UNIT-3

Java Networking

Java Networking is a concept of connecting two or more computing device together so that we can share resources. Java socket programming provides facility to share data between different computing devices

Advantages of Java Networking

1) Sharing Resources
2) Centralize software management

JAVA Networking Terminology

1) IP address
2) Protocol
3) Port Number
4) MAC Address
5) Connection - oriented and connection less Protocol
6) Socket

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

Protocol: A protocol is an ~~unique~~ assigned ~~to a~~ set of rules basically that is followed for communication. for example.

TCP
FTP
Telnet
SMTP
POP

Port Number: The port number is used to uniquely identify different applications. It acts as a communication endpoint between application. The port number is associated with the IP address for communication between two application

MAC Address: (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 Address

Connection-oriented and connection-less Protocol

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

But connection less protocol, acknowledgement is not sent by the receiver so it is not reliable but fast. eg UDP

Socket:

A socket is an endpoint between two way communications. It is bidirectional.

Types of socket

In Java, sockets are primarily used for network communication between computers or processes. Java provides several types of sockets, each serving a specific purpose and built on different protocols or communication mechanisms. Here are the main types of sockets available in Java:

1. TCP Sockets

- **Socket** (Client): Used to create a connection-oriented client socket.
- **ServerSocket** (Server): Used on the server side to listen for incoming connections from clients.
- **Usage**: TCP sockets provide reliable, two-way communication channels, where data arrives in order and without loss.
- **Example**: Commonly used for applications requiring a stable connection, such as chat servers, file transfer, or web servers.

2. UDP Sockets

- **DatagramSocket**: Used for connectionless communication using UDP (User Datagram Protocol).
- **Usage**: UDP sockets send and receive data packets independently, without guaranteeing delivery, order, or duplication.
- **Example**: Useful for real-time applications where speed is critical, like online gaming, video streaming, or broadcasting.

3. Raw Sockets

- **Java NIO Channels (SocketChannel, ServerSocketChannel)**: These aren't exactly raw sockets, but Java NIO channels provide non-blocking I/O operations and are lower-level than the traditional Socket class.
- **Usage**: Used for high-performance and scalable applications that require multiplexing, like web servers, file servers, and large-scale real-time applications.
- **Example**: These channels allow you to use selectors for managing multiple channels, avoiding the blocking behavior of traditional sockets.

JAVA Socket Programming

Java Socket Programming is used for communi-cation between the applications running on different JRE.

JAVA Socket Programming can be connection-orie-ted or connection less.

Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming

The client in socket programming must know two information:

1) IP Address of server
2) Port number

Socket Class

A socket is simply an endPoint for commun-ication between machines. The Socket class can be used to create socket.

Important Methods

1) Public InputStream getInputStream() :- It Return the InputStream attached with this Socket
2) Public OutputStream getOutput_Stream() :- Return Output stream attached with this socket
3) Public synchronized void close() : Close this socket

Server Socket class

The server socket class can be used to create a server socket. This object is used to establish communication with the clients.

Important Method

1) Public Socket accept() :- return the socket and establish a connection between server & client.

2) Public synchronized void close() :- closes the server socket.

Example

MY SERVER

```
import java.io.*;
import java.net.*;
Public class MyServer
{
public static void main(String[] args)
  {
try
 {
   ServerSocket ss = new ServerSocket(6666);
   Socket s = ss.accept(); // Create connection Establishes
   DataInputStream dis = new DataInputStream(s.getInput
                                             Stream());

String str = (String) dis.readUTF();
System.out.println("message=" + str);
      ss.close();

}
```

```
catch (Exception e)
{
 System.out.println(e);
}}}
```

Javac file name

Java class name.

MY CLIENT

```
import java.io.*;
import java.net.*;
public class MyClient
{
public static void main (String[] args)
 {
try
  {
   Socket s = new Socket("localhost", 6666);
DataOutputStream da = new DataOutputStream(s.getOutputStream());
   da.writeUTF ("Hello Server");
     da.flush();
     da.close();
     s.close();
  }
   catch (Exception e)
  {
   System.out.println(e);
}}}
```

DatagramSocket and DatagramPacket

Java DatagramSocket and DatagramPacket classes are used for connection-less socket programming using the UDP instead of TCP.

Datagram

Datagram are collection of information sent from one device to another device via established network. When the datagram is sent to the targeted device, there is no assurance that it will be reached to target device safely and completely. It may get damaged or lost in between. Likewise, the receiving device also never know if datagram received is damaged or not. The UDP protocol is used to implements the datagrams in java

DatagramSocket class

Java DatagramSocket class represent a connection-less socket for sending and recieving datagram packets. It is mechanism used to transmitting datagram packets over network

A datagram is basically an information but there is no guarantee of its content, arrival time.

Commonly used Constructors of DatagramSocket class

1) DatagramSocket() throws SocketException: It creats a datagram socket and binds it with the available

... is an unique assigned to a Port number on the localhost machine.

2) DatagramSocket(int Port) throws SocketException: It creats a datagram socket & binds it with the given Port Number.

3) DatagramSocket(int Port, InetAddress address) throws SocketException: It create a datagram socket & bind it with the specified Port number & host address

Java DatagramPacket class

Java DatagramPacket is a message that can be sent or received. It is a data container. If you send multiple packet, it may arrive in any order. Additionally, Packet delivery is not guaranteed.

Commonly Used constructors

1) DatagramPacket(byte[] barr, int length) :- It Create a datagram packet. This constructor is used to receive the Packets.

2) DatagramPacket(byte[] barr, int length, InetAddress address int port) : It creates a datagram Packet. This constructor is used to send the Packets.

Example of sending Datagram Packet by Datagram Socket

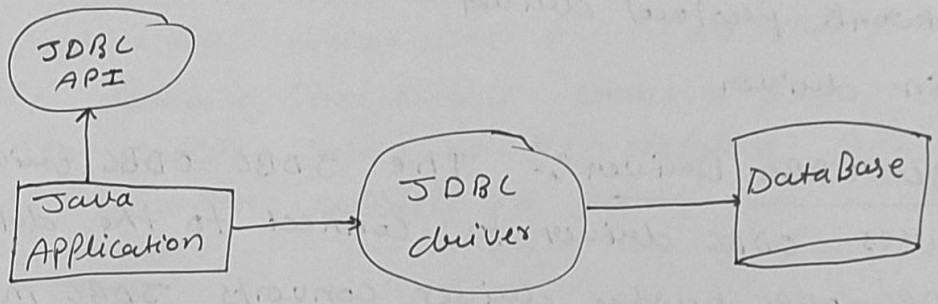
```
// Sender

import java.net.*;
public class DSender
{
public static void main(String[] args) throws
                                        Exception

{
  DatagramSocket ds = new DatagramSocket();
String str = "welcome java";
InetAddress ip = InetAddress.getByName("127.0.0.1");
DatagramPacket dp = new DatagramPacket(str.getByte(),
                    str.length(), ip, 3000);

  ds.send(dp);
  ds.close();

}}
```

Example of Receiving DatagramPacket by DatagramSocket

```
// Receiver

import java.net.*;
public class DReciver
{
public static void main(String[] args) throws Exception
{
DatagramSocket ds = new DatagramSocket(3000);
  byte[] buf = new byte[1024];
DatagramPacket dp= new DatagramPacket(buf, 1024);
   ds.recieve(dp);
  String str = new String(dp.getData(), 0, dp.getLength());
   System.out.println(str);
        ds.close();
 }
}
```

Java JDBC

JDBC stands for java Database connectivity. JDBC is Java API to connect and execute the query with the database. It is a part of JAVASE (Java Standard Edition). JDBC API user JDBC driver to connect with the database



Why should use JDBC

Before JDBC, ODBC API was the database API to connect and execute the query with database. But ODBC API uses ODBC driver which is written in C language (i.e Platform dependent and unsecured) That is why java has defined it own API (JDBC API) that uses JDBC driver (written in java language)

What is API

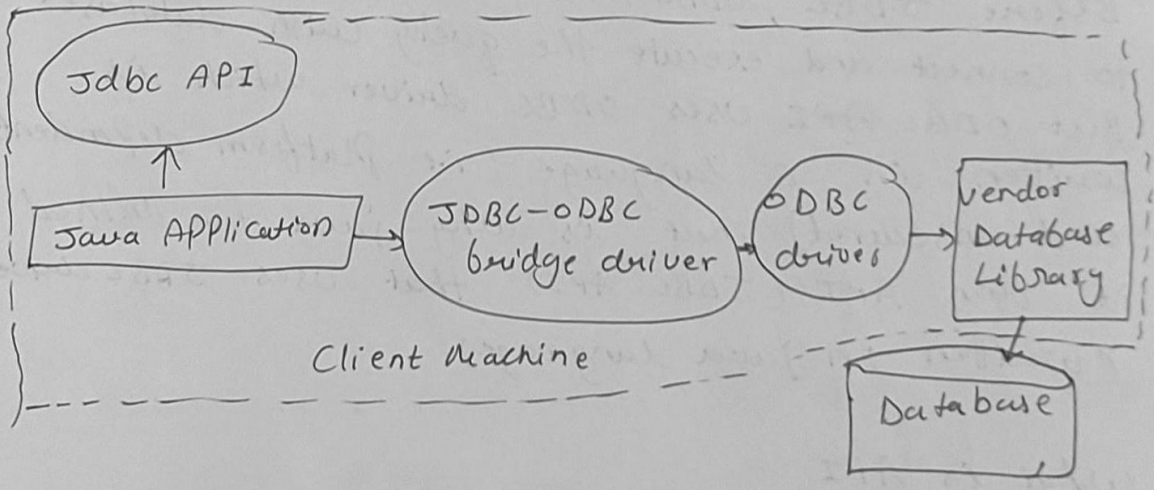
Application Programming Interface is document that contain a description of all the features of product or software. It represents classes and interface that software program can follow to communication with each other. An API can be created for application, libraries, operating system etc

JDBC Driver

JDBC driver is a software component that enables java application to interact with the database. There are 4 types of JDBC driver.

1) JDBC-ODBC driver
2) Native-API driver
3) Network Protocol driver
4) Thin driver

1) **JDBC-ODBC Driver** :- The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into ODBC function calls. This is now discouraged because of thin driver



Advantages:

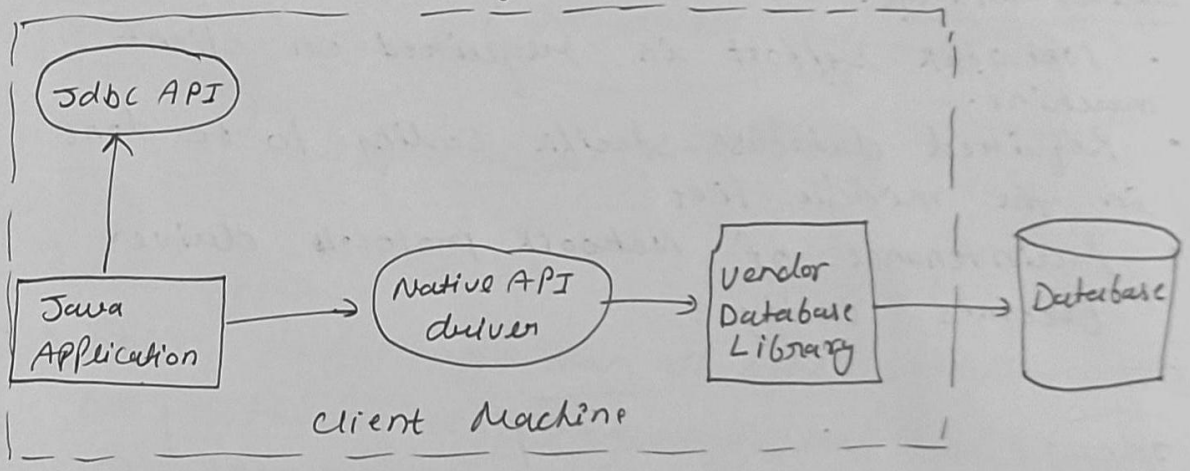
- Easy to use
- Can be easily connected to any database.

Disadvantage

- Performance degraded because JDBC method call is converted into ODBC function calls.
- The ODBC driver needs to be installed on the client machine

2) Native API Driver (Partially Java driver

Native API driver uses the client-side libraries of the database. The driver converts JDBC method call into native calls of the database API. It is not written entirely in Java.



Advantage • Performance upgraded than JDB-ODBC bridge driver.

Disadvantage:

- The Native driver needs to be installed on the each client machine
- The vendor client library needs to be installed on client machine

3) Network Protocol driver

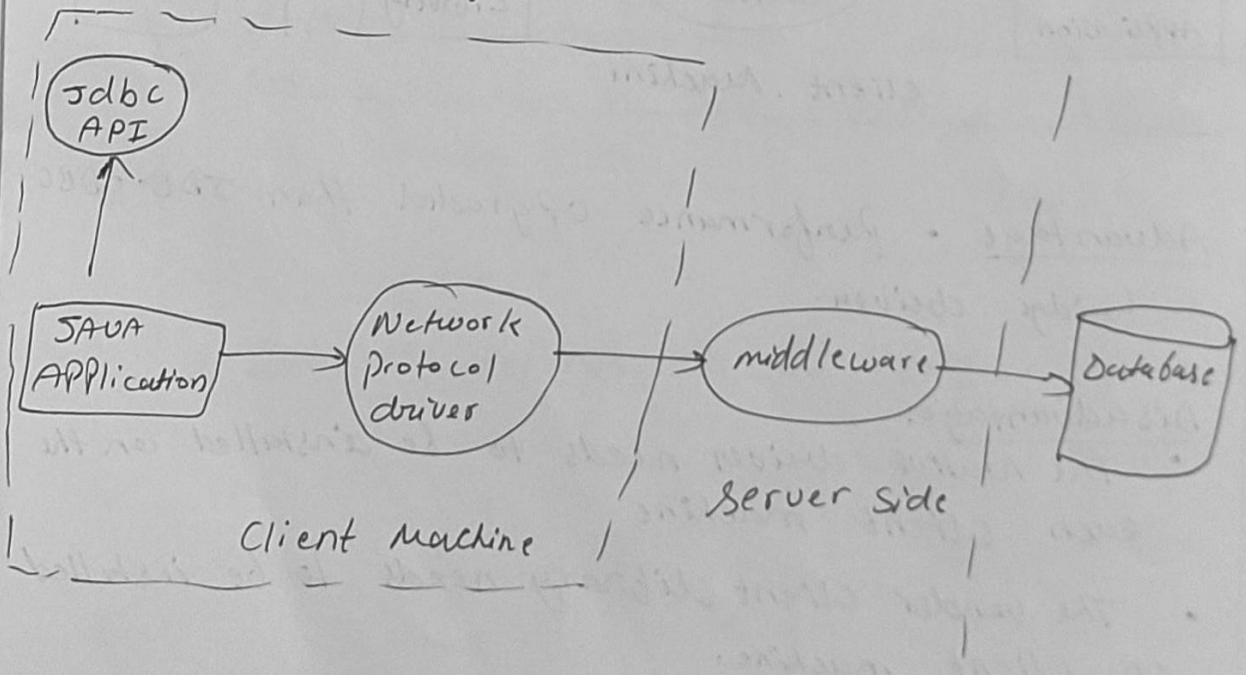
The network protocol driver uses middleware (application server) that converts JDBC call directly or indirectly in to vendor-specific database protocol. It is fully written in java.

Advantage

- No client side library is required because of application server that can perform many task like auditing, load balancing, logging etc.

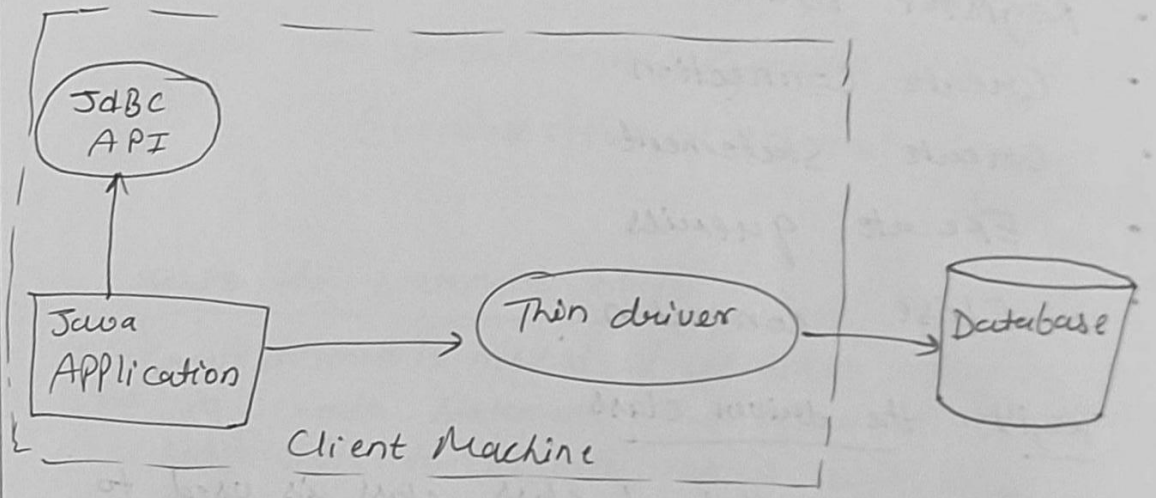
Disadvantage:

- Network support is required on client machine.
- Required database-specific coding to be done in the middle tier
- Maintenance of Network protocol driver becomes costly.



4) Thin driver

The thin driver converts JDBC calls directly into vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.



Advantage:

- Better performance than all other ~~drivers~~ driver.
- No software is required at client side or server side.

Disadvantage:

- Driver depends on the Database.

5 Steps to connect to the Database in java.

There are five steps to connect any java application with the database using JDBC.

- Register Driver class
- Create connection
- Create Statement
- Execute queries
- Close connection

1) Register the driver class

The forName() method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

Syntax of forName() method

```
public static void forName(String className) throws
                                    ClassNotFoundException
```

Example :- `Class.forName("oracle.jdbc.driver.OracleDriver");`

2) Create the connection object

The getConnection() method of DriverManager class is used to establish connection with the database.

Syntax of getConnection() method

1) Public static Connection getConnection(String url) throws SQLException

2) Public static Connection getConnection(String url, String name, String password) throws SQLException

```
Connection con = DriverManager.getConnection("jdbc:oracle:thin:
                 @localhost:1521:xe", "system", "password");
```

3) Create the statement object

The CreateStatement() method of Connection interface is used to create statement. The object of Statement is responsible to execute queries with the database

Syntax

```
Public Statement createStatement() throws SQLException
```

Example

```
Statement stmt = con.createStatement();
```

4) Execute the query

The executeQuery() method of statement interface is used to execute queries to the database. This methods returns the object of ResultSet that can be used to get all the records of a table.

Syntax:

executeQuery() method

Public ResultSet executeQuery(String sql) throws SQL-Exception

Example

```
ResultSet rs= stmt.executeQuery("select * from emp");
while(rs.next())
{
System.out.Println(rs.getInt(1)+" "+
                   getString(2));
}
```

5) Close the connection object

By closing connection object statement and ResultSet will be closed automatically. The close() method of connection interface is used to close the connection

syntax of close() method

Public void close() throws SQLException

Example to close connection

```
Con.close();
```

JDBC (Java Database Connectivity) and **ODBC (Open Database Connectivity)**. These both are API standards used for connecting applications to databases. ODBC can be used by various programming languages as it is a general API standard. But JDBC is Java-specific and it provides a Java-based interface for database access. ODBC drivers are not Java-centric which means it is designed to work with different languages through a common interface. JDBC drivers are Java-centric.

ODBC vs JDBC

In the below table, we will discuss the major difference between ODBC and JDBC:

ODBC	JDBC
1. ODBC Stands for Open Database Connectivity.	1. JDBC Stands for Java database connectivity.
2. Introduced by Microsoft in 1992.	2. Introduced by SUN Micro Systems in 1997.
3. We can use ODBC for any language like C, C++, Java etc.	3. We can use JDBC only for Java languages.
4. We can choose ODBC only Windows platform.	4. We can use JDBC on any platform.
5. Mostly ODBC Driver is developed in native languages like C, and C++.	5. JDBC Stands for Java database connectivity.
6. For Java applications it is not recommended to use ODBC because performance will be down due to internal conversion and applications will become platform-dependent.	6. For Java applications it is highly recommended to use JDBC because there are no performance & platform dependent problems.
7. ODBC is procedural.	7. JDBC is object-oriented.

Insert Data in to DataBase

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class Insert
{
public static void main(String[] args) throws Exception
{
    String name1 = "amit";
    String email1 = "amit@gmail.com";
    String city1 = "Pune";

Class.forName("com.mysql.cj.jdbc.Driver");

Comection con = DriverManager.getConnection("jdbc:mysql://
                localhost:3306/jdbc_bd", "root", "root");

PreparedStatement ps = con.prepareStatement("insert in_to
                register value(?,?,?)"); // positional
                                         parameters
  ps.setString(1, name1);
  ps.setString(2, email1);
  ps.setString(3, city1);
  int i = ps.executeUpdate();
  if(i>0)
  {
    System.out.println("Success");
  }
  else
  {
    System.out.println("Invalid details");
  }
}}
```

Update a Data in DataBase

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class Update
{
public static void main(String[] args) throws Exception
{
    String city1 = "Pune";
    String email1 = "Kamal@gmail.com";

Class.forName("com.mysql.cj.jdbc.Driver");

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

PreparedStatement ps = con.prepareStatement("Update
                register set city=? where email=?");

    ps.setString(1, city1);
    ps.setString(2, email1);

    int count = ps.executeUpdate();

      if(count=ps.
        if (count > 0)
{ System.out.println("Update Successfull");
}
else
  {System.out.println("error");
}
}}
```

Delete a data in DataBase

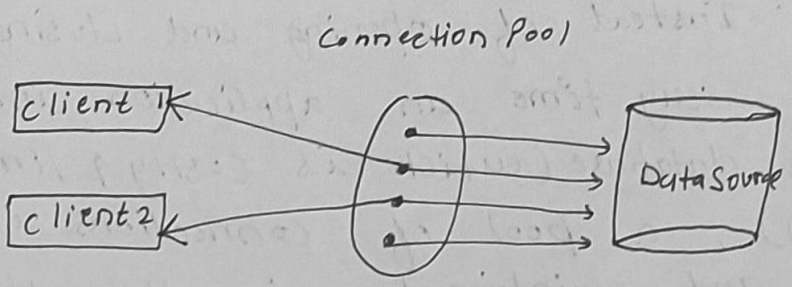

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

class Delete
{
public static void main(String[] args)
{
  String email1 = "kamal@gmail.com";
  Class.forName("com.mysql.cj.jdbc.Driver");
  Connection con = DriverManager.getConnection("jdbc:mysql:
                   //localhost:3306/jdbc_rj", "root",
                   "root");
  PreparedStatement ps= con.prepareStatement("delete
                        from register where email=?");
     ps.setString(1, email1);
       int coont = ps.executeUpdate();
      if(coont>0)
     {
      System.out.println("delete success");
     }
     else
       {  System.out.println("Delete failed");
       }
}}
```

Example to connect java Application with Database. 23

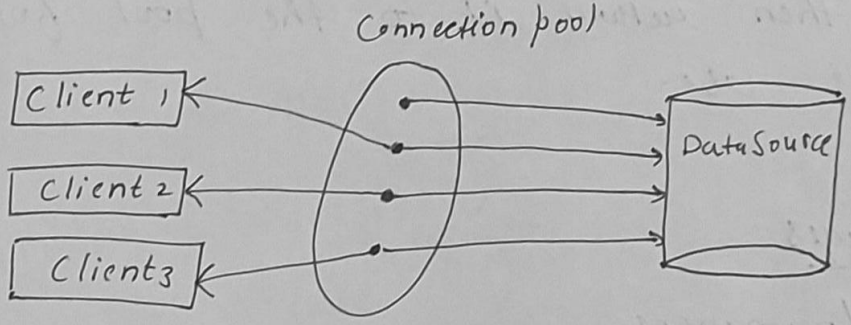
```
import java.sql.*;
Class Conn
{
Public Static void main (String [] args)
{
try
 {
   // Step 1  load the deriver class
    Class.forName("oracle.jdbc.deriver.OracleDeriver");
   // Step2  Create a connection object
   Connection con = DeriverManager.getConnection
              ("jdbc:oracle:thin:@localhost:1521:
               xe ", "system", "oracle");
  //Step-3  Create the Statement object
     Statement stmt = con.createStatement();

//Step 4 execute query
   ResultSet rs = stmt.executeQuery("select *from emp");
     while (rs.next())
     System.out.println(rs.getInt(1)+" "+rs.getString(2)
                 +" "+ rs.getstring(3));
//Step 5  Close connection object
     con.close();
}
```

```
catch(Exception e)
 {
   System.out.println(e);
 }
}}
```

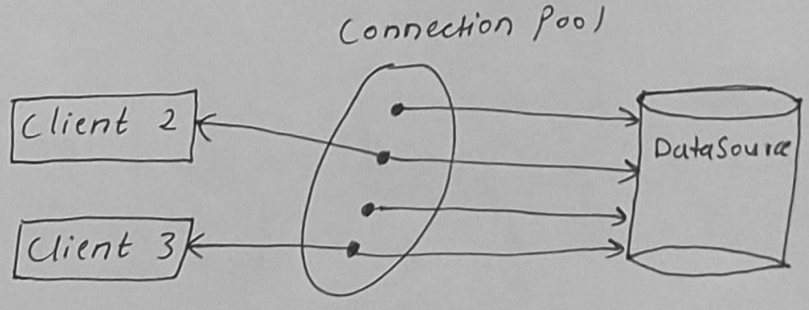
Connection Pooling



connection pool with 2 connected clients

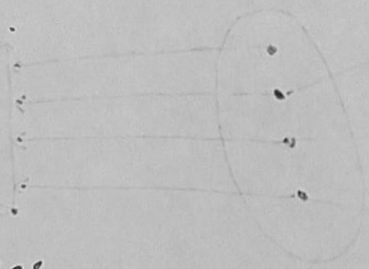


A new client #3 is assigned a free connection



Client #1 is disconnected, and free a connection for use.

Connection Pooling is a technique in JDBC that manages a pool of ~~resu~~ reusable database connections. Instead of opening and closing a connection every time an application interact with the database (which is costly & time consuming), a pool of connections is created and maintained. When the application needs a database connection, it can borrow one from the pool, use it, and then return it to the pool for future use.



Advantages

1) Performances
2) Scalability
3) Resource Management