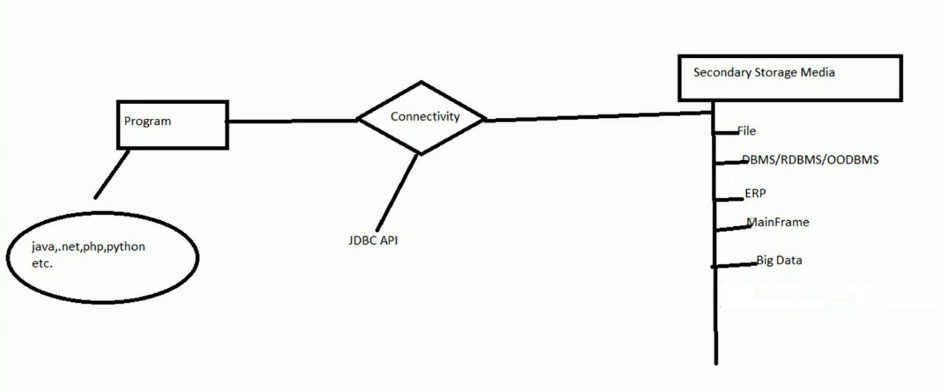
# Database

**Secondary storage media:** file, RDMS, DBMS, RADIS this are secondary storage media. bigdata, mainframe, ERP system.

**connectivity :** The process of connect the program with database is called connectivity.



# Database Connectivity :

There is many database software we use for data store. As like MySQL, oracle, MSSQL. We can use that software by install it. When we install database, all data base provides three facilities for use data.

1. **CUI (Character User Interface):** we use database by CUI in command line interface. we can create, insert, delete we with help CUI.
2. **GUI (graphical user interface):** we can also use database with GUI.
3. **API (application programming interface):** All database provides us API to use database with programming language. This API varies from database to database. We can connect programming language with database using API in three ways ----
4. using vender Specific API.
5. ODBC
6. JDC
7. ORM

### Vender Specific API:

All database have many library which are use for insert, delete, update in database. We use this library for comminicate with program. This library are called vendor specific API. Every database libarbry are different form other. MYSQL API are diffarent from Oracle API. So if we need to impliment diffarent diffarent database with diffarent diffarent vender API.

Problem: if we need multiple database we need know multiple vendor specific API for every database. This is very difficult for user to know all database API.

### ODBC(Open DataBase Connectivity):

Vendor specific API are solve by ODBC concept. ODBC are make by microsoft.ODBC are combination of ODBC driver and ODBC API.

1. **ODBC API:** OBDC collect all database vendor specific API and write it with a C programing language make ODBC API for various database.
2. **ODBC driver :** after make ODBC API implement are also make for program. This implemention called ODBC driver.

### JDBC (Java Data Base Connectivity)

1. JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database.
2. It is a part of JavaSE (Java Standard Edition).
3. JDBC API uses JDBC drivers to connect with the database.
4. JDBC solve platform dependent problem of ODBC.
5. JDBC all method stores in java.sql package.

There are four types of JDBC drivers:

1. JDBC-ODBC Bridge Driver,
2. Native Driver,
3. Network Protocol Driver
4. Thin Driver
5. Type-1) JDBC-ODBC bridge driver: this driver uses ODBC driver to connect to the database. This driver first converts JDBC calls into the ODBC calls. Then ODBC call the Database. this process is not use now.



1. Type-2) Native-API driver: this driver direct called the database specific vendor API. It not uses ODBC driver for connect Database. The driver converts JDBC calls into native calls of the database API. It is not written entirely in java.



1. **Type 3**) Network Protocol driver (**JDBC-Net pure Java**): The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.



1. **Type 4: - Thin driver (Pure Java):** The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.so it is platform independent driver.



# JDBC API

All the JDBC API are found in java.sql package. There many class and interface in JDBC API.

**Interface**: Most of the interface implementation is provide by database with vender API. All the database is own interface. Each database interface is different from other.

1. Interface Driver
2. Interface connection
3. Interface Statement
4. Interface PreparedStatement
5. Interface Resultset()
6. callableStatement
7. SQlException(C)

**Class:**  Class are creating by java program which implement interface which provide the database.

1. Class Driver implement Driver { }
2. Class Drivermanager { }
3. Class connection implement Connection { }
4. Class Statement implement Statement { }
5. Class Resultset implements ResultSet { }
6. callableStatement
7. SQlException(C)
8. **Driver:**
9. “Driver interface” is implement by “Driver Class”. “
10. Driver interface” implementation is providing accordingly different-different venders.
11. Driver convert one language called to other language.
12. It is an interface of database and program.

Code:

interface Driver { };

class Driver implements Driver{

}

|  |  |  |
| --- | --- | --- |
| RDBMS | JDBC driver name | URL format |
| MySQL | com.mysql.jdbc.Driver | jdbc:mysql://hostname/ databaseName |
| ORACLE | oracle.jdbc.driver.OracleDriver | jdbc:oracle:thin:@name:IP:databaseName |
| DB2 | COM.ibm.db2.jdbc.net.DB2Driver | jdbc:db2:hostname:port Number/databaseName |
| Sybase | com.sybase.jdbc.SybDriver | jdbc:sybase:Tds:server:IP/databaseName |

**Mysql:** com.msql.jdbc.Driver.(mysql-driver)

1. **DriverManager class:**
2. Driver manager is a class which do everything for database and programing.
3. Register the drivers with database and java program.
4. De-register the driver.
5. Create connection between java program and database.
6. Return connection.

Code:

class DriverManager{

Driver d;

public void registerDriver(Driver d) {

this.d= d;

}

public Connection getConnection(String url, String user, String pwd) {

Connection c = d.connect(url,user,pwd);

return c;

}

}

**Method of Driver manager:**

* + - 1. registerDriver ()
      2. getConnection ()

1. registerDriver():This method is use for register the driver with database. In this method we cannot change our driver name in runtime. You should use the registerDriver() method if you are using a non-JDK compliant JVM, such as the one provided by Microsoft.

Driver myDriver = new oracle.jdbc.driver.OracleDriver();

DriverManager.registerDriver( myDriver );

1. getConnection (): After the register the driver, we create connection with getConnection() method. we can have overloaded DriverManager.getConnection() methods –
2. getConnection(String url):

String URL = "jdbc:oracle:thin:username/password@amrood:1521:EMP";

Connection conn = DriverManager.getConnection(URL);

1. getConnection (String url, Properties prop):

import java.util.\*;

String URL = "jdbc:oracle:thin:@amrood:1521:EMP";

Properties info = new Properties( );

info.put( "user", "username" );

info.put( "password", "password" );

Connection conn = DriverManager.getConnection(URL, info);

1. getConnection(String url, String user, String password):

String URL = "jdbc:oracle:thin:@amrood:1521:EMP";

String USER = "username";

String PASS = "password"

Connection con = DriverManager.getConnection(URL, USER, PASS);

1. **Connection:**
2. “connection interface” which is implement by connection class.
3. Connection interface established connection with database.
4. Every database connection interface is different.
5. When getConnection method is called, program connect with database. this connection method returns a connection type object which is responsible for create a session between java program and database.

Code:

interface Connection{}

class Connection implements Connection{

public Statement createStatment() {

return new Statement();

}

public PrepareStatement prepareStatement() {

return new prepareStatement();

}

}

Connection con = DriverManager.getConnection(URL, USER, PASS);

**Method of connection class:**

1. *createStatement ():* This method is used for execute SQL queries. This method returns statement class object. This method

Statement stmt = con.createStatement();

1. *prepareStatement(string value ):*This method creates PreparedStatement object which is used to create SQL statement specified in string sql. It is usually used when parameterized SQL statement is to be executed.

PrepareStatement stmt = con.createStatement();

1. close(): this method close the connection of program and database.

Con.close();

1. **Statement:**
2. Statement interface implements by Statement class.
3. Statement class execute the query and return object of Result class which give us value.
4. it provides factory method to get the object of ResultSet.

**methods of Statement interface**

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

ResultSet rs = stmt.executeQuery("select \* from emp");

1. executeUpdate( ): the executeUpdate( ) is used to execute specified query, it may be created, drop, insert, update, delete etc.

ResultSet rs = stmt.executeUpdate ("select \* from emp");

1. execute( ): execute() is used to execute queries that may return multiple results.

ResultSet rs = stmt.execute ("select \* from emp");

1. executeBatch( ): executeBatch( ) is used to execute batch of commands.

ResultSet rs = stmt.executeBatch("select \* from emp");

1. **ResultSet:**
2. resultSet is a interface which implement by resultSet class.
3. The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.
4. The data can be retrieved from the table by using methods of the ResultSet interface.

Code:

interface Result{}

class ResultSet implements ResultSet{

public boolean next() {

}

public boolean previous() {

}

public int getInt(String columnName) {

}

public String getString(int columnIndex) {

}

}

**Method of resultSet:**

1. public boolean next():is used to move the cursor to the one row next from the current position.
2. public boolean previous():is used to move the cursor to the one row previous from the current position.
3. public boolean first():is used to move the cursor to the first row in result set object.
4. public boolean last():is used to move the cursor to the last row in result set object.
5. public int getInt(String columnName): is used to return the data of specified column name of the current row as int.
6. public String getString(int columnIndex): is used to return the data of specified column index of the current row as String.
7. **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.

con.close();

code of sample :

interface Driver{}

interface Connection{}

interface Statement{}

interface PrepareStatment{}

interface Result{}

class Driver implements Driver{

}

class prepareStatment implement prepareStatement{

}

class Connection implements Connection{

public Statement createStatment() {

return new Statement();

}

public void prepareStatement() {

return new prepareStatement();

}

}

class Statment implements Statement{

public Result executeQuery() {

new Result();

}

}

class ResultSet implements ResultSet{

public boolean next() {

}

public boolean previous() {

}

public int getInt(String columnName) {

}

public String getString(int columnIndex) {

}

}

class DriverManager{

Driver d;

public void registerDriver(Driver d) {

this.d= d;

}

public Connection getConnection(String url, String user, String pwd) {

Connection c = d.connect(url,user,pwd);

return c;

}

}

static {

DriverManeger.registerDriver(new com.mysql.jdbc.Driver());

}

# Process of Connection

There are 5 steps to connect any java application with the database using JDBC.

1. Register Driver
2. Create connection
3. Create statement
4. Execute method
5. Close connection
6. **Import JDBC Packages:**

import java.sql.\* ; // for standard JDBC programs

import java.math.\* ; // for BigDecimal and BigInteger support

1. **Register Driver:** To use JDBC we first register the driver with database. For registration we use various method.
2. Class.forName():The most common approach to register a driver is to use Java's

Class.forName() method, to dynamically load the driver's class file into memory, which automatically registers it. We can change the driver name in runtime. This method is preferable because it allows you to make the driver registration configurable and portable.

Class.forName("oracle.jdbc.driver.OracleDriver");

1. DriverManager.registerDriver():Next approach you can use to register a driver, is to use the static DriverManager.registerDriver() method. In this method we cannot change our driver name in runtime. You should use the registerDriver() method if you are using a non-JDK compliant JVM, such as the one provided by Microsoft.

Driver myDriver = new oracle.jdbc.driver.OracleDriver();

DriverManager.registerDriver( myDriver );

1. **getConnection ():** After the register the driver, we create connection with getConnection() method. we can have overloaded DriverManager.getConnection() methods –
2. **getConnection(String url)**
3. **getConnection (String url, Properties prop)**
4. **getConnection(String url, String user, String password):**

Connection con = DriverManager.getConnection(URL, USER, PASS);

1. **Statement():** after connection, statement() object create for query execute.

Statement stmt = con.createStatement();

1. **executeQuery():** with help statement object we execute the query. All data save in Result interface.

ResultSet rs = stmt.executeQuery(“select \* from user”);

1. next(): this method fatch one by one the in a row by row.

While(rs.next()){

System.out.println(rs.getInt())

}

1. **close():** then close the connection of database and program by close() method.

Con.close();

Code :

public class main{

public static void main(String[] args) {

try {

class c = Class.forName("com.mysql.jdbc.Driver");

url = jdbc.mysql://localhost:3306/shuvo";

String name = "root";

String pwd = "roo”;

Connection con = DriverManager.getConnection(url, name, pwd);

} catch (Exception e) {

// TODO: handle exception

}

}

}

# PreparedStatement:

1. A PreparedStatement is a pre-compiled SQL statement. It is a subinterface of Statement.
2. PreparedStatement object provides a feature to execute a parameterized query.
3. When PreparedStatement is created, the SQL query is passed as a parameter.
4. This Prepared Statement contains a pre-compiled SQL query, so when the PreparedStatement is executed, DBMS can just run the query instead of first compiling it.
5. We can use the same PreparedStatement and supply with different parameters at the time of execution.
6. An important advantage of PreparedStatements is that they prevent SQL injection attacks.

Code:

PreparedStatement stmt;

stmt = con.prepareStatement(select \* from students where name = ?);

stmt.setString(2,"Chhavi");

ResultSet rs = s tmt.executeQuery();

**Methods of PreparedStatement interface:**

1. setInt(int paramIndex, int value)
2. setString(int paramIndex, String value)
3. setFloat(int paramIndex, float value)
4. setDouble(int paramIndex, double value)
5. executeUpdate()
6. ResultSet executeQuery()

**code:**

import java.sql.\*;

class InsertPrepared{

public static void main(String args[]){

try{

Class.forName("oracle.jdbc.driver.OracleDriver");

String url =” jdbc:oracle:thin:@localhost:1521:xe”

Connection con = DriverManager.getConnection (url, "system", "oracle" );

PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?,?)");

stmt.setInt(1,101);

stmt.setString(2,"Ratan");

stmt.setDouble(3,5.6);

int value =stmt.executeUpdate();

System.out.println( value +" records inserted");

con.close();

}catch(Exception e){

 System.out.println(e);

}

}

}

# CallableStatement :

# Web

### Web Application:

A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter, etc. and other elements such as HTML, CSS, and JavaScript. The web components typically execute in Web Server and respond to the HTTP request.

Web Terminology:

1. Web page
2. Website
3. HTTP
4. HTTP Requests
5. Response
6. Get vs Post
7. Container
8. Server
9. Content Type
10. **Webpage:** it is a kind of internet program which is store in server for running continuously. it is collection hyperlink, hypermedia, text, audio, video, image etc. HMTL, CSS, JavaScript.

Type of webpage:

1. static web page: static web page is those which are already store on server and provide same response to the different user. HTML, CSS, JAVASCRIPT.
2. dynamic web page: Are those web which are not already store on server and provided different response to the different user. Java, PHP, .net
3. **website:** collection of webpages is called website. This two type –

**static web site:**

dynamic web site:

**Web Server:** It is also known as HTTP Server, it can handle HTTP Requests send by client and responds the request with an HTTP Response.

**Web Container:** Also known as Servlet Container and Servlet Engine. It is a part of Web Server that interacts with Servlets. This is the main component of Web Server that manages the life cycle of Servlets.

### Server Side Programming:

1. **CGI (Common Gateway Interface):** CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.



**Disadvantages of CGI:**

1. If the number of clients increases, it takes more time for sending the response.
2. For each request, it starts a process, and the web server is limited to start processes.
3. It uses platform dependent language e.g. C, C++, perl.
4. **Servlet:** Java Servlets are programs that run on a Web or Application server and act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.



**Advantages of Servlet:**

1. The web container creates threads for handling the multiple requests to the Servlet. Threads have many benefits over the Processes such as they share a common memory area
2. Better performance: because it creates a thread for each request, not process.
3. Portability: because it uses Java language.
4. JVM manages Servlets. we don't need to worry about the memory leak, garbage collection.
5. Secure: because it uses java language.

**Create Servlet Program: (before JEE):**

1. JSDK: to run servlet we need sdk which have all servlet API.
2. Server: for run servlet program we need server.(tomcat, glassfish )
3. Directory Structure: we need folder structure for run program perfectly
4. HTTP:
5. Servlet Engine:

# Servelt and JEE

### JEE:

1. The **Java EE** stands for **Java Enterprise Edition**, which was earlier known as J2EE and is currently known as Jakarta EE.
2. Java SE (Standard Edition) is use for desktop application. But The Java EE provides a platform for developers with enterprise features such as distributed computing and web services.
3. JEE, JSE, JME all java platform which is use for run various application.
4. All the library of JEE is in SDK. So that JEE run in SDK package.

JEE technology:

1. Component
2. Container
3. Server
4. Directory
5. Communication
6. Service
7. **Component:**
8. Web component: web component is those component which is responsible generate dynamic web page. Servlet, JSP.
9. EJB component: it is responsible to add business logic in any web pages.
10. **Container:** container is runtime environment for any web or EJB component.

Container = JVM implementation + JEE API implementation

Type:

1. Web container:
2. EJB container:
3. **Server:** it is an application software which is responsible for providing the service to the client.

Every server should have container that is responsible to run servlet program. Example – apache is a server and tomcat is container.

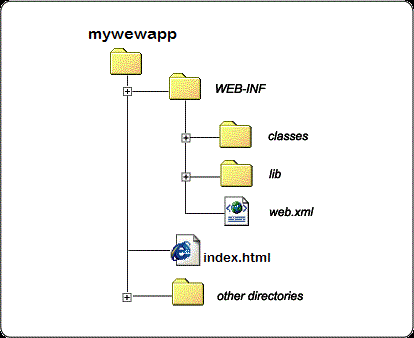
Type of server:

1. Web server: only web container. Tomcat server.
2. Application server: have web container as well as EJB container.
3. **Directory structure:**
4. **Communication:** data transfer protocol. http, ftp, SMTP
5. **Services: J**NTI, Mail service, Massage services. security

# Servlet

Before JEE servlet is server dependent application. One Servlet program run in one server, that Servlet program not run in other server. Because directory structure of every server are different. after come JEE servlet create a standard directory. So that servlet is become server independent. Directory of JEE servlet application.

1. Servlet is java program which run in web container.



### Servlet API

You need to use Servlet API to create servlets. There are two packages that you must remember while using API.

1. **javax.servlet:** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.
2. **javax.servlet.http:** package contains interfaces and classes that are responsible for http requests only.

**Hierarchy of servlet interface:**

Diagram

Description automatically generated

**The servlet interface and class:**

1. Servlet interface,
2. GenericServlet class
3. HttpServlet class

### Servlet interface:

To create any Servlet, you must implement the Servlet interface directly or indirectly. Servlet interface contains the common methods for all servlet

**Method of servlet interface:**

1. **Init() method**: It is used to initialize the servlet. It is called only when the servlet is created, and not called for any user requests afterwards. The init() method does not return within a fixed time set by the webserver. It throws a ServletException. the init() method takes a ServletConfig object as parameter.

Syntax: public void init(ServletConfig config)

1. **service():** The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service () method to handle requests coming from the client( browsers) and to write the formatted response back to the client.

Syntax: public void service (ServletRequest req, ServletResponse res)

1. **destroy():**The destroy() method is called only once at the end of the life cycle of a servlet. closing connection with the database, releasing memory allocated to the servlet, releasing resources that are allocated to the servlet and other cleanup activities.

Syntax: public void destroy()

1. **getServletConfig():** Returns a ServletConfig object, which contains initialization and startup parameters for this servlet.

Syntax: public ServletConfig getServletConfig()

1. **getServletInfo():** Returns information about the servlet, such as author, version, and copyright.

Syntax: public String getServletInfo()

**code:**

import java.io.\*;

import javax.servlet.\*;

public class DemoServlet implements Servlet{

ServletConfig config=null;

public void init(ServletConfig config){

this.config=config;

System.out.println("Initialization complete");

}

public void service(ServletRequest req, ServletResponse res) {

res.setContentType("text/html");

PrintWriter pwriter=res.getWriter();

pwriter.print("<html>");

pwriter.print("<body>");

pwriter.print("<h1>Servlet Example Program</h1>");

pwriter.print("</body>");

pwriter.print("</html>");

}

public void destroy(){

System.out.println("servlet life cycle finished");

}

public ServletConfig getServletConfig(){

return config;

}

public String getServletInfo(){

return "A Demo program written by Chaitanya";

}

}

### GenericServlet class:

1. GenericServlet class implements Servlet, ServletConfig and Serializable interfaces.
2. Generic servlet is a protocol independent Servlet that should always override the service() method to handle the client request
3. GenericServlet class can handle any type of request, so it is protocol independent.
4. You may create a generic servlet by extends the GenericServlet class and providing the implementation of the service method.

**Methods of GenericServlet class:**

1. **init(ServletConfig config):** It is used to initialize the servlet. This method is called only once by the web container when it loads the servlet.

Syntax: public void init(ServletConfig config)throws ServletException

1. **service(ServletRequest reqt,ServletResponse res):** It is used to respond to a request. It is called for every new request by web container.

Syntax: public abstract void service(ServletRequest req,ServletResponse res)throws ServletException, IOException

1. **destroy():** It is used to destroy the servlet. This method is called only once by the web container when all threads of the servlet have exited or in a timeout case.

Syntax: public void destroy()

1. **getServletConfig():** It returns a servlet config object. This config object is passed in init method.  Servlet config object contains initialization parameters and startup configuration for this servlet.

Syntax: public ServletConfig getServletConfig()

1. **getServletInfo():** It returns a string of information about servlet’s author, version, and copyright.

Syntax: public String getServletInfo()

1. **Init():** It is a convenience method which can be overridden so that there is no need to call super.init(config).

Syntax: public void init() throws ServletException

1. **getServletContext():**It returns the ServletContext object in which this servlet is running.

Syntax: public ServletContext getServletContext()

1. **getInitParameter(String name):** It returns the value for given parameter name. It returns null if parameter not exist.

Syntax: public String getInitParameter(String name)

1. **getInitParameterNames():** It returns the names of the servlet’s initialization parameters defined in web.xml file.

Syntax: public Enumeration getInitParameterNames()

1. **getServletName():** It returns the name of this servlet object.

Syntax: public String getServletName()

code:

**import** java.io.\*;

**import** javax.servlet.\*;

**public** **class** First **extends GenericServlet**{

**public** **void** service(ServletRequest req,ServletResponse res)   {

res.setContentType("text/html");

PrintWriter out=res.getWriter();

out.print("<html><body>");

out.print("<b>hello generic servlet</b>");

out.print("</body></html>");

}

}

1. **HttpServlet class**
2. The HttpServlet class extends the GenericServlet class and implements Serializable interface.
3. HTTP Servlet doesn’t override the service() method.
4. Instead it overrides the doGet() method or doPost() method or both.
5. The doGet() method is used for getting the information from server
6. the doPost() method is used for sending information to the server

**Methods of HttpServlet class:**

1. **service(ServletRequest req, ServletResponse res**): dispatches the request to the protected service method by converting the request and response object into http type.
2. **doGet(HttpServletRequest req, HttpServletResponse res):**  handles the GET request. It is invoked by the web container.
3. **doPost(HttpServletRequest req, HttpServletResponse res)** : handles the POST request. It is invoked by the web container.

**code:**

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class ExampleHttpServlet extends HttpServlet {

private String mymsg;

public void init() throws ServletException {

mymsg = "Http Servlet Demo";

}

public void doGet(HttpServletRequest request, HttpServletResponse response)  {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

out.println("<h1>" + mymsg + "</h1>");

out.println("<p>" + "Hello Friends!" + "</p>");

}

public void destroy() {

// Leaving empty. Use this if you want to perform

//something at the end of Servlet life cycle.

}

}

# Life cycle method

1. init()
2. service()
3. destroy()

When the web server (e.g. Apache Tomcat) starts up, the servlet container deploys and loads all the servlets. Once all the Servlet classes loaded, the servlet container creates instances of each servlet class. Servlet container creates only once instance per servlet class and all the requests to the servlet are executed on the same servlet instance.

**init() method:**

The init method is called only once. It is called only when the servlet is created, and not called for any user requests afterwards. When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.

**public** **void** init(ServletConfig config){

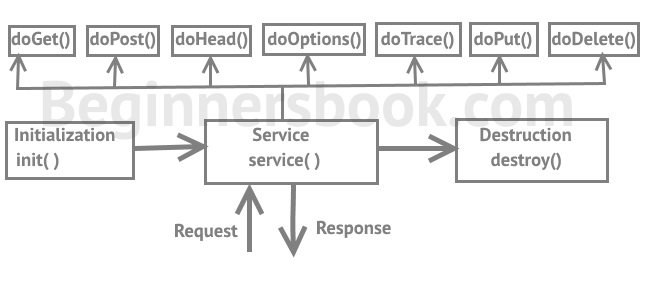
**this**.config=config;

System.out.println("servlet is initialized");

}

**The service() Method:**

The service() method is the main method to perform the actual task. the web container calls the service method each time when request for the servlet is received. Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.



Example:

**public** **void** service(ServletRequest req,ServletResponse res)  **throws** IOException,ServletException{

res.setContentType("text/html");

PrintWriter out=res.getWriter();

out.print("<html><body>");

out.print("<b>hello simple servlet</b>");

out.print("</body></html>");

}

**The destroy() Method:**

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

**public** **void** destroy(){

System.out.println("servlet is destroyed");

}

Web- xml: