JAVA

JAVA - 1

J2SE
J2EE → Java 2 Enterprise Edition
J2ME → Mobile Edition

J2EE
- RMI
- JDBC
- Java Database connectivity
- Servlet
- JSP (Java server pages)
- EJB (Enterprise JavaBean)
- Struts

Lifecycle of Thread :-

init () Initialization
start ()
run () → running
destroy () → runnable

wait ()
sleep ()
suspend ()

JAVA - 2

Thread safe → synchronized

final
- class does not extend.
- variable becomes constant.
- Method does not inherit

finalize → garbage collection

All classes in Java are under Object Class

package not needed to be imported:
java.lang.*
↳ to import everything inside any package.

In case of multiple main only one class whose main has to be used should be declared as Public.

Interface (not defined methods)
→ to extend a class to implement.

Access specifiers in Java:-

Modifiers	Default	Private	Protected	Public
Accessible inside the class	Yes	Yes	Yes	Yes

within the sub-class inside the same package	Yes	No	Yes	Yes
outside the package	No	No	No	Yes
within the sub-class outside the package	No	No	Yes	Yes

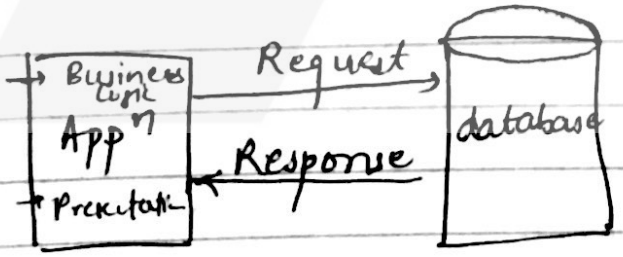
Abstract Class

Abstract class cannot be instantiated, but they can be subclassed. When an abstract class is subclassed, the subclass usually provides implementations for all of the abstract method in its parent class. However, if it does not, then the sub-class must also be declared abstract.

10/01/13

J2EE arch

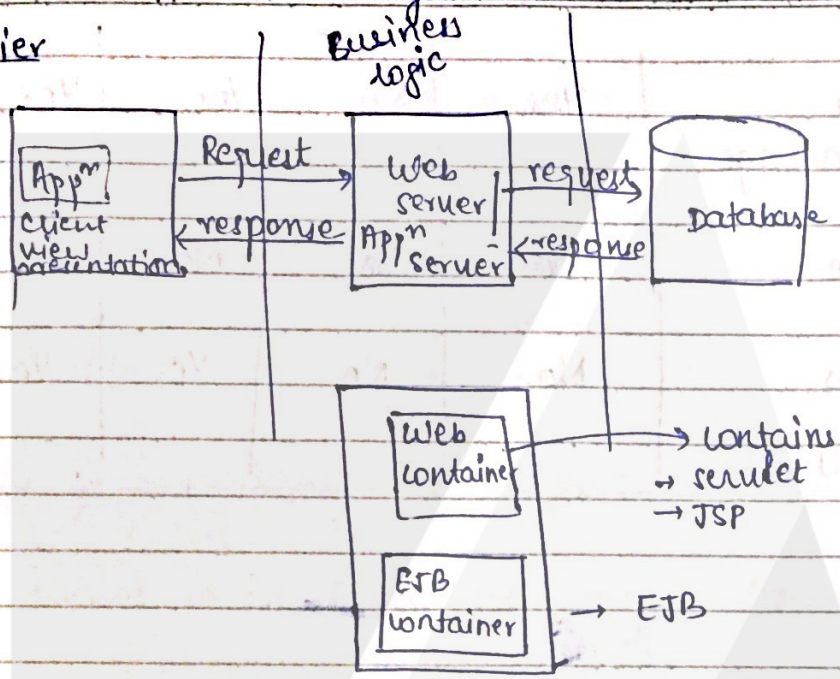
2- Tier



** J2EE → Wrox publication
Abress publication
M. Subramanyam

www.Roseindia.com

3-Tier



Bea ← Web server

IBM Websphere ← Application Server

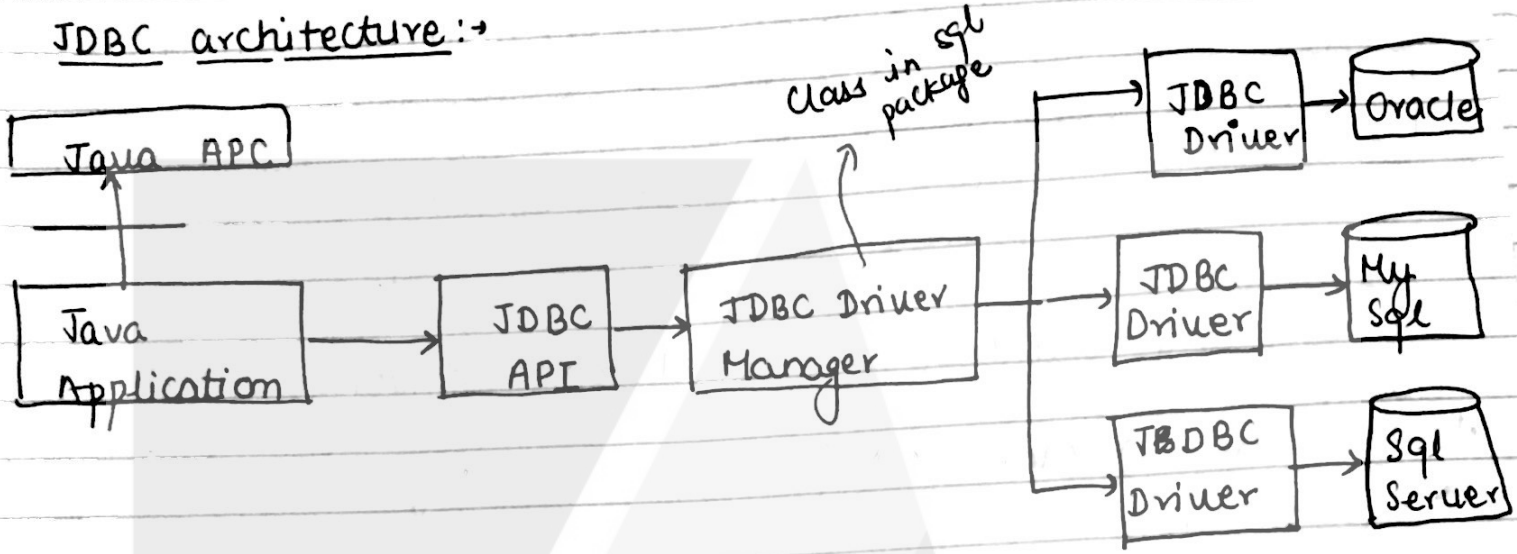
*) diff b/w Web server and Application Server?

17/01/18

Odbc → open database connectivity.
(written in c)

JDBC :→
Mc Access
Oracle
Sql Server
My Sql

JDBC architecture :→



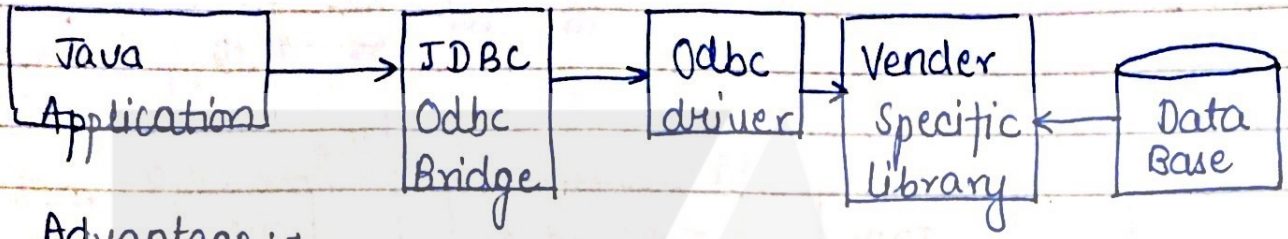
Java . Sql *
Java . Sql * ; → Connection pool (advance feature)

JDBC is an API specifications developed by Sun Microsystem that defines a uniform interface for accessing various relation databases.
JDBC driver translates standard JDBC calls into a network o database protocol or into a database library into API call that facilitates communication with a database. This id translation layer provides JDBC layer with data independe

** Type of Driver :→

1) JDBC - ODBC bridge Driver or Type 1 :
2). Native - API driver or Type 2 :
3). Network protocol driver or Type 3 :
4). Pure Java or Thin driver or Type 4 :

1) JDBC - Odbc bridge driver or Type 1:



Advantage:→

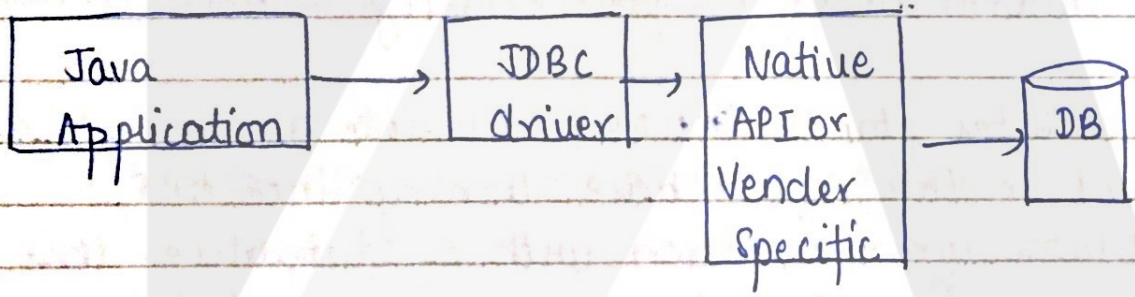
Almost any database for which odbc driver is installed can be accessed.

•) easy to use .) canbe easily connected to any database.

Disadvantage:

1. Performance degraded because JDBC method call is converted into the ODBC function call.
2. The odbc needs to be installed on the client machine.
3.

2). Native - API driver or Type 2:→



Type 2 driver use the Java Native Interface (JNI) to make calls to a local database libraries. This driver converts the JDBC calls into a data base specific calls. This driver communicates directly with the database server. It requires some native cord to connect to the database. Type 2 driver require Native database client - libraries to be installed & configured on client machine.

Advantage:

1). Better performance than JDBC-Odbc bridge driver.

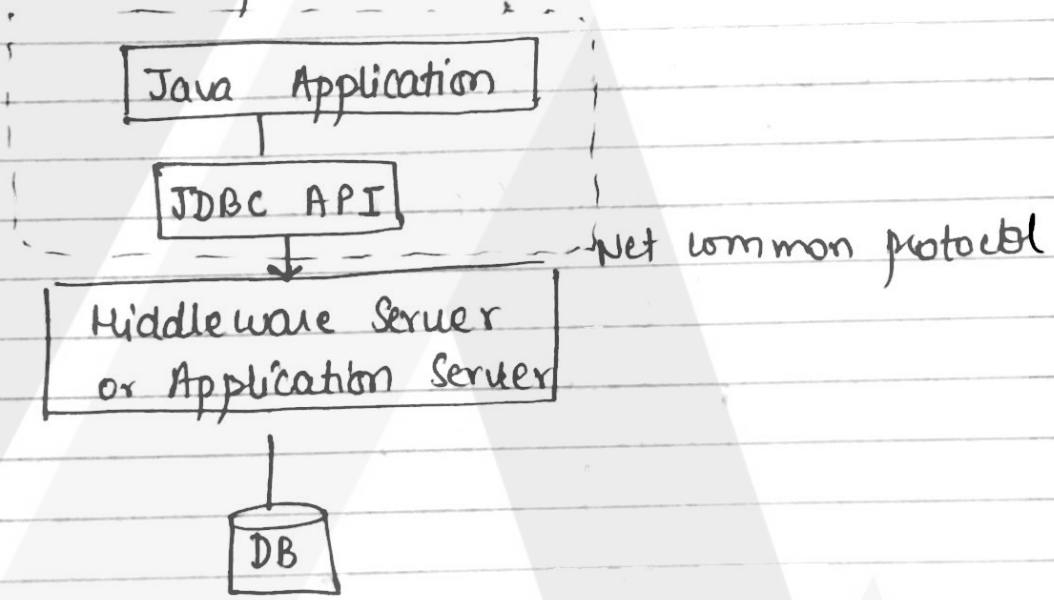
Disadvantage:

1). The native driver needs to be installed on each client machine.

2). Not all databases give the client-side libraries.

22/01/18

Type-3 (Network protocol driver):-



1). This driver is server based so there is no-need for any vender database library to be present on the client machine.

2). This driver is fully return in Java & hence portable and it is suitable for the web.

3). The Net protocol can be designed to make the client JDBC driver very small and fast to load.

4) The type-3 drivers typically provide support for features

Such as caching, load balancing & advance system administration such as login & auditing.

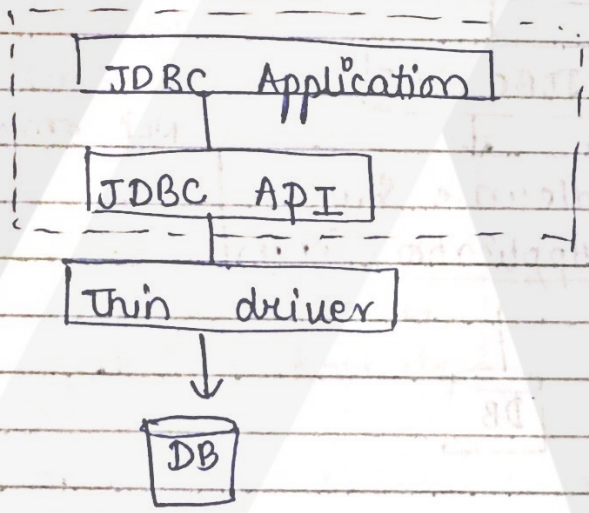
6). This driver is very flexible allow access to multiple databases using one driver.

Disadvantage :-

It requires another server application to install & maintain.

Traversing the ~~report~~ record set may take longer since the data comes ~~bac~~ through the backend server.

Type-4 (Pure Java Driver) :-



Type-4 driver use Java networking libraries to communicate directly with the data base server.

1). The Major benefit of using Type-4 driver are that they are completely written in Java to achieve platform independence & eliminate deployment administration issues. It is most suitable for the web.

2) No. of translation layer is very less i.e. type-4 JDBC drivers don't have to translate database request to ODBC or native connectivity interface or to pass the request on to another server.

3). U don't need to install special software on the client or server.

Steps for JDBC connectivity :-

1) Loading or registering drivers.
   Class.forName("driver name");

2). Creating connection object. getConnection(String url, String username, String password)
   Connection con = DriverManager.getConnection(String Url);
   (Protocol : subprotocol : database name)

3). Creating statement object.
   Statement st = con.CreateStatement(); → Statement / PreparedStatement / CallableStatement

4). Execute query using Resultset.
   Resultset rs = st.ExecuteQuery("select * from tablename").

5) Closing connection object.
   con.close();

4)
String ExecuteQuery( ) ← only select
int ExecuteUpdate( ) ← insert, delete
Boolean Execute( )
;

only class in java.sql * ← DriverManager ← java.sql *

7 interfaces
- Connection
- Statement
- Result set
;

* Prepared statement or Parameterized statement :-

* To execute a statement object many times, it usually reduces execution time to use a Prepared statement object instead.

The main feature of a prepared statement object is that, it is given a SQL statement when it is created. The advantage to this is that in most cases, this SQL statement is sent to the DBMS right away, when it is compiled. As a result, the Prepared statement object contains not just a SQL statement, but a SQL statement that has been precompiled. This means that when the Prepared statement is executed, the DBMS can just run the Prepared statement SQL statement without having to compile it first.

Stored procedure (in DBMS) :-

Ex :-

```
String updateString =
    "update" + dbName + ".COFFEES" +
        "set SALES = ? where COF_NAME = ?";
updateSales = con.prepareStatement (UpdateString);
```

* Stored Procedure :-

A Stored procedure is a group of SQL statements that form a logical unit and perform a particular task, and they are used to encapsulate a set of operations or queries to execute on a database server. for example, operations on an employ database

(hire, fire, promote, lookup) could be coded as stored procedures executed by application code. stored procedures can be compiled and executed with different parameters and results, and they can have any combination of input, output and input/output parameters.

Ex:- To print a result set that contains the names of coffee suppliers and the coffees they supply to the Coffee Break.

SHOW-SUPPLIERS:

Output:-

Acme, Inc.: Colombian-Decaf
Acme, Inc.: Colombian
Superior Coffee: French_Roast-Decaf
Superior Coffee: French-Roast
The High Ground: Espresso.

Q.1). diff. b/w Prepared & Callable statement.
Q.2). Execute query & execute update.
Q.3). Prepared statement & prepared call.
Q.4). Type 3 & Type 4 driver
Q.5). Scrollable & updatable result set
Q.6) Write short note on the following
1) driver Manager, connection interface, Statement interface result set.
Q.7). Steps for JDBC connectivity.

(Wrapper class)

```
import java.sql.*;
class abc
{
public static void main (String args[])
{
Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con = DriverManager.getConnection ("jdbc:odbc:DSN");
Statement st = con.createStatement();
ResultSet rs = st.executeQuery ("select * from student");
while (rs.next())
{
S.O.P. (rs.getString (1));
S.O.P. (rs.getString (2));
}
}
```

driver name for MS Access (always fixed)

DSN = database Name

↳ always select in execute

Name	Roll
1	23
2	24

ORACLE:-

Driver Name [oracle.jdbc.driver.OracleDriver
jdbc oracle then @ <Host>:<Port>:<SID>
OR Database name

MySQL [Driver com.mysql.mm.mysql.Driver
jdbc:mysql://<Host>{Port>|<DB>

Create Table Student (firstname varchar(32); lastname varchar(32));

int i = [st execute update (———)

execute()

Driver Manager Class: acts as an interface b/w user and driver. It keeps track of the drivers that are available & handles establishing a connection b/w a database & an appropriate driver.

Driver Manager Class maintain a list of driver classes that have registered themselves by calling the method driver Manager. register Driver ().

-) public static registerDriver ()
-) " " deregisterDriver ()
-) " " Connection getConnection ~~()~~; (String url)
-) " " " getConnection (String URL, user, password)
  getComment (String URL, user, Password)

↳ It use to establish the connection with the specified URL

Connection Interface:

A connection is the session b/w Java application and the database. The Connection Interface is a factory of Statement, prepared Statement and database metadata that is object of connection can be used to get the object of statement & database metadata. The connection

Interface provide many method for transaction management like commit, rollback etc.

public Statement. CreateStatement()
public Statement. Create Statement (int resultset type, int Resultset Concurrency)

1) pull. void commit();
2) " " rollback();
3) " " Close ();

Wrapper Class :- All the wrapper classes (int, long, byte, float, double short) are subclasses of the abstract class Number. The output of the wrapper class contains or wraps a respective primitive data type. Converting primitive data type into object and this is taken care by compiler.

Its used to convert any data type into an object.

29/01

Statement :-

Statement st = Con. CreateStatement ()
Prepared Statement ps = Con prepareStatement (String sql);

Callable Statement cs = Con. prepareCall (String sql);

Result Set :-

Id	Name
2	abc
3	cde
4	cde

```
ResultSet rs = St.executeQuery(_____)
While (rs.next())
    {
    }
```
↳ always start from first data record.

* Only used when result set is scrollable or updateable :-

next()
first()
last()
before first()
After last()
Absolute (int row)
(10) ↳ only 10th row

TYPE_FORWARD_ONLY
TYPE_SCROLL_INSENSITIVE → does not show any updatation
TYPE_SCROLL_SENSITIVE
CONCUR_READ_ONLY
CONCUR_UPDATABLE

```
Statement st = Con.createStatement(ResultSet.TYPE_SCROLL_
                    INTENSITIVE, ResultSet.CONCUR_READ_ONLY);
```

Program for Prepared Statement :-

```
Class abc
  {
  P.S.V.m(String a[])
   {
```

Class for Name (
Connection con = Driver Manager - - -
Prepared Statement ps = Con. prepare Statement ("insert into Emp values (?,?) ");

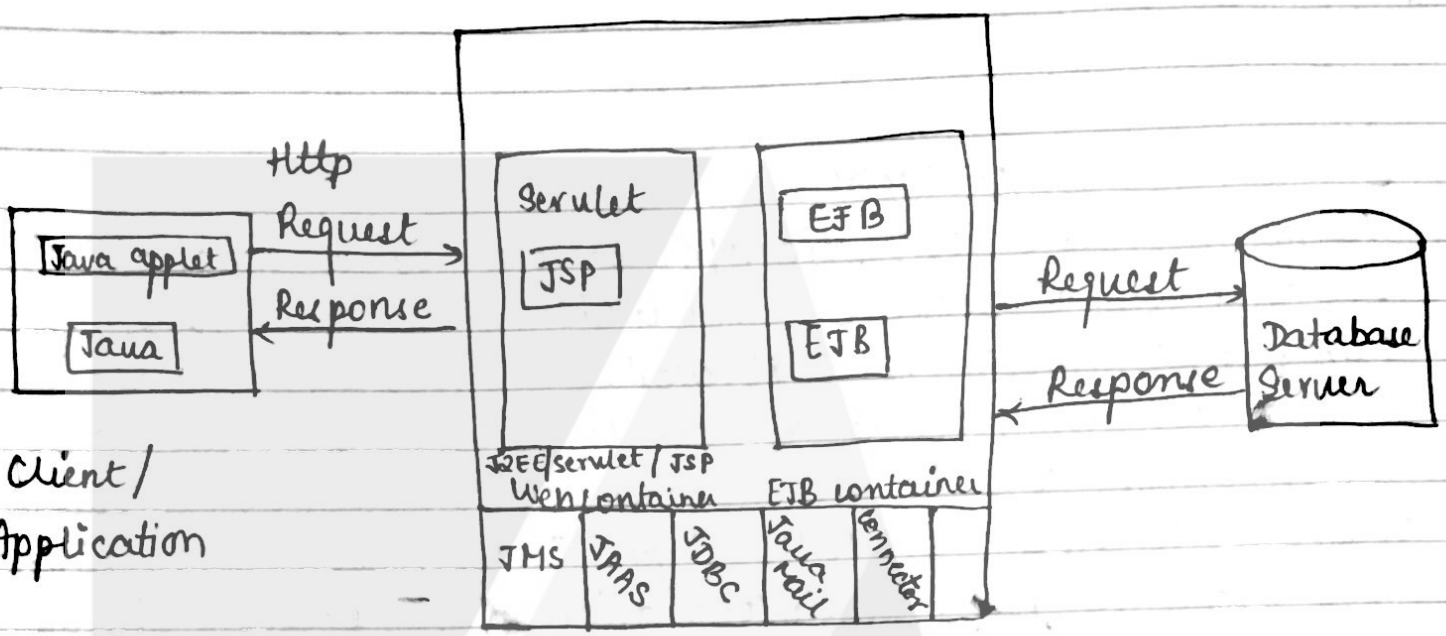
ps. set Int (1, 101);
ps. Set String (2, Amity);
int i = ps. execute Update ( ); → return ~~written~~ type is int that's why int i = ps. - - -
S.O.p. (i + " record inserted);

Result rs = ps. execute Query (- - - (— —)

Result set is the representation of a database table ~~written~~ returned from a Statement obj. A result set obj. Maintained a cursor pointing to current row of data. When the cursor is first returned it is positioned before the first row. To access the first row of the result set we need to call the next method of the result set interface.

31/01

J2EE Arch.



J2EE technologies

Stateless session bea
Stateful session plan

1) Component technology: are used to hold the most imp. part of the application i.e. business logic
   a) Web component → servlet, JSP
   b) EJB component → Session bea, Entity bea, Message dri bea

2) Service technology: provide the application component with supported service to function efficiently.
   - JDBC, - JNDI, - JAVA mail, - JAAS, - JMS, - Java transactio AP

3) Communication technology: These technology are most transparent to app^n programmers provides the mechan for communication among diff. part of the app^n whether they are local or remote.
   i) Internet protocol → Http, TCP/IP, SSL
   2) Remote object protocol → RMI, RMI-IIO

EJB component
- CMP (Container Managed Persistance)
- BMP

Http → stateless protocol

Http or Hyper text transfer protocol is a generic stateless appn level protocol which has many uses beyond Hyper text capabilities. It works on request response basis. A client sends a req. to the server in form of req. method, URI (Uniform Resource Identifier) & protocol version followed a MIME ~~type~~ like message containing req. modifier, client information and possible body content over a connection with the server. The server in turn responds with a status line followed by a MIME like message containing server info., entity meta info. & possible body content.

TCIP protocol:

TCIP is actually two seperate protocols. IP is the protocol that takes care of making sure that data is received by

When we type the add.g a website into ip is what enable that or request & the fulfillment of these requests make it to the proper destination for data being sent back and forth b/w a client and a web server is broken into several pieces or packets. These packets do not all have to take the same route when they are sent b/w the client & the web server. TCP is

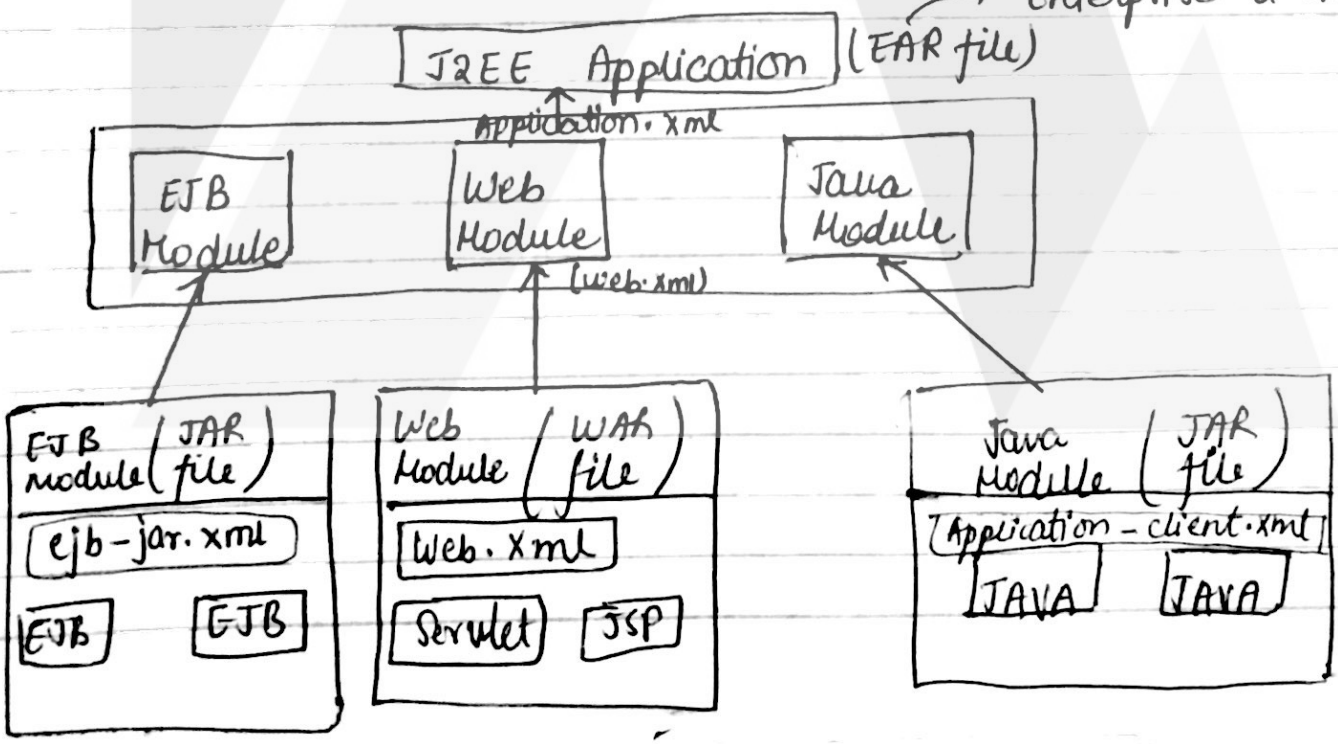
the protocol that keeps track of the packets & make sure they are assembled in the same order as they were dispatched & are error free. TCP & IP works together to move data around on the internet.

** Developing J2EE Application :-

1). Application Component Development.
2). Composition of App. components into module.
3). Composition of module into Application.
4). App Deployment.

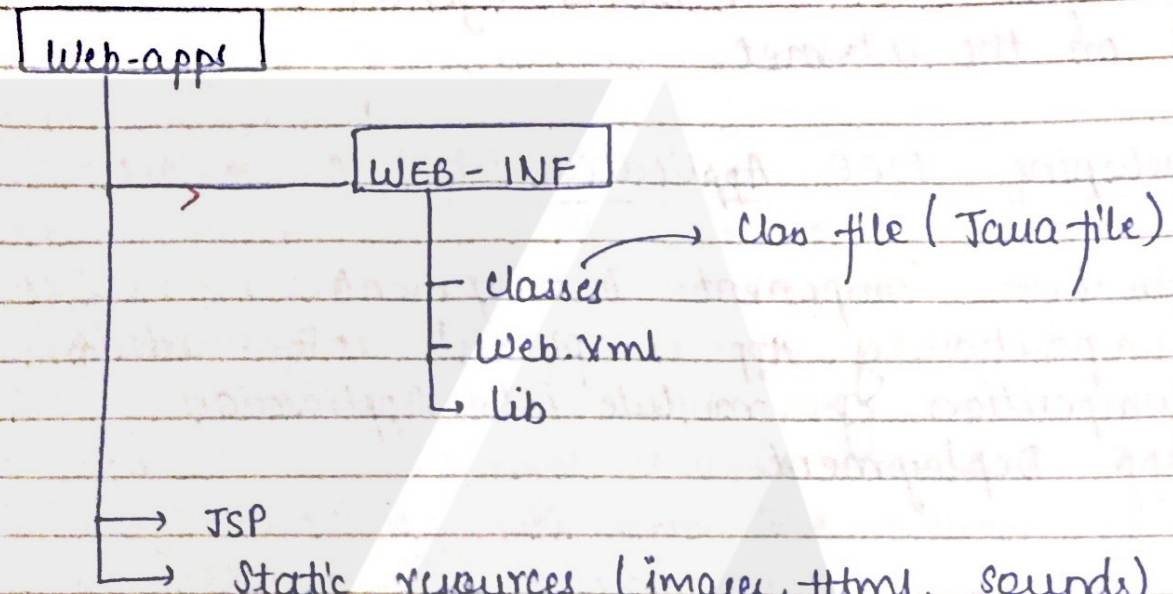
1) Java
- Servlet / JSP
- EJB

2) Module → EJB module (JAR)
→ Web module (WAR) Web archieve file
→ Java module (.jar)
→ Enterprise archieve



12/02

** Directory structure of Web Application :-



* web.xml

Servlet1.java

```
<Web-apps>
  <Servlet>
    <Servlet-name>Servlet1</Servlet-name>
    <Servlet-class>Servlet1</Servlet-class>
  </Servlet>
  <Servlet-mapping>
    <Servlet-name>Servlet1</Servlet-name>
    <Url-pattern>/Hello</Url-pattern>
  </Servlet-mapping>
</Webapps>
```

/Hello → identifier

if it matches then Run

. → No need to match with identifier, it will run.

19/02

RMI (Remote Method Invocation)

RMI is an API that provides a mechanism to create distributed appⁿ in Java. The RMI allow an object to invoke methods on an object running in another JVM.

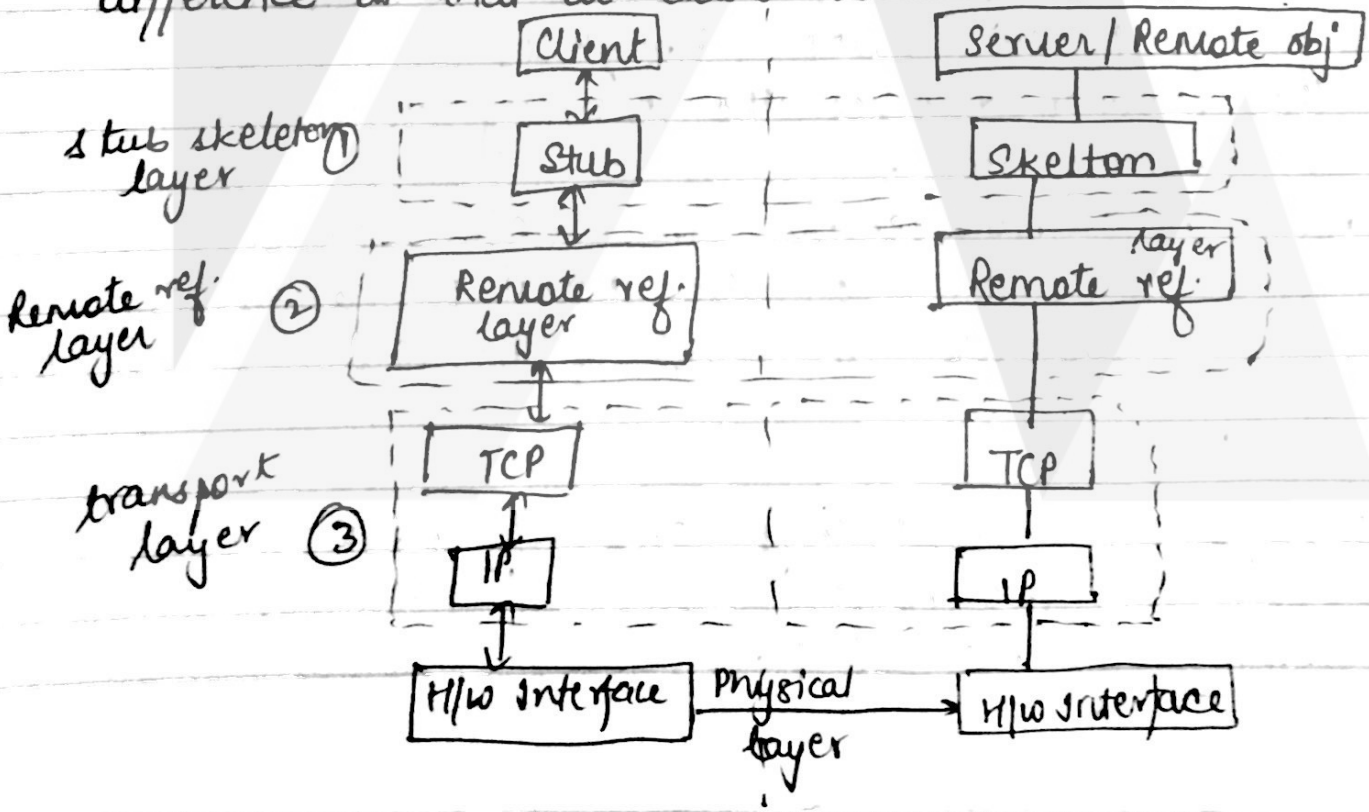
The RMI allow obj to obj communication b/w JVM.

Java.rmi * ↳ Package imported.

JRMP – Java Remote Method Protocol.
If appⁿ is in pure Java.

CORBA → Non Java application

JAVA RMI allow programmer to execute Remote fun. which behaves similar to local fun. calls but the difference is that it exists somewhere else.



1st Layer :-

Stub :-

Skeleton :-

Data —(byte conversion / Marshalling)→ Byte

Byte —(Unmarshalling)→ data

Stub :-

Stub is an obj. acts as a gateway for the client side. All the outgoing request are routed through it. It resides at the client side. When the caller invokes the method on stub obj. it does the following task :-

(i) It initiate a connection with remote virtual machine.
(ii) It writes and transmits (marshells) the parameters to remote virtual machine.
(iii) It waits for the result.
(iv) It reads (Unmarshells) the return value or exceptions.
(v) It finally return the value to the caller.

Skeleton :-

Skeleton acts as a gateway for the server side. All incoming request are routed through it.
When the skeleton receives the incoming request, it does the following task.

i) It reads the parameter for the remote method.
ii) It invoke the method on the actual remote objects.
iii) It writes & transmits the result of the caller. (Marshalling)

2nd Layer (RRL) :-

It is responsible for manage the references made by the client to the remote obj. on the server. So, it is available on both JVM.

The client side RRL receives the req. for methods from the stub that is transferred into byte stream process called serialization or Marshalling and then these data are send to the server side RRL.

* 3rd layer (transport layer) :-

**

Steps for RMI application :-

1). Create a remote interface.
2). Provide the implementation of Remote interface.
3). Create stub + skeleton obj. using rmic command.
4). Start the registry service by rmi registry tool.
5). Create & start the remote application or server.
6). Create & start the Client Application.

** Program (RMI application) :-

```
Product.java rmi.*;
public interface Product extends Remote
{
    public String getName();
}
```

Javac product.java

Product Imple.java

```
import java.rmi.*;
import java.rmi.server.*;
```

Javac product Imple.java

```
public class Product Imple extends unicastRemoteObject
                          implements Product
{
  String name;
public product Imple (String name)
  {
    this.name = name;
  }
public string getname () throws remote exception
  {
    return name;
  }
```

Product Server.java

```
import java.rmi.registry.*;
public class Product Server
{
  p.s.v. main ( String a[])
  {
try
  {
  locate Registry Create Registry (3000);

Product Imple P1 = new product imple ("PEPSI");
   "      "   P2 =  "     "    "    ("Vanila");

Naming rebind ("Cold drink" p1);
Naming rebind ("Ice cream" p2);
}
```

1049 (default port no.)

```
Catch ( Exception e)
{
  =
  }
}
```

product client.java

```
public class product client
  PSV main ( string a [])
  {
    try
    {
      product p = (product) Naming. lookup ( ".rmi://
        127.0.0.1:3000/ colddrink" );
      S.O.P ( " Product name is " + P. getName ( ));

    Catch ( exception e)
    {
      =
    }
  }
```

Type casting

Naming class provides methods for storing & obtaining reference to remote objects in a remote object registry.

4 methode in naming :⇒

static remote lookup (string)
static void bind (String, object)
" " unbind (string)
" " rebind (String, Object).

Q. diff b/w bind and Unbind.

rep
(3) rmic product 1mple
(4) start rmi registry.

rep. ques.

Q. Write short note on the following:
i) Unicast
ii) Remote obj. class
iii) Remote interface iii) Naming class iv) RMI registry.
iii) lookup method. iv) Marshalling, Unmarshalling (serialization)

·) Architecture
all 3 layer's.

·) RMI appn (Program).

22/02

Servlet → Server side components.

javax.servlet.* ← Packages import

•) Advantages
- → Portability.
- → No process creation overhead.
- → Efficient
- → Robust
- → Secure
- → Persistent

Servlet Life cycle :-

•) Instantiation :-
→ creates an instance of the servlet class with the help of new keyword.

•) Initialization :- (public void init (ServletConfig config)
init() method gets called.

•) Service :- (ServletRequest req, ServletResponse, res)
The web server has a request for a servlet it calls the servlet instance's service() method.

•) Destroy :-
Before destroying the instance, the container calls servlet instance's destroy() method.

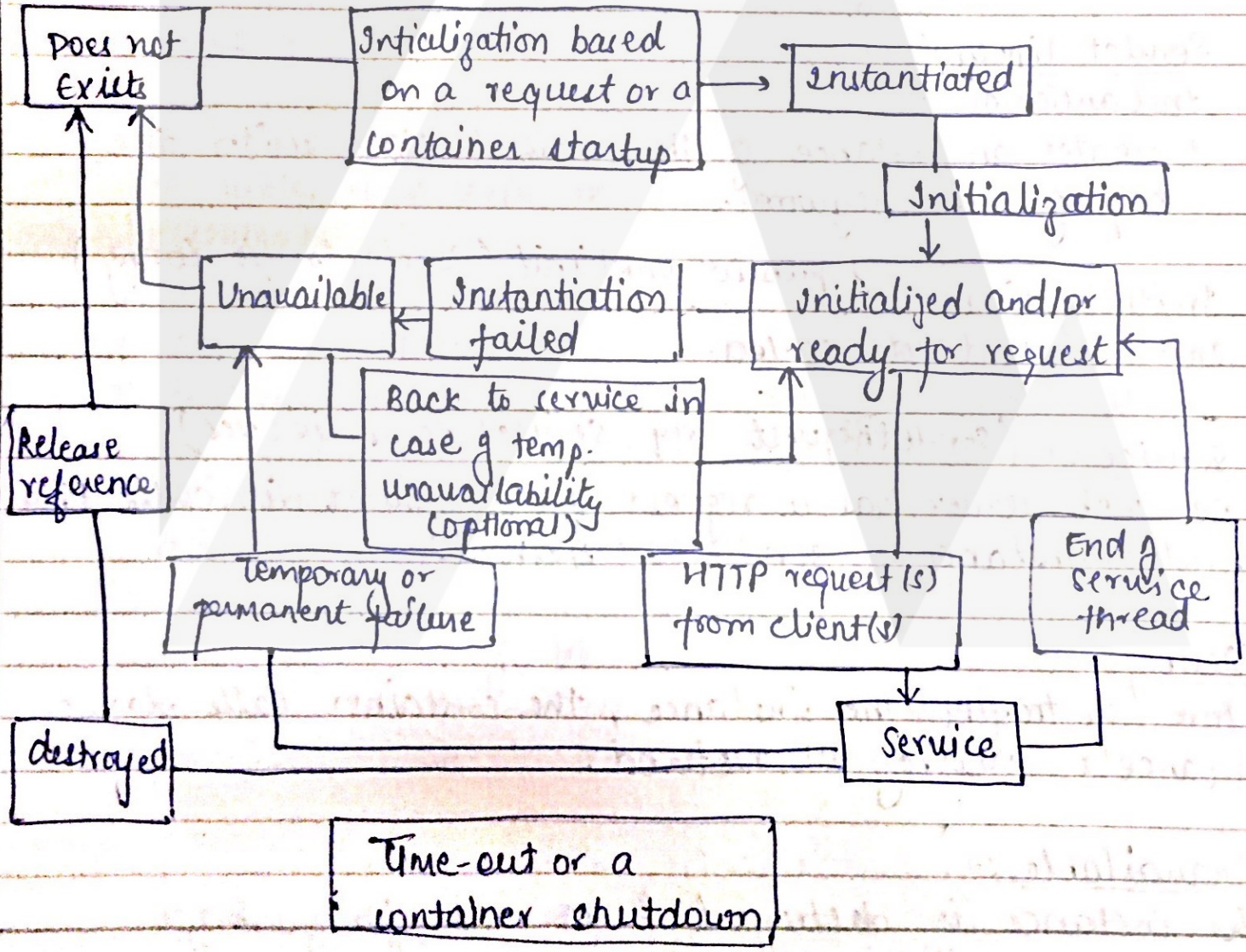
•) Unavailable :-
The instance is destroyed & marked for garbage collection.

diff. b/w servlet & CGI

servlet life cycle

- Servlet
- Servlet Request
- Servlet Response → Output print
- Servlet Config.
- Servlet context
- Single Thread Model → Thread safe & thread Synchronization are used using this.
→ Request dispatcher.

★ Servlet life cycle :-



→ Servlet Request

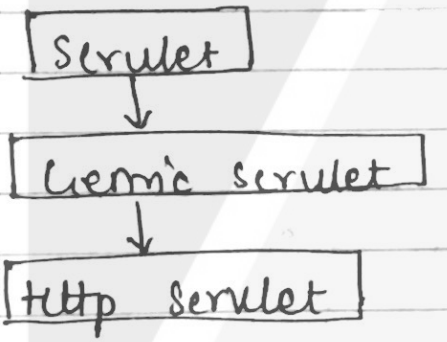
- ·) getprotocol ()
- ·) getRemote Addr. ()
- ·) get Remote Host ()
- ·) get Server Name ()
- ·) get server port ()

void ·) set Attribute (String name, obj.)

·) get Attribute (String name)

Method to Request Parameters :→

- String get Parameters (String name)
- String [] get Parameter Values (String s1)
- Enumeration get Parameter Names ()



public Service ( )
protected Service ( )
↓
doxxx
doGet
dopost

```
import javax.Servlet.X;
       javax.Servlet.http.X;

public class abc extends Http Servlet
    {
```

```
public void int ( ServletConfig config)
    {
    }       doGet
public void service (ServletRequest req , ServletResponse res)
    {
     res. SetContentType ("text/html");
     PrintWriter pw = res. getWriter();
     pw. println ("<html><body>");
          pw. println ("HELLO");

     }  pw. println ("</body></html>");
```

login html

```
<html>
 <body> <form method = "Post" action "     ">
Username < input type = "text" name = "Uname"/>
Password < input type = "Password" name = "Pan" />
          < input type = "Submit" value = "login" />
        </form>
        </body>
        </html>
```

```
import javax Servlet *
public class login extends http Servlet
{
public and depost (___req, ___ res)
```

short note on Enumeration

```
{
# res.setContentType("Text/Html");
PrintWriter pw = res.getWriter();
String Uname = req.getParameter("Uname");
String pwd = req.getParameter("pwd");

    pw.println("<Html>");
    pw.println("<Body>");
if(Uname.equals("ABC") && pwd.equals("001"))
  pw.println("Welcome" + Uname + "!!");
  else
  pw.println("login failed.");
  pw.println("</body>");
  pw.println("</Html>");
}
```

login html

```
<Html>
<body>
    <form method="Post" action=" ";>
Author <input type="text" name="Author" />
    <input type="Submit" name="Submit" />
    <input type="Reset" value="Reset" />
</form> Enumeration parameter = req.getParameterNames()
</body>
</Html>
```

```
while(parameters.hasMoreElements())
{ String ParameterName = (String) Parameters.nextElement())
out.println("Parameter Name" + ParameterName);
out.println("Parameter value" + req.getParameter(Permutations));
}
```

input java Servlet * :

```
public class login extends Http Servlet
{
public void depost (-- req, --- res)
{
 res. set Content Type (" Text / Html");
PrintWriter out = res get Writer ();
out .println ( "Server port", req. getServer Port ());
out.println ( "Server name", req. getServerName ());
 out.println (" Protocol " , req. getProtocol ());
out.println ("Content Length", req. getContentlength ());
out.println ("Remote address" , req. getRemote Addr());
 out.println (" Scheme ", req. get Scheme ());
```

Enumeration :→

The enumeration interface defines the methods by which you can enumerate (obtain one at a time) the element in a collection of objects.

This legacy interface has been superceded by Iterator. Although not deprecated, Enumeration is considered obsolete for new code. However it is used by several methods defined by the legacy class such as Vector & Properties, is used by several other API classes, and is currently in widespread used in application code.

Methods :- 1) boolean hasMoreElements ()
2). Object nextElement ()

Servlet Config :-

```
<web apps>
 <servlet>
   <servletname> - - -
   <servlet class> - - -
 </servlet>
 <init-param>
      <param_name> driver </ ___ >
      <param_value> sun jdbc odbc . . .
 </init.param>

</web Apps>
```

Config.getInit("driver")

Single Thread Model :-

also known as Marker Interface. This means that implinentation of the service method should be thread safe. The JAVA servlet API specifies the single thread model Interface for this purpose. Servlet can implinent single thread model in order to inform the container that it should make sure that only one thread is executing the servlet service method at any given moment. For this container may follow one of the following approach to ensure that each servlet instance invoked in a seperate run-time.

1). Instance pooling.
2). Request serialization.

1) In this approach the container maintains a pool of servlet instance. for each incoming req. the container allocates a servlet instance from the pool & upon completion of the service the container returns the instance on the pool. But to req. serialization in this approach the container maintains a single instance of the servlet. The container cannot send multiple req. to the instance at the same time. The container serialize the req. this means that new req. will be kept waiting while the current req. is being served.

19/03

Servlet Response :-
- setContentType ()
- getContentType()
- setContentLength(int length)
- getContentLength()
  setBufferSite(int size)
  get ___ ()
  flushBuffer()
  resetBuffer()

Method for error Handeling :
- send error (int statusCode)
- send error (int status, String msg)
- Set status (int status)
- get Status ()

1XX - informational.
100 - specifies (Http status code) that req has been received & non-yet processesed.
101 - specify that the req. protocol is being switched

2XX - Success 200 - OK
201 - created
202 - Accepted
203 - Non-authoritive information
204 - No content
205 - Reset

3XX - Redirection
300 - Multiple choices.
302 - found.
303 - see others.
307 - Temporary re-direct

4XX - Client Error
400 - Bad Req.
401 - unauthorized
403 - forbidden
404 - Not found
405 - Method not allowed
406 - Not acceptable
415 - unsupported media type.
416 - requested range not satisfiable
417 - exception field.

* 5XX - Server Error
500 - Internal Server Error.
501 - Not implimented.
502 - Bad gateway.
503 - service is available
504 - HTTP version not supported.

Servlet Context Interface:-
Servlet context is the environment where servlet runs. The servlet container creates a servlet context, a object that we can use to access information about the servlets environment.

Request Dispatcher:-
A request dispatching allow a servlet or JSP have to dispatch a req. to another servlet which will then will be responsible for any further processing & generating the response.

public void forward (servlet Request res, Servlet Response res)

Servlet Request getRequest Dispatcher (String path/name)

Servlet Content getNamed Dispatcher (String name)
Request Dispatcher dis = req. get Normal dispatcher (/)
dis.forward (req, res).

diff. the following :
Servlet context and servlet config.
HTTP servlet req. & HTTP servlet response.
Generic servlet & HTTP servlet.
do get & do post.
forward & Include method.

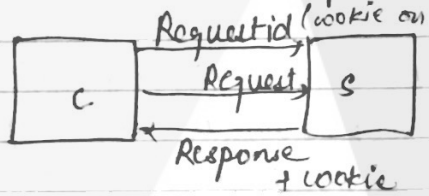
Similar to servlet collaboration.

31/03

** Session Tracking

C ⇄ S

- Cookie.
- Hidden from fields.
- Url rewritting.
- Using servlet API or Http session ~~class~~. interface.



A cookie is a small piece of information i.e. persisted b/w the multiple client req. A cookie has a name a single value & optional attribute such as a comment, path and domain quantifiers, a maximum age and a version no.

setValue & value (int value)
add Cookie ( )
get Cookie ( )

```
Cookie c = new Cookie ("uid" , "abc");
        c. setMaxAge (60 X 60);
        c. setDomain ("my server");
res. addCookie (c);
```

Hidden from field:-

```
<input type = "Hidden" name = "uid" value = "abc">
```

In hidden form field, a hidden text field is used for maintaining the states of user. In such cases we store the information in the hidden field & get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.

Disadvantage:-

1. It maintain at server site.
2. Extra form submission is required on each page.
3. Only textual info. can be used.

Url rewriting:

In this, we append a token or identifier to the URL of the next servlet or the next resource. We can send parameter, name/value pairs using the following format:-

Url ? name = value 1 & name 2 = value 2 & .....

A name and a value is seperated using an equal sign. A parameter name / value pair is seperated from another parameter using &. When the user clicks the hyperlink, the parameter name / value pair will be passed to the server. From a servlet we can use get parameter method to obtain a parameter value.

Advantage :-

1. It will always work w cookie is disabled or not.
2. Extra form submission is not required on each page.

Disadv :-

1. It will work only with links.
2. It can send only textual info.

4) HTTP session Interface :-

getsession ()
get Session (Boolean create)
getServer Id ( )
get Last Accessed Time ( )
get Creation Time ( )

23/03

JSP (Java Server Pages)

Lifecycle same as servlet.

```
<Html>
<body>
Java code --
=
</body>
</Html>
```

javax. Servlet . JSP. * ;

JSP Nuts and Bolts :-

1). Directive.
   - ① Page directive → 11 attributes
   - ② include
   - ③ tag lib. → (user define tags are made only this <Amity>)

2). Scripting Element.
   - ① declaration.
   - ② expression. (to be printed)
   - ③ Scriptlets (logic processing)

3). Standard Action

① JSP: Use Bean
② JSP: Param
③ JSP: forward
④ JSP: include
⑤ JSP: setProperty
⑥ JSP: getProperty

Total 9 implict objects :-

1. Request
2. Response
3. out
4. session
5. Page
6. Page context
7. Application
8. Config.
9. Exception.

Directive :→

JSP directive configure the code generation that container will perform in creating a servlet. They are used to set global values such as class declaration, method to be implemented, output content type etc. & they do not produce any output to the client.

Syntax : <%@ directive name attribute = "value" attribute = "value" %>

1) Page directive

** 11 attributes:

i) language.
ii) extends.
iii) import
iv) session
v) buffer
vi) autoflush
vii) IsThread safe
viii) error Page
ix) Is error Page
x) Content type
xi) Info

session = "true" (session created)
session = "false"

buffer = 8kb, by default.

autoflush = garbage collection

IsThreadSafe → synchronization

error Page

② include :

Syntax :- <%@ include file = "file name" %>

③ tag library :

<%@ taglib Uri = "tag library Uri" prefix = "tag prefix" %>

Uri → that identifies tag library description. A tag library description is used to uniquely name the set of custom tags.

tag prefix → defines the prefix stream

* Scripting elements :-

i) Decleration

<%! Java variable & methods %>

ii) Scriplets : is a block of java code that is executed during the request processing time.

<% java code statement; >

iii) expression :

<% = java exp./statement %>

Example :-

```
<Html>
<Body>
<%! int i=0%>          / decleration
<% i++ %>              / scriptlets
Hello world <%= "value of i is" +i%>   / Expression
</Body>
</Html>
```

•) Servulet context and servulet config. :-

Servulet Config	Servulet Context
•) ServuletConfig available in java.servulet.*; package.	•) Servulet context available in java.servulet.*; package.
•) Servulet Config object is one per servulet class.	•) ServuletContext object is global to entire web application.
•) Object of servuletConfig will be created during initialization process of the servulet.	•) Object of servulet context will be created at the time of web application development.
•) This config. object is public to a particular servulet only.	•) Scope: As long as web application is executing, ServuletContext object will be available, and it will be destroyed once the app^n is removed from the server.
•) Scope: As long as a servulet is executing, servuletConfig object will be available, it will be destroyed once the servulet execution is completed.	

·) Generic Servlet and HTTP servlet :-

javax.servlet.GenericServlet	Java.servlet.HttpServlet
1). GenericServlet defines a generic, protocal-independent servlet.	1). HttpServlet defines a HTTP protocal specific servlet.
2) GenericServlet gives a blueprint and makes writing servlet easier.	2). HttpServlet gives a blueprint for http servlet and makes writing them easier.
3) GenericServlet provides simple versions of the life-cycle methods init and destroy and of the methods in the servletConfig interface.	3). HttpServlet extends the GenericServlet and hence inherits the properties GenericServlet.

·) do get and do post

DoGet	DoPost
1). In doGet Method the parameters are appended to the URL and sent along with header info	·) In doPost, parameters are sent in seperate line in the body.
2). Maximum size of data that can be sent using doget is 240 bytes	·) There is no maximum size for data.
3) Parameters are not encrypted.	·) Parameters are encrypted.

1). Both include() and forward() methods are part of RequestDispatcher interface of servulets API.
2). Both methods accept objects of ServuletRequest and ServuletResponse interface.
3). Both include() and forward() can interact with static and dynamic recource e.g. another Servulet, JSP or HTML files.

1) First and foremost diff. is that include() method

JSP:

Standard actions :-

① < JSP: Usebean >
② < JSP: SetProperty >
③ < JSP: getProperty >
< JSP: Param >
< JSP: forward >
< JSP: include >
< JSP: Plugin >

① Java class (Bean)

SetName()
getName()

< JSP: Usebean. id = "SB1" Scope = "Scope Name"
Class = "beanClassname" />

** Scope of four types:
1). Page
2). Request
3). Session (Specific client)
4). Application

② < JSP. setProperty name = "SB1" property = "*" />
or Object
or Value - id

property = * (for all) property = "property name"

property = "Property name associated value =" "

③ <JSP. getProperty name = " " property = property name />

*** Call Java Bean through Use Bean :-

login. Html

```
< Html >
< Body >
< form method = " Post " action = " beans. JSP " >
< input type = " text " name = " name " >
< Slect name = " language " >
< Option value = " Java " > Java
< option value = " C++ " > C++
< option value =
</ select >
</ form >
</ body >
</ html >
```

```
    this.name = name;
}

public String getName()
{
    return name;
}
public void setlanguage (String language)
{
    this.language = language;
}
public getlanguage()
{
    return language;
}
public string getlanguage comment()
{
if( language.equals ("Java"))
{
return "Java"; } else
if ( language.equals ("C++")).
    {
      return ("Complex"; } else
if ( language.equals ("Perl"))
    {
      return "Simple"; } else
return "Sorry"; }}
```

bean.JSP

```
<html>
<body>
<JSP:Usebean id="language bean" scope="page"
                class="com.languagebean"/>
<JSP:setProperty name="languagebean" property="*"
<JSP:getProperty name="language bean" property="name"/>
                                    property
<JSP:getProperty name="language bean", = "language"/>
<JSP:getProperty name="language bean" property="
                language commet"/>
</body>
</html>
```

properties tell the no. of parameters we have ~~to~~ in our form.

* To add JDBC connectivity to JSP:

* Implicit object
* action
* action attributes.
* bean prog.