Advanced Java & Web Technology

(INFO3101)

IT 3RD YEAR 1st SEMESTER

Module-III

- *JSP: JSP architecture, JSP servers, JSP tags, understanding the layout in JSP, Declaring variables,* methods in JSP, inserting java expression in JSP, processing request from user and generating dynamic response for the user, inserting applets and java beans into JSP, using include and forward action,comparing JSP and CGI program, comparing JSP and ASP program;

  Creating ODBC data source name,
- Introduction to JDBC, prepared statement and callable statement.

JSP

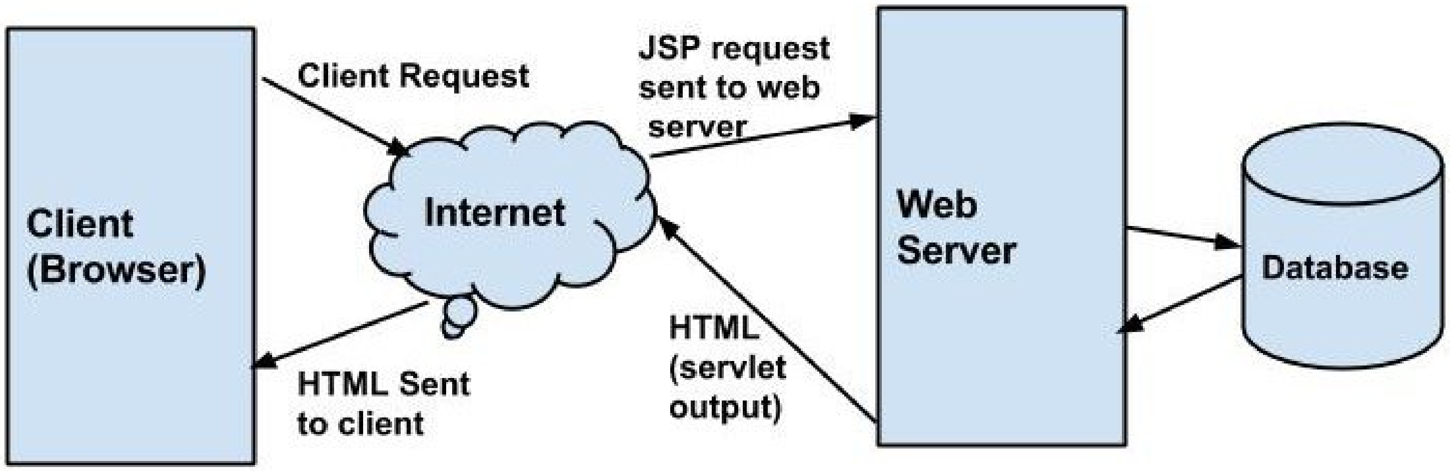
Java Server Pages (JSP) is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications. JSP have access to the entire family of Java APIs, including the JDBC API to access enterprise databases.

- JavaServer Pages (JSP) is a technology for developing Webpages that supports dynamic content. This helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.
- A JavaServer Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands.
- Using JSP, you can collect input from users through Webpage forms, present records from a database or another source, and create Webpages dynamically.
- JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages, and sharing information between requests, pages etc.

Why JSP is better than CGI

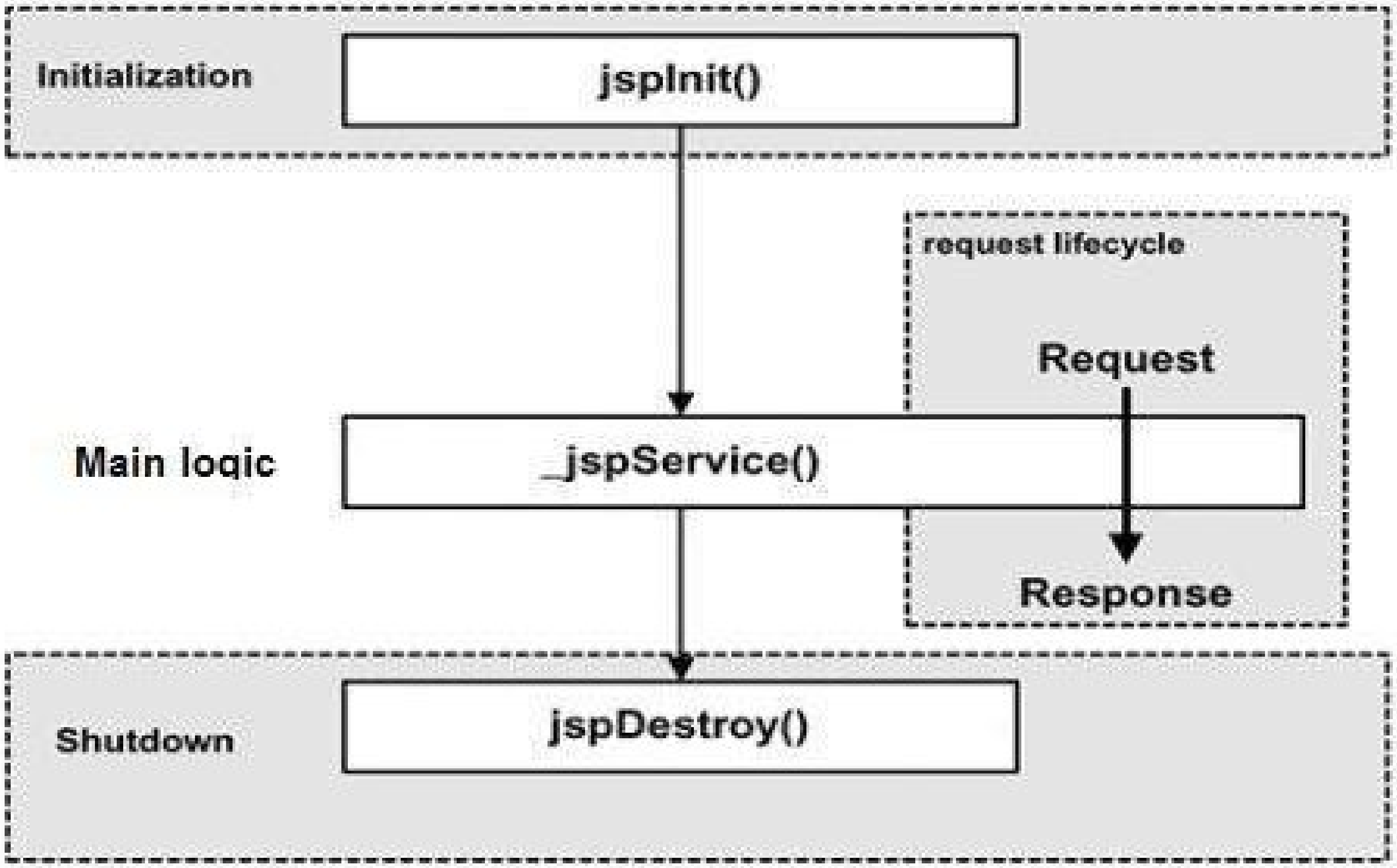
- Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having separate CGI files.
- JSP are always compiled before they are processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.
- JavaServer Pages are built on top of the Java Servlets API, so like Servlets, JSP also has access to all the powerful Enterprise Java APIs, including **JDBC, JNDI, EJB, JAXP,** etc.
- JSP pages can be used in combination with servlets that handle the business logic, the model supported by Java servlet template engines.

JSP Architecture



- The user goes to a JSP page and makes the request via internet in user’s web browser.
- The JSP request is sent to the Web Server.
- Web server accepts the requested **.jsp** file and passes the JSP file to the JSP Servlet Engine.
- If the JSP file has been called the first time then the JSP file is parsed otherwise servlet is instantiated. The next step is to generate a servlet from the JSP file. The generated servlet output is sent via the Internet form web server to users web browser.
- Now in last step, HTML results are displayed on the users web browser.

JSP Lifecycle



- The following are the paths followed by a JSP –

➢ Compilation

➢ Initialization

➢ Execution

➢ Cleanup

JSP Compilation

When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page. If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.

The compila on process involves three steps –

- Parsing the JSP.
- Converting the JSP into a servlet.
- Compiling the servlet.

JSP Initialization

- When a container loads a JSP it invokes the **jspInit()** method before servicing any requests. If you need to perform JSP-specific initialization, override the **jspInit()** method −
- public void jspInit(){ // Initialization code... }

  Typically, initialization is performed only once and as with the servlet init method, you generally initialize database connections, open files, and create lookup tables in the jspInit method.

JSP Execution

- This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed.
- Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the **_jspService()** method in the JSP.
- The _jspService() method takes an **HttpServletRequest** and an **HttpServletResponse** as its parameters as follows −
- void _jspService(HttpServletRequest request, HttpServletResponse response) { // Service handling code... }

JSP Cleanup

- The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.
- The **jspDestroy()** method is the JSP equivalent of the destroy method for servlets. Override jspDestroy when you need to perform any cleanup, such as releasing database connections or closing open files.
- public void jspDestroy() { // Your cleanup code goes here. }

JSP - Syntax

Elements of JSP

- The Scriptlet

  A scriptlet can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.

Following is the syntax of Scriptlet –

```
<% code fragment %>
```

- Example:

```
<html>
    <head>
            <title>Hello World</title>
    </head>
     <body> Hello World!<br/>
    <% out.println("Your IP address is " + request.getRemoteAddr()); %>
 </body>
 </html>
```

Let us keep the above code in JSP file **hello.jsp** and put this file in

C:\apache-tomcat7.0.2\webapps\ROOT directory. Browse through the same using URL **http://localhost:8080/hello.jsp**.

JSP Declarations

A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file.

Following is the syntax for JSP Declara   ons −

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

Example:

```
<%! int i = 0; %>
<%! int a, b, c; %>
```

JSP Expression

Syntax of JSP Expression is

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

```
<html>
<head><title>A Comment Test</title></head>

<body>
<p>Today's date: <%=(new
   java.util.Date()).toLocaleString()%></p>
 </body>
</html>
```

JSP Comments

```
<%-- This is JSP comment --%>
```

JSP - Directives

A JSP directive affects the overall structure of the servlet class. It usually has the following form –

```
<%@ directive attribute = "value" %>
```

Sr. No	Description
1	**<%@ page ... %>** Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.
2	**<%@ include ... %>** Includes a file during the translation phase.
3	**<%@ taglib ... %>** Declares a tag library, containing custom actions, used in the page

JSP - The page Directive

- The **page** directive is used to provide instructions to the container that pertain to the current JSP page.

Sr. No	Description
1	**buffer** Specifies a buffering model for the output stream.
2	**autoFlush** Controls the behavior of the servlet output buffer.
3	**contentType** Defines the character encoding scheme.
4	**errorPage** Defines the URL of another JSP that reports on Java unchecked runtime exceptions.
5	**isErrorPage** Indicates if this JSP page is a URL specified by another JSP page's errorPage attribute.
6	**extends** Specifies a superclass that the generated servlet must extend.

Sr. no	Description
7	**import** Specifies a list of packages or classes for use in the JSP as the Java import statement does for Java classes.
8	**info** Defines a string that can be accessed with the serervlet's **getServletInfo()** method.
9	**isThreadSafe** Defines the threading model for the generated servlet.
10	**language** Defines the programming language used in the JSP page.
11	**session** Specifies whether or not the JSP page participates in HTTP sessions.
12	**isELIgnored** Specifies whether or not the EL expression within the JSP page will be ignored.
13	**isScriptingEnabled** Determines if the scripting elements are allowed for use.

- ## The buffer Attribute

The **buffer** attribute specifies the buffering characteristics for the server output response object.

<%@ page buffer = "none" %>

<%@ page buffer = "8kb" %>

- ## The autoFlush Attribute

autoFlush attribute specifies whether buffered output should be flushed automatically when the buffer is filled, or whether an exception should be raised to indicate the buffer overflow.

A value of true (default) indicates automatic buffer flushing and a value of false throws an exception.

<%@ page autoFlush = "false" %>

<%@ page autoFlush = “true" %>

- ## The contentType Attribute

The contentType attribute sets the character encoding for the JSP page and for the generated response page. The default content type is **text/html**, which is the standard content type for HTML pages.

<%@ page contentType = "text/xml" %>

- The errorPage Attribute

  The errorPage attribute tells the JSP engine which page to display if there is an error while the current page runs. The value of the errorPage attribute is a relative URL.

  <%@ page errorPage = "MyErrorPage.jsp" %>

- The isErrorPage Attribute

  The isErrorPage attribute indicates that the current JSP can be used as the error page for another JSP.

  <%@ page isErrorPage = "true" %>

- The extends Attribute

  The extends attribute specifies a superclass that the generated servlet must extend.

  <%@ page extends = "somePackage.SomeClass" %>

- The import Attribute

  <%@ page import = "java.sql.*" %>

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

- The info Attribute

  The info attribute lets you provide a description of the JSP. The following is a coding example −

  <%@ page info = "This JSP Page Written By IT" %>

- The isThreadSafe Attribute

  The isThreadSafe option marks a page as being thread-safe. By default, all JSPs are considered thread-safe. If you set the isThreadSafe option to false, the JSP engine makes sure that only one thread at a time is executing your JSP.

  The following page directive sets the isThreadSafe option to false −

  <%@ page isThreadSafe = "false" %>

- The language Attribute

  The language attribute indicates the programming language used in scripting the JSP page.

  <%@ page language = "java" %>

- The session Attribute

  The session attribute indicates whether or not the JSP page uses HTTP sessions. A value of true means that the JSP page has access to a builtin **session** object and a value of false means that the JSP page cannot access the builtin session object.

  <%@ page session = "true" %>

- The isELIgnored Attribute

  The isELIgnored attribute gives you the ability to disable the evaluation of Expression Language (EL) expressions which has been introduced in JSP 2.0.

  <%@ page isELIgnored = "false" %>

- The isScriptingEnabled Attribute

  The isScriptingEnabled attribute determines if the scripting elements are allowed for use.

  The **default value (true)** enables scriptlets, expressions, and declarations.

  <%@ page isScriptingEnabled = "false" %>

▪The include Directive

The **include** directive is used to include a file during the translation phase. This directive tells the container to merge the content of other external files with the current JSP during the translation phase. You may code the ***include***directives anywhere in your JSP page.

<%@ include file = "relative url" >

▪ The taglib Directive

The JavaServer Pages API allow you to define custom JSP tags that look like HTML or XML tags and a tag library is a set of user-defined tags that implement custom behavior.

The taglib directive declares that your JSP page uses a set of custom tags, identifies the location of the library, and provides means for identifying the custom tags in your JSP page.

```
<%@ taglib uri = "http://www.example.com/custlib" prefix = "mytag" %>
<html>
    <body> <mytag:hello/> </body>
 </html>
```

JSP - Actions

JSP actions are special XML tags which control the behavior of servlet engine. **JSP actions** allow you to insert a file dynamically, reuse external JavaBean components, forward the request to the other page and generate HTML for Java Applet Plugin.

```
<jsp:action_name attribute = "value" />
```

There are two attributes that are common to all Action elements: the **id** attribute and the **scope** attribute.

- The <jsp:include> Action

```
<jsp:include page = "relative URL" flush = "true" />
```

S.No.	Attribute & Description
1	**page** The relative URL of the page to be included.
2	**flush** The boolean attribute determines whether the included resource has its buffer flushed before it is included.

Example

Let us define the following two files **(a)date.jsp** and **(b) main.jsp** as follows −

Following is the content of the **date.jsp** file −

```
<p>Today's date: <%= (new java.util.Date()).toLocaleString()%></p>
```

Following is the content of the **main.jsp** file −

```
<html>
<head>
    <title>The include Action Example</title>
</head>
    <body> <center> <h2>The include action Example</h2>
    <jsp:include page = "date.jsp" flush = "true" /> </center> </body>
</html>
```

The <jsp:useBean> Action

- The **useBean** action is quite versatile. It first searches for an existing object utilizing the id and scope variables. If an object is not found, it then tries to create the specified object.

```
<jsp:useBean id = "name" class = "package.class" />
```

Example

Calculator.java (a simple Bean class)

```
package com.abc;
public class Calculator{
  public int cube(int n){return n*n*n;}
 }
```

index.jsp file

```
<jsp:useBean id="obj" class="com.abc.Calculator"/>

<%
int m=obj.cube(5);
out.print("cube of 5 is "+m);
%>
```

Example

```
/* File: TestBean.java */
package action;
public class TestBean {
   private String message = "No message specified";
   public String getMessage() {
         return(message);
   }
   public void setMessage(String message) {
         this.message = message;
   }
}
```

Compile the above code to the generated **TestBean.class** file and make sure that you copied the TestBean.class in **C:\apache-tomcat-7.0.2\webapps\WEB-INF\classes\action** folder and the **CLASSPATH**variable should also be set to this folder –

main.jsp

```
 <html>
<head>
<title>Using JavaBeans in JSP</title>
</head>
<body>
<center>
<h2>Using JavaBeans in JSP</h2>
<jsp:useBean id = "test" class = "action.TestBean" /> <jsp:setProperty
   name = "test" property = "message" value = "Hello JSP..." />
  <p>Got message....</p>
 <jsp:getProperty name = "test" property = "message" />
</center>
</body>
</html>
```

Output

```
Using JavaBeans in JSP
Got message....
Hello JSP...
```

The <jsp:forward> Action

The **forward** action terminates the action of the current page and forwards the request to another resource such as a static page, another JSP page, or a Java Servlet.

```
<jsp:forward page = "Relative URL" />
```

Let us reuse the following two files **(a) date.jsp** and **(b) main.jsp** as follows

date.jsp

```
<p>Today's date: <%= (new java.util.Date()).toLocaleString()%></p>
```

main.jsp

```
<html>
<head>
<title>The include Action Example</title> </head>
<body>
<center>
<h2>The include action Example</h2>
<jsp:forward page = "date.jsp" />
</center>
</body>
</html>
```

What is JDBC?

JDBC stands for **J**ava **D**ata**b**ase **C**onnectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

- The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.
- Making a connection to a database.
- Creating SQL or MySQL statements.
- Executing SQL or MySQL queries in the database.
- Viewing & Modifying the resulting records.

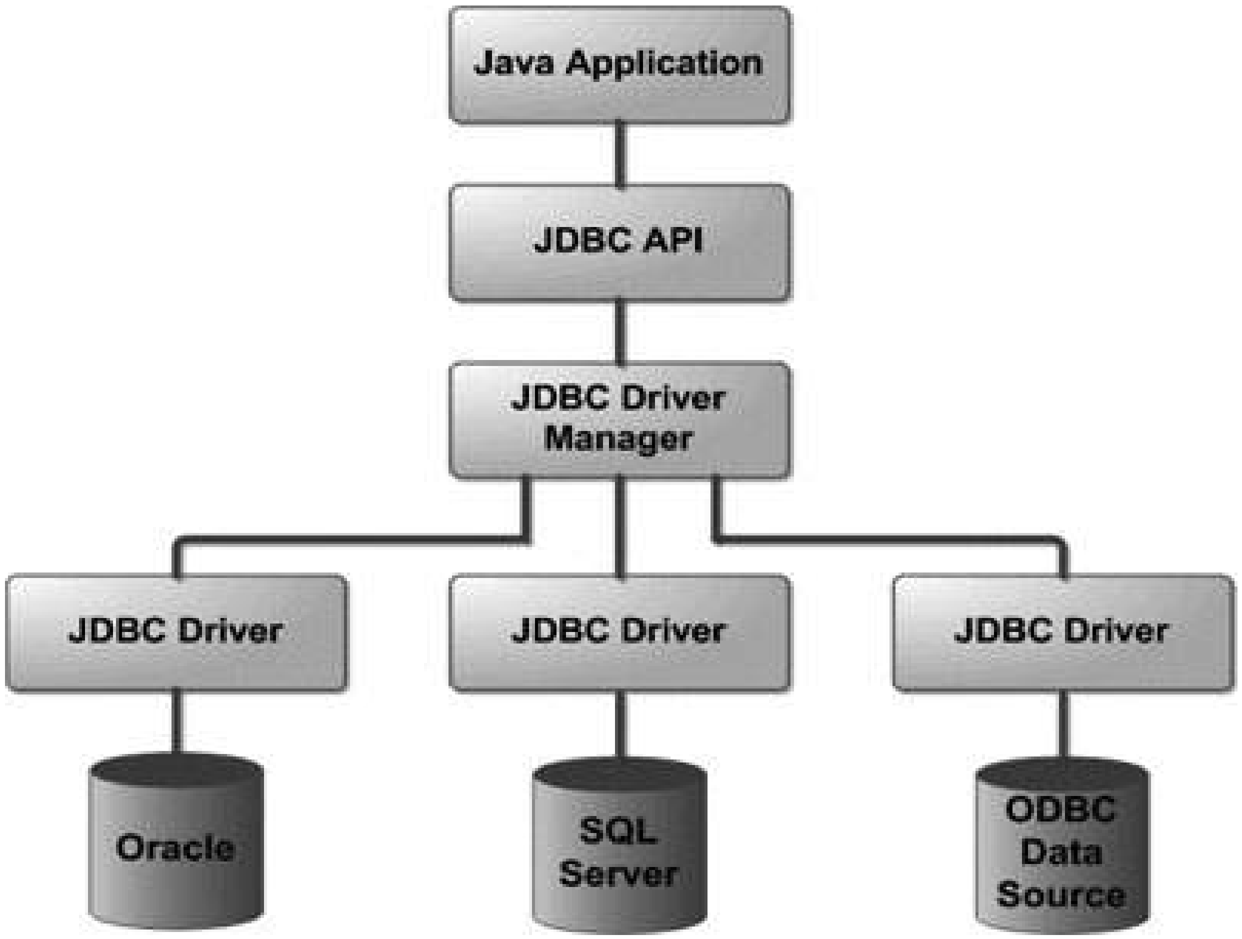
Java can be used to write different types of executables, such as –

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

JDBC Architecture

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

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


Java Application
JDBC API
JDBC Driver Manager
JDBC Driver
JDBC Driver
JDBC Driver
Oracle
SQL Server
ODBC Data Source

Common JDBC Components

- **DriverManager:** This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.

- **Driver:** This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.

- **Connection:** This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.

- **Statement:** You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.

- **ResultSet:** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.

- **SQLException:** This class handles any errors that occur in a database application.

Thank you