CSI3023-Advanced Server-Side Programming

Module-1

Servlet, JSP, JSF

Server Side Development
Tiered Architecture

✓ The two-tiered client-server model has been superseded by the multi-tiered architecture prevalent in the enterprise applications

- Allows each layer to communicate just with layers above and below it

✓ **Benefits of having a tiered application**

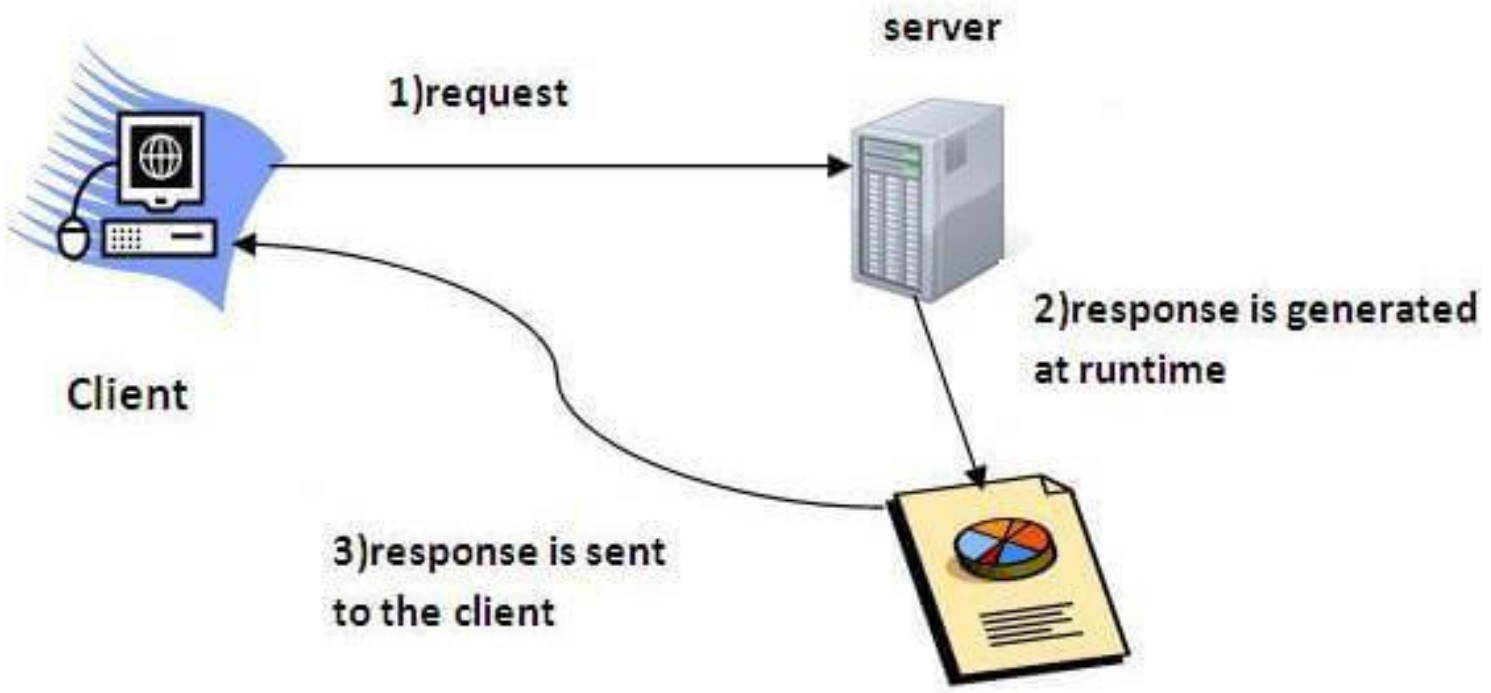
- Encapsulates rules and functionality together providing for easier maintenance & development
- Enhances flexibility and reusability of logic and software components
- Allows developers to focus on the area of their specialty e.g. database, servers, web page, etc.

Application/Browser (User Interface)

Web Server (Application Logic)

Database/ FileSystem (Persistent Storage)

Web - Tiered Architecture



Servlet, JSP, JDBC

Servlet

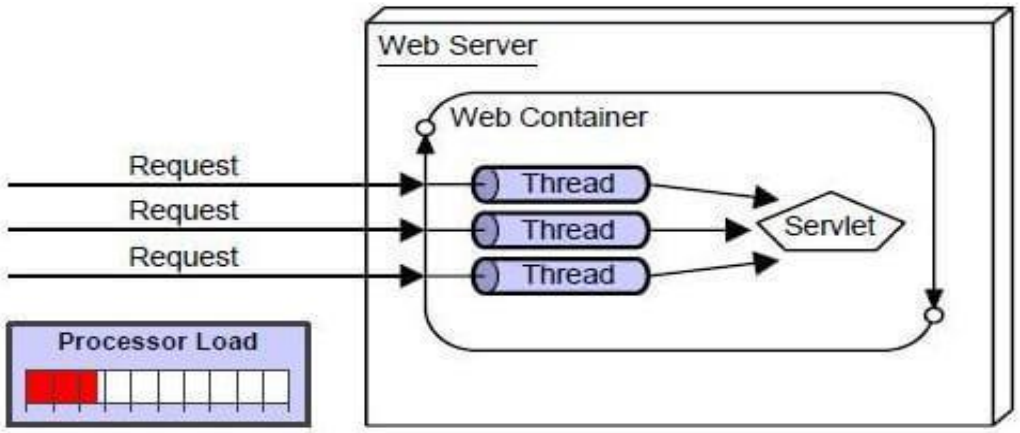
✓Servlet is a **server side Java based programming language** widely used to create dynamic web applications. Servlet runs inside a servlet container.

✓They are used to handle the request obtained from the web server, process the request, produce the response, then send a response back to the web server.

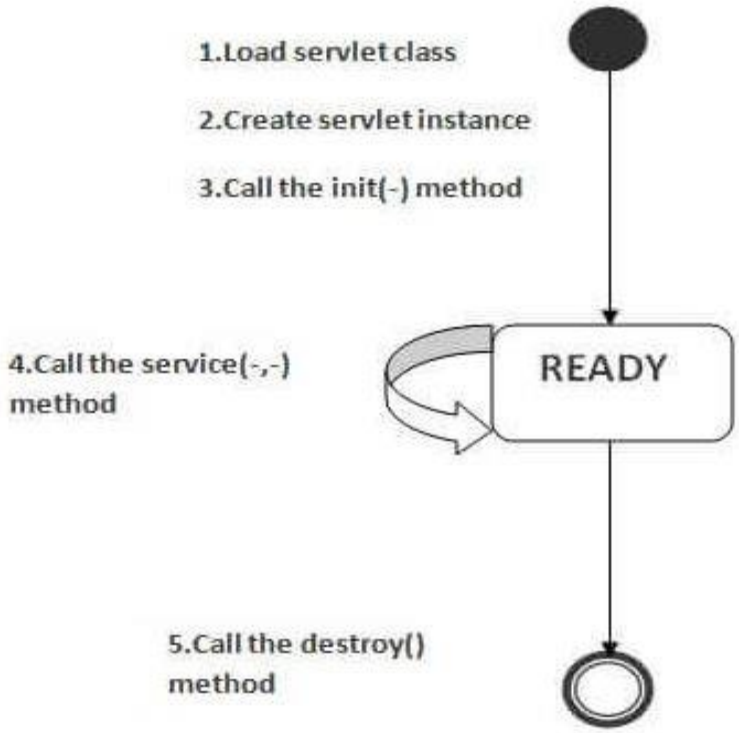
✓Servlets work on the server-side.

✓**Execution of Servlets basically involves six basic steps:**

1. Client sends a request
2. Web server receives the request
3. Sends to corresponding servlet
4. Servlet processes the request
5. Response to the server
6. Response to the client



Servlet Life Cycle



✓ The servlet is initialized by calling the **init()** method.
public void init() throws ServletException { // Initialization code... }

✓ The servlet calls **service()** method to process a client's request.

public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException { }

✓ The servlet is terminated by calling the **destroy()** method.
public void destroy() { // Finalization code... }

✓ Finally, servlet is garbage collected by the garbage collector of the JVM.

Servlet API

There are several varieties of interfaces and classes available in the Servlet API. Servlets are build from two packages:

✓ **javax.servlet** - This package contains various servlet interfaces and classes which are capable of handling **any type of protocol**.

✓ **javax.servlet.http** - This package contains various interfaces and classes which are capable of handling a **specific http type of protocol.**

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.*

Servlet Example

```
/ Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloWorld extends HttpServlet {

  private String message;

  public void init() throws ServletException {
    // Do required initialization
    message = "Hello World";
  }
```

Servlet Example

```
public void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException{

    // Set response content type
    response.setContentType("text/html");

    // Actual logic goes here.
    PrintWriter out = response.getWriter();
    out.println("<h1>" + message + "</h1>");
  }

  public void destroy() {
    // do nothing.
  }
}
```

Deployment Descriptor (web.xml)

✓ In a java web application a file named web.xml is known as deployment descriptor.

✓ It is a xml file and **<web-app>** is the root element for it.

✓ When a request comes web server uses web.xml file to map the URL of the request to the specific code that handle the request.

✓When url matched with url pattern web server try to **find the servlet name in servlet attributes** same as in servlet mapping attribute. When match found **control is goes to the associated servlet class.**

Deployment Descriptor (web.xml)

```
<web-app>
        <servlet>
                <servlet-name>HelloWorld</servlet-name>
                <servlet-class>HelloWorld</servlet-class>
        </servlet>

        <servlet-mapping>
                <servlet-name>HelloWorld</servlet-name>
                <url-pattern>/HelloWorld</url-pattern>
        </servlet-mapping>
</web-app>
```

<servlet-name>: We can provide any specific name that represents a servlet class. The name can be same as of class name.
<servlet-class>: A name of servlet class is given with this tag.
<url-patttern>: to use for requests whose URL matches the pattern

Using Annotations

- Annotations and web.xml are two ways to configure Java Servlets.
- Annotations are more convenient and prevent you from having bloated web.xml with hundreds of entries (in case of bigger application). It is also a part of Convention over configuration approach.
- The **@WebServlet** annotation is used to declare a servlet.
- The annotated class must extend

  the **javax.servlet.http.HttpServlet** class.

Using Annotations

Syntax:

```
@WebServlet(
   attribute1=value1,
   attribute2=value2,
   ...
)
public class TheServlet extends javax.servlet.http.HttpServlet
{
   // servlet code...
}
```

Example:

```
@WebServlet("/processForm")
public class MyServlet extends HttpServlet {
   public void doGet(HttpServletRequest request, HttpServletResponse
response)
         throws IOException {
      response.getWriter().println("Hello");
   }
}
```

Servlet Annotations

Attributes of @WebServlet Annotation:

Name	Type	Required	Description
value or **urlPatterns**	*String[]*	Required	Specify one or more URL patterns of the servlet. Either of attribute can be used, but not both.
name	*String*	Optional	Name of the servlet
displayName	*String*	Optional	Display name of the servlet
description	*String*	Optional	Description of the servlet
asyncSupported	*boolean*	Optional	Specify whether the servlet supports asynchronous operation mode. Default is false.
initParams	*WebInitParam[]*	Optional	Specify one or more initialization parameters of the servlet. Each parameter is specified by `@WebInitParam` annotation type.
loadOnStartup	*int*	Optional	Specify load-on-startup order of the servlet.
smallIcon	*String*	Optional	Specify name of the small icon of the servlet.
largeIcon	*String*	Optional	Specify name of the large icon of the servlet.

NOTE: the attributes *displayName*, *description*, *smallIcon* and *largeIcon* are primarily used by tools, IDEs or servlet containers, they do not affect operation of the servlet.

Servlet Request methods

String getParameter(String name)	This method returns the value given in request as a String.
Object getAttribute(String name)	This method provides the attribute of request as an Object.
String getServerName()	This method provides the server name to which request is sent.
int getServerPort()	This method returns the port number to which request is sent.
boolean isSecure()	This method indicates whether the server is secure or not.

Servlet Response Methods

Method	Description
PrintWriter getWriter()	This method is used to send character data in response.
int getBufferSize()	This method returns the capacity of buffer in sent response.
ServletOutputStream getOutputStream()	This method is used to send binary data in response.
boolean isCommited()	This method indicates the completion of response.
void reset()	This method is used to remove the data present in buffer.
void setContentType(String type)	This method is used to set the type of content.

Servlet Request and Response

```
public class HttpServletReq extends HttpServlet
{
 public void doGet(HttpServletRequest req,
HttpServletResponse res) throws ServletException,
IOException
 {
 res.setContentType("text/html");
 PrintWriter pw=res.getWriter();
 String s1=req.getParameter("username");
 pw.println("<h1>Welcome:"+s1+"</h1>");
 String s2=req.getServerName();
 pw.println("<h1>Server name:"+s2+"</h1>");
 int i=req.getServerPort();
 pw.println("<h1>Server Port:"+i+"</h1>");
 }
 }
```

```
<form action="serv" method="get">
<h3>Enter your Name:</h3>
<input type="text" name="username">
<input type="submit" value="submit">
</form>
```


SCOPE

Servlet Request Dispatcher

✓RequestDispacther is an interface used to receive requests from the users and bind it with other files such as HTML file, Servlet file, JSP file etc.

✓Servlet container is responsible to create RequestDispatcher object.

✓ RequestDispacther provides **forward()** and **include()** methods. These methods are used to call RequestDispacther.

Method	Description
void forward(ServletRequest req, ServletResponse res)	If this method is invoked then the request of current file is send forward to the another file of any type such as HTML, Servlet, JSP etc. and the response of that file is provided by the server.
void include(ServletRequest req, ServletResponse res)	If this method is invoked then the content of current file such as HTML, Servlet, JSP is included with the response.

Servlet Request Dispatcher

```
public class ServletChecker extends HttpServlet
{
 public void doPost(HttpServletRequest req,HttpServletResponse res) throws
ServletException,IOException
 {
 res.setContentType("text/html");
 PrintWriter pw=res.getWriter();
 String st1=req.getParameter("pass");
 int i1=st1.length();
 if(i1<8)
 {
 pw.println("<h1>Error:Password must be of 8 character</h1>");
 RequestDispatcher rd=req.getRequestDispatcher("/index.html");
 rd.include(req,res);
 }
 else
 {
 RequestDispatcher rd1=req.getRequestDispatcher("ServletForward");
 rd1.forward(req,res);
 }
 }
}
```

Index.html

```
<form action="serv" method="get">
<h3>Enter your Name:</h3>
<input type="text" name="username">
<input type="“password" name="“pass">
<input type="submit" value="submit">
</form>
```

Servlet Request Dispatcher

```
public class ServletForward extends HttpServlet
{
 public void doPost(HttpServletRequest
req,HttpServletResponse res) throws
ServletException,IOException
 res.setContentType("text/html");
 PrintWriter pw=res.getWriter();
 String st1=req.getParameter("name");
 pw.println("<h1>Welcome "+ st1+"</h2>");
 pw.println("<h3>You registered successfully</h3>");
 }}
```

Servlet Redirection

✓ The **sendRedirect()** method of **HttpServletResponse** interface can be used to redirect response to another resource, it may be servlet, jsp or html file.

✓ It works at client side because it uses the url of the browser to make another request. So, it can work inside and outside the server.

Syntax: **public void** sendRedirect(String URL)**throws** IOException;

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

```
public class DemoServlet extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletResponse res)
                                                    throws
ServletException,IOException
{
        res.setContentType("text/html");
        PrintWriter pw=res.getWriter();

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

Servlet Redirection

Redirected to another servlet named “redirected”

```
protected void doGet(HttpServletRequest req, HttpServletResponse
resp){
   resp.sendRedirect(req.getContextPath() + "/redirected");
}
```

forward() method	sendRedirect() method
The forward() method works at server side.	The sendRedirect() method works at client side.
It sends the same request and response objects to another servlet.	It always sends a new request.
It can work within the server only.	It can be used within and outside the server.
Example: request.getRequestDispacher("servlet2").forward(request,response);	Example: response.sendRedirect("servlet2");

Servlet Session Management

- A session is a sequence of related requests and responses between a Web client and a Web server
  - In one session, an online shopper may browse the e-catalogue of an online store, add an item into an online shopping cart, check out the shopping cart and provide his or her credit card information
  - In another session, a shopper may just browse the catalogue without putting anything in the shopping cart
- HTTP is known to be a state-less protocol
  - This means that there is nothing in the protocol to help the Web server to remember the contents of a previous GET/POST request from the Web client
    - i.e., requests to a server are independent of each other
  - In other words, the HTTP protocol does not provide direct support for us to remember previous communications

Session Tracking

- Different ways to maintain session
  1. HttpSession
  2. Cookies
  3. Hidden Form Filed
  4. URL Rewriting

Servlet Session Management

The HTTP protocol is stateless, so the server and the browser should have a way of storing the identity of the user through multiple requests

1. The browsers sends the **first request** to the server
2. The **server checks** whether the browser has identified with the session cookie.

3.1. if the server doesn't 'know' the client:

- the **server creates a new unique identifier**, and puts it, as a key, whose value is the newly created Session. It also sends a cookie response containing the unique identifier.
- the **browser stores the session cookie** containing the unique identifier, and uses it for each subsequent request to identify itself uniquely.

3.2. if the server already knows the client - the **server obtains** the Session corresponding to the passed unique identifier found in the session cookie

HttpSession

- HttpSession is an interface used by servlet container to create a session between Http client and Http server. So using this interface we can maintain the state of a user.
- The object of HttpSession stores various information about the user. This interface is present in javax.servlet.http package.

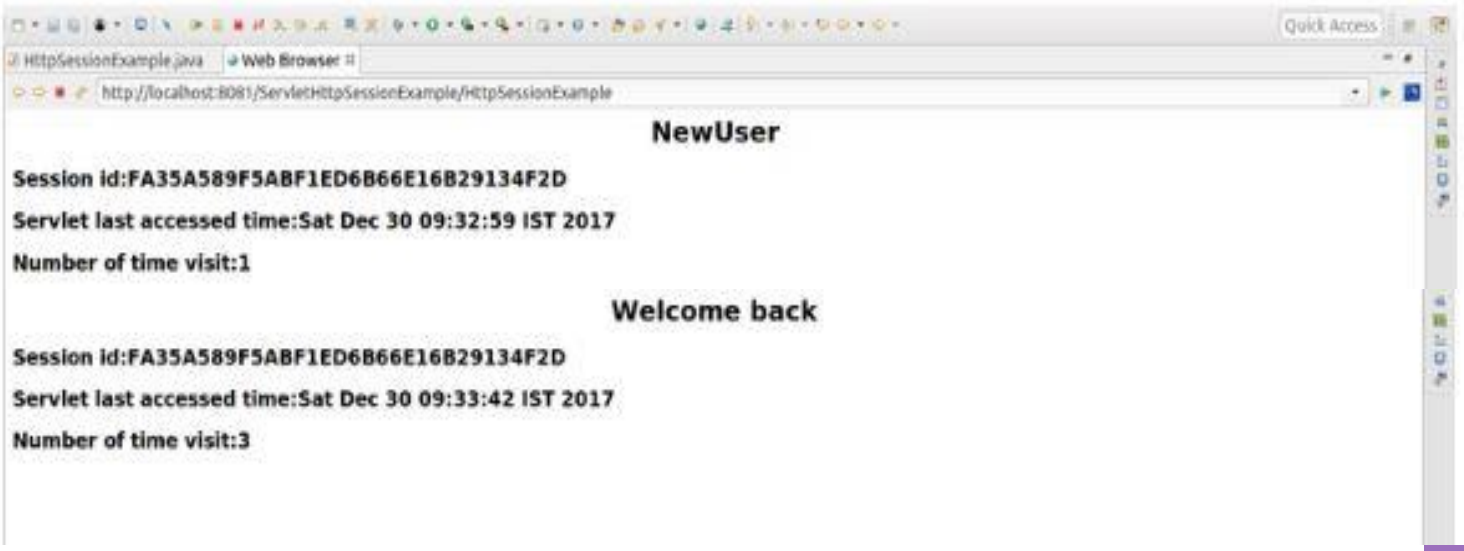
String getId()	This method returns a unique id of client used to identify it.
long getLastAccessedTime()	This method provides the time when last request is made. It returns the time in milliseconds.
Long getCreationTime()	This method provides the time when first request is made. It returns the time in milliseconds.
void setMaxInactiveInterval()	This method sets time in seconds after which the session becomes invalid.
Object getAttribute(String name)	This method provides the value of attribute bind with the name passed within it.

HttpSession Example

```
public class HttpSessionExample extends HttpServlet {
 public void doGet(HttpServletRequest req,HttpServletResponse
res) throws ServletException, IOException {
 res.setContentType("text/html");
 HttpSession session = req.getSession();
 res.setContentType("text/html");
 PrintWriter pw=res.getWriter();
 Integer attribute=(Integer)session.getAttribute("attribute");
 if (attribute == null)
 {
            attribute = new Integer(1);
            pw.println("<h1><center>NewUser</center></h1>");
 } else {
            pw.println("<h1><center>Welcome
back</center></h1>");
            attribute = new Integer(attribute.intValue() +
1);
 }
```

HttpSession Example

```
session.setAttribute("attribute",attribute);
 session.setMaxInactiveInterval(60);
 pw.println("<h2>Session id:"+session.getId()+"</h2>");
 pw.println("<h2>Servlet last accessed time:"+new
Date(session.getLastAccessedTime())+"</h2>");
 pw.println("<h2>Number of time
visit:"+attribute+"</h2>");
  }
}
```



JDBC Connectivity

- JDBC is: **J**ava **D**ata**B**ase **C**onnectivity
  - is a Java API for connecting programs written in Java to the data in relational databases.
  - consists of a set of classes and interfaces written in the Java programming language.
  - provides a standard API for tool/database developers and makes it possible to write database applications using a pure Java API.
- JDBC:
  - establishes a connection with a database
  - sends SQL statements
  - processes the results.

JDBC Steps

1. Importing Packages
2. Registering the JDBC Drivers
3. Opening a Connection to a Database
4. Creating a Statement Object
5. Executing a Query and Returning a Result Set Object
6. Processing the Result Set
7. Closing the Connection

1: Importing Packages

JDBC API 4.0 mainly provides 2 important packages:

i) java.sql - package provides classes and interfaces to perform most of the JDBC functions like creating and executing SQL queries

ii) javax.sql - a JDBC extension API and provides server-side data access and processing in Java Program.

```
//Import packages
import java.sql.*;  //JDBC packages
import java.math.*;
import java.io.*;
```

2: Registering JDBC Drivers

✓ load/register the driver in the program before connecting to the Database.

✓ You need to register it only once per database in the program.

✓ load the driver in the following 2 ways:

i) Class.forName()

ii) DriverManager.registerDriver()

DB Name	JDBC Driver Name
MySQL	com.mysql.jdbc.Driver
Oracle	oracle.jdbc.driver.OracleDriver
Microsoft SQL Server	com.microsoft.sqlserver.jdbc.SQLServerDriver
MS Access	net.ucanaccess.jdbc.UcanaccessDriver

3: Opening connection to a Database

✓ DriverManager class has the **getConnection method**, we will use this method to get the connection with Database.

getConnection(URL,username,password);

– It has 3 parameters URL, username, password.

Database	Connection String/DB URL
MySQL	jdbc:mysql://HOST_NAME:PORT/DATABASE_NAME
Oracle	jdbc:oracle:thin:@HOST_NAME:PORT:SERVICE_NAME
Microsoft SQL Server	jdbc:sqlserver://HOST_NAME:PORT;DatabaseName=< DATABASE_NAME>

Example:

Connection con = DriverManager.getConnection(jdbc:oracle:thin:@localhost:1521:xe,System,Pass123@)

4. Creating a Statement Object

(i) Create Statement

✓ create the statement object that runs the query with the connected database.

✓ use the createStatement method of the ***Connection*** class to create the query.

There are 3 statement interfaces are available in the java.sql package.

i) **Statement -** used to implement simple SQL statements with no parameter

ii) **Prepared Statement -** used to implement parameterized and precompiled SQL statements.

iii) **Callable Statement -** used to implement a parameterized SQL statement that invokes procedure or function in the database

Statement statemnt1 = conn.createStatement();
String select_query = “Select * from states where state_id = 1”;
PreparedStatement prpstmt = conn.prepareStatement(select_query);
CallableStatement callStmt = con.prepareCall("{call procedures(?,?)}");

5. Executing a Query

There are 4 important methods to execute the query in Statement interface

1. **ResultSet executeQuery(String sql)** - used to execute the SQL query and retrieve the values from DB
2. **int executeUpdate(String sql)** - used to execute value specified queries like INSERT, UPDATE, DELETE (DML statements)
3. **Boolean execute(String sql)** - used to execute the SQL query. It returns **true** if it executes the SELECT query. And, it returns **false** if it executes INSERT or UPDATE query
4. **int []executeBatch()** - to execute a batch of SQL queries to the Database and if all the queries get executed successfully, it returns an array of update counts.

Example:

ResultSet rs1= statemnt1.**executeQuery(**QUERY));

6. Processing the Result Set

✓ the queries using the executeQuery() method, the result will be stored in the ResultSet object.

ResultSet rs1= statemnt1.**executeQuery(**QUERY));

✓ A ResultSet object points to the current row in the Resultset. To iterate the data in the ResultSet object, call the **next()** method in a while loop.

✓ If there is no more record to read, it will return FALSE.

7. Closing the Connection

✓Finally, we are done with manipulating data in DB. Now we can close the JDBC connection.

✓ We need to make sure that we have closed the resource after we have used it

conn.close();

JDBC Program

```
import java.sql.*;
public class JDBCDemo {
        public static void main(String[] args) {
        try{
        Class.forName("com.mysql.cj.jdbc.Driver");

        Connection con=DriverManager.getConnection(
        "jdbc:mysql://localhost:3306/demo","root",“password");

         Statement stmt=con.createStatement();
        ResultSet rs=stmt.executeQuery("select * from student");
        while(rs.next())
        System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"
");
        con.close();
        }catch(Exception e){ System.out.println(e);}
     }
    }
```

Thank you