an exception from init() and the rest of the steps stop executing. But in the constructor we can’t use, throw in it’s prototype otherwise it is an error.
  3. This method receives only one parameter, i.e **ServletConfig** object.
  4. This method has the possibility to throw the ServletException.
  5. Once the servlet is initialized, it is ready to handle the client request.
  6. The prototype for the init() method:

public void init(ServletConfig con)throws ServletException{ }

where **con** is ServletConfig object

* **NOTE:-** In programs of servlet,we use non parameterized version of init().
* **Now,Question Arises is that:-**  
  Q. Why it is recommended to use the non parameterized version of init() instead of parameterized version as seen above?  
  Ans. To answer this, we have to go into detail. Think like developers,i.e there must be some valid reason for this and the answer will blow your mind. Coming to answer:
* **APPROACH 1**  
  Whenever the lifecycle method of a servlet starts executing,i.e when public void init(ServletConfig con) throws ServletException gets call then our class public void init(ServletConfig con) throws ServletException gets called but we have to run the code which initializes servlet config object which is written inside “HttpServlet” method public void init(ServletConfig con) throws ServletException,i.e:  
  Coding of HttpServlet class be like:
* public void init(ServletConfig con) throws ServletException
* {
* //code to initialise ServletConfig object
* init(); //This HttpServlet has 2 init() one which is parameterized and the other one is non
* //parameterized.But this non parameterized version of init() has a blank body.
* //So this call is useless.
* }
* Now see the coding of our class
* public void init(ServletConfig con) throws ServletException
* {
* super.init(con); //Since,our class init() will run first,but to run HttpServlet init() we
* // have used super keyword.And Database connectivity code will be their
* }
* **NOTE:-** As we can see, total 3 init() calls we have to make.First init() gets called of our class then of HttpServlet class then non parameterized version of HttpServlet class.
* But now, we will achieve the same thing with less number of calls:
* **APPROACH 2**  
  Coding of HttpServlet parametrized and non parameterized versions of init() will remain the same. **But in our class instead of overriding parameterized version of init(), we will override non parameterized version of init()**.
* Let’s see the coding of our class non parameterized version of init():
* public void init() throws ServletException
* {
* //database connectivity code
* }
* **NOTE:** Since this method public void init() throws ServletException ,we have override from HttpServlet class whose coding is like:
* public void init() throws ServletException
* {
* //empty body
* }
* Since it’s body is blank, therefore it is known as **“Helper method”** as it is used for overriding purpose.
* Now, as the servlet starts executing its methods, it will call the parameterized version of init(). Since we have not to override the parameterized version, therefore it will give a call to the HttpServlet parameterized version of init(). Since coding of a parameterized version of init() of HttpServlet is as same as above, therefore, from there on it will call init() (i.e non parameterized version of init). It will give a call to our class non parameterized version of init() and the code continues.  
  **Now, as you can see, total number of init() calls are 2 which is less than the first approach. Therefore, execution time is less in 2nd approach and less headache for CPU for maintaining stack and it’s speed increases as compared to 1st approach.**  
  Therefore, it is highly recommended to override non parameterized version of init().Although both will run but due to efficiency first approach is rarely used and also in first approach we have to use super keyword too.Therefore in below mentioned program,we have override non parameterized version of init().
* **service() method :**
  1. The service() method is the most important method to perform that provides the connection between client and server.
  2. The web server calls the service() method to handle requests coming from the client( web browsers) and to send response back to the client.
  3. This method determines the type of Http request (GET, POST, PUT, DELETE, etc.) .
  4. This method also calls various other methods such as doGet(), doPost(), doPut(), doDelete(), etc. as required.
  5. This method accepts two parameters.
  6. The prototype for this method:
  7. public void service(ServletRequest req, ServletResponse resp)

throws ServletException, IOException { }

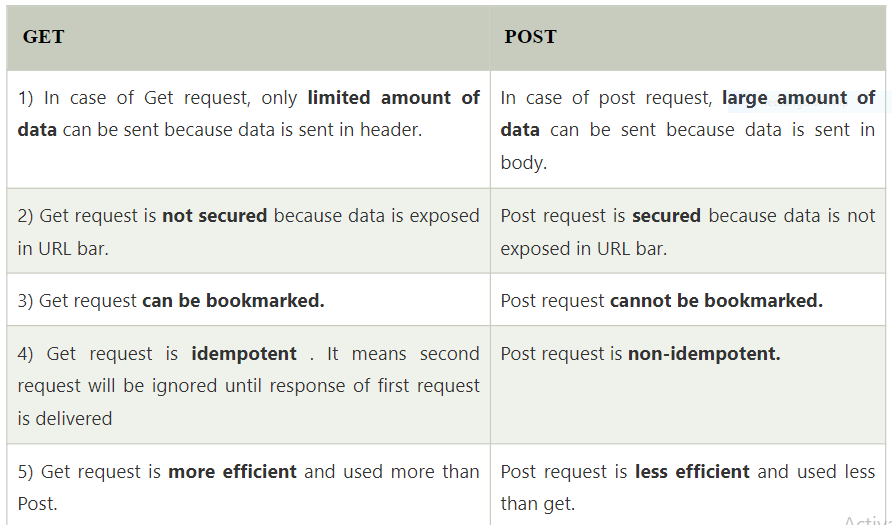
where

* + - **req** is the ServletRequest object which encapsulates the connection from client to server
    - **resp** is the ServletResponse object which encapsulates the connection from server back to the client
* **destroy() method :**
  1. The destroy() method is called only once.
  2. It is called at the end of the life cycle of the servlet.
  3. This method performs various tasks such as closing connection with the database, releasing memory allocated to the servlet, releasing resources that are allocated to the servlet and other cleanup activities.
  4. When this method is called, the garbage collector comes into action.
  5. The prototype for this method is:

public void destroy() { // Finalization code...}

Below is a sample program to illustrate Servlet in Java:

|  |
| --- |
| // Java program to show servlet example  // Importing required Java libraries  import java.io.\*;  import javax.servlet.\*;  import javax.servlet.http.\*;    // Extend HttpServlet class  public class AdvanceJavaConcepts extends HttpServlet  {     private String output;       // Initializing servlet     public void init() throws ServletException     {        output = "Advance Java Concepts";     }       // Requesting and printing the output     public void doGet(HttpServletRequest req,                      HttpServletResponse resp)        throws ServletException, IOException        {           resp.setContentType("text/html");           PrintWriter out = resp.getWriter();           out.println(output);        }          public void destroy()        {           System.out.println("Over");        }  } |



|  |  |
| --- | --- |
| **HTTP GET** | **HTTP POST** |
| In GET method we can not send large amount of data rather limited data is sent because the request parameter is appended into the URL. | In POST method large amount of data can be sent because the request parameter is appended into the body. |
| GET request is comparatively better than Post so it is used more than the  Post request. | POST request is comparatively less better than Get so it is used less than the Get request. |
| GET request is comparatively less secure because the data is exposed in the URL bar. | POST request is comparatively more secure because the data is not exposed in the URL bar. |
| Request made through GET method are stored in Browser history. | Request made through POST method is not stored in Browser history. |
| GET method request can be saved as bookmark in browser. | POST method request can not be saved as bookmark in browser. |
| Request made through GET method are stored in cache memory of Browser. | Request made through POST method are not stored in cache memory of Browser. |
| Data passed through GET method can be easily stolen by attackers. | Data passed through POST method can not be easily stolen by attackers. |
| In GET method only ASCII characters are allowed. | In POST method all types of data is allowed. |
| **PUT Request** | **DELETE Request** |
| It is used to Create or Modify a resource. | It is used to delete a resource identified by a URL. |
| It is idempotent. | It is also idempotent. |
| On successful resource creation, HTTP success code 201(Created). | On successful deletion of record, we can see 200 (OK) or 204 (No Content). |

# RequestDispatcher in Servlet

The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource also. It is one of the way of servlet collaboration.

There are two methods defined in the RequestDispatcher interface.

### Methods of RequestDispatcher interface

The RequestDispatcher interface provides two methods. They are:

1. **public void forward(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
2. **public void include(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**Includes the content of a resource (servlet, JSP page, or HTML file) in the response.



Play Videox

As you see in the above figure, response of second servlet is sent to the client. Response of the first servlet is not displayed to the user.



|  |
| --- |
| As you can see in the above figure, response of second servlet is included in the response of the first servlet that is being sent to the client.   Session Tracking in Servlets  1. [Session Tracking](https://www.javatpoint.com/session-tracking-in-servlets#session1) 2. [Session Tracking Techniques](https://www.javatpoint.com/session-tracking-in-servlets#session1tech)   **Session** simply means a particular interval of time.  **Session Tracking** is a way to maintain state (data) of an user. It is also known as **session management** in servlet.  Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.  HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below: |

# Introduction to JSP

**Introduction**

* It stands for **Java Server Pages**.
* It is a server side technology.
* It is used for creating web application.
* It is used to create dynamic web content.
* In this JSP tags are used to insert JAVA code into HTML pages.
* It is an advanced version of Servlet Technology.
* It is a Web based technology helps us to create dynamic and platform independent web pages.
* In this, Java code can be inserted in HTML/ XML pages or both.
* JSP is first converted into servlet by JSP container before processing the client’s request.

**JSP pages are more advantageous than Servlet:**

* They are easy to maintain.
* No recompilation or redeployment is required.
* JSP has access to entire API of JAVA .
* JSP are extended version of Servlet.

**Features of JSP**

* **Coding in JSP is easy** :- As it is just adding JAVA code to HTML/XML.
* **Reduction in the length of Code** :- In JSP we use action tags, custom tags etc.
* **Connection to Database is easier** :-It is easier to connect website to database and allows to read or write data easily to the database.
* **Make Interactive websites** :- In this we can create dynamic web pages which helps user to interact in real time environment.
* **Portable, Powerful, flexible and easy to maintain** :- as these are browser and server independent.
* **No Redeployment and No Re-Compilation** :- It is dynamic, secure and platform independent so no need to re-compilation.
* **Extension to Servlet** :- as it has all features of servlets, implicit objects and custom tags

**JSP syntax**

Syntax available in JSP are following

* 1. **Declaration Tag** :-It is used to declare variables.

**Syntax:-**

<%! Dec var %>

**Example:-**

<%! int var=10; %>

* 1. **Java Scriplets** :- It allows us to add any number of JAVA code, variables and expressions.

**Syntax:-**

<% java code %>

* 1. **JSP Expression** :- It evaluates and convert the expression to a string.

**Syntax:-**

<%= expression %>

**Example:-**

<% num1 = num1+num2 %>

* 1. **JAVA Comments** :- It contains the text that is added for information which has to be ignored.

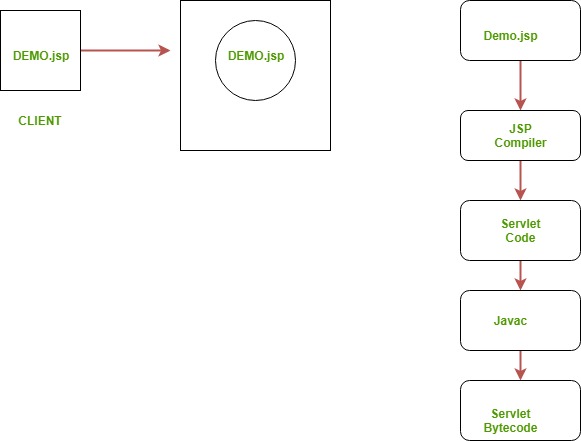
**Syntax:-**

<% -- JSP Comments %>

**Process of Execution**

Steps for Execution of JSP are following:-

* + - Create html page from where request will be sent to server eg try.html.
    - To handle to request of user next is to create .jsp file Eg. new.jsp
    - Create project folder structure.
    - Create XML file eg my.xml.
    - Create WAR file.
    - Start Tomcat
    - Run Application



**Example of Hello World**  
We will make one .html file and .jsp file

**demo.jsp**

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Hello World - JSP tutorial</title>

</head>

<body>

<%= "Hello World!" %>

</body>

</html>

**Advantages of using JSP**

* + - It does not require advanced knowledge of JAVA
    - It is capable of handling exceptions
    - Easy to use and learn
    - It can tags which are easy to use and understand
    - Implicit objects are there which reduces the length of code
    - It is suitable for both JAVA and non JAVA programmer

**Disadvantages of using JSP**

* + - Difficult to debug for errors.
    - First time access leads to wastage of time
    - It’s output is HTML which lacks features

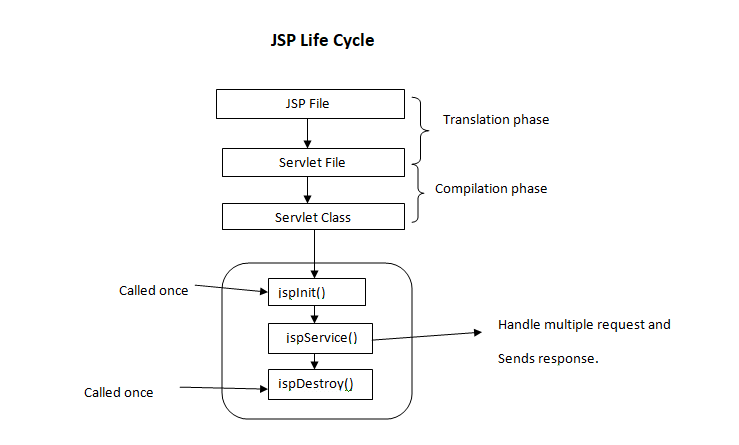
# Life cycle of JSP

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 17 Jul, 2021

 Read

 Discuss

A Java Server Page life cycle is defined as the process that started with its creation which later translated to a servlet and afterward servlet lifecycle comes into play. This is how the process goes on until its destruction.



Following steps are involved in the JSP life cycle: 

* Translation of JSP page to Servlet
* Compilation of JSP page(Compilation of JSP into test.java)
* Classloading (test.java to test.class)
* Instantiation(Object of the generated Servlet is created)
* Initialization(jspInit() method is invoked by the container)
* Request processing(\_jspService()is invoked by the container)
* JSP Cleanup (jspDestroy() method is invoked by the container)

*We can override jspInit(), jspDestroy() but we can’t override \_jspService() method.*

**Translation of JSP page to Servlet :**

This is the first step of the JSP life cycle. This translation phase deals with the Syntactic correctness of JSP. Here test.jsp file is translated to test.java.

**Compilation of JSP page :**

Here the generated java servlet file (test.java) is compiled to a class file (test.class).

**Classloading :**

Servlet class which has been loaded from the JSP source is now loaded into the container.

**Instantiation :**

Here an instance of the class is generated. The container manages one or more instances by providing responses to requests.

**Initialization :**

jspInit() method is called only once during the life cycle immediately after the generation of Servlet instance from JSP.

**Request processing :**

\_jspService() method is used to serve the raised requests by JSP. It takes request and response objects as parameters. This method cannot be overridden.

**JSP Cleanup :**

In order to remove the JSP from the use by the container or to destroy the method for servlets jspDestroy()method is used. This method is called once, if you need to perform any cleanup task like closing open files, releasing database connections jspDestroy() can be overridden.

# Difference between Servlet and JSP

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 29 Jun, 2022

 Read

 Discuss

**Brief Introduction:** **Servlet**technology is used to create a web application. A **servlet** is a Java class that is used to extend the capabilities of servers that host applications accessed by means of a request-response model. Servlets are mainly used to extend the applications hosted by web services.

**JSP** is used to create web applications just like **Servlet** technology. A **JSP** is a text document that contains two types of text: static data and dynamic data. The static data can be expressed in any text-based format (like HTML, XML, SVG, and WML), and the dynamic content can be expressed by JSP elements. Difference between Servlet and JSP

The difference between Servlet and JSP is as follows:

| Servlet | JSP |
| --- | --- |
| Servlet is a java code. | JSP is a HTML based code. |
| Writing code for servlet is harder than JSP as it is HTML in java. | JSP is easy to code as it is java in HTML. |
| Servlet plays a controller role in the hasMVC approach. | JSP is the view in the MVC approach for showing output. |
| Servlet is faster than JSP. | JSP is slower than Servlet because the first step in the hasJSP lifecycle is the translation of JSP to java code and then compile. |
| Servlet can accept all protocol requests. | JSP only accepts HTTP requests. |
| In Servlet, we can override the service() method. | In JSP, we cannot override its service() method. |
| In Servlet by default session management is not enabled, user have to enable it explicitly. | In JSP session management is automatically enabled. |
| In Servlet we have to implement everything like business logic and presentation logic in just one servlet file. | In JSP business logic is separated from presentation logic by using JavaBeansclient-side. |
| Modification in Servlet is a time-consumingcompiling task because it includes reloading, recompiling, JavaBeans and restarting the server. | JSP modification is fast, just need to click the refresh button. |
| It does not have inbuilt implicit objects. | In JSP there are inbuilt implicit objects. |
| There is no method for running JavaScript on the client side in Servlet. | While running the JavaScript at the client side in JSP, the client-side validation is used. |
| Packages are to be imported on the top of the program. | Packages can be imported into the JSP program(i.e bottom , middleclient-side, or top ) |

**What is Servlet Collaboration?**

The exchange of information among servlets of a particular Java web application is known as **Servlet Collaboration**. This enables passing/sharing information from one servlet to the other through method invocations.

**What are the principle ways provided by Java to achieve Servlet Collaboration?**  
The servlet api provides two interfaces namely:

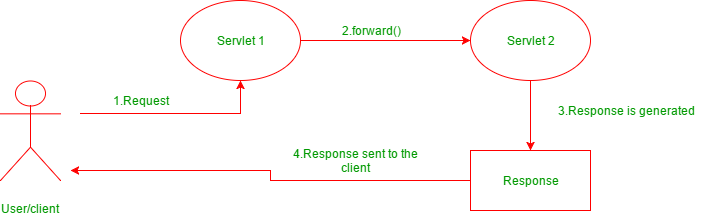
1. javax.servlet.RequestDispatcher
2. javax.servlet.http.HttpServletResponse

These two interfaces include the methods responsible for achieving the objective of sharing information between servlets.

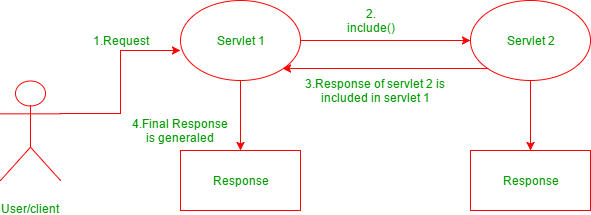
**Using RequestDispatcher Interface**

The RequestDispatcher interface provides the option of dispatching the client’s request to another web resource, which could be an HTML page, another servlet, JSP etc. It provides the following two methods:

* **public void forward(ServletRequest request, ServletResponse response)throws ServletException, java.io.IOException:**  
  The forward() method is used to transfer the client request to another resource (HTML file, servlet, jsp etc). When this method is called, the control is transferred to the next resource called. On the other hand, the include() method is used to include the content of the calling file into the called file. After calling this method, the control remains with the calling resource, but the processed output is included into the called resource.  
  The following diagram explains the way it works:



* **public void include(ServletRequest request, ServletResponse response)throws ServletException, java.io.IOException:**  
  The include() method is used to include the contents of the calling resource into the called one. When this method is called, the control still remains with the calling resource. It simply includes the processed output of the calling resource into the called one.  
  The following diagram explains how it works:



* **Example of using RequestDispatcher for Servlet Collaboration**  
  The following example explains how to use RequestDispatcher interface to achieve Servlet Collaboration:  
  **index.html**

. There are four different types of elements you can use in JSP:

Scripting elements—This is where your Java goes, as in the scriptlets we've already seen.

Comments—Comments are text you add to annotate a JSP document—they're like notes to yourself or other programmers. They're ignored by the Web server, and are only for programmers to read.

Directives—Directives are instructions specific to a particular page that let you set how the page is to be processed. For example, you can use a directive to indicate that you want the output of a page to be XML, not HTML.

Actions—Actions let you perform some operation, such as including other pages in the current page, or including Java applets, or working with JavaBean components (which you'll see in Day 6). Unlike directives, actions are re-evaluated each time the page is accessed.

Let's take a look at these various types of JSP elements, starting with scripting elements.

Scripting Elements

There are three types of scripting elements in JSP:

Scriptlets—Scriptlets can contain Java code. This is the most general of all the scripting elements.

Declarations—Declares a variable or a method for use in your code.

Expressions—Contains a Java expression that the server evaluates. The result of the expression is inserted into the Web page.

Scriptlets

The most general place for your Java code is in a scriptlet. Scriplets hold Java code fragments, and are enclosed with the tags <% and %>. We've already seen an example of a scriptlet in our first JSP, where we used the Java statement out.println("Hello there!"); to display text in a Web page:

<HTML>

<HEAD>

<TITLE>A Web Page</TITLE>

</HEAD>

<BODY>

<% out.println("Hello there!"); %>

</BODY>

</HTML>

A scriptlet can hold any number of Java statements, declarations, or expressions, making scriptlets the most general of all the JSP scripting elements. Note that they must hold valid Java statements, so you can't include direct HTML in a scriptlet.

Scriptlets are what many JSP programmers think of when they think of JSP. Scriptlets are used to embed Java code in HTML documents, turning them into JSP documents. But besides scriptlets, you can also use declarations and expressions.

Declarations

As you'll see tomorrow, you can store data using variables in Java. For example, here's how you can store our text string Hello there! in a variable of type String:

String msg = "Hello there!";

This declares a new variable named msg that holds the text Hello there!. In Java, you must declare variables before you use them, giving them a name. After you've named a variable, you can refer to it in your code using that name.

There's a special set of tags, <%! and %>, that you can use to declare variables in JSP (you can also declare variables in scriptlets, which is a more common thing to do). Here's an example: In this case, the code declares the variable msg:

<HTML>

<HEAD>

<TITLE>A Web Page</TITLE>

</HEAD>

<BODY>

<%! String msg = "Hello there!"; %>

</BODY>

</HTML>

Now you can use this new variable in a scriptlet; for example, here's how you can display the text in msg in the Web page using the out.println method:

<HTML>

<HEAD>

<TITLE>A Web Page</TITLE>

</HEAD>

<BODY>

<%! String msg = "Hello there!"; %>

<% out.println(msg); %>

</BODY>

</HTML>

That's what the process looks like in overview; you'll get all the details on declaring and using variables in Day 2.

Expressions

Expressions are any Java code fragment that can be evaluated to yield a value. For example, the expression 2 + 2 yields a value of 4, the expression 44 - 10 yields 34, and so on. In JSP, you can surround a Java expression in the tags <%= and %>. The expression's value is inserted into the Web page as text by the server.

You can even use a variable as an expression—when such an expression is evaluated, it yields the value of the variable. For example, if we had stored our string Hello there! in a variable named msg, we can insert that string into the Web page simply by using the variable as an expression this way:

<HTML>

<HEAD>

<TITLE>A Web Page</TITLE>

</HEAD>

<BODY>

<%! String msg = "Hello there!"; %>

<%= msg %>

</BODY>

</HTML>

Expressions like this are similar to scriptlets, because you can place Java code in them, but they have to be able to result in a single value when they're evaluated.

Expressions like this are useful for shorter JSP pages, but note that you don't have to use them to insert text into a Web page—you can use methods like out.println in scriptlets instead.

Comments

You can use JSP comments to document what's going on in a JSP page; they act like notes to you or other programmers. Comments are purely for the benefit of the programmers that work on the page, because the server will strip them out before sending the page back to the browser. You enclose the text in a comment between the tags <%-- and --%>.

Here's an example; in this case, the code includes the comment Display the message now. to this JSP example:

<HTML>

<HEAD>

<TITLE>A Web Page</TITLE>

</HEAD>

<BODY>

<%-- Display the message now. --%>

<% out.println("Hello there!"); %>

</BODY>

</HTML>

This comment lets you and other programmers know what's going on in your JSP. You can add as many comments to a JSP as you like—they're entirely for the benefit of programmers, and will all be stripped out before the page is sent back to the browser.

Directives

JSP directives let you give directions to the server on how a page should be processed. There are three directives in JSP; we'll see them throughout the book, but here's an overview:

page—This directive lets you configure an entire JSP page, such as whether its output is HTML or XML. You'll see more on this directive in Day 6.

include—This directive lets you include another page or resource in a JSP page.

taglib—This directive lets you use a set of custom JSP tags as defined in a tag library. More on taglibs is coming up in Day 9, "Using Custom JSP Tags," and Day 10, Creating Custom Tags."

The page directive is going to be a useful one for us as we work with Java. You can use this directive to configure the page you're working on. Like other directives, you use the special tags <%@ and %> with this directive; to create a page directive, you also include the keyword page like this: <%@ page ... %>.

Here's an example to show what the page directive can do for us. In this case, the code wants to indicate that if there's an error in the current page, the server should send an error page named error.jsp back to the browser; that we're programming in Java; that the output of the current page should be XML (not HTML, which is the default—more on XML in Day 18, "Using XML and XSLT in JSP"); and that we want to import the java.sql package so that our code can work with SQL databases. You'll see more on importing Java packages when we start working with Java in depth—Java is divided into many sections called packages, and to use a particular section's functionality, you must import the corresponding package, which makes it accessible to your code.

You can do all that with attributes of a single page directive. In JSP, attributes work just as they do in HTML—they're part of an element's opening tag, and have the form attribute=value. Here's how you can use the errorPage, language, contentType, and import attributes of the page directive to configure the page as you want (note that the page directive comes at the very beginning of the page):

<%@ page errorPage="error.jsp" language="java"

contentType="application/xml" import="java.sql.\*" %>

<HTML>

<HEAD>

.

.

.

As you can already see, the page directive is going to give us a lot of programming power.

The include directive, which looks like <%@ page ... %>, lets you include another page in the current page. When you include a page at a particular point, that page's entire contents are simply inserted at that point.

The taglib directive (which looks like this: <%@ taglib ... %>) specifies a library of custom JSP tags (see Days 9 and 10) that you want to use in the current page. Tag libraries let you define your own JSP tags. We're going to see how to use this directive later, so we won't go into the details now.

NOTE

There are rumors that scriptlets are going to be phased out in JSP 2.0 and tag libraries will be used instead, so it's a good idea to make sure you know what's going on with tag libraries in Days 9 and 10.

Actions

JSP also includes actions. As their name implies, actions let you perform some action. Unlike directives, actions are re-evaluated each time the page is accessed.

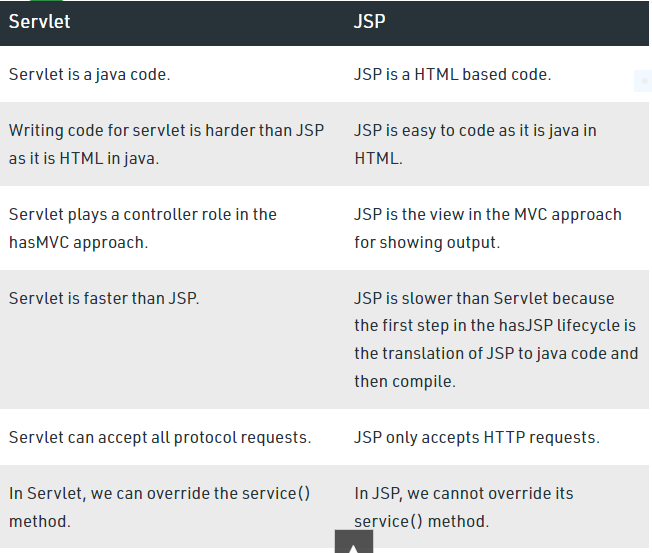
There are two types of actions: custom and standard. Custom actions are actions you create yourself, and standard actions come built into JSP. Here are the standard actions in overview:

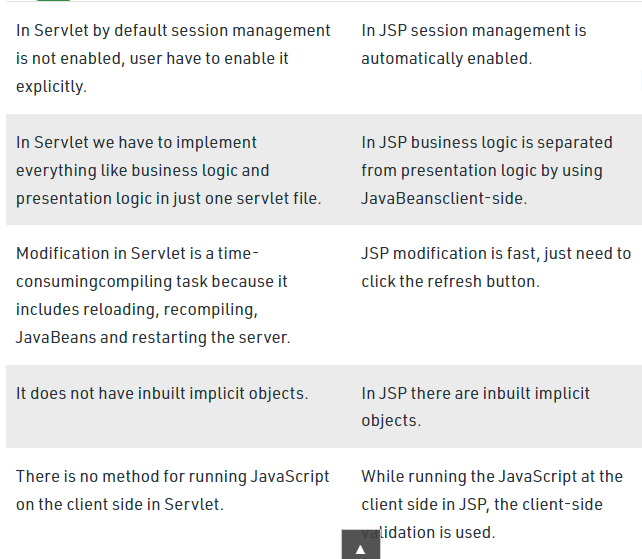
<jsp:forward>—Forwards the browser request for a new Web page to an HTML file, JSP page, or servlet. In this way, you can delegate how your Web applications respond to the browser.

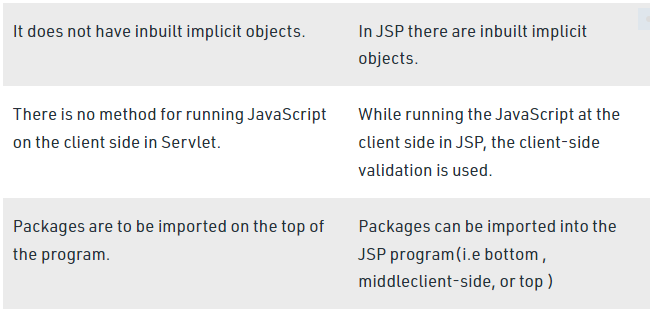
<jsp:include>— Includes a file or Web component. Note that <jsp:include> is different than the include directive because it re-evaluates the included file every time the page is accessed, whereas the include directive does not.

<jsp:plugin>— Lets you execute applets or JavaBeans with a plug-in. If the browser doesn't have the required plug-in module, it will display a dialog box asking you to download it. We'll see this one in Day 13.

<jsp:getProperty>, <jsp:setProperty>, and <jsp:useBean>—You use these actions with JavaBean components, as you'll see in Day 6.







# Establishing JDBC Connection in Java

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 20 Jun, 2022

 Read

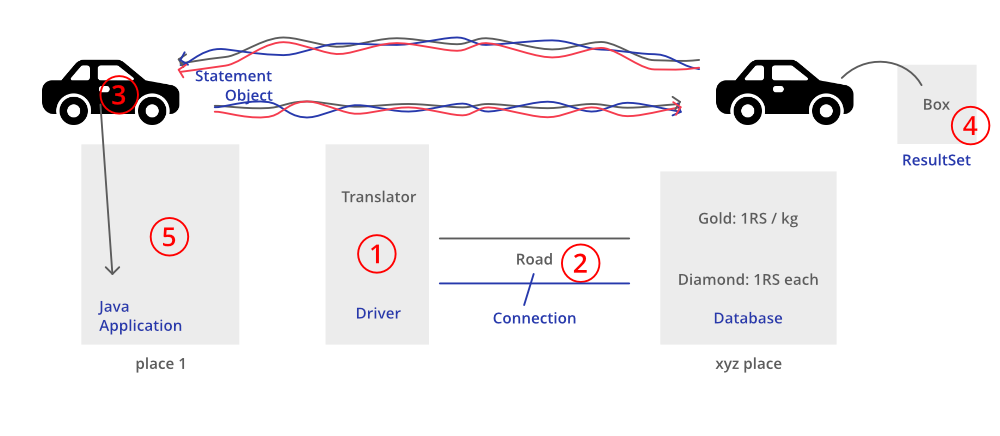
 Discuss

Before establishing a connection between the front end i.e your Java Program and the back end i.e the database we should learn what precisely a JDBC is and why it came into existence. Now let us discuss what exactly JDBC stands for and will ease out with the help of real-life illustration to get it working.

**What is JDBC?**

[JDBC](https://www.geeksforgeeks.org/introduction-to-jdbc/) is an acronym for Java Database Connectivity. It’s an advancement for ODBC ( Open Database Connectivity ). JDBC is a standard API specification developed in order to move data from frontend to the backend. This API consists of classes and interfaces written in Java. It basically acts as an interface (not the one we use in Java) or channel between your Java program and databases i.e it establishes a link between the two so that a programmer could send data from Java code and store it in the database for future use.

**Illustration:** Working of JDBC co-relating with real-time



**Why JDBC Come into Existence?**

As previously told JDBC is an advancement for ODBC, ODBC being platform-dependent had a lot of drawbacks. ODBC API was written in C, C++, Python, and Core Java and as we know above languages (except Java and some part of Python )are platform-dependent. Therefore to remove dependence, JDBC was developed by a database vendor which consisted of classes and interfaces written in Java.

**Steps For Connectivity Between Java Program and Database**

1. Import the Packages
2. Load the drivers using the *forName() method*
3. Register the drivers *using DriverManager*
4. Establish a connection*using the Connection class object*
5. Create a statement
6. Execute the query
7. Close the connections

Let us discuss these steps in brief before implementing by writing suitable code to illustrate connectivity steps for JDBC/

**Step 1:** Import the Packages

**Step 2:**Loading the drivers

In order to begin with, you first need to load the driver or register it before using it in the program. Registration is to be done once in your program. You can register a driver in one of two ways mentioned below as follows:

**2-A**Class.forName()

Here we load the driver’s class file into memory at the runtime. No need of using new or create objects. The following example uses Class.forName() to load the Oracle driver as shown below as follows:

Class.forName(“oracle.jdbc.driver.OracleDriver”);

**2-B**DriverManager.registerDriver()

DriverManager is a Java inbuilt class with a static member register. Here we call the constructor of the driver class at compile time. The following example uses DriverManager.registerDriver()to register the Oracle driver as shown below:

DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver())

**Step 3:**Establish a connection*using*the *Connection class object*

After loading the driver, establish connections as shown below as follows:

Connection con = DriverManager.getConnection(url,user,password)

* **user: U**sername from which your SQL command prompt can be accessed.
* **password:**password from which the SQL command prompt can be accessed.
* **con:** It is a reference to the Connection interface.
* **Url**: Uniform Resource Locator which is created as shown below:

String url = “ jdbc:oracle:thin:@localhost:1521:xe”

Where oracle is the database used, thin is the driver used, @localhost is the IP Address where a database is stored, 1521 is the port number and xe is the service provider. All 3 parameters above are of String type and are to be declared by the programmer before calling the function. Use of this can be referred to from the final code.

**Step 4:**Create a statement

Once a connection is established you can interact with the database. The JDBCStatement, CallableStatement, and PreparedStatement interfaces define the methods that enable you to send SQL commands and receive data from your database.   
Use of JDBC Statement is as follows:

Statement st = con.createStatement();

***Note:****Here, con is a reference to Connection interface used in previous step .*

**Step 5:**Execute the query

Now comes the most important part i.e executing the query. The query here is an SQL Query. Now we know we can have multiple types of queries. Some of them are as follows:

* The query for updating/inserting a table in a database.
* The query for retrieving data.

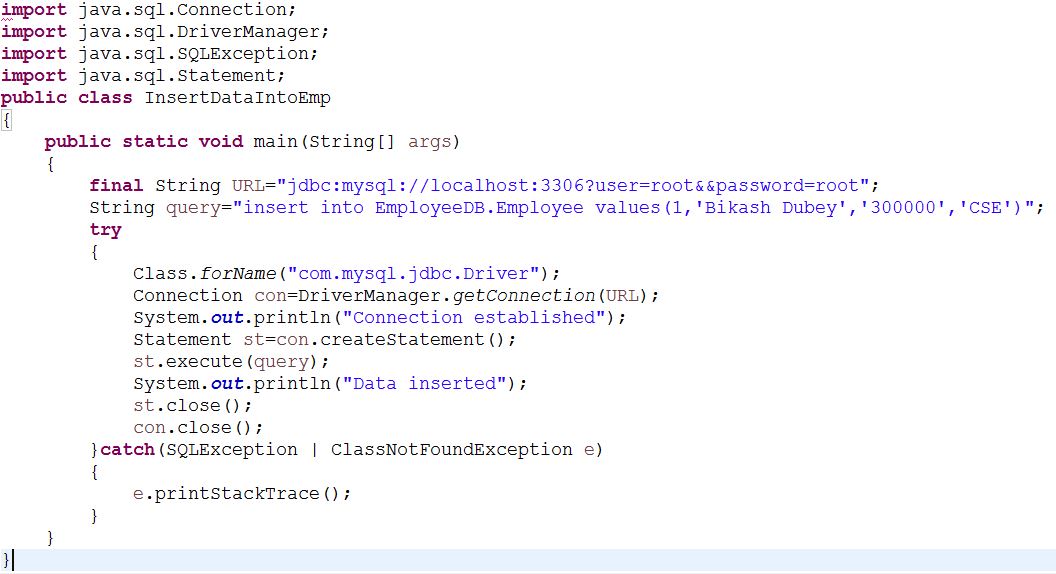
The executeQuery() method of the **Statement interface** is used to execute queries of retrieving values from the database. This method returns the object of ResultSet that can be used to get all the records of a table.   
The executeUpdate(sql query) method of the Statement interface is used to execute queries of updating/inserting.

**Pseudo Code:**

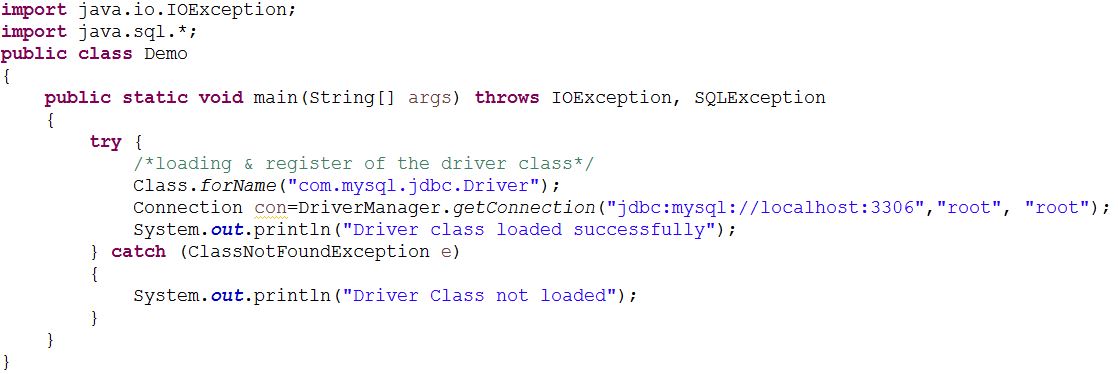
**Need of Hibernate Framework**

Hibernate is used to overcome the limitations of JDBC like:

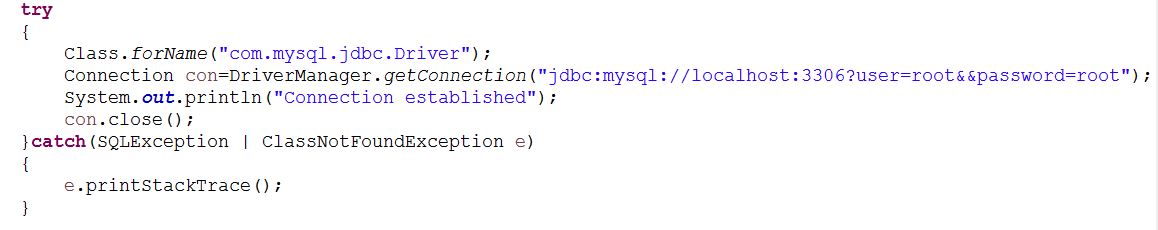
1. JDBC code is dependent upon the Database software being used i.e. our persistence logic is dependent, because of using JDBC. Here we are inserting a record into Employee table but our query is Database software-dependent i.e. Here we are using MySQL. But if we change our Database then this query won’t work.



1. If working with JDBC, changing of Database in middle of the project is very costly.
2. JDBC code is not portable code across the multiple database software.
3. In JDBC, Exception handling is mandatory. Here We can see that we are handling lots of Exception for connection.



1. While working with JDBC, There is no support Object-level relationship.
2. In JDBC, there occurs a Boilerplate problem i.e. For each and every project we have to write the below code. That increases the code length and reduce the readability.



To overcome the above problems we use ORM tool i.e. nothing but Hibernate framework. By using Hibernate we can avoid all the above problems and we can enjoy some additional set of functionalities.

**About Hibernate Framework**

Hibernate is a framework which provides some **abstraction layer**, meaning that the programmer does not have to worry about the implementations, Hibernate does the implementations for you internally like **Establishing a connection with the database, writing query to perform CRUD operations etc**.  
It is a **java framework** which is used to develop persistence logic. Persistence logic means to store and process the data for long use. More precisely Hibernate is an open-source, non-invasive, light-weight java ORM(Object-relational mapping) framework to develop objects which are independent of the database software and make independent persistence logic in all JAVA, JEE.

**Framework** means it is special install-able software that provides an abstraction layer on one or more technologies like JDBC, Servlet, etc to simplify or reduce the complexity for the development process.

**Open Source means:**

* Hibernate framework is available for everyone without any cost.
* The source code of Hibernate is also available on the Internet and we can also modify the code.

**Light-weight means:**

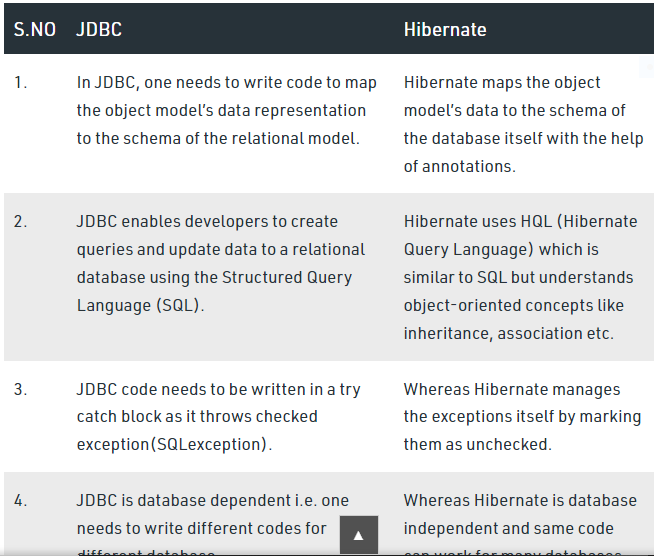
* Hibernate is less in size means the installation package is not big is size.
* Hibernate does not require any heavy container for execution.
* It does not require POJO and POJI model programming.
* Hibernate can be used alone or we can use Hibernate with other java technology and framework.

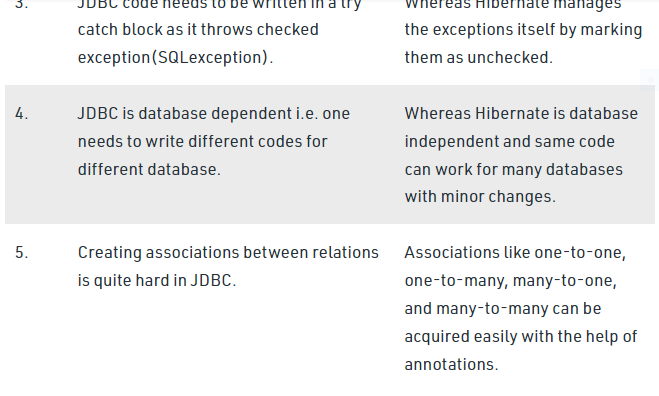
**Non-invasive means:**

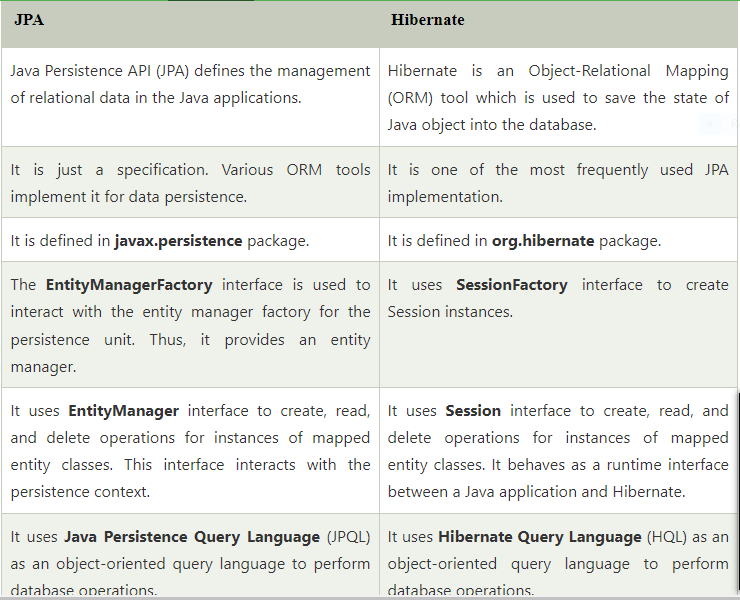
* The classes of Hibernate application development are loosely coupled classes with respect to Hibernate API i.e. Hibernate class need not implement hibernate API interfaces and need not extend from Hibernate API classes.

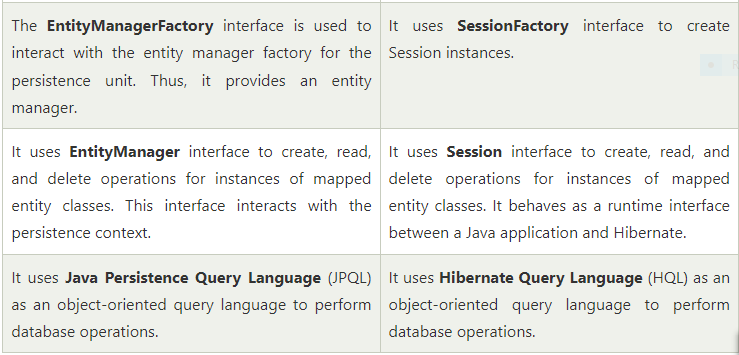
**Functionalities supported by Hibernate framework**

* Hibernate framework support **Auto DDL** operations. In JDBC manually we have to create table and declare the data-type for each and every column. But Hibernate can do **DDL operations** for you internally like creation of table,drop a table,alter a table etc.
* Hibernate supports **Auto Primary key generation**. It means in JDBC we have to manually set a primary key for a table. But Hibernate can this task for you.
* Hibernate framework is independent of Database because it supports **HQL (Hibernate Query Language)** which is not specific to any database, whereas JDBC is database dependent.
* In Hibernate, **Exception Handling is not mandatory**, whereas In JDBC exception handling is mandatory.
* Hibernate supports **Cache Memory** whereas JDBC does not support cache memory.
* Hibernate is a **ORM tool** means it support Object relational mapping. Whereas JDBC is not object oriented moreover we are dealing with values means primitive data. In hibernate each record is represented as a Object but in JDBC each record is nothing but a data which is nothing but primitive values.









## Hibernate Framework

Hibernate is a Java framework that simplifies the development of Java application to interact with the database. It is an open source, lightweight, ORM (Object Relational Mapping) tool. Hibernate implements the specifications of JPA (Java Persistence API) for data persistence.

## ORM Tool

An ORM tool simplifies the data creation, data manipulation and data access. It is a programming technique that maps the object to the data stored in the database.



The ORM tool internally uses the JDBC API to interact with the database.

## What is JPA?

Java Persistence API (JPA) is a Java specification that provides certain functionality and standard to ORM tools. The **javax.persistence** package contains the JPA classes and interfaces.

## Advantages of Hibernate Framework

Following are the advantages of hibernate framework:

### 1) Open Source and Lightweight

Hibernate framework is open source under the LGPL license and lightweight.

### 2) Fast Performance

The performance of hibernate framework is fast because cache is internally used in hibernate framework. There are two types of cache in hibernate framework first level cache and second level cache. First level cache is enabled by default.

### 3) Database Independent Query

HQL (Hibernate Query Language) is the object-oriented version of SQL. It generates the database independent queries. So you don't need to write database specific queries. Before Hibernate, if database is changed for the project, we need to change the SQL query as well that leads to the maintenance problem.

### 4) Automatic Table Creation

Hibernate framework provides the facility to create the tables of the database automatically. So there is no need to create tables in the database manually.

### 5) Simplifies Complex Join

Fetching data from multiple tables is easy in hibernate framework.

### 6) Provides Query Statistics and Database Status

Hibernate supports Query cache and provide statistics about query and database status.

# Hibernate Lifecycle

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 07 Oct, 2021

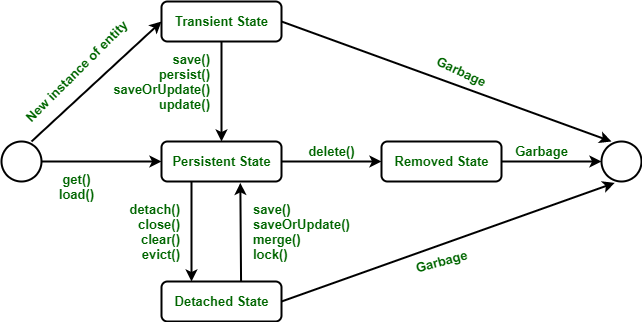
 Read

 Discuss

Here we will learn about Hibernate Lifecycle or in other words, we can say that we will learn about the lifecycle of the mapped instances of the entity/object classes in hibernate. In Hibernate, we can either create a new object of an entity and store it into the database, or we can fetch the existing data of an entity from the database. These entity is connected with the lifecycle and each object of entity passes through the various stages of the lifecycle.

There are mainly four states of the Hibernate Lifecycle :

1. Transient State
2. Persistent State
3. Detached State
4. Removed State



***Hibernate Lifecycle***

As depicted from the above media one can co-relate how they are plotted in order to plot better in our mind. Now we will be discussing the states to better interpret hibernate lifecycle. It is as follows:

**State 1:** Transient State

The transient state is the first state of an entity object. When we instantiate an object of a [POJO class](https://www.geeksforgeeks.org/pojo-vs-java-beans/) using the new operator then the object is in the transient state. This object is not connected with any hibernate session. As it is not connected to any Hibernate Session, So this state is not connected to any database table. So, if we make any changes in the data of the POJO Class then the database table is not altered. Transient objects are independent of Hibernate, and they exist in the **heap memory**.

https://media.geeksforgeeks.org/wp-content/uploads/20210626215802/GFGTransientState.png

***Changing new object to Transient State***

There are two layouts in which transient state will occur as follows:

1. When objects are generated by an application but are not connected to any session.
2. The objects are generated by a closed session.

Here, we are creating a new object for the Employee class. Below is the code which shows the initialization of the Employee object :

//Here, The object arrives in the transient state.

Employee e = new Employee();

e.setId(21);

e.setFirstName("Neha");

e.setMiddleName("Shri");

e.setLastName("Rudra");

**State 2:** Persistent State

Once the object is connected with the Hibernate Session then the object moves into the Persistent State. So, there are two ways to convert the Transient State to the Persistent State :

1. Using the hibernated session, save the entity object into the database table.
2. Using the hibernated session, load the entity object into the database table.

In this state. each object represents one row in the database table. Therefore, if we make any changes in the data then hibernate will detect these changes and make changes in the database table.



***Converting Transient State to Persistent State***

Following are the methods given for the persistent state:

* session.persist(e);
* session.save(e);
* session.saveOrUpdate(e);
* session.update(e);
* session.merge(e);
* session.lock(e);

**Example:**

// Transient State

Employee e = new Employee("Neha Shri Rudra", 21, 180103);

//Persistent State

session.save(e);

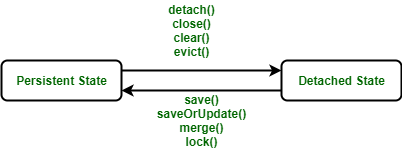
**State 3:** Detached State

For converting an object from Persistent State to Detached State, we either have to close the session or we have to clear its cache. As the session is closed here or the cache is cleared, then any changes made to the data will not affect the database table. Whenever needed, the detached object can be reconnected to a new hibernate session. To reconnect the detached object to a new hibernate session, we will use the following methods as follows:

* merge()
* update()
* load()
* refresh()
* save()
* update()

Following are the methods used for the detached state :

* session.detach(e);
* session.evict(e);
* session.clear();
* session.close();



***Converting Persistent State to Detached State***

**Example**

// Transient State

Employee e = new Employee("Neha Shri Rudra", 21, 180103);

// Persistent State

session.save(e);

// Detached State

session.close();

**State 4:** Removed State

In the hibernate lifecycle it is the last state. In the removed state, when the entity object is deleted from the database then the entity object is known to be in the removed state. It is done by calling the ***delete() operation***. As the entity object is in the removed state, if any change will be done in the data will not affect the database table.

***Note:****To make a removed entity object we will call****session.delete()****.*

https://media.geeksforgeeks.org/wp-content/uploads/20210626233515/GFGRemovedState.png

***Converting Persistent State to Removed State***

**Example**

// Java Pseudo code to Illustrate Remove State

// Transient State

Employee e = new Employee();

Session s = sessionfactory.openSession();

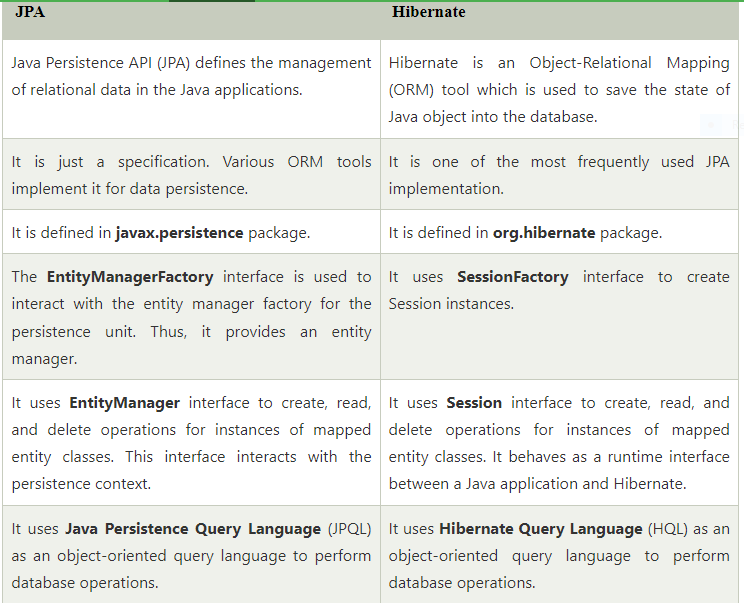
e.setId(01);

// Persistent State

session.save(e)

// Removed State

session.delete(e);



**Spring**

[Spring](https://www.geeksforgeeks.org/introduction-to-spring-framework/) is an open-source lightweight framework that allows Java EE 7 developers to build simple, reliable, and scalable enterprise applications. This framework mainly focuses on providing various ways to help you manage your business objects. It made the development of Web applications much easier than compared to classic Java frameworks and Application Programming Interfaces (APIs), such as Java database connectivity (JDBC), JavaServer Pages(JSP), and Java Servlet. This framework uses various new techniques such as Aspect-Oriented Programming (AOP), [Plain Old Java Object (POJO)](https://www.geeksforgeeks.org/pojo-vs-java-beans/), and dependency injection (DI), to develop enterprise applications.

**Spring IoC (Inversion of Control)**

Spring IoC (Inversion of Control) Container is the core of Spring Framework. It creates the objects, configures and assembles their dependencies, manages their entire life cycle. The Container uses Dependency Injection(DI) to manage the components that make up the application. It gets the information about the objects from a configuration file(XML) or Java Code or Java Annotations and Java POJO class. These objects are called Beans. Since the Controlling of Java objects and their lifecycle is not done by the developers, hence the name Inversion Of Control. The followings are some of the main features of Spring IoC,

* Creating Object for us,
* Managing our objects,
* Helping our application to be configurable,
* Managing dependencies

*To read more on Spring IoC (Inversion of Control) please refer to this article:*[*Spring IoC (Inversion of Control)*](https://www.geeksforgeeks.org/spring-ioc-container/)

**Spring Dependency Injection**

Dependency Injection is the main functionality provided by Spring IOC(Inversion of Control). The Spring-Core module is responsible for injecting dependencies through either Constructor or Setter methods. The design principle of Inversion of Control emphasizes keeping the Java classes independent of each other and the container frees them from object creation and maintenance. These classes, managed by Spring, must adhere to the standard definition of Java-Bean. Dependency Injection in Spring also ensures loose coupling between the classes. There are two types of Spring Dependency Injection.

1. Setter Dependency Injection (SDI)
2. Constructor Dependency Injection (CDI)

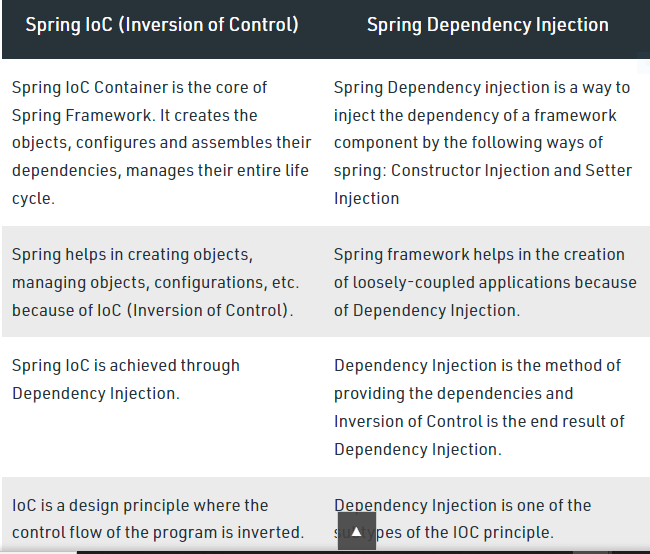
**A. Setter Dependency Injection (SDI)**

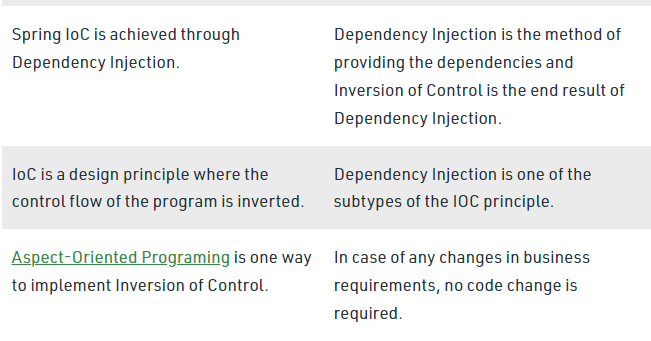
Setter Injection is the simpler of the two  Dependency Injection methods. In this, the Dependency Injection will be injected with the help of setter and/or getter methods. Now to set the Dependency Injection as Setter Injection in the bean, it is done through the bean-configuration file For this, the property to be set with the Setter Injection is declared under the **<property>** tag in the bean-config file.

**B. Constructor Dependency Injection (CDI)**

In Constructor Injection, the Dependency Injection will be injected with the help of constructors. Now to set the Dependency Injection as Constructor Dependency Injection in bean, it is done through the bean-configuration file. For this, the property to be set with the CDI is declared under the **<constructor-arg>** tag in the bean-config file.

Let us finally come up with cut-throat differences between them depicted via the table given below to get a better understanding as there persists always a dilemma if not understood to great depth.





# Jar files in Java

A [JAR (Java Archive)](https://www.geeksforgeeks.org/jar-files-java/) is a package file format typically used to aggregate many Java class files and associated metadata and resources (text, images, etc.) into one file to distribute application software or libraries on the Java platform.   
In simple words, a JAR file is a file that contains a compressed version of .class files, audio files, image files, or directories. We can imagine a .jar file as a zipped file(.zip) that is created by using WinZip software. Even, WinZip software can be used to extract the contents of a .jar . So you can use them for tasks such as lossless data compression, archiving, decompression, and archive unpacking.

# 2) Java Class File

A **Java class file** is a file containing Java bytecode and having **.class extension** that can be executed by [JVM](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/). A Java class file is created by a Java compiler from .java files as a result of successful compilation. As we know that a single Java programming language source file (or we can say .java file) may contain one class or more than one class. So if a .java file has more than one class then each class will compile into a separate class files

# How to get selected value in dropdown list using JavaScript ?

<!DOCTYPE html>

<head>

    <title>

        How to get selected value in

        dropdown list using JavaScript?

    </title>

</head>

<body>

    <h1 style="color: green">

        GeeksforGeeks

    </h1>

    <b>

        How to get selected value in dropdown

        list using JavaScript?

    </b>

<p> Select one from the given options:

        <select id="select1">

            <option value="free">Free</option>

            <option value="basic">Basic</option>

            <option value="premium">Premium</option>

        </select>

    </p>

<p> The value of the option selected is:

        <span class="output"></span>

    </p>

    <button onclick="getOption()"> Check option </button>

    <script type="text/javascript">

    function getOption() {

        selectElement = document.querySelector('#select1');

        output = selectElement.value;

        document.querySelector('.output').textContent = output;

    }

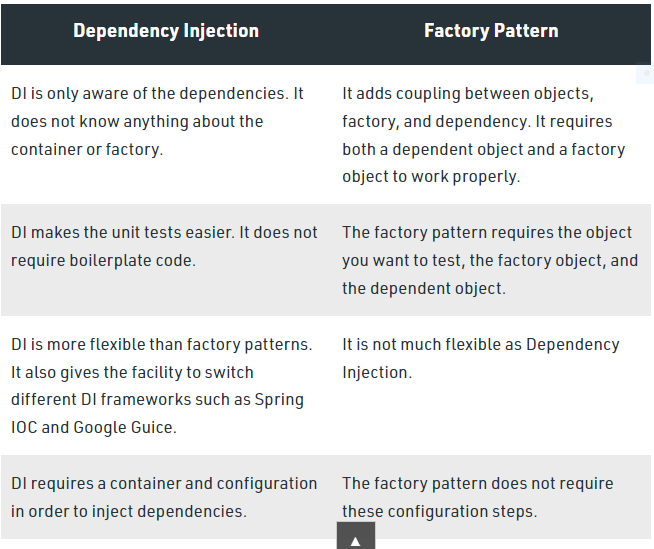
    </script>

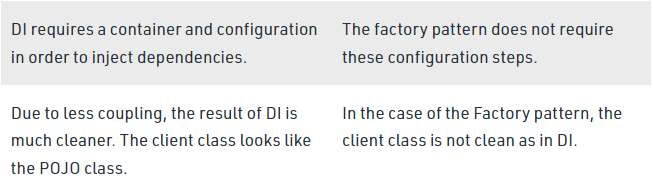
</body>

</html>

**ResultSet :**

1. **import** java.sql.\*;
2. **class** FetchRecord{
3. **public** **static** **void** main(String args[])**throws** Exception{
5. Class.forName("oracle.jdbc.driver.OracleDriver");
6. Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
7. Statement stmt=con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE,ResultSet.CONCUR\_UPDATABLE);
8. ResultSet rs=stmt.executeQuery("select \* from emp765");
10. //getting the record of 3rd row
11. rs.absolute(3);
12. System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));
14. con.close();
15. }}
16. -------------------------------------------------------------------------------------------------
17. **Prepared Statement**
18. **import** java.sql.\*;
19. **class** InsertPrepared{
20. **public** **static** **void** main(String args[]){
21. **try**{
22. Class.forName("oracle.jdbc.driver.OracleDriver");
24. Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
26. PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");
27. stmt.setInt(1,101);//1 specifies the first parameter in the query
28. stmt.setString(2,"Ratan");
30. **int** i=stmt.executeUpdate();
31. System.out.println(i+" records inserted");
33. con.close();
35. }**catch**(Exception e){ System.out.println(e);}
37. }
38. }

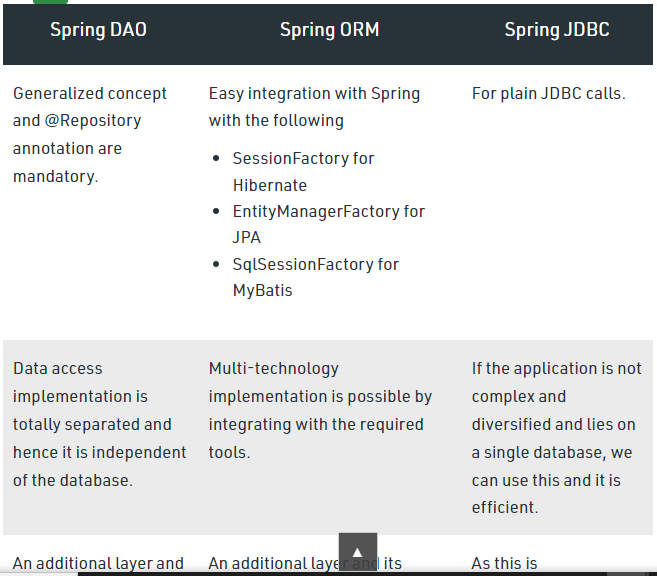


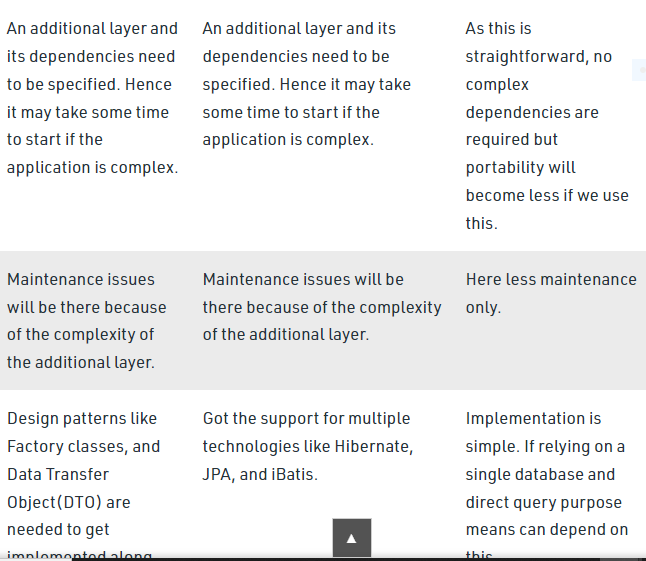


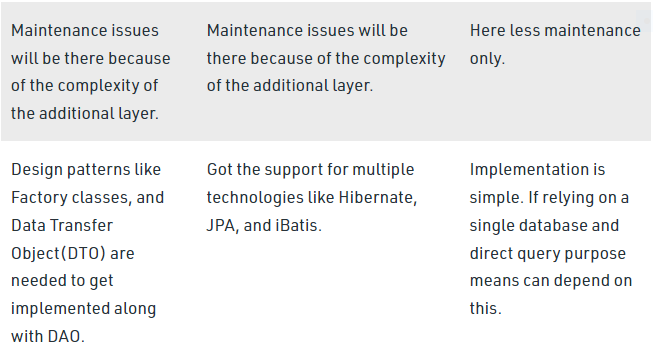
### A. Factory Pattern

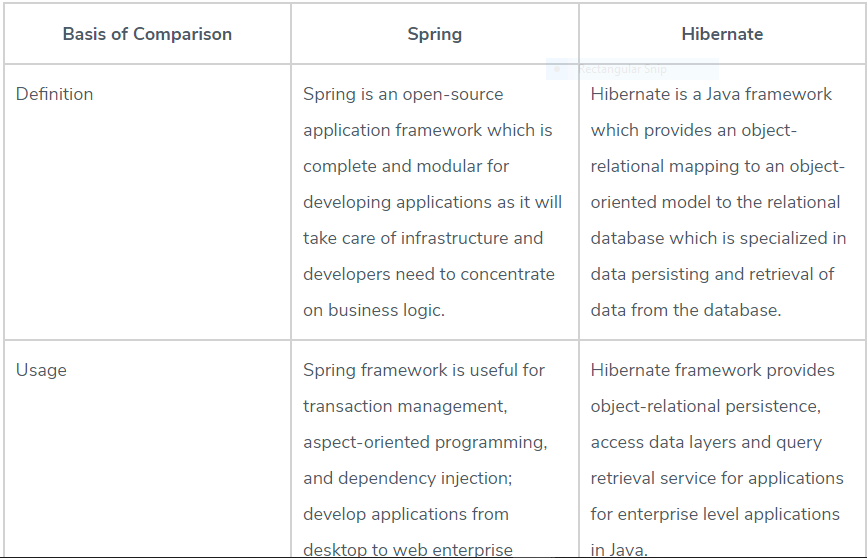
In **Factory Pattern**, the client class is still responsible for getting the instance of products by class **getInstance()**method of factory class, which means the client class is directly coupled with factory class and it can not be unit tested without the factory class.

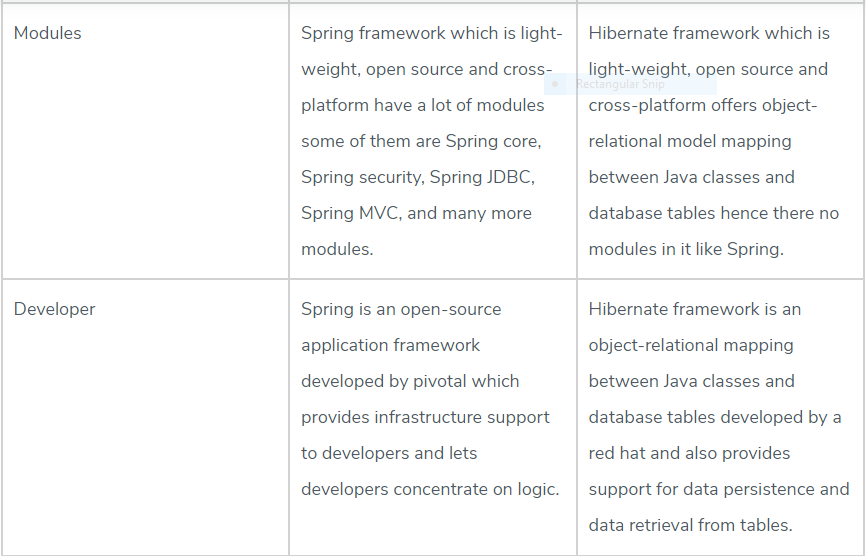
**Implementation:**In this example, we will the difference between Factory Pattern and DI with the help of the ExpenseTracker application. In this application, we have a dependent class ExpenseTracker which depends on ExpenseCalculator class. Here we will be using both Factory Pattern and Dependency Injection to understand the difference between them.

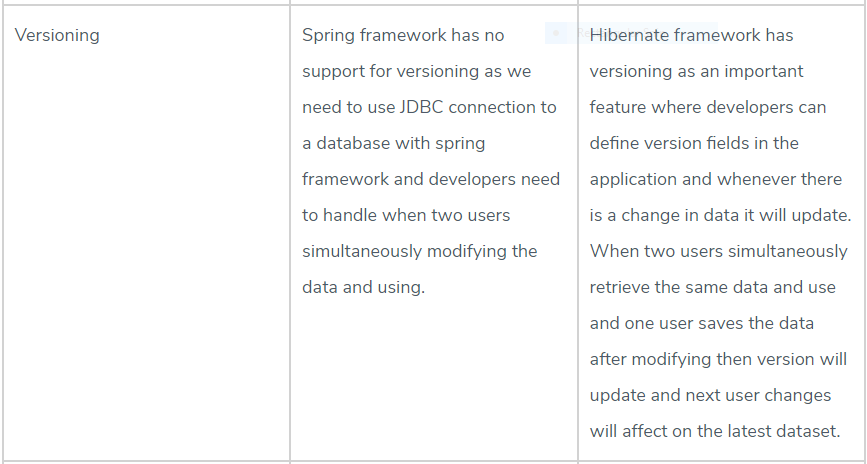


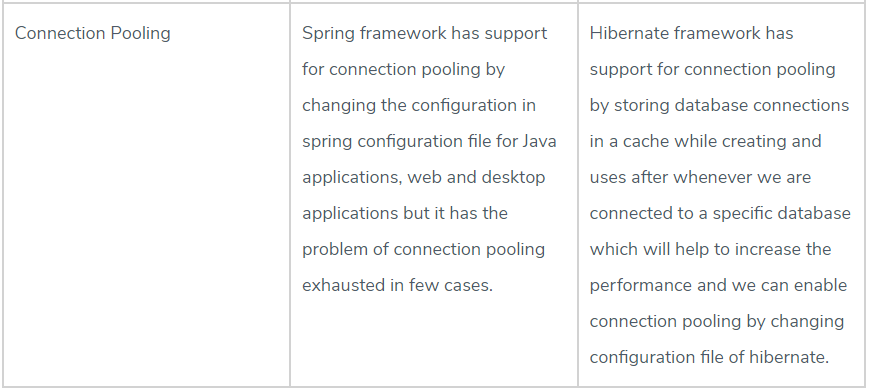


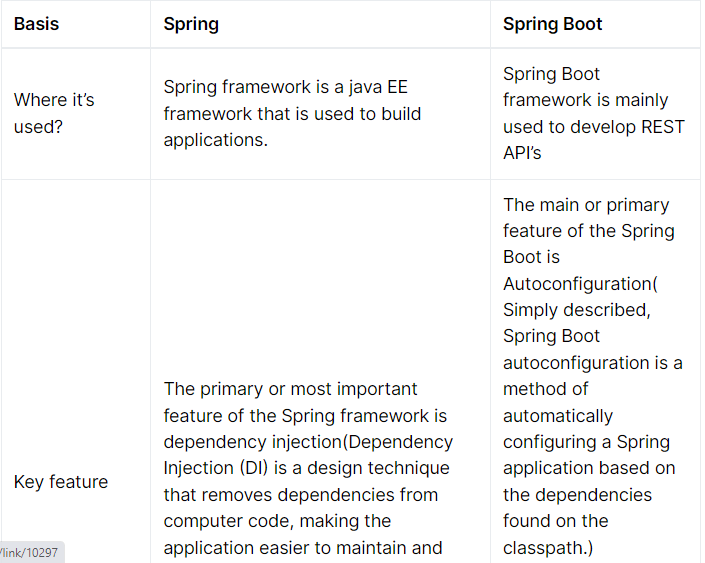


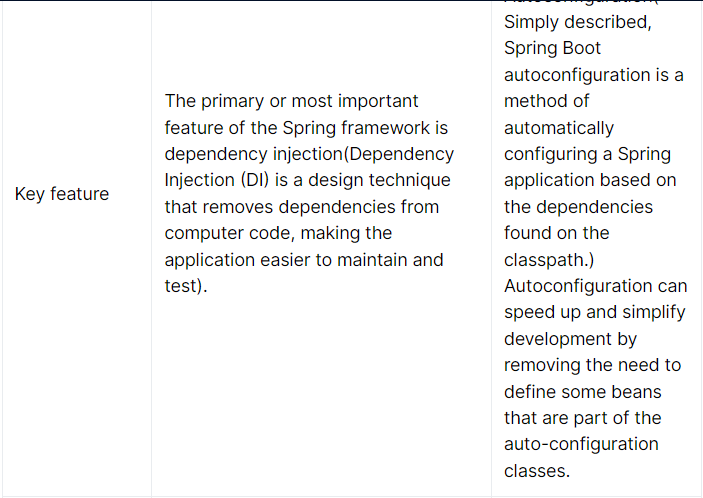


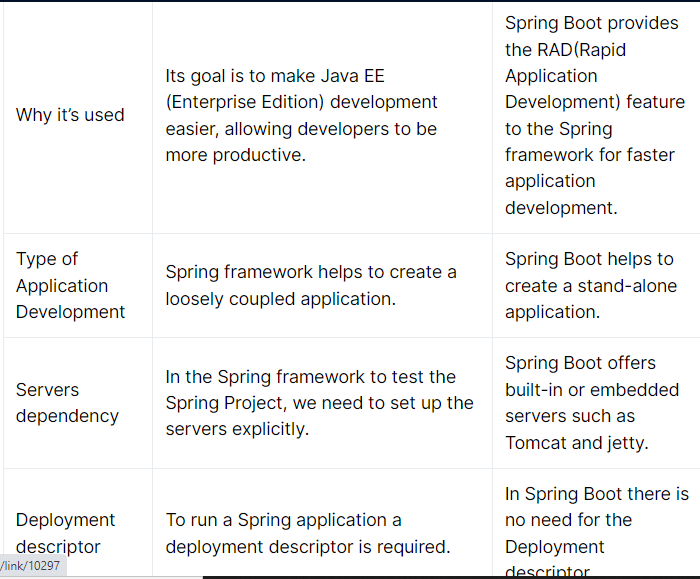


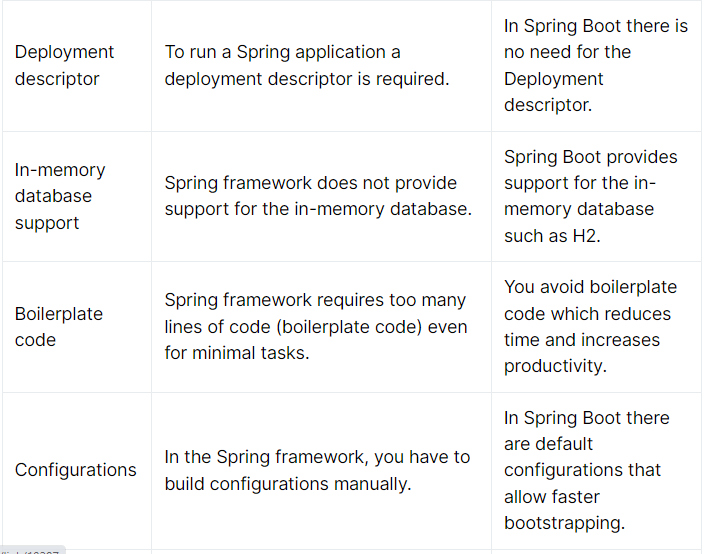


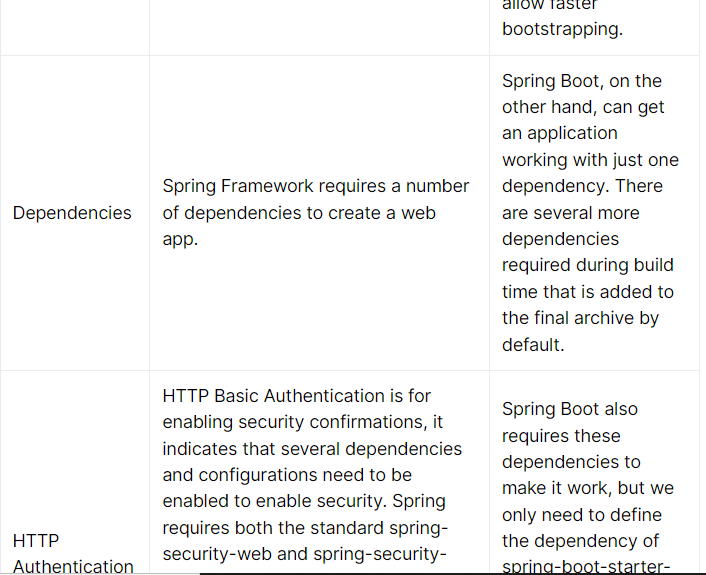


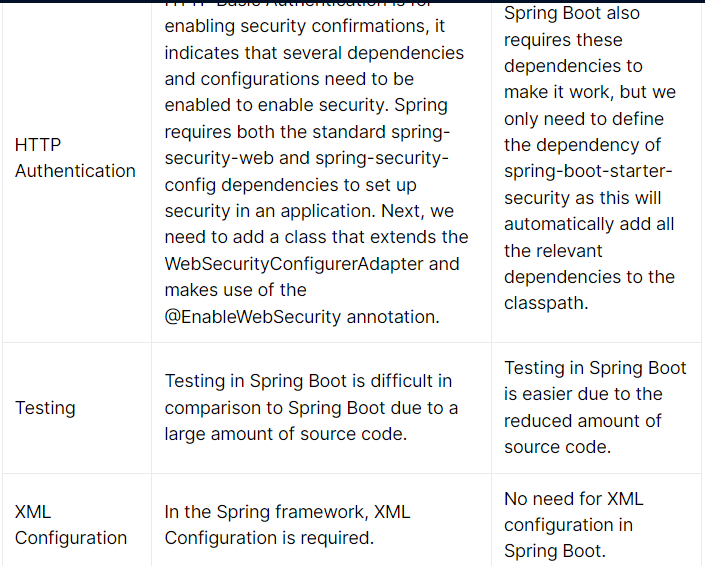


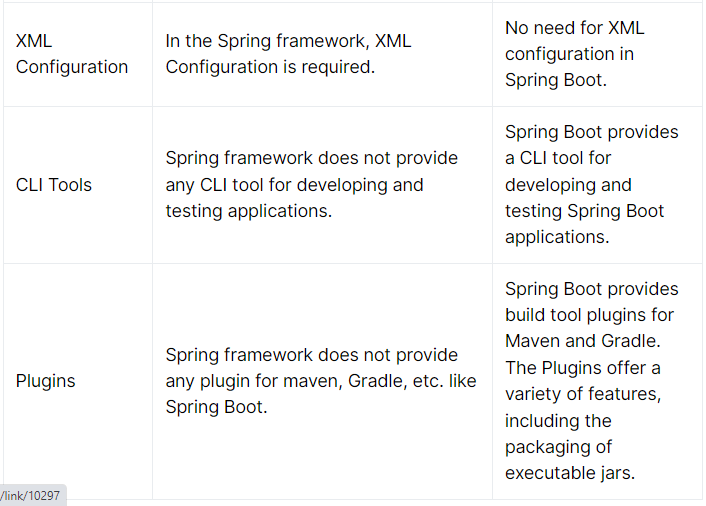


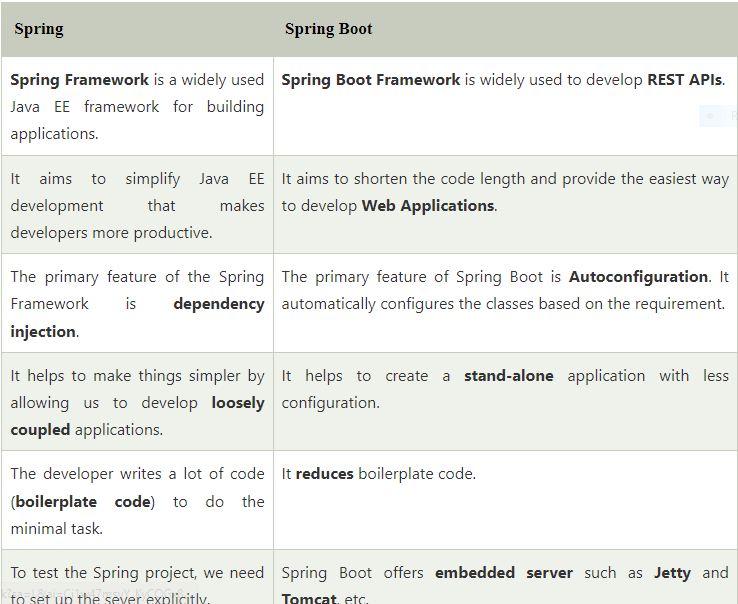


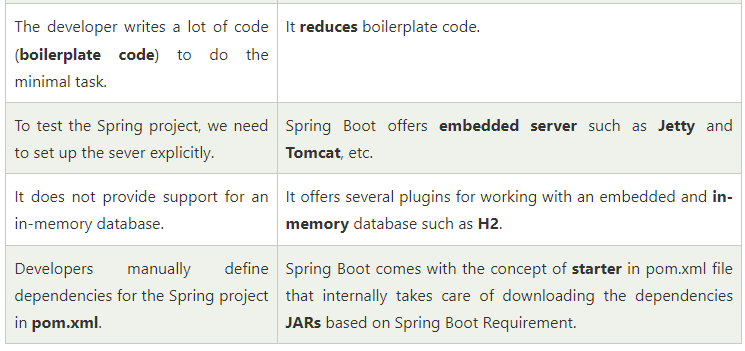












# Introduction to Spring Boot

Spring is widely used for creating scalable applications. For web applications Spring provides  
Spring MVC which is a widely used module of spring which is used to create scalable web applications.  
But main **disadvantage of spring projects is that configuration is really time-consuming and can be a bit overwhelming for the new developers.** Making the application production-ready takes some time if you are new to the spring.

Solution to this is Spring Boot. Spring Boot is built on the top of the spring and contains all the features of spring. And is becoming favourite of developer’s these days because of it’s a rapid production-ready environment which enables the developers to directly focus on the logic instead of struggling with the configuration and set up.

Spring Boot is a microservice-based framework and making a production-ready application in it takes very less time.  
Prerequisite for Spring Boot is the basic knowledge Spring framework.  
For revising the concepts of spring framework [read this article.](https://www.geeksforgeeks.org/introduction-to-spring-framework/)

### Features of Spring Boot

Spring Boot is built on the top of the conventional spring framework. So, it provides all the features of spring and is yet easier to use than spring.

* **It allows to avoid heavy configuration of XML which is present in spring:**  
  Unlike the [Spring MVC Project](https://www.geeksforgeeks.org/spring-mvc-with-jsp-view/), in spring boot everything is auto-configured. We just need to use proper configuration for utilizing a particular functionality.

**For example:** If we want to use [hibernate(ORM)](https://www.geeksforgeeks.org/introduction-to-hibernate-framework/) then we can just add **@Table annotation** above model/entity class(discussed later) and add **@Column annotation** to map it to table and columns in the database

* **It provides easy maintenance and creation of REST end points:**  
  Creating a [REST API](https://www.geeksforgeeks.org/rest-api-introduction/) is very easy in Spring Boot. Just the annotation **@RestController** and **@RequestMapping(/endPoint)** over the controller class does the work.
* **It includes embedded Tomcat-server:**

Unlike [Spring MVC project](https://www.geeksforgeeks.org/spring-mvc-with-jsp-view/) where we have to manually add and install the tomcat server, Spring Boot comes with an [embedded Tomcat server](https://www.geeksforgeeks.org/embedding-tomcat-server-in-maven-project/), so that the applications can be hosted on it.

* **Deployment is very easy, war and jar file can be easily deployed in the tomcat server:**  
  **war** or **jar** files can be directly deployed on the Tomcat Server and Spring Boot provides the facility to convert our project into war or jar files. Also, the instance of Tomcat can be run on the cloud as well.
* **Microservice Based Architecture:**  
  Microservice, as the name suggests is the name given to a module/service which focuses on a single type of feature, exposing an API(application peripheral interface).

Let us consider an example of a hospital management system.

* + In case of monolithic systems, there will be a single code containing all the features which are very tough to maintain on a huge scale.
  + But in the microservice-based system, each feature can be divided into smaller subsystems like service to handle patient registration, service to handle database management, service to handle billing etc.

Microservice based system can be easily migrated as only some services need to be altered which also makes debugging and deployment easy. Also, each service can be integrated and can be made in different technologies suited to them.

### Evolution of Spring Boot

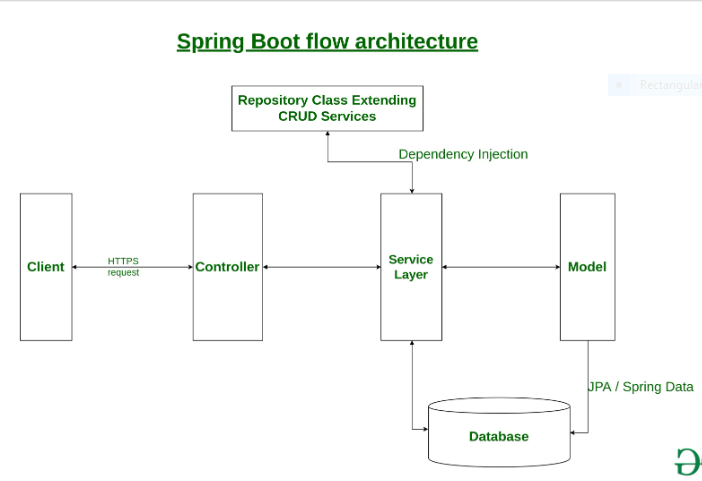
1. Spring Boot came into existence when in October 2012, a client, **Mike Youngstrom** made a Jira request asking for bootstrapping the spring framework so that it can be quickly started. And hence in early 2013, Spring Boot was made.
2. In April 2014, Spring Boot **1.0** was created followed by various versions.
3. Spring Boot **1.1** on June 2014,
4. **1.2** in March 2015,
5. **1.3** in December 2016,
6. **1.4** in January 2017 and
7. Spring Boot **1.5** on February 2017.

### Spring Boot Architecture

To understand the architecture of Spring Boot, let us first see different layers and classes present in it.

* **Layers in Spring Boot:** There are four main layers in Spring Boot:
  + **Presentation Layer:** As the name suggests, it consists of views(i.e. frontend part)
  + **Data Access Layer:** CRUD (create, retrieve, update, delete) operations on the database comes under this category.
  + **Service Layer:** This consist of service classes and uses services provided by data access layers.
  + **Integration Layer:** It consists of web different web services(any service available over the internet and uses [XML](https://www.geeksforgeeks.org/html-and-xml-gq/) messaging system).
* Then we have utility classes, validator classes and view classes.
* All the services provided by the classes are implemented in their corresponding classes and are retrieved by implementing the dependency on those interfaces.

**Spring Boot flow architecture**:



# REST API using Java Spring Boot

[Representational state transfer (REST)](https://www.geeksforgeeks.org/rest-api-introduction/) is a software architectural style that defines a set of constraints to be used for creating Web services. Web services that conform to the REST architectural style, called [RESTful Web services](https://www.geeksforgeeks.org/rest-api-architectural-constraints/), provide interoperability between computer systems on the Internet. RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of stateless operations. Other kinds of Web services, such as [SOAP Web services](https://www.geeksforgeeks.org/difference-between-rest-api-and-soap-api/), expose their own arbitrary sets of operations.

In this article, we will understand how to create a rest API using [spring boot](https://www.geeksforgeeks.org/introduction-to-spring-boot/).

[Spring](https://www.geeksforgeeks.org/introduction-to-spring-framework/) is widely used for creating scalable applications. For web applications, Spring provides Spring MVC which is a widely used module of spring which is used to create scalable web applications. But the main disadvantage of spring projects is that configuration is really time-consuming and can be a bit overwhelming for the new developers. The solution to this is [Spring Boot](https://www.geeksforgeeks.org/introduction-to-spring-boot/). Spring Boot is built on the top of the spring and contains all the features of spring. In this article, we will create a REST API to add employees to the employee list and get the list of employees

## What is Spring Boot

Spring Boot is a project that is built on the top of the Spring Framework. It provides an easier and faster way to set up, configure, and run both simple and web-based applications.

Pause

Unmute

Current TimeÂ 2:51

/

DurationÂ 4:57

Loaded: 100.00%

Â

Fullscreen

It is a Spring module that provides the **RAD (Rapid Application Development)** feature to the Spring Framework. It is used to create a stand-alone Spring-based application that you can just run because it needs minimal Spring configuration.



In short, Spring Boot is the combination of **Spring Framework** and **Embedded Servers**.

In Spring Boot, there is no requirement for XML configuration (deployment descriptor). It uses convention over configuration software design paradigm that means it decreases the effort of the developer.

We can use Spring **STS IDE** or **Spring Initializr** to develop Spring Boot Java applications.

**Why should we use Spring Boot Framework?**

We should use Spring Boot Framework because:

* The dependency injection approach is used in Spring Boot.
* It contains powerful database transaction management capabilities.
* It simplifies integration with other Java frameworks like JPA/Hibernate ORM, Struts, etc.
* It reduces the cost and development time of the application.

Along with the Spring Boot Framework, many other Spring sister projects help to build applications addressing modern business needs. There are the following Spring sister projects are as follows:

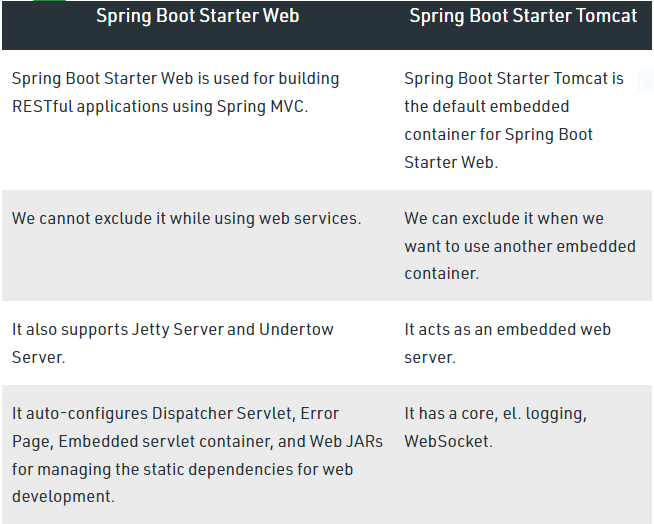
* **Spring Data:** It simplifies data access from the relational and **NoSQL** databases.
* **Spring Batch:** It provides powerful **batch** processing.
* **Spring Security:** It is a security framework that provides robust **security** to applications.
* **Spring Social:** It supports integration with **social networking** like LinkedIn.
* **Spring Integration:** It is an implementation of Enterprise Integration Patterns. It facilitates integration with other **enterprise applications** using lightweight messaging and declarative adapters.

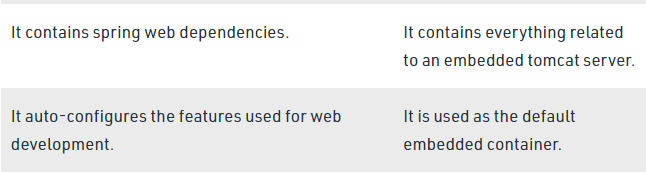
## Advantages of Spring Boot

* It creates **stand-alone** Spring applications that can be started using Java **-jar**.
* It tests web applications easily with the help of different **Embedded** HTTP servers such as **Tomcat, Jetty,** etc. We don't need to deploy WAR files.
* It provides opinionated '**starter**' POMs to simplify our Maven configuration.
* It provides **production-ready** features such as **metrics, health checks,** and **externalized configuration**.
* There is no requirement for **XML** configuration.
* It offers a **CLI** tool for developing and testing the Spring Boot application.
* It offers the number of **plug-ins**.
* It also minimizes writing multiple **boilerplate codes** (the code that has to be included in many places with little or no alteration), XML configuration, and annotations.
* It **increases productivity** and reduces development time.

## Limitations of Spring Boot

Spring Boot can use dependencies that are not going to be used in the application. These dependencies increase the size of the application.



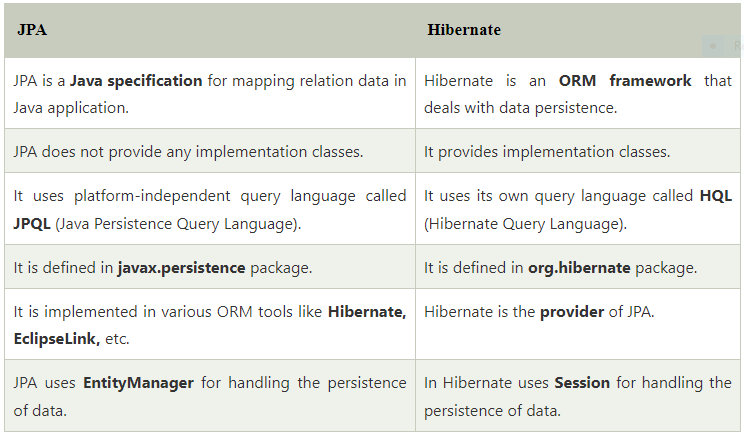


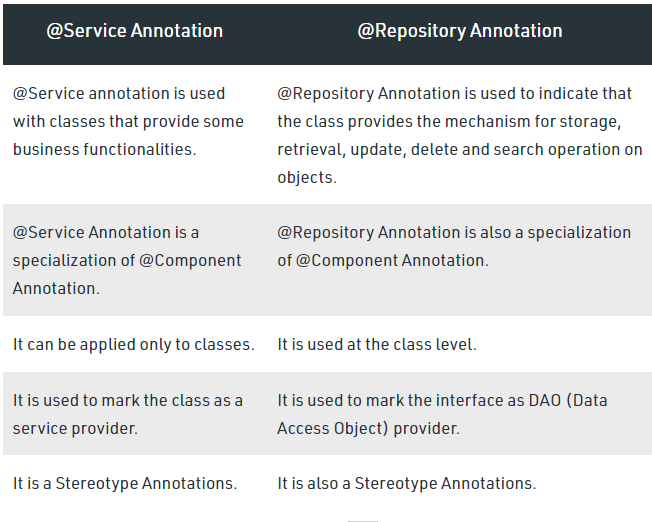
## Goals of Spring Boot

The main goal of Spring Boot is to reduce **development, unit test,** and **integration test** time.

* Provides Opinionated Development approach
* Avoids defining more Annotation Configuration
* Avoids writing lots of import statements
* Avoids XML Configuration.

By providing or avoiding the above points, Spring Boot Framework reduces **Development time, Developer Effort,** and **increases productivity**.





# Spring Boot JPA

## What is JPA?

**Spring Boot JPA**is a Java specification for managing **relational** data in Java applications. It allows us to access and persist data between Java object/ class and relational database. JPA follows **Object-Relation Mapping**(ORM). It is a set of interfaces. It also provides a runtime **EntityManager** API for processing queries and transactions on the objects against the database. It uses a platform-independent object-oriented query language JPQL (Java Persistent Query Language).

In the context of persistence, it covers three areas:

* The Java Persistence API
* **Object-Relational** metadata
* The API itself, defined in the **persistence** package

JPA is not a framework. It defines a concept that can be implemented by any framework.

# Spring Boot – Spring Data JPA

Spring Data JPA or JPA stands for**Java Persistence API,** so before looking into that, we must know about **ORM (Object Relation Mapping)**. So Object relation mapping is simply the process of persisting any java object directly into a database table. Usually, the name of the object being persisted becomes the name of the table, and each field within that object becomes a column. With the table setup set up, each row corresponds to a record in the application. Hibernate is one example of ORM. In short, JPA is the interface while hibernate is the implementation.

The java persistence API provides a specification for persisting, reading, and managing data from your java object to your relational tables in the database. JPA specifies the set of rules and guidelines for developing interfaces that follow standards. Straight to the point: JPA is just guidelines to implement ORM and there is no underlying code for the implementation. Spring Data JPA is part of the spring framework. The goal of spring data repository abstraction is to significantly reduce the amount of boilerplate code required to implement a data access layer for various persistence stores. Spring Data JPA is not a JPA provider, it is a library/framework that adds an extra layer of abstraction on the top of our JPA provider line Hibernate.

## Why should we use JPA?

JPA is simpler, cleaner, and less labor-intensive than JDBC, SQL, and hand-written mapping. JPA is suitable for non-performance oriented complex applications. The main advantage of JPA over JDBC is that, in JPA, data is represented by objects and classes while in JDBC data is represented by tables and records. It uses POJO to represent persistent data that simplifies database programming. There are some other advantages of JPA:

* JPA avoids writing DDL in a database-specific dialect of SQL. Instead of this, it allows mapping in XML or using Java annotations.
* JPA allows us to avoid writing DML in the database-specific dialect of SQL.
* JPA allows us to save and load Java objects and graphs without any DML language at all.
* When we need to perform queries JPQL, it allows us to express the queries in terms of Java entities rather than the (native) SQL table and columns.

## JPA Features

There are following features of JPA:

* It is a powerful repository and custom **object-mapping abstraction.**
* It supports for **cross-store persistence**. It means an entity can be partially stored in MySQL and Neo4j (Graph Database Management System).
* It dynamically generates queries from queries methods name.
* The domain base classes provide basic properties.
* It supports transparent auditing.
* Possibility to integrate custom repository code.
* It is easy to integrate with Spring Framework with the custom namespace.

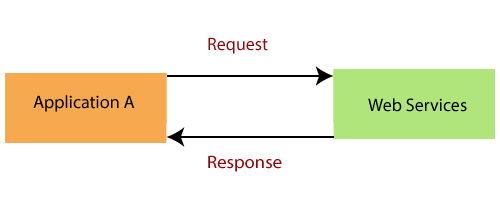
## JPA Architecture

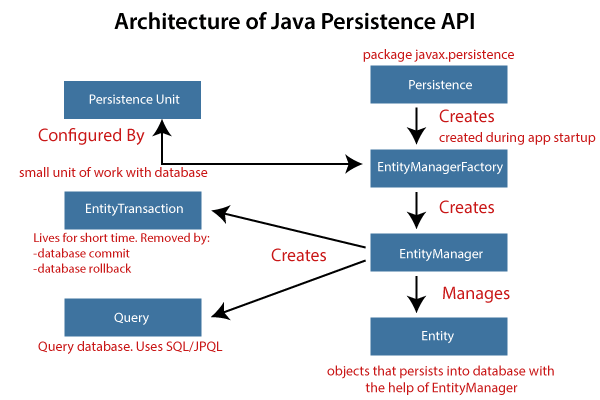
JPA is a source to store business entities as relational entities. It shows how to define a POJO as an entity and how to manage entities with relation.

The following figure describes the class-level architecture of JPA that describes the core classes and interfaces of JPA that is defined in the **javax persistence** package. The JPA architecture contains the following units:

* **Persistence:** It is a class that contains static methods to obtain an EntityManagerFactory instance.
* **EntityManagerFactory:** It is a factory class of EntityManager. It creates and manages multiple instances of EntityManager.
* **EntityManager:** It is an interface. It controls the persistence operations on objects. It works for the Query instance.
* **Entity:** The entities are the persistence objects stores as a record in the database.
* **Persistence Unit:** It defines a set of all entity classes. In an application, EntityManager instances manage it. The set of entity classes represents the data contained within a single data store.
* **EntityTransaction:** It has a **one-to-one** relationship with the EntityManager class. For each EntityManager, operations are maintained by EntityTransaction class.
* **Query:** It is an interface that is implemented by each JPA vendor to obtain relation objects that meet the criteria.

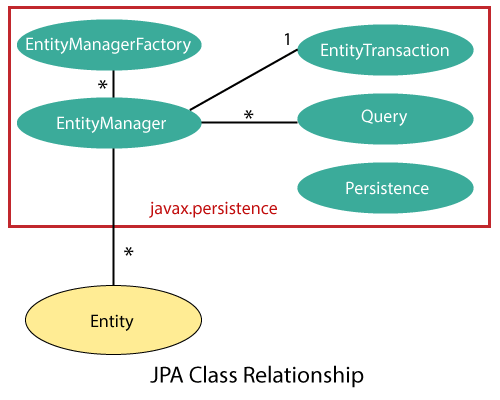
|  |  |
| --- | --- |
| **SOAP Protocol** | **RESTful Web Services** |
| SOAP is a protocol. | REST is an architectural approach. |
| SOAP acronym for Simple Object Access Protocol. | REST acronym for REpresentational State Transfer. |
| In SOAP, the data exchange format is always XML. | There is no strict data exchange format. We can use JSON, XML, etc. |
| XML is the most popular data exchange format in SOAP web services. | JSON is the most popular data exchange format in RESTful web services. |
| SOAP uses Web Service Definition Language (WSDL). | REST does not have any standard definition language. |
| SOAP does not pose any restrictions on transport. We can use either HTTP or MQ. | RESTful services use the most popular HTTP protocol. |
| SOAP web services are typical to implement. | RESTful services are easier to implement than SOAP. |
| SOAP web services use the JAX-WS API. | RESTful web services use the JAX-RS API. |
| SOAP protocol defines too many standards. | RESTful services do not emphasis on too many standards. |
| SOAP cannot use RESTful services because it is a protocol. | RESTful service can use SOAP web services because it is an architectural approach that can use any protocol like HTTP and SOAP. |
| SOAP reads cannot be cached. | REST reads can be cached. |

* 



## JPA Class Relationships

The classes and interfaces that we have discussed above maintain a relationship. The following figure shows the relationship between classes and interfaces.



* The relationship between EntityManager and EntiyTransaction is **one-to-one**. There is an EntityTransaction instance for each EntityManager operation.
* The relationship between EntityManageFactory and EntiyManager is **one-to-many**. It is a factory class to EntityManager instance.
* The relationship between EntityManager and Query is **one-to-many**. We can execute any number of queries by using an instance of EntityManager class.
* The relationship between EntityManager and Entity is **one-to-many**. An EntityManager instance can manage multiple Entities.

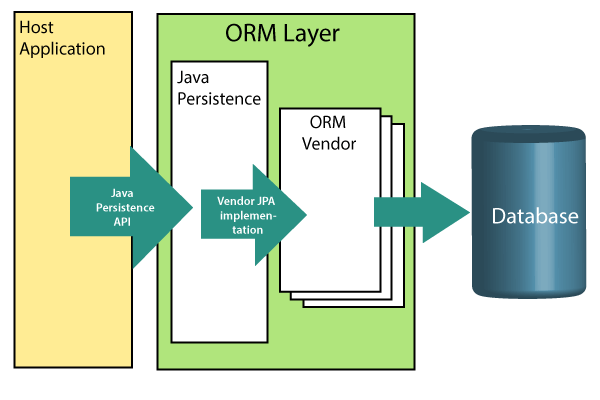
## JPA Implementations

JPA is an open-source API. There is various enterprises vendor such as Eclipse, RedHat, Oracle, etc. that provides new products by adding the JPA in them. There are some popular JPA implementations frameworks such as **Hibernate, EclipseLink, DataNucleus,** etc. It is also known as **Object-Relation Mapping** (ORM) tool.

## Object-Relation Mapping (ORM)

In ORM, the mapping of Java objects to database tables, and vice-versa is called **Object-Relational Mapping.** The ORM mapping works as a bridge between a **relational database** (tables and records) and **Java application** (classes and objects).

In the following figure, the ORM layer is an adapter layer. It adapts the language of object graphs to the language of SQL and relation tables.



The ORM layer exists between the application and the database. It converts the Java classes and objects so that they can be stored and managed in a relational database. By default, the name that persists become the name of the table, and fields become columns. Once an application sets-up, each table row corresponds to an object

| JDBC | Hibernate |
| --- | --- |
| 1. | In JDBC, one needs to write code to map the object model’s data representation to the schema of the relational model. | Hibernate maps the object model’s data to the schema of the database itself with the help of annotations. |
| 2. | JDBC enables developers to create queries and update data to a relational database using the Structured Query Language (SQL). | Hibernate uses HQL (Hibernate Query Language) which is similar to SQL but understands object-oriented concepts like inheritance, association etc. |
| 3. | JDBC code needs to be written in a try catch block as it throws checked exception(SQLexception). | Whereas Hibernate manages the exceptions itself by marking them as unchecked. |
| 4. | JDBC is database dependent i.e. one needs to write different codes for different database. | Whereas Hibernate is database independent and same code can work for many databases with minor changes. |
| 5. | Creating associations between relations is quite hard in JDBC. | Associations like one-to-one, one-to-many, many-to-one, and many-to-many can be acquired easily with the help of annotations. |

# Hibernate – Difference Between ORM and JDBC

Hibernate is a framework that is used to develop persistence logic that is independent of Database software. In JDBC to develop persistence logic, we deal with primitive types. Whereas Hibernate framework we use Objects to develop persistence logic that is independent of database software.

### ORM (Object-Relational Mapping)

ORM, an abbreviation for Object-relational mapping, is a programming approach that connects object code to a relational database via a metadata descriptor. This object code is developed in object-oriented programming (OOP) languages like Java, Python, C++, C#, etc. ORM transforms data between type systems that don’t get along in relational databases or OOP languages. In other words, it is a technique for storing, recovering, updating, as well as deleting from an object-oriented program in a relational (table) database. Now let’s first understand the term object code.

**What is Object Code?**

Object code is defined as low-level code that is comprehensible by computers. It is generated by the compiler using the source code. In other words, it is a file generated by the compiler containing the instructions for the machine in the form of binary digits.

**Advantages of ORM:**

* Resolves object code and relational mismatch
* Using ORM, the development process is quite simplified as it automates object to table and table to object conversion which results in lower development and maintenance cost
* The code is less as compared to embedded SQL
* Gives an optimized solution that results in faster application and easier maintenance.

### JDBC (Java Database Connectivity)

JDBC is an abbreviation for Java database connectivity. It is a JavaSoft (was once a division of Sun Microsystems, responsible for developing Java and for licensing the use of JVM) specification of standard API that allows Java programs to access database management systems. The JDBC API is a set of Java interfaces and classes that allow you to connect to databases and send queries. This API of JDBC makes use of drivers of JDBC in order to get connected along with the database. There are four types of JDBC drivers:

* JDBC-ODBC bridge plus ODBC drive (Type 1 driver)
* Native -API, partly Java driver (Type 2 driver)
* JDBC-Net, pure Java driver (Type 3 driver)
* Native-protocol, pure Java driver (Type 4 driver)

**Advantages of JDBC:**

* It assists in quickly establishing a connection with a data source.
* It allows you to submit queries and update statements.
* It also allows you to collect data from the database and process the results in a very simple method.

We can also perform numerous activities using JDBC API required to handle the database:

* Connection to the database
* Execution of queries and updation of statements to the database
* Retrieving results that are fetched from the database

### Difference Between ORM and JDBC

ORM when compared to JDBC is easier to work with as it does all the work by itself. It maps Java classes to the database variables via XML. While working with domain-driven applications and in the case of complex object relationships, ORM is mostly preferred but when the application is simple enough then it is better to use JDBC.

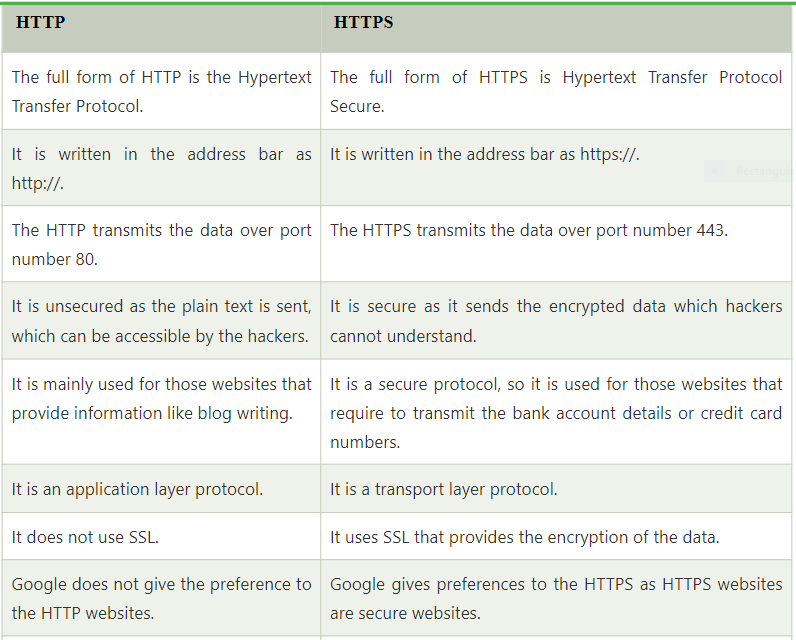
| Object Relational Mapping | Java Database Connectivity |
| --- | --- |
| Little slower than JDBC | It is faster compared to ORM |
| SQL queries requirement is comparatively quite less however this doesn’t mean that you have to do less work using ORM | SQL queries are required here |
| Hibernate framework (working on ORM technology) makes it easy to store objects/data to database automatically without writing manual code | We have to write code manually to store objects/ data in the database |
| The flow from Object/data to hibernate i.e. the frontend part is based on the ORM technique | Whereas when the data is stored in the database finally i.e., the backend part is still based on JDBCin |
| There are not many restrictions while dealing with data. Even a single database cell can be retrieved, changed, and saved. | JDBC comes with a lot of restrictions on extracting the result-set, processing it, and then committing it back to the database. |

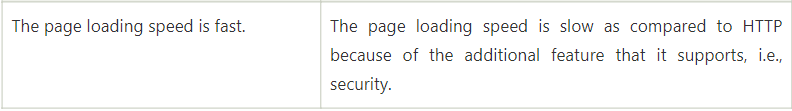
**Like**2

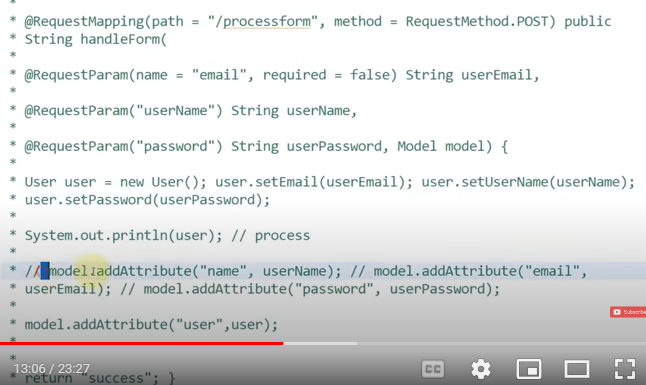
[Next](https://www.geeksforgeeks.org/spring-boot-starters/)

[**Spring Boot - Starters**](https://www.geeksforgeeks.org/spring-boot-starters/)

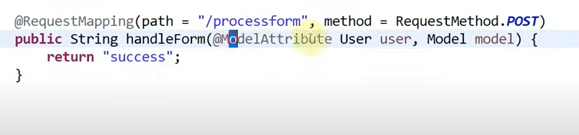
|  |  |
| --- | --- |
| ODBC | JDBC |
| ODBC Stands for Open Database Connectivity. | JDBC Stands for java database connectivity. |
| Introduced by Microsoft in 1992. | Introduced by SUN Micro Systems in 1997. |
| We can use ODBC for any language like C,C++,Java etc. | We can use JDBC only for Java languages. |
| We can choose ODBC only windows platform. | We can Use JDBC in any platform. |
| Mostly ODBC Driver developed in native languages like C,C++. | JDBC Stands for java database connectivity. |
| For Java applications it is not recommended to use ODBC because performance will be down due to internal conversion and applications will become platform Dependent. | For Java application it is highly recommended to use JDBC because there are no performance & platform dependent problem. |
| ODBC is procedural. | JDBC is object oriented. |







**Is equivalent to ->**



**Redirect in Spring MVC :**

* 1. **HttpServletResponce**
  2. **Public method(HttpServletResponce r)**
  3. **{**
  4. **R.sendRedirect(“”);**
  5. **Return””;**
  6. **}**
  7. **Redirect**
  8. 