Java Networking

Java Networking is a concept of connecting two or more computing devices together so that we can share resources. Java socket programming provides facility to share data between different computing devices. Java Networking Terminology The widely used java networking terminologies are given below:

1) IP Address

IP address is a unique number assigned to a node of a network e.g. 192.168.0.1. It is composed of octets that range from 0 to 255. It is a logical address that can be changed.

2) Protocol

A protocol is a set of rules basically that is followed for communication. For example:

- TCP
- FTP
- Telnet

3) Port Number

The port number is used to uniquely identify different applications. It acts as a communication endpoint between applications.

The port number is associated with the IP address for communication between two applications.

4) MAC Address

MAC (Media Access Control) Address is a unique identifier of NIC (Network Interface Controller). A network node can have multiple NIC but each with unique MAC.

5) Connection-oriented and connection-less protocol

In connection-oriented protocol, acknowledgement is sent by the receiver. So it is reliable but slow. The example of connection-oriented protocol is TCP.

But, in connection-less protocol, acknowledgement is not sent by the receiver. So it is not reliable but fast. The example of connection-less protocol is UDP.

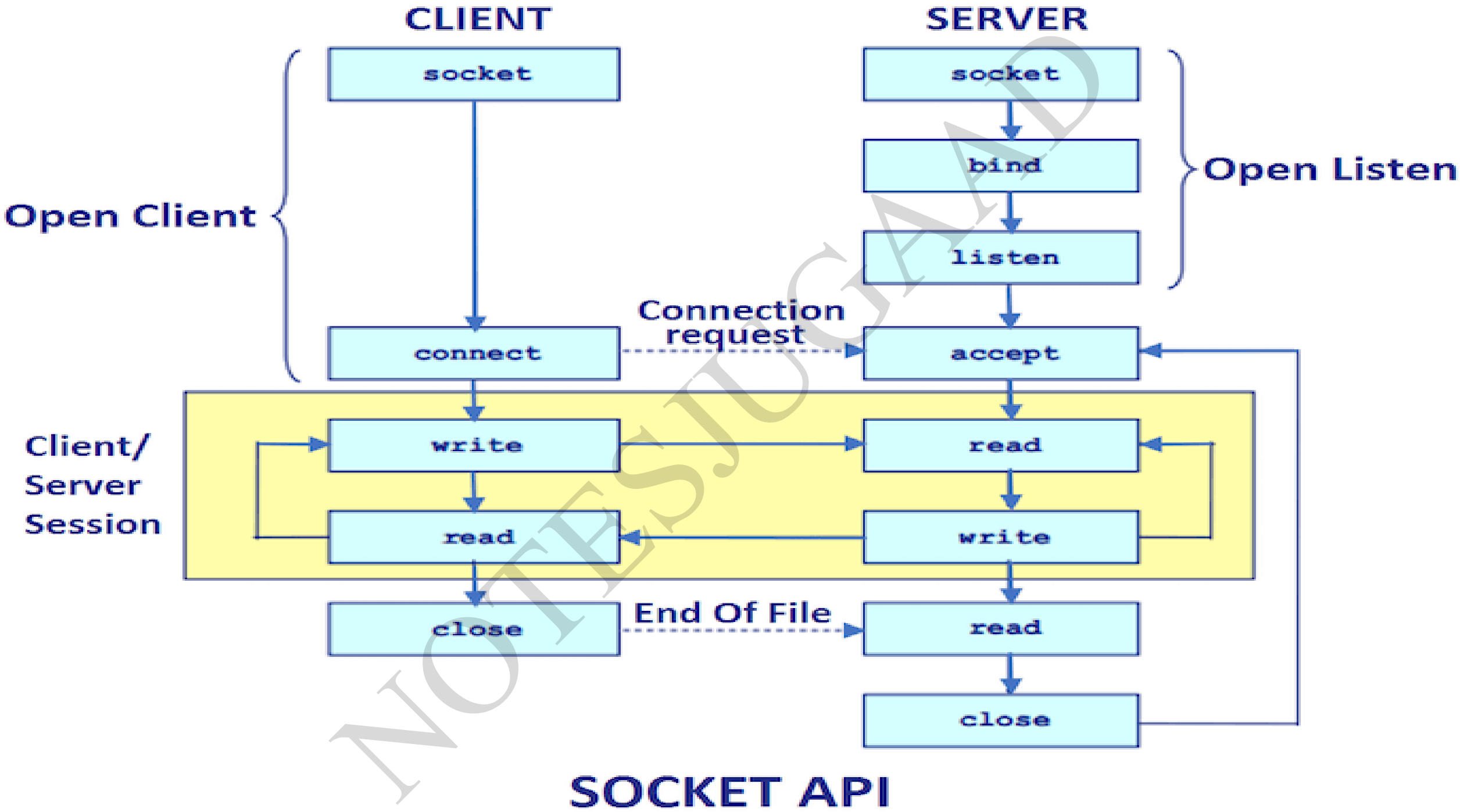
6) Socket

A socket is an endpoint between two way communication. Java Socket programming is used for communication between the applications running on different JRE. Java Socket programming can be connection-oriented or connection-less. Socket and Server Socket classes are used for connection-oriented socket programming and Datagram Socket and Datagram Packet classes are used for connection-less socket programming.

The client in socket programming must know two information:

1. IP Address of Server, and
2. Port number.

Here, we are going to make one-way client and server communication. In this application, client sends a message to the server, server reads the message and prints it. Here, two classes are being used: Socket and Server Socket. The Socket class is used to communicate client and server. Through this class, we can read and write message. The Server Socket class is used at server-side. The accept() method of Server Socket class blocks the console until the client is connected. After the successful connection of client, it returns the instance of Socket at server-side.



SOCKET API

- **Important methods**

Method	Description
1) public InputStream getInputStream()	returns the InputStream attached with this socket.
2) public OutputStream getOutputStream()	returns the OutputStream attached with this socket.
3) public synchronized void close()	closes this socket

Java Datagram Socket: -

class represents a connection-less socket for sending and receiving datagram packets. A datagram is basically an information but there is no guarantee of its content, arrival or arrival time.

Factory method: -

Is a creational design pattern which solves the problem of creating product objects without specifying their concrete classes. Factory Method defines a method, which should be used for creating objects instead of direct constructor call (new operator). Subclasses can override this method to change the class of objects that will be created.

The InetAddress class represents an IP address, both IPv4 and IPv6. Basically you create instances of this class to use with other classes such as Socket, Server Socket, Datagram Packet and Datagram Socket. In the simplest case, you can use this class to know the IP address from a hostname, and vice-versa.
The InetAddress class doesn’t have public constructors, so you create a new instance by using one of its factory methods:

- **getByName(String host)**: creates an InetAddress object based on the provided hostname.
- **getByAddress(byte[] addr)**: returns an InetAddress object from a byte array of the raw IP address.
- **getAllByName(String host)**: returns an array of InetAddress objects from the specified hostname, as a hostname can be associated with several IP addresses.
- **getLocalHost()**: returns the address of the localhost.

To get the IP address/hostname you can use a couple of methods below:

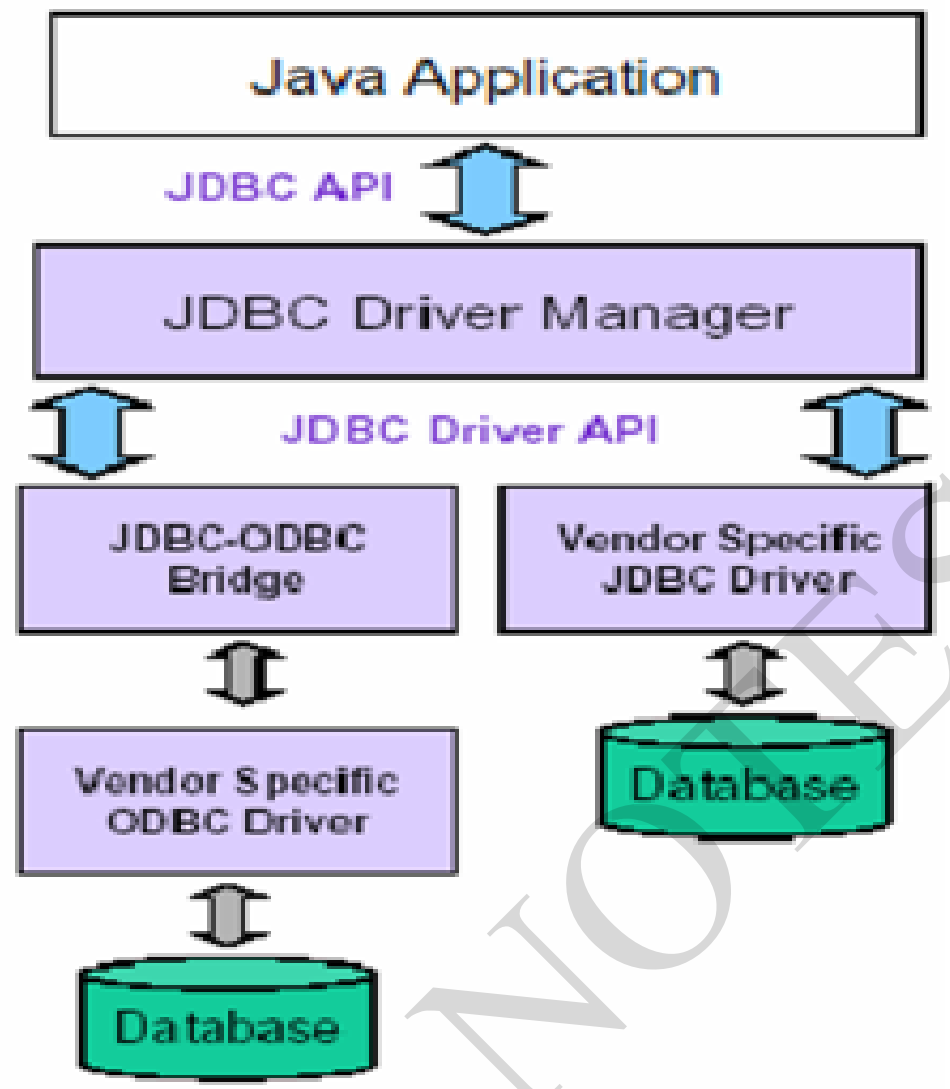
- **getHostAddress()**: returns the IP address in text.
- **getHostname()**: gets the hostname.

JDBC

What is JDBC?

- "An API that lets you access virtually any tabular data source from the Java programming language"
  - JDBC Data Access API – JDBC Technology Homepage
  - What's an API?
    - See J2SE documentation
  - What's a tabular data source?
- "... access virtually any data source, from relational databases to spreadsheets and flat files."
  - JDBC Documentation
- We'll focus on accessing Oracle databases

General Architecture



- What design pattern is implied in this architecture?
- What does it buy for us?
- Why is this architecture also multi-tiered?

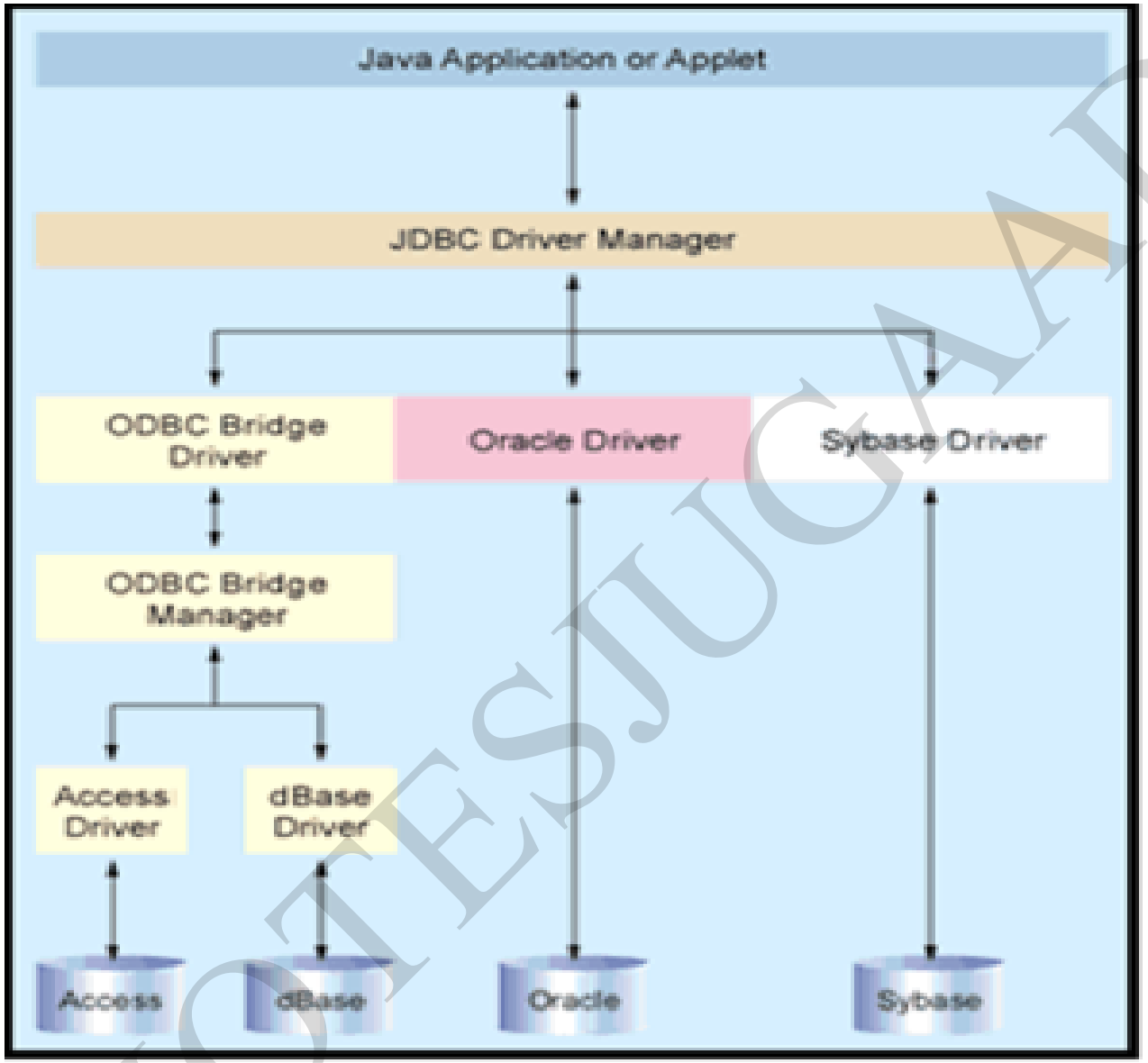


Figure 1. Anatomy of Data Access. The Driver Manager provides a consistent layer between your Java app and back-end database. JDBC works natively (such as with the Oracle driver in this example) or with any ODBC datasource.

Basic steps to use a database in Java

- 1.Establish a **connection**
- 2.Create JDBC **Statements**
- 3.Execute **SQL** Statements
- 4.GET **ResultSet**
- 5.**Close** connections

1. Establish a connection

- **import java.sql.*;**
- **Load the vendor specific driver**
  - Class.forName("oracle.jdbc.driver.OracleDriver");
    - What do you think this statement does, and how?
    - Dynamically loads a driver class, for Oracle database
- **Make the connection**
  - Connection con = DriverManager.getConnection("jdbc:oracle:thin:@oracle-prod:1521:OPROD", username, passwd);
    - What do you think this statement does?
    - Establishes connection to database by obtaining a *Connection* object

2. Create JDBC statement(s)

- Statement stmt = con.createStatement() ;
- Creates a Statement object for sending SQL statements to the database

Executing SQL Statements

- String createLehigh = "Create table Lehigh " +
  "(SSN Integer not null, Name VARCHAR(32), " +
  "Marks Integer)";
  stmt.**executeUpdate**(createLehigh);
  //What does this statement do?

- String insertLehigh = "Insert into Lehigh values“
  + "(123456789,abc,100)";
  stmt.**executeUpdate**(insertLehigh);

Get ResultSet

```
String queryLehigh = "select * from Lehigh";

ResultSet rs = Stmt.executeQuery(queryLehigh);
//What does this statement do?

while (rs.next()) {
    int ssn = rs.getInt("SSN");
    String name = rs.getString("NAME");
    int marks = rs.getInt("MARKS");
}
```

Close connection

- stmt.close();
- con.close();

Sample program

```
import java.sql.*;
class Test  {
   public static void main(String[] args)  {
      try {
         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); //dynamic loading of driver
         String filename = "c:/db1.mdb"; //Location of an Access database
         String database = "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=";
         database+= filename.trim() + ";DriverID=22;READONLY=true}"; //add on to end
         Connection con = DriverManager.getConnection( database ,"","");
         Statement s = con.createStatement();
         s.execute("create table TEST12345 ( firstcolumn integer )");
         s.execute("insert into TEST12345 values(1)");
         s.execute("select firstcolumn from TEST12345");
```

Sample program(cont)

```
        ResultSet rs = s.getResultSet();
        if (rs != null) // if rs == null, then there is no ResultSet to view
        while ( rs.next() ) // this will step through our data row-by-row
        {   /* the next line will get the first column in our current row's ResultSet
           as a String ( getString( columnNumber) ) and output it to the screen */
           System.out.println("Data from column_name: " + rs.getString(1) );
        }
        s.close(); // close Statement to let the database know we're done with it
        con.close(); //close connection
      }
     catch (Exception err) { System.out.println("ERROR: " + err);  }
   }
}
```

- JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:
- JDBC-ODBC bridge driver
- Native-API driver (partially java driver)
- Network Protocol driver (fully java driver)
- Thin driver (fully java driver)

JDBC references

- JDBC Data Access API – JDBC Technology Homepage
  - http://java.sun.com/products/jdbc/index.html
- JDBC Database Access – The Java Tutorial
  - http://java.sun.com/docs/books/tutorial/jdbc/index.html
- JDBC Documentation
  - http://java.sun.com/j2se/1.4.2/docs/guide/jdbc/index.html
- java.sql package
  - http://java.sun.com/j2se/1.4.2/docs/api/java/sql/package-summary.html
- JDBC Technology Guide: Getting Started
  - http://java.sun.com/j2se/1.4.2/docs/guide/jdbc/getstart/GettingStartedTOC.fm.html
- JDBC API Tutorial and Reference (book)
  - http://java.sun.com/docs/books/jdbc/

JDBC

- JDBC Data Access API – JDBC Technology Homepage
  - http://java.sun.com/products/jdbc/index.html
- JDBC Database Access – The Java Tutorial
  - http://java.sun.com/docs/books/tutorial/jdbc/index.html
- JDBC Documentation
  - http://java.sun.com/j2se/1.4.2/docs/guide/jdbc/index.html
- java.sql package
  - http://java.sun.com/j2se/1.4.2/docs/api/java/sql/package-summary.html
- JDBC Technology Guide: Getting Started
  - http://java.sun.com/j2se/1.4.2/docs/guide/jdbc/getstart/GettingStartedTOC.fm.html
- JDBC API Tutorial and Reference (book)
  - http://java.sun.com/docs/books/jdbc/

Servlets

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

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet.

- The servlet is initialized by calling the **init()** method.
- The servlet calls **service()** method to process a client's request.
- The servlet is terminated by calling the **destroy()** method.
- Finally, servlet is garbage collected by the garbage collector of the JVM.

Life cycle methods of a Servlet

Called only once

service() handles multiple client requests and send sends response

Called only once

Interfaces in javax.servlet package

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

1. Servlet
2. ServletRequest
3. ServletResponse
4. RequestDispatcher
5. ServletConfig
6. ServletContext
7. SingleThreadModel
8. Filter
9. FilterConfig
10. FilterChain
11. ServletRequestListener
12. ServletRequestAttributeListener
13. ServletContextListener
14. ServletContextAttributeListen

Classes in javax.servlet package

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

1. GenericServlet
2. ServletInputStream
3. ServletOutputStream
4. ServletRequestWrapper
5. ServletResponseWrapper
6. ServletRequestEvent
7. ServletContextEvent
8. ServletRequestAttributeEvent
9. ServletContextAttributeEvent
10. ServletException
11. UnavailableException