# Java and J2EE (17IS6DEJVA)

Unit 5

# J2SE(Java Platform, Standard Edition)

- Also known as Core Java, this is the most basic and standard version of Java.I t's the purest form of Java, a basic foundation for all other editions.
- It consists of a wide variety of general purpose API's (like java.lang, java.util) as well as many
  special purpose APIs.
- J2SE is mainly used to create applications for Desktop environment.
- It consist all the basics of Java the language, variables, primitive data types, Arrays, Streams, Strings Java Database Connectivity(JDBC) and much more. This is the standard, from which all other editions came out, according to the needs of the time.
- The famous JVM of Java, the heart of Java development, was also given by this edition only. It's
  because of this feature, that Java has such a wide usage.

# J2ME(Java Platform, Micro Edition)

- This version of Java is mainly concentrated for the applications running on embedded systems, mobiles and
small devices(which was a constraint before it's development).

- Constraints included limited processing power, battery limitation, small display etc.

- Also, the J2ME apps help in using web compression technologies, which in turn, reduce network usage, and hence cheap internet accessibility.

- J2ME uses many libraries and API's of J2SE, as well as, many of it's own.

- The basic aim of this edition was to work on mobiles, wireless devices, set top boxes etc.

- Most of the apps, developed for the phones (prior to smartphones era), were built on J2ME platform only(the .jar apps on Nokia app store).

# J2EE(Java Platform, Enterprise Edition)

- The Enterprise version of Java has a much larger usage of Java, like development of web services, networking, server side scripting and other various web based applications.
- J2EE is a community driven edition, i.e., there is a lot of continuous contributions from industry
  experts, Java developers and other open-source organizations.
- J2EE uses many components of J2SE, as well as, has many new features of it's own like Servlets, JavaBeans, Java Message Services, adding a whole new functionalities to the language.
- J2EE uses HTML, CSS, JavaScript etc., so as to create web pages and web services. It's also one of the most widely accepted web development standard.
- There are also many languages like .net and php, which can do that work, but what distinguishes it from other languages is the versatility, compatibility and security features, which are not that much prominent in other language

| JAVA | J2EE |
| --- | --- |
| Java is an acronym for Java Standard Edition (Java SE) which is used to better describe Core Java. | J2EE stands for Java 2 Platform, Enterprise Edition which is currently known as Java Platform, Enterprise Edition or simply "Java EE". |
| Java is a high-level programming language which derives much of its syntax from C and C++. | J2EE is a computing platform based on Java which is basically an extension of the Java Standard Edition (Java SE). |
| It is primarily used for developing desktop applications. | It is mainly used for developing multi-tiered web-based enterprise applications. It can be used for both desktop and web applications. |
| It is an OOP based language which simplifies software development. | It is a collection of Java APIs which target enterprise technologies such as EJBs, Servlets, JSPs, etc. |
| This is the beginning of Core Java which starts with the basic concepts of Java. | This is the next level of Java which implements two-tier architecture (client and server). |

# Background of J2EE

- Client/server architecture exploded from a two-tier architecture to a multi-tier architecture, where a client's request to a server generates requests to other servers that are connected together through a backbone network.

  - This is very similar to you asking a travel agent to arrange your vacation. The travel agent contacts hotels, airlines, the car rental company, restaurants, and other vendors that are necessary to fulfill your request.

- Although a multi-tier architecture provides services efficiently, it also makes it complex to design, create, debug, distribute, and maintain an application because a programmer must be assured that all tiers work together. However, the Java development team enhanced the capabilities of Java to dramatically reduce the complexity of developing a multi-tier application.

# Why J2EE?

- J2EE simplifies the creation of an enterprise-wide application, because functionality is encapsulated in components of J2EE. This enables designers and programmers to organize the application into functionality that is distributed across the server-side components built using J2EE.

- J2EE is a versatile technology because application components built using J2EE are able to communicate with each other behind the scenes using standard communications methods such as HTTP, SSL, HTML, XML, RMI, and IIOP.

- Corporations no longer need to find programmers to write programs to interface with a vendor- specific component. This also shortens the development cycle for complex programs that require multithreading and synchronization, because J2EE contains all the interfaces and libraries that handle these complex issues.

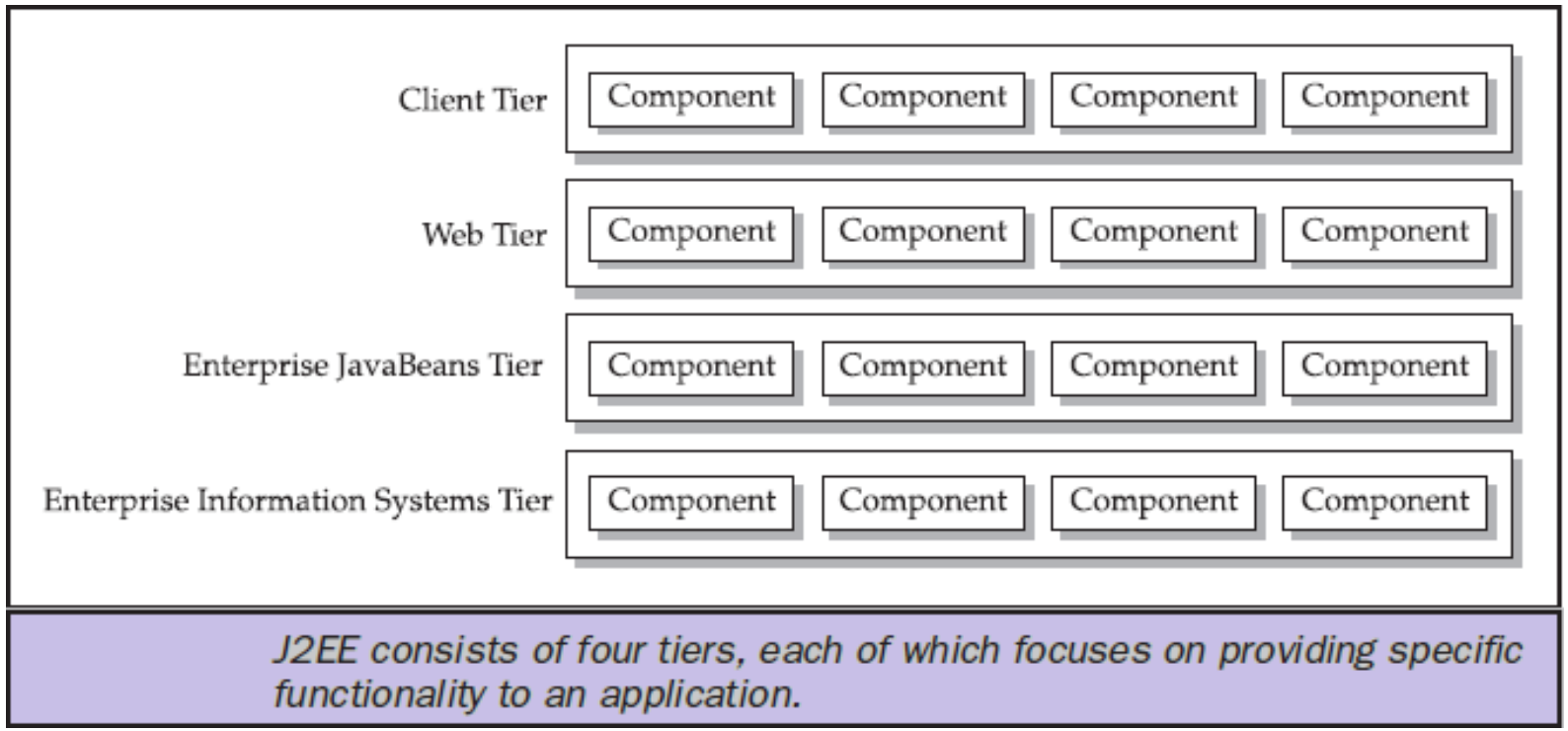
# J2EE Multi-Tier

- J2EE is a four-tier architecture. These consist of
  - the Client Tier (sometimes referred to as the Presentation Tier or Application Tier),
  - Web Tier,
  - Enterprise JavaBeans Tier (sometimes referred to as the Business Tier), and
  - the Enterprise Information Systems Tier.
  - Each tier is focused on providing a specific type of functionality to an application.

- Since the J2EE multi-tier architecture is functionally centric, a J2EE application accesses only tiers whose functionality is required by the J2EE application.

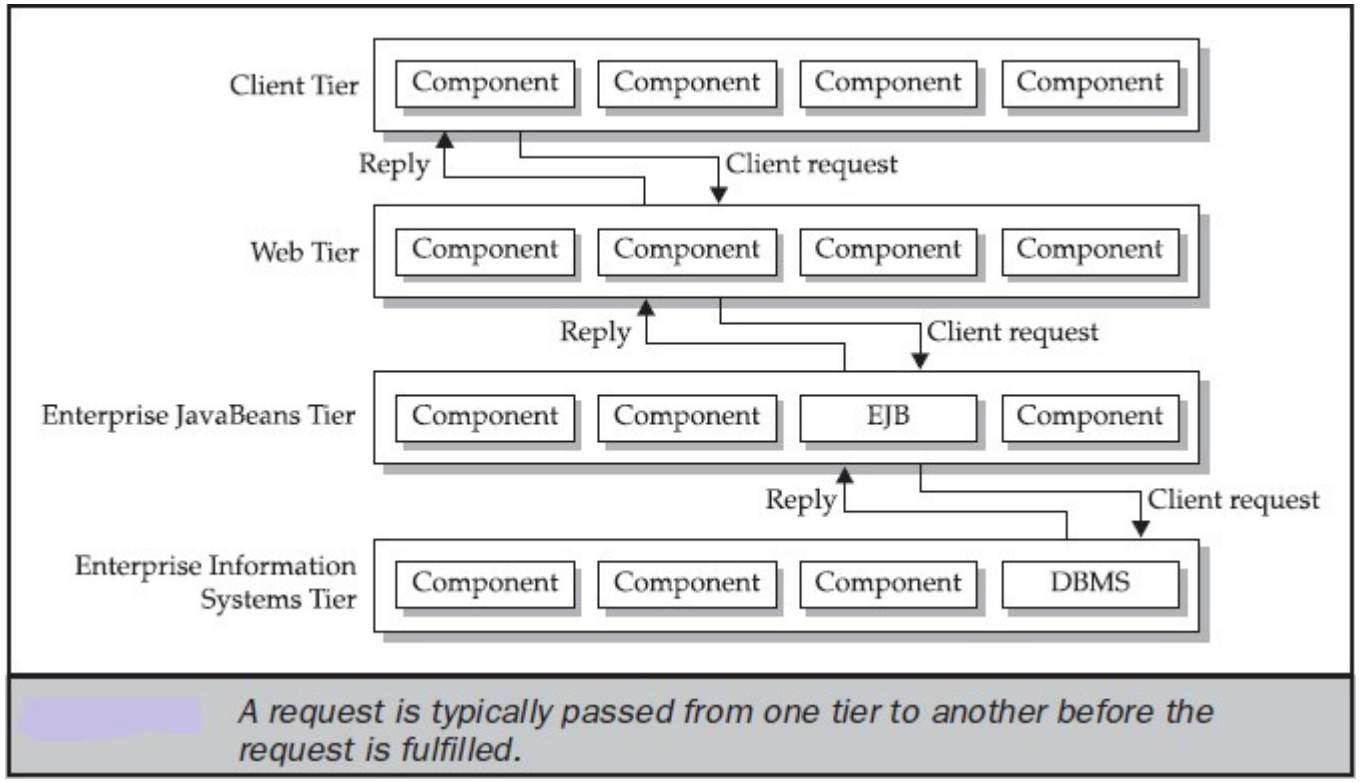
- It's also important to disassociate a J2EE API with a particular tier.
- That is, some APIs (i.e., XML API) and J2EE components can be used on more than one tier, while other APIs (i.e., Enterprise JavaBeans API) are associated with a particular tier.

# J2EE Multi-Tier




*J2EE consists of four tiers, each of which focuses on providing specific functionality to an application.*

# J2EE Multi-Tier Contd




*A request is typically passed from one tier to another before the request is fulfilled.*

# J2EE Multi-Tier Contd

- The Client Tier consists of programs that interact with the user.
  - These programs prompt the user for input and then convert the user's response into requests that are forwarded to software on a component that processes the request and returns results to the client program.
  - The client program also translates the server's response into text and screens that are presented to the user.

- The Web Tier provides Internet functionality to a J2EE application.
  - Components that operate on the Web Tier use HTTP to receive requests from and send responses to clients that could reside on any tier.

# J2EE Multi-Tier Contd

- The Enterprise JavaBeans Tier contains the business logic for J2EE applications.

- The Enterprise JavaBeans Tier is the keystone to every J2EE application because Enterprise JavaBeans working on this tier enable multiple instances of an application to concurrently access business logic and data so as not to impede the performance.

- Enterprise JavaBeans are contained on the Enterprise JavaBeans server, which is a distributed object server that works on the Enterprise JavaBeans Tier and manages transactions and security, and assures that multithreading and persistence are properly implemented whenever an Enterprise JavaBean is accessed.

- Typically an Enterprise JavaBean accesses components and resources such as DBMS on the
  Enterprise Information System (EIS) Tier.

# J2EE Multi-Tier Contd

# J2EE Multi-Tier Contd

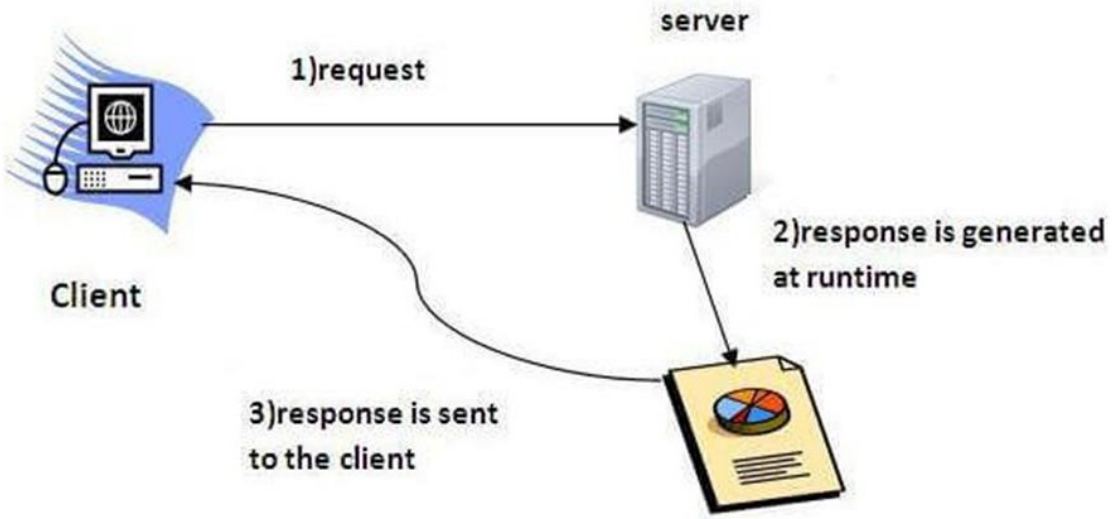
- Access Control List (ACL) is a critical design element in the J2EE multi-tier architecture that controls communication between tiers.
  - ACL bridges tiers that are typically located on different virtual local area networks
  - ACL adds a security level to web applications.
  - ACL prevents direct access to DBMS and similar resources.

# Servlets

# What is a

- 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.
- Servlet is a simple java program which runs on the server and handles http request response
- Although servlets can respond to any type of request, they are commonly used to extend the
  applications hosted by web servers.
- It is also a web component that is deployed on the server to create a dynamic web page.

# What is a


server
1)request
2)response is generated
at runtime
Client
3)response is sent
to the client

# CGI (Common Gateway

- 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

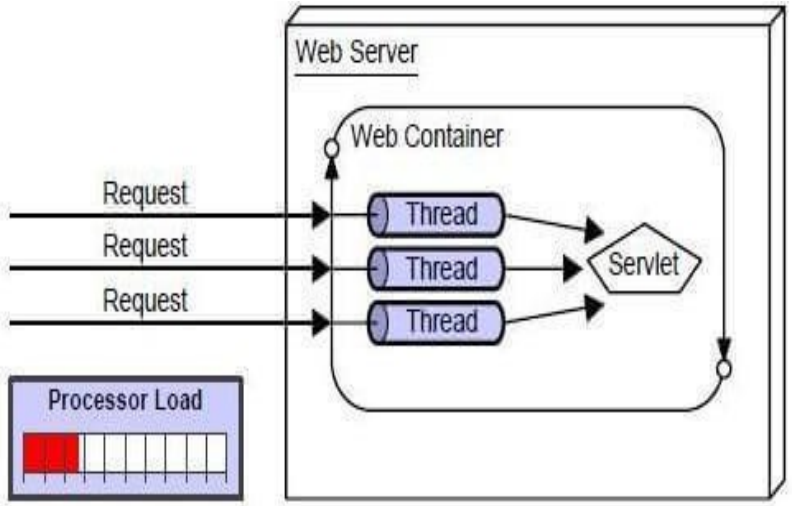
- There are many problems in CGI technology:
  - If the number of clients increases, it takes more time for sending the response.
  - For each request, it starts a process, and the web server is limited to start processes.
  - It uses platform dependent language e.g. C, C++, perl.

# Advantages of Servlet

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

The advantages of Servlet are as follows:

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



# Servlets:

- Web Application: The application which is executed in web environment or Internet environment in the process of interacting with end users through web browsers.

  To develop web application: we require
  (a)Servlets
  (b)JDBC
  (c) JSP

  Servlet programs are used to extend the functionality of web servers in the process of interacted with end users through web browsers
  When we want to execute servlet programs, then we have to install one web server. i.e. tom cat server
  Web servers is used to run web components developed through servlets and JSP, which is having a web container. Web container is used to execute web programming components.

(i)

then loads and instantiates the object of class file at (4). Once the object is loaded and instantiates, then a servlet configuration is created to hold the configuration details.

Lifecycle of Servlet:-

(1) Loading Process
(2) Instantiation Process
(3) Initialization process – Initialization Process
(4) Request process → Service(): program is ready to accept the request
(5) Destroy Process: completion of program.

(i)

then loads and instantiates the object of class file at (4). Once the object is loaded and instantiates, then a servlet configuration is created to hold the configuration details.

Lifecycle of Servlet:-

(1) Loading Process
(2) Instantiation Process
(3) Initialization process – Initialization Process
(4) Request process → Service(): program is ready to accept the request
(5) Destroy Process: completion of program.

# Introductio

- Servlet is Java EE server driven technology to create web applications in java. The **javax.servlet** and **javax.servlet.http** packages provide interfaces and classes for writing our own servlets.

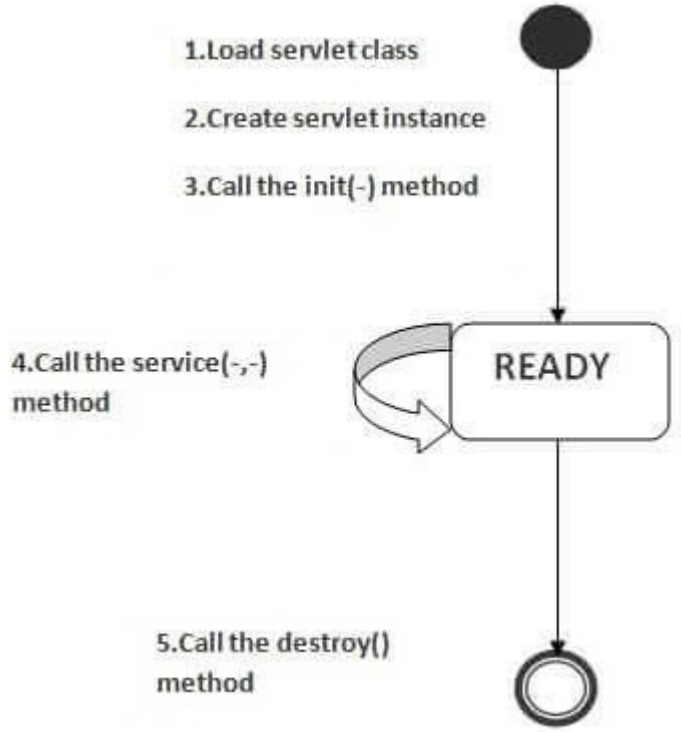
- All servlets must implement the javax. servlet. Servlet interface, which defines servlet lifecycle methods. When implementing a generic service, we can extend the GenericServlet class provided with the Java Servlet API. The HTTP Servlet class provides methods, such as doGet() and doPost(), for handling HTTP-specific services.

- Most of the times, web applications are accessed using HTTP protocol and thats why we mostly extend HttpServlet class.

# Life Cycle of a Servlet (Servlet Life Cycle)

The web container maintains the life cycle of a servlet instance. The life cycle of the servlet:

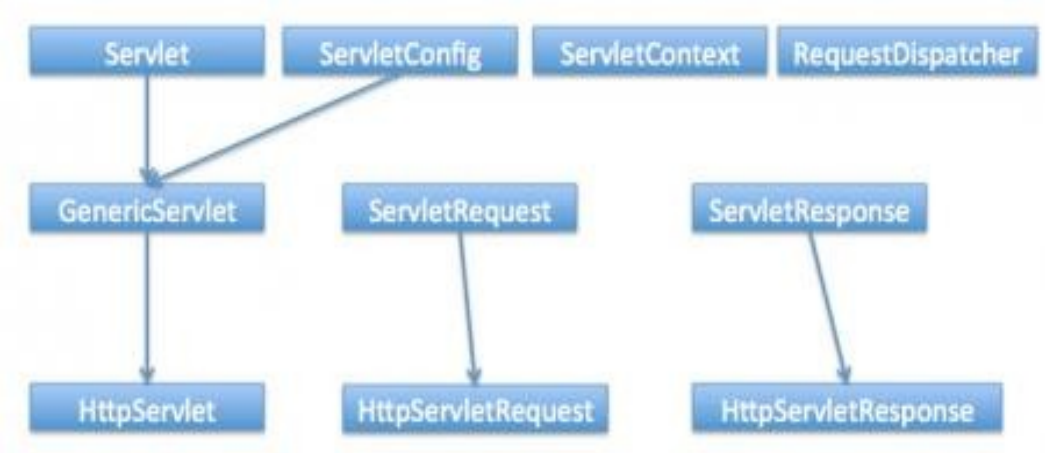
1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.




As displayed in the diagram, there are three states of a servlet: **new, ready and end.**

The servlet is in new state if servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state.

# Servlet API Hierarchy



# Servlet Contd…

- javax.servlet.Servlet is the base interface of Java Servlet API. Servlet interface declares the life cycle methods of servlet. The methods declared in this interface are:
  - **public abstract void init(ServletConfig paramServletConfig) throws ServletException** – This is the very important method that is invoked by servlet container to initialized the servlet and ServletConfig parameters. This method is called only once in servlet lifecycle and make Servlet class different from normal java objects.
  - **public abstract ServletConfig getServletConfig()** – This method returns a servlet config object, which contains any initialization parameters and startup configuration for this servlet. We can use this method to get the init parameters of servlet defines in deployment descriptor (web.xml) or through annotation in Servlet 3.
  - **public abstract void service(ServletRequest req, ServletResponse res)** throws ServletException, IOException – This method is responsible for processing the client request. Whenever servlet container receives any request, it creates a new thread and execute the service() method by passing request and response as argument.
  - **public abstract String getServletInfo()** – This method returns string containing information about the servlet, such as its author, version, and copyright. The string returned should be plain text and can't have markups.
  - **public abstract void destroy()** – This method can be called only once in servlet life cycle and used to close any open resources. This is like finalize method of a java class.

# Servl Contd

- **ServletConfig Interface**
  - javax.servlet.ServletConfig is used to pass configuration information to Servlet. Every servlet has it's own ServletConfig object and servlet container is responsible for instantiating this object. We can provide servlet init parameters in web.xml file or through use of WebInitParam annotation. We can use getServletConfig() method to get the ServletConfig object of the servlet.
  - The important methods of ServletConfig interface are:
    - public abstract ServletConfig getServletConfig() – This method returns the ServletConfig
      object for the servlet. We will look into ServletContext interface in next section.
    - public abstract String getInitParameter(String paramString) – This method can be used to get the specific init parameter value by name. If parameter is not present with the name, it returns null.
    - It has specific value for the specific servlet

# Servl Contd

- **ServletContext interface**
  - javax.servlet.ServletContext interface provides access to web application variables to the servlet. The ServletContext is unique object and available to all the servlets in the web application. When we want some init parameters to be available to multiple or all of the servlets in the web application, we can use ServletContext object and define parameters in web.xml using <context-param> element. We can get the ServletContext object via the getServletContext() method of ServletConfig.
  - Some of the important methods of ServletContext are:
    - **public abstract ServletContext getContext(String uripath)** – This method returns ServletContext object for a particular uripath or null if not available or not visible to the servlet.
    - public abstract ServletContext getServletContext() – This method returns the ServletContext
      object for the servlet. We will look into ServletContext interface in next section.
    - public abstract String getInitParameter(String paramString) – This method can be used to get the specific init parameter value by name. If parameter is not present with the name, it returns null.
    -

# Servl Contd

- **public abstract Object getAttribute (String name)** – Return the object attribute for the given name. We can get enumeration of all the attributes using **public abstract Enumeration<String> getAttributeNames ()** method.
- **public abstract void setAttribute (String paramString, Object Param Object)** – This method is used to set the attribute with application scope. The attribute will be accessible to all the other servlets having access to this ServletContext. We can remove an attribute using **public abstract void removeAttribute (String paramString)** method.
- **String getInitParameter (String name)** – This method returns the String value for the init parameter defined with name in web.xml, returns null if parameter name doesn't exist. We can use **Enumeration<String> getInitParameterNames()** to get enumeration of all the init parameter names.
- **boolean setInitParameter (String paramString1, String paramString2)** – We can use this method to set init parameters to the application.

- **Note: Ideally the name of this interface should be ApplicationContext because it's for the application and not specific to any servlet. Also don't get confused it with the servlet context passed in the URL to access the web application.**

# Servl Contd

- ServletRequest interface
  - ServletRequest interface is used to provide client request information to the servlet. Servlet container creates ServletRequest object from client request and pass it to the servlet service() method for processing.
  - Some of the important methods of ServletRequest interface are:
    - Object getAttribute (String name) – This method returns the value of named attribute as Object and null if it's not present. We can use getAttributeNames () method to get the enumeration of attribute names for the request. This interface also provides methods for setting and removing attributes.
    - String getParameter (String name) – This method returns the request parameter as String. We can use getParameterNames () method to get the enumeration of parameter names for the request.
    - String getServerName () – returns the hostname of the server.
    - int getServerPort () – returns the port number of the server on which it's listening.
    - The child interface of ServletRequest is HttpServletRequest that contains some other methods for session management, cookies and authorization of request.

# Servl Contd

- **ServletResponse interface**
  - ServletResponse interface is used by servlet in sending response to the client. Servlet container creates the ServletResponse object and pass it to servlet service() method and later use the response object to generate the HTML response for client.
  - Some of the important methods in HttpServletResponse are:
    - **void addCookie (Cookie cookie)** – Used to add cookie to the response.
    - **void addHeader (String name, String value)** – used to add a response header with the given name and value.
    - **String encodeURL (java.lang.String url)** – encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged.
    - **String getHeader (String name)** – return the value for the specified header, or null if this header has not been set.
    - **void sendRedirect (String location)** – used to send a temporary redirect response to the client using the specified redirect location URL.
    - **void setStatus (int sc)** – used to set the status code for the response.

# Servl Contd

- **RequestDispatcher interface**
  - RequestDispatcher interface is used to forward the request to another resource that can be HTML, JSP or another servlet in the same context. We can also use this to include the content of another resource to the response. This interface is used for servlet communication within the same context.
  - There are two methods defined in this interface:
    - **void forward (ServletRequest request, ServletResponse response)** – forwards the request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
      .
  - We can get RequestDispatcher in a servlet using

    ServletContext *getRequestDispatcher (String path)* method. The path must begin with a / and is interpreted as relative to the current context root.

# Servl Contd

- **GenericServlet class**
  - GenericServlet is an **abstract class** that implements Servlet, ServletConfig and Serializable interface. GenericServlet provide default implementation of all the Servlet life cycle methods and ServletConfig methods and makes our life easier when we extend this class, we need to override only the methods we want and rest of them we can work with the default implementation. Most of the methods defined in this class are only for easy access to common methods defined in Servlet and ServletConfig interfaces.

  - One of the important method in GenericServlet class is no-argument init() method and we should override this method in our servlet program if we have to initialize some resources before processing any request from servlet.

# Servl Contd

- **HTTP Servlet class**
  - HTTP Servlet is an abstract class that extends GenericServlet and provides the base for creating HTTP based web applications. There are methods defined to be overridden by subclasses for different HTTP methods.
    - doGet(), for HTTP GET requests
    - doPost(), for HTTP POST requests
    - doPut(), for HTTP PUT requests
    - doDelete(), for HTTP DELETE requests

# Servl Contd

- **Servlet Attributes**
  - Servlet attributes are used for inter-servlet communication, we can set, get and remove attributes in web application. There are three scopes for servlet attributes – **request scope**, **session scope** and **application scope**.
  - **ServletRequest**, **HTTP Session** and **ServletContext** interfaces provide methods to get/set/remove attributes from request, session and application scope respectively.
  - Servlet attributes are different from init parameters defined in **web.xml** for ServletConfig or ServletContext.

# Servl Contd

- Web Application: The application which is executed in web environment or Internet environment in the process of interacting with end users through web browsers.

  To develop web application: we require
  (a)Servlets
  (b)JDBC
  (c) JSP

  Servlet programs are used to extend the functionality of web servers in the process of interacted with end users through web browsers
  When we want to execute servlet programs, then we have to install one web server. i.e. tom cat server
  Web servers is used to run web components developed through servlets and JSP, which is having a web container. Web container is used to execute web programming components.

Java Has provided server side technologies
1)CGI
2)Servlet----.java
3)JSP

# Servl Contd

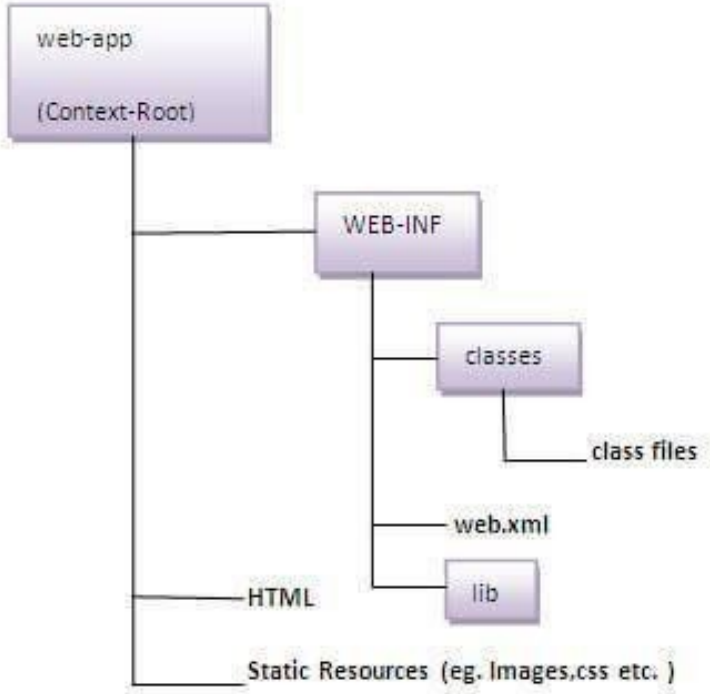
```
<!DOCTYPE html>
<html>

<body>
<form action="add" method="post">
  Enter 1st number: <input type="text" name="num1"><br>
  Enter 2st number: <input type="text" name="num2"><br>
  <input type="submit">
  </form>
</body>
          </html>
```

# Java and J2EE
# (17IS6DEJVA)

Unit 5

# Servlet Application

- **Six Steps to Running Your Servlet**
- Once Tomcat is installed and configured, you can put it to work. Six steps take you from writing your servlet to running it. These steps are as follows:
  - Create a directory structure
  - Create a Servlet
  - Compile the Servlet
  - Create a deployment descriptor
  - Start the server and deploy the project
  - Access the servlet



# Create and Compile 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
  - **Note: The HttpServlet class is widely used to create the servlet because it provides methods to handle http requests such as doGet(), doPost, doHead() etc.**
- Compile the servlet
  - For compiling the Servlet, jar file is required to be loaded. Different Servers provide different jar files.
  - Two ways to load the jar file
    - set classpath
    - 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.

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

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

  **web.xml file**

```
<web-app>

<servlet>
<servlet-name>Demo</servlet-name>
<servlet-class>DemoServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>Demol</servlet-
name>
<url-pattern>/welcome</url-
pattern>
</servlet-mapping>

                                                    </web-app>
```

**<web-app>** represents the whole application.
**<servlet>** is sub element of <web-app> and represents the  servlet.
**<servlet-name>** is sub element of <servlet> represents the name of the servlet.
**<servlet-class>** is sub element of <servlet> represents the class of the servlet.
**<servlet-mapping>** is sub element of <web-app>. It is used to map the servlet.
**<url-pattern>** is sub element of <servlet-mapping>. This pattern is used at client side to invoke the servlet.

# Example: steps to develop the LifeCycle application

- Create the index.html page
- Create the **LifeCycle** Servlet
- Create deployment descriptor

# Creating the index.html page

- For the sake of simplicity, this page will just have a button **invoke life cycle**. When you will click this button it will call **LifeCycleServlet** (which is mapped according to the entry in web.xml file).

**Html file**

```
<!DOCTYPE html>
<html>

<body>
<form action="addd" method="post">
  Enter 1st number: <input type="text" name="num1"><br>
  Enter 2st number: <input type="text" name="num2"><br>
  <input type="submit">
  </form>
</body>
</html>
```

The name of the Servlet is given in action attribute of form tag to which the request will be sent on clicking the button, in this case **LifeCycleServlet**.

# Creating the Servlet (FirstServlet)

- Now, its time to create the LifeCycleServlet which implements **init()**, **service()** and **destroy()** methods to demonstrate the lifecycle of a Servlet.

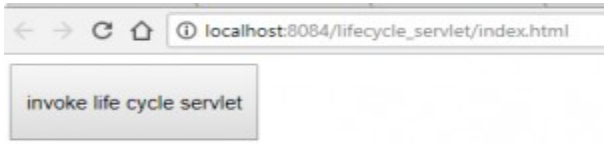
```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class AdServletdopostt extends HttpServlet
{
        public void doPost(HttpServletRequest req, HttpServletResponse res) throws IOException     //instead of doPost Service() can also be
used
{
                int i=Integer.parseInt(req.getParameter("num1"));
                  int j=Integer.parseInt(req.getParameter("num2"));
                  int k=i+j;
                  PrintWriter out = res.getWriter();
                  out.println(k);
}
}
```

# Creating deployment descriptor(web.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID" version="3.1">
  <display-name>Demodopost</display-name>
  <welcome-file-list>
    <welcome-file>NewFile3.html</welcome-file>
  </welcome-file-list>
  <servlet>
  <servlet-name>abcd</servlet-name>
  <servlet-class>AdServletdopostt</servlet-class>
  </servlet>

  <servlet-mapping>
  <servlet-name>abcd</servlet-name>
  <url-pattern>/addd</url-pattern>
  </servlet-mapping>
</web-app>
```

# Output:

- First of all the **index.html** file gets executed and then when the button is clicked the request goes to the Servlet, in this case LifeCycleServlet and the service() method handles the request.




- When the above **invoke life cycle servlet** button is clicked the code under service() method of
  LifeCycleServlet is executed and the below output is obtained :

# Cookies in Servlet

- 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.

- How Cookie works?
  - 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.

# 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.

# Advantages and Disadvantages of Cookies

- 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.**

# Cookie class

- **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 | changes the name of the cookie. |

| name) | |
|---|---|
| public void setValue(String value) | changes the value of the cookie. |

# Other methods required for 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.

- How to create Cookie?

```
Cookie ck=new Cookie("user","sonoo jaiswal");//creating cookie object
response.addCookie(ck);//adding cookie in the response
```

- How to delete Cookie?
  - It is mainly used to logout or signout the user.

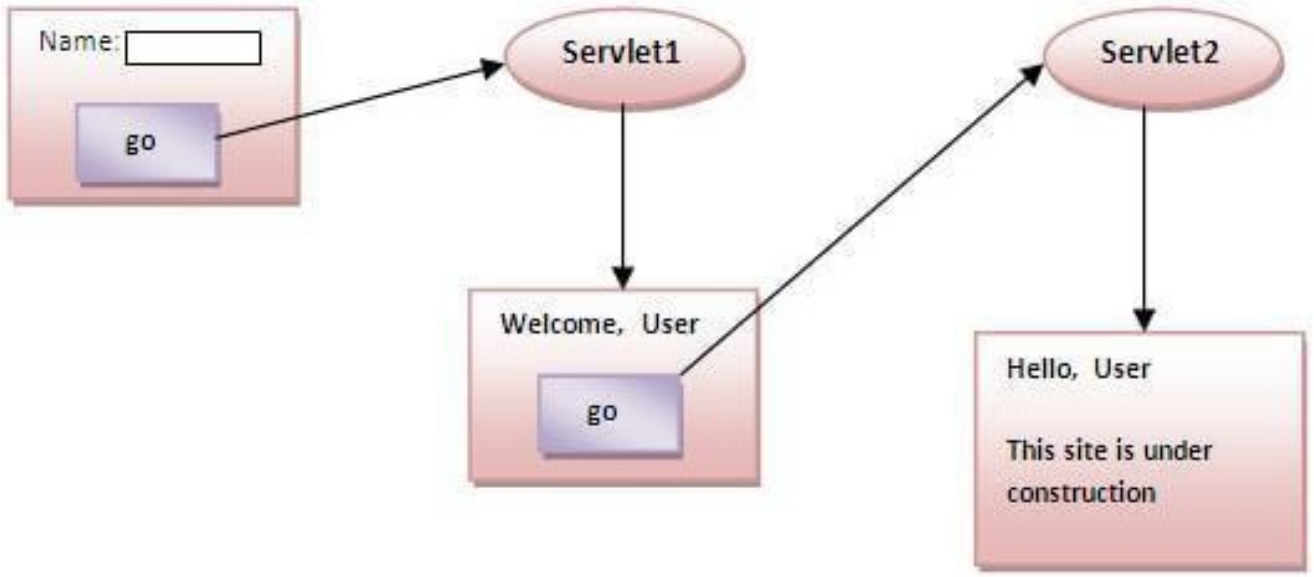
```
Cookie ck=new Cookie("user","");//deleting value of cookie ck.setMaxAge(0);//changing the maximum age to 0 seconds response.addCookie(ck);//adding cookie in the response
```

- How to get Cookies?

```
Cookie
ck[]=request.getCookies();
for(int i=0;i<ck.length;i++){
        out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name and value of cookie
}
```

# Simple example of Servlet Cookies

- In this example, we are storing the name of the user in the cookie object and accessing it in another servlet. As we know well that session corresponds to the particular user. So if you access it from too many browsers with different values, you will get the different value.



# index.html

```
<form action="servlet1" method="post">
    Name:<input type="text" name="userName"/><br/>
            <input type="submit" value="go"/>
</form>
```

FirstServlet.java

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

 public class FirstServlet extends HttpServlet {

        public void doPost(HttpServletRequest request, HttpServletResponse response){
         try{

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

         String
         n=request.getParameter("userName");
         out.print("Welcome "+n);

         Cookie ck=new Cookie("uname",n);//creating cookie
         object response.addCookie(ck);//adding cookie in the
         response

         //creating submit button
         out.print("<form action='servlet2'>");
         out.print("<input type='submit'
         value='go'>"); out.print("</form>");

         out.close();

            }catch(Exception e){System.out.println(e);}
        }
}
```

## SecondServlet.java

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

public class SecondServlet extends HttpServlet {

    public void doPost(HttpServletRequest request, HttpServletResponse response){
      try{

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

      Cookie ck[]=request.getCookies();
      out.print("Hello "+ck[0].getValue());

      out.close();

        }catch(Exception e){System.out.println(e);}
      }

}
```

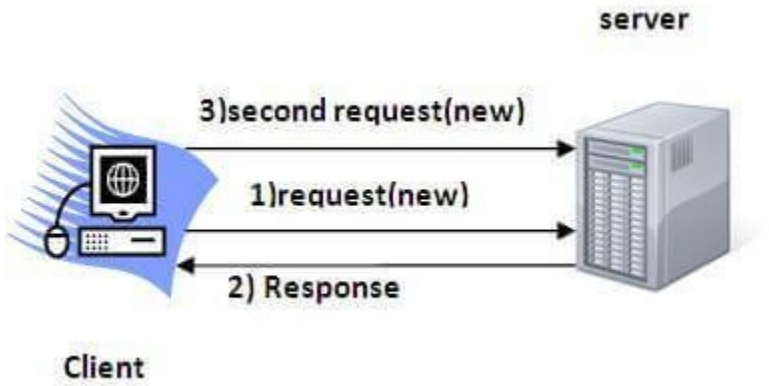
web.xml

```
<web-app>

    <servlet>
    <servlet-name>s1</servlet-name>
    <servlet-class>FirstServlet</servlet-class>
    </servlet>

    <servlet-mapping>
    <servlet-name>s1</servlet-name>
    <url-pattern>/servlet1</url-pattern>
    </servlet-mapping>

    <servlet>
    <servlet-name>s2</servlet-name>
    <servlet-class>SecondServlet</servlet-class>
    </servlet>

    <servlet-mapping>
    <servlet-name>s2</servlet-name>
    <url-pattern>/servlet2</url-pattern>
    </servlet-mapping>

</web-app>
```

# Create a Servlet Login and Logout Example using Cookies

# Session Tracking in Servlets

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




- Why use Session Tracking?
  - **To recognize the user** : It is used to recognize the particular user.

# Session Tracking Techniques

- There are four techniques used in Session tracking:

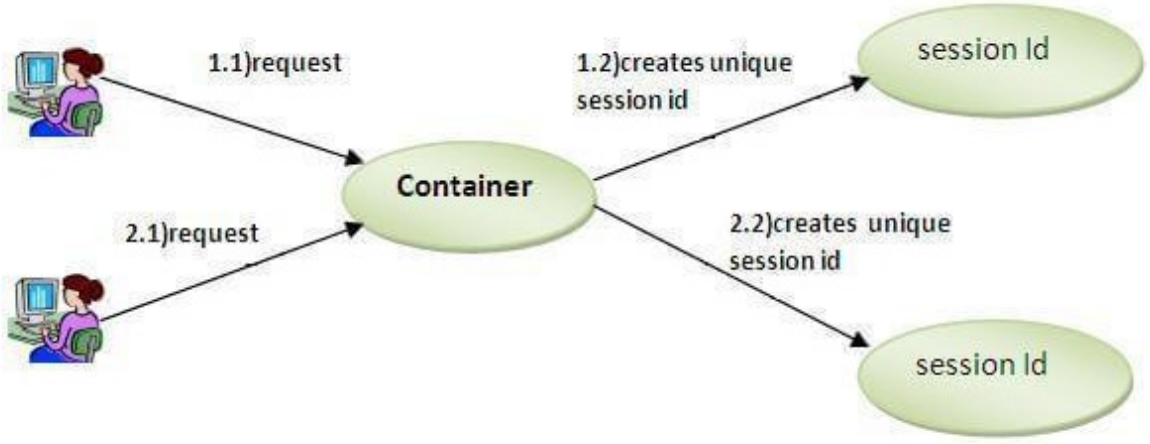
  - **Cookies:** A webserver can assign a unique session ID as a cookie to each web client and for subsequent requests from the client they can be recognized using the received cookie.
  - **Hidden Form Field:** A web server can send a hidden HTML form field along with a unique session ID as follows
    –

    ```
    <input type = "hidden" name = "sessionid" value = "12345">
    ```

    - This entry means that, when the form is submitted, the specified name and value are automatically included in the GET or POST data. Each time when web browser sends request back, then session_id value can be used to keep the track of different web browsers.
  - **URL Rewriting**: you can append some extra data on the end of each URL that identifies the session, and the server can associate that session identifier with data it has stored about that session.
    - For example, with http://dsce-ise.com/file.htm;sessionid = 12345, the session identifier is attached as sessionid = 12345 which can be accessed at the web server to identify the client.
  - **HttpSession:** Apart from the above mentioned three ways, servlet provides HttpSession Interface which provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.
    - The servlet container uses this interface to create a session between an HTTP client and an HTTP server. The session persists for a specified time period, across more than one connection or page request from the user.

# HttpSession

- HttpSession can be used to create 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.



- How to get the HttpSession object ?
  - The HttpServletRequest interface provides two methods to get the object of HttpSession:
    - **public HttpSession getSession():**Returns the current session associated with this request, or if the request does not have a session, creates one.
    - **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.

- commonly used methods of HttpSession interface
  - **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.

# Example of using HttpSession

- In this example, we are setting the attribute in the session scope in one servlet
  and getting that value from the session scope in another servlet.

  - To set the attribute in the session scope, we have used the HttpSession interface and setAttribute() method of

  - to get the attribute, we have used the getAttribute method.

index.html

```
<form action="servlet1">
    Name:<input type="text" name="userName"/><br/>
    <input type="submit" value="go"/>
</form>
```

FirstServlet.java

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


public class FirstServlet extends HttpServlet {

        public void doGet(HttpServletRequest request, HttpServletResponse response){
            try{

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

            String n=request.getParameter("userName");
            out.print("Welcome "+n);

            HttpSession
            session=request.getSession();
            session.setAttribute("uname",n);

            out.print("<a href='servlet2'>visit</a>");

            out.close();

                }catch(Exception e){System.out.println(e);}
          }

}
```

SecondServlet.java

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

public class SecondServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse
        response) try{

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

        HttpSession session=request.getSession(false);
        String n=(String)session.getAttribute("uname");
        out.print("Hello "+n);

        out.close();

            }catch(Exception e){System.out.println(e);}
      }

 }
```

## web.xml

```
<web-app>

    <servlet>
    <servlet-name>s1</servlet-name>
    <servlet-class>FirstServlet</servlet-class>
    </servlet>

    <servlet-mapping>
    <servlet-name>s1</servlet-name>
    <url-pattern>/servlet1</url-pattern>
    </servlet-mapping>

    <servlet>
    <servlet-name>s2</servlet-name>
    <servlet-class>SecondServlet</servlet-class>
    </servlet>

    <servlet-mapping>
    <servlet-name>s2</servlet-name>
    <url-pattern>/servlet2</url-pattern>
    </servlet-mapping>

</web-app>
```

Create Servlet HttpSession login and logout example Good

# Java Servlet Pages

# Introduction

- It is a server side program
- Called by the client
- Written in HTML rather than java programming language
- JSP converted to Java servlet first time when client requests JSP.

# Methods of JSP

1. jspInt()
2. Service()
3. JspDestroy

- jspInt() – called when JSP is requested and is used to initialize objects and variables.
- Service() – automatically called when it retrieves a connection to HTTP
- JspDestroy() – called when JSP terminates normally.

# JSP tags

- JSP program = HTML tags + JSP tags
- JSP tag contains the java code which is to be executed before the output of the JSP program is sent to the browser.
- 5 types of JSP Tags are

1. Comment tag –

```
<%--
    This is JSP comment
--%>
```

# JSP tags

2. Declaration statement tag – contains java declaration statements

```
<%!
    int a ; string s;
    Employee e = new employee();
%>
```

3. Directive tags- used to direct JSP virtual engine to perform a specific task such as importing a java package required, including a file etc.

```
<%@ page import= “java.sql.*” %>
```

# JSP tags

4. Expression tags – used for an expression statement whose result replaces the expression tag

<%! int a=5,b=10;%>

<% = a+b %>

5. Scriplet tags- contains commonly used java control statements

<% control statements %>

# JSP examples

```
<html>
<head><title>A Comment Test</title></head>
<body>
<h2>A Test of Comments</h2>
<%-- This comment will not be visible in the page
    source --%>
</body>
</html>
```

# JSP examples

```
<html>
<body>
<%!
      Int a=10;
      String s= “test”;
%>
<p>
      Int value is <% = a %>
      String value is <% = a %>
</p>
</body>
</html>
```

# JSP that define a method and illustrate its usage

```
<html>
<body>
<%!
    Int findsum(int a , int b)
    {
    return a+b;
    }
%>
<p>
    Sum of 2 and 5 is <%= findsum(2,5) %.
</p>
</body>
</html>
```

# Requesting string

- Browser generates user request whenever submit button is pressed
- User request consists of
  - URL
  - Query string

  http://www.gmail.com/Service/login.jsp**? – URL**

  **Username=jim&password=jim1234 – Query string**

# URL

- Divided into 4 parts
  1. Protocol –rules that are used to transfer request string from browser to JSP program.
  2. Host – It is internet protocol address (IP) or name of the server that contains the JSP program
  3. Port – It is the port that host monitors. Default is 80
  4. Virtual path or JSP program – Server maps this virtual path to physical path

# Query string

- JSP program should parse the query string to extract the values of fields that are to be processed by JSP.
- A JSP can parse the query string in 2 ways
  - Using request.getParameter()

  GetParameter() requires an argument which is the name of the field whose value you want to retrieve

  <%!

  String uname= request.getParameter(“username”);

  %>

# JSP program

- 4 predefined implicit objects of every JSP program
  - Request – instance of HttPServleRequest and represents request object
  - Response - instance of HttPServleResponse and represents response object
  - Session – Instance of HttpSession
  - Out – Instance of JspWriter that is used to send a resopnse to the client.

# User sessions

- 3 commonly used methods to track a session
  - Using hidden field
  - Using a cookie
  - Using a javaBean

  Hidden field – a field in HTML whose value is not displayed on the HTML page

  Value can be assigned to hidden field before program sends the HTML page to browser.

# Cookies

- It is a small piece of information created by JSP program that is stored in the client's hard disk by the browser.

# JSP program to create a cookie

```
<html>
<body>
<%!
        String cName= “userid”;
        String cValue= “2569855v”;
        Cookie mycookie= new Cookie(cname,cvalue);
%>
<%>
        response.addcookie(mycookie)
<%>
</body>
</html>
```

# JSP program to read a cookie

```
<html>
<body>
<%!
        String cName= “userid”;
        String name;
        String value;
        Int found=0;
%>
<%>
         Cookie[] mycookies= request.getCookies();
For(inti=; i<myCookies.length; i++)
{
Name=myCookies[i].getName();
If(name.equals(cname))
{value=mycookies[i].getvalue();
Found=1;
}
}
```

# JSP program to read a cookie

```
If(found==1
{
%>
<p> cookie name: <%= cname %> </p>
<p> cookie value: <%= value %> </p>
<%
}
Else
{
<p> Cookie not found </p>
<%>
}
%>
</body>
</html>
```

# Session objects

- JSP database system will share information among JSP programs within a session by using session object.
- Each time a session is created, a unique ID is assigned to the session and stored as a cookie.