Unit 2:
Web development using Servlets

Introduction to servlets, Servlet vs CGI, Servlet API overview Servlet Life cycle, Generic servlet, HTTPServlet, ServletConfig, ServletContext, Handling HTTP Request and response –GET /POST method, request dispatching, Using cookies, Session tracking

Introduction to servlets

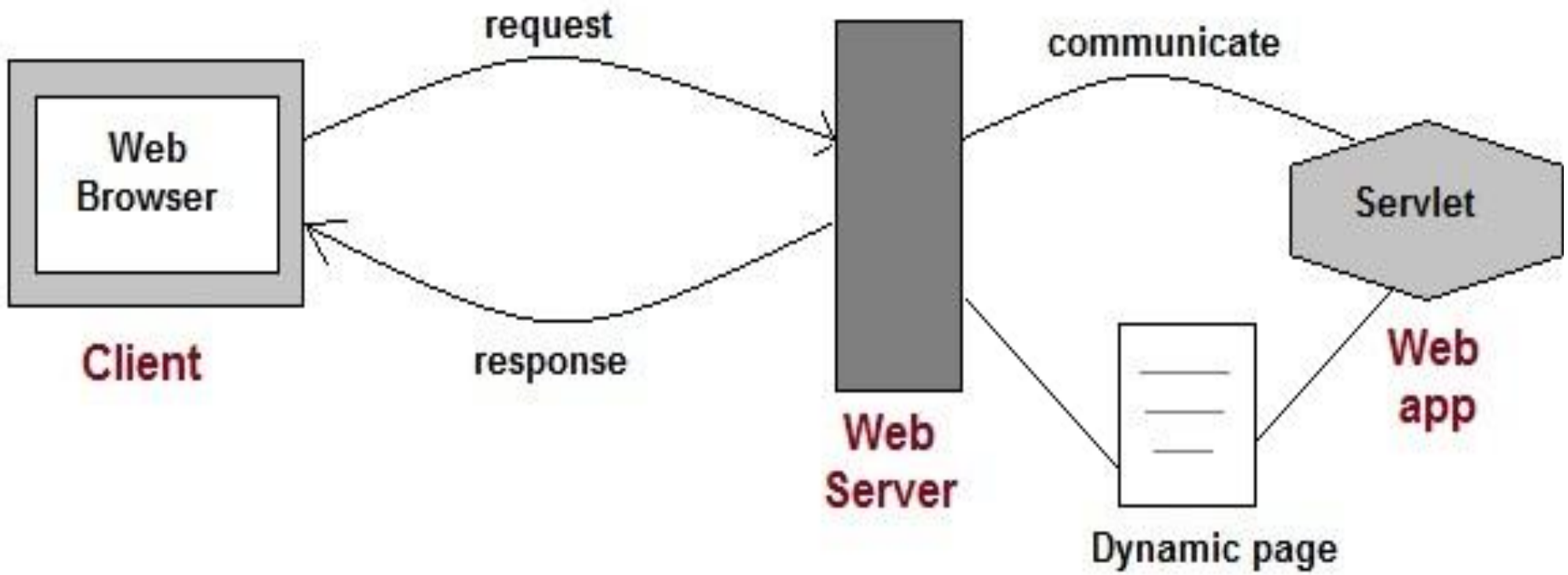
- Servlets are the Java programs that runs on the Java-enabled web server or application server.
- A **servlet** is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model.
- They are used to handle the request obtained from the web server, process the request, produce the response, then send response back to the web server.
- **Servlet** technology is used to create a web application (resides at server side and generates a dynamic web page).
- **Servlet** technology is robust and scalable because of java language.

Properties of Servlets :

- Servlets work on the server-side.
- Servlets capable of handling complex request obtained from web server.

Need For Server-Side Extensions

- The **server-side extensions** are nothing but the technologies that are used to create dynamic Web pages. Actually, to provide the facility of dynamic Web pages, Web pages need a container or Web server. To meet this requirement, independent Web server providers offer some proprietary solutions in the form of **APIs**(Application Programming Interface). These **APIs** allow us to build programs that can run with a Web server.
- Servlets are under the control of another Java application called a ***Servlet Container.***
- When an application running in a web server receives a request, the Server hands the request to the Servlet Container – which in turn passes it to the target Servlet.
- Web applications are a helper application that resides at web server and build dynamic web pages.
- A dynamic page could be anything like a page that randomly chooses picture to display or even a page that displays the current time.

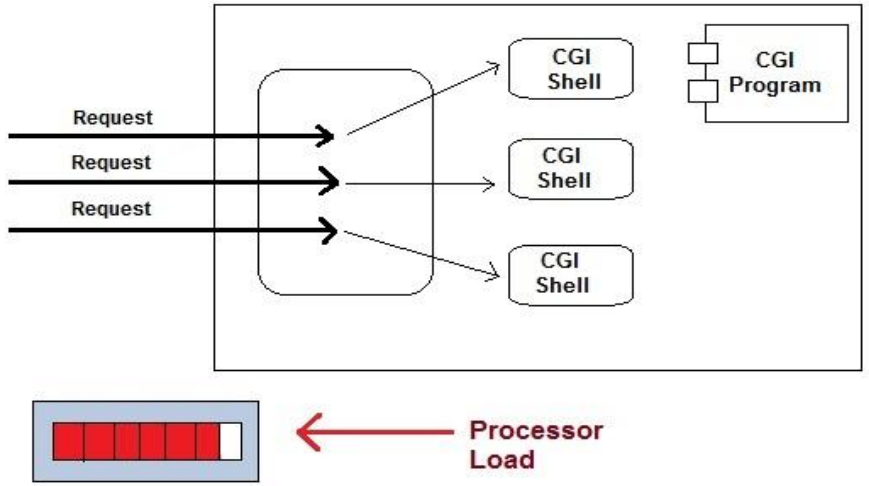

request
communicate
Web
Browser
Servlet
response
Client
Web
Server
Web
app
Dynamic page

CGI (Common Gateway Interface)

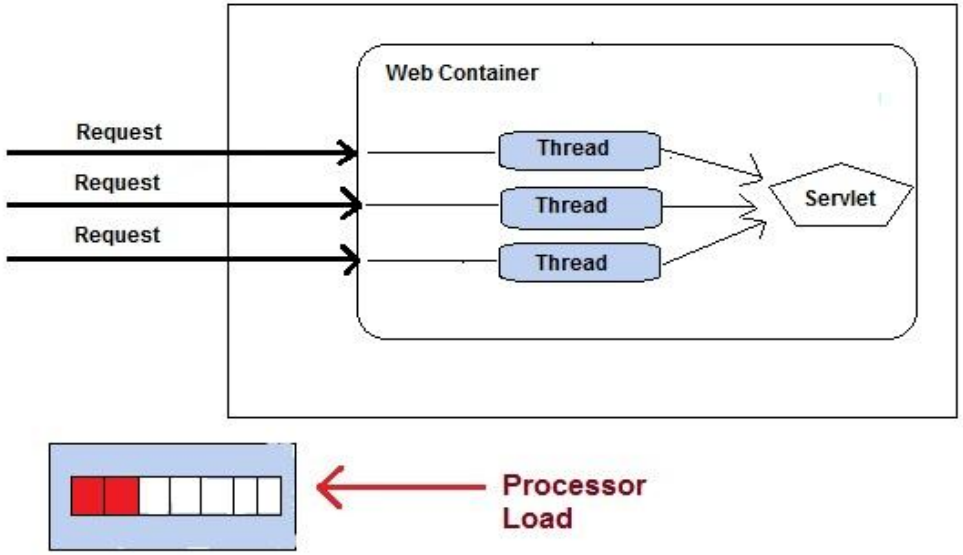
- Before Servlets, CGI(Common Gateway Interface) programming was used to create web applications.
- User clicks a link that has URL to a dynamic page instead of a static page.
- The URL decides which CGI program to execute.
- Web Servers run the CGI program in separate OS shell. The shell includes OS environment and the process to execute code of the CGI program.
- The CGI response is sent back to the Web Server, which wraps the response in an HTTP response and send it back to the web browser.

Servlet vs CGI

- **Drawbacks of CGI programs**
- High response time because CGI programs execute in their own OS shell.
- CGI is not scalable.
- CGI programs are not always secure or object-oriented.
- It is Platform dependent.
- If the number of clients increases, it takes more time for sending the response.
- For each request, it starts a process, and the web server is limited to start processes.
- It uses platform dependent language e.g. C, C++, perl.



- **Advantages of using Servlets**
- Less response time because each request runs in a separate thread.
- Servlets are scalable.
- Servlets are robust and object oriented.
- Servlets are platform independent.
- JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.



Servlets vs. Java Applications

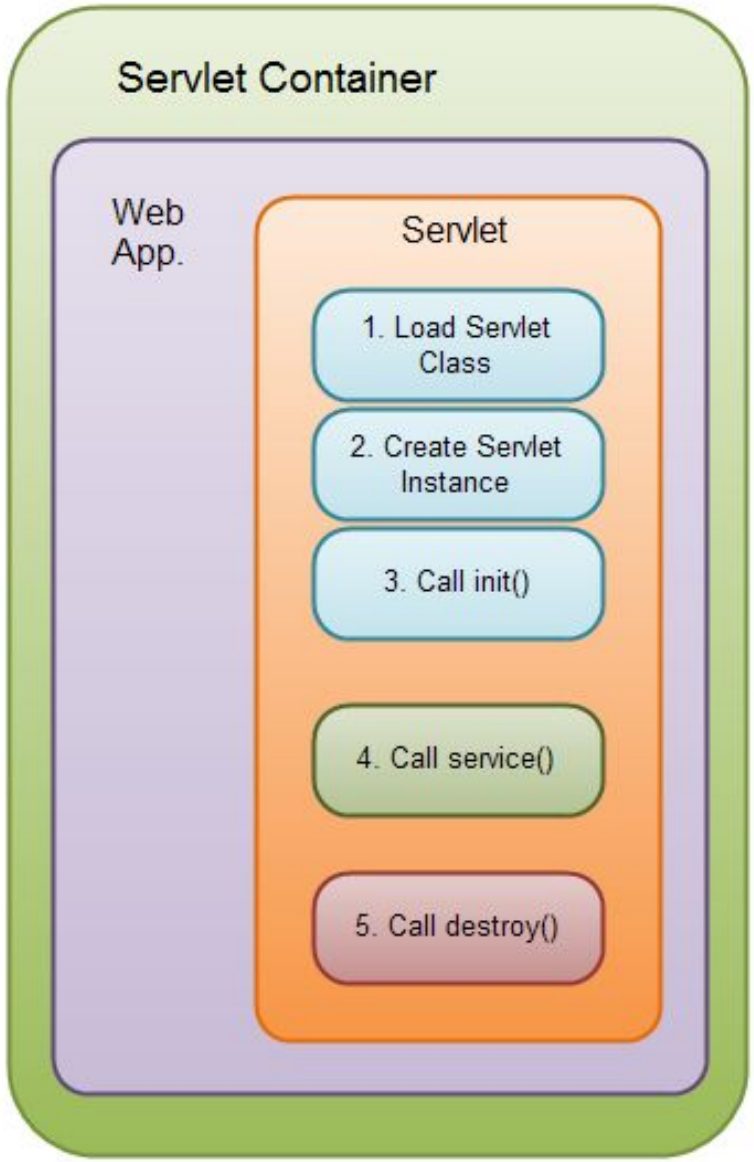
- Servlets do not have a main()
  - The main() is in the server
  - Entry point to servlet code is via call to a method (doGet() in the example)
- Servlet interaction with end user is indirect via request/response object APIs
  - Actual HTTP request/response processing is handled by the server
- Primary servlet output is typically HTML

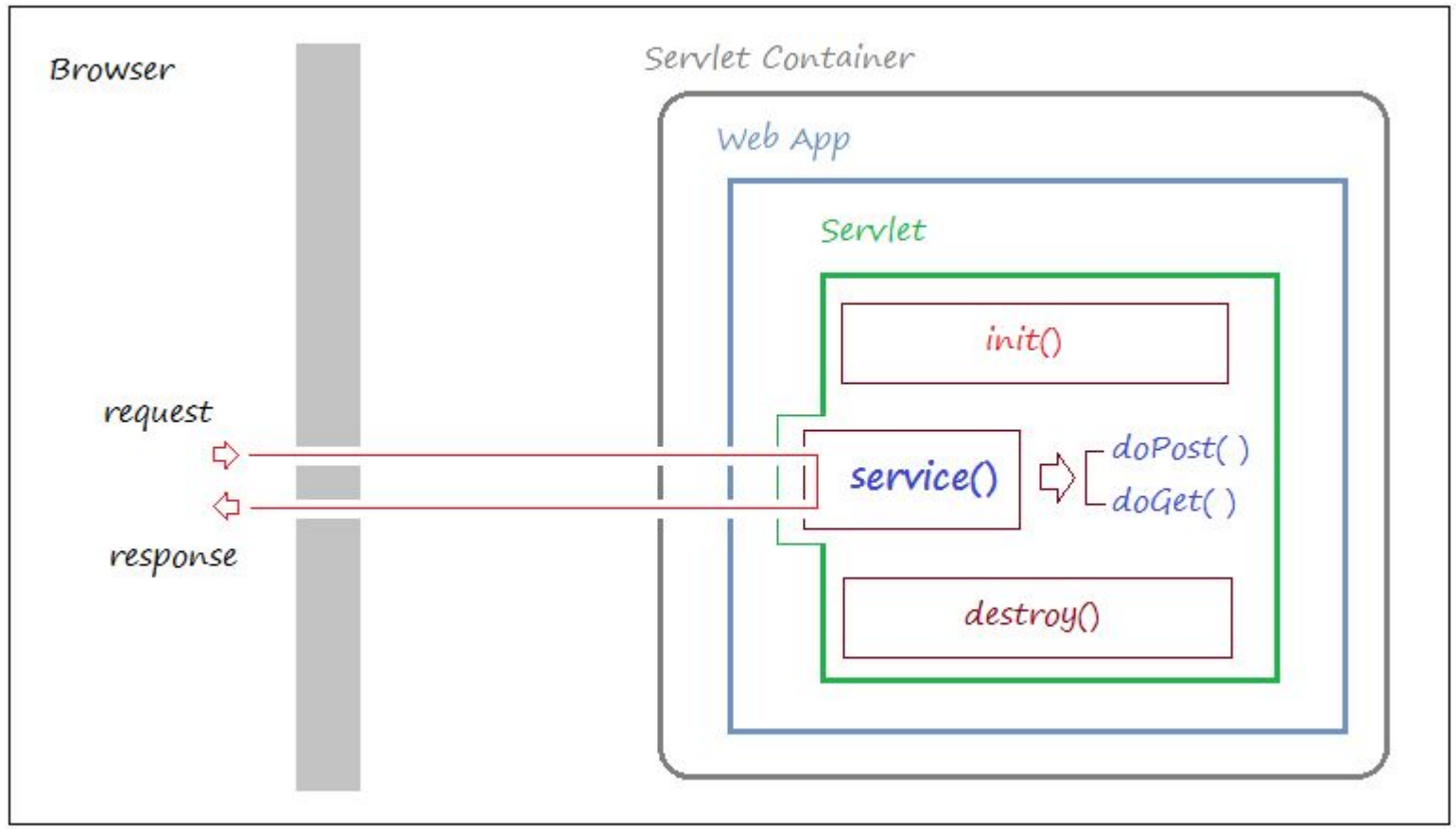
Servlet Life cycle

- Load **Servlet** Class.
- Create Instance of **Servlet**.
- Call the servlets **init()** method.
- Call the servlets **service()** method.
- Call the servlets **destroy()** method.

- Step 1, 2 and 3 are executed only once, when the servlet is initially loaded. By default the servlet is not loaded until the first request is received for it. You can force the container to load the servlet when the container starts up though.
- Step 4 is executed multiple times - once for every HTTP request to the servlet. Step 5 is executed when the servlet container unloads the servlet.

Servlet Life cycle





1. Loading Servlet Class : A Servlet class is loaded when first request for the servlet is received by the Web Container. The init method must complete successfully before the servlet can receive any requests. The servlet container cannot place the servlet into service if the init method either throws a ServletException or does not return within a time period defined by the Web server.
2. Servlet instance creation :After the Servlet class is loaded, Web Container creates the instance of it. Servlet instance is created only once in the life cycle.
3. Call to the init() method : init() method is called by the Web Container on servlet instance to initialize the servlet.

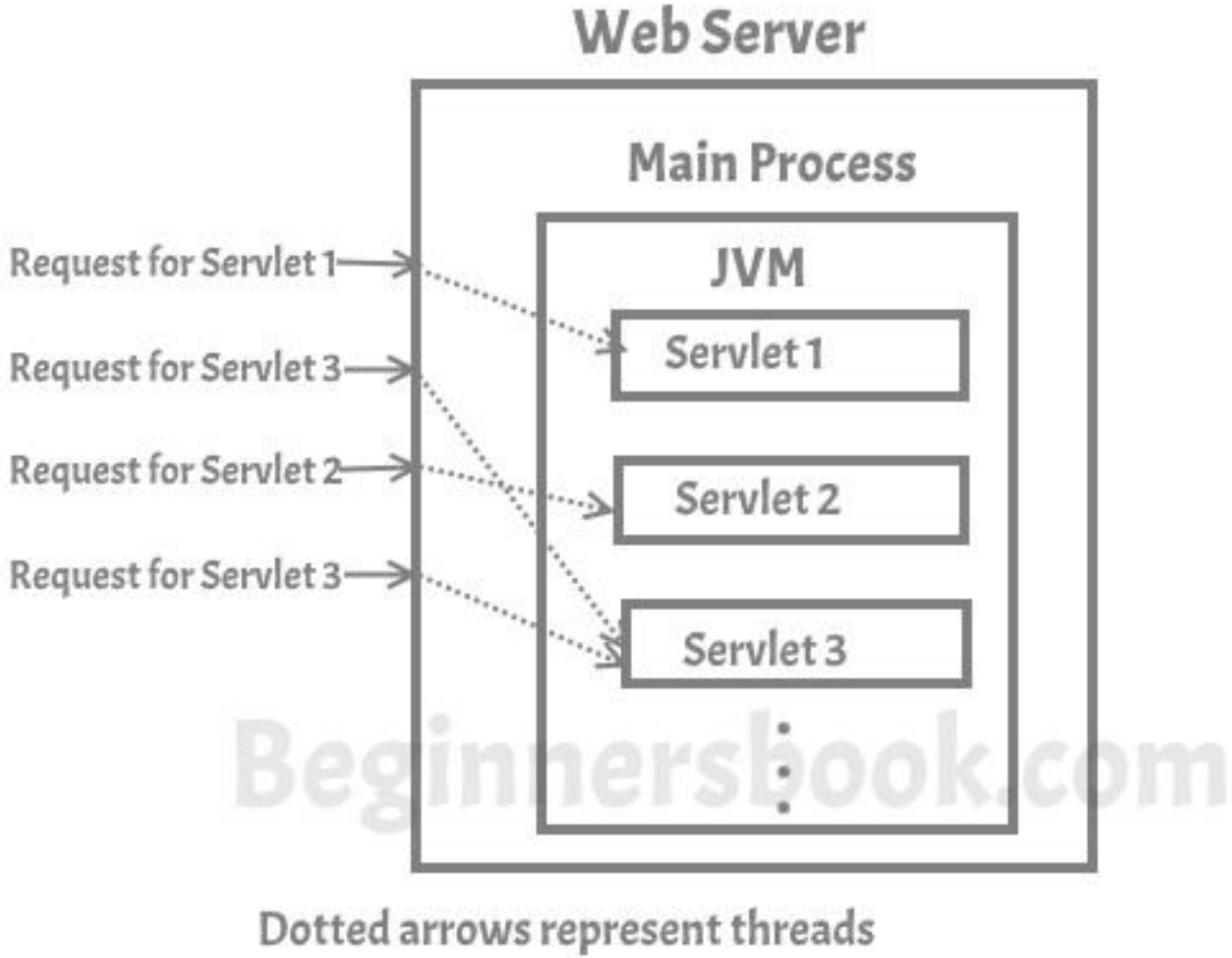
   public void init(ServletConfig config) throws ServletException

4. Call to the service() method : The containers call the service() method each time the request for servlet is received. The Container calls the service() method to handle requests coming from the client, interprets the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

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

5. Call to destroy() method: The Web Container call the destroy() method before removing servlet instance, giving it a chance for cleanup activity. Called by the Servlet Container to take the Servlet out of service. This method is only called once all threads within the servlet's services method have exited or after a timeout period has passed. After the container calls this method, it will not call the service method again on the Servlet.

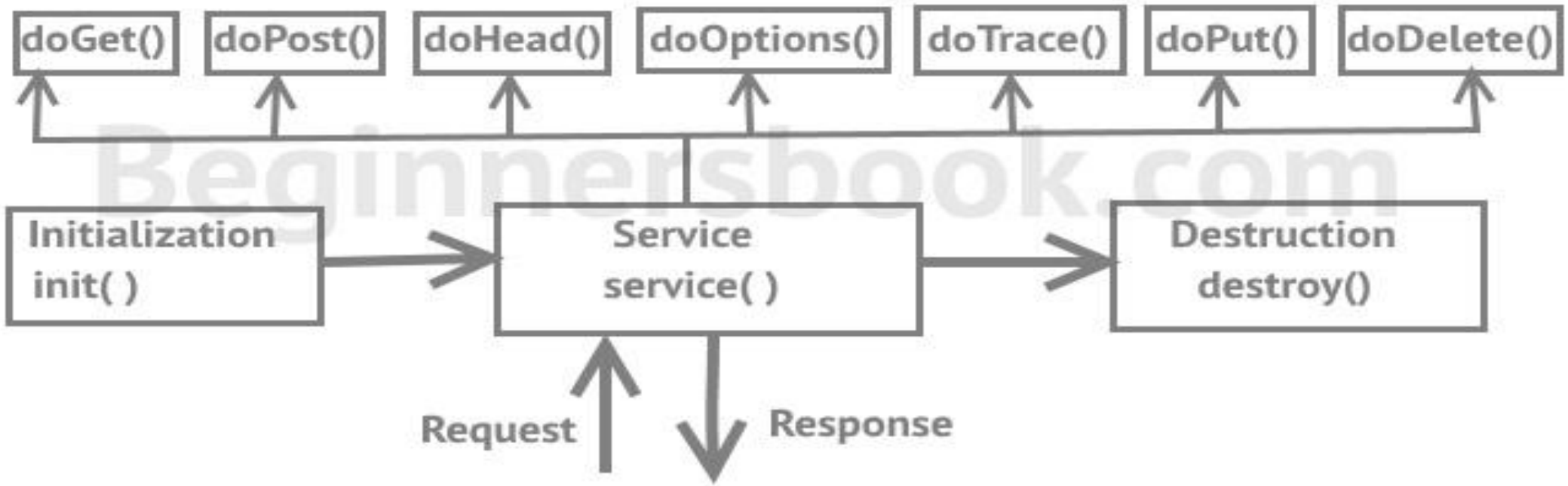
How servlet works?



Init()

- init() is **guaranteed to be called before the servlet handles its first request**.
- It is typically used to **perform servlet initialization**--creating or loading objects that are used by the servlet in the handling of its requests.
- Why not use a constructor instead? Well, in JDK 1.0 (for which servlets were originally written), **constructors for dynamically loaded Java classes (such as servlets) couldn't accept arguments**. So, in order to provide a **new servlet any information about itself and its environment, a server had to call a servlet's init() method and pass along an object that implements the ServletConfig interface**. Also, Java doesn't allow interfaces to declare constructors. This means that the javax.servlet.Servlet interface cannot declare a constructor that accepts a ServletConfig parameter.
- It's still possible to define constructors for your servlets, but in the **constructor you don't have access to the ServletConfig object or the ability to throw a ServletException**.
- init(ServletConfig) calls the getInitParameter method. This method takes the parameter name as an argument and returns a String representation of the parameter's value.
- The specification of initialization parameters is server-specific.

Service()



Unlike init() and destroy() that are called only once, the service() method can be called any number of times during servlet life cycle. As long as servlet is not destroyed, for each client request the service() method is invoked.

Destroy()

- The server calls a servlet's destroy() method when the servlet is about to be unloaded.
- In the destroy() method, a servlet should free any resources it has acquired that will not be garbage collected.
- The destroy() method also gives a servlet a chance to write out its unsaved cached information or any persistent information that should be read during the next call to init().
- A server calls the destroy method after all service calls have been completed, or a server-specific number of seconds have passed, whichever comes first. If your servlet handles any long-running operations, service methods might still be running when the server calls the destroy method.
- To destroy any resources specific to your servlet, override the destroy method. The destroy method should undo any initialization work and synchronize persistent state with the current in-memory state.

The GET Method

- In GET method the **data is sent as URL parameters** that are usually strings of name and value pairs separated by ampersands (&).
- **Advantages of Using the GET Method**
- Since the data sent by the GET method are displayed in the URL, it is possible to **bookmark the page with specific query string values**.
- GET requests can be **cached and GET requests remain in the browser history**.
- GET requests can be **bookmarked**.
- **Disadvantages of Using the GET Method**
- The GET method is **not suitable for passing sensitive information** such as the username and password, because these are fully visible in the URL query string as well as potentially stored in the client browser's memory as a visited page.
- Because the GET method assigns **data to a server environment variable**, the length of the URL is limited. So, there is a limitation for the total data to be sent.

GET Method

- A GET request **retrieves data from a web server by specifying parameters** in the URL portion of the request. This is the main method used f**or document retrieval**.

http://www.test.com/hello?key1 = value1&key2 = value2

- This method is used to handle the GET request on the server-side.
- This method also automatically supports HTTP HEAD (HEAD request is a GET request which returns nobody in response) request.
- The GET type request is usually used to *preprocess* a request.

Modifier and Type: protected void

Syntax:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException,IOException
```

HEAD Method

The HEAD method is functionally similar to GET, except that the **server replies with a response line and headers, but no entity-body**.

- This method is overridden to handle the HEAD request.
- In this method, the response contains the only header but does not contain the message body.
- This method is used to improve performance (avoid computing response body).

Modifier and Type: protected void

Syntax:

```
protected void doHead(HttpServletRequest request, HttpServletResponse response)
  throws ServletException,IOException
```

The POST Method

- In POST method the data is sent to the server as a package in a separate communication with the processing script. Data sent through POST method will not be visible in the URL.
- **Advantages of using POST Method**
- It is more secure than GET because **user-entered information is never visible** in the URL query string or in the server logs.
- **Disadvantages of using the POST Method**
- Since the data sent by the POST method is not visible in the URL, so it is not possible to bookmark the page with specific query.
- POST **requests are never cached**
- POST requests **do not remain in the browser history**.

POST Method

The POST method is used when you want to send some data to the server, for example, file update, form data, etc. This method is used to handle the POST request on the server-side.

- This method allows the client to send data of unlimited length to the webserver at a time.
- The POST type request is usually used to post-process a request.

Modifier and Type: protected void

Syntax:

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
  throws ServletException,IOException
```

PUT Method

The PUT method is used to request the **server to store the included entity-body at a location specified by the given URL**.

- This method is overridden to handle the PUT request.
- This method allows the client to store the information on the server(to save the image file on the server).
- This method is called by the server (via the service method) to handle a PUT request.

Modifier and Type: protected void

Syntax:

```
protected void doPut(HttpServletRequest request, HttpServletResponse response)
  throws ServletException,IOException
```

DELETE Method

The DELETE method is used to request the server to delete a file at a location specified by the given URL.

- This method is overridden to handle the DELETE request.
- This method allows a client to remove a document or Web page from the server.
- While using this method, it may be useful to save a copy of the affected URL in temporary storage to avoid data loss.

Modifier and Type: protected void

Syntax:

```
protected void doDelete(HttpServletRequest request, HttpServletResponse response)
  throws ServletException,IOException
```

OPTIONS Method

- The OPTIONS method is used by the client to **find out the HTTP methods and other options supported by a web server**. For security reasons, when you send data to a different domain (**cross-domain requests**), browsers usually send a 'preflight' HTTP OPTIONS request to the target server before sending the data there. The client can specify a URL for the OPTIONS method, or an asterisk (*) to refer to the entire server. The HTTP OPTIONS method is used to request information about the communication options available for a given resource. A resource can be any entity that can be identified by a URI (Uniform Resource Identifier), such as a web page, an image, a file, or an API endpoint. For example, the URI https://api.example.com/users/1 represents a resource that corresponds to a user with the ID of 1 in the API of example.com.
  - This method is overridden to handle the OPTIONS request.
  - This method is used to determine which HTTP methods the server supports and returns an appropriate header.

Modifier and Type: protected void

Syntax:

```
protected void doOptions(HttpServletRequest request, HttpServletResponse response)
throws ServletException,IOException
```

TRACE Method

The TRACE method is used to echo the contents of an HTTP Request back to the requester which can be used for debugging purpose at the time of development. The HTTP TRACE method is designed for diagnostic purposes. If enabled, the web server will respond to requests that use the TRACE method by echoing in its response the exact request that was received. This behavior is often harmless, but occasionally leads to the disclosure of sensitive information such as internal authentication headers appended by reverse proxies. This functionality could historically be used to bypass the HttpOnly cookie flag on cookies, but this is no longer possible in modern web browsers.

- This method is overridden to handle the TRACE request.
- This method returns the headers sent with the TRACE request to the client so that they can be used in debugging.

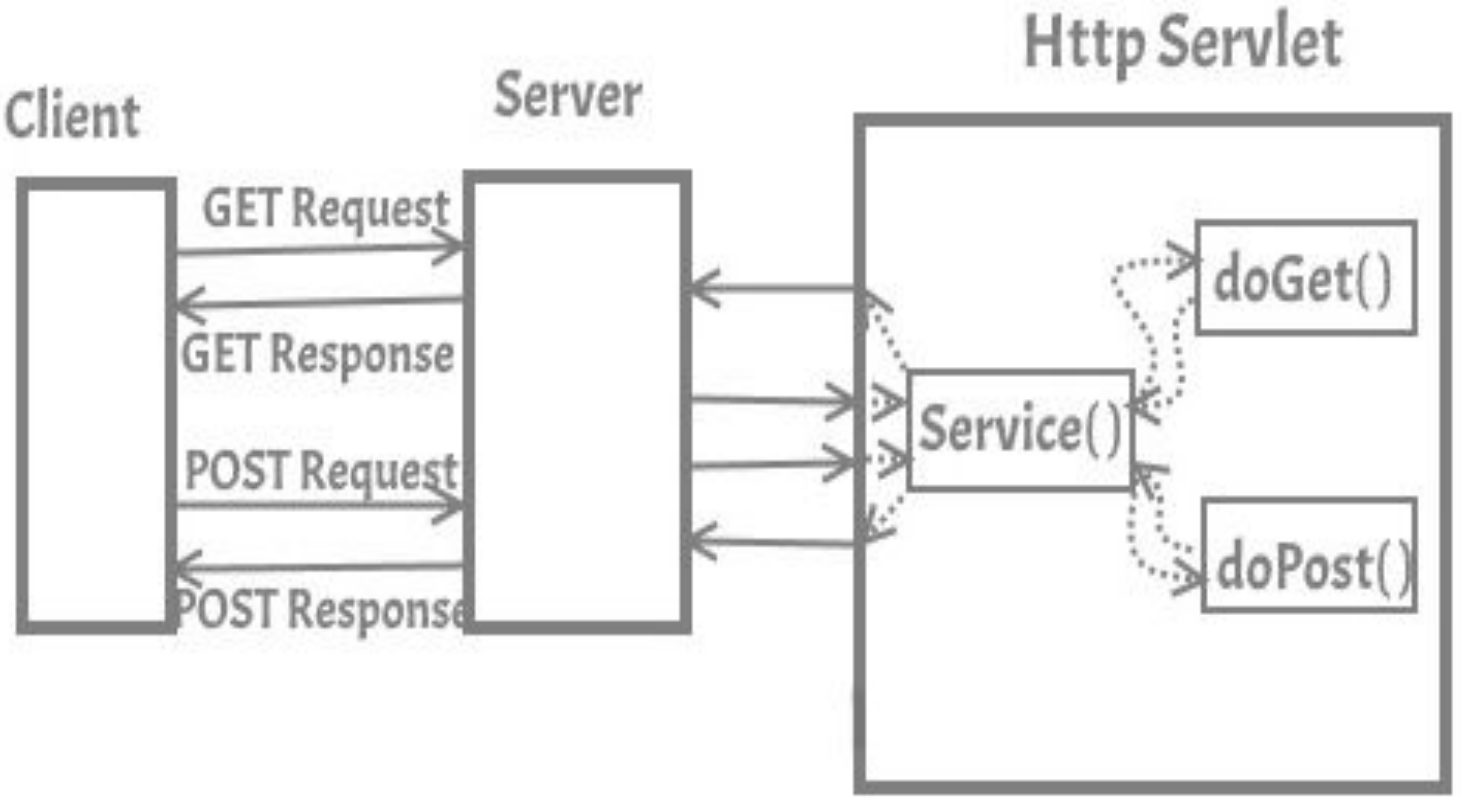
Modifier and Type: protected void

Syntax:

```
protected void doTrace(HttpServletRequest request, HttpServletResponse response)
throws ServletException,IOException
```

Http Servlet

- **In Http Servlet there is no need to override the service() method** as this method dispatches the Http Requests to the correct method handler, for example if it receives HTTP GET Request it dispatches the request to the doGet() method.

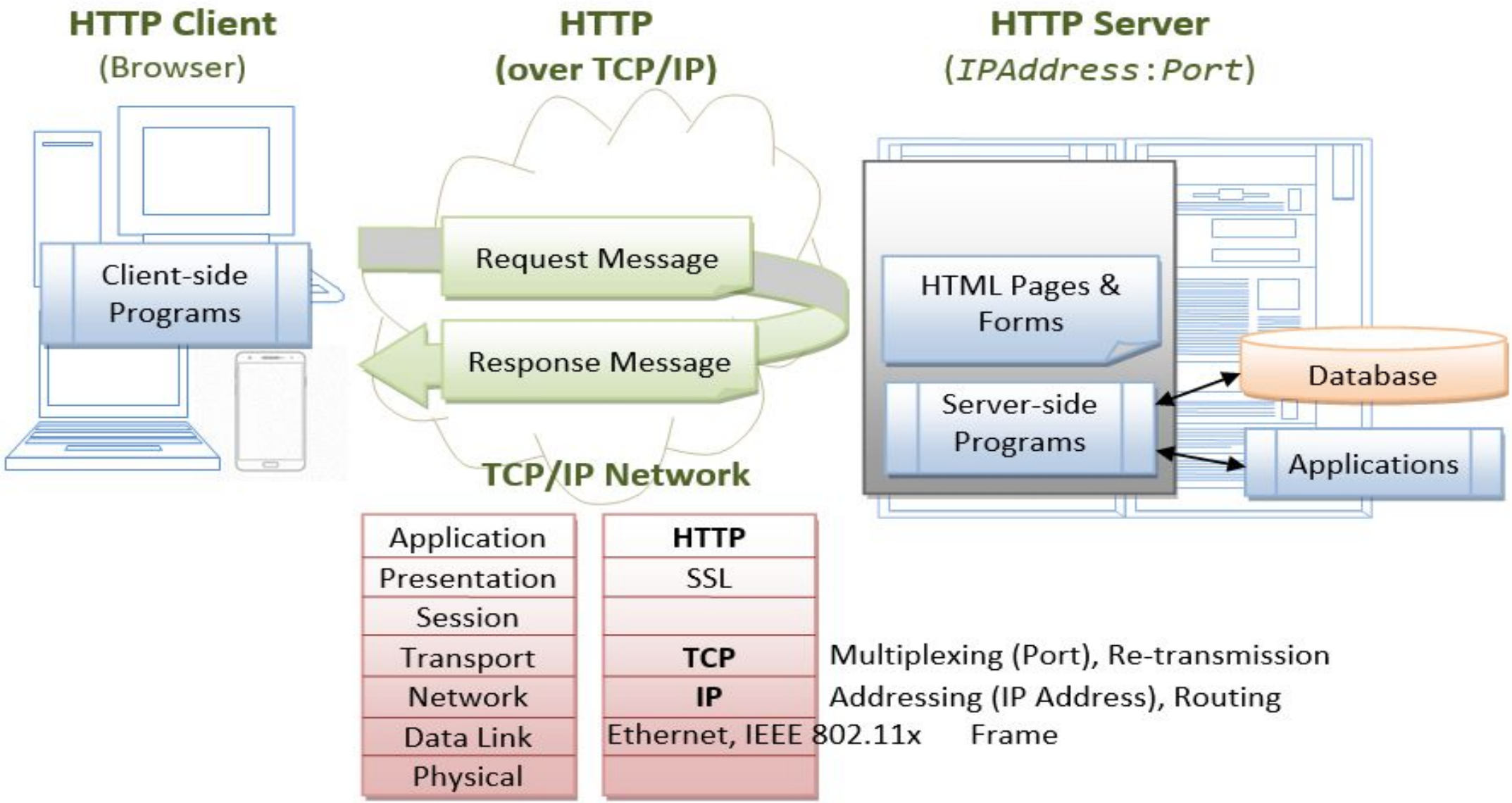


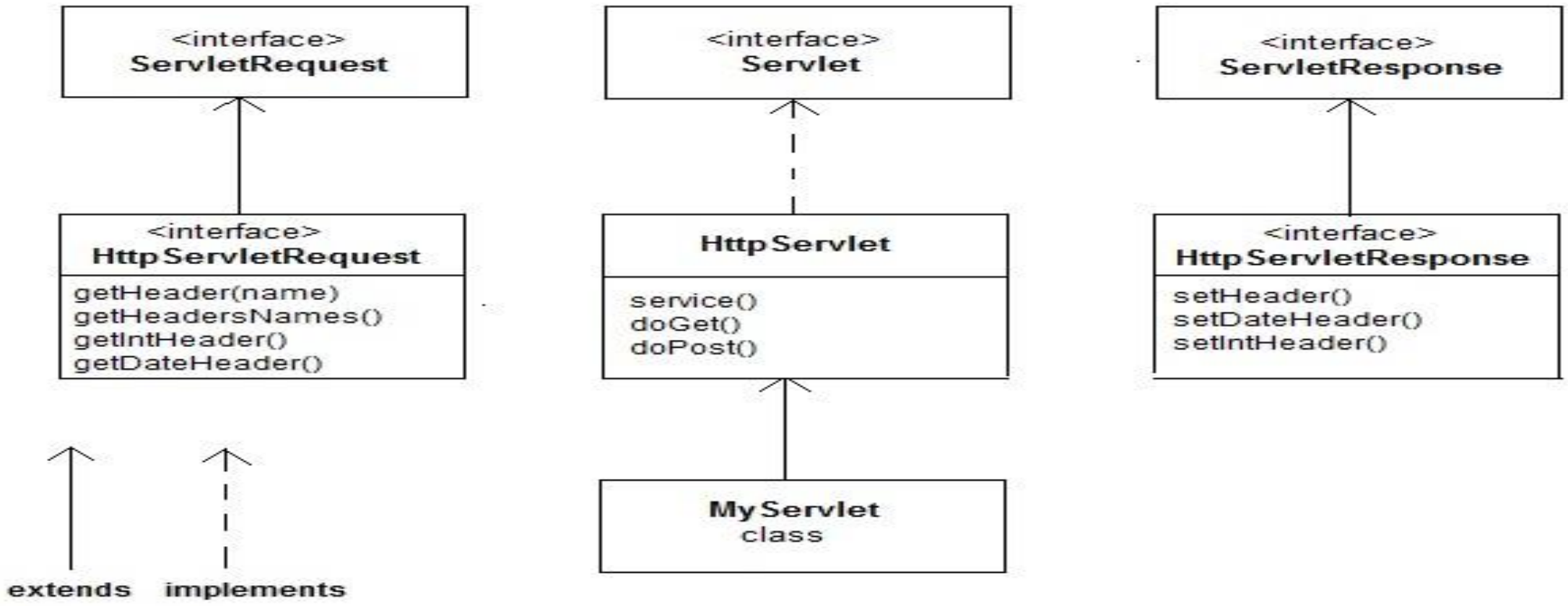
HttpServlet class

- A HTTP Servlet runs under the HTTP protocol. HttpServlet is also an abstract class.
- This class gives implementation of various services () methods of servlet interface.
- To create a servlet, we should create a class that extends HttpServlet abstract class.
- The Servlet class that we will create, must not override service () method. Our servlet class will override only the doGet() and/or doPost() methods

Anatomy of a Servlet

- HTTPServletRequest object
  - Information about an HTTP request
    - Headers
    - Query String
    - Session
    - Cookies
- HTTPServletResponse object
  - Used for formatting an HTTP response
    - Headers
    - Status codes
    - Cookies


HTTP Client
(Browser)
HTTP
(over TCP/IP)
HTTP Server
(IPAddress:Port)
Client-side Programs
Request Message
Response Message
HTML Pages & Forms
Server-side Programs
Database
Applications
TCP/IP Network
Application
Presentation
Session
Transport
Network
Data Link
Physical
HTTP
SSL
TCP
IP
Ethernet, IEEE 802.11x
Multiplexing (Port), Re-transmission
Addressing (IP Address), Routing
Frame



The web.xml file is the **deployment descriptor** for a Servlet-based Java web application (which at most Java web apps are). Among other things, it declares which Servlets exist and which URLs they handle

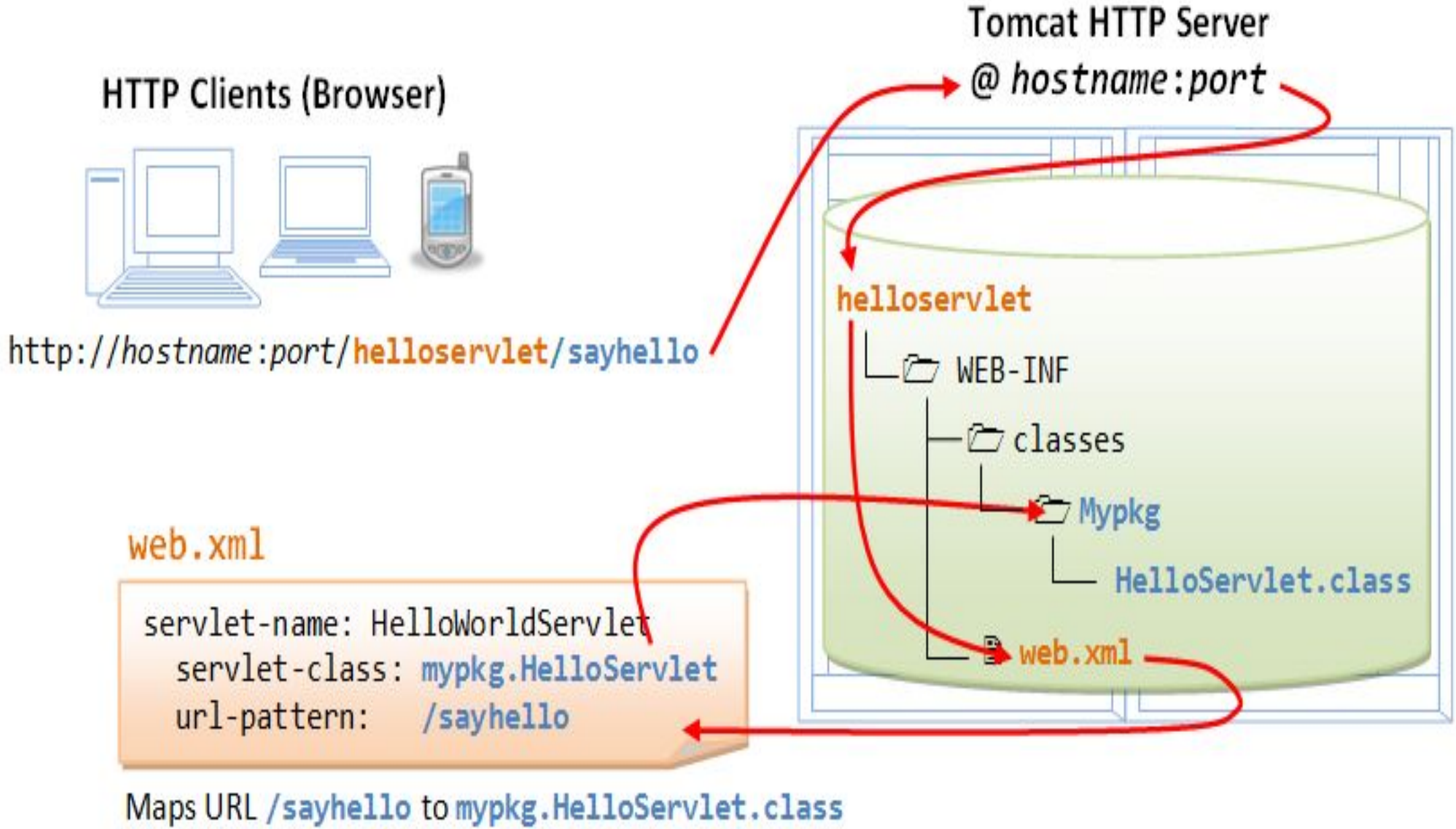
Deployment Descriptor(web.xml)

- The **web**. **xml** file is located in the **WEB**-INF directory ofyour **Web** application.
- The first entry, under the root **servlet** element in **web**. **xml**, defines a name for the **servlet** and specifies the compiled class that executes the **servlet**. ... **xml**, under the **servlet**-mapping element, **defines the URL pattern** that calls this **servlet**.
- **web**. **xml** defines mappings between URL paths and the **servlets** that handle requests with those paths.
- The **web** server uses this configuration to identify the **servlet to handle a given request and call the class method** that corresponds to the request method. For example: the doGet() method for HTTP GET requests.
- **You can** declare **multiple servlets** using the same class with different initialization parameters. The name for each **servlet** must be unique across the deployment descriptor.

```
<web-app>
  <servlet>
    <servlet-name>Validate</servlet-name>
    <servlet-class>Validate</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>Welcome</servlet-name>
    <servlet-class>Welcome</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Validate</servlet-name>
    <url-pattern>/Validate</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>Welcome</servlet-name>
    <url-pattern>/Welcome</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    </welcome-file-list>
</web-app>
```

To map a URL to a servlet, you declare the servlet with the `<servlet>` element, then define a mapping from a URL path to a servlet declaration with the `<servlet-mapping>` element.

The `<servlet>` element declares the servlet, including a name used to refer to the servlet by other elements in the file, the class to use for the servlet, and initialization parameters. You can declare multiple servlets using the same class with different initialization parameters. The name for each servlet must be unique across the deployment descriptor.


Tomcat HTTP Server
@ hostname:port
HTTP Clients (Browser)
http://hostname:port/helloservlet/sayhello
helloservlet
WEB-INF
classes
Mypkg
HelloServlet.class
web.xml
web.xml
servlet-name: HelloWorldServlet
servlet-class: mypkg.HelloServlet
url-pattern: /sayhello
Maps URL /sayhello to mypkg.HelloServlet.class

```
<servlet>
  <servlet-name>HelloWorld2</servlet-name>
  <servlet-class>examples.servlets.HelloWorld2</servlet-class>
  <init-param>
    <param-name>greeting</param-name>
    <param-value>Welcome</param-value>
  </init-param>
  <init-param>
    <param-name>person</param-name>
    <param-value>WebLogic Developer</param-value>
  </init-param>
</servlet>
```

You define initialization attributes for servlets in the Web application deployment descriptor, web.xml, in the init-param element of the servlet element, using param-name and param-value tags.

```
<servlet-mapping>
  <servlet-name>redteam</servlet-name>
  <url-pattern>/red/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
 <servlet-name>blueteam</servlet-name>
 <url-pattern>/blue/*</url-pattern>
</servlet-mapping>
```

The `<servlet-mapping>` element specifies a URL pattern and the name of a declared servlet to use for requests whose URL matches the pattern. The URL pattern can use an asterisk (`*`) at the beginning or end of the pattern to indicate zero or more of any character. (The standard does not support wildcards in the middle of a string, and does not allow multiple wildcards in one pattern.) The pattern matches the full path of the URL, starting with and including the forward slash (`/`) following the domain name.

Web.xml

It resides in the app's WAR under the WEB-INF/ directory. The file is an XML file whose root element is <web-app>.

The servlet can access its initialization parameters by getting its servlet configuration using its own getServletConfig() method, then calling the getInitParameter() method on the configuration object using the name of the parameter as an argument.

```
String teamColor = getServletConfig().getInitParameter("teamColor");
```

Servlet Load-on-Startup in web.xml

- The <servlet> element has a subelement called <load-on-startup> which you can use to control when the servlet container should load the servlet. If you do not specify a <load-on-startup> element, the servlet container will typically load your servlet when the first request arrives for it.
- By setting a <load-on-startup> element, you can tell the servlet container to load the servlet as soon as the servlet container starts. The **load-on-startup** element of **web-app** loads the servlet at the time of deployment or server start if value is positive. It is also known as **pre initialization of servlet**. Remember, the servlets init() method is called when the servlet is loaded.

```
<servlet>
   <servlet-name>controlServlet</servlet-name>
   <servlet-class>ControlServlet</servlet-class>
   <init-param><param-name>static</param-name>
            <param-value>/WEB-INF/container.script</param-value>
   </init-param>
   <load-on-startup>1</load-on-startup>
</servlet>
```

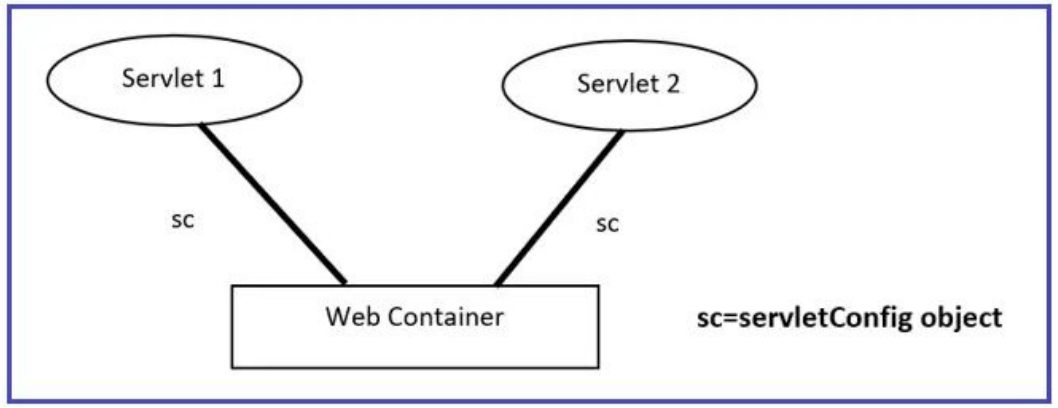
- The number inside the <load-on-startup>1</load-on-startup> element tells the servlet container in what sequence the servlets should be loaded. The lower numbers are loaded first. If the value is negative, or unspecified, the servlet container can load the servlet at any time.

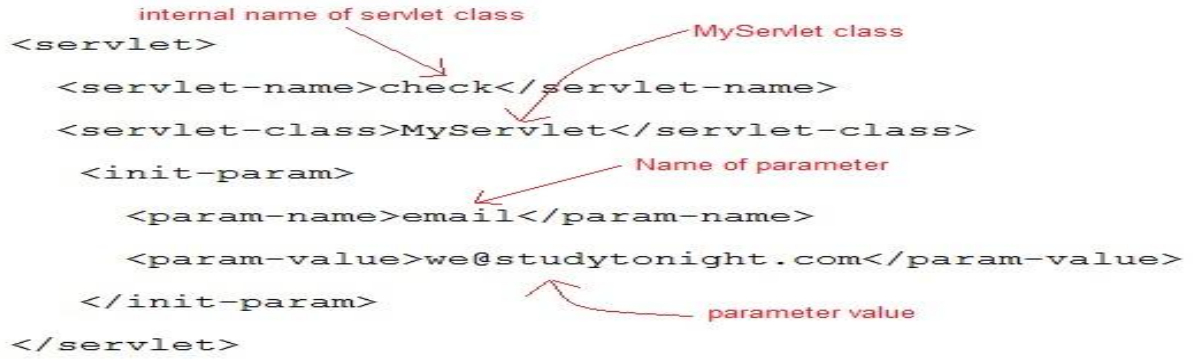
```
<servlet>     <servlet-name>servlet1</servlet-name>
 <servlet-class>com.javatpoint.FirstServlet</servlet-class>
<load-on-startup>0</load-on-startup>    </servlet>
<servlet>
  <servlet-name>servlet2</servlet-name>
<servlet-class>com.javatpoint.SecondServlet</servlet-class>
    <load-on-startup>1</load-on-startup>    </servlet>
```

There are defined 2 servlets, both servlets will be loaded at the time of project deployment or server start. But, servlet1 will be loaded first then servlet2.

ServletConfig interface

- ServletConfig object can be created and removed only by the server, not by the programmer.
- ServletConfig object is created when the client sends the request to the particular servlet and when the servlet object is created.
- ServletConfig object will be removed by the server when we close the server or when the servlet is deleted from the project.
- One ServletConfig object is created for one Servlet it means in one project we can contain multiple ServletConfig objects based on the multiple servlets available.
- This servletConfig object can be used only by the particular servlet which contains the ServletConfig object. But the ServletConfig object of one servlet cannot be used by any other servlet which maybe belongs to the same or other projects.
- In web application execution, the container will prepare ServletCofig object immediately after servlet instantiation and just before calling init() method in servlet initialization. The container will destroy the ServletConfig object just before servlet de-instantiation. Due to the above reasons, the life cycle of the ServletConfig object is up to a specific servlet.
- **Advantage**: No need to update in web.xml if any changes occur.
- In web applications, the ServletConfig object will allow only parameter data, it'll not allow attributes data

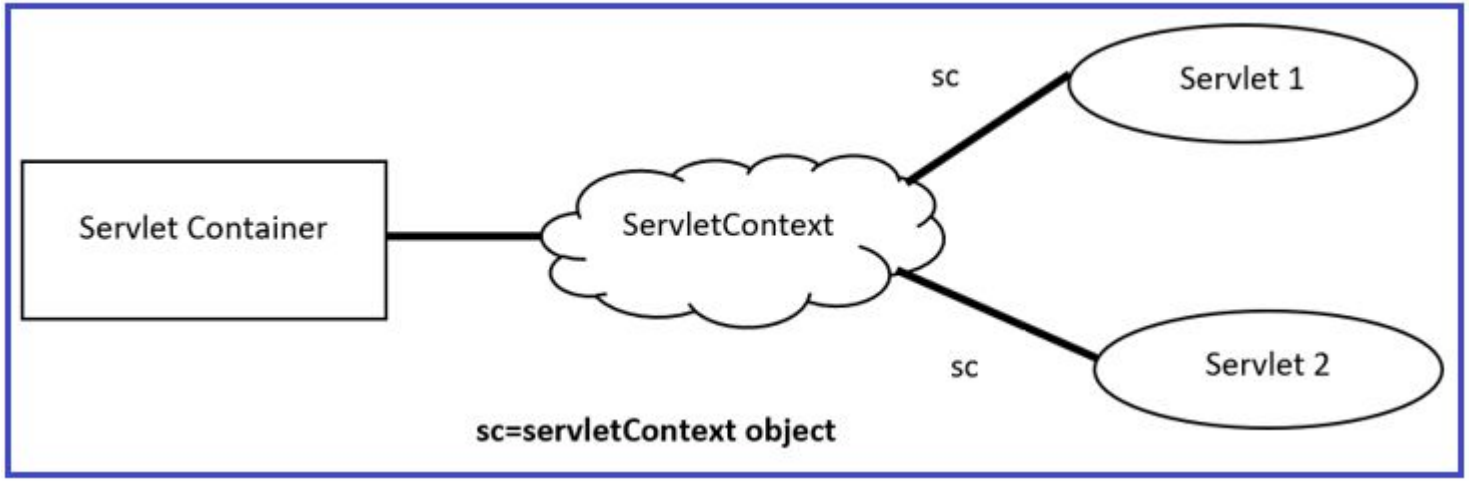




ServletContext Interface

- ServletContext is an object, it will manage all the context details of a particular web application, where the context details include the logical name of the web application and context parameters, and so on.
- ServletConetxt is an object, it will provide a complete view of a particular web application.
- ServletContext object will be prepared by the container the moment when we start the server i.e. the time when we deploy the web application.
- ServletContext object will be destroyed by the container when we shut down the server i.e. the time when we un-deploy the web application. The life of the ServletContext object is almost all the life of the respective web application.
- If we declare any data in the ServletContext object then that data will be shared with all the number of resources that are available in the present web application. ServletContext will provide more shareability.
- In web applications, the container will prepare ServletContext object irrespective of getting requests from the client. In the web application, ServletContext will allow both parameters and attributes data. ServletContext will allow both Static Inclusion and Dynamic Inclusion of data.
- ServletContext objects can be created and removed only by the server, not by the programmer.
- ServletContext object is created at the time of deploying the project. ServletContext object will be removed by the server when we un-deploy the project. One servletContext object is created from one project. The ServletContext object can be used by all the servlets which belong to the particular project. But ServletContext object of one project cannot be used by a servlet which belongs to another project.

- **Advantage of ServletContext Interface**
- It is easy to maintain. If any information is shared with all or any the servlet, it's better to form it available for all the servlets. We provide this information from the online .xml file, so if the knowledge is modified, we don't get to modify the servlet. Thus, it removes the maintenance problems.
- **Usage of ServletContext Object**
- It provides an interface between the container and the servlet.
- It is used to get configuration information from the web.xml file.
- It is used to set, get, or remove attributes from the web.xml file.
- It is used to provide inter-application communication.



Context Parameters in web.xml

```
<context-param>
   <param-name>myParam</param-name>
   <param-value>the value</param-value>
</context-param>
```

Here is how you access the parameter from inside an HttpServlet subclass:

```
String myContextParam = request.getSession().getServletContext()
.getInitParameter("myParam");
```

Defines a set of methods that a servlet uses to communicate with its servlet container, for example, to get the MIME type of a file, dispatch requests, or write to a log file. There is one context per "web application" per Java Virtual Machine.

Servlet Config	Servlet Context
Servlet config object represent single servlet	It represent whole web application running on particular JVM and common for all the servlet
Its like local parameter associated with particular servlet	Its like global parameter associated with whole application
It's a name value pair defined inside the servlet section of web.xml file so it has servlet wide scope	`ServletContext` has application wide scope so define outside of servlet tag in web.xml file.
`getServletConfig()` method is used to get the config object	`getServletContext()` method is used to get the context object.
for example shopping cart of a user is a specific to particular user so here we can use servlet config	To get the MIME type of a file or application session related information is stored using servlet context object.

Context Init parameters	**Servlet Init parameter**
Available to all servlets and JSPs that are part of web	Available to only servlet for which the <init-param> was configured
Context Init parameters are initialized within the <web-app> not within a specific <servlet> elements	Initialized within the <servlet> for each specific servlet.
ServletContext object is used to get Context Init parameters	ServletConfig object is used to get Servlet Init parameters
Only one ServletContext object for entire web app	Each servlet has its own ServletConfig object

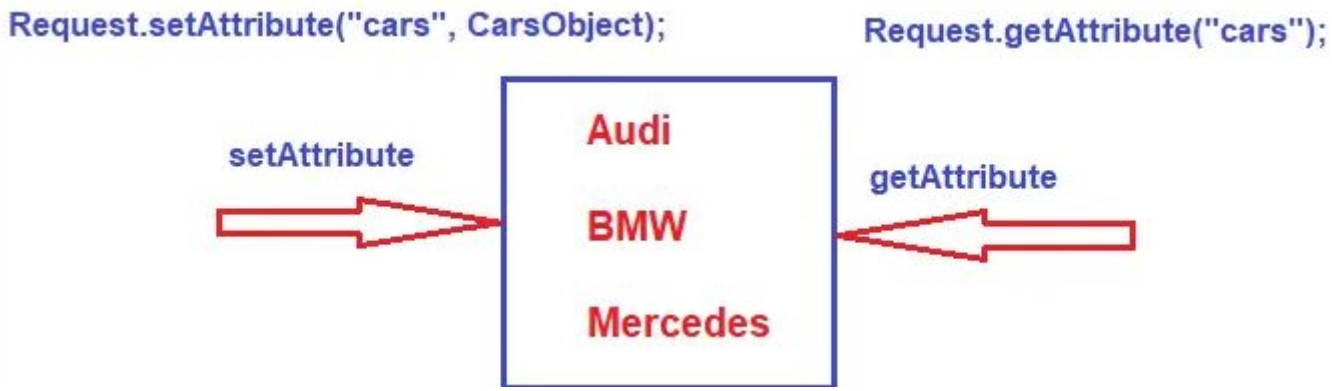
- **The pom. xml is for configure your project with Maven.**
- **The web. xml is use in all Java EE project under Tomcat**.

Servlet Attributes in Java

- The attribute is a Java object which can be set, get, or removed. It provides more visibility and accessibility to the data of the application. Scopes to set, get, or remove an object are:
- **Request attribute**
- **Session attribute**
- **Application attribute**

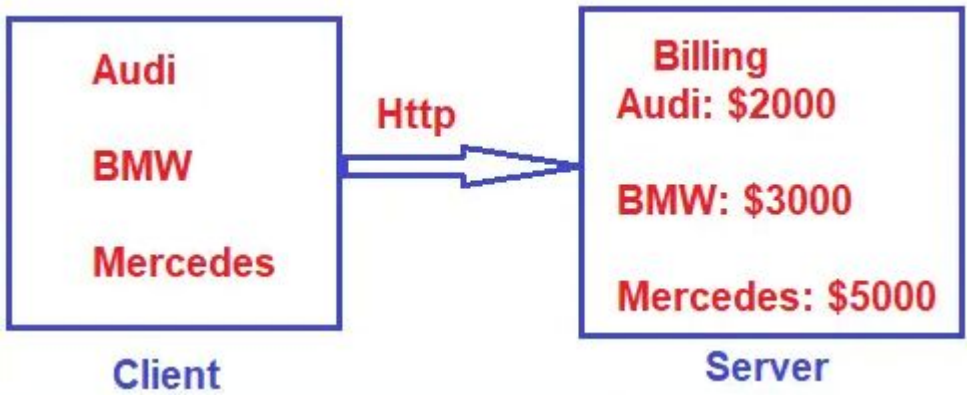
Request Attribute in Servlet

- The request scope is used while processing the results of a submitted form. We can set and get Request Attributes with an object. For example:
- **Request.setAttribute(“cars”, CarsObject);**
  Request.getAttribute(“cars”);



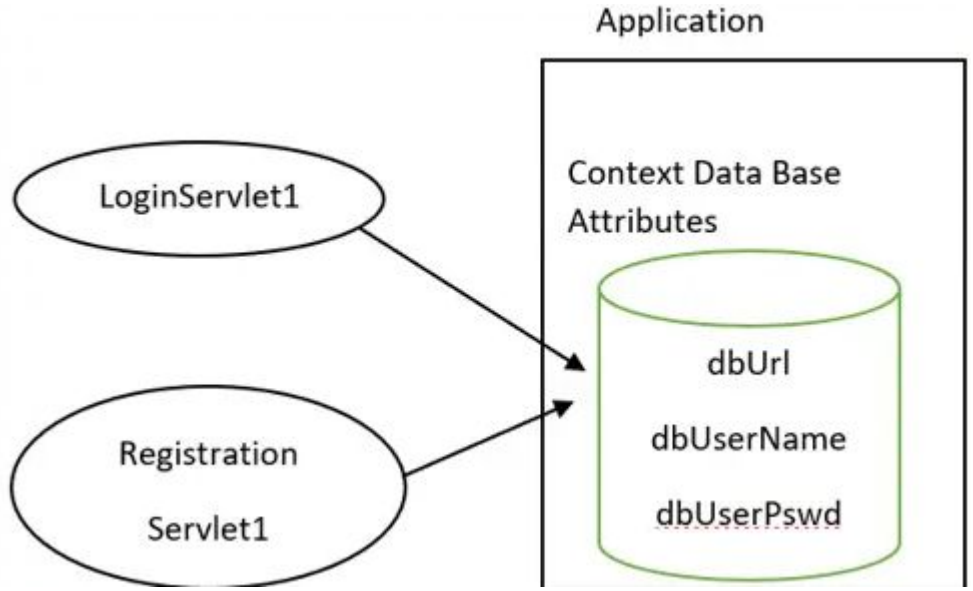
Session Attribute in Servlet

- Session Attribute is used to create the session by the Web Container when a user visits the web application. For example:
- **HttpSession session=request.getSession();**
  Session.setAttribute(“cars”,CarsObject);

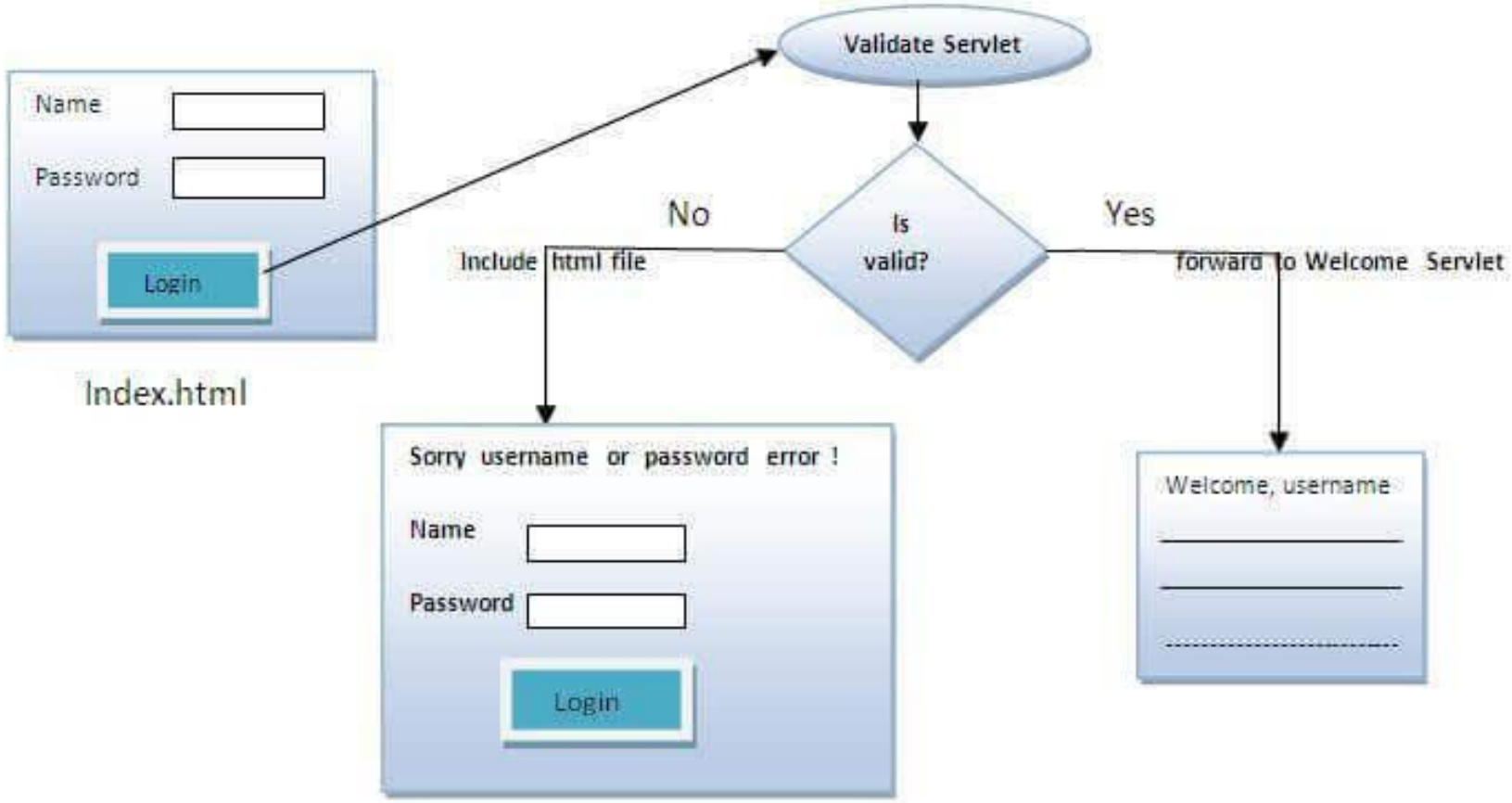


Application Attribute in Servlet

- Application Attribute scope persists until the application is delayed. It is also known as the context attribute. Any servlet within the same application can have access to context attributes. For example:
- **getServletContext().setAttribute("dbUrl","jdbc:mysql://localhost:3306/login");**
  getServletContext(),getAttribute("dbUrl");



Request Dispatcher



Request Dispatcher

- RequestDispatcher is an interface, implementation of which **defines an object which can dispatch request to any resources(such as HTML, Image, JSP, Servlet) on the server**.
- The RequestDispatcher is **an Interface** that comes under package ***javax.servlet***. Using this interface we get an object in servlet after receiving the request. Using **the RequestDispatcher object we send a request to other resources which include** (*servlet, HTML file, or JSP file*). A RequestDispatcher object can be used **to forward a request to the resource or to include the resource in a response**. The resource can be dynamic or static.

There are **three ways** to get an object:

public interface *ServletContext*. Defines a set of methods that a servlet uses to communicate with its servlet container.

1. RequestDispatcher requestDispatcher=ServletContext.getRequestDispatcher(String path);

path is a string specifying the pathname to the resource(servlet, HTML file, or JSP file).

2. RequestDispatcher requestDispatcher=ServletContext.getNamedDispatcher(String name);

name is a string specifying the name of a servlet to wrap.

3. RequestDispatcher requestDispatcher=request.getRequestDispatcher("String path");

request is the HttpServletRequest type object.

path is a string specifying the pathname to the resource. If it is relative, it must be relative to the current servlet.

sendRedirect() Method

- sendRedirect() method **redirects the response to another resource**. This method actually makes the client (browser) **to create a new request to get to the resource**.
- The client can see the new url in the browser. sendRedirect() accepts relative URL, so **it can go for resources inside or outside the server**.
- Servlets are the Java programs that **run on the server-side and generate dynamic responses to the client request**. Servlet accepts the request from the browser, processes it, and generates the response to the browser.
- While processing the request, let's say **if need to call another servlet from a different server, we need to redirect the response to that resource**. To achieve this, Java servlets provide sendRedirect() method in HttpServletResponse interface in ***javax.servlet.http*** package.

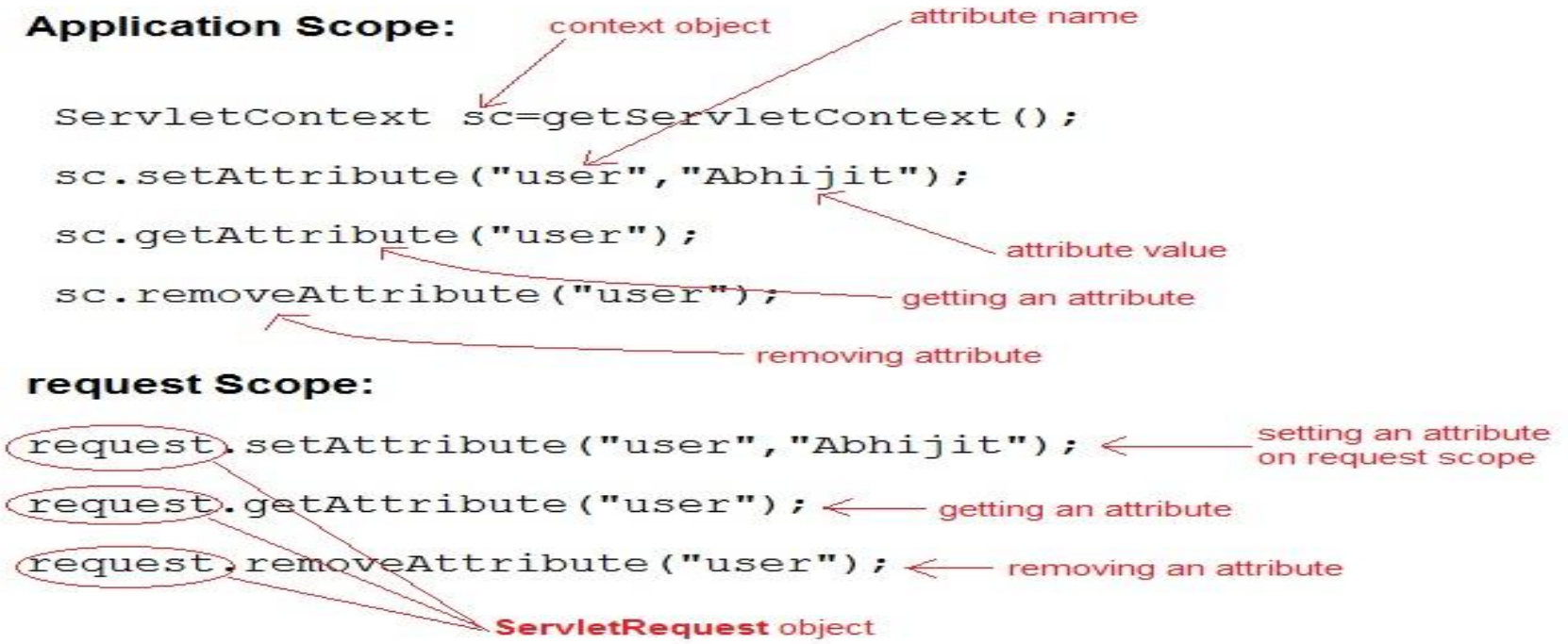
sendRedirect() and Request Dispatcher

The main difference between a **redirection** and a **request dispatching** is that,

- redirection makes the client(browser) **create a new request to get to the resource, the user can see the new URL** while request dispatch **get the resource in same request and URL does not changes**.
- sendRedirect() works on **response** object while request dispatch work on **request** object.

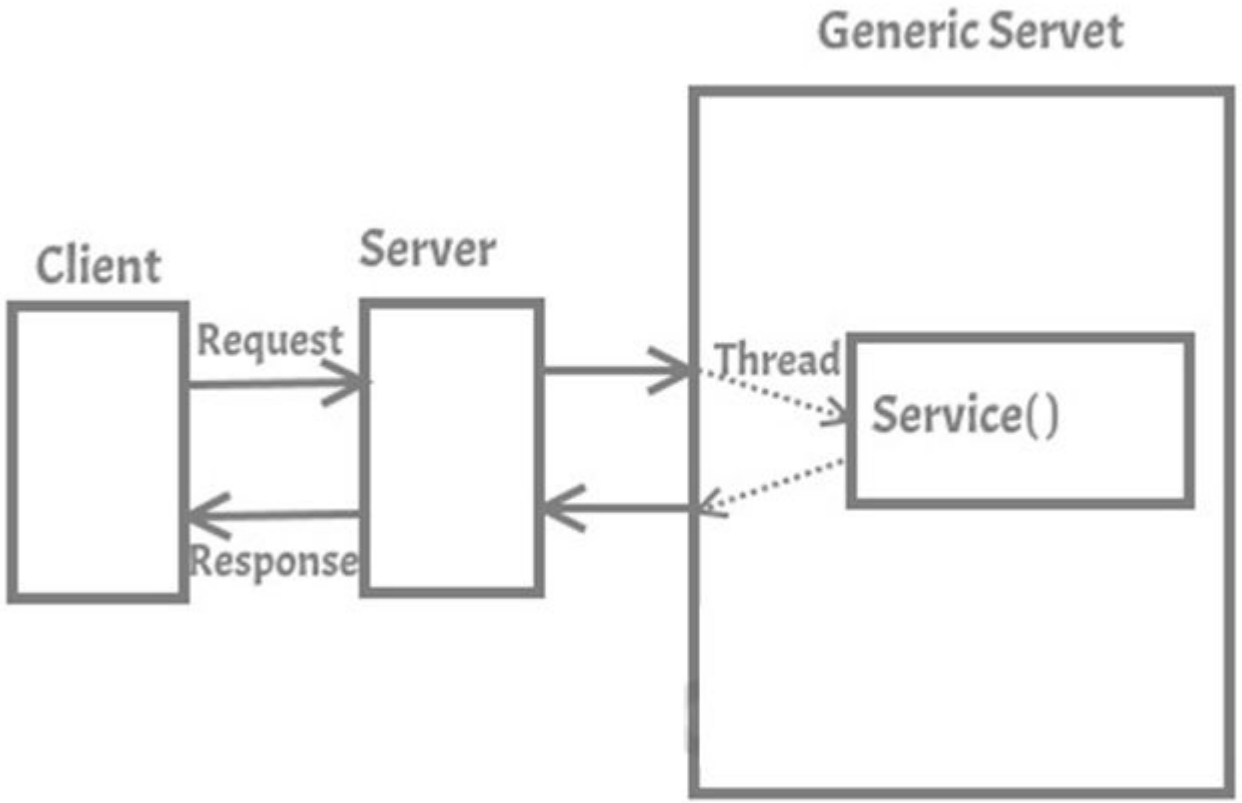
Attribute

- An **attribute** is an object that is used to share information in a web app. Attribute allows Servlets to share information among themselves. Attributes can be SET and GET from one of the following scopes :
- request
- session
- application



Generic Servlet

- public abstract void service(ServletRequest request, ServletResponse response) throws ServletException, java.io.IOException



- **Pros of using Generic Servlet:**
  1. Generic Servlet is easier to write
  2. Has simple lifecycle methods
  3. To write Generic Servlet you just need to extend javax.servlet.GenericServlet and override the service() method
- **Cons of using Generic Servlet:**
  1. Working with Generic Servlet is not that easy because we don't have convenience methods such as doGet(), doPost(), doHead() etc in Generic Servlet that we can use in Http Servlet.

  2. In Generic Servlet we only override service() method for each type of request which is cumbersome.

Methods of GenericServlet class

- public void init(ServletConfig)
- public abstract void service(ServletRequest request,ServletResposne response)
- public void destroy()
- public ServletConfig getServletConfig()
- public String getServletInfo()
- public ServletContext getServletContext()
- public String getInitParameter(String name)
- public Enumeration getInitParameterNames()
- public String getServletName()
- public void log(String msg)
- public void log(String msg, Throwable t)

```
@WebServlet(urlPatterns = "/errorHandler")
public class ErrorHandlerServlet extends HttpServlet {
@Override    protected void doGet(HttpServletRequest req,      HttpServletResponse resp) throws IOException {
resp.setContentType("text/html; charset=utf-8");
try (PrintWriter writer = resp.getWriter()) {
writer.write("<html><head><title>Error description</title></head><body>");
writer.write("<h2>Error description</h2>");
writer.write("<ul>");
 Arrays.asList(ERROR_STATUS_CODE, ERROR_EXCEPTION_TYPE, ERROR_MESSAGE)           .forEach(e -> writer.write("<li>" + e + ":" + req.getAttribute(e) + " </li>")           );          writer.write("</ul>");
writer.write("</html></body>");
}    }}
```

Now, we can access http://localhost:8080/javax-servlets/randomError to see the custom error servlet working.

One common feature is, both these Classes are Abstract Classes.

-> GenericServlet is a super class of HttpServlet class.

-> The main difference is that, HttpServlet is a protocol dependent whereas GenericServlet is protocol independent. So GenericServlet can handle all types of protocols, but HttpServlet handle only HTTP specific protocols.

-> GenericServlet belongs to javax.servlet package. HttpServlet belongs to javax.servlet.http package

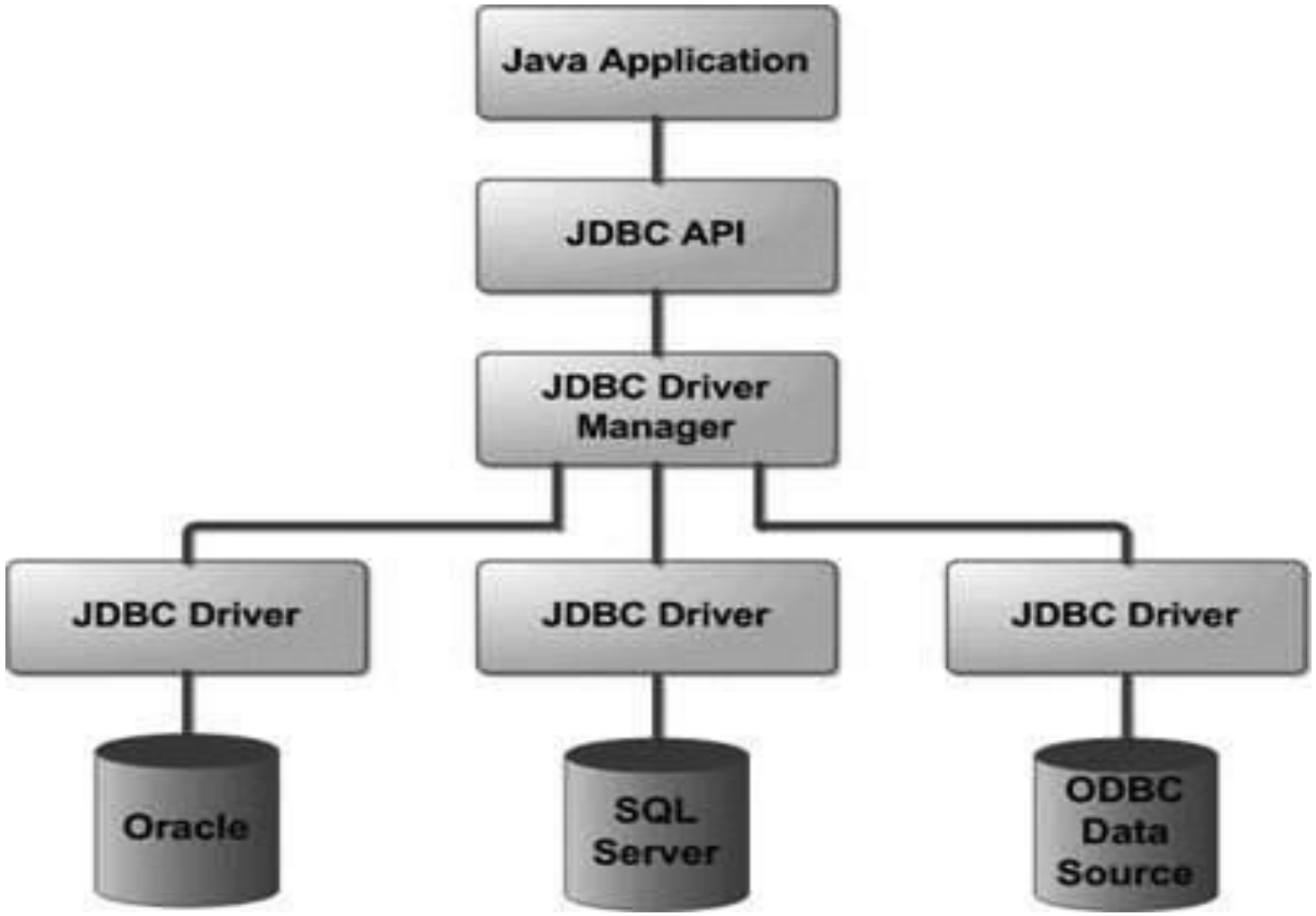
-> GenericServlet is an abstract class which extends Object and implements Servlet, ServletConfig and java.io.Serializable interfaces. HttpServlet is an abstract class which extends GenericServlet and implements java.io.Serializable interface.

-> GenericServlet supports only service() method does not contain doGet() and doPost() methods. HttpServlet support also doGet(), doPost(), doHead() methods (HTTP 1.0) plus doPut(), doOptions(), doDelete(), doTrace() methods (HTTP 1.1).

Sr	Generic Servlet	Http Servlet
1	javax.servlet.GenericServlet(abstract class)	javax.servlet.http.HttpServlet(abstract class)
2	It is the immediate subclass of Servlet interface	The immediate super class of HttpServlet is GenericServlet
3	It defines a generic, protocol-independent servlet.it can be used with any protocol, say, SMTP, FTP, CGI including HTTP etc	It defines a HTTP protocol specific servlet
4	GenericServlet is a super class of HttpServlet class.	HttpServlet is a sub class of GenericServlet class.
5	All methods are concrete except service() method. service() method is abstract method.	All methods are concrete (non-abstract). service() is non-abstract method. service() can be replaced by doGet() or doPost() methods

JSP	Servlets
JSP is a webpage scripting language that can generate dynamic content.	Servlets are Java programs that are already compiled which also creates dynamic web content.
JSP run slower compared to Servlet as it takes compilation time to convert into Java Servlets.	Servlets run faster compared to JSP.
It’s easier to code in JSP than in Java Servlets.	Its little much code to write here.
In MVC, jsp act as a view.	In MVC, servlet act as a controller.
JSP are generally preferred when there is not much processing of data required.	servlets are best for use when there is more processing and manipulation involved.
The advantage of JSP programming over servlets is that we can build custom tags which can directly call Java beans.	There is no such custom tag facility in servlets.
We can achieve functionality of JSP at client side by running JavaScript at client side.	There are no such methods for servlets.

JDBC Architecture



We can use JDBC API to handle database using Java program and can perform the following activities:

- Connect to the database
- Execute queries and update statements to the database
- Retrieve the result received from the database.

Types of JDBC architecture

- **Two-tier model:** A java application communicates directly to the data source. The JDBC driver enables the communication between the application and the data source. When a user sends a query to the data source, the answers for those queries are sent back to the user in the form of results.

  The data source can be located on a different machine on a network to which a user is connected. This is known as a **client/server configuration**, where the user's machine acts as a client and the machine having the data source running acts as the server.

- **Three-tier model:** In this, the user's queries are sent to middle-tier services, from which the commands are again sent to the data source. The results are sent back to the middle tier, and from there to the user.

  This type of model is found very useful by management information system directors.

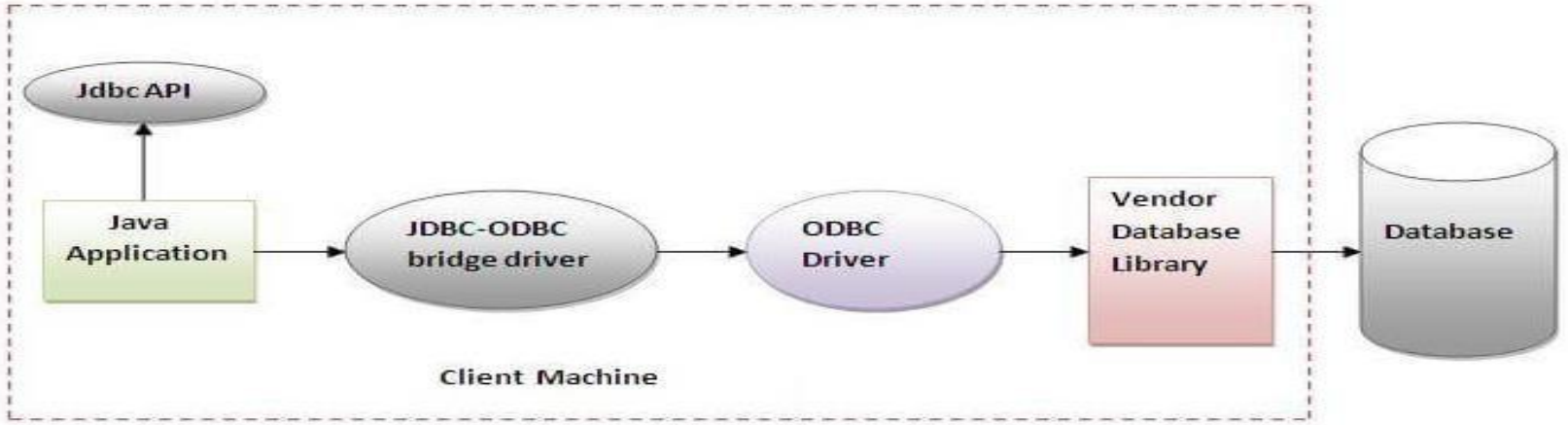
Components of JDBC

- **DriverManager:** This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.
- **Driver:** This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.
- **Connection:** This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.
- **Statement:** You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.
- **ResultSet:** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
- **SQLException:** This class handles any errors that occur in a database application.

JDBC driver

JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver



Advantages:

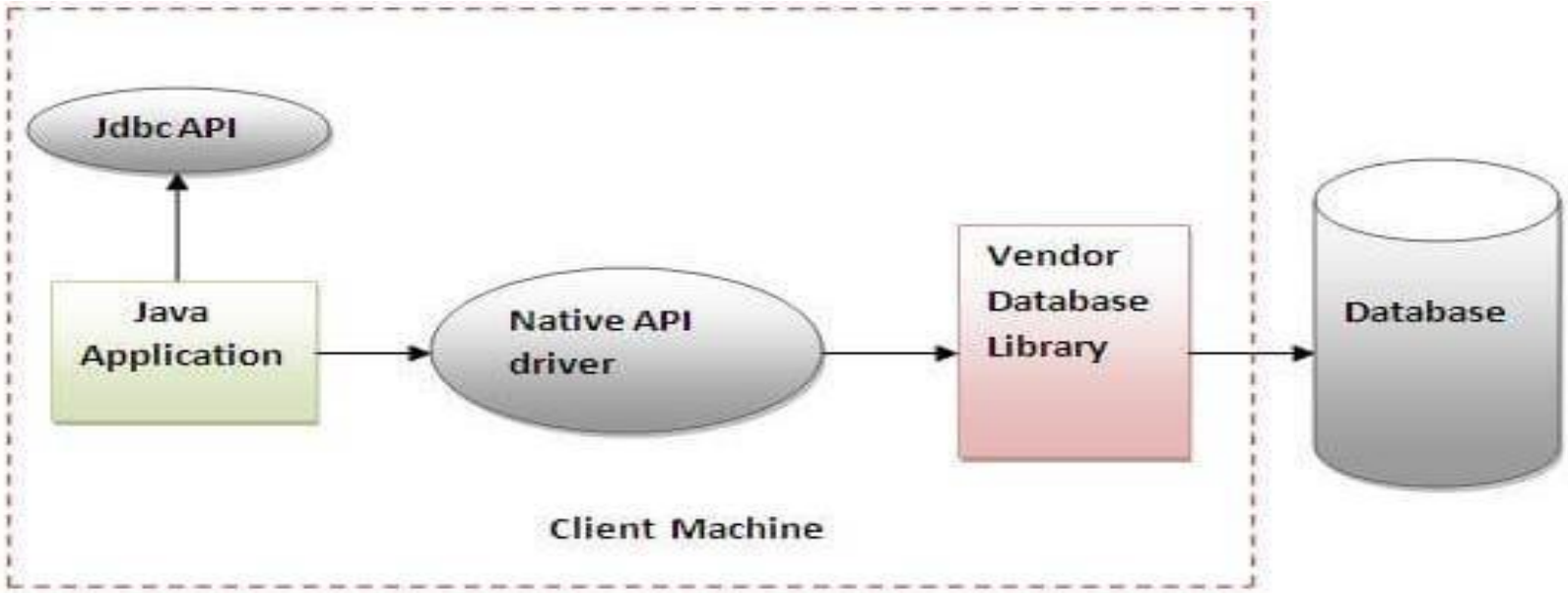
- easy to use.
- can be easily connected to any database. Disadvantages:

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.
- **In Java 8, the JDBC-ODBC Bridge has been removed**

2. Native-API driver (partially java driver)

Advantage:

- performance upgraded than JDBC-ODBC bridge driver. Disadvantage:
- The Native driver needs to be installed on the each client machine.
- The Vendor client library needs to be installed on client machine.



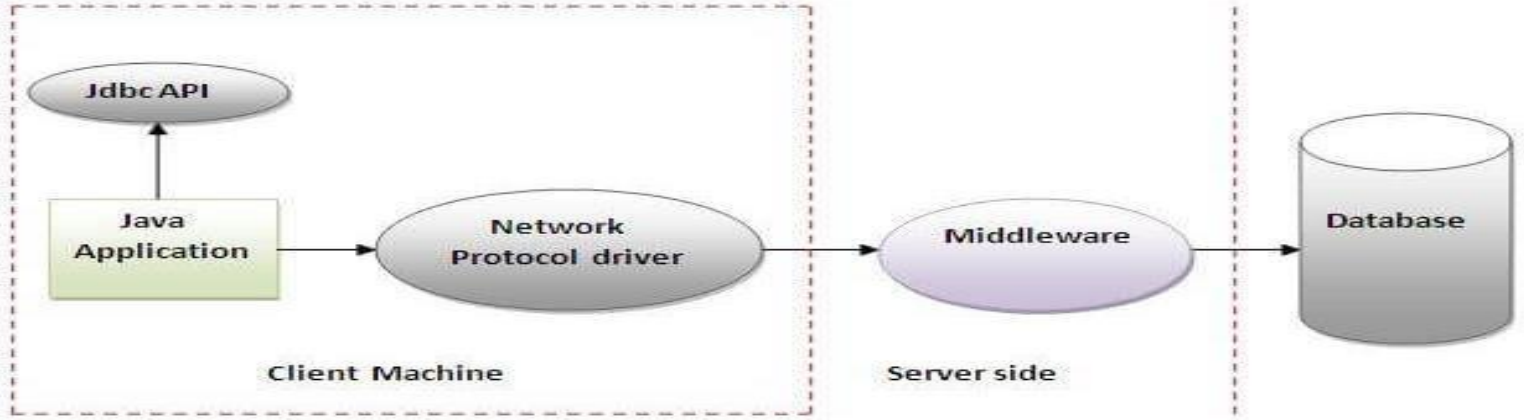
3. Network Protocol driver (fully java driver)

Advantage:

- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier



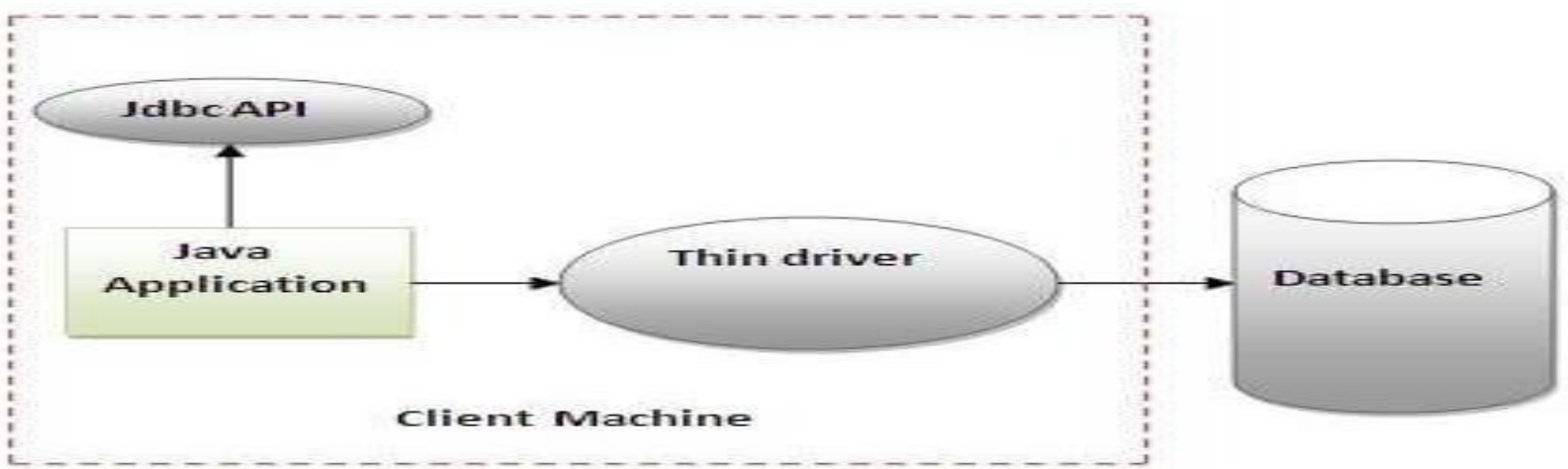
4. Thin driver (fully java driver)

Advantage:

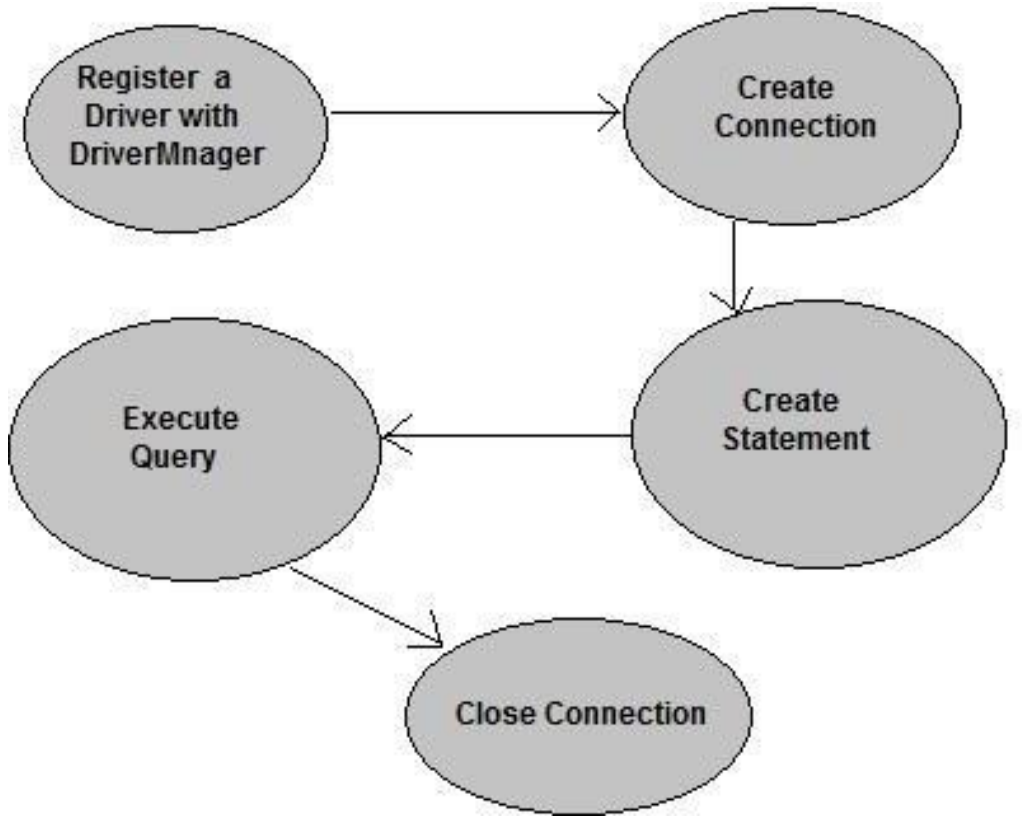
- Better performance than all other drivers.
- No software is required at client side or server side.

Disadvantage:

- Drivers depend on the Database.



- Register the Driver
- Create a Connection
- Create SQL Statement
- Execute SQL Statement
- Closing the connection



Database Connectivity with 5 Steps

1. Register the Driver class

- The **forName()** method of Class class is used to register the driver class. This method is used to dynamically load the driver class

  public static void forName(String className) **throws** ClassNotFoundException

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

- DriverManager.registerDriver(): DriverManager is a Java inbuilt class with a static member register. Here we call the constructor of the driver class at compile time. example uses DriverManager.registerDriver()to register the Oracle driver

  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver())

- The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

2. Create connection

- A Connection is the session between java application and database. The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData. The Connection interface provide many methods for transaction management like commit(), rollback() etc.
- The **getConnection()** method of DriverManager class is used to establish connection with the database.

public static Connection getConnection(String url)**throws** SQLException

public static Connection getConnection(String url, String name, String password) **throws** SQLException

{Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "system","password");}

- user – username from which your sql command prompt can be accessed.
  password – password from which your sql command prompt can be accessed.
- con: is a reference to Connection interface.

  url : Uniform Resource Locator. It can be created as follows:
- String url = " jdbc:oracle:thin:@localhost:1521:xe"
- @localhost is the IP Address where database is stored, 1521 is the port number and xe is the service provider.

3. Create SQL Statement.

Statement: Used to implement simple SQL statements with no parameters
You execute Statement objects, and they generate ResultSet objects, which is a table of data representing a database result set.
You need a Connection object to create a Statement Object

Statement is an interface under java.sql package and my understanding is that it is not possible to create an instance of an interface in Java

The driver's implementation of Connection has an implementation of the createStatement method that returns the driver's implementation of Statement.

There are three different kinds of statements

a. Create statement

- The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database

public Statement createStatement()**throws**

SQLException Statement stmt = con.createStatement();

b. PreparedStatement(Extends Statement)

It is used for precompiling SQL statements that might contain input parameters. This statement gives you the flexibility of supplying arguments dynamically

```
String SQL = "Update Employees SET age = ? WHERE id = ?";
pstmt = conn.prepareStatement(SQL)
```

- PreparedStatement interface: The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.

```
String sql="insert into emp values(?,?,?)";
```

- passing parameter (?) for the values will be set by calling the setter methods of PreparedStatement
- **Improves performance**: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.
- The prepareStatement() method of Connection interface is used to return the object of PreparedStatement. Syntax:

public PreparedStatement prepareStatement(String query)**throws** SQLException{}

c. CallableStatement: (Extends PreparedStatement.) Used to execute stored procedures that may contain both input and output parameters

```
DROP PROCEDURE IF EXISTS `EMP`.`getEmpName`
CREATE PROCEDURE `EMP`.`getEmpName` (IN EMP_ID INT, OUT EMP_FIRST VARCHAR(255))
BEGIN
  SELECT first INTO EMP_FIRST FROM Employees WHERE ID = EMP_ID;
END
```

- Three types of parameters exist: IN, OUT, and INOUT. The PreparedStatement object only uses the IN parameter. The CallableStatement object can use all the three

```
String SQL = "{call getEmpName (?, ?)}";
cstmt = conn.prepareCall (SQL);
```

- Using the CallableStatement objects is much like using the PreparedStatement objects. You must bind values to all the parameters before executing the statement, or you will receive an SQLException.
- If you have IN parameters, just follow the same rules and techniques that apply to a PreparedStatement object; use the setXXX() method that corresponds to the Java data type you are binding.
- When you use OUT and INOUT parameters you must employ an additional CallableStatement method, registerOutParameter(). The registerOutParameter() method binds the JDBC data type, to the data type that the stored procedure is expected to return

Commonly used methods of Connection interface

1) public Statement createStatement(): **creates a statement object that can be used to execute SQL queries.**
2) public Statement createStatement(int resultSetType,int resultSetConcurrency): **Creates a Statement object that will generate ResultSet objects with the given type and concurrency**
3) public void setAutoCommit(boolean status): **is used to set the commit status. By default it is true.**
4) public void commit(): **saves the changes made since the previous commit/rollback permanent.**
5) public void rollback(): **Drops all changes made since the previous commit/rollback**
6) public void close(): **closes the connection and Releases a JDBC resources immediately**

Statement interface

4. Execute SQL Statement

After creating statement, now execute using executeQuery() method of Statement interface. This method is used to execute SQL statements. Syntax of the method is given below.

public ResultSet executeQuery(String query) throws SQLException

Example:

```
ResultSet rs=s.executeQuery("select * from user");
  while(rs.next())
  {
   System.out.println(rs.getString(1)+" "+rs.getString(2));
 }
```

Statement Interface

The **Statement interface** provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

- Commonly used methods of Statement interface:

1) **public ResultSet executeQuery(String sql):** is used to execute SELECT query. It returns the object of ResultSet.
2) **public int executeUpdate(String sql):** is used to execute specified query, it may be create, drop, insert, update, delete etc
3) **public boolean execute(String sql):** is used to execute queries that may return multiple results.
4) **public int[] executeBatch():** is used to execute batch of commands

ResultSet interface

- The ResultSet interface can be three categories −
- **Navigational methods:** Used to move the cursor around.
- **Get methods:** Used to view the data in the columns of the current row being pointed by the cursor.
- **Update methods:** Used to update the data in the columns of the current row. The updates can then be updated in the underlying database as well.

Type of ResultSet

Type	Description
ResultSet.TYPE_FORW A RD_ONLY	The cursor can only move forward in the result set.
ResultSet.TYPE_SCROLL_ INSENSITIVE	The cursor can scroll forward and backward, and the result set is not sensitive to changes made by others to the database that occur after the result set was created.
ResultSet.TYPE_SCROLL_ SENSITIVE.	The cursor can scroll forward and backward, and the result set is sensitive to changes made by others to the database that occur after the result set was created.

If you do not specify any ResultSet type, you will automatically get one that is TYPE_FORWARD_ONLY.

Concurrency of ResultSet

Concurrency	Description
ResultSet.CONCUR_READ_ONLY	Creates a read-only result set. This is the default
ResultSet.CONCUR_UPDATABLE	Creates an updateable result set.

Statement stmt = conn.createStatement(ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_READ_ONLY);

createStatement(int RSType, int RSConcurrency);
prepareStatement(String SQL, int RSType, int RSConcurrency);
prepareCall(String sql, int RSType, int RSConcurrency);

If you do not specify any Concurrency type, you will automatically get one that is CONCUR_READ_ONLY.

Execute queries

- To execute a query, call an execute method such as the following:
- execute: Returns true if the first object that the query returns is a ResultSet object. Use this method if the query could return one or more ResultSet objects. Retrieve the ResultSet objects returned from the query by repeatedly calling Statement.getResultSet.
- 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.
- executeUpdate: Returns an integer representing the number of rows affected by the SQL statement. Use this method if you are using INSERT, DELETE, or UPDATE SQL statements.

Viewing a Result Set

- The ResultSet interface contains dozens of methods for getting the data of the current row. There is a get method for each of the possible data types, and each get method has two versions −
- One that takes in a column name.
- One that takes in a column index.

S.N.	Methods & Description
1	public int getInt(String columnName) throws SQLException Returns the int in the current row in the column named columnName.
2	public int getInt(int columnIndex) throws SQLException Returns the int in the current row in the specified column index. The column index starts at 1, meaning the first column of a row is 1, the second column of a row is 2, and so on.

Updating a Result Set

- The ResultSet interface contains a collection of update methods for updating the data of a result set.
  - One that takes in a column name.
  - One that takes in a column index.

S.N.	Methods & Description
1	public void updateString(int columnIndex, String s) throws SQLException Changes the String in the specified column to the value of s.
2	public void updateString(String columnName, String s) throws SQLException Similar to the previous method, except that the column is specified by its name instead of its index.

5. Close connection

- Close connection By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection
- **public void** close()**throws**

  SQLException Example

- con.close();

ResultSet interface

- The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row. **By default, ResultSet object can be moved forward only and it is not updatable**
- But we can make this object to move forward and backward direction by passing either TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE in createStatement(int,int) method as well as we can make this object as updatable by:
- Statement stmt =

  con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,

  ResultSet.CONCUR_UPDATABLE);

Useful methods of DriverManager class

Method	Description
1) public static void registerDriver(Driver driver):	is used to register the given driver with DriverManager.
2) public static void deregisterDriver(Driver driver):	is used to deregister the given driver (drop the driver from the list) with DriverManager.
3) public static Connection getConnection(String url):	is used to establish the connection with the specified url.
4) public static Connection getConnection(String url,String userName,String password):	is used to establish the connection with the specified url, username and password.

Commonly used methods of Connection interface:

1) **public Statement createStatement():** creates a statement object that can be used to execute SQL queries.
2) **public Statement createStatement(int resultSetType,int resultSetConcurrency):** Creates a Statement object that will generate ResultSet objects with the given type and concurrency.
3) **public void setAutoCommit(boolean status):** is used to set the commit status.By default it is true.
4) **public void commit():** saves the changes made since the previous commit/rollback permanent.
5) **public void rollback():** Drops all changes made since the previous commit/rollback.
6) **public void close():** closes the connection and Releases a JDBC resources immediately

methods of Statement interface

1) public ResultSet executeQuery(String sql): is used to execute SELECT query. It returns the object of ResultSet.

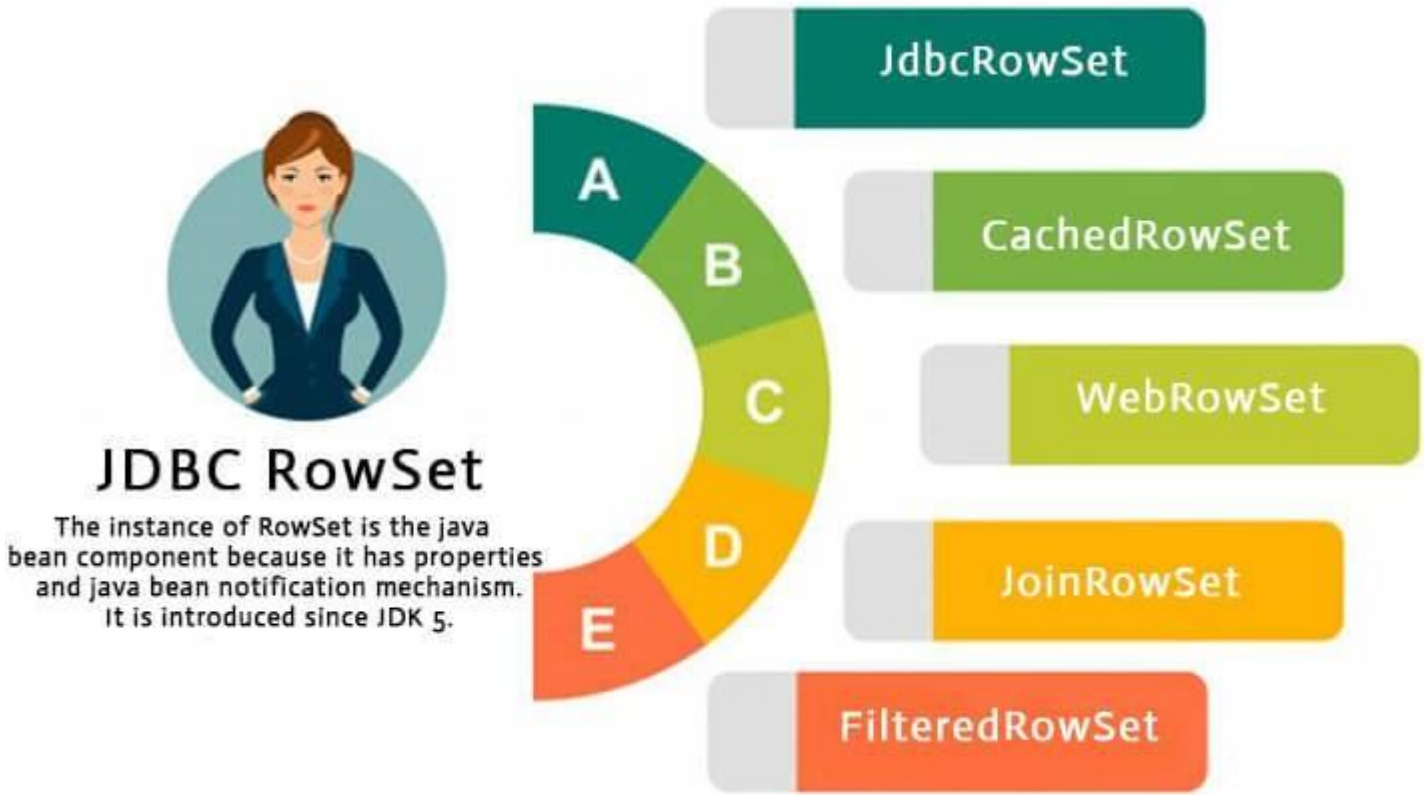
2) public int executeUpdate(String sql): is used to execute specified query, it may be create, drop, insert, update, delete etc.

3) public boolean execute(String sql): is used to execute queries that may return multiple results.

4) public int[] executeBatch(): is used to execute batch of commands.

difference between stored procedures and functions.

Stored Procedure	Function
is used to perform business logic.	is used to perform calculation.
must not have the return type.	must have the return type.
may return 0 or more values.	may return only one values.
We can call functions from the procedure.	Procedure cannot be called from function.
Procedure supports input and output parameters.	Function supports only input parameter.
Exception handling using try/catch block can be used in stored procedures.	Exception handling using try/catch can't be used in user defined functions.


JDBC RowSet
The instance of RowSet is the java bean component because it has properties and java bean notification mechanism. It is introduced since JDK 5.
A
B
C
D
E
JdbcRowSet
CachedRowSet
WebRowSet
JoinRowSet
FilteredRowSet

features of JDBC API 4.0

- **Automatic Loading of Driver class** You don't need to write Class.forName() now because it is loaded bydefault since jdbc4.
- **Subclasses of SQLException** Jdbc 4 provides new subclasses of SQLException class for better readability and handling.
- **New methods** There are many new methods introduced in Connection, PreparedStatement, CallableStatement, ResultSet etc.
- **Improved DataSource** Now data source implementation is improved.
- **Event Handling support in Statement for Connection Pooling** Now Connection Pooling can listen statement error and statement closing events.