UNIT-3

Introduction Servlets: Common Interface (CGI), Lifecycle of a Servlet, deploying a Servlet, The Servlet API Reading Servlet parameters, Reading Initialization parameters, Handling Http Request & Responses, Using Cookies and Sessions, Connecting to a database using JDBC.

Introduction

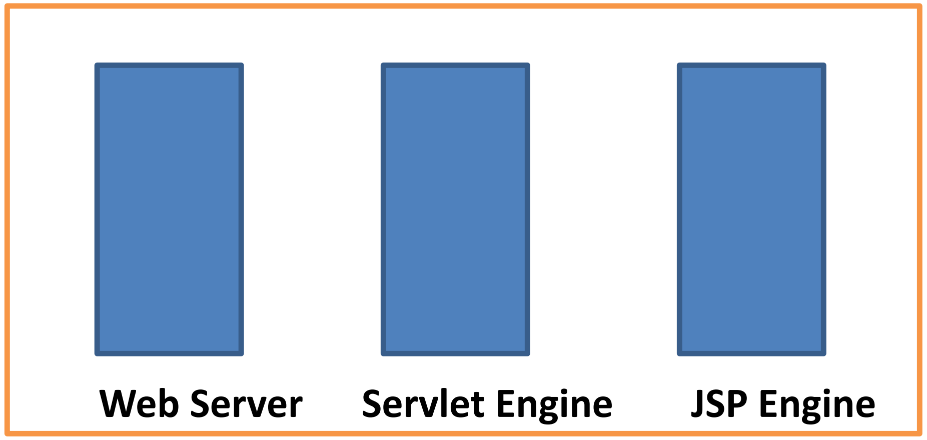
- Servlet is a server side piece of code that generates dynamic web page.
- A servlet is a web component, it is not web application. It is part of web application.
- Servlet is a specialized java class.
- Produces dynamic web pages.
- Contains Java code and HTML code.
- Commonly used with HTTP(Hyper Text Transfer Protocol)- called as HTTP Servlet.
- Servlets make use of Java standard extension classes in the packages javax.servlet and javax,servlet.http

Introduction

- Web Container:

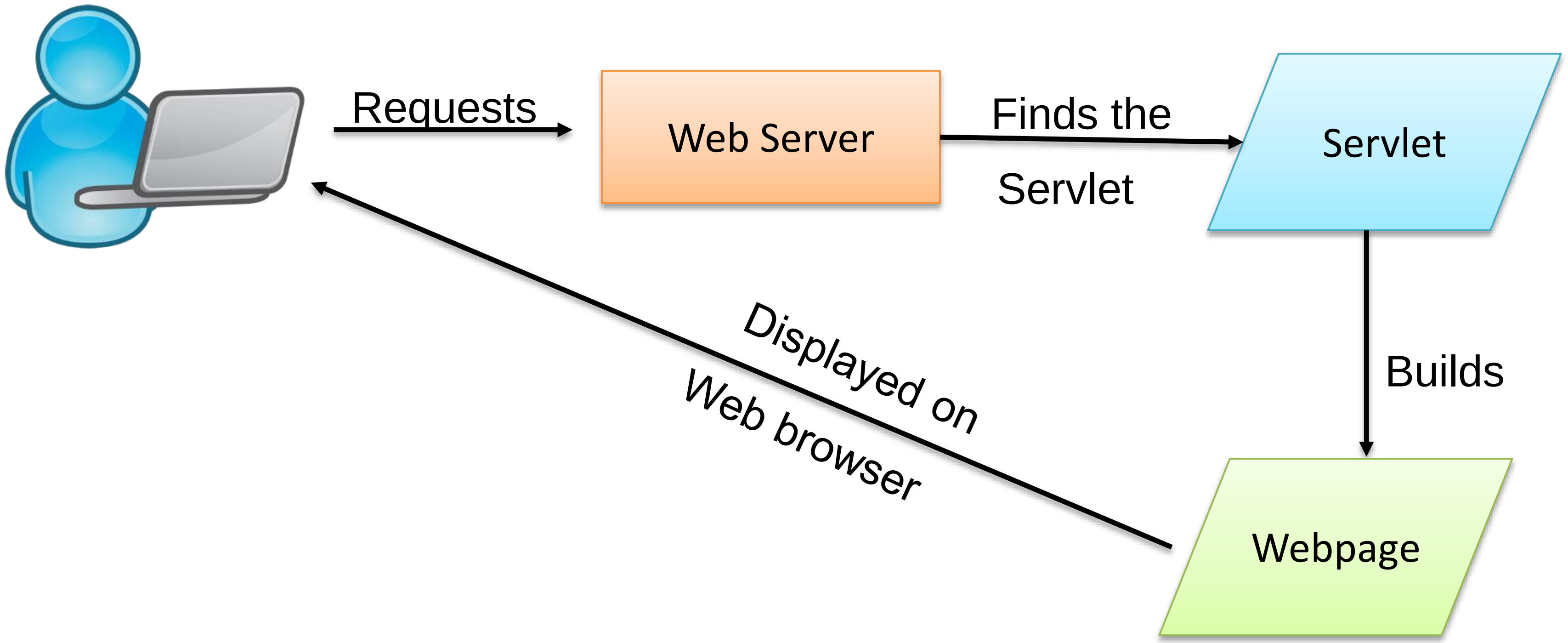
It is a server software written in Java according to Servlets and JSP specifications.

A web container has 3 modules which interact with each other in application development.



Introduction

How Servlet Works ??

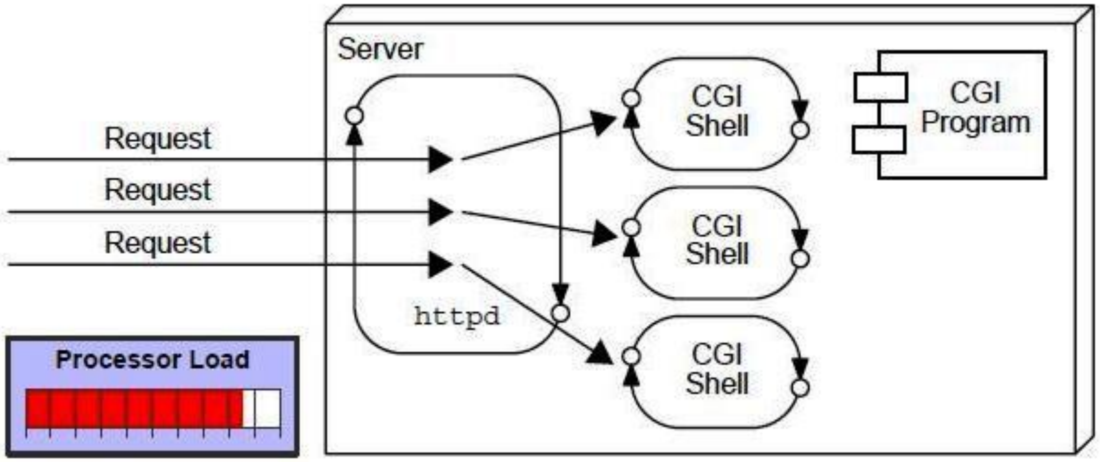


Common Gateway Interface(CGI)

- 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 components such as HTML. The web components typically execute in Web Server and respond to HTTP request.
- In a client-server architecture when the client requests for a static web page to the server then that file is loaded on the server and is returned to he web client and getting on the web browser.
- For CGI applications this scenario is slightly different. When a browser of client sends a request for the data on the server then the server specially the web server locates the CGI program and passes this request to the CGI program. The CGI program decodes the information sent by the client and performs the necessary computations. The output of the CGI program is thensent back to the clients browser.

Common Gateway Interface(CGI)

- 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.

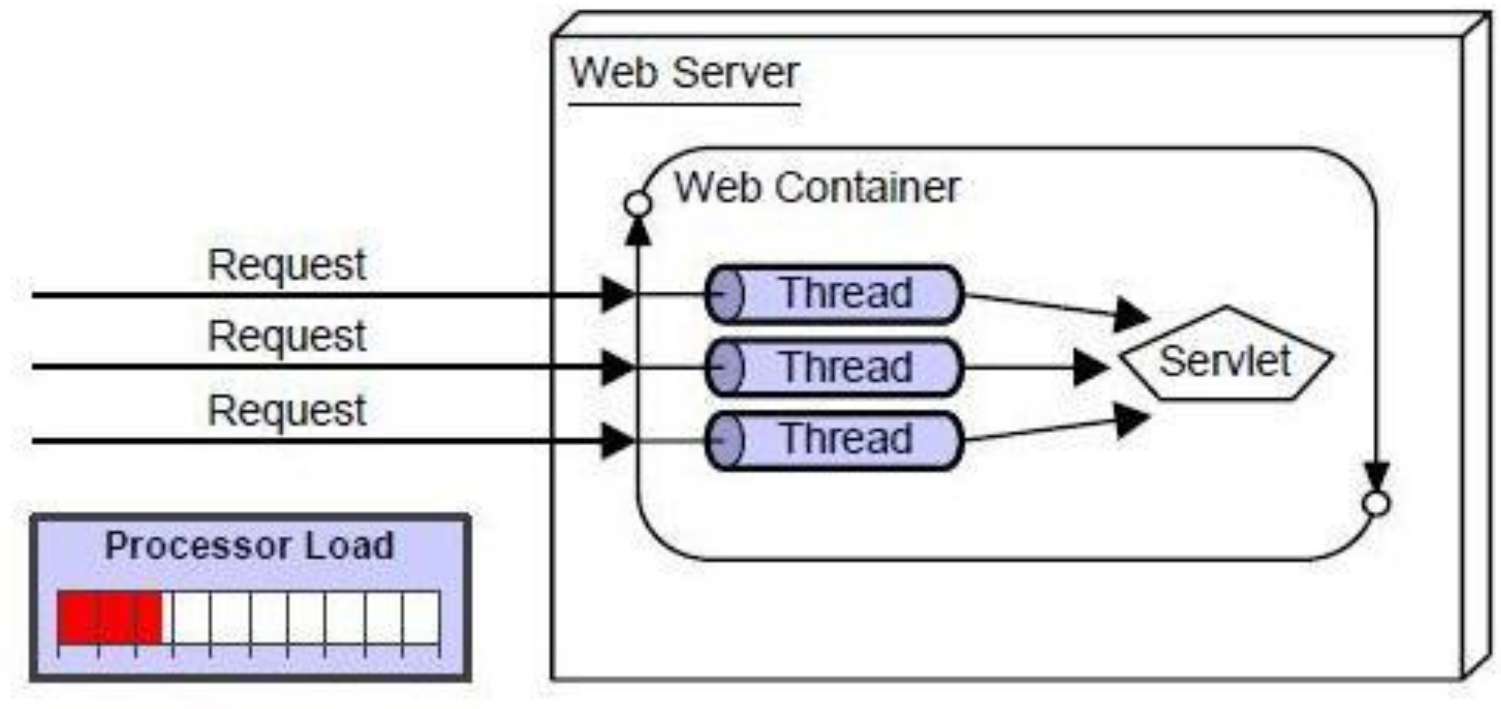


1.For each request, it starts a process and Web server is limited to start processes.

2.If number of clients increases, it takes more time for sending response. As it needs to be loaded and started on each request.

3.It uses platform dependent language e.g. C, C++, perl.

Common Gateway Interface(CGI)



Common Gateway Interface(CGI)

There are many advantages of servlet over CGI. The web container creates threads for handling the multiple requests to the servlet. Threads have a lot of benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The basic benefits of servlet are as follows:

- Better performance: because it creates a thread for each request not process.
- Portability: because it uses java language.
- Robust: Servlets are managed by JVM so we don't need to worry about memory leak, garbage collection etc.
- Secure: because it uses java language..

Lifecycle of a Servlet

Servlet life cycle is described by 3-life cycle methods and 4-life cycle phases.

3 life cycle methods are

init()

service()

destroy() → servlet engine calls them at appropriate time.

- ❑ Servlet engine calls init() soon after it has created the Servlet Instance. (When the servlet is loaded in the memory for the first time). It is called only once implicitly by the Servlet Engine.
- ❑ Just before the servlet instance is destroyed (garbage collected), servlet engine calls destroy() method. It is invoked implicitly (automatically) by the servlet engine only once on the servlet instance. (server unloads the servlet from the memory).

Lifecycle of a Servlet

- Each time client request comes, servlet engine calls the service(). It is called ‘n’ times implicitly by the servlet engine on servlet instance.
- **Skeleton:**

```
public class OurServlet extends GenericServlet
{
        public void init(ServletConfig config)
        {
                //Resource allocation
        }
        public void service(ServletRequest request,ServletResponse response)
        {
                //Client request processing
        }
        public void destroy()
        {
                //Resource release
```

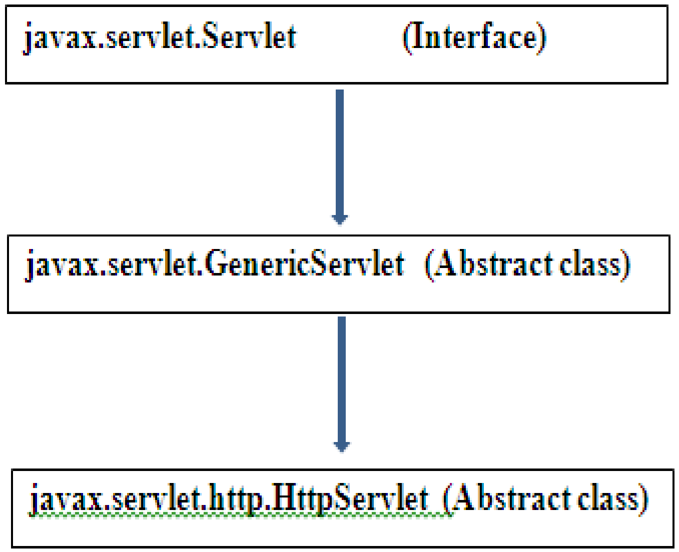
Lifecycle of a Servlet

To write a Servlet three ways exist.

a) by implementing Servlet (it is interface)

b) by extending GenericServlet (it is abstract class)

c) by extending HttpServlet (it is abstract class)

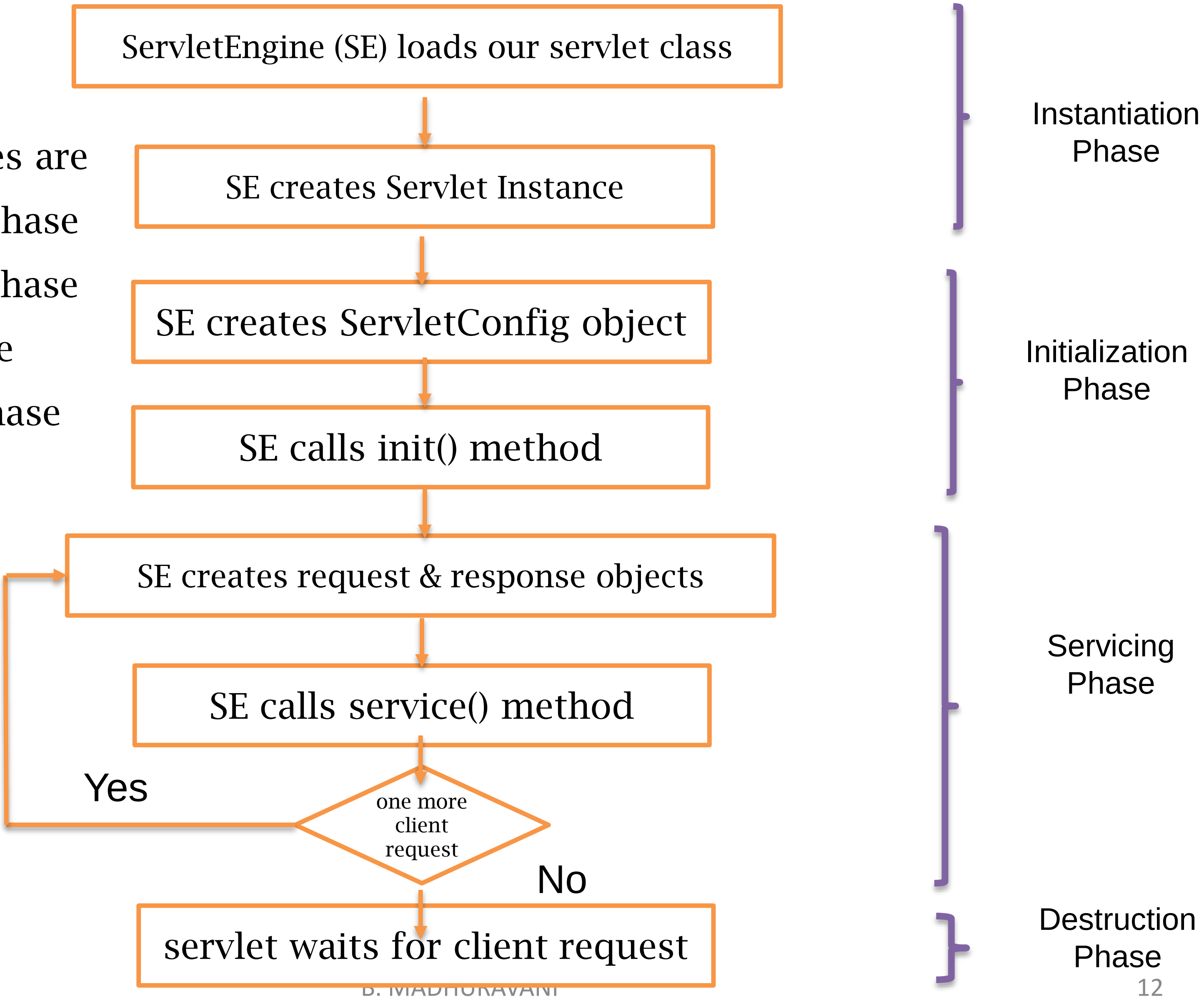

javax.servlet.Servlet (Interface)
javax.servlet.GenericServlet (Abstract class)
javax.servlet.http.HttpServlet (Abstract class)

- Servlet interface contains 5 abstract methods
- GenericServlet contains only one abstract method **service()**
- HttpServlet is abstract class but without any abstract methods. With HttpServlet extension, **service()** method can be replaced by **doGet()** or **doPost()** with the same parameters of service() method.

Lifecycle of a Servlet

4- Life cycle phases are

1. Instantiation Phase
2. Initialization Phase
3. Servicing Phase
4. Destruction Phase



ServletEngine (SE) loads our servlet class
SE creates Servlet Instance
SE creates ServletConfig object
SE calls init() method
SE creates request & response objects
SE calls service() method
one more client request
Yes
No
servlet waits for client request
Instantiation Phase
Initialization Phase
Servicing Phase
Destruction Phase

Lifecycle of a Servlet

Instantiation Phase:

When servlet engine receives first client request, it loads the servlet into memory. Servlet engine loads the servlet instance of the loads servlet. During this phase the servlet instance is missing a) Initialization information b) context information (i.e. environmental information on which field to work on)

Initialization Phase:

Servlet engine creates ServletConfig object which is an interface and calls init() method by supplying ServletConfig object as parameter. Once init() method is executed completely then servlet is ready to serve the client request.

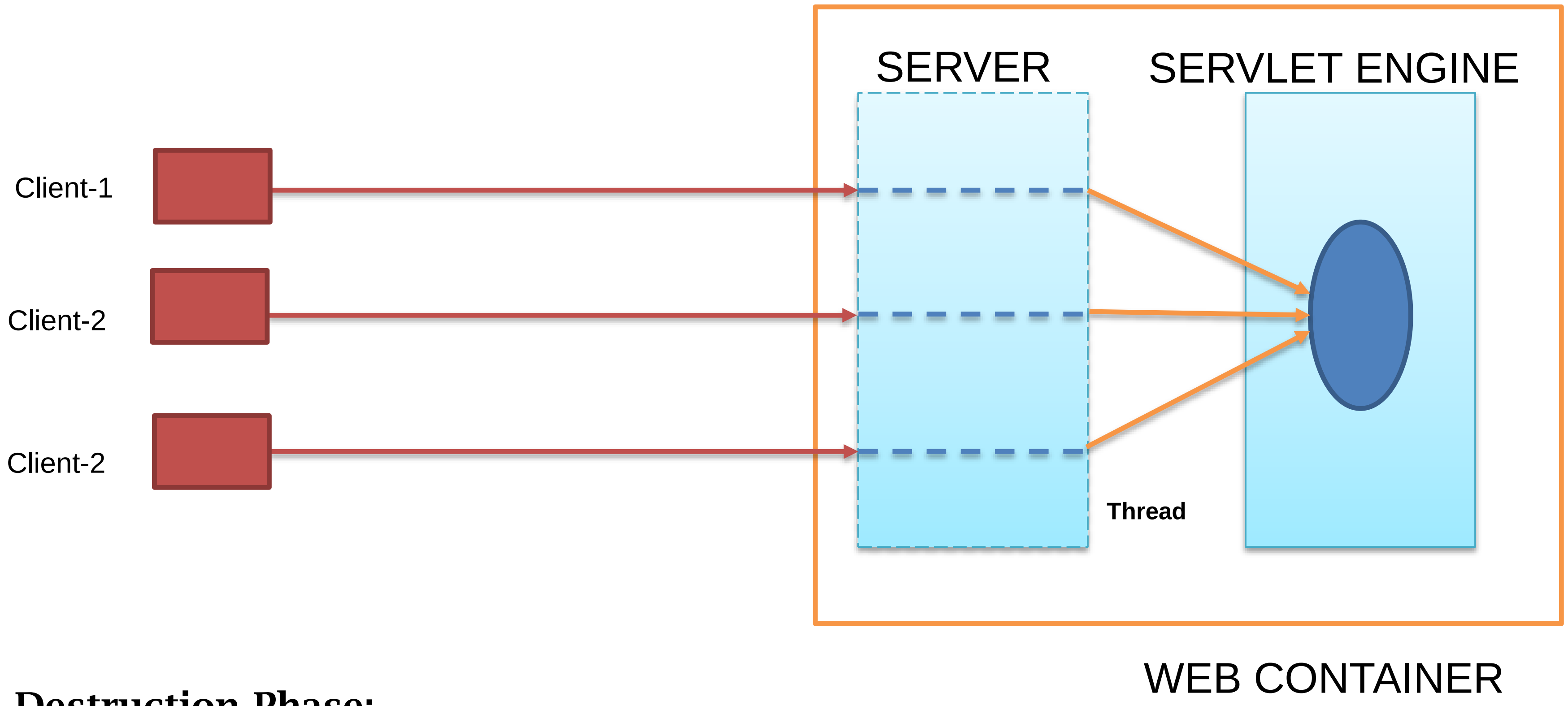
Note: Instantiation and Initialization happens only once in the life cycle of servlet.

Servicing Phase:

The information coming from web server(user input data+client info+server details) is not understandable to servlet. Hence this raw info is given by web server to SE which converts it into the request & response objects. SE calls service() by supplying req & resp objects as arguments.

Lifecycle of a Servlet

Once service() is completely executed, one client request is processed and response is delivered i.e HTTP request-response cycle is completed.



Destruction Phase:

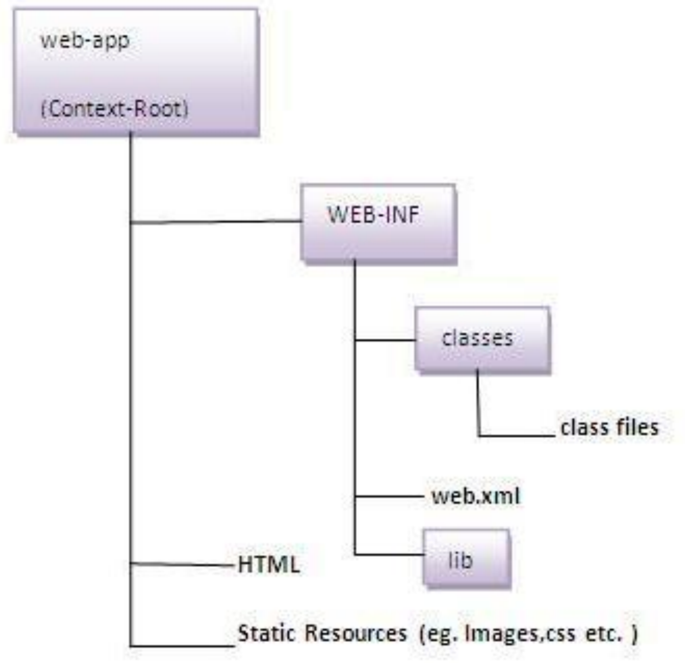
Just before the servlet instance is garbage collected, servlet engine calls destroy() on it.

Deploying a Servlet

1)Create a directory structures

The directory structure defines that where to put the different types of files so that web container may get the information and respond to the client.

The Sun Microsystem defines a unique standard to be followed by all the server vendors.



Deploying a Servlet

2)Create a servlet

There are three ways to create the servlet.

By implementing the Servlet interface

By inheriting the GenericServlet class

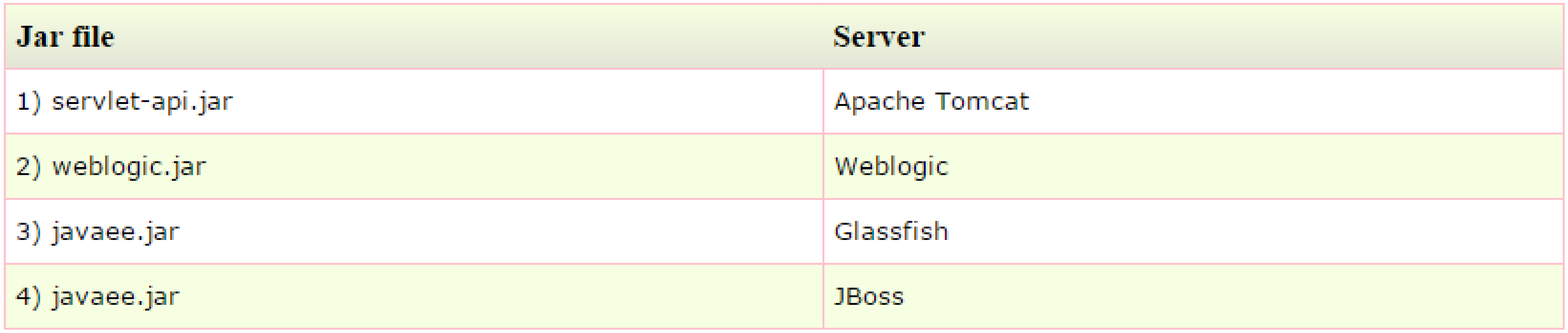
By inheriting the HttpServlet class

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Extend HttpServlet class
public class FirstServlet extends HttpServlet
{
   public void doGet(HttpServletRequest request,HttpServletResponse
response) throws ServletException, IOException
  {
      // Set response content type
      response.setContentType("text/html");
      // Actual logic goes here.
      PrintWriter out = response.getWriter();
      out.println("<html>");
      out.println("<head>");
      out.println("<title>FirstServlet</title>");
      out.println("<body>");
      out.println("<h1>Welcome to servlet</h1>");
      out.println("</body>");
      out.println("</html>");
```

Deploying a Servlet

3)Compile the servlet

For compiling the Servlet, jar file is required to be loaded. Different Servers provide different jar files:



Jar file	Server
1) servlet-api.jar	Apache Tomcat
2) weblogic.jar	Weblogic
3) javaee.jar	Glassfish
4) javaee.jar	JBoss

Two ways to load the jar file

1. set classpath
2. paste the jar file in JRE/lib/ext folder

Put the java file in any folder. After compiling the java file, paste the class file of servlet in WEB-INF/classes directory.

Deploying a Servlet

4)Create the deployment descriptor (web.xml file)

The deployment descriptor is an xml file, from which Web Container gets the information about the servet to be invoked.

The web container uses the Parser to get the information from the web.xml file. There are many xml parsers such as SAX, DOM and Pull.

There are many elements in the web.xml file. Here is given some necessary elements to run the simple servlet program.

Deploying a Servlet

(web.xml file)

```
<web-app>

<servlet>
<servlet-name>FirstServlet</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>FirstServlet</servlet-name>
<url-pattern>/servlet/FirstServlet</url-pattern>
</servlet-mapping>

</web-app>
```

Deploying a Servlet

5)Start the Server and deploy the project

To start Apache Tomcat server, double click on the startup.bat file under apache-tomcat/bin directory.

One Time Configuration for Apache Tomcat Server

You need to perform following tasks:

1. Set CATALINA_HOME in environmental variable:

C:\Program Files\Apache Software Foundation\Tomcat 7.0

2. Set JAVA_HOME in environmental variable:

C:\Program Files\Java\jdk1.7.0_06

3. Set CLASSPATH

C:\Program Files\Apache Software Foundation\Tomcat 7.0\lib\servlet-api.jar;

Deploying a Servlet

6) How to access the servlet

Open browser and write http://hostname:portno/contextroot/urlpatternofservlet.

For example:

http://localhost:8080/sample/servlet/FirstServlet

The Servlet API

The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api.

The javax.servlet package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.

The javax.servlet.http package contains interfaces and classes that are responsible for http requests only.

The Servlet API

Interfaces in javax.servlet package

There are many interfaces in javax.servlet package. They are as follows:

1. Servlet
2. ServletRequest
3. ServletResponse
4. RequestDispatcher
5. ServletConfig
6. ServletContext
7. SingleThreadModel
8. Filter
9. FilterConfig
10. FilterChain
11. ServletRequestListener
12. ServletRequestAttributeListener
13. ServletContextListener
14. ServletContextAttributeListener

The Servlet API

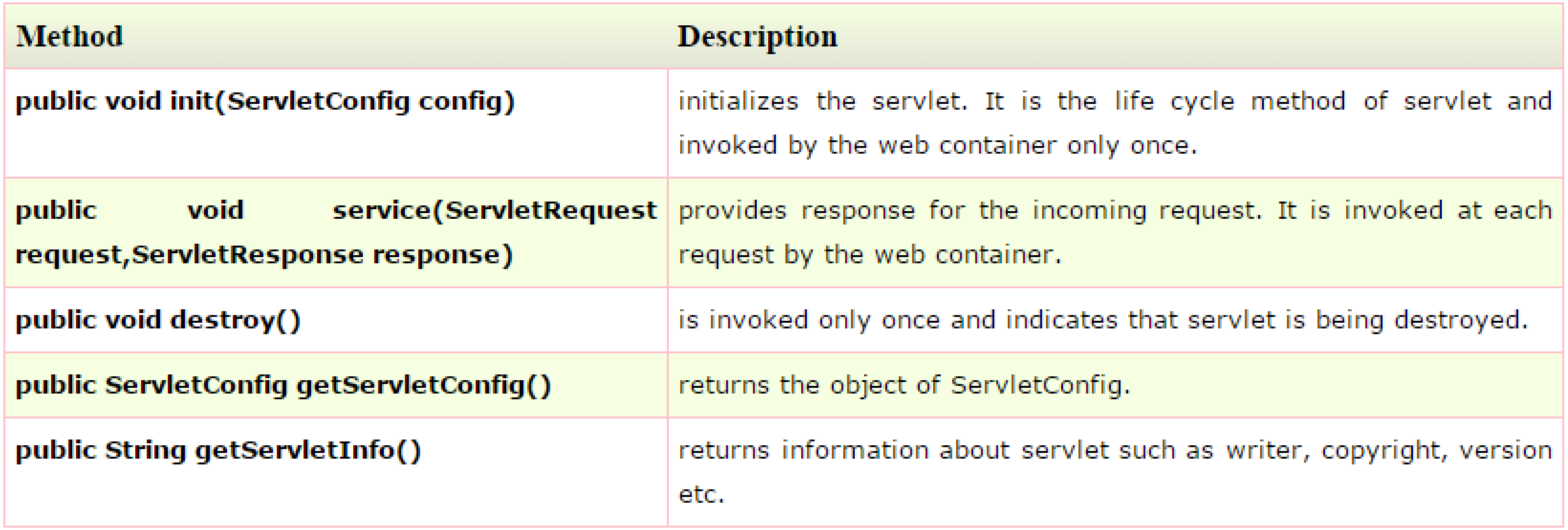
Servlet Interface

Servlet interface provides common behavior to all the servlets.

Servlet interface needs to be implemented for creating any servlet (either directly or indirectly).

It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods.

There are 5 methods in Servlet interface. The init, service and destroy are the life cycle methods of servlet. These are invoked by the web container.



Method	Description
public void init(ServletConfig config)	initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
public void service(ServletRequest request,ServletResponse response)	provides response for the incoming request. It is invoked at each request by the web container.
public void destroy()	is invoked only once and indicates that servlet is being destroyed.
public ServletConfig getServletConfig()	returns the object of ServletConfig.
public String getServletInfo()	returns information about servlet such as writer, copyright, version etc.

The Servlet API

ServletConfig Interface

An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file.

If the configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time.

Advantage of ServletConfig

The core advantage of ServletConfig is that you don't need to edit the servlet file if information is modified from the web.xml file.

The Servlet API

Methods of ServletConfig interface

1.public String getInitParameter(String name): Returns the parameter value for the specified parameter name.

2.public Enumeration getInitParameterNames(): Returns an enumeration of all the initialization parameter names.

3.public String getServletName(): Returns the name of the servlet.

4.public ServletContext getServletContext(): Returns an object of ServletContext.

The Servlet API

ServletContext interface

An object of ServletContext is created by the web container at time of deploying the project. This object can be used to get configuration information from web.xml file. There is only one ServletContext object per web application.

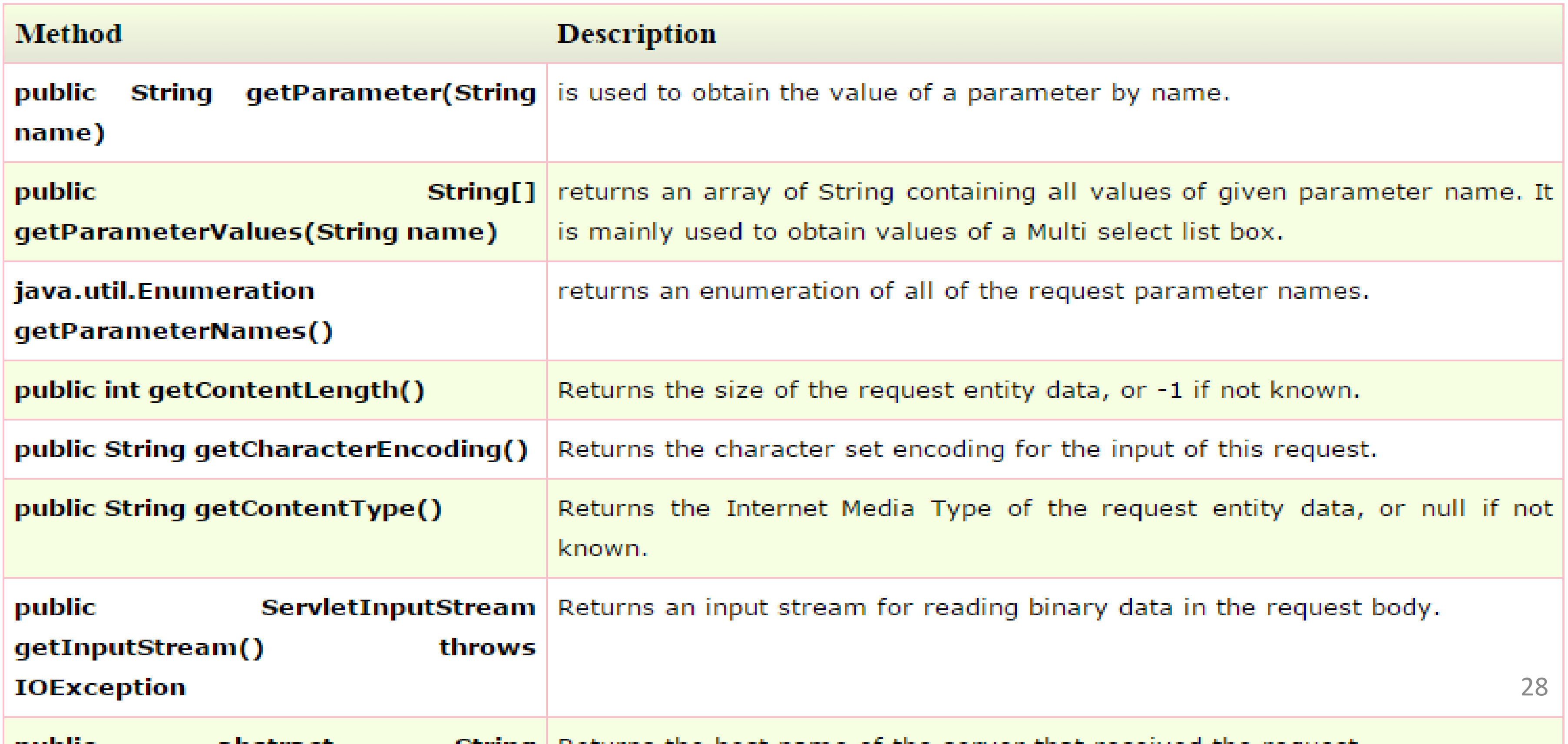
There is given some commonly used methods of ServletContext interface.

public String getInitParameter(String name): Returns the parameter value for the specified parameter name.
public Enumeration getInitParameterNames(): Returns the names of the context's initialization parameters.
public void setAttribute(String name,Object object):sets the given object in the application scope.
public Object getAttribute(String name): Returns the attribute for the specified name.
public Enumeration getInitParameterNames(): Returns the names of the context's initialization parameters as an Enumeration of String objects.
public void removeAttribute(String name): Removes the attribute with the given name from the servlet context.

The Servlet API

ServletRequest interface

An object of ServletRequest is used to provide the client request information to a servlet such as content type, content length, parameter names and values, header informations, attributes etc.



Method	Description
public String getParameter(String name)	is used to obtain the value of a parameter by name.
public String[] getParameterValues(String name)	returns an array of String containing all values of given parameter name. It is mainly used to obtain values of a Multi select list box.
java.util.Enumeration getParameterNames()	returns an enumeration of all of the request parameter names.
public int getContentLength()	Returns the size of the request entity data, or -1 if not known.
public String getCharacterEncoding()	Returns the character set encoding for the input of this request.
public String getContentType()	Returns the Internet Media Type of the request entity data, or null if not known.
public ServletInputStream getInputStream() throws IOException	Returns an input stream for reading binary data in the request body.
public abstract String	Returns the host name of the server that received the request.

The Servlet API

ServletResponse interface

The ServletResponse interface contains various methods that enable a servlet to respond to the client requests.
A servlet can send the response either as character or binary data. The PrintWriter stream can be used to send character data as servlet response, and ServletOutputStream stream to send binary data as servlet response. Here are the methods of ServletResponse interface:

1) java.lang.String getCharacterEncoding(): It returns the name of the MIME charset used in the response sent to the client.

2) java.lang.String getContentType(): It returns the response content type. e.g. text, html etc.

3) ServletOutputStream getOutputStream(): Returns a ServletOutputStream suitable for writing binary data in the response.

4) java.io.PrintWriter getWriter(): Returns the PrintWriter object.

5) void setCharacterEncoding(java.lang.String charset): Set the MIME charset (character encoding) of the response.

6) void setContentLength(int len): It sets the length of the response body.

7) void setContentType(java.lang.String type): Sets the type of the response data.

8) void setBufferSize(int size): Sets the buffer size.

9) int getBufferSize(): Returns the buffer size.

10) void flushBuffer(): Forces any content in the buffer to be written to the client.

11) boolean isCommitted(): Returns a boolean indicating if the response has been committed.

12) void reset(): Clears the data of the buffer along with the headers and status code.

The Servlet API

Classes in javax.servlet package

There are many classes in javax.servlet package. They are as follows:

1.GenericServlet

2.ServletInputStream

3.ServletOutputStream

4.ServletRequestWrapper

5.ServletResponseWrapper

6.ServletRequestEvent

7.ServletContextEvent

8.ServletRequestAttributeEvent

9.ServletContextAttributeEvent

10.ServletException

11.UnavailableException

The Servlet API

GenericServlet class

GenericServlet class implements Servlet, ServletConfig and Serializable interfaces. It provides the implementation of all the methods of these interfaces except the service method.

GenericServlet class can handle any type of request so it is protocol-independent.

The Servlet API

Methods of GenericServlet class

There are many methods in GenericServlet class. They are as follows:

public void init(ServletConfig config) is used to initialize the servlet.

public abstract void service(ServletRequest request, ServletResponse response) provides service for the incoming request. It is invoked at each time when user requests for a servlet.

public void destroy() is invoked only once throughout the life cycle and indicates that servlet is being destroyed.

public ServletConfig getServletConfig() returns the object of ServletConfig.

public String getServletInfo() returns information about servlet such as writer, copyright, version etc.

public void init() it is a convenient method for the servlet programmers, now there is no need to call super.init(config)

public ServletContext getServletContext() returns the object of ServletContext.

public String getInitParameter(String name) returns the parameter value for the given parameter name.

public Enumeration getInitParameterNames() returns all the parameters defined in the web.xml file.

public String getServletName() returns the name of the servlet object.

public void log(String msg) writes the given message in the servlet log file.

public void log(String msg,Throwable t) writes the explanatory message in the servlet log file and a stack trace.

The Servlet API

ServletInputStream class

ServletOutputStream class provides a stream to write binary data into the response. It is an abstract class.
The **getOutputStream()** method of **ServletResponse** interface returns the instance of ServletOutputStream class.
The ServletOutputStream class provides print() and println() methods that are overloaded.

void print(boolean b){}
void print(char c){}
void print(int i){}
void print(long l){}
void print(float f){}
void print(double d){}
void print(String s){}
void println{}
void println(boolean b){}
void println(char c){}
void println(int i){}
void println(long l){}
void println(float f){}
void println(double d){}
void println(String s){}

The Servlet API

Interfaces in javax.servlet.http package

There are many interfaces in javax.servlet.http package. They are as follows:

HttpServletRequest
HttpServletResponse
HttpSession
HttpSessionListener
HttpSessionAttributeListener
HttpSessionBindingListener
HttpSessionActivationListener
HttpSessionContext (deprecated now)

The Servlet API

HttpSession

In such case, container creates a session id for each user.The container uses this id to identify the particular user.An object of HttpSession can be used to perform two tasks:

- bind objects
- view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.

public String getId(): Returns a string containing the unique identifier value.
public long getCreationTime(): Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
public long getLastAccessedTime(): Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
public void invalidate(): Invalidates this session then unbinds any objects bound to it.

The Servlet API

HttpServeltRequest interface:

It is used to obtain the information from clients HTTP request.
1. String getMethod()
2. String getPAthInfo()
3. HttpSession getSession()
4. String getHeader(String fields)
5. Cookie[] getCookies()
6. String getAuthType()

HttpServeltResponse interface:

It is used to formulate an HTTP response to the client.
1. Void add/cookie(Cookie cookie)
2. String encodeURL(String url)
3. Boolean containsHeader(String f)
4. Void sendError(int code)

The Servlet API

Classes in javax.servlet.http package

There are many classes in javax.servlet.http package. They are as follows:

1. HttpServlet
2. Cookie
3. HttpServletRequestWrapper
4. HttpServletResponseWrapper
5. HttpSessionEvent
6. HttpSessionBindingEvent
7. HttpUtils (deprecated now)

The Servlet API

HttpServlet class

The HttpServlet class extends the GenericServlet class and implements Serializable interface. It provides http specific methods such as doGet, doPost, doHead, doTrace etc.

protected void doGet(HttpServletRequest req, HttpServletResponse res) handles the GET request. It is invoked by the web container.
protected void doPost(HttpServletRequest req, HttpServletResponse res) handles the POST request. It is invoked by the web container.
protected void doPut(HttpServletRequest req, HttpServletResponse res) handles the PUT request. It is invoked by the web container.
public void service(ServletRequest req,ServletResponse res) dispatches the request to the protected service method by converting the request and response object into http type.

The Servlet API

HttpSessionEvent class

The HttpSessionEvent is notified when session object is changed. The corresponding Listener interface for this event is HttpSessionListener.

We can perform some operations at this event such as counting total and current logged-in users, maintaing a log of user details such as login time, logout time etc.

There are two methods declared in the HttpSessionListener interface which must be implemented by the servlet programmer to perform some action.

public void sessionCreated(HttpSessionEvent e): is invoked when session object is created.
public void sessionDestroyed(ServletContextEvent e): is invoked when session is invalidated.

Reading Servlet parameters

Servlets read data automatically using the following methods depending on the situation:

getParameter(): to get the value of a form parameter.

getParameterValues(): Call this method if the parameter appears more than once and returns multiple values, for example checkbox.

getParameterNames(): Call this method if you want a complete list of all parameters in the current request.

Reading Servlet parameters

Here is a simple URL which will pass two values to HelloForm program using GET method.

http://localhost:8080/HelloForm?first_name=ZARA&last_name=ALI

Reading Servlet parameters

Get Method

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloForm extends HttpServlet {

   public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
      // Set response content type
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "Using GET Method to Read Form Data";

       out.println( "<html><head><title>" + title + "</title></head>" );
       out.println("<b>First Name</b>: "+ request.getParameter("first_name") );
       out.println("<b>Last Name</b>: "+ request.getParameter("last_name"));
        out.println("</body></html>");
   }
}
```

```
<servlet>
   <servlet-name>HelloForm</servlet-name>
   <servlet-class>HelloForm</servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>HelloForm</servlet-name>
   <url-pattern>/HelloForm</url-pattern>
</servlet-mapping>
```

Now type *http://localhost:8080/HelloForm?first_name=Hello&last_name=World*

Reading Servlet parameters

Post Method

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class GetParameterServlet extends HttpServlet
{
public void doPost(HttpServletRequest req,HttpServletResponse res) throws
IOException,ServletException
{
                res.setContentType("text/html");
                PrintWriter out=res.getWriter();
out.println("<html><head><title>Hello World Servlet</title></head>");
                out.println("<body>");
                String login=req.getParameter("login");
                String password=req.getParameter("password");
out.println("<h2> login:" + login +"<br>password:"+password+"</h2>");
                out.println("<body></html>");
                }
}
```

HelloForm.html

```
<html>
  <body>
    <form action='http://localhost:8080/GetParameterServlet'
method="POST">

              Login: <input type="text" name="login">
      <br />
      Password: <input type="text" name="password" />
      <input type="submit" value="Login" />
    </form>
  </body>
</html>
```

web.xml

```
<servlet>
<servlet-name>GetParameterServlet</servlet-name>
<servlet-class>GetParameterServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>GetParameterServlet</servlet-name>
<url-pattern>/GetParameterServlet</url-pattern>
</servlet-mapping>
```

Reading initialization parameter

Syntax to provide the initialization parameter for a servlet

The init-param sub-element of servlet is used to specify the initialization parameter for a servlet.

```
<web-app>
  <servlet>
    ......

    <init-param>
      <param-name>parametername</param-name>
      <param-value>parametervalue</param-value>
    </init-param>
    ......
  </servlet>
</web-app>
```

Reading initialization parameter

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class DemoServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    ServletConfig config=getServletConfig();
    String driver=config.getInitParameter("driver");
    out.print("Driver is: "+driver);

    out.close();
    }

}
```

```xml
<servlet>
<servlet-name>DemoServlet</servlet-name>
<servlet-class>DemoServlet</servlet-class>

<init-param>
<param-name>driver</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</init-param>

</servlet>

<servlet-mapping>
<servlet-name>DemoServlet</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>
```

Handling Http Request & Responses

When the client make a request to the web server using HTTP protocol, request comes to web server. HttpServlet has some special methos which can be used to handle HTTP requests These are

doDelete()
doGet()
doHead()
doOption()
doPost()
doPut()
doTrace()

For handling the input, the HTTP request makes use of two commonly used methods such as GET and POST.

When the user sumbits his request using doGet() then the URL string displays the request submitted by the user. But if the doPost() is used then URL string does not show the submitted contents.

Handling Http Request & Responses

Handling HTTP-GET Request

Name of the servlet which is to be invoked

```
<html>
<body>
<center><form                    name="/form1"                    method=GET
action=http://localhost:8080/sample/servlet/my_choiceservlet>
<b>Fruit:</b>
<select name="Fruit" size="1">
<option value="Apple">Apple</option>
<option value="Mango">Mango</option>
<option value="Banana">Banana</option>
<option value="Strawberry">Strawberry</option>
</select>
<br><br>
<input type="submit" value="Submit">
</form>
</center>
</body>
</html>
```

Handling Http Request & Responses

Handling HTTP-GET Request
my_choiceservlet.java

```
import java.io.*;
import javax.servlet.http.*;
import java.io.*;
public class my_choiseservlet extends HttpServlet
{
        public void doGet(HttpServletRequest req,HttpServletResponse res)
throws ServletException, IOException
        {
                String Fruit=req.getPArameter(“Fruit”);
                res.setContentType(“text/html);
                PrintWriter out=res.getWriter();
                out.println(“<b>Done!!”);
                out.println(“You have chosen”+Fruit);
                out.close();
        }
}
```

Handling Http Request & Responses

Handling HTTP-PUT Request

Name of the servlet which is to be invoked

```
<html>
<body>
<center><form name="/form1" method=POST
action=http://localhost:8080/sample/servlet/my_choiceservlet1>
<b>Fruit:</b>
<select name="Fruit" size="1">
<option value="Apple">Apple</option>
<option value="Mango">Mango</option>
<option value="Banana">Banana</option>
<option value="Strawberry">Strawberry</option>
</select>
<br><br>
<input type="submit" value="Submit">
</form>
</center>
</body>
</html>
```

Handling Http Request & Responses

Handling HTTP-POST Request
my_choiceservlet1.java

```
import java.io.*;
import javax.servlet.http.*;
import java.io.*;
public class my_choiseservlet1 extends HttpServlet
{
		public void doPost(HttpServletRequest req,HttpServletResponse res)
throws ServletException, IOException
		{
				String Fruit=req.getPArameter(“Fruit”);
				res.setContentType(“text/html);
				PrintWriter out=res.getWriter();
				out.println(“<b>Done!!”);
				out.println(Fruit+”is selcted by you!!”);
				out.close();
		}
}
```

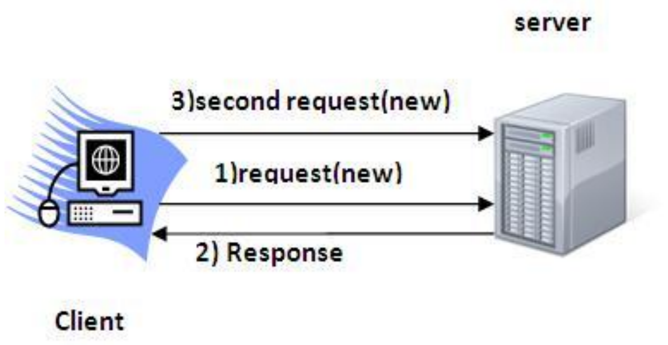
Session Tracking Using Servlet

Session simply means a particular interval of time.

Session Tracking is a way to maintain state (data) of an user. It is also known as session management in servlet.

Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:



Session Tracking Using Servlet

- getSession() is used to create sessions in servlets.
- It returns the object which stores the bindings with names that use this object.
- The bindings can be managed using getAttribute(), setAttribute, removeAttribute().

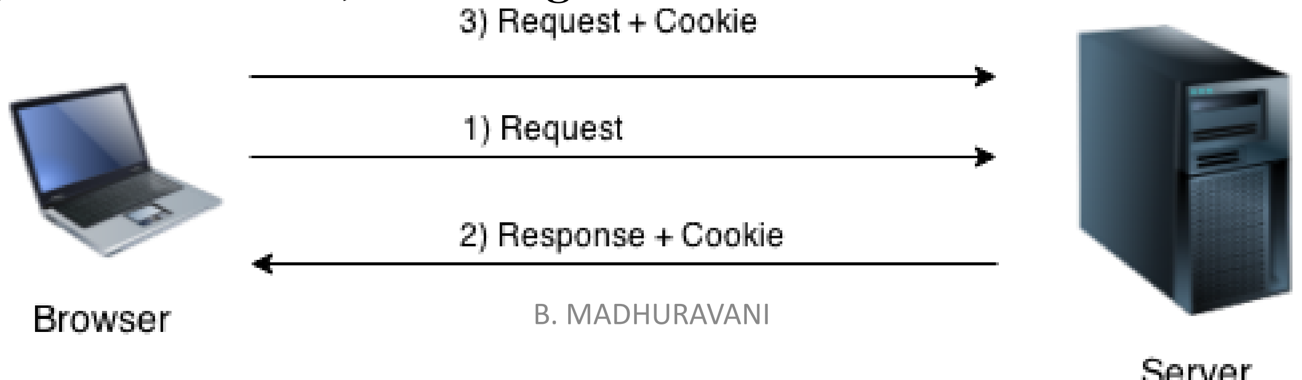
Two things play an important role in session tracking

1. HttpServletRequest interface which supports getSession().
2. HttpSession class which supports binding managing functions such as getAttribute(), setAttribute(), removeAttribute() and geAttributeNames().

SessionsServletDemo.java

Using Cookies

- A cookie is a small piece of information that is persisted between the multiple client requests.
- A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.
- By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.



Using Cookies

Types of Cookie

There are 2 types of cookies in servlets.

- Non-persistent cookie
- Persistent cookie

Non-persistent cookie : It is valid for single session only. It is removed each time when user closes the browser.

Persistent cookie: It is valid for multiple session . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

Advantage of Cookies

- Simplest technique of maintaining the state.
- Cookies are maintained at client side.

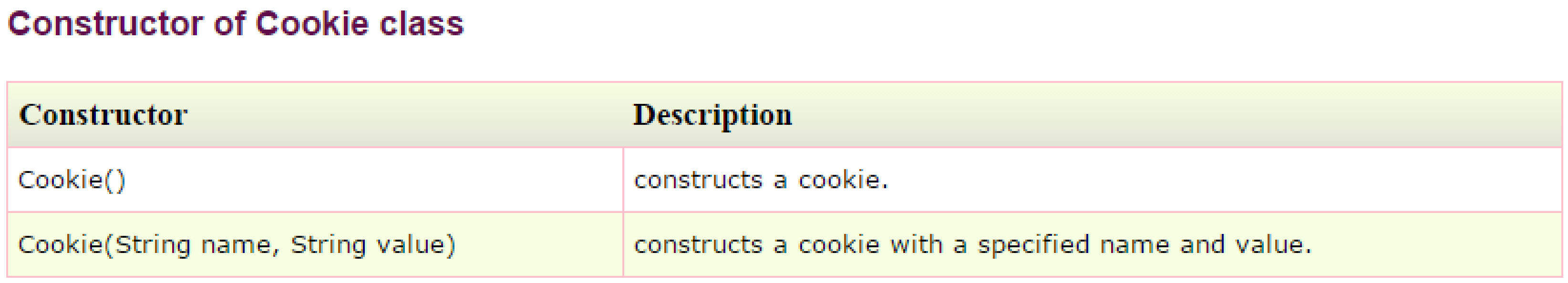
Disadvantage of Cookies

- It will not work if cookie is disabled from the browser.
- Only textual information can be set in Cookie object.

Note: Gmail uses cookie technique for login. If you disable the cookie, gmail won't work.

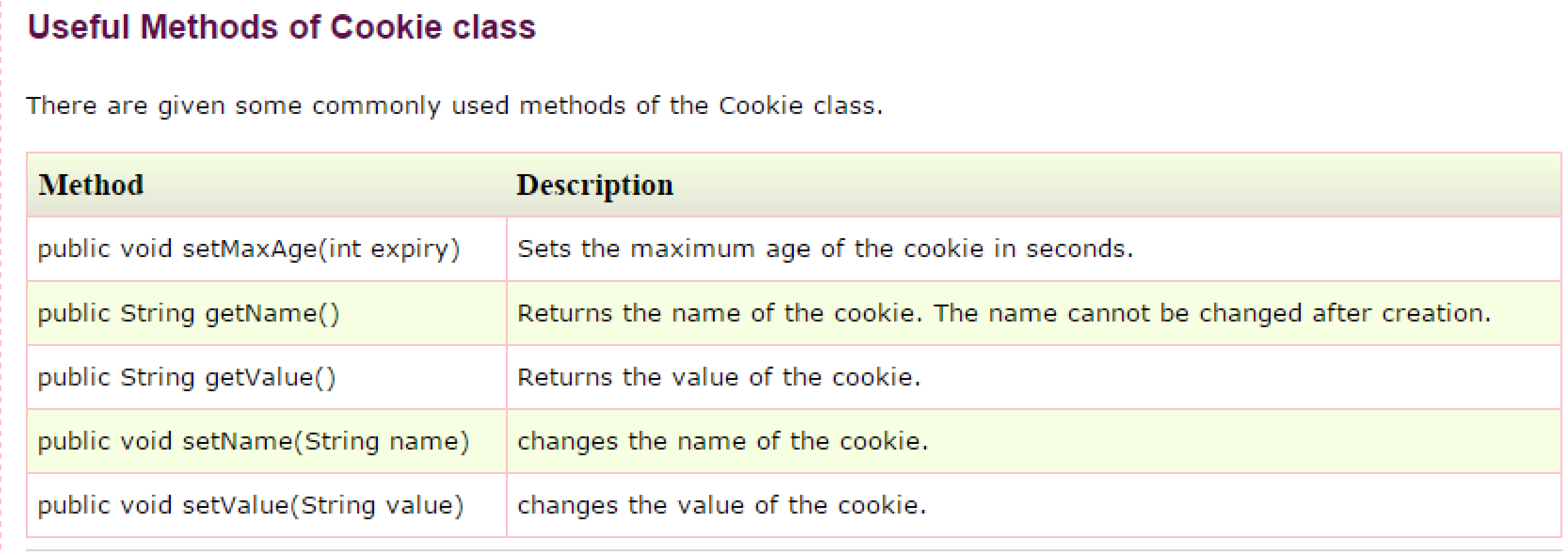
Using Cookies

javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.



Constructor of Cookie class

Constructor	Description
Cookie()	constructs a cookie.
Cookie(String name, String value)	constructs a cookie with a specified name and value.



Useful Methods of Cookie class

There are given some commonly used methods of the Cookie class.

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot be changed after creation.
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void setValue(String value)	changes the value of the cookie.

Using Cookies

For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces.

They are:

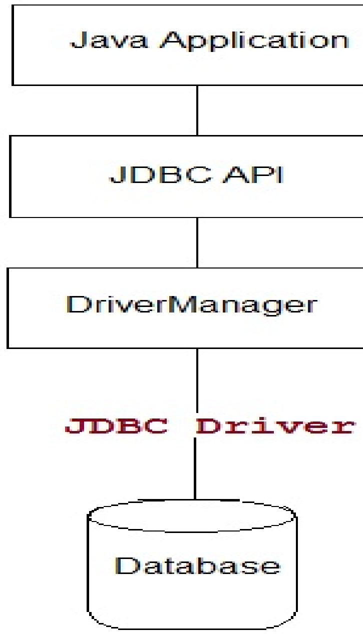
public void addCookie(Cookie ck):method of HttpServletResponse interface is used to add cookie in response object.

public Cookie[] getCookies():method of HttpServletRequest interface is used to return all the cookies from the browser.

Connecting to a database using JDBC

Java Database Connectivity (JDBC) is an Application Programming Interface (API) used to connect Java application with Database. JDBC is used to interact with various types of Databases such as Oracle, MS Access, My SQL and SQL Server**.**

JDBC ARCHITECTURE



Connecting to a database using JDBC

The JDBC architecture consists of two major components:

1. JDBC API
2. JDBC driver types

The JDBC API is a set of classes, interfaces and exceptions used for establishing connection with data source. It is defined in java.sql and javax.sql package.

Driver Manager :
It establishes database connection for the client Java application.
It receives JDBC method calls
Translating method calls into DBMS understandable calls
Receiving results from the database.
Translating the results into java format and passing them to client.

Connecting to a database using JDBC

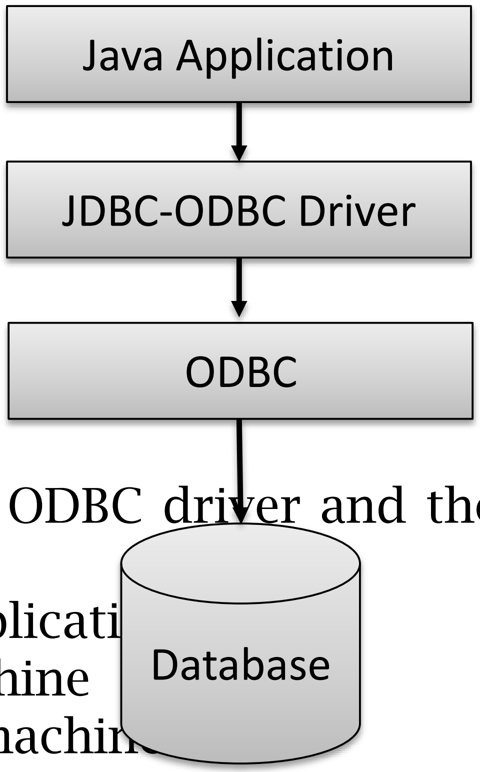
JDBC Driver Types

There are four types of JDBC drivers

1. Type 1:JDBC-ODBC Bridge Driver
2. Type 2:Native-API/Partly Java Driver
3. Type 3:Network Protocol Driver
4. Type 4:Native Protocol Pure Driver / Thin Driver

Connecting to a database using JDBC

Type 1:JDBC-ODBC Bridge Driver



Merit:

Using the JDBC-ODBC bridge access to any database
Is possible.

Demerits:

1. This is slowest driver because the calls are sent to ODBC driver and then to the native database connectivity interface.
2. This type of driver is not suitable for large scale applicati
3. The native database must present on the client machine

And the ODBC driver must be installed on the clients machin

Connecting to a database using JDBC

Type 2:Native-API/Partly Java Driver

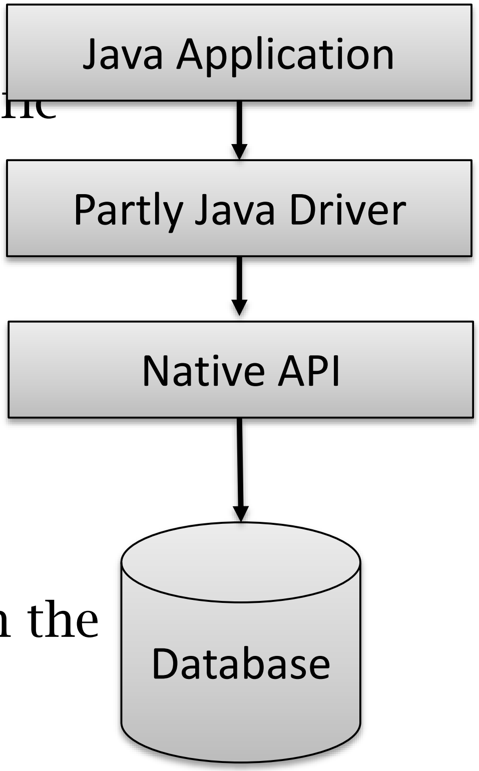
This driver translates all JDBC calls into database-specific
Calls.

Merit:

JDBC calls are directly converted into database-specific
Calls.

Demerits:

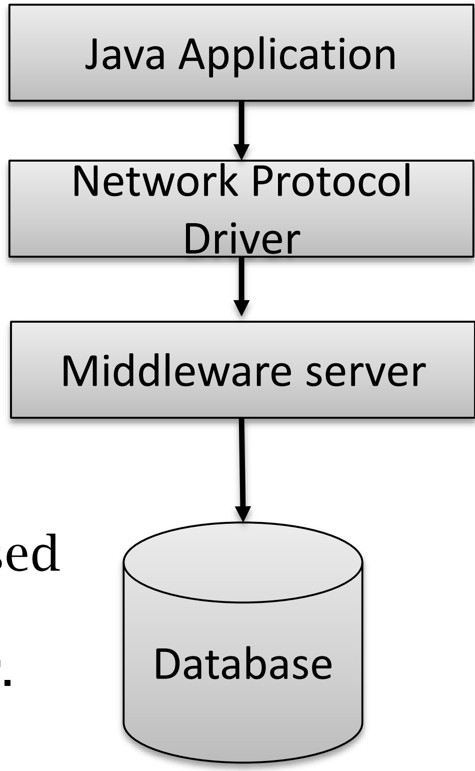
1. The library of required databases must be loaded on the
client machine.
2. It is not useful for the internet.
3. If some modification is made in the database then the native API must also be modified because it is specific to a database.



Connecting to a database using JDBC

Type 3:Network Protocol Driver

all JDBC calls are passed through the network to the middleware server. The middleware server then translates the request to the DB-specific native-connectivity interface and then the request is sent to the DB server.



Merit:

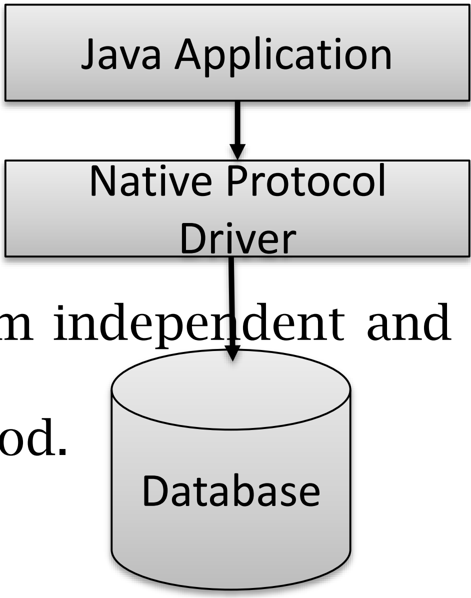
1. As it is server based driver there is no need to keep Library of required databases on the client machine.
2. It is fully written in Java, hence it is portable and can be used On internet.
3. It is possible to access multiple databases using one driver.

Demerits:

The middleware server application needs to be installed and maintained.

Connecting to a database using JDBC

Type 4:Native Protocol Pure Driver / Thin Driver



Clients application directly communicates to the database server.

Merit:

1. As it is completely written in Java, it is portable, platform independent and can be used on internet.
2. There is no translation layer. Hence the performance is good.
3. No need to install specific software on the client machine.

Demerits:

At client side a separate driver is needed for each database.