Chapter :- 7

Java Network programming and socket programming

URL class

- Class URL represents a Uniform Resource Locator, a pointer to a 'resource' on the World Wide Web.
- A resource can be as simple as a file or a directory or it can be refernce to more complicated object such as query to a database or a search engine.
- A url is divided into many sections :-

eg :-

https://www.example.com:8000/search?q=123#section-id

Protocol, Hostname, port, File, query, Refrence.

- Java has different functions (Methods) for URL class. they are :-

i) public string getQuery() :- It returns the query of the URL.

ii) public String getRef() :- It returns the refrence of the URL object. usually, it is a part of # in the url.

iii) public int getPort() :- It returns the port number of the URL

iv) public int getDefaultPort() :- It returns the defult port used.

v) public string toString() :- It returns the String representation of the URL

vi) public URI toURI() :- It returns URI equivalent of the URL.

vii) public string getProtocol() :- It returns the protocol of URL

viii) public string getHost() :- It returns the Hostname of URL.

ix) public string getFile() :- It returns the Filename of the URL

x) public String getAuthority() :- It returns the authority part of the URL or null if it is empty.

eg :- URL

```
import java.net.*;

Class URLDemo{
public static void main(string[] args){
try{

        URL url = new URL("https://www.youtube.com/
results?search_query=redneck");

System.out.println("protocol:" + url.getProtocol());

System.out.println("Hostname:" + url.getHost());

System.out.println("port number:" + url.getport());

System.out.println("content:" + url.getContent());

System.out.println("query:" + url.getquery());

System.out.println("Refrence:" + url.getRef());

System.out.println("Filename:" url.getFile());
```

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

outputs :-

protocol : https
Hostname : www.youtube.com
port number : -1
content : sun.net.www.protocol.http.HttpURLConnection
$HttpInputStream@1b73begf.
query :- search_query = redneck

Reference : null
Filename : /results?search_query = redneck.

URLconnection class

- The java URLconnection class represents a communication link between the URL and the application.
- The class URLconnection can be used to read and write data to the specified resource referred by the URL.
- The openconnection() Method of URL class returns the object of URLconnection class.
- SYNTAX :-
  Public URLconnection openconnection throws IoException{}

vi) public URI toURI() :- It returns URI equivalent of the URL.

vii) public string getProtocol() :- It returns the protocol of URL

viii) public string getHost() :- It returns the Hostname of URL.

ix) public string getFile() :- It returns the Filename of the URL.

x) public String getAuthority() :- It returns the authority part of the URL or null if it is empty.

eg :- URL

```
import java.net.*;

Class URLDemo{
public static void main(string[] args){
 try{

    URL url = new URL("https://www.youtube.com/results?search_query=redneck");

System.out.println("protocol :" + url.getprotocol());

System.out.println("Hostname :" + url.getHost());

System.out.println("port number :" + url.getport());

system.out.println("content :" + url.getlontent());

System.out.println("query :" + url.getquery(););

System.out.println("Refrence :" + url.getRef());

System.out.println("Filename :" url.getFile());
```

Reads the content of URL

eg:-

```
import java.net.*;
import java.io.*;

Public class URLConnectionDemo {

Public static void Main ( string [] args) {
 try {

      URL url= new URL ("https://www.google.com/");

URLconnection urlcon = url.openConnection();

InputStream input= urlcon.getInputstream();

int i;
While ((i= input.read()) != -1) {

   System.out.println ((char) i) ;
  }

}catch ( Exception e ){

   system.out.println( e.getMessage());
  }
}
}
```

- This program displays the source code of web page.

HttpResponse code

```
import java.io.*;
import java.net.*;

public class Httpresponsecode {

public static void main(String[] args) {
try{

  URL url = new URL("http://www.google.com");

HttpURLConnection urlconxn = (HttpURLConnection) url.
        openConnection();

int code = urlconxn.getResponseCode();

 if (code == HttpURLConnection.HTTP_OK){

  InputStream stream = urlconxn.getInputstream();
   int i;
while(i = stream.read()) != -1)
  {
    System.out.println((char) i);
  }
}
else{
    system.out.println("server can't print");
  }
}catch(Exception e){
   System.out.println(e.getMessage());
}}}
```

Header connection content.

```
import java.io.*;
import java.net.*;

Public class HeaderConnectionContent {

 Public static void main(String[] args) {
  try {

  URL url = new URL("http://www.google.com");

HttpURLConnection urlconxn = (HttpURLConnection) url.openConnecti
on();
        int i;
  for(i=1; i<=8; i++)
    {
      System.out.println(urlconxn.getHeaderFieldKey(i) +
                    "=" + urlconxn.getHeaderField(i));
    }

} catch (Exception e) {

     System.out.println(e.getMessage());
    }
  }
}
```

InetAddress class (Stream communication)

- Java InetAddress class Represents Ip address.
- InetAddress class provides Methods to get the Ip address of any host name.
some of the Methods are :-

i) public string getHostName() :- Returns hostname of the address.

ii) public string getHostAddress() :- Returns Ip address in string format.

iii) public static InetAddress getLocalHost() throws UnknownHostException :- Returns the instance of the local host.
and Throws UnknownHostException : if the local host name could not be resolved into an instance.

iv) public static InetAddress getByName (string host) throws UnknownHostException :- Returns an InetAddress object based on the host name provided.

eg:-

```
import java.net.*;

class Inet{
public static void main(String[] args){
try{
InetAddress address = InetAddress.getByName(
"www.youtube.com");
```

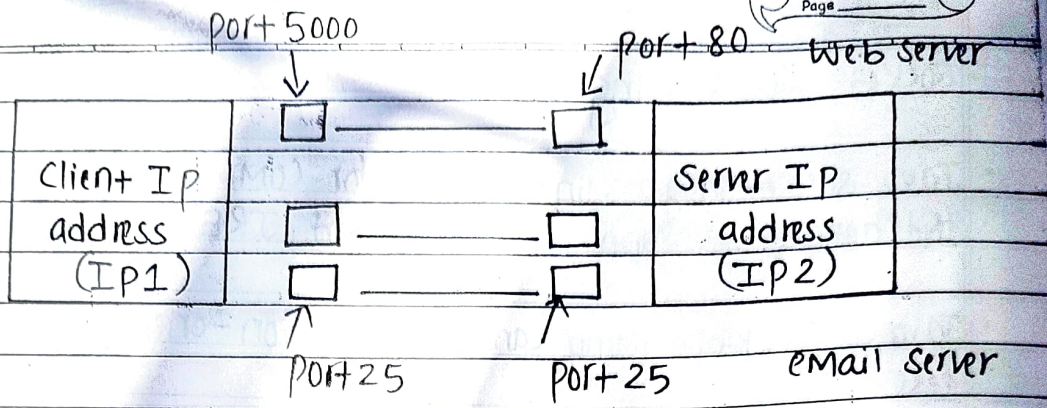
```
System.out.println(address);

InetAddress add = InetAddress.getByName("172.
217.166.174");

System.out.println(add.getHostName());

InetAddress me = InetAddress.getLocalHost();

System.out.println("My address is" + me.getHost
Address());

System.out.println("My localhost name" + me.get
HostName());

}
Catch(unKnownHostException e)
{
    System.out.println(e.getMessage());
}

}
}
```

Socket Programming

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

PORT VS SOCKET.

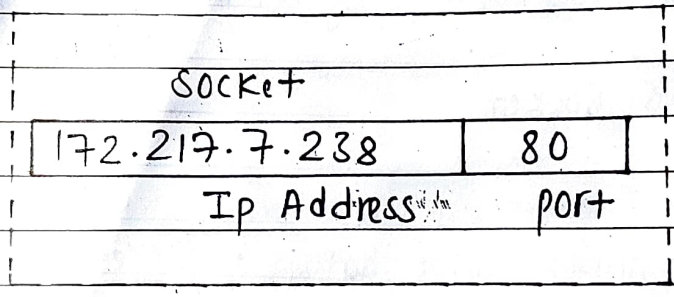
Port	Socket
• Port is a numeric value that is assigned to an application to an endpoint of communication.	• Sockets is an internal endpoint for sending and receiving data within a node on a computer Network.
• It helps to identify a specific application or a process.	• It works as the interface to send and receive data through specific port.
• Ports operate at the Transport layer of the OSI layer.	• Sockets are a means of plugging the application layer in.



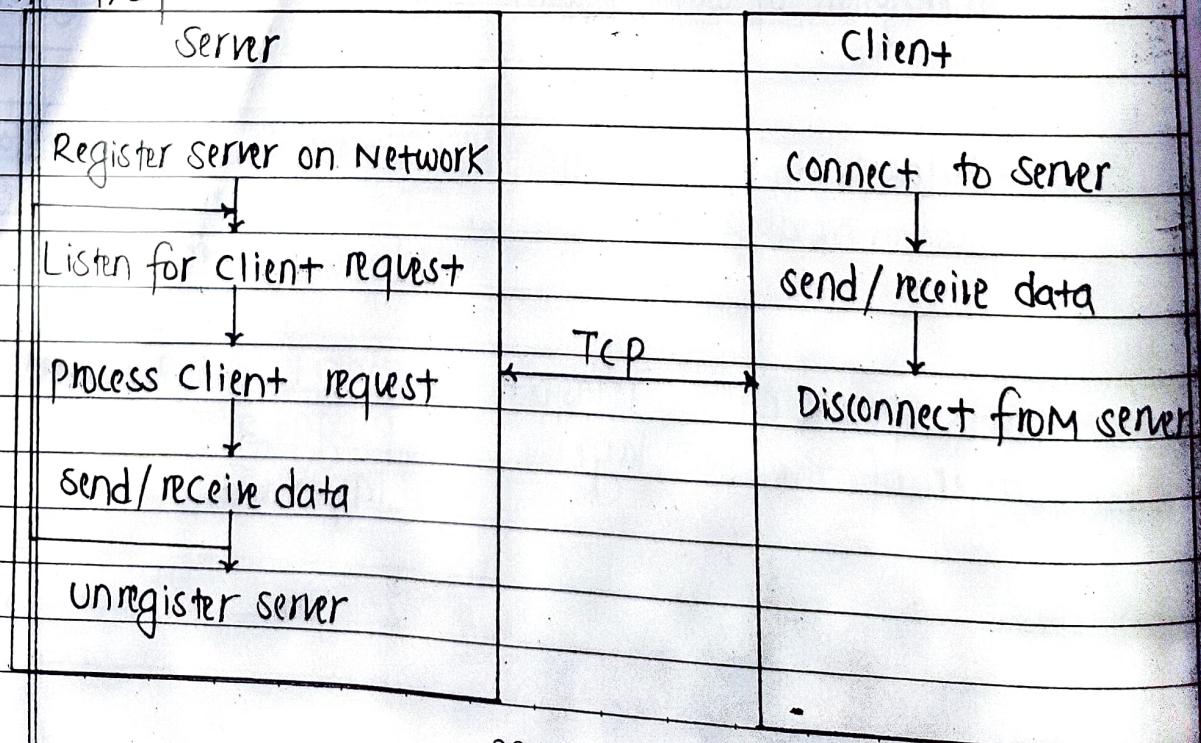
Ip Address + port number = socket.

Fig:- ports and sockets.

eg:-



Tcp/Ip



Tcp/Ip servers

Step1 :- Create a server socket and bind it with a port.

SYNTAX :-

```
ServerSocket serverSocket = new serverSocket(6848);
```

Step2 :- Listen for connection

SYNTAX :-

```
Socket socket = serversocket.accept();
```

Step3 :- Read data from client

SYNTAX :-

```
InputStream input = socket.getInputStream();

BufferedReader reader = new BufferedReader( new
  InputStreamReader(input);

  String line = reader.readLine(); // reads a line of
        text.
```

Step4 :- Send data to the client.

SYNTAX :-

```
OutputStream output = socket.getoutputStream();
PrintWriter writer = new printWriter(output, true);
Writer.println("This is a Message sent to server");
```

Step5 :- Close connection

SYNTAX :- `socket.close();`

Step6 :- Terminates the server.

SYNTAX :- `serversocket.close();`

Let's see a simple java socket programming where client sends a text and server receives and prints it.

=> File : Myserver.java

```
import java.io.*;
import java.net.*;

public class Myserver{

public static void main(string[] args){

try{

ServerSocket ss = new serverSocket(6666);

System.out.println("server is running on port 6666");

Socket s = ss.accept();

system.out.println("client connected");

DataInputStream din = new DataInputStream(s.get
InputStream());

String str = (String) din.readUTF();

System.out.println("client says=" +str);

  ss.close();
  s.close();
  din.close();
```

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

⇒ File Myclient.java.

```
import java.io.*;
import java.net.*;

//(public client) ⇒ My File.

public class Myclient {

Public static void Main(String[] args) {
try {
    Socket s=new Socket("localhost", 6666);

System.out.println("Reply to the server");

DataOutputStream dout= new DataOutputStream(s.get
outputstream());

    dout.writeUTF("Hello server");

    dout.flush();
    dout.close();
    s.close();
```

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

Example of Java socket programming (Read-write both side) once chatting.

File myserver.java

```
import java.net.*;
import java.io.*;

public class myserver{

public static void main(String[] args){
try{

ServerSocket ss = new ServerSocket(4000);
system.out.println("server is running on the port+4000");
// Socket s => new listing .socket accepting class
Socket s = ss.accept();
System.out.println("client connected");
DataInputStream din = new DataInputStream(s.getInput
Stream());

String clientMsg = (String) din.readUTF();
System.out.println("the client says:" +clientMsg);
```

```
System.out.println("Reply to the client Message");
String reply = "Hello I am server";
DataOutputStream dout = new DataOutputStream(s.get
OutputStream());
    dout.writeUTF(reply);

    dout.close();
    din.close();
    s.close();
    ss.close();
}catch(Exception e){

    System.out.println(e.getMessage());
}
}
}.
```

⇒ File Myclient.java

```
import java.net.*;
import java.io.*;

class MyClient {

public static void main(String[] args){
try{

Socket s= new Socket("localhost",4000);
system.out.println("Reply to the server");
```

```
DataOutputStream dout = new DataOutputStream(s.getOutputStream());

String servermsg = "Hello I am client";

    dout.writeUTF(serverMsg);

DataInputStream din = new DataInputStream(s.getInputStream());

    String clientmsg = (String) din.readUTF();

System.out.println("the client says :"+clientMsg);

    dout.flush();

    dout.close();

    din.close();

    s.close();

} catch (Exception e)
{

    system.out.println(e.getMessage());
  }
 }
}
```

Java Socket programming taking inputs from user and continous chat application.

```
import java.net.*;
import java.io.*;
import java.util.*;

Class Server{
public static void main(String[] args){
  try{

ServerSocket ss = new ServerSocket(3333);
System.out.println("server is running on port 3333");

Socket s = ss.accept();
System.out.println("client connected");

DataInputStream din = new DataInputStream(s.getInput
  Stream());

DataOutputStream dout = new DataOutputStream(s.getOutput
  Stream());

Scanner br = new Scanner(system.in);

String clientMsg = "", serverMsg = "";

While (!ClientMsg.equals("bye")){

 ClientMsg = din.readUTF();

 System.out.println("Client says:" + ClientMsg);
```

```
System.out.println("Reply to the client");
serverMsg = br.nextLine();
dout.writeUTF(serverMsg);
dout.flush();
}

din.close();
s.close();
ss.close();
dout.close();
} catch(Exception e){

System.out.println(e.getMessage());
}

}
}



import java.net.*;
import java.io.*;
import java.util.*;

class Client{
public static void Main(String[] args){
try{
Socket s = new Socket("localhost", 3333);

DataInputStream din = new DataInputStream(s.getInputStream());
DataOutputStream dout = new DataOutputStream(s.getOutputStream());
Scanner br = new Scanner(System.in);
```

```
String serverMsg = "" ; String clientMsg = ""

while (!serverMsg.equals("bye")){

 serverMsg = br.nextLine();
System.out.println(" send Message to a server");

 dout.writeUTF(serverMsg);

 dout.flush();

 clientMsg = din.readUTF();

System.out.println("server says :" + clientMsg);
}

   dout.close();
    s.close();

}catch(Exception e)
  {

     System.out.println(e.getMessage());
  }
}
}
```

Tcp vs UDP

Tcp	UDP.
1) Transmission control protocol	1) User Datagram protocol.
2) Tcp is connection oriented.	2) UDP is connection less protocol.
3) It is used by HTTP, HTTPS, FTP, SMTP.	3) It is used for DNS, DHCP, VOIP.
4) Slower than UDP.	4) UDP is faster because error recovery is not attempted
5) It's header size is 20 bytes	5) It's header size is 8 bytes.
6) Tcp does Flow control	6) UDP does not have option for Flow control.
7) Tcp does error checking and error recovery.	7) UDP does error checking but discard erroneous packets.
8) Tcp rearranges data packets in the order specified.	8) UDP has no inherent order as all packets are independent of each other.
9) Tcp is suited for applications that requires high reliability and transmission time is less critical.	9) UDP is suitable for applications that need fast, efficient transmission such as games.
10) Tcp is heavyweight. It requires three packets to setup a socket connection, before any user data can be sent.	10) UDP is lightweight. There is no ordering of messages, no tracking connection, etc.

UDP Server

```
import java.net.*;

class Server{
public static void main(String[] args){
  try{

    DatagramSocket socket = new DatagramSocket(9090);

byte[] receiveData = new byte[1024];
byte[] sendData = new byte[1024];

While(true){

DatagramPacket receivePacket= new DatagramPacket(receiveData,
                receiveData.length);

  Socket.recein(receivePacket);

String sentence = new string(receivePacket.getData());

System.out.println("Received=" + sentence);

String stringData = "Hello client!";
  sendData =(string)Data.getBytes();

InetAddress clientIpAddress = receivepacket.getAddress();

int clientport = receivepacket.getport();
DatagramPacket sendpacket =new DatagramPacket(
    sendData, sendData.length, clientIpAddress,
  clientport);
  Socket.send(sendpacket);   socket.close();
}
```

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

UDP client

```
import java.net.*;

class Client{
public static void main (String[] args){
try{

DatagramSocket clientSocket = new DatagramSocket();
byte[] sendData = new byte[1024];
byte[] receiveData = new byte[1024];

String stringSendData = "Hellowserver";
sendData = (String)sendData.getBytes();

InetAddress ip = InetAddress.getLocalHost();

DatagramPacket sendpacket = new DatagramPacket(sendData,
        sendData.length, ip.9090);

clientSocket.receive(receivepacket);
DatagramPacket receivepacket = new DatagramPacket(receiveData,
        receiveData.length);

clientSocket.receive(receivepacket);
//receivepacket)//it receives packets.
```

```
receiveData = receivePacket.getData();

String stringReceiveData = new String(receiveData);

System.out.println("FROM server :" + stringReceiveData);
    clientSocket.close();
}
 catch(Exception e)
 {
     System.out.println(e.getMessage());
 }
}
}
```

Write a client program which connects with server and also (print) running on port 5000 and send "I am client" Message to server and also print received Message from server in console.

```
import java.io.*;
import java.net.*;

Class Server{
Public static void main(String[] args){
 try{
ServerSocket ss = new ServerSocket(5000);

Socket s = ss.accept();
```

```
DataInputStream din = new DataInputStream(s.getInputStream());
String clientMsg = (String) din.readUTF();
System.out.println(clientMsg);
String reply = "Received Message";
DataOutputStream dout = new DataOutputStream(s.getOutput
    Stream());
    dout.writeUTF(reply);
    dout.close();
    din.close();
    s.close();
    ss.close();
} catch (Exception e)
{
    System.out.println(e.getMessage());
}
}

import java.io.*;
import java.net.*;

class Client{
public static void main(String[] args){
try{

    Socket s = new Socket("localhost", 5000);
```

```
DataOutputStream dout= new DataOutputStream(s.getOutput
Stream());

String serverMsg = " I am client";

    dout.writeUTF(serverMsg);

DataInputStream din= new DataInputStream(s.getInputStream())

    String clientMsg=(String) din.readUTF();

System.out.println("the client says:" + clientMsg);

    dout.flush();
    dout.close();
    dout.close();
    din.close();
    s.close();
} catch(Exception e){

    System.out.println(e.getMessage());
}
}
}
```

Key classes of Datagram.

** DatagramSocket class represents a connection-less socket for sending and receiving datagram packets.

** DatagramPacket is a Message that can be sent or received. If we send multiple packet, it may arrive in any order. Additionally packets delivery is not guaranteed.

Types of sockets provided by java are :-

1) Stream sockets :- It allows processes to communicate using TCP. It provides bidirectional, reliable, sequenced and unduplicated flow of data with no record boundries. After the connection has been established, data can be read from and written to these sockets as a byte stream.

2) Datagram sockets :- It allows processes to use UDP to communicate. A datagram socket supports bidirectional flow of messages. A process on a datagram socket can receive duplicate message. Record boundries in the data are preserved.

3) Raw sockets :- It provide access to ICMP. These sockets are normally datagram oriented, although their exact characteristics are dependent on the interface provided to support by the protocols. only superuser processes can use raw sockets.

2018 spring

5b) How do you get the list of Ip address that are assigned to Network interface?

- We can obtain this information of Ip address from a NetworkInterface instance by using one of two methods.
- The first method is getInetAddress(), returns an Enumeration of InteAddrss.
- The other method is getInterfaceAddress() returns the list of java.net.InterfaceAddress instances. This Method is used when we need more information about the interface address beyond its Ip address.

The following programs lists all the network interface and their addresses on a machine.

```
import java.io.*;
import java.net.*;
import java.util.*;
import static java.lang.system.out;

Public class IpList {

    Public static void main(string[] args) throws
        SocketException {

Enumeration <NetworkInterface> nets = NetworkInterface.
        getNetworkInterfaces();

    for(NetworkInterface netint: collections.list(nets))
            displayInterfaceInformation(netint);
}

Static void displayInterfaceInformation(NetworkInterface
    netint) throws SocketException {

    out.printf("Display name: %s\n", netint.
            getDisplayName());

    out.printf("Name: %s\n", netint.getName());

Enumeration <InetAddress> inetAddresses = netint.
    getInetAddresses();
```

```
for ( InetAddress inetAddress : Collections.list
        (inetAddresses));

    out.printf ("InetAddress : %s\n",
                inetAddress);
}
    out.printf ("\n");
}
}
```

Sample output of this program

output :-

Display name : Tcp Loopback interface
Name : Io
InetAddress : /127.0.0.1

Display name : Wireless Network connection
Name : etho

InetAddress : /102.0.2.0

Describe the JDBC Architecture.

The JDBC- Architecture consists of two-tier and three-tier processing models to access a database.

i) Two-tier model :-

- A Java application communicates directly to the data sources. The JDBC drivers enables the communication between the application and the data source.
- When a user sends a data query to the data source, the answers for these queries are sent back to the user in the form of results.
- The data source can be located on a different machine on a network to which a user is connected.
- This is also known as client/server configuration, where the user's machine acts as a client and the machine having the data source running acts as the server.

ii) Three-tier model :-

- In this, the user's queries are sent to middle-tier services, from which the commands are again sent to the data ~~der~~ source.
- The results are sent back to the middle tier, and from there to the user.
- This type of model is found very useful by management information system directors.

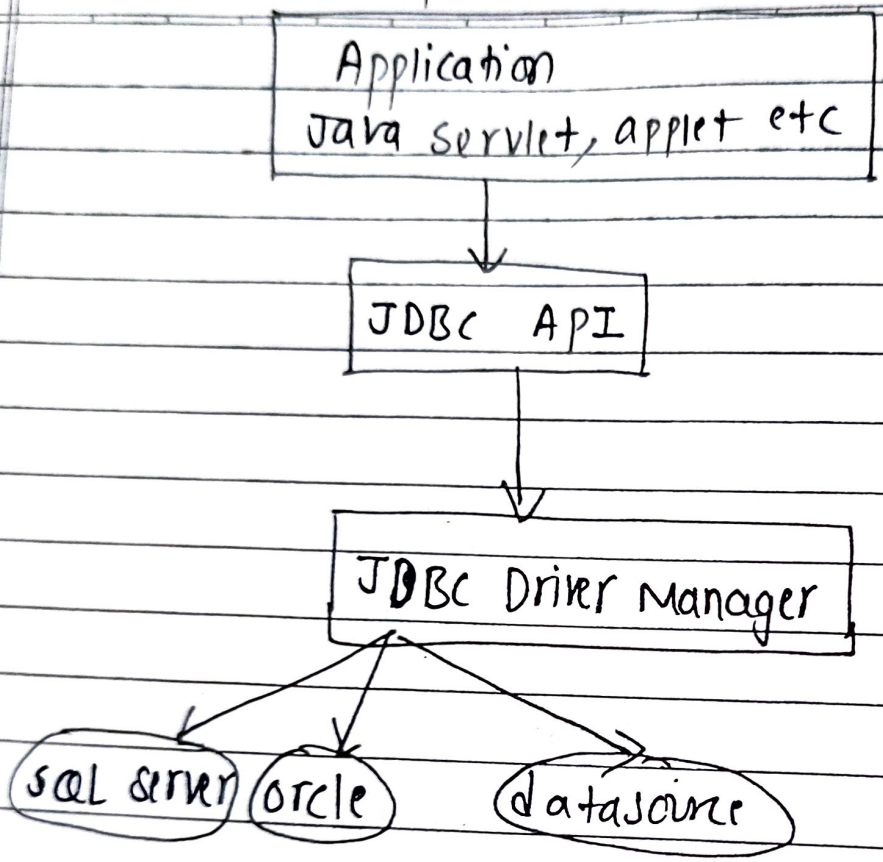


Fig:- JDBC Architecture.