

3

Explain J2EE Architecture.

J2EE stands for Java Enterprise Edition which is used to develop large ~~so~~ scale applications.

J2EE is mainly used for developing web applications.

J2EE has a four-tier architecture which is also called multi-tier architecture.

These are the four-tiers of J2EE.

- a) Client Tier
- b) Web Tier
- c) Business Tier
- d) EIS Tier.

Java EE Application 1	Java EE Application 2	Client Tier
JSP, Servlets		Web Tier
Enterprise Beans		Business Tier
Database		EIS Tier

a Client Tier :

Client Tier is also called Presentation Tier which is contains Client Side Server.

Client Tier consists of programs that interact with the users.

It consists all the web application which is operate by users.

This Tier prompt the user for input as a request to forward to software.

b Web Tier :

Web Tier consists all the Jsp and servlets content.

The components of web tier use HTTP to receive requests and send responses to the client.

Web Tier Provides service to Client-tier using HTTP.

c Business Tier :

Business Tier is also called

Business Tier contains business logic for J2EE application which is contain Java EE server.

In this tier two or more Enterprise Beans reside.

This tier provides concurrency, scalability, life cycle management and back up to J2EE application.

d) J EIS tier:

EIS stands for Enterprise Information System.

This tier provides directly interface to J2EE application to the databases.

EIS tier is also called Databases Server.

g) Describe URL and URL Connection class in Network Programming?

=> URL: URL stands for Uniform Resource Locator.

URL is one of the key concepts of web which provides published resource on the web.

URL has main two components:

- a) Protocol Identifier
- b) Resource name

Protocol Identifier contain name of Protocol which is used by URL.

Resource name contain name of Resource which you want to use in web.

Ex. https:// thebrainspot.in

↓
Protocol
Identifier

↓
Resource
name

→ URLConnection Class:

This class is used to communicate with URL over the network.

The URLConnection class contains many methods to communicate with URL.

URLConnection class is abstract class which contains two subclasses.

- 1) HttpURLConnection
- 2) JarURLConnection

1 HttpURL Connection:

This class is works for HTTP protocol which is used to retrieve information of URL which contains http protocol

2 JarURL Connection:

This class is used represents URL connection to a jar file.

This are the Basic Method of URLConnection class.

(i) `getContent()`: Reterieves the contents of the URL connection.

(ii) `getDate()`: Return the value of the date header field.

(iii) `getExpiration()`: Return the value of the expires header files.

(iv) `getLastModified()`: Return the value of last modified header file.

(v) `getURL()`: Returns the value of the URLConnection's URL Field.

- Syntax to Create URL Connection:

```
URL Object name . = new URL("URL Value");
```

```
URLConnection Object =
```

```
URL Object name . openConnection();
```

- Example:

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

```
public class java  
{
```

```
    public static void main(String  
        args[])
```

```
{
```

```
    try
```

```
{
```

```
        URL url = new URL("https://  
        thebrainspot.in");
```

```
HttpURLConnection con =  
    (HttpURLConnection) url.  
        openConnection();  
  
long time = con.getLastModified();  
  
if (time == 0)  
{  
    System.out.println("No  
        Update");  
}  
else  
{  
    Date date = new Date(time);  
  
    System.out.println("Last-  
        Modification Date: " + time  
        + date);  
}  
  
con.disconnect();  
}  
catch (IOException e)  
{  
    e.printStackTrace();  
}  
}
```


What is TCP/IP Server Socket and Socket?

Socket and ServerSocket classes are used for connection-oriented socket programming.

ServerSocket:

The ServerSocket class is used to create a server socket.

ServerSocket Object is used to communicate with clients in application.

Syntax to create ServerSocket Connection:

```
ServerSocket ServerSocket =  
Object
```

```
new ServerSocket (Port  
Number);
```

```
Socket Socket = ServerSocket  
Object . Object .
```

```
accept();
```

- ServerSocket Method :

There are two important method for ServerSocket.

- ci) accept() : create connection between server and client.
- cii) close() : close connection between server and client.

→ Socket Class :

The socket class is used to create client side socket.

Socket object is communicate with Server Socket.

- Syntax For Creating Client Side Socket :

```
Socket socket = new
    Object
```

```
Socket ("host", Port );
        name    Number
```

- Socket Method :

1. `getInputStream()` : Return the input value of socket.
2. `getOutputStream()` : Return the output value of socket.
3. `close()` : close the client side socket.

→ Example :

Server Socket :

```
class Server
{
    public static void main(String
        args[])
    {
        try {
            ServerSocket s = new
                ServerSocket(3201);
            Socket c = s.accept();
            DataInputStream dis = new
                DataInputStream(c.getInputStream());
```

```

        String str = (String)dis.readUTF();
        System.out.println("Message = " +
            str);
    }
    s.close();
}
catch (Exception e)
{
    System.out.println(e);
}
}
}

```

```

Client (Socket):

```

```

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

```

```

        try {

```

```

            Socket c = new Socket
                ("localhost", 3201);

```

```

            DataOutputStream dout = new

```

```

                DataOutputStream (c.getOutputStream());

```

```
dout.writeUTF("Hello Khushi");
```

```
dout.flush();
```

```
dout.close();
```

```
c.close();
```

```
}
```

```
catch (Exception e)
```

```
{
```

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

```
}
```

```
}
```

Explain DatagramSocket and DatagramPacket class.

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

These classes are used in UDP network communication for providing message.

-> DatagramSocket class:

DatagramSocket class represents a connection-less socket.

DatagramSocket class provides method to transmitting datagram in the network.

- Syntax For create class:

```
DatagramSocket DatagramSocket  
Object
```

```
= new DatagramSocket();
```

- Method:

ci) close(): Closes the datagram socket.

cii) disconnect(): Disconnect the socket.

ciii) getPort(): Returns the port number of socket.

iv) send(): Sends the Datagram Packet from socket.

(v) receive() : Receives the datagram packet from the socket.

→ DatagramPacket class:

DatagramPacket class is a message that can be send or received by Socket.

DatagramPacket class is a one type of data containe to use to send the Packet from the Socket.

- Syntax for create class:

```
DatagramPacket DatagramPacket  
Object
```

```
= new DatagramPacket (byte[]  
    buff, int length);
```

- Method:

(i) byte[] getData() : Returns the data buffer.

(ii) getPort() : Returns the port number on the remote host.

- iii) `setPort(int iport)`: Sets the Port number of the remote host.
- iv) `setLength(int Length)`: Sets the length of the packet.
- v) `setData(byte[] buff)`: Sets the data buffer for the packet.

→ Example:

UDPClient:

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

```
class UDPClient
{
```

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

```
        try {
```

```
            DatagramSocket ds = new
                DatagramSocket();
```

```
            String str = "Hello";
```

```
            byte[] send = str.getBytes();
```



```
InetAddress sa = InetAddress.  
    getByName("localhost");
```

```
int Port = 3201;
```

```
DatagramPacket dp = new Datagram  
    Packet(send, send.length, sa, Port);
```

```
ds.send(sendp);
```

```
byte[] receive = new byte[1024];
```

```
DatagramPacket dr = new  
    DatagramPacket(receive,  
        receive.length);
```

```
ds.receive(receive);
```

```
String c = new String(receive.  
    getData(), 0, dr.getLength()  
    ());
```

```
System.out.println("Received  
    data " + c);
```

```
    }  
    ds.close();
```

```
catch (Exception e) {  
    e.printStackTrace();  
}
```

```
}  
}  
UDP Server:
```

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

```
class server {
```

```
    public static void main (String  
        args[])
```

```
    {  
        try {
```

```
            DatagramPacket rp = new
```

```
            DatagramSocket ds = new  
            DatagramSocket(32010);
```

```
            byte[] rd = new byte [1024];
```

```
            while (true) {
```

```
                DatagramPacket rp = new  
                DatagramPacket (rd, rd.length);
```

```
                ds.receive (rp);
```

```
String SM = new String(rp.getData()  
    c, 0, rp.getLength());
```

```
System.out.println("Received "  
    + SM);
```

```
String C = SM.toUpperCase();
```

```
InetAddress clientAddress =  
    rp.getAddress();
```

```
int port = rp.getPort();  
byte[] sd = C.getBytes();
```

```
DatagramPacket sp = new  
    DatagramPacket(sd, sd.length,  
    clientAddress, port);
```

```
rp.send(sp);
```

```
}  
}
```

```
catch (Exception e)
```

```
{
```

```
    e.printStackTrace();
```

```
}
```

```
}
```

```
}
```

=> UDP Server :

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

class server
{
    public static void main (String
        args[])
    {
        try {
            DatagramSocket ds = new
                DatagramSocket (3201);

            byte[] rd = new byte [1024];

            while (true)
            {
                DatagramPacket rp = new
                    DatagramPacket (rd, rd.
                        length());
                ds.receive (rp);

                String sm = new String
                    (rp.getData(), 0, rp.
                        getLength());

                System.out.println (sm);
            }
        }
    }
}
```

```
String c = sm.toUpperCase();  
InetAddress ca = rp.getAddress  
();  
int port = rp.getPort();  
byte[] sd = r1.getBytes();  
DatagramPacket sp = new  
DatagramPacket(sd, sd.length,  
ca, port);  
rd.send(sp); }  
catch (Exception e)  
{  
e.printStackTrace();  
}  
}  
}
```