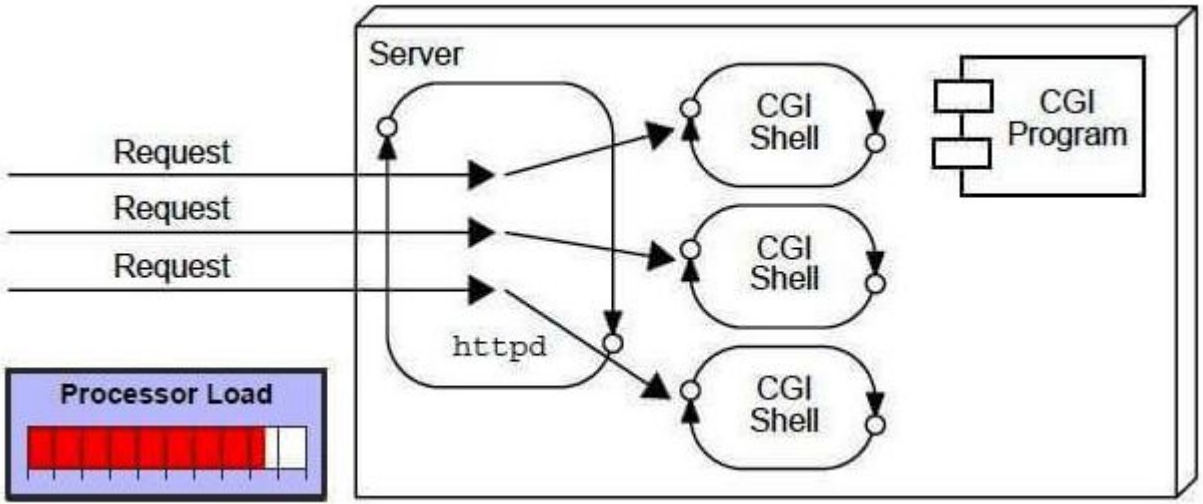
Servlet

Introduction to Servlet

- Servlet is a technology which is used to create a web application.
- Servlet is an API that provides many interfaces and classes including documentation.
- Servlet is an interface that must be implemented for creating any Servlet.
- Servlet(HttpServlet) is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
- Servlet is a web component that is deployed on the server to create a dynamic web page.

CGI (Common Gateway Interface)

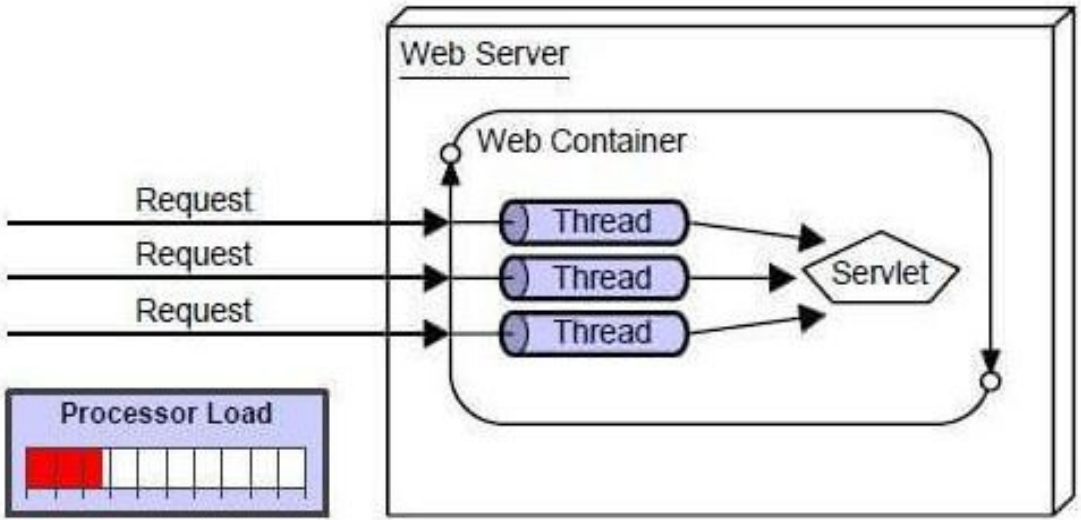
CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process



Disadvantages of CGI

There are many problems in CGI technology:

1. If the number of clients increases, it takes more time for sending the response.
2. For each request, it starts a process, and the web server is limited to start processes.
3. It uses platform dependent language e.g. C, C++, perl.


Web Server
Web Container
Request
Request
Request
Thread
Thread
Thread
Servlet
Processor Load

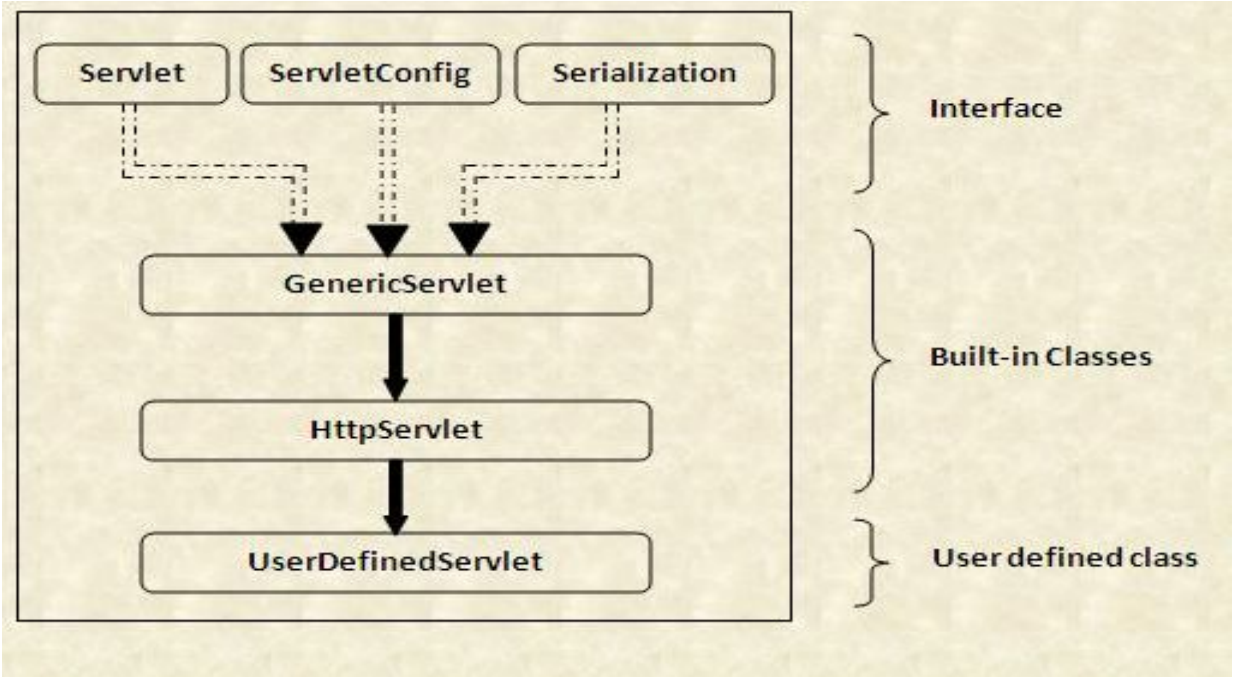
Advantages of Servlet

1. **Better performance:** because it creates a thread for each request, not process.
2. **Portability:** because it uses Java language.
3. **Robust:** JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.
4. **Secure:** because it uses java language.

Difference between Servlet and CGI

Servlet	CGI (Common Gateway Interface)
Servlets are portable and efficient.	CGI is not portable.
In Servlets, sharing data is possible.	In CGI, sharing data is not possible.
Servlets can directly communicate with the webserver.	CGI cannot directly communicate with the webserver.
Servlets are less expensive than CGI.	CGI is more expensive than Servlets.
Servlets can handle the cookies.	CGI cannot handle the cookies.

Hierarchy of Servlet



- The Servlet Class Hierarchy consists of two top level interfaces which are implemented by the GenericServlet class:
  - javax.servlet.Servlet
  - javax.servlet.ServletConfig
- The GenericServlet class is extended by the HttpServlet class which in turn is extended by a user defined class.

Servlet Interface

- Servlet interface provides common behavior to all the servlets. Servlet interface defines methods that all servlets must implement.
- Servlet interface needs to be implemented for creating any servlet
- 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.

Methods of Servlet interface:

1. public void init(ServletConfig config)
2. public void service(ServletRequest request,ServletResponse response)
3. public void destroy()
4. public ServletConfig getServletConfig()
5. public String getServletInfo()

HttpServlet class

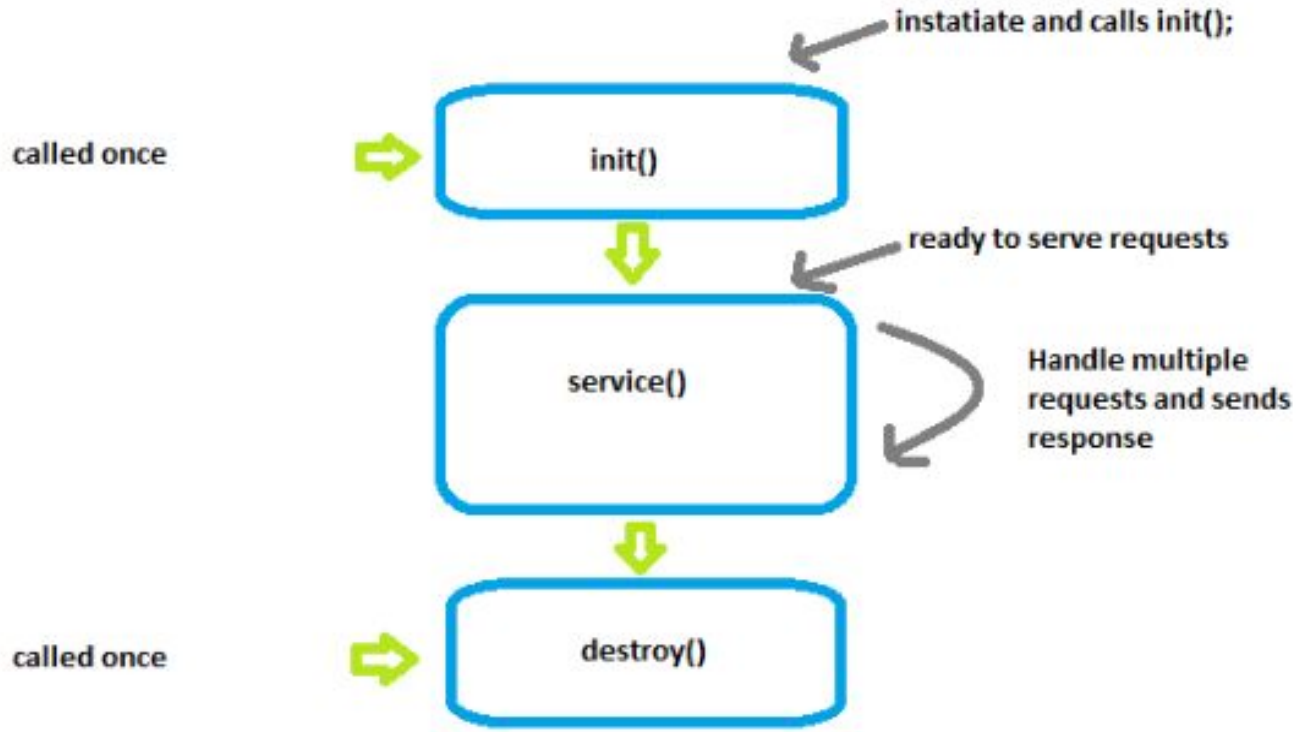
The HttpServlet class extends the GenericServlet class and implements Serializable interface. It provides http specific methods such as doGet, doPost, doHead, doTrace etc.

Methods of HttpServlet class:

1. **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.
2. **protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
3. **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.

- **protected void doOptions(HttpServletRequest req, HttpServletResponse res)** handles the OPTIONS 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.
- **protected void doTrace(HttpServletRequest req, HttpServletResponse res)** handles the TRACE request. It is invoked by the web container.
- **protected void doDelete(HttpServletRequest req, HttpServletResponse res)** handles the DELETE request. It is invoked by the web container.
- **protected long getLastModified(HttpServletRequest req)** returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

Servlet Life Cycle



Stages of the Servlet Life Cycle:

The Servlet life cycle mainly goes through four stages,

- Loading a Servlet.
- Initializing the Servlet.
- Request handling.
- Destroying the Servlet.

Loading a Servlet: The first stage of the Servlet lifecycle involves loading and instantiating the Servlet by the Servlet container.

The Servlet container performs two operations in this stage :

- **Loading :** Loads the Servlet class.
- **Instantiation :** Creates an instance of the Servlet. To create a new instance of the Servlet, the container uses the no-argument constructor.

Initializing a Servlet: After the Servlet is instantiated successfully, the Servlet container initializes the instantiated Servlet object. The container initializes the Servlet object by invoking the **Servlet.init(ServletConfig)** method which accepts ServletConfig object reference as parameter.

The Servlet container invokes the **Servlet.init(ServletConfig)** method only once, immediately after the **Servlet.init(ServletConfig)** object is instantiated successfully. This method is used to initialize the resources, such as JDBC datasource.

Now, if the Servlet fails to initialize, then it informs the Servlet container by throwing the **ServletException** or **UnavailableException**.

init()

```
public class MyServlet implements Servlet
{
   public void init(ServletConfig config) throws ServletException
{
        //initialization code
   }
    //rest of code
   }
```

Handling request: After initialization, the Servlet instance is ready to serve the client requests. The Servlet container performs the following operations when the Servlet instance is located to service a request :

- It creates the **ServletRequest** and **ServletResponse** objects. In this case, if this is a HTTP request, then the Web container creates **HttpServletRequest** and **HttpServletResponse** objects which are subtypes of the **ServletRequest** and **ServletResponse** objects respectively.

- After creating the request and response objects it invokes the Servlet.service(ServletRequest, ServletResponse) method by passing the request and response objects.

The **service()** method while processing the request may throw the **ServletException** or **UnavailableException** or **IOException**.

service()

```
public class MyServlet implements Servlet
{
    public void service(ServletRequest req, ServletResponse res)
    throws ServletException, IOException
{
            // request handling code
    }
    // rest of code
   }
```

doGet and doPost are methods of the **javax.servlet.http.HttpServlet** class that are used to handle HTTP GET and POST requests, respectively.

The doGet method is called by the server (via the service method) when the client requests a GET request. It is used to retrieve information from the server.

The doPost method is called by the server (via the service method) when the client requests a POST request. It is used to send information to the server.

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

public class MyServlet extends HttpServlet {

  @Override
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
    // handle GET request
  }

  @Override
  protected void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
    // handle POST request
  }
}
```

Destroying a Servlet: When a Servlet container decides to destroy the Servlet, it performs the following operations,

- It allows all the threads currently running in the service method of the Servlet instance to complete their jobs and get released.
- After currently running threads have completed their jobs, the Servlet container calls the **destroy()** method on the Servlet instance.

After the **destroy()** method is executed, the Servlet container releases all the references of this Servlet instance so that it becomes eligible for garbage collection.

destroy()

```
public void destroy()
   {
   //Finalization code
   }
```

As soon as the **destroy()** method is activated, the Servlet container releases the Servlet instance.

Apache

- Apache is a *very* popular server
- 66% of the web sites on the Internet use Apache
- Apache is:
- Full-featured and extensible
- Efficient
- Robust
- Secure (at least, more secure than other servers)
- Up to date with current standards
- Open source
- Free

Tomcat

- Tomcat is the Servlet Engine than handles servlet requests for Apache
- Tomcat is a “helper application” for Apache
- It’s best to think of Tomcat as a “servlet container”
- Apache can handle many types of web services
- Apache can be installed without Tomcat
- Tomcat can be installed without Apache
- It’s easier to install Tomcat standalone than as part of Apache
- By itself, Tomcat can handle web pages, servlets, and JSP
- Apache and Tomcat are open source (and therefore free)

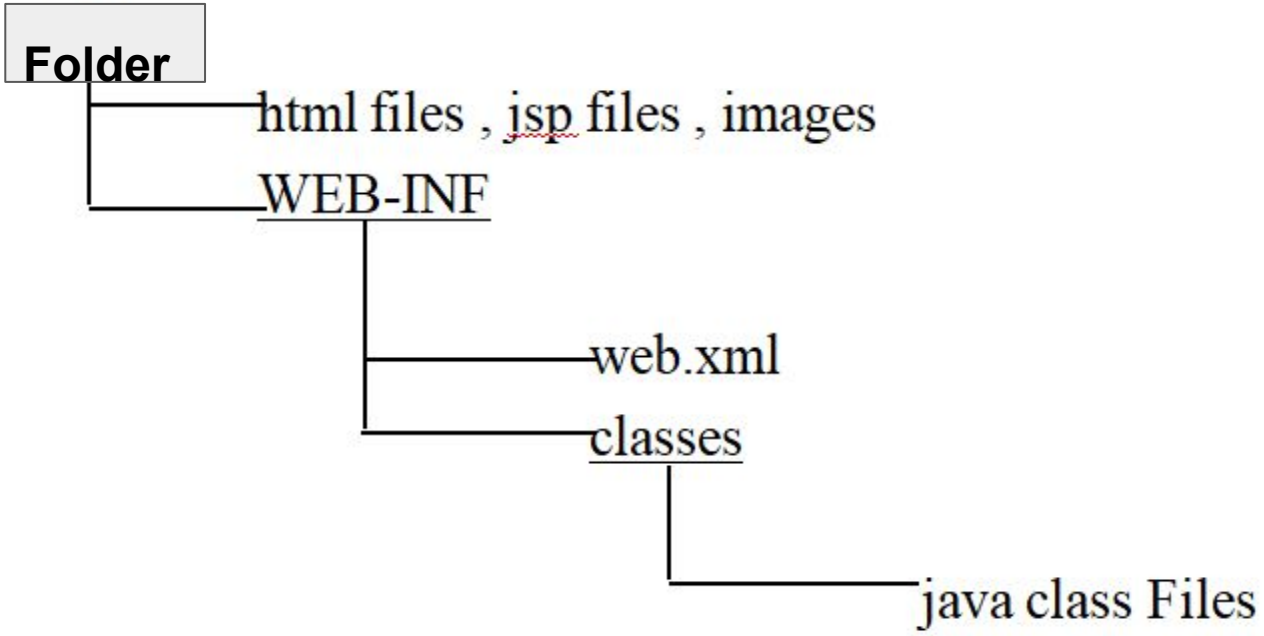
Creating Servlet

Here, we are going to use **apache tomcat server** in this example. The steps are as follows:

1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the project
6. Access the servlet

Directory Structure

cd /usr/java/jdk1.5.0/appache-tomcat-5.5.12/webapps/



To start Tomcat

To start:

cd /usr/java/jdk1.5.0/appache-tomcat-5.5.12/bin>
sh startup.sh than enter

To Check:

Open mozilla web browser
http://localhost:8080 than enter

To Shutdown Tomcat:

cd /usr/java/jdk1.5.0/appache-tomcat-5.5.12/bin>
sh shutdown.sh than enter

Steps to running first servlet

After, you have installed & configured Tomcat, you can put it into service. Basically, you need to follow six steps to go from writing your servlet to running it.

- Create a directory structure under Tomcat for your application.
- Write the servlet source code, you need to import the javax.servlet.* and javax.servlet.http.* packages in your source file.
- Compile your source code.
- Create a deployment descriptor
- Run Tomcat
- Call your servlet from a web browser.

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

public class Hello extends HttpServlet
{
        public void init() throws ServletException
{
        output = "Hello Pune!!!";
}
public void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException
        {
                resp.setContentType("text/html");
                PrintWriter out = resp.getWriter();
                out.println(output);
        }

        public void destroy()
        {
                System.out.println("Over");
        }
}
```

<web-app>

<servlet>

<servlet-name>H</servlet-name>

<servlet-class>Hello</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>H</servlet-name>

<url-pattern>/Hellopune</url-pattern>

</servlet-mapping>

</web-app>

Reading Data from a client

A client uses either the GET or POST method to pass information to the java servlet.

- The doGet() or doPost() method is called in the java servet depending on the method used by the client.
- Input parameters are retrieved via messages to the HttpServletRequest object
- **public Enumeration getParameterNames()**

  Returns an Enumeration of the parameter names

  If no parameters, returns an empty Enumeration
- **public String getParameter(String name)**

  Returns the value of the parameter name as a String

  If the parameter doesn't exist, returns null

  If name has multiple values, only the first is returned
- **public String[] getParameterValues(name)**

  Returns an array of values of the parameter name

  If the parameter doesn't exist, returns null

n

Session Tracking

- As we know that the Http is a stateless protocol, means that is cannot persist the information.
- It always treats each request as a new request.
- In http client makes a connection to the server, sends the request, gets the response & closes the connection.
- Session tracking can be done in four ways.

1. Hidden form fields
2. url rewriting
3. Cookies
4. HttpSession

Cookies

- A cookie is a small amount of data that is saved on the client's machine and can be created by referenced by a java servlet using the java servlet cookie.
- A cookie is composed of two pieces.
- These are the cookie name & the cookie value, both of which are created by the java servlet.
- The cookie name is used to identify a particular cookie from among other cookies stored at the client.
- The cookie value is data associated with the cookie.
- The java servlet writes a cookie by passing the constructor of the cookie object two arguments.

First argument is string object that contains the name of the cookies.

The other argument a string object that contains the value of the cookie

Constructor of Cookie:

- Cookie()
- Cookie(String name, String value)

Methods of Cookie:

- **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.

- **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.

- Ex.

Cookie myC = new Cookie(“firstname”,”Pooja”);

- Once a cookie is created, the java servlet must insert the cookie into the http response header by calling the addCookie() method of the HttpServletResponse object.
- That is

response.addCookie(myC);

- You can read a cookie by calling the getCookies() method of the HttpServeltRequest object.
- The getCookies() method returns an array of cookie objects.

```
<html>
<head>
    <title>Create Cookie Form</title>
</head>
<body>
    <h2>Create Cookie Form</h2>
    <form action="createCookieServlet" method="get">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username">
        <br><br>
        <input type="submit" value="Create Cookie">
    </form>
</body>
</html>
```

```
import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.*;

public class CreateCookieServlet extends HttpServlet {

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

      String username = request.getParameter("username");

      Cookie cookie = new Cookie("username", username);

      cookie.setMaxAge(24 * 60 * 60);

      response.addCookie(cookie);

      response.getWriter().println("Cookie for username created successfully!");

   }

}
```

HttpSession API

- The HttpSession API uses the HttpSession object that is associated with the HttpSevletRequest to determine if the request is a continuation from an existing session or is a new session.
- A java servlet calls the getSession ( ) method of the HttpServletRequest object, which returns an HttpSession object.

i. e. HttpSession s1 = request.getSession (boolean);

- If the getSession ( ) method doesn't use an argument, then the getSession() method returns either the existing session or creates a new session and returns the session object.
- If the getSession () method takes a boolean true as an argument, then the getSession () method returns the current session as a session object. If the current session doesn't exist, the getSession () method creates a new session & return a session object.
- If the getSession ( ) method takes a boolean false as an argument, the getSession() method returns the current session as a session object. If the current session doesn't exist, the getSesion() method returns a null

- An HttpSession object contains a data structure that is used to store key and a value associated with each key.
- A key is an attribute of the session.
- You can read a value of an attribute by calling the HttpSession object's getAttribute () method, which requires one argument, which is a String object that contains one argument of the attribute whose value you want to retrieve.
- The getAttribute ( ) returns the value of the attribute, which is an String object.
- We can modify the value of an attribute by calling setAttribute ( ) method, which requires two arguments.
- The first argument is a String object that contains the name of the attribute.
- The other argument is an String object that contains value of the attribute.

The HttpServletRequest interface provides two methods to get the object of HttpSession:

1. **public HttpSession getSession():**Returns the current session associated with this request, or if the request does not have a session, creates one.
2. **public HttpSession getSession(boolean create):**Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

1. **public String getId():**Returns a string containing the unique identifier value.
2. **public long getCreationTime():**Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
3. **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.
4. **public void invalidate():**Invalidates this session then unbinds any objects bound to it.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SessionExample extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)    throws ServletException, IOException {
        HttpSession session = request.getSession();
       session.setAttribute("username", "Pooja");
        session.setAttribute("email", "pooja@gmail.com");
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("<h1>Session Created</h1>");
        out.println("<p>Username: " + session.getAttribute("username") + "</p>");
        out.println("<p>Email: " + session.getAttribute("email") + "</p>");
        out.println("</body></html>");
    }
```