Servlet , JDBC

1. What are advantages of servlet over CGI?

Advantages of Servlets over CGI:

- **Platform independence:**
  Servlets run on top of Java Virtual Machine (JVM) so they are platform independent components. On the other hand CGIs run on the operating system itself so they are platform dependent.
- **Performance**
  CGIs run on separate processes, so it takes more time to start a new process for each and every requests. Servlets are accessed through threads and those threads also being kept in a thread pool. It increases the performance.
- **Security**
  Servlets are running inside the sand box of JVM, so it is hard to damage the server side modules by malfunctioning the Servlets. Since CGIs are native applications, using CGIs a hacker can damage the server side components easily compared to Servlets.
- **ErrorHandling**
  Servlets provides easiest error handling mechanism jointly with web container. CGIs do not have such a powerful error handling support.
- **Extensibility**
  Servlets are just Java classes so it is easy maintain and extend their functionality. All the object oriented concepts can be directly applied to Servlets as well. CGIs are mostly written in scripting languages like Perl so the extensibility is very poor in case of CGIs.

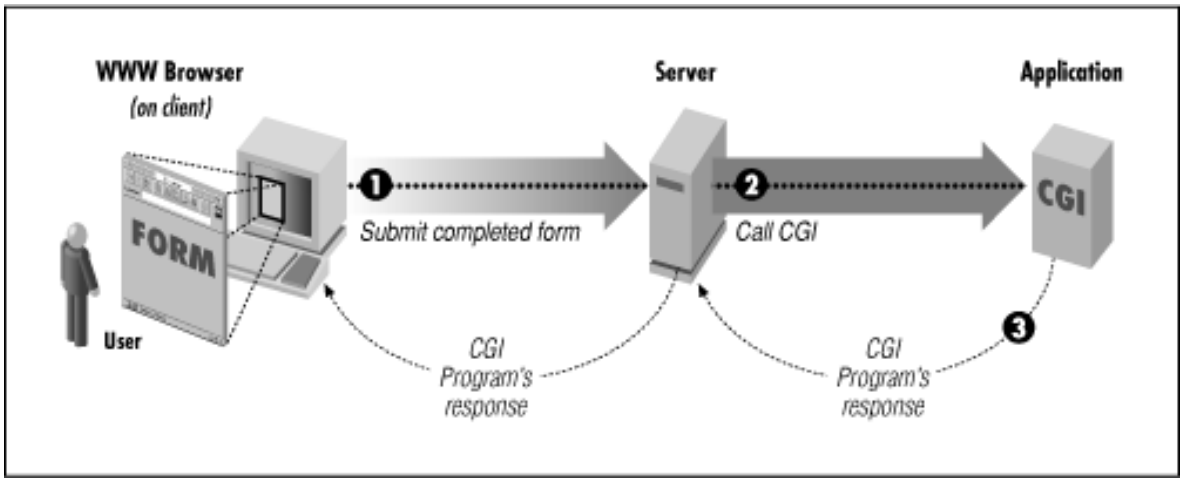
2. Differentiate CGI and Servlet

CGI	Servlet
It is a process based. i.e., for every request, a new process will be created and that process is responsible to generate required response.	It is thread based. i.e., for every request new thread will be created and that thread is responsible to generate required response.
Creation and destruction of new process for every request is costly, if the number of requests increases then the performance of the system goes down. Hence CGI technology fails to deliver scalable applications.	Creation and destruction of a new thread for every request is not costly, hence if the number of requests increases there is no change in response time. Due to this, servlet technology succeeds to deliver scalable applications.
Two processes never share common address space. Hence there is no chance of occurring concurrence problems in CGI	All the threads shares the same address space, Hence concurrence problem is very common in servlets.
We can write CGI program in variety of languages, But most popular language is perl.	We can write servlets only in java.
Most of the CGI languages are not object oriented. Hence we miss the benefits of oops.	Java language itself is object oriented. Hence we can get all the key benefits of oops.
Most of CGI languages are platform dependent.	Java language is platform independent.

3. What is Servlet? Write Short note on Java Servlet Technology.

Servlet: It's a module written using Java that runs in a server application to answer client requests.

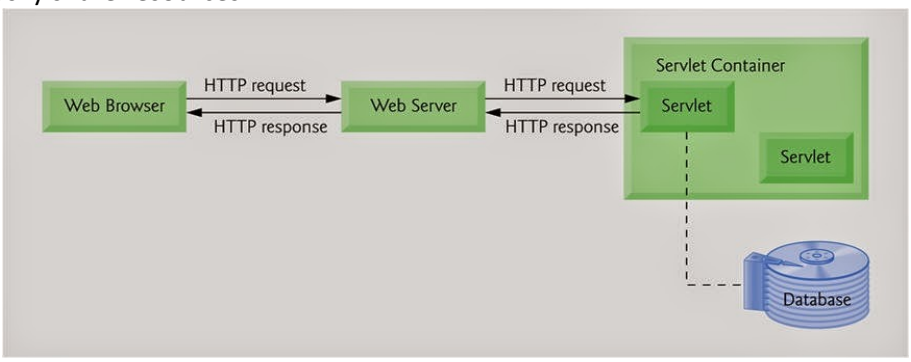
- ¡ It's most commonly used with HTTP and the word servlet is often used in the meaning of HTTP servlet
- ¡ It's supported by virtually all web servers and application servers
- ¡ Can easily share resources



4. What is Servlet? Write a short note on Java Servlet Technology. Describe Servlet Architecture.

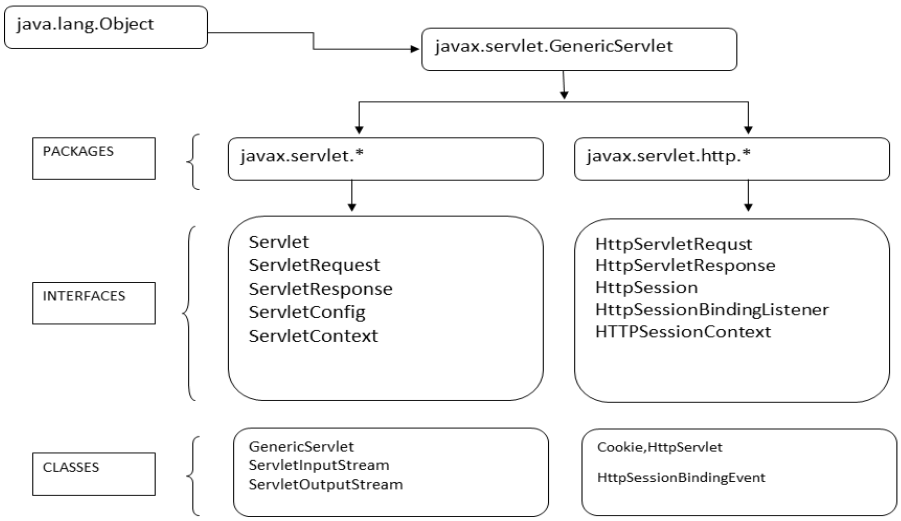
Servlet:

- It's a module written using Java that runs in a server application to answer client requests.
- It's most commonly used with HTTP and the word servlet is often used in the meaning of HTTP servlet
- It's supported by virtually all web servers and application servers
- Can easily share resources



- Servlets read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
- Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
- Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
- Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
- Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

Servlet Architecture:



Servlet architecture made by two packages:

1. javax.servlet.Servlet
2. javax.servlet.http

The **javax.servlet.Servlet** package contains the generic interfaces and Classes that are implemented and extended by all **Servlets**. The **javax.servlet.http** package contains the Classes that are extended when creating **HTTP-specific Servlets**.

The main root of the **Servlet Architecture** is the **javax.servlet.Servlet**. it has following interfaces:

- Servlet
- ServletRequest
- ServletResponse
- RequestDispatcher
- ServletConfig
- ServletContext
- SingleThreadModel
- Filter
- FilterConfig

Classes included in javax.servlet package are as follows:

- GenericServlet
- ServletInputStream
- ServletOutputStream
- ServletRequestWrapper
- ServletResponseWrapper
- ServletRequestEvent
- ServletContextEvent
- ServletRequestAttributeEvent

- ServletException

Interfaces in javax.servlet.http package

There are many interfaces in javax.servlet.http package. They are as follows:

- HttpServletRequest
- HttpServletResponse

- HttpSessionContext (deprecated now)
- Classes in javax.servlet.http package

There are many classes in javax.servlet.http package. They are as follows:

- HttpServlet
- Cookie
- HttpServletRequestWrapper
- HttpServletResponseWrapper
- HttpSessionEvent
- HttpSessionBindingEvent
- Htt

5. Describe servlet life cycle.

A servlet life cycle can be defined as the entire process from its creation till the destruction. The lifecycle of a servlet is controlled by the container in which the servlet has been deployed. When a request is mapped to a servlet, the container performs the following steps:

- If an instance of the servlet does not exist, the web container
  - Loads the servlet class.
  - Creates an instance of the servlet class.
  - Initializes the servlet instance by calling the init method.
- Invokes the service method, passing request and response objects. If it needs to remove the servlet, the container finalizes the servlet by calling the servlet's destroy method.

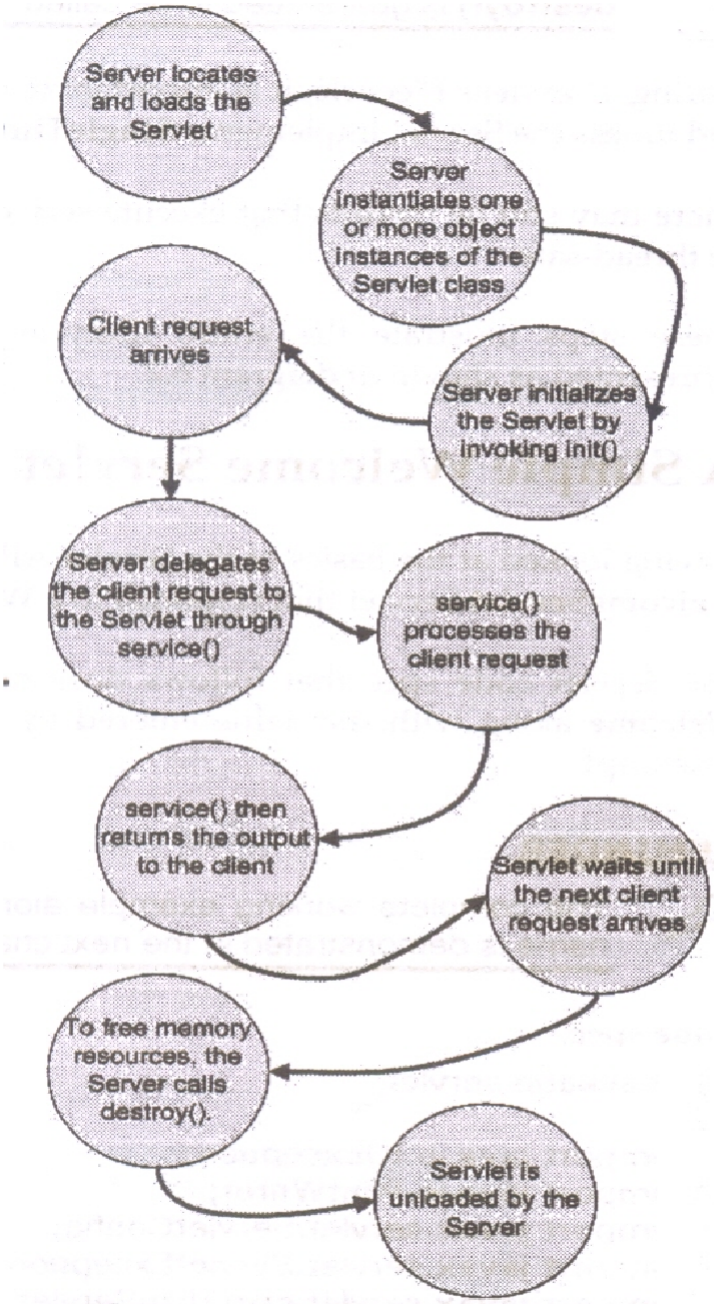


Diagram 6.2: The Servlet Lifecycle

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. 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.
2. **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.
   -

3. 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 invoke 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.

4. 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 conatiner releases all the references of this Servlet instance so that it becomes eligible for garbage collection.

6. Write a note on RequestDispatcher

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.

To obtain the object of RequestDispatcher:-

The getRequestDispatcher() method of ServletRequest interface returns the object of RequestDispatcher. Syntax:

- public RequestDispatcher getRequestDispatcher(String resource);

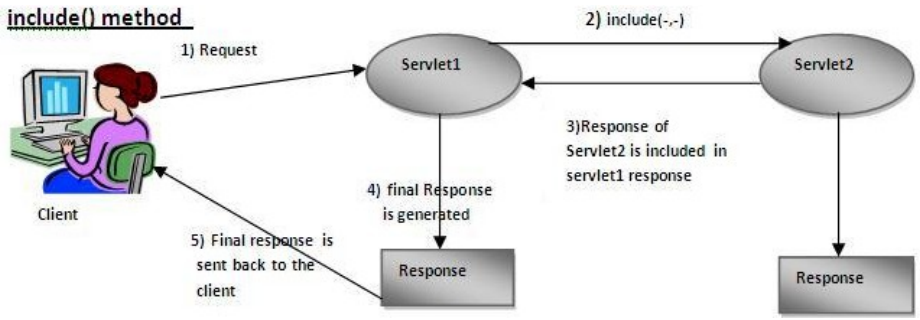
Example of using getRequestDispatcher method

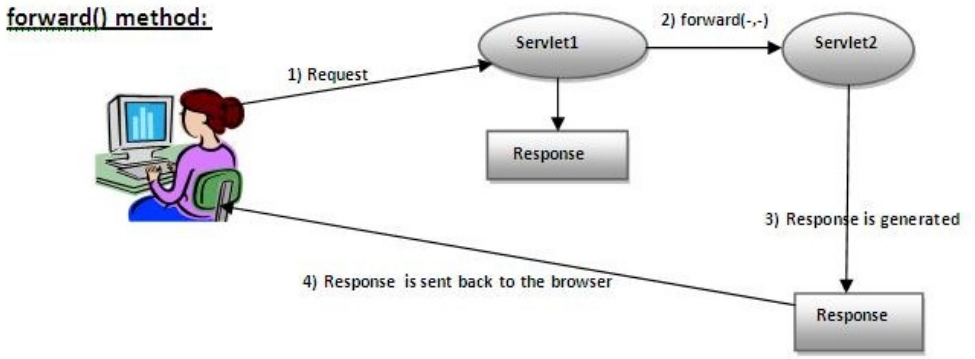
- RequestDispatcher rd=request.getRequestDispatcher("servlet2");
- //servlet2 is the url-pattern of the second servlet

There are two methods defined in the RequestDispatcher interface.

- **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.

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





index.html

```
<form action="servlet1" method="post">
Name:<input type="text"
name="userName"/><br/>
Password:<input type="password" name="userPass"/><br/>
<input type="submit" value="login"/>
</form>
```

Login.java

```
import
java.io.*;
import javax.servlet.*;
import
javax.servlet.http.*;
public class Login extends HttpServlet {
  public void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletExcep tion, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String
    n=request.getParameter("userName");
    String
    p=request.getParameter("userPass");

    if(p.equals("servlet"){
       RequestDispatcher rd=request.getRequestDispatcher("servlet2");
       rd.forward(request, response);


    }
    else{
       out.print("Sorry UserName or Password Error!");
       RequestDispatcher rd=request.getRequestDispatcher("/index.html");
       rd.include(request, response);
     }
    }
}
```

WelcomeServlet.ja

```
va import java.io.*;
import
javax.servlet.*;
import javax.servlet.http.*;
public class WelcomeServlet extends HttpServlet {
     public void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletExc eption, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
```

}

7. Differentiate ServletConfig and Servlet Context

ServletConfig	ServletContext
• ServletConfig available in javax.servlet.*; package • ServletConfig object is one per servlet class • Object of ServletConfig will be created during initialization process of the servlet • This Config object is public to a particular servlet only • ***Scope*: As long as a servlet is executing, ServletConfig object will be available, it will be destroyed once the servlet execution is completed.** • We should give request explicitly, in order to create ServletConfig object for the first time • In web.xml – <*init-param*> tag will be appear under <*servlet-class*> tag	• ServletContext available in javax.servlet.*; package • ServletContext object is global to entire web application • Object of ServletContext will be created at the time of web application deployment • *Scope*: As long as web application is executing, ServletContext object will be available, and it will be destroyed once the application is removed from the server. • ServletContext object will be available even before giving the first request • In web.xml – <*context-param*> tag will be appear under <*web-app*> tag

GET	POST
1) In case of Get request, only **limited amount of data** can be sent because data is sent in header.	In case of post request, **large amount of data** can be sent because data is sent in body.
2) Get request is **not secured** because data is exposed in URL bar.	Post request is **secured** because data is not exposed in URL bar.
3) Get request **can be bookmarked**	Post request **cannot be** bookmarked
4) Get request is **idempotent**. It means second request will be ignored until response of first request is delivered.	Post request is **non-idempotent**
5) Get request is **more efficient** and used more than Post	Post request is **less efficient** and used less than get.

	GenericServlet	HTTPServlet
1	Generic Servlet class is super class/parent class of HTTP servlet in servlet architecture.	HTTP servlet class is sub class/child class of Generic Servlet class in servlet architecture.
2	The javax.servlet package contains the generic interfaces and classes that are implemented &extended by the servlets.	The javax.servlet.http package contains the classes that are extended when creating http specific servlets.
3	This class implements Servlet interface.	This class extends Generic Servlet class.
4	This class implements Servlet Config which handles initialization parameter & Servlet Context.	This class implements service() of generic class by calling method specific implementation to http request method(i.e. doGet() and doPost()).
5	Servlet that extends this class are protocol independent. They do not contain any support for HTTP or other protocol.	Servlet that extends this class have built-in support for HTTP protocol.
6	The GenericServlet does not include protocol-specific methods for handling request parameters, cookies, sessions and setting response headers.	The HttpServlet subclass passes generic service method requests to the relevant doGet() or doPost() method.
7	GenericServlet is not specific to any protocol.	HttpServlet only supports HTTP and HTTPS protocol.
8	javax.servlet.GenericServlet	javax.servlet.http.HttpServlet
9	GenericServlet defines a generic, protocol-independent servlet.	HttpServlet defines a HTTP protocol specific servlet.

Q. What is JDBC?

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

The JDBC library includes APIs for each of the tasks commonly associated with database usage:

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

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

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

Q. Explain the architecture of JDBC in detail.

JDBC Architecture:

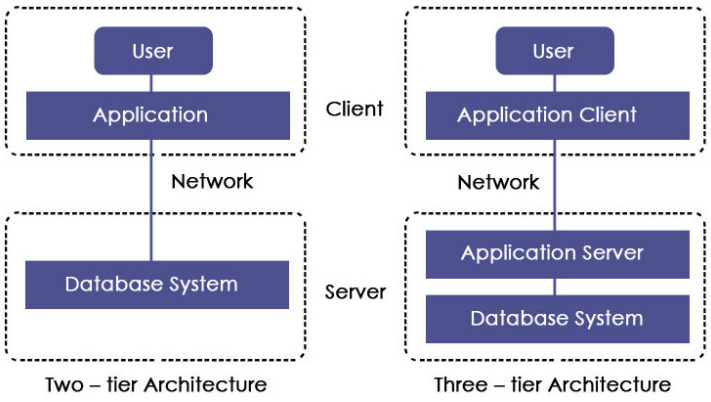
JDBC supports two types of processing models for accessing database i.e. two-tier and three-tier.

1. Two-tier Architecture:

This architecture helps java program or application to directly communicate with the database. It needs a JDBC driver to communicate with a specific database. Query or request is sent by the user to the database and results are received back by the user. The database may be present on the same machine or any remote machine connected via a network. This approach is called client-server architecture or configuration.

2. Three-tier Architecture:

In this, there is no direct communication. Requests are sent to the middle tier i.e. HTML browser sends a request to java application which is then further sent to the database. Database processes the request and sends the result back to the middle tier which then communicates with the user. It increases the performance and simplifies the application deployment.



The JDBC API provides the following interfaces and classes:

- **DriverManager:** This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication subprotocol. 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

- **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.

Q. List and explain advantages of JDBC

- The combination of JAVA API and JDBC API results into the easy way and economically more efficient application development.
- With JDBC API, no configuration information is required on the client side. It is a JDBC URL or a Data Source object registered with a JNDI(Java Naming and Directory Interface) which defines all information required to establish a connection with a driver written in java programming language.
- JDBC API is available everywhere on the Java EE platform, which provides support for write once, Run anywhere paradigm as a developer can actually write database application once and access data anywhere.
- JDBC API facilitates the development of sophisticated application by providing metadata access

Q. Write and explain the steps for Java Database Connectivity.

The java program can access the database using JDBC. The seven steps that are needed to access database are

1. Import the package
2. Load and register the driver
3. Establish the Connection
4. Create a Statement object
5. Execute a query
6. Process the result
7. Close the connection

Step1: Import java.sql package

The important classes and interfaces of JDBC are available in the java.sql package . So inorder to access the databases it must be imported in our application.

import java.sql.*;

Step 2: Loading and registering the driver

In order to access database it must have the particular driver for that database. In Java we have to inform the Drive Manager about the required driver this is done with the help of the method forName(String str) available in the class named Class

Class.forName("path with driver name");

In an application the user can register more than one driver. The statement used to load and register the JDBC-ODBC bridge driver is

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

Step 3: Establishing the connection

Establishing connection means making a connection to access RDBMS through JDBC driver. The connection can be established with the help of getConnection() method available in DriverManager class. The established connection is then set to new connection object.

Connection cn=DriverManager.getConnetion ("jdbc:odbc:DSN", "administrator", "password");

Where DSN-Data Source Name

When the statement is executed the connection is set to the Connection object cn. For example to connect

The methods used for creating the statements are

- **Statement createStatement()**
  Returns a new Statement object.Used for general queries
- **PreparedStatement prepareStatement(String sql)**
  Returns a new PreparedStatement object.For a statement called multiple times with different values (precompiled to reduce parsing time)
- **CallableStatement prepareCall(String sql)**
  Returns a new CallableStatement object for stored procedures

Step 5: Executing the Query

After making connection to the database we are ready to execute the SQL statements. The various methods available in Statement object to execute the query are

- **ResultSet executeQuery(String)**
  Execute a SQL statement that returns a single ResultSet. After executing the SQL statements the requested data is stored in the ResultSet object.
- **int executeUpdate(String)**
  Execute a SQL INSERT, UPDATE or DELETE statement. Returns the number of rows changed.
- **boolean execute(String)**
  Execute a SQL statement that may return multiple results.

Step 6: Retrieving the result.

A ResultSet provides access to a table of data generated by executing a Statement.Only one ResultSet per Statement can be open at once.The table rows are retrieved in sequence. A ResultSet maintains a cursor pointing to its current row of data. The 'next' method moves the cursor to the next row.

Methods:

- **public boolean first()**
  Moves the cursor to the first row in this ResultSet object.
- **public boolean last()**
  Moves the cursor to the last row in this ResultSet object.
- **public boolean next()**
  Moves the cursor down one row from its current position.
  The methods used to retrieve the values from the current row
- **Type getType(int columnIndex)**
  Returns the given field as the given type. Fields indexed starting at 1 (not 0)
- **Type getType(String columnName)**
  Returns the data at the specified column

In the above methods the Type should be replaced by the valid datatypes such as Boolean, String, Int, Long, Float etc as getString(Column name), getInt(column index) etc.

Step 7: Closing the connection and statement

When the client request is completed we have to close the created objects of Connection and Statement using close() method.

```
st.close();
cn.close();
```

Q. What is JDBC Driver ?

JDBC drivers are client-side adapters(installed on the client machine, not on the server) that convert requests from java programs to a protocol that the DBMS can understand. There are commercial and free drivers available for most relational database severs.

JDBC Drivers Types:

JDBC driver implementations vary because of the wide variety of operating systems and hardware platforms in which Java operates. Sun has divided the implementation types into four categories, Types 1, 2, 3, and 4, which is explained below:

Type I - JDBC-ODBC Driver

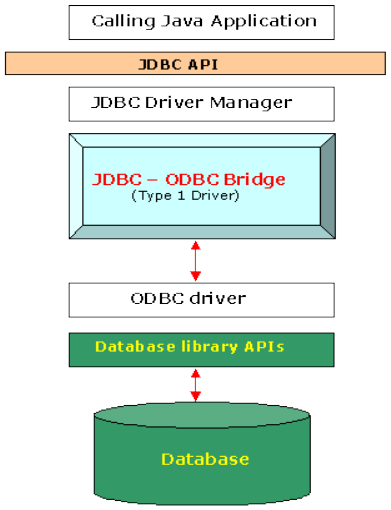


Fig: TYPE I Driver

Advantages:

- Almost any database, for which ODBC driver is installed, can be accessed.

Disadvantages:

- Performance overhead since the calls have to go through the jdbc Overhead bridge to the ODBC driver, then to the native db connectivity interface (thus may be slower than other types of drivers).
- The ODBC driver needs to be installed on the client machine.
- Not suitable for applets, because the ODBC driver needs to be installed on the client.

Type II - Native Interface/API Driver

- It uses the client-side libraries of the database.
- It mainly uses Java Native Interface to translatae calls to the local database API.
- It also provide java wrapper classes that are invoked using JDBC drivers.

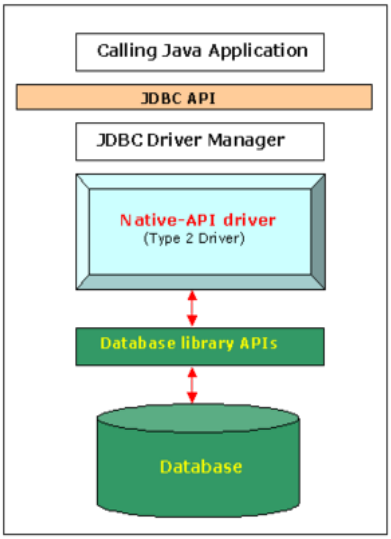


Fig: TYPE II Driver

Advantages:

- As there is no implementation of jdbc-odbc bridge, its considerably faster than a type 1 driver.

Disadvantages:

- The vendor client library needs to be installed on the client machine.
- Not all databases have a client side library
- This driver is platform dependent
- This driver supports all java applications except Applets

Type III - Network Protocol Driver

- Sends JDBC API calls to a middle-tier net server that translates the calls into the DBMS-specific network protocol. The translated calls are then sent to a particular DBMS.
- Follows a three tier communication approach.
- Can interface to multiple databases - Not vendor specific.
- The JDBC Client driver written in java, communicates with a middleware-net-server using a database independent protocol, and then this net server translates this request into database commands for that database.
- Thus the client driver to middleware communication is database independent.

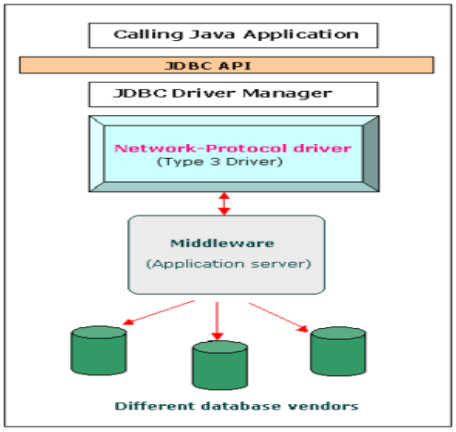


Fig: TYPE III Driver

Advantages:

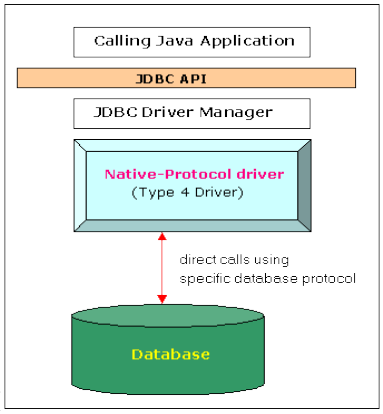
- Since the communication between client and the middleware server is database independent, there is no need for the database vendor library on the client. The client need not be changed for a new database.
- A single driver can handle any database, provided the middleware supports it.

Disadvantages:

- Requires database-specific coding to be done in the middle tier.
- The middleware layer added may result in additional latency, but is typically overcome by using better middleware services.

Type IV - Pure Java Driver/Database Protocol Driver

- They are also written in pure java and implement a database protocol such as Oracle's SQL*NET, to communicate directly with oracle.
- They do not require any native database libraries to be loaded on the client and can be deployed over the internet.



I

Advantages:

- Completely implemented in Java to achieve platform independence.
- These drivers don't translate the requests into an intermediary format (such as ODBC).

Disadvantages:

- Drivers are database dependent, as different database vendors use wildly different (and usually proprietary) network protocols.

Q. Write short notes on:

1.The Statement Objects:

It is Used this for the general-purpose access to database. It is useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters.

Creating Statement Object:

We need to create statement object using the Connection's createStatement( ) method, as in the following example:

```
Connection con = DriverManager. getConnection("jdbc:oracle:thin:@localhost:1521:XE","scott","tiger");
Statement st = con.createStatement();
ResultSet rs = st.executeQuery("select * from student");
```

Once you've created a Statement object, you can then use it to execute a SQL statement with one of its three execute methods.

- **boolean execute(String SQL)** : Returns a boolean value of true if a ResultSet object can be retrieved; otherwise, it returns false. Use this method to execute SQL DDL statements or when you need to use truly dynamic SQL.
- **int executeUpdate(String SQL)** : Returns the numbers of rows affected by the execution of the SQL statement. Use this method to execute SQL statements for which you expect to get a number of rows affected - for example, an INSERT, UPDATE, or DELETE statement.
- **ResultSet executeQuery(String SQL)** : Returns a ResultSet object. Use this method when you expect to get a result set, as you would with a SELECT statement.

2. The PreparedStatement Objects:

- PreparedStatement is a class in java.sql package and allows Java programmer to execute SQL queries by using JDBC package.
- We can get PreparedStatement object by calling connection.prepareStatement()method.
- SQL queries passed to this method goes to Database for pre-compilation if JDBC driver supports it. If it doesn't than pre-compilation occurs when you execute prepared queries.
- Prepared Statement queries are pre-compiled on database and there access plan will be reused to execute further queries which allows them to execute much quicker than normal queries generated by Statement object.

Creating PreparedStatement Object:

```
PreparedStatement pstmt = null;
try {
  String SQL = "Update Employees SET age = ? WHERE id = ?";
  pstmt = conn.prepareStatement(SQL);
  . . .
}
catch (SQLException e) {
  . . .
}
finally {
  . . .
}
```

- All parameters in JDBC are represented by the **?** symbol, which is known as the parameter marker. You must supply values for every parameter before executing the SQL statement.
- The **setXXX()** methods bind values to the parameters, where **XXX** represents the Java data type of the value you wish to bind to the input parameter. If you forget to supply the values, you will receive an SQLException.
- Each parameter marker is referred to by its ordinal position. The first marker represents position 1, the next position 2, and so forth. This method differs from that of Java array indices, which start at 0.
- All of the **Statement object's** methods for interacting with the database (a) execute(), (b) executeQuery(), and (c) executeUpdate() also work with the PreparedStatement object. However, the methods are modified to use SQL

4. At last `PreparedStatement` queries are more readable and secure than cluttered `string` concatenated queries.

3 The CallableStatement Objects:

Just as a Connection object creates the Statement and PreparedStatement objects, it also creates the CallableStatement object, which would be used to execute a call to a database stored procedure.

Creating CallableStatement Object:

Suppose, you need to execute the following Oracle stored procedure:

```
CREATE OR REPLACE PROCEDURE getEmpName
  (EMP_ID IN NUMBER, EMP_FIRST OUT VARCHAR) AS
BEGIN
  SELECT first INTO EMP_FIRST
  FROM Employees
  WHERE ID = EMP_ID;
END;
```

NOTE: Above stored procedure has been written for Oracle, but we are working with MySQL database so let us write same stored procedure for MySQL as follows to create it in EMP database:

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `EMP`.`getEmpName` $$
CREATE PROCEDURE `EMP`.`getEmpName`
  (IN EMP_ID INT, OUT EMP_FIRST VARCHAR(255))
BEGIN
  SELECT first INTO EMP_FIRST
  FROM Employees
  WHERE ID = EMP_ID;
END $$

DELIMITER ;
```

Three types of parameters exist: IN, OUT, and INOUT. The PreparedStatement object only uses the IN parameter. The CallableStatement object can use all three.

Here are the definitions of each:

Parameter	Description
IN	A parameter whose value is unknown when the SQL statement is created. You bind values to IN parameters with the setXXX() methods.
OUT	A parameter whose value is supplied by the SQL statement it returns. You retrieve values from theOUT parameters with the getXXX() methods.
INOUT	A parameter that provides both input and output values. You bind variables with the setXXX() methods and retrieve values with the getXXX() methods.

The following code snippet shows how to employ the **Connection.prepareCall()** method to instantiate a**CallableStatement** object based on the preceding stored procedure:

```
CallableStatement cstmt = null;
try {
  String SQL = "{call getEmpName (?, ?)}";
  cstmt = conn.prepareCall (SQL);
```

```
finally {
  . . .
}
```

The String variable SQL represents the stored procedure, with parameter placeholders.

Using CallableStatement objects is much like using PreparedStatement objects. You must bind values to all parameters before executing the statement, or you will receive an SQLException.

If you have IN parameters, just follow the same rules and techniques that apply to a PreparedStatement object; use the setXXX() method that corresponds to the Java data type you are binding.

When you use OUT and INOUT parameters you must employ an additional CallableStatement method, registerOutParameter(). The registerOutParameter() method binds the JDBC data type to the data type the stored procedure is expected to return.

Once you call your stored procedure, you retrieve the value from the OUT parameter with the appropriate getXXX() method. This method casts the retrieved value of SQL type to a Java data type.

Closing CallableStatement Obeject:

Just as you close other Statement object, for the same reason you should also close the CallableStatement object.

A simple call to the close() method will do the job. If you close the Connection object first it will close the CallableStatement object as well. However, you should always explicitly close the CallableStatement object to ensure proper cleanup.

```
CallableStatement cstmt = null;
try {
  String SQL = "{call getEmpName (?, ?)}";
  cstmt = conn.prepareCall (SQL);
  . . .
}
catch (SQLException e) {
  . . .
}
finally {
  cstmt.close();
}
```

Q. Write a note on ResultSet interface

The object of ResultSet maintains a cursor pointing to a particular row of data. Initially, cursor points to before the first row. By default, ResultSet object can be moved forward only and it is not updatable.

But we can make this object to move forward and backward direction by passing either TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE in createStatement(int,int) method as well as we can make this object as updatable by:

Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);

Commonly used methods of ResultSet interface

1) public boolean next():	is used to move the cursor to the one row next from the current position.

	object.
4) public boolean last():	is used to move the cursor to the last row in result set object.
5) public boolean absolute(int row):	is used to move the cursor to the specified row number in the ResultSet object.
6) public boolean relative(int row):	is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.
7) public int getInt(int columnIndex):	is used to return the data of specified column index of the current row as int.
8) public int getInt(String columnName):	is used to return the data of specified column name of the current row as int.
9) public String getString(int columnIndex):	is used to return the data of specified column index of the current row as String.
10) public String getString(String columnName):	is used to return the data of specified column name of the current row as String.

Example of Scrollable ResultSet

Let's see the simple example of ResultSet interface to retrieve the data of 3rd row.

```
import java.sql.*;
class FetchRecord
{
	public static void main(String args[])throws Exception
	{

	Class.forName("oracle.jdbc.driver.OracleDriver");
	Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle")
	;

	Statement stmt=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABL
	E);
	ResultSet rs=stmt.executeQuery("select * from emp765");

	//getting the record of 3rd row
	rs.absolute(3);
	System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));

	con.close();
	}
}
```

JDBC RowSet Example

JDBC RowSet is an interface of javax.sql.rowset interface. This interface is wrapper around a ResultSet object that makes possible to use resultSet as java beans object. It can be one bean that makes available for composing of an application.

An Example of Row Set Event listener is given below, To run this example at first create a database name student and create a table also named student

```
CREATE TABLE student (
RollNo int(9)  PRIMARY KEY NOT NULL,
Name tinytext NOT NULL,
Course varchar(25) NOT NULL,
Address text
 );
```

Then insert the value into it as

```
NSERT INTO student VALUES(1, 'Ram', 'B.Tech', 'Delhi') ;

NSERT INTO student VALUES(2, 'Syam', 'M.Tech', 'Mumbai') ;
```

JDBCRowSetExample.java

```
import java.sql.*;
import javax.sql.rowset.JdbcRowSet;
import com.sun.rowset.JdbcRowSetImpl;

public class JDBCRowSetExample {
      public static void main(String[] args) throws Exception {
            Connection connection = getMySqlConnection();
            System.out.println("Connection Done");
            Statement statement = connection.createStatement();
            JdbcRowSet jdbcRowSet;
            jdbcRowSet = new JdbcRowSetImpl(connection);
            jdbcRowSet.setType(ResultSet.TYPE_SCROLL_INSENSITIVE);
            String queryString = "SELECT * FROM student";
            jdbcRowSet.setCommand(queryString);
            jdbcRowSet.execute();
            jdbcRowSet.addRowSetListener(new ExampleListener());

            while (jdbcRowSet.next())
                        {
                  // Generating cursor Moved event
              System.out.println("Roll No- " + jdbcRowSet.getString(1));
              System.out.println("name- " + jdbcRowSet.getString(2));
            }
```

```
    public static Connection getMySqlConnection() throws Exception {
     Class.forName("com.mysql.jdbc.Driver");
     Connection connection =
            DriverManager.getConnection("jdbc:mysql://localhost:3306/student");
     return connection;
    }

}
class ExampleListener implements RowSetListener {

    @Override
    public void cursorMoved(RowSetEvent event) {
        // TODO Auto-generated method stub
        System.out.println("Cursor Moved Listener");
        System.out.println(event.toString());
    }

    @Override
    public void rowChanged(RowSetEvent event) {
        // TODO Auto-generated method stub
        System.out.println("Cursor Changed Listener");
        System.out.println(event.toString());
    }

    @Override
    public void rowSetChanged(RowSetEvent event) {
        // TODO Auto-generated method stub
        System.out.println("RowSet changed Listener");
        System.out.println(event.toString());
    }
}
```

JDBC Transactions:

A transaction is a set of actions to be carried out as a single, atomic action. Either all of the actions are carried out, or none of them are. JDBC Transaction let you control how and when a transaction should commit into database. JDBC transaction make sure SQL statements within a transaction block are all executed successful, if either one of the SQL statement within transaction block is failed, abort and rollback everything within the transaction block. If your JDBC Connection is in auto-commit mode, which it is by default, then every SQL statement is committed to the database upon its completion.

Transactions enable you to control if, and when, changes are applied to the database. It treats a single SQL statement or a group of SQL statements as one logical unit, and if any statement fails, the whole transaction fails.

To enable manual- transaction support instead of the auto-commit mode that the JDBC driver uses by default, use the

```
conn.setAutoCommit(false);
```

1. Commit & Rollback

Once you are done with your changes and you want to commit the changes then call commit() method on connection object as follows:

```
conn.commit( );
```

Otherwise, to roll back updates to the database made using the Connection named conn, use the following code:

```
conn.rollback( );
```

The following example illustrates the use of a commit and rollback object:

```
try{
   //Assume a valid connection object conn
   conn.setAutoCommit(false);
   Statement stmt = conn.createStatement();

   String SQL = "INSERT INTO Employees  " +
                "VALUES (106, 20, 'Rita', 'Tez')";
   stmt.executeUpdate(SQL);
   //Submit a malformed SQL statement that breaks
   String SQL = "INSERTED IN Employees  " +
                "VALUES (107, 22, 'Sita', 'Singh')";
   stmt.executeUpdate(SQL);
   // If there is no error.
   conn.commit();
}catch(SQLException se){
   // If there is any error.
   conn.rollback();
}
```

In this case none of the above INSERT statement would success and everything would be rolled back.

2. Savepoints:

The new JDBC 3.0 Savepoint interface gives you additional transactional control.

When you set a savepoint you define a logical rollback point within a transaction. If an error occurs past a savepoint, you can use the rollback method to undo either all the changes or only the changes made after the savepoint.

The Connection object has two new methods that help you manage savepoints:

- **setSavepoint(String savepointName):** defines a new savepoint. It also returns a Savepoint object.
- **releaseSavepoint(Savepoint savepointName):** deletes a savepoint.

Notice that it requires a Savepoint object as a parameter. This object is usually a savepoint generated by the setSavepoint() method.
There is one rollback (String savepointName) method which rolls back work to the specified savepoint.

The following example illustrates the use of a Savepoint object:

```
try{
   //Assume a valid connection object conn
   conn.setAutoCommit(false);
   Statement stmt = conn.createStatement();
```

```
   String SQL = "INSERTED IN Employees " +
                "VALUES (107, 22, 'Sita', 'Tez')";
   stmt.executeUpdate(SQL);
   // If there is no error, commit the changes.
   conn.commit();

}catch(SQLException se){
   // If there is any error.
   conn.rollback(savepoint1);
}
```

In this case none of the above INSERT statement would success and everything would be rolled back.